Univerzita Pardubice
Dopravní fakulta Jana Pernera

Kurz celoživotního vzdělávání

Elektrotechnické systémy pro environmentálně příznivou dopravu

Předmět

Pokročilé systémy digitálního řízení a komunikace v dopravní technice

Studijní materiál

Mikroprocesorová řídící technika, 1. část

Ing. Zdeněk Mašek



Obsah

1 Aplikace běžící v reálném čase ... 3
2 Architektura procesoru ... 4
2.1 Von Neumannova architektura ... 4
2.2 Harvardská architektura ... 7
2.3 Modifikovaná harvardská architektura ... 9
2.4 Popis jednotlivých částí struktury počítače ... 10
3 Historický vývoj ... 26
4 MCU AVR – hlavní rysy ... 27
5 Architektura MCU AVR ... 29
5.1 Vnitřní struktura procesoru (CPU) ... 30
5.2 CISC vs. RISC procesory ... 38
5.3 Činnost CPU ... 43
5.4 Zdroje taktovacího kmitočtu ... 59
6 Paměťový prostor ... 72
6.1 Paměť programu ... 73
6.2 Paměť dat ... 76
6.3 Zásobník (stack) ... 78
6.4 Prostor I/O registrů – přístup k periferiím ... 82
6.5 Adresovací módy paměti programu a dat ... 84
7 Vstupně-výstupní porty ... 95
7.1 Vlastnosti ... 96
7.2 Nastavení portu ... 98
7.3 Alternativní funkce portů/pinů ... 100
8 Zapojení uživatelského vstupu k DI ... 101
8.1 Přímé připojení k digitálnímu vstupu MCU ... 101
8.2 Připojení většího množství tlačítek ... 102
8.3 Ošetření zákmitů tlačítek ... 111
8.4 Maticová klávesnice ... 120
8.5 Přepěťová ochrana vstupů ... 125
9 Zapojení uživatelského výstupu k DO ... 137
9.1 Malá zátěž (např. LED) ... 137
9.2 Posílení digitálního výstupu tranzistorem ... 138
9.3 Galvanické oddělení zátěže ... 143
10 Rozšíření počtu vstupů a výstupů ... 146
10.1 Mapování I/O do adresního prostoru externí paměti ... 147
11 Konverze napěťových úrovní ... 154
11.1 Napěťové úrovně ... 154
11.2 Způsoby propojení ... 155
12 Zobrazovací jednotky ... 169
12.1 LED displeje ... 169

12.2 Znakové LCD ... 184
13 Použití vyššího programovacího jazyka na MCU ... 198
13.1 Úvod ... 198
13.2 Kompilátor HLL a ne-PC aplikace ... 198
13.3 Optimalizace ... 199
13.4 Ukázky optimalizací ... 201
13.5 Obecná pravidla při psaní programu ... 204
13.6 Přehled C kompilátorů pro AVR ... 207
14 Přerušovací systém ... 211
14.1 Úvod a princip přerušení ... 211
14.2 Přerušení na AVR ... 220
14.3 Externí přerušení ... 234
15 Programovací techniky v embedded aplikacích ... 240
15.1 Základní pojmy ... 240
15.2 Základní techniky tvorby kódu ... 241
15.3 Funkční fronty ... 256

1 Aplikace běžící v reálném čase

Lze rozdělit na:

- Monitorování – sběr dat z procesu
- Řízení – ovlivňování procesu v reálném čase
    - Logické – kombinační a sekvenční logické obvody
    - Spojité (kvazispojité) – procesor má kontakt se soustavou v diskrétních okamžicích, různé regulační úlohy

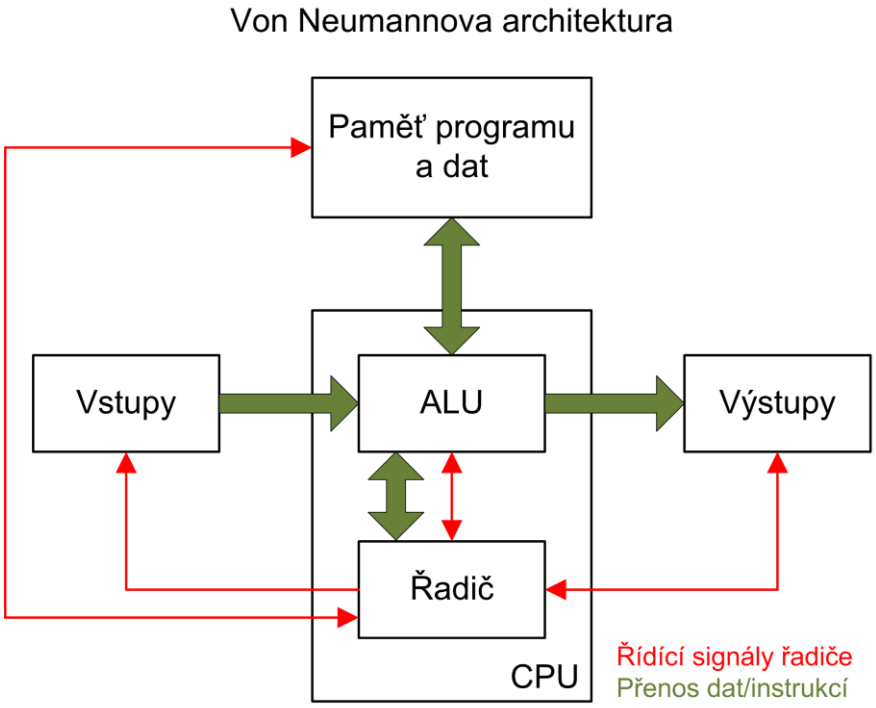
Co je to „reálný čas“ ?

- Úlohy s malou časovou náročností
    - Perioda vzorkování je 10 až 100 ms.
    - Použití 8-bit a 16-bit procesorů.
    - Logické řízení, spojité řízení soustav s dlouhými časovými konstantami (např. tepelné soustavy).
- Úlohy se střední časovou náročností
    - Perioda vzorkování je 1 až 10 ms.
    - Použití 8-bit a 16-bit procesorů.
    - Např. regulace otáček, polohy, regulace mechanických soustav.
- Úlohy s vysokou časovou náročností
    - Perioda vzorkování je 10 μs až 100 μs.
    - Použití 16-bit a 32-bit procesorů.
    - Např. řízení polovodičových měničů (výkonové elektroniky).
- Úlohy s extrémní časovou náročností
    - Perioda vzorkování jsou zlomky μs až 1 μs.
    - 16-bit a 32-bit procesory DSP (DSP = digitální signálový procesor).
    - Např. zpracování zvuku nebo obrazu v reálném čase.

2 Architektura procesoru

2.1 Von Neumannova architektura

- Definuje strukturu počítače.
- Byla vynalezena jako první a platí dodnes.
- Von Neumann – americký matematik, autor architektury, rok 1945
    - EDVAC – elektronkový počítač, nástupce ENIACu (prvního el. počítače)
        - EDVAC - program uložený v paměti, ENIAC – přepínače, děrné štítky
- Z novější doby např. PC, 80C166, některé ARMy.
- Vlastnosti:
    - Počítač se skládá ze základních bloků: paměť, aritmeticko-logická jednotka, vstupní a výstupní jednotky, řadič.



Obrázek 1 Von Neumannova architektura

    - Struktura počítače je nezávislá na typu řešené úlohy. Úlohu definuje program uložený v paměti.
    - **Instrukce a data jsou uložena v jedné paměti**
        - Paměť je rozdělena na buňky o stejné velikosti, kterým je přiřazeno číslo – adresa.
        - Existuje pouze jedna adresová a datová sběrnice.
        - Jeden společný adresní prostor pro kód a data – v paměti jsou uloženy instrukce i data. Číselná hodnota adresy udává, zda se přistupuje do kódu nebo do dat.

- Jednotný přístup k instrukcím i datům - není třeba rozlišovat, zda se jedná o přístup k programu nebo k datům.
- Zjednodušení návrhu čipu, nižší cena.
- Kód (instrukce) a data nelze přenášet najednou – snížení rychlosti.
- Chyba v programu může zapříčinit pád programu (přepsání instrukcí).

o Program je tvořen posloupností příkazů - instrukcí. Instrukce se provádějí postupně v pořadí, ve kterém jsou v programu uvedeny.

```
lds R0, adr1      ;R0 = obsah na adrese adr1
lds R1, adr2      ;R1 = obsah na adrese adr2
add R0,R1         ;R0 = R0 + R1 (součet čísel v reg. R0 a R1)
```

o Změna pořadí provádění instrukcí se realizuje pomocí instrukcí skoku (nepodmíněný, podmíněný).

Ukázka podmíněného skoku:

```
        ldi R18,0xFF      ;naplnění registru R18 konstantou 255(10)
hop:    dec R18           ;dekrementuj obsah v R18
        brne hop          ;skákej pokud v R18 není 0
```

Ukázka nepodmíněného skoku:

```
        …                 ;nějaká instrukce
        …                 ;nějaká instrukce
        jmp hop           ;skoč na řádek hop:
        add R0,R1         ;nevykoná se, bude vždy přeskočeno
hop:    …                 ;nějaká instrukce
        …                 ;nějaká instrukce
```

o Instrukce a čísla jsou reprezentována ve dvojkové soustavě.

Např. Strojový kód (Opcode) instrukce SEI u mikrokontroléru (MCU) AVR:

SEI – Set Global Interrupt Flag

Description:

Sets the Global Interrupt Flag (I) in SREG (Status Register). The instruction following SEI will be executed before any pending interrupts.

Operation:

(i) I ← 1

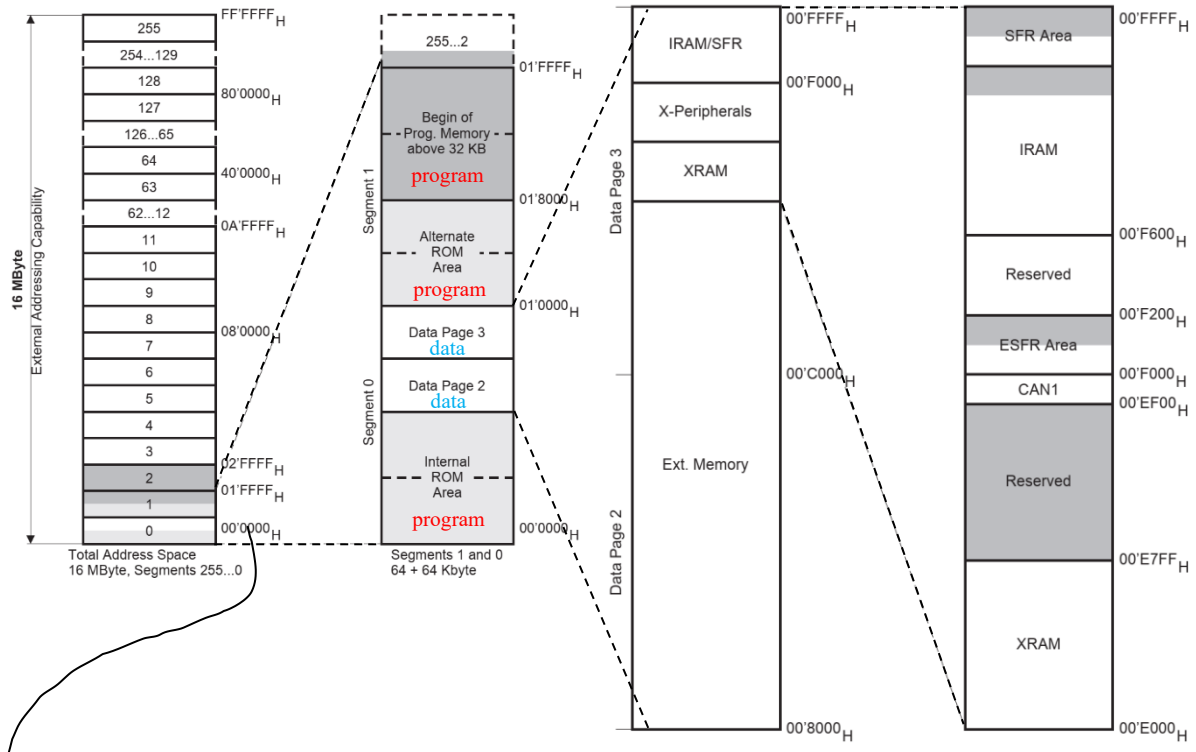
	Syntax:	Operands:	Program Counter:
(i)	SEI	None	PC ← PC + 1

16-bit Opcode:

1001	0100	0111	1000

- I systémy s fyzicky oddělenými pamětmi, ale společnou sběrnicí přenášející instrukce i data, a jedním adresním prostorem, se označují jako systémy založené na Von Neumannově architektuře.
- Unifikovaný přístup k instrukcím a datům umožňuje zacházet s instrukcemi jako s daty a opačně.

- Umožňuje např. načíst program z pevného disku, uložit ho do RAM a z ní ho následně spustit.
- Záleží tedy jen na interpretaci jedniček a nul uložených na paměťovém médiu - při kopírování programu z pevného disku do RAM se jedná o kopírování dat (s instrukcemi se zachází jako s daty) a teprve, když se program spustí z RAM, jsou tato data chápána jako instrukce.

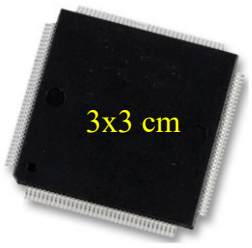


Obrázek 2 Společný adresní prostor (16-bit MCU 80C167)

Adresa zapsaná v šestnáctkové soustavě. Např $01FFF_H$ = 8191 v desítkové soustavě. Zde tyto adresy označují počátky a konce segmentů nebo bloků paměti.

U MCU 80C167 je paměťový prostor rozdělen na 256 segmentů, každý o velikosti 64 kByte (64 kB). Max. velikost paměti je 256 x 64 kByte = 16 384 kByte = 16 MByte.

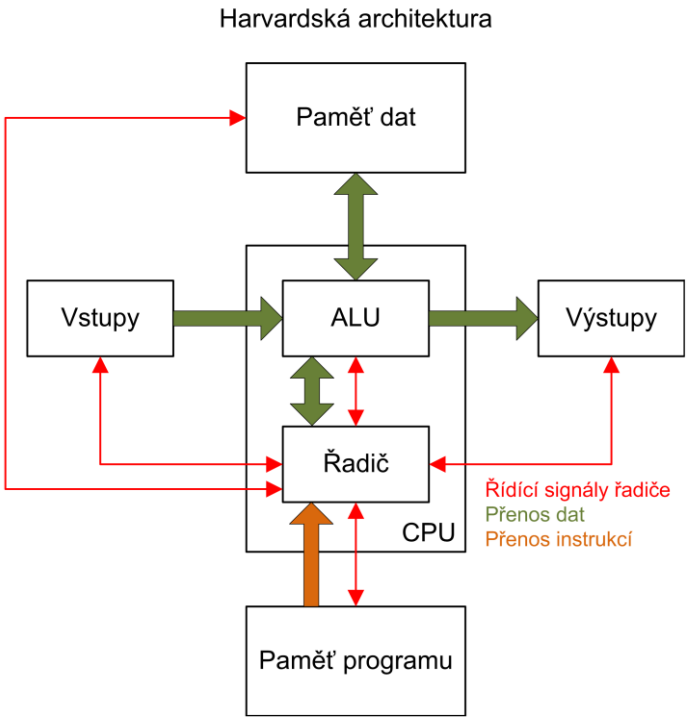
Segmenty se dále dělí na stránky (page) o různé velikosti a významu (to je dáno pevně



Obrázek 3 Mikrokontrolér (jednočipový mikrokopočítač) 80C167 v SMD pouzdru P-MQFP-144-8

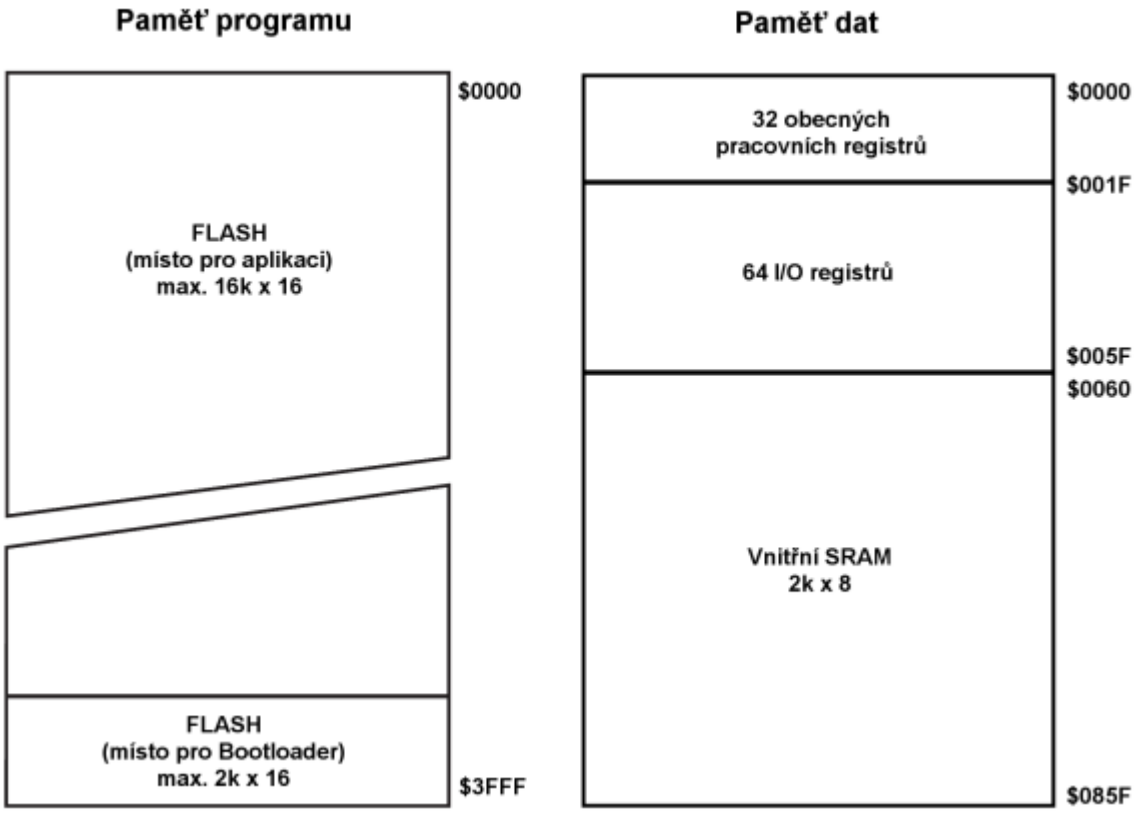
2.2 Harvardská architektura

- Struktura vytvořená v roce 1947 na Harvardově univerzitě v Cambridge ve státě Massachusetts (USA) – počítač MARK II.
- Z novější doby např. 8051, AVR, PIC, některé ARMy, digitální signálové procesory (výčet zahrnuje i systémy s modifikovanou harvardskou architekturou).
- Fyzicky oddělená paměť programu (kód) i data a oddělené sběrnice pro přístup do těchto pamětí.
- Vlastnosti:
    - Není nebezpečí, že se programátorskou chybou poškodí program (nelze zapisovat data do paměti programu, ani instrukce do paměti dat).
    - Paralelní (současný) přístup do obou pamětí (instrukce i data se čtou najednou) = zvýšení rychlosti.
    - Díky fyzickému oddělení pamětí lze použít různé typy pamětí:
        - FLASH, EEPROM nebo ROM pro kód
        - SRAM nebo DRAM pro data
        - Různá bitová šířka sběrnice pro každou z pamětí (např. 16-bit pro kód, 8-bit pro data).
    - Dva adresní prostory – pro instrukce a pro data.
    - Různé instrukce pro přístup do oddělených adresních prostor.
    - Složitější a dražší hardware než u Von Neumannovy architektury.



Obrázek 4 Harvardská architektura

- Čistá harvardská architektura neumožňuje používat paměť programu pro uložení dat a paměť dat pro uložení instrukcí, ačkoliv je to v některých případech užitečné. Proto vznikají různé **modifikace základní harvardské architektury**, které toto umožňují.
    - Vybrané případy, kdy je vhodné mít do paměti programu přístup na data:
        - Uložení konstant do paměti programu při jejím programování a vyčítání těchto konstant za chodu programu – řetězce, různé převodní tabulky apod., za účelem šetření kapacity paměti dat (je obvykle menší než paměť programu).
        - Programování paměti programu – ukládání instrukcí do paměti dat při programování paměti programu – aktualizace SW zařízení. Procesor sám paměť programu přeprogramuje bez nutnosti jeho vyndávání z DPS za účelem programování paměti.
- Moderní procesory nemají čistě Von Neumannovu nebo harvardskou architekturu, obsahují prvky z obou dvou.



Obrázek 5 Oddělené adresní prostory (8-bit MCU AVR)

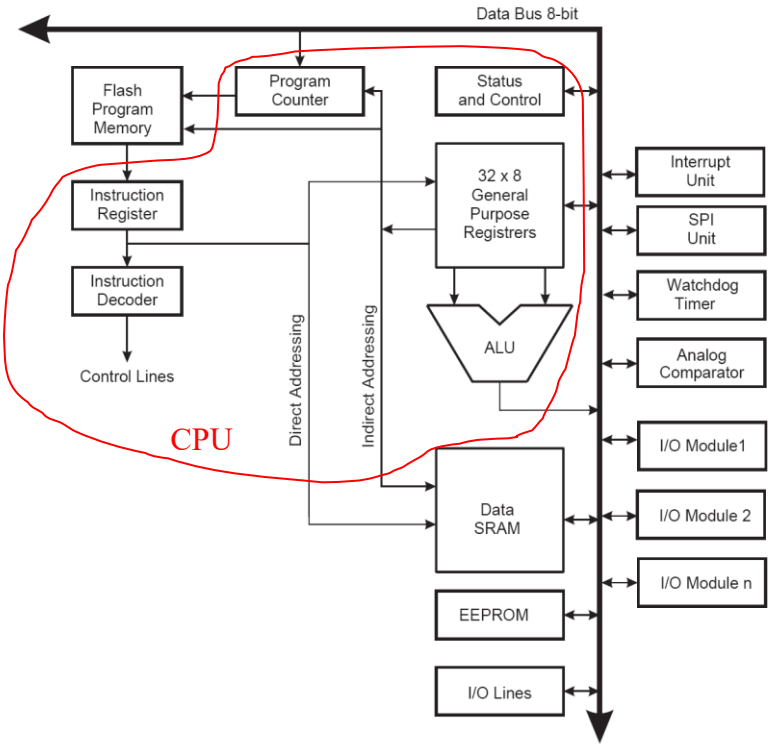
2.3 Modifikovaná harvardská architektura

- Lze říci, že se jedná o kombinaci Von Neumannovy a harvardské architektury.
- Většina harvardských procesorů nemá čistě harvardskou architekturu.
- Modifikovaná harvardská architektura vznikla za účelem možnosti zacházet s instrukcemi jako s daty a opačně (u Von Neumannovy architektury je to přirozené) při zachování rychlého současného přístupu do paměti programu a dat (typická vlastnost čistě harvardské architektury).
- Prakticky mohou procesory založené na modifikované harvardské architektuře vykazovat jednu nebo více z níže uvedených vlastností nebo jejich kombinace:
    - Paměti pro instrukce a pro data jsou oddělené, ale sběrnice pro přenos instrukcí i dat je jen jedna společná.
        - Oddělené rychlé vyrovnávací (cache) paměti zvlášť pro instrukce a pro data.
    - Jeden společný adresní prostor pro instrukce i data – z pohledu programátora se procesor jeví jako Von Neumannovský.
- Některé procesory s modifikovanou harvardskou architekturou jsou modifikované méně (blíží se k čisté harvardské architektuře), některé více (blíže k Von Neumannově architektuře), např.:
    - 8-bit AVR – malé odchylky od čistě harvardské architektury
        - Zavedení instrukcí pro zápis/čtení dat do/z paměti programu.
    - 32-bit ARM9 – větší odchylky od čistě harvardské architektury
        - Společný adresní prostor paměti programu a dat.
        - Jedna společná sběrnice do paměti programu a dat.
- Oddělené rychlé vyrovnávací (cache) paměti zvlášť pro instrukce a pro data.

2.4 Popis jednotlivých částí struktury počítače

2.4.1 Procesor (CPU – Central Procesor Unit)

- CPU = řadič + ALU (aritmeticko-logická jednotka)
- Řadič podle programu dává pokyny ke zpracování dat a produkuje výstupní data.
- Dekóduje instrukce.
  - Řídí pořadí vykonávání instrukcí.
  - Pomocí signálů řídí ostatní části počítače (ALU, paměť,…).
    - Červené spoje v Obrázek 1 a Obrázek 4 jsou obousměrné, protože znamenají jak přenos řídících signálů z řadiče do jednotlivých jednotek, tak přenos stavových signálů z jednotek do řadiče (informování řadiče o stavu dané jednotky).
  - Řadič není fyzicky oddělen od jednotlivých částí, nýbrž je „rozprostřen“ po ploše čipu, jednotlivé jeho části jsou co nejblíže k řízeným prvkům.
- ALU podle pokynů řadiče provádí jednotlivé výpočty a zpracovává data.

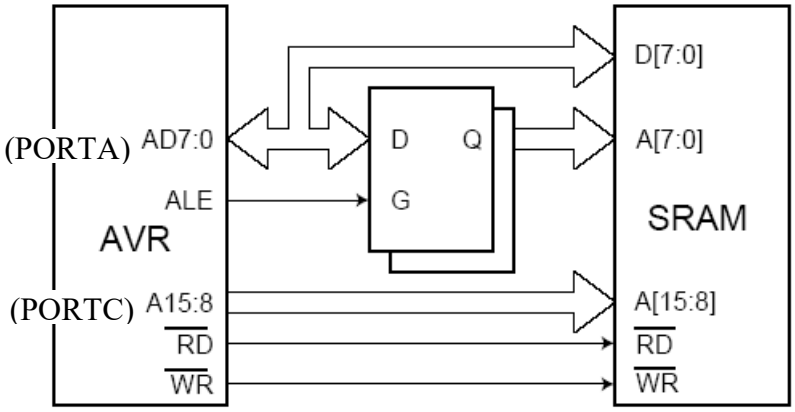


Obrázek 6 CPU (řadič + ALU)

2.4.2 Sběrnice (BUS)

- Sada vodičů propojující jednotlivé části počítače.
- Dělení sběrnic:
  - **Datová sběrnice** – pro přenos dat mezi jednotlivými částmi počítače, její šířka je obvykle rovna šířce slova ALU (např. 8-bit, 16-bit atd.).
  - **Adresová sběrnice** – pro přenos adresy (výběr buňky v paměti). Její šířka závisí na velikosti paměti, kterou je možné připojit (kolik paměti lze uadresovat). Např. 16-bit adresa → 216 adres → 65536 paměťových buněk.
  - **Řídící sběrnice** – pro přenos řídících signálů, např. signály čtení/zápis z/do paměti, signalizace přerušení apod.

Příklad připojení externí paměti dat k 8-bit MCU AVR



Obrázek 7 Připojení externí paměti dat k 8-bit MCU AVR

Ext. paměť SRAM

D[7:0] – datová sběrnice (8-bit), vodiče D7 až D0

A[7:0] a A[15:8] – adresová sběrnice (16-bit), vodiče A0 až A15, max. velikost externí paměti dat je 2^{16} = 65536 B (64 kB)

/RD – čtení dat (/RD v logické 0)

/WR – zápis dat (/WR v logické 0)

MCU AVR

A15:8 (PORTC) – horní byte 16-bit adresy

AD7:0 (PORTA) – multiplexovaný port, přenáší dolní byte 16-bit adresy a zároveň datový byte (hodnotu pro zápis do ext. paměti nebo hodnotu vyčtenou z ext. paměti)

řídící signály:

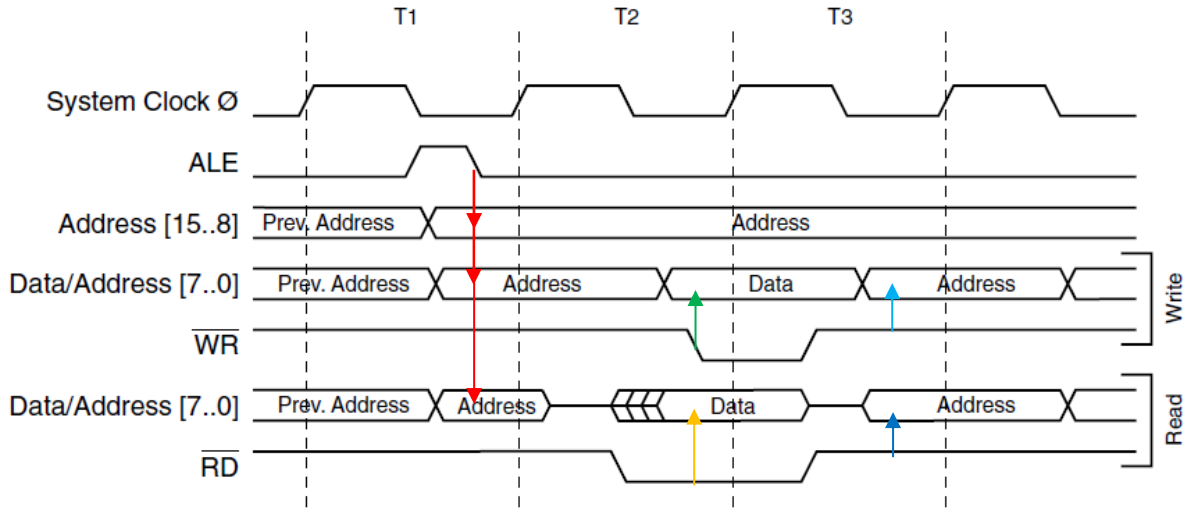
ALE – data platná (sestupná hrana ALE)

/RD – čtení dat (/RD v logické 0)

/WR – zápis dat (/WR v logické 0)

Pozn.: Lomítko „/“ označuje signál aktivní v logické 0.

Komunikace s externí pamětí dat je multiplexovaná. Datová sběrnice je multiplexovaná s adresní sběrnicí (po datové sběrnici se přenáší i dolní bajt adresy). Na straně MCU AVR stačí pro připojení externí paměti pouze dva 8-bit porty, PORTA – multiplexovaný port pro přenos dolního byte adresy a pro přenos jednoho byte dat (AD7:0), PORTC – port pro přenos horního byte adresy (A15:8). 8-mi násobný D klopný obvod slouží pro zachycení dolního byte 16-bit adresy a jeho udržení na adresních vstupech A7:0 externí paměti SRAM v průběhu přenosu datového byte mezi MCU a externí SRAM.



Obrázek 8 Časování signálů pro přístup do externí paměti SRAM

Přístup do externí paměti SRAM je v tomto případě 3-cyklový (periody oscilátoru T1, T2, T3).

Vystavení adresy

Nejprve MCU v taktu T1 vystaví adresu (horních 8-bitů A15:8 a dolních 8-bitů AD7:0) a platnou adresu MCU potvrdí sestupnou hranou signálu ALE (červené šipky). Tím se dolních 8-bitů adresy zachytí do externího záchytného registru, který podrží na svém výstupu dolní půlku 16-bit adresy během doby, kdy se na sběrnici AD7:0 objeví data. V průběhu přístupu do externí paměti pak signál ALE zůstává trvale v logické 0 (úrovni LOW).

Zápis do ext. paměti SRAM

V průběhu taktu T2 vystaví MCU na multiplexovaném portu (PORTA) datový byte (Data) pro zápis do ext. paměti SRAM a potvrdí ho sestupnou hranou signálu /WR (zelená šipka). Tím se datový byte zapíše do ext. paměti SRAM.

V průběhu taktu T3 se signál /WR vrátí do úrovně HIGH a na multiplexovaném portu (PORTA - AD7:0) se opět objeví adresa (světle modrá šipka).

Tím je zápis dokončen.

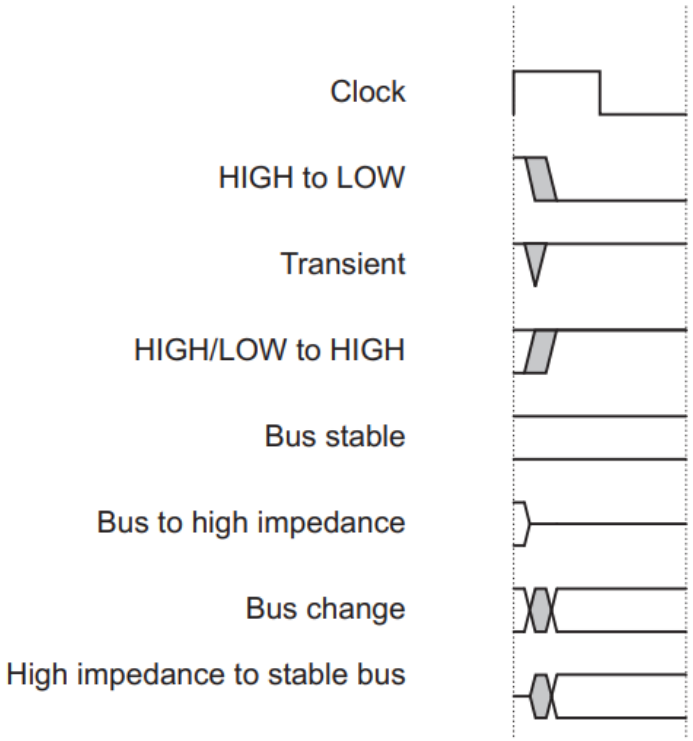
Čtení z ext. paměti SRAM

V průběhu taktu T2 dá MCU povel ke čtení z ext. paměti SRAM tím, že signál /RD nastaví z logické 1 do logické 0 po dobu jednoho taktu. Tím započne čtení datového byte z ext. paměti

SRAM. Externí paměť SRAM vystaví na datovou sběrnici jeden byte dat (PORTA - AD7:0), oranžová šipka.

Po dokončeném čtení se v průběhu taktu T3 signál /RD vrátí do úrovně HIGH a na multiplexovaném portu (PORTA) se opět objeví adresa (tmavě modrá šipka).

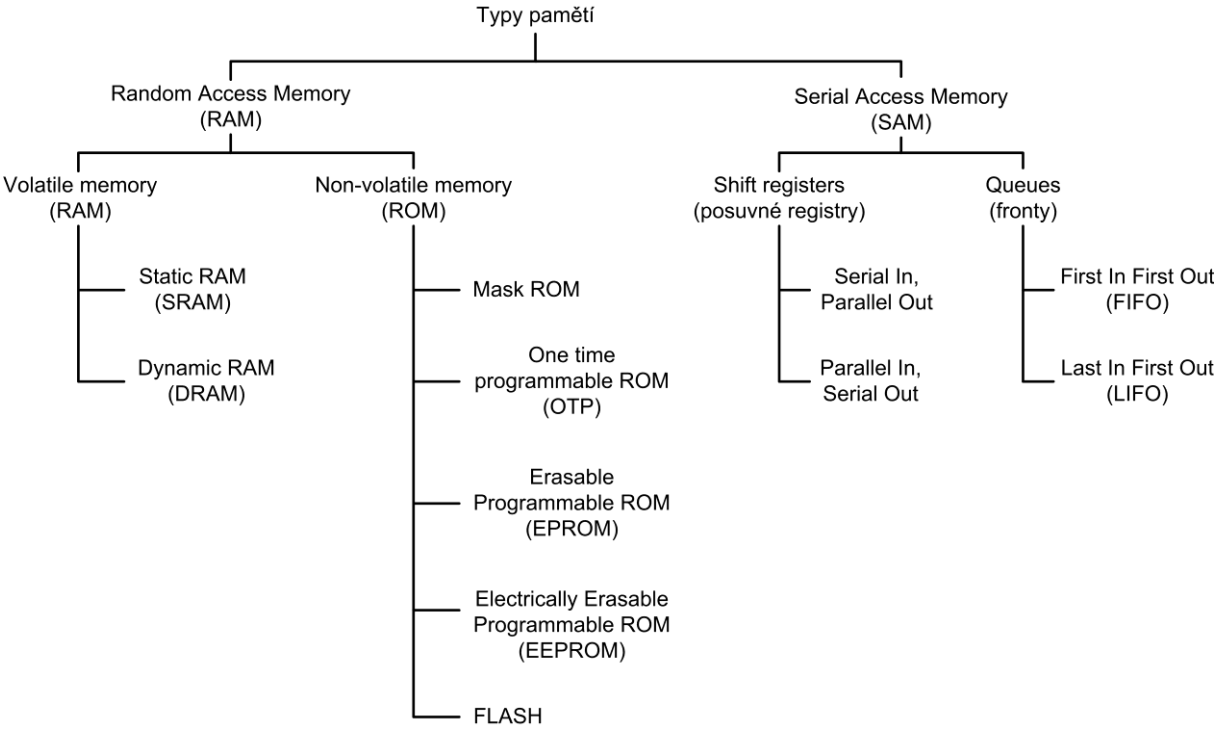
Tím je čtení dokončeno.



Obrázek 9 Značky používané v popisu časování komunikace na sběrnicích

2.4.3 Druhy pamětí

Paměti můžeme dělit podle různých hledisek. Např. podle způsobu přístupu k datům (zda je možné čtení i zápis nebo jen čtení, zda je přístup k datům přes paralelní nebo sériovou sběrnici), podle výrobní technologie, podle rychlosti přístupu k datům, podle hustoty informace uložené na čipu….



Obrázek 10 Typy pamětí

Zabývejme se nyní levou částí pavouka zobrazeného na Obrázek 10.

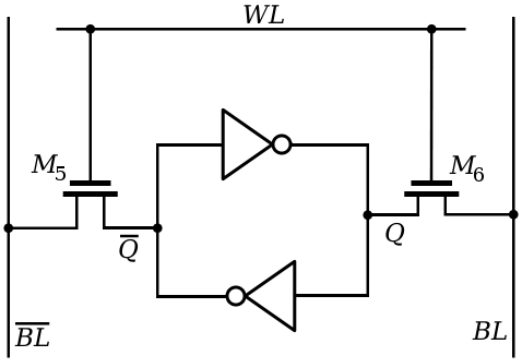
Pojem RAM lze chápat buď velmi obecně jako pamět, u které lze k datům ležícím na určité adrese přistupovat přímo, anebo lze pojem chápat jako paměť, do které lze zapisovat i číst a po odpojení napájení se data v ní ztratí (převažující definice). Ztrátu dat po odpojení napájení vyjadřuje anglické slovo „volatile“.

Druhý typ pamětí, mající v označení ROM nebo FLASH nebo mající v názvu anglické slovo „non-volatile“, jsou paměti, které svůj obsah udrží i po ztrátě napájení. Z těchto pamětí lze vždy číst, zda do nich jde zapisovat, kolikrát a jakým způsobem, to záleží na jejich typu (konstrukci).

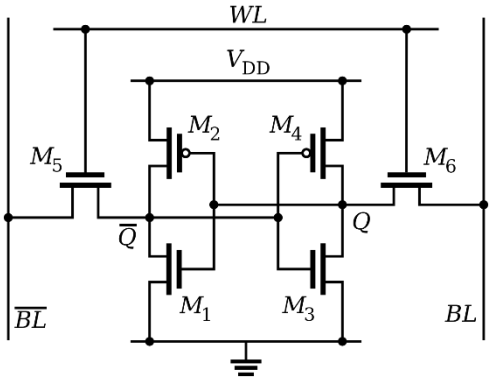
Paměti RAM

Dělíme na dvě základní kategorie, podle konstrukce paměťové buňky.

- **Statická RAM (SRAM)**
  - Jeden bit informace je v každé paměťové buňce uchován pomocí bistabilního klopného obvodu (anglicky „flip-flop“). Buňka (1 bit) může typicky obsahovat celkem 6 tranzistorů, z toho 4 tvoří bistabilní klopný obvod a dva ostatní slouží pro čtení a zápis hodnoty bitu.
  - Vlastnosti
    - Vysoká rychlost přístupu datům, nepotřebuje obnovovat data (refresh) jako DRAM.
    - Nižší spotřeba energie oproti DRAM (odpadají obnovovací cykly), a to zejména v klidovém stavu, když se obsah paměti nemění.
    - Malá hustota informace na čipu – SRAM bude mít oproti DRAM na stejné ploše čipu výrazně menší kapacitu (protože buňka obsahuje moc tranzistorů).
    - Vyšší cena oproti DRAM.
    - Použití jako hlavní paměť (když stačí malá kapacita v porovnání s DRAM) nebo jako rychlá vyrovnávací paměť (tzv. CACHE) v kombinaci s DRAM (vysoká kapacita paměti).



Obrázek 11 Paměťová buňky (1 bit) SRAM s hradly (invertory) tvořící bistabilní klopný obvod místo tranzistorů pro snazší představu funkce



Obrázek 12 Paměťová buňka (1 bit) SRAM se šesti MOS tranzistory

WL (word line) – adresní vodič pro výběr skupiny paměťových buněk (= 1 řádek v paměti)
BL (bit line) – pro čtení a zápis hodnoty (1 bitu) do buňky, $\overline{BL}$ je negovaná hodnota
Q a /Q – vstupy (zápis) nebo výstupy (čtení) bistabilního KO, drží hodnotu bitu (0 nebo 1)

Bistabilní klopný obvod tvořený čtveřicí MOS tranzistorů na Obrázek 12, resp. dvojicí invertorů pro snazší představu funkce na Obrázek 11, má dva stabilní stavy, které se v čase nemění:

- Q = 0, /Q = 1 (tranzistor M3 = ON, M4 = OFF, M1 = OFF, M2 = ON)
- Q = 1, /Q = 0 (tranzistor M3 = OFF, M4 = ON, M1 = ON, M2 = OFF)

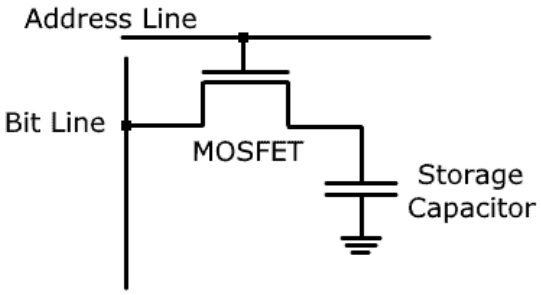
Čtení probíhá takto:

- Sepnutí tranzistorů M5 a M6 napětím z adresního vodiče WL.
- Hodnoty Q a /Q se dostanou na bitové vodiče BL a /BL a dál do zesilovacího obvodu (není zakreslen), který z rozdílu napětí mezi linkami BL a /BL vyhodnotí logickou úroveň buňky.

Zápis probíhá takto:

- Vystavení nové hodnoty bitu na bitové vodiče BL a /BL, např. BL = 1, /BL = 0.
- Sepnutí tranzistorů M5 a M6 napětím z adresního vodiče WL.
- Pokud je nově zapisovaná hodnota bitu stejná jako předchozí, nic se nemění.
- Pokud je nově zapisovaná hodnota bitu jiná než předchozí, protlačí se přes tranzistory M5 a M6 do bistabilního klopného obvodu a donutí ho ke změně stavu.

- **Dynamická RAM (DRAM)**
    - Jeden bit informace je v paměťové buňce uchován ne pomocí bistabilního klopného obvodu, nýbrž pomocí jednoho MOSFET tranzistoru a kondenzátoru. Jako kondenzátor se využívá kapacita izolované řídící elektrody (gate) MOSFET tranzistoru.
        - Tranzistor sepnut – lze číst nebo zapisovat hodnotu bitu.
        - Tranzistor vypnut – uchování informace v kondenzátoru.



Obrázek 13 Paměťová buňka DRAM (1 bit informace)

- Čtení
    - Aktivace Address Line (přivedení napětí na gate MOSFETu).
    - Náboj kondenzátoru nabíjí vodič Bit Line, který má parazitní kapacitu. Stoupá na něm napětí, po překročení prahové úrovně je rozpoznán logický stav (hodnota bitu). Tranzistor musí být otevřen dostatečně dlouho – čtení nějakou dobu trvá.

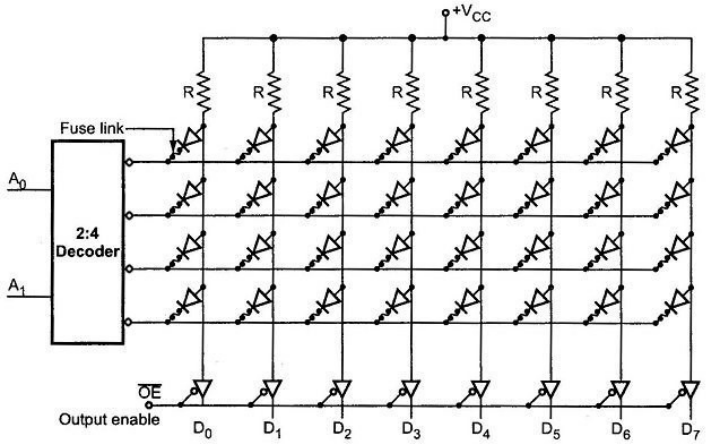
        - Zápis
            - Zamýšlená logická úroveň HIGH (vysoké napětí) nebo LOW (nízké napětí) se přivede na vodič Bit Line.
            - Tranzsitor se sepne a kondenzátor se nabije/vybije na napětí vodiče Bit Line. Tím se do buňky uloží informace HIGH (log. 1) nebo LOW (log. 0). Tranzistor musí být sepnut dostatečně dlouho, aby se kondenzátor stihl nabít (úroveň HIGH) nebo vybít (úroveň LOW) – zápis nějakou dobu trvá.
    - Vlastnosti
        - Vysoká hustota informace na čipu – DRAM bude mít oproti SRAM na stejné ploše čipu výrazně větší kapacitu (protože buňka DRAM obsahuje jen jeden tranzistor).
        - Nižší cena oproti SRAM.
        - Náboj uskladněný v kondenzátoru se samovolně a při každém čtení vybíjí (data se při čtení buněk vymažou).
        - Náboj v buňce se musí periodicky doplňovat (refresh).
        - To řeší řídící obvod paměti, který buňky periodicky čte (jednou za několik ms), přečtená data uloží do vyrovnávací paměti, ze které do buňek data zapíše zpět, tím se informace obnoví.
        - Nižší rychlost přístupu k datům oproti SRAM z důvodu jiné konstrukce buňky a kvůli nutnosti vkládání obnovovacích (refresh) cyklů.
        - Vyšší spotřeba energie oproti SRAM kvůli nutným obnovovacím cyklům, zejména v klidovém režimu, když se obsah paměti nemění.

Paměti ROM

Jedná se o paměti, jejichž obsah se po vypnutí napájení neztratí. V závislosti na možnostech zapisování do paměti a přepisování obsahu paměti se dělí do několika typů.

- **Mask ROM**
    - Obsah paměti je dán při jejím výrobním procesu a následně ho již nelze změnit – např. během života zařízení, ve kterém je tato paměť použita, nelze v zařízení aktualizovat program ani měnit read-only data.
    - Vyplatí se jen výroba velkého množství pamětí.
    - Nejstarší typ paměti.
- **One Time Programmable ROM (OTP ROM nebo OTP EPROM)**
    - Paměť je možné pouze jednou naprogramovat (z výroby je nenaprogramovaná).
    - Paměť v každé paměťové buňce obsahuje propojku (tenký drátek) nastavující bit na úroveň LOW, takže z výroby jsou v paměti bitově samé 0.

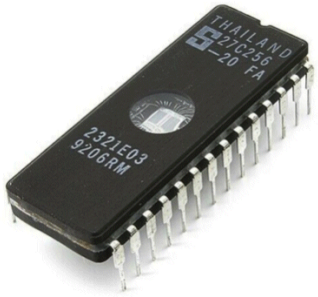
- Po vložení do programátoru se krátkým (20 us) pulzem o zvýšeném napětí (typicky 12 V) vybrané propojky (podle zamýšleného obsahu v paměti) přepálí, čímž se naprogramují jedničky (bity v úrovni HIGH).
- Je to levnější varianta paměti EPROM, která nemá sklíčko pro smazání jejího obsahu ultrafialovým světlem.



Obrázek 14 Paměť PROM o kapacitě 4x8 bitů (vysvětlení principu)

Příslušný řádek v paměti vybírá adresní dekodér 1 ze 4 (v obrázku 2:4 decoder, protože má 2 vstupy a 4 výstupy) podle logické úrovně přivedené z venku na adresní vodiče A1, A0. Podle logické kombinace na adresních vodičích A1, A0 se aktivuje jeden výstup dekodéru. Aktivuje v tomto případě znamená, že se na daném výstupu z dekodéru objeví úroveň LOW, na zbylých jeho výstupech je úroveň HIGH. Pokud je pojistka (Fuse link) v příslušném sloupci (bitu) nepřepálená, je tento bit v úrovni LOW, protože daný sloupec je přes diodu ukostřen a při čtení (signál Output enable = LOW) se na příslušném výstupu D_x objeví úroveň LOW. Pokud je pojistka přepálená, je bit v úrovni HIGH, což zajišťuje pull-up rezistor R a při čtení (signál Output enable = LOW) se na příslušném výstupu D_x objeví HIGH. Pamět na Obrázek 14 má 4 bajty a je tzv. organizovaná jako 4 x 8 (4 řádky a v každém 8 bitů). Zápis 32k x 8 by znamenal 2^{15} = 32768 řádků a v každém je 8 bitů, celkem 32768 B (32 kB). Zápis 32k x 16 by znamenal 2^{15} = 32768 řádků a v každém je 16 bitů, celkem 65536 B (64 kB).

- **Erasable Programmable ROM (EPROM)**
  - Odstraňuje nedostatek paměti PROM co do počtu programování.
  - Paměť EPROM lze programovat vícekrát po sobě (vydrží jednotky tisíc cyklů smazání/programování).
  - Před novým naprogramováním je nutné paměť nejprve smazat pomocí ultrafialového světla – paměti měly na sobě za tímto účelem skleněné okénko. Mazání může trvat od 5 do 30 minut.

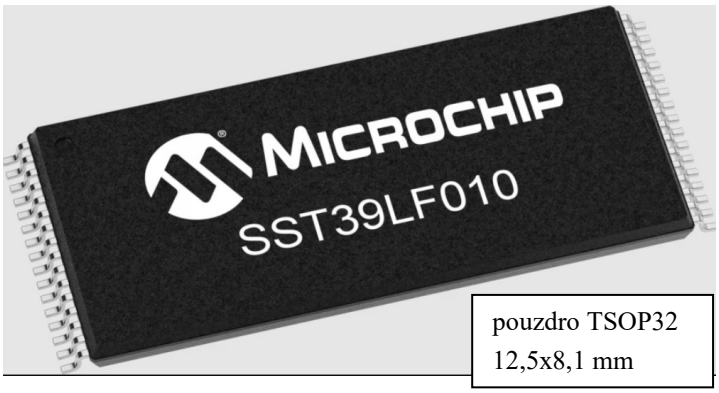


- **Electrically Erasable Programmable ROM (EEPROM)**
  - Na rozdíl od EPROM se paměť EEPROM maže elektricky, ne ultrafialovým světlem.
  - EEPROM není nutné vyndavat z desky plošného spoje (DPS), lze jí programovat na DPS.
  - Lze programovat jednotlivé bajty paměti (tj. zapisovat na libovolnou adresu).
  - Před zápisem hodnoty na nějakou adresu není nutné buňku předem smazat (nastavit v ní mazacím procesem log. 1). Smazání buňky před zápisem si obvody v paměti udělají automaticky samy.
  - Při zápisu do paměti se programují log. 0, při mazání se programují log. 1.
  - Zápis trvá výrazně delší dobu (jednotky milisekund) než čtení (jednotky až desítky mikrosekund).
  - Počet přepisů dnes obvykle 100 000 až 1 000 000.
  - Garantovaná doba uchování dat beze změny desítky let (dnes běžně 100 let).
  - Nižší hustota informace na plochu oproti FLASH, proto se vyrábí v malých kapacitách, jinak by čip vycházel moc velký.
  - Typická paměť pro uložení konfigurace zařízení, která se může během života zařízení měnit, dále pro uložení různých adaptačních hodnot měnících se v delším časovém horizontu během života zařízení apod.
- **FLASH**
  - Další generace pamětí, které přišly po EEPROM.
  - FLASH paměti jsou rychlejší, mají větší kapacitu (vyšší hustotu informace na jednotku plochy) a nižší spotřebu el. energie.
  - FLASH vydrží o řád nižší počet přepisů než EEPROM (10 000 až 100 000).
  - Garantovaná doba uchování dat beze změny desítky let (dnes běžně 100 let).
  - U některých typů FLASH lze číst a zapisovat libovolně po bajtech, ale před zápisem je nutné dotčenou část paměti nejprve smazat. Mazání probíhá vždy po blocích, nelze smazat jen jeden vybraný byte, což je určitá nevýhoda oproti EEPROM. Při mazání se buňky nastavují do logické 1, při zápisu se programují logické 0.

- EEPROM se hodí pro častější náhodné zapisování malého počtu dat, FLASH se hodí pro sekvenční zápis většího množství dat. EEPROM bude rychlejší než FLASH při zápisu malého množství dat, protože u FLASH před zápisem musí proběhnout mazání celého bloku, což bude časově náročnější. Rychlost čtení může být u obou typů pamětí shodná.
- U mikropočítačů je FLASH běžný typ paměti, ve které je uložen program vykonávaný procesorem.
- Dva typy FLASH pamětí (dle způsobu interního propojení paměťových buněk)
  - **NOR FLASH** – čtení a zápis po bajtech, náhodný přístup, vhodné pro mikroprocesorové systémy (uložení programu a konstantních dat, které se nemění).
  - **NAND FLASH** – čtení a zápis po stránkách velkých typicky 512 až 4096 bajtů, vyšší kapacita (GB u NOR vs. TB u NAND FLASH), rychlejší zápis a mazání než NOR FLASH, vhodné pro sekvenční čtení a ukládání objemnějších dat (paměťové karty, USB klíčenky, SSD).
- Paměti FLASH jsou obvykle pomalejší než paměti RAM.
- U mikrokontrolérů bude mít FLASH paměť na čipu několikanásobně větší kapacitu než paměť SRAM na čipu, protože FLASH má vyšší hustotu informace na plochu než SRAM.

Paralelní vs sériové paměti

Podle způsobu připojení paměti (připojovacího rozhraní), jde o počty vodičů.



pouzdro TSOP32
12,5x8,1 mm



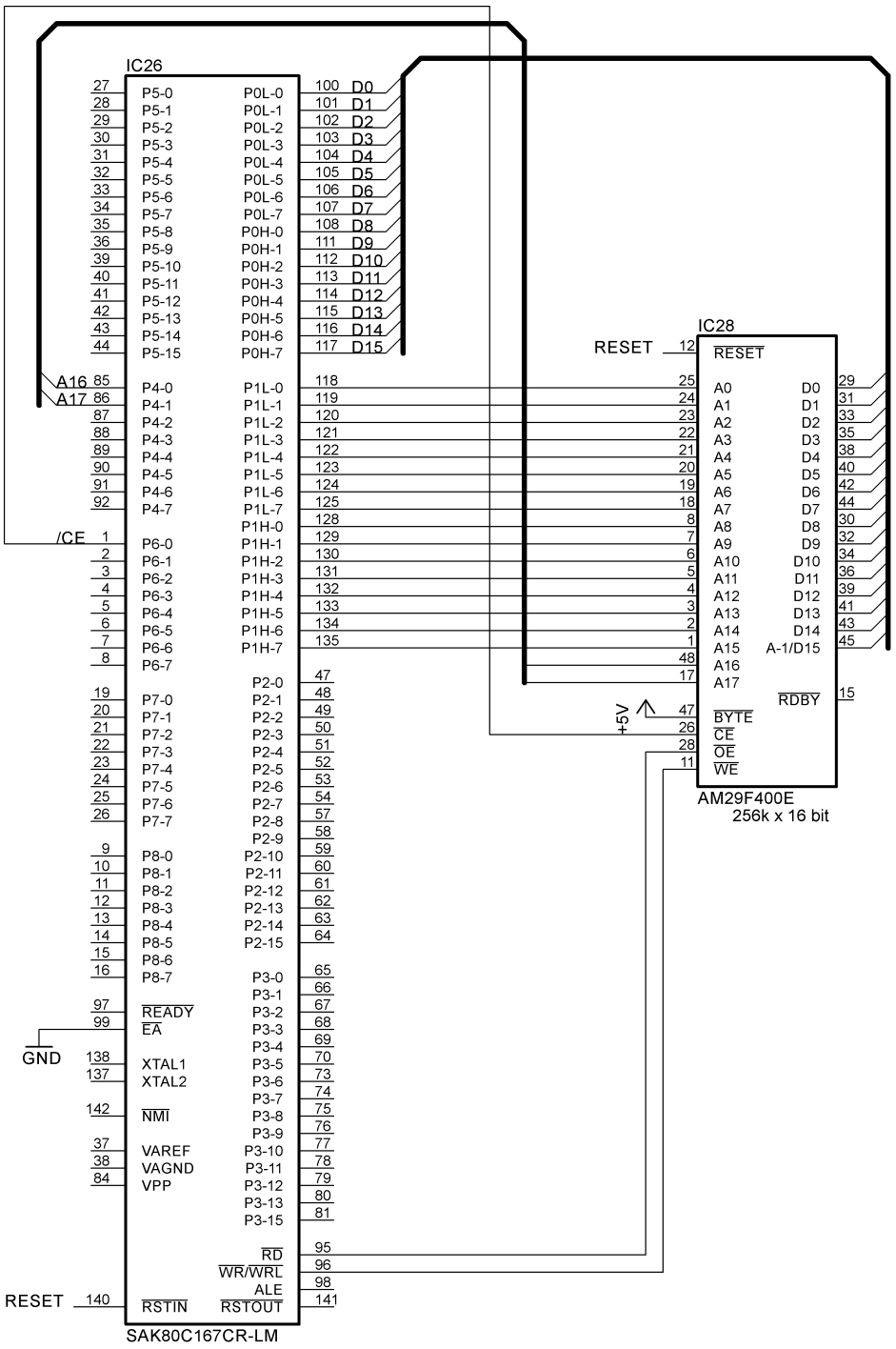
pouzdro SO8
4,9x3,9 mm

Obrázek 15 Paralelní (vlevo) a sériová FLASH (vpravo) o kapacitě 1 Mbit od výrobce Microchip

- **Paralelní paměti**
  - Přenos dat po více vodičích najednou – v každý okamžik se přenáší skupina bitů.
  - Potřebujeme tyto počty vodičů pro připojení paměti:
    - N adresových vodičů A_{N-1} až A_0
    - M datových vodičů D_{M-1} až D_0
    - Několik řídících vodičů (např. pro řízení směru přenosu dat).
  - Kapacita paměti bude 2N x M bitů, např. při N=16, M=8: 216 x 8 = 65 535 B (64 kB). Při každém přístupu do paměti se po datových vodičích najednou přenese 8 bitů (jeden byte).
  - Výhody:
    - Vysoká přenosová rychlost.
    - Přímé napojení na sběrnici procesoru.
  - Nevýhody:
    - Velké pouzdro s množstvím nožiček, zabírá plochu na desce plošného spoje (DPS), spotřeba velkého množství pinů na procesoru/MCU pro připojení paměti.
    - Vyšší spotřeba el. energie.
    - Možné problémy s přeslechy signálu mezi paralelně vedenými vodiči blízko sebe (kapacitní vazba) - musí být krátké spoje na DPS (komplikovanější návrh DPS), krátké připojovací kabely.
    - Vyšší cena.
  - Typický zástupce této kategore jsou DRAM paměťové moduly v PC.
  - Pro malé jednočipové mikropočítače se tento způsob připojení externích pamětí nehodí.

Na Obrázek 16 je znázorněno připojení externí paralelní FLASH paměti AM29F400 o velikosti 256k x 16 (4 Mbit = 512 kB) k 16-bit mikrokontroléru řady 80C167. Paměť obsahuje program, který se má v MCU vykonávat. Varianta MCU na obrázku není vybavena žádným typem vnitřní paměti ROM, proto je externí paměť programu nezbytná.

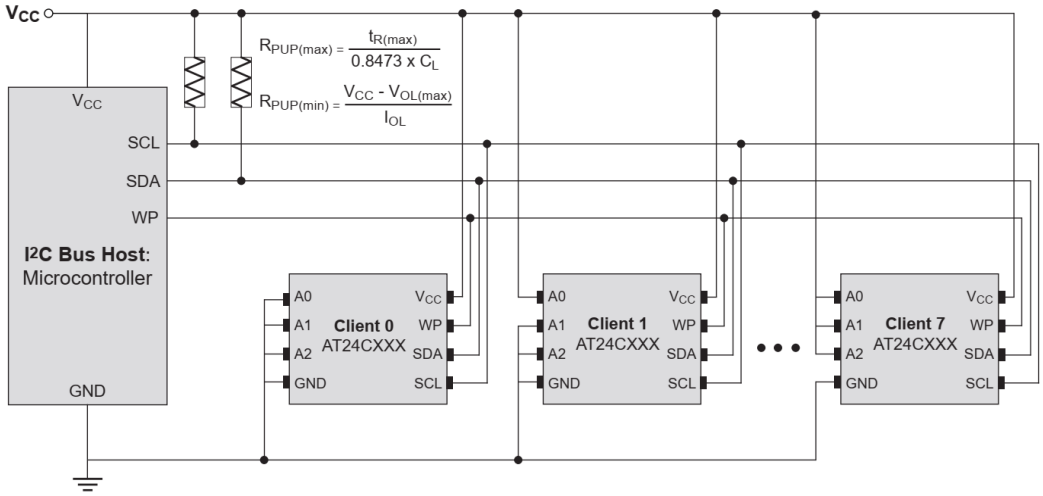
Paměť je připojena k MCU přímo přes 18-bit adresovou sběrnici (vodiče A0 až A17) a 16-bit datovou sběrnici (vodiče D0 až D15). Řídící signály jsou /CE (chip enable – aktivace paměti), /OE (otuput enable – signál čtení, zdrojem signálu je MCU), /WE (write enable – zdrojem signálu je MCU).



Obrázek 16 Připojení paralelní FLASH 256k x 16 k MCU 80C167

- **Sériové paměti**
  - Přenos jednotlivých bitů postupně jeden za druhým typicky po jednom vodiči v jednom směru. Počet připojovacích vodičů neroste s rostoucí kapacitou paměti.
  - Existují standardní komunikační rozhraní, např.:
    - SPI (3 vodiče + chip select), max. rychlost obvykle 10-ky Mb/s
    - I2C (2 vodiče), max. rychlosti 100 kb/s, 400 kb/s, 3,4 Mb/s
  - Výhody:
    - Malá velikost pouzdra (např. pouzdro jen s 8 vývody) – zabírá menší plochu na DPS, snadnější tahání spojů na DPS, malá spotřeba nožiček na MCU pro připojení paměti.
    - Nižší spotřeba el. energie.
  - Nevýhody:
    - Obecně menší přenosová rychlost než u paralelních pamětí.
  - Vhodný a běžně používaný způsob připojení externích pamětí k malým jednočipovým mikropočítačům.
  - Paměti RAM, EEPROM i FLASH se sériovým rozhraním.

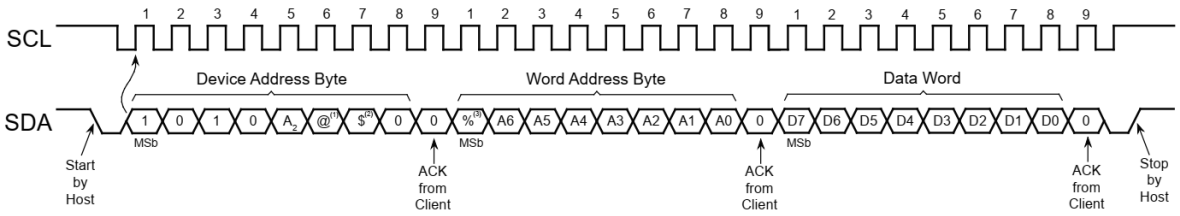
Na Obrázek 17 je příklad připojení EEPROM řady AT24C od firmy Microchip k obecnému mikrokontroléru pomocí sériové sběrnice I2C. Paměť se vyrábí ve velikostech 128 B (128x8) u AT24C01, 256 B (256x8) u AT24C02, 512 B (512x8) u AT24C04 a 1024 B (1024x8) u AT24C08. Vodiči SCL přenáší obdélníkový hodinový signál pro taktování komunikace, který generuje MCU, když chce z paměti číst nebo do ní zapisovat. Vodič SDA slouží pro přenos dat - při čtení paměti ve směru paměť→MCU, při zápisu do paměti ve směru MCU→paměť. Vodič WP (write protect) ovládaný z MCU je pouze doplňkový a není nutné aby byl zapojen (pin WP uvnitř paměti obsahuje interní pull-down rezistor zajišťující, že do paměti je umožněn zápis i při nezapojení pinu). Pokud je WP v úrovni LOW, je možné do paměti zapisovat, pokud je v HIGH, do paměti není možné zapisovat.



Obrázek 17 EEPROM připojená k MCU přes sériovou sběrnici I2C

Komunikaci na sbwěrnici I2C vždy zahajuje MCU (Master) tak, že na lince SDA nejprve vystaví adresu obvodu, se kterým chce na sběrnici I2C komunikovat (tím obvod na sběrnici vybere ke komunikaci). Současně s přenosem adresy sdělí, zda bude do paměti zapisovat nebo z ní číst (rozhoduje úroveň bitu 0 v Device Address Byte).

Při zápisu jednoho bajtu do EEPROM následně MCU na lince SDA vystaví adresu, na kterou chce datový byte zapsat (Word Address Byte). Následuje vystavení datového bajtu z MCU pro zápis do paměti (Data Word).

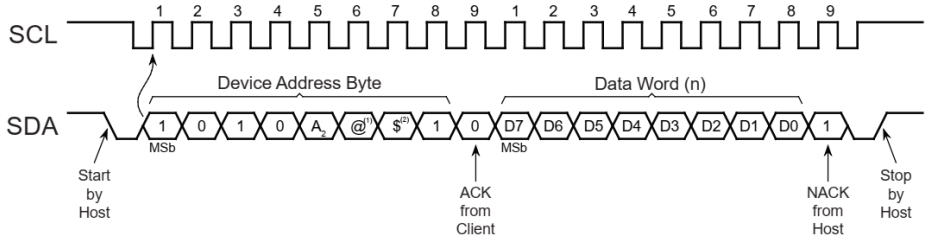


Notes:

1. For the AT24C01C, AT24C02C and AT24C04C, the @ indicates the A1 hardware client address bit. For the AT24C08C, the @ indicates the A9 word address bit.
2. For the AT24C01C and AT24C02C, the $ indicates the A0 hardware client address bit. For the AT24C04C and AT24C08C, the $ indicates the A8 word address bit.
3. For the AT24C01C, the % indicates a “don’t care” bit. For the AT24C02C, AT24C04C and AT24C08C, the % indicates the A7 word address bit.

Obrázek 18 Zápis jednoho bajtu do sériové EEPROM

Čtení jednoho bajtu z paměti může probíhat bud z aktuální adresy v paměti nebo z jiné adersy adresy. V prvním případě je situace jednodušší, do EEPROM stačí pro zahájení čtení odeslat Device Address Byte s příznakem pro čtení, generovat dál hodiny a číst hodnoty bitů, které na sběrnici následně vystaví paměť (Data Word).



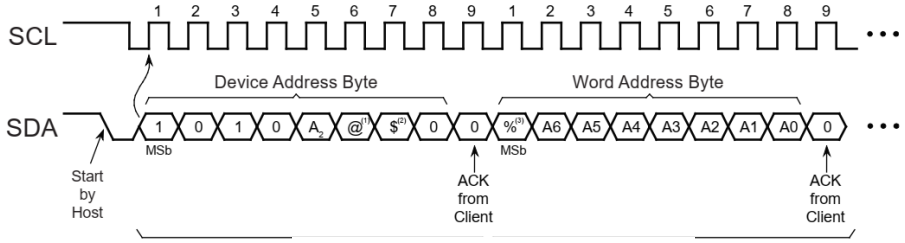
Notes:

1. For the AT24C01C, AT24C02C and AT24C04C, the @ indicates the A1 hardware client address bit. For the AT24C08C, the @ indicates the A9 word address bit.
2. For the AT24C01C and AT24C02C, the $ indicates the A0 hardware client address bit. For the AT24C04C and AT24C08C, the $ indicates the A8 word address bit.

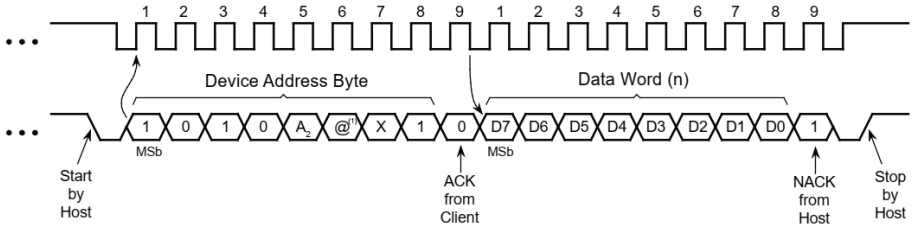
Obrázek 19 Čtení z aktuální adresy ze sériové EEPROM

Paměť umožnuje i zápis celé jedné stránky najednou, kdy se do paměti nejprve přenese max. 8 nebo 16 datových bajtů (= velikost jedné stránky) dle velikosti použité paměti a pak se spustí jejich zápis.

Čtení z náhodné adresy. Při čtení z jiné adresy, než která odpovídá aktuální hodnotě adresního čítače v EEPROM, je nejprve nutné v EEPROM nastavit adresu (Word Address Byte) a teprve pak z ní vyčíst hodnotu (Data Word). Na začátku komunikace tak přibudou dva další bajty pro přenos (celkem 4 bajty se během komunikace budou přenášet), nejprve se z MCU do paměti přenese adresa a potom se zní vyčte hodnota.



nastavení adresy pro čtení



vyčtení hodnoty z adresy

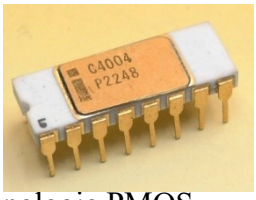
Notes:

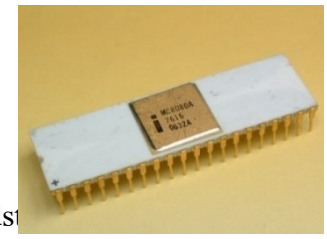
1. For the AT24C01C, AT24C02C and AT24C04C, the @ indicates the A1 hardware client address bit. For the AT24C08C, the @ indicates the A9 word address bit (R/$\overline{W}$ = 0) or a “don't care” bit (R/$\overline{W}$ = 1).
2. For the AT24C01C and AT24C02C, the $ indicates the A0 hardware client address bit. For the AT24C04C and AT24C08C, the $ indicates the A8 word address bit (R/$\overline{W}$ = 0) or a “don't care” bit (R/$\overline{W}$ = 1).
3. For the AT24C01C, the % indicates indicates a “don’t care” bit. For the AT24C02C, AT24C04C and AT24C08C, the % indicates the A7 word address bit.

Obrázek 20 Čtení z náhodné adresy ze sériové EEPROM

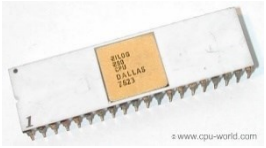
Paměť umožnuje i sekvenční (postupné) čtení bajtů po sobě, kdy se po vyčtení každého z nich automaticky uvnitř paměti inkrementuje adresní čítač. Tímto způsobem je možné vyčíst bajtů kolik chceme.

3 Historický vývoj





- I. generace mikroprocesorů
  - Počátek 70. let 20. století.
  - Intel 4004 – 4-bit procesor, technologie PMOS
- II. generace mikroprocesorů
  - 1974
  - Intel 8080 – 8-bit procesor, technologie NMOS, podst
- III. generace mikroprocesorů
  - 1976
  - Zilog Z80 – nástupce Intel 8080, 8-bit, rozšířená instrukční sada, velmi rozšířený, přinesl boom domácích počítačů, např. Sinclair ZX Spectrum, Nintendo Game Boy, herní automaty, dále kalkulačky, tiskárny apod.



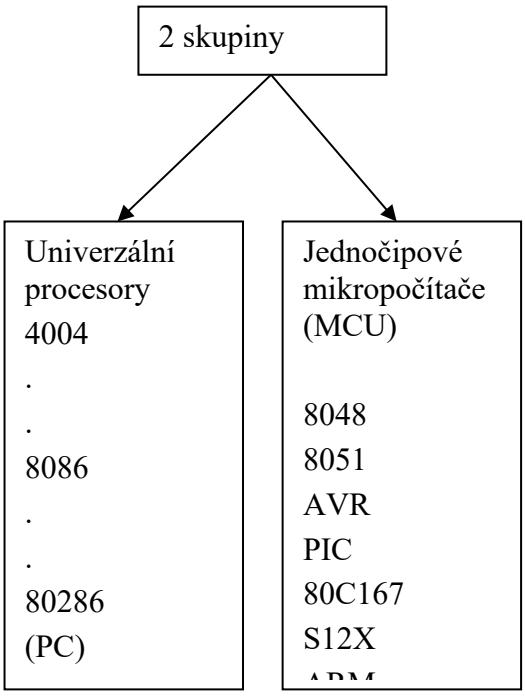
- IV. generace mikroprocesorů
  - Vznik jednočipových mikropočítačů (mikrokontrolérů, nebo-li MCU, µC).
  - MCU (mikrokontrolér) = CPU (procesor) + paměť + integrované periferie (časovače, A/D, komunikační linky apod.) na jednom čipu. Je to kompletní mikropočítač na jednom čipu.
  - MCU jsou určeny pro jednoúčelové aplikace, hardware je orientován na styk s reálným prostředím.
  - Intel 8048 – 8-bit, rok 1976, později nahrazen populárnějším 8051
  - Další výrobci MCU: Texas Instruments, Analog Devices, Infineon, Freescale, Renesas, Microchip, Atmel, Dallas, Phillips atd.











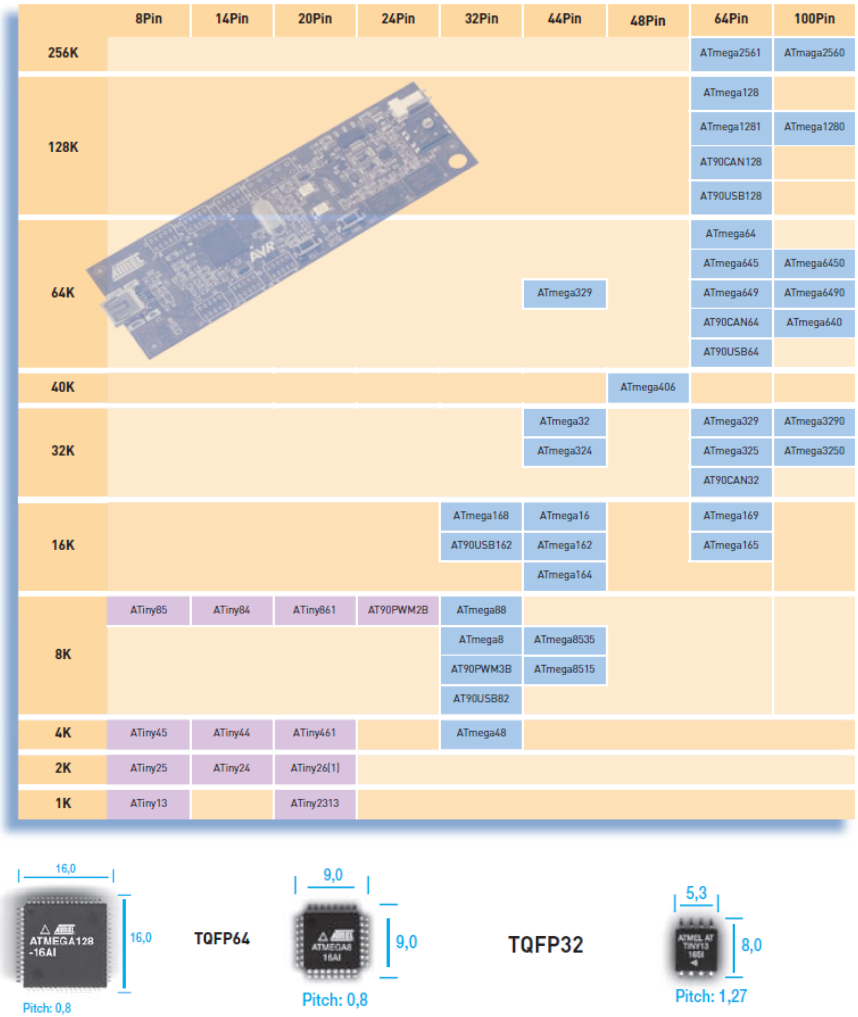


4 MCU AVR – hlavní rysy

Funkci mikrontroléru a jeho programování si v tomto předmětu ukážeme na konkrétním typu. Bude to 8-bit mikrokontrolér AVR vyvinutý v druhé polovině 90. letech 20. století firmou Nordic VLSI. Design byl prodán firmě Atmel, která mikrokontroléry uvedla na trh. Jedná se o úspěšný a oblíbený mikrokontrolér v profi i hobby sféře (viz Arduino) používaný dodnes.

- 8-bitový mikrokontrolér
- Max. frekvence oscilátoru 16-20 MHz. Max. výkon 16-20 MIPS.
- Napájení 5 V nebo 1,8 až 5 V (low-voltage varianty s nižší taktovací frekv.).
- RISC architektura
    - Instrukční soubor obsahuje pouze jednoduché a nejčastěji využívané instrukce (cca 135 instrukcí).
    - Většina instrukcí je stejně dlouhá a mají pevný formát – to umožňuje jejich jednodušší dekódování – zrychlení.
    - Pipelining – zřetězení instrukcí. Dokončování instrukcí v jednom taktu.
    - AVR má 32 8-bitových pracovních registrů přímo napojených na ALU, každý z nich může obsahovat data i adresy, obecně operandy – rozdíl oproti 8051, což je procesor akumulátorového typu.
- Harvardská architektura – oddělené paměťové prostory pro kód a data
- Velikost interních pamětí
    - až 256 kB interní paměť programu FLASH (životnost 10 000 zápisů)
    - až 8 kB interní paměť dat (až 64 kB externí paměť dat) SRAM
    - až 4 kB interní EEPROM (životnost 100 000 zápisů)
- Šíře instrukčního slova 16 bitů (šířka paměti programu 16 bitů)
    - Výhoda - zrychlení načtení instrukcí (většina instrukcí načtena během 1 strojového cyklu), instrukce jsou 16 a 32 bitové
    - Nevýhoda - zvětšení požadavků na paměť (nevadí)
- Strojový cyklus trvá jeden takt oscilátoru, není použita žádná dělička jako např. u 8051, kde 1SC = 12/Fosc.
- Implementace jednoduchého zřetězení instrukcí – pipelining
    - Umožňuje aby instrukce typu registr-registr byly vykonány během 1 taktu – tím se dosahuje výkonu 1 MIPS na 1 MHz.
    - Výkon oproti 8051 až 12x větší při stejném kmitočtu.
- Architektura vhodná pro programování v jazyce C.
- Vestavěné periferie (dle typu):
    - Časovače (vč. PWM a záchytné jednotky), watchdog.
    - 10-bit A/D převodník
    - Analogový komparátor
    - Sběrnice: USART, SPI, I2C, CAN, USB

- Různé typy AVR dle velikosti pouzdra, velikosti paměti, vybavenosti periferiemi:
  - Tiny
  - Mega
  - CAN
  - USB
  - LCD
  - PWM (motor control)



	8Pin	14Pin	20Pin	24Pin	32Pin	44Pin	48Pin	64Pin	100Pin
256K								ATmega2561	ATmega2560
128K								ATmega128	
								ATmega1281	ATmega1280
								AT90CAN128	
								AT90USB128	
64K								ATmega64	
								ATmega645	ATmega6450
						ATmega329		ATmega649	ATmega6490
								AT90CAN64	ATmega640
								AT90USB64	
40K							ATmega406		
32K						ATmega32		ATmega329	ATmega3290
						ATmega324		ATmega325	ATmega3250
								AT90CAN32	
16K					ATmega168	ATmega16		ATmega169	
					AT90USB162	ATmega162		ATmega165	
						ATmega164			
8K	ATiny85	ATiny84	ATiny861	AT90PWM2B	ATmega88				
					ATmega8	ATmega8535			
					AT90PWM3B	ATmega8515			
					AT90USB82				
4K	ATiny45	ATiny44	ATiny461		ATmega48				
2K	ATiny25	ATiny24	ATiny26[1]						
1K	ATiny13		ATiny2313						

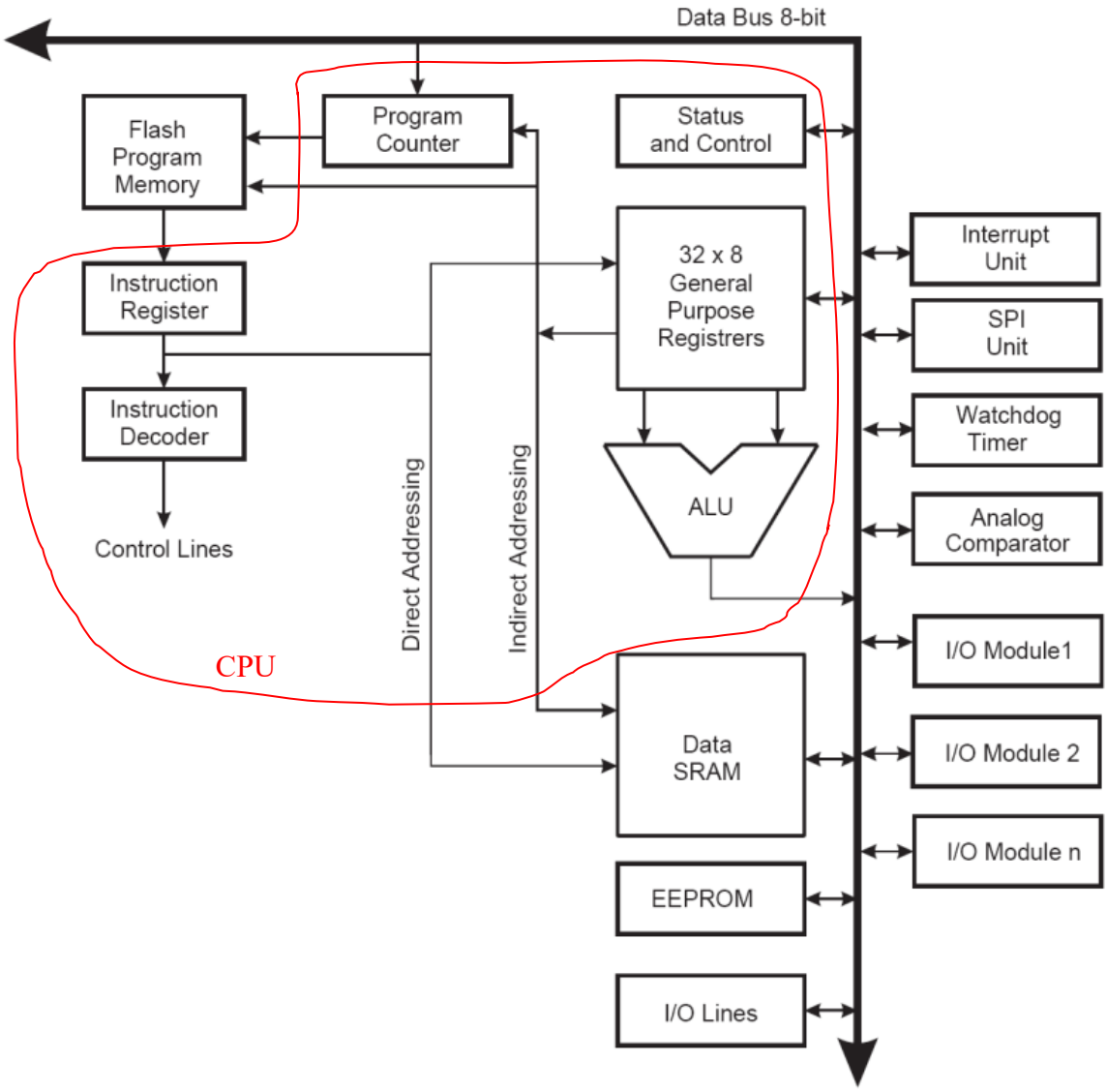
- AVR je použito v oblíbeném Arduinu – Nano, Micro, UNO, Leonardo, MEGA2560…



5 Architektura MCU AVR

Vlastnosti:

- harvardská architektura – oddělená paměť dat od paměti programu
- RISC CPU
- využití pipeliningu při práci s instrukcemi (načtení, dekódování, vykonání)



5.1 Vnitřní struktura procesoru (CPU)

ALU – aritmeticko logická jednotka

Je kombinační obvod, kterí vykonává aritmetické a logické operace nad vstupními daty. Podle šířky zpracovávaných dat je procesor zařazen mezi 8-bit, 16-bit, atd. procesory.

Pracovní registry (General Purpose Registers)

Data, se kterými ALU pracuje jsou uložena v těchto registrech. Výsledky výpočtů se ukládají do těchto registrů.

Programový čítač (Program Counter)

Tento registr slouží pro uložení adresy instrukce, která má být načtena z paměti programu. Při načítání instrukcí je automaticky inkrementován. V případě instrukce skoku je jeho obsah přepsán novou adresou od které se bude program vykonávat.

Registr instrukcí (Instruction register)

Obsahuje aktuální právě vykonávanou instrukci. Podle jeho obsahu řídí dekodér instrukcí činnost procesoru.

Dekodér instrukcí (Instruction decoder)

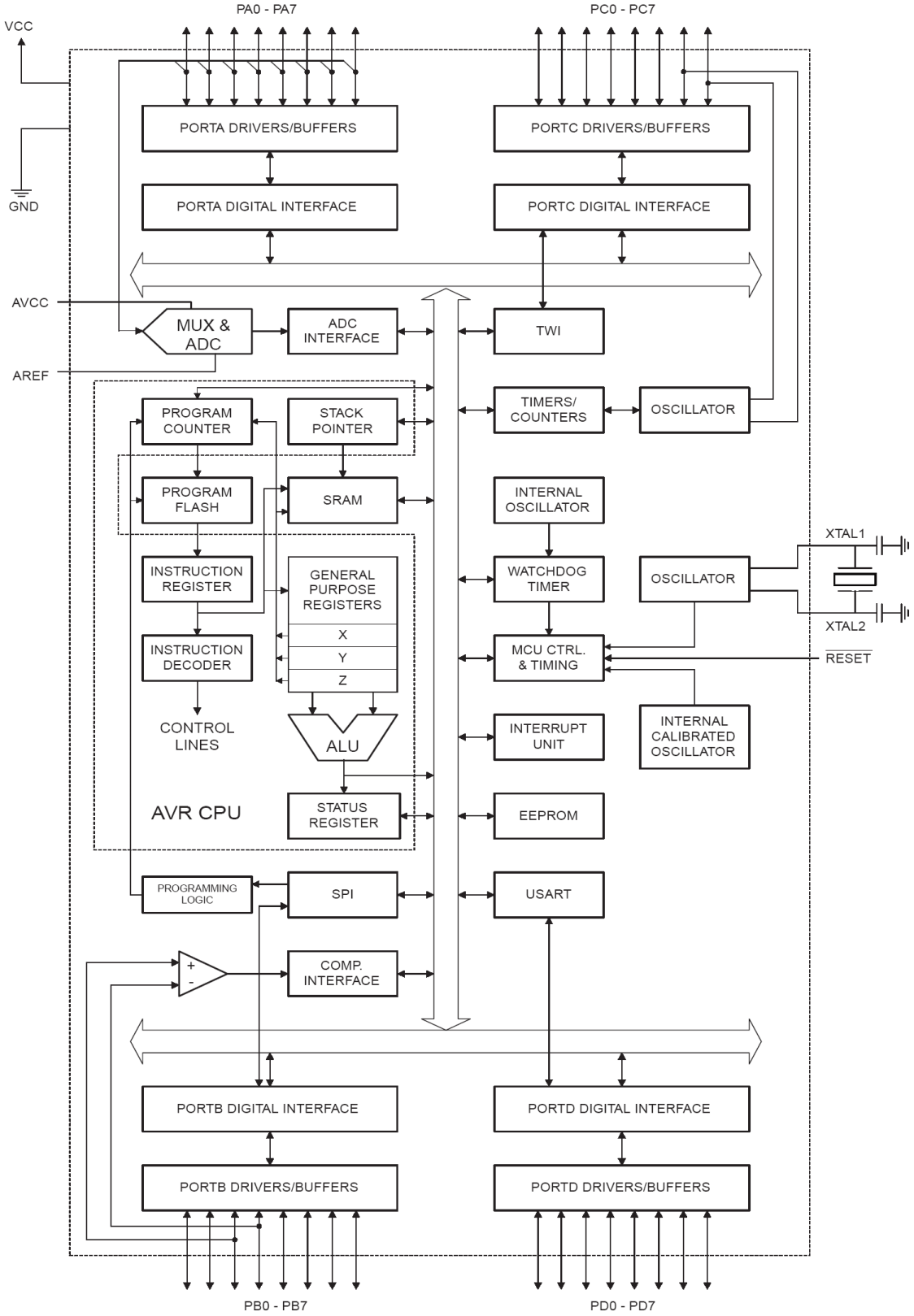
Dekóduje obsah resigstru instrukcí.

Řadič

Řadič (na obrázku není zobrazen, pouze náčrtek „Control Lines“) na základě dekódované instrukce řídí činnost vnitřních částí procesoru i vnější sběrnice. Jeho činnost je řízena generátorem hodin (externí krystal, keramický rezonátor, RC oscilátor apod.).

Další registry

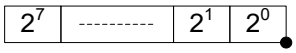
- Registry pro obsluhu vestavěných periferií a I/O portů.
- **Stavové slovo procesoru** – registr s příznaky, které se nastavují podle výsledku aritmetické nebo logické operace provedené v ALU.
- **Stack pointer** (ukazatel na datový zásobník) – především slouží k dočasné úschově návratových adres podprogramů a obsahu registrů.



Obrázek 21 Blokové schéma MCU ATmega32

ALU

- Řádová mřížka:



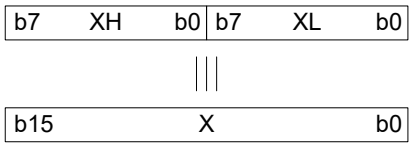
  - $L = 8$ bitů
  - $Z = N = 256$ (kapacita čísla), $\varepsilon = 1$ (nejmenší krok)
- Pouze celá čísla, omezená podpora pro práci se zápornými čísly (ve 2D).
- Pracuje v přímém spojení s 32 univerzálními registry.
- Podporuje aritmetické a logické operace mezi registry, registrem a konstantou nebo pracuje se samotným registrem.
- ALU operace jsou rozděleny na tři druhy
  - aritmetické
  - logické
  - bitové
- Výsledek operace v ALU ovlivňuje bity ve stavovém registru SREG (jako PSW u 8051)
- AVR typu ATmega obsahují **hardwarovou násobičku** (násobení pak trvá pouze 2 takty)

Pracovní registry

	7 … 0	Addr.	
General Purpose Working Registers	R0	$00	
	R1	$01	
	R2	$02	
	…		
	R13	$0D	
	R14	$0E	
	R15	$0F	
	R16	$10	
	R17	$11	
	…		
	R26	$1A	X-register Low Byte
	R27	$1B	X-register High Byte
	R28	$1C	Y-register Low Byte
	R29	$1D	Y-register High Byte
	R30	$1E	Z-register Low Byte
	R31	$1F	Z-register High Byte

- Registry jsou mapovány do začátku datové paměti.
- Obsahem registru mohou být data nebo adresa.

- Téměř všechny instrukce pracující s registry mají jednocyklový přístup (existují vyjímky). Některé instrukce mají přístup jen k R16 až R31.
- Registry R26 až R31 mohou sloužit jako tři 16-bitové registry X, Y, Z použitelné pro **nepřímé adresování** (způsoby použití budou uvedeny v jedné z dalších kapitol)
    - příklad registr X



Stavový registr

- 8-bitový registr SREG
- obsahuje příznaky (bity), které jsou modifikovány v závislosti na výsledku výpočetní operace provedené v ALU – stejný princip jako PSW u 8051
- hodnoty jedotlivých bitů lze testovat a využít např. pro podmíněné větvení programu, skoky apod. (např. instrukce BREQ, BRNE, BRTS, BRTC atd.)
- existují instrukce SEx a CLx pro nastavování jednotlivých bitů do 1 či do 0 (např. SEI, CLI, SET, CLT atd.)

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

BIT7 - I – Global Interrupt Enable

- Globální povolení všech přeušení (I = 1). Globální zákaz všech přerušení (I = 0).
- Povolení jednotlivých přerušení se provádí v příslušných registrech periferií.
- Po vyvolání přerušení je bit I nulován, po vykonání instrukce RETI (návrat z obsluhy přerušení) je opět nastaven na jedničku.
- Lze ho nastavovat i softwarově pomocí instrukcí SEI (I = 1) a CLI (I = 0)

BIT6 – T – Bit Copy Storage

- Uživatelsky nastavitelný bit pomocí instrukcí BST Rr,b (kopie určitého bitu pracovního registru Rr do bitu T) a BLD Rd,b (kopie bitu T do určitého bitu pracovního registru Rd).

BIT5 – H – Half Carry Flag

- Indikuje přenos mezi dolní a horní čtveřicí bitů (mezi bitem b_3 a b_4), využití při počítání s BCD čísly pro dekadickou korekci.
- Pokud dojde při binárním sčítání k přenosu mezi dolní a horní čtveřicí bitů nebo výsledek obsahuje nižší číslici mimo rozsah 0 až 9, tak se musí k výsledku přičíst číslo 0x06 (provést tzv. dekadickou korekci).
- Stejně tak pokud dojde při binárním sčítání k nastavení příznaku přetečení ALU (příznak Carry), tj. při přenosu mezi bitem b7 a (neexistujícím) bitem b8, nebo pokud výsledek obsahuje vyšší číslici mimo rozsah 0 až 9, musí se k výsledku přičíst číslo 0x60 (provést tzv. dekadickou korekci).
- Př. 1: Přenos z dolního do horního nibblu

```
bit č.7654 3210
      0010 0101     25 v BCD kódu
     +0100 1000     48 v BCD kódu
     ----------
      0110 1101     0x6D – špatně (není číslo v BCD)
     +0000 0110     +6 (dekadická korekce)
     ----------
      0111 0011     73 v BCD kódu – správně
```

- Př. 2: V dolním nibblu výsledku je číslo větší než 9

```
bit č.7654 3210
      0011 1001     39 v BCD kódu
     +0100 1000     48 v BCD kódu
     ----------
      1000 0001     81 v BCD kódu – špatně
     +0000 0110     +6 (dekadická korekce)
     ----------
      1000 0111     87 v BCD kódu – správně
```

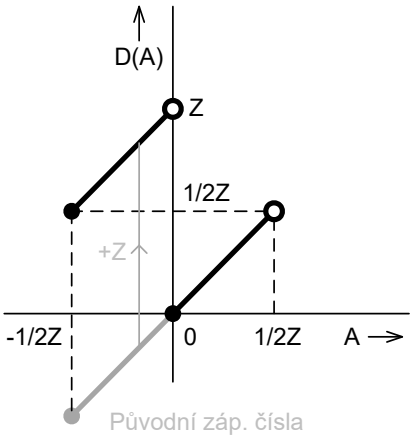
BIT4 – S – Sign Bit, S = N xor V

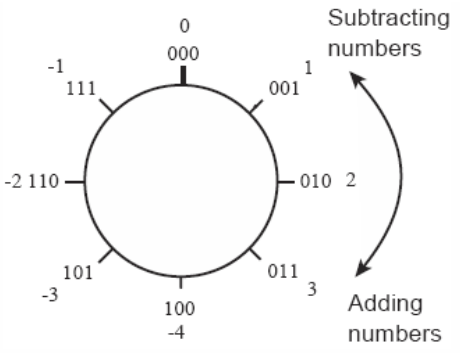
- Tento bit je výlučným součtem (XOR) mezi příznakem záporného výsledku N a příznakem přetečení V. Používá se pro znaménkové testy, při porovnávaní.

N	V	S = N xor V
0	0	0
0	1	1
1	0	1
1	1	0

BIT3 – V – Two's Complement Overflow Flag

- Příznak přetečení čísla se znaménkem (číslo je mimo rozsah zobrazitelných hodnot). Číslo se znaménkem je uloženo ve dvojkovém doplňku. Pozor, nejedná se o přetečení ALU – to je indikováno příznakem C (carry)!
- Princip dvojkového doplňku (demonstrováno na 3-bitovém čísle) – opakování.





Z – modul (počet různých hodnot n-bitového čísla), $Z = 2^n$, $n = 3 \rightarrow Z = 8$

A – celé číslo (kladné nebo záporné), které se má uložit ve dvojkovém doplňku

D(A) – dvojkový doplněk čísla A

D(A) = A pro kladná čísla A

D(A) = A + Z pro záporná čísla A

Rozsah 3-bit čísla ve 2D: ⟨-4 ; 3⟩

Obecně $\langle -2^{n-1} ; 2^{n-1} - 1 \rangle$ kde n – počet bitů čísla.

Pro 8-bit číslo ve 2D: ⟨-128 ; 127⟩

- Příznak V se vypočte jako V = C xor $C_{b6 \rightarrow b7}$. Kde $C_{b6 \rightarrow b7}$ indikuje, že nastal přenos z bitu b_6 do bitu b_7 (bity jsou číslovány od 0).
- Mohou nastat 2 varianty, při kterých se příznak V nastaví do 1:
  - Součet dvou kladných čísel je záporný:
    - C = 0, $C_{b6 \rightarrow b7}$ = 1, tj. nedojde k přenosu z bitu b_7 do bitu b_8 (C = 0), ale zároveň dojde k přenosu z bitu b_6 do bitu b_7
  - **Příklad:** 127 + 1 = 128 (uvažujeme 8-bitová čísla)

	Bit č. 7654 3210
A(127)	0111 1111
+B(1)	+0000 0001
=128 (-128 ve 2D)	1000 0000

  - Došlo k přenosu mezi dvěma nejvýznamnějšími bity výsledku ($C_{b6 \rightarrow b7}$ = 1), došlo ke změně znaménka výsledku ⇒ příznak V se nastaví do 1.

- Součet dvou záporných čísel je kladný:
  - C = 1, $C_{b6 \rightarrow b7}$ = 0, tj. dojde k přenosu z bitu b_7 do bitu b_8 (C = 1), ale zároveň nedojde k přenosu z bitu b_6 do bitu b_7.
- **Příklad:** -128 - 1 = -129 (uvažujeme 8-bitová čísla)

```
                          Bit č. 8 7654 3210
 A(-128)                        1000 0000
+B(-1)                         +1111 1111
=127 (127 ve 2D)              1 0111 1111
                                  ↓
                                 C = 1
```

- Došlo k přetečení výsledku – k přenosu z nejvýznamnějšího bitu (b_7) výsledku do (fiktivního) bitu b_8 (C = 1), došlo ke změně znaménka výsledku ⇒ příznak V se nastaví do 1.

BIT2 – N – Negative Flag

- Příznak záporného výsledku aritmetických nebo logických operací.
- N = 1, když je nejvyšší bit výsledku = 1 (ve dvojkovém doplňku znamená, že číslo je záporné).

```
                            Bit č. 7654 3210
 A(127)                           0111 1111
+B(1)                            +0000 0001
=-128 ve 2D                      1000 0000
                                     ↓
                                   N = 1
```

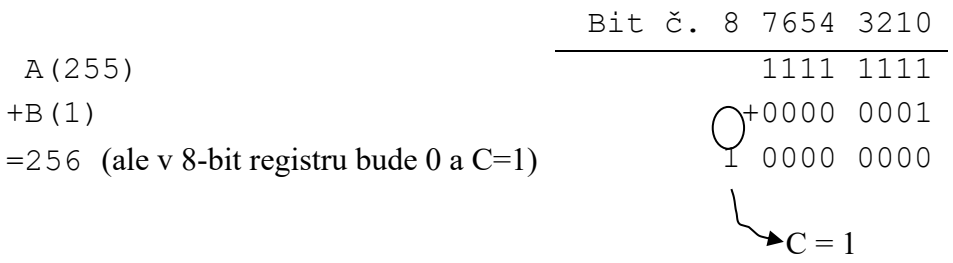
BIT1 – Z – Zero Flag

- Příznak Z indikuje nulový výsledek aritmetických nebo logických operací.

```
                          Bit č. 8 7654 3210
 A(255)                         1111 1111
+B(1)                          +0000 0001
=0 a C=1                      1 0000 0000
                                  ↓    ↓
                                 C=1  Z=1
```

BIT0 – C – Carry Flag

- Přetečení výsledku při aritmetických a logických operacích, indikuje, že výsledek je větší než 255.
- Používá se pro detekci překročení rozsahu neznaménkového čísla.
- Např. při součtu 255 + 1 se C nastaví do 1 a v registru bude výsledek 0, protože číslo 256 přeteklo kapacitu 8-bit registru.



5.2 CISC vs. RISC procesory

5.2.1 CISC

- CISC = Complex Instruction Set Computer
- vývojově starší, dnes postupný ústup ze slávy
- přístup: „pomoci konstrukci překladače z VPJ co nejpodobnějšími instrukcemi s příkazy VPJ“:
  - volání podprogramů
  - cykly (např. 8051 – `djnz` = Decrement Jump if Not Zero)
  - komplexní větvení (např. 8051 – `cjne` = Compare and Jump if Not Equal)
  - výpočty s operandy v paměti během jednoho cyklu
    - např. 8051 – add ACC, adr `// ACC = ACC + obsah adresy přístup do paměti „zadrátovaný“ do instrukce add, neoddělený do jiné instrukce`
    - `ACC` je tzv. akumulátor – pro všechny ALU instrukce je ACC jeden z operandů instrukce a zároveň je to cíl pro uložení výsledku
- Základní znaky:
  - komplikovaný instrukční soubor (100ky instrukcí) – snaha přiblížit se VPJ
  - složité instrukce
    - složitý návrh čipu → vysoká cena
    - zpracování instrukce → nutné více kroků řadiče → dlouhý instrukční cyklus (viz dále)
    - **kratší program** (nutné použít méně instrukcí) → **méně přístupů k paměti**, **úspora nákladů na µP systém** (60. léta 20 stol.) → ale ne vždy vede ke zrychlení běhu programu
  - nestejně dlouhé instrukce – obtížně proveditelné zřetězené (proudové) vykonávání instrukcí (pipelining)
- Zástupci: 8051, Motorola 68000, Intel x86 (do 80486)

5.2.2 RISC

- RISC = Reduced Instruction Set Computer
- Jak zvýšit výkon procesoru? Rok 1970 u IBM – výzkum, výsledky:
  - velký počet instrukcí = složité optimalizace
  - programátor ani překladače nejsou schopni využít všechny instrukce u CISC CPU
  - až 50 % instrukcí nelze využít optimálně
  - v „průměrných“ programech:
    - většina „výkonných instrukcí“ (aritmetika, cykly – obvodově nejsložitější) je v kódu velmi málo zastoupena
    - nejčastější instrukce – přesuny, skoky, porovnání
- charakteristika:
  - malý počet jednoduchých instrukcí (10ky až cca 130, ale najdou se výjimky i s ještě větším počtem instrukcí)
  - kratší doba provedení jedné instrukce než u CISC
  - instrukce jsou stejně dlouhé
  - lze realizovat zřetězené (překrývané) zpracování instrukcí, tzv. pipelining (snaha o to, aby každý strojový cyklus byla dokončena jedna instrukce)
  - velký počet pracovních registrů → 32 a více (za účelem omezení „pomalých“ přístupů do paměti)
  - malý počet tranzistorů na čipu CPU oproti CISC
  - menší spotřeba el. energie
  - integrace více periférií
  - základní filozofie → „load / store architektura“
    - ALU pracuje pouze nad pracovními registry (GPR)
    - Operandy uložené v paměti musejí být do pracovních registrů nejprve nahrány pomocí load / store instrukcí
    - Takto nelze (CISC):
      ```
      add ACC, adr   // ACC = ACC + obsah adr
      ```
    - Takto se musí (RISC):
      ```
      lds R0, adr1   // R0 = obsah adr1
      lds R1, adr2   // R1 = obsah adr2
      add R0, R1     // R0 = R0 + R1
      ```
- RISC je dnes nejrozšířenější architektura CPU
  - 8-bit MCU – Atmel AVR, Michrochip PIC
  - 32-bit MCU – ARM (Acorn Risc Machine, až 75% trhu – Smart Phones…), různí výrobci (STM, NXP, Cypress, Renesas, Infineon, TI, Apple, Qualcom, Samsung, …)
  - „CPU pro PC“

- x86 od Intel Pentium výše (instrukční soubor CISC, ale vnitřní architektura je RISC, použití mikrokódu na rozložení složitých instrukcí na jednoduché)
- IBM PowerPC (Apple/Macintosh do r. 2004, dále použití v automobilních ECU – Freesscale MPC5xx)
- A další typy RISC CPU…..

8051 – příklad instrukcí pro přesun dat

„Jednou“ instrukcí MOV lze přistupovat do různých oblastí v paměti.

Data transfer

MOV	A,Rn	Move register to accumulator	1	1
MOV	A,direct*)	Move direct byte to accumulator	2	1
MOV	A,@Ri	Move indirect RAM to accumulator	1	1
MOV	A,#data	Move immediate data to accumulator	2	1
MOV	Rn,A	Move accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct byte	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load data pointer with a 16-bit constant	3	2

AVR – příklad instrukcí pro přesun dat

Pro přístup do různých paměťových oblastí jsou použity různé instrukce. Instrukce MOV má jen jeden význam.

Data Transfer Instructions					
MOV	Rd, Rr	Copy Register	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Pair	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LDS	Rd, k	Load Direct from data space	Rd ← (k)	None	2
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Increment	Rd ← (X), X ← X + 1	None	2
LD	Rd, -X	Load Indirect and Pre-Decrement	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Increment	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, -Y	Load Indirect and Pre-Decrement	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Increment	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Decrement	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
STS	k, Rr	Store Direct to data space	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Increment	(X) ← Rr, X ← X + 1	None	2
ST	-X, Rr	Store Indirect and Pre-Decrement	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Increment	(Y) ← Rr, Y ← Y + 1	None	2
ST	-Y, Rr	Store Indirect and Pre-Decrement	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Increment	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Decrement	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Increment	Rd ← (Z), Z ← Z + 1	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (RAMPZ:Z)	None	3
ELPM	Rd, Z+	Extended Load Program Memory and Post-Increment	Rd ← (RAMPZ:Z), Z ← Z + 1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
ESPM		Extended Store Program Memory	(RAMPZ:Z) ← R1:R0	None	-
IN	Rd, A	In From I/O Location	Rd ← I/O(A)	None	1
OUT	A, Rr	Out To I/O Location	I/O(A) ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2

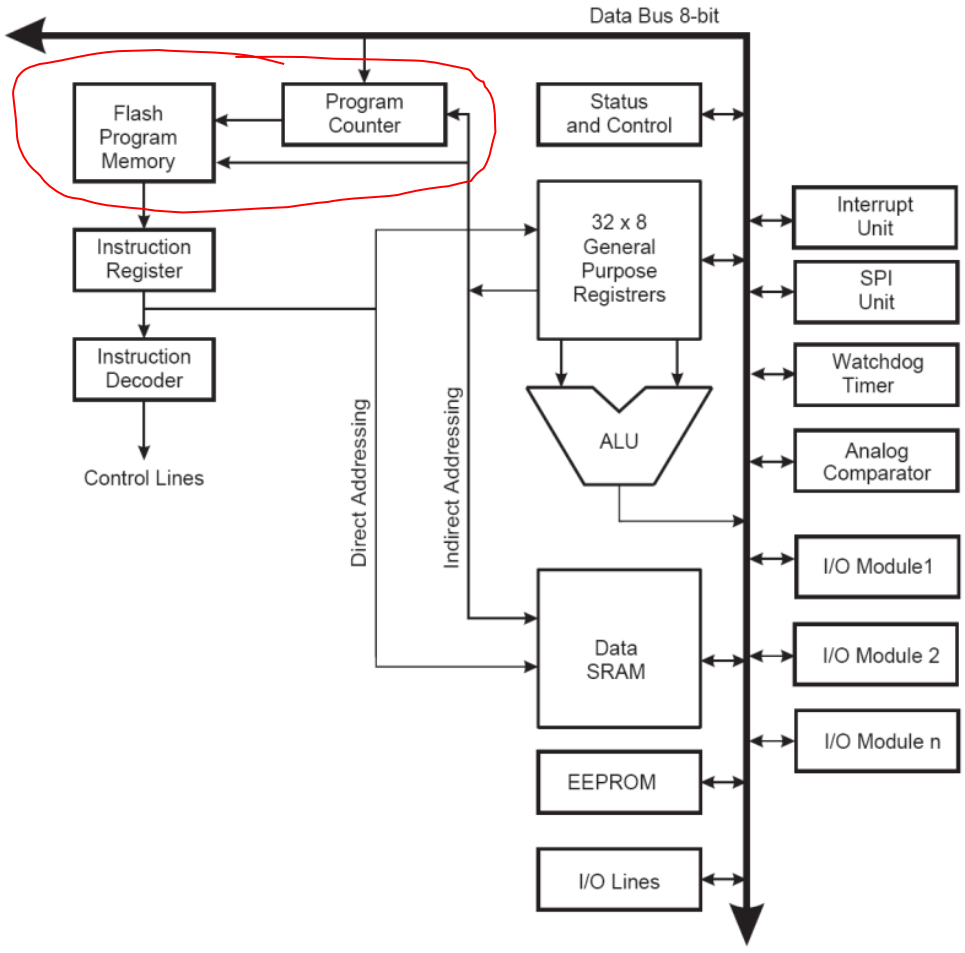
5.3 Činnost CPU

5.3.1 Programový čítač (PC)

- PC = programový čítač (Program Counter)
- pointer do paměti programu – CPU vykonává instrukci, jejíž adresa je v PC
- výchozí hodnota po resetu: 0 (program začíná od adresy 0)
- PC je automaticky inkrementován o délku právě zpracovávané instrukce
    - po vykonání instrukce *i* následuje instrukce *i*+1

```
adresa ROM   instrukce
0x0000       instrukceDlouhá1Slovo
0x0001       instrukceDlouhá2Slova
0x0003       instrukceDlouhá1Slovo
```

    - adresa v ROM = obsah PC před vykonáním instrukce na dané adrese
- změna pořadí provádění instrukcí → manipulace s PC
    - skoky, větvení, volání podprogramů



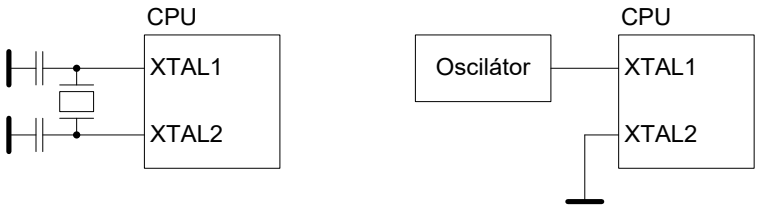
Obrázek 22 Program Counter uvnitř CPU AVR

5.3.2 Instrukční cyklus

- Instruction cycle (IC)
- CPU = sekvenční číslicový obvod
- CPU je řízeno synchronizačním tzv. hodinovým signálem (clock)

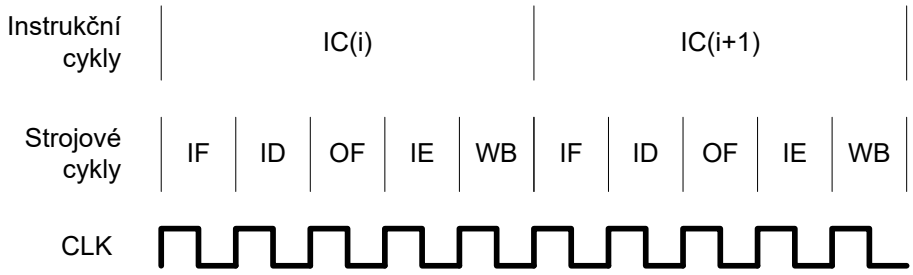
$$f_{CLK} = \frac{1}{T_{CLK}} \quad f_{CLK} = \frac{\mathbf{1}}{T_{CLK}} \qquad \text{[Hz; s]}$$

  - určuje rychlost provádění instrukcí („výkon“ CPU)
    - výkonné CPU – 100ky MHz ÷ 1ky GHz
    - MCU 1ky ÷ 100 MHz
  - pro každé CPU je definováno rozmezí f_{CLK}, kdy je schopno funkce
- nejčastější provedení generátoru hodin



interní oscilátor externí oscilátor

- **IC = doba pro provedení jedné instrukce**
  - ideální varianta $IC = T_{CLK}$ (např. AVR: $N = 1$)
  - často $IC = N^* T_{CLK}$ (např. 8051: $N = 12$)
  - instrukce mohou být různě dlouhé
  - v instrukčním souboru daného MCU je u každé instrukce napsáno kolik instrukčních cyklů je potřeba na její vykonání
- IC se dělí na fáze nazvané strojové cykly (úroveň taktu hodin)
- typický CISC IC se skládá z těchto fází (strojových cyklů):
  - čtení instrukce (Instruction Fetch)
  - dekódování instrukce (Instruction Decode)
  - čtení operandů (Operand Fetch)
  - vykonání instrukce (Instruction Execution)
  - zápis výsledku (Write Back)
- **každou fázi vykonání instrukce provádí jiná část CPU**



- **různé CPU → různý počet a jména strojových cyklů**
- **různé CPU → různé délky IC, různé frekvence** → nelze jednoduše porovnat výkon, proto → MIPS (Million Instructions Per Second) je měřítkem výkonu
  - AVR – 16 MIPS při 16 MHz (1 MIPS / 1 MHz)
  - ARM Cortex M3 – 125 MIPS při 100 MHz (1,25 MIPS / 1 MHz)
  - i486DX 8,7 MIPS při 25 MHz (0,348 MIPS / 1 MHz)
  - Intel Core i7 – 92 100 MIPS při 2,93 GHz (31 MIPS / 1 MHz)

Příklad různě dlouhých instrukcí (AVR):

Instrukce ADD – sečtení čísel uložených ve dvou pracovních registrech

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	$0 \le d \le 31, 0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

Example:

```
add   r1,r2     ; Add r2 to r1 (r1=r1+r2)
add   r28,r28   ; Add r28 to itself (r28=r28+r28)
```

Words: 1 (2 bytes)
Cycles: 1

Instrukce LDS – vyčtení hodnoty z RAM a její uložení do pracovního registru

	Syntax:	Operands:	Program Counter:
(i)	LDS Rd,k	$0 \le d \le 31$, $0 \le k \le 65535$	PC ← PC + 2

32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Example:

```
lds   r2,$FF00     ; Load r2 with the contents of data space location $FF00
add   r2,r1        ; add r1 to r2
sts   $FF00,r2     ; Write back
```

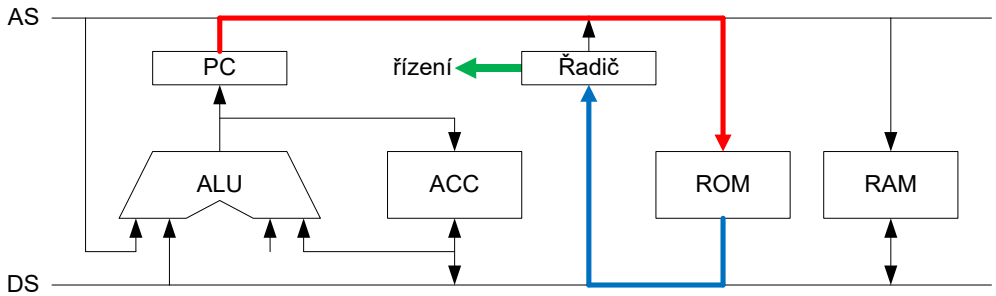
Words: 2 (4 bytes)
Cycles: 2

5.3.3 Instrukční cyklus CISC

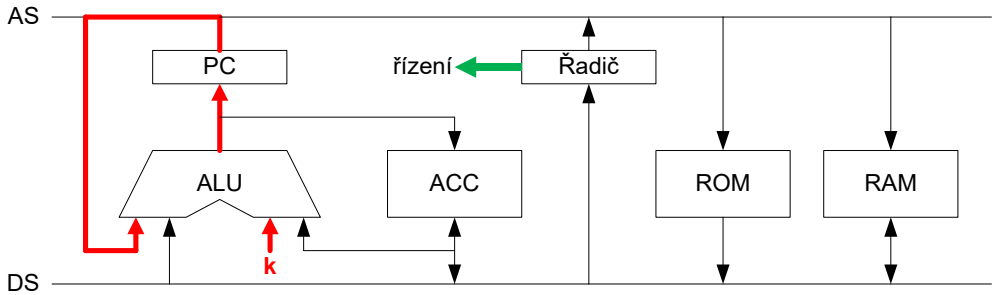
- CISC je obvykle tzv. akumulátorově orientovaná architektura
  - nemá GPR (General Purpose Registers), má pouze akumulátor ACC
  - ALU:
    - jeden operand a výsledek výpočtu je uložen vždy v ACC
    - druhý operand: konstanta, přímá adresa, nepřímá adresa, ...

```
add ACC, adr        // ACC = ACC + obsah na adr
```

- Dále příklad provedení instrukce `add ACC, adr`
- **čtení instrukce (Instruction Fetch):**

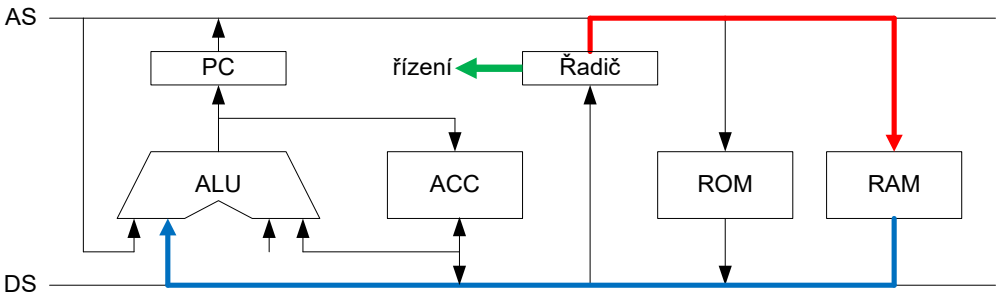


  - V PC je uložena adresa další instrukce, která se bude číst.
  - Řadič: PC (adresa) → AS (adresová sběrnice), generování řídích signálů (ŘS) pro ROM (čtení)
  - Obsah ROM → IR (registr instrukcí v řadiči)
- **dekódování instrukce (Instruction Decode):**

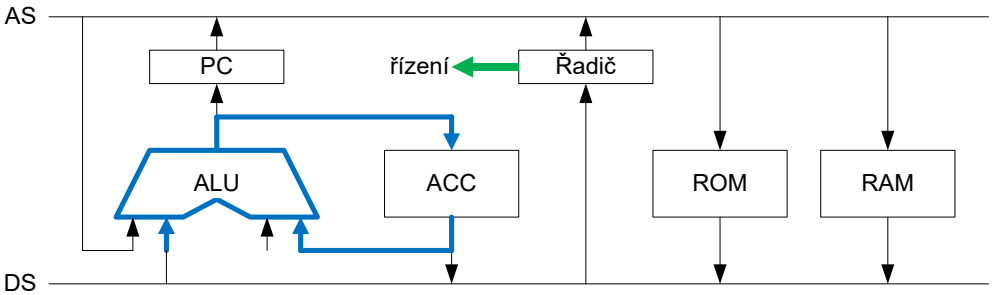


  - Řadič generuje ŘS pro ALU, ta vypočte adresu následující instrukce:
    - z délky právě zpracovávané (uloženo v opcode instrukce)
    - skok (k)
  - výsledek (= adresa následující instrukce) → PC

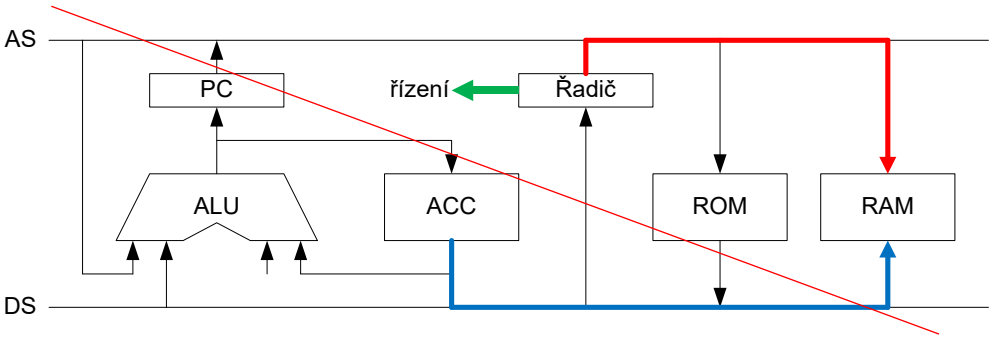
- **čtení operandu z RAM (Operand Fetch):**



  - řadič vystaví adresu operandu na adresovou sběrnici (AS) a generuje ŘS (řídící signály) pro RAM
  - RAM vystaví na datovou sběrnici (DS) obsah a ten je zapsán do ALU
- **vykonání instrukce (Instruction Exection):**

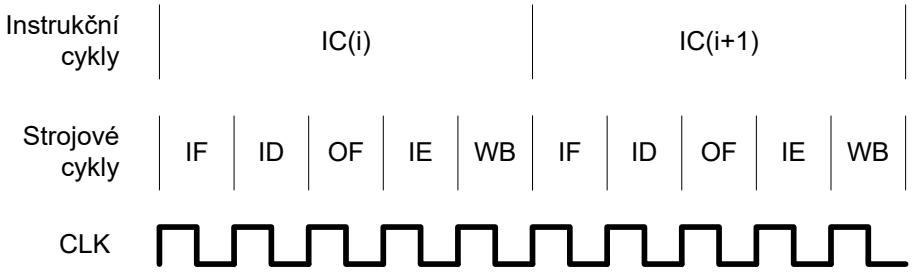


  - řadič generuje ŘS pro ALU, ta instrukci vykoná
  - výsledek je uložen do akumulátoru ACC
- **zápis výsledku (Write Back):**



  - řadič vystaví adresu výsledku na AS a generuje ŘS pro RAM (zápis)
  - ACC → RAM

- **CISC = Nepřekrývané zpracování instrukcí**
  - komplikovaná instrukce = nutno strojové cykly zpracovat za sebou



5.3.4 Instrukční cyklus RISC

- Má méně fází než u CISC.
- **Sčítání čísel ležících v RAM na adresách ADR1 + ADR2:**
  - **CISC (např. 8051):**

```
mov ACC, adr1
add ACC, adr2       // složité CISC instrukce
```

  - **RISC (např. AVR):**

```
lds r0, adr1
lds r1, adr2
add r0, r1          // jednoduché RISC instrukce
```

    - důsledek: v IC u RISC nemusí být OF (skryto v IE u `lds`)
- **Čekací smyčka**

AVR

```
cekej:
      ldi R18,0xFF      ;1IC naplneni registru konstantou 255(10)
hop:  ldi R19,0xFF      ;1IC naplneni registru konstantou 255(10)
hop2: dec R19           ;1IC dekrementuj
      brne hop2         ;2IC skakej pokud je R19!=0
      dec R18           ;1IC dekrementuj
      brne hop          ;2IC skakej pokud je R18!=0
```

  - jedná se o 2 cykly, jeden vnořený do druhého (vnější má počítadlo R18, vnitřní má počítadlo R19), celkový počet průchodů cyklem: 255*255=65025
  - kombinace instrukcí `dec` a `brne` je ekvivaletnem instrukce `djnz` u 8051

8051

```
cekej:
      mov R3,#FFh       ;1IC naplneni registru konstantou
hop:  mov R2,#FFh       ;1IC naplneni registru konstantou
hop2: djnz R2,hop2      ;2IC dekrementuj a skoč jestliže R2!=0
      djnz R3,hop       ;2IC dekrementuj a skoč jestliže R3!=0
```

  - `djnz` u 8051 trvá 2 IC (**2280 ns**), `dec` + `brne` u AVR trvá 3 IC (ale **190 ns**)

5.3.5 Pipelining

- Pipelining – překrývané / zřetězené / proudové zpracování instrukcí.
- Myšlenka – zaměstnat nevytížené části CPU v jednotlivých fázích IC.
- Pipeline je rozdělena do částí (stupňů), každá část pracuje na jedné instrukci, všechny části pipeline pracují současně, části pipeline se vzájemně neovlivňují (každá dělá něco jiného).
- Pro realizaci pipeliningu jsou potřeba stejně dlouhé instrukce, aby nedocházelo k porušení pipeline (např. 1 IC instrukce následovaná 2 IC instrukcí).

IF1	IF2	IF3	IF4	IF5
	ID1	ID2	ID3	ID4
		IE1	IE2	IE3



(na obr. je 3 stupňová pipeline)

  - **v každém taktu hodin CPU je dokončena jedna instrukce**, $IC = T_{CLK}$

- **počet stupňů pipeline**
  - odvislé od architektury CPU a od plánovaných f_{CLK} (vyšší CLK = delší pipeline)
  - MCU obvykle 2 až 5
  - Pentium 4 – 20 až 31 (kvůli cílení na f_{CLK} až 10 GHz)
  - Intel Core 2 Duo – 14 stupňů pipeline
- **problémy pipeliningu:**
  - instrukce musí být stejně dlouhé (pokud nejsou, tak pipeline není optimálně využita) – u RISC splněno lépe než u CISC
  - po sobě jdoucí instrukce pro zápis a čtení ze stejné adresy – následná instrukce potřebuje výsledek předchozí instrukce, jenže ta ještě není dokončena
  - skoky, větvení, volání podprogramů – např. u podmíněného skoku se včas neví, zda se bude nebo nebude skákat, hrozí výpadek pipeline

```
        a           // představuje výpočet podmínky
        jmp   sem   // představuje podmíněný skok: if (podmínka)
                    // then skákej na sem
        b           // else jdi dál a vykonej instrukci b
        c
sem:    d
```

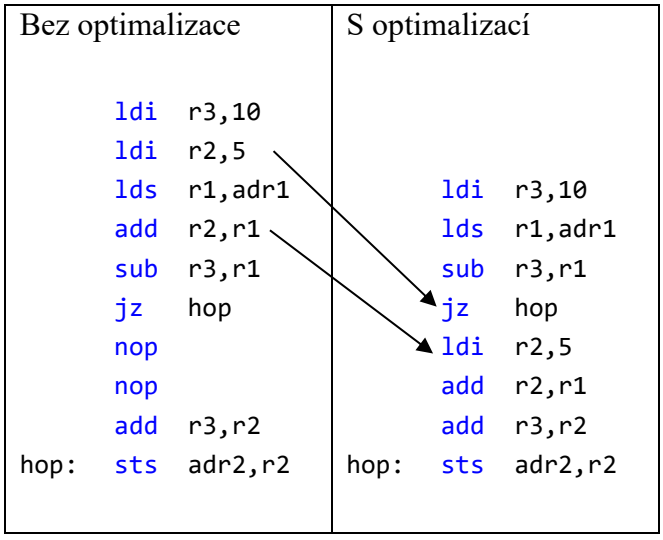
F	a	jmp	b	c	d			
D		a	jmp	b	c	d		
E			a	jmp	b	c	d	

    - pokud se bude skákat, tak je nutné zahodit obsah pipeline (rozpracované instrukce b a c) → chvilkové zpomalení ve vykonávání instrukcí než se pipeline zase zaplní dalšími instrukcemi

- CPU a/nebo kompilátor jsou navrženy tak, aby uměly předcházet výše uvedeným problémům pipeliningu – cílem je mít pipeline neustále zaplněnou správnými instrukcemi za účelem maximalizace výkonu.
- Typy optimalizací:
  - **skoky** - různé metody predikce skoku, zpoždění instrukce skoku
  - **předávání dat mezi po sobě jdoucími instrukcemi** – např. umělé zpoždění druhé instrukce o tolik cyklů, kolik je jich potřeba na dokončení předchozí instrukce
  - **čtení dat z paměti** (vykonání instrukce přístupu do paměti trvá obvykle déle než jiná instrukce) – např. změnou pořadí vykonávání instrukcí za účelem zaplnění pipeline během vykonávání instrukce čtení z paměti
  - Těmito optimalizacemi lze dosáhnout plného vytížení CPU.

Příklad optimalizace programu zpožděním instrukce skoku

Předpokládejme 3 stupňovou pipeline a že je doba vykonání všech instrukcí v příkladu stejná.

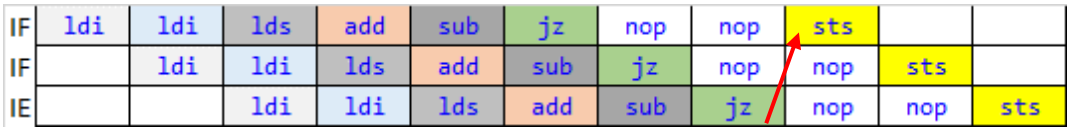


Význam instrukcí:

`ldi Rd,K` - nahrání číselné konstanty K do registru Rd
`lds Rd,adr` - vyčtení hodnoty z adresy adr z RAM a její uložení do registru Rd
`add Rd,Rr` - sečtení obsahu dvou registrů, Rd = Rd + Rr
`sub Rd,Rr` - odečtení obsahu dvou registrů, Rd = Rd – Rr
`jz hop` - skáče na návěští hop v kódu, pokud byl výsledek předchozí operace 0
`nop` - „no operation“, instrukce, která nic nedělá
`sts adr,Rr` - zápis hodnoty na adresu adr do RAM, zdrojem hodnoty je registr Rr

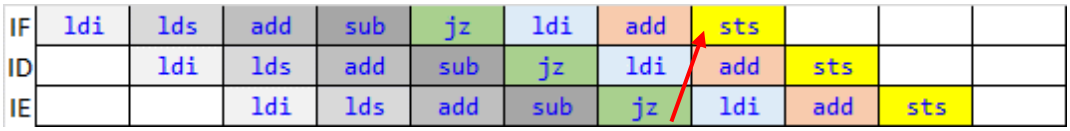
Označení `d` a `r` v legendě jsou pořadová čísla registrů.

V levém sloupci je neoptimalizovaná varianta - přidané dvě „prázdné“ instrukce `nop` za instrukci podmíněného skoku `jz`. Tyto dvě instrukce `nop` zajistí, že se v případě skoku nevykoná instrukce `add r3,r2`. Tyto instrukce jsou tam vždy, i v případě že se skákat nebude. To ale znamená zdržení ve vykonávání programu. Vložené instrukce `nop` jsou tedy jen nutná výplň, která zajišťuje správné vykonání programu (ať se bude nebo nebude skákat), ale zdržuje vykonávání.



Obrázek 23 Obsah pipeline – neoptimalizovaná varianta skoku

V pravém sloupci je optimalizovaná varianta, ve které jsou instrukce `nop` nahrazeny instrukcemi `ldi r2,5` a `add r2,r1` z jiné části programu (před instrukcí podmíněného skoku). Došlo tedy ke změně pořadí vykonávání instrukcí. To ale nevadí, protože ani jedna z instrukcí nemá vliv na nastavení podmínky skoku, a vykonají se vždy, ať se bude nebo nebude skákat. Původní záměr programátora tedy není touto změnou porušen.



Obrázek 24 Obsah pipeline – optimalizovaná varianta skoku

Vkládání instrukcí `nop` nebo optimalizaci kódu změnou pořadí vykonávání instrukcí **provádí kompilátor**.

Oba dva obrázky s obsahem pipeline znázorňují situaci, kdy je podmínka skoku splněna a bude se skákat na návěští `hop`, na kterém leží instrukce `sts adr2,r2`. V případě, že by se neskákalo, tak jediná změna bude v tom, že instrukce `sts adr2,r2` v obrázcích bude nahrazena instrukcí `add r3,r2`, až po které bude následovat instrukce `sts adr2,r2`.

Červená šipka v obrázcích označuje okamžik vykonání skoku – instrukce `jz` změní obsah programového čítače (PC), který obsahuje adresu, ze které se bude načítat další instrukce, a do 1. stupně pipeline se v dalším taktu načte instrukce na místě odskoku (instrukce `sts adr2,r2` u návěští `hop`).

5.3.6 Rychlost CISC vs. RISC

- **8051 (CISC)**
  - IC = 12*T_{CLK}
  - většina instrukcí trvá 2 IC (násobení a dělení 4 IC)
  - 48 instrukcí (ale významově odpovídá 111 instrukcím)
- **AVR (RISC)**
  - IC = 1*T_{CLK}
  - drtivá většina instrukcí trvá 1 IC (několik 2 IC – typicky přístup do RAM, násobení, relativní skoky, a jen málo 3/4/5 IC – volání podprogramu a návrat, čtení konstant z paměti programu)
  - 118 instrukcí
- AVR je více než 12x výkonnější než 8051, ale program má více instrukcí (navíc např. instrukce dělení chybí, nutno naprogramovat „ručně“ → cca 60 IC)
- výkon AVR = cca Intel 80386 (CISC) na stejné f_{CLK}

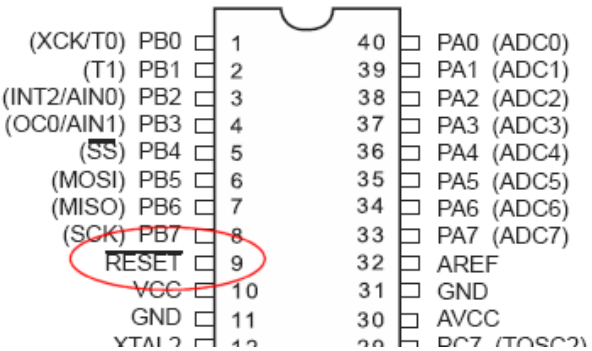
5.3.7 Reset

Pro správnou funkci procesoru je nutné splnit následující podmínky, po jejichž splnění může být procesor spuštěn:

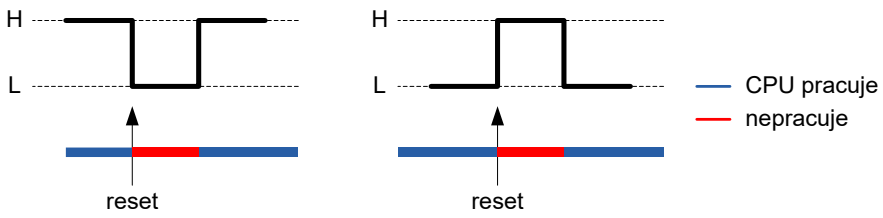
- dosažní správného napájecího napětí
- rozběh a ustálení oscilátoru (systémových hodin)

Uvedení procesoru do výchozího stavu po zapnutí napájení zajišťuje signál RESET.

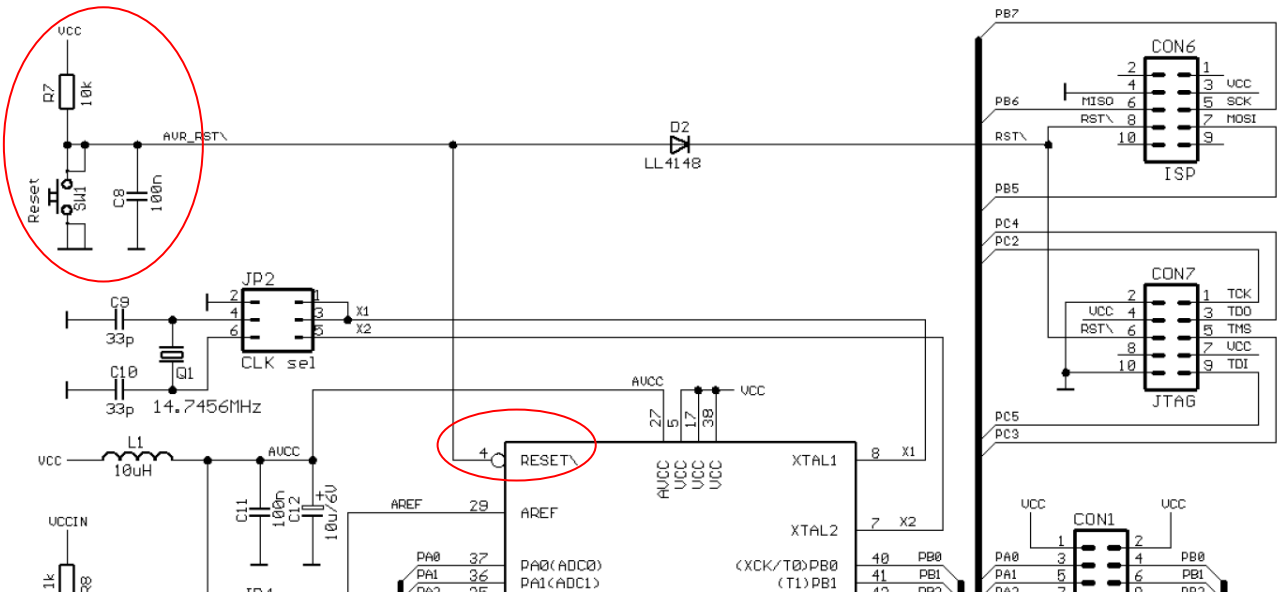
- Hodnoty registrů a dalších obvodů MCU AVR po resetu:
  - PC = 0
  - SP = 0x60 (první volná adresa v RAM za IO registry)
  - GPR = 0
  - Registry v I/O prostoru – viz. datasheet
  - RAM = 0
  - Přerušení vypnuto (viz později)
  - I/O porty do stavu vysoké impedance, aby neovlivňovaly k nim připojené obvody – viz později
- Jak provést reset MCU ? → hardwarový vstup RESET
  - Příklad ATmega32 (pouzdro DIL40)



  - Polarita externího signálu RESET
    - aktivní v L (např. AVR) aktivní v H (např. 8051)



  - Nutno dodržet min. délku pulzu, např. 1,5 µs u AVR.
  - Pro jednoduché aplikace postačuje jednoduchý časovací RC článek připojený na vstup RESET. Pro složitější aplikace (nutno resetovat více obvodů na DPS při startu) se používají specializované resetovací obvody.



Obrázek 25 Signál RESET generovaný pomocí RC článku

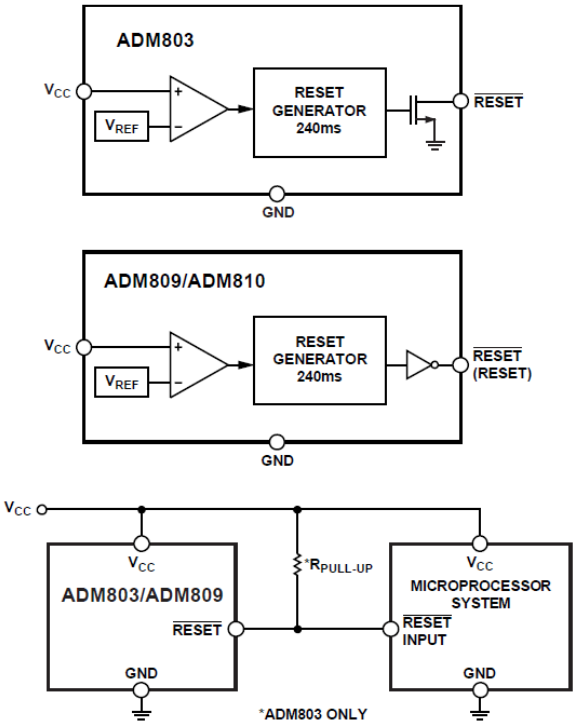
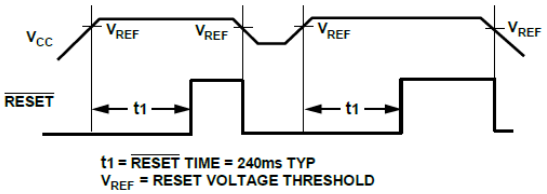


Table 3. Reset Threshold Options

RESET Model	Threshold (V)
ADM8xxL	4.63
ADM8xxM	4.38
ADM8xxJ (ADM809/ADM810 Only)	4.00
ADM8xxT	3.08
ADM8xxS	2.93
ADM8xxR	2.63
ADM8xxZ	2.32



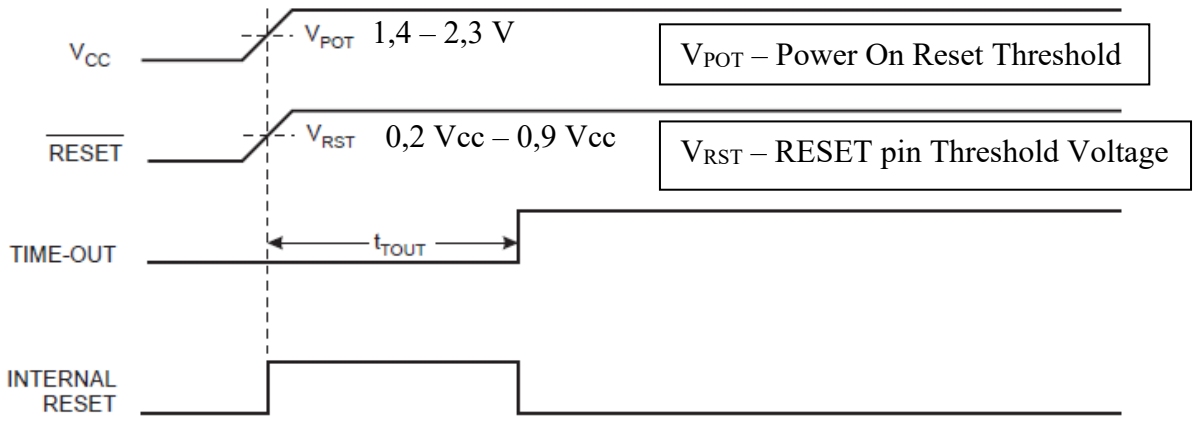
Přechod do resetu pokud je Vcc < V_{REF}.

Opuštění resetu až pokud je Vcc > V_{REF} po dobu t1.

Obrázek 26 Signál RESET generovaný speciálním obvodem

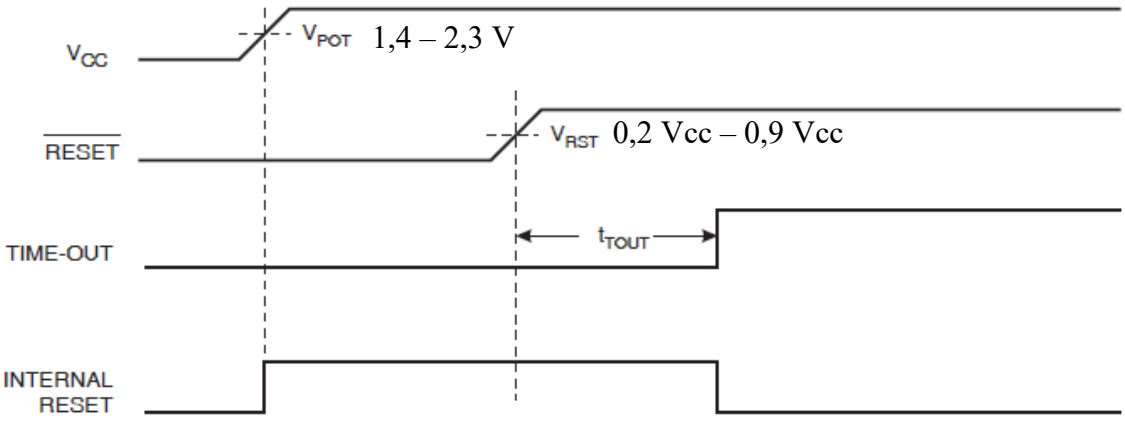
- Speciální externí resetovací obvody sledují velikost napájecího napětí a při podpětí generují signál reset. Tím je zajištěno definované chování procesoru. Dle typu obvodu lze/nelze nastavovat prahovou hodnotu napětí, při kterém obvod vybaví.

Figure 16. MCU Start-up, $\overline{\text{RESET}}$ Tied to V_{CC}.



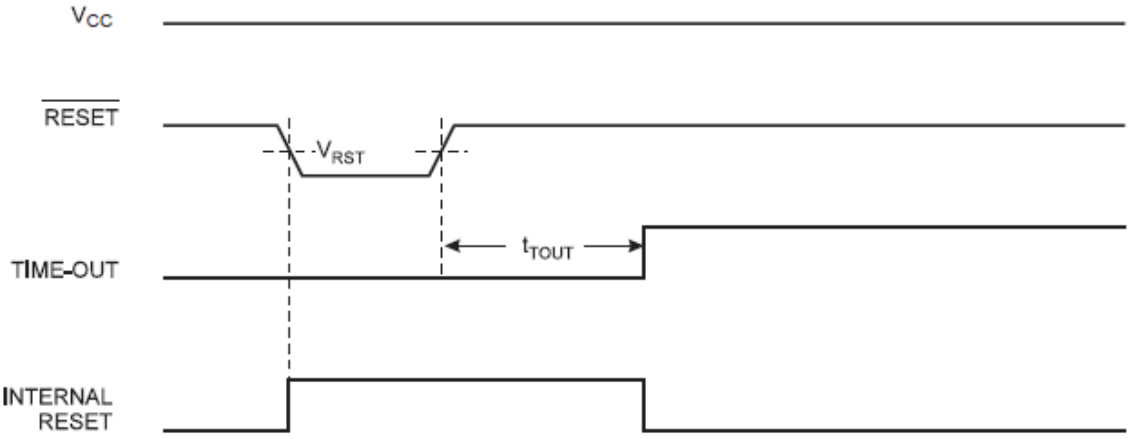
Obrázek 27 POWER-ON reset bez externího signálu RESET

Figure 17. MCU Start-up, $\overline{\text{RESET}}$ Extended Externally



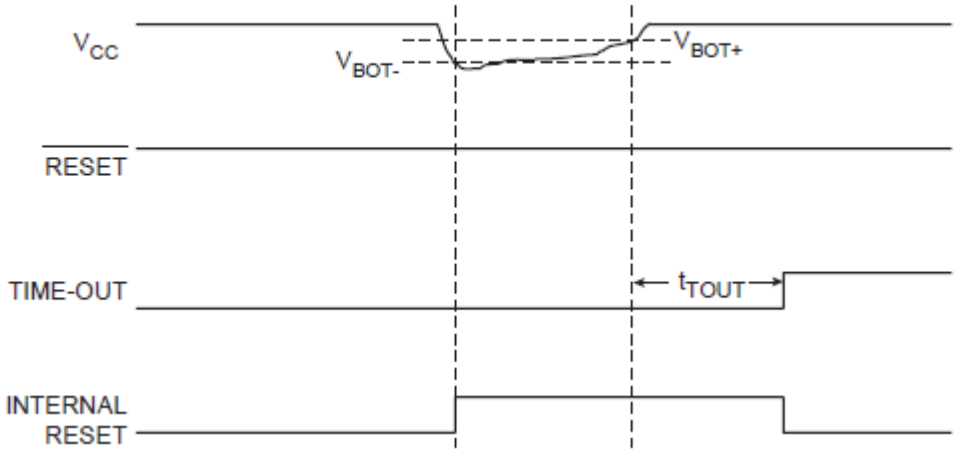
Obrázek 28 POWER-ON reset s externím signálem RESET

Figure 18. External Reset During Operation



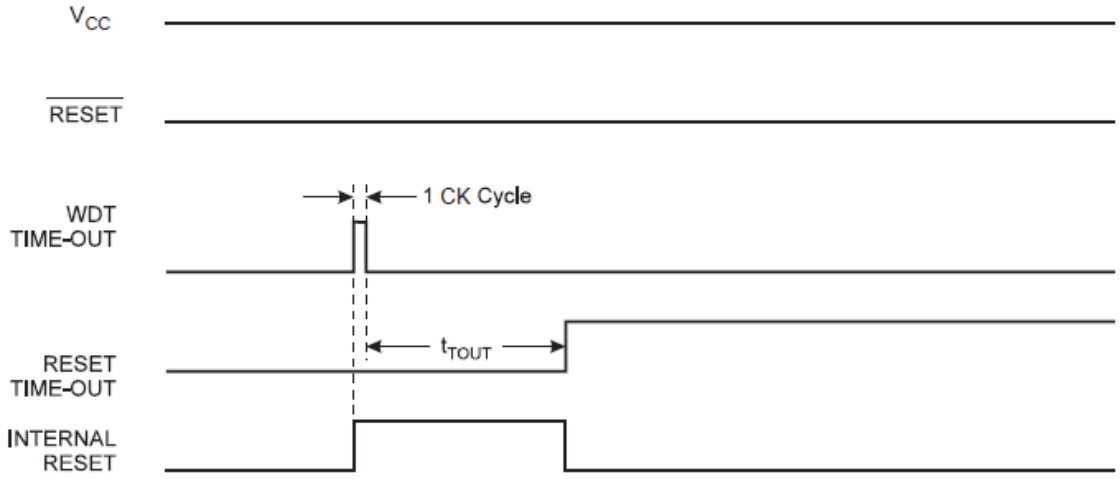
Obrázek 29 Vyvolání resetu externím signálem RESET za běhu MCU

Figure 19. Brown-out Reset During Operation



Obrázek 30 Interní reset MCU vyvolaný poklesem napájecího napětí

Figure 20. Watchdog Reset During Operation



Obrázek 31 Interní reset MCU vyvolaný watchdogem

Vcc – napájecí napětí MCU

/RESET – pin /RESET na MCU

TIME-OUT (t_{TOUT}) – časové zpoždění, po kterém interní signál reset odezní (je to přídavné zpoždění). U AVR se nastavuje při programování MCU externím programátorem v tzv. pojistkách (FUSES), lze nastavit v rozsahu 4,1 až 65 ms v závislosti na rychlosti náběhu napájecího napětí a náběhu oscilátoru. FUSES nelze nastavovat z aplikace běžící v MCU, nýbrž pouze externím programátorem.

INTERNAL RESET – interní signál RESET v MCU

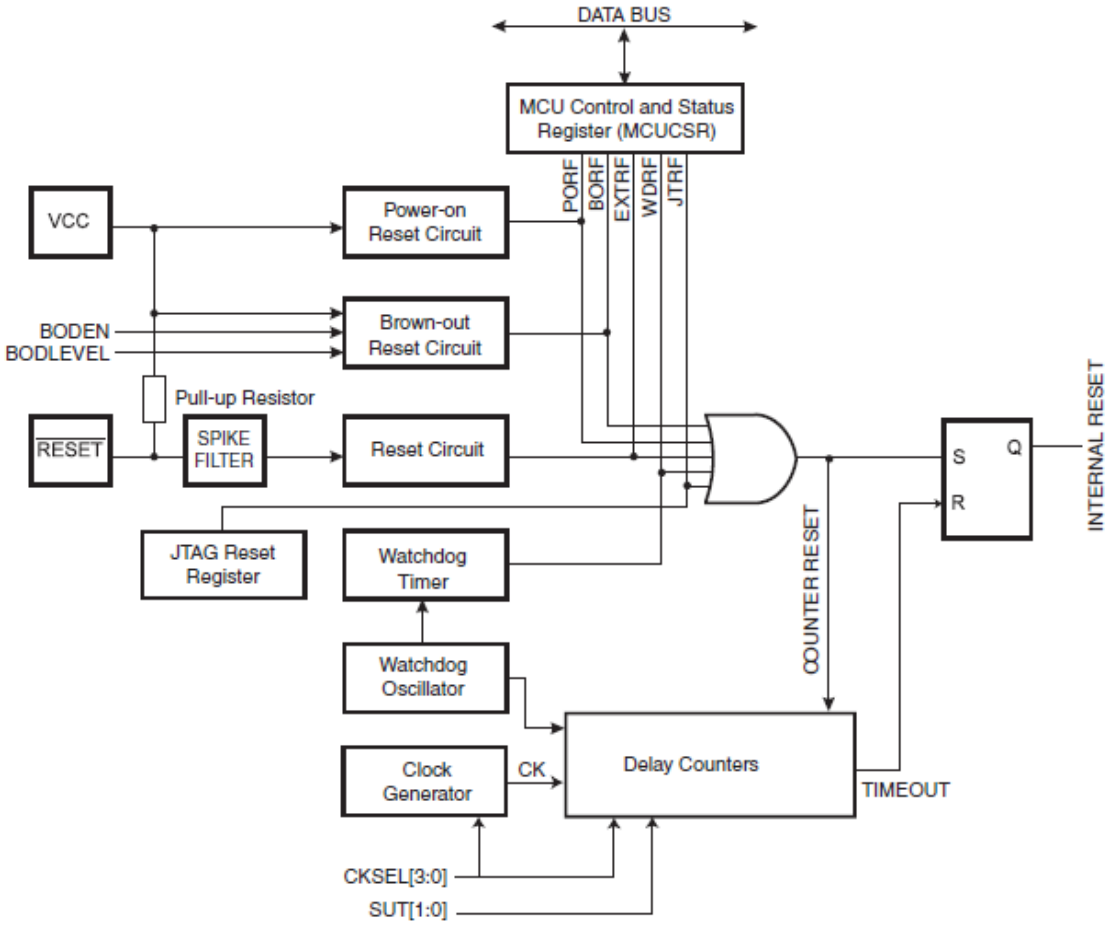
Table 6. Start-up Times for the Low-frequency Crystal Oscillator Clock Selection

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	1K CK(1)	4.1 ms	Fast rising power or BOD enabled
01	1K CK(1)	65 ms	Slowly rising power
10	32K CK	65 ms	Stable frequency at start-up
11	Reserved		

Note: 1. These options should only be used if frequency stability at start-up is not important for the application.

Obrázek 32 Nastavení dodatečné prodlevy během resetu v tzv. pojistkách (fuses) u MCU AVR

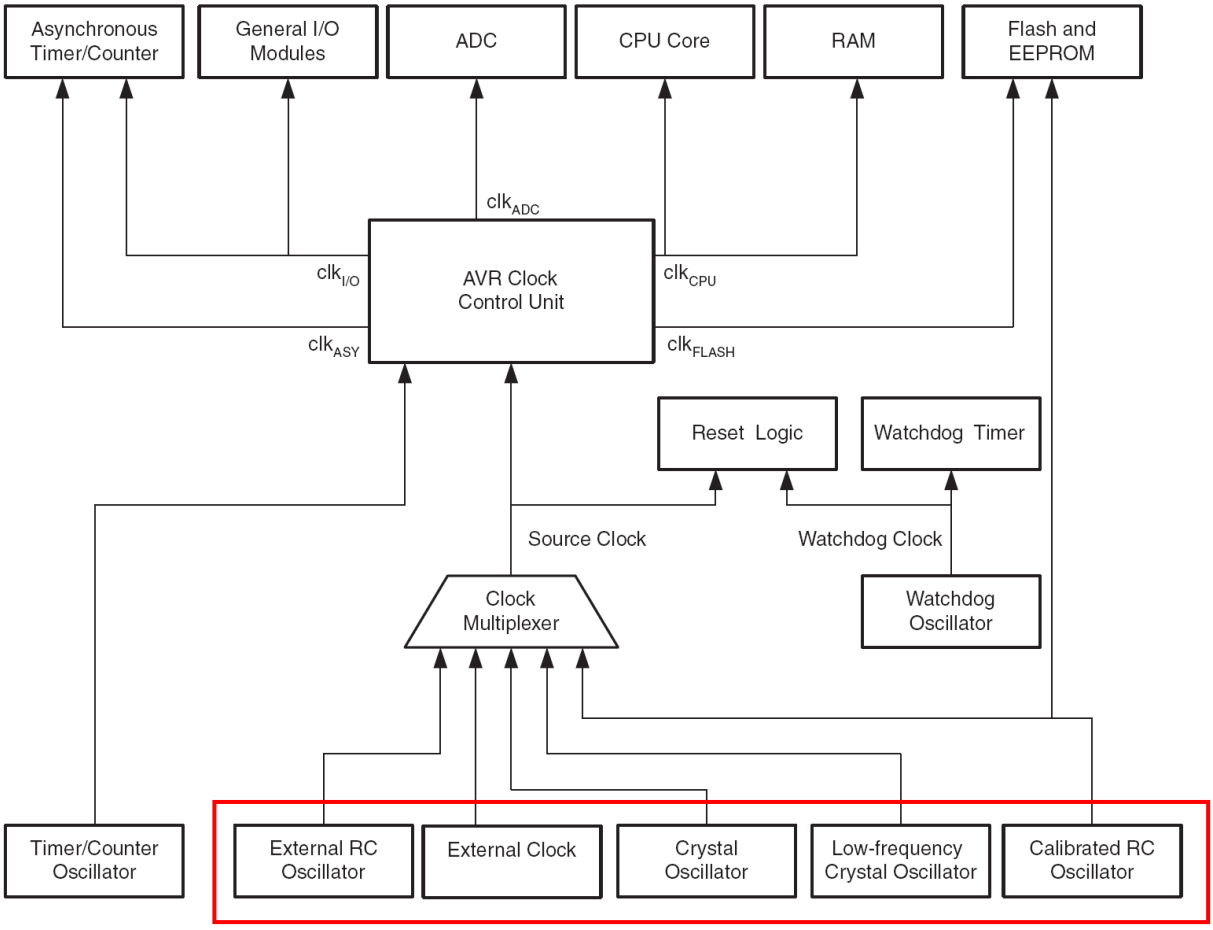
- MCU má obvykle více zdrojů signálu RESET, např. MCU AVR má tyto:
  - Externí pin /RESET
  - Vnitřní obvod detekce podpětí (BOD = Brown Out Detector)
  - Watchdog (časovač, který když vyprší, tak generuje signál RESET)
  - Příkaz RESET na komunikačním (programovacím) rozhraní JTAG



Obrázek 33 Logika RESETu (MCU AVR)

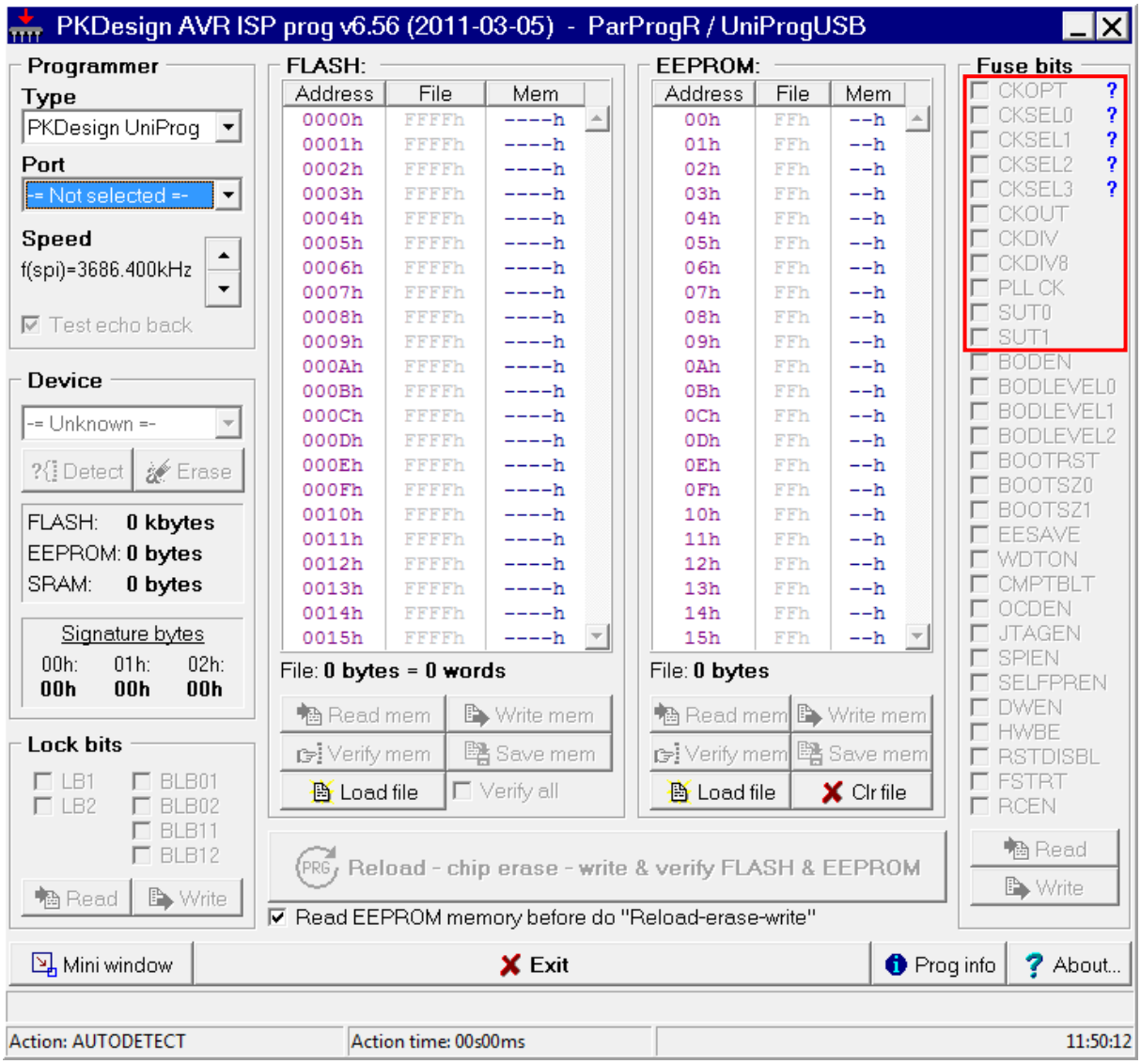
5.4 Zdroje taktovacího kmitočtu

- externí krystal + interní oscilátor – nejpřesnější, teplotně nejstabilnější
- externí keramický rezonátor + interní oscilátor
- externí RC oscilátor
- interní kalibrovaný RC oscilátor
- externí zdroj taktovacího kmitočtu (vytvořený externím oscilátorem – používá se, pokud má být více obvodů na DPS taktováno stejným hodinovým signálem)
- externí nízkofrekvenční krystal (32,768 kHz) – druhý zdroj taktovacího kmitočtu použitý obvykle pro RTC (hodiny reálného času)



Obrázek 34 Rozvedení taktovacích signálů v MCU

- U MCU AVR se výběr zdroje taktovacího kmitočtu se provádí nastavením tzv. **Flash Fuse bits** při nahrávání programu do MCU.
- Standardně je nový MCU AVR dodáván s nastaveným interním RC oscilátorem 1 MHz.



Obrázek 35 Fuse bits (ISP programátor Uniprog)

- Červeně jsou v Obrázek 35 orámovány **Fuse bits**, kterými se vybírá zdroj kmitočtu a nastavují se parametry oscilátoru.
- Význam Fuse bits (při programování MCU AVR programátorem):
  - CKOPT Fuse volí mezi dvěma zesíleními zesilovače oscilátoru. Když není naprogramovaná je max. Fosc = 8 MHz, v opačném případě max. Fosc = 16 MHz. Uvedené hodnoty platí pro ATmega32.
  - Programováním CKSEL Fuses se vybírá mód oscilátoru v závislosti na frekvenci krystalu.
  - Programováním SUT Fuses se volí doba náběhu oscilátoru.
  - Detaily viz datasheet.

- V tabulkách níže jsou uvedeny možnosti nastavení z datasheetu MCU ATmega32.

Table 2. Device Clocking Options Select(1)

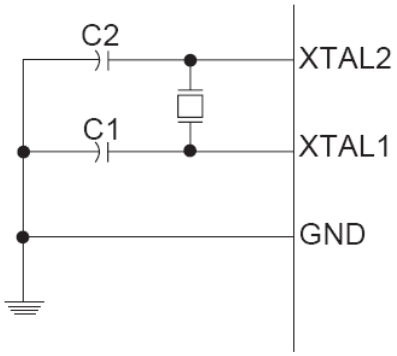
Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

Note: 1. For all fuses "1" means unprogrammed while "0" means programmed.

Table 4. Crystal Oscillator Operating Modes

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101(1)	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

Note: 1. This option should not be used with crystals, only with ceramic resonators.



Obrázek 36 Připojení externího krystalu k MCU AVR

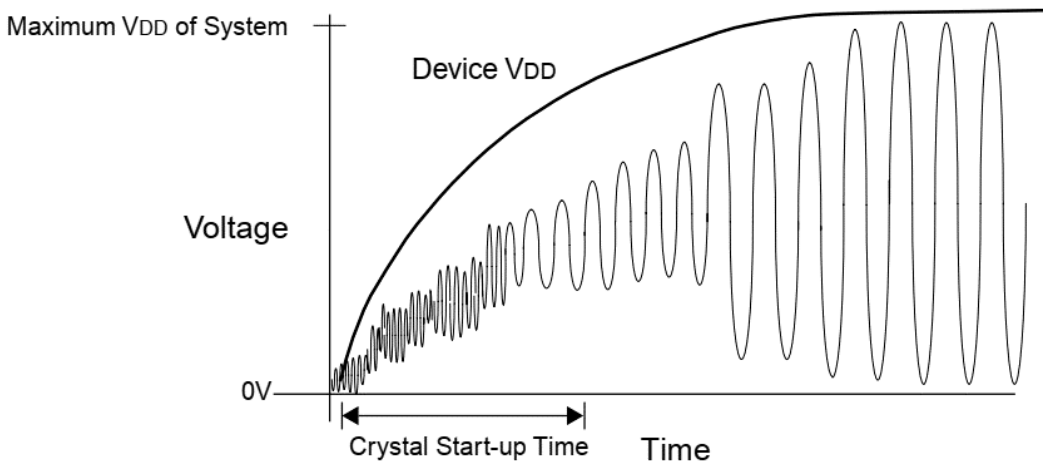
Table 5. Start-up Times for the Crystal Oscillator Clock Selection

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK(1)	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK(1)	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK(2)	–	Ceramic resonator, BOD enabled
0	11	1K CK(2)	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK(2)	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

Notes:
1. These options should only be used when not operating close to the maximum frequency of the device, and only if frequency stability at start-up is not important for the application. These options are not suitable for crystals.
2. These options are intended for use with ceramic resonators and will ensure frequency stability at start-up. They can also be used with crystals when not operating close to the maximum frequency of the device, and if frequency stability at start-up is not important for the application.

Start-up Time – rychlost probuzení MCU z režimu spánku (jako počet taktů oscilátoru)

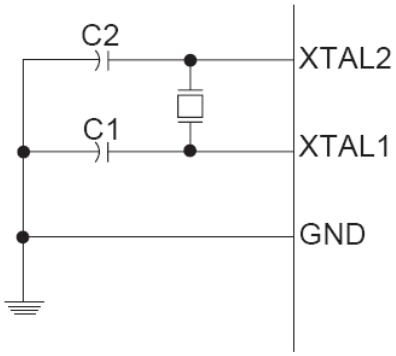
Additional Delay from Reset – prodleva t_{out} v opuštění resetu (viz např. Obrázek 29)



Obrázek 37 Náběh oscilátoru po zapnutí napájení MCU

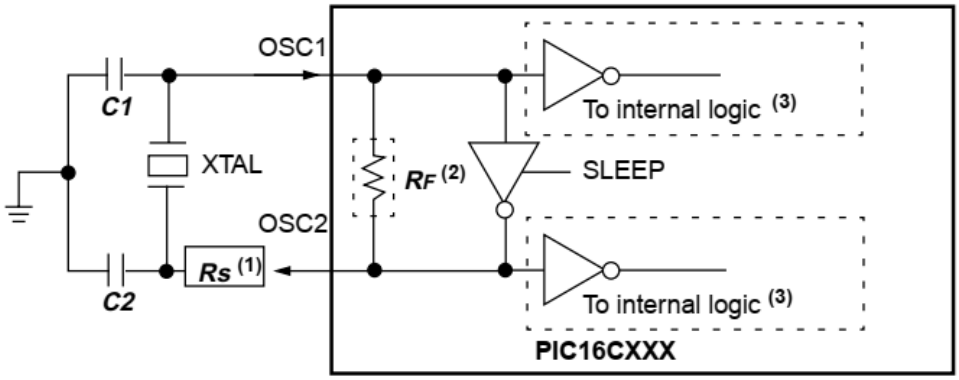
5.4.1 Externí krystal nebo keramický rezonátor

- Připojuje se na piny MCU obvykle pojmenované XTAL1 (vstup) a XTAL2 (výstup).
- Krystal zaručuje vysokou přesnost a stabilitu frekvence
  - Přesnost frekvence např. 50 ppm = 0,005 % při teplotě 25 °C.
  - Stabilita frekvence např. 50 ppm = 0,005 % pro celý teplotní rozsah -10 °C až +70 °C.



Obrázek 38 Připojení externího krystalu k MCU AVR

- Krystal / keramický rezonátor je uvnitř MCU připojen k oscilátoru:



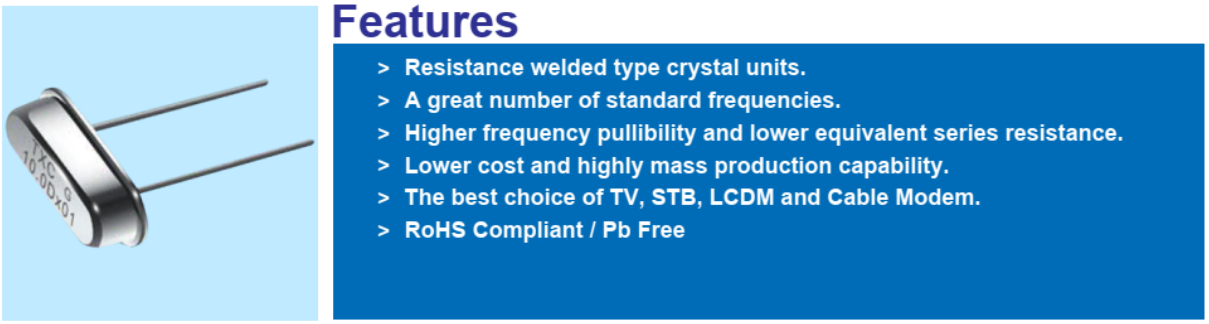
Note 1: A series resistor, Rs, may be required for AT strip cut crystals.
2: The feedback resistor, RF, is typically in the range of 2 to 10 MΩ.
3: Depending on the device, the buffer to the internal logic may be either before or after the oscillator inverter.

Obrázek 39 Oscilátor s logickými hradly a krystalem

  - Jedná se o paralelní rezonanční obvod.
  - Logické hradlo – invertor – aktivní člen oscilátoru, zajišťuje fázový posuv 180°.
    - Signálem SLEEP se oscilátor zapíná/vypíná.
  - XTAL – krystal kmitající na základní frekvenci.
  - C1 a C2 jsou externí kondenzátory umožňující fázový posuv 180°, jejich velikost má velký význam pro stabilitu oscilací. Vyšší hodnoty zvyšují

stabilitu oscilátoru, ale výkonově zatěžují krystal a prodlužují čas náběhu oscilátoru po zapnutí napájení.

- Doporučená hodnota zatěžovacích kondenzátorů C1 a C2 uvedená v datasheetech MCU bývá v rozsahu 12-30 pF.
- Je třeba se řídit doporučením výrobce krystalu (viz datasheet ke krystalu, viz Obrázek 40).



Features

> Resistance welded type crystal units.
> A great number of standard frequencies.
> Higher frequency pullibility and lower equivalent series resistance.
> Lower cost and highly mass production capability.
> The best choice of TV, STB, LCDM and Cable Modem.
> RoHS Compliant / Pb Free

Electrical Specifications

Item / Type	9B
Frequency Range	3.2 ~ 90 MHz
Frequency Tolerance (at 25 °C)	± 30 ppm, or specify
Frequency Stability Over Operating Temperature Range	± 30 ppm (- 20 ~ +70 °C), or specify
Operating Temperature Range	- 20 ~ +70 °C
Shunt Capacitance (C0)	7 pF Max.
Drive Level	1 ~ 500 µW (100 µW typical)
Load Capacitance	18 pF, or specify
Aging (at 25 °C)	± 3 ppm / year Max.
Storage Temperature Range	- 40 ~ + 85 °C

Obrázek 40 Parametry krystalu

- Rs – tlumící rezistor, který s C1 funguje jako dolní propust, redukuje zesílení na vyšších kmitočtech. Má hodnotu desítky až stovky Ohm. Obvykle není potřeba.
- R_F – zpětnovazební rezistor, který určuje velikost zpětné vazby. Rezistor je uvnitř MCU.

Co je jednotka ppm?

ppm = parts per million (1 ppm je jedna miliontina z nějakého celku)

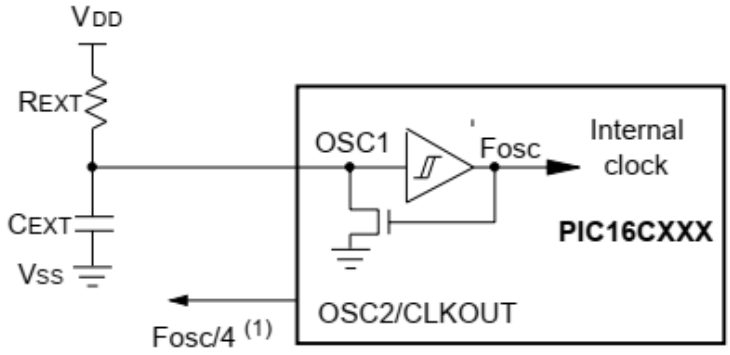
1 % = 10 000 ppm (jedna setina)

0,1 % (1 promile) = 1000 ppm (jedna tisícina)

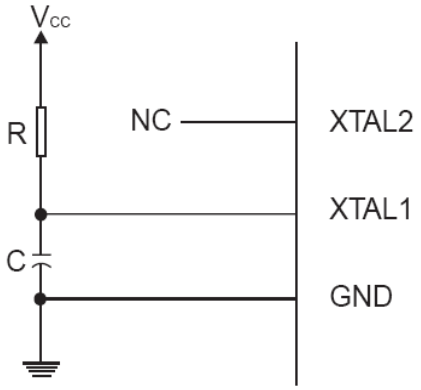
Pro lepší představu, udávaná přesnost frekvence krystalu o hodnotě ±30 ppm vyjádřená v % činí ±0,003 %.

5.4.2 Externí RC oscilátor (oscilátor s externím RC článkem)

- Pro aplikace nenáročné na stabilitu a přesnost taktovacího kmitočtu (± 4 %).
- Kmitočet se mění (jednotky procent) v závislosti na teplotě a napájecím napětí.



Obrázek 41 Zapojení RC oscilátoru u MCU PIC16

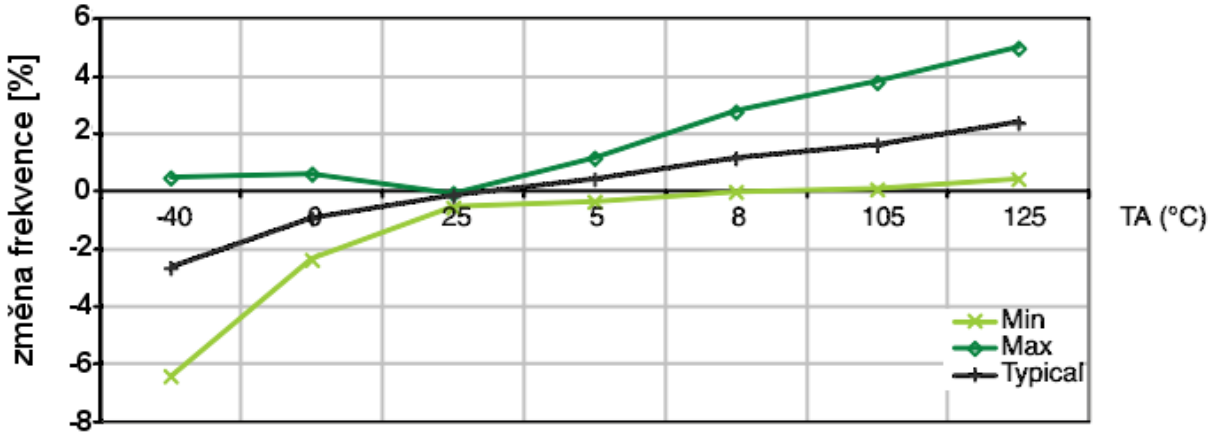


Obrázek 42 Oscilátor s externím RC článkem u MCU AVR

- Kmitočet se nastavuje pomocí externích R a C zapojených do vstupu XTAL1. Pin XTAL2 zůstává nezapojen.
  - $F_{OSC} = 1 / (3*RC)$
  - R – obvykle jednotky až desítky kOhm (např. 3 až 100 kOhm). Čím nižší hodnota, tím lepší (zvýšení stability, zmenšení citlivosti na šum).
  - C – čím větší hodnota, tím lepší (zvyšuje se stabilita, redukuje se citlivost na parazitní kapacity v obvodu).
  - Nelze však kombinovat jakékoliv hodnoty R a C, doporučené hodnoty R a C jsou uvedeny v datasheetu MCU.
  - U AVR činí minimální hodnota kondenzátoru C = 22 pF. Lze aktivovat vnitřní kapacitu na vstupu XTAL1 pomocí CKOPT Fuse, pak stačí k pinu XTAL1 připojit pouze rezistor R.
- Mód oscilátoru a doba náběhu se u AVR opět volí pomocí CKSEL a SUT Fuse při programování MCU externím programátorem.

5.4.3 Interní kalibrovaný RC oscilátor

- Kalibrovaný = výrobce MCU donastaví frekvenci oscilátoru v jednom bodě (při jednom napájecím napětí a jedné teplotě okolí), aby se co nejvíce blížila jmenovité hodnotě, tím se dosáhne odchylka frekvence do cca 1 až 3 %.
- Opět pro aplikace nenáročné na přesnost časování (frekvence se mění s teplotou).

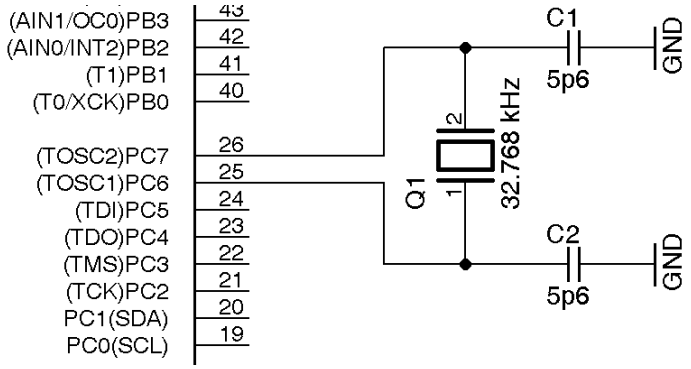


Obrázek 43 Teplotní stabilita interního RC oscilátoru (MCU STM32F446)

- U MCU AVR je na výběr je několik pevně daných kmitočtů, závisí na typu AVR. Výběr se provádí programováním CKSEL Fuse. Např. pro ATmega32 lze volit 1, 2, 4, 8 MHz.
- Odpadá připojení externích součástek R a C.
- Standardně je z výroby interní RC oscilátor zapnut a běží na 1 MHz.
- **U MCU AVR výrobce provádí kalibraci interního RC oscilátoru pro všechny dostupné hodnoty frekvence, při napájecím napětí 5 V a teplotě 25 °C. Tím se dosáhne přesnosti frekvence ±3 %.**
- Kalibrační hodnoty pro jednotlivé frekvence jsou uloženy ve speciální oblasti paměti.
- Po resetu se automaticky registr OSCCAL naplní kalibrační hodnotou pro 1 MHz což zajistí kalibraci RC oscilátoru při napájecím napětí 5 V a teplotě 25 °C.
- V případě, že se použije jiná frekvence než 1 MHz, musí se kalibrační hodnota nahrát do registru OSCCAL ručně. To se provede nejprve vyčtením kalibrační hodnoty pro zvolenou frekvenci a jejím následným uložením do FLASH nebo EEPROM. Po té ji lze softwarově načíst a naplnit registr OSCCAL.
- Tímto způsobem lze kalibrovat RC oscilátor pro určité napájecí napětí a teplotu a tím dosáhnout nezávislosti frekvence na napětí a teplotě. Postup, jakým způsobem lze zjistit správnou kalibrační hodnotu v OSCCAL registru pro napětí jiné než 5 V a teplotu různou od 25 °C, je popsán v aplikační poznámce AVR053 od Atmelu.

5.4.4 Externí oscilátor pro jeden z časovačů

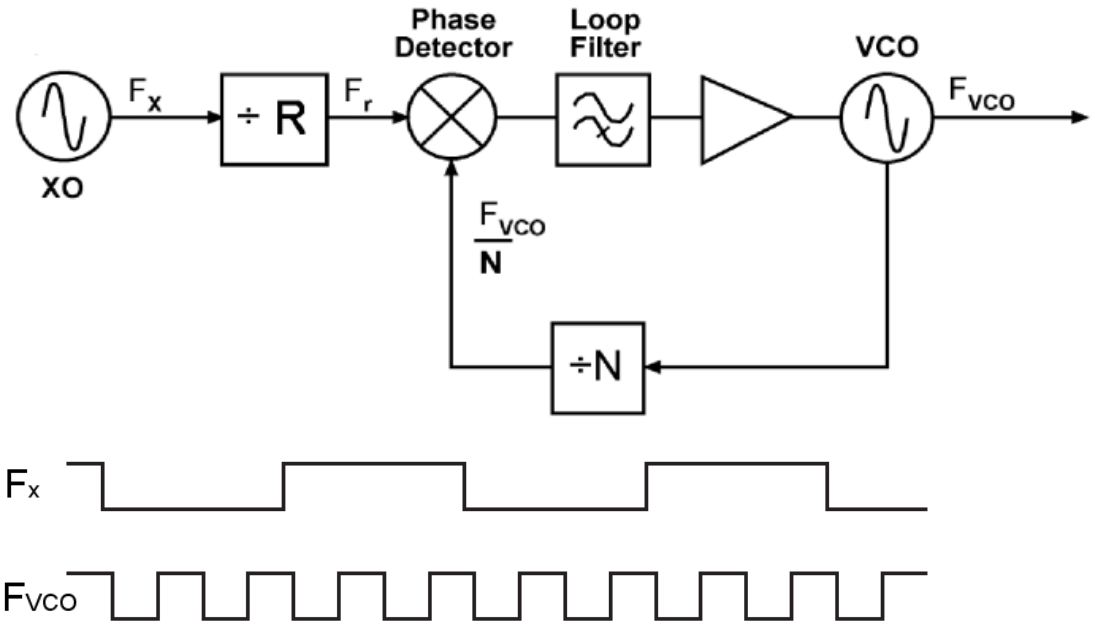
- Některé typy AVR dovolují připojit na vstupy TOSC1 a TOSC2, patřící časovači, externí krystal o frekvenci 32,768 kHz.



- Časovač (obvykle časovač TCNT2) pak není taktován interním signálem odvozeným z hodin procesoru, nýbrž je taktován tímto externím krystalem. Taktovací signál pro časovač tak není synchronní s taktovacím signálem ve zbytku MCU, který je generován jiným oscilátorem.
- **Lze využít pro generování hodin reálného času (RTC) – odměřování sekund.**

5.4.5 Fázový závěs (PLL)

- PLL = Phase Locked Loop
- Netýká se MCU AVR, ale často se používá u jiných typů MCU, proto je vhodné se o něm zmínit.
- U MCU slouží PLL jako násobič kmitočtu.
- Základní kmitočet, jehož zdrojem je obvykle stabilní krystalový oscilátor, se pomocí PLL vynásobí a vznikne kmitočet vyšší, kterým je taktováno CPU a interní periferie.
- Např. z krystalového oscilátoru 10 MHz se pomocí PLL vytvoří taktovací signál o kmitočtu 40 MHz (10 MHz x 4).
- Mimo oblast výpočetní techniky se PLL používá také např. pro demodulaci FM, PM, AM.
- Popis funkce PLL
    - PLL je zpětnovazební regulační obvod.
    - Pomocí zpětné vazby se nastavuje frekvence výstupního signálu tak, aby byla násobkem pevné vstupní frekvence, jejímž zdrojem je obvykle stabilní krystalový oscilátor.



Obrázek 44 Blokové schéma PLL (zdroj: TI Technical brief SWRA029)

    - Význam bloků:
        - XO – krystalový oscilátor zdroj pevného vstupního kmitočtu
        - Phase detector – fázový detektor
        - Loop filter – filtru typu dolní propust
        - VCO – napětím řízený oscilátor
        - R a N – děličky frekvence

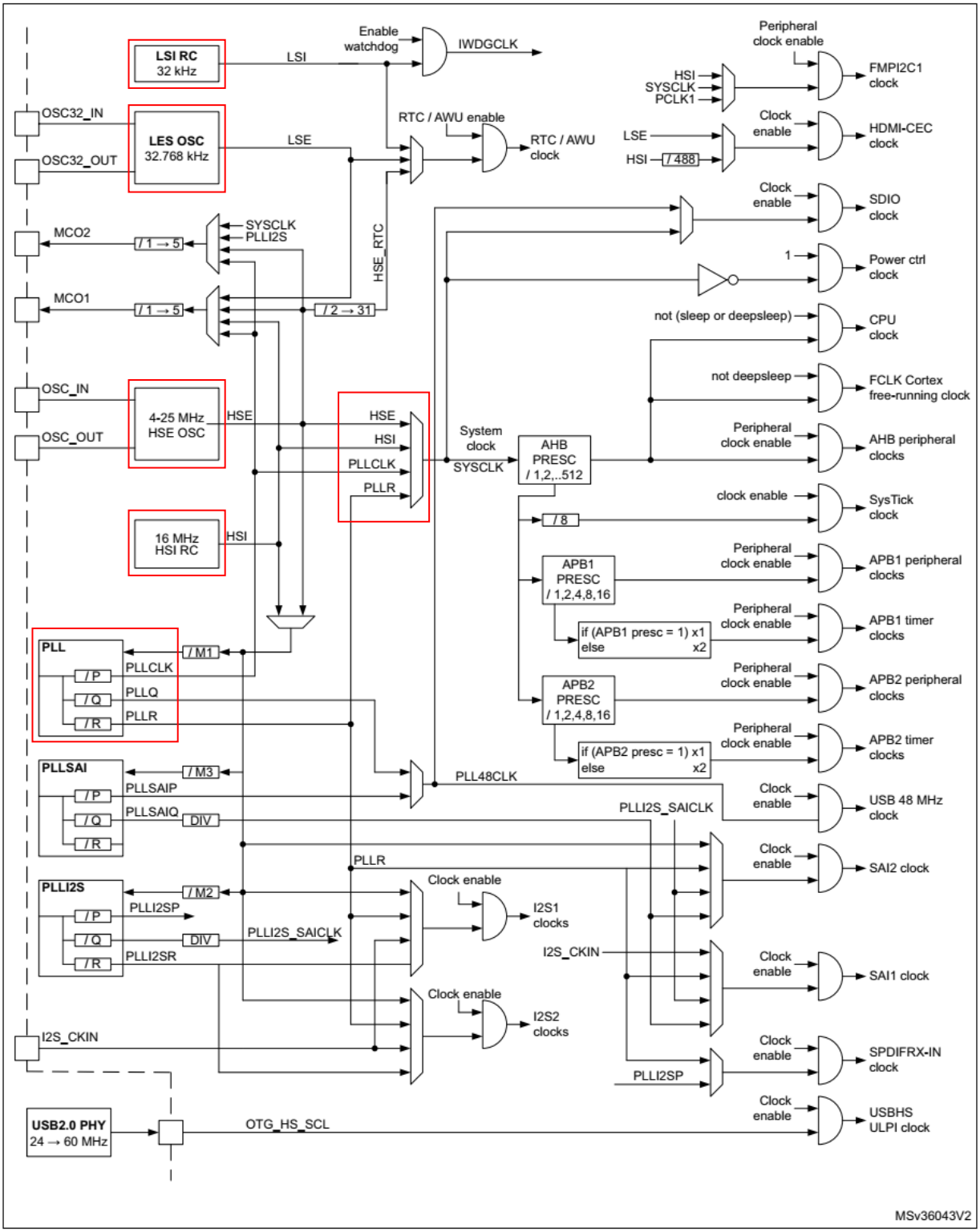
- XO je krystalový oscilátor, který je zdrojem obdélníkového signálu o velmi stabilní frekvenci F_x, která bude smyčkou PLL zvýšena na frekvenci F_{VCO}:

$$F_{VCO} = N \cdot F_r = \frac{N}{R} \cdot F_X \tag{1}$$

  - Výstupní frekvenci F_{VCO} lze nastavovat v diskrétních krocích pomocí programovatelných děliček R a N, které jsou realizovány pomocí čítačů.
  - R a N jsou celá čísla.
  - Velikost kroku je rovna frekvenci F_r.
  - U MCU může být zdrojem frekvence F_x např. krystalový oscilátor nebo RC oscilátor (externí/interní), podle požadavků na přesnost a stabilitu frekvence.
- Signál s frekvencí F_x je vydělen děličkou R (R je celé číslo). Tím dostáváme frekvenci $\mathbf{F_r = F_x / R}$, která vstupuje do fázového detektoru, společně se signálem o frekvenci $\mathbf{F_{VCO}/N}$ (výstupní signál s kmitočtem F_{VCO} podělený N-krát ve zpětné vazbě).
- **Fázový detektor** porovnává fázi vstupního signálu o frekvenci F_x/R s fází signálu ve zpětné vazbě (F_{VCO}/N). Výstupní signál z fázového detektoru prochází filtrem typu dolní propust, pomocí kterého se získá stejnosměrné analogové napětí, jehož velikost je úměrná fázové chybě. Pomocí tohoto napětí se snižuje nebo zvyšuje frekvence VCO až do doby, než se fáze (frekvence) signálů vstupujících do fázového detektoru rovná, pak je dosaženo tzv. „závěsu“. Signál pro řízení VCO je úměrný rozdílu fází a kompenzuje chybu kmitočtu.
- Stavy PLL:
  - **Zachycování** – probíhá proces dolaďování výstupní frekvence, to trvá určitou dobu (tzv. **lock time**), u MCU např. 300 μs.
  - **Sledování** – již došlo k zachycení, výstupní frekvence je nastavena, fázová chyba je na minimální hodnotě.

- Vlastnosti PLL:
  - Vstupní a výstupní frekvenční rozsah.
  - Velikost frekvenčního kroku (ve výše uvedeném příkladu dána poměrem F_r/N).
  - Rychlost přeladění na jinou frekvenci při skokové změně frekvence („**lock time**“).
  - Fázový šum – určuje kvalitu výstupního signálu, udává se jako „**time jitter**“ – neustálá drobná změna periody výstupního signálu v čase (např. ±2 %).
- V MCU může být i více PLL než jedna, např. u ARM Cortex M4 (STM32F446):
  - Hlavní systémový kmitočet a z něj dělením vytvořený taktovací kmitočet pro CPU a periferie.

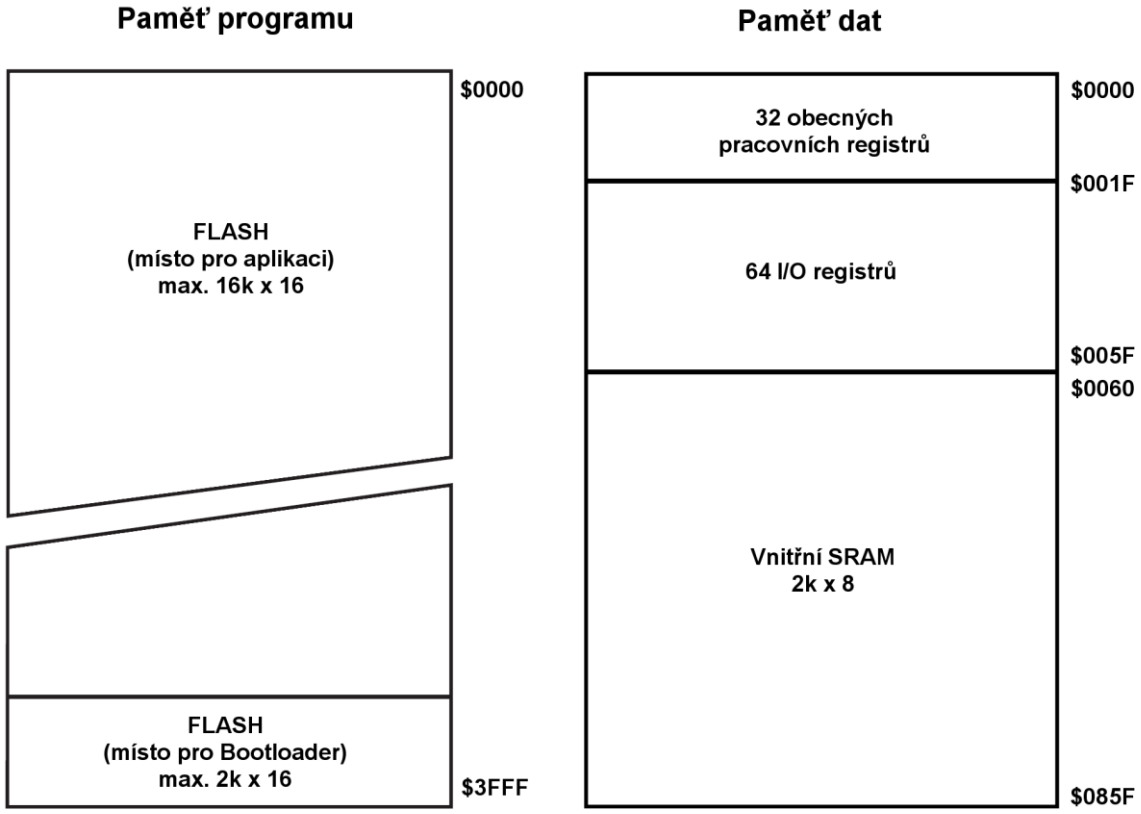
    - Taktovací kmitočet pro digitální audio rozhraní I2S.
    - Taktovací kmitočet pro USB rozhraní.
- Konfigurace PLL se v MCU provádí pomocí registrů (stejně jako obsluha jiných periferií).
    - Hodnoty děliček v PLL.
    - Zapnutí/vypnutí PLL.
    - Čtení stavu zavěšení (PLL locked/unlocked).
- Příklad - zdroj hodin u MCU STM32F446 (ARM Cortex M4):
    - Má několik zdrojů kmitočtu:
        - 1x HSE – High Speed External clock (krystal nebo keramický rezonátor 4 až 25 MHz)
        - 1x LSE – Low Speed External clock (krystal 32,768 kHz)
        - 1x HSI – High Speed Internal clock - interní kalibrovaný RC oscilátor 16 MHz
        - 1x LSI – Low Speed internal clock - interní RC oscilátor 32 kHz
        - 3x PLL – 3 nezávislé PLL pro „system clock“ a periferie (UART, I2C, HDMI, USB, Ethernet, dig. audio, apod.)
    - Samostatné děličky kmitočtu pro CPU a vybrané periferie – různé taktovací frekvence pro různé části MCU.
    - Hodiny lze softwarově vypínat a tím šetřit spotřebu el. energie, když periferie z nich taktované nejsou používány.
    - Frekvence CPU až 180 MHz.
    - PLL:
        - Zdrojem vstupní frekvence je HSE nebo HSI.
        - Frekvence na výstupu VCO je 100 až 432 MHz.
        - Programovatelná dělička M vstupního kmitočtu PLL v rozsahu 2 až 63.
        - Programovatelná dělička N ve zpětné vazbě PLL v rozsahu 50 až 432.
    - Hlídání funkce oscilátoru HSE i PLL (Clock Security System).
        - Při poruše oscilátoru se automaticky generuje událost (nemaskovatelné přerušení), vypíná se HSE oscilátor a přechází se na interní RC oscilátor HSI.
        - Chod aplikace není přerušen, aplikace se navíc dozví o poruše HSE oscilátoru nebo PLL a může na poruchu bezpečně reagovat.



Obrázek 45 Rozvod taktovacích hodin uvnitř MCU STM32F446 (ARM Cortex M4)

6 Paměťový prostor

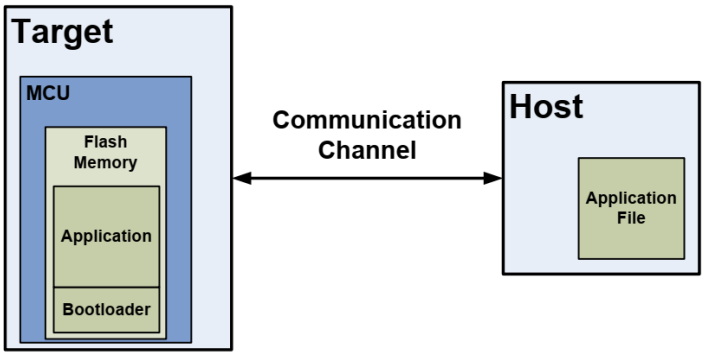
- AVR obsahuje dva (případně tři) oddělené paměťové prostory (paměť programu, paměť dat a případně interní paměť EEPROM pro data).
- MCU je vybaveno na čipu těmito pamětmi (jsou to paměti uvnitř MCU):
    - FLASH - paměť programu
    - SRAM - paměť dat
    - EEPROM – paměť pro data, obsah je uchován i po vypnutí napájení



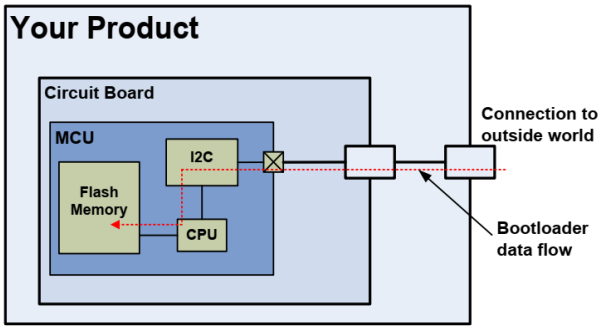
Obrázek 46 Paměťový prostor programu a dat u MCU AVR

6.1 Paměť programu

- Pro uložení kódu (instrukcí) programu.
- typu FLASH, In-System Programmable Memory - přeprogramovatelná v cílovém zařízení (pomocí SPI nebo JTAG rozhraní nebo za chodu aplikace z boot loaderu)
    - FLASH = elektricky mazatelná/programovatelná po blocích 128 bajtů (nelze mazat nebo programovat bajt po bajtu na rozdíl od běžné EEPROM), rychlý přístup (100 ns), životnost 10 000 zápisů
- organizovaná jako N x 16 (to zn. po dvou bajtech), kde N je závislé na velikosti paměti programu u daného typu AVR, např. 512x16 = 8192 B (8 kB paměti)
- šířka programového čítače závisí na velikosti paměti u daného typu MCU (např. u ATmega32 je PC 14-ti bitový – uadresuje tedy 16k paměťových pozic nebo-li 32 kB paměti programu)
- velikost interní FLASH dle typu AVR (1 až 256 kB)
- nelze připojit externí paměť programu
- Bootloader
    - Je to část programu, která umožňuje aktualizovat aplikaci (přeprogramováním paměti programu) za chodu bez nutnosti použít externí programátor.
    - Nový program je do MCU přenesen přes nějaký typ běžného komunikačního rozhraní (UART, USB, CAN, SPI, I2C, …..podle typu zařízení).

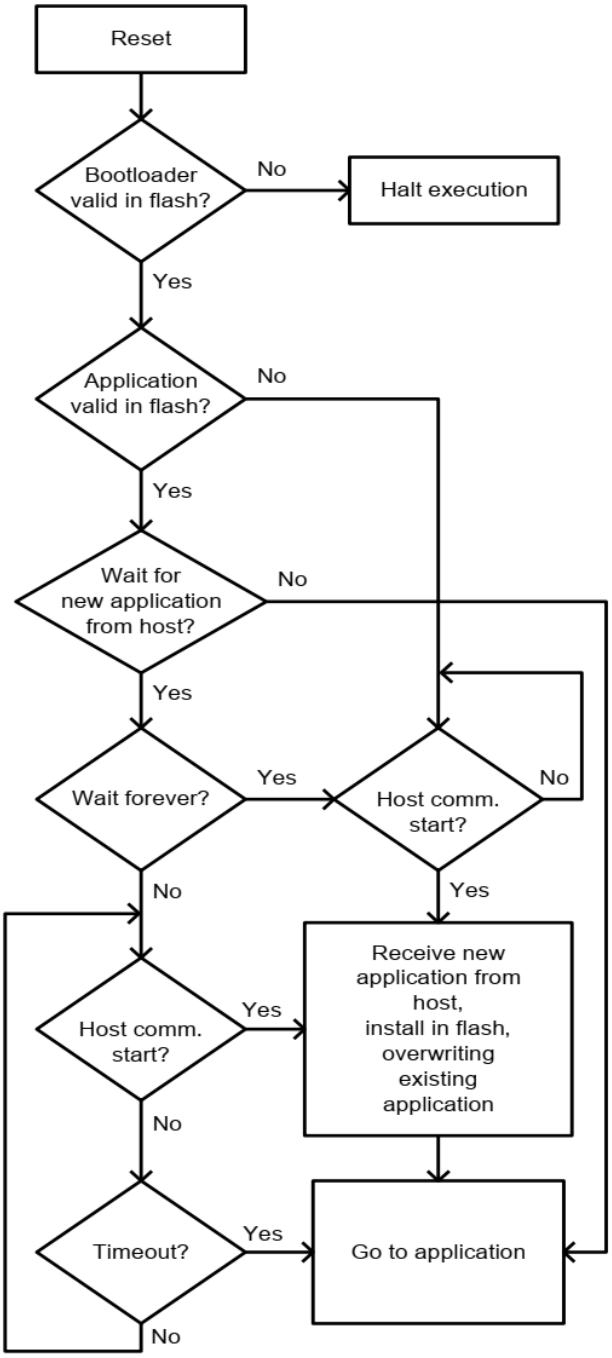


Obrázek 47 Zařízení vybavené bootloaderem pro aktualizaci aplikace, zdroj: Cypress AN73854



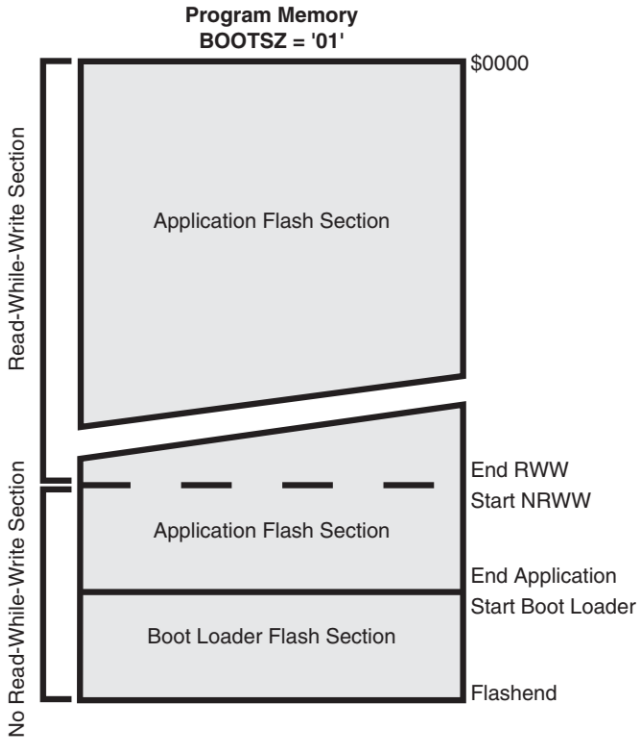
Obrázek 48 Přenos nového programu do zařízení, zdroj: Cypress AN73854

- Jak se přeprogramování aktivuje?
  - Externím pinem.
  - MCU po resetu „poslouchá“ na komunikační lince a čeká na povel. Pokud do určité doby povel nepřijde, spustí se aplikace v MCU. Pokud povel přijde, začne přijímat kód nové aplikace a postupně přeprogramovávat paměť programu, ve které sídlila předchozí verze aplikace. Viz Obrázek 49.



Obrázek 49 Co se děje po zapnutí zařízení, které je vybaveno bootloaderem, zdroj: Cypress AN73854

- Po resetu se nespustí aplikace, ale bootloader, který až následně předá běh aplikaci.
- U MCU AVR je bootloader umístěn v paměti programu v oblasti Boot Block (velikost tohoto bloku je od 512 byte do 8 kbyte, jeho velikost lze zvolit).

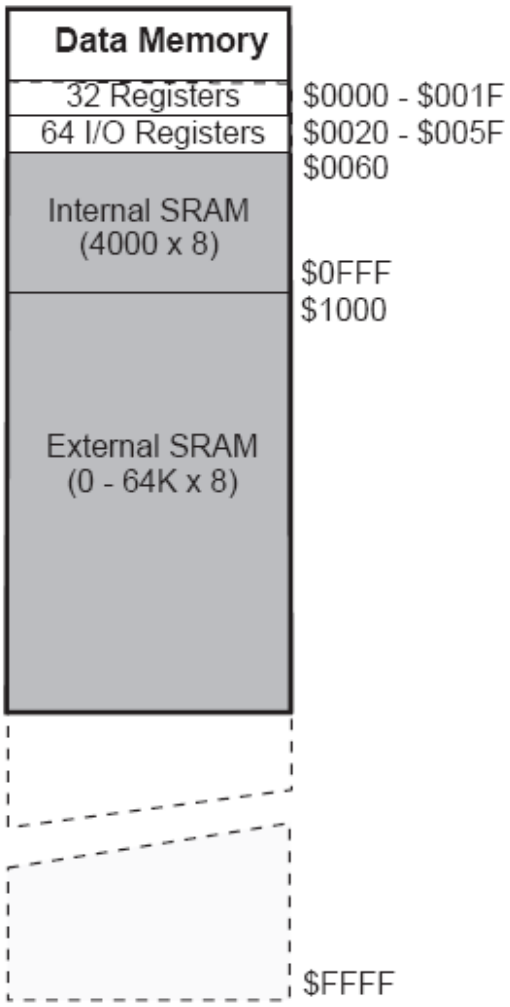


Obrázek 50 Paměť programu s oblastí pro bootloader (MCU AVR)

- Část **Application Flash Section** – zde sídlí aplikace.
- Část **Boot Loader Flash Section** – zde sídlí bootloader.
  - Boot loader je nutné nejprve povolit při prvním programování MCU externím programátorem (u AVR se programují tzv. lockbits).
  - Do oblasti bootloaderu je také nutné nahrát samotný bootloader, který zajistí funkčnost znázorněnou na Obrázek 49.
  - Teprve pak je možné aktualizovat aplikaci i bootloader v zařízení bez externího programátoru.

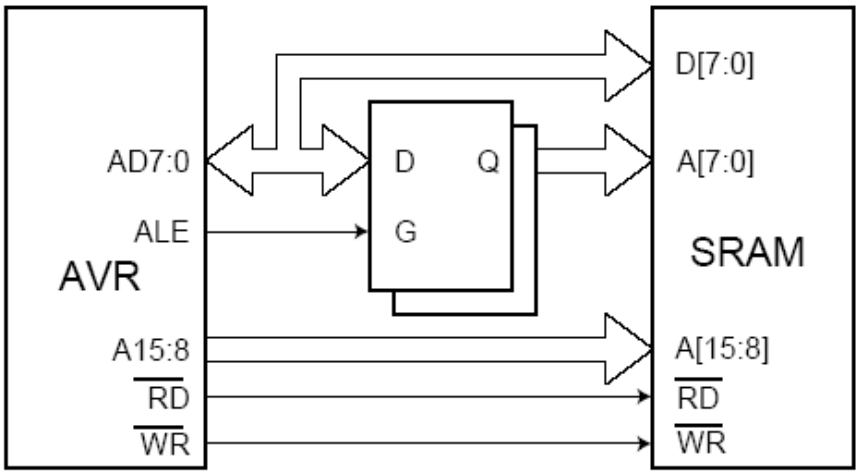
6.2 Paměť dat

- je typu SRAM (static RAM) – paměť s rychlým přístupem, tvořená bistabilními klopnými obvody, které nevyžadují doplňování náboje pro udržení údaje na rozdíl od dynamické paměti DRAM
- slouží pro ukládání proměnných (obecně dat, se kterými pracuje program)
- velikost interní SRAM dle typu AVR (0 až 8kB)
- do prvních 96-ti adres jsou mapovány pracovní a I/O registry – to má výhodu v přístupu na tyto oblasti (přistupuje se do nich stejně jako do SRAM)
- teprve od adresy 0x0060 začíná blok interní paměti SRAM pro ukládání dat
- V paměti SRAM je umístěn zásobník (data stack), jeho velikost je omezená pouze velikostí SRAM. Ukazatel zásobníku (SP) musí být vždy inicializován na začátku programu. Registr SPL a SPH se nachází v I/O prostoru.



Obrázek 51 Paměť dat SRAM

- existuje 5 adresovacích módu pro paměť dat:
    - přímé adresování
    - nepřímé adresování
    - nepřímé adresování s posunutím (6-ti bitový posun)
    - nepřímé adresování s dekrementací ukazatele adresy před zpracováním instrukce
    - nepřímé adresování s inkrementací ukazatele instrukce po zpracování instrukce
- některé typy AVR mají možnost připojení externí paralelní paměti dat (ATmega128, Atmega1280/2560/2561, ATmega162/8515/103)
    - maximální velikost externí paralelní SRAM je 64 kB



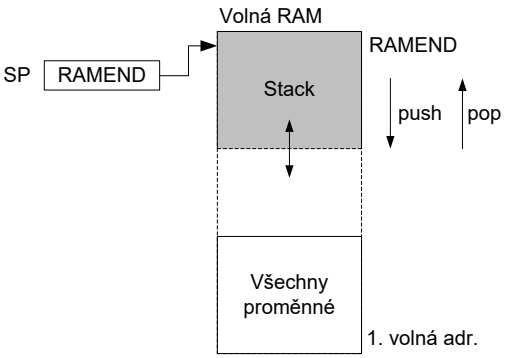
Obrázek 52 Multiplexované připojení externí RAM

- Připojení paměti je klasické s multiplexovanou adresovou a datovou sběrnicí.
- Při překročení rozsahu interní SRAM se automaticky přistupuje do externí paměti, která navazuje na poslední adresu interní SRAM. **K přístupu do externí SRAM se používají stejné instrukce jako při přístupu do interní paměti**.
- Přístup do externí paměti trvá o jeden takt déle. Jestliže je zásobník umístěn v externí paměti, trvá volání podprogramů, přerušení a návrat z nich déle (navíc tři takty), kvůli ukládání návratové adresy (2 bajty) do zásobníku.

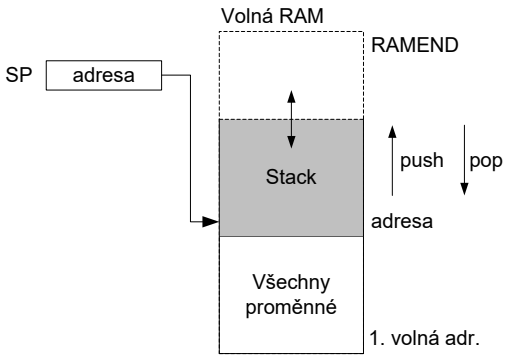
6.3 Zásobník (stack)

- Zásobník je vyhrazená oblast datové paměti.
- Obsluha zásobníku:
  - Automatická – na HW úrovni (není nutné psát kód).
  - Ruční – zajišťuje programátor nebo kompilátor vyššího programovacího jazyka.
- Co se do zásobníku ukládá:
  - při obsluze přerušení návratová adresa (CPU - automaticky)
  - při volání podprogramů:
    - návratové adresy (CPU - automaticky)
    - parametry podprogramů, návratové hodnoty (programátor v ASM, kompilátor VPJ)
  - uložení pomocných dat, lokálních proměnných (programátor v ASM, kompilátor VPJ – vyšší programovací jazyk, např. C)
- Princip:
  - Prostor pro data zásobníku
    - v RAM, umístění (obvykle) volitelné
    - vyhrazení provede programátor (v ASM) nebo kompilátor (VPJ)
  - Registr SP (stack pointer)
    - = ukazatel na vrchol zásobníku, je to registr v prostoru I/O registrů
    - Při uložení dat (`push`) se SP automaticky ++ a při vyzvednutí dat (`pop`) se SP automaticky --. Může to být i naopak, závisí na provedení.
- Dle provedení (dáno CPU):
  - Zásobník je rostoucí:
    - Směrem k vyšším adresám – např. 8051
    - Směrem k nižším adresám – např. AVR, Intel x86
  - Způsob obsluhy SP; SP ukazuje na:
    - první volnou adresu (++ po `push`, -- před `pop`)
    - poslední obsazenou adresu (++ před `push`, -- po `pop`)

- Možné strategie umístění v RAM dle směru růstu SP



Roste k **nižším** adresám

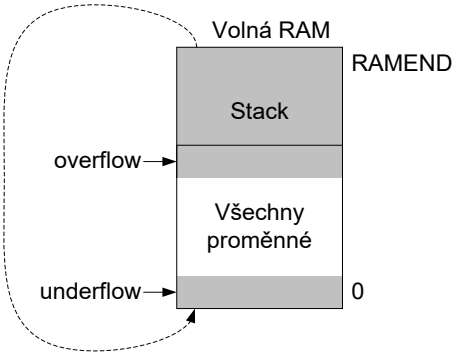


Roste k **vyšším** adresám

- instrukce pro obsluhu: `push coUlozit`, `pop kamObnovit`
  - zásobník typu LIFO (Last In First Out) → `pop` v opačném pořadí než `push`!!!

```
push   A
push   B
push   C
// …
pop    C
pop    B
pop    A
```

- Časté chyby při práci se zásobníkem v ASM
  - neinicializovaný `SP`
  - nekonzistence `push` / `pop` (počet, směr práce – viz výše)
  - přetečení / podtečení zásobníku – v ASM hlídá programátor!!!



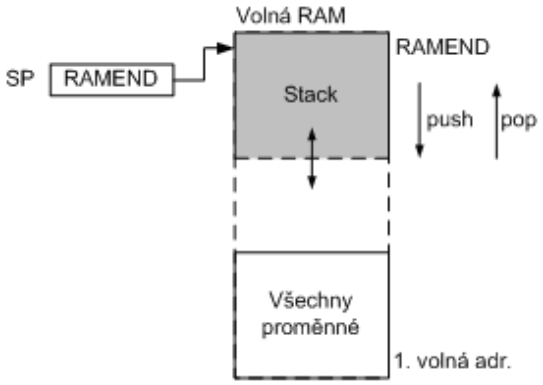
(pro zásobník rostoucí dolů)

6.3.1 Zásobník u AVR

- Zásobník je typu LIFO (poslední uložená hodnota se první vyjme).
- U AVR je ukazatel zásobníku (SP) 16-ti bitový, rozdělen na dva registry SPH a SPL. Nalézá se v I/O prostoru.

Bit	15	14	13	12	11	10	9	8	
	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

- **SP roste směrem k nižším adresám** → uložením jedné hodnoty do zásobníku se SP sníží o jedničku (u 8051 naopak !).



- **Po resetu je SP = 0x00 (kolize s pracovními registry), je nutné ho přenastavit na adresu ≥ 0x60, protože nižší adresy patří pracovním a IO registrům. Je zvykem ho nastavit na konec SRAM**:

```
ldi R16,HIGH(RAMEND)
out SPH,R16
ldi R16,LOW(RAMEND)
out SPL,R16
;Konstanta RAMEND určuje poslední adresu ve SRAM, je
;definovaná v hlavičkovém souboru a je závislá na typu
;AVR – dle toho jakou má kapacitu SRAM.
;HIGH() a LOW() jsou funkce definované překladačem, které
;vrací horní bajt výrazu (HIGH) nebo dolní bajt výrazu
;(LOW).
```

- Uložení hodnoty do zásobníku a vyjmutí hodnoty ze zásobníku:

```
push R0   ;uložení obsahu registru R0 do zásobníku
```

  - Provede
    - (SP) ← R0
    - SP--
    -

```
pop R0     ;vyjmutí hodnoty a uložení do registru R0
```

- Provede
  - SP++
  - (SP) → R0

- **SP ukazuje na první volnou adresu !**
- Vždy je nutno dbát na pravidlo kolik jsem toho uložil, tolik toho i vyberu. Přičemž vyčítání hodnot ze zásobníku je v opačném pořadí než ukládání.

```
push R0    ;uložení obsahu registru R0 do zásobníku
push R1    ;uložení obsahu registru R1 do zásobníku
pop R1     ;vyjmutí hodnoty a uložení do R1
pop R0     ;vyjmutí hodnoty a uložení do R0
```

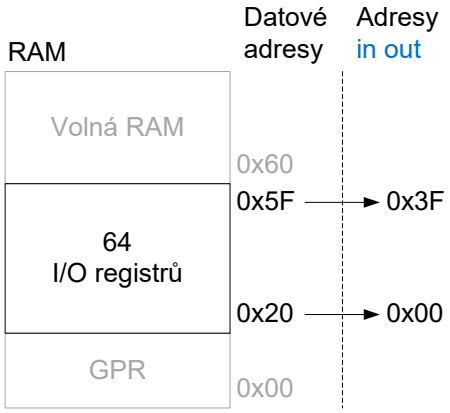
- Příklad práce se zásobníkem – uložení a vyzvednutí registru R0 a SREG

```
push r10
in   r10, SREG
push r10  ; push SREG
;    ...
pop  r10  ; pop SREG
out  SREG, r10
pop  r10
```

- Vhodné použití zásobníku v ASM je v následujících případech:
  - Všechny pracovní registry jsou již použité.
  - Volání podprogramů.
    - Záloha registrů použitých v podprogramu na zásobník.
    - Předávání parametrů do podprogramu přes zásobník.

6.4 Prostor I/O registrů – přístup k periferiím

- Všechny I/O porty (brány) a registry periferií jsou umístěny v oblasti 64 I/O registrů.
- Přístup na ně je pomocí instrukcí `in` a `out` (číslo – adresa I/O registru od 0x00 do 0x3F)
    - Alternativně lze na ně přistupovat jako do paměti SRAM pomocí instrukcí `ld` a `st` (I/O registr je dostupný na adrese ve SRAM – pozor na offset 0x20 !).



- Oblast I/O registrů v rozsahu 0x0000 až 0x001F je bitově adresovatelná (využití instrukcí pro bitový přístup `sbi, cbi, sbis, sbic`).

Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register							
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
⋮									
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR9	EEAR8
$1E ($3E)	EEARL	EEPROM Address Register Low Byte							
$1D ($3D)	EEDR	EEPROM Data Register							
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
⋮									
$05 ($25)	ADCH	ADC Data Register High Byte							
$04 ($24)	ADCL	ADC Data Register Low Byte							
$03 ($23)	TWDR	Two-wire Serial Interface Data Register							
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register							

Obrázek 53 Příklad konkrétního uspořádání I/O registrů u ATmega32

- Pro typy, které mají méně periferií, je tabulka příslušně zredukována.
- Pozor, obsazení I/O registrů na příslušných adresách se může dle typu AVR lišit! Při vlastní tvorbě programu to ovšem moc nevadí, protože se na I/O registry přistupuje pomocí jejich názvů, které jsou zaneseny v příslušném hlavičkovém souboru a namapovány na správné adresy v I/O prostoru.

- **Příklad: práce s I/O registrem UDR na ATmega32**
  - Registr `UDR` má IO adresu `0x0C`, adresu v paměti dat `0x2C`
  - soubor `m32def.inc` se musí vložit do zdrojového souboru s kódem pomocí direktivy `.INCLUDE "m32def.inc"`, tento soubor obsahuje definici symbolu `UDR` (a také dalších registrů):

    ```
    .EQU   UDR    0x0C
    ```

  - zápis do UDR (`UDR = 0x45`)

    ```
    ldi    r16, 0x45
    out    UDR, r16           // preferovaný způsob
    sts    UDR+0x20, r16      // variantně – přístup na UDR jako do RAM
    ```

  - čtení z UDR (`r0 = UDR`)

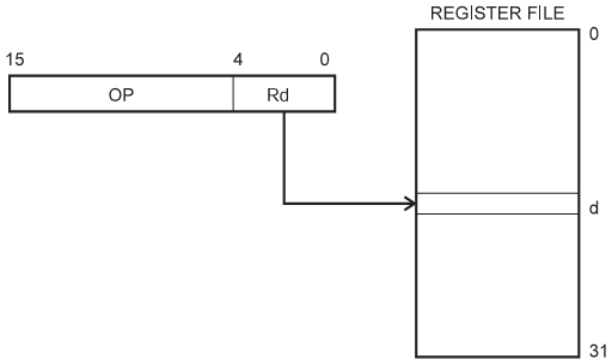
    ```
    in     r0, UDR            // preferovaný způsob
    lds    r5, UDR+0x20       // variantně – přístup na UDR jako do RAM
    ```

6.5 Adresovací módy paměti programu a dat

- AVR architektura umožňuje použít několik adresovacích módů pro přístup k paměti programu (FLASH) a k datové paměti (SRAM, pracovní registry a I/O registry)
- **Mód adresování** = způsob, jak se dostat k vlastnímu obsahu operandu
    - Implicitní – instrukce má operand (cíl) přímo v OP code
    - Konstantou – operandem je konstanta
    - Přímé – operandem je adresa
    - Nepřímé – operandem je tzv. pointer
    - Nepřímé relativní – operandem je tzv. pointer
- **Jak rozlišit / použít jednotlivé adresovací módy ?**
  Použitím správné instrukce a správného registru ! Co způsob adresování, to jiná instrukce. Nebo-li každá instrukce v sobě zahrnuje i způsob adresování.
    - Např. 4 různé instrukce pro čtení ze SRAM, dle způsobu adresování
- Následuje popis způsobů adresování s uvedením obsahu příslušné instrukce.
- Použitá symbolika:
    - OP – kód instrukce v instrukčním souboru
    - Rr – zdrojový pracovní registr (R0 až R31)
    - Rd – zdrojový a zároveň i cílový pracovní registr (R0 až R31)
    - k – přímá adresa v RAM
    - X, Y, Z – registry pro nepřímé adresování (vždy dvojice registrů, např. X = XH+XL = R26 + R27)
    - A – adresa I/O registru v I/O prostoru
    - q – 6-bit. konstanta udávající relativní posunutí u nepřímého adresování

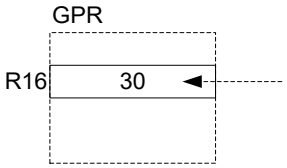
6.5.1 Přímé adresování jednoho pracovního registru

- Operand je uložen v pracovním registru Rd
- Např. instrukce `inc` Rd, `rol` Rd nebo `ldi` Rd, k



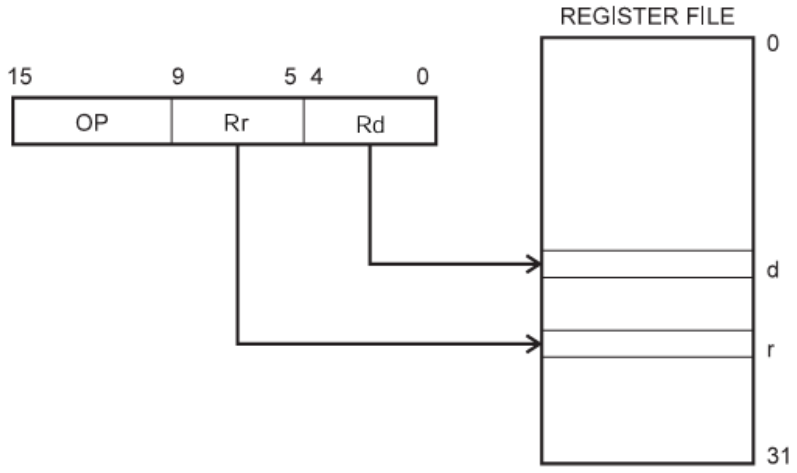
`ldi`

- "load immediately" – naplní pracovní registr konstantou
- Syntaxe: `ldi Re, k`
- Příklad: `ldi r16, 30 // r16 = 30`



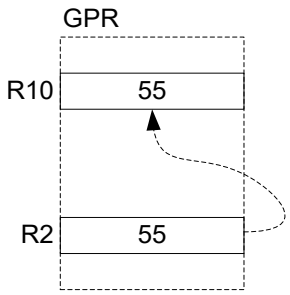
6.5.2 Přímé adresování dvou pracovních registrů

- Operand je uložen v registru Rr a Rd, výsledek je uložen zpět do registru Rd
- Např. instrukce `mov Rd,Rr` nebo `add Rd,Rr`



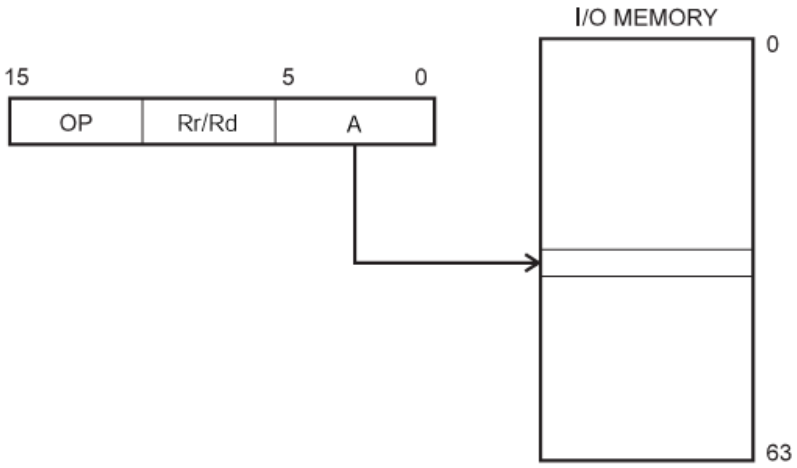
mov

- „Copy Register“
- Syntaxe: `mov Rd, Rr // rd = rr`
- Příklad: `mov r10, r2`



6.5.3 Přímé adresování I/O prostoru – přístup na periferie

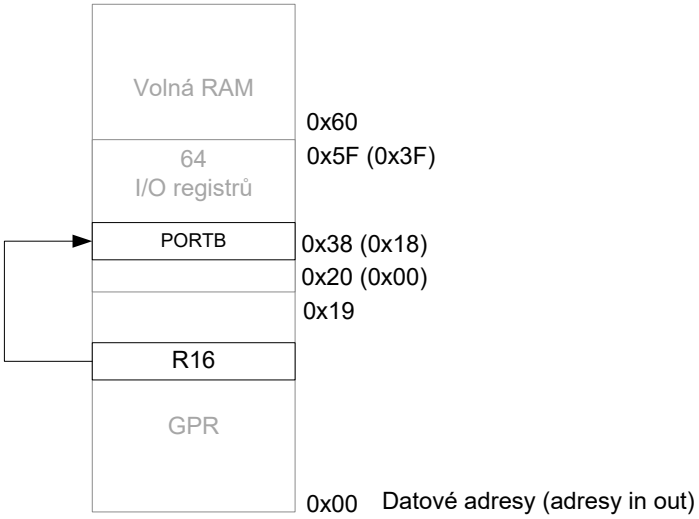
- Jedním z operandů instrukce je adresa v I/O prostoru (A)
- Druhým operandem je adresa zdrojového nebo cílového registru (Rr/Rd)
- Např. instrukce `in Rd,A` nebo `out A,Rr`



out

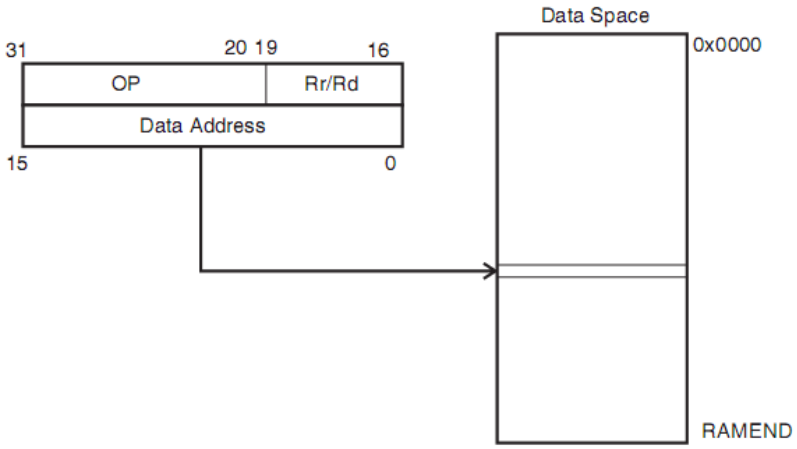
- Store register to IO location“
- Syntaxe: `out A, Rr // A = Rr`
- Příklad:

```
ldi   r16,0xFF    // r16 = 255 (0b11111111)
out   PORTB,r16   // PORTB = r16 (nelze přímo PORTB = 255)
```



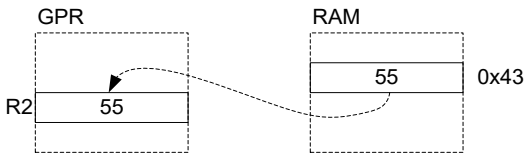
6.5.4 Přímé adresování dat (ve SRAM)

- 16-ti bitová adresa je obsažena v dolních 16-ti bitech (LSB) dvouslovové instrukce. Rd/Rr je cílový nebo zdrojový registr



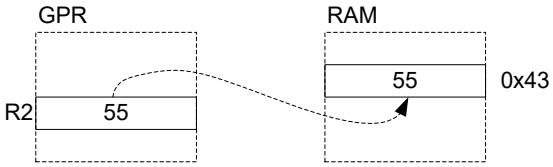
lds

- „**L**oad **D**irect from data **S**pace“
- Syntaxe: `lds Rd, k`
- Příklad: `lds r2, 0x43 // r2 = 0x43`



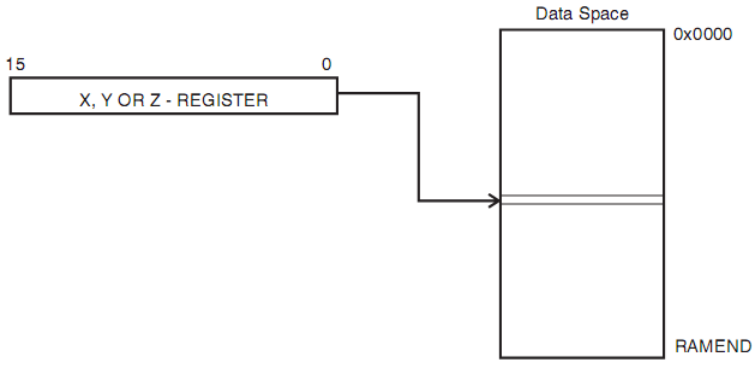
sts

- „**ST**ore direct to data **S**pace“
- Syntaxe: `sts k, Rd`
- Příklad: `sts 0x43, r2 // (0x43) = r2`



6.5.5 Nepřímé adresování dat (ve SRAM)

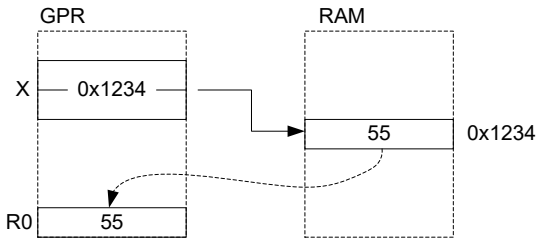
- Prostřednictvím dvojic pracovních registrů (pointerů)
  - X (XL:XH = R26:R27)
  - Y (YL:YH = R28:R29)
  - Z (ZL:ZH = R30:R31)
- Např. instrukce
  - `ld Rd,X` (načte obsah adresy uložené v reg. X do registru Rd)
  - `st X,Rr` (uloží obsah Rr na adresu v registru X),
  - stejně pro Y a Z.
- Využití pro přístup k polím, tabulkám a ukazateli na zásobník.



ld

- „Load Indirect from data space to register“
  - Syntaxe: `ld Rd, P`
  - Příklad:

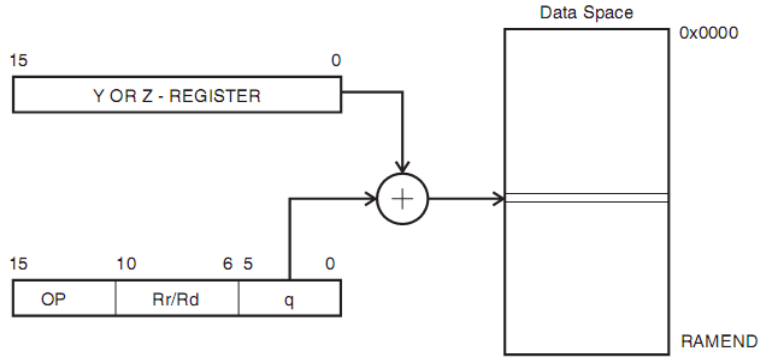
```
ldi XL, LOW(0x1234)    // r26 = 0x34
ldi XH, HIGH(0x1234)   // r27 = 0x12
ld  r0, X              // r0 = (0x1234)
```



operand X = pointer

6.5.6 Nepřímé adresování dat s posunem

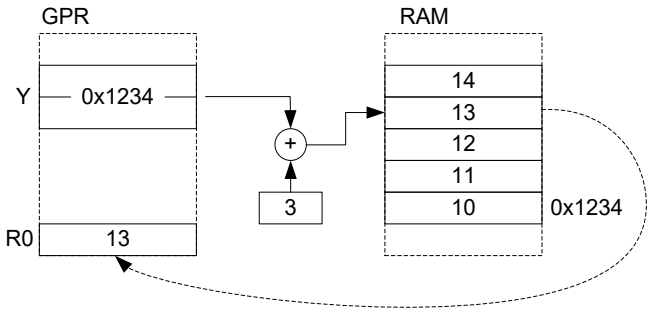
- Relativní adresování s offsetem
- Adresa operandu se získá přičtením obsahu registru **Y** nebo **Z** k adrese umístěné v dolních 6-ti bitech instrukčního slova.
- Druhým operandem je adresa zdrojového nebo cílového pracovního registru.
- Např. instrukce `ldd Rd,Y+q` nebo `std Y+q,Rr` , stejně pro Z
- Využití pro **náhodný přístup** k polím, tabulkám.



`ldd`

- Syntaxe: `ldd Rd, P+q // q je konstanta`
  - $q \in \langle 0; 63 \rangle$
- Příklad:
  - V paměti od adresy 0x1234 pole o délce 5, přístup na pole[3]

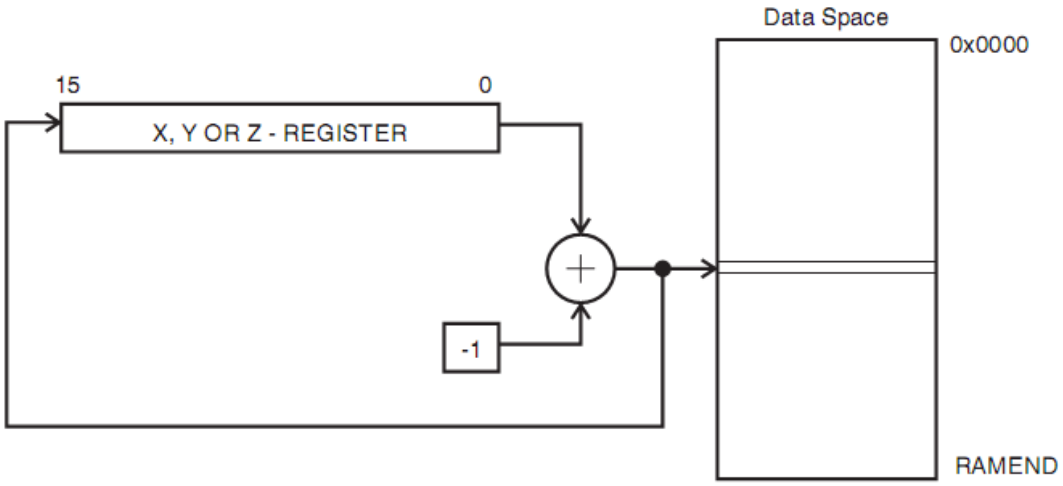
```
// pointer na zacatek pole
ldi   YH, HIGH(0x1234)
ldi   YL, LOW(0x1234)     // predchozi dva radky: R0 = pole[3]
ldd   r0, Y+3             // r0 = (0x1234 + 3)
```



  - Výpočet adresy probíhá v ALU

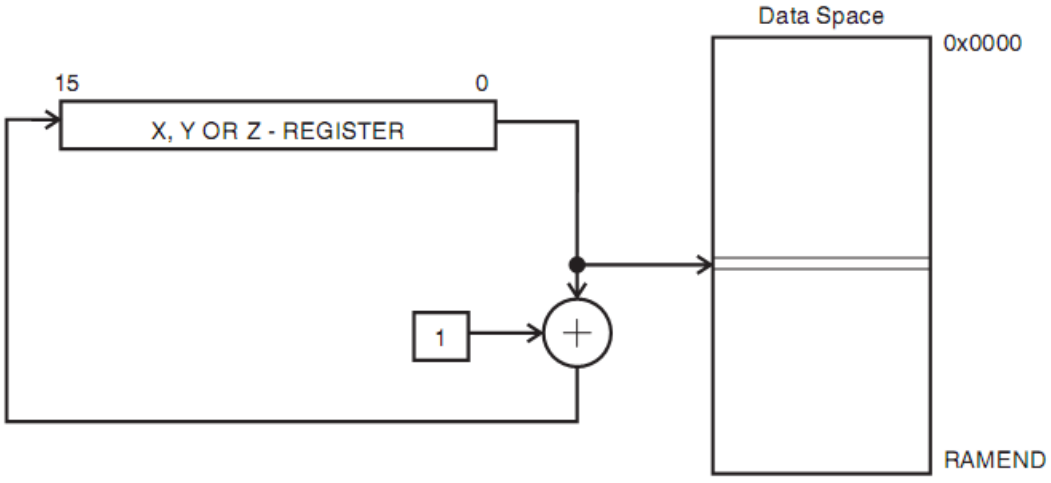
6.5.7 Nepřímé adresování dat s pre-dekrementací

- Registr **X**, **Y** nebo **Z** je před provedením operace dekrementován. Adresa v operandu je dekrementovaný obsah registru X, Y nebo Z.
- Např. instrukce `ld Rd,-X` nebo `st -X,Rr` , stejně pro Y a Z
- Využití pro **sekvenční přístup** k polím, tabulkám a ukazateli na zásobník.



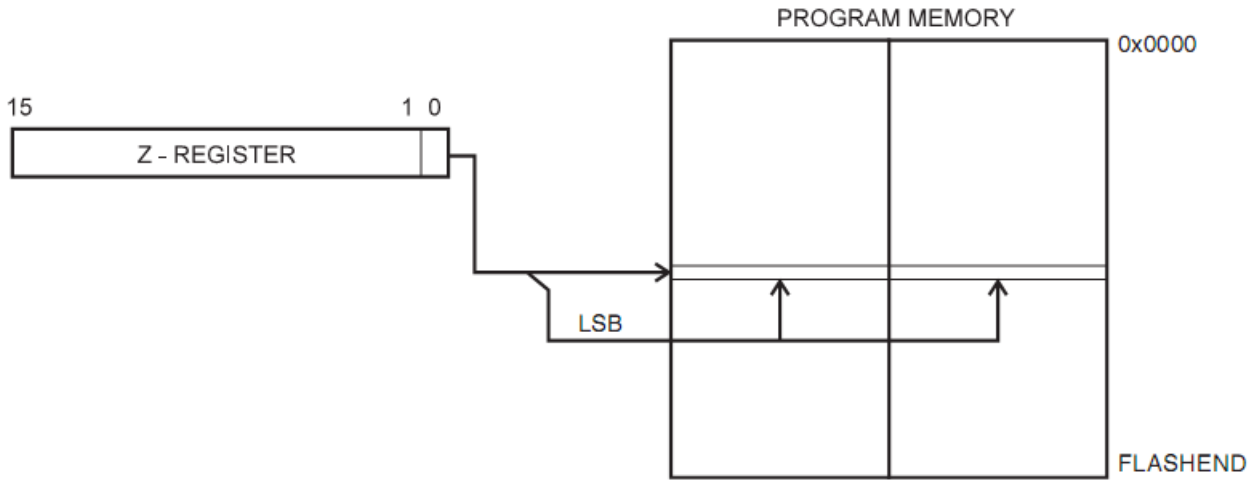
6.5.8 Nepřímé adresování dat s post-inkrementací

- Registr **X**, **Y** nebo **Z** je inkrementován po provedení operace. Adresa v operandu je obsah registru X, Y nebo Z před inkrementací.
- Např. instrukce `ld Rd,X+` nebo `st X+,Rr` , stejně pro Y a Z
- Využití pro **sekvenční přístup** k polím, tabulkám a ukazateli na zásobník.



6.5.9 Přímé adresování konstanty v paměti programu

- Adresa konstanty zapsaná v registru **Z**.
- Např. instrukce `lpm` (naplní registr R0 obsahem paměti programu kam ukazuje Z)
- V Z musí být uložena bajtová adresa do paměti programu, ačkoliv je paměť programu organizovaná po slovech délky 2 bajtů
  - Jelikož je paměť programu organizovaná po slovech, tak 15 horních bitů v registru Z adresuje slovo v paměti programu (určuje řádek), nultý bit vybírá horní (lsb = 1) nebo dolní (lsb = 0) bajt slova (určuje sloupec). Viz obrázek níže.
- Mimo AVR základní řady jsou možné další instrukce:
  - `lpm Rd,Z` - můžeme určit cílový registr pro zápis hodnoty
  - `lpm Rd,Z+` - můžeme určit cílový registr, ukazatel Z se automaticky postinkrementuje
  - `spm` – zápis jednoho slova do paměti programu (zdrojem je R0:R1)
- Využití pro vyčítání tabulek, řetězců apod. uložených v paměti programu.

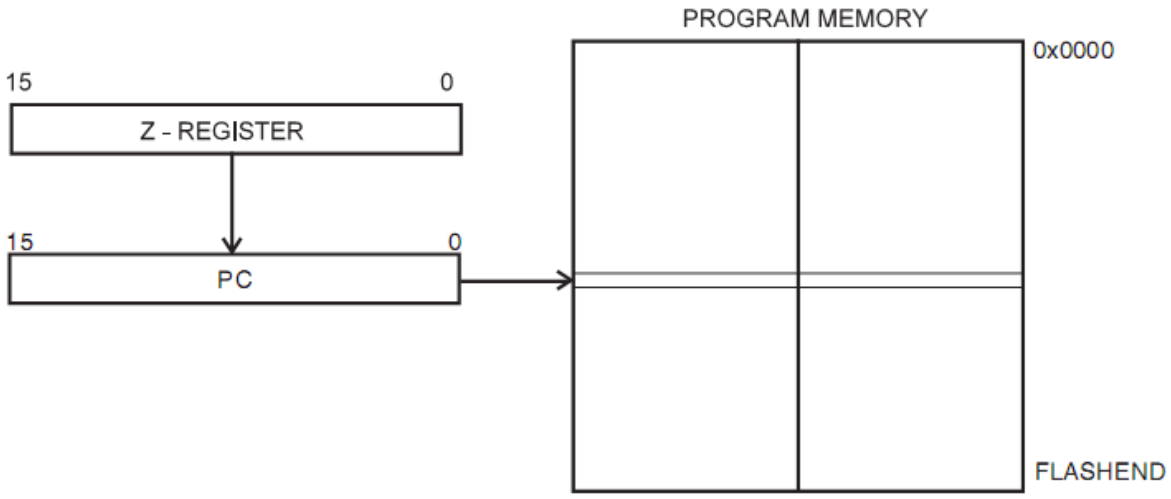


Example:

```
        ldi   ZH, high(Table_1<<1); Initialize Z-pointer
        ldi   ZL, low(Table_1<<1)
        lpm   r16, Z              ; Load constant from Program
                                  ; Memory pointed to by Z (r31:r30)
...
Table_1:
.dw 0x5876                        ; 0x76 is addresses when Z_LSB = 0
                                  ; 0x58 is addresses when Z_LSB = 1
...
```

6.5.10 Nepřímé adresování paměti programu

- Provádění programu pokračuje na adrese obsažené v registru **Z**, tj. programový čítač PC je naplněn obsahem registru Z.
- instrukce `IJMP` a `ICALL`.



Example:

```
mov     r30,r0    ; Set offset to jump table
ijmp              ; Jump to routine pointed to by r31:r30
```

Words: 1 (2 bytes)
Cycles: 2

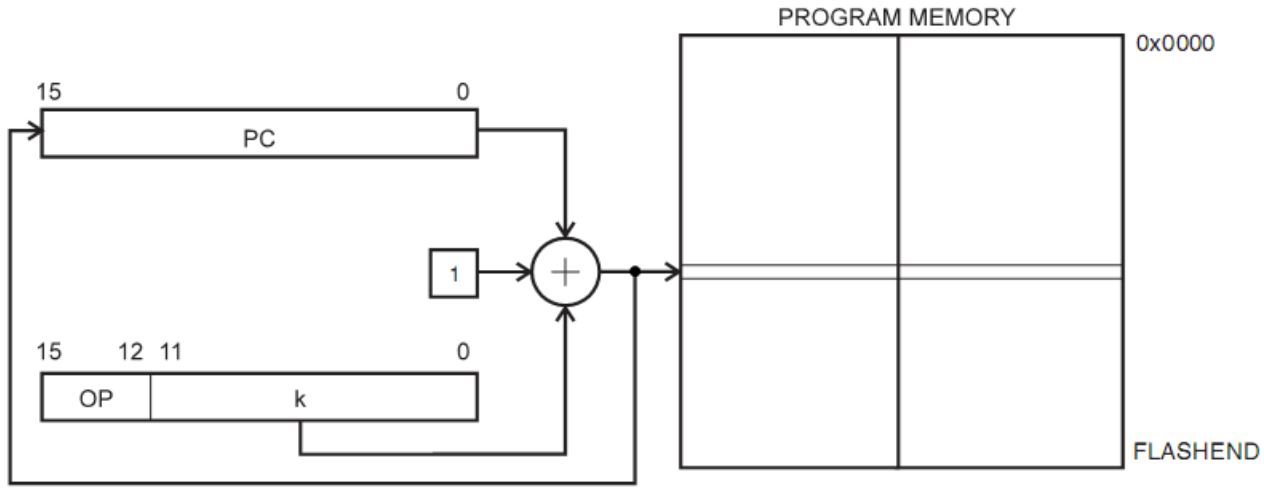
Example:

```
mov       r30,r0    ; Set offset to call table
icall               ; Call routine pointed to by r31:r30
```

Words: 1 (2 bytes)
Cycles: 3 devices with 16 bit PC
4 devices with 22 bit PC

6.5.11 Relativní adresování v paměti programu

- Provádění programu pokračuje na adrese PC + k + 1. Relativní adresa k může být v rozsahu -2048 až +2047.
- instrukce `RJMP k` a `RCALL k`
  - relative jump / relative call



Example:

```
            cpi     r16,$42   ; Compare r16 to $42
            brne    error     ; Branch if r16 <> $42
            rjmp    ok        ; Unconditional branch
error:      add     r16,r17   ; Add r17 to r16
            inc     r16       ; Increment r16
ok:         nop               ; Destination for rjmp (do nothing)
```

Words: 1 (2 bytes)
Cycles: 2

Example:

```
            rcall   routine   ; Call subroutine
            ...
routine:    push    r14       ; Save r14 on the Stack
            ...
            pop     r14       ; Restore r14
            ret               ; Return from subroutine
```

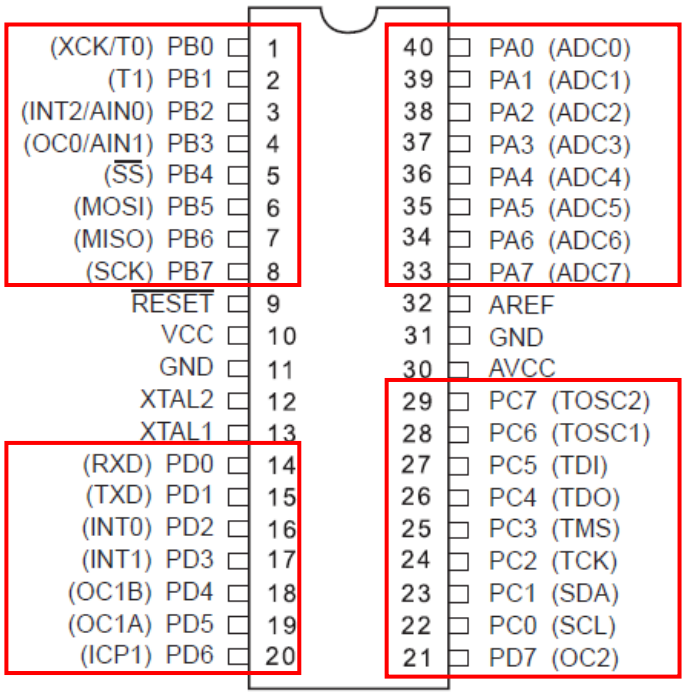
Words: 1 (2 bytes)
Cycles: 3 devices with 16-bit PC
4 devices with 22-bit PC

6.5.12 Souhrn

- pro přístup k I/O registrům, paměti dat a pro přesun dat mezi pracovními registry se používají rozdílné instrukce – odlišnost oproti CISC procesorům (např. 8051), kde jsme si vystačili s instrukcí MOV
- registry X, Y, Z slouží pro nepřímé adresování paměti dat (slouží jako ukazatele)
- registr Z slouží také pro nepřímé adresování paměti programu
- lze využít pre-dekrementaci a post-inkrementaci ukazatele

7 Vstupně-výstupní porty

- MCU AVR mohou mít dle typu až 10 8-bitových obousměrných bran označených PORT A až PORT L (některá písmena jsou v řadě vynechána)

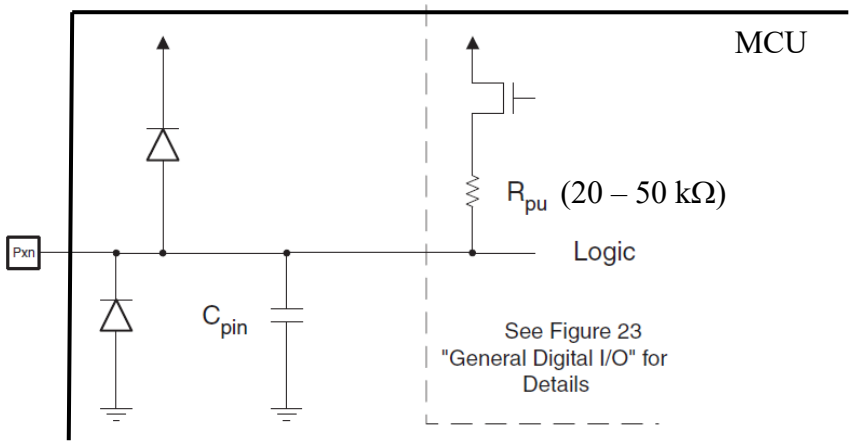


Obrázek 54 I/O porty u ATmega32

- Jednotlivé piny portů mohou sloužit jako obecně vstupně/výstupní piny nebo mohou mít alternativní funkci (pin je spojen s nějakou periferií, např. komunikační linkou, časovačem, A/D převodníkem apod.). Alternativní funkce je u jména pinu uvedena v závorce, např. PA0 (ADC0) znamená, že pin PA0 portu A slouží alternativně jako analogový vstup (kanál č. 0) A/D převodníku.

7.1 Vlastnosti

- Pin může být nastaven jako vstupní nebo výstupní (směr se určuje programově).
- Každý pin je na vstupu chráněn diodami proti zemi GND a proti napájecímu napětí Vcc. Diodami je realizována ochrana proti přepětí - napětí na pinu nesmí být větší než Vcc + 0,5 a nesmí být menší než -0,5 V proti GND. Max. proud diod je 1 mA. Aby nedošlo ke zničení diod při přepětí, je nutné vložit do série se vstupem rezistor a tím omezit poruchový proud – podrobnosti viz kapitola „Přepěťová ochrana vstupů“.



Obrázek 55 Zjednodušená interní struktura pinu každého portu

28.1 Absolute Maximum Ratings

Stresses beyond those listed under “Absolute Maximum Ratings” may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

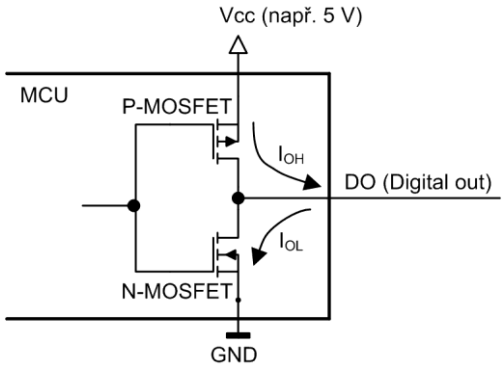
Parameters	Min.	Typ.	Max.	Unit
Operating temperature	–55		+125	°C
Storage temperature	–65		+150	°C
Voltage on any pin except $\overline{\text{RESET}}$ with respect to ground	–0.5		V_{CC} + 0.5	V
Voltage on RESET with respect to ground	–0.5		+13.0	V
Maximum operating voltage		6.0		V
DC current per I/O pin		40.0		mA
DC current V_{CC} and GND pins		200.0		mA
Injection current at V_{CC} = 0V		±5.0(1)		mA
Injection current at V_{CC} = 5V		±1.0		mA

Note: 1. Maximum current per port = ±30mA

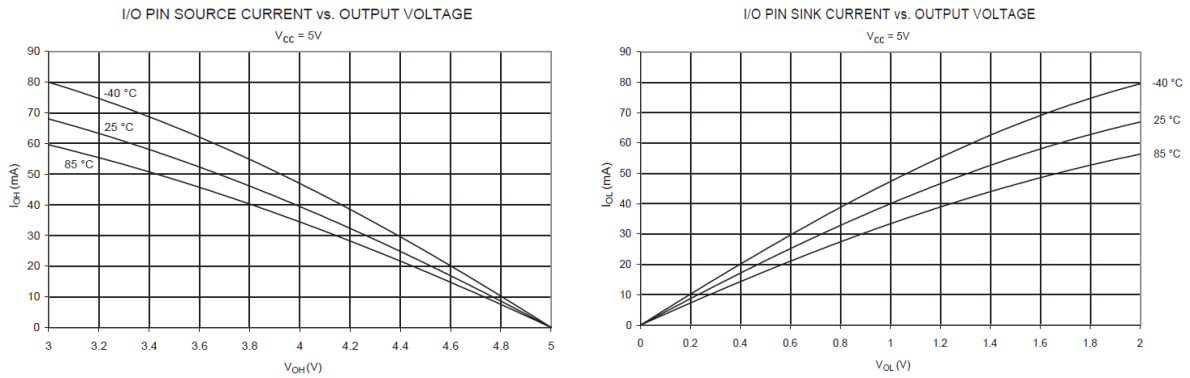
Obrázek 56 Max. proud ochrannými diodami na vstupu pinu (MCU AVR)

- **Pokud je pin nastaven jako vstupní** (Obrázek 55), může se programově povolit interní pull-up rezistor (cca 20 – 50 kΩ), který na pinu definuje logickou úroveň 1 (HIGH), pokud k pinu není z vnějšku nic připojeno (používá se typicky pro připojení tlačítka nebo propojení se zařízením, které je vybaveno výstupem typu otevřený kolektor). Vstupní odpor pinu v ustáleném stavu je velký, proud tekoucí do pinu je obvykle v řádu do 1 µA.

- **Pokud je pin nastaven jako výstupní** (Obrázek 57), uplatňuje se výstupní komplementární (push-pull) dvojice MOSFET tranzistorů (CMOS logic = „complementary MOS“), která definuje logickou úroveň na pinu. V jeden okamžik je sepnut pouze jeden tranzistor z komplementární dvojice (buď horní tranzistor – výstup v logické 1 nebo dolní tranzistor – výstup v logické 0). Výstup snese proudové zatížení až 40 mA v logické úrovni 0 (tzv. proud I_{OL} – proud teče ze zátěže směrem do pinu a přes dolní tranzistor z push-pull dvojice následně do záporného pólu napájení, tzv. „**sink**“) i v logické úrovni 1 (tzv. proud I_{OH} – proud teče ven z pinu přes horní tranzistor z push-pull dvojice, proud je odebírán z kladného pólu napájení - napájecího pinu Vcc mikrokontroléru, tzv. „**source**“). Výstupy dovolují tedy připojit přímo např. 20 mA LED. S rostoucím proudem klesá napětí na pinu v logické 1 (proud vytéká z pinu – „**source**“), resp. roste napětí na pinu v logické 0 (proud teče do pinu – „**sink**“) – viz zatěžovací charakteristika uvedená na následujících obrázcích.



Obrázek 57 Obvyklé vnitřní zapojení jednoho digitálního výstupu (PUSH-PULL, CMOS)



Obrázek 58 Zatěžovací charakteristika výstupního pinu

- Maximální odběr proudu z jednoho portu nesmí překročit 100 mA (I_{OH} i I_{OL}).
- Maximální odběr proudu ze všech portů nesmí překročit 400 mA (I_{OH} i I_{OL}).
- **Každá brána (port) má tři ovládací registry** – datový registr **PORTx** , směrový registr **DDRx** a registr **PINx** nesoucí informaci o skutečné log. úrovni pinů portu.

7.2 Nastavení portu

- spočívá v zapsání hodnot do registrů PORTx a DDRx
- určujeme zda bude pin určitého portu vstupní nebo výstupní a zda bude v log. 0 nebo v log. 1, případně bude-li aktivován interní pull-up rezistor či nikoliv

7.2.1 DDRx registr

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- určuje zda pin bude vstupní (DDRx.n = 0) nebo výstupní (DDRx.n = 1)

7.2.2 PORTx registr

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- určuje jaká bude úroveň na daném pinu, je-li pin nastaven jako výstupní
- určuje zda bude aktivován pull-up, je-li pin nastaven jako vstupní (pull-up je aktivní když PORTx.n = 1, jinak pull-up není aktivován)

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Obrázek 59 Možné kombinace nastavení vývodu

- zápisem 1 do bitu PUD v registru SFIOR lze globálně zakázat pull-up rezistory na všech portech

7.2.3 PINx registr

Port B Input Pins Address – PINB

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

- obsahuje informaci o fyzickém stavu pinů daného portu
- určen pouze pro čtení

Příklad nastavení portu a čtení logických úrovní na portu

Assembly Code Example[1]

```
...
; Define pull-ups and set outputs high
; Define directions for port pins
ldi   r16,(1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi   r17,(1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out   PORTB,r16
out   DDRB,r17
; Insert nop for synchronization
nop
; Read port pins
in    r16,PINB
...
```

C Code Example[1]

```
unsigned char i;
...
/* Define pull-ups and set outputs high */
/* Define directions for port pins */
PORTB = (1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0);
DDRB = (1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0);
/* Insert nop for synchronization*/
_NOP();
/* Read port pins */
i = PINB;
...
```

- Lze použít i bitový přístup na jednotlivé piny portu:
  zápis 1 na pin: `sbi A,b` kde `b` je číslo pinu (0 až 7)
  zápis 0 na pin: `cbi A,b` kde `A` je název registru portu (např. PORTA, DDRA)
- Zápis si lze usnadnit zavedením maker, ukázka pro jazyk C:

```
#define sbit(val, bit)     ((val) |= (1 << (bit)))
#define cbit(val, bit)     ((val) &= ~(1 << (bit)))
#define tglbit(val, bit)   ((val) ^= (1 << (bit)))
#define rdbit(val, bit)    (((val) >> (bit)) & 0x01)
#define wrbit(val, bit, bitvalue) (bitvalue ? sbit(val, bit) : cbit(val, bit))
```

7.3 Alternativní funkce portů/pinů

- Většina portů obsahuje piny s alternativními funkcemi.
- Aktivace alternativní funkce se provede v příslušném řídícím registru dané periferie, které tento pin náleží (např. povolením externího přerušení INT0 dojde k propojení pinu s alternativní funkcí INT0 s logikou přerušení).
- Při aktivaci alternativní funkce se i přesto u většiny pinů musí nastavit směr (určit zda je pin vstupní či výstupní zápisem do registru DDRx) !

Výčet alternativních funkcí na portech ATmega32

Port A Pins Alternate Functions

Port Pin	Alternate Function
PA7	ADC7 (ADC input channel 7)
PA6	ADC6 (ADC input channel 6)
PA5	ADC5 (ADC input channel 5)
PA4	ADC4 (ADC input channel 4)
PA3	ADC3 (ADC input channel 3)
PA2	ADC2 (ADC input channel 2)
PA1	ADC1 (ADC input channel 1)
PA0	ADC0 (ADC input channel 0)

Port B Pins Alternate Functions

Port Pin	Alternate Functions
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	$\overline{SS}$ (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negative Input) OC0 (Timer/Counter0 Output Compare Match Output)
PB2	AIN0 (Analog Comparator Positive Input) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

Port D Pins Alternate Functions

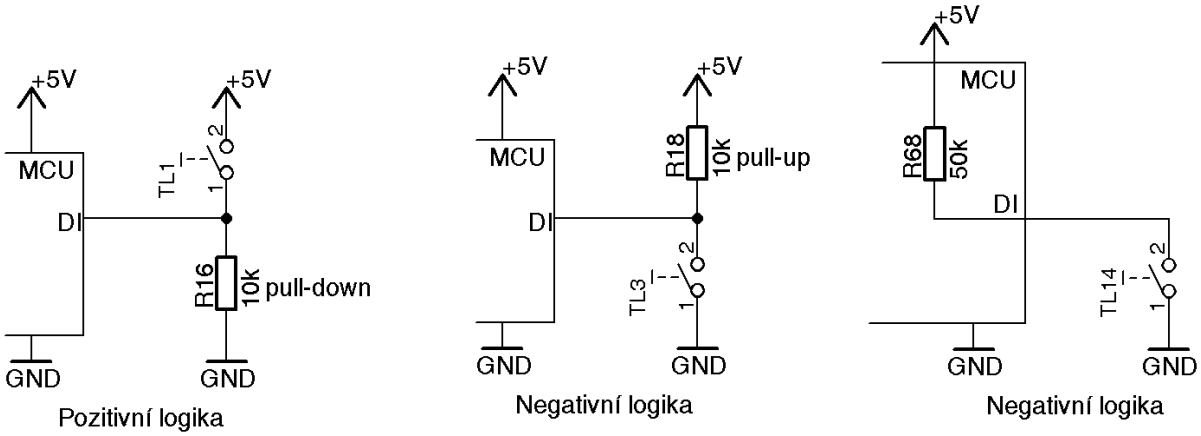
Port Pin	Alternate Function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP1 (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

8 Zapojení uživatelského vstupu k DI

Základní zásady:

- Pin MCU musí být nastaven jako vstupní.
- Vstupní odpor pinu je obvykle větší než 1 MΩ (do pinu teče proud řádově 1 μA).
- Vždy musí být definována logická úroveň na vstupním pinu (napětí na pinu nesmí „plavat“).
    - Pomocí pull-up nebo pull-down rezistoru.
    - Lze využít interní pull-up rezistor v MCU, pokud je jím MCU vybaveno.
- Pozitivní vs. negativní logika
    - Pozitivní logika = logická 1 při vybuzení (např. stisknuté tlačítko)
    - Negativní logika = logická 0 při vybuzení (např. stisknuté tlačítko)
- Ošetření zákmitů u mechanických tlačítek a relé
    - hardwarově
    - softwarově
- Připojení více tlačítek (nebo obecně zdrojů signálu)
    - Nejlépe všechna k jednomu portu – jednodušší čtení stavů v programu
    - Přímo – 1 tlačítko = 1 pin na MCU
    - Nepřímo – při větším počtu tlačítek, pro úsporu pinů na MCU, různé možnosti:
        - Kodér
        - Multiplexer
        - Posuvný registr
        - A/D převodník
        - Maticová klávesnice

8.1 Přímé připojení k digitálnímu vstupu MCU



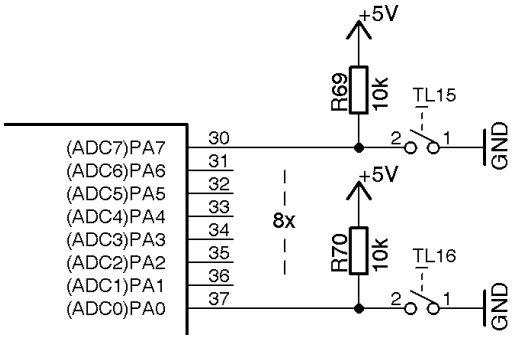
Obrázek 60 Přímé připojení jednoho tlačítka k MCU

Velikost pull-up nebo pull-down rezistoru:

- Čím je odpor menší, tím lepší je odolnost vůči rušení.
- Minimální odpor je dán max. povoleným proudem tlačítka.
- Obvykle 1ky až 10ky kΩ (proud tlačítkem cca 1 mA).

8.2 Připojení většího množství tlačítek

8.2.1 Přímé připojení

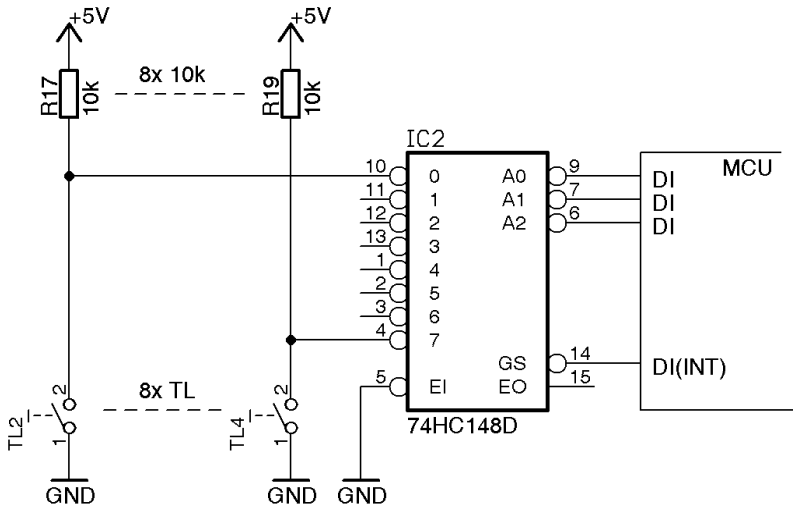


Obrázek 61 Přímé připojení více tlačítek k MCU

- Každý bit v registru PINA odpovídá jednomu tlačítku.
- Lze zjistit stisk více tlačítek najednou.
- Pokud např. PINA = 0xF6 (0b11110110) → jsou stisknuta tlačítka připojená k pinům PA0 a PA3.

8.2.2 Nepřímé připojení – pro úsporu pinů MCU, více možností

8.2.2.1 Prioritní kodér



Obrázek 62 Připojení více tlačítek pomocí prioritního kodéru

- Prioritní kodér (např. 74x148) přiřazuje každému tlačítku kód.
    - Kód tlačítka se objeví na výstupech A0:2.

- Prioritní = na výstupu kodéru se objeví kód tlačítka s nejvyšší prioritou (nelze rozpoznat stisk více tlačítek najednou).
- Nejvyšší prioritu má tlačítko připojené ke vstupu č. 7

FUNCTION TABLE

	INPUTS								OUTPUTS				
	EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
	H	X	X	X	X	X	X	X	X	H	H	H	H	H
žádné tl. nestisknuto	L	H	H	H	H	H	H	H	H	H	H	H	H	L
některé tl. stisknuto	L	X	X	X	X	X	X	X	L	L	L	L	L	H
	L	X	X	X	X	X	X	L	H	L	L	H	L	H
	L	X	X	X	X	X	L	H	H	L	H	L	L	H
	L	X	X	X	X	L	H	H	H	L	H	H	L	H
	L	X	X	X	L	H	H	H	H	H	L	L	L	H
	L	X	X	L	H	H	H	H	H	H	L	H	L	H
	L	X	L	H	H	H	H	H	H	H	H	L	L	H
	L	L	H	H	H	H	H	H	H	H	H	H	L	H

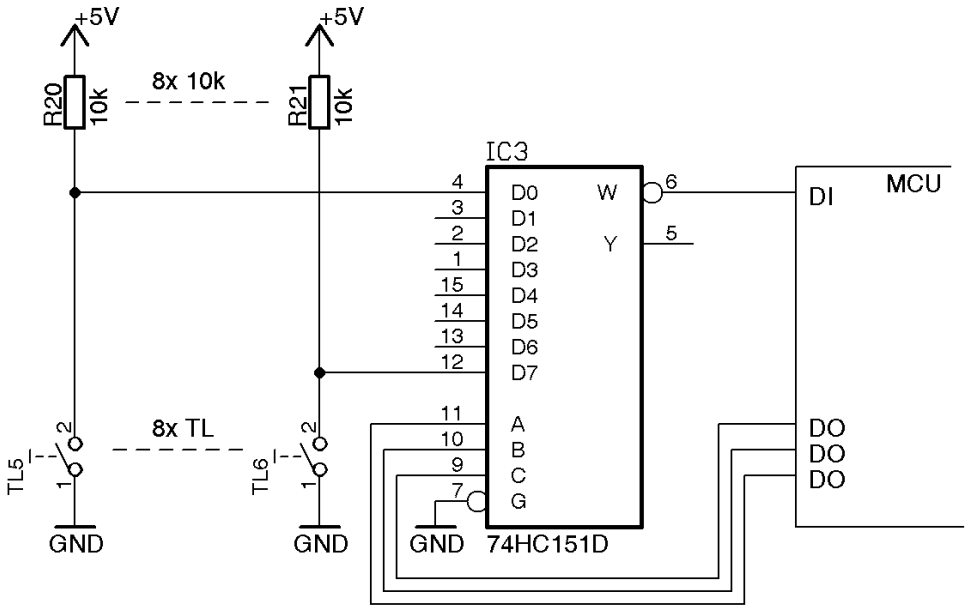
Obrázek 63 Tabulka činnosti prioritního kodéru 74HC148

- Význam pinů prioritního kodéru:
  - EI – povoluje (pin v úrovni L) nebo zakazuje (pin v úrovni H) činnost kodéru
  - Vstupy 0 až 7 – pro připojení logických vstupních signálů, které se mají zakódovat
  - Výstupy A0 až A2 – kód odpovídající logické kombinaci na vstupech 0 až 7
  - GS – signalizuje (GS = L), že alespoň na jednom vstupu je úroveň L (že je stisknuté alespoň jedno tlačítko). Signál lze zavést na pin externího přerušení MCU a obsloužit klávesnici až v okamžiku, kdy došlo ke stisku tlačítka (viz Obrázek 62).
  - EO – pro kaskádní řazení kodérů. Výstup EO je v úrovni L pokud EI = L a není stisknuto žádné tlačítko (na všech vstupech je úroveň H). Výstup EO pak připojíme do vstupu EI dalšího kodéru (tím povolujeme jeho funkci).
- Způsob čtení tlačítek v programu:
  - Periodicky (např. v nekonečné smyčce) čteme stav výstupů A0:2.
  - Nebo využíváme externího přerušení – stav výstupů A0:2 čteme v obsluze přerušení (obsluha přerušení na sestupnou hranu nebo L úroveň signálu GS).
  - V programu máme zaneseny kódy všech tlačítek (např. ve formě symbolických konstant) vůči kterým porovnáváme přečtený kód.

Tabulka 1 Tabulka kódu tlačítek (prioritní kodér)

Tlačítko	Kód (BIN)	Kód (DEC)
TL0 (vstup 0)	0b111	7
TL1	0b110	6
TL2	0b101	5
TL3	0b100	4
TL4	0b011	3
TL5	0b010	2
TL6	0b001	1
TL7 (vstup 7)	0b000	0

8.2.2.2 Multiplexer



Obrázek 64 Připojení více tlačítek pomocí multiplexeru

- Na výstupu /W poskytuje multiplexer negovaný stav vstupu D0 až D7, který je zrovna adresován vstupy A,B,C (kde A je nejméně významný bit adresy). Tj. v našem případě stisknutému tlačítku odpovídá úroveň H na výstupu /W.

Tabulka 2 Tabulka činnosti multiplexeru 74HC151

FUNCTION TABLE

INPUTS												OUTPUTS	
$\overline{E}$	S_2	S_1	S_0	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$\overline{Y}$	Y
H	X	X	X	X	X	X	X	X	X	X	X	H	L
L	L	L	L	L	X	X	X	X	X	X	X	H	L
L	L	L	L	H	X	X	X	X	X	X	X	L	H
L	L	L	H	X	L	X	X	X	X	X	X	H	L
L	L	L	H	X	H	X	X	X	X	X	X	L	H
L	L	H	L	X	X	L	X	X	X	X	X	H	L
L	L	H	L	X	X	H	X	X	X	X	X	L	H
L	L	H	H	X	X	X	L	X	X	X	X	H	L
L	L	H	H	X	X	X	H	X	X	X	X	L	H
L	H	L	L	X	X	X	X	L	X	X	X	H	L
L	H	L	L	X	X	X	X	H	X	X	X	L	H
L	H	L	H	X	X	X	X	X	L	X	X	H	L
L	H	L	H	X	X	X	X	X	H	X	X	L	H
L	H	H	L	X	X	X	X	X	X	L	X	H	L
L	H	H	L	X	X	X	X	X	X	H	X	L	H
L	H	H	H	X	X	X	X	X	X	X	L	H	L
L	H	H	H	X	X	X	X	X	X	X	H	L	H

Notes

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care.

- Význam pinů multiplexeru:
  - E – povoluje (pin v úrovni L) nebo zakazuje (pin v úrovni H) činnost obvodu
  - Adresní vstupy S0 až S2 (A až C) – adresa datového vstupu (vybírá vstup D0 až D7)
  - Datové vstupy I0 až I7 (D0 až D7) – pro připojení logických vstupních signálů, které se mají zakódovat
  - Výstup /Y (W) – negovaný stav vybraného vstupu
  - Výstup Y – přímý stav vybraného vstupu

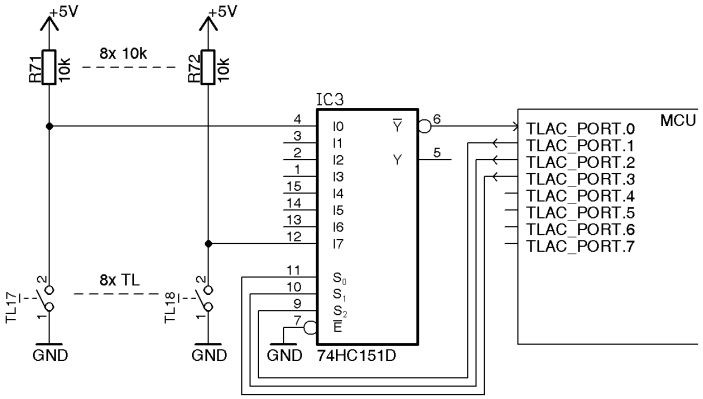
- Způsob čtení tlačítek v programu:
  - Na vstupy A:C multiplexeru vystavíme adresu tlačítka a na výstupu /W multiplexeru přečteme stav adresovaného tlačítka.
  - Stav tlačítka uložíme do proměnné „stav tlačítek“ na příslušnou bitovou pozici, která odpovídá danému tlačítku.
  - Každé tlačítko má v programu svůj kód, který odpovídá pozici bitu v proměnné „stav tlačítek“. Vůči těmto kódům porovnáváme přečtený kód.

Ukázkový kód pro čtení stavu tlačítek přes multiplexer.

```
include <avr/io.h>

/*
        Multiplexer musi byt pripojen k MCU takto:
        TLAC_PORT.0 - vystup /Y (W) - pin na MCU vstupni
        TLAC_PORT.1 - adresni vstup S0 (A) - pin na MCU vystupni
        TLAC_PORT.2 - adresni vstup S1 (B) - pin na MCU vystupni
        TLAC_PORT.3 - adresni vstup S2 (C) - pin na MCU vystupni
*/

/* pripojeni multiplexeru k MCU */
#define TLAC_PORT       PORTB
#define TLAC_DDR        DDRB
#define TLAC_PIN        PINB
/* kody tlacitek */
#define TLAC_NESTISKNUTO        0       // 0
#define TLAC_0                  (1<<0)  // 1
#define TLAC_1                  (1<<1)  // 2
#define TLAC_2                  (1<<2)  // 4
#define TLAC_3                  (1<<3)  // 8
#define TLAC_4                  (1<<4)  // 16
#define TLAC_5                  (1<<5)  // 32
#define TLAC_6                  (1<<6)  // 64
#define TLAC_7                  (1<<7)  // 128
```




```
/* Cte stav 8-mi tlacitek pripojenych k multiplexeru.
        parametry:      zadne
        vraci:          byte, ve kterem je stav 8-mi tlacitek, co bit to jedno tlacitko
                        bit v H odpovida stisknutemu tlacitku
*/
uint8_t cti_tlacitka(void)
{
        uint8_t stav_tlacitek = 0; // priznaky vynulovane = zadne tlacitko neni stisknute

        for (uint8_t adresa=0;adresa<8;adresa++) // postupne vystavovani adresy a cteni stavu tlacitka
        {
                TLAC_PORT = adresa<<1;  // adresa se na portu vystavuje az od pinu TLAC_PORT.1
                stav_tlacitek |= (TLAC_PIN & 0x01)<<adresa;     // nastaveni priznaku stisku tlacitka
        }
        return stav_tlacitek;
}

int main(void)
{
    uint8_t tlacitka;

        // nastaveni smeru pinu na portu
        TLAC_DDR = 0b00001110;  // pin TLAC_PORT.0 vstupni, piny TLAC_PORT.1 az 3 vystupni, zbyle piny
vstupni

        while(1)
        {
                // nejaky dalsi kod

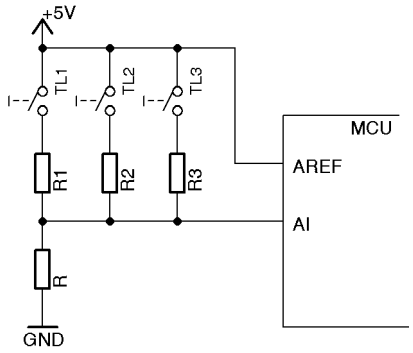
                tlacitka = cti_tlacitka();      // cteni stavu tlacitek
                switch (tlacitka)               // reakce na stisk tlacitka
                {
                        case TLAC_0:    // reakce na stisk tlacitka c.0
                                // nejaky kod
                                break;
                        case TLAC_7:    // reakce na stisk tlacitka c.7
                                // nejaky kod
                                break;
                        case (TLAC_0 | TLAC_7): // reakce na soucasny stisk tlacitek c.0 a c.7
                                // nejaky kod
                                break;
                }

                // nejaky dalsi kod
        }
}
```

Obsah proměnné `tlacitka`								
Bit	7	6	5	4	3	2	1	0
Tlačítko	7	6	5	4	3	2	1	0

8.2.2.3 A/D převodník

- Pro úsporu pinů MCU potřebných pro připojení tlačítek lze využít interní A/D převodník MCU.



Obrázek 65 Připojení více tlačítek pomocí A/D převodníku

- Tlačítka se zapojí do odporové sítě, kterou se realizuje napěťový dělič s proměnným dělícím poměrem podle stisknutého tlačítka a napětí na výstupu děliče se zavede do jednoho analogového vstupu A/D převodníku.
- Po stisku tlačítka se na analogovém vstupu MCU objeví odpovídající napětí. Toto napětí je A/D převodníkem převedeno na číslo. Pro 10-bit A/D převodník platí:

$$ADC = \frac{AI \cdot 2^{10}}{AREF} = \frac{AI \cdot 1024}{AREF} \tag{2}$$

Kde: ADC – napětí převedené na číslo, rozsah 0 až 1023 (odpovídá napětí 0 až AREF – AREF/1024 [V]), kvantizační krok v napětí je AREF/1024 [V], pokud je AREF = 5 V, je kvantizační krok 5 V / 1024 = 4,88 mV

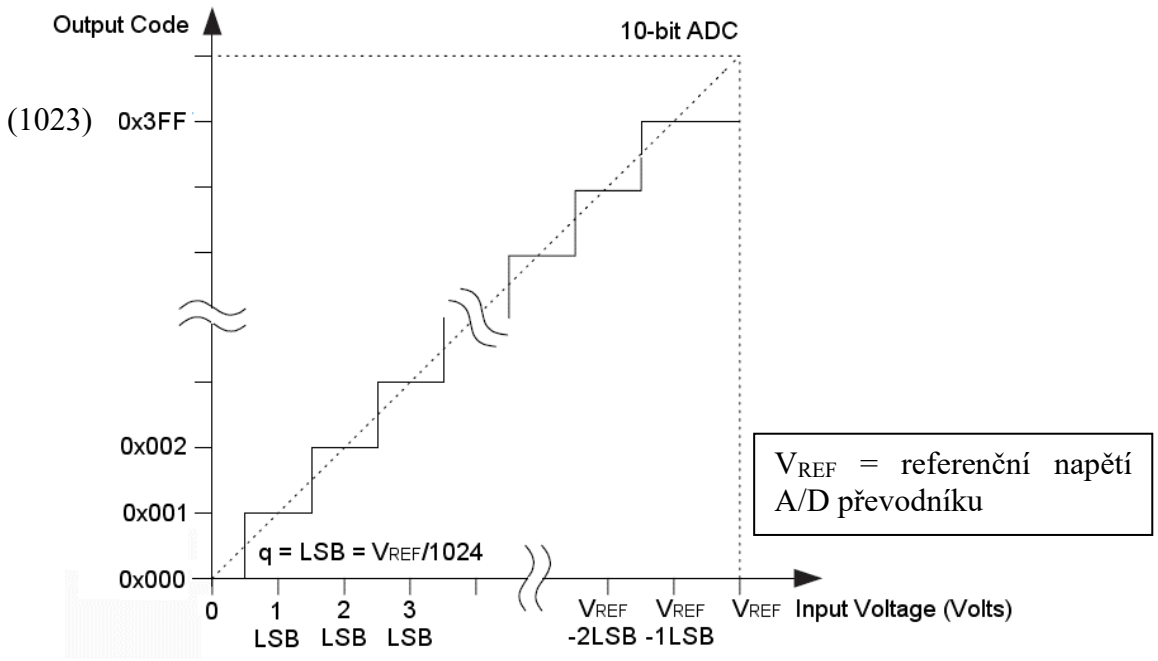
AI – napětí na analogovém vstupu A/D převodníku [V]

AREF – referenční napětí A/D převodníku [V], např. 5 V

Tabulka 3 Vztah mezi vstupním napětím a výstupním kódem 10-bit A/D převodníku

Napětí (AI)	Číslo (ADC)
0 V	0
1/1024 * AREF = 4,88 mV	1
2/1024 * AREF = 9,76 mV	2
.	
.	
512/1024 * AREF = 2,500 V	512
.	
.	
1022/1024 * AREF = 4,990 V	1022
1023/1024 * AREF = 4,995 V	1023

Pozn.: AREF = 5 V, 10-bit A/D převodník



Obrázek 66 Převodní charakteristika 10-bit A/D převodníku

- Volbu dělícího poměru je třeba provést s rozmyslem v závislosti na rozlišení A/D převodníku a počtu tlačítek.
- Počet tlačítek je omezen:
  - rozlišením A/D převodníku
  - kolísáním napájecího napětí (lze potlačit tím, že se pro referenční napětí A/D převodníku použije napájecí napětí odporového děliče resp. napájecí napětí MCU – referenční napětí pak „pluje“ s tímto napájecím napětím a kód z A/D převodníku je stabilní, nezávislý na napájecím napětí odporového děliče)
  - tolerancí rezistorů
  - požadavkem na odolnost vůči šumu
- U 10-bit A/D převodníku by bylo teoreticky možné připojit 1023 tlačítek. Všechny bity převodníku by tvořily dvojkové číslo stisknutého tlačítka. Krok napětí pro dvě sousední tlačítka by při napájecím napětí (a zároveň referenčním napětím A/D převodníku) 5 V byl $5/2^{10}$ = 5/1024 = cca 4,88 mV. Rezistory by se musely přesně vybírat a napájecí napětí by muselo být velmi dobře stabilizováno. Prakticky nerealizovatelné.

- Příklad: Mějme připojit 8 tlačítek. Napájecí napětí je 5 V.
    - Rozsah 0 až 5 V rozdělíme na 9 částí (8 tlačítek + žádné nestisknuto).
    - Krok napětí = 5 V / 9 = 0,556 V
    - Napěťový krok určuje odolnost vůči šumu a kolísání napájecího napětí. Čím větší tím lepší.
    - Napětí a číselné hodnoty odpovídající jednotlivým tlačítkům:

Vcc = AREF =	5	[V]		
rozlišení =	1024	[-]		
počet tlač. =	8	[-]		
krok =	0,556	[V]	113	[adc]
krok/2 =	0,278	[V]	56	[adc]

Tlačítko	**Napětí na výstupu děliče [V]**	**Číslo (ADC)**	**Rozsah napětí [V]**	**Číslo od (ADC)**	**Číslo do (ADC)**
žádné	0,000	0	0,000 až 0,278	0	55
TL1	0,556	113	0,556 ± 0,278	56	169
TL2	1,111	227	1,111 ± 0,278	170	283
TL3	1,667	341	1,667 ± 0,278	284	397
TL4	2,222	455	2,222 ± 0,278	398	511
TL5	2,778	568	2,778 ± 0,278	511	624
TL6	3,333	682	3,333 ± 0,278	625	738
TL7	3,889	796	3,889 ± 0,278	739	852
TL8	4,444	910	4,444 ± 0,278	853	966

    - Pro každé tlačítko musí být zaveden rozsah napětí (tomu odpovídající rozsah čísla z A/D převodníku), který odpovídá stisku tlačítka, protože napětí kolísá vlivem šumu a kolísání napájecího napětí.
    - Každému tlačítku odpovídá určitý rozsah napětí, šířka tohoto rozsahu je rovna napěťovému kroku (rozdílu dvou sousedních napětí děliče).
    - Pokud v programu vyčteme z A/D převodníku hodnotu např. 700, budeme vědět, že je stisknuté tlačítko TL6, protože hodnota 700 spadá do rozsahu 625 až 738.
    - Výpočet hodnot rezistorů v děliči
        - Pro napětí na výstupu děliče platí:

$$AI = V_{cc} \cdot \frac{R}{R + R_i} [V] \quad (3)$$

Kde: AI – napětí na výstupu děliče

V_{cc} – napájecí napětí děliče

R – dolní rezistor v děliči

R_i – horní rezistory jednotlivých tlačítek

Vstupní odpor A/D převodníku je veliký, můžeme ho zanedbat.

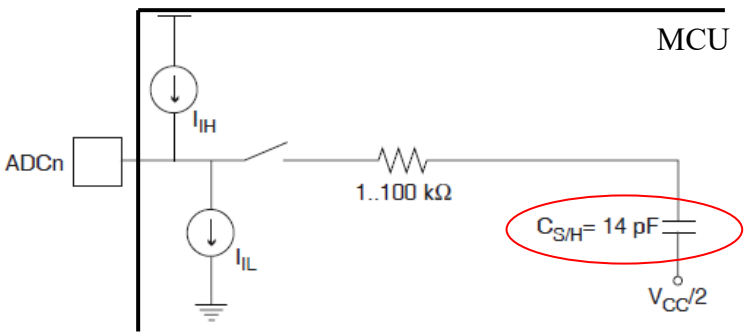
- Odpor R volíme a dopočítáme odpory R_i:

$$R_i = V_{cc} \cdot \frac{R}{AI} - R = R \cdot \left(\frac{V_{cc}}{AI} - 1 \right) \tag{4}$$

Vcc =	5	[V]
R =	1000	[Ω]

Tlačítko	**Napětí na výstupu děliče [V]**	**R_i [Ω]**
žádné	0,000	-
TL1	0,556	8000
TL2	1,111	3500
TL3	1,667	2000
TL4	2,222	1250
TL5	2,778	800
TL6	3,333	500
TL7	3,889	286
TL8	4,444	125

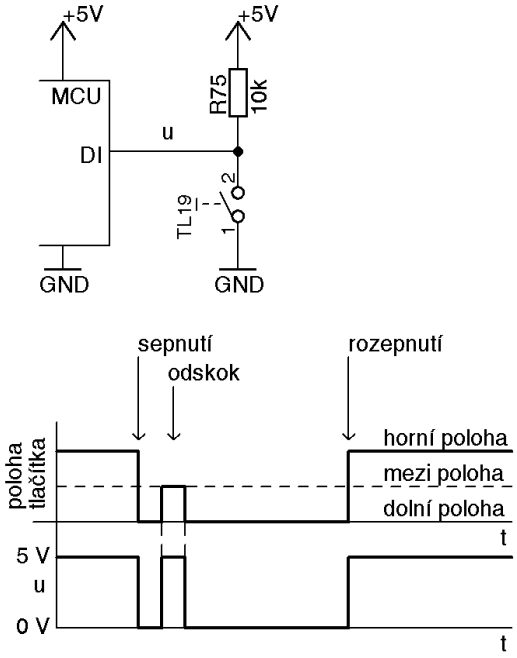
- Odpory R_i následně vybereme z řady (např. E12).
- Odpor R volíme tak, aby nebyla překročena maximální doporučená velikost výstupního odporu děliče (pro AVR činí 10 kΩ), jinak se nestihne během jednoho odběru vzorku nabít SAMPLE and HOLD kondenzátor na vstupu A/D převodníku a převod bude nepřesný. Pokud to nemůže být dodrženo, uděláme několik převodů (např. 3) a použijeme číslo z posledního (to už bude kondenzátor nabit) nebo snížíme taktovací kmitočet A/D převodníku (nevýhodné).



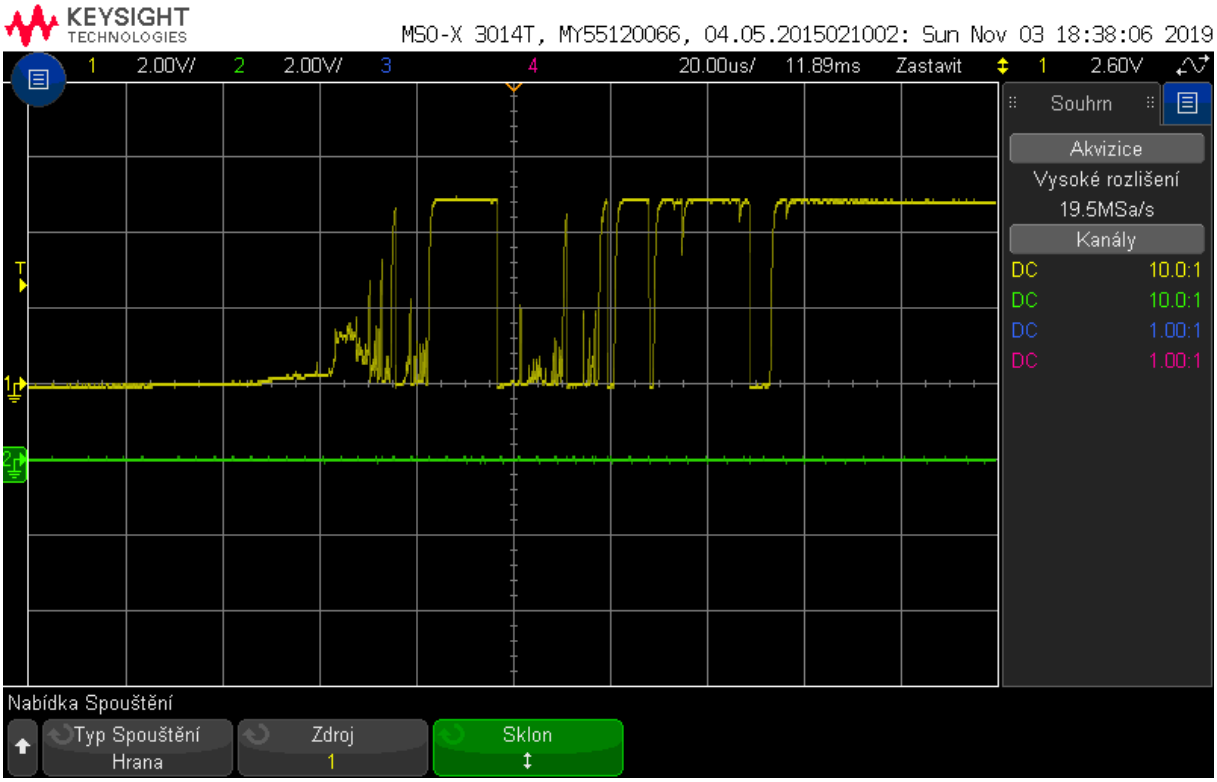
Obrázek 67 Vstupní obvod analogového vstupu

8.3 Ošetření zákmitů tlačítek

- Kontakty běžných mechanických tlačítek, přepínačů, rotačních enkodérů apod. odskakují při změně polohy.



Obrázek 68 Princip zakmitávání tlačítka



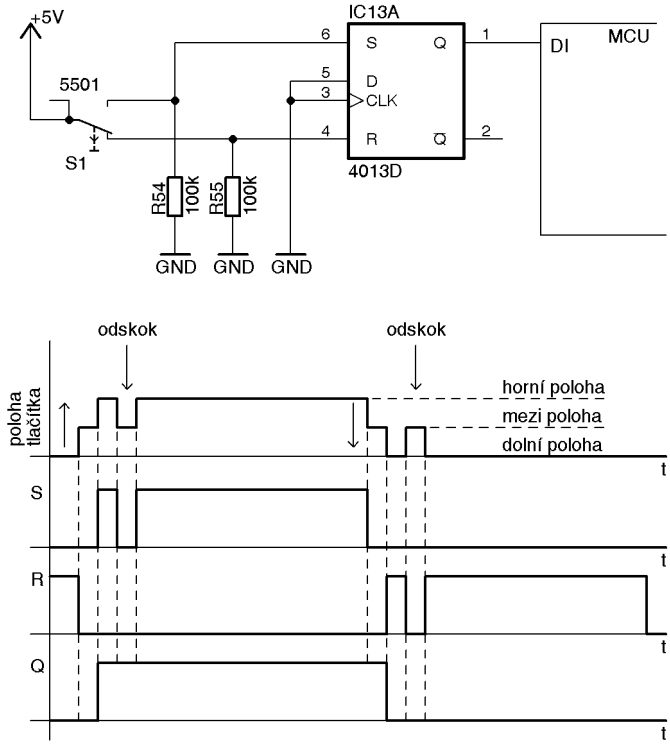
Obrázek 69 Zákmity na tlačítku (CH1 žlutý průběh)

- **Doba zakmitávání kontaktů činí řádově desítky až stovky μs nebo až milisekundy. Dle (ne)kvality spínače.**
- Bez ošetření se během čtení v MCU jeví jeden stisk tlačítka jako několikanásobný stisk (nebo obecně změna stavu pokud se nebudeme bavit jen o tlačítkách).
- **Ošetření zákmitů**:
    - **Hardwarové** – roste složitost návrhu zapojení a potřebná plocha DPS, roste počet součástek, roste cena
    - **Softwarové** – není třeba dodatečný HW, zjednodušení návrhu DPS, úspora nákladů

8.3.1 Hardwarové ošetření zákmitů

- Cílem je překlenout dobu zakmitávání po změně stavu tlačítka.
- Různá řešení
  - Použít bezzákmitová tlačítka (např. membránové klávesnice)
  - RC článek
  - R-S KO
  - Schmittův KO
  - MKO

8.3.1.1 R-S klopný obvod



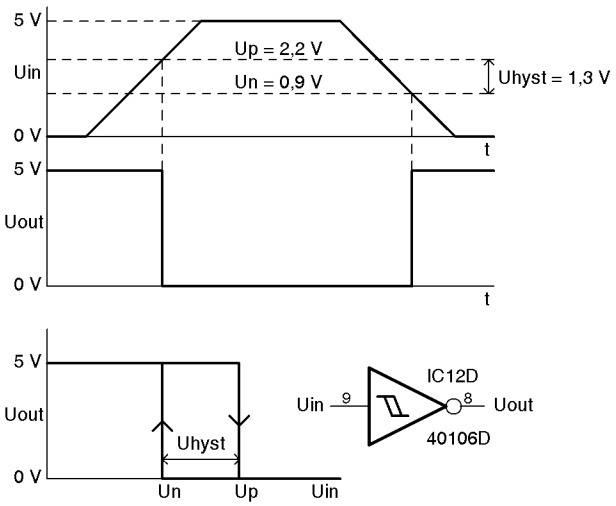
R	S	Q
1	0	0
0	1	1
0	0	Q
1	1	-

Obrázek 70 Ošetření zákmitů tlačítka pomocí R-S KO

- Při použití RS KO je třeba použít tlačítko s přepínacími kontakty.
- Pozitivní logika – tlačítko stisknuto → na vstupu MCU je úroveň H.
- Pokud se kontakt tlačítka nachází v mezipoloze, odpory udržují na vstupech R a S logickou 0 a výstup Q se nemění, je na něm udržován poslední stav (viz pravdivostní tabulka R-S KO). Odskok kontaktu tedy nepůsobí rušivě.
- Jakmile se jeden ze vstupů R nebo S propojí přes kontakt tlačítka s 5 V (logická 1), dojde ke změně výstupu Q do logické 0 (kontakt v horní poloze – stisk tlačítka) nebo do logické 1 (kontakt v dolní poloze – uvolnění tlačítka).

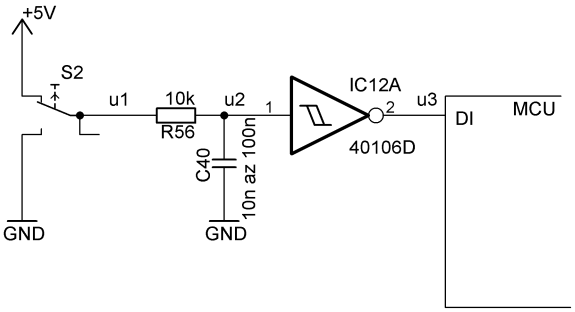
8.3.1.2 Schmittův klopný obvod

- 40106 – invertor se Schmittovým vstupem
- SKO zavádí na svém vstupu hysterezi. Překlápěcí úroveň napětí je různá pro přechod z úrovně L do H a opačně z H do L (Obrázek 71).



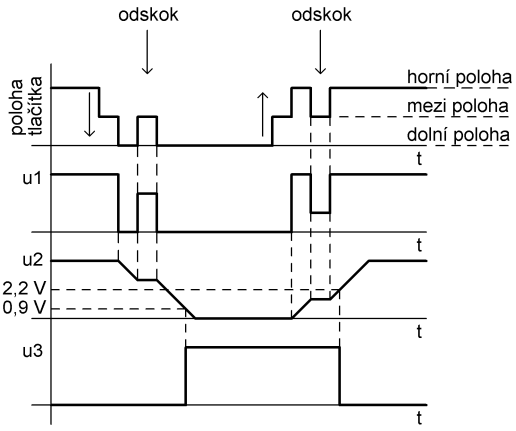
Obrázek 71 Princip hystereze u SKO 40106

- Použití SKO pro ošetření zákmitu tlačítka je uvedeno na Obrázek 72.



Stisk tlačítka

Kondenzátor se vybíjí, dojde k odskoku kontaktu, napětí na kondenzátoru setrvává na poslední hodnotě, odskok pomine, kondenzátor se dál vybíjí až napětí na něm dosáhne překlápěcí úrovně Un. Výstup SKO následně překlopí do úrovně H.



Uvolnění tlačítka

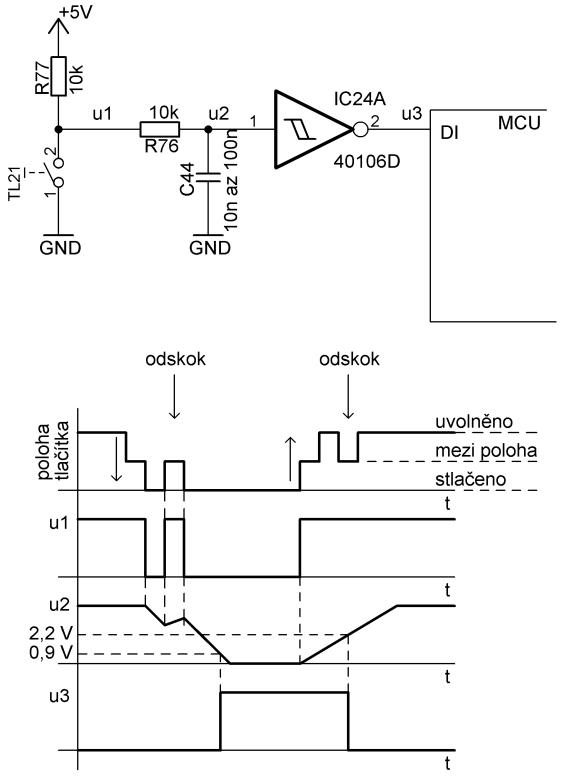
Kondenzátor se nabíjí, dojde k odskoku kontaktu, napětí na kondenzátoru setrvává na poslední hodnotě, odskok pomine, kondenzátor se dál nabíjí až napětí na něm dosáhne překlápěcí úrovně Up. Výstup SKO následně překlopí do úrovně L.

Obrázek 72 Ošetření zákmitů tlačítka s přepínacím kontaktem pomocí SKO (varianta 1)

- Pozitivní logika – tlačítko stisknuto → na vstupu MCU je úroveň H.

- RC článek slouží pro zpomalení růstu/poklesu napětí (jeho filtraci).
- Při přepnutí kontaktu tlačítka se RC článek vybíjí nebo nabíjí. Rychlost nabíjení i vybíjení je stejná a úměrná časové konstantě $\tau = RC$. Časovou konstantu je třeba zvolit experimentálně dle kvality tlačítka.
- Jakmile dosáhne napětí na kondenzátoru překlápěcí úrovně, tak výstup SKO se překlopí. Překlápěcí úroveň napětí je různá pro přechod z úrovně L do H a H do L. SKO má na svém vstupu hysterezi.
- SKO slouží jako tvarovací obvod, který obnovuje hrany signálu z tlačítka (kondenzátoru). Z pomalého přechodu z úrovně L do H a opačně (díky nabíjení/vybíjení kondenzátoru) vytváří rychlý přechod (strmou hranu).
- Stisk tlačítka
    - Kondenzátor se vybíjí, dojde k odskoku kontaktu, napětí na kondenzátoru setrvává na poslední hodnotě, odskok pomine, kondenzátor se dál vybíjí až napětí na něm dosáhne překlápěcí úrovně Un. Výstup SKO následně překlopí do úrovně H.
- Uvolnění tlačítka
    - Kondenzátor se nabíjí, dojde k odskoku kontaktu, napětí na kondenzátoru setrvává na poslední hodnotě, odskok pomine, kondenzátor se dál nabíjí až napětí na něm dosáhne překlápěcí úrovně Up. Výstup SKO následně překlopí do úrovně L.

Použití SKO pro ošetření zákmitu tlačítka se spínacím kontaktem je uvedeno na Obrázek 73.

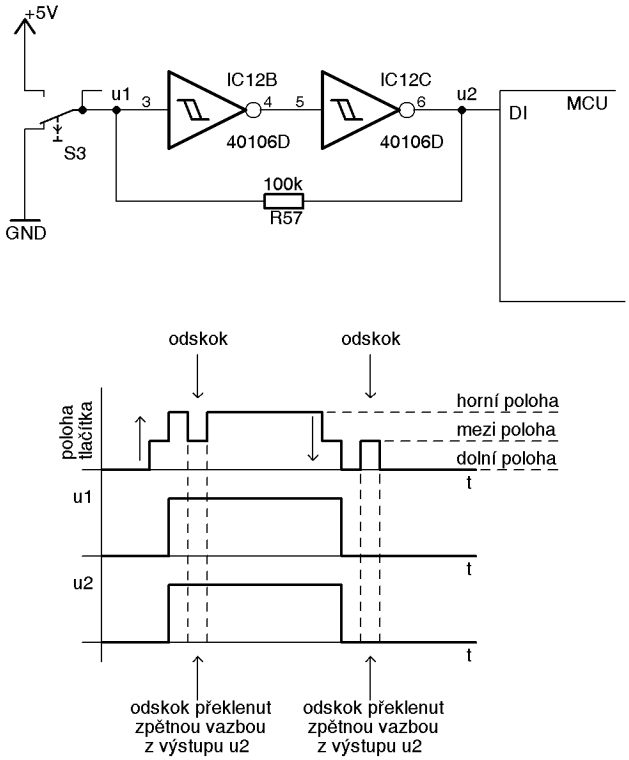


Obrázek 73 Ošetření zákmitů tlačítka se spínacím kontaktem pomocí SKO (varianta 2)

- Logickou úroveň při uvolněném tlačítku definuje pull-up rezistor k +5V.
- Funkce obvodu je podobná jako v případě použití tlačítka s přepínacím kontaktem.
- Při přepnutí kontaktu tlačítka se RC článek vybíjí nebo nabíjí, ale rychlost nabíjení i vybíjení je různá (nabíjecí časová konstanta je 2x delší než vybíjecí). Hodnoty R a C je třeba zvolit experimentálně dle kvality tlačítka (obvykle jednotky ms).

Varianta nevyžadující RC článek

- Můžeme použít 1x SKO (1/6 40106) + 1x obyčejný invertor (1/6 4069) nebo 1x SKO (1/6 40106) + 1x SKO (1/6 40106), což má výhodu v tom, že oba SKO jsou v jednom pouzdru (ve 40106 je 6x SKO).
- Zavedeme zpětnou vazbu z výstupu obvodu na vstup.
    - Pomocí ní překleneme odskok kontaktu (zpětná vazba nám na vstupu udrží napětí).



Obrázek 74 Ošetření zákmitů tlačítka pomocí SKO (varianta 2)

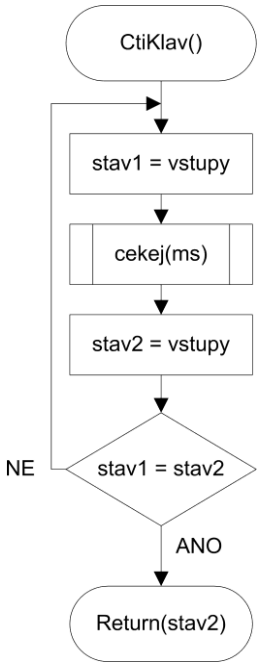
- Stisk tlačítka
    - Kontakt se odlepí od GND a dostane se do mezipolohy, napětí na vstupu u1 je drženo zpětnou vazbou z výstupu u2 přes odpor 100k v úrovni L.
    - Kontakt se dosedne na +5V, na u1 je napětí 5 V, na u2 se také objeví napětí 5 V.
    - Kontakt odskočí do mezipolohy, napětí na vstupu u1 je drženo zpětnou vazbou z výstupu u2 přes odpor 100k v úrovni H.
    - Kontakt se vrátí do polohy +5V, napětí na u1 a u2 setrvává na 5 V.
    - Odskok byl překlenut zpětnou vazbou z výstupu u2 na vstup u1.
- Uvolnění tlačítka
    - Obdobně jako u stisku, akorát v opačné logice.

8.3.2 Softwarové ošetření zákmitů

- Řešení nevyžadující přídavné součástky.
- V nenáročných případech postačující řešení.

8.3.2.1 Varianta 1

- Spočívá ve dvojím čtení stavu klávesy s prodlevou a následné porovnání stavů.
- Pokud jsou stavy shodné, program pokračuje dál, jinak se čtení opakuje.
- Prodlevu mezi čteními je třeba zvolit takovou, aby překlenula obvyklé trvání zákmitů kontaktu.
- Na Obrázek 75 je uveden vývojový diagram podprogramu, který toto řeší.

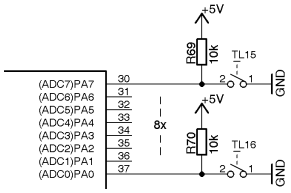


Obrázek 75 Vývojový diagram SW ošetření zákmitů (varianta 1)

Kód implementující variantu 1:

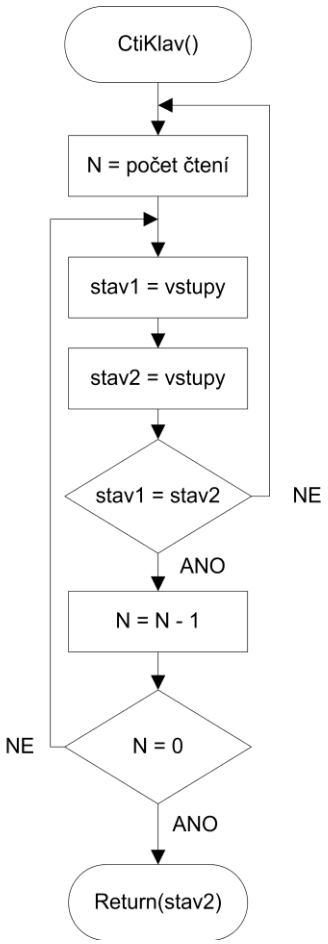
```
/* Cte stav tlacitek s osetrenim zakmitu.
        parametry:      zadne
        vraci:          byte, ve kterem je stav 8-mi tlacitek, co bit to jedno tlacitko
                                bit v H odpovida stisknutemu tlacitku
        Interne vola low-level funkci cti_tlacitka(), ktera slouzi pro cteni stavu tlacitek.
*/
uint8_t cti_stav_tlacitek(void)
{
        uint8_t stav;


        do
        {
                stav = cti_tlacitka();      // cte stav tlacitek, co bit to jedno tlacitko
                _delay_ms(10);              // ceka definovanou dobu [ms]
        } while (cti_tlacitka() != stav);   // cte a porovnava stav tlacitek s predchozim ctenim
        return stav;                        // stav je OK, bez zakmitu, koncime
}
```



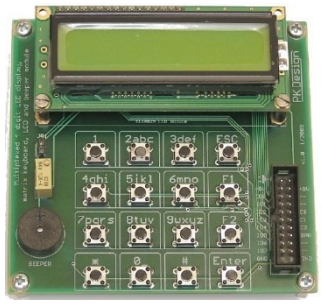
8.3.2.2 Varianta 2

- Spočívá ve snímání stavu N-krát po sobě.
- Pokud se všechny vzorky shodují (tj. vstup se během opakovaného čtení nezměnil), je stav považován za správný.
- Pokud se stav během opakovaného čtení změní, celý cyklus se opakuje.
- Počet opakování N je závislý na rychlosti vykonávání programu a musí se stanovit tak, aby se překlenula doba zakmitávání kontaktu.
- Na Obrázek 76 je uveden vývojový diagram podprogramu, který toto řeší.

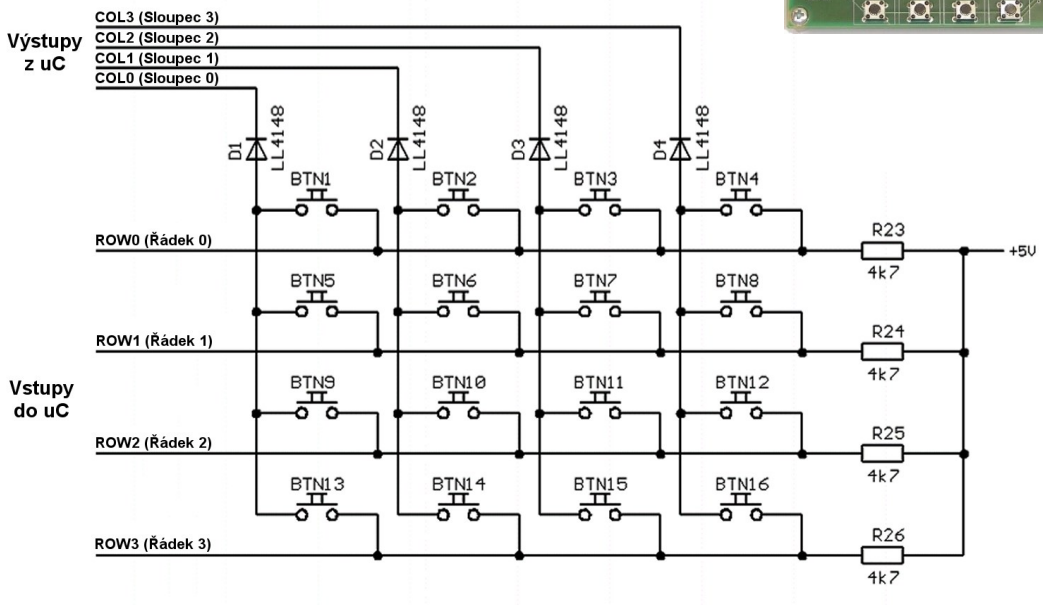


Obrázek 76 Vývojový diagram SW ošetření zákmitů (varianta 2)

8.4 Maticová klávesnice



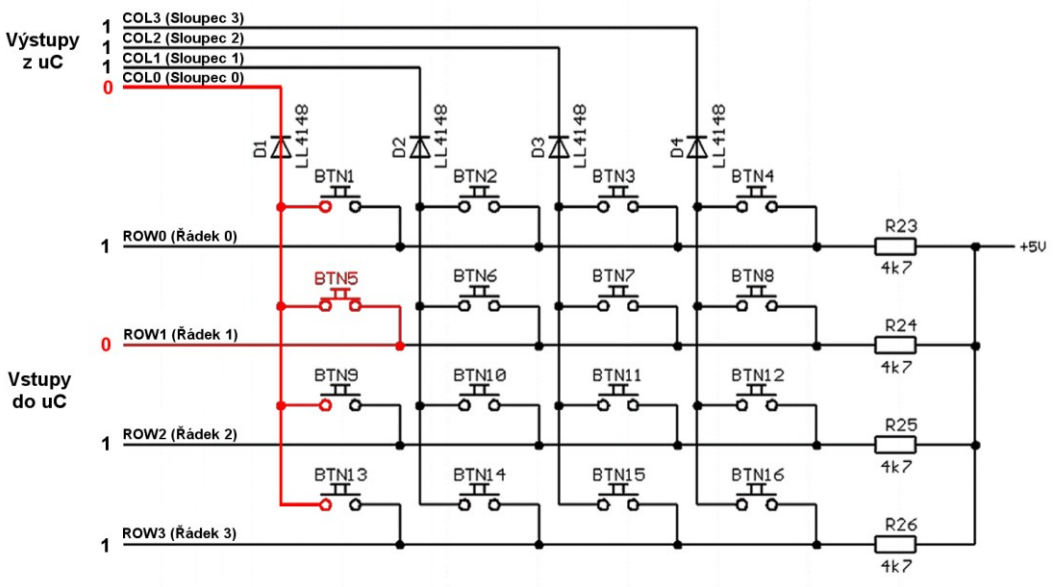
- Klávesy jsou uspořádány do matice NxN (například 4x4).
- Čtení stavu kláves je nutno provádět multiplexovaně.
- Klasické zapojení klávesnice je na následujícím obrázku:



Obrázek 77 El. zapojení maticové klávesnice 4x4

8.4.1 Čtení stavu tlačítek

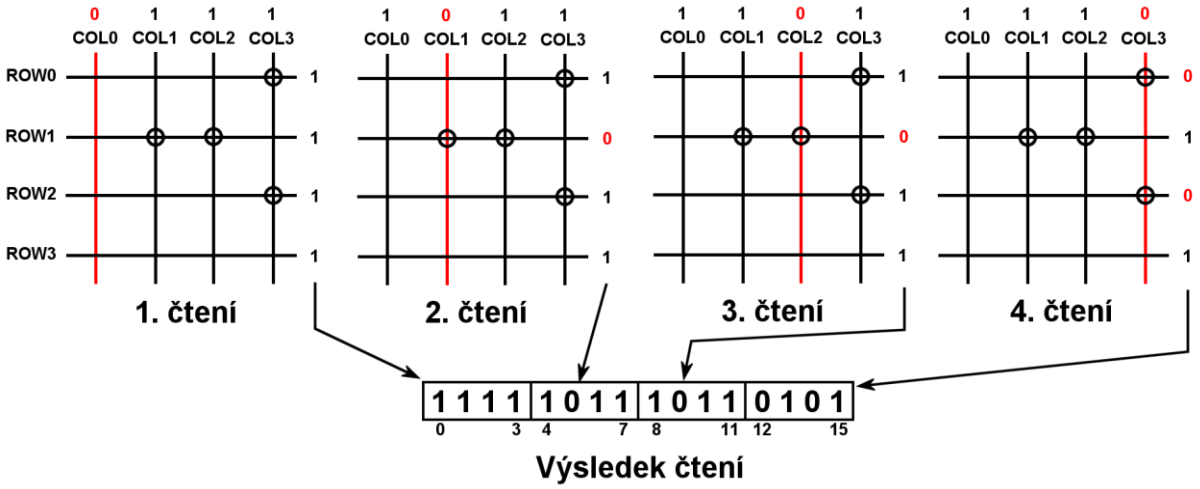
- MCU postupně aktivuje jednotlivé sloupce (MCU postupně uzemňuje výstupy COLx) a následně čte stav všech řádek (ROW0 – ROW3). Je-li tlačítko stisknuto, bude na příslušném řádku logická 0.



Obrázek 78 Čtení stavů tlačítek maticové klávesnice 4x4

- V případě maticové klávesnice 4x4 je nejlepším řešením připojit všechny její vstupy a výstupy na jeden 8-bit port mikrokontroléru (4 výstupy + 4 vstupy = 8 pinů).

- V případě matice 4x4 dostaneme čtyři 4-bitové výsledky, které je nejlepší uložit do jedné 16-bitové proměnné. Princip je zřejmý z dalšího obrázku.



Obrázek 79 Sestavení výsledku čtení stavů tlačítek maticové klávesníice 4x4

- Jednotlivé bity ve výsledné proměnné označují stisk příslušného tlačítka v daném sloupci (0 – tlačítko stisknuto, 1 – tlačítko nestisknuto).
- Tento způsob čtení dovoluje rozpoznat i stisk několika tlačítek najednou (ve stejném řádku i sloupci).
- Každá klávesa (když je stisknuta) má svůj jedinečný kód:

	COL0	**COL1**	**COL2**	**COL3**
ROW0	0xFFFE	0xFFEF	0xFEFF	0xEFFF
ROW1	0xFFFD	0xFFDF	0xFDFF	0xDFFF
ROW2	0xFFFB	0xFFBF	0xFBFF	0xBFFF
ROW3	0xFFF7	0xFF7F	0xF7FF	0x7FFF

- Předchozí tabulka odpovídá negativní logice (klávesa stisknuta = log. 0). **Přirozenější je pozitivní logika (klávesa stisknuta = log. 1), výsledek čtení je tedy nutno invertovat.** Kódy kláves v tomto případě budou následující:

	COL0	**COL1**	**COL2**	**COL3**
ROW0	0x0001	0x0010	0x0100	0x1000
ROW1	0x0002	0x0020	0x0200	0x2000
ROW2	0x0004	0x0040	0x0400	0x4000
ROW3	0x0008	0x0080	0x0800	0x8000

Z této tabulky je vidět, že kódy kláves ve sloupcích COL1 – COL3 jsou vždy násobkem 16-ti vůči předchozímu sloupci (bitový posun o 4 bity doleva).

8.4.2 Softwarová implementace

- Pro obsluhu maticové klávesnice je vhodné napsat funkci, která provede výše uvedený algoritmus a vrátí výsledek čtení.

Implementace funkce pro čtení stavu kláves v jazyce C

Předpokládá se, že výstupy COL0 - COL3 jsou připojeny na spodní 4 bity portu a vstupy ROW0 - ROW3 jsou připojeny na horní 4 bity portu.

Funkce vrací 16-bit celé číslo, ve kterém každému bitu odpovídá jedno tlačítko maticové klávesnice. Bit má hodnotu 1, pokud je tlačítko stisknuto, jinak má bit hodnotu 0.

```
#include <avr/io.h>

// definice pripojeni klvesnice k MCU
#define MATIC_KL_PORT         PORTA
#define MATIC_KL_DDR          DDRA
#define MATIC_KL_PIN          PINA

unsigned int cti_matic_kl(void)
{
        unsigned int klavesa = 0;
        unsigned char i, s, r;
        unsigned char zaloha_DDR;

        // zaloha registru smeru
        zaloha_DDR = MATIC_KL_DDR;

        // nastaveni smeru pinu, radky vstupni, sloupce vystupni
        MATIC_KL_DDR = 0x0F;
        s = 1;                  // zacinam 0.sloupcem
        for (i=0; i<4; i++)     // aktivuji postupne sloupce (uzemnuji je)
        {
                // aktivace daneho sloupce do 0, (maskovani 0x0F neni nutne)
                MATIC_KL_PORT = (~s) & 0x0F;
                // pauza kvuli prestaveni hodnoty na pinech, NUTNE !!
                asm volatile("nop\n\t"
                "nop\n\t"
                "nop\n\t"
                "nop\n\t"
                "nop\n\t"
                "nop\n\t"
                "nop\n\t"
                "nop\n\t"
                ::);
                /*
                Precteni radku v danem sloupci, prevod do pozitivni logiky a posun
                o 4 bity kvuli zarovnani vysledku doprava.
                */
                r = MATIC_KL_PIN;
                r = ~r;
                r = r >> 4;
                /*
                napr. MATIC_KL_PIN = 0b11101111, pak ~ MATIC_KL_PIN = 0b00010000 a
                (~MATIC_KL_PIN) >> 4 = 0b00000001
                */
                /*
                Kod klavesy je 4 bitovy a vsechny 4 sloupce jsou ulozeny vedle
                sebe do 16-bitove promenne. Jednicka na prislusne bitove pozici
                znaci zmacknutou klavesu.
                */
                /*
                Akt. stav klaves je rotovan o 4 bity doleva a bitove secten
                s minulym vysledkem
                */
                klavesa |= (unsigned int)(r) << (i<<2);
                s = s << 1;     // dalsi sloupec
        }
```

- Tuto funkci pak zavoláme, potřebujeme-li zjistit stav tlačítek. Ten se nejčastěji zjišťuje periodicky – funkce je volána buď v nekonečné smyčce nebo např. v přerušení od časovače, který generuje požadovaný časový interval čtení stavu kláves.

8.4.2.1 Čtení stavu tlačítek v nekonečné smyčce

```
// definice kodu klaves
#define BTN1 0x0001
#define BTN2 0x0002

void main(void)
{
        unsigned int klavesa = 0;
        /*

        nejaky kod

        */
        while (1)       // nekonecna smycka
        {
                /*
                nejaky kod
                */
                klavesa = cti_matic_kl();

                // reakce na klavesu
                if (klavesa == BTN1)
                {
                        // reakce na BTN1
                }
                else if (klavesa == BTN2)
                {
                        // reakce na BTN2
                }
                /*
                nejaky kod
                */
        };
}
```

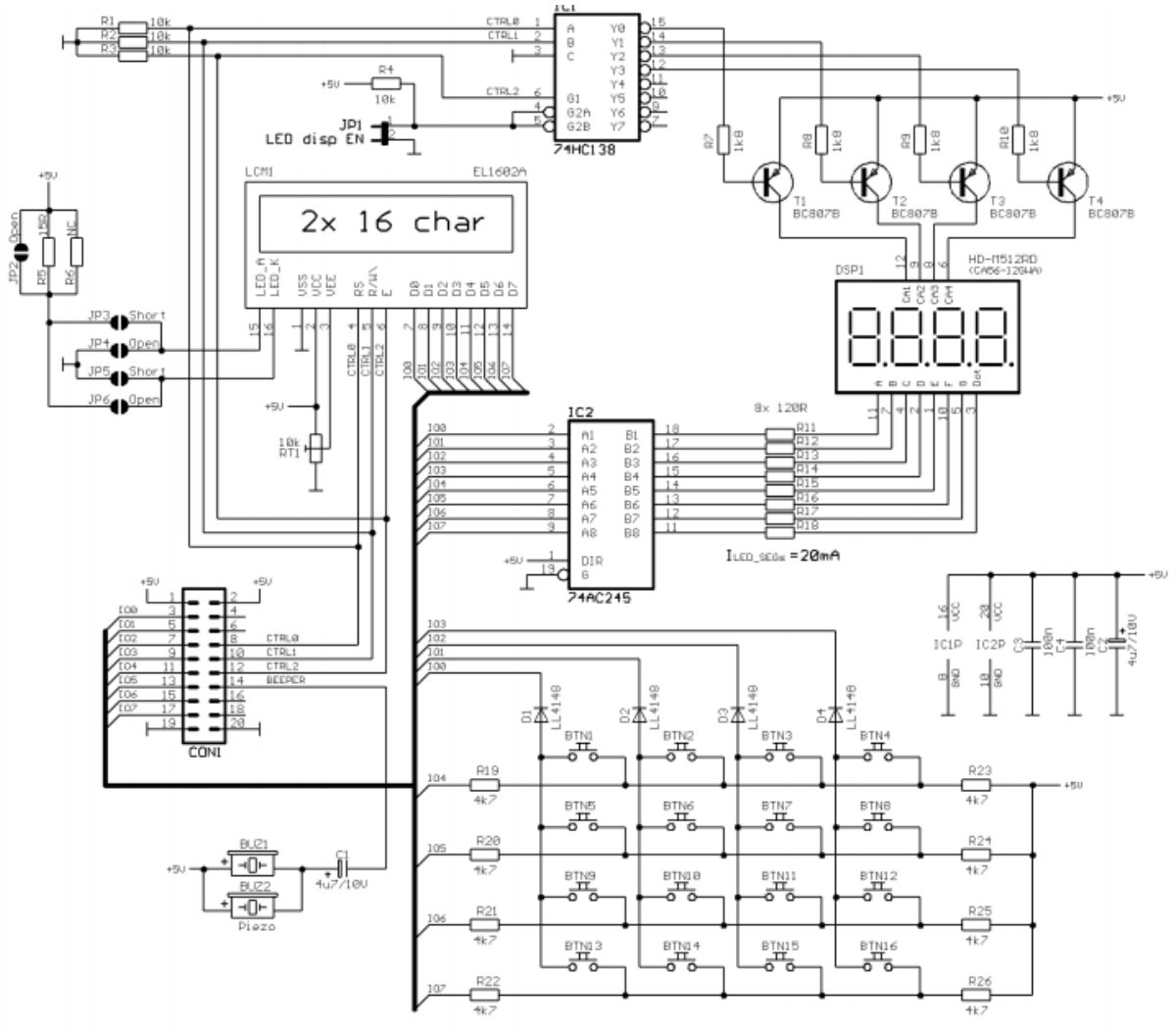
- Chceme-li reagovat na kombinaci kláves (2 a více stisknutých kláves najednou), použijeme bitový součet kódů kláves.

```
if (klavesa == (BTN1 | BTN2))
{
        // reakce na soucasny stisk BTN1 a BTN2
}
```

- Je samozřejmě možné použít i příkaz `switch`

```
switch (klavesa)
{
        case BTN1:      // reakce na BTN1
                break;
        case BTN2:      // reakce na BTN2
                break;
        case (BTN1|BTN2): // reakce na soucasny stisk BTN1+BTN2
                break;
}
```

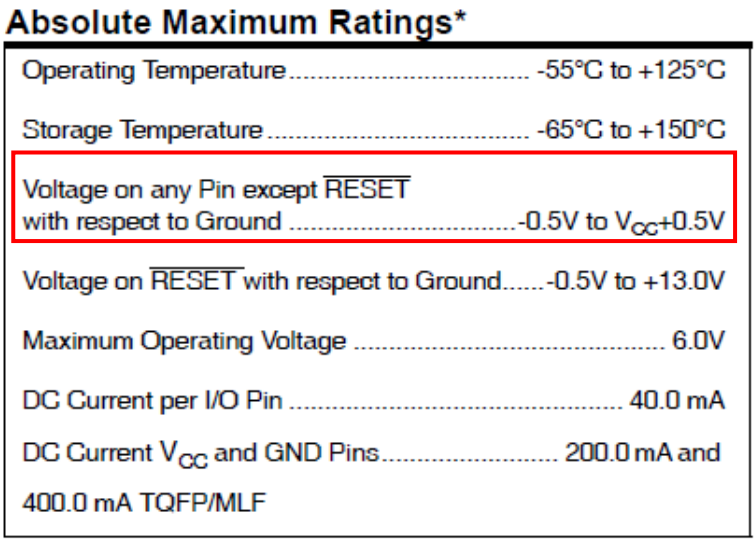
8.4.2.2 Zapojení přípravku Maticová klávesnice + LCD/7-segm.



Obrázek 80 Zapojení přípravku Maticová klávesnice + LCD/7-segm. na kitech AVR v laboratoři

8.5 Přepěťová ochrana vstupů

- Max. vstupní napětí na pinech obvykle -0,5 V až Vcc + 0,5 V jinak hrozí poškození obvodu (např. MCU).



Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA and 400.0 mA TQFP/MLF

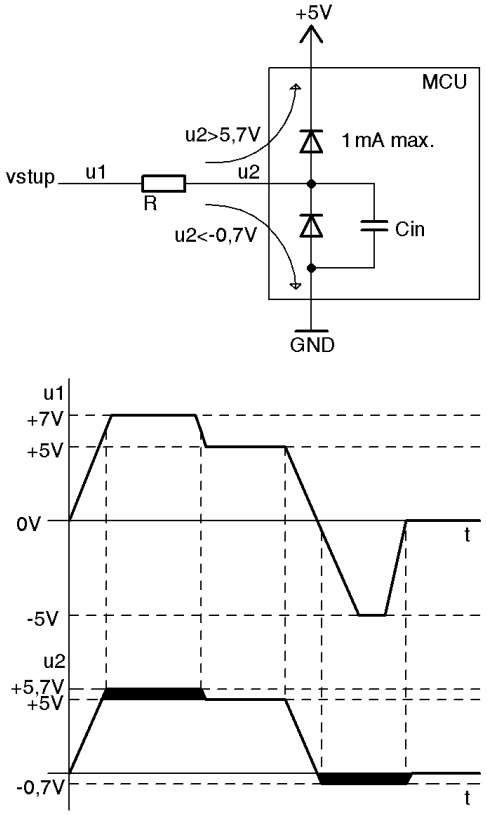
Obrázek 81 AVR Electrical parameters - Absolute Maximum Ratings

Proč a kdy chránit ?

- Možný dotyk rukou (odhalené konektory apod.) – elektrostatický výboj (ESD).
- Výskyt přepětí na komunikačních linkách a dlouhých vedeních.
- Porucha připojeného zařízení.

8.5.1 Využití interních ochranných diod

- Řada obvodů má své vstupy chráněny interními ochrannými diodami vůči Vcc a GND.
- Diody nedovolí překročit max. napětí na pinu, otevřou se a proud svedou do Vcc nebo do GND. Ořezávají přepětí.



Obrázek 82 Princip přepěťové ochrany s využitím interních ochranných diod na pinu v MCU

- Horní dioda otvírá pokud: u2 > 5 V + 0,5 V
- Dolní dioda otvírá pokud: u2 < -0,5 V
- Omezená proudová zatížitelnost interních ochranných diod, **proud max. 1 mA u MCU AVR**.
    - Zjištěno dotazem na support Atmel.
    - V datasheetu k AVR není zmínka o interních ochranných diodách.
- Je nutný předřadný odpor R pro omezení proudu ochrannou diodou.
    - Velikost R dle předpokládaného max. napětí na pinu a max. povoleného proudu ochrannou diodou.

$$R_{+} = \frac{U_{\max +} - (V_{CC} + 0,5)}{I_{\mathrm{D}\max}} \tag{5}$$

$$R_{-} = \frac{U_{\max -} + 0,5}{I_{\mathrm{D}\max}} \tag{6}$$

$$R = \max(R_+, R_-) \tag{7}$$

Kde: R_+ - předřadný odpor pro kladné přepětí [Ω]
R_- - předřadný odpor pro záporné přepětí [Ω]
R – výsledná hodnota předřadného odporu [Ω]
U_{max+} – maximální kladné přepětí [V]
U_{max-} – maximální záporné přepětí [V], zadává se záporné
0,5 – úbytek napětí na ochranné diodě [V]
I_{Dmax} – max. proud ochrannou diodou [A]

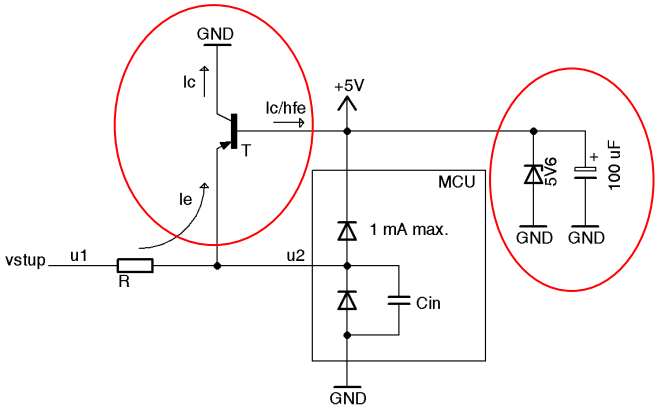
Např.: U_{max+} = 30 V, U_{max-} = -30 V, I_{Dmax} = 1 mA, Vcc = 5 V

$$R_+ = \frac{30-(5+0,5)}{0,001} = 25,500\,\text{k}\Omega \rightarrow 27\,\text{k}\Omega$$

$$R_- = \frac{-30+0,5}{0,001} = 29,5\,\text{k}\Omega \rightarrow 33\,\text{k}\Omega$$

$$R = \max(27\,\text{k}\Omega, 33\,\text{k}\Omega) = 33\,\text{k}\,\Omega$$

- Problémy:
  - **Možný růst Vcc** při otevření horní ochranné diody pokud je z Vcc jen slabý odběr a zdroj přepětí je silný. V této situaci zdroj přepětí dodává proud do napájecí větve Vcc a napětí Vcc se může zvýšit nad limit → hrozí poškození všech obvodů připojených k Vcc.
  - Řešení (Obrázek 83):
    - Na větev Vcc doplnit velký kondenzátor (omezí rychlost růstu Vcc).
    - Napájecí větev Vcc chránit Zenerovou diodou.
    - Na vstupu použít aktivní ochranu - PNP tranzistor, který otevře při přepětí a svede proud do GND. Proud injektovaný do Vcc se zmenší o faktor zesílení h_{FE} tranzistoru, navíc předřadný odpor R může být menší, což je výhodné jak uvidíme dále.

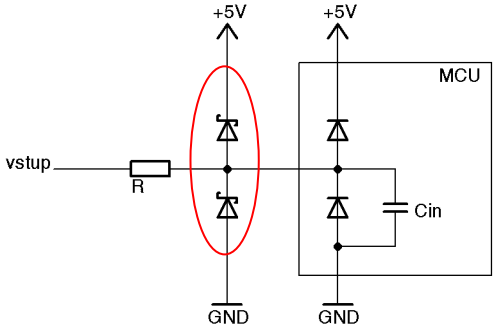


Obrázek 83 Aktivní přepěťová ochrana s PNP tranzistorem, kombinovaná s ochranou Vcc

- **Velký R snižuje přenosovou rychlost**
  - Parazitní kapacita pinu C_{in} tvoří s předřadným odporem R a kapacitou případně připojené komunikační sběrnice C_{bus} dolní propust.
  - **Kazí hrany obdélníkového signálu**.
  - **Omezuje max. použitelnou frekvenci digitálního signálu na pinu**.
  - **Čím větší R tím nižší max. přenosová rychlost**.
  - $\tau = R*(C_{in} + C_{bus})$
  - $C_{in} \approx$ jednotky pF, $C_{bus} \approx$ desítky až stovky pF
  - **Např.:**
    R = 30 kΩ, $C_{in} + C_{bus}$ = 100 pF → τ = 3 µs → $f_{SW} = 1/(3*\tau)$ = 111 kHz

8.5.2 Využití externích ochranných diod vně MCU

- Za účelem proudového posílení (desítky a stovky mA).
- Hodnota předřadného odporu R pak vychází menší → menší parazitní časová konstanta → vyšší přenosová rychlost.
- **Využití Schottkyho diod**
  - Malý úbytek v dopředném směru (typ. 0,3 V) → interní ochranné diody v MCU se při přepětí ani neotevřou, neteče jimi žádný proud.



Obrázek 84 Přepěťová ochrana s externími ochrannými diodami

  - **Pozor, Schottkyho diody mají cca o řád vyšší závěrný proud než křemíkové diody** (zejména při vyšších teplotách) – to může vadit při ochraně analogových vstupů (vlivem proudu dochází k úbytku napětí na vnitřním odporu zdroje signálu a tím ke zkreslení signálu).

1N4148 (Si dioda):

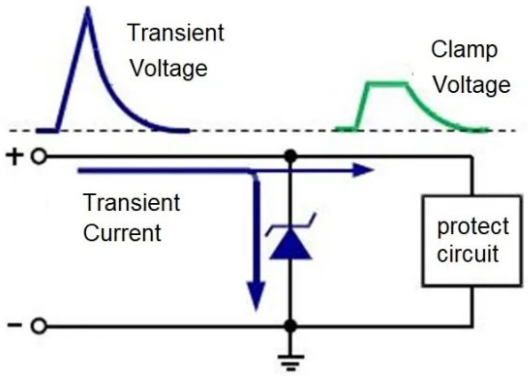
Reverse current	V_R = 20 V		I_R			25	nA

BAT46 (Schottkyho dioda):

Parameter	Test condition	Symbol	Min	Typ.	Max	Unit
Reverse breakdown voltage	I_R = 100 µA (pulsed)	$V_{(BR)R}$	100			V
Leakage current	Pulse test t_p < 300 µs, δ < 2 % V_R = 1.5 V	I_R			0.5	µA
	Pulse test t_p < 300 µs, δ < 2 % V_R = 1.5 V, T_j = 60 °C	I_R			5	µA
	Pulse test t_p < 300 µs, δ < 2 % V_R = 10 V	I_R			0.8	µA
	Pulse test t_p < 300 µs, δ < 2 % V_R = 10 V, T_j = 60 °C	I_R			7.5	µA
	Pulse test t_p < 300 µs, δ < 2 % V_R = 50 V	I_R			2	µA
	Pulse test t_p < 300 µs, δ < 2 % V_R = 50 V, T_j = 60 °C	I_R			15	µA
	Pulse test t_p < 300 µs, δ < 2 % V_R = 75 V	I_R			5	µA

8.5.3 Použití TVS

- TVS = Transient Voltage Suppressor
- Skupina součástek určených pro omezení škodlivého přepětí:
  - Zenerova dioda
  - Transil
  - Varistor



Obrázek 85 Princip omezení přepětí pomocí prvků TVS

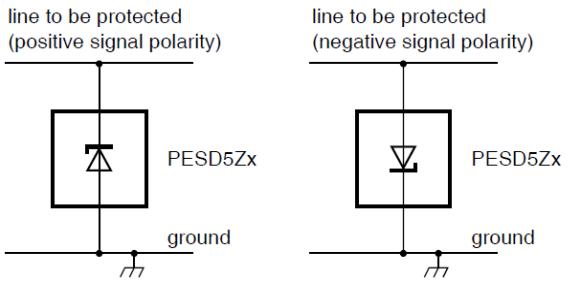


Zenerova dioda (THT) | Transil (SMD) | Transil (THT) | Varistor

Obrázek 86 Jednotlivé typy TVS prvků v provedení pro připájení na DPS

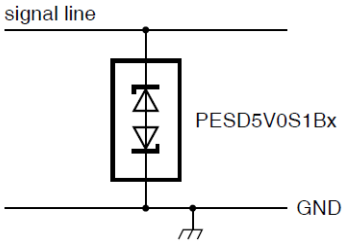
- Důležité parametry TVS:
  - Velikost závěrného napětí – určuje úroveň napětí, od které bude napětí považováno za přepětí, které bude součástkou omezováno.
  - Max. velikost proudu - od jednotek A po desítky kA.
  - Určené pro imuplsní režim (velké kráktodobé proudy).
  - Rychlost odezvy na přepětí – od jednotek ps po jednotky μs.
  - Parazitní kapacita – jednotky pF až nF. Ovlivňuje použití TVS na komunikačních sběrnicích.
  - Velikost svodového proudu – desítky pA až μA. Ovlivňuje použití TVS na analogových linkách.
  - Životnost – počet cyklů přepětí, které TVS vydrží (varistor stárne, transil ne).

- **Velký výběr TVS od význačných výrobců** (např. NXP Semiconductors, ST Microelectronics)
    - Dle kategorie použití:
        - Obecné použití – ochrana analogových I/O, digitálních I/O
        - Ochrana sběrnic – SPI, I2C, RS-232, RS-485, USB, ETH, CAN, LIN,...
        - Ochrana rozhraní pro přenos obrazu – HDMI, DisplayPort,…
        - Automotive
    - Dle polarity pracovního napětí:
        - **Unidirectional** – pouze jedna polarita napětí vůči GND (0 V až Vcc nebo -Vcc až 0 V)



unidirectional protection of one line

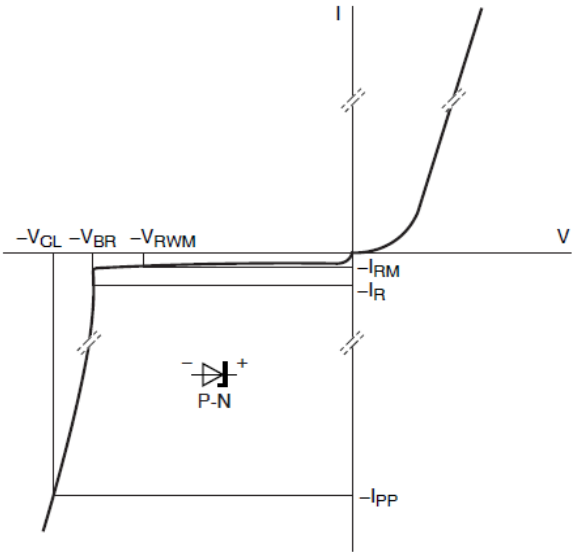
        - **Bidirectional** – obě polarity napětí vůči GND (-Vcc až +Vcc)



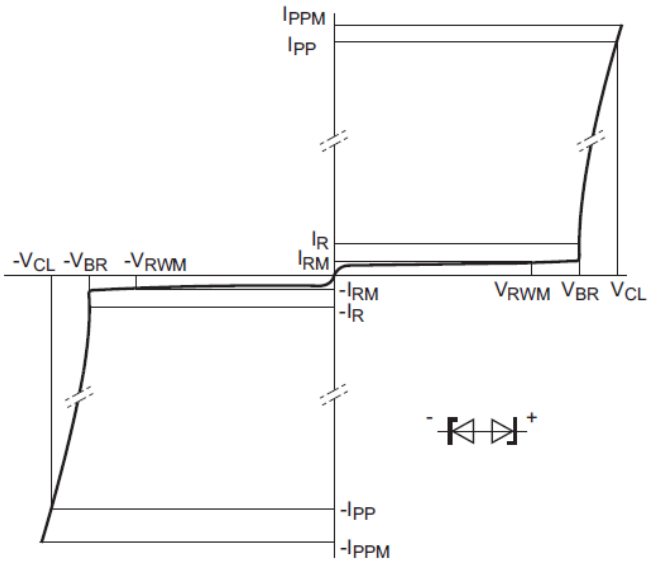
    - Jednoduché použití – obvykle bez dalších externích součástek.
    - Několik TVS v jednom pouzdru, pinout přizpůsoben aplikaci.
    - Malé rozměry.
    - **Aplikační příručky** (**Application Notes**) dostupné zdarma na webech výrobců – návod na výběr a použití součástky.

8.5.3.1 Základní parametry TVS

- V-A charakteristika



Obrázek 87 V-A char. unipolárního transilu



Obrázek 88 V-A char. bipolárního transilu

Normální pracovní oblast bez omezování přepětí – od 0 do V_{RWM}. Signál není výrazně ovlivněn funkcí přepěťové ochrany.

Oblast omezování napětí – od V_{BR} do V_{CL}. Signál je omezen přepěťovou ochranou. V této oblasti je požadováno, aby V-A char. byla co nejstrmější, tj. aby se napětí V_{CL} blížilo napětí V_{BR}. Tuto vlastnost lze vyjádřit pomocí ukazatele „clamping factor“ C_F. Tento ukazatel souvisí s dynamickým odporem R_d součástky v oblasti omezování napětí.

$$C_F = \frac{V_{CL}}{V_{BR}} \tag{8}$$

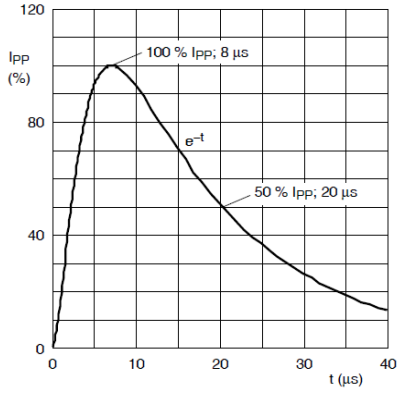
V_{RWM} – napětí, při kterém zaručeně nedojde k vybavení přepěťové ochrany. Do této hodnoty se musí normálně pohybovat signál na lince, která je chráněna přepěťovou ochranou.

V_{BR} – průrazné napětí, přepěťová ochrana začíná vybavovat

V_{CL} – napětí, na které je přepětí omezeno při proudu I_{PP}

I_{PPM} – neopakovatelný špičkový proud (testovací rázový impuls 8/20 µs dle IEC 61000-4-5)

I_{RM} – svodový proud „zavřené“ přepěťové ochrany



Obrázek 89 Standardizovaný rázový impuls dle normy IEC 61000-4-5

- Další parametry uváděné v datasheetu:
  - P_{PPM} - neopakovatelný špičkový výkon $P_{PPM} = I_{PPM} \cdot V_{CL}$
  - **R_d** - diferenciální odpor v oblasti omezování přepětí (definuje sklon V-A char. mezi napětími V_{BR} a V_{CL}). Čím je R_d menší, tím je V-A char. v oblasti omezování napětí strmější a napětí V_{CL} se tolik nevzdaluje od V_{BR}.

$$V_{CL} = V_{BR} + R_d \cdot I_{PP} \tag{9}$$

  - **C_d** – kapacita diody – čím je nižší, tím vyšší frekvence je možné přenášet po vodiči, ke kterému je přepěťová ochrana připojena.

- **Ukázka parametrů uváděných v datasheetu přepěťové ochrany typu PESD1CAN-U**:

Table 5. Limiting values

In accordance with the Absolute Maximum Rating System (IEC 60134).

Symbol	Parameter	Conditions		Min	Max	Unit
P_{PPM}	rated peak pulse power		[1][2]	-	150	W
I_{PPM}	rated peak pulse current		[1][2]	-	3	A
T_j	junction temperature			-	150	°C
T_{amb}	ambient temperature			–55	+150	°C
T_{stg}	storage temperature			–65	+150	°C

[1] Device stressed with ten non-repetitive current pulses (8/20 μs exponential decay waveform according to IEC 61000-4-5 and IEC 61643-321).

[2] Measured from pin 1 or 2 to 3.

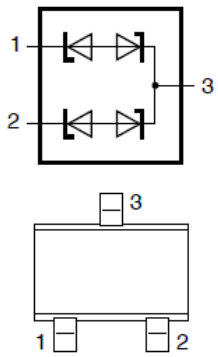


Table 6. ESD maximum ratings

Symbol	Parameter	Conditions		Min	Max	Unit
V_{ESD}	electrostatic discharge voltage	IEC 61000-4-2 (contact discharge)	[1][2]	-	23	kV
		machine model	[2]	-	400	V
		MIL-STD-883 (human body model)		-	10	kV

[1] Device stressed with ten non-repetitive ESD pulses.

[2] Measured from pin 1 or 2 to 3.

Table 8. Characteristics

T_{amb} = 25 °C unless otherwise specified.

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{RWM}	reverse standoff voltage			-	-	24	V
I_{RM}	reverse leakage current	V_{RWM} = 24 V		-	1	50	nA
V_{BR}	breakdown voltage	I_R = 5 mA		25.4	27.8	30.3	V
V_{CL}	clamping voltage	I_{PP} = 1 A	[1][2]	-	-	35	V
		I_{PPM} = 3 A		-	-	50	V
C_d	diode capacitance	f = 1 MHz; V_R = 0 V		-	9.3	12	pF
		f = 1 MHz; V_R = 2.5 V		-	7.2	-	pF
$\Delta C_d/C_d$	capacitance matching	f = 1 MHz; V_R = 0 V	[3]	-	0.1	-	%
		f = 1 MHz; V_R = 2.5 V		-	0.1	-	%
r_{dyn}	dynamic resistance	I_R = 10 A	[2][4]	-	1.5	-	Ω

[1] Device stressed with 8/20 μs exponential decay waveform according to IEC 61000-4-5 and IEC 61643-321.

[2] Measured from pin 1 or 2 to 3.

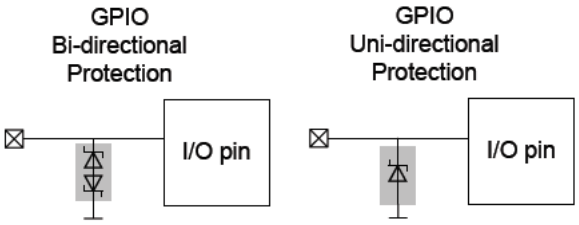
[3] ΔC_d is the difference of the capacitance measured between pin 1 and pin 3 and the capacitance measured between pin 2 and pin 3.

[4] Non-repetitive current pulse, Transmission Line Pulse (TLP) t_p = 100 ns; square pulse; ANS/IESD STM5.1-2008.

- **Dále následují ukázky TVS od firmy Nexperia.**
  - Jedná se o výstřižky z příručky „*Application guide: ESD protection*“
  - https://assets.nexperia.com/documents/brochure/Nexperia_document_brochure_ESD-Protection-Applications_022017.pdf

8.5.3.2 Ochrana univerzálních digitálních I/O

- Pro pomalé DI a DO (TVS má dovolenu vyšší parazitní kapacitu - desítky až stovky pF).



Obrázek 90 TVS - ochrana GPIO – zapojení

Tabulka 4 TVS - ochrana GPIO – seznam vhodných obvodů

Selection guide

Part	Package	Size (mm)	ESD robustness (IEC61000-4-2)	Configuration
PESD5Z6.0	SOD523	1.2 x 0.8 x 0.6	30 kV	1 channel unidirectional
PESD3V3U1UL	DFN1006-2	1.0 x 0.6 x 0.48	9 kV	1 channel unidirectional
PESD5V0S1BL	DFN1006-2	1.0 x 0.6 x 0.48	30 kV	1 channel bidirectional
PESD5V0S1BA	SOD323	1.7 x 1.25 x 0.95	30 kV	1 channel bidirectional
PESD5V2S2UT	SOT23	2.9 x 1.3 x 1.0	30 kV	2 channel unidirectional/ 1 channel bidirectional
BZA456A	SOT457	2.9 x 1.5 x 1.0	8 kV	4 channel unidirectional
PESD3V3S4UD	SOT457	2.9 x 1.5 x 1.0	30 kV	4 channel unidirectional
PESD5V0V1BL	DFN1006-2	1.0 x 0.6 x 0.48	30 kV	1 channel bidirectional
PESD5V0X1UALD	DFN1006D-2	1.0 x 0.6 x 0.37	15 kV	1 channel unidirectional



Obrázek 91 Pouzdro SOT23



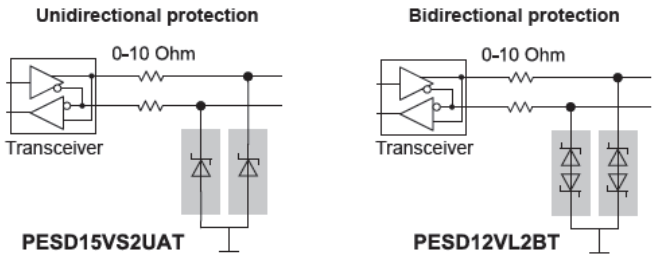
Obrázek 92 Pouzdro SOT457



Obrázek 93 Pouzdro DFN1006-2

8.5.3.3 Ochrana komunikační linky RS-232 a RS-485

- Komunikační rychlost do 1 Mbit/s.
- Pracovní napětí na datových vodičích ±12 V.



Obrázek 94 TVS – ochrana RS-232 – zapojení

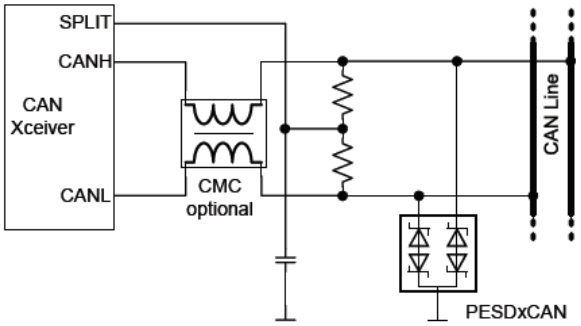
Tabulka 5 TVS - ochrana RS-232 – seznam vhodných obvodů

Selection guide

Part	Package	Size (mm)	No. of Lines	V_{RWM}	ESD robustness (IEC61000-4-2)
PESD15VS2UT	SOT23	2.9 x 1.3 x 1.0	2 unidirectional / 1 bidirectional	15 V	30 kV
PESD12VL2BT	SOT23	2.9 x 1.3 x 1.0	2 bidirectional	12 V	30 kV

8.5.3.4 Ochrana komunikační linky CAN

- Komunikační rychlost do 1 Mbit/s.
- Pracovní napětí na datových vodičích 5 V.
- Automotive – pro napětí AKU 12 V.



Obrázek 95 TVS – ochrana CAN – zapojení

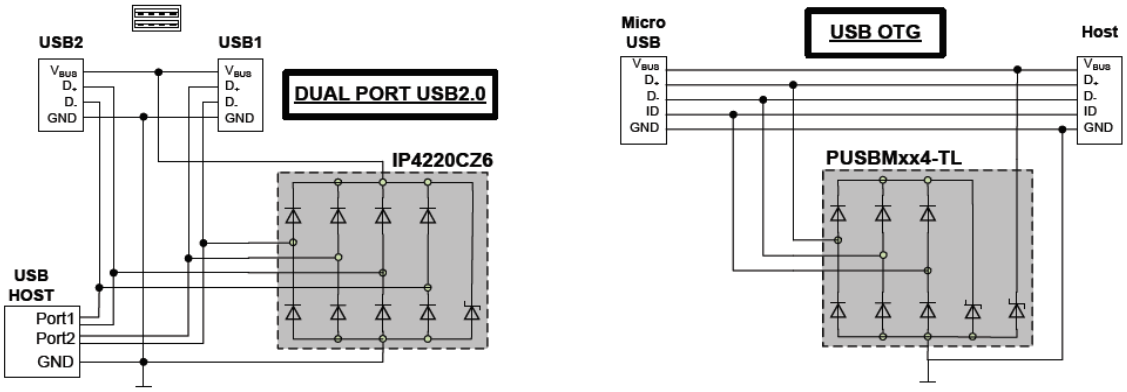
Tabulka 6 TVS – ochrana CAN – seznam vhodných obvodů

Selection guide

Part	Package	Size (mm)	ESD robustness (IEC61000-4-2)	C_d (max)	P_{ppm} at t_p = 8/20µs	V_{RWM}	AEC-Q101	Note
PESD2IVN-U	SOT323	2.0 x 1.25 x 0.95	18 kV	11 pF	150 W	26.5 V	Yes	NEW
PESD1CAN-U	SOT323	2.0 x 1.25 x 0.95	23 kV	12 pF	150 W	24 V	Yes	
PESD1CAN	SOT23	2.9 x 1.3 x 1.0	23 kV	17 pF	200 W	24 V	Yes	
PESD2CAN	SOT23	2.9 x 1.3 x 1.0	30 kV	30 pF	230 W	24 V	Yes	

8.5.3.5 Ochrana komunikační linky USB

- USB 2.0 - max. komunikační rychlost 480 Mbit/s
- Vysoká komunikační rychlost → je potřeba TVS s malou parazitní kapacitou (jednotky pF).
- Pracovní napětí: 3,3 V data a 5 V napájení V_{BUS}.
- Miniaturní rozměry TVS – dnes nutnost (mobily apod.).



Obrázek 96 TVS - ochrana USB – zapojení

Tabulka 7 TVS - ochrana USB – seznam vhodných obvodů

Selection guide

Part	Package	Size (mm)	C_d	ESD robustness (IEC61000-4-2)	Note
PUSB2X4Y	SOT363	2.0 x 1.25 x 0.95	0.55 pF	12 kV	4 lines, ultra-low clamping NEW
PUSB2X4D	SOT457	2.9 x 1.5 x 1.0	0.55 pF	12 kV	4 lines, ultra-low clamping NEW
PESD5V0F5UV	SOT666	1.6 x 1.2 x 0.55	0.55 pF	8 kV	5 lines, uni-directional
PUSBMxxVX4-TL	DFN1616-6	1.6 x 1.6 x 0.5	0.8 pF	8 kV	Add Vbus protection for USB OTG & charging
PESD5V0X1ULD	DFN1006D-2	1.0 x 0.6 x 0.37	0.95 pF	8 kV	1 line, unidirectional
IP4220CZ6	SOT457	2.9 x 1.5 x 1.0	1 pF	8 kV	4 lines
IP4221CZ6-S	DFN1410-6	1.45 x 1.0 x 0.48	1 pF	8 kV	IP4220 with smaller package
IP4221CZ6-XS	DFN1010-6 (XSON6)	1.0 x 1.0 x 0.48	1 pF	8 kV	IP4220 with smaller package
PRTR5V0U4D	SOT457	2.9 x 1.5 x 1.0	1 pF	8 kV	4 lines, AEC-Q101 qualified
IP3319CX6	WLCSP6	1.35 x 0.95 x 0.57	1.5 pF	15 kV	Common Mode Filter + ESD protection (1 line pair CMF + 1 line pair ESD protection) NEW

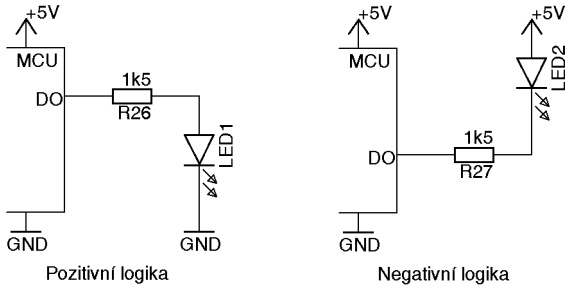
9 Zapojení uživatelského výstupu k DO

Základní zásady:

- Pin MCU musí být nastaven jako výstupní.
- Výstupní pin MCU má omezenou proudovou zatížitelnost (obvykle do 20 mA), pro ovládání větších zátěží je výstup nutné posílit (např. tranzistorem).
    - U některých MCU mohou být vybrané porty typu otevřený kolektor, tj. nejsou schopné dodat žádný proud v úrovni H (protože spínají pouze k L). Úroveň H pak musí být zajištěna externím obvodem vně MCU (např. pull-up rezistorem).
- Zajistit definovanou úroveň na pinu během resetu MCU (pokud je třeba)
    - Pomocí pull-up nebo pull-down rezistoru.
    - Některé MCU mají v průběhu resetu piny ve stavu vysoké impedance, některé je mají v definované logické úrovni (např. L).

9.1 Malá zátěž (např. LED)

- Pozitivní logika = LED svítí v úrovni H na DO
- Negativní logika = LED svítí v úrovni L na DO



Barva	Úbytek napětí
Infračervená	1,6 V
Červená	1,8 V až 2,1 V
Oranžová	2,2 V
Žlutá	2,4 V
Zelená	2,6 V
Modrá	3,0 V až 3,5 V
Bílá	3,0 V až 3,5 V
Ultrafialová	3,5 V

Obrázek 97 Připojení LED k výstupu MCU

- Výpočet předřadného odporu:

$$R = \frac{Vcc - U_{LED}}{I_{LED}} \tag{10}$$

Ve výpočtu se zanedbává výstupní odpor digitálního výstupu MCU.

Kde: Vcc – napájecí napětí [V]
U_{LED} – úbytek napětí LED v propustném směru [V] při proudu I_{LED}
I_{LED} – proud rozsvícenou LED [A] při napětí U_{LED}
R – předřadný odpor [Ω]

Např. pro červenou 20 mA LED:

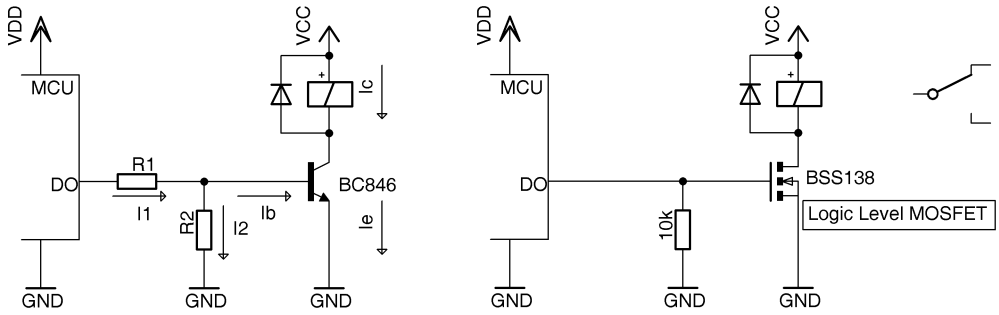
$$R = \frac{5-2}{0,020} = 150\,\Omega \tag{11}$$

Napětí LED v propustném směru závisí na barvě LED!

9.2 Posílení digitálního výstupu tranzistorem

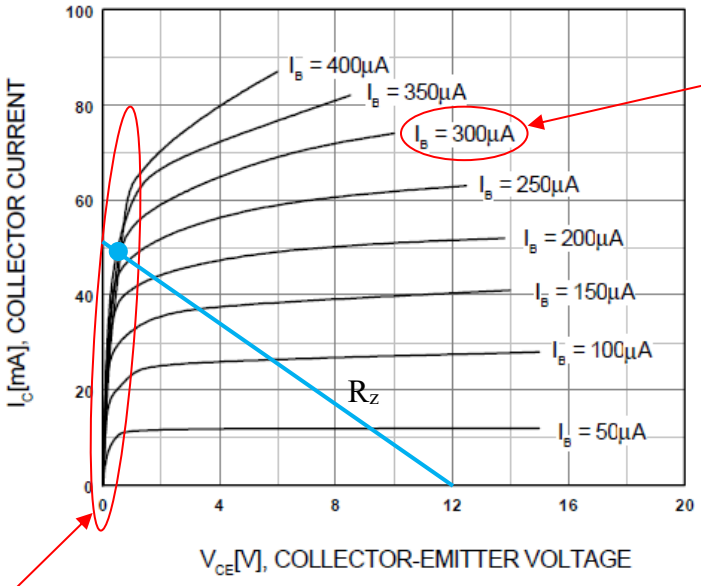
9.2.1 Bipolární NPN tranzistor a unipolární N-MOSFET

- Spíná zátěž ke GND.
- Zátěž sepnuta pokud DO = H.
- Zátěž může být napájena z jiného napětí než MCU.



Obrázek 98 Posílení DO tranzistorem NPN nebo N-MOSFET

- Při spínání **induktivní zátěže** je nutná **nulová dioda**. Dimenzovat jí podle napájecího napětí a proudu zátěže.
- Důležité jsou resistor R2 umístěné vůči GND v bázi bipolárního, respektive u G elektrody N-MOSFET. Rezistor zajišťuje, že transistor zůstane zavřený pokud bude MCU v resetu nebo bez napájení (DO na MCU ve stavu vysoké impedance).
- Tranzistory pracují ve spínacím režimu, tj. při sepnutí musí být dostatečně saturovány.



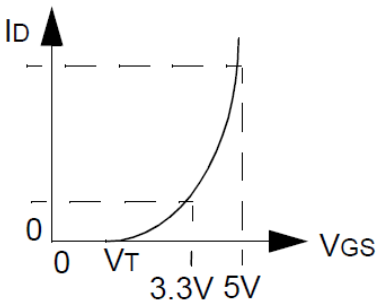
Min. proud do báze pro zátěž Rz.

Obrázek 99 Výstupní charakteristika NPN tranzistoru BC846

V této oblasti musí tranzistor pracovat v sepnutém stavu.

Při použití tranzistoru **N-MOSFET** musí tento být **logic-level**, tj. musí otevírat již při velmi malých napětích U_{GS}, když ho chceme spínat přímo z MCU. Vždy zkontrolovat prahové napětí $U_{GS(th)}$ v datasheetu tranzistoru, aby $U_{GS(th)} < V_{DD}$.

Tabulka 8 Parametry N-MOSFET tranzistoru IRF7467



RDS(on)	Static Drain-to-Source On-Resistance	—	9.4	12	mΩ	VGS = 10V, ID = 11A
		—	10.6	13.5		VGS = 4.5V, ID = 9.0A
		—	17	35		VGS = 2.8V, ID = 5.5A
VGS(th)	Gate Threshold Voltage	0.6	—	2.0	V	VDS = VGS, ID = 250 µA

Z datasheetu vyplývá, že tranzistor IRF7467 je vhodný i pro 3,3 V logiku.

Výpočet bázového rezistoru pro NPN tranzistor

Tabulka 9 Parametry NPN tranzistoru

Characteristic	Sym	Min	Max	Unit	Test Condition
OFF CHARACTERISTICS					
Collector-Base Breakdown voltage	V(BR)CBO	60	—	V	IC = 50 µA, IE = 0
Collector-Emitter Break-down Voltage	V(BR)CEO	50	—	V	IC = 1.0 mA, IB = 0
Emitter-Base Breakdown Voltage	V(BR)EBO	7.0	—	V	IE = 50 µA, IC = 0
Collector Cutoff Current	ICBO	—	100	nA	VCB = 60V
Emitter Cutoff Current	IEBO	—	100	nA	VEB = 7.0V
ON CHARACTERISTICS					
DC Current Gain	h_{FE}	120 180 270	270 390 560	—	VCE = 6.0V, IC = 1.0 mA
Collector-Emitter Saturation Voltage	VCE(SAT)	—	0.4	V	IC = 50 mA, IB = 5.0 mA

Nejprve stanovíme proud do báze tranzistoru. Pro výpočet použijeme nejmenší součinitel zesílení h_{FE} uváděné v datasheetu (aby platilo $I_B \geq \frac{I_C}{h_{FE\,\min}}$).

$$I_B = \frac{I_C}{h_{FE\min}} = \frac{\frac{V_{CC}}{R_Z}}{h_{FE\min}} = \frac{V_{CC}}{h_{FE\min} \cdot R_Z} \quad (12)$$

Kde: I_B – proud báze tranzistoru v sepnutém stavu

I_C – proud kolektorem tranzistoru (proud protékající zátěží) v sepnutém stavu

h_{FEmin} – zesilovací činitel tranzistoru (min. hodnota)

V_{CC} – napájecí napětí zátěže (a zároveň napětí na zátěži při sepnutém tranzistoru při zanedbání úbytku napětí kolektor-emitor tranzistoru)

R_Z – odpor zátěže

Nyní vypočteme potřebnou velikost odporu do báze. Pokud platí $R_1 = R_2$ (obvykle se takto volí), platí pro odpor $R_1 = R_2$ vztah:

$$R_1 = R_2 = \frac{V_{DD} - 2 \cdot U_{BE}}{I_B} \quad (13)$$

Kde: R_1, R_2 – odpory v bázi tranzistoru

I_B – proud báze tranzistoru v sepnutém stavu

V_{DD} – napájecí napětí MCU (napětí na pinu DO v logické úrovni H)

U_{BE} – napětí báze-emitor tranzistoru v sepnutém stavu

Pokud platí $R_1 = R_2$, rezistorem R_2 prochází proud I_2 o velikosti cca 10% I_B. Proud I_1, který prochází přes pin MCU do GND činí cca 110% I_B.

V odvození byl zanedbán úbytek napětí kolektor-emitor tranzistoru v sepnutém stavu (parametr $V_{CE(SAT)}$). Pokud by tento úbytek nebyl zanedbatelný, nahradíme ve vztazích napětí V_{CC} napětím $V_{CC} - V_{CE(SAT)}$, kde $V_{CE(SAT)}$ je úbytek napětí kolektor-emitor v sepnutém stavu (závislý na proudu I_B).

Př.: Tranzistor BC846 (U_{CE} = 80 V, I_{Cmax} = 100 mA)

V_{DD} = 5 V

V_{CC} = 12 V

R_Z = 200 Ω

h_{FEmin} = 110

$$I_Z = \frac{V_{CC}}{R_Z} = \frac{12}{200} = 0,060\,\text{A}$$

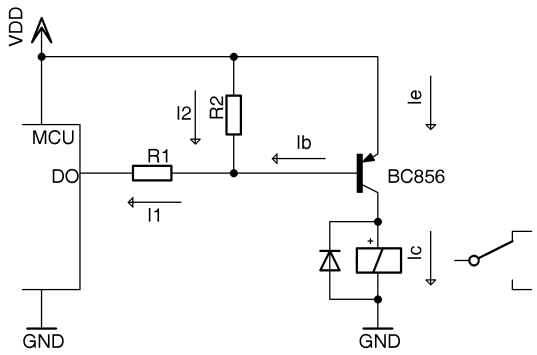
$$I_B = \frac{I_C}{h_{FE\min}} = \frac{0,060}{110} = 0,55\,\text{mA}$$

$$R_1 = R_2 = \frac{V_{DD} - 2 \cdot U_{BE}}{I_B} = \frac{5 - 2 \cdot 0,7}{0,00055} = 6545\,\Omega \rightarrow 6200\,\Omega\,(\text{z řady E12})$$

Pro zajištění toho, že tranzistor **bude vždy dostatečně saturován a naplno otevřen, je vhodné I_B zvýšit např. 2x** za cenu růstu celkové spotřeby zařízení.

9.2.2 Bipolární PNP tranzistor

- Spíná zátěž k Vcc.
- Zátěž sepnuta pokud DO = L.
- V následujícím jednoduchém zapojení nelze zátěž napájet z napětí vyššího než je napájecí napětí MCU, byl by problém udržet tranzistor ve vypnutém stavu.



Obrázek 100 Posílení DO tranzistorem PNP

- Při spínání induktivní zátěže je nutná nulová dioda. Dimenzovat jí podle napájecího napětí a proudu zátěže.

Výpočet rezistorů v bázi PNP tranzistoru

Výpočet je stejný jako u NPN tranzistoru. Pro úplnost je zde opět uveden.
Nejprve stanovíme proud do báze tranzistoru (stejně jako u NPN). Pro výpočet použijeme nejmenší součinitel zesílení h_{FE} uváděné v datasheetu (aby platilo $I_B \geq \frac{I_C}{h_{FE\min}}$).

$$I_B = \frac{I_C}{h_{FE\min}} = \frac{\frac{V_{DD}}{R_Z}}{h_{FE\min}} = \frac{V_{DD}}{h_{FE\min} \cdot R_Z} \tag{14}$$

Kde: I_B – proud báze tranzistoru v sepnutém stavu
I_C – proud kolektorem tranzistoru (proud protékající zátěží) v sepnutém stavu
h_{FEmin} – zesilovací činitel tranzistoru (min. hodnota)
V_{DD} – napájecí napětí zátěže (a zároveň napětí na zátěži při sepnutém tranzistoru při zanedbání úbytku napětí kolektor-emitor tranzistoru)
R_Z – odpor zátěže

Nyní vypočteme potřebnou velikost odporu do báze. Pokud platí $R_1 = R_2$ (obvykle se takto volí), platí pro odpor R_B vztah:

$$R_1 = R_2 = \frac{V_{DD} - 2 \cdot U_{BE}}{I_B} \tag{15}$$

Kde: R_1, R_2 – odpory v bázi tranzistoru
I_B – proud báze tranzistoru v sepnutém stavu

V_{DD} – napájecí napětí MCU (napětí na pinu DO v logické úrovni H)

U_{BE} – napětí báze-emitor tranzistoru v sepnutém stavu

Pokud platí $R_1 = R_2$, rezistorem R_2 prochází proud I_2 o velikosti cca 10% I_B. Proud I_1, který prochází přes pin MCU do GND činí cca 110% I_B.

Př.: Tranzistor BC556 (U_{CE} = -65 V, I_{Cmax} = -100 mA)

V_{DD} = 5 V

R_Z = 100 Ω

h_{FEmin} = 110

$$I_C = \frac{V_{DD}}{R_Z} = \frac{5}{100} = 0,05\,\text{A}$$

$$I_B = \frac{I_C}{h_{FE\min}} = \frac{0,05}{110} = 0,454\,\text{mA}$$

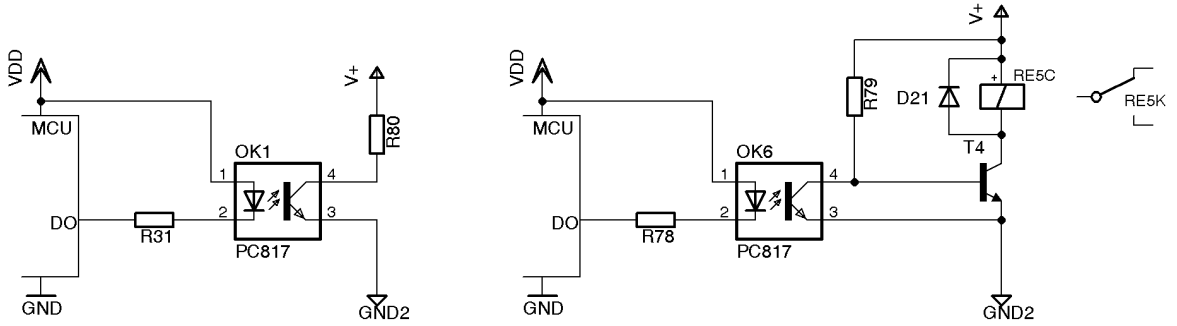
$$R_B = \frac{\left(V_{DD} - 2 \cdot U_{BE}\right)}{I_B} = \frac{\left(5 - 1,4\right)}{0,000454} = 7929\,\Omega \rightarrow 6800\,\Omega \text{ (z řady E12)}$$

Pro zajištění toho, že tranzistor bude vždy dostatečně saturován a naplno otevřen, je vhodné I_B zvýšit např. 2x za cenu růstu celkové spotřeby zařízení.

9.3 Galvanické oddělení zátěže

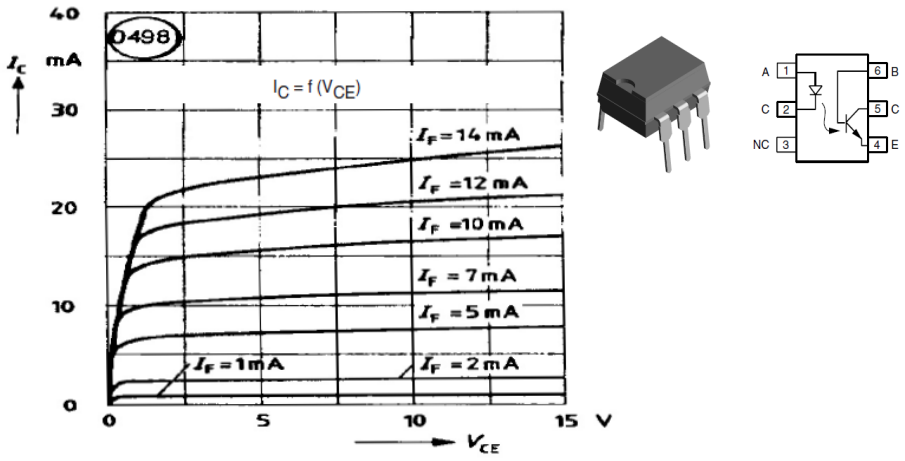
- GO zátěže = MCU a spínaná zátěž jsou napájené ze dvou galvanicky oddělených zdrojů (zdroje nemají společný žádný z pólů)
- V některých případech je vhodné (nutné) galvanicky oddělit spínanou zátěž od MCU.
    - Oddělení od výkonových el. obvodů (ochrana při poruše výkonové části, omezení průniku rušení od výkonové elektroniky).
    - Zamezení zemních smyček (omezení rušení).

9.3.1 Pomalé DO



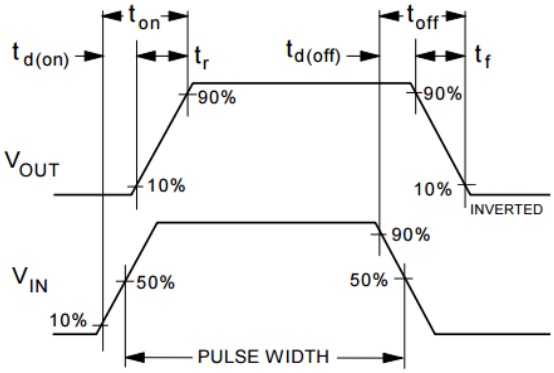
Obrázek 101 Galvanické oddělení zátěže od MCU

- **Optron** - pro realizaci GO
    - Max. trvalý I_F (proud diodou) a I_C (proud kolektorem tranzistoru) obvykle 50 až 100 mA.
    - Napětí U_F obvykle 1,4 V a U_{CE} do 70 V.
    - Optrony mají malé zesílení (je třeba velký I_F pro daný I_C) → nelze jimi spínat velké zátěže.
        - Parametr CTR (Current Transfer Ratio = poměr I_C/I_F) v rozsahu 0,4 až 3.



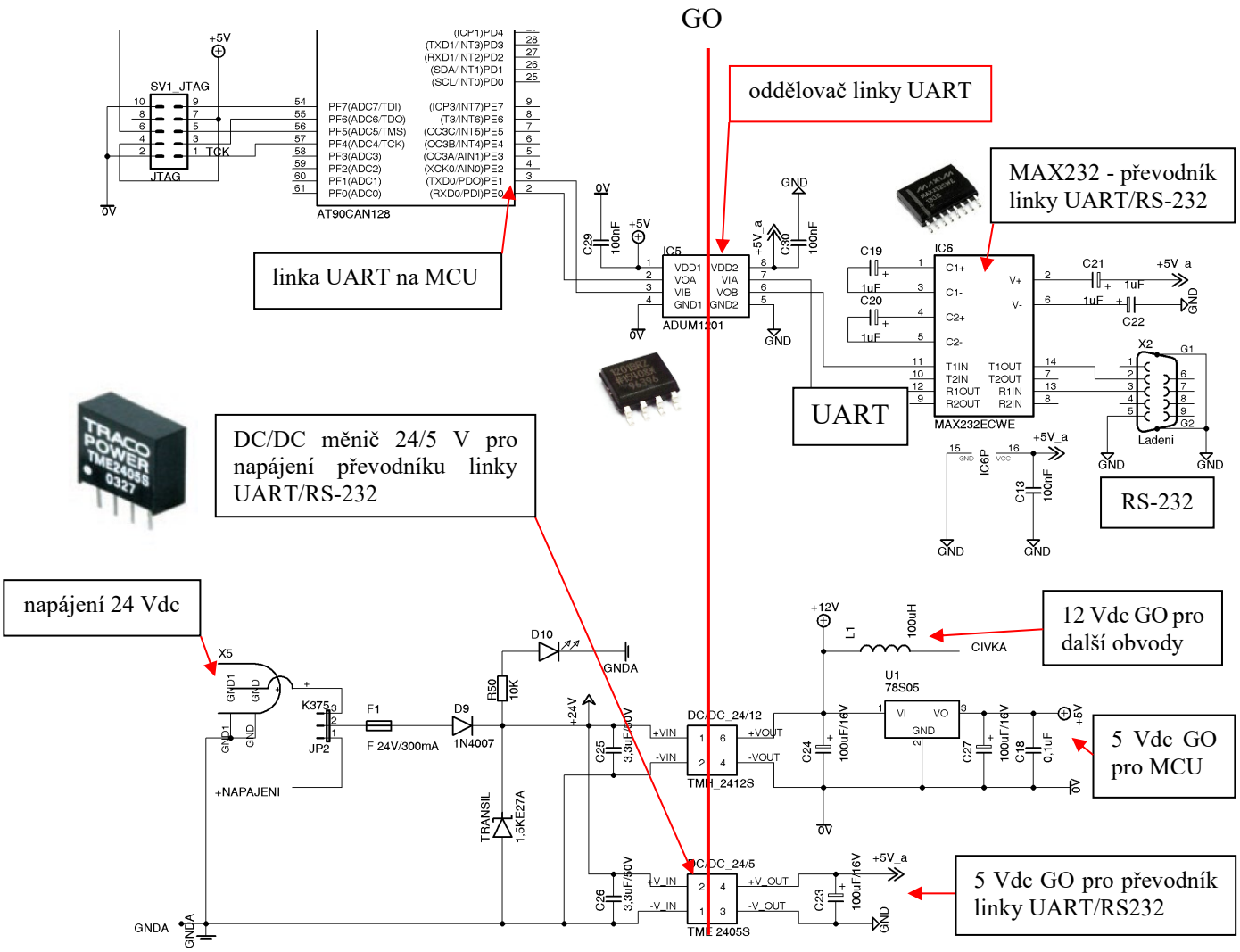
Obrázek 102 Výstupní charakteristika optronu CNY17

- Izolační pevnost.
  - Řádově kV.
- Rychlost spínání.
  - Do stovek Hz až jednotek kHz.
  - Doba náběhu (t_r) – jednotky µs.
  - Doba poklesu (t_f) – jednotky µs.
  - Doba sepnutí (t_{on}) – jednotky µs.
  - Doba vypnutí (t_{off}) – desítky µs.



9.3.2 Rychlé DO

- Pro galvanické oddělení komunikačních linek (sběrnic).
- Požadovaná rychlost spínání desítky kHz až desítky MHz.
- Používají se speciální oddělovače schopné pracovat na vysokých frekvencích.



Obrázek 103 Galvanické oddělení sběrnic

- Např. řada **ADuM** digitálních oddělovačů od firmy Analog Devices.
  - Není to optron, používá monolitický transformátor.
  - Přenosová rychlost až 150 Mbps, dle typu.
  - Napájecí napětí 5 V nebo 3 V (na každé straně může být jiné).
  - Až 6 kanálů (linek).
  - Spotřeba proudu - jednotky mA na kanál, dle přenosové rychlosti.
  - El. pevnost 2,5 kV i vyšší.
  - SMD pouzdro, např. SO-8.
  - Jednoduché zapojení, nevyžaduje žádné okolní součástky.

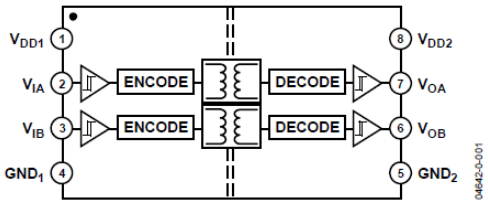


Figure 1. ADuM1200 Functional Block Diagram

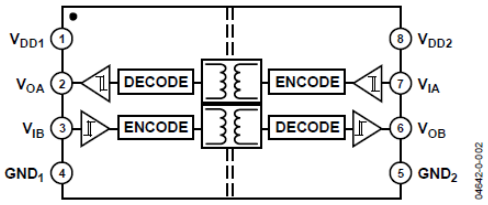
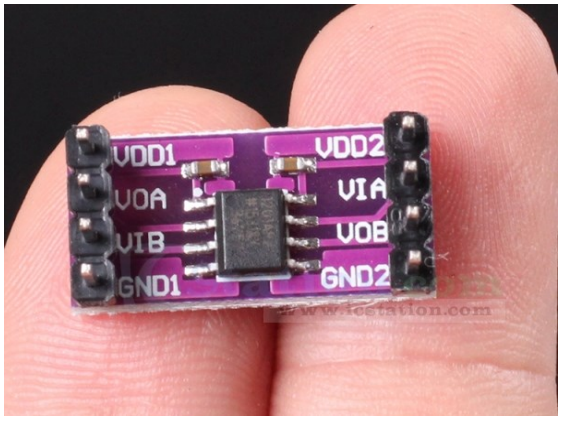


Figure 2. ADuM1201 Functional Block Diagram

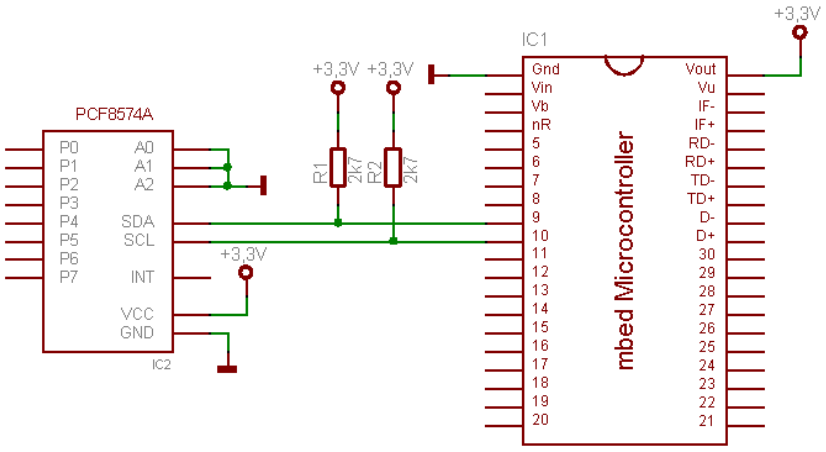
Obrázek 104 Bloková struktura digitálního oddělovače ADuM1201



Obrázek 105 ADuM1201 v pouzdře SO-8

10 Rozšíření počtu vstupů a výstupů

- Proč?
  - Počet potřebných vstupů a/nebo výstupů je větší než počet volných pinů na MCU.
- Jak?
  - Použití expandérů I/O – připojitelných k MCU např. přes sériovou komunikační sběrnici I2C nebo SPI.

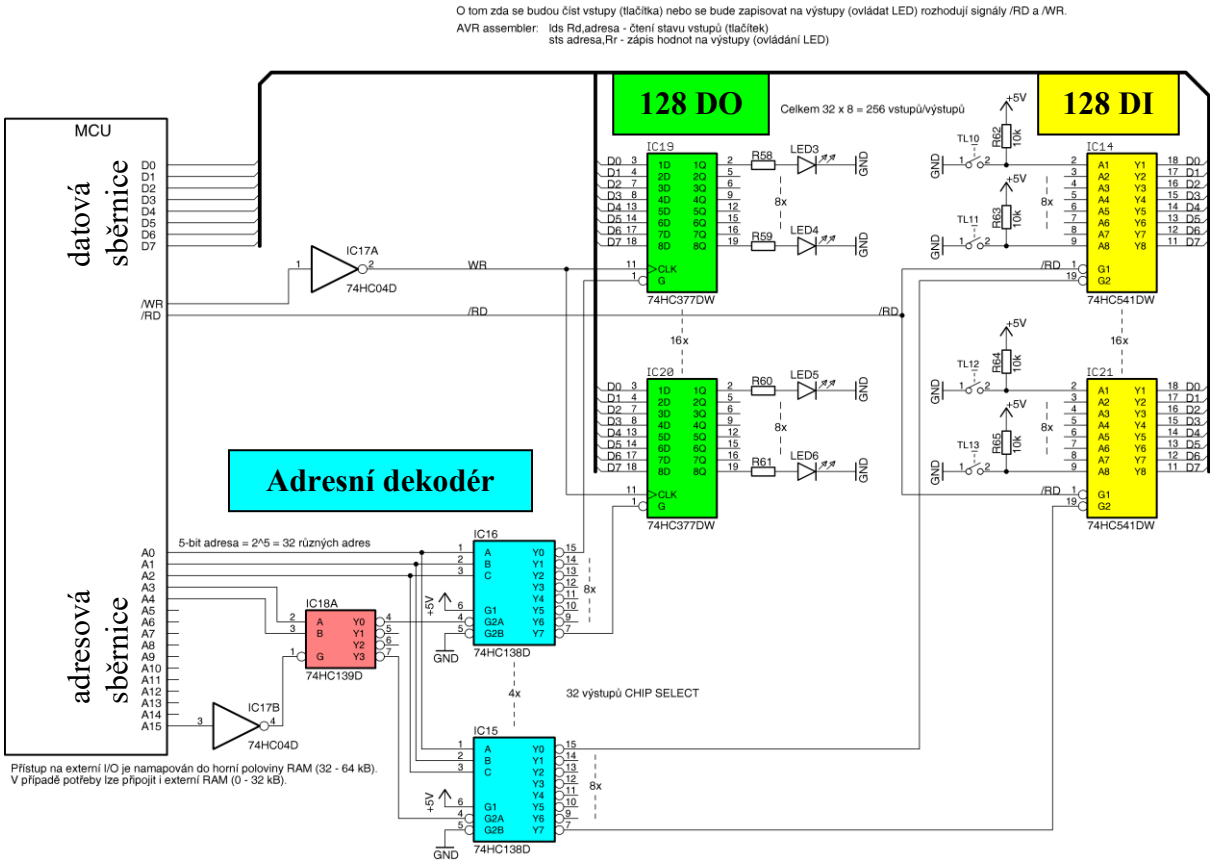


Obrázek 106 Expandér I/O PCF8574A a jeho připojení k MCU přčes sběrnici I2C

  - Použití pomocných logických obvodů (řady 4xxx a 74xxx) vně MCU.
    - Rozšíření vstupů – kodéry, multiplexery.
    - Rozšíření výstupů – dekodéry, demultiplexery.
    - Mapování I/O do paměti dat – přístup na I/O jako do externí RAM.

10.1 Mapování I/O do adresního prostoru externí paměti

- MCU musí podporovat připojení externí RAM (mít vyvedeno rozhraní).
- Čtení vstupů a zápis na výstupy probíhá jako čtení a zápis z/do externí RAM.
- Každý externí digitální vstup (DI) a výstup (DO) má svojí adresu v RAM.



Obrázek 107 Rozšíření I/O mapováním do adresního prostoru RAM

Parametry obvodu

- Počet DI: 128 (tlačítka) → 16x 74HC541 (16x buffer/line driver)
- Počet DO: 128 (LED) → 16x 74HC377 (16x D flip-flop)
- Celkem DI + DO (DI/O): 256
- 8-bit datová sběrnice (D0:7) – pro čtení stavu z osmice digitálních vstupů (čtení) nebo pro zápis stavu na osmici digitálních výstupů (zápis). Co bit sběrnice to jeden digitální vstup (DI) nebo digitální výstup (DO).
- 16-bit adresová sběrnice (A0:A15) – pro výběh osmice DI nebo DO.
- Řídící signály /RD a /WR pro výběr mezi čtením DI a zápisem na DO.

DI

- Interface k DI je realizován pomocí několika 74HC541 (neinvertující 8 kanálový budič sběrnice).
- Po aktivaci čte logické hodnoty na svých vstupech A1:8 a beze změny je vystavuje na svých výstupech Y1:8.
- Pokud není aktivní, tak jsou výstupy ve stavu vysoké impedance (Z).
- Aktivace obvodu pomocí vstupů /G1 a /G2 (v Tabulka 10 označeny jako /OE1 a /OE2)

Tabulka 10 Pravdivostní tabulka obvodu 74HC541

INPUTS			OUTPUT
/G1	/G2	A_n	Y_n
L	L	L	L
L	L	H	H
X	H	X	Z
H	X	X	Z
Notes H = HIGH voltage level L = LOW voltage level X = don't care Z = high impedance OFF-state			

DO

- Interface k DO je realizován pomocí několika 74HC377 (8x D klopný obvod).
- Obvod na vzestupnou hranu na svém vstupu CLK (za předpokladu, že je aktivován, tj. /G = L), přenese data (požadovaný stav osmice DO) na své výstupy Q1:8. Tento stav na svých výstupech udržuje až do příchodu další vzestupné hrany na CLK.

Tabulka 11 Pravdivostní tabulka obvodu 74HC377

Operating modes	INPUTS			OUTPUTS
	CLK	/G	D_n	Q_n
load "1"	↑	l	h	H
load "0"	↑	l	l	L
hold (do nothing)	↑	h	X	no change
	X	H	X	no change
Notes H = HIGH voltage level h = HIGH voltage level one set-up time prior to the LOW-to-HGH clock transition L = LOW voltage level l = LOW voltage level one set-up time prior to the LOW-to-HGH clock transition X = don't care ↑ = LOW-to-HIGH transition				

Adresní dekodér

- Je tvořen jedním dekodérem 1 ze 4 (74HC139) a čtyřmi dekodéry 1 z 8 (74HC138).
- Dle adresy A0:15 se aktivuje jeden CHIP SELECT (CS) signál (CS = L), kterým se aktivuje 1 ze 32 obvodů, které realizují rozhraní pro digitálních vstupů (74HC541) a výstupů (74HC377).
- Signálů CS je tolik, kolik je obvodů realizujících rozhraní DI a DO. V uvedeném zapojení jich je 32 (16x 74HC541 + 16x 74HC377).
- V každém okamžiku je aktivní vždy pouze 1 signál CS nebo není aktivní žádný, pokud se nepřistupuje na externí DI/O.
- Adresování
    - Je využito pouze 6 bitů ze 16-bitové adresy: A0:A4 (spodních 5 bitů) + A15 (nejvyšší bit)
    - A15 (nejvyšší bit adresy) – rozděluje adresní prostor na 2 poloviny.
        - Adresní prostor v rozsahu adres 0 až 32767 lze využít pro připojení externí RAM (není použito ve výše uvedeném příkladu).

        - Adresní prostor v rozsahu adres od 32768 do 65535 je použit pro přístup na externí DI a DO.
    - A0:4 jsou zavedené do dekodérů 1 ze 4 a 1 z 8 a na jejich základě se vybírá jeden ze signálů CHIP SELECT.
    - Mapa adres jednotlivých osmic DI a DO je uvedena v Tabulka 12.

Tabulka 12 Mapa adres expandéru DI / DO

pořadí	adresa (DEC)	adresa (HEX)	A15	A4	A3	A2	A1	A0	DI/DO IC	typ DI nebo DO
0	32768	8000	1	0	0	0	0	0	74HC377 č. 0	LED 0 až 7
1	32769	8001	1	0	0	0	0	1	74HC377 č. 1	LED 8 až 15
2	32770	8002	1	0	0	0	1	0	74HC377 č. 2	LED 16 až 23
3	32771	8003	1	0	0	0	1	1	74HC377 č. 3	LED 24 až 31
4	32772	8004	1	0	0	1	0	0	74HC377 č. 4	LED 32 až 39
5	32773	8005	1	0	0	1	0	1	74HC377 č. 5	LED 40 až 47
6	32774	8006	1	0	0	1	1	0	74HC377 č. 6	LED 48 až 55
7	32775	8007	1	0	0	1	1	1	74HC377 č. 7	LED 56 až 63
8	32776	8008	1	0	1	0	0	0	74HC377 č. 8	LED 64 až 71
9	32777	8009	1	0	1	0	0	1	74HC377 č. 9	LED 72 až 79
10	32778	800A	1	0	1	0	1	0	74HC377 č. 10	LED 80 až 87
11	32779	800B	1	0	1	0	1	1	74HC377 č. 11	LED 88 až 95
12	32780	800C	1	0	1	1	0	0	74HC377 č. 12	LED 96 až 103
13	32781	800D	1	0	1	1	0	1	74HC377 č. 13	LED 104 až 111
14	32782	800E	1	0	1	1	1	0	74HC377 č. 14	LED 112 až 119
15	32783	800F	1	0	1	1	1	1	74HC377 č. 15	LED 119 až 127
16	32784	8010	1	1	0	0	0	0	74HC541 č. 0	TL 0 až 7
17	32785	8011	1	1	0	0	0	1	74HC541 č. 1	TL 8 až 15
18	32786	8012	1	1	0	0	1	0	74HC541 č. 2	TL 16 až 23
19	32787	8013	1	1	0	0	1	1	74HC541 č. 3	TL 24 až 31
20	32788	8014	1	1	0	1	0	0	74HC541 č. 4	TL 32 až 39
21	32789	8015	1	1	0	1	0	1	74HC541 č. 5	TL 40 až 47
22	32790	8016	1	1	0	1	1	0	74HC541 č. 6	TL 48 až 57
23	32791	8017	1	1	0	1	1	1	74HC541 č. 7	TL 58 až 63
24	32792	8018	1	1	1	0	0	0	74HC541 č. 8	TL 64 až 71
25	32793	8019	1	1	1	0	0	1	74HC541 č. 9	TL 72 až 79
26	32794	801A	1	1	1	0	1	0	74HC541 č. 10	TL 80 až 87
27	32795	801B	1	1	1	0	1	1	74HC541 č. 11	TL 88 až 95
28	32796	801C	1	1	1	1	0	0	74HC541 č. 12	TL 96 až 103
29	32797	801D	1	1	1	1	0	1	74HC541 č. 13	TL 104 až 111
30	32798	801E	1	1	1	1	1	0	74HC541 č. 14	TL 112 až 119
31	32799	801F	1	1	1	1	1	1	74HC541 č. 15	TL 119 až 127

V tabulce se uvažuje A14:5 = 0

Přístup na externí I/O je do adresního prostoru RAM namapován celkem 1024x (opakuje se 1024x).

od adr. (HEX)	do adr. (HEX)	
0	7FFF	Externí RAM
8000	801F	Expanze I/O (32 adres)
8020	803F	Expanze I/O (32 adres)
.	.	Expanze I/O (32 adres)
FFE0	FFFF	Expanze I/O (32 adres)

Stručný popis funkce celého obvodu

- **Čtení DI**
  - MCU vystaví adresu na A0:15.
  - Vygenerování signálu CS (Chip Select).
  - MCU aktivuje čtení (/RD = L).
  - Aktivace vybraného 74HC541 → vystaví stav osmice DI na datovou sběrnici D0:7.
  - MCU přečte stav vybrané osmice DI z datové sběrnice D0:7.
- **Zápis na DO**
  - MCU vystaví adresu na A0:15.
  - Vygenerování signálu CS (Chip Select).
  - MCU vystaví požadovaný stav vybrané osmice DO na datovou sběrnici D0:7.
  - MCU aktivuje zápis (/WR = L).
  - Aktivace vybraného 74HC377 → přečte data z D0:7 a přepíše je na své výstupy Q1:8 (osmice DO).

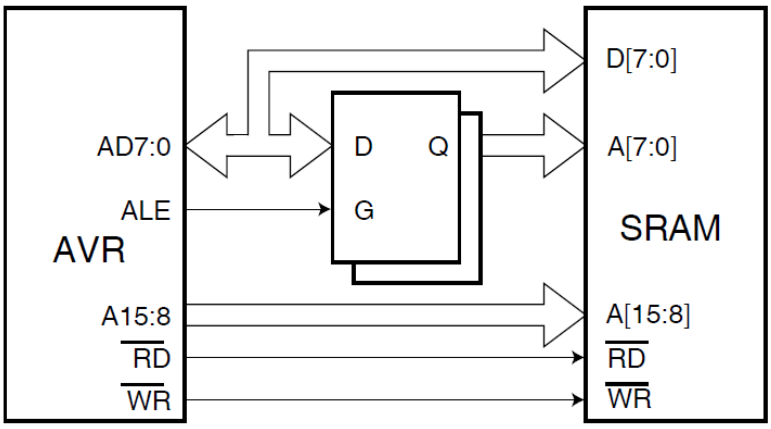
Podrobný popis funkce celého obvodu

- MCU vystaví na adresní sběrnici A0:15 adresu.
- A15 ovládá enable vstup /G prvního dekodéru 1 ze 4 (74HC139). Pokud výstup A15 = H → vstup /G = L a dle adresy A3:4 se na jednom z výstupů Y0 až Y3 obvodu 74HC139 objeví úroveň L, která následně aktivuje jeden ze 4 dekodérů 1 z 8 (74HC138). Podle adresy A0:2 se následně aktivuje jeden z osmi výstupů Y0 až Y7 na vybraném dekodéru 1 z 8. Aktivace znamená, že Yn = L. Tímto byl vygenerován signál CHIP SELECT, který aktivuje 1 ze 32 obvodů sloužících jako interface vždy pro osmici DI nebo DO (obvody 74HC377 pro DO a 74HC541 pro DI).
- Podle směru komunikace, který je dán použitou instrukcí pro přístup do externí RAM (buď čtení nebo zápis), MCU aktivuje (úroveň L) buď signál /RD pro čtení nebo signál /WR pro zápis dat na datové sběrnici D0:7. Signál /RD je zaveden do jednoho z povolovacích vstupů všech obvodů 74HC541 (DI). Signál /WR přes je invertor zaveden do vstupu CLK všech obvodů 74HC377 (DO), invertor je použit z toho důvodu, že vstup CLK u D klopných obvodů 74HC377 reaguje na vzestupnou hranu, ale MCU v uvedeném schématu má opačnou polaritu signálu /WR (aktivní v L, nikoliv v H jak by bylo potřeba). Signály /RD a /WR společně se signálem CS slouží pro výběr (aktivaci) jednoho z výše uvedených obvodů, tedy pro přístup na vybranou osmici vstupů nebo výstupů.

- Čtení DI
  - MCU generuje signál pro čtení: /RD = L, /WR = H.
  - Dle CS dojde k aktivaci některého z obvodů 74HC541 (vstupy /G1 = /G2 = L u vybraného obvodu), přičemž signál CS již byl vygenerován před aktivací signálu /RD.
  - Vybraný 74HC541 přepíše na své výstupy Y1:8 hodnoty na svých vstupech A1:8 (stavy tlačítek), tím je vystaví na datovou sběrnici D0:7.
  - MCU si z datové sběrnice D0:7 přečte stav osmice DI.
- Zápis na DO
  - MCU vystaví na datové sběrnici D0:7 data (požadovaný stav vybrané osmice DO).
  - MCU generuje signál pro zápis: /WR = L, /RD = H.
  - Dle CS dojde k aktivaci některého z obvodů 74HC377 (vstup /G = L), přičemž signál CS již byl vygenerován před aktivací signálu /WR.
  - Vybraný 74HC377 registruje na svém vstupu CLK vzestupnou hranu, na kterou reaguje přepsáním bajtu z datové sběrnice D0:7 na své výstupy Q1:8.

Poznámky

- Zapojení uvedené na Obrázek 107 vyžaduje, aby MCU měl vyvedenou celou externí 16-bit adresovou sběrnici (tj. všech 16 bitů vyvedených na 16 pinů). Proto nelze uvedené zapojení použít bez úprav s MCU AVR, protože MCU AVR má multiplexovanou adresní sběrnici s datovou. Zapojení by se muselo upravit (přidat D KO pro multiplexování datové a adresní sběrnice).



Obrázek 108 Multiplexované připojení externí RAM k MCU AVR

11 Konverze napěťových úrovní

Proč?

- Existence digitálních obvodů s různým napájecím napětím (5 V, 3,3 V, …) na DPS.
- Logické úrovně digitálních obvodů mají jiné napěťové hladiny (podle napájecího napětí daného obvodu), výstupy se vstupy nelze přímo spojovat.
- Jak propojit tyto digitální obvody mezi sebou?
    - Používat pouze obvody se stejným napájecím napětím (někdy nelze).
    - Použít přizpůsobení – mezi výstupy a vstupy vložit mezičlánek (od jednoduchého mezičlánku tvořeného pasivními součástkami až po za tímto účelem určenému integrovanému obvodu - konvertoru napěťových úrovní).

11.1 Napěťové úrovně

- Různé rodiny logických obvodů (resp. typy vstupů/výstupů), různá napájecí napětí.

Tabulka 13 Rozhodné napěťové úrovně různých rodin logických obvodů

	V_{OH} (min)	V_{OL} (max)	V_{IH} (min)	V_{IL} (max)
5 V TTL	2,4 V	0,5 V	2,0 V	0,8 V
3,3 V LVTTL	2,4 V	0,4 V	2,0 V	0,8 V
5 V CMOS	4,7 V (Vcc – 0,3 V)	0,5 V	3,5 (0,7xVcc)	1,5 V (0,3xVcc)
3,3 V LVCMOS	3,0 V (Vcc – 0,3 V)	0,5 V	2,3 V (0,7xVcc)	1,0 V (0,3xVcc)
Poznámka: LV – Low Voltage OH – Output High OL – Output Low V_{OH} – min. výstupní napětí v úrovni H V_{OL} – max. výstupní napětí v úrovni L V_{IH} – min. vstupní napětí pro úroveň H V_{IL} – max. vstupní napětí pro úroveň L				

- Propojovat lze pouze obvody, jejichž výstupy a vstupy jsou navzájem kompatibilní, co se týče napěťových úrovní pro logické úrovně L a H.
- Výstup se vstupem dvou obvodů lze propojit pouze tehdy, pokud platí:

$$V_{OH} > V_{IH} \text{ a } V_{OL} < V_{IL}$$

11.2 Způsoby propojení

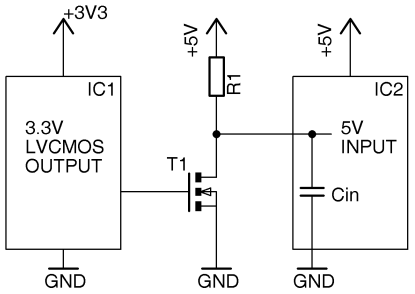
- Různé způsoby, podle složitosti a kvality mezičlenu.

11.2.1 3,3V → 5V přímé propojení

- Přímo lze propojit pouze:
  - 3,3V LVCMOS → 5V TTL (protože 3,0 V > 2,0 V a 0,5 V < 0,8 V)
  - 3,3V LVTTL → 5V TTL (protože 2,4 V > 2,0 V a 0,4 V < 0,8 V)

11.2.2 3,3V → 5V pomocí N-MOSFET tranzistoru

- Nevýhody
  - Provádí inverzi.
  - Velké množství součástek.
  - Nehodí se pro rychlý přenos dat (pull-up odpor omezuje rychlost spínání).



Obrázek 109 3,3V → 5V pomocí N-MOSFET

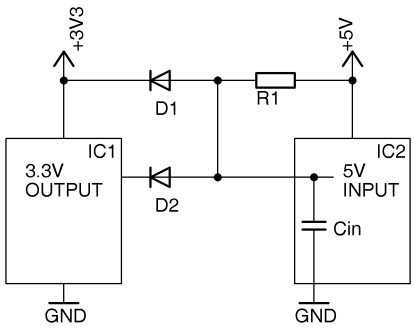
- Vlastnosti
  - T_1 musí být logic-level MOSFET ($U_{GS(th)} < 3$ V).
  - R_1 je třeba volit obezřetně na základě:
    - Požadované frekvence spínání.
    - Proudu skrz R_1 (T_1).
  - Různá rychlost spínání pro H → L a L → H (z pohledu 5V INPUT).
    - L → H pomalejší (C_{in} se nabíjí přes R_1).
    - H → L rychlejší (R_{dsON} <<< R_1 → C_{in} se vybíjí rychle).
  - Maximální rychlost spínání závisí na velikosti R_1, parazitní kapacitě 5 V vstupu (C_{in}) a parazitní kapacitě DPS (C_S) – řádově jednotky pF.

$$T_{SW} = 3 \cdot R_1 \cdot (C_{in} + C_S)$$
$$f_{SW} = \frac{1}{T_{SW}} \quad (16)$$

  - Za účelem zvýšení f_{SW} je možné R_1 zmenšit za cenu zvýšení spotřeby proudu.

11.2.3 3,3V → 5V pomocí děliče s diodami

- Výhody
  - Cena.
- Nevýhody
  - Velké množství součástek.
  - Nehodí se pro rychlý přenos dat (pull-up odpor omezuje rychlost spínání).



Obrázek 110 3,3V → 5V pomocí děliče s diodami

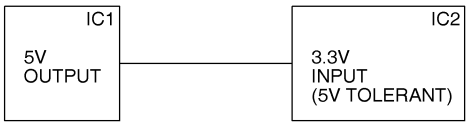
- Vlastnosti
  - Předpokládejme propojení 3,3 V výstupu LVTTL nebo LVCMOS se vstupem 5 V CMOS.
    - Přímo propojit nelze, protože: 2,4 V < 3,5 V i 3,0 V < 3,5 V

Tabulka 14 Prahová napětí vybraných rodin

	5 V CMOS input	3,3 V LVTTL output	3,3 V LVCMOS output
práh pro H	>3,5 V	>2,4 V	>3,0 V
práh pro L	<1,5 V	<0,4 V	<0,5 V

  - Pomocí diody D1 zvýšíme potřebné napětí výstupu nad 3,5 V.
  - Úroveň H je tvořena děličem R1 – D1. Dioda D1 pomáhá zvýšit potřebné napětí pro splnění $V_{IH(5V\ CMOS)}$.
    - Nové $V_{OH(3,3V)}$ = 3,3 V + 0,7 V = 4,1 V > 3,5 V.
  - Úroveň L je tvořena děličem R1 – D2. Původní $V_{OL(3,3V\ LVCMOS)}$ se zvýší o úbytek napětí na diodě, ale je stále s rezervou pod $V_{IL(5V\ CMOS)}$.
    - Nové $V_{OL(3,3V)}$ = 0,5 V + 0,7 V = 1,2 V < 1,5 V.
  - Velikost odporu R1
    - R1 musí být řádově menší než vstupní odpor 5 V vstupu, jinak by dělící poměr odporů znemožnil funkci obvodu.
    - R1 nesmí být příliš malý, musí se brát ohled na max. proud 3,3 V výstupu v úrovni L.
    - Maximální rychlost spínání závisí na velikosti R1, parazitní kapacitě 5 V vstupu (C_{in}), diod a DPS.

11.2.4 5V → 3,3V přímé propojení



Obrázek 111 Přímé propojení 5 V výstupu se 3,3 V vstupem

- Teoreticky lze obvody přímo propojit, protože napětí na 5 V výstupu vždy splňuje požadavky 3,3 V vstupu pro L i H úroveň, nezávisle na rodině (CMOS, TTL, LVCMOS, LVTTL).
  - **V_{OH}** 5 V výstupu je vždy vyšší než **V_{IH}** 3,3 V vstupu.
  - **V_{OL}** 5 V výstupu je vždy nižší než **V_{IL}** 3,3 V vstupu.

	V_{OH} (min)	V_{OL} (max)	V_{IH} (min)	V_{IL} (max)
5 V TTL	2,4 V	0,5 V	2,0 V	0,8 V
3,3 V LVTTL	2,4 V	0,4 V	2,0 V	0,8 V
5 V CMOS	4,7 V (Vcc – 0,3 V)	0,5 V	3,5 (0,7xVcc)	1,5 V (0,3xVcc)
3,3 V LVCMOS	3,0 V (Vcc – 0,3 V)	0,5 V	2,3 V (0,7xVcc)	1,0 V (0,3xVcc)

Poznámka:
LV – Low Voltage
OH – Output High
OL – Output Low
V_{OH} – min. výstupní napětí v úrovni H
V_{OL} – max. výstupní napětí v úrovni L
V_{IH} – min. vstupní napětí pro úroveň H
V_{IL} – max. vstupní napětí pro úroveň L

- Prakticky lze pouze tehdy, pokud je 3,3 V vstup „**5 V tolerant**“.
  - Protože napětí 5 V CMOS výstupu v úrovni H (4,7 V) překračuje maximální povolené napětí na 3,3 V vstupu.

Table 45-1. Absolute Maximum Ratings*

Operating Temperature (Industrial).....-40°C to +85°C
Storage Temperature.....-60°C to +150°C
Voltage on Input Pins with Respect to Ground (except VBUS).....-0.3V to +4.0V
Voltage on VBUS Pin with Respect to Ground.....-0.3V to +6.0V
Maximum Operating Voltage (VDDCORE).....2.0V
Maximum Operating Voltage (VDDIO).....4.0V
Total DC Output Current on all I/O lines
100-lead LQFP.....100 mA
144-lead LQFP.....130 mA
100-ball TFBGA.....150 mA
144-ball LFBGA.....150 mA

Obrázek 112 Povolené napětí na vstupních pinech MCU Atmel SAM3X/A (3,3 V obvod)

- Jak zjistím, zda je 3,3 V vstup „5 V tolerant“ ?
  - Pohledem do datasheetu dané součástky – viz níže úryvky z datasheetu dvou různých obvodů.

3,3 V varianta | 5 V varianta

RECOMMENDED OPERATING CONDITIONS

SYMBOL	PARAMETER	CONDITIONS	74AHC1G			74AHCT1G			UNIT
			MIN.	TYP.	MAX.	MIN.	TYP.	MAX.	
V_{CC}	supply voltage		2.0	5.0	5.5	4.5	5.0	5.5	V
V_I	input voltage		0	–	5.5	0	–	5.5	V
V_O	output voltage		0	–	V_{CC}	0	–	V_{CC}	V
T_{amb}	operating ambient temperature	see DC and AC characteristics per device	–40	+25	+125	–40	+25	+125	°C
t_r, t_f	input rise and fall times	V_{CC} = 3.3 ±0.3 V	–	–	100	–	–	–	ns/V
		V_{CC} = 5 ±0.5 V	–	–	20	–	–	20	ns/V

Obrázek 113 Vstup je „5 V tolerant“ (obvod 74AHC1G)

recommended operating conditions (see Note 4)

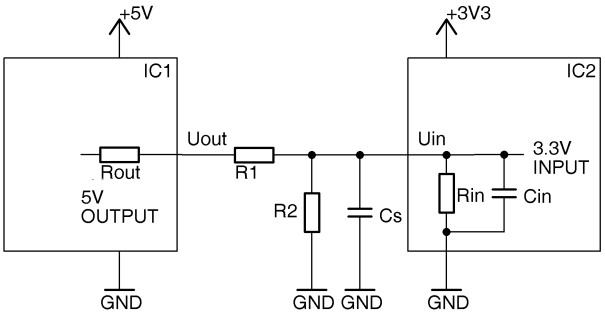
			MIN	MAX	UNIT
V_{CC}	Supply voltage	Operating	1.65	5.5	V
		Data retention only	1.5		
V_{IH}	High-level input voltage	V_{CC} = 1.65 V to 1.95 V	0.65 × V_{CC}		V
		V_{CC} = 2.3 V to 2.7 V	1.7		
		V_{CC} = 3 V to 3.6 V	2		
		V_{CC} = 4.5 V to 5.5 V	0.7 × V_{CC}		
V_{IL}	Low-level input voltage	V_{CC} = 1.65 V to 1.95 V		0.35 × V_{CC}	V
		V_{CC} = 2.3 V to 2.7 V		0.7	
		V_{CC} = 3 V to 3.6 V		0.8	
		V_{CC} = 4.5 V to 5.5 V		0.3 × V_{CC}	
V_I	Input voltage		0	5.5	V
V_O	Output voltage		0	V_{CC}	V

Obrázek 114 Vstup je „5 V tolerant“ (obvod 74LVC1G125)

- Pokud 3,3 V vstup není „5 V tolerant“, nelze obvody přímo propojit a musí se použít přizpůsobovací mezičlen.

11.2.5 5V → 3,3V s využitím odporového děliče

- Varianta, která napadne každého jako první.



Obrázek 115 5V → 3,3V s využitím odporového děliče

- Vlastnosti
  - Rout – typicky 10 Ω.
  - Rin – typicky větší než 500 kΩ.
  - Cin – parazitní kapacita vstupu (jednotky pF).
  - Cs – parazitní kapacita DPS (jednotky až desítky pF).
  - Parazitní zatěžovací kapacity Cin a Cs omezují frekvenci spínání.
  - Vztahy pro odporový dělič R1-R2 (zanedbány odpory Rout a Rin):

$$\frac{Vcc_{vyšší}}{Vcc_{nižší}} = \frac{R1 + R2}{R2} \tag{17}$$

$$R1 = \frac{Vcc_{vyšší} - Vcc_{nižší}}{Vcc_{nižší}} \cdot R2 \tag{18}$$

$$R2 = \frac{Vcc_{nižší}}{Vcc_{vyšší} - Vcc_{nižší}} \cdot R1 \tag{19}$$

  - Vztahy pro dělič při konverzi z 5 V na 3,3 V:

$$R1 = \frac{5 - 3,3}{3,3} \cdot R2 = 0,515 \cdot R2 \tag{20}$$

$$R2 = \frac{3,3}{5 - 3,3} \cdot R1 = 1,941 \cdot R1 \tag{21}$$

  - Volba odporů R1 a R2:
    - R1 >> Rout
    - R2 << Rin
    - R1 a R2 co největší – minimalizace spotřeby proudu
    - R1 a R2 co nejmenší – aby nabíjení a vybíjení parazitních kapacit bylo co nejrychlejší.

- Výsledek - kompromis ve stanovení hodnot R1 a R2.

- Maximální hodnoty R1, R2 lze určit na základě **odhadu doby náběhu/poklesu** signálu na vstupu:

$$t_r = -R \cdot C \cdot \ln\left(\frac{U_\infty - U_V}{U_0 - U_V}\right) \tag{22}$$

Kde: t_r – doba náběhu a zároveň poklesu
$R = 0{,}66 * R1$
$C = Cs + Cin$
$U_V = 0{,}66 * Uout$
U_∞ - koncové napětí na C (Uin)
U_0 – počáteční napětí na C (Uin)

$$R = -\frac{t_r}{C \cdot \ln\left(\frac{U_\infty - U_V}{U_0 - U_V}\right)} \tag{23}$$

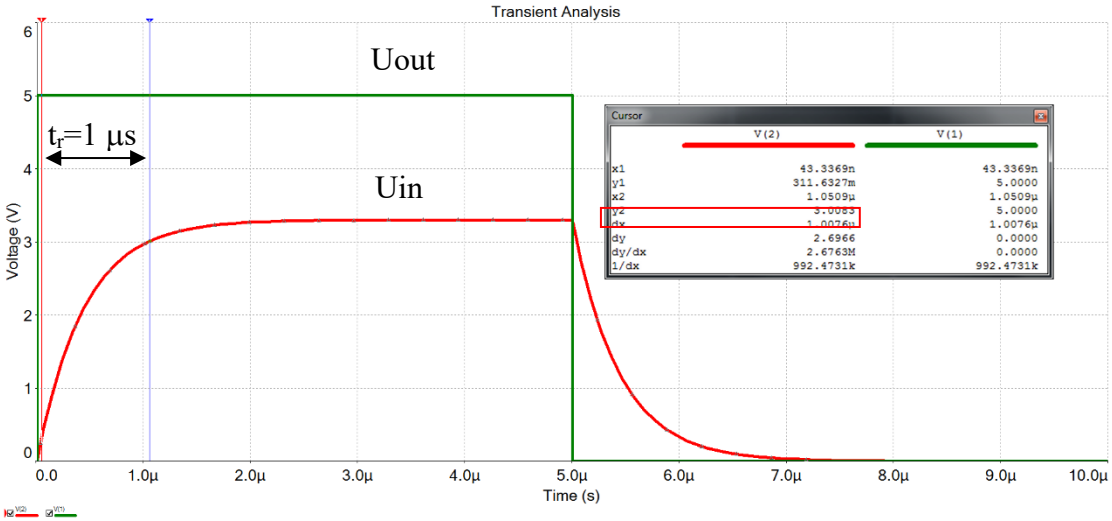
Př.: Cs = 30 pF
Cin = 5 pF
Max. doba náběhu: t_r = 1 µs (10 % a 90 % Vcc → 0,3 V až 3 V)
Uout = 5 V

$$R = -\frac{1 \cdot 10^{-6}}{35 \cdot 10^{-12} \cdot \ln\left(\frac{3 - 0{,}66 \cdot 5}{0{,}3 - 0{,}66 \cdot 5}\right)} = 12408\,\Omega$$

$f_{SWmax} \cong 1/(2*t_r) = 500$ kHz

$$R1 = \frac{R}{0{,}66} = \frac{12408}{0{,}66} = 18800\,\Omega$$

$$R2 = 1{,}941 \cdot R1 = 1{,}941 \cdot 18800 = 36490\,\Omega$$

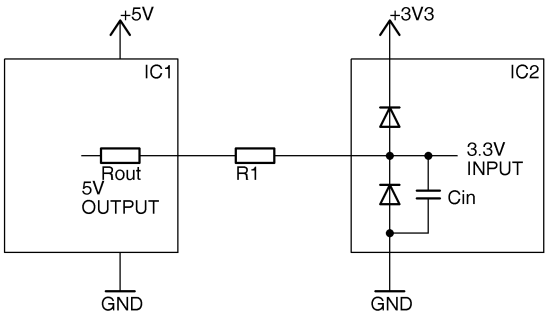


Obrázek 116 Vliv parazitních kapacit na průběh napětí na 3,3 V vstupu

- **Výhody**
  - Jednoduchost.
  - Cena.
- **Nevýhody**
  - Nehodí se pro rychlý přenos dat (odpory děliče omezují rychlost spínání).

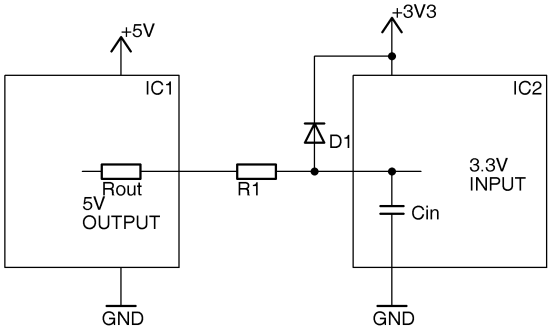
11.2.6 5V → 3,3V s využitím ochranných diod

- Využívá ochranné diody na 3,3 V vstupu – otevře se při úrovni H na 5 V výstupu.



Obrázek 117 5V → 3,3V s využitím interních ochranných diod

- Horní ochranná dioda na 3,3 V vstupu se při překročení napětí 3,3 V + 0,7 V otevře a svede proud z 5 V výstupu do napájecí větve 3,3 V.
- Pokud obvod nemá na vstupu ochranné diody, musí se doplnit externí dioda.

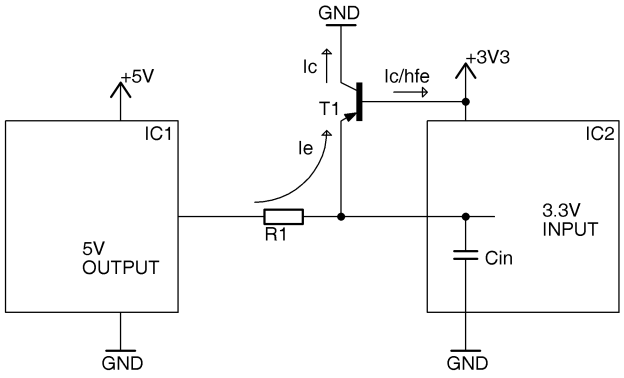


Obrázek 118 5V → 3,3V s využitím externí ochranné diody

- **Vlastnosti**
  - Interní ochranné diody jsou obvykle slabě dimenzovány (proud do 1 mA).
  - R_{out} je typicky 10 Ω → je nutné vložit R1 pro omezení proudu.
  - Přes R1 se nabíjí parazitní kapacita 3,3 V vstupu C_{in} → omezuje rychlost spínání.
  - Proud svedený do napájecí větve 3,3 V může zvýšit napájecí napětí nad bezpečnou mez → je nutné zajistit ochranu napájecí větve.
- **Výhody**
  - Jednoduchost (zejména při využití interních ochranných diod), pouze jeden přidaný rezistor.
  - Cena.
- **Nevýhody**
  - Nehodí se pro rychlý přenos dat (odpor R1 omezuje rychlost spínání).
  - Riziko zvýšení napájecího 3,3 V napětí nad bezpečný limit.

11.2.7 5V → 3,3V s využitím ochranného tranzistoru

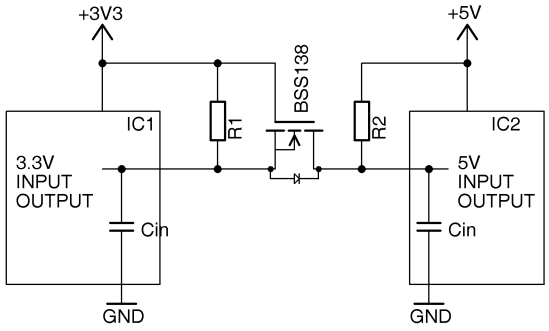
- **Vlastnosti**
  - Při úrovni H na 5 V výstupu se tranzistor otevře, omezí napětí na 3,3 V vstupu na cca 3,3 V + 0,7 V = 4 V, a svede tranzistor proud do GND.
  - Do napájecí větve 3,3 V prochází pouze malý proud, který je dán zesilovacím činitelem tranzistoru ($I_B = \frac{I_C}{h_{FE}} \approx \frac{I_C}{10 \; až \; 400}$). Nehrozí nebezpečné zvýšení napětí napájecí větve 3,3 V.



Obrázek 119 5V → 3,3V s využitím externího ochranného tranzistoru

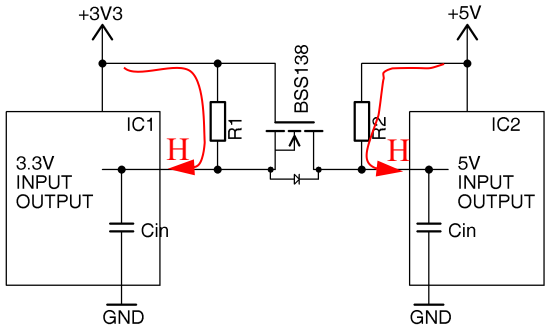
- **Výhody**
  - Nehrozí nebezpečný růst napětí na 3,3 V napájecí větvi.
  - Cena.
- **Nevýhody**
  - Někdy komplikuje motiv DPS.
  - Nehodí se pro rychlý přenos dat (odpor R1 omezuje rychlost spínání), ačkoliv R1 může být nižší než v případě použití interních ochranných diod, protože tranzistor je dimenzován na vyšší proud před interní ochranná dioda. Nízký R1 snižuje časovou konstantu R1*C_{in} a umožňuje rychlejší spínání.

11.2.8 5V → 3,3V a 3,3V → 5V s využitím N-MOSFET tranzistoru

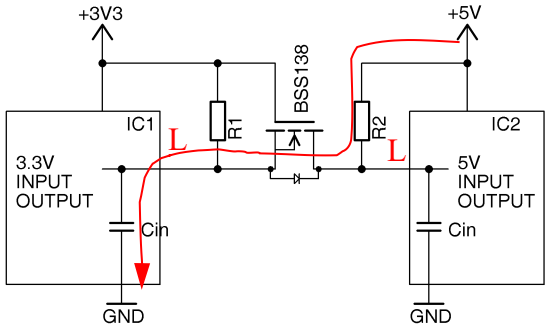


Obrázek 120 Obousměrná konverze úrovní s N-MOSFET tranzistorem

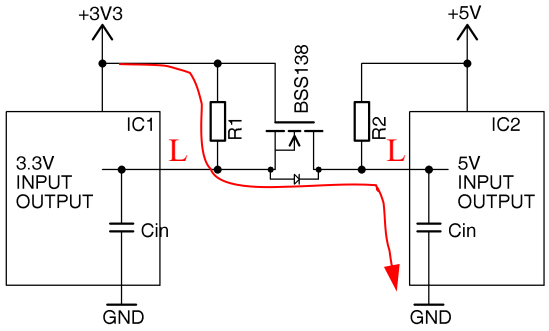
- Vlastnosti
  - Obousměrná konverze úrovní (z vyššího napětí na nižší a obráceně).
  - Lez použít, pokud se směr komunikace na lince mění (např. komunikace po sběrnici I2C).
  - Pokud **ani jedna strana nevysílá úroveň L**, zajišťují pull-up rezistory R1 a R2 úroveň H. Tranzistor je uzavřen, protože napětí U_{GS} je rovno (nebo blízké) 0 V.



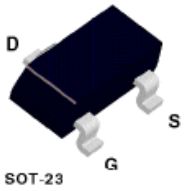
  - Pokud **levá (3,3 V) strana vysílá úroveň L**, dojde k přizemnění S elektrody tranzistoru a k jeho otevření ($U_{GS} \cong 3{,}3$ V). Úroveň L se objeví na pravé (5 V) straně.



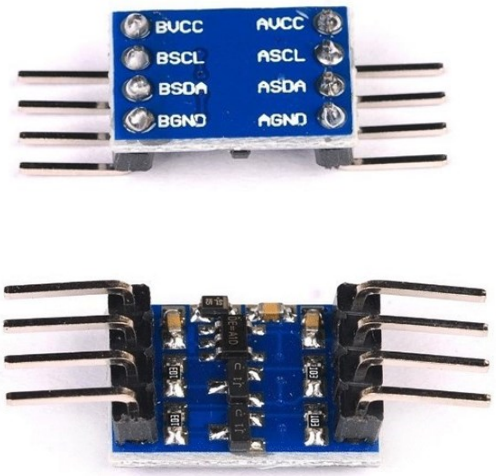
- Pokud **pravá (5 V) strana vysílá úroveň L**, dojde k přizemnění D elektrody tranzistoru. Proud začne protékat z levé strany na pravou přes substrátovou diodu tranzistoru, napětí na S elektrodě tranzistoru klesne, U_{GS} stoupne a tranzistor se otevře. Na levou (3,3 V) stranu se dostane úroveň L.



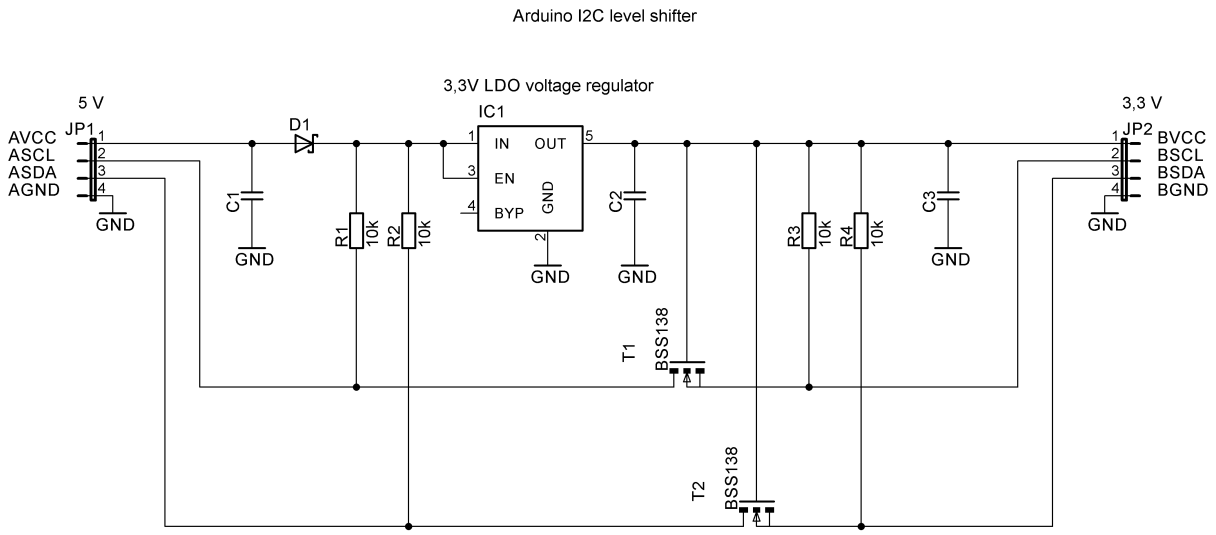
- R1 a R2 omezují rychlost spínání.
  - Stejně jako v jiných zapojeních s pull-up rezistorem i zde se přes R1 a R2 nabíjí parazitní kapacita vstupu. Tím se prodlužuje doba náběhu t_r při přechodu z úrovně L na H.
- Hojně se používají tranzistory BSS138 a BSS123.



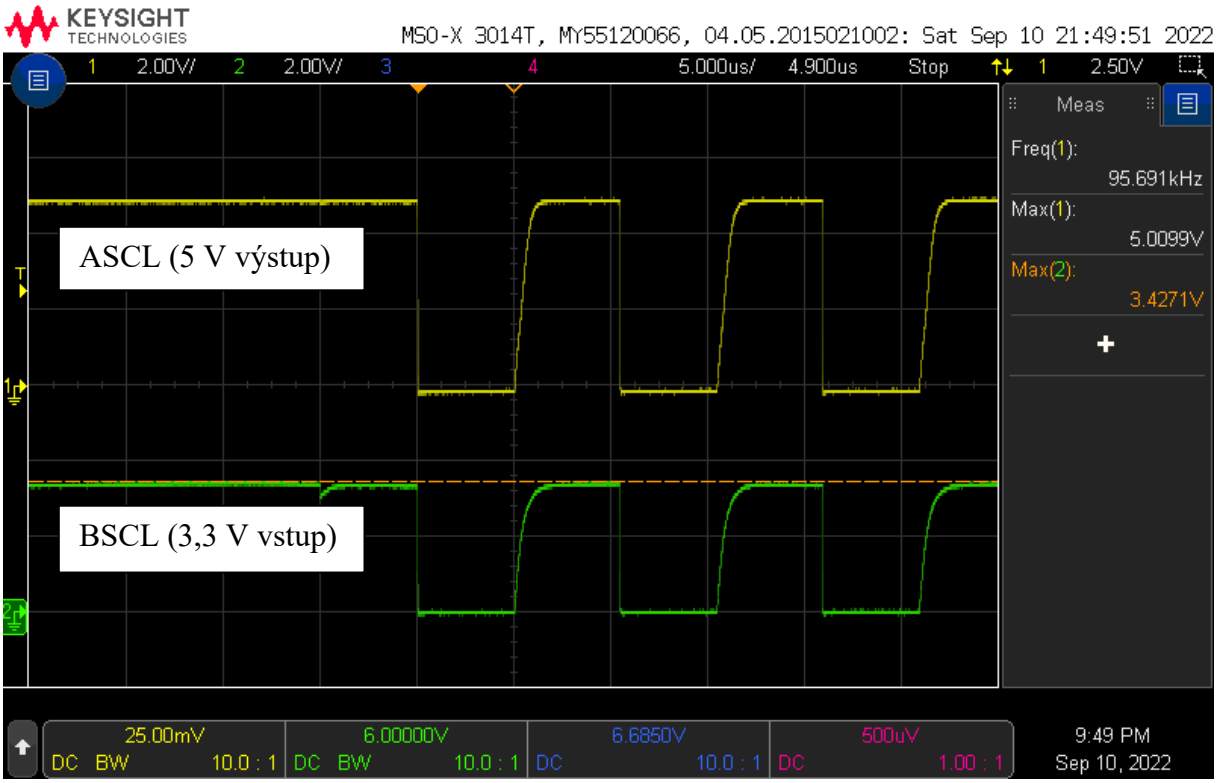
- **Výhody**
  - Obousměrná komunikace.
  - Cena.
- **Nevýhody**
  - Velké množství součástek.
  - Nehodí se pro rychlý přenos dat (pull-up odpory omezují rychlost spínání).



Obrázek 121 Obousměrný konvertor úrovní 5 V ↔ 3,3 V s N-MOSFET tranzistory pro I2C sběrnici



Obrázek 122 El. schéma zapojení obousměrného I2C konvertoru úrovní s N-MOSFET tranzistory

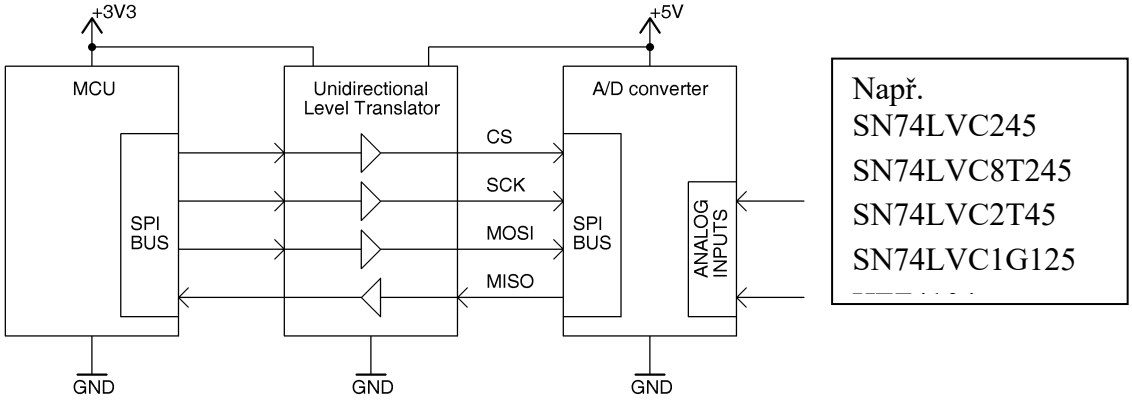


Obrázek 123 Oscilogram signálu SCL (jedno zařízení na sběrnici I2C, pull-up rezistory 10k)

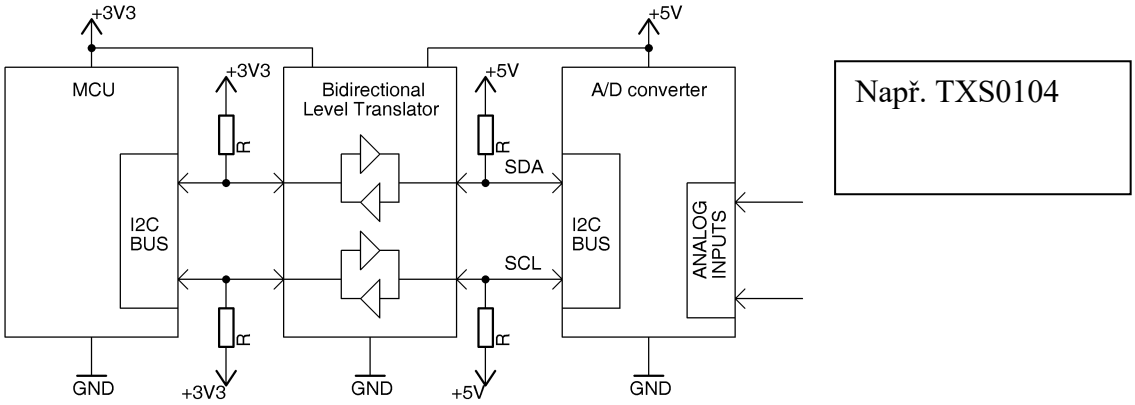
Všimněte si strmých sestupných hran (sepnutý tranzistoru) a pomalých náběžných hran signálu (vypnutí tranzistoru a nabíjení parazitních kapacit přes rezistor).

11.2.9 5V → 3,3V a 3,3V → 5V s využitím konvertoru úrovní

- Nejlepší řešení, zejména pro větší množství vstupů/výstupů.
- Výhody
  - Jednoduchost.
  - Neomezuje rychlost spínání (desítky až stovky Mbps).
  - Možnost obousměrné komunikace.
- Nevýhody
  - Někdy cena.



Obrázek 124 Jednosměrný konvertor úrovní



R - pull-up odpory volitelně, závisí na typu translátoru

Obrázek 125 Obousměrný konvertor úrovní

- Vlastnosti konvertorů úrovní (level translators)
  - Jednosměrné (unidirectional) – pouze jeden směr konverze (jeden směr komunikace)
  - Obousměrné (bidirectional) – obousměrná konverze (z vyššího napětí na nižší a obráceně) dle aktuálního směru komunikace.
  - Různý počet linek (1-bit, 2-bit, 8-bit).

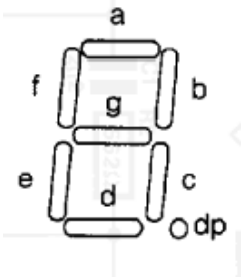
- Různá napájecí napětí (od 1,2 V po 15 V).
- Různé rychlosti.
- Různé typy – např. přímo určené pro dané komunikační rozhraní (SPI, I2C, …).

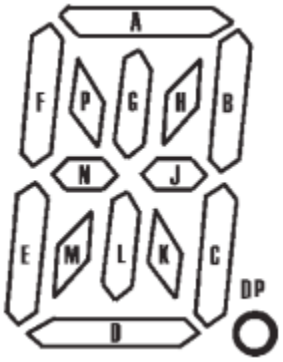
12 Zobrazovací jednotky

- Dělení dle způsobu zobrazení
  - Segmentové
  - Bodové/maticové
- Dle technologie
  - LED – aktivní zdroj světla
  - LCD – pasivní zdroj světla

12.1 LED displeje

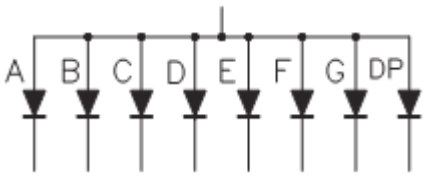
- Např. segmentovky, dva typy:



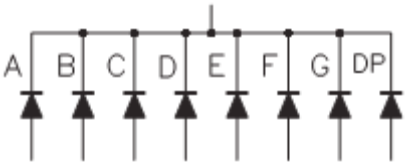


Sedmisegmentový displej Alfanumerický displej

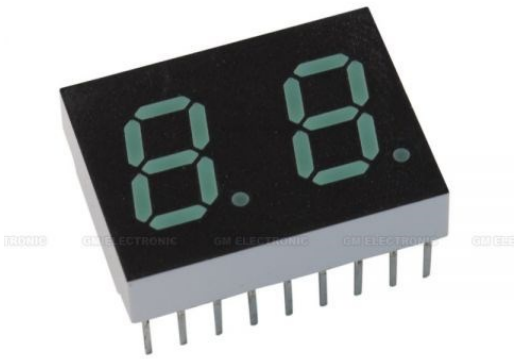
- Zapojení může být buď společná anoda (SA) nebo společná katoda (SC):



SA

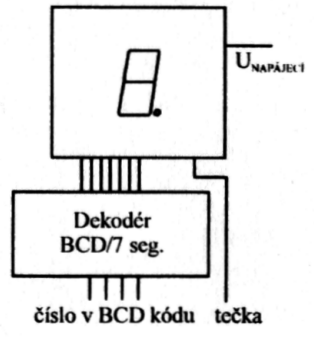


SC

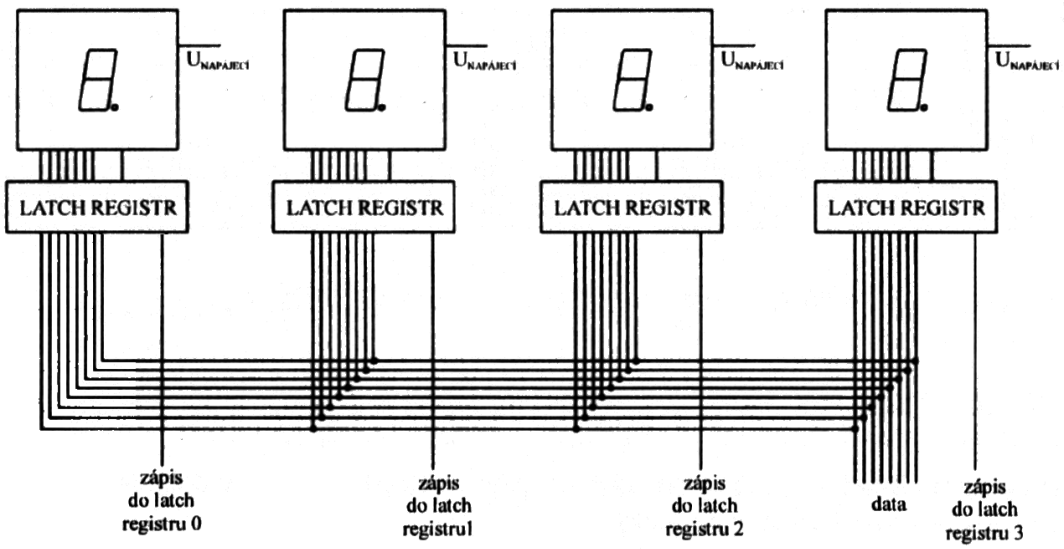


12.1.1 Způsoby zapojení 7-seg. displejů k MCU

- **Přímé připojení**
  - Nevýhodné – velká spotřeba pinů, co segment, to potřeba jeden pin
- **Připojení pomocí dekodéru BCD/7-seg.**
  - Nutné „pouze“ 4 vývody + tečka pro jednu číslici
  - Pro 4 místní displej nutno 4x4 + 4 = 20 vývodů

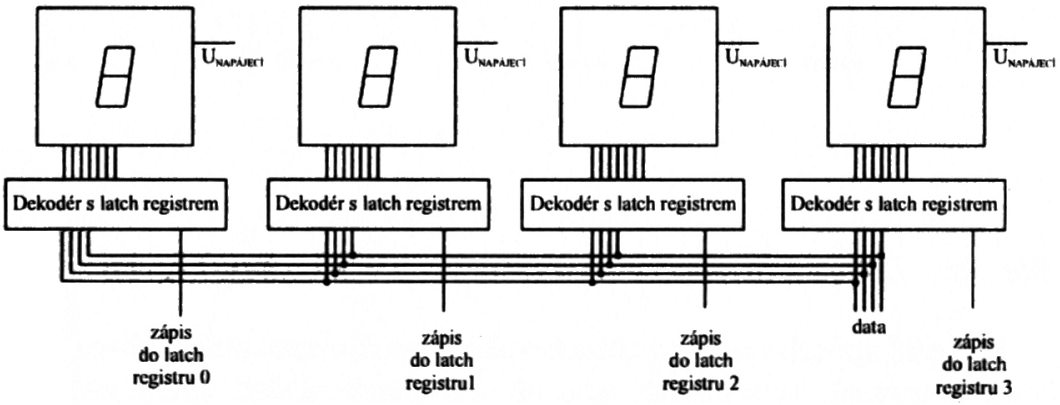


- **Použití záchytného (LATCH) registru – expanze výstupů**
  - Další redukce počtu vývodů, pro 4 displeje potřeba 8 + 4 = 12 vývodů

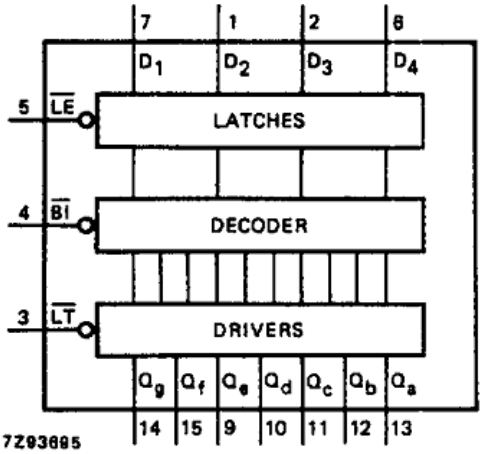


  - Do LATCH registru se uloží stav výstupů pro každý displej, aktivací hodinového vstupu se přepíší vstupy na výstup registru a setrvávají v tomto stavu do dalšího přepsání.
  - Např. 74HCT574

- **Dekodér se záchytným (LATCH) registrem**
  - Další ušetření vývodů MCU → 4 + 4 = 8 vývodů pro 4 displeje



  - Např. 74HC(T)4511 – dekodér BCD/7-seg. a budič displeje
    - Pro displeje se společnou katodou.
    - Proud max. 10 mA (nutno omezit rezistory).
    - Vstup /LE (Latch Enable) – V Log.0 $D_x = Q_x$, v Log. 1 uchová výstupy Q_x neměnné
    - Vstup /BI (Blanking Input) – v Log. 0 na výstupech Q_x Log. 0 (zhasne displej)
    - Vstup /LT (Lamp Test) – v Log. 0 na výstupech Q_x Log. 1 (rozsvítí všechny segmenty)



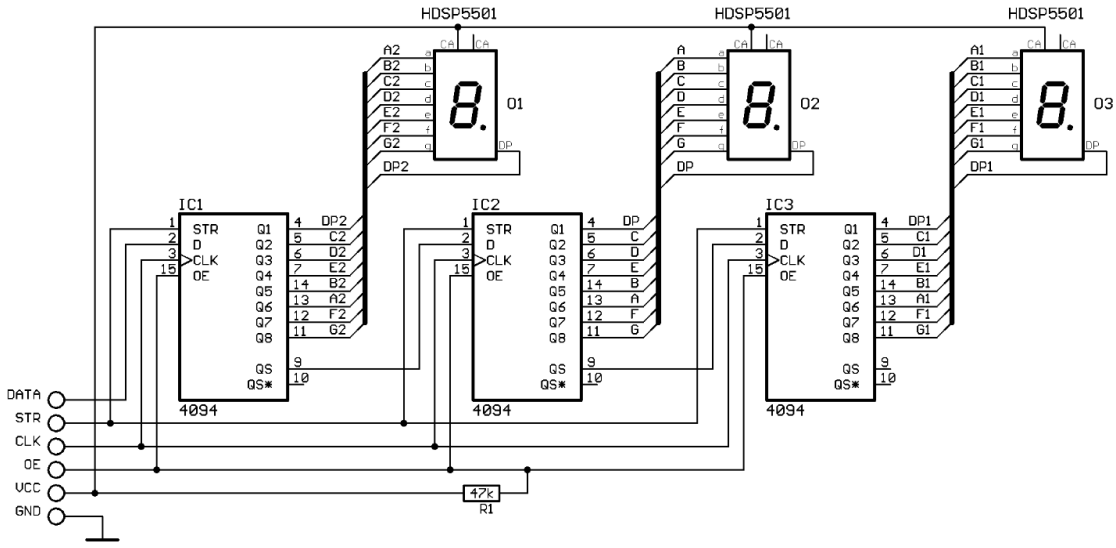
FUNCTION TABLE

INPUTS							OUTPUTS							DISPLAY
$\overline{LE}$	$\overline{BI}$	$\overline{LT}$	D_4	D_3	D_2	D_1	Q_a	Q_b	Q_c	Q_d	Q_e	Q_f	Q_g	
X	X	L	X	X	X	X	H	H	H	H	H	H	H	8
X	L	H	X	X	X	X	L	L	L	L	L	L	L	blank
L	H	H	L	L	L	L	H	H	H	H	H	H	L	0
L	H	H	L	L	L	H	L	H	H	L	L	L	L	1
L	H	H	L	L	H	L	H	H	L	H	H	L	H	2
L	H	H	L	L	H	H	H	H	H	H	L	L	H	3
L	H	H	L	H	L	L	L	H	H	L	L	H	H	4
L	H	H	L	H	L	H	H	L	H	H	L	H	H	5
L	H	H	L	H	H	L	L	L	H	H	H	H	H	6
L	H	H	L	H	H	H	H	H	H	L	L	L	L	7
L	H	H	H	L	L	L	H	H	H	H	H	H	H	8
L	H	H	H	L	L	H	H	H	H	L	L	H	H	9
L	H	H	H	L	H	L	L	L	L	L	L	L	L	blank
L	H	H	H	L	H	H	L	L	L	L	L	L	L	blank
L	H	H	H	H	L	L	L	L	L	L	L	L	L	blank
L	H	H	H	H	L	H	L	L	L	L	L	L	L	blank
L	H	H	H	H	H	L	L	L	L	L	L	L	L	blank
L	H	H	H	H	H	H	L	L	L	L	L	L	L	blank
H	H	H	X	X	X	X				(1)				(1)

Note

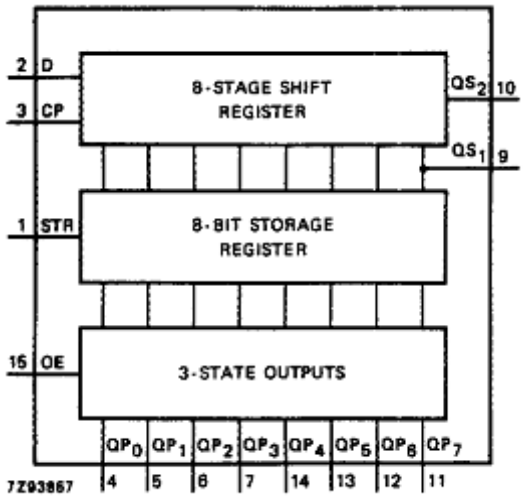
1. Depends upon the BCD-code applied during the LOW-to-HIGH transition of $\overline{LE}$.
   H = HIGH voltage level
   L = LOW voltage level
   X = don't care

- **Displej s posuvnými registry („sériový displej“)**



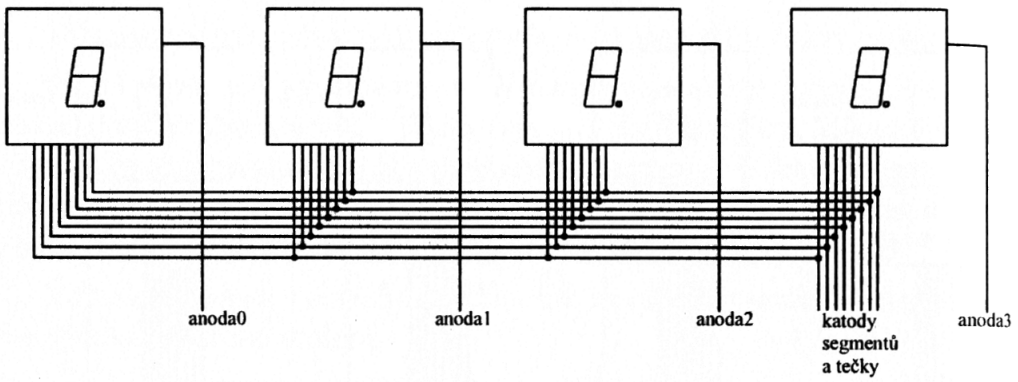
  - Nutné pouze 4 vodiče (DATA, STROBE, CLK, OE) pro jakýkoliv počet displejů.
  - Data se posouvají sériově bit po bitu do jednotlivých posuvných registrů – ty jsou propojeny do kaskády.

- Např. 74HC(T)4094 – obsahuje posuvný registr a střadač

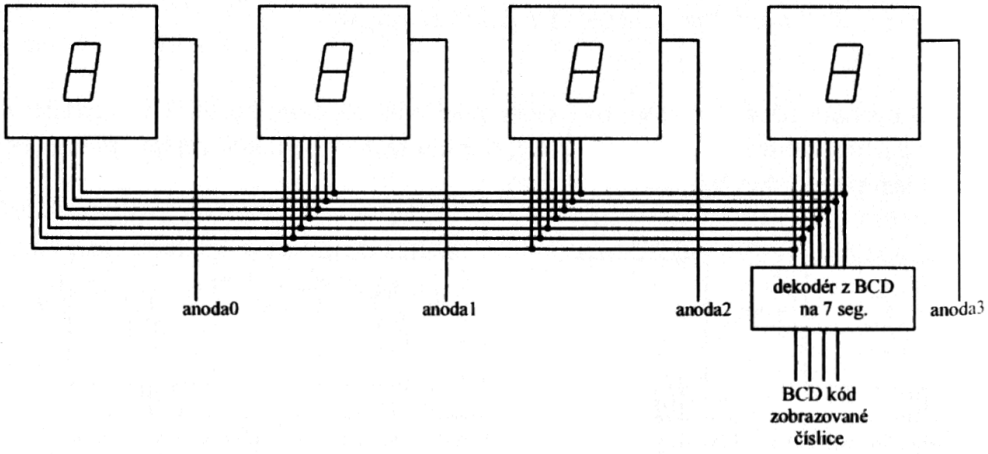


- Princip funkce zapojení (a posuvného registru) je následující:
  - Data jsou v registru posouvána na náběžnou hranu signálu CP (CLK ve schématu). Ze vstupu D jsou přesunuta do bitu 0 a z bitu 7 do bitu 0 následujícího posuvného registru (přes výstup QS), který je zapojen do kaskády.
  - Vstup STR slouží k zachycení dat z posuvného registru do záchytného registru (má stejnou funkci jako /LE u předchozího obvodu 4511) při vzestupné hraně signálu.
  - Vstup OE povoluje zobrazení informace v záchytném registru. Když je vstup OE = Log. 0, jsou výstupy Q_x ve stavu vysoké impedance, je-li OE = Log. 1, objeví se na Q_x obsah registru.
- Ovládání zobrazovače:
  - Základní stav: STR = L, CLK = L, OE = 1.
  - Vysouvám jednotlivé bity na DATA a zároveň generuji CLK.
  - Po vysunutí všech bitů (3 displeje: 3x8 = 24 bitů) STR = H, menší pauza (100 ns), STR = L, CLK = L (není-li již).

- **Multiplexovaný displej**
  - Postupné zapínání/vypínání displejů a obnova jejich zobrazení v časovém multiplexu.
  - Děje se s určitou frekvencí, takovou aby nápis neblikal (alespoň 50 Hz – pozor, platí pro jednu 7-seg. !), lidské oko má setrvačnost, nepostřehne to.
  - Základní zapojení:

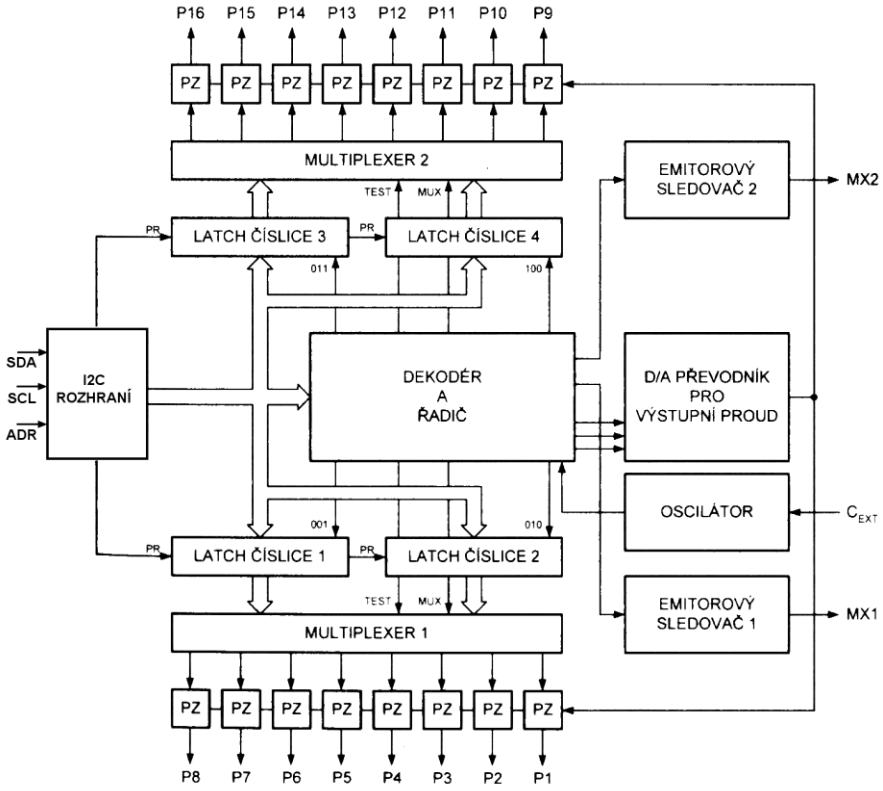


  - Ušetření vývodů pomocí dekodéru:

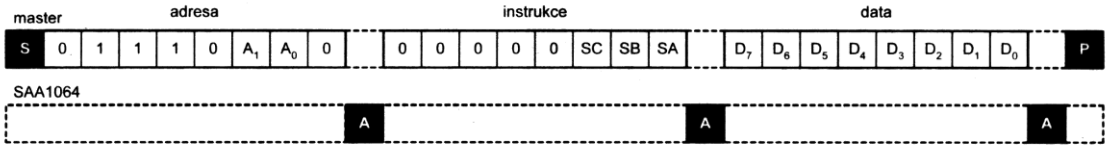


  - **Výhody:** ušetření vývodů, nízká cena
  - **Nevýhody:** Proud displejem musí být větší než při statickém řízení a to N-krát při N-znakovém displeji. Zatěžuje CPU – nutnost periodicky obnovovat displej.

- **Použití speciálních obvodů pro zobrazovače**
  - Komunikace s těmito obvody po paralelní či sériové sběrnici (např. I^2C).
  - MCU posílá kód znaku, který má být zobrazen. Dále instrukce (také nějaký kód) pro nastavení zobrazení (např. pozice na displeji apod.).
  - Výhodou je komfortní obsluha, nevýhodou většinou vyšší cena tohoto I/O.
  - Např. řadiče LCD, různé obvody pro řízení 7-segm. apod.
  - Příklad speciálního I/O – Philips SAA1064
    - Pro buzení 2 nebo 4 7-segm. se společnou anodou.
    - Možnost práce ve statickém (2 displeje) nebo dynamickém režimu (4 displeje).
    - Regulace proudu 0 – 21 mA s krokem 3 mA, nejsou nutné předřadné rezistory pro jednotlivé segmenty.
    - Komunikace po sběrnici I^2C.
  - Blokové schéma:



- Zapisovací sekvence, instrukce a význam bitů řídícího registru:



Obr. 8.10 Zapisovací sekvence obvodu SAA1064

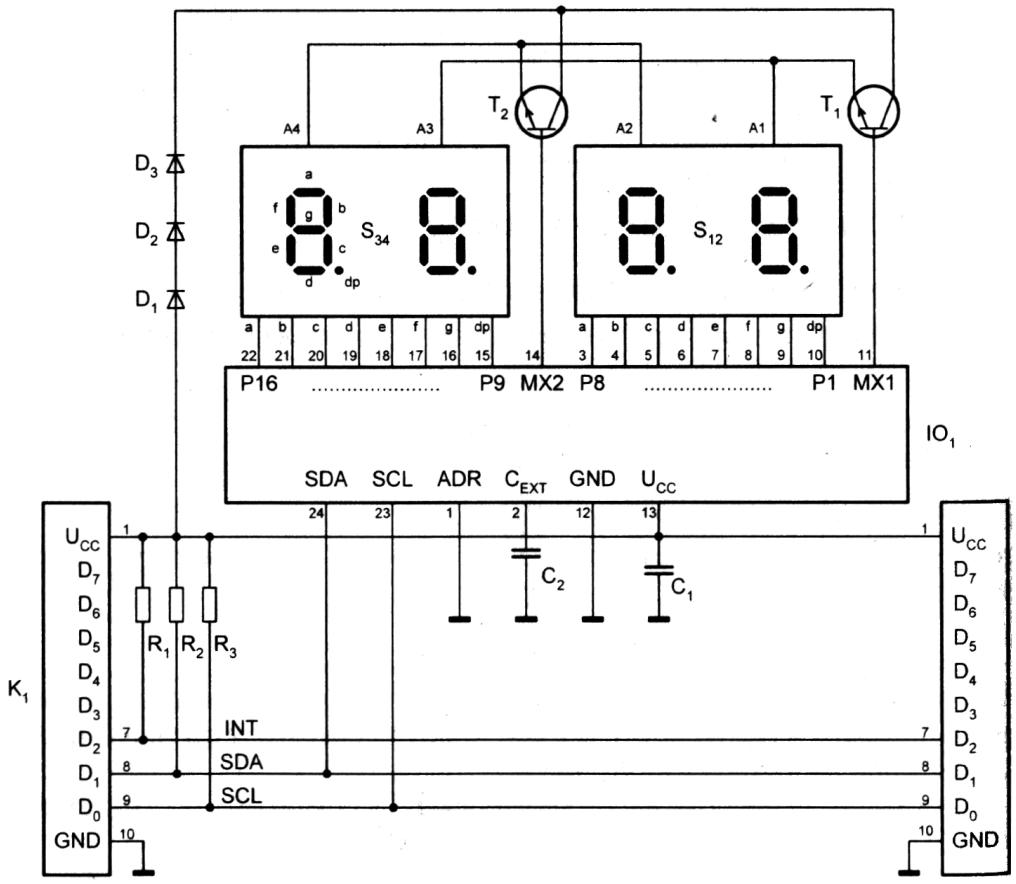
Tab. 8.7 Instrukce SAA1064

SC	SB	SA	Subadresa	Místo určení
0	0	0	0	řídicí registr
0	0	1	1	segmentovka 1
0	1	0	2	segmentovka 2
0	1	1	3	segmentovka 3
1	0	0	4	segmentovka 4
1	0	1	5	vyhrazeno
1	1	0	6	vyhrazeno
1	1	1	7	vyhrazeno

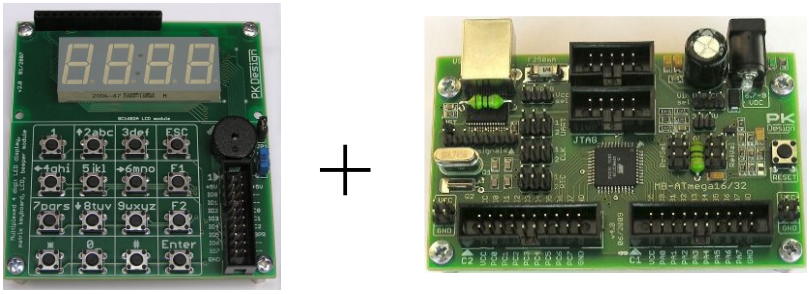
Tab. 8.8 Řídicí registr a význam jeho bitů

Bit	Význam
D_7	vyhrazeno, musí být 0
D_6 D_5 D_4	řídí výstupní proud, viz *tab. 8.9*
D_3	zapíná testovací režim (všechny segmenty aktivovány)
D_2	zapíná segmentovky 2 a 4
D_1	zapíná segmentovky 1 a 3
D_0	volba režimu: 0 – statický režim, aktivní segmentovky 1 a 2, 1 – dynamický režim, aktivní segmentovky 1 + 3 a 2 + 4

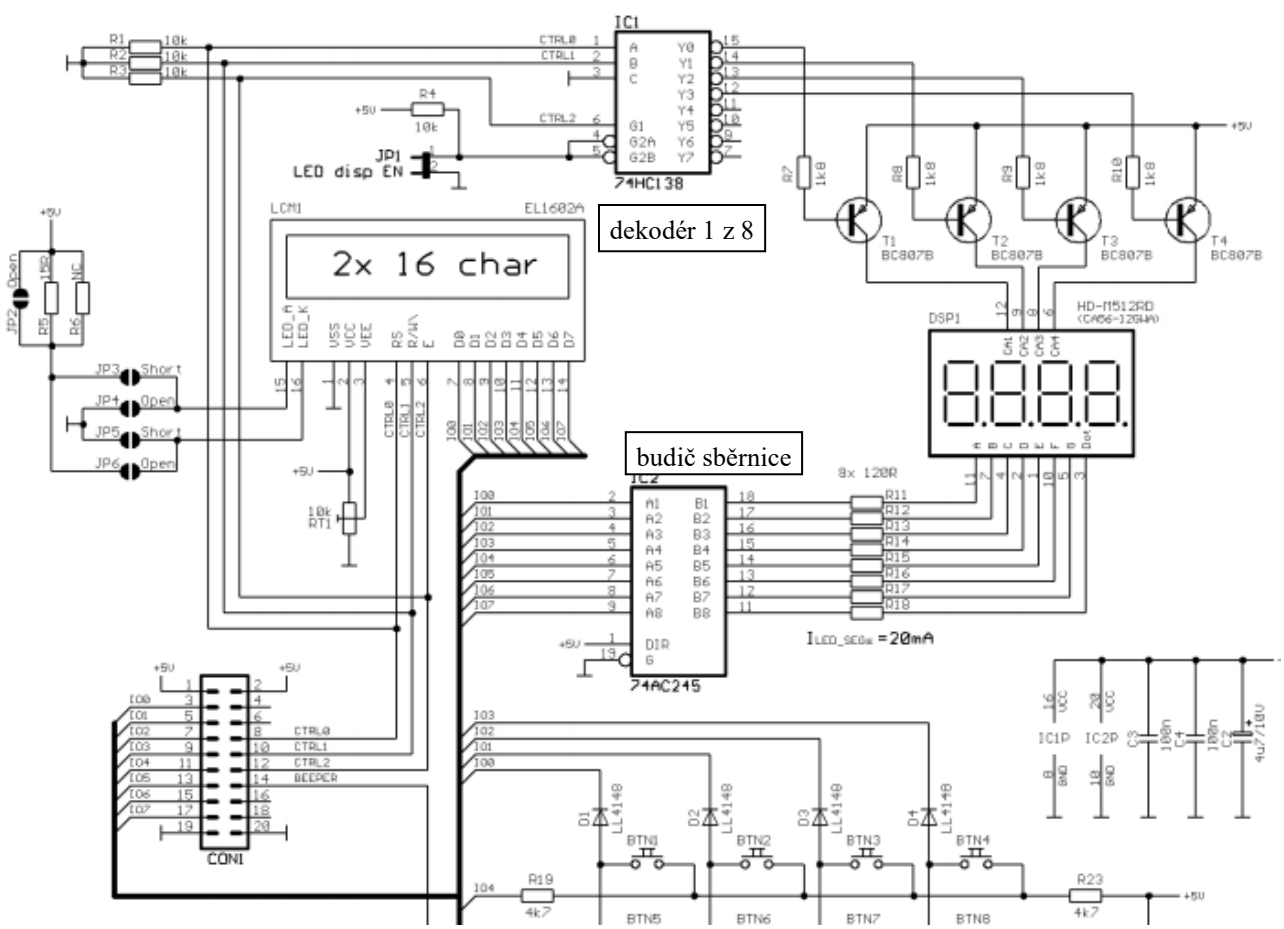
- Schéma zapojení:



12.1.2 Obsluha a připojení multiplexovaného 4-místného 7-segm. LED displeje ke kitu ATmega32 od PK Design



12.1.2.1 Zapojení multiplexovaného displeje



Obrázek 126 El. schéma zapojení multiplexovaného displeje na školním kitu

- LED displej se společnou katodou.
- Je použit neinvertující budič 74HC245 pro odlehčení zátěže portů mikrokontroléru.
- Segment svítí při log. 0 na pinu příslušného portu mikrokontroléru.
- V závislosti na připojení k základové desce ATmega32 bude LED displej dostupný na bráně PORTA nebo PORTC mikrokontroléru (datové vodiče).
- Napájení jednotlivých sedmisegmentovek displeje je ovládáno PNP tranzistory.

- Pro ušetření vývodů jsou tyto tranzistory spínány pomocí dekodéru 1 z 8 (74HC138).
  - Pravdivostní tabulka dekodéru 1 z 8 (74HC138)

FUNCTION TABLE

INPUTS							OUTPUTS							
$\overline{G2A}$	$\overline{G2B}$	G1	A	A_1	B	A_2	C	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$
H	X	X	X	X	X	X	H	H	H	H	H	H	H	H	H
X	H	X	X	X	X	X	H	H	H	H	H	H	H	H	H
X	X	L	X	X	X	X	H	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	L	H	H	H	H	H	H	H	H
L	L	H	H	L	L	L	H	L	H	H	H	H	H	H	H
L	L	H	L	H	L	L	H	H	L	H	H	H	H	H	H
L	L	H	H	H	L	L	H	H	H	L	H	H	H	H	H
L	L	H	L	L	H	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	L	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	L	H	H	H	H	H	H	H	H	L

Notes

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care

Červeně jsou označeny kombinace, které přicházejí v našem zapojení v úvahu (máme pouze 4 místný displej).

  - Vývody CTRL0 (odpovídá vstupu A na 74HC138) a CTRL1 (odpovídá vstupu B na 74HC138) určují aktivní sedmisegmentovku.
  - Vývod CTRL2 (odpovídá vstupu G1 na 74HC138) zapíná/vypíná celý displej.
    LOW – zhasne celý displej
    HIGH – pouze vybraná sedmisegmentovka má napájení, podle kombinace vystavené na CTRL0 a CTRL1
  - Ovládací piny CTLR0 až CTLR2 budou v závislosti na připojení k základové desce ATmega32 dostupné na bráně PORTB nebo PORTD.

Tabulka 15 Zjednodušená pravdivostní tabulka platná pro Obrázek 126

CTRL0 (A)	CTRL1 (B)	CTRL2 (C)	Y0	Y1	Y2	Y3	Poznámka
0	0	1	0	1	1	1	svítí 1. sedmisegmentovka zleva
1	0	1	1	0	1	1	svítí 2. sedmisegmentovka zleva
0	1	1	1	1	0	1	svítí 3. sedmisegmentovka zleva
1	1	1	1	1	1	0	svítí 4. sedmisegmentovka zleva
X	X	0	1	1	1	1	všechny sedmisegmentovky zhasnuté

Vstupy C = 0, G2A = G2B = 0 na 74HC138.

12.1.2.2 Princip obsluhy multiplexovaného LED displeje

- Vyhrazený časovač pro periodickou obnovu zobrazení na displeji.
- Kód pro obnovu zobrazení na displeji je umístěn ve funkci obsluhy přerušení při přetečení časovače.
- Časovač musí přetékat s periodou minimálně 4x50 Hz = 200 Hz (displej je 4 místný), aby docházelo k obnově zobrazení na každé sedmisegmentovce s frekvencí 50 Hz (hranice pro lidské oko).

12.1.2.3 Činnost kódu v obsluze přerušení od přetečení časovače

1. Zhasnout displej (CTRL2 = 0).
2. Vystavit na port novou hodnotu (odpovídající kombinaci 0 a 1 pro zobrazení chtěného čísla).
    - Správnou kombinaci 0 a 1 pro zobrazení chtěného čísla lze vyčíst z překódovací tabulky.
3. Aktivovat následující 7-mi segmentovku (CTRL0 = 0 nebo 1 , CTRL1 = 0 nebo 1).
    - Nutno si pamatovat pořadí aktuálně rozsvícené 7-mi segmentovky.
4. Rozsvítit displej (CTRL2 = 1).

12.1.2.4 Překódovácí tabulka

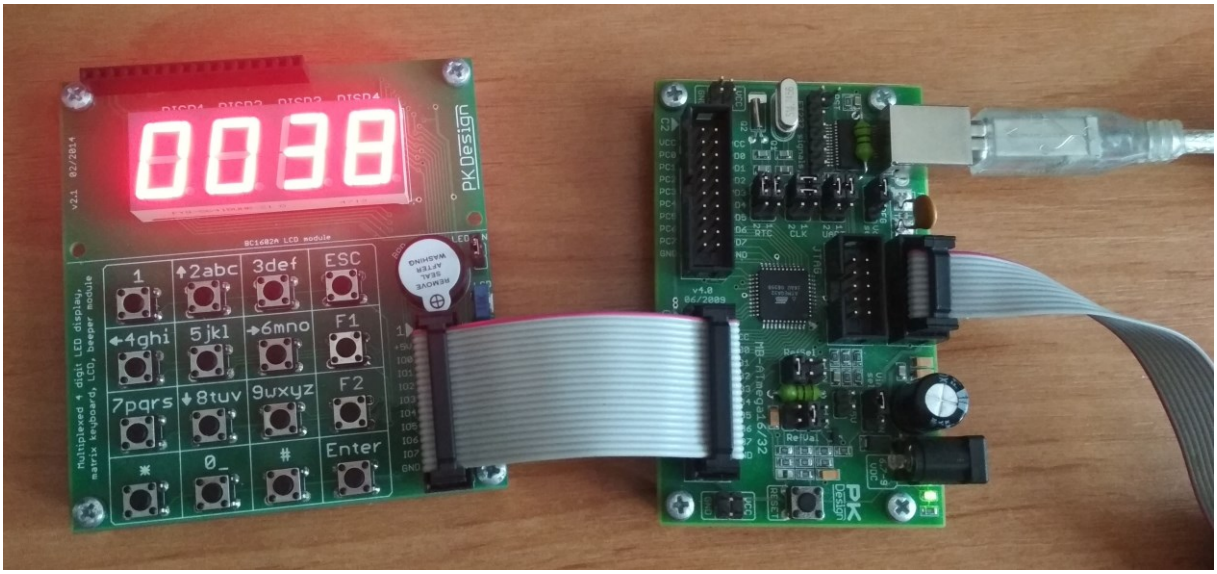
- Řeší převod mezi zobrazovaným číslem (0 až 9) a správnou binární kombinací 0 a 1 pro rozsvícení čísla na 7-mi segmentovce.
- Pro zapojení LED displeje na modulu maticové klávesnice bude překódovací tabulka vypadat takto:

Hodnota na portu	Zobrazené číslo na 7-segm.
0b11000000	0
0b11111001	1
0b10100100	2
0b10110000	3
0b10011001	4
0b10010010	5
0b10000010	6
0b11111000	7
0b10000000	8
0b10010000	9

- Segment je rozsvícen při úrovni logická 0 na daném pinu portu (je použit LED displej se společnou katodou a neinvertující budič 74HC245)

12.1.2.5 Ukázka obsluhy multiplexovaného 4-místného LED displeje

Základová deska s ATmega32 + modul LED a LCD displeje s maticovou klávesnicí od PK Design.



Obrázek 127 Multiplexovaný 4-místný LED displej v kitu od PK Design

```
/*
	Multiplex4x7segmDisplej
	Ukazka obsluhy 4-mistneho 7-segm. displeje od PK design.
*/

#define F_CPU 14745600 // [Hz]
#include <avr/io.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#include "bcd.h"	// pro prevod z int na BCD

//definice pripojeni displeje k MCU
#define LED_PORT		PORTA
#define LED_DDR			DDRA
#define LED_PIN			PINA
#define LED_CTRL_PORT	PORTB
#define LED_CTRL_DDR	DDRB
#define LED_CTRL_PIN	PINB
#define LED_CTRL0		2		//cislo pinu na portu
#define LED_CTRL1		3
#define LED_CTRL2		4

// prevodni tabulka BCD->7 seg., ulozena v pameti programu
const unsigned char tab_seg[10] PROGMEM =
{0b11000000,0b11111001,0b10100100,0b10110000,0b10011001,0b10010010,0b10000010,0b11111000,0b10000000,0b10010000};

// 4-mistne cislo, které se ma zobrazit na displeji (v BCD)
unsigned short int cislo_bcd = 0x0000; // musi byt globalni promenna, protoze se pouziva v ISR
```

```
// Timer 0 overflow interrupt service routine
ISR(TIMER0_OVF_vect)
{
	static unsigned char pozice; // pamatuje si posledni aktivni sedmisegmentovku
	unsigned char cislice; // hodnota aktualni cislice

	// prepnuti sedmisegmentovky
	if (++pozice == 4) pozice = 0; // 0 az 3
	LED_DDR = 0xFF;		// cely port vystupni (ovladani 7-segm.)
	LED_CTRL_DDR |= (1<<LED_CTRL0) | (1<<LED_CTRL1) | (1<<LED_CTRL2); // pouze piny
CTLR0,1 a 2 vystupni (ovladani dekoderu 1 z 8 74HC138)

	LED_CTRL_PORT &= ~(1<<LED_CTRL2); // vypne cely displej kvuli prosvecovani
	LED_CTRL_PORT = (pozice<<2);		// vyber dalsi sedmisegmentovky v poradi

	// zapis cislice na vybranou segmisegmentovku
	// v promenne cislo_bcd je ulozeno ctyrmistne cislo, ktere se ma zobrazit na
displeji, cislo je ve formatu BCD !!
	switch (pozice)
	{
		case 0:		cislice = (cislo_bcd >> 12) & 0x000F;	// 1. cislice
		break;
		case 1:		cislice = (cislo_bcd >> 8) & 0x000F;	// 2. cislice
		break;
		case 2:		cislice = (cislo_bcd >> 4) & 0x000F;	// 3. cislice
		break;
		case 3:		cislice = (cislo_bcd >> 0) & 0x000F;	// 4. cislice
		break;
		default: cislice = 0; // nemelo by nastat
		break;
	}
	LED_PORT = pgm_read_byte(&tab_seg[cislice]);	// prevod BCD -> 7-segm. a
vystaveni cislice na port
	LED_CTRL_PORT |= 1<<LED_CTRL2; // zapne displej, podle kombinace vystavene na
vyvody CTRL0 a CTRL1 sviti jen jedna sedmisegmentovka
}

int main(void)
{
	// 16-bit neznamenkove cislo
	unsigned short int cislo = 0;

	cislo_bcd = short2bcd(cislo);		// prevod na BCD 4-mistne cislo (napr. 0x9999)

	// nastaveni 8-bit casovace Timer/Counter 0 pro obsluhu multiplexovaneho 4-
mistneho LED displeje
	/*
		mode NORMAL
		delicka N = 256
		Ftimer = Fclk/N Hz
		Fclk = 14745600 Hz
		frekvence preteceni = Ftimer/2^8 = 14745600/256/256 = 225 Hz
		perioda preteceni = 2^8/Ftimer = 256*256/14745600 = 4.444 ms

		Obnovovaci frekvence celeho 4-mistneho displeje je 225 Hz.
		Obnovovaci frkevence jedne sedmisegmentovky je 225/4 = 56.25 Hz,
		coz je dostacujici, aby displej rusive neblikal.
	*/
	TCNT0 = 0;	// radeji vynulovani citaciho registru
	TCCR0 = (1<<CS02);	// nastaveni rezimu NORMAL a delicky N = 256 -> spusteni
casovace
	//TCCR0 = (1<<CS02)|(1<<CS00);	// N = 1024 -> DISPLEJ UZ BUDE RUSIVE BLIKAK
```

```
	TIMSK = (1<<TOIE0); // povoleni preruseni pri preteceni casovace Timer/Couner 0

	// globalni povoleni preruseni - pro obsluhu preruseni od casovace (pro obsluhu
LED displeje)
	sei();

	// stale dokola inkrementujeme cislo od 0 do 9999 a zobrazujeme ho na displeji
	while(1)
	{

		// prevod na BCD 4-mistne cislo (packed BCD format)
		// napr. cislo = 9999 (0x270F) -> cislo_bcd = 39321 (0x9999)
		// funkce short2bcd se nachazi v souboru bcd.h
		cislo_bcd = short2bcd(cislo);

		// pauza 1000 ms
		_delay_ms(1000);

		// inkrementace cisla zobrazovaneho na displeji
		if (cislo < 9999) cislo++;
		else cislo = 0;
	}
}
```

Tabulka 16 Hodnoty ovládacích signálů CTRLx podle obsahu proměnné `pozice` po vykonání řádku s kódem `LED_CTRL_PORT = (pozice<<2);`

pozice	CTRL0	CTRL1	CTRL2	Poznámka
0	0	0	0	Vybírá 1. sedmisegmentovku zleva. Displej zhasnut.
1	1	0	0	Vybírá 2. sedmisegmentovku zleva. Displej zhasnut.
2	0	1	0	Vybírá 3. sedmisegmentovku zleva. Displej zhasnut.
3	1	1	0	Vybírá 4. sedmisegmentovku zleva. Displej zhasnut.

Stručné vysvětlení způsobu zobrazení čísla na LED displej.

V těle příkazu switch se na aktuálně aktivní sedmisegmentovku zapíše to správné číslo. Celé číslo, které se má zobrazit na 4-místném displeji je uloženo v proměnné `cislo_bcd`. Proč v BCD? Protože jednotlivé nibbly (skupiny 4 bitů), které odpovídají jednotlivým číslům na 7-mi segmentovkách, jsou pak přímo indexy do překódovaní tabulky (indexy 0 až 9). V těle příkazu `switch` se pak pouze vybere ten správný nibble pro danou 7-mi segmentovku a pomocí kódovací tabulky se vybere správná binární kombinace, která se vystaví na datové linky displeje.

Např. `cislo_bcd = 0x1234` (číslo 1234 ve formátu BCD) – binárně 4660

	`cislo_bcd`	
byte	HIGH BYTE	LOW BYTE

bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BCD	1				2				3				4			

Převod binárního čísla na BCD číslo

Pokud máme v našem programu číslo, které chceme zobrazit na LED displeji, a které je uložené do běžné 16-bit neznaménkové int proměnné (`unsigned short int`), musíme ho před jeho vypsáním na displej převést do packed BCD formátu.

To můžeme nejjednodušeji provést pomocí funkcí, které je buď součástí toolchainu nebo pomocí knihovny, kterou vytvořil někdo jiný.

Knihovna (soubory `bcd.h bcd.c`) pro BCD konverzi, která umí (obsah souboru `bcd.h`):

```
/*
Converts 16-bit unsigned integer to 4 digit BCD number.
input:        val - number to convert (0..9999)
return:       BCD number (4 digits long)
*/
extern unsigned short int short2bcd(unsigned short int val);

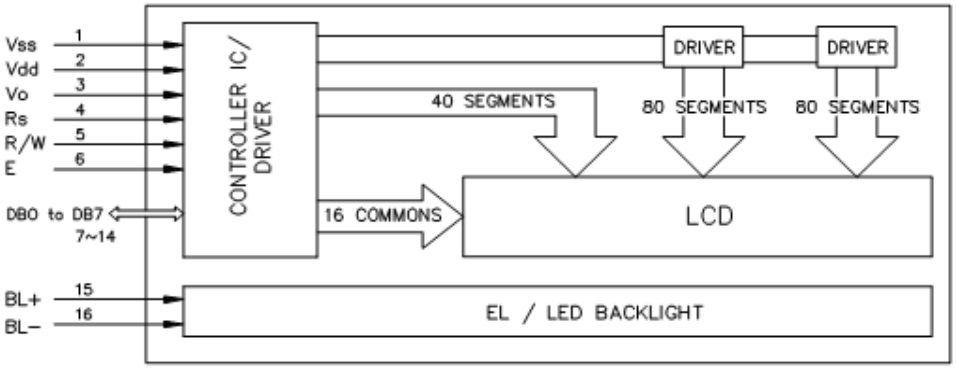
/*
Converts 8-bit binary number to 2 digit BCD number.
input:        val - number to convert (0..99)
return:       BCD number (2 digits long)
*/
extern unsigned char char2bcd(unsigned char val);


/************************ BCD to BIN ************************/
/*
Converts 4 digit BCD number to 16-bit binary number.
input:        BCD number (4 digits long)
return:       binary number
*/
extern unsigned short int bcd2short(unsigned short int bcd);

/*
Converts 2 digit BCD number to 8-bit binary number.
input:        BCD number (2 digits long)
return:       binary number
*/
extern unsigned char bcd2char(unsigned char bcd);
```

12.2 Znakové LCD

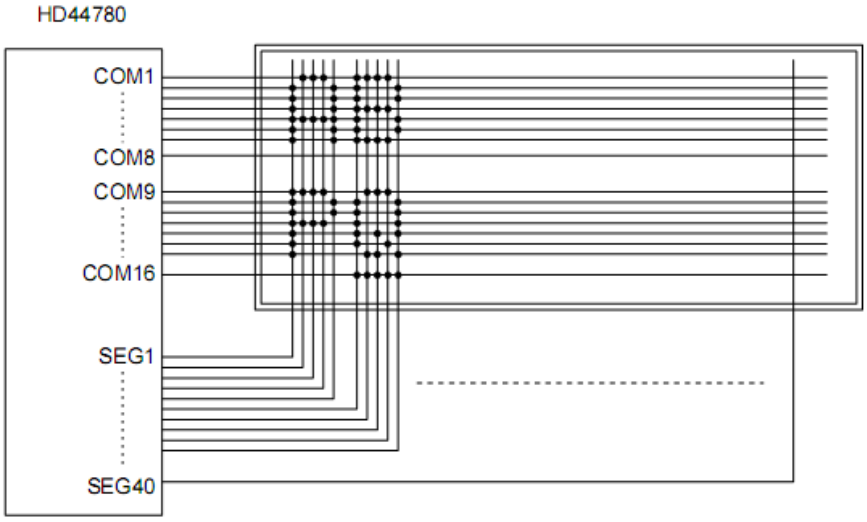
- Jsou řízeny speciálním řadičem, většinou řadič Sharp HD44780 kompatibilní.
- Varianty s různým počtem znaků na řádek a různý počet řádků, od 1x8 až 4x40 znaků.
- S podsvětlením nebo bez podsvětlení.
- Pro připojení k MCU je potřeba 4 nebo 8 datových vodičů a 2 nebo 3 řídící vodiče.
- **Obvyklé zapojení displeje:**



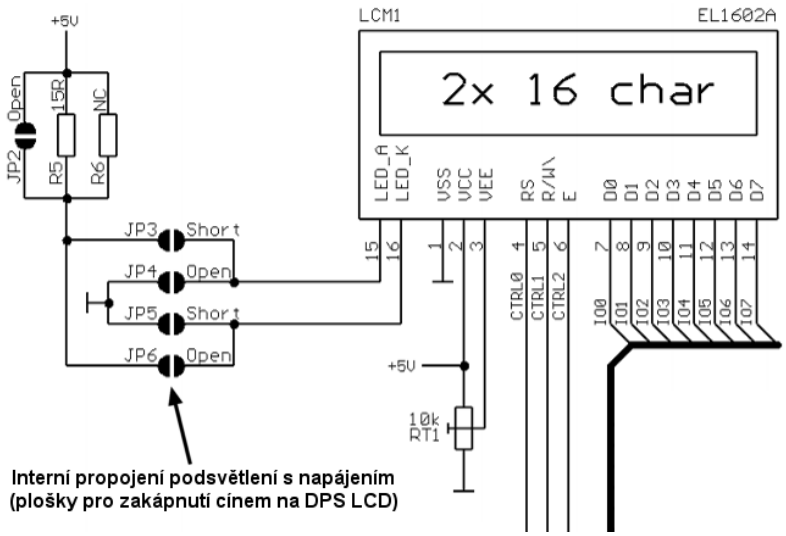


Vývod č.	Název	Popis
1	Vss	GND
2	Vdd	+5V
3	Vo	Kontrast
4	Rs	Instrukce (0) / Data (1)
5	R/W	Zápis (0) / Čtení (1)
6	E	Takt. signál (hodiny)
7 - 14	DB0 – DB7	Data/Instrukce 8-bit
15	BL+	Podsvětlení anoda
16	BL-	Podsvětlení katoda

- **Interní maticové uspořádání** (LCD o vel. 2 řádky x 8 znaků, velikost znaku 5x8 bodů)



- **Zapojení LCD v aplikaci**



- **Znaková sada**
  - Znak má velikost 5x8 bodů.
  - Tabulka znaků obsahuje základní anglickou sadu a také některé další „exotické“ znaky.
  - Kód znaku je ASCII.

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	q	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	ɾ	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		−	=	M	]	m	}			ュ	ス	ヘ	ン	₺	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	゜	ö	

- **Znaky určené k zobrazení jsou ukládány do DDRAM** (Display Data RAM).
  - Velikost DDRAM je 80 x 8-bit nebo-li 80 znaků.
  - Každé pozici znaku na displeji odpovídá určitá adresa v paměti.
  - Adresa je 7-bitová.
  - Příklad mapy adres v DDRAM pro LCD 2x16 (2 řádky, 16 znaků na řádek):

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

  - Na adrese se nachází kód znaku z tabulky kódů.
  - U 3 řádkového LCD začíná 3. řádek na adrese 0x14, u 4 řádkového LCD začíná 4. řádek na adrese 0x54.
  - Po vypnutí napájení ztratí obsah (při dalším zapnutí nebude na LCD svítit text, který na něm svítil před tím).
- **Lze definovat i vlastní znaky** (max. 8). Ty jsou uloženy v paměti **CGRAM** (Character Generator RAM).
  - Až 8 vlastních znaků.
  - Znaky jsou dostupné pod kódy 0 – 7 a opakovaně pod kódy 8 – 15, viz předchozí tabulka znaků.
  - Adresa do CGRAM je 6-ti bitová. Její horní 3 bity adresují uživatelem definovaný znak (kód uživatelského znaku je tedy vždy 0000xkkk), dolní 3 bity adresují řádek matice znaku. Postup tvorby uživatelského znaku je zobrazen na následujícím obrázku.

Kód znaku	Adresa CGRAM						Data							
	Kód zn.			Řádek zn.										
	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0 nebo 8	0	0	0	0	0	0	x	x	x	0	1	0	1	0
				0	0	1				0	0	1	0	0
				0	1	0				0	1	1	1	0
				0	1	1				1	0	0	0	1
				1	0	0				1	1	1	1	1
				1	0	1				1	0	0	0	0
				1	1	0				0	1	1	1	0
				1	1	1				0	0	0	0	0

  - Spodních 5 bitů dat definuje podobu bitu na daném řádku. Je-li bit v log. 1, je bod černý. Je-li v log. 0, je průhledný. Na horních 3 bitech dat nezáleží. Poslední řádek je vyhrazen pro kurzor.

- **Instrukce**
  - Princip ovládání displeje spočívá v zasílání instrukcí a dat řadiči.
  - Lze mazat displej, nastavovat polohu kurzoru, zapínat/vypínat kurzor, zapisovat nebo číst data do/z DDRAM a CGRAM a další.
  - Po odeslání příkazu je nutno vždy vyčkat určitý čas, který řadič LCD potřebuje na zpracování – viz tabulka.

Tabulka 17 Instrukce pro ovládání alfanumerického LCD

Instrukce	RS	R/-W	DATA								Čas prov.
			7	6	5	4	3	2	1	0	
Clear display	0	0	0	0	0	0	0	0	0	1	1,64 ms
Return home	0	0	0	0	0	0	0	0	1	X	1,64 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	40 µs
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	40 µs
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	X	X	40 µs
Function set	0	0	0	0	1	DL	N	F	X	X	40 µs
Set CGRAM address	0	0	0	1	MSB ADR CG LSB						40 µs
Set DDRAM address	0	0	1	MSB ADR DD LSB							40 µs
Ready BUSY flag and addr.	0	1	BF	MSB ADR C LSB							40 µs
Write data to CG/DD RAM	1	0	MSB DATA LSB								40 µs
Čti data z CG/DD	1	1	MSB DATA LSB								40 µs

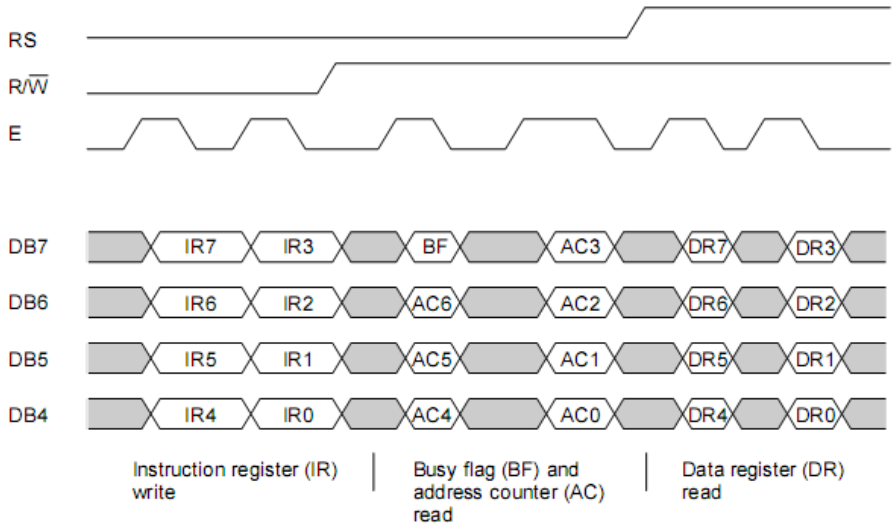
Význam jednotlivých bitů:

I/D	Směr posuvu kurzoru (0 – vlevo, 1 – vpravo) po zápisu do / čtení z DDRAM
S	Posuv textu (0 – zakázán, 1 – povolen)
D	1 – displej zapnut, 0 – displej vypnut
C	0 – kurzor vypnut, 1 – kurzor zapnut
B	0 – blikání kurzoru vyp., 1 – blikání zap.
S/C	0 – posun kurzoru , 1 – posun textu
R/L	0 – vlevo, 1 - vpravo
DL	0 – 4-bitová komunikace, 1 – 8-bitová komunikace
N	0 – jednořádkový displej, 1 – dvouřádkový displej
F	Výběr fontu, 0 – 5x8 bodů, 1 – 5x10 bodů
ADR CG	Adresa do CGRAM (6 bitů)
ADR DD	Adresa do DDRAM (7 bitů)
BF	Busy Flag – poskytuje informaci o aktivitě řadiče, 0 – příjem povolen, 1 – řadič zaneprázdněn
ADR C	Adresa ukazatele na místo kam ukazuje kurzor

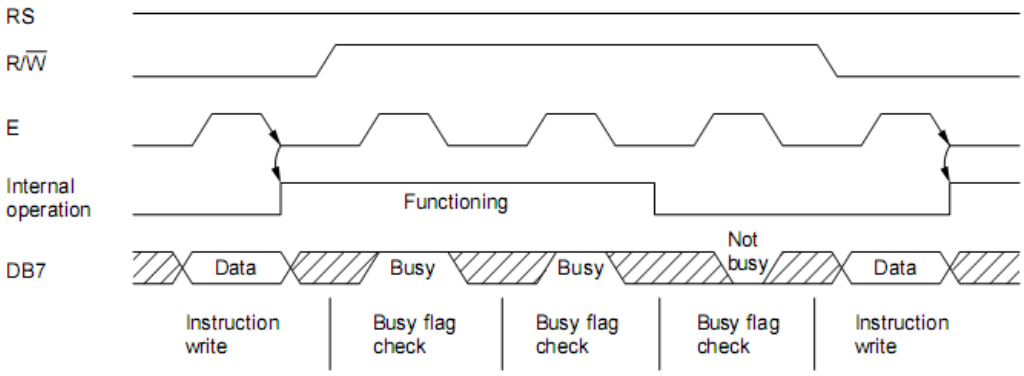
- **Komunikace s LCD displejem**
  - Probíhá po datových vodičích (4 nebo 8) a je potřeba obsluhovat vodič RS – přepínání zápisu instrukcí nebo dat a vodič E – taktovací signál. Vodič R/W nemusí být použit, nevyžadujeme-li čtení dat z řadiče, v tomto případě bude trvale uzemněn.
  - Datová komunikace může být 8-bitová nebo 4-bitová (ušetření vývodů MCU).
  - Dostupnost LCD (zda je zaneprázdněno) se testuje pomocí příznaku BF nebo stačí dodržet čekací dobu – viz tabulka instrukcí. Obvykle se používá pouze jednosměrná komunikace MCU→LCD (řídící linka R/W je trvale v log. 0), takže se příznak BF nevyčítá, pouze se jednotlivé příkazy oddělují prodlevami.
  - **8-bitová komunikace**
    - Využívá se všem 8 datových vodičů.
    - První inicializační instrukce nastaví 8-bit komunikaci (bit DL = 1).
  - **4-bitová komunikace**
    - Využívají se jen 4 datové vodiče, a to DB4 – DB7 (horní nibble).
    - Data a instrukce jsou vysílány **nadvakrát**. Nejprve se na DB4 - DB7 nastaví horní nibble, potvrdí se vzestupnou hranou E, pak se nastaví dolní nibble a opět se potvrdí. Výjimkou je první inicializační instrukce, kdy se pošlou jen horní 4 bity pro nastavení 4-bitového módu komunikace (DL = 0) a potvrdí se E jen jednou. Pak se instrukce odešle znova ale už celá – tedy nadvakrát 4 bity aby se nastavily i spodní 4 bity (počet řádků - bit N a font bit - F).
  - Posloupnost při zápisu a čtení:

	1. krok	**2. krok**	**3. krok**	**4. krok**
Zápis	RS, R/W = 0	E = 1	Vystavení dat	E = 0
Čtení	RS, R/W = 1	E = 1	Čtení dat	E = 0

- Příklad 4-bitové komunikace:



- Testování Busy Flag:
  Pro jednoduchost zobrazen pouze jeden datový vodič DB7. Předpokládá se 8-bit komunikace.



12.2.1 Obsluha a připojení LCD ke kitu ATmega32 od PK Design

- Dvouřádkové LCD, 16 znaků na každém řádku.
- LCD modul je k základové desce AVR (ATmega32) připojeno přes 20-pinový plochý kabel následujícím způsobem:

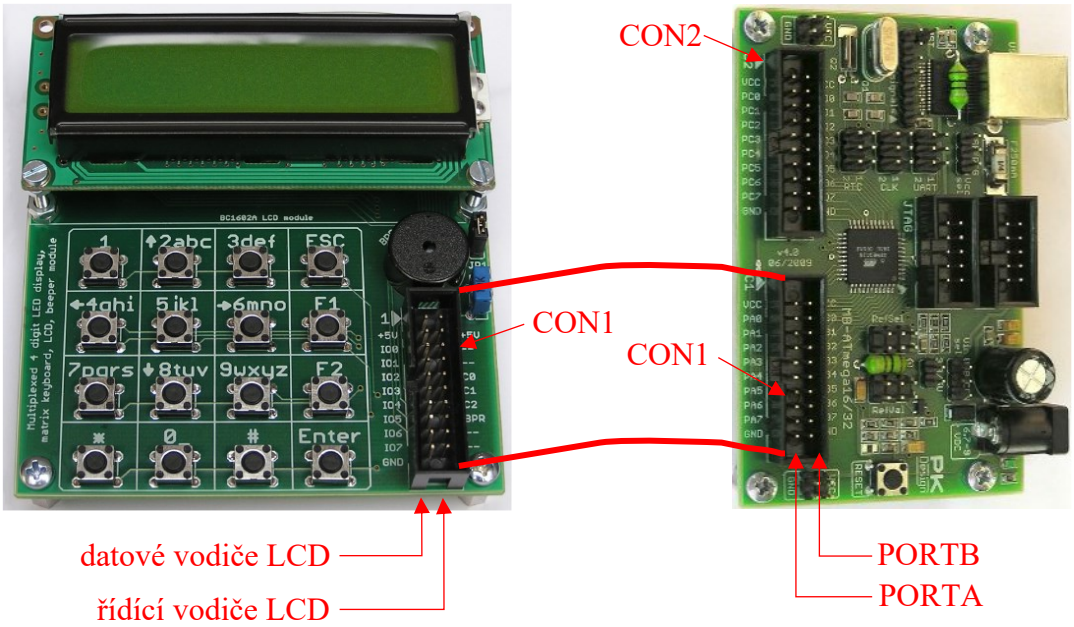
[LCD modul] [AVR PortX]
RS (pin 4) ------ bit 2
RD (pin 5) ------ bit 3
EN (pin 6) ------ bit 4

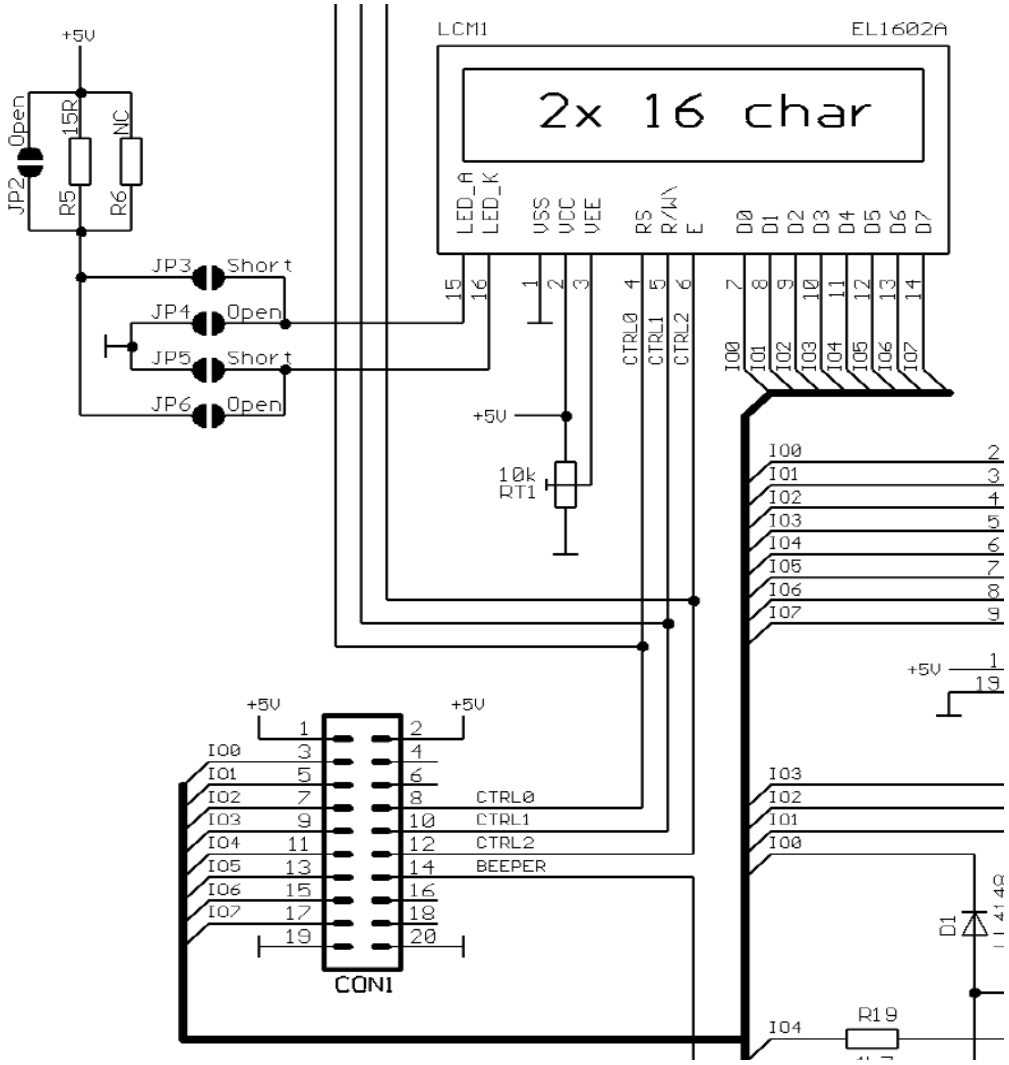
Řídící vodiče LCD.

[LCD modul] [AVR PortY]
DB0 (pin 3) --- bit 0
DB1 (pin 5) --- bit 1
DB2 (pin 7) --- bit 2
DB3 (pin 9) --- bit 3
DB4 (pin 11) --- bit 4
DB5 (pin 12) --- bit 5
DB6 (pin 13) --- bit 6
DB7 (pin 14) --- bit 7

Datové vodiče LCD.

- Při zapojení LCD modulu do CON1 na základové desce ATmega32 se LCD nachází na portech: řídící signály = PORTB, data = PORTA
- Při zapojení LCD modulu do CON2 na základové desce ATmega32 se LCD nachází na portech: řídící signály = PORTD, data = PORTC

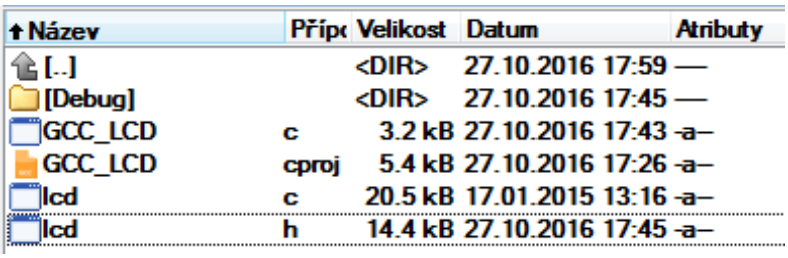




Obrázek 128 Schéma zapojení vývodů LCD na LCD modulu od PK-design

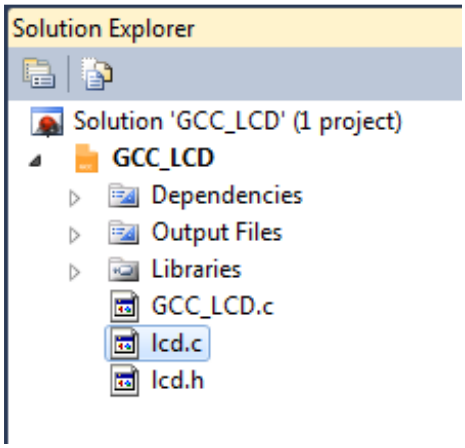
12.2.1.1 Knihovna pro komunikaci s LCD displejem

- Pokud používáme kompilátor GCC, použijeme pro obsluhu LCD externí knihovnu.
  - `lcd.h`
  - `lcd.c`
- Komunikace s LCD probíhá po datových vodičích 4-bitově.
  - Používají se pouze horní 4 bity (DB4 až DB7).
- Příprava před voláním funkcí z knihovny:
  - Nakopírování souborů `lcd.h` a `lcd.c` do adresáře projektu.



Název	Přípc	Velikost	Datum	Atributy
[..]		<DIR>	27.10.2016 17:59	—
[Debug]		<DIR>	27.10.2016 17:45	—
GCC_LCD	c	3.2 kB	27.10.2016 17:43	-a–
GCC_LCD	cproj	5.4 kB	27.10.2016 17:26	-a–
lcd	c	20.5 kB	17.01.2015 13:16	-a–
lcd	h	14.4 kB	27.10.2016 17:45	-a–

  - Přidání souborů `lcd.h` a `lcd.c` do projektu v Atmel studiu.



  - Úprava definic připojení LCD k AVR – editace konstant v souboru `lcd.h`.
    - Definují se porty a piny na AVR, ke kterým je LCD připojeno.
    - Pro připojení kitu LCD do konektoru CON1 na základové desce ATmega32 nastavit tak, jak je uvedeno níže. Při zapojení do CON2 místo PORTA a PORTB napsat PORTC a PORTD.

```
#ifndef LCD_DATA_PORT
#define LCD_DATA_PORT       PORTA        /**< port for the LCD data lines   */
#endif
#ifndef LCD_CONTROL_PORT
#define LCD_CONTROL_PORT    PORTB        /**< port for the LCD control lines   */
#endif
#ifndef LCD_DATA0_PORT
#define LCD_DATA0_PORT    LCD_DATA_PORT     /**< port for 4bit data bit 0 */
#endif
#ifndef LCD_DATA1_PORT
#define LCD_DATA1_PORT    LCD_DATA_PORT     /**< port for 4bit data bit 1 */
#endif
```

```
#ifndef LCD_DATA2_PORT
#define LCD_DATA2_PORT   LCD_DATA_PORT     /**< port for 4bit data bit 2 */
#endif
#ifndef LCD_DATA3_PORT
#define LCD_DATA3_PORT   LCD_DATA_PORT     /**< port for 4bit data bit 3 */
#endif
#ifndef LCD_DATA0_PIN
#define LCD_DATA0_PIN    4            /**< pin for 4bit data bit 0  */
#endif
#ifndef LCD_DATA1_PIN
#define LCD_DATA1_PIN    5            /**< pin for 4bit data bit 1  */
#endif
#ifndef LCD_DATA2_PIN
#define LCD_DATA2_PIN    6            /**< pin for 4bit data bit 2  */
#endif
#ifndef LCD_DATA3_PIN
#define LCD_DATA3_PIN    7            /**< pin for 4bit data bit 3  */
#endif
#ifndef LCD_RS_PORT
#define LCD_RS_PORT      LCD_CONTROL_PORT     /**< port for RS line         */
#endif
#ifndef LCD_RS_PIN
#define LCD_RS_PIN       2            /**< pin  for RS line         */
#endif
#ifndef LCD_RW_PORT
#define LCD_RW_PORT      LCD_CONTROL_PORT     /**< port for RW line         */
#endif
#ifndef LCD_RW_PIN
#define LCD_RW_PIN       3            /**< pin  for RW line         */
#endif
#ifndef LCD_E_PORT
#define LCD_E_PORT       LCD_CONTROL_PORT     /**< port for Enable line     */
#endif
#ifndef LCD_E_PIN
#define LCD_E_PIN        4            /**< pin  for Enable line     */
#endif
```

- Přidání `#include "lcd.h"` na začátek zdrojáku naší aplikace.



12.2.1.2 Funkce pro práci s LCD

- Kompletní nápověda ke knihovně je uvedena v `manual_lcd.html`
  - Nebo viz hlavičkový soubor `lcd.h`, jehož kód je dostatečně okomentován.
- Zde pouze seznam funkcí, které je možné pro obsluhu LCD používat.

void	**lcd_init** (uint8_t dispAttr) Initialize display and select type of cursor. More...
void	**lcd_clrscr** (void) Clear display and set cursor to home position. More...
void	**lcd_home** (void) Set cursor to home position. More...
void	**lcd_gotoxy** (uint8_t x, uint8_t y) Set cursor to specified position. More...
void	**lcd_putc** (char c) Display character at current cursor position. More...
void	**lcd_puts** (const char *s) Display string without auto linefeed. More...
void	**lcd_puts_p** (const char *progmem_s) Display string from program memory without auto linefeed. More...
void	**lcd_command** (uint8_t cmd) Send LCD controller instruction command. More...
void	**lcd_data** (uint8_t data) Send data byte to LCD controller. More...
#define	**lcd_puts_P**(__s) **lcd_puts_p**(PSTR(__s)) macros for automatically storing string constant in program memory

Demonstrační aplikace pro Atmel Studio

```
/*
 * GCC_LCD.c
 *
 * Created: 27.10.2016 16:53:39
 *  Author: zdma0726
 */


#include <stdlib.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include "lcd.h"


/*
** Two user defined characters placed in program memory
*/
static const PROGMEM unsigned char copyRightChar[] =
{
       0x07, 0x08, 0x13, 0x14, 0x14, 0x13, 0x08, 0x07,
       0x00, 0x10, 0x08, 0x08, 0x08, 0x08, 0x10, 0x00
};


int main(void)
{
    char buffer[7];
    int  num;
    unsigned char i, j;

    /* initialize display, cursor off */
    lcd_init(LCD_DISP_ON);

    for (;;)  /* loop forever */
       {
        /*
         * Test 1:  write text to display
         */

        /* clear display and home cursor */
        lcd_clrscr();

        /* put string to display (line 1) with linefeed */
        lcd_puts("LCD Test Line 1\n");

        /* cursor is now on second line, write second line */
        lcd_puts("Line 2");

        /* move cursor to position 8 on line 2 */
        lcd_gotoxy(7,1);

        /* write single char to display */
        lcd_putc(':');

        _delay_ms(2000);


        /*
```

```
 * Test 2: use lcd_command() to turn on cursor
 */

/* turn on cursor */
lcd_command(LCD_DISP_ON_CURSOR);

/* put string */
lcd_puts( "CurOn");

_delay_ms(2000);


/*
 * Test 3: display shift
 */

lcd_clrscr();     /* clear display home cursor */

/* put string from program memory to display */
lcd_puts_P( "Line 1 longer than 14 characters\n" );
lcd_puts_P( "Line 2 longer than 14 characters" );

for (j=1;j<20;j++)
      {
            _delay_ms(1000);

            /* move BOTH lines one position to the left */
            lcd_command(LCD_MOVE_DISP_LEFT);
      }
      _delay_ms(2000);

/* turn off cursor */
lcd_command(LCD_DISP_OFF);


/*
 *   Test: Display integer values
 */

lcd_clrscr();   /* clear display home cursor */

/* convert interger into string */
      num = 134;
itoa( num , buffer, 10);

/* put converted string to display */
lcd_puts(buffer);

_delay_ms(2000);


/*
 *  Test: Display userdefined characters
 */

lcd_clrscr();   /* clear display home cursor */

lcd_puts("Copyright: ");

/*
 * load two userdefined characters from program memory
 * into LCD controller CG RAM location 0 and 1
```

```
         */
        lcd_command(_BV(LCD_CGRAM));  /* set CG RAM start address 0 */
        for(i=0; i<16; i++)
        {
            lcd_data(pgm_read_byte_near(&copyRightChar[i]));
        }

        /* move cursor to position 0 on line 2 */
        /* Note: this switched back to DD RAM adresses */
        lcd_gotoxy(0,1);

        /* display user defined (c), built using two user defined chars */
        lcd_putc(0);
        lcd_putc(1);


        _delay_ms(5000);

    }
}
```

13 Použití vyššího programovacího jazyka na MCU

13.1 Úvod

- vyšší programovací jazyk = High Level Language (HLL)
- malé MCU – dříve nemyslitelné, důvody: nedostatek RAM, ROM, malý výkon CPU
- dnes poměrně běžné, používané jazyky:
  - C → absolutně nejčastěji
  - Pascal, Basic, atd. → také, ale v daleko menší míře
- důvody:
  - rychlost vývoje
  - přenositelnost na jiné typy CPU (MCU)
- ideální stav:
  - architektura MCU je přizpůsobena požadavkům překladače
  - kód je stejně efektivní jako Assembler
- skutečnost → použití HLL = nárůst kódu (code overhead):
  - nepřizpůsobená architektura – nárůst kódu o 100ky %
  - přizpůsobená – nárůst kódu o 50 – 100% (závisí na programátorovi)
- dále HLL = C

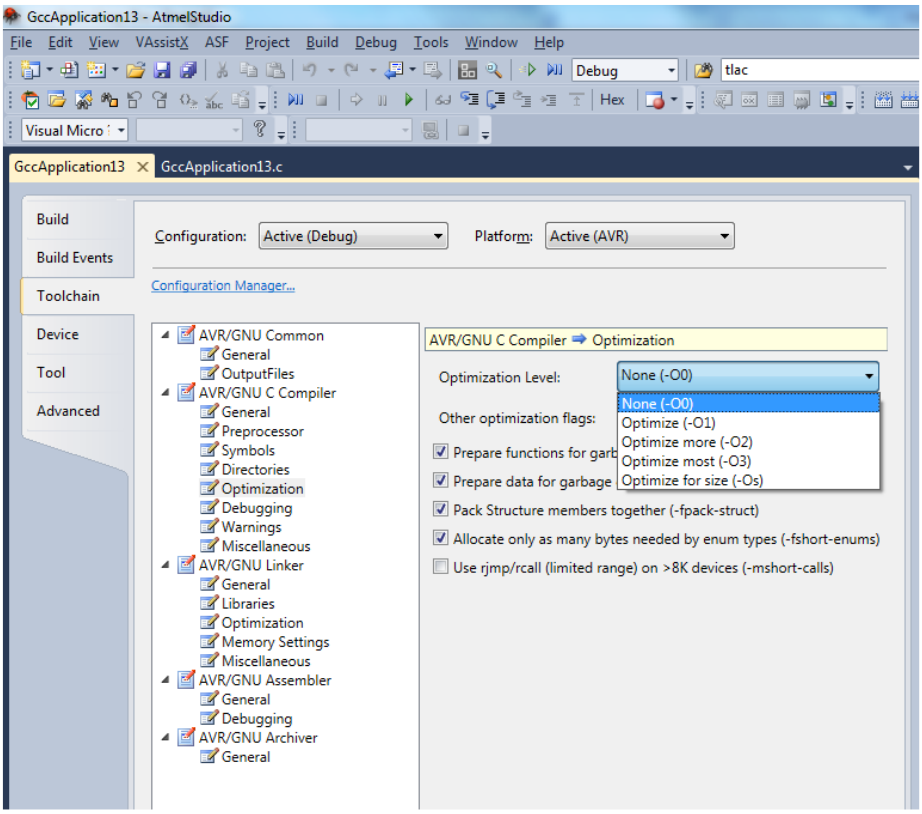
13.2 Kompilátor HLL a ne-PC aplikace

- nutné omezení specifikace HLL (např. dle ANSI C):
  - standardní vstupní a výstupní operace (`printf()`, `scanf()` atd.)
  - omezení je nutné z důvodu omezených zdrojů příslušného MCU, čím velkorysejší jsou systémové zdroje příslušného MCU, tím menší je nutnost zavádět různá rozšíření jazyka nebo jiné odlišnosti od standardů jazyka C
- rozšíření specifikace HLL
  - ne-PC hardware (porty, periferie)
  - nové datové typy (bitová proměnná)
  - harvardská architektura (program je fyzicky oddělen od dat)
  - přerušení (obsluha, povolení/zakázání)
- program v assembleru = pouze výkonný kód
- program v HLL =
  - výkonný kód
  - inicializace CPU pro běh programu (Run Time Environment, Start-up code)
    - u překladačů pro AVR je to cca 100-200 B kódu

- Velikost a rychlost výsledného programu napsaného v HLL je dána:
  - Kvalitou kompilátoru (jak dobře přeloží program z HLL do ASM).
  - Je silně ovlivněna samotným programátorem – nutno promyšleně volit datové typy, paměťové třídy, algoritmy využívající danou architekturu MCU apod.
  - A v neposlední řadě je určena vhodností daného MCU pro programování ve vyšším jazyce (např. v jazyce C/C++).

13.3 Optimalizace

- Každý kompilátor nabízí volbu optimalizace kódu.
- Různé typy optimalizace:
  - Na rychlost kódu (speed)
  - Na velikost kódu (size)
- Různé stupně síly optimalizace:
  - CodeVision: Low, Medium, Maximal
  - GCC: -O0 (bez optimalizace) až -O3 (max. optimalizace), -Os (optimalizace na velikost)
- V závislosti na použitém kompilátoru a úrovni optimalizace se může dosáhnout zmenšení kódu v jednotkách až desítkách procent.



Obrázek 129 Nastavení úrovně optimalizace překladače GCC v Atmel Studio

Tabulka 18 Přehled možných optimalizací v překladu programu nezávislých na hardwaru

Způsob optimalizace	Popis
Constant folding (zprocání konstant)	Výpočty, které obsahují konstanty, jsou v co největší míře prováděny již kompilátorem.
Local Common/Global Common Subexpression Elimination (vyloučení opakujících se výpočtů nebo částí výrazů)	Vícenásobné (opakované) výpočty nebo stejné části výrazů uvnitř výrazu nebo funkce jsou pokud možno co nejvíce eliminovány a počítají se jen jednou, přičemž výsledek prvního výpočtu se ukládá do registru. To vyžaduje velmi přesnou analýzu programu prováděnou kompilátorem.
Optimization of span-dependent instructions and Jumps (optimalizace skokových příkazů)	Meziskoky (skok na jiný skok) jsou odstraňovány tak, že se nahradí skokem na konečný cíl. Zahrnuje to také volbu optimálního skokového příkazu v závislosti na délce skoku (16-bitový nebo 32-bitový příkaz).
Dead Code Elimination (vyloučení „mrtvého“ kódu)	Odstranění nepoužívaného pasivního zdrojového kódu z programu.
Cross-jumping and hoisting (náhrada opakujících se úseků programu skoky)	Identické úseky kódu na různých místech v programu se vytvářejí jen jednou a zpracovávají pomocí skokových příkazů.
Jump Negation (negace skoků)	Testy pro podmíněné skoky se invertují, lze-li tím odstranit jiné skoky nebo nepoužívaný kód.
Data Overlay (překrývání dat)	Datové segmenty funkcí jsou označeny jako staticky překrývané. Spojovací program pak má možnost překrývat segmenty.
Peephole Optimizing (optimalizace plnění)	Redundantní příkazy plnění (zavádění) se odstraňují. Vylučuje se tak nepotřebné zavádění dat a konstant z paměti. Složitější operace se nahrazují jednoduššími, je-li tím možno ušetřit paměťové místo nebo čas běhu programu.
Simple loop Optimization (optimalizace jednoduchých smyček)	Jednoduché smyčky se ptimalizují z hlediska doby běhu tak, že se kód vysune ze smyčky. Optimalizace obzvlášť „horkých“ míst programu, většinou smyček (90 % času v 10 % kódu).
Loop Rotation (rotace smyček)	V programových smyčkách se zamění uspořádání kódu, dosáhne-li se tím rychlejšího a efektivnějšího kódu.
Control Flow Optimization (optimalizace řídícího toku)	Výrazy typu switch – case se optimalizují a zjednodušují jako skokové tabulky nebo skokové řetězce. Proměnné z různých oblastí paměti se zavádějí přímo do operace.

Tabulka 19 Optimalizace závislé na hardwaru

Způsob optimalizace	Popis
Register Variable (registrové proměnné)	Automatické proměnné a parametry funkcí se pokud možno co nejvíce umísťují do registrů. Tím je přístup na ně efektivnější a nezabírají žádné místo v RAM.
Simple Acces Optimization (optimalizace jednoduchým přístupem)	Přístupy na interní datové a bitové adresy jsou optimalizovány strojovými instrukcemi závislými na MCU, využívají se speciální posloupnosti strojových příkazů.
Code Rearrangement (reorganizace kódu)	Je-li smyčka FOR efektivnější než programátorem použitá smyčka WHILE, kompilátor kód změní. Nebo jestliže smyčka čítá nahoru a pak následuje dotaz <> 0, zkouší se, zda existuje příkaz procesoru pro ?:<>. Pokud ne, smyčka se otočí a testuje na = 0.

13.4 Ukázky optimalizací

13.4.1 Constant folding (zpracování konstant)

Kód v C	Kód v ASM

`#define K 1000`
 	
`void main(void)`
`{`
`  int itemp = 0;`
 	
`  while (1)`
`  {`
`  itemp = itemp + K + 5;`
`  }`
`}`
	`.MACRO __GETWRN`
	`  LDI  R@0,LOW(@2)`
	`  LDI  R@1,HIGH(@2)`
	`.ENDM`
	
	`.MACRO __ADDWRN`
	`  SUBI R@0,LOW(-@2)`
	`  SBCI R@1,HIGH(-@2)`
	`.ENDM`
	
	`.CSEG`
	`_main:`
	`; 0000 001D  int itemp = 0;`
	`; 0000 001E`
	`; 0000 001F  while (1)`
	`;  itemp -> R16,R17`
	`  __GETWRN 16,17,0`
	`_0x3:`
	`;0000 0020  {`
	`;0000 0021 itemp = itemp + K + 5;`
	`  __ADDWRN 16,17,1005`
	`;0000 0022  };`
	`  RJMP _0x3`
	`;0000 0023 }`
	`_0x6:`
	`  RJMP _0x6`

13.4.2 Dead Code Elimination (vyloučení „mrtvého“ kódu)

Kód v C

```
#include <mega32.h>
#define K 1000

void main(void)
{
      int itemp = 0;

      while (1)
      {
            itemp = itemp + K + 5;
      };

      itemp++; // nikdy se neprovede
}
```

Kód v ASM

```
.MACRO __GETWRN
      LDI  R@0,LOW(@2)
      LDI  R@1,HIGH(@2)
.ENDM

.MACRO __ADDWRN
      SUBI R@0,LOW(-@2)
      SBCI R@1,HIGH(-@2)
.ENDM

.CSEG
_main:
; 0000 001D       int itemp = 0;
; 0000 001E
; 0000 001F       while (1)
;      itemp -> R16,R17
      __GETWRN 16,17,0
_0x3:
;0000 0020     {
;0000 0021 itemp = itemp + K + 5;
      __ADDWRN 16,17,1005
;0000 0022     };
      RJMP _0x3
; 0000 000C
; 0000 000D     itemp++;       //
nikdy se neprovede
; OPERACE itemp++ ZDE CHYBI !!!
;0000 000E }
_0x6:
      RJMP _0x6
```

13.4.3 Vyloučení opakujících se výpočtů nebo částí výrazů

Nefunkční příklad – optimalizace nefunguje, výraz se počítá vždy znovu.

Kód v C

```
#include <mega32.h>

void main(void)
{
      char a = 0;
      char b = 1;
      char c,d;

      c = a + b + 1;
      d = a + b + c; // vyraz (a+b)
                        se opakuje
}
```

Kód v ASM

```
;void main(void)
;___ 0000 0005 {

.CSEG
_main:
;___ 0000 0006     char a = 0;
;___ 0000 0007     char b = 1;
;___ 0000 0008     char c,d;
;___ 0000 0009
;___ 0000 000A     c = a + b + 1;
;      a -> R17
;      b -> R16
;      c -> R19
;      d -> R18
      LDI  R17,0    ; a = 0
      LDI  R16,1    ; b = 1
      MOV  R26,R17
      CLR  R27
      MOV  R30,R16
      LDI  R31,0
      ADD  R30,R26 ; a + b
```

	``
	ADC R31,R27
	ADIW R30,1 ; + 1
	MOV R19,R30 ; c = a + b + 1
	;___ 0000 000B ; d = a + b + c
	MOV R26,R17
	CLR R27
	CLR R30
	ADD R26,R16 ; a + b
	ADC R27,R30
	MOV R30,R19
	LDI R31,0
	ADD R30,R26 ; + c
	MOV R18,R30 ; d = a + b + c
	;___ 0000 000C }
	_0x3:
	RJMP _0x3
	``

Správně by mělo po optimalizaci vypadat takto:

```
void main(void)
{
      char a = 0;
      char b = 1;
      char c,d;

      t = a + b;
      c = t + 1;
      d = t + c;
}
```

13.4.4 Optimalizace kódu ve smyčce

Jestliže je výsledek výrazu uvnitř smyčky stále stejný (nemění se), pak nezávisí na smyčce a kompilátor může výraz přesunout před smyčku. To vede ke zrychlení – výraz se počítá pouze jednou, nikoliv při každém průchodu smyčkou.

Z výrazu:

```
while (f())       // b * c se ve smycce nemeni
    g(b * c);
```

se stane:

```
T = b * c;        // výpočet výrazu b * c vysunut před smyčku
while (f())
    g(T);
```

13.5 Obecná pravidla při psaní programu

- Používat co nejmenší velikostí postačující typ proměnných
  - Např. místo `long` použiji `short int`, když vím, že rozsahem bude dostačovat – šetření pamětí a výpočetního výkonu.
  - Menší datový typ = menší paměťová náročnost, rychlejší kód.
  - Nutné zejména u malých (8-bit) MCU s malou datovou pamětí.
- Pozor na automatickou konverzi celočíselných typů z „nižších“ na „vyšší“ (Integral Promotion)
  - Aritmetické operace (+,-,*,/,++,--,~) jsou vždy provedeny na typech `int` nebo `long` (`signed` nebo `unsigned`).
  - Automatická konverze proměnných typu `char`, `short`, `enum` na typ `int` nebo `unsigned int` před spuštěním výpočtu.
    - Jestliže se operand vejde do `signed int`, tak je převeden na `signed int`, pokud se nevejde, je převeden na `unsigned int`.
  - **Nebezpečí, které z toho vyplývá:**
    - Pokud je typ `int` velký 32-bitů a dojde k násobení dvou operandů typu `short int` (16-bitů), je výsledek 32-bitový a vejde se do typu `int`.
    - Pokud je typ `int` velký jen 16-bitů (např. na AVR), tak výsledek násobení přeteče (výsledek bude 16-bitový, oříznutý).
  - Šířka slova, ve které proběhne výpočet závisí na velikosti typu `int` – nekonzistentnost v jazyce C.
  - **Příklad**:

```
unsigned int a = 40000; /* uint16 nebo uint32 */
unsigned int b = 30000; /* uint16 nebo uint32 */
unsigned int c;
c = a + b;  /* c = 4464 pokud unsigned int == uint16
               c = 70000 pokud unsigned int == uint32
                  */
```

    Neznalý by předpokládal výsledek 70000, ale hodnota přiřazená do `u` závisí na velikosti typu `int`. Pokud je `int` 32-bitový, sčítání proběhne ve 32-bit aritmetice v `c` bude uložena správná hodnota. Pokud je `int` pouze 16-bitový, sčítání proběhne v 16-bit aritmetice, výsledek přeteče rozsah 16-bitů a v `c` bude 4464 (70000 modulo 65536).

  - Zdroj problému – velikost datového typu `int` je platformově závislá !
  - Místo standardních datových typů by se v programu měly používat **uživatelsky definované datové** typy u nichž je **zaručena platformová nezávislost** (`uint8_t, uint16_t, uint32_t, …`).

Řešení – explicitní typová konverze operandů na typ, ve kterém je má proběhnout aritmetická operace (přetypování na typ výsledku aritmetické operace).

V příkladu se předpokládají tyto velikosti datových typů:

```
typedef unsigned short uint16_t //(16-bit neznaménkové celé číslo)
typedef unsigned long uint32_t //(32-bit neznaménkové celé číslo)

uint16_t a = 40000;
uint16_t b = 30000;
uint32_t c;
c = (uint32_t)a + b;    /* c = 70000 */
/* staci pretypovat pouze jeden operand, druhy se pretypuje automaticky diky implicitini typove konverzi */
```

- **Chybné příklady (programátor si myslí, že výsledek bude OK, ale výsledek bude nesprávný):**

V příkladu se předpokládají tyto velikosti datových typů:

```
uint16_t    je unsigned short int (16-bit)
uint32_t    je unsigned long int (32-bit)
float32_t   je float (32-bit)
float64_t   je double (64-bit)

/* násobení proběhne na 16-bitech, výsledek přeteče */
uint32_t u32a = (uint32_t)(u16a * u16b);

/* proběhne jako celočíselné dělení */
float64_t f64a = u16a / u16b;

/* proběhne jako celočíselné dělení */
float32_t f32a = (float32_t)(u16a / u16b);

/* sčítání proběhne na 32-bitech, výsledek přeteče */
float64_t f64a = f32a + f32b;

/* sčítání proběhne na 32-bitech, výsledek přeteče */
float64_t f64a = (float64_t)(f32a + f32b);
```

- **Používat modifikátor `volatile` pro proměnné, jejichž hodnota se mění asynchronně** např. někde v přerušení. Je to ochrana před možnou chybnou optimalizací této proměnné překladačem, která by vedla k nefunkčnosti kódu.

```
volatile unsigned long time_ms = 0;

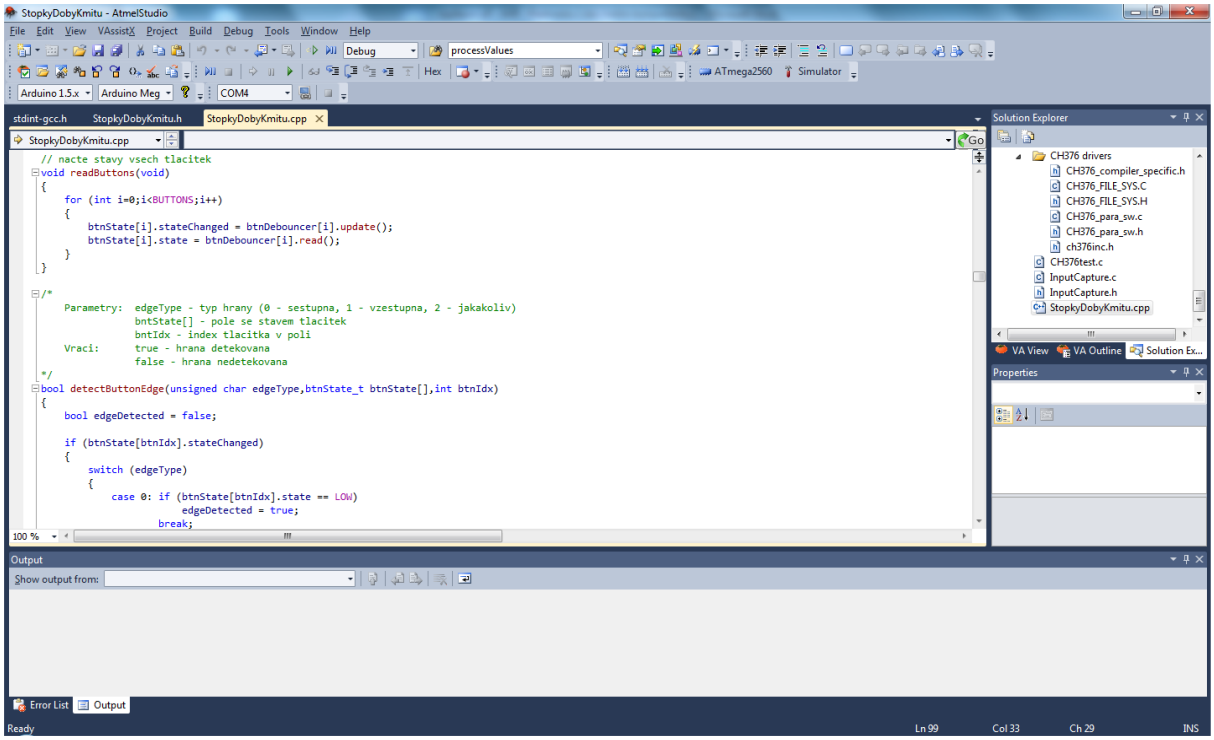
// Timer 0 overflow interrupt service routine
ISR(TIMER0_OVF_vect)
{
   time_ms++;
   // nejaky dalsi kod
}

// Timer 0 overflow interrupt service routine
void main(void)
{
   // kompilator tento kod uplne vynecha pokud nebude uvedeno slovo
   // volatile před promennou time_ms
   if (time_ms == 1000)
   {
         // nejaky dalsi kod
   }
}
```

- **Vyhýbat se rekurzivnímu volání funkcí**
    - Rekurzivním voláním velmi silně narůstá spotřeba datové paměti a hrozí zaplnění zásobníku.
    - Při každém volání funkce se do zásobníku uloží návratová adresa, obsah použitých pracovních registrů, SREG.

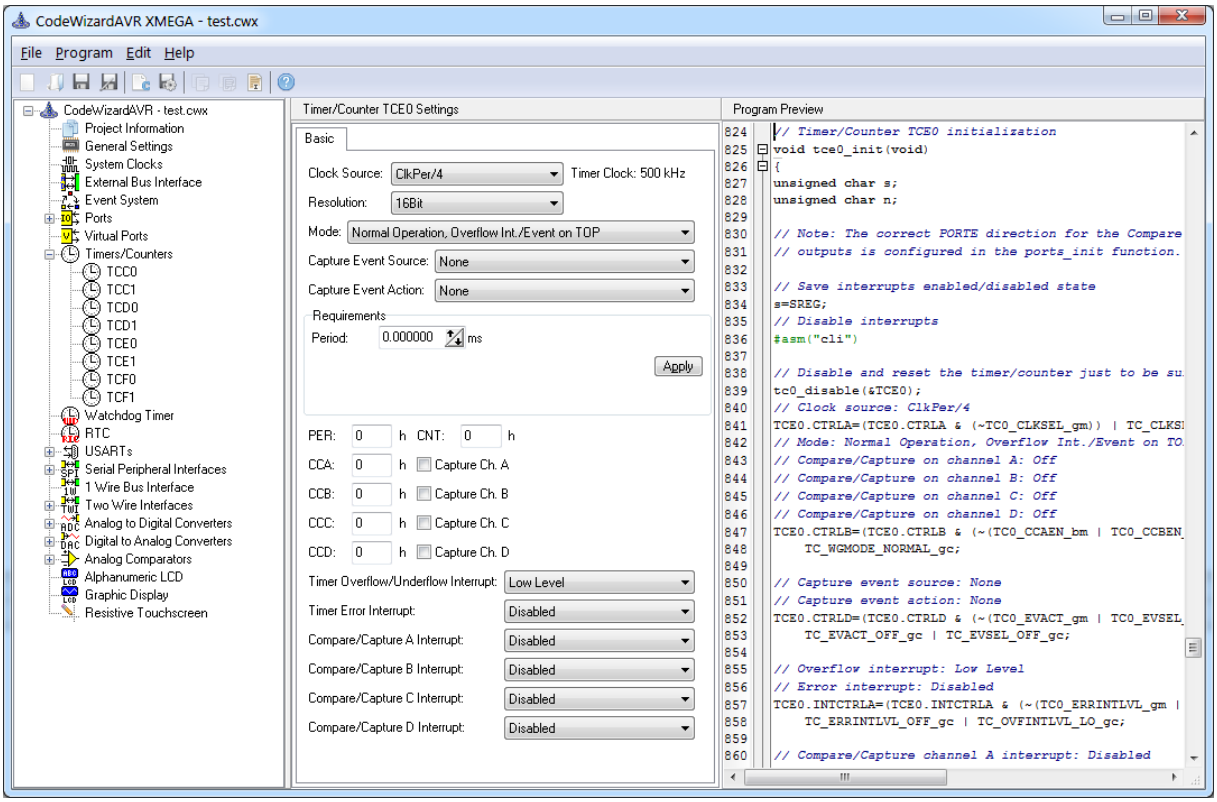
13.6 Přehled C kompilátorů pro AVR

- **AVR GCC**
  - GCC = GNU Compiler Collection
  - Je free (volně šiřitelný pod GNU licencí).
  - Nemá vlastní vývojové prostředí, využívá integrace do AVR studia (Atmel Studia).
  - Stále vylepšován a zdokonalován komunitou (http://winavr.sourceforge.net/).
  - Používá knihovnu AVR-libc, která zajišťuje implementaci funkcí standardní C knihovny (vstupně/výstupní funkce, práce s řetězci, matematické funkce, dynamická alokace paměti atd.) na AVR.
    - http://www.nongnu.org/avr-libc/user-manual/index.html
    - https://www.nongnu.org/avr-libc/user-manual/modules.html



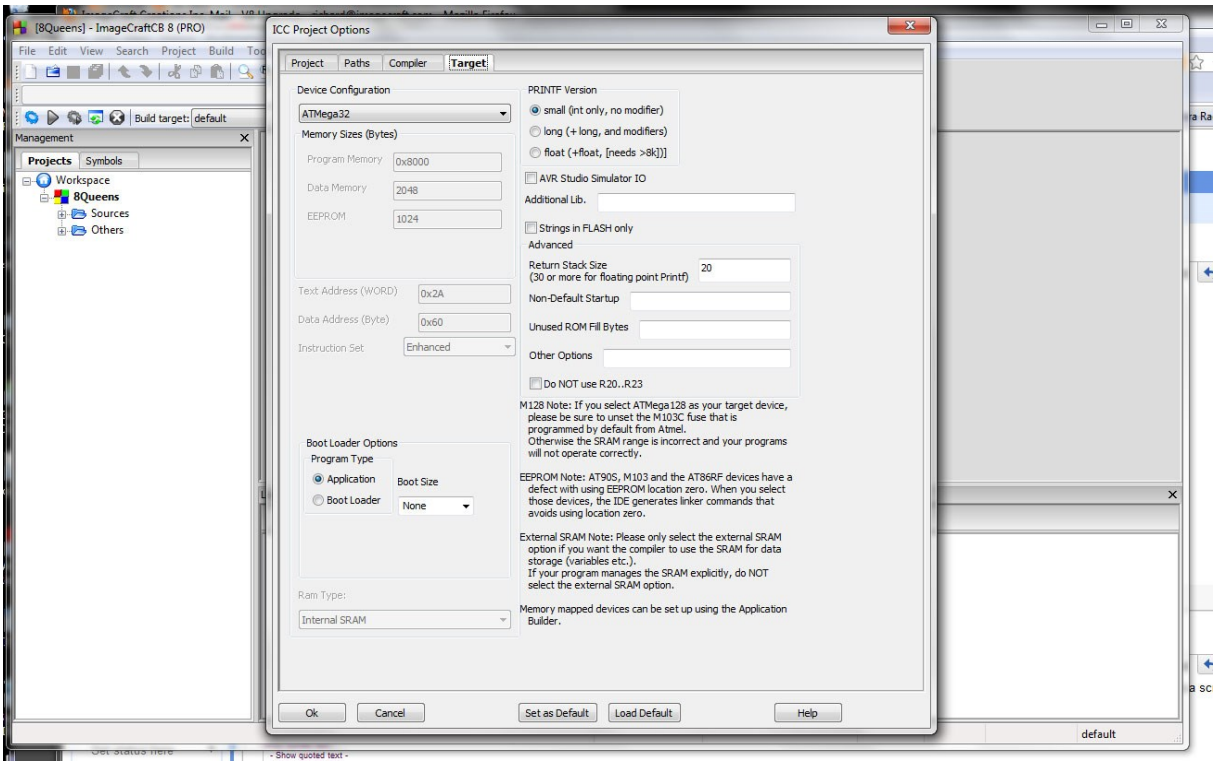
Obrázek 130 Atmel Studio 6.1

- **CodeVision AVR**
  - Vývojové prostředí (editor, C kompilátor, neobsahuje simulátor).
  - Obsahuje CodeWizard pro automatické generování kódu dle nastavení periferií.
  - Obsahuje velké množství knihovních funkcí pro ovládání interních periferií a externích zařízení (LCD, RTC, I2C a SPI sběrnice, senzory teploty apod.)
  - Zabudovaný ISP programátor a terminál.
  - Generuje objektový COFF soubor, který lze ladit v debuggeru AVR studia.
  - Vhodný pro začátečníky (CodeWizard, knihovny pro externí obvody).
  - Nevýhody
    - V zápisu kódu možno použít řadu ne-ANSI-C konstrukcí, které sice zjednodušují programátorovi práci, ale kód se tak stává hůře přenositelný (nelze vzít program napsaný v CodeVision a přenést ho jednoduše např. na GCC, bylo by třeba vynaložit úsilí na jeho úpravu).
    - Je placený. K dispozici je verze zdarma s velkým omezením kódu na 4 kB.
  - http://www.hpinfotech.ro/index.html



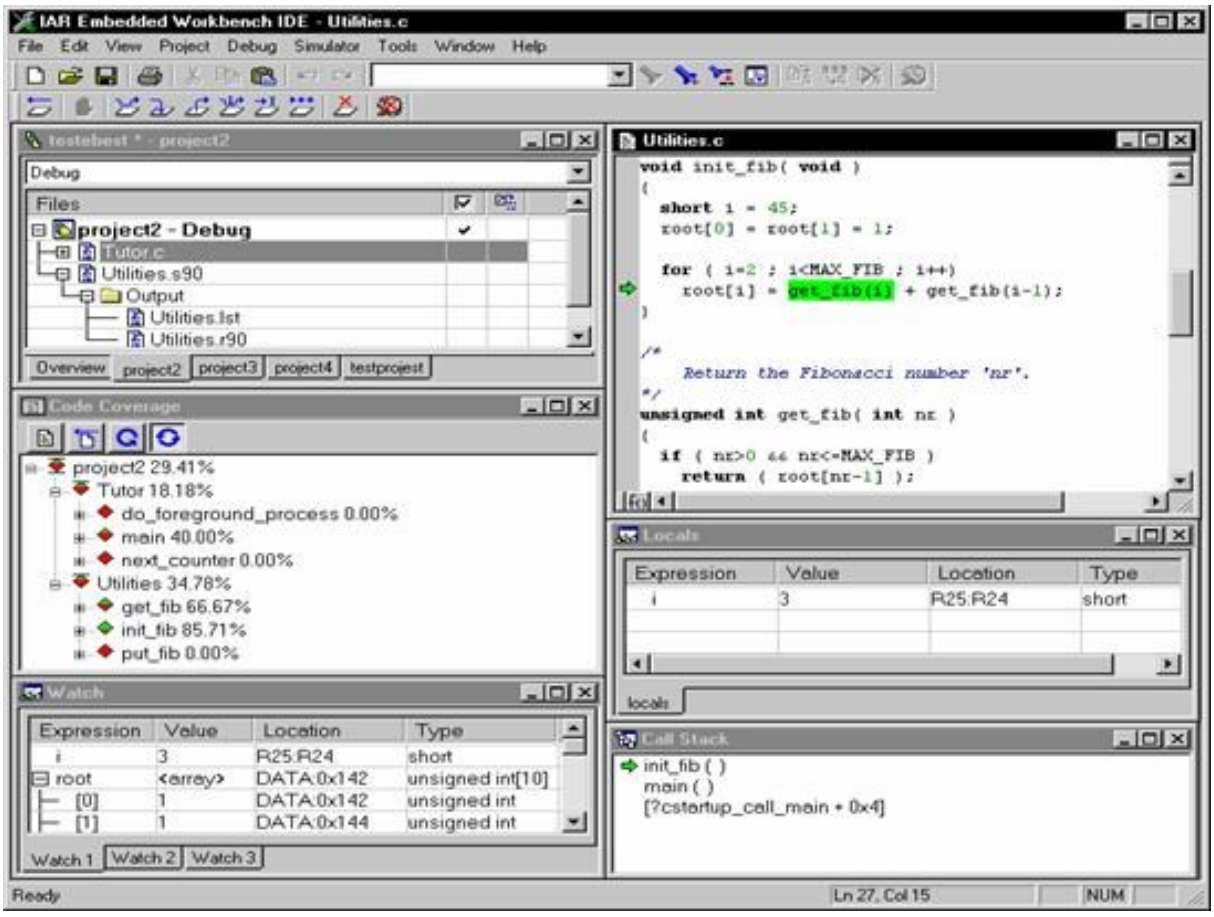
Obrázek 131 CodeVision AVR IDE

- **ImageCraft Compiler AVR**
  - IDE, C/C++ kompilátor, průvodce nastavením periferií
  - https://imagecraft.com/
  - Zastavení vývoje v roce 2020.



Obrázek 132 ImageCraft Compiler

- **IAR Embedded Workbench**
  - Firma IAR Systems je zkušený výrobce kompilátorů pro MCU (první firma, která nabízela ANSI-C kompilátor pro 8051).
  - Kompletní vývojové prostředí (editor, C/C++ kompilátor, debugger)
  - Velmi dobrá optimalizace kódu, doporučen přímo firmou ATMEL.
  - MISRA C checker (kontrola kvality kódu)
  - Podpora vybraných RTOS (real time operating systems)
  - Certifikován pro použití v bezpečnostně relevantních aplikacích.
  - Velmi drahý kompilátor
  - https://www.iar.com/embedded-development-tools/iar-embedded-workbench



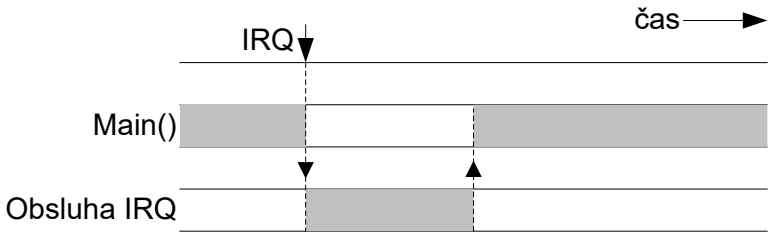
Obrázek 133 IAR Embedded Workbench

O tom, který kompilátor je lepší a který horší se na internetu vedou debaty. Nedá se přesně určit vítěz, ale lze téměř s jistotou říci, že nejkvalitnější bude IAR compiler. Na druhém konci žebříčku se bude nacházet CodeVision a GCC. Kompilátory se také navzájem liší tím, v jakém rozsahu dodržují standard ANSI-C (i když všichni výrobci uvádějí, že jejich kompilátor dodržuje normu ANSI-C).

14 Přerušovací systém

14.1 Úvod a princip přerušení

- Přerušení = **interrupt**
- Důvod – obsluha asynchronních událostí (CPU mnohem rychlejší než pomalé periferie – „má klávesnice nějaké znaky?“)
- Typy přerušení:
  - SW = pouze jiný způsob volání podprogramů
  - HW = CPU vykoná kód na základě signalizované externí (vzhledem k CPU) události (stisknuta klávesa, časovač ukončil odpočet atd.):
    1) Periferie → žádost o obsluhu přerušení (**interrupt request**, **IRQ**)
    2) CPU → Přerušení vykonávání „normálního“ programu
    3) CPU → Obsluha události (= vykonání kódu pro obsluhu)
    4) CPU → Návrat do hlavního programu na místo přerušení a jeho pokračování



  - Pozn.: často pojem HW přerušení = přerušení obecně (dále také)
- **Řadič přerušení** (interrupt controller) = obvod pro řízení přerušení
  - rozhoduje o:
    - signalizovat/nesignalizovat žádost o přerušení
    - prioritě přerušení
  - dle umístění vzhledem k CPU:
    - interní: u MCU (i periferie jsou součástí čipu)
    - externí: je ve zvláštním obvodu, připojen na sběrnici

14.1.1 Obslužné programy přerušení

- **Interrupt Service Routine (ISR)**, jiný název např. **Interrupt Handler**
- = asynchronně volaný podprogram (okamžik volání nelze určit, v programu není `call` podprogramu)

14.1.2 Tabulka vektorů přerušení

- **Interrupt Vector Table (IVT)**
- Po IRQ CPU provede skok na pevné místo v paměti programu (dále jen ROM), každý zdroj přerušení má svůj vlastní vektor
- Typická paměťová mapa ROM s vyznačenou IVT:

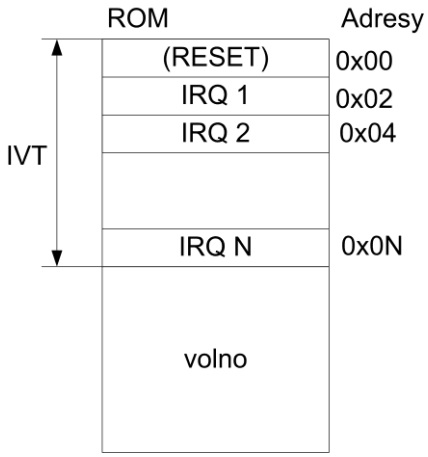


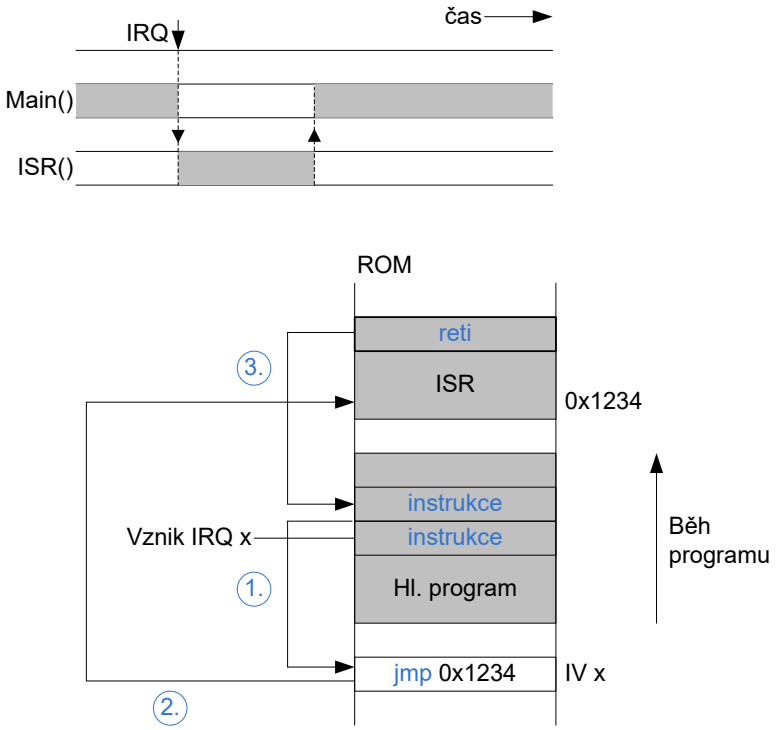
Table 18. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Obrázek 134 Tabulka vektorů přerušení ATmega32

- zajištění asynchronního volání ISR
  - vektory jsou obvykle blízko sebe (u AVR na vzdálenost 2 adres) → na vektor nelze umístit ISR, ale pouze jednu instrukci `jmp` → z vektoru se skáče dál na jinou adresu v ROM, tam kde leží ISR

14.1.3 Činnost CPU



ad 1. – automatický HW skok na IV x po IRQ x

- Postup:
  - vznik žádosti o přerušení (IRQ)
  - rozhodnutí o přijetí/nepřijetí IRQ (viz dále), pokud ano, pak
  - dokončení právě zpracovávané instrukce I
  - skok na odpovídající vektor přerušení (IV) (1.)
    - (SP) ← návratová adr. (= adr. I+1)
    - PC = IV
  - z vektoru přerušení (IV) skok na začátek ISR (2.)
  - běh ISR
    - ISR zakončena instrukcí `reti`
    - Obnovení stavu logiky přerušení
    - Návrat do místa odskoku, pokračování další instrukcí za místem odskoku (3.)
  - `ret` `PC ← (SP)`

14.1.4 Programy s přerušením

- kostra programu v ASM s přerušením

```
        .ORG 0x0000
        jmp  main       // v ROM 0x0000, Reset, přeskočení IVT

        .ORG IV1        // IV1 = adresa vektoru přerušení
        jmp  ISR1       // skok na rutinu ISR1

        .ORG IV2        // IV1 = adresa vektoru přerušení
        jmp  ISR2       // skok na rutinu ISR2


main:
        // instrukce hlavního programu
        // ...
        jmp main        // nekonečná smyčka


//podprogram pro obsluhu přerušení od zdroje 1
ISR1:       // nějaký kód
        reti


//podprogram pro obsluhu přerušení od zdroje 2
ISR2:       // nějaký kód
reti
```

  - pokud program přerušení nevyužívá – na adresách IVT může ležet normální instrukce

14.1.5 Problém asynchronního volání ISR

- ISR by se mělo chovat vzhledem k CPU transparentně - stav CPU po vykonání ISR musí zůstat nezměněn.
- Příklad problémové situace:
  - `Main()` – obsahuje cyklus, `r18` jako počítadlo

```
        nop
        nop             // instrukce X
        nop
        ldi  r18,10
cyklus:
        // nějaké operace
        nop
        nop             // instrukce Y
        dec  r18        // instrukce Z
        brne cyklus
```

  - ISR() – sčítá dva registry, jeden z nich je r18

```
ISR:
        // nějaké operace
        ldi  r18, 10
        add  r0, r18
        // nějaké operace
        reti
```

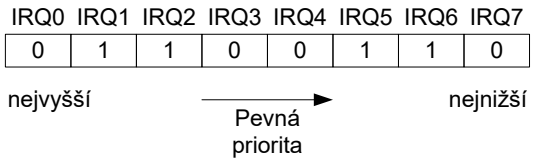
  - IRQ po vykonání instrukce X – vše OK
  - IRQ po vykonání instrukce Y = zničení hodnoty počítadla (= změna počtu opakování na 10 díky činnosti ISR)
  - IRQ po vykonání instrukce Z = ovlivnění chodu smyčky na základě výsledku součtu v ISR (pokud `add` nastaví příznak Z v SREG, tak to ovlivní chování `brne` – smyčka bude předčasně ukončena)
- Řešení: ISR musí zajistit uložení/obnovu všech používaných prostředků
  - Použitím zásobníku.
  - Na začátku ISR na zásobník uložit obsah všech používaných registrů v ISR + SREG.
  - Na konci ISR ze zásobníku obnovit tyto registry + SREG.

- Správný kód ISR:

```
ISR:
        // uložení registrů
        push r0
        push r18
        in   r18, SREG // uložení registru stavu CPU
        push r18
        // vlastní užitečný kód
        ldi  r18, 10
        add  r0, r18
        // obnova registrů
        pop  r18
        out  SREG, r18 // obnova registru stavu CPU
        pop  r18
        pop  r0
        reti
```

14.1.6 Priorita přerušení

- Rozhoduje o tom, která žádost o přerušení bude obsloužena.
- Priorita hraje roli v těchto případech:
  - najednou se sejde více IRQ v jeden okamžik
  - právě je obsluhováno přerušení X je vyvolán požadavek na přerušení Y
- Různě u různých CPU.
- Pevná priorita
  - dána pořadím vektorů v IVT (obvykle vektory s nižší adresou = vyšší priorita)
- Volitelná priorita
  - programátor může určit důležitost vektorů
- Volitelná priorita, jednodušší případ – prioritní skupiny
  - registr úrovně priority – každý vektor bit: 0 = nižší, 1 = vyšší priorita

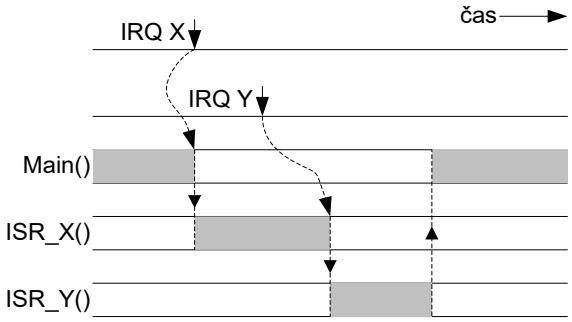


  - Pokud bude priorita dle výše uvedeného obrázku a sejde-li se najednou:
    - IRQ0 a IRQ1 – bude obslouženo nejdříve IRQ1 (skupina s vyšší prioritou
    - IRQ1 a IRQ2 – bude obslouženo nejdříve IRQ1 (dle pevné priority v IVT)

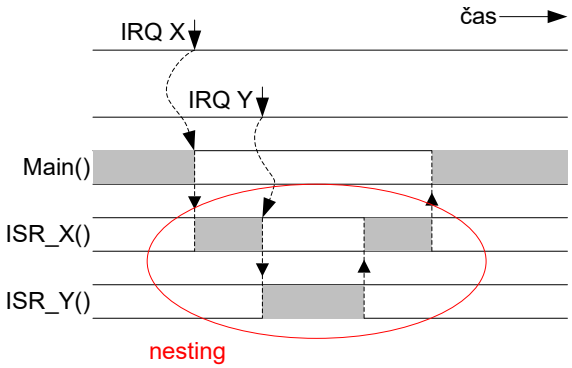
- Volitelná priorita, složitější případ – každý vektor má registr, číslo = priorita (obvykle nesmí mít dva vektory stejnou prioritu).
- Pro oba případy – lze měnit prioritu za běhu programu.

14.1.7 Vnořená přerušení

- Nested Interrupts
- = probíhající přerušení je přerušeno dalším přerušením
- Chování opět dle typu CPU, možné způsoby:
  - Uvažujme, že IRQ Y má vyšší prioritu než IRQ X.
  - Probíhající ISR nemůže být přerušena:

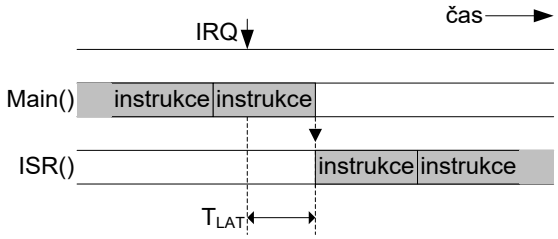


  - Probíhající ISR může být přerušena – nesting:



14.1.8 Latence přerušení

- = doba od vzniku IRQ do vstupu do ISR a vykonáním první instrukce obsluhy
- Doba latence TLAT:



- T_{LAT} není konstantní:

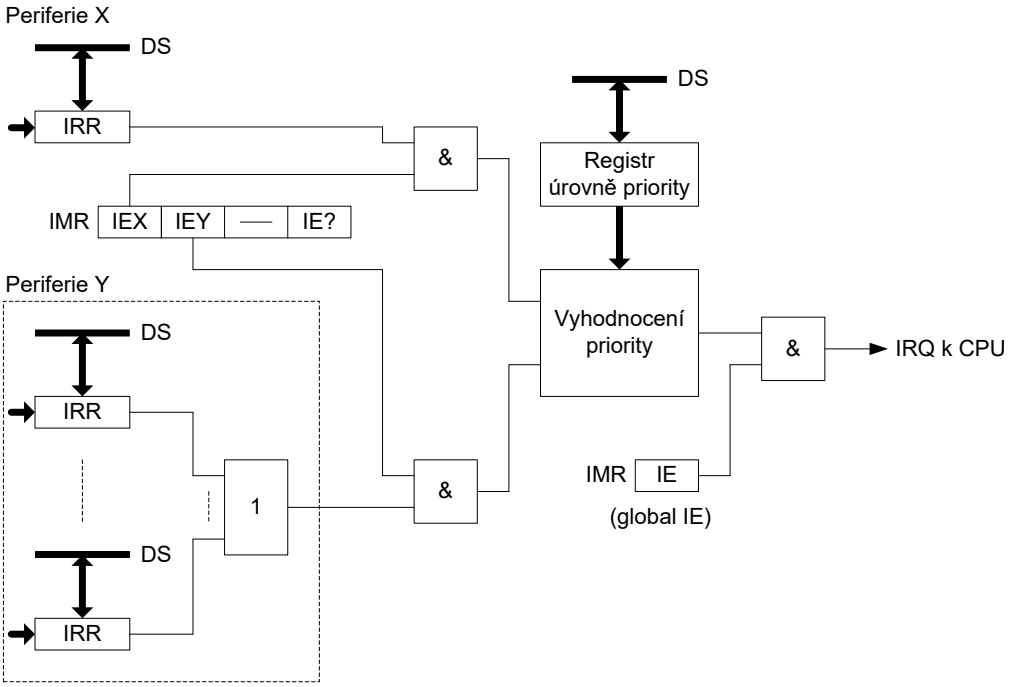
  $T_{LAT} = T_{INST} + T_{ZPRAC}$

  - T_{INST} – doba trvání instrukce (musí se dokončit, různé instrukce trvají různou dobu)
  - T_{ZPRAC} – odskok na IV a dále na ISR
  - T_{LAT} může být poměrně dlouhá, pokud se právě vykonává jiné přerušení s vyšší prioritou. Musí se počkat, než právě vykonávané ISR dokončí. Pak závisí ne délce kódu v právě vykonávané ISR.
- Může být problém např. při realizaci přesného časování na základě ISR + časovač.

14.1.9 Logika přerušení u MCU

- Výhoda: vše je na čipu (řadič, periferie,…)
- **Maskovatelná přerušení** – programátor může zvolit, bude-li přerušení obsluhovat či nikoli
  - **I**nterrupt **M**ask **R**egister (**IMR**) – obsahuje bity (= maska), kterými se povoluje/zakazuje dané přerušení
    - bit = 1 – přerušení se vyvolá
    - bit = 0 – přerušení se nevyvolá
  - **G**lobal **I**nterrupt **E**nable (bit; zpravidla označen **IE**) – maska pro všechna přerušení (**globální zákaz/povolení všech přerušení**)
- **Nemaskovatelné přerušení**
  - Non-maskable Interrupt (NMI).
  - Vždy se obslouží, nelze jej vypnout (např. PC – chyba parity dat v RAM).
  - AVR nemá
- **I**nterrupt **R**equest **R**egister (**IRR)** nebo **I**nterrupt **F**lag **R**egister (**IFR**) = registr, který obsahuje příznaky (bity), každý bit představuje žádost o přerušení, bit je přiveden na logiku přerušení.
  - bit = 1 $\rightarrow$ = IRQ

- bit = 0 → není žádost o přerušení



- Periferie Y – více zdrojů přerušení pod jedním vektorem – který zdroj vyvolal přerušení, musí zjistit programátor programově čtením IRR jednotlivých potenciálních zdrojů přerušení.

14.2 Přerušení na AVR

- U řady ATmega může být až 34 zdrojů přerušení.
- Adresy vektorů přerušení jsou uspořádány za sebou od adresy 0x0000 paměti programu. Počet vektorů přerušení a jejich umístění v tabulce závisí na konkrétním typu AVR a počtu periferií, které obsahuje.
- Priorita přerušení je určena pořadím v tabulce – nejvyšší má RESET.

Tabulka 20 Tabulka vektorů přerušení u ATmega32

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

- Při psaní programů v assembleru a používání přerušení je nutno na daný vektor umístit instrukci skoku do rutiny pro obsluhu přerušení.
- První instrukcí vůbec (ta je umístěná na vektoru RESET) by měl být odskok na **hlavní program**, který **začíná až za tabulkou přerušení**. V případě, že přerušení nevyužívám, je možné začít psát program již od adresy 0x0000.

Příklad kódu v ASM (využívá všechna přerušení):

```
Address   Labels    Code                    Comments
$000                jmp   RESET             ; Reset Handler
$002                jmp   EXT_INT0          ; IRQ0 Handler
$004                jmp   EXT_INT1          ; IRQ1 Handler
$006                jmp   EXT_INT2          ; IRQ2 Handler
$008                jmp   TIM2_COMP         ; Timer2 Compare Handler
$00A                jmp   TIM2_OVF          ; Timer2 Overflow Handler
$00C                jmp   TIM1_CAPT         ; Timer1 Capture Handler
$00E                jmp   TIM1_COMPA        ; Timer1 CompareA Handler
$010                jmp   TIM1_COMPB        ; Timer1 CompareB Handler
$012                jmp   TIM1_OVF          ; Timer1 Overflow Handler
$014                jmp   TIM0_COMP         ; Timer0 Compare Handler
$016                jmp   TIM0_OVF          ; Timer0 Overflow Handler
$018                jmp   SPI_STC           ; SPI Transfer Complete Handler
$01A                jmp   USART_RXC         ; USART RX Complete Handler
$01C                jmp   USART_UDRE        ; UDR Empty Handler
$01E                jmp   USART_TXC         ; USART TX Complete Handler
$020                jmp   ADC               ; ADC Conversion Complete Handler
$022                jmp   EE_RDY            ; EEPROM Ready Handler
$024                jmp   ANA_COMP          ; Analog Comparator Handler
$026                jmp   TWI               ; Two-wire Serial Interface Handler
$028                jmp   SPM_RDY           ; Store Program Memory Ready Handler
;
$02A      RESET:    ldi   r16,high(RAMEND)  ; Main program start
$02B                out   SPH,r16           ; Set Stack Pointer to top of RAM
$02C                ldi   r16,low(RAMEND)
$02D                out   SPL,r16
$02E                sei                     ; Enable interrupts
$02F                <instr>  xxx
...       ...       ...
```

- **Globální povolení** (či zákaz) přerušení se provede nastavením bitu **I** v registru **SREG**.

Bit	7	6	5	4	3	2	1	0	
	I	**T**	**H**	**S**	**V**	**N**	**Z**	**C**	**SREG**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

I = 1 ⇒ přerušení povolena

I = 0 ⇒ přerušení zakázána

Nastavení bitu `I` v registru **SREG** se provede instrukcí `sei` (I=1) nebo `cli` (I=0).

- **Každá periferie má ve svých registrech „Interrupt Mask Register“,** ve kterém lze povolit/zakázat přerušení od dané periferie (bity končící na –IE).

  Např. IMR časovače T1

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Dále má každá periferie **registr s příznaky přerušení** („**Interrupt Flag Register**“)

  Např. IFR časovače T1

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

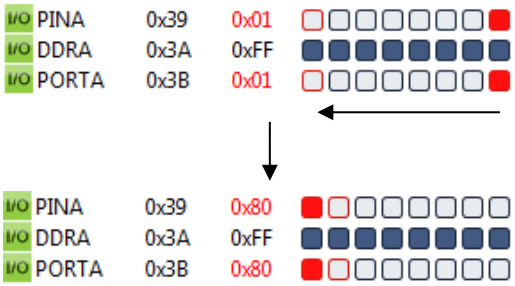
  - Příznaky jsou nastaveny do 1 při aktivní žádosti o přerušení.
  - **Příznak se automaticky nuluje po vstupu do přerušovací rutiny.**
  - **Ručně (v programu) lze příznak vynulovat zápisem 1 do příslušného bitu !**

- **Mechanismus přerušení**
  - Jsou-li přerušení globálně povolena a dojde-li ke splnění podmínky, která vede k vygenerování žádosti o přerušení (např. přetečení časovače), dokončí se právě vykonávaná instrukce, PC se zvýší o délku aktuálně zpracovávané instrukce (např. o 1 u instrukce o délce 1 slova) a dojde k jeho uložení do zásobníku.
  - Poté dojde ke skoku na vektor přerušení. Odtud se obvykle odskočí do rutiny pro obsluhu přerušení.
  - **Hardware MCU automaticky zakáže globálně přerušení (I = 0) !**
  - Po skončení rutiny, která musí být zakončena instrukcí `reti`, dojde k automatickému vyzvednutí návratové adresy ze zásobníku – naplní se jí PC a program pokračuje instrukcí, která následuje za místem odskoku z hlavního programu.
  - **Hardware MCU automaticky povolí globálně přerušení (I = 1) !** (zajišťuje instrukce `reti`)

- **Důležité informace o prioritě přerušení, vnořená přerušení**
  - Sejde-li se naráz více přerušení, je vykonáno to s největší prioritou. Další přerušení se vykonají postupně po vykonání předešlých opět dle priority.
  - **AVR nepodporuje vnořená přerušení - aktuálně vykonávané přerušení nemůže být přerušeno již žádným jiným, a to ani přerušením s vyšší prioritou !**
    - Protože po vstupu do přerušovací rutiny hardware MCU globálně zakáže všechna přerušení (I = 0).
    - To je rozdíl oproti 8051 !
  - **Jak umožnit nesting na AVR, tj. aby právě vykonávané přerušení mohlo být přerušeno jiným?**
    - Jedině povolením globálního přerušení (SREG bit I =1) v obsluze ISR → nevýhoda → při další žádosti o obsluhu přerušení bude aktuální ISR přerušena bez ohledu na prioritu nově příchozího přerušení.
  - Závěr je ten, že priorita přerušení na AVR je rozhodující pouze v okamžiku, kdy se sejde více žádostí o přerušení v jeden okamžik.
- **Doplňkové informace**
  - Je možné volit umístění tabulky vektoru přerušení. Tabulka může být na začátku paměti programu od adresy 0x0000 nebo může být přesunuta na začátek oblasti pro Bootloader, pokud je Bootloader použit.
  - Odezva na přerušení trvá v nejnepříznivějším případě 9 instrukčních cyklů:
    - dokončení právě prováděné instrukce (nejdelší instrukce se vykonává 3 IC),
    - skok na vektor přerušení (3 IC) a z něj skok do přerušovací rutiny (3 IC) – to platí při použití instrukce absolutního skoku `jmp`,
    - návrat z přerušení trvá 4 IC (vykonání instrukce `reti`).

14.2.1 Obsluha přerušení v ASM

Příklad – Běžící had využívající přerušení od časovače 1 (ATmega32)




```
;definice pripojeni LED k MCU
.equ   LED_PORT = PORTA
.equ   LED_DDR = DDRA

.dseg
.org 0x0060          ; prvni volna adresa v RAM
had: .DB      1      ;definice promenne had

.cseg
.org 0x0000          jmp start           ;reset vektor
.org 0x0012          jmp ISR_TIMER1_OVF  ;ISR vektor pro obsluhu preteceni TIMER1

.org 0x0030 ;hlavni program zacina az za IVT
start:               ldi r16,HIGH(RAMEND) ;nastaveni SP
                     out SPH,r16
                     ldi r16,LOW(RAMEND)
                     out SPL,r16

                     ldi r16,0xFF
                     out LED_DDR,r16      ;DDRA = 0xFF
                     ldi r16,0x01
                     sts had,r16          ;had = 1

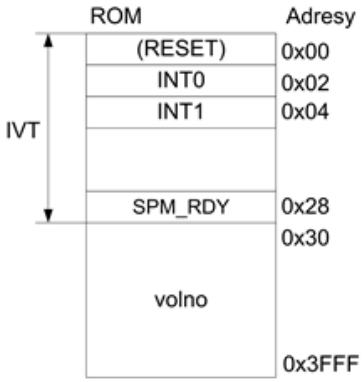
                     ldi r16,1<<TOIE1     ;zapnout preruseni pri preteceni casovace
                     out TIMSK,r16
                     sei                  ;globalni povoleni preruseni
                     ;ldi r16,1<<CS12           ;spusteni TIMER1 - frekvence = Fclk/256
                     ldi r16,1<<CS10            ;spusteni TIMER1 - frekvence = Fclk/1
                     out TCCR1B,r16

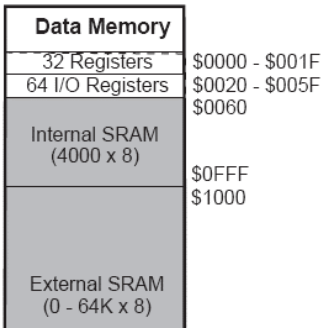
hop:                 ;zde by mohl byt nejaky dalsi vykonny kod
                     rjmp hop      ;nekonecna smycka

ISR_TIMER1_OVF:      push r16      ;zaloha obsahu r16
                     in r16,SREG
                     push r16      ;zaloha SREG

                     ;algoritmus hada
                     lds r16,had           ;nacteni promenne had do r16
                     out LED_PORT,r16      ;zapis na port s LED
                     lsl r16               ;bitovy posun doleva
                     brcc had_konec        ;skace kdyz Carry = 0
                     ldi r16,0x01          ;had jede od zacatku
        had_konec:   sts had,r16           ;ulozeni r16 do promenne had

                     pop r16
                     out SREG,r16          ;obnoveni obsahu SREG
                     pop r16               ;obnoveni obsahu r16
                     reti
```





Záloha registrů, používaných v ISR, na zásobník.

Obnovení registrů, používaných v ISR, ze zásobníku.

14.2.2 Obsluha přerušení v CodeVision

- **ISR se zapisuje speciálním způsobem** – funkcí, která začíná klíčovým slovem i**nterrupt**, následovaným **číslem vektoru přerušení** v hranatých závorkách [ ].

```
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
      // kod v obluze preruseni
}
```

  - Číslování je od jedničky, v kódu se pro přehlednost místo čísla vektoru odvoláváme na název vektoru přerušení z datasheetu daného MCU.
    - Akorát mezery v názvu vektoru v datasheetu vyměníme za podtržítko.

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow

  - Název funkce může být jakýkoliv (vhodné pojmenovat podle zdroje přerušení).
  - Funkce nepřebírá ani nevrací žádné parametry.
- Globální povolení/zakázání obsluhy přerušení – voláním instrukcí
  - `#asm("sei")` – nastaví bit I = 1(globální povolení obsluhy přerušení)
  - `#asm("cli")` – nastaví bit I = 0 (globální zákaz obsluhy přerušení)
- **Předávání hodnot do ISR** – přes globální proměnné
  - Globální proměnné používané uvnitř funkce obsluhy přerušení by měly být v definici označeny jako `volatile` (kompilátor nebude optimalizovat přístup k těmto proměnným).

Příklad – Běžící had využívající přerušení od časovače 1 (ATmega32)

```
#include <mega32.h>
#include <mega32bitdef.h>

volatile unsigned char had; // stav 8 LED, bit v 0 - LED nesviti, bit v 1 - LED sviti

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
      PORTA = had;
      if (had == 0x80) had = 0x01;
      else had <<= 1;
}

void main(void)
{
      had = 1;                    // sviti prvni LED
      DDRA = 0xFF;                // 8 LED pripojeno k portu A
      //TCCR1B = (1 << CS10);     // spusteni TIMER1 - frekvence = Fclk/1
      TCCR1B = (1 << CS12);       // spusteni TIMER1 - frekvence = Fclk/256
      TIMSK = (1 << TOIE1);       // zapnout preruseni pri preteceni casovace
      #asm("sei")                 // povolit globalne preruseni

      while (1) // nekonecna smycka
      {
            // nejaky dalsi vykonny kod
      }

}
```

- Při vstupu do ISR se do zásobníku Data Stack **automaticky ukládají použité registry a SREG**. Při skončení funkce se opět vyzvedávají. Funkce je automaticky ukončená instrukcí `reti`.
- Lze využít CodeWizard pro vygenerování prázdné funkce obsluhy přerušení.

Jak to vypadá po přeložení ?
(Velikost programu 125 slov (250 Bajtů), optimalizace „size, maximal“, ve výpisu chybí statup-code – cca 150 Bajtů)

```
.DSEG
      .ORG 0x260
;____       1 #include <mega32.h>
;____       2 #include <mega32bitdef.h>
;____       3
;____       4 volatile unsigned char had;
_had:
      .BYTE 0x1
;____       5
;____       6 // Timer 1 overflow interrupt service routine
;____       7 interrupt [TIM1_OVF] void timer1_ovf_isr(void)
;____       8 {
```

Proměnná `had` je v RAM na drese 0x260.

```
        .CSEG
_timer1_ovf_isr:
        ST   -Y,R26
        ST   -Y,R30
        IN   R30,SREG
        ST   -Y,R30
;___         9 // Place your code here
;___        10      PORTA = had;
        LDS  R30,_had
        OUT  0x1B,R30
;___        11      if (had == 0x80) had = 0x01;
        LDS  R26,_had
        CPI  R26,LOW(0x80)
        BRNE _0x3
        LDI  R30,LOW(1)
        RJMP _0x9
;___        12      else  had <<= 1;
_0x3:
        LDS  R30,_had
        LSL  R30
_0x9:
        STS  _had,R30
;___        13 }
        LD   R30,Y+
        OUT  SREG,R30
        LD   R30,Y+
        LD   R26,Y+
        RETI
;___        14
;___        15 void main(void)
;___        16 {
_main:
;___        17      had = 1;
        LDI  R30,LOW(1)
        STS  _had,R30
;___        18      DDRA = 0xFF;
        LDI  R30,LOW(255)
        OUT  0x1A,R30
;___        19      TCCR1B = (1 << CS12);   //   zapnuti   T1   -   frekvence   =
Fclk/256
        LDI  R30,LOW(4)
        OUT  0x2E,R30
;___        20      TIMSK = (1 << TOIE1);   // zapnout preruseni pri preteceni
        OUT  0x39,R30
;___        21      #asm("sei")             // povolit globalne preruseni
        sei
;___        22
;___        23      while (1) ;             // nekonecna smycka - ceka se na
preruseni
_0x5:
        RJMP _0x5
;___        24 }
_0x8:
        RJMP _0x8
```

Uložení registrů, které se budou v obsluze přerušení používat, na zásobník (DATA_STACK). Také uložení SREG na zásobník.

Vyzvednutí původního obsahu použitých registrů ze zásobníku.

14.2.3 Obsluha přerušení v AVR GCC

- Na začátek programu je třeba vložit: `#include <avr/interrupt.h>`
- **ISR se zapisuje speciálním způsobem:**

```
ISR(TIMER1_OVF_vect)
{
        // kod v obluze preruseni
}
```

- **Definice hlavičky ISR pomocí makra**: `#define ISR(vector, attributes)`
  - Makro je definované v souboru `avr/interrupt.h`.
  - `vector` – číslo vektoru přerušení, místo čísla používáme zásadně symbol, který se tvoří takto:
    - Podíváme se do tabulky vektorů přerušení v datasheetu AVR na název vektoru.

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow

    - Mezery v názvu vektoru v datasheetu nahradíme podtržítkem _
    - K názvu vektoru doplníme koncovku `_vect`
    - Např. `ISR(TIMER1_OVF_vect)`
  - `attributes` – nepovinný parametr, kterým se upravuje chování funkce (ovlivňuje se tím výsledný vygenerovaný kód rutiny pro obsluhu přerušení)
    - `ISR_BLOCK` – stejné jako volání makra ISR bez `attributes`. Po vstupu do ISR zůstane obsluha dalších ISR globálně zakázána (I=0).
    - `ISR_NOBLOCK` – umožňuje vnořená přerušení. Po vstupu do ISR se globálně povolí přerušení (I=1).

        - `ISR_NAKED` – po vstupu do ISR se nezálohuje SREG a na konci ISR se nevolá `reti`. Programátor musí tyto části do ISR doplnit ručně.
- Podrobnosti viz http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html#gaa87c0c624b62f40a17539be6946c3e26
- **Globální povolení/zakázání obsluhy přerušení** – voláním maker definovaných v `avr/interrupt.h`.
    - `sei()` – nastaví bit I v SREG na 1 (globální povolení obsluhy přerušení)
    - `cli()` – nastaví bit I v SREG na 0 (globální zákaz obsluhy přerušení)
- **Předávání hodnot do ISR** – přes globální proměnné
    - **Globální proměnné používané uvnitř funkce obsluhy přerušení musí být v definici označeny jako** `volatile` (kompilátor nebude optimalizovat přístup k těmto proměnným a program bude fungovat správně).

Příklad – Běžící had využívající přerušení od časovače 1 (ATmega32)

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile unsigned char had;// stav 8 LED, bit v 0 - LED nesviti, bit v 1 - LED sviti

// Obsluha preteceni casovace
ISR(TIMER1_OVF_vect)
{
	PORTA = had;
	if (had == 0x80) had = 0x01;
	else had <<= 1;
}

int main(void)
{
	had = 1;			// sviti prvni LED
	DDRA = 0xFF;			// 8 LED pripojeno k portu A
	TIMSK = (1 << TOIE1);		// zapnout preruseni pri preteceni casovace
	sei();				// povolit globalne preruseni
	//TCCR1B = (1 << CS10);		// spusteni TIMER1 - frekvence = Fclk/1
	TCCR1B = (1 << CS12);		// spusteni TIMER1 - frekvence = Fclk/256

	while (1) // nekonecna smycka
	{
		// nejaky dalsi vykonny kod
	}
}
```

- Pokud není atributy makra **ISR()** stanoveno jinak, tak při vstupu do ISR se do zásobníku **automaticky ukládají použité registry a SREG**. Při skončení funkce se opět vyzvedávají. Funkce je automaticky ukončená instrukcí `reti`. Programátor se pak o toto nemusí starat, probíhá to automaticky.

Příklad – Blikání LED na Arduino Leonardo (ATmega32U4)
Nastavení časovače přes registry.

```
#include "Arduino.h"
#include <avr/io.h>

#define LED_PIN     13

uint8_t stavLed = LOW;     // zadany stav LED, globalni promenna, aby byla viditelna v
preruseni

void setup()
{
       // nastaveni smeru na pinu
       digitalWrite(LED_PIN,stavLed);
       pinMode(LED_PIN,OUTPUT);

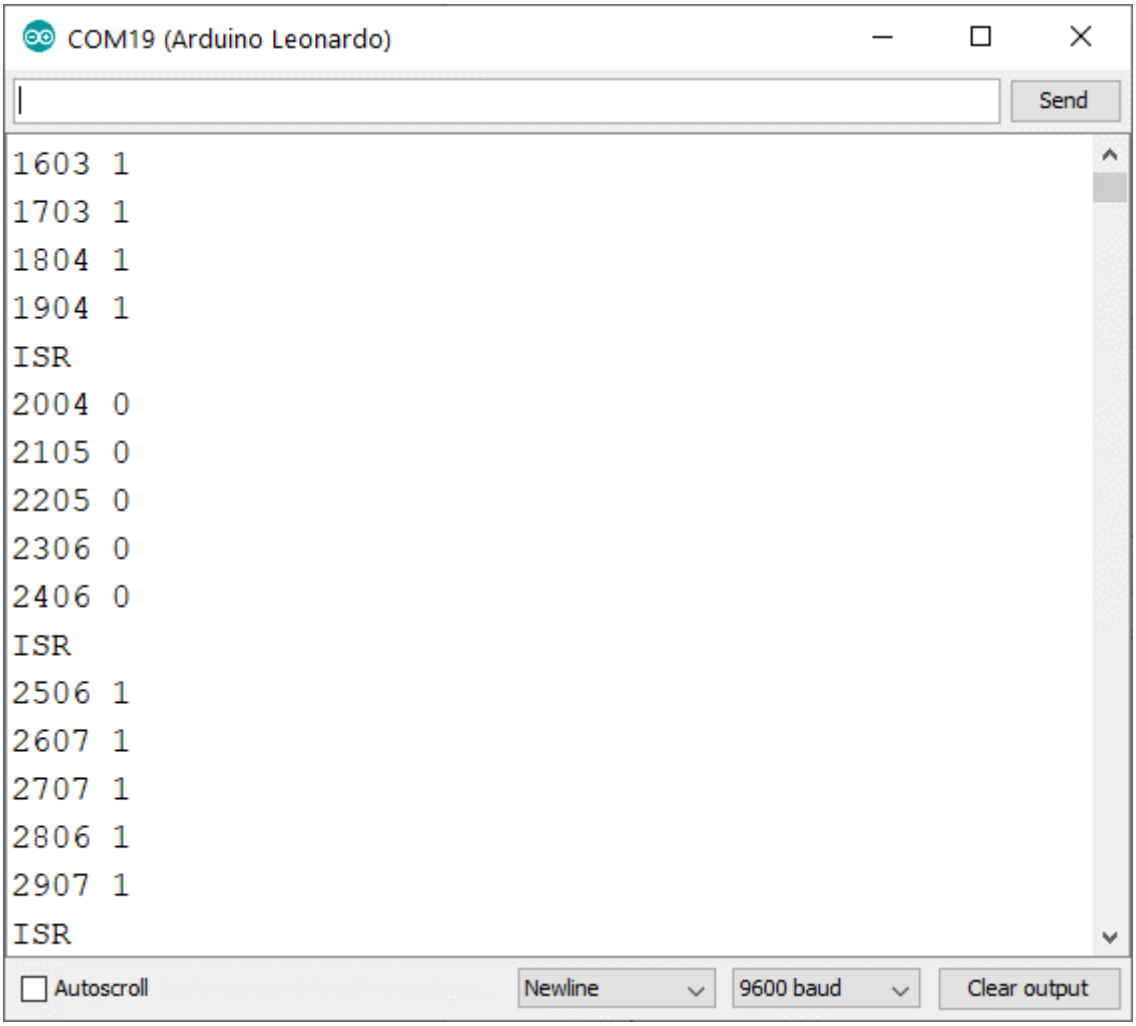
       // nastaveni Serial pro vypisovani ladicich hlasek pres USB
       Serial.begin(9600);  // baudrate 9600 Bd

       // nastaveni casovace pro generovani intervalu blikani LED 500 ms
       // Phase and Frequency Correct PMW mode (mode 8)
       // Timer/Counter 1
       TCCR1A = 0;
       TCCR1B = (1<<WGM13) | (1<<CS11) | (1<<CS10);    // preddelicka N = 64
       ICR1 = 62500; // urcuje TOP pro Fpwm = 2 Hz (500 ms interval blikani)
       //ICR1 = 31250;      // urcuje TOP pro Fpwm = 4 Hz (250 ms interval blikani)
       TIMSK1 = (1<<TOIE1); // timer1 overflow interrupt enable
       sei(); // globalne povolit preruseni (neni treba, Arduinem je jiz povoleno)
}

void loop(void)
{
       // demonstrace nezavislosti kodu v loop() na obsluze preruseni
       Serial.print(millis());
       Serial.print(" ");
       Serial.println(stavLed);
       delay(100); // pro zpomaleni vypisovani
}

// obsluha preruseni pri preteceni casovace Timer/Counter 1
ISR(TIMER1_OVF_vect)
{
       stavLed = !stavLed;  // zmena zadaneho stavu LED
       digitalWrite(LED_PIN,stavLed);
       Serial.println("ISR");
}
```

Výstup do terminálu na PC:



Příklad – Blikání LED na Arduino Leonardo (ATmega32U4)

Nastavení časovače 1 pomocí knihovny TimerOne.

```
/*
Blikání LED v přerušení od časovače, každých 500 ms změna stavu LED.
Použita knihovna TimerOne pro nastavení časovače.
Kód je určen pro Arduino Nano/UNO/Leonardo/MEGA2560.
*/

#include "Arduino.h"
#include <avr/io.h>
#include <TimerOne.h>

// pripojeni LED k Arduinu
#define LED_PIN     13

// globalni promenne
uint8_t stavLed = LOW;     // zadany stav LED, globalni promenna, aby byla viditelna v
preruseni

// deklarace funkci
void blinkLed(void);

void setup()
{
	// nastaveni smeru na pinu
	digitalWrite(LED_PIN,stavLed);
	pinMode(LED_PIN,OUTPUT);

	// nastaveni Serial pro vypisovani ladicich hlasek pres USB
	Serial.begin(9600); // baudrate 9600 Bd

	// nastaveni casovace pro generovani intervalu blikani LED 500 ms

	Timer1.attachInterrupt(blinkLed); // pripojeni callback funkce, bude se vyvolavat
pri preruseni od casovace, po odmereni casoveho intervalu

	Timer1.initialize(500000); // periodicke odmerovani intervalu 500 000 us, casovac
se hned spusti

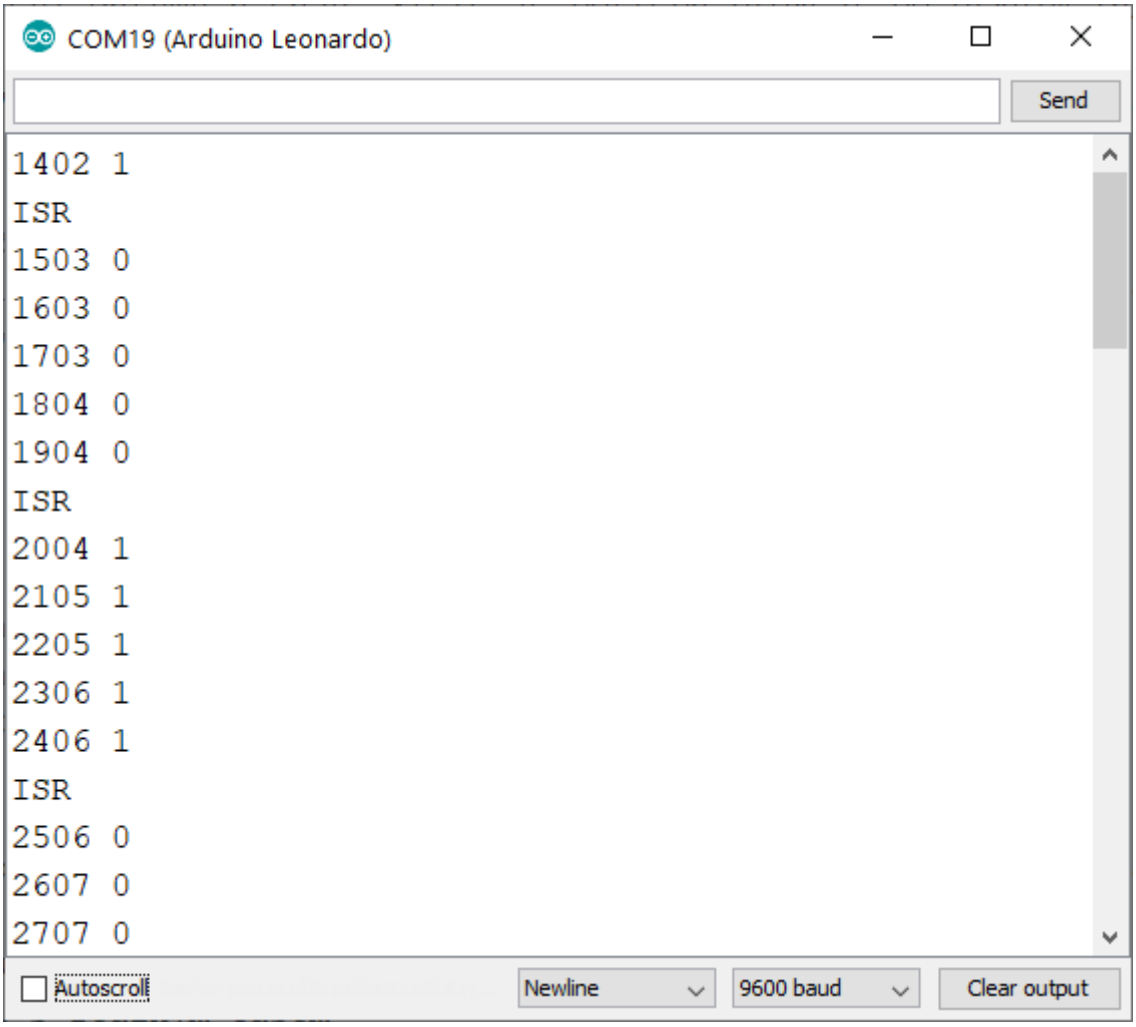
	// nebo lze rovnou i takto
	//Timer1.attachInterrupt(blinkLed,500000); // periodicke odmerovani intervalu 500
000 us, casovac se hned spusti

}

void loop(void)
{
	// demonstrace nezavislosti kodu v loop() na obsluze preruseni
	Serial.print(millis());
	Serial.print(" ");
	Serial.println(stavLed);
	delay(100); // pro zpomaleni vypisovani
}

// callback funkce volana z ISR Timer1
void blinkLed(void)
{
	stavLed = !stavLed; // zmena zadaneho stavu LED
	digitalWrite(LED_PIN,stavLed);
	Serial.println("ISR");
}
```

Výstup do terminálu na PC:

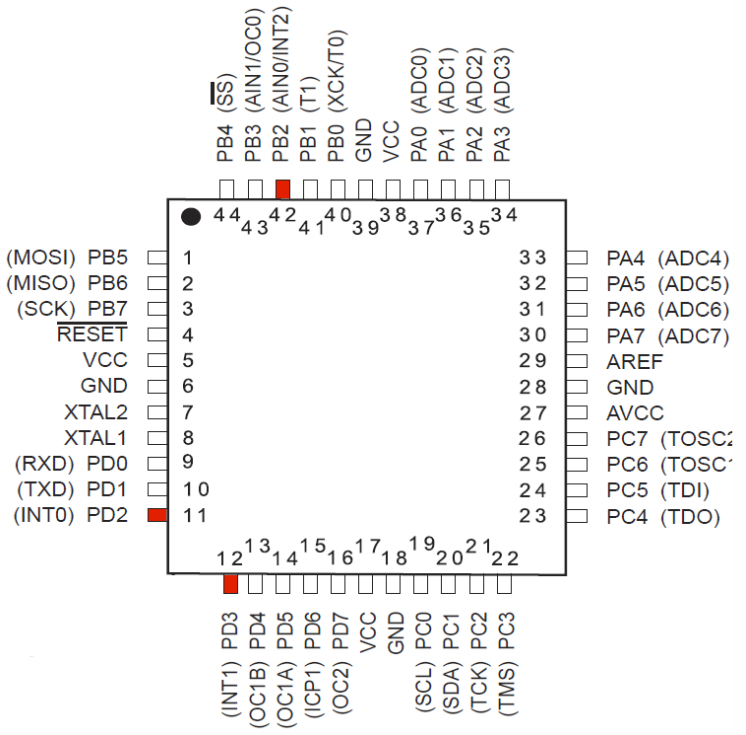


14.3 Externí přerušení

- Některé piny na MCU mají alternativní funkci „externí přerušení“.
- Externí přerušení = je vyvoláno přerušení pokud se na pinu objeví dohodnutá událost.
- Na ATmega32 máme k dispozici 3 externí přerušení, označené INT0, INT1 a INT2.

Tabulka 21 Seznam pinů pro externí přerušení na ATmega32

Číslo pinu	Alternativní funkce
PD2	INT0
PD3	INT1
PB2	INT2



Obrázek 135 Rozmístění pinů externího přerušení na ATmega32 v pouzdru TQFP44

- Počet externích přerušení závisí na typu AVR
  - např. u ATmega32 jsou 3, ATmega128 jich má 8
  - u některých AVR je možné zapnout externí přerušení na každém pinu vybraných portů
- Názvy ovládacích registrů externích přerušení, příznakových registrů a umístění jednotlivých bitů v nich se liší dle typu AVR. Proto budou na následujících řádcích popsány registry a použití externího přerušení pouze u typu ATmega32. Pro ostatní typy to bude analogické. Vždy je nutno vycházet z datasheetu konkrétního typu AVR.

- Externí přerušení může být vyvoláno
  - nízkou úrovní na pinu INTx
  - vzestupnou hranou na pinu INTx
  - sestupnou hranou na pinu INTx
  - jakoukoliv změnou úrovně na pinu INTx
  - k nastavení slouží registr **MCUCR** (pro **INT0** a **INT1**)

Bit	7	6	5	4	3	2	1	0	
	SE	**SM2**	**SM1**	**SM0**	**ISC11**	**ISC10**	**ISC01**	**ISC00**	**MCUCR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

INT0

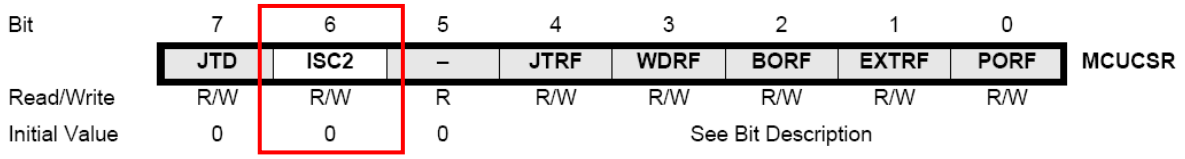
ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

INT1

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

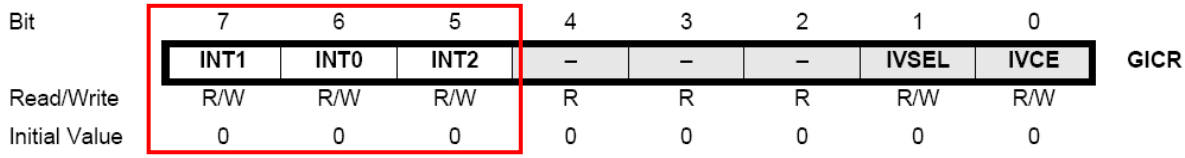
  - **INT2** může reagovat pouze na vzestupnou nebo sestupnou hranu, k nastavení slouží bit ISC2 v registru **MCUCSR**
    ISC2 = 1 ⇒ vzestupná hrana
    ISC2 = 0 ⇒ sestupná hrana

    Zároveň se doporučuje, aby se před změnou hodnoty bitu ISC2 zakázalo přerušení INT2, poté se provedla změna bitu ISC2, poté se vynuloval příznak přerušení a nakonec opět povolilo přerušení INT2. Jinak by při změně ISC2 mohlo dojít k nechtěnému vygenerování přerušení.



Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

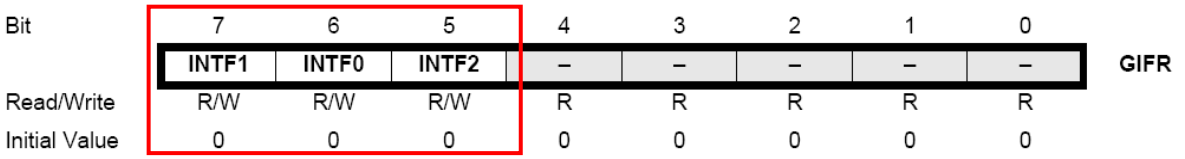
- Poznámky
  - Pokud je nastaveno přerušení na nízkou úroveň, vyvolává se přerušení stále dokuď na pinu trvá nízká úroveň.
  - Je-li nastaveno přerušení na nízkou úroveň (log. 0 na pinu INTx), musí na pinu setrvat minimálně po dobu než se dokončí aktuálně zpracovávaná instrukce, aby došlo k detekci tohoto stavu na pinu.
  - Aby byla změna úrovně na pinu INT0 a INT1 detekovatelná, musí mít puls délku minimálně rovnou délce jednoho cyklu oscilátoru MCU.
  - Změny na INT2 jsou registrovány asynchronně. Proto, aby byly detekovatelné, musí být puls široký minimálně 50 ns.
- Externí přerušení se **povolují/zakazují** v registru **GICR**
  - Přerušení je povoleno je-li příslušný bit nastaven do 1
  - Přerušení je zakázáno je-li příslušný bit nastaven do 0



Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  -

- Registr s **příznaky** externích přerušení se nazývá **GIFR**



Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

  - Jakmile dojde k detekci přerušení, nastaví se konkrétní příznak **INTFx** do 1.
  - Je-li povoleno konkrétní externí přerušení a pokudé jsou globálně povolena přerušení (I = 1), dojde ke skoku na vektor daného přerušení a poté k odskoku do rutiny pro obsluhu přerušení.
  - Příznak **INTFx** se **automaticky nuluje** po vstupu do rutiny na obsluhu přerušení.
  - Alternativně (pokud nejsou použita přerušení a program periodicky testuje příznak **INTx**) může být příznak ručně smazán **zápisem jedničky (!)** do příslušného bitu **INTFx**.

Příklad:
Napište program, který bude počítat počet náběžných hran logického signálu, přivedeného na vstup INT0. MCU ATmega32.

Kompilátor CodeVision:

```
#include <mega32.h>
#include <mega32bitdef.h>     // obsahuje cisla bitu v IO registrech

volatile unsigned char pocetHran = 0;

// Obsluha externiho preruseni INT0
interrupt [EXT_INT0] void ext_int0_isr(void)
{
      pocetHran++;
}

// hlavni program
void main(void)
{
      // Nastaveni externiho preruseni INT0
      // INT0: On
      // INT0 Mode: Rising Edge
      // INT1: Off
      // INT2: Off
      MCUCR = (1 << ISC01) | (1 << ISC00);
      GICR = (1 << INT0);

      // Globalni povoleni preruseni (bit I v SREG nastavi do 1)
      #asm("sei")

      // nekonecna smycka
   while (1)
   {
      if (pocetHran == 10)      // kazdou 10 hranu neco udelej
      {
            pocetHran = 0;      // mazani pocitadla hran
            // tady by mohl byt nejaky dalsi kod, napr vypis na LCD
      }
   };
}
```

Kompilátor AVR GCC:

```
#include <avr/io.h>
#include <avr/interrupt.h>

volatile unsigned char pocetHran = 0;    //  nejaka  globalni  promenna,  aktualizovana
asynchronne v preruseni

/* Funkce je automaticky zavolána při přerušení */
ISR(INT0_vect)
{
	pocetHran++;  // aktualizace promenne pocitadlo
}


int main(void)
{
	// nastaveni a povoleni externiho preruseni INT0
	MCUCR = (1<<ISC00) | (1<<ISC01);  // reakce na vzestupnou hranu na pinu INT0
	GICR = 1<<INT0;      // povoleni externiho preruseni INT0

	sei(); // globani povoleni preruseni (bit I v SREG nastavi do 1)

	while(1) // nekoncecna smycka
	{
		if (pocetHran == 10)// kazdou 10 hranu neco udelej
		{
			pocetHran = 0;       // mazani pocitadla hran
			// tady by mohl byt nejaky dalsi kod, napr vypis na LCD
		}
	}
}
```

V ASM:

```
.include "m32def.inc"

     .cseg

     .org 0x0000
     rjmp reset

     .org INT0addr
     rjmp INT0_ISR

reset:    // nastaveni zasobniku
     ldi  r16, LOW(RAMEND)
     out  SPL, r16
     ldi  r16, HIGH(RAMEND)
     out  SPH, r16
     // konfigurace INT0
     ldi  r16, (1 << ISC01) | (1 << ISC00)
     out  MCUCR, r16
     ldi  r16, (1 << INT0)
     out  GICR, r16
     ldi r17,0      // počet hran = 0
     sei            // globální povolení přerušení

loop:
     cpi r17,10     // v r17 je počet hran
     brne loop      // if (počet hran == 10)
     ldi r17,0      // počet hran = 0
     rjmp loop      // while(1)

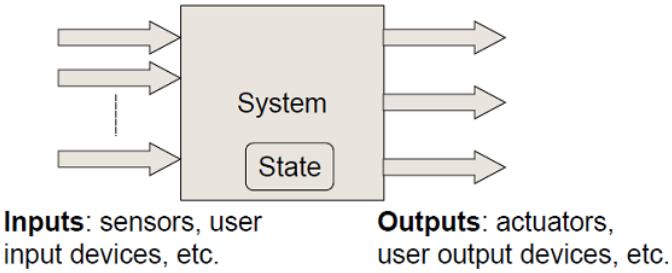
INT0_ISR:
     push r10       // zaloha r10
     in   r10, SREG
     push r10       // zaloha SREG
     inc  r17       // pocetHran = pocetHran + 1
     pop  r10       // obnova SREG
     out  SREG,r10
     pop  r10       // obnova r10
     reti
```

- Do cvičení:
  - odsimulovat (IRQ vyvolat zaškrtnutím INTF0), všimnout si automatického `cli` a nulování INTF0.
  - Dodělat ještě INT1 a vyzkoušet vnořené přerušení, doplnit `sei` do ISR a prokrokovat.

15 Programovací techniky v embedded aplikacích

15.1 Základní pojmy

15.1.1 Systém



- mapuje vstupy na výstupy
- stav – paměť systému, jak mapování probíhá
- reakční doba – doba mezi stimulem a reakcí na něj
- deterministický systém – pro každý stav a vstup lze určit jednoznačný výstup a další stav

15.1.2 Reálný čas

- Znamená, že je zaručena odezva systému v daném čase (čas – dle aplikace) = reakční doba omezena shora („správná odpověď ve špatný čas je špatná odpověď“).
- Všechna data zpracovat v jedné periodě vzorkování (načtení vstupů, regulace, aktivace výstupů...).
- U běžných MCU vzorkovací frekvence max. 10ky kHz.

15.1.3 Událost

- Jakákoli příhoda, která změní nesekvenčně PC (programový čítač) – změna postupu chování.
- Dle synchronnosti s taktem CPU
  - Synchronní – výskyt v predikovatelných časech (synchronizovaná s taktem CPU).
  - Asynchronní – vysyktující se v nahodilých časech (nesynchronizovaná s taktem CPU).
- Dle periodicity:
  - Periodická – opakující se v pravidelných intervalech.
  - Aperiodická – opakující se, ale v nahodilých intervalech.
  - Sporadická – vyskytující se občas, velmi nepravidelné intervaly.

- Příklady událostí:

	Periodická	**Aperiodická**	**Sporadická**
Synchronní	Opakovaně se vykonávající kód Procesy spouštěné v taktu interních hodin	Instrukce skoku Garbage collection	Zotavení z chyb Systémová volání
Asynchronní	Opakované přerušení, taktované hodinami	Přerušení s proměnnou periodou	Externí přerušení

15.2 Základní techniky tvorby kódu

15.2.1 Polling

- polling = opakovaný dotaz na stav
- Lze použít i u HW s přerušením – přerušení vypnuto, místo tohoto periodické testování stavu příznaku „žádost o přerušení“ v IRR (Interrupt Request Register)

```
int main(void)
{
  setup();        // inicializace periferii
  while (1)       // nekonecna smycka
  {
      if (tlacitko stisknuto)
      {
            // obsluha stisku tlacitka
      }

      if (UCSRA & (1 << RXC))  // příjem bajtu po USART
      {
        neco = UDR; // vyčtení bajtu z přijímacího registru
        // zpracování dat
      }

      while (!(TIFR & (1 << TOVF1))); // čekání na přetečení časovače

      TIFR |= 1 << TOVF1; // příznak nutno nulovat zápisem 1
  }
  return 0;
}
```

- **Výhody:** jednoduché
- **Nevýhody**: dlouhé zpracování jedné události blokuje další

15.2.2 Přerušení

- Použijeme, pokud potřebujeme:
  - obsloužit kód s vysokou prioritou
  - rychle reagovat na událost
  - periodicky a bez výrazného zpoždění reagovat na událost
- **Příklad**:
  - **Nejprve řešení s nekonečnou smyčkou a bez přerušení**

```
int main(void)
{
  setup();  // inicializace periferií
  while(1)  // hlavní nekonečná smyčka programu
  {
      // čtení dat ze senzorů
      // zpracování dat a ...
      // ... výpočet nových hodnot procesu
      // nastavení akčních členů
  }
  return 0;
}
```

    - **Nevýhoda:** minimální, ale nekonstantní perioda zpracování
  - **Řešení s přerušením**
    - Výkonný kód umístěn do obsluhy přetečení časovače.
    - Časovač odměřuje konstantní periodu vykonávání kódu.

```
int main(void)
{
      setup();    // inicializace periferií
      while(1)    // hlavní nekonečná smyčka programu
      {
            // prázdná
      }
      return 0;
}

ISR(TIMER1_OVF_vect)
{
      // čtení dat ze senzorů
      // zpracování dat a ...
      // ... výpočet nových hodnot procesu
      // nastavení akčních členů
}
```

    - **Výhoda:** konstantní perioda zpracování
    - **Nevýhoda:** dlouhý kód v ISR blokuje obsluhu dalších případných IRQ (pokud nelze použít vnořená přerušení, a na AVR nelze)

15.2.3 Událostní programování

- Jednotlivé části programu (moduly) si předávají data ke zpracování.
- Příklady:
  - vyvolání přerušení (přetekl časovač, USART přijal data),
  - tlačítko bylo stisknuto,
  - obsluha displeje,
  - atd.
- Řešení - různé způsoby:
  - **Obsluha události je krátká:**
    - Kompletní obsluha přímo u zdroje události – obsluha je krátká, nebude blokovat zpracování dalších událostí.
  - **Obsluha události je dlouhá:**
    - Základní (nejnutnější) zpracování u zdroje + vyvolání události (nastavení příznaku) a dokončení zpracování jinde (např. v nekonečné smyčce hlavního programu), aby dlouhotrvající zpracování události neblokovalo obsluhy dalších událostí (přerušení).

Příklad – příjem dat ze sériové linky USART a obsluha stisku tlačítek

```
// globální proměnné
#define RX_BUFFER_LENGTH 20   // velikost prijimaciho bufferu
volatile uint8_t rxBuffer[RX_BUFFER_LENGTH] = {0};    // prijimaci buffer pro
data
volatile uint8_t fRxComplete = 0;   // priznat kompletne prijate zpravy

// program
int main(void)
{
	setup();			// inicializace periferii
	while(1)			// hlavni nekocnena smycka programu
	{
		// zpracovani prijate zpravy po seriove lince UART
		if (fRxComplete)
		{
			fRxComplete = 0;  // událost obsloužena
			// zpracovaní přijaté zprávy uložené v rxBuffer
		}

		// reakce na tlacitka
		if (tlacitko bylo stisknuto)
		{
			// reakce na stisk tlacitka
		}

		// vypis na displej
	}
	return 0;
}
```

```
// preruseni od prijmu na seriove lince USART
ISR(USART_RXC_vect)
{
	uint8_t rxData;		// přijatý bajt

	rxData = UDR;		// vyčtení přijatého bajtu
	if (rxData == '\n')	// pokud byla zpráva přijata celá
	{
		rxBuffer[i] = '\0'; // tak je řetězec kompletní
		i = 0;
		fRxComplete = 1;  // nahodíme pouze příznak a zpracování zprávy
	}				// se dokončí jinde
	else
	{
		// uložení přijatého znaku do řetězce
		rxBuffer[i] = rxData;
		if (i < RX_BUFFER_LENGTH-1) i++; // ochrana proti přetečení
	přijímacího bufferu
	}
}
```

- Vlastnosti toho typu programování
  - **Krátká přerušení** (pouze nejnutnější zpracování události) = rychlá reakce systému na všechny události.
  - Ale zároveň kombinace přerušení + příznaku = zpomalení reakce na konkrétní událost
    - Okamžik zahájení úplného zpracování události závisí na okamžiku testování příznaku v nekonečné smyčce hl. programu.
    - Možnost prioritizace častější kontrolou vybraných příznaků v místě úplného zpracování požadavku.

15.2.4 Synchronizace dat

- Příklad – teplotní alarm

```
volatile int temp[2];   // pole pro ulozeni teplot, globalni promenna

int main(void)
{
	int temp1, temp2;

	setup();    // inicializace periferií
	while(1)    // hlavní nekonečná smyčka programu
	{
		temp1 = temp[0];
		temp2 = temp[1];
		if (temp1 != temp2)     // pokud jsou T rozdilné, vyhlaš
			.....           // poplach

	}
	return 0;
}
```

```
// periodicke cteni teplot
ISR(TIMER0_OVF_vect)
{
      temp[0] = ...;    // přečtu teplotu z čidla
      temp[1] = ...;    // přečtu teplotu z čidla
}
```

- Možný běh programu
  - `temp1 = temp[0];`
  - `ISR()`
  - `temp2 = temp[1];`
  - test na teploty → rozdíl → poplach
- zdroj problému: `temp[2]` je sdíleno mezi `main()` a `ISR()` – možná nekonzistentnost dat (teploty nepatří k sobě)
- **atomická operace** = dále nerozložitelná (např. vykonání jedné instrukce)
  - jeden řádek v C (obvykle) není atomická operace
- **Řešení: kritická sekce – zajistí atomičnost:**

```
while(1)
{
      cli(); // zakaz preruseni
      temp1 = temp[0];
      temp2 = temp[1];
      sei(); // povoleni preruseni
      if (temp1 != temp2)
            // poplach
}
```

Kritická sekce

Ještě o trochu lepší řešení:

Přerušení mohlo, ale nemuselo být před jeho zakázáním povoleno. Protože nějaká jiná část kódu ho mohla mezi tím zakázat. V SREG je bit I.

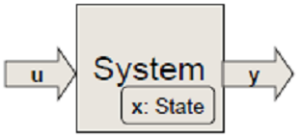
```
unsigned char zaloha;
while(1)
{
      zaloha = SREG;
      cli(); // zakaz preruseni
      temp1 = temp[0];
      temp2 = temp[1];
      SREG = zaloha; // povoleni preruseni
      if (temp1 != temp2)
            // poplach
}
```

Kritická sekce

Povolí přerušení pouze pokud bylo před vstupem do kritické sekce povoleno. Tím se nezmění chod aplikace, pokud před vstupem do kritické sekce nějaká jiná část kódu přerušení zakázala.

15.2.5 Stavové programování

- Program na bázi stavového automatu



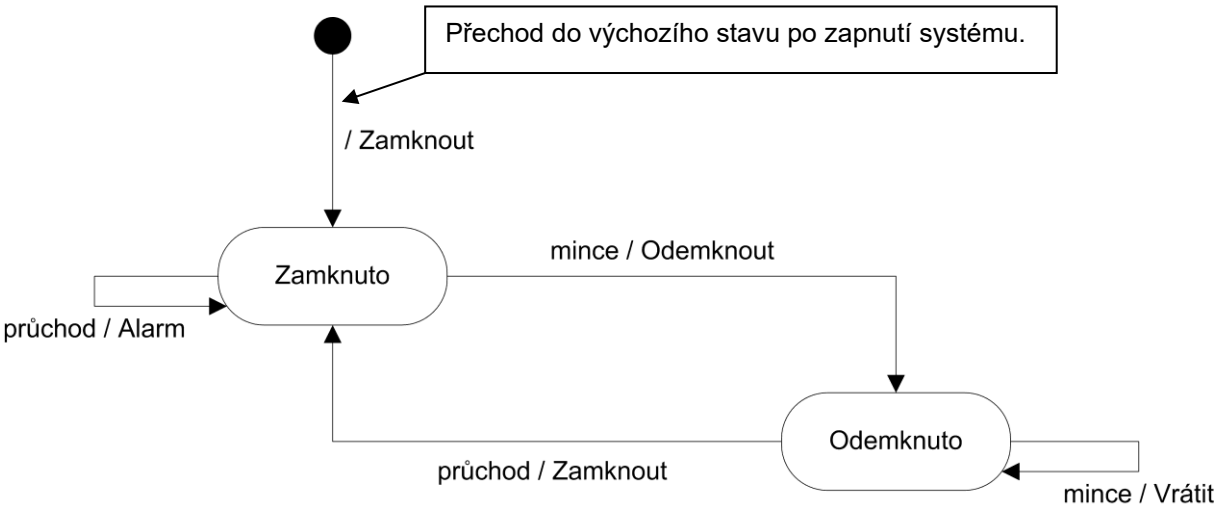
Discrete-time system	Continuous-time system
$x(k+1) = f(x(k), u(k))$	$\dot{x}(t) = f(x(t), u(t))$
$y(k) = g(x(k),u(k))$	$y(t) = g(x(t),u(t))$

- Výstup *y(k)* ze systému je dán:
  - Aktuální hodnotou vstupů *u(k)* a vnitřních stavů *x(k)*
- Nový stav *x(k+1)* je dán:
  - Aktuální hodnotou vstupů *u(k)* a vnitřních stavů *x(k)*
- Program prochází jednotlivé stavy, ve kterých testuje vstupní signály nebo jejich data a na jejich základě realizuje přechod do stavů následujících a produkuje výstupy.

Příklad – turniket:

- Stavy – Zamknuto, Odemknuto
- Události – triggery (spouštěče) přechodů mezi stavy:
  - mince – znamená vhození mince do turniketu
  - průchod – znamená, že osoba prošla turniketem
- Akce – Odemknout, Zamknout, Vrátit peníze, Alarm

- **Stavový diagram popisující chování systému**



  - Šipky – čáry přechodu mezi stavy.
  - Čáry přechodu jsou označeny:
    - *událostí, která spouští přechod* – obecně např. změna hodnoty nějaké proměnné, přijmutí nějakých dat po sběrnici, vypršení time-outu apod.
    - *podmínkou, která musí být splněna* – např. nějaká proměnná musí nabývat určité hodnoty
    - *akcí, která se má při přechodu vykonat* – nastavení výstupů
  - Syntaxe: **událost [podmínka] / akce**
    - Ani jedna část není povinná.
    - Různé možnosti:
      - mince
      - [hodnota = 10 Kč]
      - mince [hodnota = 10 Kč]
      - mince / Odemknout
      - mince [hodnota = 10 Kč] / Odemknout
      - [hodnota = 10 Kč] / Odemknout
  - Počáteční stav
    - Označuje (definuje) výchozí stav automatu po zapnutí systému.
    - Přechod do výchozího stavu nesmí obsahovat žádnou podmínku.
  - Po spuštění přechodu dojde ke změně stavu systému.

- **Tabulka přechodů**
  - Textové vyjádření stavového diagramu.
  - Existuje 1D a 2D provedení.
  - **1D tabulka přechodů:**

číslo přechodu	Aktuální stav	Událost	Nový stav	Akce
1	Zamknuto	mince	Odemknuto	Odemknout
2	Zamknuto	průchod	Zamknuto	Alarm
3	Odemknuto	mince	Odemknuto	Vrátit
4	Odemknuto	průchod	Zamknuto	Zamknout

    - Každý řádek představuje jeden přechod.
    - Tabulka se čte takto:

      „*Pokud jsme ve stavu Zamknuto a dojde k události (ke vhození) mince, přecházíme do stavu Odemknuto a provedeme akci Odemknout.*“
    - Tabulka se prochází od shora dolů, až se narazí na správný přechod - rychlost procházení tabulky klesá s rostoucím počtem přechodů.
  - **2D tabulka přechodů:**

Aktuální stav\Událost	mince	průchod
Zamknuto	Odemknuto/Odemknout	Zamknuto/Alarm
Odemknuto	Odemknuto/Vrátit	Zamknuto/Zamknout

    - Řádky – aktuální stav
    - Sloupce – událost
    - Prvek tabulky – nový stav/akce při přechodu do nového stavu
    - U 2D tabulky je rychlé vyčtení nového stavu a akce, rychlost nezávisí na počtu přechodů v tabulce (aktuální stav a událost jsou přímo indexy do tabulky).
- **Implementace stavového automatu**
  - Existuje několik způsobů implementace
    - Vnořené příkazy `switch - case`
    - Tabulka přechodů (1D nebo 2D)
    - SMC (State Machine Class) – třída v OOP (C++, C#, Java…)

15.2.5.1 Implementace pomocí `switch – case`

- Vhodné pro jednoduché stavové automaty
  - Všechny stavy a přechody jsou viditelné na 1 až 2 obrazovky kódu – snadná údržba a přehlednost.
  - Elegantní a efektivní.
- Nevhodné pro složité stavové automaty s vysokým počtem stavů a událostí
  - Mnoho stran kódu – obtížná orientace a údržba kódu.
  - Míchání logiky automatu s výkonným kódem, který implementuje jednotlivé akce – ztráta přehlednosti.

```
// definice vsech stavu stavoveho automatu (jako novy datovy typ StavFSM_t)
typedef enum
{
	ZAMKNUTO,
	ODEMKNUTO
} StavFSM_t;

// definice udalosti (jako novy datovy typ UdalostFSM_t)
typedef enum
{
	MINCE,
	PRUCHOD,
	ZADNA		// pomocna udalost, rika ze nedoslo k zadne udalosti
} UdalostFSM_t;

// globalni promenne
StavFSM_t stav;
UdalostFSM_t udalost;

// funkce realizujici logiku stavoveho automatu
void SpustFSM(eUdalostFSM_t u);
// funkce, ktere realizuji akce
void Zamknout(void);
void Odemknout(void);
void Vratit(void);
void Alarm(void);

void main(void)
{
	stav = ZAMKNUTO;  // vychozi stav po zapnuti
	Zamknout();       // zamknuti
	while (1)   // nekonecna smycka
	{
		/*
			udalost = ZADNA; // vychozi udalost
			.
			.
			nejaky kod, ktery vytvari udalost...
			udalost = ..........;
		*/
		SpustFSM(udalost);      // funkce pro obsluhu stavoveho automatu
	}
}
// Implementace stavoveho automatu
void SpustFSM(UdalostFSM_t u)
```

```
{
	switch (stav)
	{
	case ZAMKNUTO:
		switch (u)
		{
		case MINCE:		// prechod c.1
			stav = ODEMKNUTO; // prechod do dalsiho stavu
			Odemknout();// akce
			break;
		case PRUCHOD:		// prechod c.2, zustavame ve stejnem stavu
			Alarm();	// akce
			break;
		case ZADNA:		// nic nedelej, zustan ve stejnem stavu
			break;		// (obsluha teto udalosti zde nemusi byt)
		}
		break;

	case ODEMKNUTO:
		switch (u)
		{
		case MINCE:		// prechod c.3, zustavame ve stejnem stavu
			Vratit();	// akce
			break;
		case PRUCHOD:		// prechod c.4
			stav = ZAMKNUTO;  // prechod do dalsiho stavu
			Zamknout(); // akce
			break;
		case ZADNA:		// nic nedelej, zustan ve stejnem stavu
			break;		// (obsluha teto udalosti zde nemusi byt)
		}
		break;
	}
}

// Zamkne turniket
void Zamknout(void)
{
	// nejaky kod
}


// Odemkne turniket
void Odemknout(void)
{
	// nejaky kod
}

// Vrati vhozene mince
void Vratit(void)
{
	// nejaky kod
}

// Spusti alarm
void Alarm(void)
{
	// nejaky kod
}
```

- Vnořené příkazy `switch - case` rozdělují kód do 4 částí, přičemž v daném čase je aktivní vždy pouze jedna z nich.
  - Každý vnitřní `case` představuje jeden přechod ve stavovém automatu.
    - Např. část pro stav *ZAMKNUTO* a událost *MINCE* mění stav na *ODEMKNUTO* a spouští akci (volá funkci) *ODEMKNI*.
- Pokud jsou akce jednoduché, tak není třeba vytvářet funkce pro obsluhu akcí a kód akce lze umístit přímo do patřičného `case`.

- V závislosti na konkrétní aplikaci není vždy nutné vytvářet události, které spouštějí přechod do nového stavu. Někdy je přechod do nového stavu spuštěn pouze tím, že vstupy nabývají požadovaných hodnot.
  - Viz kód níže - nemá vyloženě události, které spouštějí přechod do nového stavu. Do nového stavu se přechází, pokud vstupní proměnné (`drivingRegimeSwitchPos`, `velocity`, `throttle_perc`) nabývají požadovaných hodnot.

```
void DrivingRegimeFSM(tDrivingRegime drivingRegimeSwitchPos,int16 velocity,int16 throttle_perc)
{
	switch (_rezimJizdyZad)		// Stav je zaroven zadany rezim jizdy
	{
		case NORMAL: // stav NORMAL
			// prechod c.1
			if (drivingRegimeSwitchPos == MUV74::EDrivingRegime::NORMAL)
			{
				// nic nedelej, setrvej v aktualnim stavu
				_cisloPrechoduRezimJizdy = 1; // pouze pro ladeni
			}
			// prechod c.2
			else if ((drivingRegimeSwitchPos == EMERGENCY) &&
				  (velocity == 0) &&
				  (throttle_perc == 0))
			{
				_rezimJizdyZad = EMERGENCY;   // novy stav
				_cisloPrechoduRezimJizdy = 2; // pouze pro ladeni
			}

			// prechod c.3
			else if (drivingRegimeSwitchPos == UNKNOWN)

			{
				// nic nedelej, setrvej v aktualnim stavu
				_cisloPrechoduRezimJizdy = 3; // pouze pro ladeni
			}
			// nesplneny podminky prechodu
			else
			{
				// nic nedelej, setrvej v aktualnim stavu
				_cisloPrechoduRezimJizdy = -1; // pouze pro ladeni
			}
			break;
```

15.2.5.2 Implementace pomocí tabulky přechodů

- Opakování
  - Tabulka přechodů (1D)
    - Textové vyjádření stavového diagramu.

číslo přechodu	Aktuální stav	Událost	Nový stav	Akce
1	Zamknuto	mince	Odemknuto	Odemknout
2	Zamknuto	průchod	Zamknuto	Alarm
3	Odemknuto	mince	Odemknuto	Vrátit
4	Odemknuto	průchod	Zamknuto	Zamknout

    - Každý řádek představuje jeden přechod.
- Vhodné pro složité stavové automaty, kde by `switch - case` implementace byla neefektivní a nepřehledná.
- Používá se pole pointerů na funkce, které implementují akce.
- **Výhody:**
  - Tabulka přechodů vytvořená v kódu se čte stejně jako tabulka přechodů v textové podobě.
  - Kód logiky automatu je oddělený od implementace akcí.
  - Přidání přechodu do kódu je snadné.
  - Dává prostor pro implementaci strojového načtení tabulky přechodů z externího souboru (tabulku v kódu pak není nutné při změně nebo doplnění ručně vyplňovat/modifikovat).
  - Lze měnit strukturu stavového automatu za běhu aplikace – v aplikaci může být více různých tabulek přechodu, každá reprezentující jiný stavový automat. Výběr tabulky může být proveden až za běhu aplikace (např. podle počátečních podmínek po startu aplikace), mezi tabulkami lze přepínat za běhu.
  - U 2D tabulky rychlé vyčtení nového stavu a akce, tabulka se neprochází jako v případě 1D tabulky.
- **Nevýhody:**
  - 1D tabulka
    - Při velkém počtu přechodů se přímo úměrně (lineárně) snižuje rychlost procházení tabulky.
    - U 2D tabulky procházení odpadá, protože jí není třeba postupně procházet, aktuální stav a událost jsou přímo indexy do tabulky (rychlost určení nového stavu a akce nezávisí na počtu prvků v tabulce).

- 2D tabulka
  - Vyšší paměťová náročnost (počet stavů krát počet událostí) vůči 1D tabulce. Tabulka musí být vyplněna vždy celá, i když některé kombinace stav/událost nejsou definované a zbytečně zabírají místo v tabulce (předpokládá se tabulka ve formě statického 2D pole).
  - Lze použít jen tehdy, pokud jsou stavy a události reprezentovány celými čísly jdoucími za sebou, protože představují indexy do 2D tabulky (0, 1, 2, …).
  - Pracnější přidávání nových stavů, událostí a přechodů do tabulky v kódu (protože je 2D).

Ukázka implementace formou 1D tabulky přechodů

```
// definice vsech stavu stavoveho automatu (jako novy datovy typ StavFSM_t)
typedef enum
{
	ZAMKNUTO,
	ODEMKNUTO
} StavFSM_t;

// definice udalosti (jako novy datovy typ UdalostFSM_t)
typedef enum
{
	MINCE,
	PRUCHOD,
	ZADNA			// pomocna udalost, rika ze nedoslo k zadne udalosti
} UdalostFSM_t;

// novy datovy typ akce_t (pro zjednoduseni definice) - je to pointer na funkci, ktera nema
zadne parametry ani navratovou hodnotu
// pomoci tohoto pointeru se bude volat prislusna akce pri zmene stavu
typedef void (*akce_t)(void);

// aktualni stav, udalost, novy stav a pointer na funkci pro vykonani akce pri zmene stavu
// jsou zapouzdreny do struktury, ktera je definovana jako novy datovy typ Prechod_t
typedef struct
{
    StavFSM_t aktStav;		// aktualni stav
    UdalostFSM_t u;		// udalost, ktera vyvola prechod do noveho stavu
    StavFSM_t novyStav;		// novy stav, do ktereho se prejde
    akce_t akce;		// pointer na funkci (akci), ktera se ma vykonat behem prechodu
				// do noveho stavu
} Prechod_t;

// deklarace funkci pro obsluhu akci
void Zamknout(void);
void Odemknout(void);
void Vratit(void);
void Alarm(void);
```

```
/*
    1D tabulka prechodu
    Je jadrem stavoveho automatu.
    Tabulka je implementovana jako 1D pole. Jejimi prvky jsou prechody mezi stavy.
    Tabulka obsahuje tolik prvku kolik je prechodu mezi stavy.
    Pridani prechodu ve stavovem diagramu/v textove tabulce prechodu znamena v kodu pouhe
doplneni prechodu do tabulky a implementaci funkce, ktera obslouzi prechod, nic vic.
*/
Prechod_t tabulkaPrechodu[] =
{
    // {aktualni stav, udalost, novy stav, nazev funkce pro vykonani akce}
       {ZAMKNUTO,MINCE,ODEMKNUTO,Odemknout},
       {ZAMKNUTO,PRUCHOD,ZAMKNUTO,Alarm},
       {ODEMKNUTO,MINCE,ODEMKNUTO,Vratit},
       {ODEMKNUTO,PRUCHOD,ZAMKNUTO,Zamknout}
};

#define POCET_PRECHODU  (sizeof(tabulkaPrechodu)/sizeof(Prechod_t))  // pocet prvku v tabulce
prechodu (spocita se v dobe prekladu)

// globalni promenne
StavFSM_t stav;       // drzi aktualni stav
UdalostFSM_t udalost;

// deklarace funkci
void SpustFSM(Prechod_t tab[], int prvku, StavFSM_t *aktStav, UdalostFSM_t udalost);

void main(void)
{
       stav = ZAMKNUTO;      // vychozi stav po zapnuti
       Zamknout();           // zamkneme turniket
       while (1)      // nekonecna smycka
       {
              /*
                     udalost = ZADNA;      // vychozi udalost
                     .
                     .
                     nejaky kod, ktery vytvari udalost...
                     udalost = ...........;
              */
              SpustFSM(tabulkaPrechodu,POCET_PRECHODU,&stav,udalost);  // volani funkce pro
obsluhu stavoveho automatu
       }
}

// Implementace stavoveho automatu
void SpustFSM(Prechod_t tab[], int prvku, StavFSM_t *aktStav, UdalostFSM_t udalost)
{
    int i;
    Prechod_t prechod;

    // prochazi tabulku prechodu a hleda odpovidajici kombinaci aktualniho stavu a udalosti
    for (i=0;i<prvku;i++)
    {
        prechod = tab[i];
        // urceni noveho stavu a akce v prechodu do noveho stavu na zaklade aktualniho stavu
a udalosti
        if (prechod.aktStav == *aktStav &&
            prechod.u == udalost)
        {
            *aktStav = prechod.novyStav; // novy stav
            if (prechod.akce != NULL) prechod.akce(); // akce pri prechodu do noveho stavu
            break;  // po vykonani prechodu ukonci prohledavani
        }
    }
}
```

```
// Zamkne turniket
void Zamknout(void)
{
	// nejaky kod
}

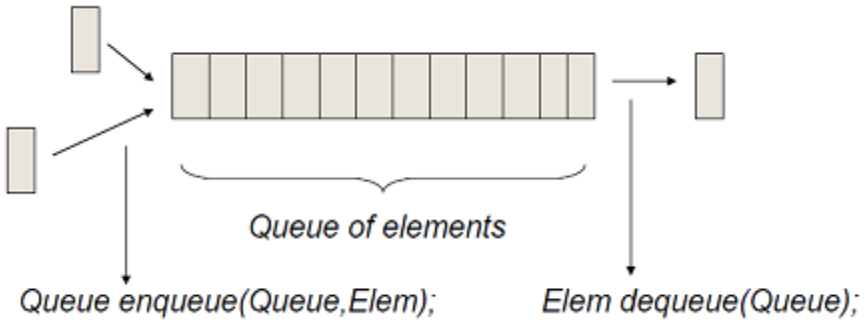
// Odemkne turniket
void Odemknout(void)
{
	// nejaky kod
}

// Vrati vhozene mince
void Vratit(void)
{
	// nejaky kod
}

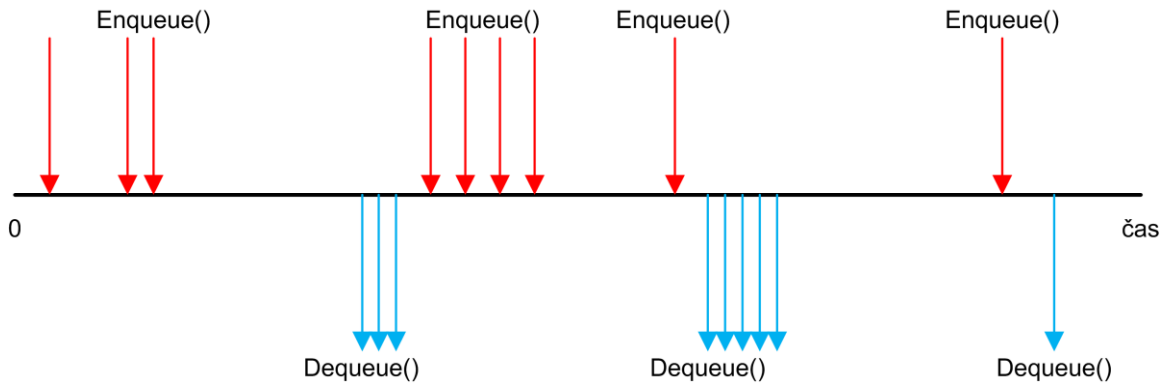
// Spusti alarm
void Alarm(void)
{
	// nejaky kod
}
```

15.3 Funkční fronty

- Použití:
  - Pro systémy nárazově zatížené větším množstvím požadavků (dat).
  - Pro předávání dat mezi smyčkami (procesy), které běží různou rychlostí, nejsou mezi sebou synchronizované.

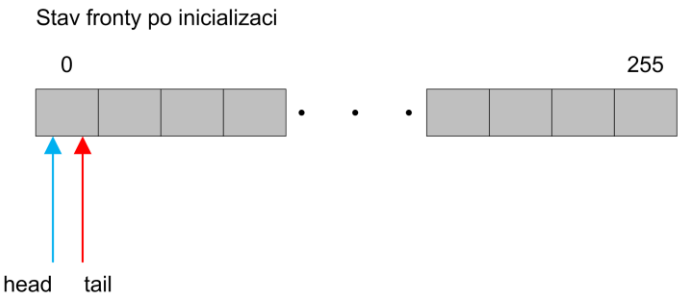


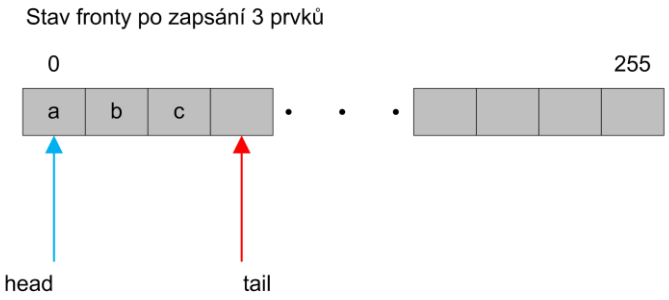
- Zásobník typu **FIFO** (**F**irst **I**nput **F**irst **O**utput).
- **Princip použití:**
  - Typicky pro předávání dat mezi procesy, které běží různě rychle a nesynchronizovaně.
  - Nějaký proces zaplní nárazově frontu novými prvky (daty).
  - Jiný proces periodicky sleduje stav fronty, a pokud ta obsahuje nějaká data, tak je vyčte a zpracuje.
    - I během právě probíhajícího zpracování dat mohl klidně zase druhý proces do fronty zapsat další nová data….
  - Typické použití:
    - Komunikační fronty pro vysílání (Tx) a příjem (Rx) dat.

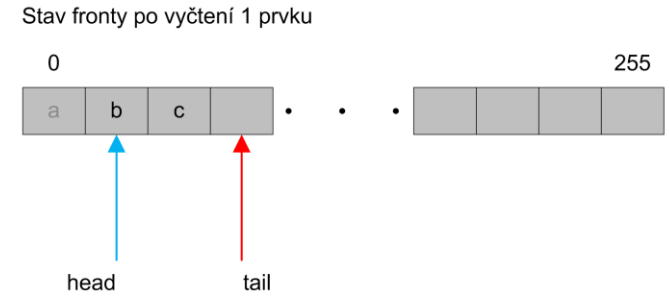


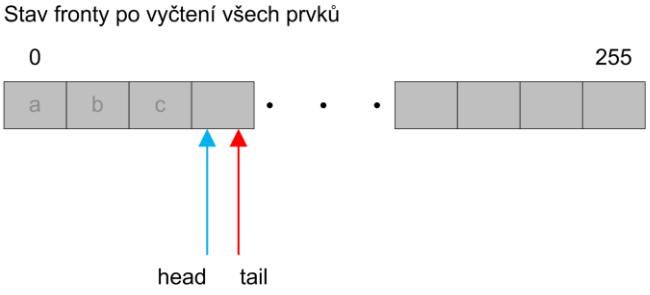
Enqueue() – vložení dat (zápis) do fronty
Dequeue() – vyzvednutí dat (čtení) z fronty

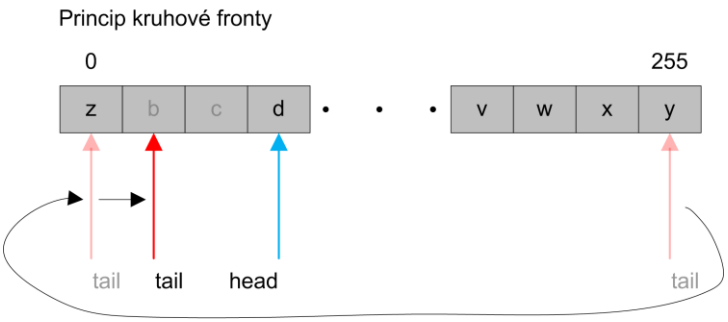
- **Manipulace s frontou** (modrá šipka - čtení z fronty, červená šipka zápis do fronty):
    - Předpokládejme frontu dlouhou 256 prvků.











Důležité jsou ukazatele pro pohyb ve frontě:

- Jsou v nich uloženy indexy na prvky v poli.
- `head` – pro čtení, ukazuje na nejstarší prvek k vyzvednutí, při každém přečtení prvku se ukazatel posune o 1 prvek vpravo
- `tail` – pro zápis, ukazuje na volné místo v poli pro uložení dalšího prvku, při každém uložení nového prvku do fronty se ukazatel posune o 1 prvek vpravo
- Po dosažení konce fronty se ukazatel přesune na začátek fronty → princip kruhového bufferu.

- **Fronta bývá implementována jako:**
  - Kruhový buffer - pole prvků nějakého datového typu (viz kód níže).

    nebo
  - Zřetězený seznam.

- **Ukázka implementace fronty pomocí kruhového bufferu**

```
// obsah souboru queue256.h
#ifndef QUEUE256_H
#define QUEUE256_H

typedef struct
{
      unsigned char data[256];      // buffer o delce 256 bajtu
      unsigned char head;           // zakladni myslenka: 255 + 1 = 0
      unsigned char tail;
} tQ256;    // kruhovy buffer delky 256 bajtu

// makra pro praci s frontou
#define InitQ256(q)     q.head = 0; q.tail = 0
#define Enq256(q, item) q.data[q.tail++] = item
#define Deq256(q)       (q.data[q.head++])
// volat napr. x = Deq256(q); x je promenna typu char
#define Q256isEmpty(q)  (q.head == q.tail)      // 1 kdyz je fronta prazdna

/*
Poznamky:
      - fronta je nedokonala
      - nelze menit velikost fronty, vzdy musi byt 256 prvku
      - neni hlidano a osetreno preteceni fronty (tail muze dobehnout a
      predbehnout head)
      - pevny datovy typ prvku fronty (unsigned char)
*/
#endif
```

- Základem je pole, do kterého se ukládají data: `unsigned char data[256];`
- Do fronty lze uložit max. 256 prvků a fronta umí pracovat pouze s daty typu `unsigned char` a s žádnými jinými (fronta není generická – neumí pracovat s jakýmkoliv datovým typem).
- Důležité jsou ukazatele pro pohyb ve frontě:
  - Jsou v nich uloženy indexy na prvky v poli `data`.
  - `head` – pro čtení, ukazuje na nejstarší prvek k vyzvednutí
  - `tail` – pro zápis, ukazuje na volné místo v poli pro uložení dalšího prvku
  - Výše uvedená fronta musí mít vždy 256 prvků, aby fungoval princip kruhového bufferu. Protože ukazatele `head` a `tail` jsou 8-bitové a spoléhá se na to, že platí 255 + 1 = 0 (přetečení 8-bit indexu zajistí automatický přesun pointeru z konce na začátek fronty).

- Pole a ukazatele jsou zapouzdřeny do datové struktury, pro kterou byl vytvořen nový datový typ `tQ256`.
- Logika, která ukládá a vyzvedává data z fronty prostřednictvím ukazatelů `head` a `tail` je v tomto konkrétním případě umístěna do maker. Tím je umožněno mít v aplikaci více kruhových front, které se obsluhují jednotným způsobem (do makra se vždy předá jen proměnná reprezentující danou frontu).
- **Buffer + ukazatele + logika = „kruhový buffer“** („circular buffer“).
- Použití fronty v programu spočívá v provedení těchto kroků:
    - Vytvoření proměnné typu `tQ256`: `tQ256 fronta;`
    - Inicializace fronty – zavolání makra `InitQ256(fronta);`
    - Pak už můžeme data do fronty ukládat nebo z ní číst:
        - Uložení dat do fronty: `Enq256(fronta,10);`
        - Zjištění zda jsou ve frontě nějaká nepřečtená data:

```
if (!Q256isEmpty(fronta))
{
   // vyzvednutí jednoho prvku z fronty
   /*
         nejake zpracovani dat
   */
}
```

        - Vyzvednutí dat z fronty:

```
unsigned char data;

if (!Q256isEmpty(fronta))
{
   data = Deq256(mojeFronta);
   /*
         nejake zpracovani dat
   */
}
```

        - Čtení dat z fronty se obvykle provádí v cyklu – postupně vyzvedneme a zpracujeme všechna data čekající ve frontě:

```
// vycitej data z fronty, pokud nejaka obsahuje
while (!Q256isEmpty(fronta))
{
      data = Deq256(fronta);  // vycte jeden prvek z fronty
      /*
            nejake zpracovani dat
      */
}
```

- **Uvedená implementace fronty má tato negativa:**
  - Nelze měnit velikost fronty, musí být vždy 256 prvků dlouhá.
  - Fronta umí pracovat pouze s daty typu `unsigned char` a s žádnými jinými.
  - Není hlídáno a ošetřeno přetečení fronty (`tail` může doběhnout a předběhnout `head`).
  - Pomocí „magie s makry“ lze udělat generickou frontu a uživatelsky definovaným počtem prvků. Viz knihovna `ringbuffer.h` nacházející se v adresáři „Frotna“ v ZIP archivu knihoven, které používáme na cvičeních.

Příklad použití fronty:

```
// Demonstrace pouziti fronty.
#include "queue256.h"

// fronta 256-ti prvku typu int - globalni promenna, aby mohla byt sdilena
vsemi castmi aplikace
tQ256 fronta;

// program
void main(void)
{
	InitQ256(fronta); // inicializace fronty
	/*
		inicializace periferii,
		povoleni preruseni
	*/

	while(1)	// hlavni smycka programu
	{
		// zpracovani dat ulozenych ve fronte
		// vycita data z fronty, pokud nejaka obsahuje
		while (!Q256isEmpty(fronta))
		{
			data = Deq256(fronta);  // vycte jeden prvek z fronty
			/*
				nejake zpracovani dat
				.
				.
			*/
		}

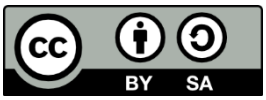
		// nějaký další kód, který trvá nějak dlouho
	}
}

// udalost, ktera prida prvek do fronty, napr. prijem dat po sbernici
ISR(IRQ1_vect)
{
	char data;
	/*
		nejaky kod, ktery naplni promennou data
		.
		.
		data = ......;
	*/
	Enq256(fronta,data); // prida prvek do fronty
}
```

UNIVERZITA PARDUBICE
DOPRAVNÍ FAKULTA
JANA PERNERA

Vytvořeno v rámci projektu **Green Deal UPCE**, reg. č. NPO_UPCE_MSMT-2142/2024-4.

Toto dílo podléhá licenci Creative Commons BY-SA 4.0. Pro zobrazení licenčních podmínek navštivte https://creativecommons.org/licenses/by-sa/4.0/.







NÁRODNÍ PLÁN OBNOVY



Financováno Evropskou unií
NextGenerationEU

Univerzita Pardubice
Dopravní fakulta Jana Pernera

Kurz celoživotního vzdělávání

Elektrotechnické systémy pro environmentálně příznivou dopravu

Předmět

Pokročilé systémy digitálního řízení a komunikace v dopravní technice

Studijní materiál

Mikroprocesorová řídící technika, 2. část

Ing. Zdeněk Mašek



Obsah

1 Čítače/časovače ... 3
1.1 Úvod ... 3
1.2 Taktování časovače ... 4
1.3 Základní použití časovače (režim Normal) ... 5
1.4 Použití předděličky časovače ... 10
1.5 Realizace dlouhých zpoždění v programu ... 13
1.6 Využití CTC režimu časovače ... 15
1.7 Přetečení časovače a jeho využití ... 33
1.8 Předvolba časovače v režimu Normal ... 40
1.9 Generování PWM ... 44
1.10 Záchytná jednotka (Input Capture Unit) ... 65
1.11 Měření délky impulsu ... 66
1.12 Asnychronní časovač Timer/Counter 2 ... 69
1.13 Příloha 1 – vzorce pro ne-PWM režimy ... 70
1.14 Příloha 2 – vzorce pro PWM režimy ... 71
1.15 Příloha 3 – přehled registrů časovačů na ATmega32 ... 72
1.16 Příloha 4 – obsazenost časovačů na Arduino ... 78
2 Měření otáček a polohy (úhlového natočení) ... 80
2.1 Inkrementální čidlo ... 80
2.2 Inkrementální čidlo s kvadraturním výstupem ... 81
2.3 Mechanický inkrementální enkodér ... 85
2.4 Vyhodnocení rychlosti otáčení z IRC čidel (detailněji) ... 97
2.5 Čidlo absolutní polohy (natočení) ... 109
3 A/D převodník ... 111
3.1 Vzorkování ... 111
3.2 Kvantování ... 118
3.3 Chyby A/D převodníků ... 121
3.4 Parametry reálného A/D převodníku ... 125
3.5 Základní typy A/D převodníků ... 127
4 Interní A/D převodník v AVR ... 130
4.1 Volba referenčního napětí obecně ... 132
4.2 Referenční napětí A/D převodníku na AVR ... 132
4.3 Práce s interním A/D převodníkem v AVR ATmega32 ... 134
4.4 Registry pro obsluhu A/D převodníku v AVR ... 147
4.5 Obsluha A/D převodníku v jazyku C ... 149
5 Vybrané programové techniky při použití A/D převodníku ... 158
5.1 Digitální filtrace ... 158
5.2 Lineární interpolace ... 172
6 Sériový přenos dat ... 179
7 Přístrojové sběrnice ... 180
8 Sběrnice SPI (Serial Peripheral Interface) ... 180
8.1 Vlastnosti SPI rozhraní ... 180
8.2 Topologie sběrnice ... 182
8.3 Přenos dat ... 183
8.4 SPI u AVR ... 187
8.5 Registry SPI v MCU AVR Atmega32 ... 190
8.6 Vysílání a příjem po SPI na AVR ... 192
8.7 SPI a kompilátor AVR-GCC ... 196

8.8 SPI a Arduino 202
9 Sběrnice I²C 213
9.1 Fyzické uspořádání sběrnice 213
9.2 Přenosový protokol 215
9.3 Příklad I²C obvodu – A/D převodník ADS7830 222
9.4 Příklad I²C obvodu – EEPROM 24Cxx 225
9.5 I²C u AVR 228
9.6 I²C a kompilátor CodeVision 229
9.7 I²C a kompilátor AVR-GCC 230
9.8 I²C a Arduino 235
10 Sériová linka RS-232 249
10.1 Parametry fyzické vrstvy 250
10.2 Přenášené signály 251
10.3 Konektory 252
10.4 Kabely 252
10.5 RS-232 na PC 253
11 UART 255
11.1 Formát rámce UART (frame format) 257
11.2 Přenosová rychlost (Baud Rate) v_B 258
11.3 Handshaking 259
11.4 UART v PC 260
11.5 UART v MCU 260
11.6 Převodník UART / RS-232 261
11.7 Převodník UART / USB 262
11.8 USART v AVR 264
11.9 USART a kompilátor CodeVision 276
11.10 USART a kompilátor AVR-GCC 278
12 Komunikační protokoly 284
12.1 ISO/OSI komunikační model a sběrnice 284
12.2 Úvod do problematiky komunikačních protokolů 286

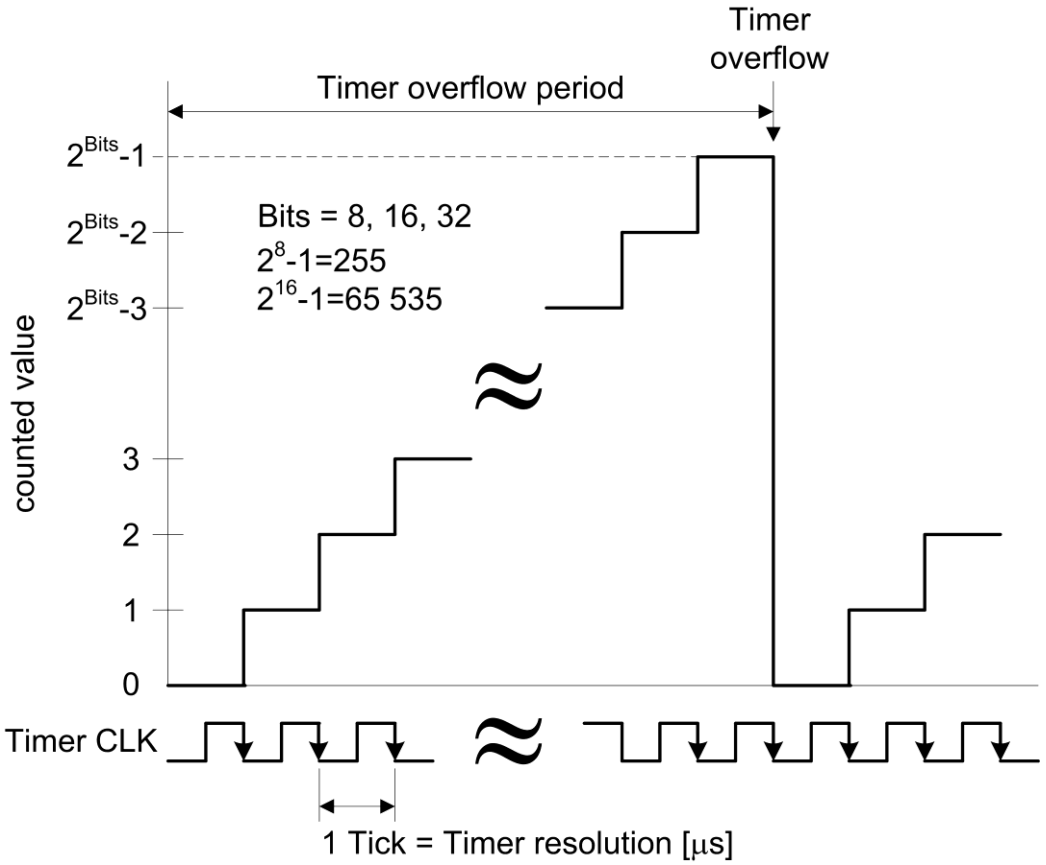
1 Čítače/časovače

1.1 Úvod

- Základní funkce časovače/čítače je **odměřování času**.
- Časovač/čítač je přídavná hardwarová periferie – běží nezávisle na CPU (nezávisle na vykonávaném hlavním programu).
- K čemu lze časovač/čítač využít?
    - Časování chodu aplikace.
    - Generování signálu na přiřazeném výstupním pinu
        - Např. signál typu PWM.
    - Měření parametrů vstupního signálu.
        - Délka pulsu
        - Perioda
        - Střída
- Časovač vs. čítač
    - Liší se pouze zdrojem taktovacího kmitočtu („hodin“).
    - Časovač – je taktován z interních hodin MCU.
    - Čítač – je taktován vnějším zdrojem signálu mimo MCU.
- Interakce aplikace a časovače
    - Nastavení časovače - řídící registry, čítací registr.
    - Programová obsluha časovače
        - periodické dotazováním na stav časovače (tzv. „polling“)
        - přerušení (při přetečení časovače, při komparaci atd.)
        - plně hardwarové generování průběhu na přiděleném pinu – nevyžadující polling ani přerušení (princip „spusť časovač a zapomeň“)
    - Snížení zátěže CPU.

1.2 Taktování časovače

- Každý časovač potřebuje ke své funkci taktovací signál (tzv. **hodiny**, **clock** - **CLK**).
- Každým tikem „hodin“ se zvýší hodnota čítacího registru časovače o 1.



- **Časové rozlišení časovače** (**Timer resolution**) = perioda taktovacích hodin (jak dlouho trvá jeden tik)
    - Časovač odměřuje čas v násobcích rozlišení časovače.
    - Je to nejmenší časový krok časovače.

$$\text{časové rozlišení časovače [s]} = \frac{1}{\text{taktovací frekvence časovače [Hz]}} \tag{1}$$

    - Rozlišení časovače při taktovací frekvenci např. 1 MHz bude:

$$\text{časové rozlišení časovače} = \frac{1}{1000000} = 1\ \mu\text{s} \tag{2}$$

    - Pokud uběhne např. 50 period taktovacích hodin, časovač načítá 50 tiků, které odpovídají době 50 x 1 μs = 50 μs. Časovač odměřil časový interval 50 μs.

V následujících příkladech uvedených v dalších kapitolách uvažujme MCU ATmega32 na frekvenci 1 MHz, LED připojenou k portu PORTA na pin č. 0 tohoto portu (pin PA0).

1.3 Základní použití časovače (režim Normal)

- Uvažujme následující jednoduchý příklad.
- Vytvořme program, který bude blikat s LED periodou 0,1 s (f = 10 Hz).
  - To znamená, že stav LED budeme měnit každých 0,05 s (f = 20 Hz).
- Odpovídající pseudokód:

```
Nastav port pro LED
Vynuluj časovač
Nastav časovač do režimu Normal
Spusť časovač

WHILE (nekonečná smyčka)
       IF hodnota časovače je rovna nebo větší než 1/20 s THEN
              Vynuluj časovač
              Změň stav LED
       END IF
END WHILE
```

- Rozpracovaný kód (pro překladač AVR-GCC):

```
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT     PORTA
#define LED_DDR      DDRA
#define LED_PIN_NUM  0

int main(void)
{
       LED_DDR |= (1<<LED_PIN_NUM); // vystup pro LED

       // TODO: vynulovat casovac
       // TODO: nastavit a spustit casovac


       while(1)
       {
              if () // TODO: zjistit stav casovace, jestli uz odmeril 1/20 s
              {
                     // ubehla 1/20 s

                     // TODO: vynulovat casovac

                     LED_PORT ^= (1<<LED_PIN_NUM);       // zmenit stav LED
              }
       }
}
```

- Nyní potřebujeme do kódu doplnit nastavení časovače, čtení hodnoty čítacího registru pro kontrolu uplynutí 1/20 s a časovač vynulovat. (vyměnit TODO komentáře za kód)
- Musíme znát, jaké časovače jsou na MCU AVR ATmega32 k dispozici a jaké mají vlastnosti.

Časovače na MCU AVR ATmega32

- ATmega32 je vybavena těmito časovači/čítači (Timer/Counter)
  - Timer/Counter 0 – 8-bit časovač
  - Timer/Counter 2 – 8-bit časovač
  - Timer/Counter 1 – 16-bit časovač, má 2 kanály (A a B, které mají společný čítací registr)
- Vlastnosti časovačů
  - Volitelná předdělička (např. 1, 8, 64, 256, 1024).
  - Funkce časovače nebo čítače.
  - Záchytná (Input Capture) jednotka např. pro měření délky časového intervalu.
  - Výstupní komparační jednotky (Output Compare) pro generování obdélníkového průběhu s určitou frekvencí a střídou.
  - Různé režimy generování PWM.

- V základním nastavení (režim časovač, normální režim – jednoduché počítání tiků) se časovače 8-bit a 16-bit časovače liší pouze tím do jaké hodnoty mohou čítat, než dojde k přetečení časovače.

Tabulka 1 Rozsah čítání dle bitové šířky časovače

Šířka časovače (čítacího registru)	Rozsah čítání	Max. časovačem měřitelný časový interval (časovač taktován na 1 MHz)
8-bit	0 – 255 (2^8-1)	255 μs
16-bit	0 – 65535 (2^{16}-1)	65535 μs

  - Po přetečení začne časovač počítat tiky zase od 0.

- Pro náš příklad bude časovač taktován na frekvenci rovné taktovací frekvenci MCU (tj. předdělička hodin časovače = 1), budeme předpokládat frekvenci **1 MHz**.
- **Musíme vypočítat potřebný počet tiků časovače**, které odpovídají požadovanému časovému intervalu pro změnu stavu LED (stav LED se mění každých **1/20 s**):

$$\text{počet tiků časovače} = \frac{\text{požadovaný časový interval}}{\text{perioda hodin časovače}} = \text{požadovaný časový interval} \cdot f_{timer}$$

$$\text{počet tiků časovače} = \frac{\frac{1}{20}}{0{,}000001} = \frac{1}{20} \cdot 1000000 = 50000 \quad (3)$$

- Vypočtená hodnota potřebného počtu tiků (50 000) je **větší než 255**, takže **použijeme 16-bit časovač Timer/Counter 1**.

- Dále musíme časovači nastavit zdroj hodin a časovač musíme spustit.
  - To se provede nastavením řídícího registru **TCCR1B**.
  - Řídící registr TCCR1B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **ICNC1** – aktivace/deaktivace omezovače šumu pro záchytnou jednotku
  - **ICES1** – určuje při jaké hraně vstupního signálu dojde k zachycení, vzestupná (ICES1 = 0) nebo sestupná (ICES0 = 1) hrana
  - **WGM13** a **WGM12** – výběr módů časovače spolu s WGM11 a WGM10
  - **CS12, CS11 a CS10** – volba zdroje hodin a předděličky hodin časovače

Tabulka 2 Výběr zdroje hodin pro časovač 1

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

  - V našem příkladu chceme, aby časovač tikal na stejné frekvenci jako CPU (tj. časovač bez předděličky).
    - V registru **TCCR1B** nastavíme bit **CS10 = 1** a bity **CS11**, **CS12 = 0**.
    - Nic ostatního nemusíme měnit, po resetu je obsah registru TCCR1B vynulován, což nám vyhovuje (je nastaven normální režim a časovač je zastaven).
  - Nastavením zdroje hodin se časovač rovnou spustí.

Tabulka 3 Režimy časovače Timer/Counter 1

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

- **Zbývá vynulovat časovač po uplynutí 1/20 s**
  - Vynulováním čítacího registru **TCNT1** (TCNT1 = 0).
  - V registru **TCNT1** je uložena aktuální načítaná hodnota časovače.
  - Registr **TCNT1** je 16-bitový (protože je časovač Timer/Counter 1 je 16-bitový).
    - Je rozdělen na 2 poloviny (horní **TCNT1H** a dolní **TCNT1L**).
    - V AVR-GCC ovšem může přistupovat jednoduše na celý 16-bit registr **TCNT1**, správný přístup si zajistí kompilátor.

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 1 Čítací registr 16-bit časovače Timer/Counter 1

- Úplný program (pro překladač AVR-GCC):

```
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT     PORTA
#define LED_DDR      DDRA
#define LED_PIN_NUM  0

int main(void)
{
	LED_DDR |= (1<<LED_PIN_NUM); // vystup pro LED
	TCNT1 = 0;           // vynulovani casovace
	TCCR1B = (1<<CS10);  // nastaveni a spusteni casovace (normal mode, no prescaler)

	while(1)
	{
		if (TCNT1 >= 50000) // zjisti, jestli uz se odmerila 1/20 s
		{
			// ubehla 1/20 s
			TCNT1 = 0;    // vynulovani casovace
			LED_PORT ^= (1<<LED_PIN_NUM);      // zmenit stav LED
		}
	}
}
```

- Připomínky k programu:
  - V porovnání TCNT1 vůči 50000 se nepoužívá operátor rovnost (==) ale operátor větší nebo rovno (>=) za účelem ochrany proti vynechanému porovnávání (když časovač tikne dvakrát mezi dvěma kontrolami), jinak by k porovnání nemuselo nikdy dojít. K vynechání porovnání dojde tehdy, pokud vykonání kódu v nekonečné smyčce while(1) bude trvat příliš dlouho (déle než je perioda tiku časovače).

1.4 Použití předděličky časovače

- V předchozím příkladu jsme odměřovali zpoždění 1/20 s. Počet tiků 16-bit časovače vyšel 50000 při taktovací frekvenci časovače 1 MHz, skoro jsme tedy vyčerpali rozsah čítacího registru časovače (max. počet tiků před přetečením 16-bit časovače je 65535).
- Co když budeme chtít generovat delší zpoždění?
- Použijeme předděličku časovače – za cenu horšího rozlišení v čase budeme schopni odměřit delší časový interval.
    - Předdělička je součástí obvodu časovače.
    - Předdělička dělí frekvenci vstupující do časovače N-krát (kde $N = 2^n$), do časovače tak vstupuje frekvence vydělená, nižší.

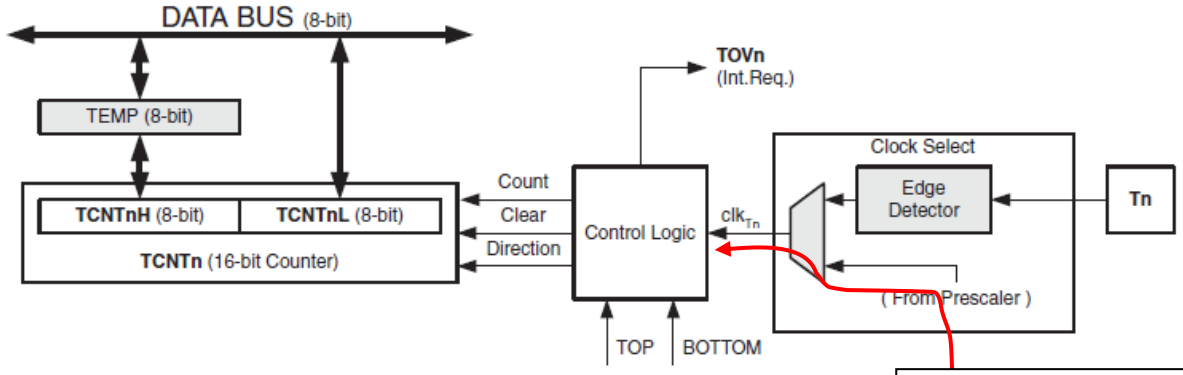
$$\text{taktovací frekvence časovače} = \frac{\text{vstupní frekvence}}{N} \tag{4}$$

- U AVR je taktovací frekvence časovače odvozena od tak. frekv. CPU (vydělením předděličkou). Pro vztahy zde uvedené platí: **vstupní frekvence = frekvence CPU**

$$\text{taktovací frekvence časovače} = \frac{\text{frekvence CPU}}{N} \tag{5}$$

- Vzorec pro výpočet časového rozlišení časovače po zavedení předděličky:

$$\text{časové rozlišení časovače [s]} = \frac{N}{\text{frekvence CPU [Hz]}} \tag{6}$$



Obrázek 2 Část vnitřní struktury Timer/Counter 1

Taktovací frekvence časovače, již po vydělení předděličkou.

- **Pojďme v našem příkladu s blikající LED snížit frekvenci blikání LED z 10 Hz na 0,5 Hz (perioda blikání LED 2 sekundy → změna stavu LED každou 1 s).**
  - Využijme za tímto účelem předdělička.
  - Postup výpočtu požadované hodnoty tiků časovače pro odměření zpoždění 0,5 s bude stejný jako v prvním příkladu, akorát taktovací frekvence časovače bude nižší (vydělená předdělička).
- Na MCU AVR ATmega32 máme u Timer/Counter 1 k dispozici tyto dělící poměry:
  - N = 1, 8, 64, 256, 1024
- Při uvažování vstupní frekvence 1 MHz můžeme vypočítat **časové rozlišení pro každou předdělička**:

Tabulka 4 Vliv předdělička rozlišení časovače

Předdělička (N)	Rozlišení při 1 MHz
1	1 µs
8	8 µs
64	64 µs
256	256 µs
1024	1024 µs

- Vzorec pro počet tiků časovače se zavedením předděličky:

$$\text{počet tiků časovače} = \frac{\text{požadovaný časový interval}}{\text{N} \cdot \text{perioda hodin CPU}}$$

$$\text{počet tiků časovače} = \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{N} \tag{7}$$

- **Nyní najdeme vhodnou předdělička N**, takovou, která umožní generovat požadované zpoždění 1 s:

Tabulka 5 Počet tiků časovače s předděličkou pro vytvoření zpoždění 0,5 s

Předdělička (N)	Počet tiků časovače
1	1 000 000
8	125 000
64	15 625
256	3 906,25
1024	976,525

  - Z výsledků musíme vyloučit předděličky 256 a 1024, protože při jejich použití by nebylo zpoždění generováno přesně (zbytek po dělení N není nulový).
  - Dále musíme vyloučit i předděličku 8 i provoz bez předděličky (N=1), protože 125 000 tiků ani 1 000 000 tiků nelze uložit do 16-bit čítacího registru (max. 65535).
  - **Použitelná je pouze předdělička 64**. A to pouze se 16-bit časovačem, pro 8-bit časovač jsou všechny předděličky nepoužitelné, protože počet tiků vychází ve všech případech větší než 2^8 (255).

- Nyní musíme správně nastavit řídící registr TCCR1B časovače Timer/Counter 1.

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

  - Bity **CS10, CS11 = 1**, bit **CS12 = 0** zajistí nastavení předděličky N=64 a spuštění časovače.
- Úplný program (pro překladač AVR-GCC):

```
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT      PORTA
#define LED_DDR       DDRA
#define LED_PIN_NUM   0

// Dokonceny program
int main(void)
{
        LED_DDR |= (1<<LED_PIN_NUM); // vystup pro LED
        TCCR1B = (1<<CS11) | (1<<CS10);     // nastaveni a spusteni casovace (normal
                                            mode, clock Fcpu/64)
        TCNT1 = 0;     // vynulovani casovace

        while(1)
        {
                if (TCNT1 >= 15625) // zjisti, jestli uz se odmerila 1 s
                {
                        // ubehla 1 s
                        TCNT1 = 0;     // vynulovani casovace
                        LED_PORT ^= (1<<LED_PIN_NUM);       // zmenit stav LED
                }
        }
}
```

- Připomínky k programu:
  - Žlutě jsou vyznačeny změny vůči předchozí variantě, která nepoužívala předděličku časovače.

1.5 Realizace dlouhých zpoždění v programu

- Předchozí příklady ukázaly, že **délka zpoždění** realizovaná časovačem má své omezení, které závisí na:
  - **Taktovací frekvence CPU** (z ní je odvozená taktovací frekvence časovače) – čím je nižší, tím delší zpoždění lze realizovat, ale instrukce programu se vykonávají pomaleji.
  - **Použitá předdělička časovače** - čím je větší, tím delší zpoždění lze realizovat.
  - **Bitová šířka čítacího registru časovače** (16-bit časovač dokáže generovat 256x delší zpoždění než 8-bit časovač).
- **Jak realizovat zpoždění dlouhá několik sekund, minut, hodin, dní….?**
  - **Vytvořením vlastního softwarového čítače** – celočíselné proměnné, kterou budeme inkrementovat při každém odměření základní periody, kterou nám bude odměřovat hardwarový časovač.
  - Hardwarový časovač nastavíme např. na odměřování periody 1 sekunda.
  - Z této základní periody si můžeme v SW odvodit dílčí časování dle libosti jako násobky této základní periody (2, 3, 10, 60 sekund….).
- **Pozměňme náš příklad s blikáním LED tak, aby program odměřoval časový interval 60 sekund (LED bude měnit svůj stav každých 60 sekund).**
- **Odpovídající pseudokód:**

```
Nastav port pro LED
Vynuluj časovač
Vynuluj počítadlo sekund
Nastav časovač do režimu Normal
Spusť časovač

WHILE (nekonečná smyčka)
       IF hodnota časovače je rovna nebo větší než 1s THEN
              Vynuluj časovač
              Zvyš počítadlo sekund o 1
              IF počítadlo sekund je rovno nebo větší než 60 THEN
                     Vynuluj počítadlo sekund
                     Změň stav LED
              END IF
       END IF
END WHILE
```

- **Úplný program (pro překladač AVR-GCC):**

```
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT      PORTA
#define LED_DDR       DDRA
#define LED_PIN_NUM   0

// Dokonceny program
int main(void)
{
	unsigned char pocitadloSekund = 0;

	LED_DDR |= (1<<LED_PIN_NUM); // vystup pro LED
	TCNT1 = 0;            // vynulovani casovace
	TCCR1B = (1<<CS11) | (1<<CS10);     // nastaveni a spusteni casovace (normal
	                                    mode, clock Fcpu/64)

	while(1)
	{
		if (TCNT1 >= 15625) // zjisti, jestli uz se odmerila 1 s
		{
			// ubehla 1 s
			TCNT1 = 0;            // vynulovani casovace
			pocitadloSekund++;    // zvyseni pocitadla sekund o 1

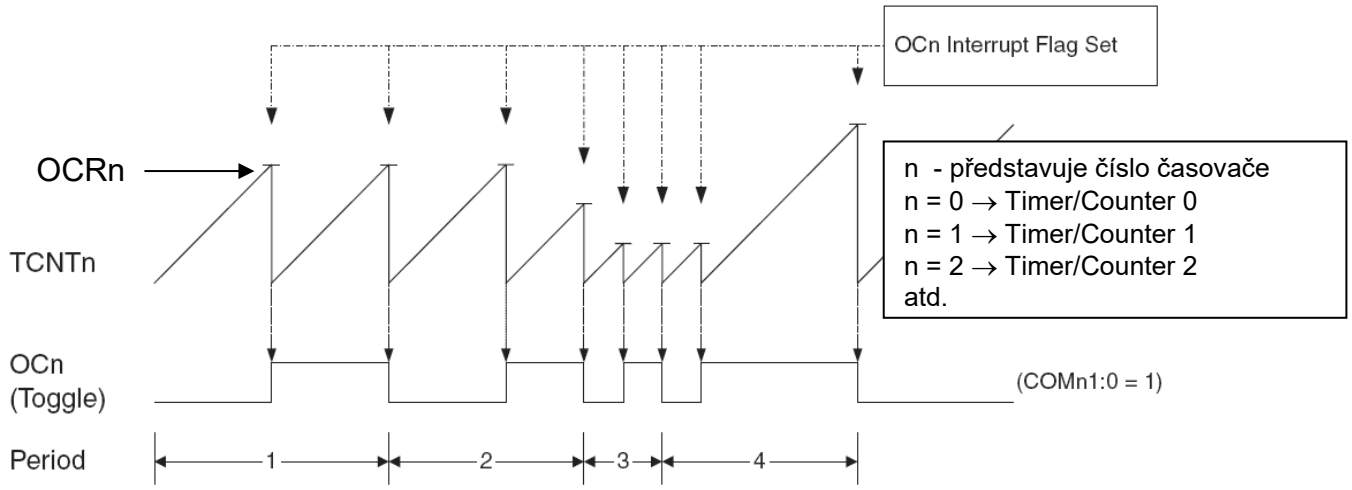
		}

		if (pocitadloSekund >= 60)   // zjisti zda ubehlo 60 s
		{
			// ubehlo 60 s
			pocitadloSekund = 0;  // vynulovani pocitadla sekund
			LED_PORT ^= (1<<LED_PIN_NUM);       // zmenit stav LED
		}
	}
}
```

- Připomínky k programu:
  - Žlutě jsou vyznačeny části, které pracují se softwarovým čítačem.
  - Pro dlouhá zpoždění je potřeba, aby hodiny hardwarového časovače tikaly přesně. Odchylka taktovací frekvence časovače se výrazně projeví do dlouhého zpoždění. Nepřesnost 1 % taktovací frekvence časovače vytvoří na 60 sekundách nepřesnost 0,6 sekundy!
  - **Uvedený princip softwarových čítačů/děličeček lze použít obecně pro odvození dílčích časování od základní periody vytvořené HW časovačem, ne jen pro dlouhá zpoždění.**

1.6 Využití CTC režimu časovače

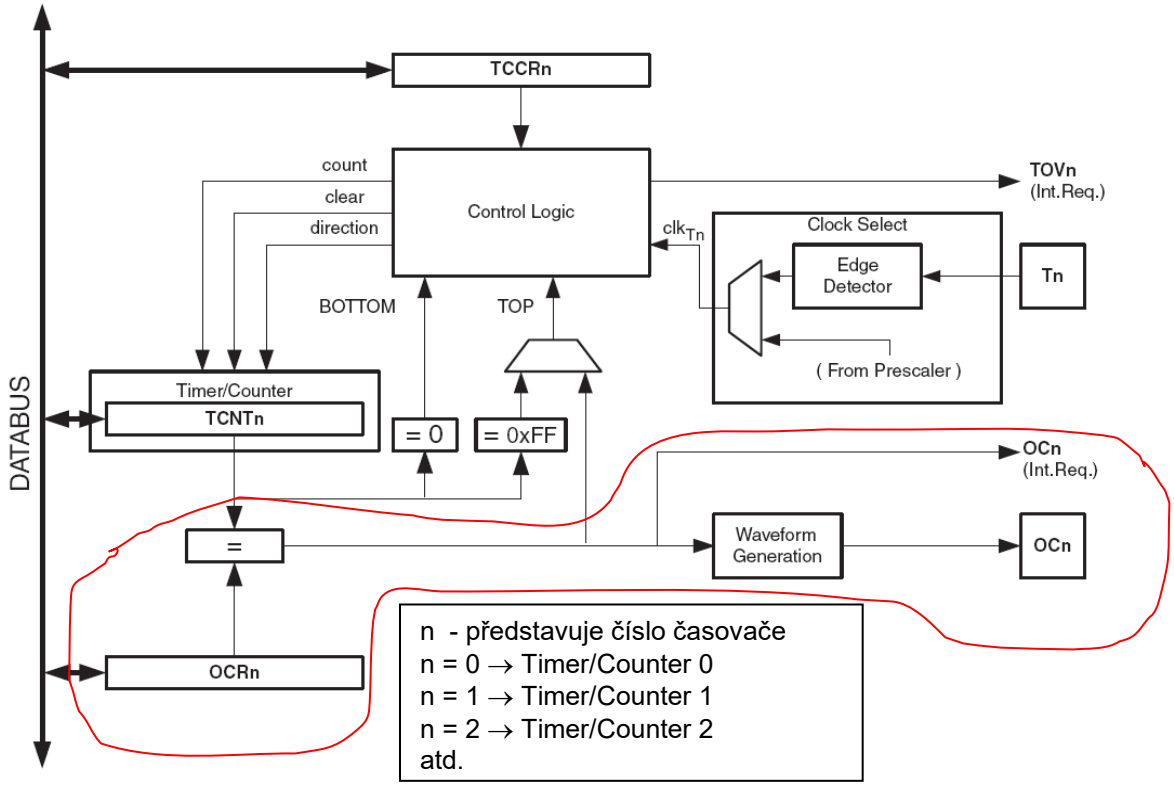
- V předchozích kapitolách jsme časovač používali v normálním režimu a navíc velmi základním způsobem (ruční kontrolou načítané hodnoty, bez využití přerušení).
- Uvedený způsob použití má tyto nevýhody:
  - „Ruční“ komparace vůči požadované hodnotě a „ruční“ nulování čítacího registru – nahradíme HW komparací v režimu CTC (dozvíte se v této kapitole).
  - Nepřesné generování požadovaného zpoždění
    - Okamžik „ručního“ porovnávání závisí na časové náročnosti vykonání kódu v nekonečné smyčce hl. programu.
    - Mezi tím mohl časovač udělat pár tiků nad komparační hodnotu – tím vznikne chyba časování.
    - Tento nedostatek lze odstranit použitím přerušení (dozvíte se v dalších kapitolách).
- **CTC** = **C**lear **T**imer **O**n **C**ompare
  - Hardware časovače při každém tiku zcela nezávisle na programu porovná obsah čítacího registru **TCNT** s komparační hodnotou uloženou v komparačním registru **OCR** (**O**utput **C**ompare **R**egister).
  - Při dosažení komparační hodnoty:
    - se časovač automaticky vynuluje
    - nastaví se příznak komparace **OCn** v registru příznaků **TIFR**



Obrázek 3 Časovač v režimu CTC

  - V programu stačí **„ručně“ kontrolovat pouze příznak komparace**
    - To je o něco rychlejší než porovnávání dvou jednobajtových nebo dvoubajtových (v případě 16-bit časovače) čísel.
  - Nebo **povolit přerušení při komparaci** a **kód umístit do rutiny pro obsluhu přerušení** – z hlediska přesnosti generování času **nejlepší řešení**.

- Nebo **využít hardware časovače, který dokáže automaticky zcela nezávisle na SW ovládat k časovači přidružený pin** – lze na něm generovat obdélníkový signál o frekvenci a střídě 1:1.
    - Např. blikat s LED bez nutnosti psát kód pro ovládání LED, stačí pouze nastavit časovač do módu CTC a povolit generování signálu na výstupním pinu označeném OCn.



Obrázek 4 Struktura 8-bit časovače v AVR

- **TCNTn** (Timer/Counter Register) - čítací registr, 8-bit.
- **OCRn** (Output Compare Register) - předvolba pro komparační jednotku, 8-bit.
- **TCCRn** (Timer/Counter Control Register) - řídící registr (konfigurace časovače)
- **TIFR** (Timer Interrupt Flag Register) – registr s příznaky přerušení (v obr. označené jako Int.Req. **TOVn** a **OCn**)
- **TIMSK** (Timer Interrupt Mask Register) - povolení různých zdrojů přerušení od časovače

1.6.1 CTC režim bez použití přerušení

- Funkci budeme opět demonstrovat na aplikaci blikající LED s periodou 2 s (změna stavu LED každou 1 s).
- V prvopočátku pouze vyměníme normální režim časovače za režim CTC.
- **Odpovídající pseudokód:**

```
Nastav port pro LED
Nastav časovač do režimu CTC
Nastav komparační hodnotu na 1 sekundu
Vynuluj časovač
Spusť časovač

WHILE (nekonečná smyčka)
       IF je příznak komparace nastaven do 1 THEN
              Smaž příznak komparace
              Změň stav LED
       END IF
END WHILE
```

- V režimu CTC po dosažení komparační hodnoty není třeba čítací registr časovače nulovat – čítací registr se nuluje sám automaticky po dosažení komparační hodnoty (přesněji jeden tik po dosažení komparační hodnoty).
- Místo nulování čítacího registru je třeba mazat příznak komparace – hardware časovače ho při odměření další periody zase automaticky nastaví do 1.

- **Rozpracovaný kód (pro překladač AVR-GCC):**

```
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT     PORTA
#define LED_DDR      DDRA
#define LED_PIN_NUM  0

// Rozpracovany program
int main(void)
{
       LED_DDR |= (1<<LED_PIN_NUM); // vystup pro LED

       // TODO: nastaveni casovace do rezimu CTC
       // TODO: nastaveni komparacni hodnoty pro periodu 1 s
       TCNT1 = 0;           // vynulovani casovace
       // spusteni casovace s frekvenci Fcpu/64
       TCCR1B &= ~(1<<CS12);              // bit CS12=0
       TCCR1B |= (1<<CS11) | (1<<CS10);  // bity CS11=1, CS10=1

       while(1)
       {
              if () // TODO: kontrola priznaku komparace
              {
                     // ubehla 1 s

                     // TODO: smazani priznaku komparace

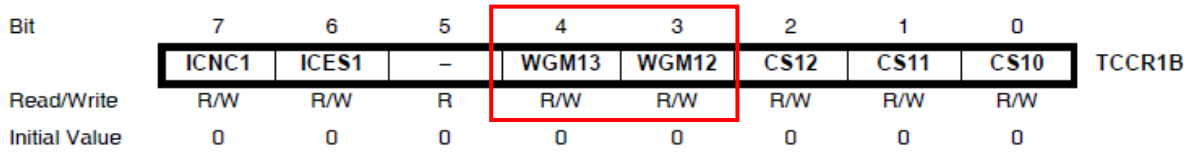
                     LED_PORT ^= (1<<LED_PIN_NUM);              // zmenit stav LED
              }
```

```
        }
}
```

- Komentáře označené TODO musíme zaměnit za kód.
- **Nastavení časovače Timer/Counter 1 do režimu CTC**
  - Timer/Counter 1 má dva kanály (A, B), použijeme pouze jeden z nich, v režimu CTC lze využít pouze kanál A.
    - Každý kanál má vlastní komparační jednotku a jednotku pro generování průběhu (Waveform Generation Unit) na přidruženém výstupním pinu.
    - Oba kanály mají společný čítací registr TCNT1.
  - Pohledem do datasheetu k ATmega32, kapitoly „16-bit Timer/Counter 1“ zjistíme, že máme 2 možnosti nastavení časovače do režimu CTC. Liší se pouze tím, jaký registr je použit pro uložení komparační hodnoty.
    - Komparační hodnota uložena v registru **OCR1A** (Mode = 4).
    - Komparační hodnota uložena v registru **ICR1** (Mode = 12).
    - Je jedno, který registr použijeme, pouze pak musíme testovat správný příznak komparace.
  - Použijeme Mode 4 – komparační hodnota bude uložena v registru **OCR1A**.
    - Bity **WGM10 až WGM13** je třeba nastavit podle Tabulka 6.
    - Bit **WGM10 a WGM11** se nachází v řídícím registru **TCCR1A**.
    - Bity **WGM12** a **WGM13** se nachází v řídícím registru **TCCR1B**.

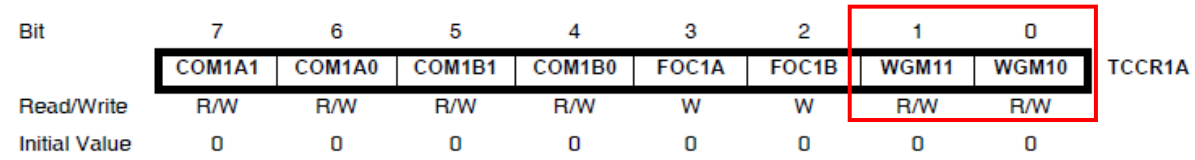
Tabulka 6 Režimy časovače Timer/Counter 1

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP



Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 5 Řídící registr TCCR1B časovače Timer/Counter 1



Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 6 Řídící registr TCCR1A časovače Timer/Counter 1

- Kód pro nastavení Timer/Counter 1 do režimu CTC:

```
TCCR1A &= ~((1<<WGM10) | (1<<WGM11));    // bit WGM10 = 0, WGM11 = 0
TCCR1B |= (1<<WGM12);       // bit WGM12 = 1
TCCR1B &= ~(1<<WGM13);      // bit WGM13 = 0
```

- **Výpočet komparační hodnoty:**
  - Výpočet potřebného počet tiků časovače pro vytvoření požadovaného zpoždění se nemění:

$$\text{počet tiků časovače} = \frac{\text{požadovaný časový interval}}{\text{N} \cdot \text{perioda hodin CPU}}$$

$$\text{počet tiků časovače} = \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{N} \qquad (8)$$

  - Kde: N – předdělička časovače
  - Komparační hodnota ale bude o jeden tik menší:

$$\text{komparační hodnota} = \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{N} - 1 \qquad (9)$$

  - Je to z toho důvodu, že vynulování čítacího registru časovače po dosažení komparační hodnoty trvá právě jeden tik navíc.
    - Nejprve je dosažena komparační hodnota.
    - Následující tik je čítací registr vynulován a je nastaven příznak komparace.

Tabulka 7 Počet tiků časovače s předděličkou pro vytvoření zpoždění 1 s

Předdělička (N)	Počet tiků časovače	Komparační hodnota (režim CTC)
1	1 000 000	999 999
8	125 000	124 999
64	15 625	15 624
256	3 906,25	3 905,25
1024	976,525	975,525

- Komparační hodnota pro zpoždění 1 s při taktovací frekvenci časovače $F_{CPU}/64$ činí 15 624.
    - Tuto hodnotu zapíšeme před spuštěním časovače do komparačního registru **OCR1A**.
- **Rozpracovaný kód (pro překladač AVR-GCC):**

```
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT     PORTA
#define LED_DDR              DDRA
#define LED_PIN_NUM          0

// Dokonceny program
int main(void)
{
	LED_DDR |= (1<<LED_PIN_NUM); // vystup pro LED

	// nastaveni casovace do rezimu CTC
	TCCR1A &= ~((1<<WGM10) | (1<<WGM11));      // bit WGM10 = 0, WGM11 = 0
	TCCR1B |= (1<<WGM12); // bit WGM12 = 1
	TCCR1B &= ~(1<<WGM13);       // bit WGM13 = 0

	OCR1A = 15624;       // nastaveni komparacni hodnoty pro periodu 1 s
	TCNT1 = 0;           // vynulovani casovace
	// spusteni casovace s frekvenci Fcpu/64
	TCCR1B &= ~(1<<CS12);             // bit CS12=0
	TCCR1B |= (1<<CS11) | (1<<CS10);  // bity CS11=1, CS10=1

	while(1)
	{
		if () // TODO: kontrola priznaku komparace
		{
			// ubehla 1 s

			// TODO: smazani priznaku komparace

			LED_PORT ^= (1<<LED_PIN_NUM);              // zmenit stav LED

		}
	}
}
```

- **Do kódu zbývá doplnit manipulaci s příznakem komparace.**
- Použijeme příznak komparace **OCF1A**.
    - Protože používáme komparační jednotku na kanálu A časovače Timer/Counter 1.
    - Příznak **OCF1A** se nachází v registru příznaků **TIFR** (**Timer/Counter Interrupt Flag Register**).
        - Registr obashuje příznaku všech časovačů, v Obrázek 7 jsou bílým podbarvením označeny příznaky náležející časovači Timer/Counter 1.
        - Nás ale zajímá pouze příznak **OCF1A**.

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: This register contains flag bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

Obrázek 7 Registr příznaků TIFR

- Čtení stavu příznaku **OC1FA** je snadné.
- Mazání příznaku **OC1FA** vyžaduje pečlivé pročtení datasheetu, protože je v něm uvedeno toto:

- **Bit 4 – OCF1A: Timer/Counter1, Output Compare A Match Flag**

This flag is set in the timer clock cycle after the counter (TCNT1) value matches the Output Compare Register A (OCR1A).

Note that a Forced Output Compare (FOC1A) strobe will not set the OCF1A Flag.

OCF1A is automatically cleared when the Output Compare Match A Interrupt Vector is executed. Alternatively, OCF1A can be cleared by writing a logic one to its bit location.

  - **Příznak OCF1A se maže tím, že se do něj zapíše logická jednička!**
  - To je běžná praxe u AVR, netýká se pouze příznaku OCF1A, nýbrž všech příznaků žádosti o přerušení u interních periferií AVR.
  - Je třeba pečlivě číst datasheet!

- **Úplný program (pro překladač AVR-GCC):**

```
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT     PORTA
#define LED_DDR      DDRA
#define LED_PIN_NUM  0

// Dokonceny program
int main(void)
{
	LED_DDR |= (1<<LED_PIN_NUM); // vystup pro LED

	// nastaveni casovace do rezimu CTC
	TCCR1A &= ~((1<<WGM10) | (1<<WGM11));      // bit WGM10 = 0, WGM11 = 0
	TCCR1B |= (1<<WGM12);        // bit WGM12 = 1
	TCCR1B &= ~(1<<WGM13);       // bit WGM13 = 0

	OCR1A = 15624;// nastaveni komparacni hodnoty pro periodu 1 s
	TCNT1 = 0;           // vynulovani casovace
	// spusteni casovace s frekvenci Fcpu/64
	TCCR1B &= ~(1<<CS12);              // bit CS12=0
	TCCR1B |= (1<<CS11) | (1<<CS10);  // bity CS11=1, CS10=1

	while(1)
	{
		if (TIFR & (1<<OCF1A)) // kontrola priznaku komparace
		{
			// ubehla 1 s
			TIFR |= (1<<OCF1A);                // smazani priznaku komparace
			LED_PORT ^= (1<<LED_PIN_NUM);      // zmenit stav LED
		}
	}
}
```

1.6.2 CTC režim s přerušením

- Ve všech předchozích příkladech jsme „ručně“ ve smyčce hlavního programu testovali, zda už časovač udělal potřebný počet tiků pro vytvoření požadovaného zpoždění.
    - Tento způsob má nevýhodu v tom, že okamžik testování stavu čítacího registru nebo příznaku komparace závisí na časové náročnosti kódu v hlavní smyčce, což způsobuje nepřesnost v časování (prodloužení zpoždění a navíc variabilní).
    - **Řešení:**
        - Odpovědnost přeneseme na hardware časovače.
        - **Použijeme přerušení od časovače – kód, který se má po odměření zpoždění vykonat umístíme do rutiny pro obsluhu přerušení.**
- **Princip přerušení** (opakování)
    - Pokud je přerušení povoleno a jsou splněny podmínky pro vyvolání přerušení, spustí se speciální rutina nazvaná **ISR** (**I**nterrupt **S**ervice **R**outine) – rutina pro obsluhu přerušení.
    - Přerušení může nastat kdykoliv za běhu programu, pokud jsou přerušení globálně povolena a přerušení od daného zdroje je povoleno.
    - Co se děje při obsluze žádosti o přerušení (zjednodušeně):
        - Hlavní program je přerušen.
        - Následuje odskok do rutiny pro obsluhu přerušení (ISR) a její vykonání.
        - Návrat do hlavního programu a jeho pokračování.
- Zdroje přerušení u časovačů na AVR
    - Přerušení při přetečení časovače (Overflow interrupt)
    - Přerušení při komparaci (Compare Match interrupt)
    - Přerušení od záchytné jednotky (Input Capture interrupt)
- **Pro náš případ blikající LED s periodou 2 s použijeme přerušení při komparaci (Output Compare interrupt).**

- **Odpovídající pseudokód:**

```
Nastav port pro LED
Nastav časovač do režimu CTC
Povol přerušení při komparaci
Povol globálně přerušení
Nastav komparační hodnotu na 1 sekundu
Vynuluj časovač
Spusť časovač

WHILE (nekonečná smyčka)
END WHILE

ISR Timer Compare
       Změň stav LED
END ISR
```

- Vidíme, že kód v nekonečné smyčce hl. programu zcela zmizel (smyčka je prázdná).
- Ovládání LED se přesunulo do ISR.
- Do úvodní inicializace přibyly navíc dvě věci:
  - Povolení přerušení při komparaci časovače.
  - Globální povolení přerušení.

- **Rozpracovaný kód (pro překladač AVR-GCC):**

```
// Rozpracovany program
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT     PORTA
#define LED_DDR      DDRA
#define LED_PIN_NUM  0

int main(void)
{
       LED_DDR |= (1<< LED_PIN_NUM);     // vystup pro LED

       // nastaveni casovace do rezimu CTC
       TCCR1A &= ~((1<<WGM10) | (1<<WGM11));    // bit WGM10 = 0, WGM11 = 0
       TCCR1B |= (1<<WGM12);      // bit WGM12 = 1
       TCCR1B &= ~(1<<WGM13);     // bit WGM13 = 0

       // TODO: povoleni preruseni pri komparaci
       // TODO: globalni povoleni preruseni

       OCR1A = 15625;      // nastaveni komparacni hodnoty pro periodu 1 s
       TCNT1 = 0;          // vynulovani casovace
       // spusteni casovace s frekvenci Fcpu/64
       TCCR1B &= ~(1<<CS12);             // bit CS12=0
       TCCR1B |= (1<<CS11) | (1<<CS10);  // bity CS11=1, CS10=1
       while(1)
       {
       }
}

// TODO: doplnit ISR
```

- Komentáře označené TODO musíme zaměnit za kód.
- Nejprve doplníme rutinu pro obsluhu přerušení (ISR).
- Je třeba vědět, jak se zapisuje ISR, když používáme překladač AVR-GCC a knihovnu AVR-libc.
    - Do programu je třeba připojit hlavičkový soubor `interrupt.h`, který zpřístupní prostředky pro vytvoření ISR a ovládání globálního povolení přerušení (obecně pro zacházení s přerušeními).

```
#include <avr/io.h>
#include <avr/interrupt.h>

// definice pripojeni LED k MCU
#define LED_PORT    PORTA
#define LED_DDR     DDRA
#define LED_PIN     0

// Rozpracovany program
int main(void)
{
        . . .
```

- V AVR-GCC se ISR se zapisuje obecně ve tvaru:

```
ISR(VectorName_vect)
{
        // kod pro obsluhu ISR
}
```

    - Kde `VectorName_vect` je název vektoru přerušení, pro které má být ISR vytvořena. Název se tvoří takto:
        - Podíváme se do tabulky vektorů přerušení v datasheetu AVR na název vektoru.

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow

        - Mezery v názvu vektoru v datasheetu vyměníme za podtržítko _
        - K názvu vektoru doplníme koncovku `_vect`
        - Např. ISR(TIMER1_COMPA_vect)
- **Rozpracovaný kód (pro překladač AVR-GCC):**
    - Po doplnění ISR.

```
// Rozpracovany program
#include <avr/io.h>
#include <avr/interrupt.h>

// definice pripojeni LED k MCU
#define LED_PORT    PORTA
#define LED_DDR     DDRA
#define LED_PIN_NUM 0

int main(void)
{
	LED_DDR |= (1<<LED_PIN_NUM);	// vystup pro LED

	// nastaveni casovace do rezimu CTC
	TCCR1A &= ~((1<<WGM10) | (1<<WGM11));	// bit WGM10 = 0, WGM11 = 0
	TCCR1B |= (1<<WGM12);		// bit WGM12 = 1
	TCCR1B &= ~(1<<WGM13);		// bit WGM13 = 0

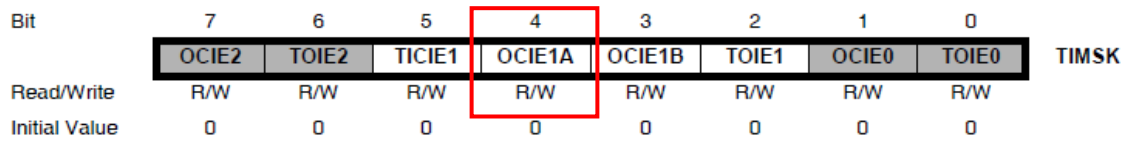
	// TODO: povoleni preruseni pri komparaci
	// TODO: globalni povoleni preruseni

	OCR1A = 15625;		// nastaveni komparacni hodnoty pro periodu 1 s
	TCNT1 = 0;			// vynulovani casovace
	// spusteni casovace s frekvenci Fcpu/64
	TCCR1B &= ~(1<<CS12);			// bit CS12=0
	TCCR1B |= (1<<CS11) | (1<<CS10);	// bity CS11=1, CS10=1
	while(1)
	{
	}
}

ISR(TIMER1_COMPA_vect)
{
	LED_PORT ^= (1<<LED_PIN_NUM);	// zmena stavu LED
}
```

- Připomínky k programu:
    - V ISR nemusíme „ručně“ mazat příznak komparace.
        - Příznak se maže automaticky sám po přechodu do ISR.
    - Nekonečná smyčka v `main()` je zcela prázdná – tj. nic se v nekonečné smyčce hl. programu nevykonává.
        - Snížili jsme zatížení CPU.
        - V nekonečné smyčce hl. programu bychom mohli vykonávat další úlohy nesouvisející s časováním LED nebo MCU mezi obsluhou ISR uspat a snížit tím příkon MCU.

- Do kódu zbývá doplnit dvě věci:
  - **U časovače povolit přerušení při komparaci.**
    - Pohledem do datasheetu AVR ATmega32 zjistíme, že se musí nastavit bit **OCIE1A** (Timer/Counter 1, Output Compare A Match Interrupt Enable) v registru **TIMSK** (Timer/Counter Interrupt Mask Register).



Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: 1. This register contains interrupt control bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

- **Bit 4 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Output Compare A match interrupt is enabled. The corresponding Interrupt Vector (See "Interrupts" on page 42.) is executed when the OCF1A Flag, located in TIFR, is set.

    - Kód:

```
TIMSK |= (1<<OCIE1A); // povoleni preruseni pri komparaci
```

  - **Globálně povolit přerušení.**
    - Pro povolení a zákaz všech přerušení na globální úrovni slouží makra:
    - `sei()` – nastaví bit I v SREG na 1 (globální povolení obsluhy přerušení)
    - `cli()` – nastaví bit I v SREG na 0 (globální zákaz obsluhy přerušení)

- **Úplný program (pro překladač AVR-GCC):**

```
// Dokonceny program
#include <avr/io.h>
#include <avr/interrupt.h>

// definice pripojeni LED k MCU
#define LED_PORT            PORTA
#define LED_DDR             DDRA
#define LED_PIN_NUM         0

int main(void)
{
        LED_DDR |= (1<<LED_PIN_NUM);       // vystup pro LED

        // nastaveni casovace do rezimu CTC
        TCCR1A &= ~((1<<WGM10) | (1<<WGM11));    // bit WGM10 = 0, WGM11 = 0
        TCCR1B |= (1<<WGM12);       // bit WGM12 = 1
        TCCR1B &= ~(1<<WGM13);      // bit WGM13 = 0

        TIMSK |= (1<<OCIE1A);       // povoleni preruseni pri komparaci
        sei(); // globalni povoleni preruseni

        OCR1A = 15625;       // nastaveni komparacni hodnoty pro periodu 1 s
        TCNT1 = 0;           // vynulovani casovace

        // spusteni casovace s frekvenci Fcpu/64
        TCCR1B &= ~(1<<CS12);             // bit CS12=0
        TCCR1B |= (1<<CS11) | (1<<CS10);  // bity CS11=1, CS10=1

        while(1)
        {
        }
}

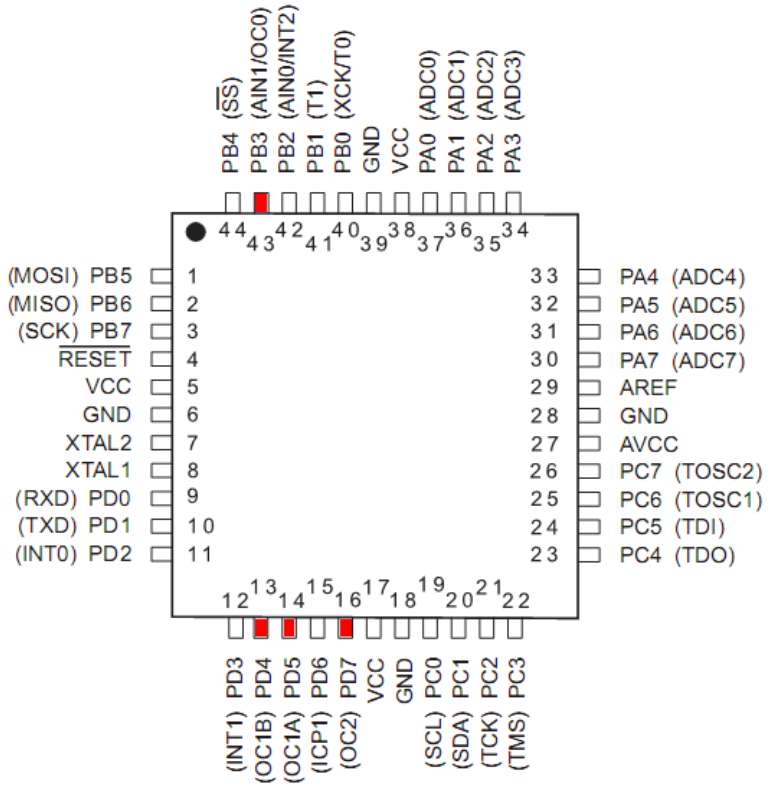
ISR(TIMER1_COMPA_vect)
{
        LED_PORT ^= (1<<LED_PIN_NUM);      // zmena stavu LED
}
```

1.6.3 CTC režim s HW ovládáním výstupního pinu

- Předchozí řešení využívající přerušení pro ovládání LED lze ještě vylepšit.
- Ovládání LED můžeme nechat čistě na hardwaru.
- **Hardware v AVR umožňuje k logice časovače připojit výstupní pin na některém z portů AVR (platí obecně pro všechny režimy časovače, ne jen pro režim CTC).**
  - Tento pin pak nelze ovládat „ručně“, nýbrž je ovládán časovačem.
  - Nastavením časovače do správného režimu lze na tomto pinu generovat obdélníkový průběh o určité frekvenci a střídě.
  - **Hardwarová záležitost** → nevyžaduje obsluhu v SW → CPU se nemusí vůbec starat o generování průběhu na výstupním pinu → časování signálu na výstupním pinu je **přesné**.
  - Piny, které lze propojit s logikou časovače mají alternativní označení **OCnx**.
    - n – číslo časovače (0, 1, 2, …)
    - x – kanál časovače (A, B, …) – pouze u časovačů s více kanály

Tabulka 8 Výstupní piny OCnx přidružené k časovačům (ATmega32)

Pin	Alternativní funkce	Časovač
PB3	OC0	Timer/Counter 0
PD5	OC1A	Timer/Counter 1, kanál A
PD4	OC1B	Timer/Counter 1, kanál B
PD7	OC2	Timer/Counter 2



- **Chování pinů OCnx** se nastavuje bity **COMnx0** a **COMnx1** v řídím registru časovače, např. pro časovač Timer/Counter 1:

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7:6 – COM1A1:0: Compare Output Mode for Compare unit A**
- **Bit 5:4 – COM1B1:0: Compare Output Mode for Compare unit B**

The COM1A1:0 and COM1B1:0 control the Output Compare pins (OC1A and OC1B respectively) behavior. If one or both of the COM1A1:0 bits are written to one, the OC1A output overrides the normal port functionality of the I/O pin it is connected to. If one or both of the COM1B1:0 bit are written to one, the OC1B output overrides the normal port functionality of the I/O pin it is connected to. However, note that the *Data Direction Register* (DDR) bit corresponding to the OC1A or OC1B pin must be set in order to enable the output driver.

When the OC1A or OC1B is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. Table 44 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to a normal or a CTC mode (non-PWM).

Table 44. Compare Output Mode, non-PWM

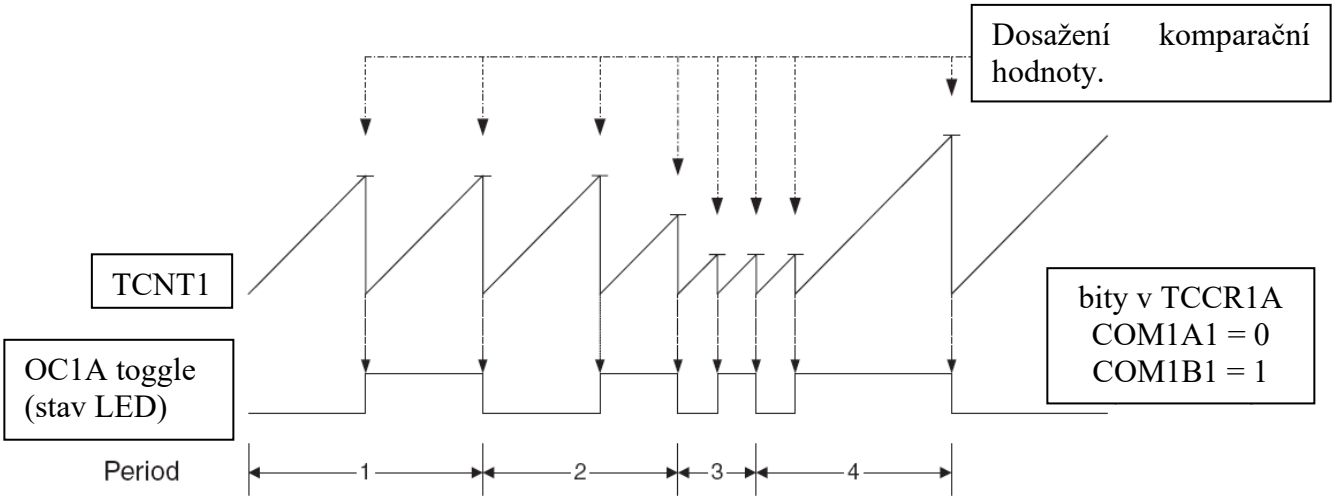
COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on compare match
1	0	Clear OC1A/OC1B on compare match (Set output to low level)
1	1	Set OC1A/OC1B on compare match (Set output to high level)

Obrázek 8 Nastavení chování výstupních pinů OCnx (ATmega32)

- Chování výstupních pinů **OCnx** závisí na režimu časovače (zde zobrazené chování je platné v režimech **Normal** a **CTC**, v režimech generování **PWM** je chování pinů **OCnx** jiné).

- V našem příkladu musíme:
  - **Přepojit LED** z pinu **PA0** na **PD5** (OC1A), abychom jí připojili na výstup **OC1A**, který je ovládán hardwarem časovače **Timer/Counter 1**.
  - **Nastavit chování pinu OC1A** tak, aby při komparaci měnil svůj stav („toggle“), tj. nastavit bit **COM1A0** do **1** a bit **COM1A1** do **0**.

```
TCCR1A &= ~(1<<COM1A1);      // bit COM1A1 = 0
TCCR1A |= (1<<COM1A0);       // bit COM1A0 = 1
```



Obrázek 9 Změna stavu LED při dosažení komparační hodnoty časovače TCTN1

- Odpovídající pseudokód:

```
Nastav port pro LED
Nastav časovač do režimu CTC
Povol ovládání pinu OC1A časovačem
Nastav komparační hodnotu na 1 sekundu
Vynuluj časovač
Spusť časovač

WHILE (nekonečná smyčka)
END WHILE
```

- Vidíme, že zmizela obsluha přerušení a také kód v nekonečné smyčce hl. programu.
- Ovládání LED zcela zajišťuje hardware.
- CPU se nemusí vůbec starat o řízení LED → může vykonávat jiný užitečnější kód.

- **Úplný program (pro překladač AVR-GCC):**

```
// Dokonceny program
#include <avr/io.h>

// definice pripojeni LED k MCU
#define LED_PORT            PORTD
#define LED_DDR             DDRD
#define LED_PIN_NUM         5       // pin OC1A


int main(void)
{
	LED_DDR |= (1<<LED_PIN_NUM);      // vystup pro LED

	// nastaveni casovace do rezimu CTC
	TCCR1A &= ~((1<<WGM10) | (1<<WGM11));    // bit WGM10 = 0, WGM11 = 0
	TCCR1B |= (1<<WGM12);       // bit WGM12 = 1
	TCCR1B &= ~(1<<WGM13);      // bit WGM13 = 0

	// povoleni ovladani pinu OC1A casovacem
```

```
        TCCR1A &= ~(1<<COM1A1);     // bit COM1A1 = 0
        TCCR1A |= (1<<COM1A0);      // bit COM1A0 = 1

        OCR1A = 15624;       // nastaveni komparacni hodnoty pro periodu 1 s
        TCNT1 = 0;           // vynulovani casovace

        // spusteni casovace s frekvenci Fcpu/64
        TCCR1B &= ~(1<<CS12);             // bit CS12=0
        TCCR1B |= (1<<CS11) | (1<<CS10);  // bity CS11=1, CS10=1

        while(1)
        {
        }
}
```

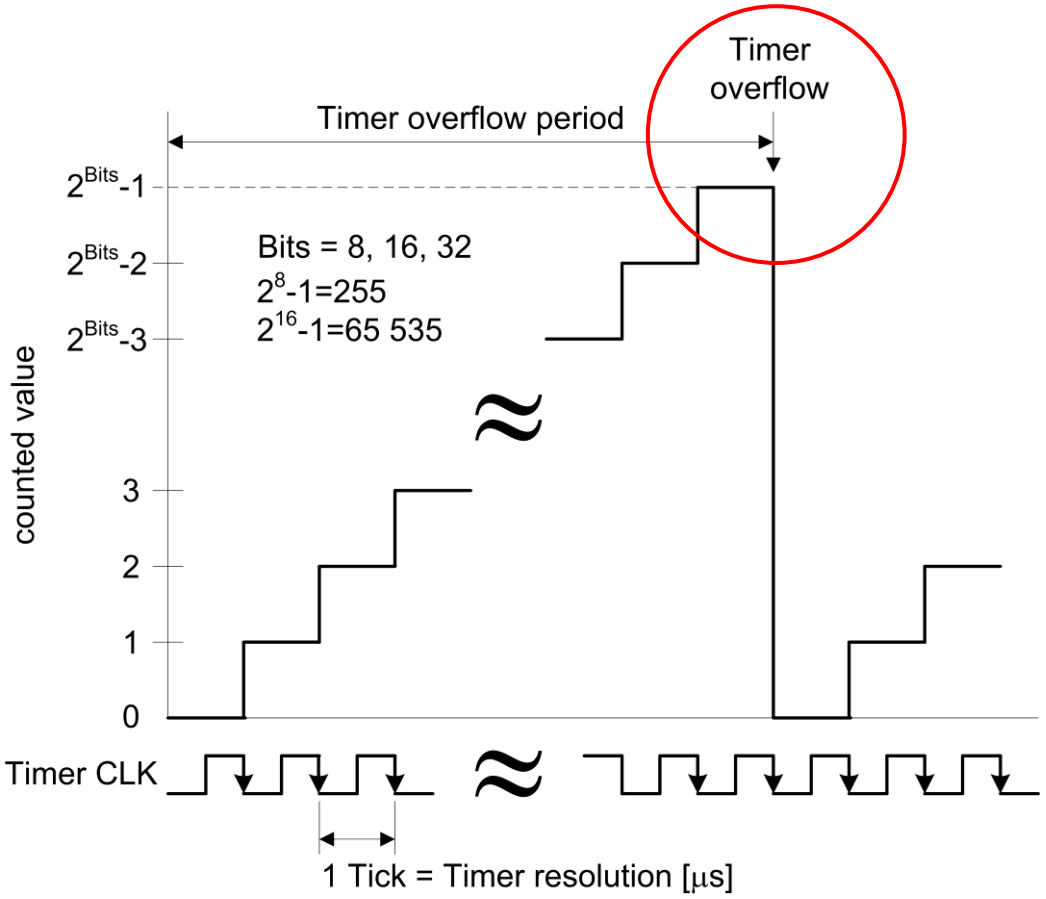
- Připomínky k programu:
  - Žlutě jsou vyznačeny části, které jsou odlišné od předchozí varianty s přerušením.

1.7 Přetečení časovače a jeho využití

- V dříve uvedených případech jsme se nezabývali tím, co se stane, když časovač přeteče, respektive, k čemu přetečení časovače využít. K přetečení časovače totiž v předchozích případech vůbec nedocházelo, nebylo dokonce ani chtěné.
- **Co je to „přetečení“ časovače (Timer Overflow)?**
  - Čítací registr časovače se každým tikem zvýší o 1.
  - Maximální hodnota, která může být uložená v čítacím registru, závisí na bitové šířce časovače (bits).

$$\boxed{\text{max. načítaná hodnota} = 2^{\text{bits}} - 1} \tag{10}$$

    - 8-bit časovač → čítá v rozsahu 0 až 2^8-1 (0 až 255)
    - 16-bit časovač → čítá v rozsahu 0 až 2^{16}-1 (0 až 65 535)
    - 32-bit časovač → čítá v rozsahu 0 až 2^{32}-1 (0 až 4 294 967 295)
  - **Jakmile čítací registr obsahuje max. hodnotu a dojde k dalšímu tiku, časovač „přeteče“ a čítací registr se vynuluje. Časovač běží dál, čítání začíná opět od nuly.**



Obrázek 10 Přeteční časovače

- **K čemu lze přetečení časovače využít?**
  - K vytvoření časování
    - Jednorázová událost „přetečení časovače“ – generování jednorázového zpoždění, vygenerování jednorázového pulsu na výstupním pinu apod.
    - Periodická událost „přetečení časovače“ – časování aplikace, generování periodického signálu na výstupním pinu, atd…
- **Jak rychle časovač přeteče?**
  - To závisí na taktovací frekvenci časovače a bitové šířce časovače:

    perioda přetečení časovače = počet tiků do přetečení ∗ trvání jednoho tiku (11)

    $$\boxed{\text{počet tiků do přetečení} = 2^{\text{bits}}} \quad (12)$$

    $$\boxed{\text{trvání jednoho tiku} = \frac{\text{N}}{\text{taktovací frekvence CPU}}} \quad (13)$$

    - Kde: N – předdělička časovače (1, 8, 64, ….)
      bits – bitová šířka časovače (8, 16, 32)
    - **Výsledný vztah:**

    $$\boxed{\text{perioda přetečení časovače} = \frac{2^{bits} \cdot \text{N}}{\text{taktovací frekvence CPU}}} \quad (14)$$

  - Můžeme vypočítat periodu přetečení časovače pro různé F_{CPU}.

Tabulka 9 Perioda přetečení časovače pro F_{CPU} = 1 MHz

	Perioda přetečení časovače	
Předdělička (N)	**8-bit časovač**	**16-bit časovač**
1	256 μs	65,536 ms
8	2048 μs	524,288 ms
64	16 384 μs	4,194304 s
256	65 536 μs	16,777216 s
1024	262 144 μs	67,108864 s

Tabulka 10 Perioda přetečení časovače pro F_{CPU} = 14,7456 MHz

	Perioda přetečení časovače	
Předdělička (N)	**8-bit časovač**	**16-bit časovač**
1	17,361 μs	4,444 ms
8	138,888 μs	35,555 ms
64	1,111 ms	284,444 ms
256	4,444 ms	1,13777 s
1024	17,777 ms	4,55111 s

- Na rozdíl od režimu **CTC** jsme při použití principu přetečení velmi omezeni při odměřování časového intervalu (zpoždění) – **odměřovaný časový interval (zpoždění) je roven periodě přetečení časovače**.
    - To znamená, že můžeme vybírat pouze z předem daných hodnot zpoždění a měnit ho můžeme jen v hrubých krocích (daných výběrem předděliček).
    - To je poměrně omezující vlastnost, ztrácíme flexibilitu a přesnost, kterou jsme měli v módu CTC.
    - **Periodu přetečení časovače můžeme ovlivnit pouze těmito způsoby:**
        - **Změnou předdělička** – čím větší předdělička, tím je možné odměřovat delší zpoždění.
        - **Výběrem časovače** (8-bit nebo 16-bit) – bitově širší časovač přetéká pomaleji, je možné odměřovat delší zpoždění.
        - **Změnou taktovací frekvence CPU** – čím nižší tím je možné odměřovat delší zpoždění.
- **Nyní upravme náš příklad blikající LED tak, abychom místo režimu CTC využili přetečení časovače.**
- Použijeme opět 16-bit Timer/Counter 1, protože dokáže odměřovat delší zpoždění než 8-bit časovač Timer/Counter 0, a musíme pro něj vybrat vhodnou předděličku.
    - Předpokládejme taktovací frekvenci CPU opět 1 MHz (F_{CPU} = 1 MHz).
    - **LED má blikat s periodou 2 s (tj. každou 1 sekundu měnit stav)**, takže potřebujeme, **aby časovač přetékal s periodou 1 sekunda**.
    - **Periodu 1 sekunda nelze splnit**, protože **nejvhodnější předdělička N = 8** zajistí periodu přetečení časovače **524,288 ms**. Další nejbližší předdělička je N = 64, se tou by ale byla perioda přetečení již příliš dlouhá (4,194 s).
    - Spokojme se s periodou přetečení 0,524288 s, LED tedy bude blikat cca 2x rychleji než chceme.
- Pojďme vytvořit program.
- Použijeme přetečení časovače a povolíme přerušení, ovládání LED umístíme do ISR přetečení časovače.
- **Odpovídající pseudokód:**

```
Nastav port pro LED
Nastav časovač do normálního režimu
Povol přerušení při přetečení časovače
Povol globálně přerušení
Vynuluj časovač
Spusť časovač s taktovací frekvencí Fcpu/8

WHILE (nekonečná smyčka)
END WHILE

ISR Timer Overflow
       Změň stav LED
END ISR
```

- **Rozpracovaný kód (pro překladač AVR-GCC):**

```
// Rozpracovany program
#include <avr/io.h>
#include <avr/interrupt.h>

// definice pripojeni LED k MCU
#define LED_PORT            PORTA
#define LED_DDR             DDRA
#define LED_PIN_NUM         0

int main(void)
{
        LED_DDR |= (1<<LED_PIN_NUM);      // vystup pro LED

        // TODO: nastaveni casovace do normalniho rezimu

        // TODO: povoleni preruseni pri preteceni casovace

        sei(); // globalni povoleni preruseni

        // vynulovani casovace (po resetu se nemusi delat, TCNT1 je po resetu vynulovan)
        TCNT1 = 0;

        // TODO: spusteni casovace s frekvenci Fcpu/8

        while(1)
        {
        }
}

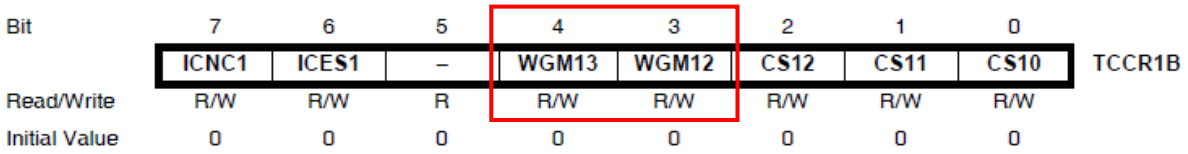
// TODO: doplnit ISR
```

- Opět musíme TODO komentáře nahradit kódem.
- **Nastavení časovače do normálního režimu:**
    - Pohledem do datasheetu ATmega32 kapitoly „16-bit Timer/Counter 1 – Registr Description“ zjistíme, že bity **WGM10** až **WGM13** v řídícím registrech **TCCR1A** a **TCCR1B** musíme vynulovat.
    - Poznámka:
        - Po resetu MCU je obsah registrů TCCR1A i TCCR1B nulový, takže po resetu nemusíme vůbec bity **WGM10:13** nulovat.
        - Pokud, ale časovač byl předtím nastaven do jiného režimu, musíme tyto bity vynulovat.

Tabulka 11 Režimy časovače Timer/Counter 1

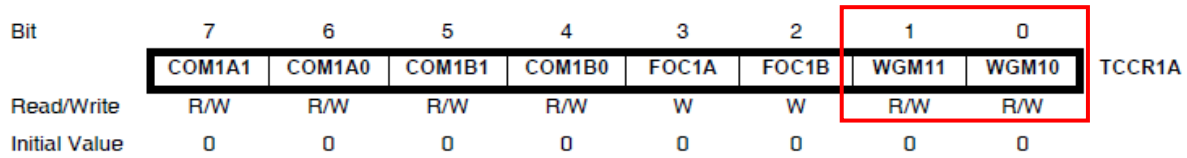
Table 47. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM



Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 11 Řídící registr TCCR1B časovače Timer/Counter 1



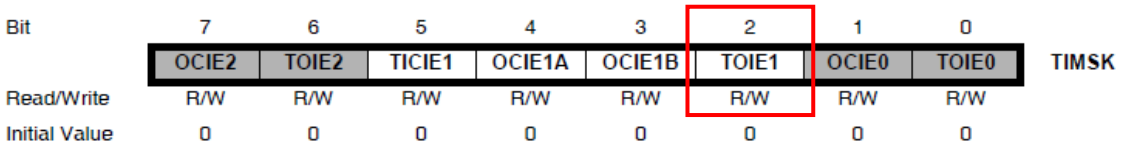
Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 12 Řídící registr TCCR1A časovače Timer/Counter 1

- Kód pro nastavení Timer/Counter 1 do režimu Normal:

```
TCCR1A &= ~((1<<WGM10) | (1<<WGM11));    // bit WGM10 = 0, WGM11 = 0
TCCR1B &= ~((1<<WGM12) | (1<<WGM13));    // bit WGM12 = 0, WGM13 = 0
```

- **Povolení přerušení při přetečení časovače:**
  - Pohledem do datasheetu ATmega32 zjistíme, že musíme nastavit bit **TOIE1** v registru **TIMSK**.



Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Note: 1. This register contains interrupt control bits for several Timer/Counters, but only Timer1 bits are described in this section. The remaining bits are described in their respective timer sections.

- **Bit 2 – TOIE1: Timer/Counter1, Overflow Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter1 Overflow Interrupt is enabled. The corresponding Interrupt Vector (See "Interrupts" on page 42.) is executed when the TOV1 Flag, located in TIFR, is set.

  - Kód pro povolení přerušení při přetečení časovače Timer/Counter 1:

```
TIMSK |= (1<<TOIE1); // povoleni preruseni pri preteceni casovace 1
```

- **Vytvoření ISR:**
  - Musíme se podívat do datasheetu na název vektoru do tabulky vektorů přerušení (kapitola „Interrupts" v datasheetu ATmega32).

Table 18. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow

- Kód ISR pro obsluhu přerušení při přetečení časovače Timer/Counter 1:

```
ISR(TIMER1_OVF_vect)
{
        LED_PORT ^= (1<<LED_PIN_NUM);      // zmena stavu LED
}
```

- Zbývá už jen nastavit taktovací frekvenci časovač na hodnotu $F_{CPU}/8$:
  - Opět pohledem do datasheetu ATmega32….
  - Je třeba nastavit bity **CS10** až **CS12** v řídícím registru **TCCR1B** na správné hodnoty.

Tabulka 12 Výběr zdroje kmitočtu pro časovač Timer/Counter 1

Table 48. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Poznámka:
    - Po resetu MCU je obsah registru **TCCR1B** nulový, takže po resetu nemusíme bity **CS10 a CS12** nulovat.
    - Pokud, ale časovač předtím běžel, musíme všechny tyto bity vynulovat.

- Kód pro spuštění časovače na frekvenci $F_{CPU}/8$:

```
// spusteni casovace s frekvenci Fcpu/8
TCCR1B &= ~((1<<CS10) | (1<<CS12));      // po resetu se nemusi CS10 a CS12 nulovat, protoze registr TCCR1B je po resetu vynulovan
TCCR1B |= (1<<CS11);
```

- **Úplný program (pro překladač AVR-GCC):**

```
// Dokonceny program
#include <avr/io.h>
#include <avr/interrupt.h>

// definice pripojeni LED k MCU
#define LED_PORT            PORTA
#define LED_DDR             DDRA
#define LED_PIN_NUM         0

int main(void)
{
	LED_DDR |= (1<<LED_PIN_NUM);      // vystup pro LED

	// nastaveni casovace do normalniho rezimu
	// (po resetu se nemusi delat, protoze registry TCCR1A a B jsou vynulovany -> casovac jiz je po resetu v normalnim rezimu)
	TCCR1A &= ~((1<<WGM10) | (1<<WGM11));    // bit WGM10 = 0, WGM11 = 0
	TCCR1B &= ~((1<<WGM12) | (1<<WGM13));    // bit WGM12 = 0, WGM13 = 0

	TIMSK |= (1<<TOIE1); // povoleni preruseni pri preteceni casovace 1
	sei(); // globalni povoleni preruseni

	// vynulovani casovace (po resetu se nemusi delat, TCNT1 je po resetu vynulovan)
	TCNT1 = 0;

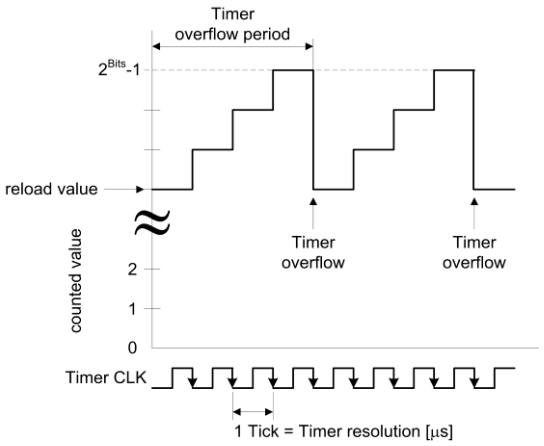
	// spusteni casovace s frekvenci Fcpu/8
	TCCR1B &= ~((1<<CS10) | (1<<CS12));      // po resetu se nemusi bity CS10 a CS12 nulovat, protoze registr TCCR1B je po resetu vynulovan
	TCCR1B |= (1<<CS11); // bit CS11=1

	while(1)
	{
	}
}

ISR(TIMER1_OVF_vect)
{
	LED_PORT ^= (1<<LED_PIN_NUM);     // zmena stavu LED
}
```

1.8 Předvolba časovače v režimu Normal

- I přes uvedenou neschopnost využít přetečení časovače v režimu Normal pro odměřování libovolného časového intervalu, lze chování časovače upravit tak, aby se choval podobně jako v režimu CTC.
    - Tímto prostředkem je zavedení **předvolby časovače**.
- **Porovnání režimu Normal a CTC**
    - **CTC**
        - Fixní BOTTOM hodnota (rovna nule), ze které časovač začíná počítat tiky.
        - Nastavitelná TOP hodnota, která se nastaví podle požadovaného časového intervalu, který má časovač odměřit.
    - **Normal**
        - Fixní BOTTOM hodnota (rovna nule), ze které časovač začíná počítat tiky.
        - Fixní TOP hodnota – dána bitovou šířkou časovače (rovna 2^{bits}-1).
        - → omezená schopnost odměřit libovolný časový interval
        - **Řešení: zavedení předvolby časovače → flexibilní BOTTOM hodnota**
- **Princip softwarové předvolby časovače v režimu Normal**
    - Cílem je, aby rozdíl TOP – BOTTOM hodnot byl nastavitelný.
    - TOP hodnota je fixní, tu nelze měnit, je určena bitovou šířkou časovače.
    - BOTTOM hodnotu lze udělat nastavitelnou tím, že se v SW „ručně“ nastaví čítací registr časovače na hodnotu v rozsahu 0 až 2^{bits}-1 = **předvolba časovače (reload value)**.
        - A to před prvním spuštěním časovače.
        - A znovu po každém přetečení časovače.
        - Tím se urychlí přetečení časovače tak, aby přetékal podle požadovaného časového intervalu, který má časovač odměřit.



- **Rozpracovaný kód s implementací softwarové předvolby časovače v režimu Normal (pro překladač AVR-GCC):**

```
#include <avr/io.h>
#include <avr/interrupt.h>

// definice pripojeni LED k MCU
#define LED_PORT            PORTA
#define LED_DDR             DDRA
#define LED_PIN_NUM         0

int main(void)
{
	LED_DDR |= (1<<LED_PIN_NUM);	// vystup pro LED

	// nastaveni casovace do normalniho rezimu
	// (po resetu se nemusi delat, protoze registry TCCR1A a B jsou vynulovany ->
	casovac jiz je po resetu v normalnim rezimu)
	TCCR1A &= ~((1<<WGM10) | (1<<WGM11));	// bit WGM10 = 0, WGM11 = 0
	TCCR1B &= ~((1<<WGM12) | (1<<WGM13));	// bit WGM12 = 0, WGM13 = 0

	TIMSK |= (1<<TOIE1); // povoleni preruseni pri preteceni casovace 1
	sei(); // globalni povoleni preruseni

	// TODO: nastavit predvolbu casovace pred prvnim spustenim casovace

	// spusteni casovace s frekvenci Fcpu/8
	TCCR1B &= ~((1<<CS10) | (1<<CS12));	// po resetu se nemusi bity CS10 a CS12
	nulovat, protoze registr TCCR1B je po resetu vynulovan
	TCCR1B |= (1<<CS11); // bit CS11=1

	while(1)
	{
	}
}

ISR(TIMER1_OVF_vect)
{
	// TODO: nastavit predvolbu casovace po kazdem pretecni
	LED_PORT ^= (1<<LED_PIN_NUM);	// zmena stavu LED
}
```

  - TODO komentáře je nutné opět vyměnit za funkční kód

- **Jak vypočítat předvolbu časovače (reload value)?**
  - Spočítáme počet tiků do přetečení časovače na základě požadovaného časového intervalu, který má časovač odměřit (vzorec už známe, není nový):

$$\text{počet tiků časovače} = \frac{\text{požadovaný časový interval}}{\text{perioda hodin časovače}} \quad (15)$$

  - Nyní odečteme potřebný počet tiků časovače od maximální hodnoty, kterou pojme čítací registr (2^{Bits}-1) a přičteme jedničku, která zohledňuje jeden tik navíc na přetečení:

$$\text{předvolba} = 2^{Bits}\text{-1 - počet tiků časovače +1} = 2^{Bits}\text{ - počet tiků časovače} \quad (16)$$

- **Výsledný vztah pro výpočet předvolby se zavedením deličky časovače:**

$$\boxed{\text{předvolba} = 2^{\text{Bits}} - \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{N}} \tag{17}$$

  - Kde: N – předdělička časovače (1, 8, 64, …)
    F_{CPU} – taktovací frekvence CPU
    Bits – bitová šířka časovače (8, 16, 32)

- Pro náš příklad blikání LED s periodou 2 s (změna stavu LED každou 1 s) je předvolba rovna hodnotě:

$$\boxed{\text{předvolba} = 2^{16} - 1 \cdot \frac{1 \cdot 10^{6}}{64} = 65536 - 15625 = 49911} \tag{18}$$

  - N = 64
  - F_{CPU} = 1 MHz
  - Bits = 16 (16-bit časovač Timer/Counter 1)

- **Úplný program (pro překladač AVR-GCC):**

```
#include <avr/io.h>
#include <avr/interrupt.h>

// definice pripojeni LED k MCU
#define LED_PORT            PORTA
#define LED_DDR             DDRA
#define LED_PIN_NUM         0

#define RELOAD_TCNT1        49911  // predvolba casovace TCNT1

int main(void)
{
        LED_DDR |= (1<<LED_PIN_NUM);      // vystup pro LED

        // nastaveni casovace do normalniho rezimu
        // (po resetu se nemusi delat, protoze registry TCCR1A a B jsou vynulovany ->
        casovac jiz je po resetu v normalnim rezimu)
        TCCR1A &= ~((1<<WGM10) | (1<<WGM11));    // bit WGM10 = 0, WGM11 = 0
        TCCR1B &= ~((1<<WGM12) | (1<<WGM13));    // bit WGM12 = 0, WGM13 = 0

        TIMSK |= (1<<TOIE1); // povoleni preruseni pri preteceni casovace 1
        sei(); // globalni povoleni preruseni

        // nastaveni predvolby casovace pred prvnim spustenim casovace
        TCNT1 = RELOAD_TCNT1;

        // spusteni casovace s frekvenci Fcpu/64
        TCCR1B &= ~(1<<CS12);       // po resetu se nemusi bit CS12 nulovat, protoze
                                    registr TCCR1B je po resetu vynulovan
        TCCR1B |= (1<<CS11) | (1<<CS10);  // bity CS11=1, CS10=1

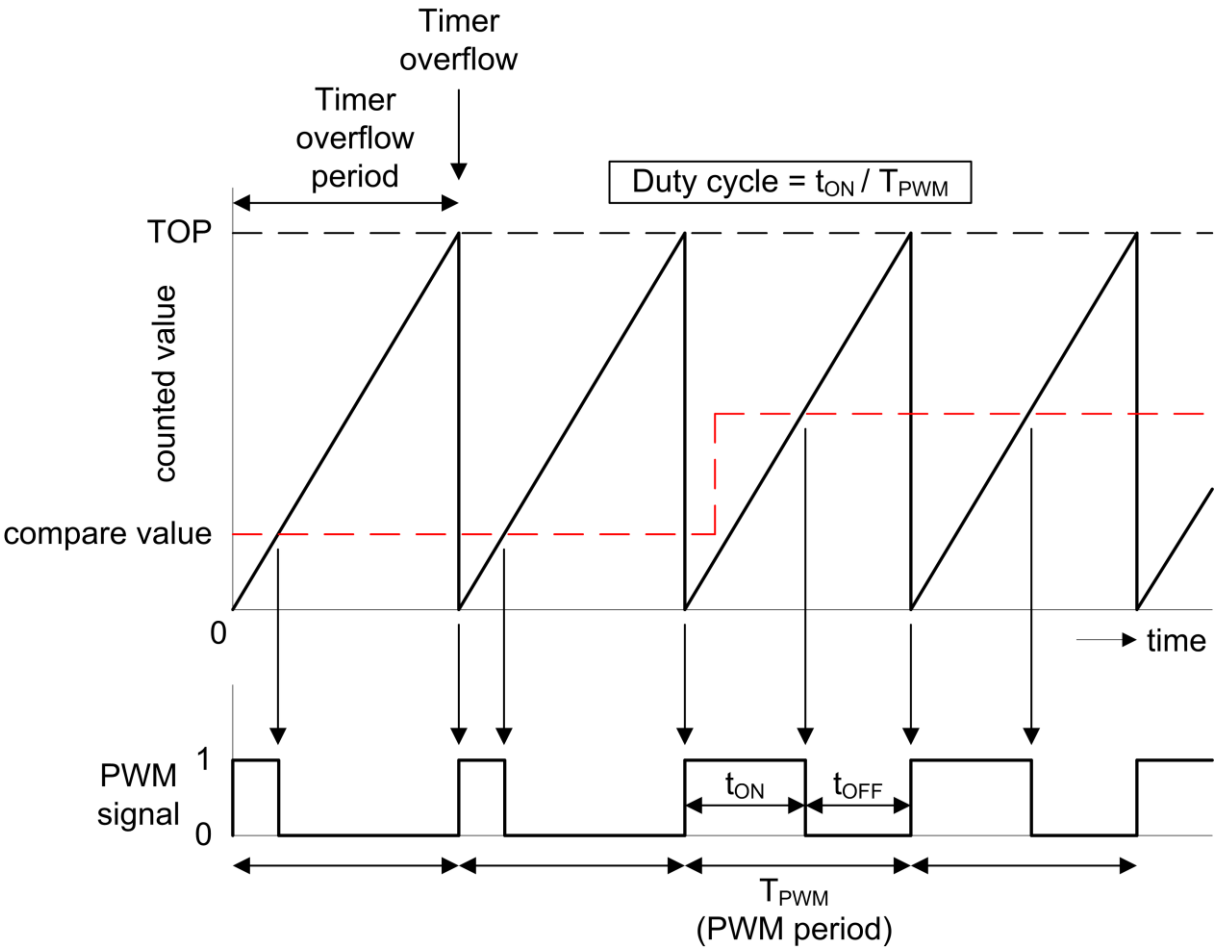
        while(1)
        {
        }
}

ISR(TIMER1_OVF_vect)
{
        TCNT1 = RELOAD_TCNT1;             // nastaveni predvolby casovace
        LED_PORT ^= (1<<LED_PIN_NUM);     // zmena stavu LED
}
```

- **Nevýhody SW předvolby časovače:**
    - Nepřesné odměřování času (prodloužení časového intervalu) pokud dojde ke zpožděnému vstupu do ISR obsluhy přetečení časovače, tedy k pozdnímu zápisu předvolby do čítacího registru (časovač mezi tím mohl udělat několik tiků).

1.9 Generování PWM

- **PWM** = **P**ulse **W**idth **M**odulation (pulsně šířková modulace)
- Princip generování PWM pomocí časovače:



Obrázek 13 Princip generování PWM pomocí časovače

  - Perioda přetečení časovače = perioda PWM
  - TOP hodnota určuje
    - periodu PWM (společně s taktovací frekvencí časovače)
    - rozlišení ve střídě (čím je TOP větší, tím více kroků a tím jemnější nastavení střídy)
  - **střída [-] = doba zapnutí / perioda PWM = t_{ON}/T_{PWM}**
  - Komparační hodnota určuje střídu:

$$\text{duty cycle} = \frac{\text{compare value}}{\text{TOP}} \tag{19}$$

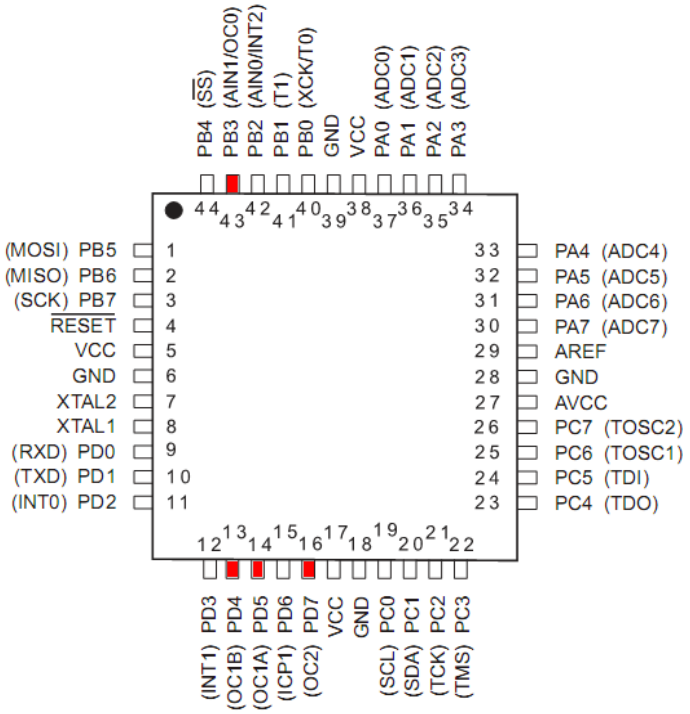
- **Způsoby vytvoření PWM signálu**
  - **Ručně** – časovač v režimu Normal
    - Při komparaci nastavit pin OCn = 0
    - Při přetečení nastavit pin OCn = 1
    - Střída se nastavuje pomocí komparační úrovně (OCRn)
    - Nevýhoda – nutná obsluha přerušení – latence – problém při vyšších frekvencích PWM (MCU je plně zaneprázdněn generováním PWM, nezbývá čas na jiné úkoly)
  - **Hardwarově** – pomocí PWM módů časovače
    - Signál PWM je generován na výstupním pinu hardwarem časovače, SW pouze nastavuje požadovanou střídu skrz komparační hodnotu.
    - Odpadá obsluha přerušení – snížení zatížení CPU

1.9.1 Piny s hardwarovou funkcí PWM

PWM lze u MCU AVR generovat pouze na vybraných pinech, které mají alternativní funkci Output Comapre (piny označené OCx, kde x je číslo časovače).

Na ATmega32 jsou to tyto piny:

Pin	Příslušnost pinu k časovači
PB3 (OC0)	8-bit Timer/Counter 0
PD4 (OC1B)	16-bit Timer/Counter 1 kanál B
PD5 (OC1A)	16-bit Timer/Counter 1 kanál A
PD7 (OC2)	8-bit Timer/Counter 2



Obrázek 14 Piny s hardwarovou funkcí PWM na MCU ATmega32

1.9.2 Režimy hardwarového generování PWM na AVR

- Na MCU AVR existují 3 základní režimy hardwarového generování PWM signálu:
  - **Fast PWM**
  - **Phase Correct PWM**
  - **Phase and Frequency Correct PWM**
- Režimy se liší:
  - Způsobem generování PWM
    - Nesymetrická nebo symetrická PWM.
  - (Ne)schopností nastavit 0 % střídu PWM.
  - Rozlišením ve střídě PWM (pevná nebo nastavitelná TOP hodnota).
    - Nastavitelná TOP hodnota umožňuje přesněji nastavit požadovanou frekvenci PWM (ne jen v hrubých krocích daných periodou přetečení časovače pokud je TOP hodnota pevná).
  - (Ne)schopností plynule měnit frekvenci PWM za chodu (umí jen jeden z režimů).

Tabulka 13 Porovnání režimů PWM na MCU AVR

Režim PWM	Symetrie pulzu vůči periodě	Min. střída	Max. střída	Max. frekvence	Lze plynule měnit frekvenci PWM za chodu?
Fast PWM	NE	větší než 0 %	100 %		NE
Phase Correct PWM	ANO	0 %	100 %	cca poloviční vůči Fast PWM	NE
Phase and Frequency Correct PWM	ANO	0 %	100 %	cca poloviční vůči Fast PWM	ANO

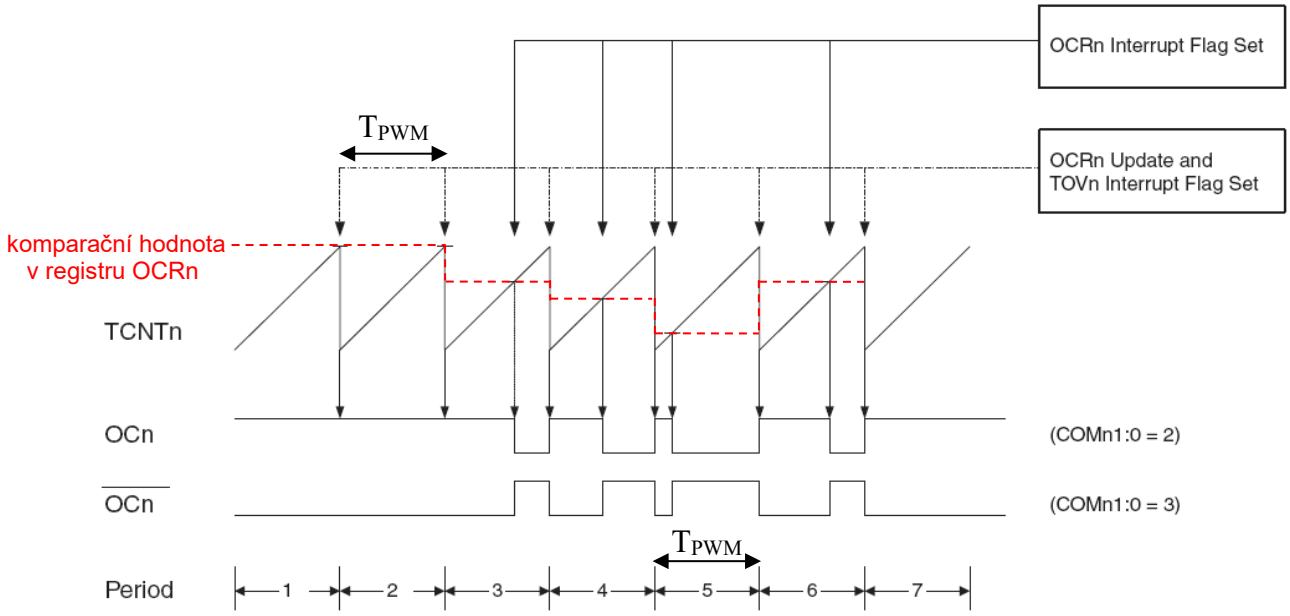
1.9.3 Časovač Timer/Counter 0

1.9.3.1 Fast PWM mód

- Hardwarové generování **nesymetrické PWM**.
  - Puls je zarovnaný na začátek periody PWM.

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Obrázek 15 Režim Fast PWM u Timer/Counter 0



Obrázek 16 Princip generování signálu PWM (režim Fast PWM, časovač Timer/Counter 0)

- **Signál PWM je generován na výstupním pinu OC0**
  - Při dosažení komparační úrovně pin OC0 automaticky nastaví do 0, při přetečení časovače se pin OC0 automaticky nastaví do 1 (nebo obráceně, podle nastavené polarity signálu na výstupním pinu OC0 - neinvertovaná vs. invertovaná PWM).
- **Polarita PWM signálu na pinu OC0**
  - neinvertovaná PWM vs. invertovaná PWM – viz Obrázek 16. Volba proběhne nastavením bitů **COM00** a **COM01** v registru **TCCR0**.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

Obrázek 17 Nastavení chování výstupního pinu OC0 (režim Fast PWM)

- Při komparaci a přetečení se opět nastavují příslušné příznaky v registru příznaků TIFR a lze využít i jim odpovídající přerušení (ale není vůbec nutné ho zapínat a obsluhovat).
- **Frekvence PWM** je určena taktovací frekvencí AVR a předděličkou časovače.
  - Nelze hýbat s TOP hodnotou (**TOP = 255**), časovač vždy čítá **od 0 do 255**.

$$f_{OC0} = \frac{F_{CPU}}{N \cdot 256} \tag{20}$$

Kde: f_{OC0} – frekvence PWM signálu na pinu OC0
N - dělící poměr předděličky časovače
F_{CPU} – taktovací frekvence CPU

Tabulka 14 Frekvence PWM v závislosti na taktovací frekvenci časovače (režim Fast PWM, 8-bit časovač)

Předdělička (N)	Frekvence PWM při F_{CPU} = 1 MHz	Frekvence PWM při F_{CPU} = 14,7456 MHz
1	3 906,25 Hz	57 600 Hz
8	488,28 Hz	7200 Hz
64	61,04 Hz	900 Hz
256	15,26 Hz	225 Hz
1024	3,81 Hz	56,25 Hz

- Střída PWM je určena komparační hodnotou v registru OCR0
  - Rozlišení 8 bitů (256 kroků střídy).

$$\text{duty cycle} = \frac{OCR0}{255} \tag{21}$$

Kde: *duty cycle* – požadovaná střída [-] (v rozsahu 0 až 1)
OCR0 – hodnota zapsaná v komparačním registru OCR0
255 – TOP hodnota časovače

- **Chování PWM signálu v krajních hodnotách střídy (0 a 100 %):**
  - Požadovaná střída 100% - lze - do OCR0 zapsat 255 - signál na pinu OC0 se nebude měnit (bude trvale v 0 nebo v 1 – závisí na nastavení bitů COM00 a COM01).
  - Požadovaná střída 0% - nelze. Když OCR0 = 0 (požadovaná střída 0 %), v signálu na pinu OC0 se budou objevovat úzké špičky, jejichž délka trvání bude rovna 1 tiku časovače (nelze docílit střídy rovné 0 %).

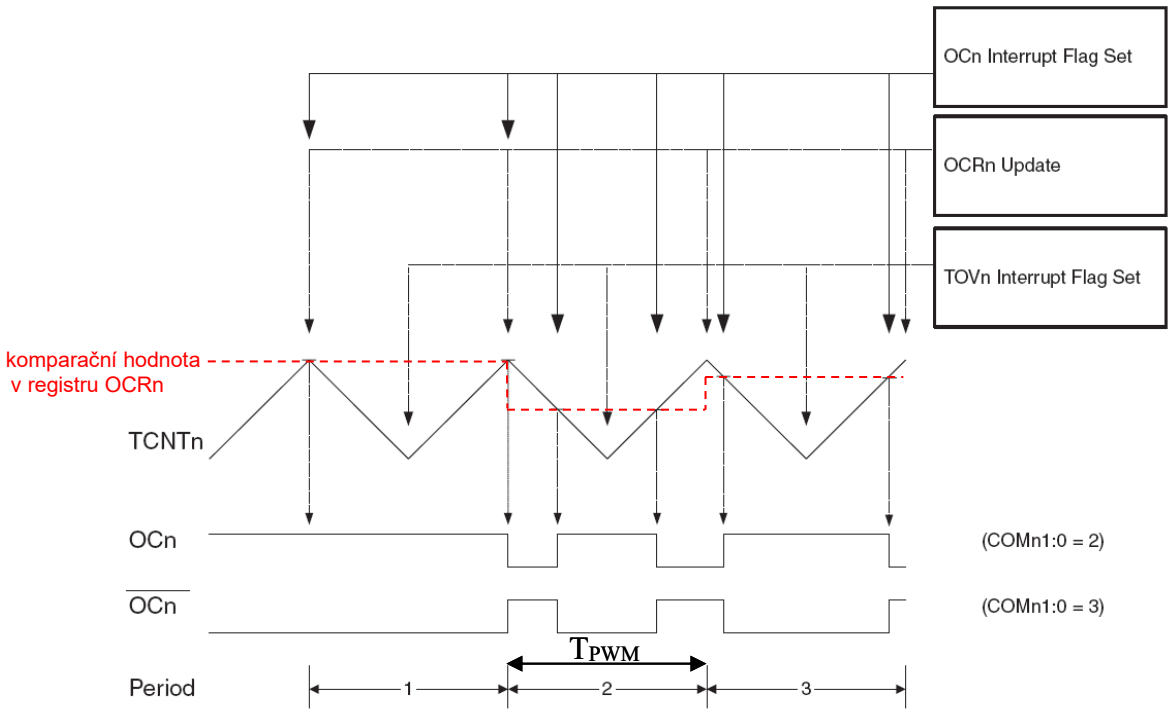
- **Komparační hodnotu lze do registru OCR0 zapisovat kdykoliv, protože:**
  - Registr OCR0 je bufferován.
  - Zápis do OCR0 v SW způsobí zapsání hodnoty do bufferu OCR0.
  - Hardware přepíše komparační hodnotu z bufferu do registru OCR0 až ve správném okamžiku – až po dosažení TOP hodnoty.
  - Toto zabraňuje vzniku tzv. „glitche“, protože by se mohlo stát, že by při přímém zápisu do registru OCR0 byla nová hodnota v OCR0 menší než aktuální hodnota v TCNT0 a tím by v aktuální periodě PWM nedošlo vůbec ke komparaci (to by vedlo ke chvilkovému nastavení střídy 100% v dotčené jedné periodě PWM)!

1.9.3.2 Phase Correct PWM mód

- Hardwarové generování **symetrické PWM**.
  - Puls je ve středu periody PWM. Doporučené pro řízení el. pohonů.

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

Obrázek 18 Režim Phase Correct PWM u Timer/Counter 0



Obrázek 19 Princip generování PWM (režim Phase Correct PWM, časovač Timer/Counter 0)

- **Maximální frekvence PWM je cca poloviční oproti režimu Fast PWM.**
  - Časovač čítá nejprve nahoru (od 0 do 255 = TOP) a pak dolů zpět k 0.

$$f_{OC0} = \frac{F_{CPU}}{N \cdot 510} \tag{22}$$

Tabulka 15 Frekvence PWM v závislosti na taktovací frekvenci časovače (režim Phase Correct PWM, 8-bit časovač)

Předdělička (N)	Frekvence PWM při F_{CPU} = 1 MHz	Frekvence PWM při F_{CPU} = 14,7456 MHz
1	1960,78 Hz	28 913,94 Hz
8	245,10 Hz	3614,12 Hz
64	30,64 Hz	451,76 Hz
256	7,66 Hz	112,94 Hz
1024	1,91 Hz	28,26 Hz

- **Rozlišení ve střídě je opět 8 bit.**
- Při dosažení komparační úrovně se mění logická hodnota na pinu OC0.
- **Chování PWM signálu v krajních hodnotách střídy (0 a 100 %):**
  - Lze nastavit krajní hodnoty střídy 100 % (OCR0 = 255) i 0 % (OCR0 = 0).
- **Komparační hodnotu lze do registru OCR0 zapisovat kdykoliv.** Protože komparační registry jsou dvojitě bufferované. Programátor zapisuje do bufferu, hardware si hodnotu z bufferu zapíše do registru sám ve vhodném okamžiku.
- **Polarita signálu na pinu OC0**
  - Neinvertovaný signál PWM vs. invertovaný signál PWM – viz Obrázek 20. Volba proběhne nastavením bitů **COM00** a **COM01** v registru **TCCR0**.

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

Note: 1. A special case occurs when OCR0 equals TOP and COM01 is set. In this case, the compare match is ignored, but the set or clear is done at TOP. See "Phase Correct PWM Mode" on page 74 for more details.

Obrázek 20 Nastavení chování výstupního pinu OC0 (režim Phase Correct PWM)

1.9.4 Časovač Timer/Counter 1

- Timer/Counter 1 je 16-bit časovač
- **Dva kanály PWM** (kanál A, kanál B)
  - Dvě komparační hodnoty, každá pro jeden kanál (registry OCR1A a OCR1B).
  - Dva výstupní piny (OC10 a OC11), každý pro jeden kanál.
  - Společný čítací registr (TCNT1).
- **3 základní režimy PWM:**
  - **Fast PWM**
  - **Phase Correct PWM**
  - **Phase and Frequency Correct PWM**
- **Možnost nastavení TOP hodnoty** – lze jemněji nastavit frekvenci PWM a rozlišení ve střídě.

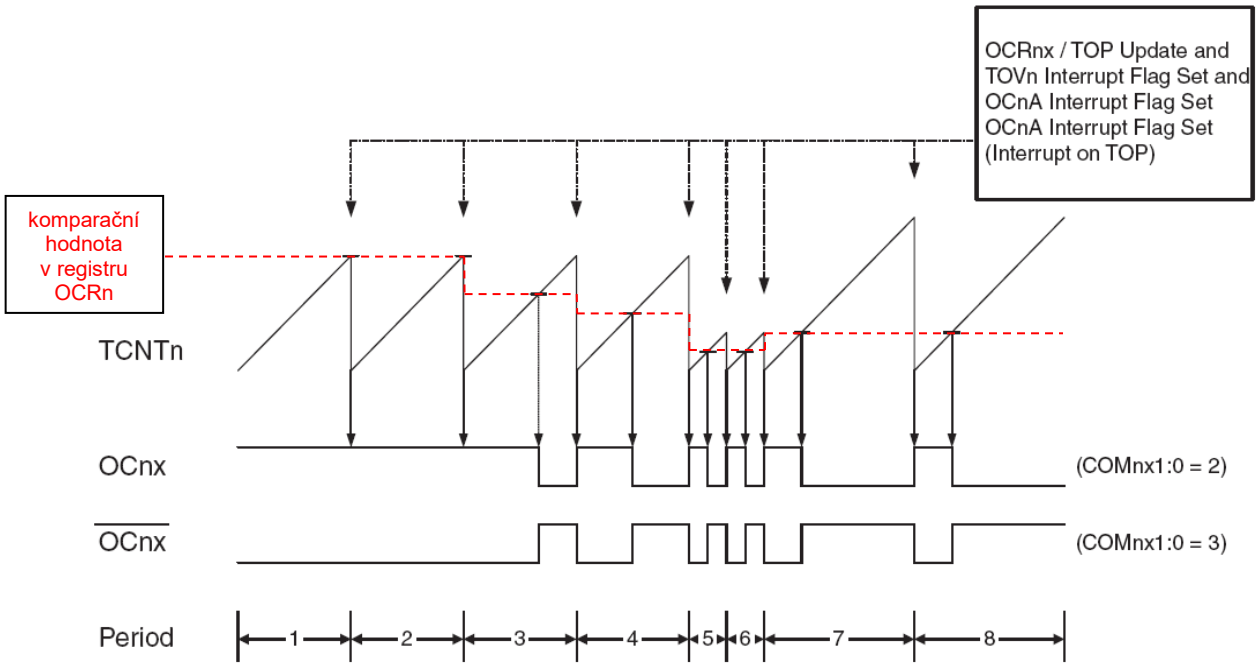
Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Obrázek 21 Režimy PWM u Timer/Counter 1

- Možnosti nastavení rozlišení PWM:
  - Pevná TOP hodnota – lze volit z 8, 9, 10-bit rozlišení PWM
  - Volně nastavitelná TOP hodnota – v rozsahu 3 až 65535 (odpovídá rozlišení 2-bit až 16-bit).
    - TOP hodnota se zapisuje se buď do registru ICR1 nebo OCR1A.

1.9.4.1.1.1 Fast PWM mód

- Princip stejný jako u Timer/Counter 0 v tomto módu, ale ……
- **Lze hýbat s TOP hodnotou čímž lze jemněji nastavovat frekvenci PWM**. (To je rozdíl oproti Timer/Counter 0, kde je TOP napevno 8 bit – čítá od 0 do 255).
    - TOP hodnota také udává rozlišení ve střídě.
    - Rozlišení zde může nabývat pevných hodnot 8, 9 nebo 10 bitů.
    - Nebo může být určeno hodnotou v registru ICR1 nebo OCR1A (dle zvoleného režimu PWM). Maximální rozlišení je 16 bit (ICR1 nebo OCR1A = 65535). Minimální je 2 bity (ICR1 nebo OCR1A = 3).
    - **Chceme-li uživatelsky nastavit rozlišení, je nejlepší použít registr ICR1 pro definování TOP hodnoty. Jinak použijeme jednu z přednastavených hodnot (8, 9 nebo 10-bit). Registry OCR1A a OCR1B pak slouží jako komparační.**



Obrázek 22 Princip generování signálu PWM (režim Fast PWM, časovač Timer/Counter 1)

- Frekvence PWM se vypočte jako:

$$f_{OC1xPWM} = \frac{F_{CPU}}{N \cdot (1 + TOP)} \tag{23}$$

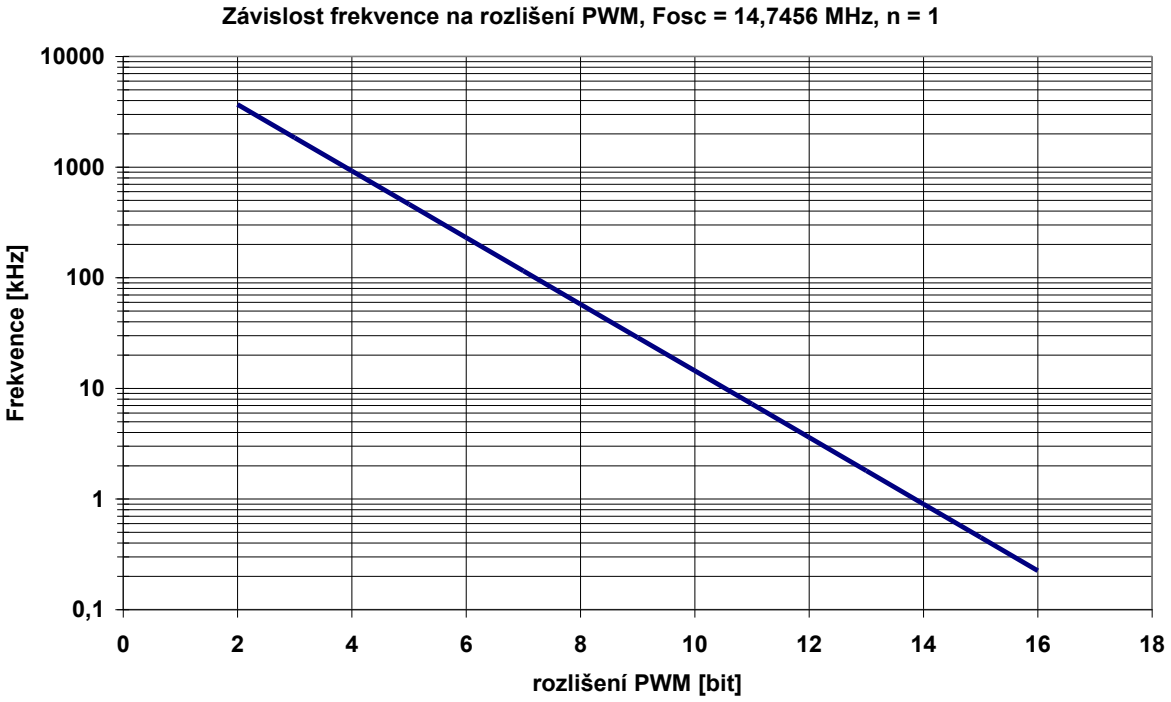
Kde: $f_{OC1xPWM}$ – frekvence PWM signálu na výstupním pinu OC1n
F_{CPU} – taktovací frekvence CPU
N – dělící poměr předděličky časovače
TOP – top hodnota časovače

- Střída PWM je určena komparační hodnotou v komparačním registru:

$$\boxed{\text{duty cycle} = \frac{\text{compare value}}{TOP}} \tag{24}$$

Kde: *duty cycle* – požadovaná střída [-] (v rozsahu 0 až 1)
compare value – hodnota zapsaná v komparačním registru (buď OCR1A nebo OCR1B nebo ICR1 dle zvoleného režimu PWM)
TOP – TOP hodnota časovače zapsaná v registru buď ICR1 nebo OCR1A (dle zvoleného režimu PWM)

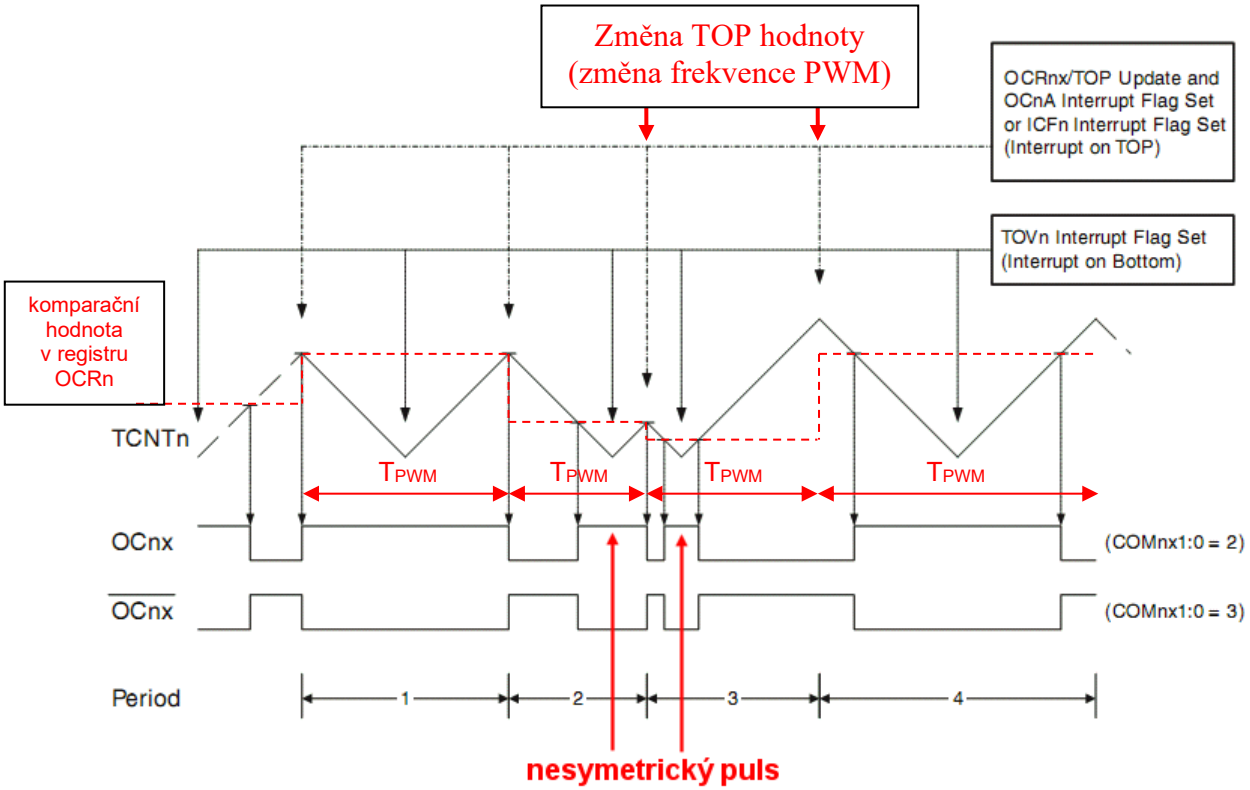
- **Všimněte si, že frekvence PWM souvisí s rozlišením PWM:**
  - Rozlišení ve střídě ↑ , frekvence PWM ↓ a naopak



Obrázek 23 Závislost mezi frekvencí a rozlišením PWM (16-bit časovač, frekvence časovače 14,7456 MHz)

1.9.4.2 Phase Correct PWM mód

- Princip stejný jako u Timer/Counter 0 v tomto módu, ale …….
- Rozlišení PWM lze opět volit
  - Vybrat z pevných hodnot 8, 9 nebo 10 bitů.
  - Nastavit dle libosti, TOP = OCR1A nebo TOP = ICR1 (dle zvoleného režimu PWM).
- Časovač čítá nahoru i dolů – max. frekvence PWM je poloviční oproti režimu Fast PWM.



Obrázek 24 Princip generování signálu PWM
(režim Phase Correct PWM, časovač Timer/Counter 1)

- **Změna frekvence PWM (změna TOP hodnoty) za běhu časovače může způsobit vadný puls v rámci periody PWM (puls s neodpovídající střídou) – viz** Obrázek 24.
  - Proto se doporučuje použít nově zavedený režim **Phase and Frequency Correct Mode PWM**, kde je toto ošetřeno.
  - V případě, že se TOP hodnota (a tedy frekvence PWM) za běhu nemění, jsou tyto dva módy rovnocenné.

- **Frekvence PWM se vypočte jako:**

$$f_{OC1xPCPWM} = \frac{F_{OSC}}{2 \cdot N \cdot TOP} \tag{25}$$

Kde: $f_{OC1xPWM}$ – frekvence PWM signálu na výstupním pinu OC1n
F_{OSC} – taktovací frekvence CPU
N - dělící poměr předděličky časovače
TOP – top hodnota časovače
2 – frekvence PWM je poloviční vůči režimu Fast PWM, protože časovač čítá nahoru i dolů

- **Rozlišení PWM závisí na TOP hodnotě následovně:**

$$R_{PWM}[bits] = \frac{\log(\text{TOP}+1)}{\log(2)} \tag{26}$$

Kde: R_{PWM} – rozlišení PWM ve střídě [bitů]
TOP – top hodnota časovače

Tabulka 16 Rozlišení PWM

TOP hodnota	Rozlišení PWM
65535	16-bit
32767	15-bit
16383	14-bit
8191	13-bit
4095	12-bit
2047	11-bit
1023	10-bit
511	9-bit
255	8-bit
127	7-bit

- **Střída PWM je určena komparační hodnotou v komparačním registru:**

$$\text{duty cycle} = \frac{\text{compare value}}{TOP} \tag{27}$$

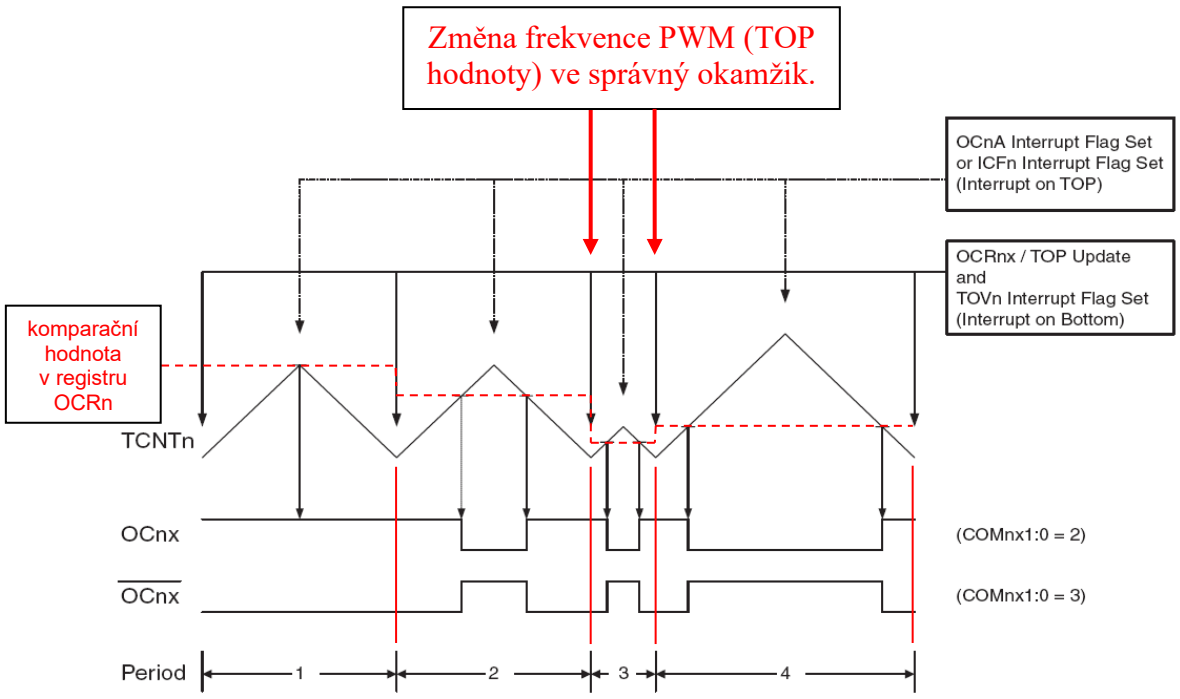
Kde: *duty cycle* – požadovaná střída [-] (v rozsahu 0 až 1)
compare value – hodnota zapsaná v komparačním registru (buď OCR1A nebo OCR1B nebo ICR1 dle zvoleného režimu PWM)
TOP – TOP hodnota časovače zapsaná v registru buď ICR1 nebo OCR1A (dle zvoleného režimu PWM)

1.9.4.3 Phase and Frequency Correct PWM mód

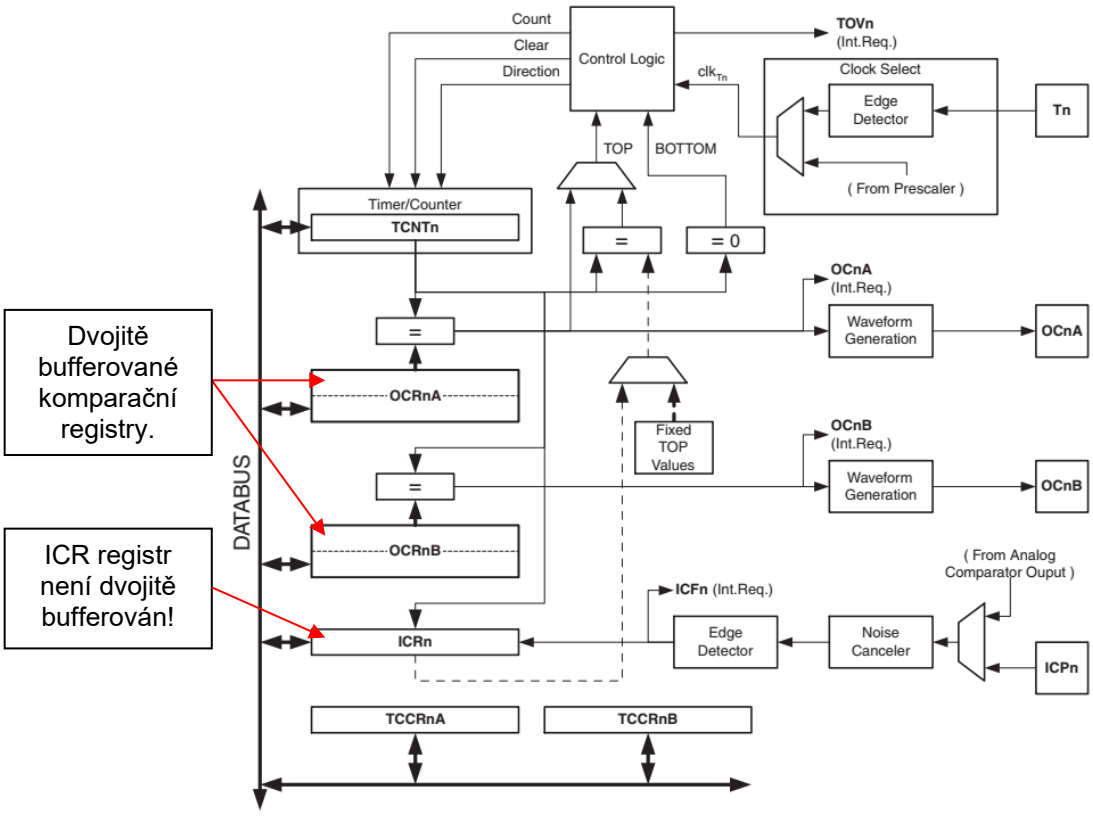
- Funkčně je tento režim stejný jako režim **Phase Correct PWM**, ale umožňuje jednoduše měnit frekvenci PWM za chodu.
- **Tento režim použijeme, když potřebujeme za běhu měnit frekvenci PWM**.
  - Časovač nastavíme do MODE = 9.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

  - **Frekvence** se nastavuje komparačním registrem OCR1A (TOP).
  - **Střída** se nastavuje komparačním registrem OCR1B.
  - Při změně frekvence je současně nutné přepočítat komparační hodnotu, aby byla dodržena požadovaná střída!
  - Do registrů OCR1A i OCR1B můžeme zapsat kdykoliv (nesynchronizovaně s tikáním časovače), protože registry OCR1x jsou dvojitě bufferované (Obrázek 26). Hardware si hodnotu v registru aktualizuje sám až ve vhodný okamžik (po dosažení BOTTOM hodnoty - Obrázek 25).
  - **PWM signál lze odebírat pouze z výstupního pinu OC1B**, výstupní pin OC1A nelze použít!
  - V případě, že bychom využili pro nastavování měnící se frekvence PWM registr ICR1 (časovač v režimu MODE = 8), který není dvojitě bufferován, tak bychom ztratily výhodu, kterou nám hardware nabízí. Do registru ICR1 bychom museli zapisovat až po dosažení BOTTOM hodnoty, aby nedošlo k chybnému. To by znamenalo reagovat na přerušení Timer Overflow – ztráta principu „nastav a zapomeň“.
  - Pokud není potřeba frekvenci PWM za chodu měnit, je režim **Phase and Frequency Correct PWM** shodný s režimem **Phase Correct PWM**.



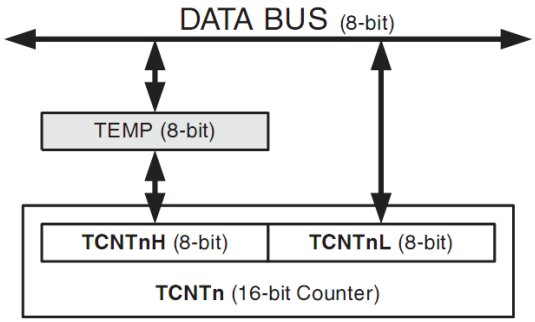
Obrázek 25 Princip generování signálu PWM
(režim Phase and Frequency Correct PWM, časovač Timer/Counter 1)



Obrázek 26 Vnitřní bloková struktura 16-bit časovače Timer/Counter 1

Přístup k 16-ti bitovým registrům časovače Timer/Counter 1

- Registry TCNT1, OCR1A, OCR1B, ICR1 jsou 16-bitové.
  - Přístup k nim je po bajtech = přístup k 16-bit registru na dvakrát.
- Je nutné zabezpečit, že se během čtení nebo zápisu nezmění druhý bajt, pak by již obě poloviny k sobě nepatřily.
  - Může se např. stát, že vyčtu horní bajt a něž přečtu dolní bajt, může mezi tím dojít k jeho změně. Vyčtený horní a dolní bajt pak k sobě nebudou patřit.
- Řešením je bufferovat jednu polovinu 16-ti bitového registru.
- Každý 16-ti bitový časovač má speciální 8-mi bitový registr (buffer), do kterého se dočasně ukládá **horní bajt** 16-ti bitového čísla.



Obrázek 27 Bufferování 16-bit registrů časovače Timer/Counter 1

- Programátor tento buffer nevidí (není pro něj přístupný). Aby bufferování fungovalo je nutné dodržet následující postup zápisu a čtení 16-bit hodnoty:
  - **Zápis do 16-bit registru časovače**
    - **Nejprve zapíšeme horní bajt a pak teprve dolní bajt.**
    - Při zápisu se nejprve horní bajt uloží do bufferu a pak se společně se zápisem dolního bajtu zapíše i obsah buffferu do vyššího bajtu 16-bit registru.
  - **Čtení z 16-bit registru časovače**
    - **Nejprve čteme dolní bajt a pak teprve horní bajt.**
    - Při čtení dolního bajtu se zároveň horní bajt uloží do bufferu a následné čtení horního bajtu vyčte hodnotu z bufferu. Tím je zajištěno, že k sobě obě poloviny patří.
- **Ne všechny 16-ti bitové registry jsou vybaveny bufferem.**
  - Seznam bufferovaných registrů je uveden v datasheetu AVR.
- **Při programování v assembleru je nutno dbát na uvedené pravidlo.**
- **Při programování v C** přistupujeme rovnou na celý 16-ti bitový registr (je-li to možné), správný přístup zajistí kompilátor.

```
unsigned short int capturedValue;
unsigned short int compareValue;

capturedValue = ICR1;
OCR1A = compareValue;
```

1.9.5 Příklad

Generujte PWM o pevné frekvenci 10 kHz. Rozlišení PWM min. 8-bit. Střída musí jít nastavit od 0 % do 100 %. Taktovací frekvence CPU je F_{CPU} = 14,7456 MHz.

- Ze zadání vyplývá, že je třeba nastavovat střídu od 0 % včetně → musíme vyloučit režim **Fast PWM**, který neumí nastavit střídu 0 %.
- Ze zadání vyplývá požadavek na pevnou frekvenci PWM 10 kHz:
  - Volíme **Timer/Counter 1**, u kterého můžeme frekvenci PWM nastavit přesně dle zadání.
  - Volíme režim **Phase and Frequency Correct PWM** (lze také **Phase Correct PWM**, zde je to jedno, frekvence PWM je pevná, můžeme použít oba režimy).
- Stanovíme vhodnou děličku časovače N a TOP hodnotu pro dosažení požadované frekvence a rozlišení PWM:

$$f_{PWM} = \frac{F_{CPU}}{2 \cdot N \cdot TOP} \tag{28}$$

Kde: f_{PWM} – frekvence PWM signálu
F_{CPU} – taktovací frekvence CPU
N - dělící poměr předděličky časovače
TOP – top hodnota časovače

  - Najděme vhodnou předděličku N a TOP hodnotu:

$$TOP = \frac{F_{CPU}}{2 \cdot N \cdot f_{PWM}} \tag{29}$$

Předdělička (N)	**TOP hodnota** **F_{CPU} = 14,7456 MHz** **f_{PWM} = 10 kHz**
1	737,28
8	92,16
64	11,52
256	2,88
1024	0,72

  - Jediná použitelná předdělička časovače je **N = 1**
    - Protože **TOP hodnota vychází 737** (po odstranění desetinné části), což je více než požadovaných 255 (min. 8-bit rozlišení).
    - Rozlišení PWM s TOP = 737 činí:

$$R_{PWM} = \frac{\log(\text{TOP}+1)}{\log(2)} = \frac{2,868}{0,301} = 9,53\,bit \tag{30}$$

Což je více než požadovaných 8-bit.

- U ostatních předděličeк bychom již nedosáhli požadovaného rozlišení PWM.

- Po zvolení N=1 bude mít frekvence PWM hodnotu přesně:

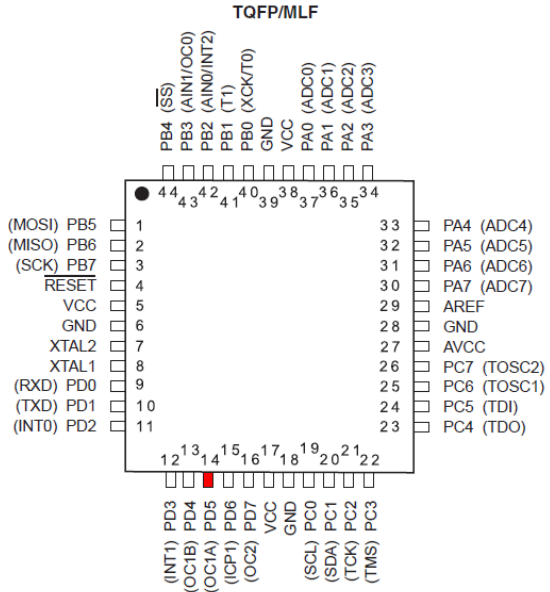
$$f_{PWM} = \frac{F_{CPU}}{2 \cdot N \cdot TOP} = \frac{14745600}{2 \cdot 1 \cdot 737} = 10,0038\,kHz \tag{31}$$

- Odpovídající pseudokód:

```
Nastav pin pro generování PWM jako výstup
Vynuluj časovač
Nastav časovač do režimu PWM
Nastav TOP hodnotu
Nastav počáteční komparační hodnotu dle požadované střídy
Spusť časovač

WHILE (nekonečná smyčka)
       Nastav komparační hodnotu dle požadované střídy
END WHILE
```

- Nastavení pinu pro generování PWM
  - Použijeme pin PD5 (OC1A) – kanál A časovače Timer/Counter 1.



- Kód:

```
#include <avr/io.h>

int main(void)
{
       // nastaveni smeru na pinu OC1A
       DDRD |= (1<<PD5);    // vystup OC1A (signal PWM)
```

- **Připojení pinu OC1A (PD5) k časovači**
  - Je třeba nastavit bity **COM1A1** a **COM1A0** do správných hodnot dle Tabulka 17.

Tabulka 17 Nastavení chování pinu OC1A v režimu Phase and Frequency Correct PWM

Table 45. Compare Output Mode, Fast PWM(1)

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM13:0 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on compare match, set OC1A/OC1B at TOP
1	1	Set OC1A/OC1B on compare match, clear OC1A/OC1B at TOP

Note: 1. A special case occurs when OCR1A/OCR1B equals TOP and COM1A1/COM1B1 is set. In this case the compare match is ignored, but the set or clear is done at TOP. See "Fast PWM Mode" on page 97. for more details.

Table 46 shows the COM1x1:0 bit functionality when the WGM13:0 bits are set to the phase correct or the phase and frequency correct, PWM mode.

  - Bity **COM1A1** a **COM1A0** se nachází v registru **TCCR1A**.

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

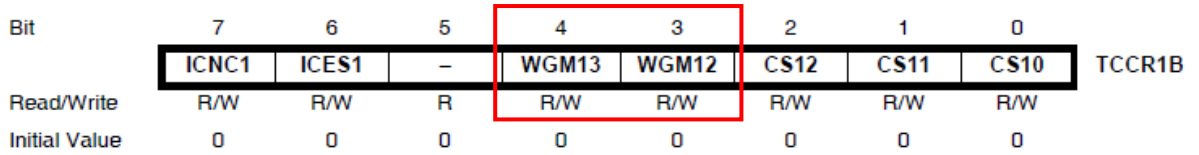
  - Kód:

```
// povoleni ovladani pinu OC1A casovacem (neinvertovana PWM)
TCCR1A |= (1<<COM1A1);     // bit COM1A1 = 1
TCCR1A &= ~(1<<COM1A0);    // bit COM1A0 = 0
```

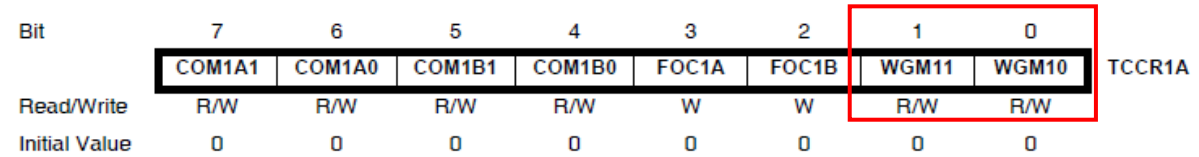
- **Nastavení časovače Timer/Counter 1 do režimu Phase and Frequency Correct PWM.**
  - Použijeme **Mode 8** – komparační hodnota bude uložena v registru **OCR1A**.
    - Bity **WGM10 až WGM13** je třeba nastavit podle níže uvedené tabulky (tabulka z datasheetu ATmega32).
    - Bit **WGM10 a WGM11** se nachází v řídícím registru **TCCR1A**.
    - Bity **WGM12** a **WGM13** se nachází v řídícím registru **TCCR1B**.

Tabulka 18 Režimy časovače Timer/Counter 1

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP



Obrázek 28 Řídící registr TCCR1B časovače Timer/Counter 1



Obrázek 29 Řídící registr TCCR1A časovače Timer/Counter 1

- Kód pro nastavení Timer/Counter 1 do režimu CTC:

```
// nastaveni casovace do rezimu Phase and Frequency correct PWM
TCCR1A &= ~(1<<WGM10);      // bit WGM10 = 0
TCCR1A &= ~(1<<WGM11);      // bit WGM11 = 0
TCCR1B &= ~(1<<WGM12);      // bit WGM12 = 0
TCCR1B |= (1<<WGM13);       // bit WGM13 = 1
```

- **Nastavení TOP hodnoty**
  - Zápisem do registru ICR1 – registr ICR1 určuje TOP hodnotu.
  - Kód:
```
ICR1 = 737;   // nastaveni TOP hodnoty
```

- **Nastavení počáteční komparační hodnoty**
  - Uvažujme počáteční střídu 0 %.
  - Kód:
```
OCR1A = 0;    // pocatecni strida 0 %
```

- **Vynulování čítacího registru časovače**
  - Kód:
```
TCNT1 = 0;    // vynulovani casovace
```

- **Spuštění časovače na taktovací frekvenci F_{CPU}**
  - Předdělička časovače N = 1 → časovač běží na frekvenci CPU
  - Bity **CS12, CS11 a CS10** v registru **TCCR1B** – volba zdroje hodin a předděličky hodin časovače.
  - Řídící registr TCCR1B

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Tabulka 19 Výběr zdroje hodin pro časovač Timer/Counter 1

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

  - Nastavením zdroje hodin se časovač rovnou spustí.
  - Kód:

```
// spusteni casovace na frekvenci Fcpu/1
TCCR1B &= ~((1<<CS12)|(1<<CS11));       // bity CS11=0, CS12=0
TCCR1B |= (1<<CS10);                    // bit CS10=1
```

- **Úplný program (pro překladač AVR-GCC):**

```
#include <avr/io.h>

int main(void)
{
	// nastaveni smeru na pinu OC1A
	DDRD |= (1<<PD5);    // vystup OC1A (signal PWM)

	// nastaveni casovace do rezimu Phase and Frequency correct PWM
	TCCR1A |= (1<<WGM10);      // bit WGM10 = 1
	TCCR1A &= ~(1<<WGM11);     // bit WGM11 = 0
	TCCR1B &= ~(1<<WGM12);     // bit WGM12 = 0
	TCCR1B |= (1<<WGM13);      // bit WGM13 = 1

	// povoleni ovladani pinu OC1A casovacem (neinvertovana PWM)
	TCCR1A |= (1<<COM1A1);     // bit COM1A1 = 1
	TCCR1A &= ~(1<<COM1A0);    // bit COM1A0 = 0

	ICR1 = 737;   // nastaveni TOP hodnoty
	OCR1A = 0;    // pocatecni strida 0 %
	TCNT1 = 0;    // vynulovani casovace

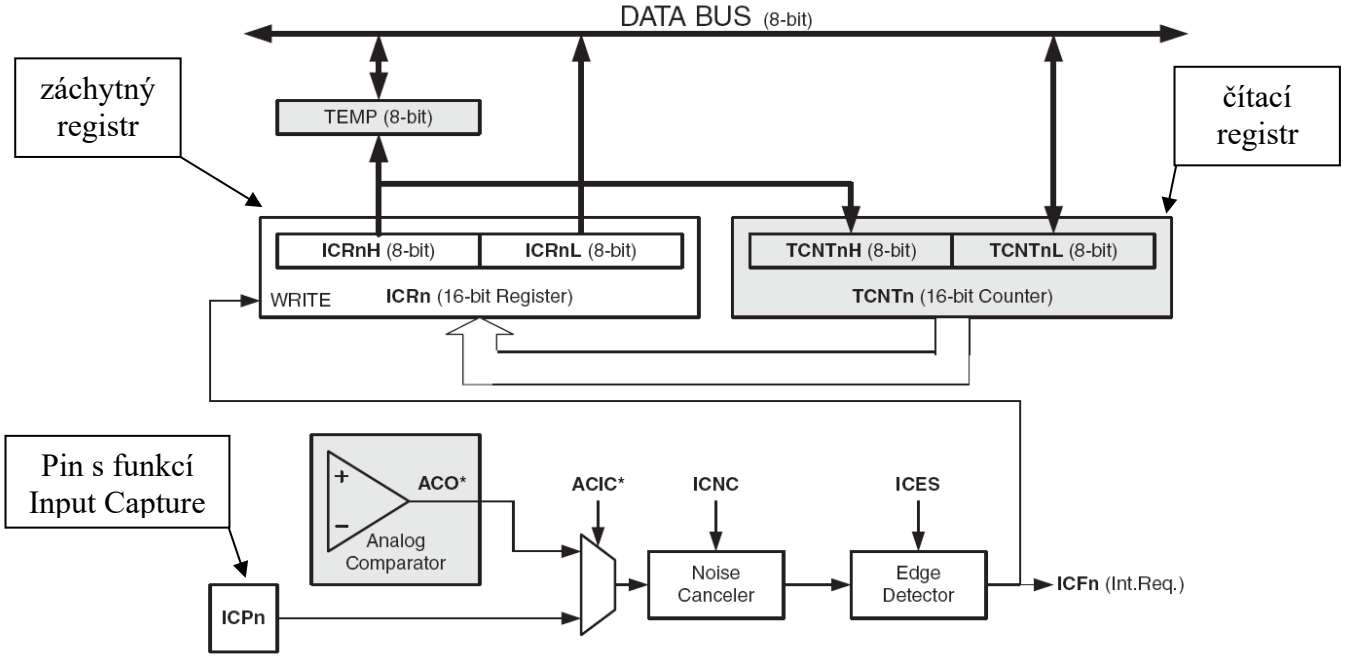
	// spusteni casovace na frekvenci Fcpu/1
	TCCR1B &= ~((1<<CS12)|(1<<CS11));        // bity CS11=0, CS12=0
	TCCR1B |= (1<<CS10);                     // bit CS10=1


	while(1)
	{
		// prubezne nastavovani stridy jak je potreba - zapisem hodnoty do 16-it
		// komparacniho registru OCR1A

		//OCR1A = 0;  // strida 0%
		OCR1A = 737/2;      // strida 50%
		//OCR1A = 737;      // strida 100%
	}
}
```

1.10 Záchytná jednotka (Input Capture Unit)

- Slouží pro hardwarové zachycení aktuální hodnoty časovače/čítače po výskytu definované události.
- Použití – přesné měření délku pulsu, frekvence, střídy vstupního signálu.
- Záchytnou jednotkou jsou na AVR vybaveny pouze 16-bit časovače.
- Dále předpokládejme 16-bit časovač Timer/Counter 1, který má na ATmega32 jako jediný tuto funkci.



Obrázek 30 16-bit časovač s funkcí Input Capture

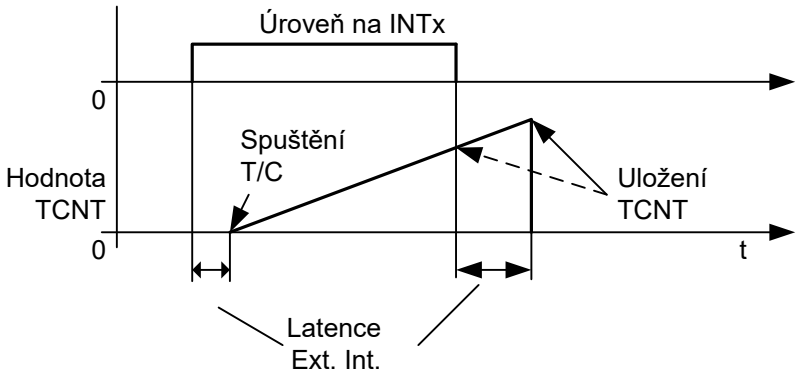
- **Po příchodu události dojde k automatickému zachycení 16-ti bitové hodnoty časovače TCNT1 do registru ICR1** (Input Capture Register) a zároveň se nastaví příznak **ICF1** (Input Capture Flag). Je-li povoleno přerušení, vykoná se.
- **Zdrojem vstupní události je:**
  - změna logické úrovně na pinu ICP1 (vzestupná nebo sestupná hrana) nebo
  - výstup z analogového komparátoru (ACO).
- Normálně je registr **ICR1** určen pouze pro čtení a při použití funkce Input Capture je v něm uložena zachycená hodnota časovače, ale v režimu generování PWM může sloužit i jako zdroj pro definování TOP hodnoty, takže do něj lze zapisovat.

1.11 Měření délky impulsu

- Je možný dvojí způsob:
  - Pomocí externího přerušení – méně přesný způsob
  - Pomocí záchytné jednotky časovače – přesnější způsob

1.11.1 Pomocí externího přerušení

- Na AVR vstupy s alternativní funkcí **INTx** (kde x je číslo externího přerušení).
- Chceme měřit délku kladného pulsu.

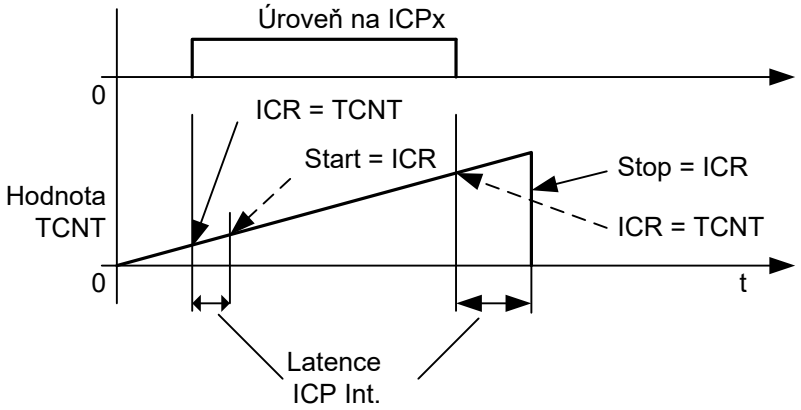


Obrázek 31 Měření délky pulsu pomocí externího přerušení

- **Algoritmus programu**
  - **Hlavní program**
    - Konfigurace externího přerušení na vzestupnou hranu, povolení přerušení.
    - Stavový automat v ISR externího přerušení do stavu „začátek“.
  - **Rutina pro obsluhu ext. přerušení**
    - Stav „začátek“ – přijde vzestupná hrana pulsu
      - Spuštění časovače.
      - Nastavení externího přerušení na sestupnou hranu.
      - Stavový automat v ISR externího přerušení do stavu „konec“.
    - Stav „konec“ – přijde sestupná hrana pulsu
      - Vyčtení hodnoty z čítacího registru časovače (TCNT) a přepočet na skutečný čas – vynásobením periodou „tikání“ časovače.
      - Zastavení časovače.
      - Příprava na měření délky dalšího pulsu
        - Nastavení externího přerušení na vzestupnou hranu.
        - Stavový automat v ISR externího přerušení do stavu „začátek“.
- Problém v latenci vstupu do ISR.
- Lepší řešení – použití záchytné jednotky **Input Capture**.

1.11.2 Pomocí záchytné jednotky Input Capture

- Přesnější měření než pomocí externího přerušení, latence obsluhy přerušení od zachycení nehraje roli.
- Na AVR digitální vstupy s alternativní funkcí **ICPn** (kde n je číslo časovače).
- Chceme měřit délku kladného pulsu.



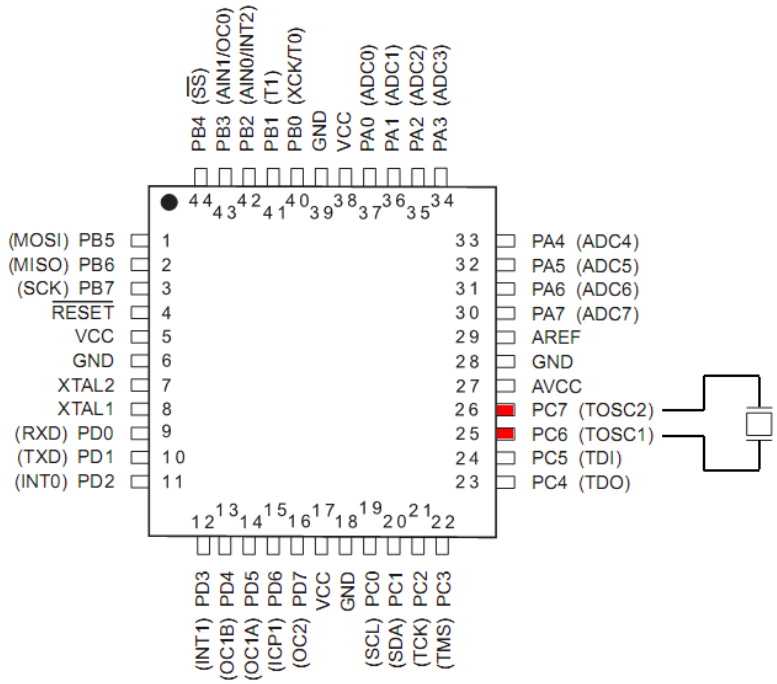
Obrázek 32 Měření délky pulsu pomocí záchytné jednotky

- **Algoritmus programu**:
  - **Hlavní program** (měření délky kladného pulsu):
    - Konfigurace ICP na vzestupnou hranu, spuštění časovače, povolení přerušení od záchytné jednotky (ICP).
    - Stavový automat v ISR pro ICP nastavit na „začátek“.
  - **Rutina pro obsluhu přerušení od ICP**
    - Stav „začátek“ – přijde vzestupná hrana pulsu
      - Proměnná start = ICR
      - Nastavit ICP na sestupnou hranu.
      - Stavový automat ISR pro ICP nastavit na „konec“.
    - Stav „konec“ – přijde sestupná hrana pulsu
      - Proměnná `stop = ICR`
      - Výpočet délky pulsu jako: `(stop - start)*`T_{tick}
      -
      - Kde: T_{tick} – perioda tiků časovače
      -
      - Příprava na měření délky dalšího pulsu
        - Nastavit ICP na vzestupnou hranu.
        - Stavový automat ISR pro ICP nastavit na „začátek“.
      - Časovač se nechá volně běžet, nezastavuje se.

- Pokud má MCU více jednotek – lze provádět i měření fázového posuvu.

1.12 Asnychronní časovač Timer/Counter 2

- 8-mi bitový
- Má stejné vlastnosti jako Timer/Counter 0.
- Navíc může být taktován asynchronně z externího zdroje (krystal 32,768 kHz) připojeného na vývody TOSC1 a TOSC2 mikrokontroléru.
    - **Využití pro generování RTC (hodin reálného času).**
    - **Časovač odměřuje sekundy.**
- Podrobnosti viz datasheet.



Obrázek 33 Připojení externího krystalu 32,768 kHz k Timer/Counter 2 (ATmega32)

1.13 Příloha 1 – vzorce pro ne-PWM režimy

- **Obecně platné vztahy**

$$\text{taktovací frekvence časovače} = \frac{\text{frekvence CPU}}{N} \tag{32}$$

$$\text{časové rozlišení časovače [s]} = \frac{N}{\text{frekvence CPU [Hz]}} \tag{33}$$

$$\text{počet tiků časovače} = \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{N} \tag{34}$$

$$N \geq \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{\text{počet tiků časovače}} \tag{35}$$

  - Kde: N – předdělička časovače (1, 8, 64, …)
    F_{CPU} – taktovací frekvence CPU

- **Režim CTC (Clear Timer on Compare)**
  - **Výpočet komparační hodnoty v režimu CTC**

$$\text{komparační hodnota} = \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{N} - 1 \tag{36}$$

    - Kde: N – předdělička časovače (1, 8, 64, …)
      F_{CPU} – taktovací frekvence CPU

- **Režim Normal**
  - **Perioda přetečení časovače**

$$\text{perioda přetečení časovače} = \frac{2^{bits} \cdot N}{\text{taktovací frekvence CPU}} \tag{37}$$

    - Kde: N – předdělička časovače (1, 8, 64, …)
      F_{CPU} – taktovací frekvence CPU

  - **Výpočet SW předvolby časovače v režimu Normal**

$$\text{předvolba} = 2^{Bits} - \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{N} \tag{38}$$

    - Kde: N – předdělička časovače (1, 8, 64, …)
      F_{CPU} – taktovací frekvence CPU
      Bits – bitová šířka časovače (8, 16, 32)

1.14 Příloha 2 – vzorce pro PWM režimy

- **Fast PWM**
  - **Frekvence PWM se vypočte jako:**

$$f_{PWM} = \frac{F_{CPU}}{N \cdot (1 + TOP)} \tag{39}$$

Kde: f_{PWM} – frekvence PWM signálu
F_{CPU} – taktovací frekvence CPU
N - dělící poměr předděličky časovače
TOP – top hodnota časovače

  - **Střída PWM je určena komparační hodnotou v komparačním registru:**

$$\text{duty cycle} = \frac{\text{compare value}}{TOP} \tag{40}$$

Kde: *duty cycle* – požadovaná střída [-] (v rozsahu 0 až 1)
compare value – hodnota zapsaná v komparačním registru
TOP – TOP hodnota časovače

- **Phase Correct PWM a Phase and Frequency Correct PWM**
  - **Frekvence PWM se vypočte jako:**

$$f_{PWM} = \frac{F_{CPU}}{2 \cdot N \cdot TOP} \tag{41}$$

Kde: f_{PWM} – frekvence PWM signálu
F_{CPU} – taktovací frekvence CPU
N - dělící poměr předděličky časovače
TOP – top hodnota časovače
2 – frekvence PWM je poloviční vůči režimu Fast PWM, protože časovač čítá nahoru i dolů

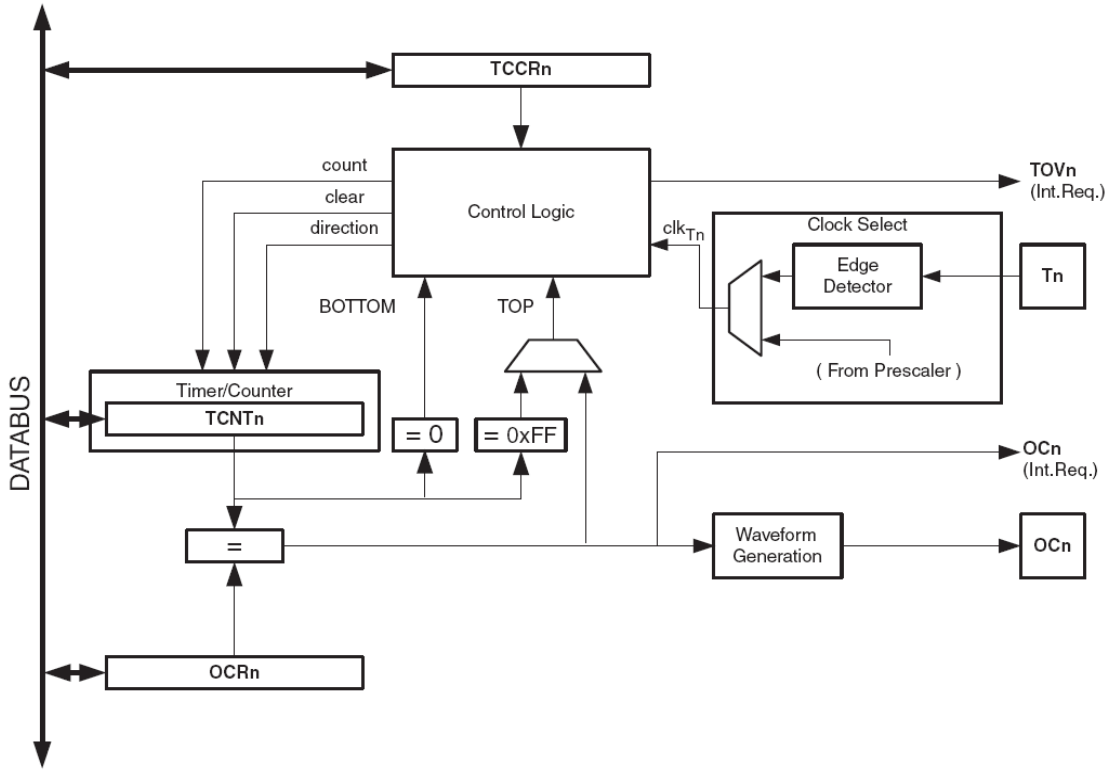
  - **Střída PWM je určena komparační hodnotou v komparačním registru:**

$$\text{duty cycle} = \frac{\text{compare value}}{TOP} \tag{42}$$

Kde: *duty cycle* – požadovaná střída [-] (v rozsahu 0 až 1)
compare value – hodnota zapsaná v komparačním registru
TOP – TOP hodnota časovače

1.15 Příloha 3 – přehled registrů časovačů na ATmega32

1.15.1 Registry časovače Timer/Counter 0



Obrázek 34 Struktura časovače Timer/Counter 0

- **Řídící registr TCCR0**

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

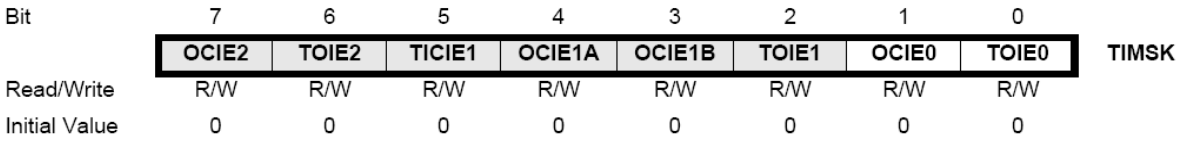
  - **FOC0** – aktivní pouze při normálním módu časovače, zápisem jedničky do FOC0 se vynutí signál shody, který vstupuje do jednotky generování průběhu, čímž dojde ke změně hodnoty na výstupním pinu OC0 dle nastavení daného bity COM0x. Nenastavuje žádné příznaky.
  - **WGM00** a **WGM01** – výběr módů časovače
  - **COM00** a **COM01** – výběr chování pinu OC0, závisí na zvoleném módu časovače
  - **CS00**, **CS01** a **CS02** – výběr zdroje taktovacího kmitočtu pro časovač

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Důležitá poznámka !!

Časovač se zapíná nastavením zdroje taktovacího kmitočtu, není zde přítomen žádný samostatný bit, který by zapnul nebo vypnul časovač jako tomu bylo u 8051.

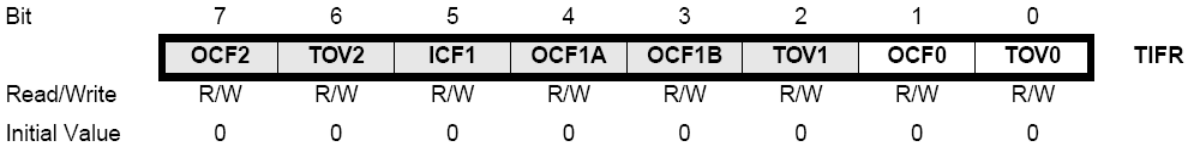
- **Povolení/zákaz přerušení – registr TIMSK**



Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **TOIE0** – povolení (1) / zákaz (0) přerušení při přetečení časovače 0
  - **OCIE0** – povolení (1) / zákaz (0) přerušení při komparaci

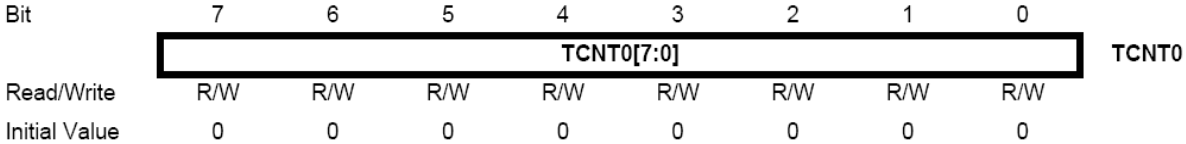
- **Registr s příznaky – registr TIFR**



Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **TOV0** – příznak přetečení časovače 0
  - **OCF0** – příznak dosažení shody při komparaci OCR0 a TCNT0
  - **Příznaky se automaticky nulují při vstupu do rutiny pro obsluhu přerušení.**
  - **Při ručním mazání příznaků je do nich nutno zapsat 1 !**

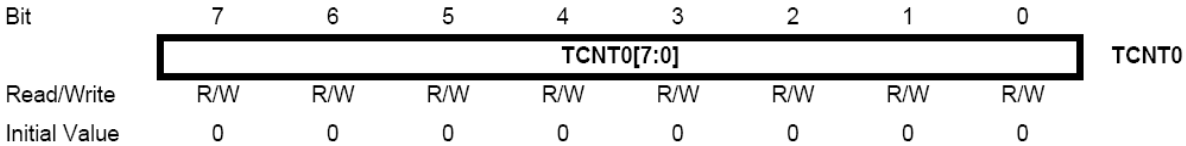
- **Registr TCNT0**



Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Uchovává aktuálně načítanou hodnotu. Lze ho číst i do něj zapisovat.

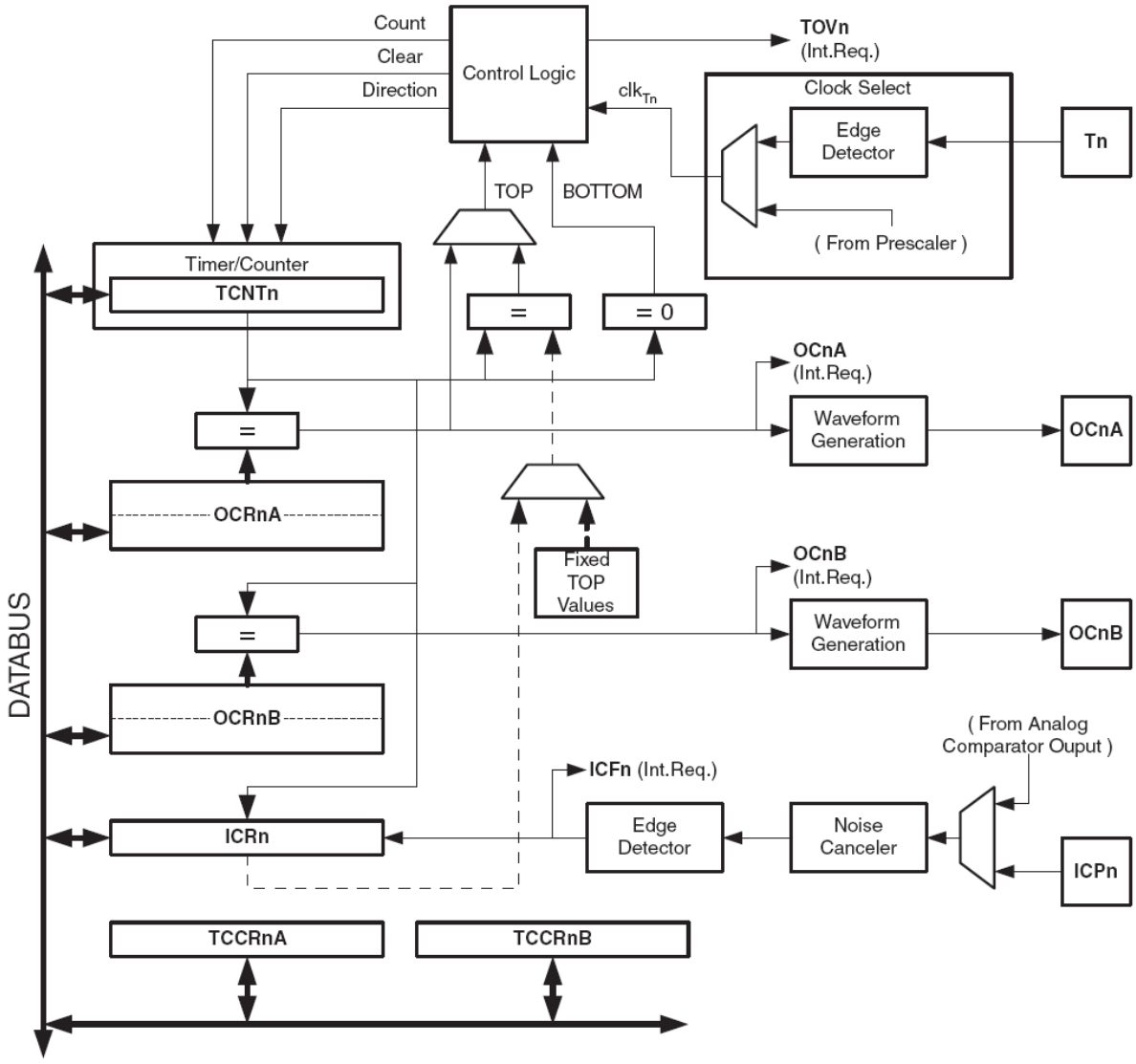
- **Registr OCR0**



Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Obsahuje komparační úroveň, která je neustále porovnávána s TCNT0. Při shodě lze vygenerovat přerušení nebo lze využít jednotku pro generování signálu na pinu OC0.
  - Lze ho normálně číst i do něj zapisovat.

1.15.2 Registry časovače Timer/Counter 1



Obrázek 35 Struktura časovače Timer/Counter 1

- **Řídící registr TCCR1A**

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - A – bity patřící komparační jednotce A
  - B – bity patřící komparační jednotce B
  - **FOC1A** a **FOC1B** – vynucení signálu shody komparace (pouze v ne-PWM módu časovače)
  - **WGM11** a **WGM10** – výběr módů časovače spolu s WGM13 a WGM12
  - **COM1A(B)1** a **COM1A(B)0** – výběr chování pinu OC1A(B), závisí na zvoleném módu časovače

- **Řídící registr TCCR1B**

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **ICNC1** – aktivace/deaktivace omezovače šumu pro záchytnou jednotku
  - **ICES1** – určuje při jaké hraně vstupního signálu dojde k zachycení, vzestupná (ICES1 = 0) nebo sestupná (ICES0 = 1) hrana
  - **WGM13** a **WGM12** – výběr módů časovače spolu s WGM11 a WGM10
  - **CS12, CS11 a CS10** – volba zdroje signálu a předděličky

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

- **Režimy časovače 1**

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

- **Povolení/zákaz přerušení – registr TIMSK**

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **TICIE1** – povolení (1) / zákaz (0) přerušení při zachycení
  - **OCIE1A(B)** – povolení (1) / zákaz (0) přerušení při komparaci (jednotka A nebo B)
  - **TOIE1** – povolení (1) / zákaz (0) přerušení při přetečení časovače

- **Registr s příznaky – registr TIFR**

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **ICF1** – příznak, že došlo k zachycení na pinu ICP1 nebo je příznak nastaven, při dosažení TOP hodnoty, když je ICR1 použit v módech PWM
  - **TOV1** – příznak přetečení časovače 1
  - **OCF1A(B)** – příznak dosažení shody při komparaci OCR1A(B) a TCNT1

1.16 Příloha 4 – obsazenost časovačů na Arduino

Tabulka 20 Obsazenost časovačů na Arduino

	Obsazené časovače	Volné časovače
Arduino Leonardo (ATmega32u4)	Timer0 (8-bit) OVF ISR, millis() Timer3 (16-bit) COMPA ISR, tone()	Timer1 (16-bit) Timer4 (10-bit)
Arduino Nano, UNO (ATmega328P)	Timer0 (8-bit) OVF ISR, millis() Timer2 (8-bit) COMPA ISR, tone()	Timer1 (16-bit)
Arduino MEGA2560 (ATmega2560)	Timer0 (8-bit) OVF ISR, millis() Timer2 (8-bit) COMPA ISR, tone()	Timer1 (16-bit) Timer3 (16-bit) Timer4 (16-bit) Timer5 (16-bit)

Časovač **Timer0** zajišťuje měření času počítáním počtu přetečení časovače. Používají ho funkce `millis()` a `micros()`.

Další časovač (**Timer2** nebo **Timer3** dle typu Arduina) je používán funkcí `tone()` pro generování obdélníkového průběhu o zadané frekvenci a střídě 50 % na vybraném pinu a to i tehdy, když se funkce v kódu nepoužije.

Při použití funkce `analogWrite()` pro generování PWM může dojít k obsazení dalších časovačů (těch které jsou v tabulce označeny jako volné), v závilosti na číslu výstupního pinu, na kterém má být generována PWM a v závislosti na typu Arduina. Pokud se funkce `analogWrite()` v kódu nepoužije, tak neobsazuje časovač.

https://docs.arduino.cc/language-reference/en/functions/analog-io/analogWrite/
https://docs.arduino.cc/learn/microcontrollers/analog-output/

Tabulka 21 Obsazení časovačů funkcí analogWrite() na Arduino Leonardo

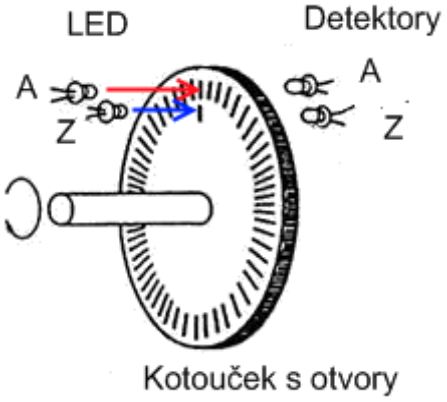
Výstupní PWM pin	Obsazené časovače	Volné časovače
žádný	Timer0 (8-bit) millis() Timer3 (16-bit) tone()	Timer1 (16-bit) Timer4 (10-bit)
3	Timer0 (8-bit) millis(), analogWrite() Timer3 (16-bit) tone()	Timer1 (16-bit) Timer4 (10-bit)
5	Timer0 (8-bit) millis() Timer3 (16-bit) tone(), analogWrite()	Timer1 (16-bit) Timer4 (10-bit)
6	Timer0 (8-bit) millis() Timer3 (16-bit) tone() Timer4 (10-bit) analogWrite()	Timer1 (16-bit)
9	Timer0 (8-bit) millis() Timer3 (16-bit) tone() Timer1 (16-bit) analogWrite()	Timer4 (10-bit)
10	Timer0 (8-bit) millis() Timer3 (16-bit) tone() Timer1 (16-bit) analogWrite()	Timer4 (10-bit)
11	Timer0 (8-bit) millis(), analogWrite() Timer3 (16-bit) tone()	Timer1 (16-bit) Timer4 (10-bit)
13	Timer0 (8-bit) millis() Timer3 (16-bit) tone() Timer4 (10-bit) analogWrite()	Timer1 (16-bit)

Pro další typy Arduin lze obsazenost časovače funkcí `analogWrite()` zjistit z tabulky `const uint8_t PROGMEM digital_pin_to_timer_PGM[]` udefinované v souboru `pins_Arduino.h`.

2 Měření otáček a polohy (úhlového natočení)

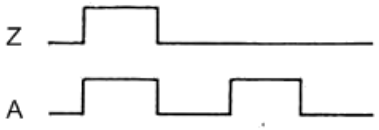
- Možnosti:
  - Analogově – DC motor zapojený jako dynamo, $U_i = k \cdot \omega$, zastaralý způsob, již se nepoužívá
  - Digitálně – pulzní snímání, běžný (moderní) způsob

2.1 Inkrementální čidlo



Obrázek 36 Princip konstrukce inkrementálního snímače otáček (jedna stopa + nulová značka)

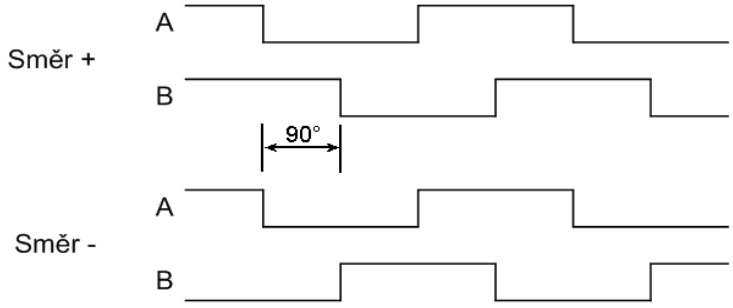
- Kanál A = měření rychlosti otáčení
- Kanál Z (volitelně, není nutné) = absolutní poloha (referenční značka)



- Nevýhoda: nelze rozeznat směr
- Vyhodnocení rychlosti otáčení (nyní pouze stručně)
  - Počet pulzů (hran) za jednotku času - velký počet pulzů / pro vysoké otáčky
  - Vzdálenost hran (perioda signálu) - malý počet pulzů / pro nízké otáčky

2.2 Inkrementální čidlo s kvadraturním výstupem

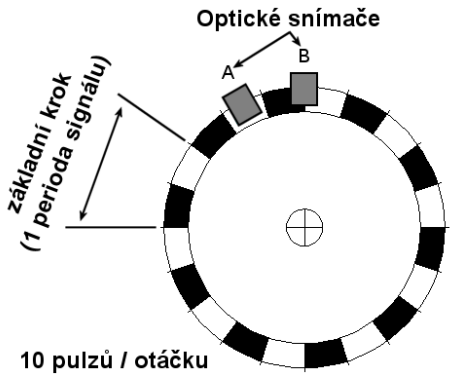
- Rotační kodér (rotary encoder)
- Dva kanály (A, B) vzájemně posunuté o $\Delta\varphi = 90°$ (= kvadraturní)
- Poskytuje informaci o rychlosti i směru otáčení, volitelně 3. kanál (referenční značka Z – jeden puls na mechanickou otáčku, udává zákl. polohu).
    - Směr+ : kanál A předbíhá kanál B
    - Směr- : kanál B předbíhá kanál A



- Vyhodnocení směru otáčení
    - Čtení úrovně druhého kanálu při výskytu hrany prvého kanálu (nebo naopak).
- **Dle provedení**
    - Mechanické snímání (spínače)
        - Spotřební elektronika (nast. hlasitosti)
        - Low-cost aplikace (omezená rychlost otáčení, odskoky kontaktů, mech. opotřebení)



    - Optické snímání
        - Při nárocích na vysokou přesnost a spolehlivost (průmyslové aplikace)
        - Dva optické snímače posunuté vzájemně o lichý násobek 90° elektrických – nejběžnější řešení.



- Rozlišení desítky až tisíce pulzů / otáčku.

2.2.1 Elektrické rozhraní (konkrétní příklad – snímač LARM IRC 300)

- Napájecí napětí
  - +5 V nebo 10 až 30 V
  - Spotřeba cca 1 až 2 W



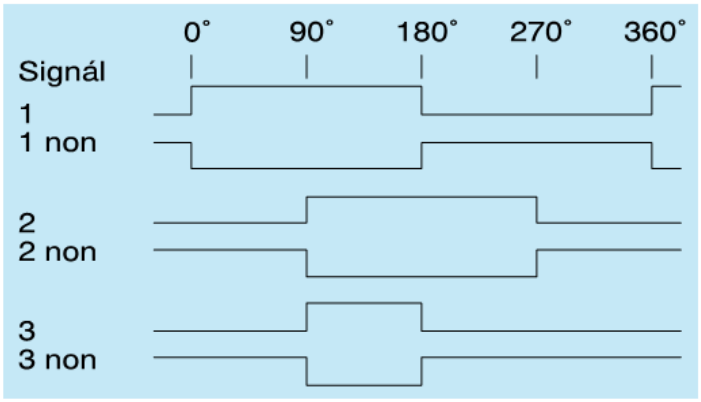
IRC 300 – 305

- Provedení el. výstupů
  - Push-pull (pro U_{nap} = 10 až 30 V)
  - Linkový budič (pro U_{nap} = 5 V)
  - Otevřený kolektor NPN nebo PNP
- Max. výstupní frekvence 200 kHz
  - Určuje max. rychlost otáčení:
    Např. 200 kHz / 2048 = 97,65 Hz → 5860 ot/min

Rozlišení 2048 pulzů /otáčku

- Výstupní signály:

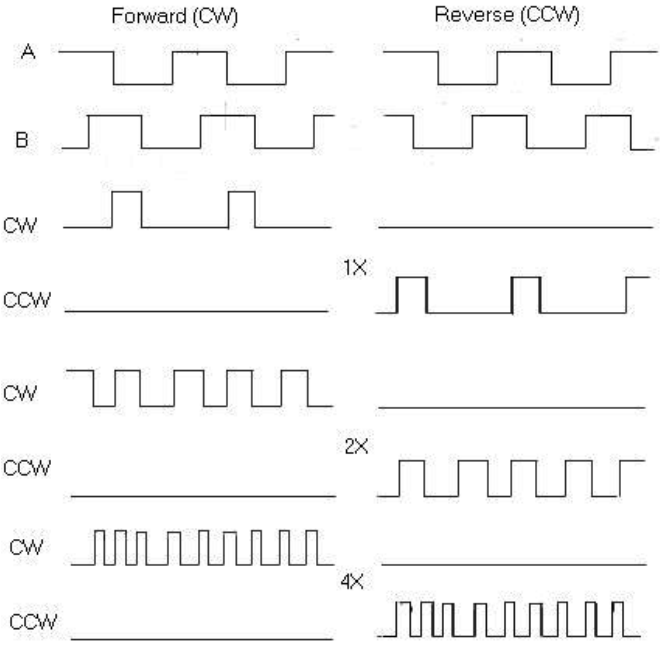
Výstupní signály IRC300 až 325
2 základní signály (1,2) posunuté o 90° elektrických, 1 nulový impulz (3) a jejich negace. Nad 100 kHz se nulový pulz nezaručuje.



  - Obvykle diferenciální signály (např. 1 a 1 non) pro zvýšení odolnosti proti rušení (2 stopy a nulová značka → celkem 6 vodičů). Logický stav je určen z rozdílu napětí mezi vodiči.

2.2.2 Zpracování signálu z kvalitního optického čidla v MCU

- **Možnosti čtení signálů z čidla**
  - Reakce na jednu z hran jednoho kanálu (rozlišení 1x)
  - Reakce na obě hrany jednoho kanálu (rozlišení x2)
  - Reakce na obě hrany obou kanálů (rozlišení x4)



Quadrature direction sensing and resolution enhancement. (CW = clockwise, CCW= counter-clockwise)

- **Princip zpracování signálů**
  1) zjištění směru otáčení – detekcí logické úrovně na druhém kanálu v okamžiku hrany na prvním kanálu
  2) výpočet relativní polohy – inkrementace / dekrementace čítače podle směru otáčení
  3) výpočet rychlosti otáčení – měření časové vzdálenosti hran (měření periody signálu) nebo počítáním počtu hran za definovanou dobu
- **Připojení čidla k MCU (různé možnosti)**
  - **Na externí přerušení – SW zpracování signálu z čidla**
    - Jedna stopa z čidla na ext. přerušení, druhá stopa na běžný digitální vstup MCU (rozlišení x1 nebo x2).

      nebo
    - Obě stopy z čidla na ext. přerušení MCU (rozlišení x4).
    - Výpočet rychlosti otáčení - měřením periody signálu z čidla nebo počítáním počtu pulsů za definovaný čas.

- **Použití Input Capture jednotky – SW zpracování signálu z čidla**
  - Jedna stopa z čidla na vstup Inpute Capture, druhá stopa na běžný digitální vstup MCU (rozlišení x1 nebo x2).
  - Výpočet rychlosti otáčení - měřením periody signálu z čidla.
- **MCU obsahuje speciální HW pro zpracování signálu z kvadraturního IRC čidla – hardwarové zpracování signálu z čidla**. Čidlo se připojí k vyhrazeným vstupům na MCU.
  - Hardware automaticky detekuje směr otáčení a dle něj inkrementuje nebo dekrementuje hardwarový čítač (čítač QPOSCNT v obrázku níže).

Figure 6. Quadrature Decoder State Machine

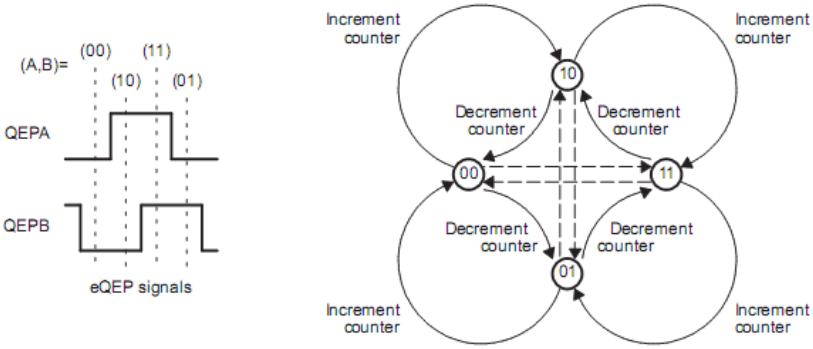
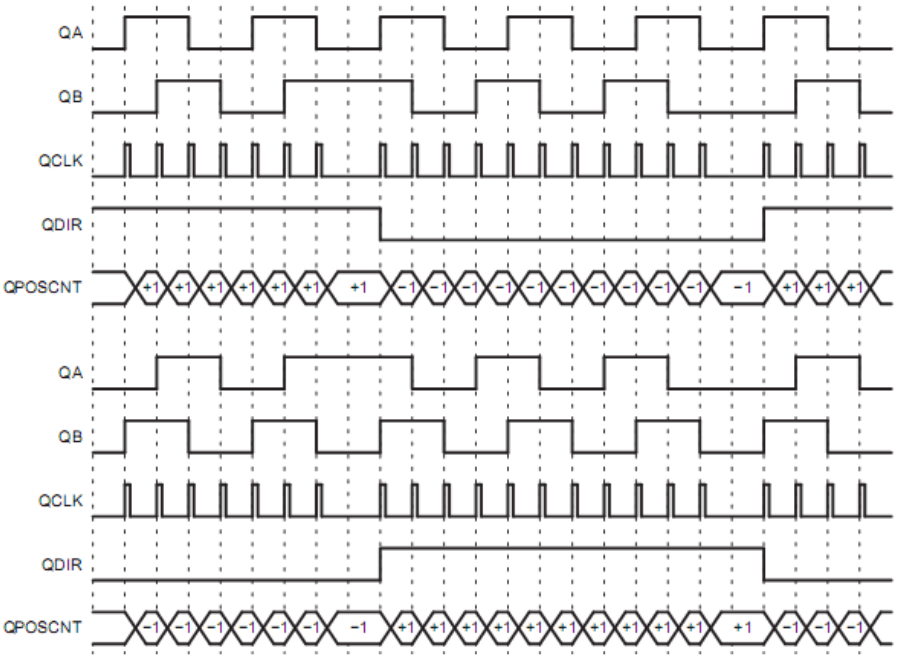


Figure 7. Quadrature-clock and Direction Decoding



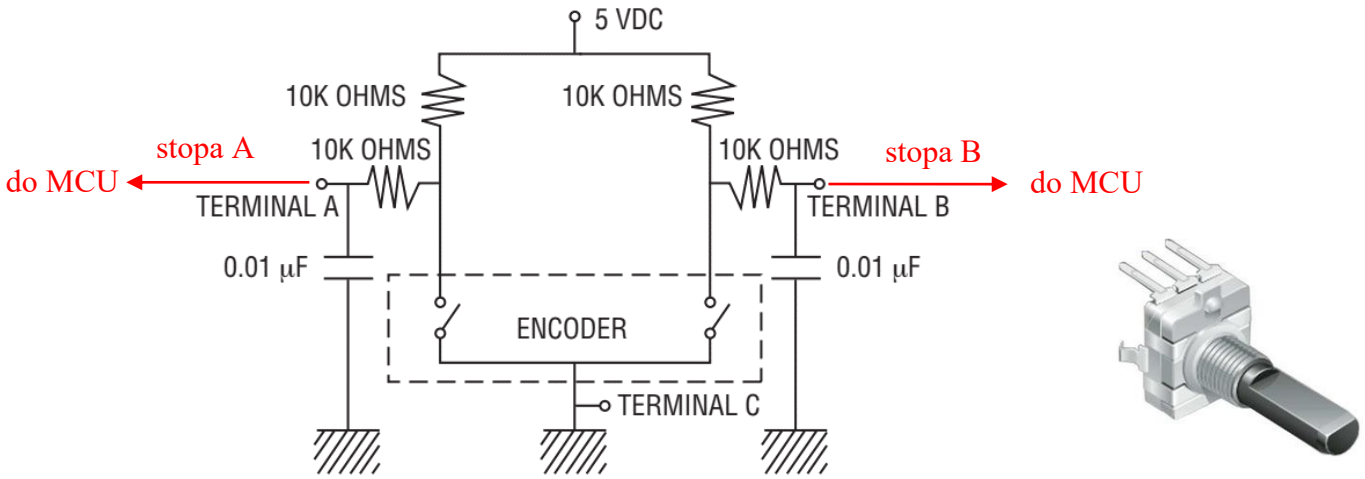
QA, QB – signály ze snímače

QCLK – interní signál odvozený z hran signálů QA, QB (rozlišení x4)

QDIR – interní signál směru otáčení odvozený ze signálů QA, QB

2.3 Mechanický inkrementální enkodér

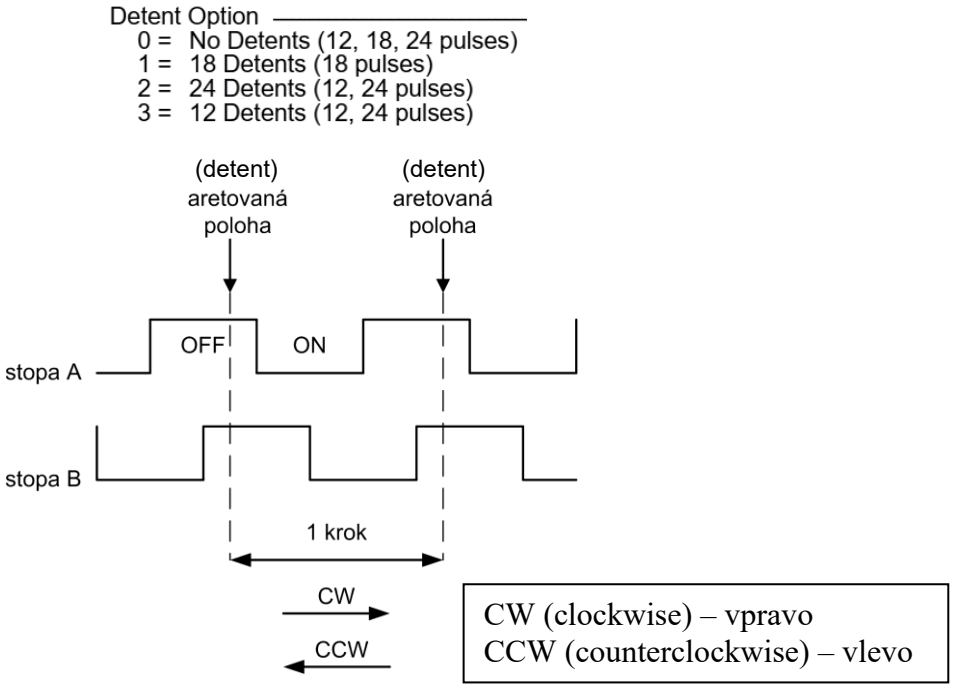
- Je inkrementální čidlo s kvadraturním výstupem.
- Vlastnosti mechanických inkrementálních enkodérů
  - Počet pulsů na otáčku - např. 20.
  - S aretací / bez aretace v jednotlivých polohách během otáčení.
  - Osa často plní funkci spínače.
  - Omezená max. rychlost otáčení – např. 60 ot/min.
  - Omezená životnost – např. 100 000 cyklů.
  - Vývody:
    - A, B – dvě stopy
    - GND
    - SW1,2 – dva piny spínací kontakt (jen pokud osa slouží jako tlačítko)
  - Proudová zatížitelnost kontaktů – např. 10 mA / 5 V.
  - Obvykle výrazné zakmitávání (odskoky) kontaktů – komplikují vyhodnocení signálu ze snímače.
    - Vyžaduje HW a/nebo SW filtraci signálů ze snímače.
    - HW filtrace – min. vyhlazovací RC článek na obou stopách.
    - SW filtrace – opakovaná čtení, vkládání prodlev mezi čteními nebo použití vhodného algoritmu (tabulky přechodů) pro zpracování signálů z enkodéru.
- Doporučené el. zapojení mech. rotačního enkodéru
  - Včetně HW filtrace zákmitů pomocí RC článku.
  - V základní (aretované) poloze jsou mech. kontakty na stopě A i B rozpojené.



Obrázek 37 Doporučené el. zapojení mech. rotačního enkodéru Bourns PEC11L

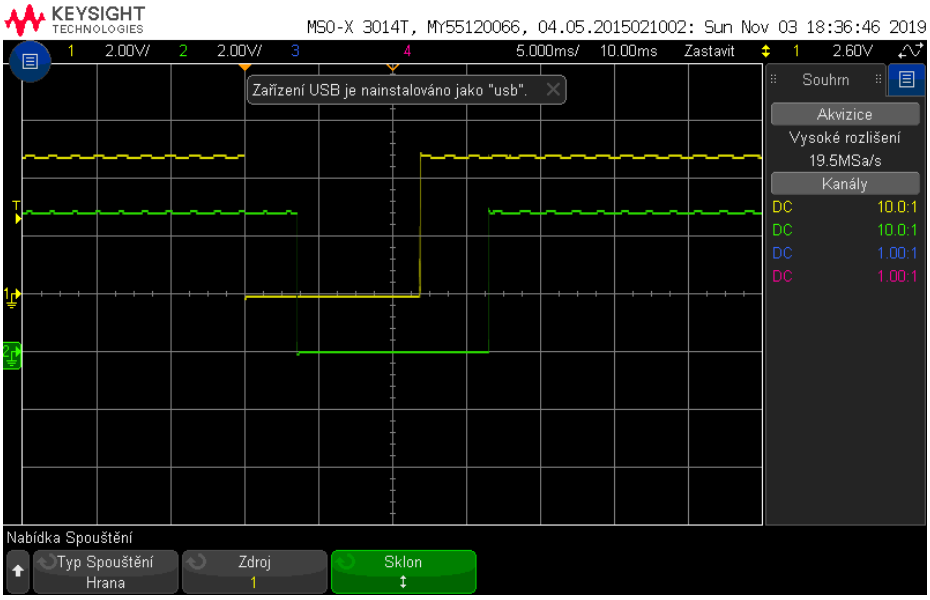
- Časová konstanta RC článku při rozepnutí kontaktu = (10 kΩ + 10 kΩ) * 10 nF = 0,2 ms.
- Časová konstanta RC článku při sepnutí kontaktu = 10 kΩ * 10 nF = 0,1 ms.

- **Signály z mech. rotačního enkodéru**
  - Zde uvedené průběhy signálů platí jen za podmínky **Detents = Pulses (počet aretovaných poloh = počet pulsů)**.

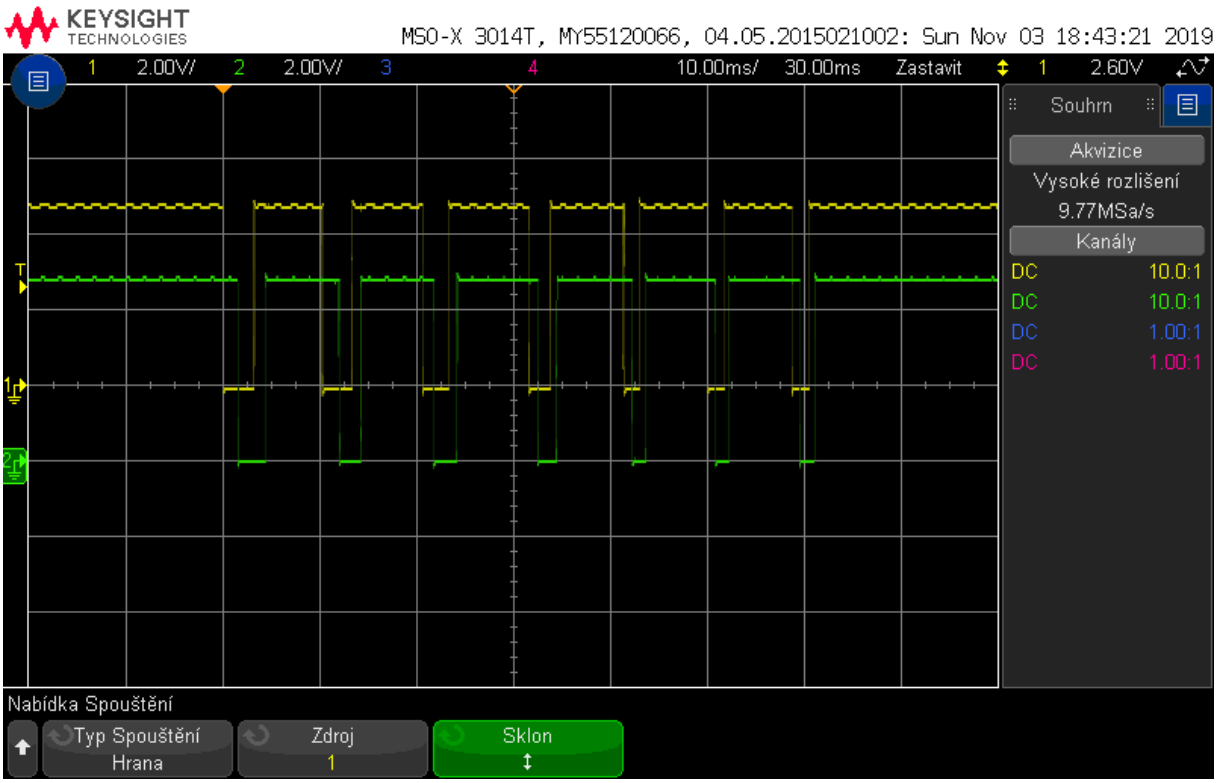


Obrázek 38 Signály z mech. rotačního enkodéru

- Skutečné signály z mech. rotačního enkodéru vypadají takto:

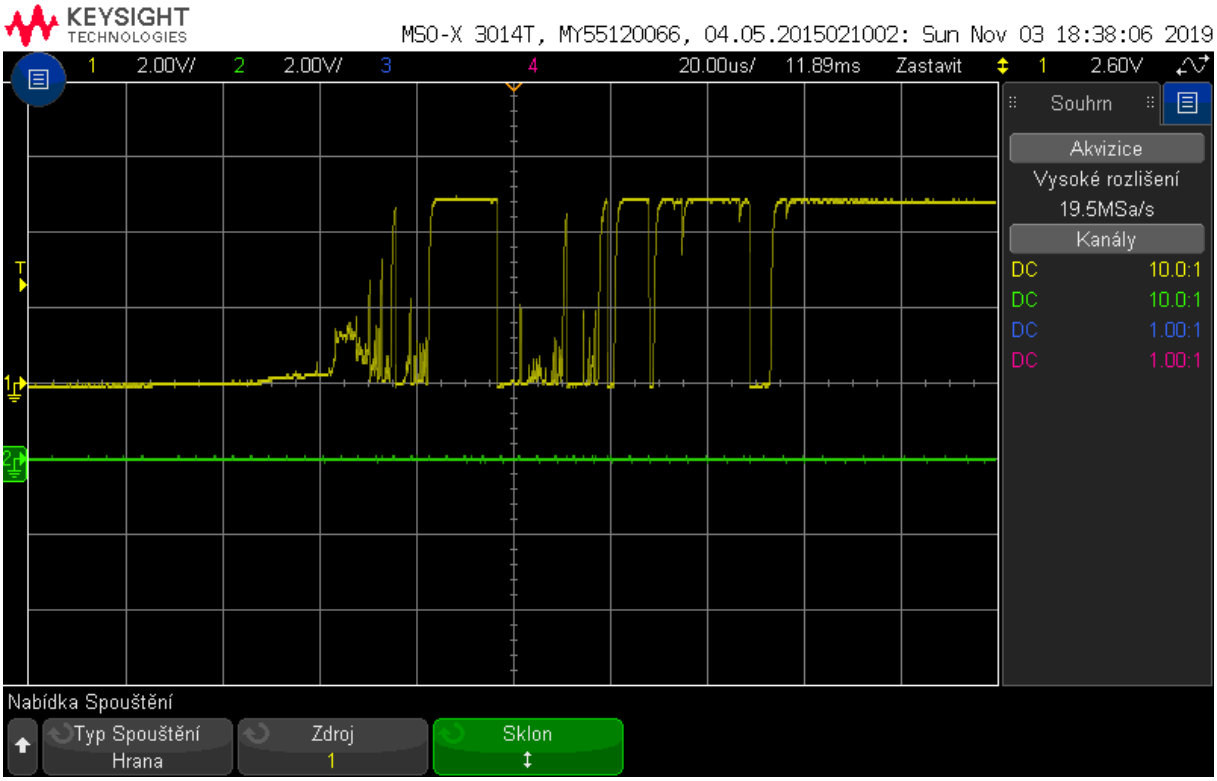


Obrázek 39 Stopa A a B mech. enkodéru – záznam z OSC – jeden krok vpravo



Obrázek 40 Stopa A a B mech. enkodéru – záznam z OSC – 7 kroků vpravo

- Dochází k zákmitům na hranách signálu stopy A i B:



Obrázek 41 Stopa A a B mech. enkodéru – záznam z OSC – odskakování kontaktu na stopě A

- Zpracování signálu z mech. enkodéru se provádí obvykle softwarově v MCU.
- Různé metody SW zpracování se liší:
    - Složitostí.
    - Výpočetní náročností.
    - Odolností vůči zákmitům kontaktů snímače.
    - Jak velkou rychlost otáčení dovolí.
- Obvyklé připojení čidla k MCU (různě, dle způsobu zpracování signálů):
    - **Obě stopy z čidla na jakékoliv dva digitální vstupy MCU - polling**
        - SW musí neustále a dostatečně rychle (alespoň každou 1 ms) číst stavy na těchto digitálních vstupech, aby nedošlo k vynechání kroku a i rychlá otočení byla zaznamenána.
    - **Jednu stopu z čidla na externí přerušení, druhou stopu na jakýkoliv digitální vstup MCU**
        - Odpadá neustálé zjišťování stavu signálů čidla (polling), MCU reaguje jen, když je potřeba (tj. když se osa enkodéru otáčí).
        - Přerušení zajistí, že nedojde k vynechání kroku ani při vysokých rychlostech otáčení.
        - Riziko častého vyvolávání přerušení při zakmitávání mech. kontaktů.
- **Princip zpracování signálů**
    1) SW filtrace zákmitů.
    2) Zjištění směru otáčení – např. detekcí logické úrovně na druhém kanálu v okamžiku hrany na prvním kanálu nebo i jinak (např. tabulka přechodů).
    3) Výpočet relativní polohy – inkrementace / dekrementace čítače podle směru otáčení.

2.3.1 Základní způsob vyhodnocení signálů

- Používá polling (neustále periodické čtení stavu stop A a B).
- Základní nejjednodušší způsob, nepříliš dokonalý.
    - Výhody:
        - Jednoduchost.
    - Nevýhody
        - Nedostatečně odolné vůči zákmitům mech. kontaktů (nespolehlivé).
        - Omezení max. rychlosti otáčení (čím více chceme potlačovat zákmity, tím více klesá max. rychlost otáčení, kterou dokážeme zpracovat).
        - Polling.
- Obě stopy ze snímače se připojí na jakékoliv dva digitální vstupy MCU
    - SW musí neustále a dostatečně rychle (alespoň každou 1 ms) číst stavy na těchto digitálních vstupech, aby nedošlo k vynechání kroku a i rychlá otočení byla zaznamenána.
- Postup:
    1) Filtrace signálů na stopách A a B.
    2) Detekce sestupné hrany na stopě A – tím se zjistí, že došlo k pohybu.
    3) Zjištění logické úrovně na stopě B – tím se určí směr otáčení.
    4) Inkrementace nebo dekrementace čítače podle směru otáčení.
- Celý kód pro AVR:

Pro enkodér, u něhož platí:
Detents = Pulses
(počet aretovaných poloh = počet pulsů)

```
#include <avr/io.h>
#define F_CPU 147456000
#include <util/delay.h>

// definice pripojeni mechanickeho inkrementalniho enkoderu k MCU
#define ENC_PORT      PORTD
#define ENC_DDR       DDRD
#define ENC_PIN       PIND
#define ENC_A_PIN     PD2    // stopa A
#define ENC_B_PIN     PD3    // stopa B

// promenne pro enkoder
uint8_t enc_state, enc_state_old; // aktualni a minuly stav enkoderu
int8_t steps = 0; // citac kroku enkoderu

int main(void)
{
	uint8_t enc_A_moved;

	// piny pro enkoder vstupni, interni pull-upy vypnuty, musi byt externi
	ENC_PORT &= ~((1<<ENC_A_PIN)|(1<<ENC_B_PIN));
	ENC_DDR &= ~((1<<ENC_A_PIN)|(1<<ENC_B_PIN));

	// nastaveni vychoziho stavu enkoderu po zapnuti
	enc_state = ENC_PIN;  // cteni urovni na stopach A a B (na celem portu)
```

```
    while(1)
    {
        enc_state_old = enc_state;  // zaloha minuleho stavu stop A a B

        // cteni logickych urovni na stopach A a B (cte se cely port) s osetrenim zakmitu
       do
       {
              enc_state = ENC_PIN;
              delay_ms(1);  // prodlevu mezi ctenimi nastavit zkusmo
       } while (enc_state != ENC_PIN);

       // detekce sestupne hrany na stope A
        enc_A_moved = enc_state_old ^ enc_state;  // detekce zmeny stavu na vsech pinech
portu
        enc_A_moved &= ~enc_state;         // zajima nas pouze prechod z urovne HIGH do LOW
        enc_A_moved &= 1<<ENC_A_PIN;       // zajima nas pouze stopa A

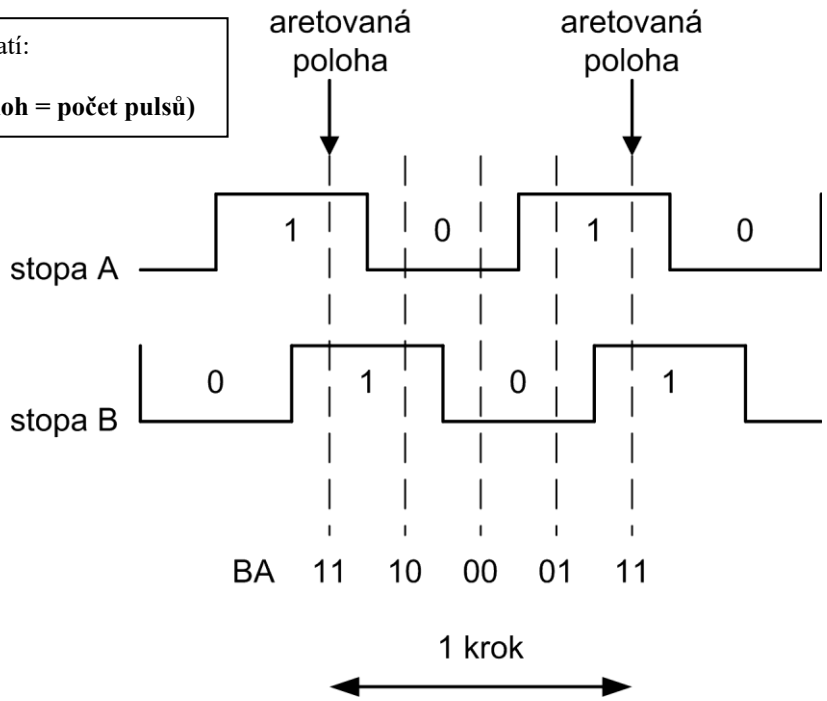
        if (enc_A_moved) // reagujeme pouze pokud na stope A doslo k prechodu z HIGH do LOW
        {
              if (enc_state & (1<<ENC_B_PIN)) steps++;  // pokud je na stope B uroven HIGH,
tak byl krok vpravo
              else steps--; // pokud je na stope B uroven LOW, tak byl krok vlevo
        }
    }
}
```

- Výstupem z algoritmu je počet kroků od zapnutí MCU, kroky jsou uloženy v proměnné `steps`.
- Časovou prodlevu v části kódu, který ošetřuje zákmity, je potřeba nastavit metodou pokus-omyl pro konkrétní enkodér. Dobrému enkodéru bude stačit krátká prodleva, špatný enkodér potřebuje delší prodlevu.
    - Příliš dlouhá prodleva znemožní rychlé krokování (program přestane reagovat na rychlé otáčení osy enkodéru).
    - Příliš krátká prodleva způsobí reagování programu i na zákmity.
- Důsledek vůbec neošetřených nebo špatně ošetřených zákmitů:
    - Např. při jednom kroku vpravo by se měl obsah čítače `counter` zvýšit o 1, např. z hodnoty 0 na 1. Ale zákmity způsobí hned několik po sobě jdoucích změn, např. $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 4 \rightarrow 5$.

2.3.2 Použití tabulky přechodů

- Zde použijeme polling, ale lze použít i přerušení.
- Složitější algoritmus, ale má výhodnější vlastnosti.
    - Výhody:
        - Dobrá odolnost vůči zákmitům.
        - Umožňuje vysokou rychlost otáčení enkodéru.
    - Nevýhody:
        - Složitější.
        - Delší kód, více proměnných v RAM.
- Uvedený algoritmus je odolný proti zákmitům mech. kontaktů enkodéru, měl by si poradit i se signály z enkodéru bez filtračních kondenzátorů. Ošetření zákmitů je provedeno bez vkládání prodlev mezi opakovaným čtením stavu signálů enkodéru – neomezuje rychlost otáčení enkodéru.
- **Obě stopy ze snímače se připojí na jakékoliv dva digitální vstupy MCU – použijeme polling**
    - SW musí neustále a rychle (alespoň každou 1 ms) číst stavy na těchto digitálních vstupech, aby nedošlo k vynechání kroku a i rychlá otočení byla zaznamenána.
- Během jednoho kroku (od jedné aretované polohy do další aretované polohy) se na stopách A, B objevují tyto logické úrovně:

Pro enkodér, u něhož platí:
Detents = Pulses
(počet aretovaných poloh = počet pulsů)



Obrázek 42 Přechody během jednoho kroku

- Celkem máme 4 stavy a celkem 8 možných přechodů mezi nimi (4 přechody pro otáčení vpravo a 4 přechody pro otáčení vlevo).

- **Směr otáčení se určuje z posloupnosti stavů.**
- Uvažujeme, že každý stav je zakódován do 2-bitového čísla. Úroveň na stopě B je více význačný (horní) bit, úroveň na stopě A je méně význačný (spodní) bit.
- Pro směr otáčení **vpravo** (CW – clockwise – po směru hodinových ručiček) se přechází mezi stavy v tomto pořadí: **0b11 → 0b10 → 0b00 → 0b01 → 0b11**
- Ve směru otáčení **vlevo** (CCW – counterclockwise – proti směru hodinových ručiček) se přechází mezi stavy v opačném pořadí: **0b11 → 0b01 → 0b00 → 0b10 → 0b11**
- K zákmitům dochází pouze při změně logické úrovně na jedné stopě, logická úroveň na druhé stopě se v ten okamžik nemění - toho lze využít pro vyfiltrování zákmitů.
- **Vytvoří se tabulka přechodů**
  - Definuje správné a chybné přechody. Chybné přechody jsou ignorovány.
  - **Každý přechod je zakódován do 4-bit čísla, které se skládá z:**
    - Kódu minulého stavu (horní dva bity).
    - Kódu aktuálního stavu (dolní dva bity).
    - **Např. 0b1110 – minulý stav byl 0b11, nový stav je 0b10**
  - Validní přechody pro směr **vpravo** (CW) jsou:

    BIN: 0b1110 → 0b1000 → 0b0001 → 0b0111

    HEX: 0xE → 0x8 → 0x1 → 0x7

    DEC: 14 → 8 → 1 → 7
  - Validní přechody pro směr **vlevo** (CCW) jsou:

    BIN: 0b1101 → 0b0100 → 0b0010 → 0b1011

    HEX: 0xD → 0x4 → 0x2 → 0xB

    DEC: 13 → 4 → 2 → 11

Tabulka 22 Tabulka přechodů pro inkrementální kvadraturní enkodér

Kód přechodu			Validní přechod	Směr otáčení
(DEC)	(HEX)	(BIN)		
0	0	0000	NE	-
1	1	0001	ANO	Vpravo (CW)
2	2	0010	ANO	Vlevo (CCW)
3	3	0011	NE	-
4	4	0100	ANO	Vlevo (CCW)
5	5	0101	NE	-
6	6	0110	NE	-
7	7	0111	ANO	Vpravo (CW)
8	8	1000	ANO	Vpravo (CW)
9	9	1001	NE	-
10	A	1010	NE	-
11	B	1011	ANO	Vlevo (CCW)
12	C	1100	NE	
13	D	1101	ANO	Vlevo (CCW)
14	E	1110	ANO	Vpravo (CW)
15	F	1111	NE	-

minulý stav — nový stav

- **Vstupem do tabulky je kód přechodu, výstupem z tabulky je směr otáčení.**
- Směru otáčení vpravo (CW) přiřadíme číselnou hodnotu +1.
- Směru otáčení vlevo (CCW) přiřadíme číselnou hodnotu -1.
- Směru otáčení při neplatném přechodu přiřadíme číselnou hodnotu 0.
- V kódu pak vytvoříme tabulku přechodů takto:

```
int8_t enc_trans_table[16] = {0,+1,-1,0,-1,0,0,+1,+1,0,0,-1,0,-1,+1,0};
```

- Index do tabulky (kód přechodu) vytvoříme takto:

```
// cteni stop enkoderu
uint8_t enc_state = ENC_PIN;     // cteni stavu celeho portu
enc_state >>= ((ENC_A_PIN < ENC_B_PIN) ? ENC_A_PIN : ENC_B_PIN);   // posunuti
vpravo, aby nejnizsi dva bity byly stopa B a stopa A
enc_state &= 0x03; // zajimaji nas pouze nejnizsi dva bity (stopa B a stopa A)

// vytvoreni 4-bit kodu prechodu (indexu do tabulky)
enc_index <<= 2;          // stary stav horni dva bity
enc_index |= enc_state;   // novy stav dolni dva bity
enc_index &= 0x0F;        // 4-bit kod prechodu (index do tabulky)
```

Stav enkodéru `enc_state`:

7	6	5	4	3	2	1	0
0	0	0	0	0	0	B	A

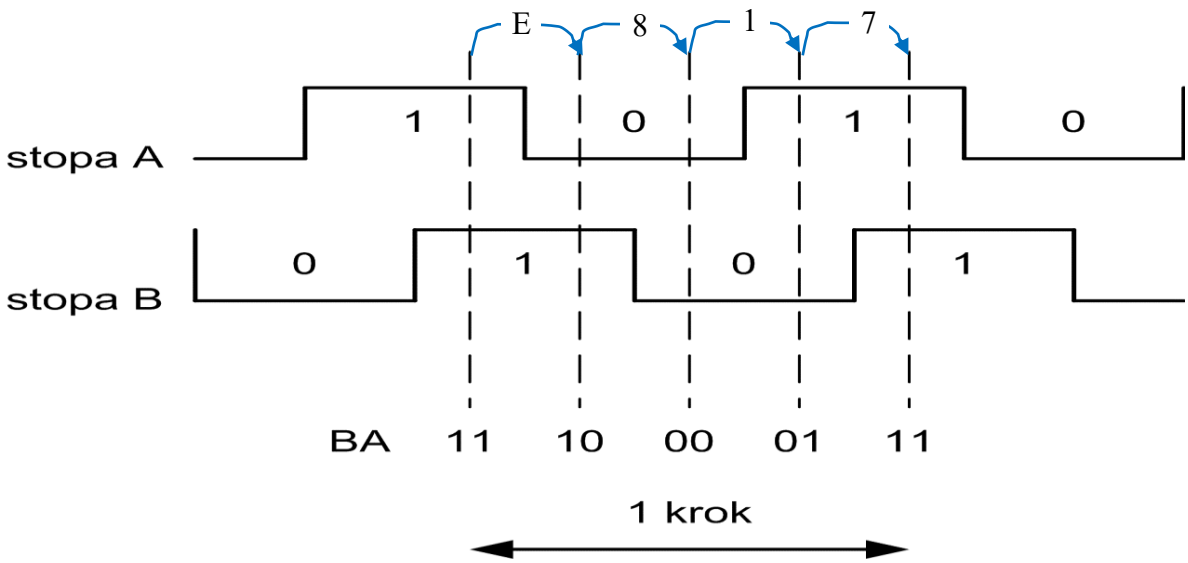
Kód přechodu `enc_index`:

7	6	5	4	3	2	1	0
0	0	0	0	B_{old}	A_{old}	B	A

Př. 1: krok vpravo – **kvalitní enkodér** (nezakmitává), posloupnost přechodů bude:

HEX: 0xE → 0x8 → 0x1 → 0x7

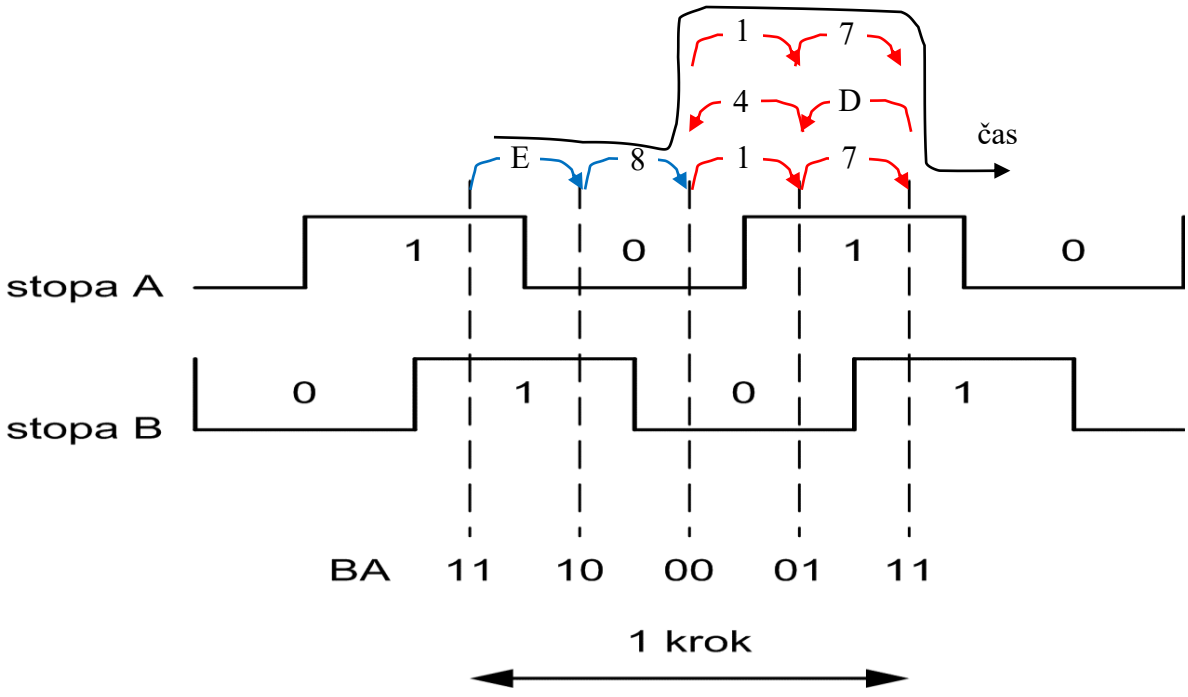
BIN: 0b1110 → 0b1000 → 0b0001 → 0b0111



- **Př. 2:** krok vpravo – **nekvalitní enkodér** (zakmitává), posloupnost přechodů bude:

HEX: 0xE → 0x8 → 0x1 → 0x4 → 0x1 → 0x7 → 0xD → 0x7

BIN: 0b1110 → 0b1000 → 0b0001 → 0b0100 → 0b0001 → 0b0111 → 0b1101 → 0b0111



- Při každém přechodu se podle směru otáčení vyčteného z tabulky inkrementuje nebo dekrementuje průběžné počítadlo `enc_counter`, které ve výsledku slouží pro určení směru otáčení po dosažení nové aretované polohy (po dosažení stavu 0b11).

```
dir = enc_trans_table[enc_index];
if (dir == 1) enc_counter++;
if (dir == -1) enc_counter--;
```

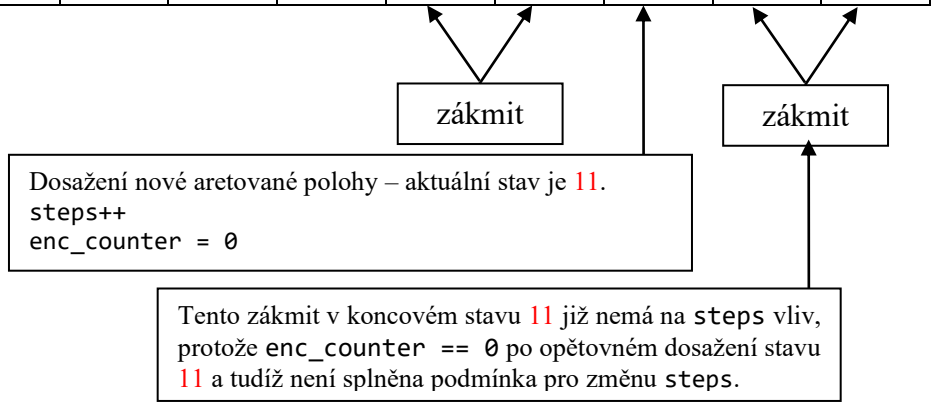
- Po dosažení nové aretované polohy, tj. stavu 0b11 (desítkově 3), se podle hodnoty čítače `enc_counter` určí „výsledný“ směr otáčení a teprve podle něj se buď inkrementuje nebo dekrementuje čítač kroků enkodéru `steps`. Následně se `enc_counter` vynuluje.

```
if (enc_state == 3) // pri dosazeni nove aretovane polohy 0b11
{
        if (enc_counter > 0) steps++;
        if (enc_counter < 0) steps--;
        enc_counter = 0;
}
```

- Zákmity jsou efektivně odfiltrovány tím, že se jejich projevy vyskytují „v párech“ (odskok – doskok), tudíž neovlivňují výslednou hodnotu čítače `enc_counter` po dosažení aretované polohy 0b11.
    - Odskok při otáčení vpravo: odečte se 1 a následně se přičte 1 → -1+1 = 0
    - Odskok při otáčení vlevo: přičte se 1 a následně se odečte 1 → 1-1 = 0
    - **Příklad:**

      1 krok vpravo, nekvalitní enkodér, odskoky (obrázek s přechody je výše)

č. přechodu HEX	0xE	0x8	0x1	0x4	0x1	0x7	0xD	0x7
č. přechodu BIN	0b1110	0b1000	0b0001	0b0100	0b0001	0b0111	0b1101	0b0111
vyhodnocený směr otáčení z tabulky	CW	CW	CW	CCW	CW	CW	CCW	CW
`enc_counter`	1	2	3	2	3	4 (0)	-1	0
`steps`	0	0	0	0	0	1	1	1



- Celý kód pro AVR:

```
#include <avr/io.h>

// definice pripojeni mechanickeho inkrementalniho enkoderu, musi byt kterekoliv dva piny
vedle sebe
#define ENC_PORT      PORTD
#define ENC_DDR       DDRD
#define ENC_PIN       PIND
#define ENC_A_PIN     PD2     // stopa A
#define ENC_B_PIN     PD3     // stopa B

// definice smeru otaceni
#define CW    1       // vpravo
#define CCW   -1      // vlevo

// promenne spojene s obsluhou enkoderu
int8_t  enc_trans_table[16]  =  {0,CW,CCW,0,CCW,0,0,CW,CW,0,0,CCW,0,CCW,CW,0};  //  tabulka
prechodu pro mech. enkoder
uint8_t enc_state;           // aktualni stav
uint8_t enc_index = 0;       // kod prechodu (index do tabulky prechodu)
int8_t enc_counter = 0;      // prubezne pocitadlo prechodu

// pocitadlo kroku enkoderu
int8_t steps = 0;

int main(void)
{
       int8_t dir;

       // piny pro enkoder vstupni, interni pull-upy vypnuty, musi byt externi
       ENC_PORT &= ~((1<<ENC_A_PIN)|(1<<ENC_B_PIN));
       ENC_DDR &= ~((1<<ENC_A_PIN)|(1<<ENC_B_PIN));

       // nastaveni vychoziho stavu enkoderu po zapnuti
       enc_index = 0x03; // v aretovane poloze

    while(1)
    {
        // cteni stop enkoderu
        enc_state = ENC_PIN; // cteni stavu celeho portu
        enc_state >>= ((ENC_A_PIN < ENC_B_PIN) ? ENC_A_PIN : ENC_B_PIN);       // posunuti
vpravo, aby nejnizsi dva bity byly stopa B a stopa A
        enc_state &= 0x03;   // zajimaji nas pouze nejnizsi dva bity (stopa B a stopa A)

        // vytvoreni 4-bit kodu prechodu (indexu do tabulky)
        enc_index <<= 2;            // stary stav horni dva bity
        enc_index |= enc_state;     // novy stav dolni dva bity
        enc_index &= 0x0F;          // zajimaji nas pouze dolni 4 bity

        // cteni smeru z tabulky prechodu
        dir = enc_trans_table[enc_index];
        if (dir == CW) enc_counter++;
        if (dir == CCW) enc_counter--;

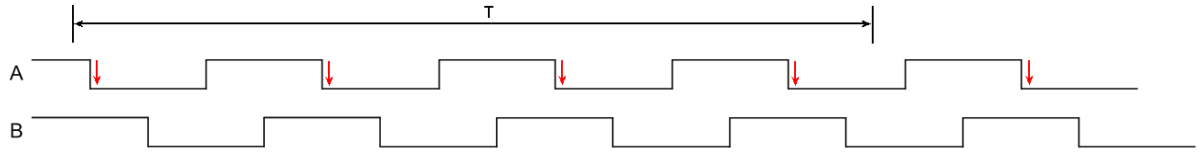
        if (enc_state == 3)  // pri dosazeni nove aretovane polohy
        {
               // inkrementuj nebo dekrementuj pocitadlo kroku podle smeru otaceni
               if (enc_counter > 0) steps++;
               if (enc_counter < 0) steps--;
               enc_counter = 0;     // nuluj prubezne pocitadlo prechodu
        }
    }
}
```

Kód lze použít jen pro enkodér, u něhož platí:
Detents = Pulses
(počet aretovaných poloh = počet pulsů)

Výstupem z algoritmu je počet kroků od zapnutí MCU, kroky jsou uloženy v proměnné `steps`.

2.4 Vyhodnocení rychlosti otáčení z IRC čidel (detailněji)

2.4.1 Měřením počtu pulsů (hran) za jednotku času



- Vhodné pro čidla s velkým počtem pulzů / otáčku
- Vhodné pro měření vyšších otáček (roste relativní přesnost měření)
- Informace o otáčkách je poskytovaná s konstantní periodou T
- **Princip**
  - Počítají se hrany (režim 1x, 2x nebo 4x) za pevný časový interval T.
  - Pevný časový interval T je odměřován časovačem.

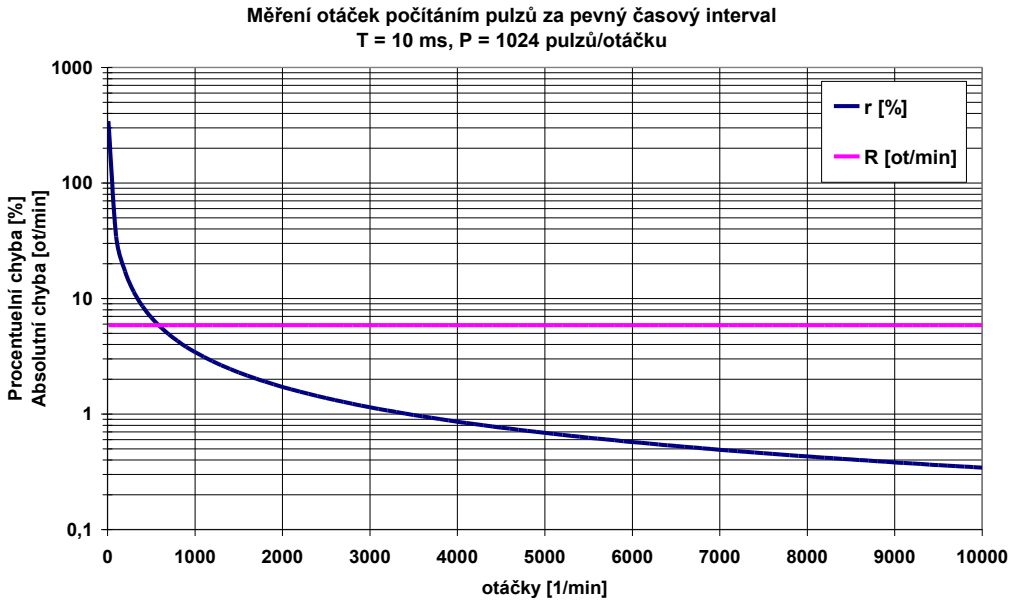
$$\omega(k) = \frac{\varphi(k) - \varphi(k-1)}{T} = \frac{\Delta\varphi(k)}{T} \tag{43}$$

    - ω(k) – úhlová rychlost otáčení v čase kT
    - φ(k) – úhlové natočení (poloha) v čase kT
    - φ(k-1) – poloha v čase (k-1)T
    - Δφ(k) – přírůstek polohy za čas T
    - T – pevný časový interval měření

$$\omega(k) = \text{Konstanta} \cdot \text{Počet načítaných hran za periodu T} \tag{44}$$

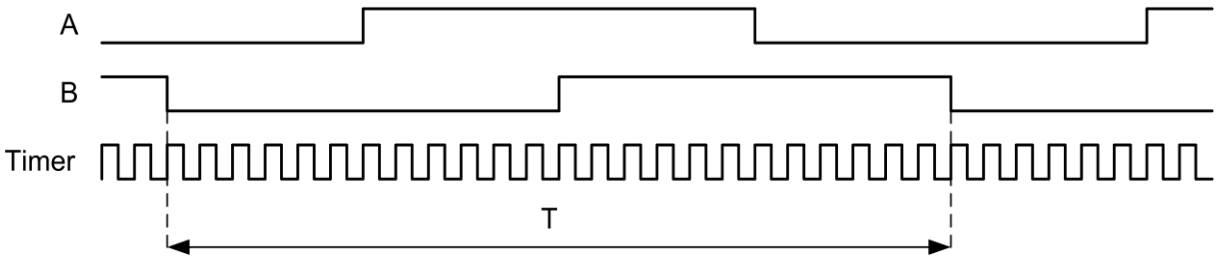
  - **Připojení k MCU**
    - Ruční počítání vzest./sest. hran za periodu T
      - Signál z jednoho (nebo obou, pokud je třeba rozlišovat směr) kanálu(ů) IRC snímače přiveden na vstup(y) externího přerušení INTx.
      - Časově náročné na zpracování (frekvence signálu z IRC čidla závisí na měřených otáčkách a počtu pulzů na otáčku, může dosahovat až desítek kHz).
    - Využití čítače – není nutné reagovat na každou hranu.
      - Rozpoznání směru – čtením logické úrovně druhého kanálu jednou za periodu T.
    - Použití externího obvodu pro zpracování signálu.

- **Např.** T = 10 ms, P = 1024 pulzů / otáčku
  - Předpokládám počítání vzest. nebo sest. hran z jednoho kanálu IRC.
  - Pevná rozlišovací schopnost: $R = \frac{60}{P \cdot T} = \frac{60}{1024 \cdot 0,010} = 5,88$ ot/min
  - **n = 100 ot/min**
    - Za 10 ms se při 100 ot/min načítá $N = \frac{n \cdot P \cdot T}{60} = \frac{100 \cdot 1024 \cdot 0,01}{60} = 17$ pulzů
    - N znám → otáčky se vypočtou jako: $n = \frac{N \cdot 60}{P \cdot T}$ ot/min
    - Rozlišení v měření otáček: $R = \frac{n}{N} = \frac{100}{17} = 5,88$ ot/min
    - Procentuální chyba měření otáček: $r = \frac{R}{n} \cdot 100 = \frac{5,88}{100} \cdot 100 = 5,88\%$
  - **n = 1000 ot/min**
    - Za 10 ms se při 1000 ot/min načítá $N = \frac{n \cdot P \cdot T}{60} = \frac{1000 \cdot 1024 \cdot 0,01}{60} = 170$ pulzů
    - Rozlišení v měření otáček: $R = \frac{n}{N} = \frac{1000}{170} = 5,88$ ot/min
    - Procentuální chyba měření otáček: $r = \frac{R}{n} \cdot 100 = \frac{5,88}{1000} \cdot 100 = 0,588\%$
  - **Závěr**
    - Konstantní rozlišení v otáčkách (konst. absolutní chyba měření) → při vysokých otáčkách malá procentuální chyba, ale při nízkých velká !
    - Viz příklad nahoře: R = 5,88 ot/min → otáčky pod 5,88 ot/min nelze měřit (bude po většinu času ukazovat 0 ot/min).
    - Tento způsob vyhodnocení se nehodí pro měření velmi malých otáček.



- **Způsoby zvýšení přesnosti (rozlišení)**
  - Delší časový interval čítání T
  - Počítání vzestupných i sestupných hran (rozlišení x2), na obou kanálech (rozlišení x4)
  - IRC s vyšším počtem pulzů na otáčku

2.4.2 Měřením periody (vzdálenost hran)



- Měří se perioda T signálu ze snímače otáček pomocí volně běžícího časovače (Timer).
- Vhodné pro čidla s malým počtem pulzů na otáčku.
- Vhodné pro měření malých otáček.
- Informace o otáčkách je poskytována nepravidelně (T není konstantní a závisí na otáčkách).
- **Princip**
  - Měří se počet tiků N časovače za jednu periodu T signálu ze snímače otáček → N je následně převedeno na otáčky *n* (vynásobením konstantou).
  - Přesnost měření závisí na frekvenci volně běžícího časovače, který odměřuje periodu signálu T:
    - Vysoká frekv. časovače → vysoký počet načítaných tiků za periodu T→ vysoké rozlišení v periodě T (a tím v otáčkách).

Nevýhoda: časovač rychleji přetéká → omezení měřitelných otáček ze spodu, pokud není přetečení časovače zahrnuto do výpočtu.

- **Čím vyšší jsou měřené otáčky**, tím kratší je perioda T a tím menší počet tiků časovač načítá, **klesá přesnost měření**.
- **Čím nižší jsou měřené otáčky**, tím delší je perioda T a tím větší počet tiků časovač načítá, a **roste přesnost měření**.

- Dynamický rozsah měřitelných otáček závisí na šířce slova časovače:
  - Šířka časovače (např. 16-bit) – čím větší, tím pomaleji přetéká → to dovoluje rozšíření měřícího rozsahu směrem k nižším otáčkám nebo zvýšení frekvence (a tím rozlišení) při zachování měřícího rozsahu.
- **Využití Input Capture jednotky** (vyhnutí se latencím při obsluze přerušení).

- **Např.** 16-bit T/C, $F_{T/C}$ =14,7456 MHz, P = 1024 pulzů / otáčku
  - Perioda přetečení T/C: $T_{OVF} = \frac{2^{16}}{F_{T/C}} = \frac{65536}{14745600} = 4,444 \text{ ms}$
  - Minimální měřitelná rychlost (než dojde k přetečení T/C), pokud není do výpočtu zahrnuto přetečení časovače:

$$n_{\min} = \frac{60}{T_{OVF} \cdot P} = \frac{60}{4,444 \cdot 10^{-3} \cdot 1024} = 13,185 \text{ ot/min}$$

  - Kdyby byl T/C 32-bit:

$$T_{OVF} = \frac{2^{32}}{F_{T/C}} = \frac{4294967296}{14745600} = 291,71 \text{ s}$$

$$n_{\min} = \frac{60}{T_{OVF} \cdot P} = \frac{60}{291,71 \cdot 1024} = 0,0002 \text{ ot/min}$$

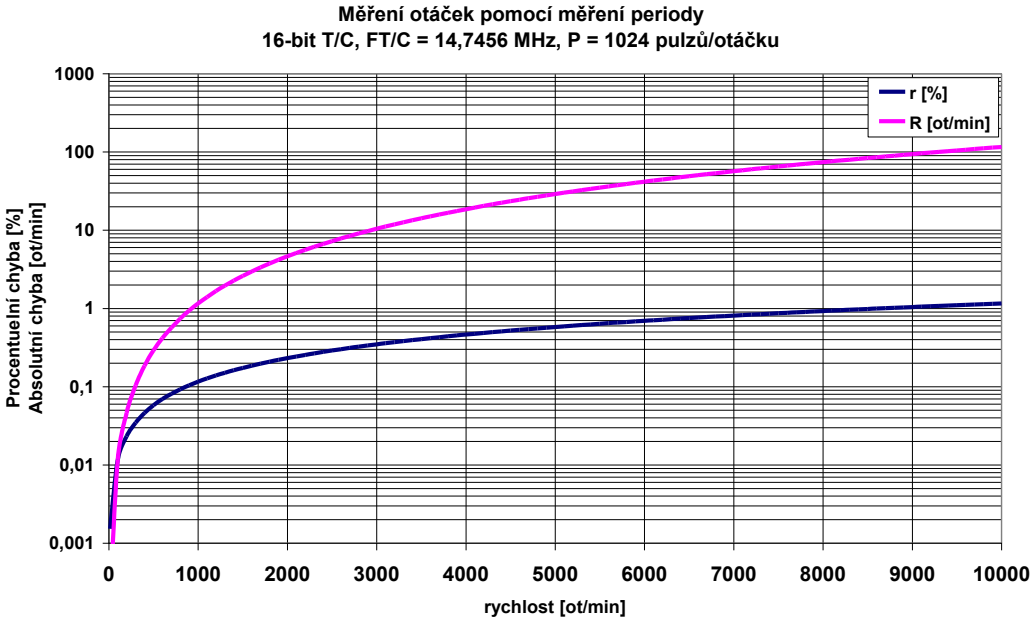
- **n = 100 ot/min**
  - $T = \frac{60}{n} = \frac{60}{100} = 0,6 \text{ s}$
  - Za T = 0,6 s časovač načítá $N = \frac{F_{T/C} \cdot T}{P} = \frac{14745600 \cdot 0,6}{1024} = 8640$ tiků
  - Převod na otáčky:

$$n_{vypoč} = \frac{k}{N} = \frac{60 \cdot 2^{16}}{P \cdot T_{OVF}} \cdot \frac{1}{N} = \frac{60 \cdot 65536}{1024 \cdot 4,44 \cdot 10^{-3}} \cdot \frac{1}{8640} = 100 \text{ ot/min}$$

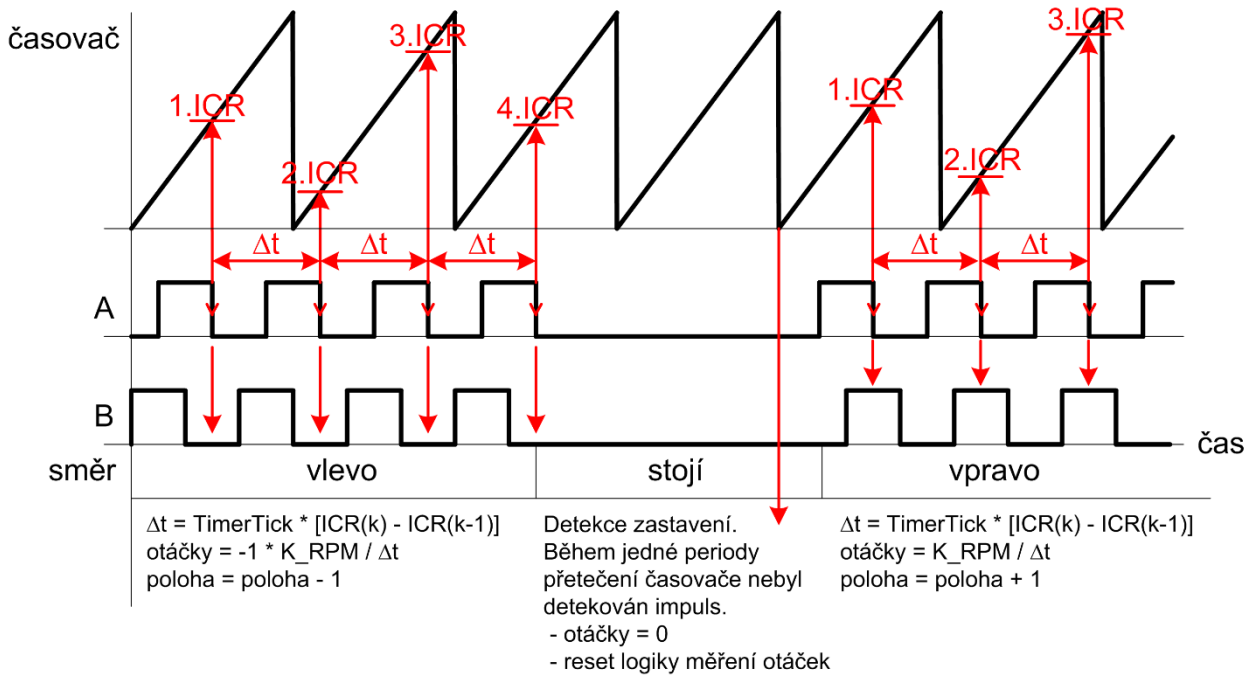
  - Rozlišení v otáčkách: $R = \frac{n}{N} = \frac{100}{8640} = 0,01157$ ot/min
  - Procentuální chyba měření otáček:

$$r = \frac{R}{n} \cdot 100 = \frac{0,01157}{100} \cdot 100 = 0,01157\,\%$$

- **n = 1000 ot/min**
  - $T = \frac{60}{n} = \frac{60}{1000} = 0,06 \text{ s}$
  - Za T = 0,06 s časovač načítá $N = \frac{F_{T/C} \cdot T}{P} = \frac{14745600 \cdot 0,06}{1024} = 864$ tiků
  - Rozlišení v otáčkách: $R = \frac{n}{N} = \frac{1000}{864} = 1,157$ ot/min
- Procentuální chyba měření otáček: $r = \frac{R}{n} \cdot 100 = \frac{1,157}{1000} \cdot 100 = 0,1157\,\%$



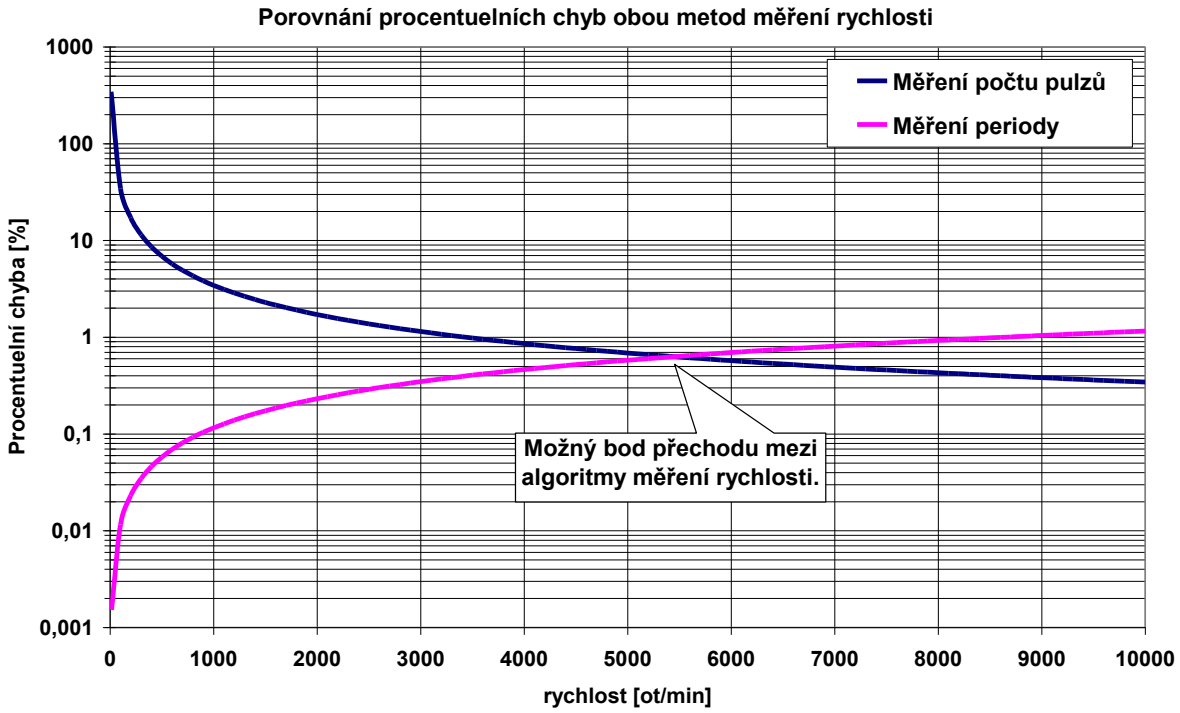
- **Způsoby zvýšení přesnosti (rozlišení)**
  - Vyšší frekvence časovače
  - Paradoxně IRC čidlo s nižším počtem pulsů na otáčku
    - Nevýhoda - informace o otáčkách méně často (na delším úhlovém natočení měříme spíše průměrnou hodnotu otáček než okamžitou)



Obrázek 43 Princip měření otáček pomocí záchytné jednotky

2.4.3 Kombinace metod

- Pro nízké otáčky – měření periody
- Pro vysoké otáčky – počítání pulsů za pevnou periodu
- On-line přechod mezi způsoby měření

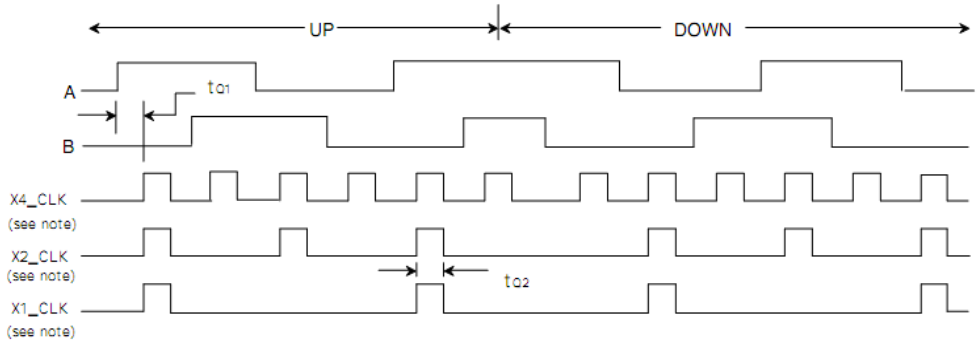


2.4.4 Obvody pro dekódování signálu z IRC čidel s kv. výstupem

- Odlehčení zátěže MCU – dekódování provádí spec. obvod
- Dvě skupiny obvodů:
  - Dekodéry kv. signálu
    - Výstupem je signál „Clock“ a „Direction“.
  - Čítače
    - Obsahují dekodér a čítač pulsů (hran).
    - Výstupem je obsah čítače (poloha).
    - Spojení s MCU přes sériovou nebo paralelní sběrnici.

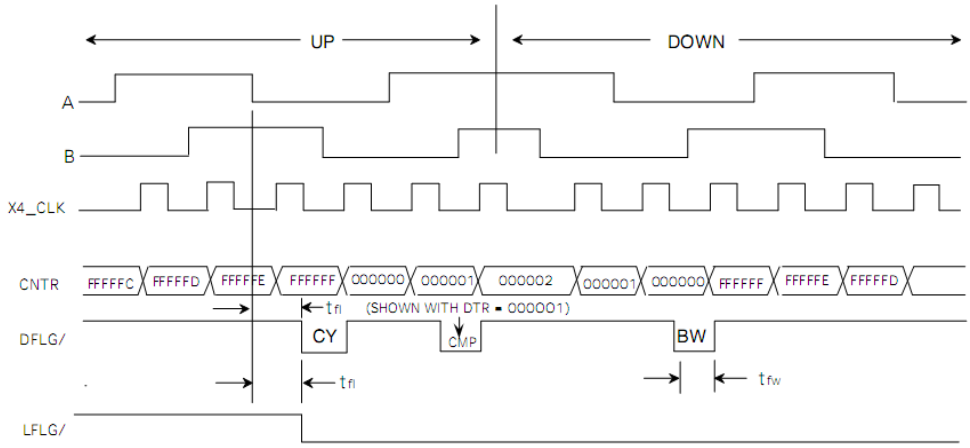
Např. obvod LS7366R (fy LSI Computer Systems)

- 32-bit Quadrature Counter with SPI interface
- Dekódování signálu z IRC čidel s kv. výstupem (stopy A, B, Index) nebo s nekv. výstupem (stopy A- signál, B – směr otáčení)
- Režim dekódování: 1x, 2x, 4x rozlišení

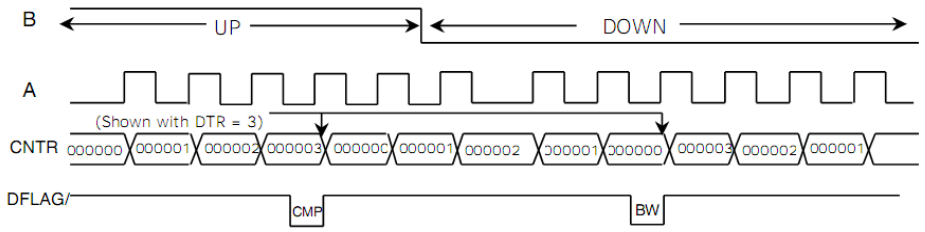


Note: x1, x2, and x4 CLKs are internal up/down clocks derived from filtered and decoded quadrature clocks.

- Obsahuje 32-bit čítač, konfigurovatelný jako 8, 16, 24 nebo 32-bit
- Dva konfigurovatelné výstupy (LFLAG, DFLAG)
  - Pro generování signálu externího přerušení
    - Výskytu ref. značky (Index)
    - Při přetečení čítače (v obrázku CY)
    - Podtečení čítače (v obrázku BW)
    - Dosažení komparační úrovně (v obrázku CMP)



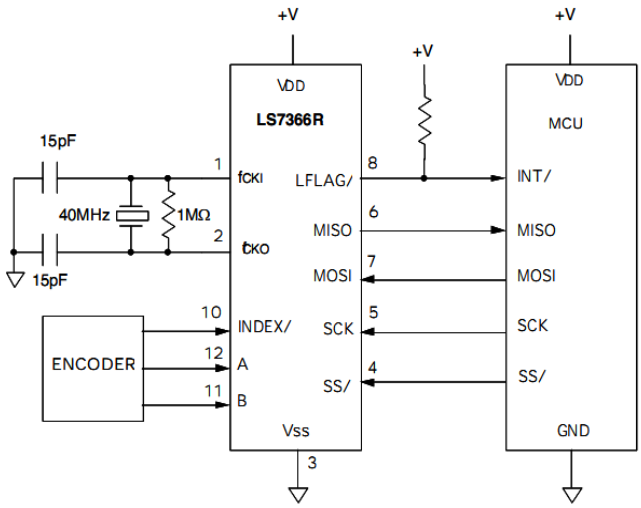
- Konfigurovatelné možnosti čítání
  - Free-running (čítá od 0 do max. rozsahu, pak opět od 0)
  - Modulo N (čítá od 0 do N, pak opět od 0)
  - Range-limit (čítá od 0 do nastaveného horního limitu, pak se zastaví)
  - Single-cycle (čítač se zastaví při podtečení/přetečení, rozběhne se po příchodu nulové značky Index)



NOTE: CNTR values are indicated in 1-byte mode

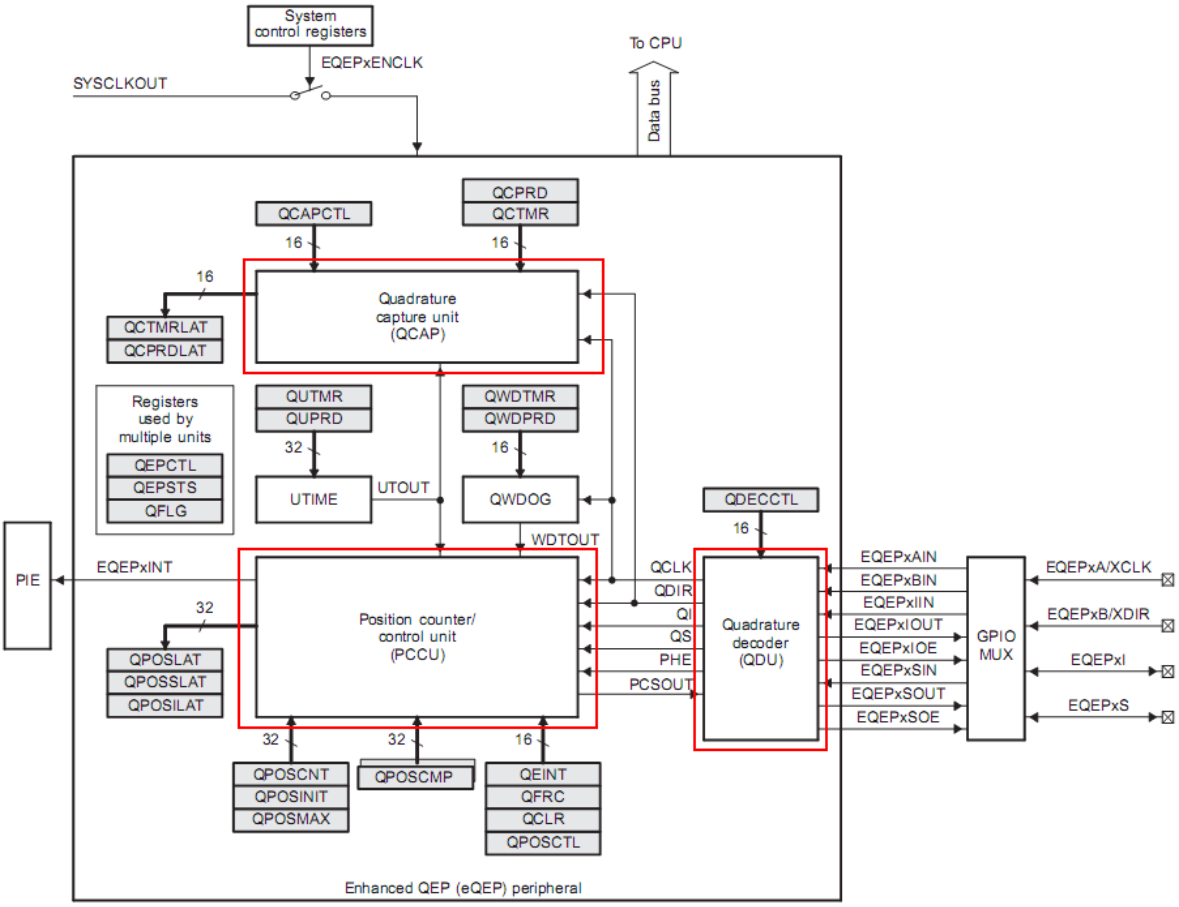
FIGURE 10. MODULO-N, NON-QUADRATURE

- Komunikace s MCU po sériové sběrnici SPI (MISO, MOSI, SCK, SS)



2.4.5 Rozhraní MCU (DSP) pro připojení IRC čidel s kvadraturním výstupem

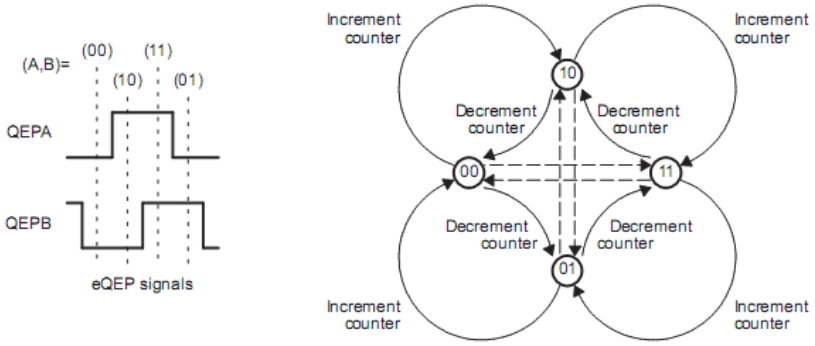
- Rozhraní pro připojení IRC čidel integrované v MCU.
- Výrazně usnadňuje měření otáček a polohy.
- Běžně se vyskytuje u DSP pro řízení el. pohonů.



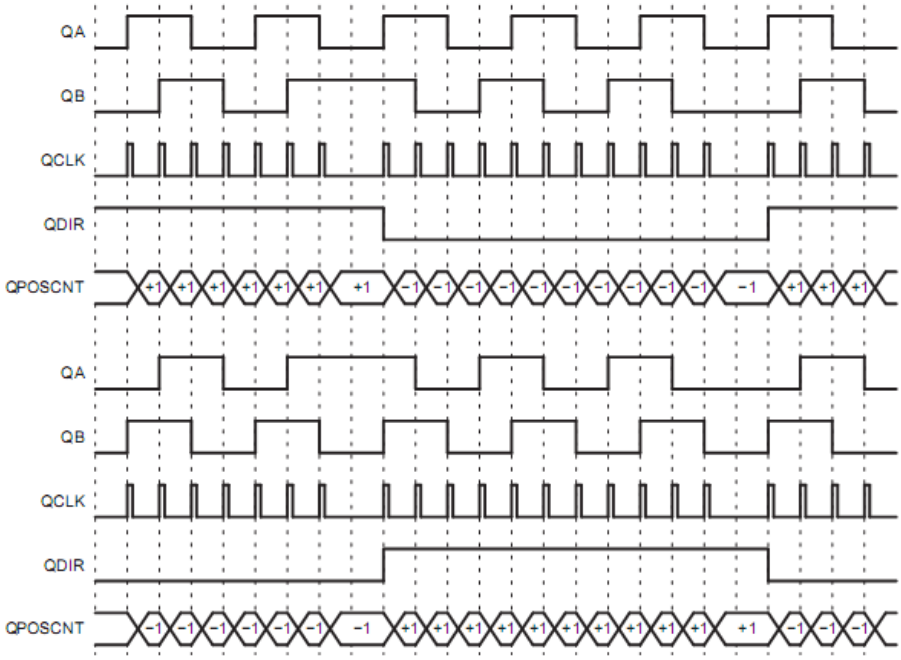
Obrázek 44 QEP jednotka u DSP TMS320Cxxx

Základní vlastnosti QEP jednotky

- **Vstupy**
  - kv. signál - A, B (fázově posunuté o 90°), Index, Strobe pin (uživ. definovatelná reakce na tento vstup – např. zachycení hodnoty čítače)
  - nekv. signál – A signál rychlosti, B směr, Index, Strobe pin
- **Quadrature Decoder (QDU)**
  - Vytváří signály pro čítač
    - Hodiny (QCLK) – 4x rozlišení
    - Směr čítání (QDIR)
  - Hlídá správný sled signálů na kanálech A a B
    - Při nesprávném přechodu nastaví Phase Error Flag (PHE)



Obrázek 45 Stavový automat kvadraturního dekodéru



Obrázek 46 Generování hodin (QCLK) a signálu směru otáčení (QDIR)

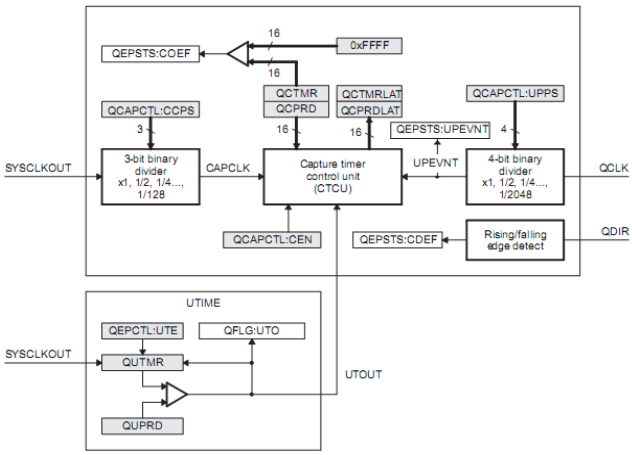
- **Position Counter (QPOSCNT)**
  - 32-bit čítač pulsů – zdrojem jsou hodiny QCLK z kv. dekodéru
  - Čítá nahoru i dolů – směr určen signálem QDIR z kv. dekodéru
  - Různé módy čítání – v závislosti na potřebách aplikace
    - **Reset on Maximum Position** (čítá 0 → max, 0 → max atd.)
      - Hodnotu Max lze určit (musí být v rozsahu 32-bit)
      - Běžné chování čítače (přetéká/podtéká)
      - Použití tam, kde se čítají pulzy přes několik otáček. Poloha je vztažena k referenční poloze (v ní je čítač vynulován).
    - **Reset on Index Event** (čítač do 0 při výskytu Index značky)
      - Použití tam, kde nás zajímá pouze poloha v rámci jedné otáčky. Na konci každé otáčky je čítač automaticky vynulován. Referenční poloha je dána značkou Index.

        - a další módy...
    - Možnost generovat přerušení při:
        - přetečení/podtečení čítače
        - komparaci - aktuální hodnota čítače je porovnávána s komparační hodnotou v registru QPOSCNT
    - Záchytné (Latch) registry QPOSxLAT
        - Ukládají aktuální hodnotu čítače při definované události na vstupu Index (nulová značka - referenční poloha) – např. při vzestupné hraně
        - Může sloužit pro kontrolu počtu načítaných pulzů během jedné otáčky
            - Např. u IRC s 1024 pulzy / otáčku čítač načítá během jedné otáčky hodnotu 4x1024 = 4096

- **Edge Capture Unit**
    - Pro měření malých rychlostí (otáček).
    - Měří se perioda signálu, přesněji doba potřebná na jednotkovou změnu polohy:

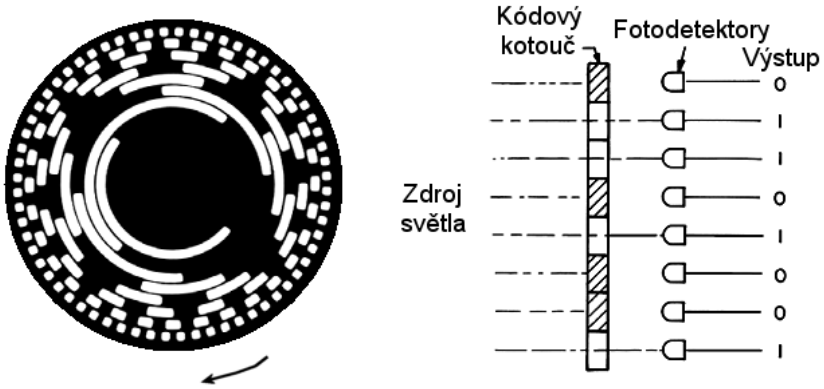
$$v(k) = \frac{X}{t(k) - t(k-1)} = \frac{X}{\Delta T}$$

        - v(k) – rychlost (nebo otáčky) v čase kT
        - X – jednotkový (pevný) interval polohy (dráha, kterou urazí snímač např. za jednu periodu signálu - např. 360°/1024 = 0,351°)
        - ΔT – změřená doba (závisí na rychlosti otáčení)
    - Princip: X se nastaví na pevnou hodnotu a ΔT se měří časovačem → pak se vypočte v(k).
    - Jednotková změna polohy X je definovaná jako celočíselný násobek hran hodin QCLK (hodiny QLCK jsou poděleny v předděličce).
    - Časovač QCTMR je taktován ze systémových hodin přes předděličku.
    - Každou událostí (odměření X) je obsah časovače zachycen do Capture Period Register (QCPRD) a časovač je vynulován. Tím je změřeno **ΔT**.

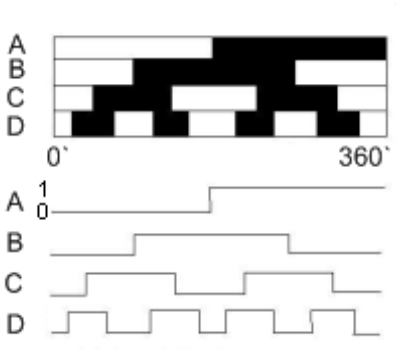


Obrázek 47 eQEP Edge Capture Unit

2.5 Čidlo absolutní polohy (natočení)



- Otvory v kotoučku = Grayův kód
  - Změna vždy pouze v jednom bitu (jedné stopě)
  - **Příklad 4-bit snímače (4 bity = 4 stopy)**
    - Rozlišení v úhlu: 360° / 2^4 = 22,5°



Desítkový kód	Úhel [°]	Binární kód	Grayův kód
0	0 - 22,5	0000	0000
1	22,5 - 45	0001	0001
2	45 - 67,5	0010	0011
3	.	0011	0010
4	.	0100	0110
5	.	0101	0111
6	.	0110	0101
7	.	0111	0100
8	.	1000	1100
9	.	1001	1101
10	.	1010	1111
11	.	1011	1110
12	.	1100	1010
13	.	1101	1011
14	.	1110	1001
15	337,5 - 360	1111	1000

  - Vlastnosti Grayova kódu
    - Změna vždy pouze v jednom bitu (jedné stopě)
    - Zabraňuje chybě při čtení

- U bin kódování se sousední čísla liší ve více než jednom bitu – ke změně nemusí dojít najednou – přečtená kombinace tak může být vzdálená o několik kroků od skutečné.

- Příklad kotoučku s 8 stopami = 8-bit → 256 poloh / otáčku



- Rozlišení běžně až 14-bit (16384 poloh / otáčku)
- Dražší oproti IRC
- Použití v aplikacích, kde je nutné znát natočení hřídele při zapnutí systému (oblast el. pohonů).

2.5.1 Elektrické rozhraní (konkrétní příklad - snímač LARM ARC 4xx)

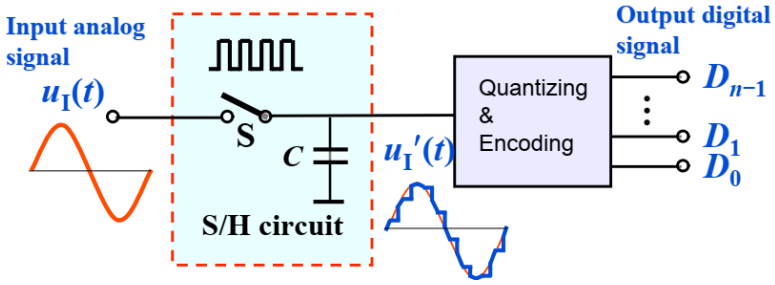
- Napájecí napětí
  - +5 V nebo 10 až 30 V
  - Spotřeba cca 1 až 2 W
- Datový výstup paralelní
- Provedení el. výstupů
  - Push-pull (pro U_{nap} = 10 až 30 V)
  - Linkový budič (pro U_{nap} = 5 V)
- Max. výstupní frekvence 200 kHz



3 A/D převodník

- Analogově/digitální převodník převádí analogové napětí na binární číslo.

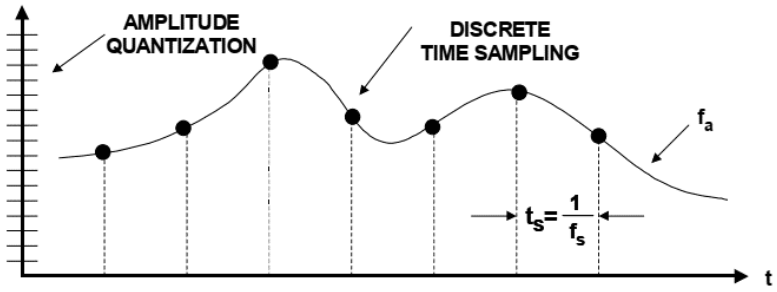
$$D = round((U_I / U_{ref}) \cdot 2^n) \tag{45}$$



Legenda:
n – počet bitů A/D převodníku
U_I – měřené napětí
U_{ref} – referenční napětí
U_I je v rozsahu od 0 do U_{ref}

Obrázek 48 Proces analogově-digitálního převodu

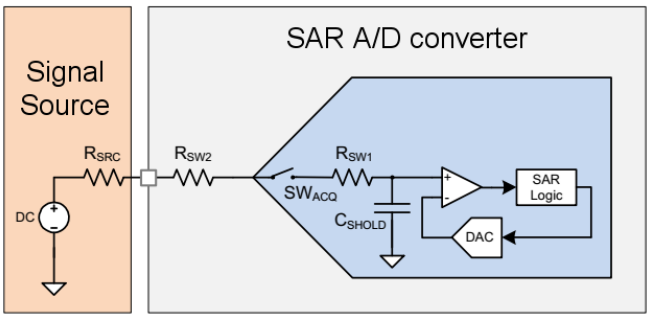
- Proces převodu je rozdělen na 2 části:
  - **Vzorkování analogového signálu** – odebrání vzorku napětí.
  - **Kvantování** – přiřazení binární číselné hodnoty měřenému napětí.



Obrázek 49 Vzorkování a kvantování

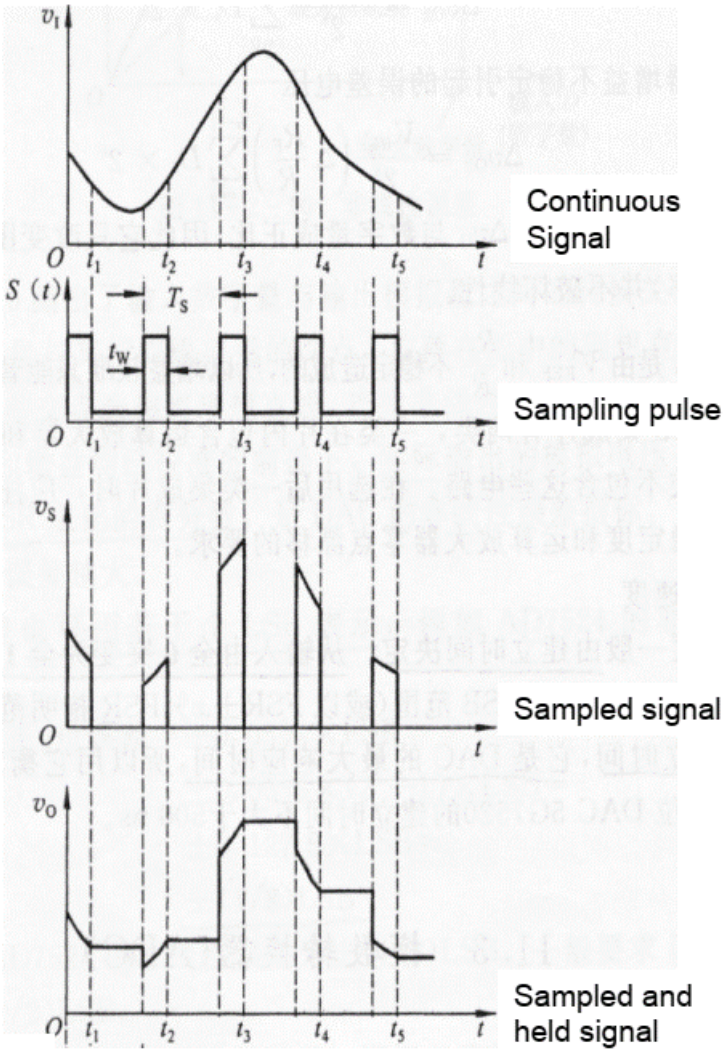
3.1 Vzorkování

- Odebrání vzorku napětí pomocí obvodu **Sample and Hold** (S/H), který se nachází uvnitř A/D převodníku.
- S/H obvod zajišťuje konstantní hodnotu napětí vzorku během A/D převodu.



Obrázek 50 Sample and Hold obvod na vstupu A/D převodníku

- **Sample and Hold** obvod v Obrázek 50 je tvořen:
    - Spínačem SWACQ (představuje vzorkovač) a odporem RSW1 (představuje odpor spínače).
    - Kondenzátorem C_{SHOLD}, který se nabije na napětí vstupního signálu a s ním pak pracuje vlastní A/D převodník, který následuje za S/H obvodem.

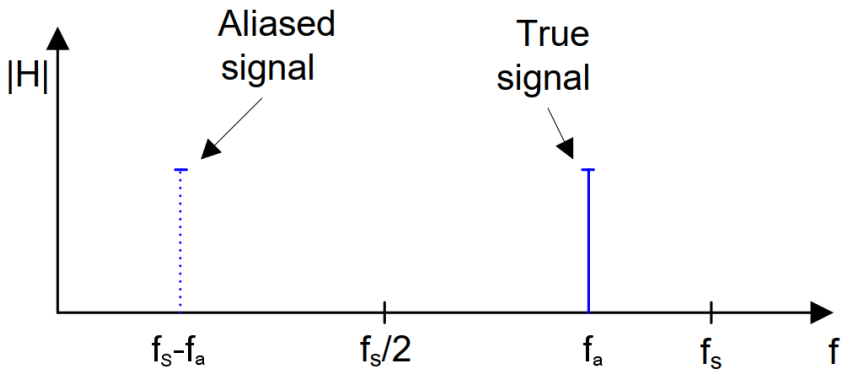


Obrázek 51 Vzorkování analogového signálu (napětí)

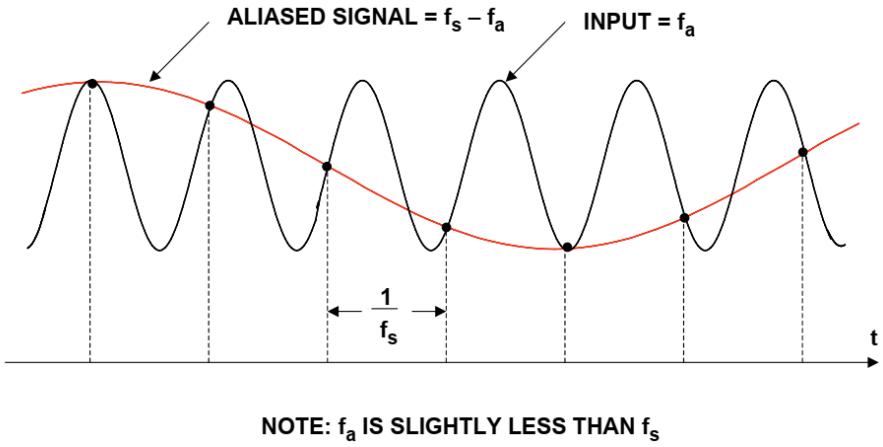
- Doba odběru vzorku t_w by měla být co nejkratší (nabití vstupní kapacity C_{SHOLD} co nejrychlejší).

3.1.1 Frekvence vzorkování F_s (sampling frequency)

- Při použití A/D převodníku obvykle pracujeme s konstantní vzorkovací frekvencí f_s (resp. periodou vzorkování T_s).
- F_s by měla být min. 2x větší než je nejvyšší frekvence obsažená v měřeném signálu (Nyquistův vzorkovací teorém).
- Pokud není, tak v důsledku podvzorkování vzniká tzv. **aliasing** → frekvence vyšší než $f_s/2$ se tváří jako frekvence mezi 0 až $f_s/2$ (Obrázek 52) a dostáváme zkreslený (nepravdivý) průběh signálu (Obrázek 53).



Obrázek 52 Aliasing ve frekvenčním spektru

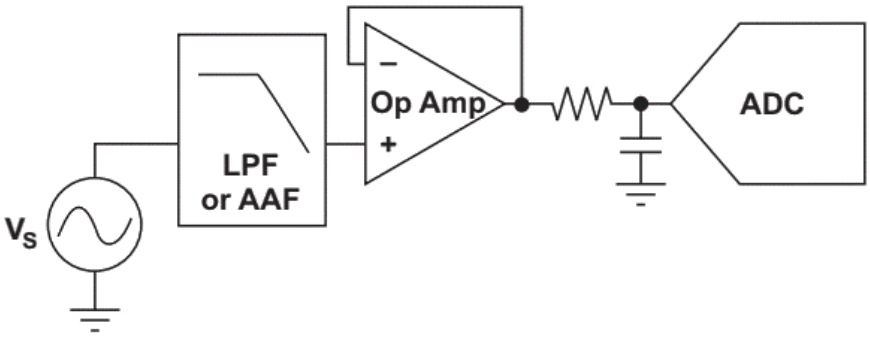


NOTE: f_a IS SLIGHTLY LESS THAN f_s

Obrázek 53 Aliasing vzniká kvůli podvzorkování signálu

f_a – frekvence vstupního analogového signálu
f_s – vzorkovací frekvence A/D převodu

- Aby nedocházelo k aliasingu, používáme na vstupu A/D převodníku anti-aliasingový filtr (**AAF – Anti Aliasing Filter**).

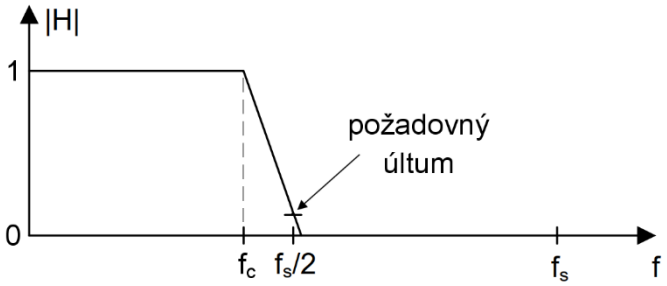


Obrázek 54 Antialiasignový filtr (AAF) na vstupu A/D převodníku (ADC)

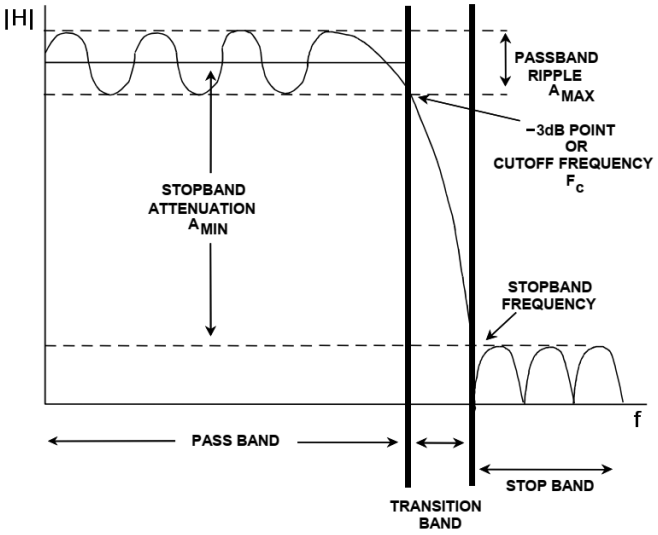
Struktura obvodu:

- V_S – představuje zdroj napěťového signálu, který vzorkujeme A/D převodníkem ADC.
- AAF – antialiasignový filtr, dolní propust (LPF = Low Pass Freq.).
- Napěťový sledovač (Op Amp) pro impedanční oddělení (má malý výstupní odpor, který je potřeba pro rychlé nabití kondenzátoru v obvodu Sample and Hold uvnitř A/D převodníku).
- ADC – A/D převodník.

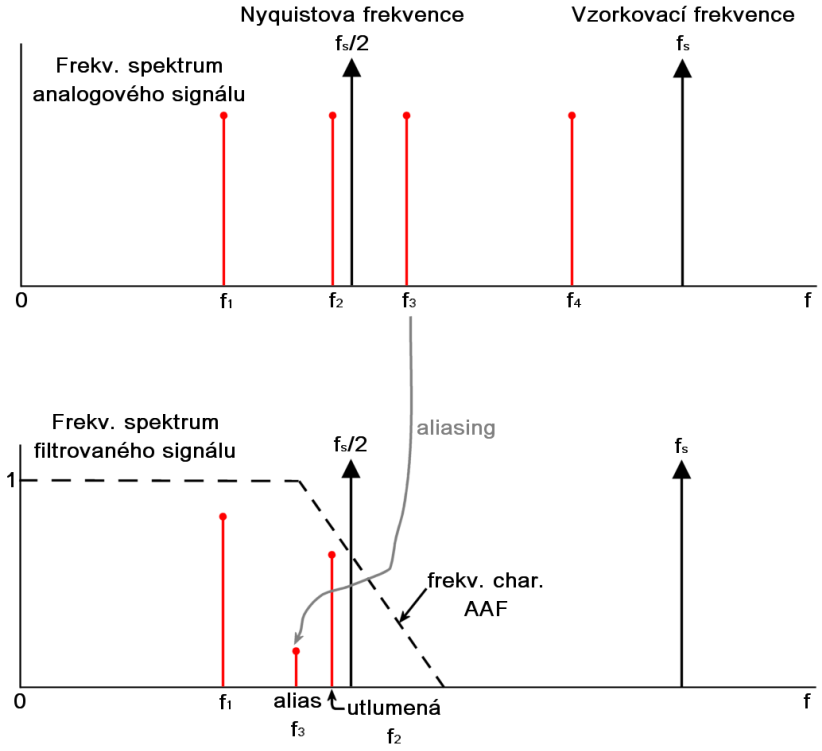
- **AAF je dolní propust**, která z analogového signálu odstraní (nebo spíše dostatečně utlumí) frekvenční složky nad frekvencí $f_s/2$.
  - Útlum složek nad $f_s/2$ je potřeba ideálně takový, aby byla úroveň frekvenčních složek signálu nad $f_s/2$ pod rozlišovací schopností A/D převodníku.
  - Dle požadovaného útlumu a strmosti útlumu (resp. šířky přechodového pásma) se stanoví řád filtru (obvykle 2. a vyššího řádu).
  - AAF je realizován pomocí operačního zesilovače nebo kaskády operačních zesilovačů, pokud je AAF vysokého řádu.



Obrázek 55 Požadovaná frekvenční amplitudová charakteristika AAF

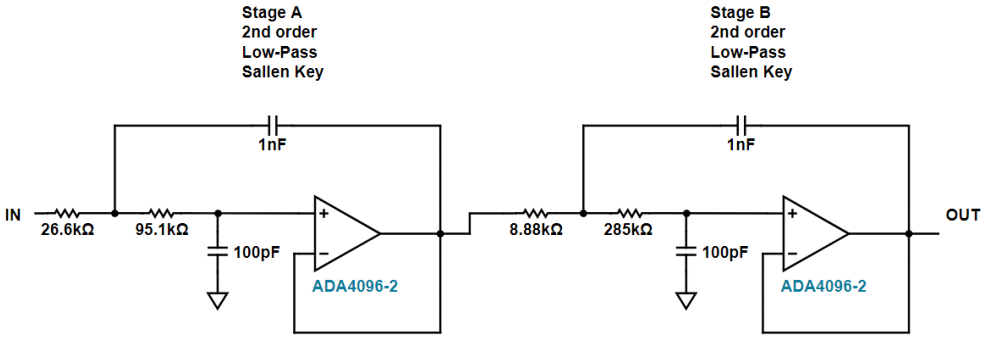


Obrázek 56 Frekvenční amplitudová charakteristika skutečného filtru typu dolní propust



Obrázek 57 Funkce AAF

Na Obrázek 57 je znázorněn vliv anti-aliasingového filtru (AAF). Vstupní signál, který má být vzorkován, obsahuje frekvenční složky f_1 až f_4 (horní graf). Frekvenční složky f_3 a f_4 je nutné zatlumit (odfiltrovat), jinak dojde k jejich ozrcadlení do pásma 0 až $f_s/2$. Po průchodu signálu filtrem AAF (spodní graf) je frekvenční složka f_4 odfiltrovaná úplně, frekv. složka f_3 je pouze částečně utlumená a bude přítomna v signálu, který bude A/D převodníkem vzorkován. Pokud je její velikost pod rozlišovací schopností A/D převodníku, tak to nevadí. Pokud není, tak je potřeba buď zvýšit vzorkovací frekvenci f_s, aby f_3 ležela pod $f_s/2$ nebo zvýšit útlum AAF.

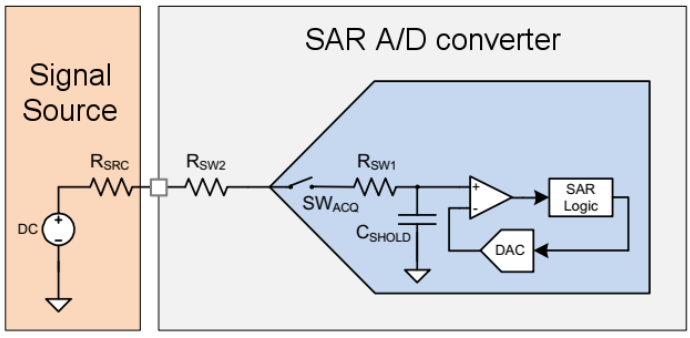


Obrázek 58 Praktická realizace Butterworth AAF 4. řádu (topologie Sallen Key)

Pro návrh AFF lze použít např. webovou aplikaci *Analog Filter Design* od Analog Devices: https://tools.analog.com/en/filterwizard/

3.1.2 Doba odebrání vzorku a velikost vnitřního odporu zdroje signálu

- **Acquisition Time** = doba potřebná na nabití kapacity C_{SHOLD} v Sample and Hold (S/H) obvodu



Obrázek 59 Sample and Hold obvod na vstupu A/D převodníku

- **Sample and Hold** obvod v Obrázek 59 je tvořen:
  - Spínačem SW_{ACQ} (představuje vzorkovač).
  - Odporem RSW1 (představuje odpor spínače).
  - Kondenzátorem CSHOLD, který se nabije na napětí vstupního signálu a s ním pak pracuje vlastní A/D převodník, který následuje za S/H obvodem.
- Rychlost nabíjení kondenzátoru C_{SHOLD}, je dána odpory v cestě signálu od jeho zdroje ke kondenzátoru C_{SHOLD} a velikostí kondenzátoru C_{SHOLD}.
- Výrobce A/D převodníku stanovuje minimální **Acquisition Time** pro bezchybný A/D převod, např.:

$$t_{ACQ} \geq 9 \cdot (R_{SRC} + R_{SW2} + R_{SW1}) \cdot C_{SHOLD} \quad (46)$$

Běžné hodnoty R a C u SAR ADC s max. vzorkovací frekvencí 1 MSps:

$R_{SW2} + R_{SW1} = 1$ kOhm

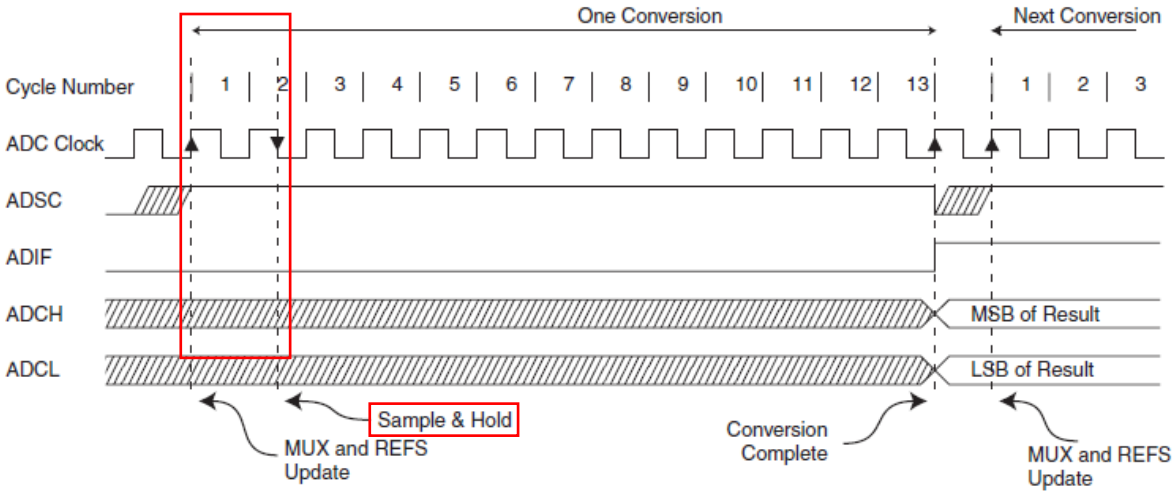
$C_{SHOLD} = 10$ pF

R_{SRC} – vnitřní odpor zdroje signálu

Např. pro R_{SRC} = 10 kOhm vychází min. hodnota Acquisition Time:

$$t_{ACQ} \geq 9 \cdot (10000 + 1000) \cdot 10^{-11} \geq 990ns \quad (47)$$

- Čím vyšší je vnitřní odpor zdroje signálu, tím delší musí být Acquisition Time a tím klesá max. použitelná vzorkovací frekvence.

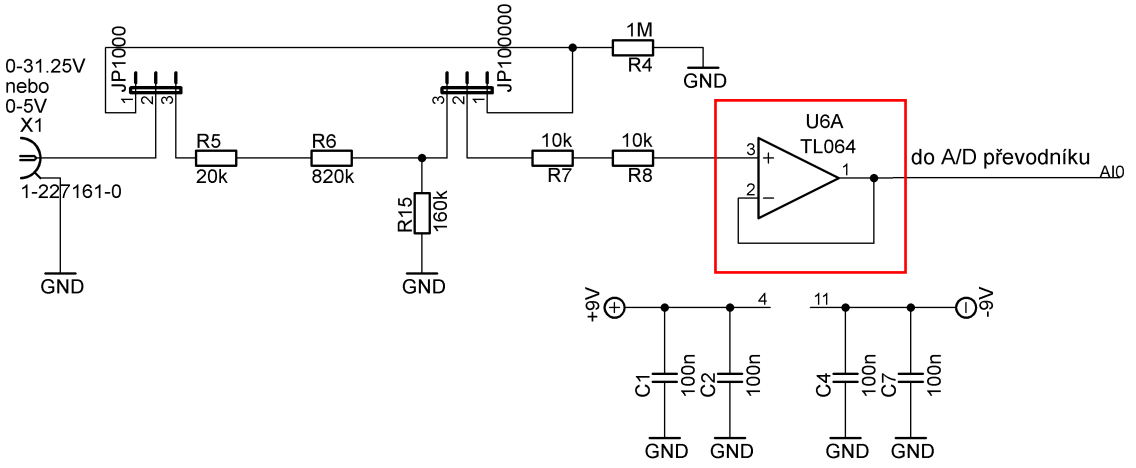


Obrázek 60 10-bit SAR ADC Timing Diagram, Single Conversion, MCU AVR

- Výrobce někdy přímo specifikuje maximální povolený vnitřní odpor zdroje signálu, např. v datasheetu k AVR ATmega32 je uvedeno:

The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedant sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

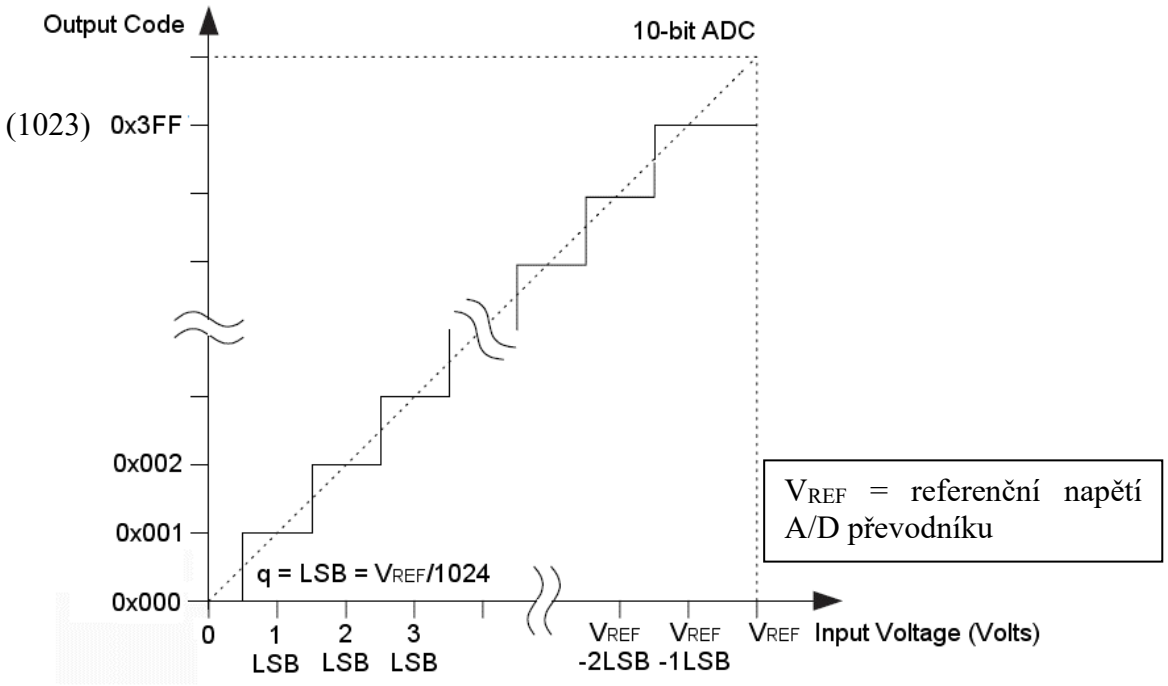
- Pokud je vnitřní odpor zdroje signálu příliš velký, použijeme **napěťový sledovač**, který impedančně oddělí zdroj signálu od vstupu A/D převodníku (Obrázek 61).



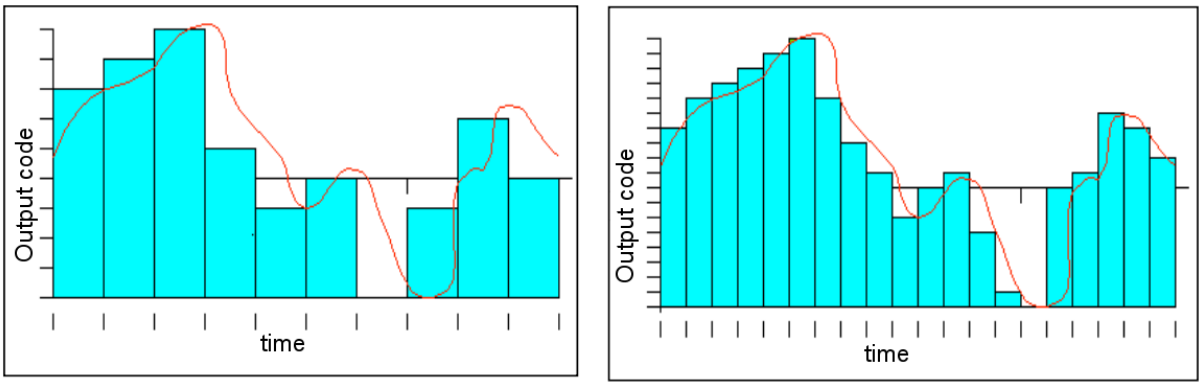
Obrázek 61 Oddělení zdroje signálu napěťovým sledovačem

3.2 Kvantování

- Vzorku napětí se přiřadí jedna z dostupných celočíselných hodnot.
- Počet těchto úrovní je dán počtem bitů (rozlišením) A/D převodníku, např.
  - 8-bit A/D převodník → 28 = 256 úrovní
  - 10-bit A/D převodník → 210 = 1024 úrovní
  - 16-bit A/D převodník → 2^{16} = 65536 úrovní



Obrázek 62 Ideální převodní charakteristika 10-bit A/D převodníku



Obrázek 63 Vliv rozlišení A/D převodníku na počet kvantizačních úrovní

- **Referenční napětí UREF určuje konverzní rozsah A/D převodníku.**
- Napětí přiváděné na analogový vstup se musí pohybovat v rozsahu 0 až UREF [V].
- Vždy se snažíme referenční napětí přizpůsobit max. napětí na analogovém vstupu pro dosažení maximální přesnosti měření (využití rozlišení A/D převodníku).
    - Pokud je např. max. U_{IN} do 2,5 V a U_{REF} = 5 V, tak není rozlišení A/D převodníku plně využito, 10-bit A/D převodník je degradován na 9-bit, protože využíváme pouze polovinu rozsahu referenčního napětí. Nastavení U_{REF} na 2,5 V docílíme využití plného rozlišení 10-bit, ale vzroste vliv šumu.
- **A/D převodník převádí vstupní analogové napětí na číslo podle vztahu:**

$$ADC = \frac{U_{IN} \cdot 2^N}{U_{REF}} \tag{48}$$

Kde: ADC – napětí převedené na celé číslo, rozsah 0 až $2^N - 1$ [-]

U_{IN} – napětí na analogovém vstupu A/D převodníku [V]

U_{REF} – referenční napětí A/D převodníku [V], např. 5 V

N – počet bitů A/D převodníku

- **Máme číslo z A/D převodníku a chceme určit napětí, jaké mu odpovídá:**

$$U_{IN} = U_{REF} \cdot \frac{ADC}{2^N} \tag{49}$$

- **Pro 10-bit A/D převodník (N=10):**

$$ADC = \frac{U_{IN} \cdot 2^{10}}{U_{REF}} = \frac{U_{IN} \cdot 1024}{U_{REF}} \tag{50}$$

$$U_{IN} = U_{REF} \cdot \frac{ADC}{1024} \tag{51}$$

Kde: ADC – napětí převedené na číslo, rozsah 0 až 1023 ($2^{10} - 1$) [-]

U_{IN} – napětí na analogovém vstupu A/D převodníku [V]

U_{REF} – referenční napětí A/D převodníku [V], např. 5 V

- N-bitový A/D převodník dokáže rozlišit 2^N napěťových úrovní.
- **Kvantizační krok** (hodnota **1 LSB = Least Significant Bit**) činí:

$$q = U_{LSB} = \frac{U_{REF}}{2^N} \tag{52}$$

Kde: q – kvantizační krok [V], je roven napěťové hodnotě 1 LSB

U_{REF} – referenční napětí A/D převodníku [V], např. 5 V

N – počet bitů A/D převodníku

- Např. pro U_{REF} = 5 V a 10-bit A/D převodník činí kvantizační krok:

$$q = U_{LSB} = \frac{5}{1024} = 4,8828 \text{ mV} \tag{53}$$

Tabulka 23 Vztah mezi vstupním napětím a výstupním kódem 10-bit A/D převodníku

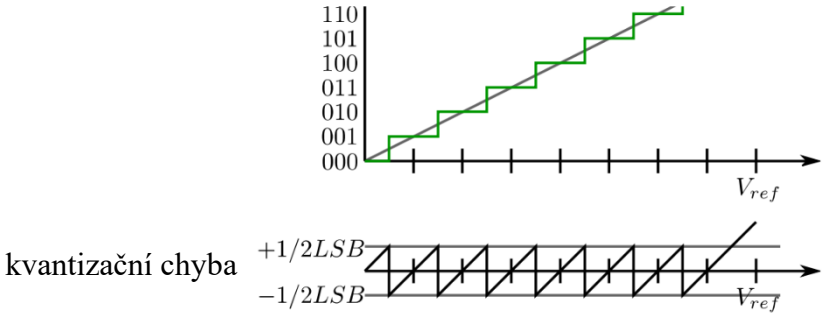
Napětí na analogovém vstupu (U_{IN})	**Číslo (ADC)**
0 V	0
1/1024 * U_{REF} = 4,88 mV	1
2/1024 * U_{REF} = 9,76 mV	2
.	
.	
512/1024 * U_{REF} = 2,5 V	512
.	
.	
1022/1024 * U_{REF} = 4,99 V	1022
1023/1024 * U_{REF} = 4,995 V	1023

Pozn.: U_{REF} = 5 V, 10-bit A/D převodník

- Krajní hodnoty:
  - ADC = 0 [-] odpovídá napětí na analogovém vstupu UIN = 0 [V]
  - ADC = 0x3FF (1023 dekadicky) [-] odpovídá napětí analogovém vstupu $U_{IN} = U_{REF}$ – 1 LSB [V]

3.3 Chyby A/D převodníků

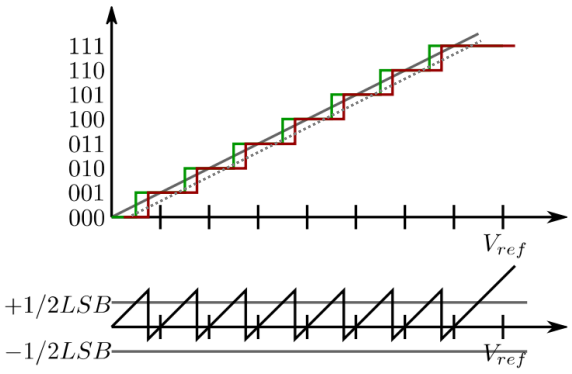
- Skutečné A/D převodníky vykazují chyby vůči ideálnímu N-bitovému A/D přev.



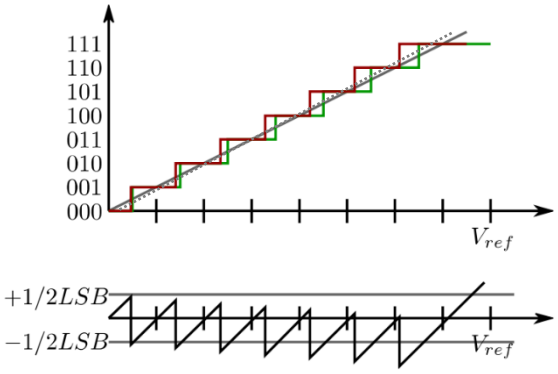
Obrázek 64 Převodní charakteristika ideálního 3-bitového A/D převodníku

- **Statické chyby** – způsobují odchylnost průběhu od ideální převodní charakteristiky
  - **Chyba zesílení** (Gain error) – chybný sklon převodní charakteristiky.
  - **Chyba nuly** (Offset error) – posun přev. char. v celém rozsahu napětí.

} lze odstranit kalibrací

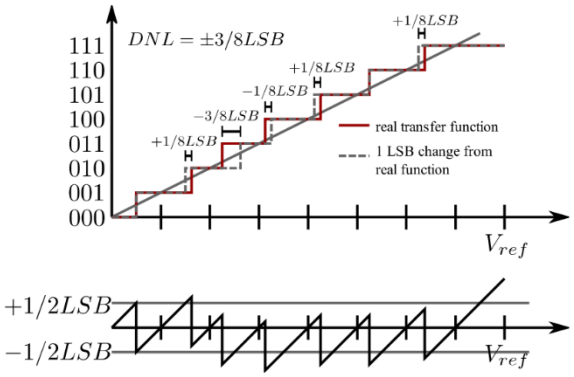


Obrázek 65 Chyba nuly

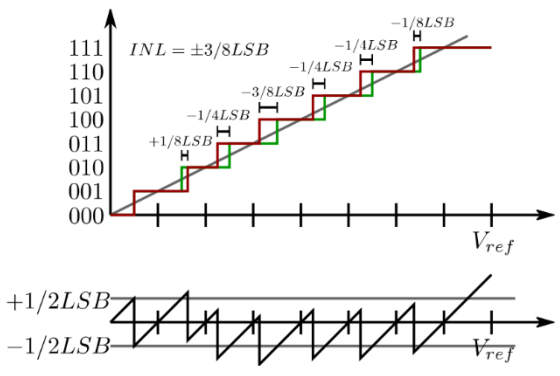


Obrázek 66 Chyba zesílení

  - **Chyba linearity**
    - **Diferenciální nelinearita** (DNL) - jak se liší šířka jednotlivých schodů skutečné charakteristiky od ideální. Ideálně je přírůstek napětí, který vyvolá změnu kódu o 1, konstantní v celém rozsahu napětí a roven 1 LSB. DNL nabývá hodnot od -1 do +1. Udává se max. hodnota a také může být ve formě grafu.
      - DNL kladné – kód má větší šířku než ideální
      - DNL záporné – kód má kratší šířku než ideální
      - DNL = -1 – kód chybí (neexistuje vstupní napětí pro daný kód)
    - **Integrální nelinearita** (INL) – vyjadřuje největší odchylku skutečné průměrné charakteristiky od ideální charakteristiky. Udává se max. hodnota a také může být ve formě grafu (postupná suma DNL). Pokud se hovoří o linearitě A/D převodníku, tak se tím míní INL.

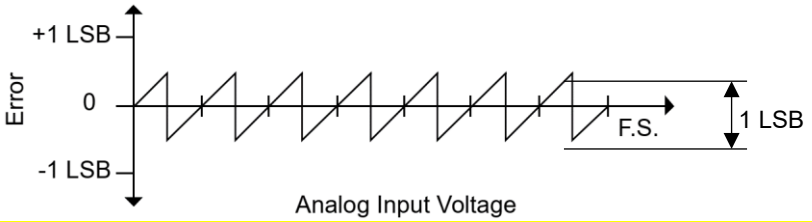


Obrázek 67 Diferenciální nelinearita (DNL)



Obrázek 68 Integrální nelinearita (INL)

- **Dynamické (AC) chyby** – projevují se při převodu rychle se měnícího napětí (frekvence řádu kHz nebo MHz)
  - Říkají kolik šumu a zkreslení bude obsahovat ovzorkovaný signál pro danou frekvenci vstupního signálu a frekvenci vzorkování.
  - Závisejí na frekvenci vstupního analogového signálu, na vzorkovací frekvenci, na napájecím napětí, na teplotě, případně na dalších parametrech.
  - **Ukazatele kvality A/D převodníku v AC oblasti:**
    - **SNR** – Signal to Noise Ratio (odstup signál – šum).
      - U ideálního A/D převodníku je šum v jeho digitálním výstupu způsoben kvantizační chybou. Maximální kvantizační chyba je u ideálního A/D převodníku rovna hodnotě ± ½ LSB.
      - Pokud na vstup A/D převodníku přivedeme lineárně rostoucí signál od 0 V do U_{ref}-1LSB V (Full Scale), bude mít průběh kvantizační chyby pilový tvar s Pk-Pk (špička-špička) hodnotou 1 LSB:



      - SNR ideálního ADC je dán velikostí kvantizačního šumu, tedy rozlišením A/D převodníku.

$$SNR[dB] = 20\log\left(\frac{rms\ SIGNAL}{rms\ NOISE}\right) = 20\log\left(\frac{\frac{2^{N-1}\cdot q}{\sqrt{2}}}{\frac{q}{\sqrt{12}}}\right) \tag{54}$$

$$\boxed{SNR = 6,02\cdot N + 1,76\,dB} \tag{55}$$

Kde: N – počet bitů A/D převodníku

q – velikost kvantizačního kroku (1 LSB) [V]

rms SIGNAL – efektivní hodnota vstupního signálu [V]

rms NOISE – efektivní hodnota kvanzitačního šumu [V]

Při určování SNR se předpokládá vstupní sinusový signál o amplitudě $2^{N-1} \cdot q$ [V], tedy polovině vstupního napěťového rozsahu A/D převodníku (½ Full Scale). Napěťový rozsah špička-špička tohoto signálu je roven plnému vstupnímu napěťovému rozsahu A/D převodníku (Full Scale). Efektivní (rms) hodnota sinusového průběhu je pak rovna jeho amplitudě dělené $\sqrt{2}$, tedy: $rms\,SIGNAL = \frac{2^{N-1} \cdot q}{\sqrt{2}}$.

Efektivní hodnota kvanzitačního šumu, který má pilovitý průběh, se vypočte jako podíl jeho amplitudy dělené $\sqrt{3}$, tedy:

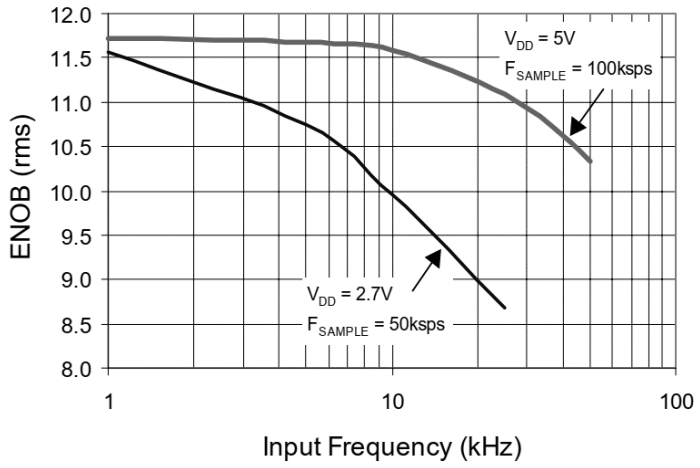
$$rms\,NOISE = \frac{\frac{1}{2}LSB}{\sqrt{3}} = \frac{LSB}{\sqrt{12}} = \frac{q}{\sqrt{12}}.$$

Pomůcka:
6,02 dB 2x
20 dB 10x
40 dB 100x
60 dB 1000x
80 dB 10000x
100 dB 100000x

Rozlišení A/D převodníku [bit]	Počet kvantizačních úrovní	SNR [dB]
8	256	49,92
10	1 024	61,96
12	4 096	74,00
14	16 384	86,04
16	65 536	98,08
24	16 277 216	146,24

- **SINAD** – Signal to Noise Ratio + Distorsion.
  - Zahrnuje SNR + zkreslení + úroveň harmonických složek způsobených A/D převodníkem.
  - Významný ukazatel kvality A/D převodníku v AC oblasti.
- **Počet efektivních bitů** (ENOB – Effective Number of Bits)
  - Rozlišení A/D převodníku klesá s rostoucí frekvencí. Vlivem dynamických chyb se může např. z 12-bit A/D převodníku stát převodník pouze 10-bitový.
  - Vypočte se porovnáním SINAD vůči SNR ideálního A/D převodníku.

$$ENOB = \frac{SINAD - 1,76\,dB}{6,02} \tag{56}$$



Obrázek 69 Efektivní počet bitů 12-bit A/D převodníku v závislosti na frekv. vstupního signálu

- SNR a SINAD se určují ze změřeného spektra ovzorkovaného sign. pomocí FFT.
- Chyby A/D převodníku jsou hezky vysvětleny např. v dokumentu „*Microchip AN693 - Understanding A-D Converter Performance Specifications.PDF*“ (dostupný online).

3.3.1 Kdy jsou jaké chyby důležité

- **Statické chyby** – převažují v případě, když se měřené napětí mění jen velmi pomalu. Např.:
  - Měření hmotnosti – elektronická váha.
  - Statická tenzometrická měření – měření síly, zatížení.
  - Měření teploty – teplota se mění pomalu.
- **Dynamické chyby** – nabývají na významu při rychle se měnícím vstupním signálu.
  - A/D převodník v osciloskopu – velký frekvenční rozsah vstupního signálu.
  - Vibrační stolice pro únavové testy – z ovzorkovaného průběhu signálu se počítá frekvenční analýza – zjišťuje se frekvenční spektrum signálu (obsah a velikost frekvenčních složek v signálu).

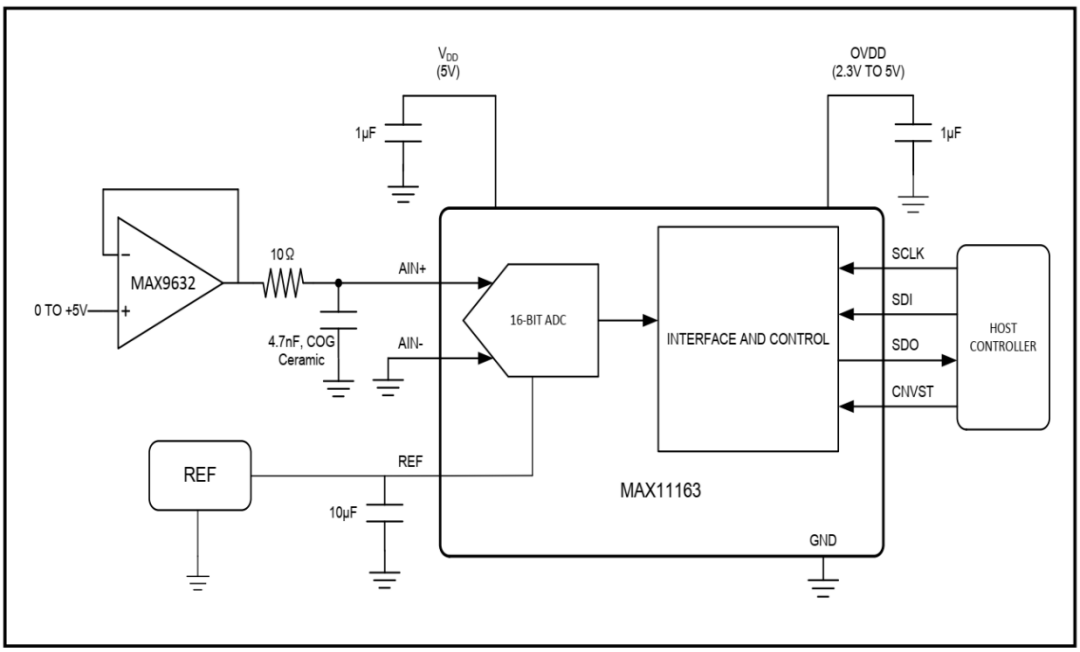
3.4 Parametry reálného A/D převodníku

- Např. jednokanálový 16-bit A/D převodník s postupnou aproximací (SAR) MAX11163.

Benefits and Features

- High DC/AC Accuracy Improves Measurement Quality
  - 16-Bit Resolution with No Missing Codes
  - 250ksps Throughput Rates Without Pipeline Delay/ Latency
  - 93.2dB SNR and -105dB THD at 10kHz
  - 0.38 LSB_{RMS} Transition Noise
  - ±0.2 LSB DNL (typ) and ±0.5 LSB INL (typ)
- Wide Supply Range and Low Power Simplify Power-Supply Design
  - +5V Analog Supply
  - +2.3V to +5V Digital Supply
  - 19mW Power Consumption at 250ksps
  - 10µA in Shutdown Mode
- Multi-Industry Standard Serial Interface and Small Package Reduces Size
  - SPI/QSPI™/MICROWIRE®/DSP-Compatible Serial Interface
  - 3mm x 5mm Tiny 10-Pin µMAX Package

Typical Operating Circuit



Electrical Characteristics

(V_{DD} = 4.75V to 5.25V, V_{OVDD} = 2.3V to 5.25V, f_{SAMPLE} = 250ksps, V_{REF} = 5V; T_A = T_{MIN} to T_{MAX}, unless otherwise noted. Typical values are at T_A = +25°C.) (Note 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
ANALOG INPUT (Note 3)						
Input Voltage Range		AIN+ to AIN-	0		V_{REF}	V
Absolute Input Voltage Range		AIN+ to GND	-0.1		V_{REF} + 0.1	V
		AIN- to GND	-0.1		+0.1	
Input Leakage Current		Acquisition phase	-10	+0.001	+10	µA
Input Capacitance				40		pF
Input-Clamp Protection Current		Both inputs	-20		+20	mA
STATIC PERFOMANCE (Note 4)						
Resolution	N		16			Bits
No Missing Codes			16			Bits
Offset Error			-3.5	±0.5	+3.5	LSB
Offset Error Temperature Coefficient				±0.006		LSB/°C
Gain Error			-7.5	±2	+7.5	LSB
Gain Error Temperature Coefficient				±0.04		LSB/°C
Integral Nonlinearity	INL		-1.2	±0.5	+1.2	LSB
Differential Nonlinearity	DNL		-0.5	±0.2	+0.5	LSB
Positive Full-Scale Error			-7.25		+7.25	LSB
Analog Input CMR	CMR	Referred to the output		-2.2		LSB/V
Power-Supply Rejection (Note 5)	PSR	PSR vs. V_{DD}, referred to the output		-4.64		LSB/V
Transition Noise				0.38		LSB_{RMS}

statické chyby

Note 4: Static performance limits are guaranteed by design and device characterization. For definitions, see the *Definitions* section.

Electrical Characteristics (continued)

(V_{DD} = 4.75V to 5.25V, V_{OVDD} = 2.3V to 5.25V, f_{SAMPLE} = 250ksps, V_{REF} = 5V; T_A = T_{MIN} to T_{MAX}, unless otherwise noted. Typical values are at T_A = +25°C.) (Note 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
EXTERNAL REFERENCE						
REF Voltage Input Range	V_{REF}		2.5		V_{DD}	V
REF Input Capacitance				20		pF
REF Load Current				78.5		µA
DYNAMIC PERFOMANCE (Note 6)						
Signal-to-Noise Ratio	SNR		92.1	93.2		dB
		V_{REF} = 2.5V		87.6		dB
Signal-to-Noise Plus Distortion	SINAD		91.6	93.0		dB
Spurious-Free Dynamic Range	SFDR		99.2	107		dB
Total Harmonic Distortion	THD			-105	-98	dB
Intermodulation Distortion (Note 7)	IMD			-119.4		dBFS
SAMPLING DYNAMICS						
Throughput Sample Rate			0		250	ksps
Transient Response		Full-scale step			400	ns
Full-Power Bandwidth		-3dB point		6		MHz
		-0.1dB point		> 0.2		
Aperture Delay				2.5		ns
Aperture Jitter				50		ps_{RMS}

dynamické chyby

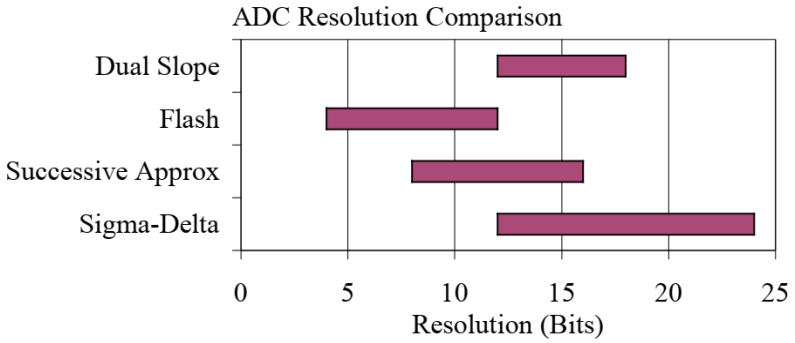
Note 6: 10kHz sine wave input, -0.1dB below full scale.

Efektivní počet bitů, který odpovídá parametru SINAD u výše uvedeného 16-bit ADC, je:

$$ENOB = \frac{SINAD - 1,76\,dB}{6,02} = \frac{93 - 1,76\,dB}{6,02} = 15,15\,bit \qquad (57)$$

3.5 Základní typy A/D převodníků

- Integrační A/D převodník (dual slope ADC)
- Paralelní (Flash) A/D převodník
- A/D převodník s postupnou aproximací (SAR)
- Sigma-Delta A/D převodník



Obrázek 70 Porovnání rozlišení různých typů A/D převodníků

Tabulka 24 Provnání různých typů A/D převodníku dle rychlosti vzorkování a ceny

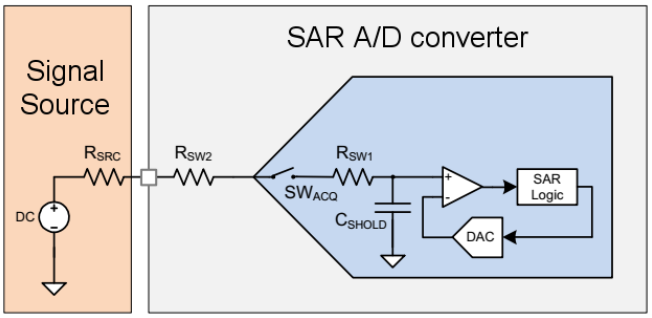
Typ	Frekvence vzorkování	Cena
Integrační	malá	střední
Flash	velmi vysoká	vysoká
S postupnou aproximací	střední až vysoká	nízká
Sigma-Delta	nízká	nízká

Tabulka 25 Oblasti použití A/D převodníků různých typů

Typ	Použití	Poznámka
Integrační	voltmetry a multimetry	Frekvence vzorkování 10 Hz. Měří průměrnou hodnotu napětí. Odolnost proti rušivému napětí síťové frekvence a jejích násobků.
Flash	digitální osciloskopy, vysoká frekvence vzorkování až GHz	Nejrychlejší A/D převodníky. Celý převod se vykoná v jednom taktu.
S postupnou aproximací	měřící karty, MCU, rychlé voltmetry, pomalejší digitální osciloskopy	Vyžadují obvod Sample and Hold na vstupu, nejsou odolné vůči sériovému rušení.
Sigma-Delta	přesné číslicové voltmetry, přesné digitální teploměry	Malá rychlost převodu. Vysoké rozlišení. Pro měření pomalu se měnících signálů (např. napětí z termočlánku).

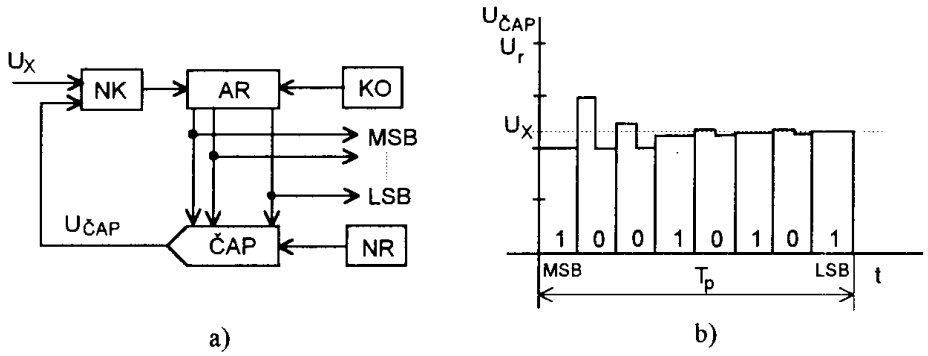
3.5.1 A/D převodníky s postupnou aproximací (SAR)

- Velmi často používaný typ A/D převodníku.
- Jedná se o obvyklý typ interního A/D převodníku u MCU.
- Vyrábějí se 8 až 16-bitové.
- Doba převodu jednotky □s i méně (dle typu) → 1-ky kSps až 10-ky MSps.
- Vyžadují konstantní vstupní napětí během převodu → na jejich vstup se umísťuje vzorkovač s pamětí (tzv. **Sample and Hold** obvod).



Obrázek 71 Sample and Hold obvod na vstupu A/D převodníku

- Vnitřní struktura tohoto typu A/D převodníku je na Obrázek 72.

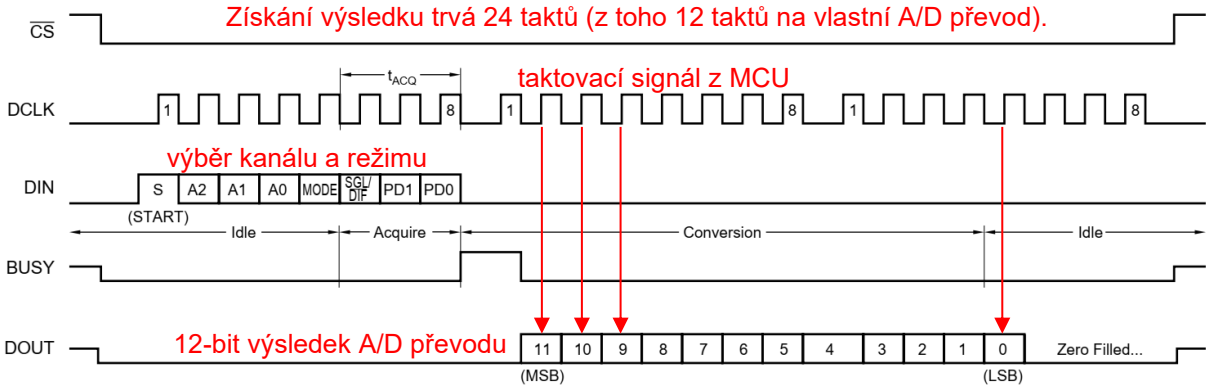


Obrázek 72 Blokové schéma A/D převodníku s postupnou aproximací (a) a průběh vyvažování tohoto převodníku (b)
zdroj: [Haasz V. *Elektrická měření a metody*]

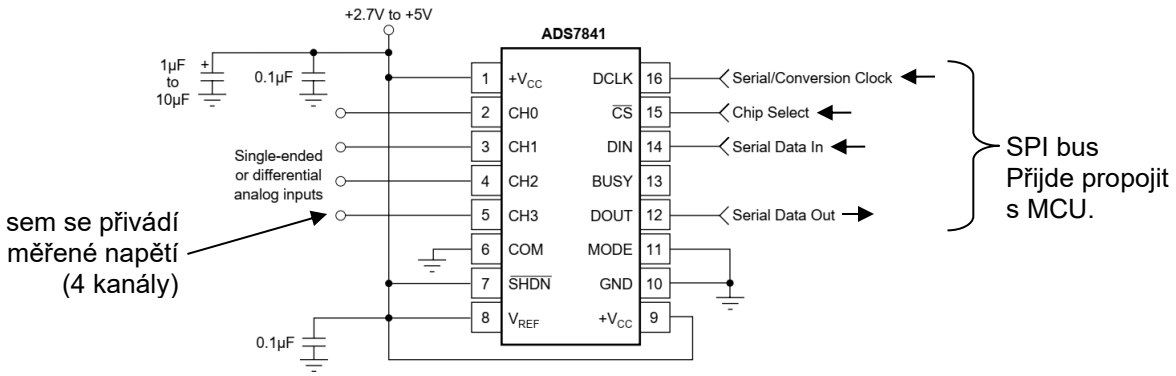
- Popis funkce (zdroj: [Haasz V. *Elektrická měření a metody*]):
  - Vstupní analogové napětí Ux se srovnává se zpětnovazebním kompenzačním napětím UČAP na výstupu D/A převodníku (ČAP), které se mění tak dlouho, dokud rozdíl mezi oběma napětími není menší než rozlišovací schopnost A/D převodníku. Pak je odpovídající číslo na vstupu D/A převodníku výstupem A/D převodníku.
  - Převod probíhá v n taktech (n je počet bitů A/D převodníku).
  - V prvém taktu je určena hodnota nejvýznamnějšího bitu (MSB), odpovídající signál je vyslán do aproximačního registru AR (ostatní výstupy AR jsou v logické 0).

- Pomocí D/A převodníku je číslo převedeno na napětí UČAP = Ur/2 (Ur je referenční napětí A/D převodníku) a porovnáno pomocí napěťového komparátoru NK s Ux. Je-li UČAP < Ux (jako v Obrázek 72), ponechá se MSB = 1, v opačném případě by se nastavilo MSB = 0.
- V dalším taktu je testován bit MSB-1 pomocí jemu odpovídajícího napětí UČAP = Ur/4, které se přičte k napětí na výstupu D/A převodníku z předchozího taktu. Výsledné napětí je opět srovnáno s Ux. Je-li UČAP > Ux, je zkoušený bit nastaven na 0, jinak se ponechá v logické 1. Stejná procedura se opakuje pro zbývající bity D/A převodníku.
- Počet taktů převodu je tedy roven počtu bitů D/A převodníku. Ve skutečnosti je potřeba jeden tak navíc pro počáteční vynulování AR. Celková doba převodu je v Obrázek 72 označena T_p.

Na Obrázek 73 je znázorněn průběh jednoho A/D převodu u externího 12-bit A/D převodníku ADS7841 s postupnou aproximací (SAR) a sériovým komunikačním rozhraním SPI. U externích SAR A/D převodníků se hodiny DCLK rozhraní SPI používají nejen pro komunikaci s A/D převodníkem, ale současně také pro taktování vlastního A/D převodu.



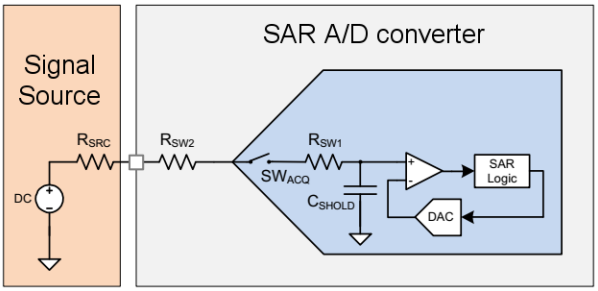
Obrázek 73 Taktování A/D převodu u externího 12-bit SAR A/D převodníku ADS7841 s SPI rozhraním



Obrázek 74 Základní el. zapojení externího 12-bit SAR A/D převodníku s SPI rozhraním

4 Interní A/D převodník v AVR

- Většina AVR obsahuje jeden 10-ti bitový A/D převodník s postupnou aproximací (anglicky tzv. **SAR ADC** = **S**uccessive **A**pproximation **R**egister **A**nalog to **D**igital **C**onverter).

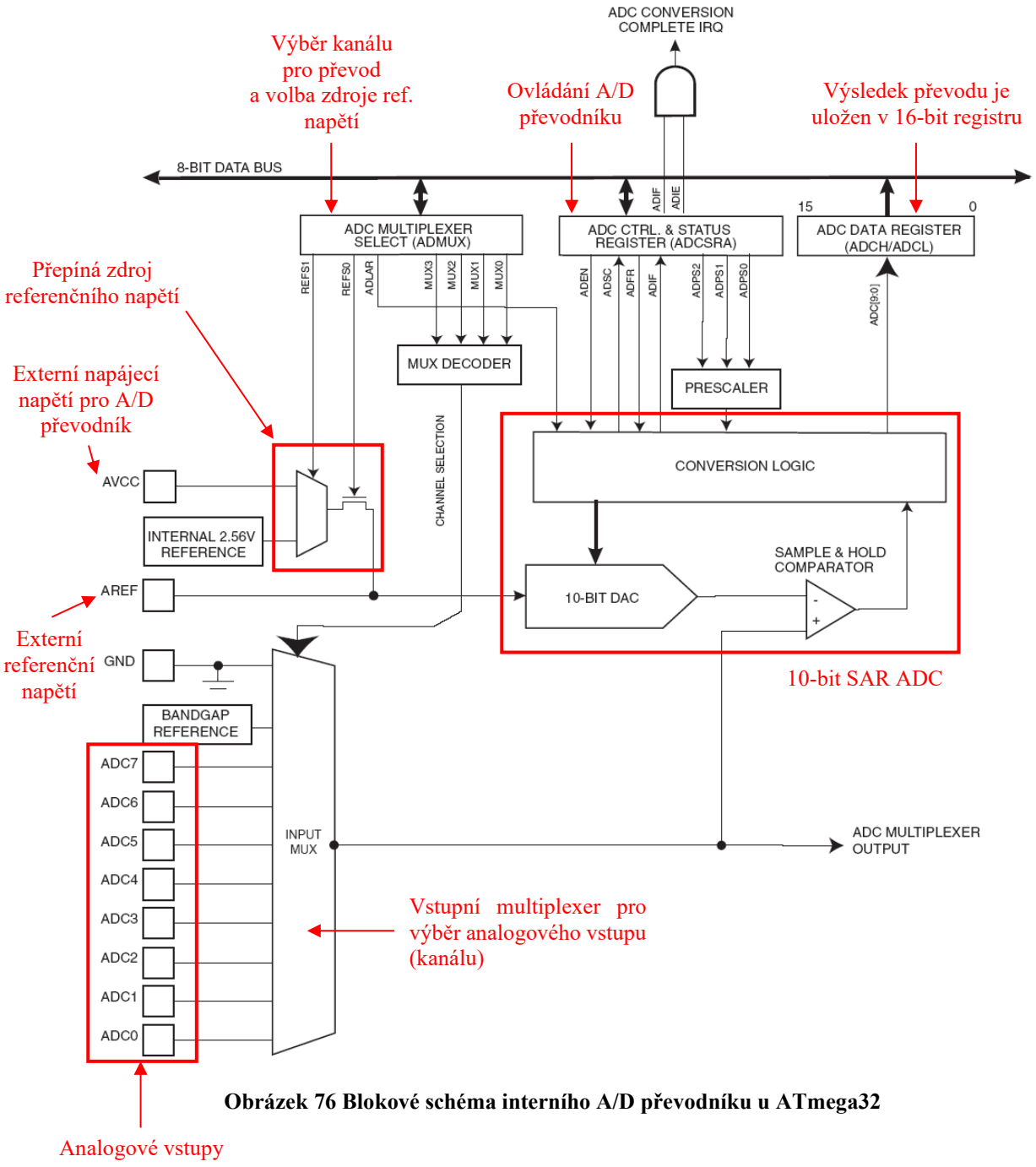


Obrázek 75 Připojení signálu k A/D převodníku

- Počet analogových kanálů závisí na typu AVR (0 až 16).
  - Kanály jsou multiplexovány, protože A/D převodník je pouze jeden.
  - V jeden okamžik lze zpracovávat pouze jeden kanál.
  - Nevýhoda - při vzorkování více kanálů jsou vzorky vůči sobě časově posunuty (to může a nemusí vadit).
- Parametry interního A/D převodníku v AVR:
  - A/D převodník s postupnou aproximací.
  - Absolutní přesnost 1,5 LSB při ADC_{clock} = 200 kHz a U_{REF} = 4 V (zahrnující všechny statické chyby - kvantizační šum, INL, DNL, Gain, Offset Error).
    - LSB = Least Significant Bit = nejméně význačný bit.
  - Frekveční rozsah vstupního signálu 4 kHz.
  - Zdroj referenčního napětí – možnost volby
    - Externí v rozsahu 0 až VCC.
    - Interní napěťová reference 2,56 V.
  - Doba převodu 13 μs – 260 μs (dle nastaveného taktovacího kmitočtu A/D převodníku), přičemž při maximální přesnosti (10 bit) je převodník schopen vzorkovat s **frekvencí max. 15 ksps** (doba převodu 67 μs) **na jednom kanálu**.
  - S počtem převáděných kanálů klesá frekvence vzorkování, např.
    - 1 kanál 15 kHz
    - 2 kanály 7,5 kHz
    - 3 kanály 5 kHz
    - 4 kanály 3,75 kHz
    - atd…
  - Volitelná předdělička taktovacího kmitočtu.

- Pro dosažení maximální přesnosti se musí kmitočet pohybovat v rozmezí 50 kHz až 200 kHz.
- Doba převodu závisí na velikosti taktovacího kmitočtu.
- Je možné použít i vyšší kmitočty, ale sníží se tím rozlišovací schopnost pod 10 bit.

- Některé typy AVR mají diferenciální analogové vstupy s volitelným zesílením analogového signálu před vstupem do A/D převodníku – lze použít pro zesílení malých napětí.



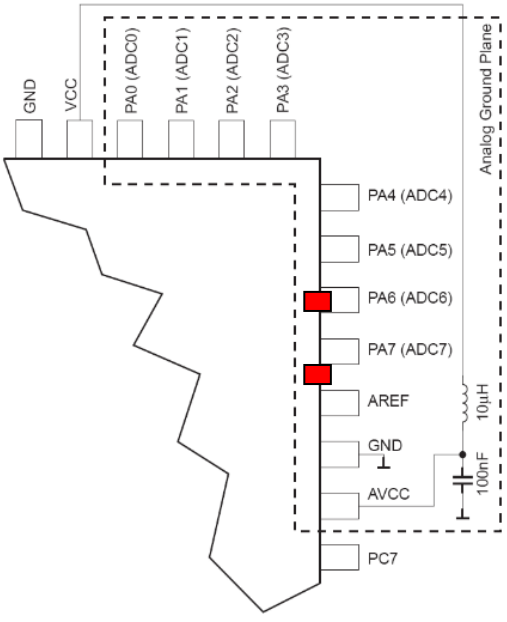
Obrázek 76 Blokové schéma interního A/D převodníku u ATmega32

4.1 Volba referenčního napětí obecně

- Referenční napětí UREF určuje konverzní rozsah A/D převodníku.
- Napětí přiváděné na analogový vstup se musí pohybovat v rozsahu 0 až UREF [V].
- Vždy se snažíme referenční napětí přizpůsobit max. napětí na analogovém vstupu pro dosažení maximální přesnosti měření (využití rozlišení A/D převodníku).
    - Pokud je např. max. U_{IN} do 2,5 V a U_{REF} = 5 V, tak není rozlišení A/D převodníku plně využito, 10-bit A/D převodník je degradován na 9-bit, protože využíváme pouze polovinu rozsahu referenčního napětí. Nastavení U_{REF} na 2,5 V docílíme využití plného rozlišení 10-bit.

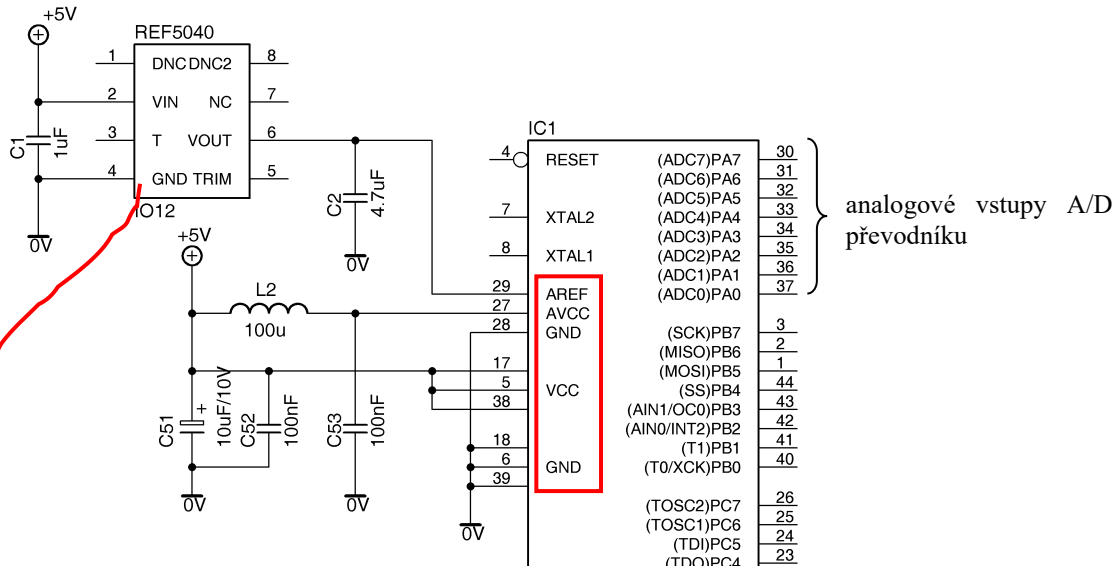
4.2 Referenční napětí A/D převodníku na AVR

- Zdroj referenčního napětí u MCU AVR lze volit, jsou tři možnosti:
    - U_{REF} = AVCC (jako zdroj UREF se použije napájecí napětí MCU)
    - U_{REF} = 2,56 V (interní napěťová reference 2,56 V uvnitř MCU)
    - Externí U_{REF} - napětí přivedené na pin AREF z vnějšku.
- Je-li u AVR přivedeno externí napětí na pin AREF, nesmí se zvolit žádná jiná možnost než externí reference, jinak dojde ke zkratu externí reference s interní referencí, resp. s AVCC (viz interní zapojení A/D převodníku na Obrázek 76).
- Volba referenčního napětí se provede pomocí bitů **REFS0** a **REF1** v registru **ADMUX**.
- **Pro dosažení maximální přesnosti A/D převodu je nutné zabezpečit čistotu a stálost napájecího napětí A/D převodníku (vstup AVCC) a externího referenčního napětí (vstup AREF), pokud je externí napěťová reference použita**.
    - Na Obrázek 77 je znázorněn LC filtr u pinu AVCC pro odfiltrování digitálního šumu (doporučené zapojení z datasheetu AVR).



Obrázek 77 LC filtr v napájení A/D převodníku

- Na Obrázek 78 je znázorněno připojení kvalitní externí napěťové reference 4,096 V ke vstupu AREF.



Obrázek 78 Připojení externí napěťové reference k ATmega32

TEXAS INSTRUMENTS

www.ti.com

REF5010, REF5020
REF5025, REF5030
REF5040, REF5045, REF5050

SBOS410E – JUNE 2007 – REVISED JUNE 2010

Low-Noise, Very Low Drift, Precision VOLTAGE REFERENCE

Check for Samples: REF5010, REF5020, REF5025, REF5030, REF5040, REF5045, REF5050

FEATURES

- LOW TEMPERATURE DRIFT:
  - High-Grade: 3ppm/°C (max)
  - Standard-Grade: 8ppm/°C (max)
- HIGH ACCURACY:
  - High-Grade: 0.05% (max)
  - Standard-Grade: 0.1% (max)
- LOW NOISE: 3μV_{PP}/V
- EXCELLENT LONG-TERM STABILITY:
  - 5ppm/1000hr (typ) after 1000 hours
- HIGH OUTPUT CURRENT: ±10mA
- TEMPERATURE RANGE: –40°C to +125°C

APPLICATIONS

- 16-BIT DATA ACQUISITION SYSTEMS
- ATE EQUIPMENT
- INDUSTRIAL PROCESS CONTROL
- MEDICAL INSTRUMENTATION
- OPTICAL CONTROL SYSTEMS
- PRECISION INSTRUMENTATION

DESCRIPTION

The REF50xx is a family of low-noise, low-drift, very high precision voltage references. These references are capable of both sinking and sourcing, and are very robust with regard to line and load changes.

Excellent temperature drift (3ppm/°C) and high accuracy (0.05%) are achieved using proprietary design techniques. These features, combined with very low noise, make the REF50xx family ideal for use in high-precision data acquisition systems.

Each reference voltage is available in both standard- and high-grade versions. They are offered in MSOP-8 and SO-8 packages, and are specified from –40°C to +125°C.

REF50xx Family

MODEL	OUTPUT VOLTAGE
REF5020	2.048V
REF5025	2.5V
REF5030	3.0V
REF5040	4.096V
REF5045	4.5V
REF5050	5.0V
REF5010	10.0V

Obrázek 79 Parametry přesné a stabilní napěťové reference REF50xx

4.3 Práce s interním A/D převodníkem v AVR ATmega32

- Pro ovládání interního A/D převodníku jsou vyhrazeny tyto registry:
  - **ADMUX** – volba napěťové reference, volba kanálu pro převod, volba zarovnání výsledku převodu
  - **ADCSRA** – řídící a příznakový registr – pro nastavení typu převodu, povolení/zákaz přerušení po dokončeném převodu, nastavení děličky taktovací frekvence, spuštění převodu apod.
  - **SFIOR** - volba spouštěcího signálu převodu v autotrigger módu

- **Ovládání A/D převodníku lze shrnout do těchto kroků**:
  - Povolení A/D převodníku.
  - Nastavení typu převodu (jednorázový, kontinuální).
  - Výběr zdroje napěťové reference.
  - Výběr analogového kanálu.
  - Spuštění převodu.
  - Čekání na dokončení převodu.
  - Vyčtení výsledku převodu.

  → Provádí se periodicky

- Obvykle se A/D převod opakuje periodicky s konstantní periodou vzorkování – zajistíme časovačem.

4.3.1 Povolení funkce A/D převodníku

- Převodník se povoluje bitem **ADEN** (0 – zakázán, 1 – povolen) v řídícím registru **ADCSRA (ADC Control and Status Register)**.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

4.3.2 Výběr zdroje napěťové reference

- Napěťová reference se vybírá pomocí bitů **REFS0** a **REFS1** v registru **ADMUX (ADC Multiplexer Selection Register)**

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **REFS1** a **REFS0** – výběr napěťové reference

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

4.3.3 Výběr zarovnání výsledku

- **10-bitový výsledek převodu** je uložen v 16-bit registru ADC (tvořený dvěma 8-bit registry **ADCL** a **ADCH**) a může být zarovnán buď doleva nebo doprava.
- Zarovnání výsledku doleva se používá tehdy, **pokud nám stačí 8-bit výsledek**, stačí pak číst **pouze registr ADCH**.
- Zarovnání výsledku se nastavuje bitem **ADLAR** v registru **ADMUX**.
- Defaultně je po resetu nastaveno zarovnání doprava.

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADLAR = 0 — Zarovnání doprava

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR = 1 — Zarovnání doleva

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

Obrázek 80 Výsledek převodu uložený v registrech ADCL a ADCH

4.3.4 Nastavení taktovacího kmitočtu A/D převodu

- Pro využití maximálního rozlišení 10-bit se doporučuje max. taktovací frekvence A/D převodníku do 200 kHz.
- Pokud se spokojíme jen s 8-bit rozlišením, tak lze použít i vyšší frekvenci.
- Taktovací frekvence A/D převodníku je odvozena z frekvence CPU přes děličku:

$$F_{ADCLK} = \frac{F_{CPU}}{N} \tag{58}$$

- **Výběr děličky N se provádí bity ADPS2 až ADPS0 v registru ADCSRA**

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **ADPS2** až **ADPS0** – nastavení děličky taktovacího kmitočtu pro ADC

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Tabulka 26 Taktovací frekvence A/D převodníku pro různé dělící poměry N

dělička N	F_{ADCLK} při F_{CPU} = 14,7456 MHz	doba převodu	
2	7 372,8 kHz	1,76 µs	8-bit a menší rozlišení
4	3 686,4 kHz	3,57 µs	
8	1 843,2 kHz	7,05 µs	
16	921,6 kHz	14,10 µs	
32	460,8 kHz	28,21 µs	
64	230,4 kHz	56,42 µs	
128	115,2 kHz	112,84 µs	10-bit rozlišení

doba převodu = 13 / F_{ADCLK}

4.3.5 Výběr analogového kanálu pro převod

- Kanály jsou číslovány od nuly.
- **Volba kanálu** se provede bity **MUX3** až **MUX0** v registru **ADMUX**.
- U verzí AVR s diferenciálními vstupy je ještě přítomen bit **MUX4**, počet možností je tak rozšířen (výběr samostatného kanálu či diferenciálního a určení zesílení).

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Typ převodu**
  - **Single Ended** - měří se napětí na jednom vybraném kanálu vůči GND
    - Výsledek převodu je:

$$ADC = \frac{U_{IN} \cdot 2^{10}}{U_{REF}} = \frac{U_{IN} \cdot 1024}{U_{REF}} \tag{59}$$

Kde: ADC – napětí převedené na číslo, rozsah 0 až 1023 ($2^{10} - 1$) [-]
U_{IN} – napětí na analogovém vstupu A/D převodníku [V]
U_{REF} – referenční napětí A/D převodníku [V], např. 5 V

  - **Differential** – měří se rozdílové napětí mezi dvěma kanály
    - Výsledek převodu je:

$$ADC = \frac{(U_{POS} - U_{NEG}) \cdot GAIN \cdot 1024}{U_{REF}} \tag{60}$$

    - a je reprezentován ve dvojkovém doplňku v rozsahu **od -512** (0x200) **do +511** (0x1FF)

Kde: ADC – napětí převedené na číslo, rozsah 0 až 1023 ($2^{10} - 1$) [-]
U_{POS} – napětí na „kladném“ analogovém vstupu A/D převodníku [V]
U_{NEG} – napětí na „záporném“ analogovém vstupu A/D převodníku [V]
GAIN – zesílení
U_{REF} – referenční napětí A/D převodníku [V], např. 5 V

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010(1)		ADC0	ADC0	200x
01011(1)		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110(1)		ADC2	ADC2	200x
01111(1)		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

Note: 1. The differential input channels are not tested for devices in PDIP Package. This feature is only guaranteed to work for devices in TQFP and QFN/MLF Packages

Obrázek 81 Výběr analogového kanálu pro převod

4.3.5.1 Změna kanálu

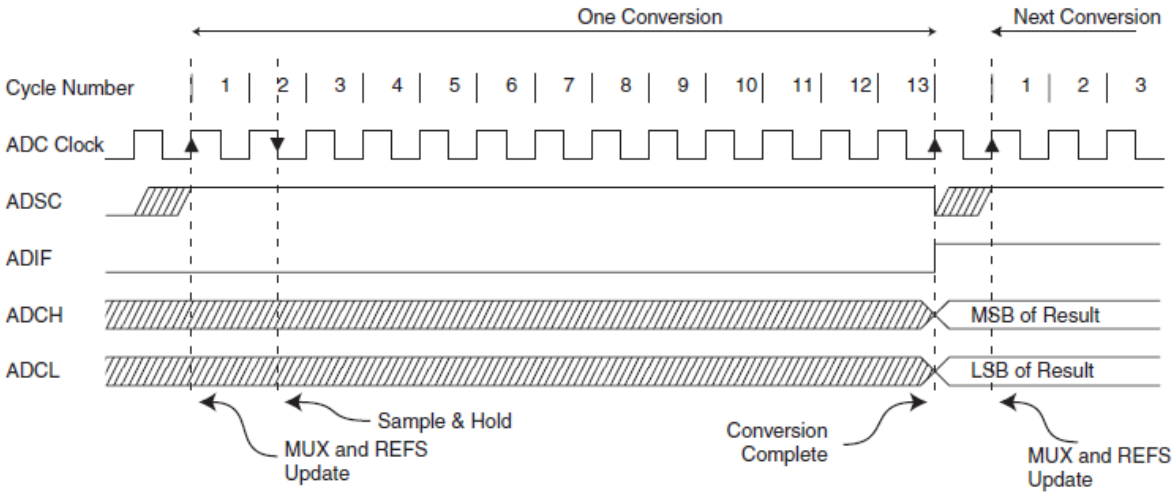
- Obecné zásady
    - Kanál je možné změnit během převodu, ale změna se projeví až po skončení převodu.
    - Kanál lze změnit nejdříve jeden takt po spuštění převodu, nedodržením této prodlevy by mohlo dojít k chybě.
- V případě jednorázového převodu
    - Počkat na dokončení převodu, nastavit kanál, spustit převod.
- V případě kontinuálního převodu
    - Počkat na dokončení převodu, pak změnit kanál, je vhodné využít přerušení při dokončeném převodu.
    - Nutno si ovšem uvědomit, že automatický převod se spustí dříve, než proběhne změna kanálu, takže **výsledek následujícího převodu bude odpovídat předchozímu kanálu**.
- V případě automatického spouštění (Auto-Trigger)
    - Kanál změnit během převodu, např. v obsluze přerušení od dané periferie nebo…
    - Až po dokončení převodu, např. v jeho přerušení.
    - Nové nastavení kanálu se projeví až v následujícím převodu.

4.3.6 Zahájení A/D převodu, módy A/D převodu

- Převodník může pracovat ve třech módech
  - jednorázový převod
  - kontinuální převádění (volně běžící)
  - převod spouštěný od nějaké události (např. přetečení časovače, externí přerušení apod.) – mód je dostupný pouze u některých AVR
- **Jednorázový převod (Single Conversion)**
  - Provede pouze jeden převod, převod se spouští „ručně“.
  - **A/D převod se spustí zapsáním 1 do bitu ADSC** v registru **ADSCRA**.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Bit **ADSC** zůstává v 1 po celou dobu převodu, po dokončení převodu ho hardware automaticky vynuluje. Tím jsme informování o dokončení převodu.
  - O skončení převodu jsme také informování příznakem **ADIF** (ADC Interrupt Flag) – po skončení převodu bude nastaven do 1. Příznak je pak nutné ručně smazat zápisem jedničky, pokud nevyužíváme přerušení.
  - **Převod trvá 13 taktů**, úplně první převod trvá 25 taktů.
  - Např. F_{CPU} = 16 MHz, $F_{ADCLK} = F_{CPU} / 128$ = 125 kHz → **doba převodu** = 13/125 000 = **104 µs → 9,615 kSps**



Obrázek 82 ADC Timing Diagram, Single Conversion

- **Kontinuální převádění (Free Running Mode)**
  - V tomto módu se neustále dokola spouští převod a aktualizuje se **ADC Data Register** s výsledkem převodu.
    - Po dokončení předchozího převodu se automaticky spustí další.
    - Výsledek převodu zůstává v 16-bit registru ADC.
  - Převod musí být spuštěn „ručně“ pouze poprvé (zápis 1 do bitu **ADSC** v registru **ADCSRA**).
  - **Tento mód použijeme zejména tehdy, pokud potřebujeme dosáhnout vysoké vzorkovací frekvence**.
    - Používá se obvykle v kombinaci s obsluhou přerušení při dokončeném převodu, abychom byli schopni rychle sbírat výsledky.
    - V ISR je třeba vyčíst výsledek převodu a uložit ho do proměnné pro další zpracování mimo ISR.
    - Mám-li převádět více kanálů, je ISR dobré místo pro změnu kanálu (např. postupně vzorkujeme 8 kanálů a vzorky postupně ukládáme do pole).
    - Zpracování vzorků se provede jinde, mimo ISR, protože obsluha ISR musí kratší než je perioda vzorkování.
  - Free Running Mode je u ATmega32 součástí tzv. Auto Triggering módu.
  - **Free Running Mode se u ATmega32 zapíná zápisem 1** do bitu **ADATE** v registru **ADCSRA**, před tím je nutné v registru **SFIOR** pomocí bitů **ADTS0:2** nastavit zdroj spouštění převodu na „Free Running mode“ (bity ADTS0:2 vynulovat).

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 83 Zapnutí Auto Trigger módu (bit ADATE = 1)

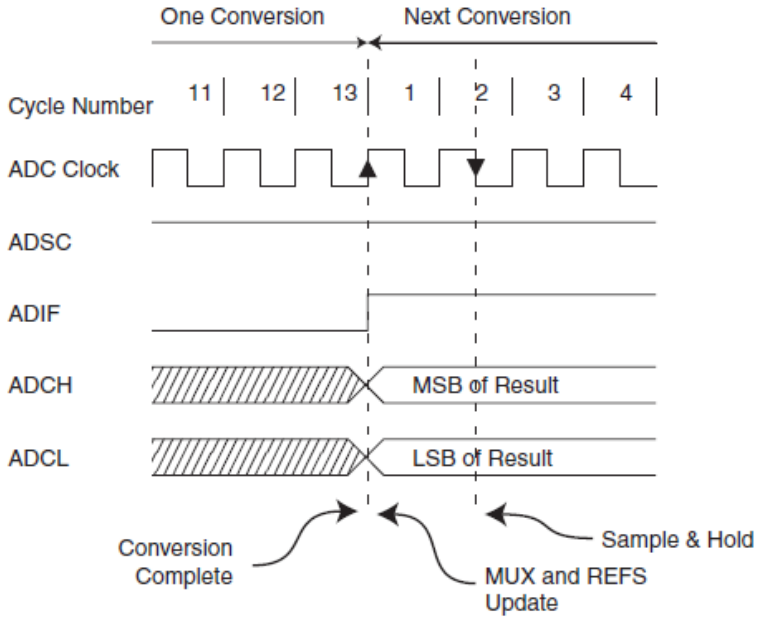
Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 84 Výběr zdroje spouštění A/D převodu (bity ADTS0:2)

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

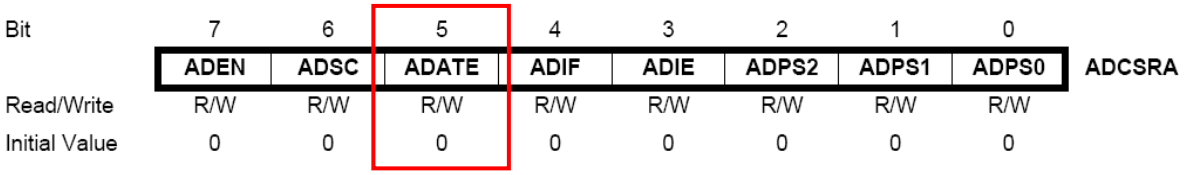
Obrázek 85 Zdroje spuštění A/D převodu v Auto Trigger módu

- **Převod trvá 13 taktů, úplně první převod trvá 25 taktů.**



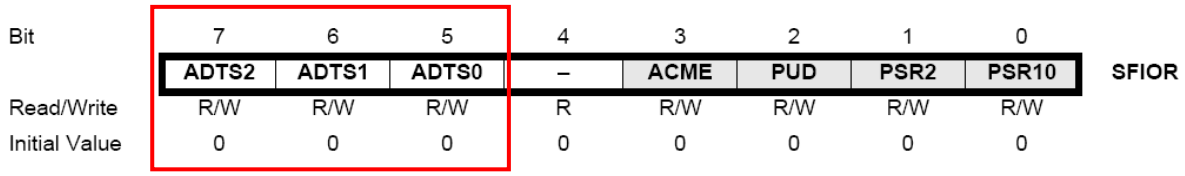
Obrázek 86 ADC Timing Diagram, Free Running Conversion

- **Automaticky spouštěný převod pomocí události (Auto Triggering)**
  - Dostupný jen u některých AVR, např. od ATmega16 a výše a u některých ATtiny.
  - Tento mód se zapíná zápisem 1 do bitu **ADATE** v registru **ADCSRA** a výběr zdroje spouštěcího pulsu se provádí bity **ADTS0:2** v registru **SFIOR**.



Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 87 Zapnutí Auto Trigger módu (bit ADATE = 1)



Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

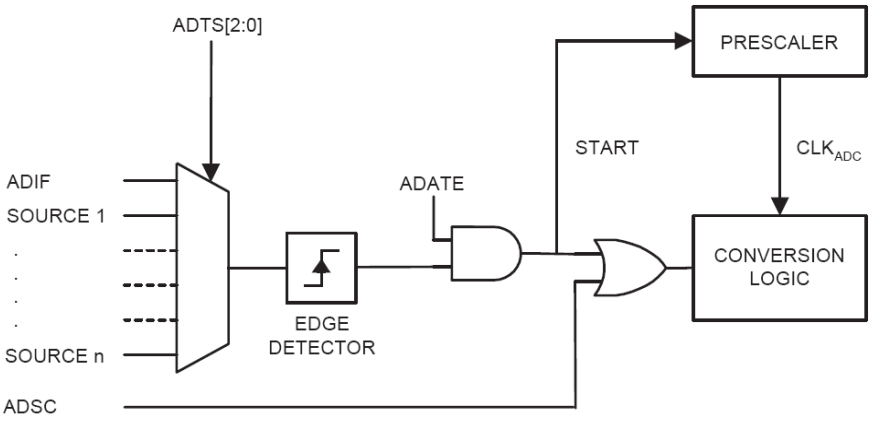
Obrázek 88 Výběr zdroje spouštění A/D převodu (bity ADTS0:2)

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Obrázek 89 Zdroje spuštění A/D převodu v Auto Trigger módu

  - Převod je spuštěn při náběžné hraně spouštěcího signálu (tzn. změna z log. 0 na log. 1).
  - **Např. při přetečení časovače (Timer/Counter Overflow) dojde k nastavení příznaku přetečení a spustí se automaticky A/D převod**.
    - Přerušení od dané periferie nemusí být povoleno a nemusí být povoleno ani globální přerušení. Vždy dojde k nastavení příznaku přerušení a spuštění A/D převodu. Nevyužívám-li přerušení, je nutné příznak u oné periferie nulovat ručně, jinak by příště nedošlo ke spuštění převodu (A/D převod se spouští vzestupnou hranou spouštěcího signálu).
    - Periodickým triggerováním od časovače lze zajistit periodické vzorkování se stabilní vzorkovací frekvencí.
  - Automatické kontinuální spouštění převodu lze zařídit tak, že se nastaví jako zdroj spouštěcího signálu vlastní příznak přerušení A/D převodníku „převod dokončen“ (bit **ADIF** v registru **ADCSRA**). Bity **ADTS0:2 = 0**. V tomto

případě není nutné příznak **ADIF** mazat po dokončeném převodu. Tento režim již byl vysvětlen v předchozím odstavci.



Obrázek 90 Logika automatického spouštění A/D převodu (mód Auto Trigger)

4.3.7 Výsledek A/D převodu

Kde je uložen výsledek A/D převodu?

- Po dokončení A/D převodu je výsledek převodu uložen v 16-bit registru **ADC**, který je tvořen dvěma 8-bit registry **ADCL** (dolní byte výsledku) a **ADCH** (horní byte výsledku).
- Výsledek může být v registru ADC zarovnán doleva nebo doprava, podle nastavení bitu **ADLAR** v registru **ADCSRA**.

ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

Obrázek 91 16-bit registr ADC s výsledkem A/D převodu

- Při čtení výsledku z 16-bit registru **ADC** je třeba dodržet postup:
  - Čte se nejdříve **ADCL** a pak **ADCH** aby bylo zaručeno, že horní bajt náleží k dolnímu (případná změna údaje v registru **ADCH** je blokována jakmile přistoupím k **ADCL**, pak přečtu **ADCH** a teprve pak dojde k odblokování registru **ADCH** pro zápis nového výsledku).
  - V jazyku C toto není třeba řešit, lze číst celý 16-bit registr najednou (registr je dostupný pod názvem **ADCW**), správný přístup si zajistí kompilátor.

Jak zjistit, že byl převod dokončen?

- O dokončeném převodu nás informuje příznak **ADIF** (ADC Interrupt Flag) v registru **ADCSRA**.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 92 Příznak dokončeného A/D převodu v registru ADCSRA

- Po dokončeném A/D převodu je v 16-bit datovém registru ADC uložen výsledek převodu, který mohu vyčíst.
- Mazání příznaku ADIF:
  - Pokud se nepoužívá přerušení „převod dokončen“, je nutné po dokončení převodu příznak **ADIF** „ručně“ mazat (zápisem 1 do bitu ADIF).

- Pokud se používá přerušení „převod dokončen“, je bit ADIF automaticky smazán po vstupu do obsluhy přerušení a není třeba ho „ručně“ mazat.

Přerušení „převod dokončen“

- A/D převodník disponuje jedním zdrojem přerušení – „převod dokončen“.
- Přerušení použijeme zejména tehdy, pokud potřebujeme vzorkovat s co nejvyšší frekvencí. V přerušení vyčteme výsledek, uložíme si ho např. do pole a zpracujeme mimo přerušení. Mezi tím již běží další A/D převod.
- Přerušení se povoluje/zakazuje bitem **ADIE** v registru **ADCSRA**.
- Žádost o přerušení se vyvolá, jakmile je příznak dokončení převodu **ADIF** v registru **ADCSRA** nastaven do 1 (za předpokladu, že je přerušení „převod dokončen“ povoleno a globálně jsou přerušení povolena).
- Příznak **ADIF** se maže automaticky po vstupu do ISR.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obrázek 93 Bity související s přerušením „převod dokončen“

- **ADIF** – příznak ukončeného převodu, je-li zapnuto přerušení, vygeneruje se
- **ADIE** – povoluje/zakazuje přerušení „převod dokončen“

Omezovač šumu (Noise Canceler)

- Pro snížení hladiny šumu během A/D převodu a tím zvýšení přesnosti výsledku.
- Princip spočívá v uspání CPU a některých dalších periferií během A/D převodu, čímž se sníží hladina produkovaného šumu a zvýší se tím přesnost A/D převodu.
- Lze provádět pouze **u jednorázového převodu** a musí být **povoleno přerušení** „A/D převod dokončen“.
- Postup:
  - Nastavit jednorázový převod a povolit přerušení „A/D převod dokončen“.
  - Povolit A/D převodník (bit ADEN = 1 v registru ADCSRA).
  - Nastavit **Sleep mód “ADC Noise Reduction”** pomocí bitů **SM0** až **SM2** v registru **MCUCR**. A/D převod se automaticky spustí jakmile CPU usne (zastaví se).
  - Po dokončení A/D převodu se vygeneruje IRQ, které „vzbudí“ CPU, následuje odskok do ISR. CPU může v tomto módu vzbudit nejen přerušení od A/D převodníku, ale i od dalších periferií (viz datasheet k AVR - režimy se sníženou spotřebou).

4.4 Registry pro obsluhu A/D převodníku v AVR

- Platí pro ATmega32
- **Registr ADMUX (ADC Multiplexer Selection Register)**

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **REFS1** a **REFS0** – výběr napěťové reference

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

  - **ADLAR** – zarovnání výsledku doprava (0) nebo doleva (1)
  - **MUX4** až **MUX0** – výběr kanálu pro převod a případně i zesílení při použití diferenciálních kanálů

- **Registr ADCSRA (ADC Control and Status Register)**

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **ADEN** – povolení A/D převodníku (0 – zakázán, 1 – povolen)
  - **ADSC** – zahájení A/D převodu. Bit je v jedničce tak dlouho, dokud není převod u konce, po dokončení převodu bude nulový.
  - **ADATE** – zapíná automaticky spouštěný převod na událost (tzv. autotriggering), volba zdroje spouštěcího signálu se volí bity **ADTS** v registru **SFIOR**
  - **ADIF** – příznak dokončeného převodu, je-li povoleno přerušení, vygeneruje se
  - **ADIE** – povoluje přerušení „převod dokončen“
  - **ADPS2** až **ADPS0** – nastavení děličky taktovacího kmitočtu pro ADC

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

- **Registr SFIOR (Special Function IO Register)**

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **ADTS2** až **ADTS0** – volba spouštěcího signálu převodu v autotrigger módu

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

  - **Bit 4** – reservováno, při zápisu do SFIOR musí být nastaven do nuly

4.5 Obsluha A/D převodníku v jazyku C

4.5.1 Jednorázový převod

- **Použijeme polling - spustíme převod a počkáme na jeho dokončení**
  - Je vhodné si vytvořit funkci pro obsluhu A/D převodníku.
  - Funkce vrací výsledek A/D převodu a jako parametr přebírá číslo kanálu.
- **Pseudokód hl. programu:**

```
Nastav A/D převodník (jednorázový převod, dělička, zdroj ref. napětí)
Povol A/D převodník

WHILE (nekonečná smyčka)
       Přečti hodnotu na vybraném kanálu
       Převeď hodnotu na napětí v milivoltech
END WHILE
```

- **Pseudokód funkce pro spuštění převodu a čtení výsledku:**

```
Nastav číslo kanálu
Spusť převod
Čekej, dokud není převod hotový (testováním příznaku dokončeného převodu ve smyčce)
Smaž příznak dokončeného převodu
Vyčti výsledek
```

- **Kód (pro překladač AVR-GCC):**

```
#include <avr/io.h>

#define ADC_VREF_TYPE 0x00   // volba napetove reference interniho A/D prevodniku
                             // 0x00 - externi napetova reference na pinu AREF
                             // 0x01 - AVCC jako napetova reference
                             // 0x03 - interni napetova reference 2,56 V

#define ADC_VREF_MV   5000U  // hodnota referencniho napeti A/D prevodniku [mV]

// Deklarace funkci
unsigned short read_adc(unsigned char channel);

int main(void)
{
       unsigned short result, result_mV;

       // Inicializace interniho A/D prevodniku
       ADMUX = (ADC_VREF_TYPE & 0x03)<<REFS0;     // nastaveni zdroje napetove reference,
vysledek prevodu zarovnan doprava
       ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1); // povoleni A/D prevodniku, taktovaci
frekvence Fcpu/64

       while(1)
       {
              // cteni kanalu 0
              result = read_adc(0);
              // prevod na napeti [mV] (napeti [mV] = (result*ADC_VREF_MV)/1024)
              result_mV = (unsigned short)(((unsigned long)result*ADC_VREF_MV)>>10);

              // nejaky dalsi kod, ktery bude vyuzivat vysledek prevodu
       }
}

/*
Provede prevod na vybranem kanalu interniho A/D prevodniku
Vstupy:channel - cislo kanalu (od 0)
Vraci: 10-bit vysledek A/D prevodu
*/
unsigned short read_adc(unsigned char channel)
{
       ADMUX &= ~0x1F;                    // bity MUX4:0 je treba vynulovat
       ADMUX |= channel;                  // vyber kanalu
       ADCSRA |= 1<<ADSC;                 // spusteni prevodu
       while ((ADCSRA & (1<<ADIF))==0);   // cekani na dokonceni prevodu
       ADCSRA |= 1<<ADIF;                 // smazani priznaku ADIF
       return ADCW;                       // vycteni vysledku prevodu
}
```

- Připomínky k programu:
  - Nastavení taktovací frekvence A/D převodníku
    - Frekvence nastavena na 14745600 Hz / 64 = 230 400 Hz, což je cca maximální doporučená frekvence, při které lze ještě využít plných 10 bitů A/D převodníku.
  - Převod na napětí v [mV]
    - Probíhá podle vzorce (49).

        - Bitový posun `>>10` znamená dělení 1024 (výsledek A/D převodu je 10-bitový).
        - Násobení `result*ADC_VREF_MV` musí proběhnout na 32 bitech, protože se výsledek násobení se nevejde do proměnné `unsigned short`. Proto explicitní přetypování `(unsigned long)result`.
    - **Nekonstantní perioda vzorkování**, určená dobou vykonání kódu v nekonečné smyčce while(1).
        - Pro jednoduché aplikace nemusí vadit.
        - Pro aplikace typu regulace, digitální filtrace apod. to vadí, u nich je třeba zajistit konstantní periodu vzorkování.

- **Kód pro Arduino (také AVR-GCC, ale ovládání A/D převodníku přes knihovní funkce, ne přes registry):**

```
void setup()
{

}

void loop(void)
{
    int result, result_mV;

    // cteni napeti na A0, vysledek prevodu je bezrozmerne cislo 0-1023 [ad counts]
    result = analogRead(0);

    // prevod na napeti v mV: 0-1024 -> 0-5000 mV
    result_mV = (int)(((long)result*5000)>>10);
}
```

    - A/D převod probíhá na plném počtu bitů (10 nebo 12 bitů dle typu Arduina).
    - A/D převod probíhá s maximální rychlostí tak, aby taktovací frekvence A/D převodníku byla v doporučeném rozmezí 50-200 kHz. Např. u NANO, UNO, MEGA2560, Leonardo je taktovací frekvence A/D převodníku 125 kHz.
    - **Napěťová reference interního A/D převodníku je standardně Vcc (je rovna napájecímu napětí MCU, tedy 5 V nebo 3 V)**. Pozor, při napájení Arduina z USB není napětí přesně 5 V, obvykle je menší a na každém počítači obecně trochu jiné. Tím bude trpět přesnost měření napětí.
    - V případě nutntosti je možné napěťovou referenci změnit zavoláním funkce `analogReference()`.
        - https://docs.arduino.cc/language-reference/en/functions/analog-io/analogReference/
    - Další informace:
        - https://docs.arduino.cc/language-reference/en/functions/analog-io/analogRead/

- **Upravme předchozí příklad tak, abychom zajistili konstantní periodu vzorkování**
    - Konstantní periodu vzorkování (zvolme jí např. 20 ms) vytvoříme časovačem v režimu CTC (umožní přesné nastavení požadované periody vzorkování).
    - Uvažujme frekvenci CPU 14,7456 MHz.
    - Do ISR časovače umístíme kód pro čtení hodnoty z A/D převodníku.
    - V ISR časovače by mohl být i další kód, který hodnotu z A/D převodníku dále zpracuje (např. provede digitální filtraci, spustí regulátor, nastaví výstupy do požadovaných hodnot…). Důležité je, aby se kód stihl vykonat za jednu periodu vzorkování a to navíc s určitou rezervou.
- Pseudokód:

```
Nastav A/D převodník (jednorázový převod, dělička, zdroj ref. napětí)
Povol A/D převodník

Nastav časovač do režimu CTC
Povol přerušení při komparaci
Povol globálně přerušení
Nastav komparační hodnotu na 20 ms
Vynuluj časovač
Spusť časovač

WHILE (nekonečná smyčka)
END WHILE

ISR Timer Compare
       Přečti hodnotu na vybraném kanálu
       Převeď hodnotu na napětí v milivoltech
END ISR
```

- **Kód (pro překladač AVR-GCC):**

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define ADC_VREF_TYPE 0x00    // volba napetove reference interniho A/D prevodniku
                              // 0x00 - externi napetova reference na pinu AREF
                              // 0x01 - AVCC jako napetova reference
                              // 0x03 - interni napetova reference 2,56 V

#define ADC_VREF_MV   5000U   // hodnota referencniho napeti A/D prevodniku [mV]

// Deklarace funkci
unsigned short read_adc(unsigned char channel);

// Globalni promenne
volatile unsigned short result, result_mV;

int main(void)
{
       // Inicializace interniho A/D prevodniku
       ADMUX = (ADC_VREF_TYPE & 0x03)<<REFS0;     // nastaveni zdroje napetove reference,
vysledek prevodu zarovnan doprava
       ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1);  // povoleni A/D prevodniku, taktovaci
frekvence Fcpu/64

       // Nastaveni casovace Timer/Counter 1 do rezimu CTC
       TCCR1A &= ~((1<<WGM10) | (1<<WGM11));      // bit WGM10 = 0, WGM11 = 0
       TCCR1B |= (1<<WGM12); // bit WGM12 = 1
       TCCR1B &= ~(1<<WGM13);       // bit WGM13 = 0

       TIMSK |= (1<<OCIE1A); // povoleni preruseni pri komparaci
       sei(); // globalni povoleni preruseni

       OCR1A = 4607; // nastaveni komparacni hodnoty pro periodu 20 s (Fcpu = 14745600 Hz)
       TCNT1 = 0;            // vynulovani casovace
       TCCR1B |= (1<<CS11) | (1<<CS10);    // spusteni casovace s frekvenci Fcpu/64

       while(1)
       {
       }
}

/*
Provede prevod na vybranem kanalu interniho A/D prevodniku
Vstupy:channel - cislo kanalu (od 0)
Vraci: 10-bit vysledek A/D prevodu
*/
unsigned short read_adc(unsigned char channel)
{
       ADMUX &= ~0x1F;                    // bity MUX4:0 je treba vynulovat
       ADMUX |= channel;                  // vyber kanalu
       ADCSRA |= 1<<ADSC;                 // spusteni prevodu
       while ((ADCSRA & (1<<ADIF))==0);   // cekani na dokonceni prevodu
       ADCSRA |= 1<<ADIF;                 // smazani priznaku ADIF
       return ADCW;                       // vycteni vysledku prevodu
}
```

```
// Odmerovani periody vzorkovani
ISR(TIMER1_COMPA_vect)
{
        // cteni kanalu 0
        result = read_adc(0);
        // prevod na napeti [mV]
        result_mV = (unsigned short)(((unsigned long)result * ADC_VREF_MV) >> 10);

        // zde by mohl byt kod, ktery by zpracoval hodnotu z A/D prevodniku
}
```

- Připomínky k programu:
  - Žlutě jsou označeny změny vůči předchozí variantě.
  - Proměnné `result` a `result_mV` jsou nyní globální a označené `volatile`, protože potřebujeme, aby byly přístupné v `ISR` i v `main()`.
  - Pro odměřování periody vzorkování 20 ms byl použit časovač 1 v režimu CTC.
    - Komparační hodnota časovače se vypočetla takto:

$$\text{komparační hodnota} = \text{OCR1A} = \text{požadovaný časový interval} \cdot \frac{F_{CPU}}{N} - 1 \tag{61}$$

$$\text{komparační hodnota} = \text{OCR1A} = 0{,}02 \cdot \frac{1474600}{64} - 1 = 4607 \tag{62}$$

  - Spouštění A/D převodu se přesunulo do `ISR` časovače. V `ISR` by mohl být umístěn kód, který by vzorek z A/D převodníku dále zpracoval.
  - Nekonečná smyčka v `main()` zůstala prázdná. Do ní by se mohl umístit nějaký jiný užitečný kód, např. výpis na LCD apod. V tomto případě je nutné zajistit při čtení proměnným `result` a `result_mV` atomický přístup k nim, aby během čtení v `main()` nemohlo dojít k jejich změně (což by nastalo po vyvolání ISR časovače).

4.5.2 Kontinuální převod

- Vytvořme takovou obsluhu A/D převodníku, která umožní kontinuální převádění vybraných kanálů a ukládání výsledků převodů do pole.
- Použijeme přerušení A/D převodníku „převod dokončen“.
- Do ISR „převod dokončen“ umístíme kód, který zajistí
    - Uložení výsledku převodu do pole pod index odpovídající číslu aktuálně vybraného kanálu, kterému patří výsledek převodu.
    - Změnu kanálu.
    - Spuštění převodu na vybraném kanálu.
- Nepoužijeme hardwarový Free Running mód A/D převodníku pro kontinuální převod, nýbrž převod budeme spouštět ručně. Kód bude kompatibilní se všemi AVR, tedy i s těmi, které hardwarový Free Running mód nemají.
- Výsledky převodu budeme zpracovávat jednou za čas v jiné části programu, ne v ISR A/D převodníku.
- **Vlastnosti navrhovaného řešení**
    - Pole hodnot s výsledky A/D převodu na vybraných kanálech bude obsahovat vždy jen poslední nejaktuálnější hodnoty.
    - Perioda vzorkování je závislá na počtu převáděných kanálů.
    - Data zpracujeme jednou za čas v jiné části programu
        - Použijeme hodnoty uložené v poli z naposledy provedených převodů.
        - Program se nemusí zdržovat se spouštěním převodu – data již má připravená.
        - Pole hodnot je sdílené různými částmi programu – při čtení hodnot z pole musíme zajistit atomický přístup, aby nedošlo k chybnému vyčtení dat z pole.
- Vyjdeme z programu, který byl uveden v předchozí kapitole, který upravíme.
- **Pseudokód:**

```
Nastav A/D převodník (jednorázový převod, dělička, zdroj ref. napětí, povol přerušení)
Povol A/D převodník

Nastav časovač do režimu CTC
Povol přerušení při komparaci
Povol globálně přerušení
Nastav komparační hodnotu na 20 ms
Vynuluj časovač
Spusť časovač
Spusť A/D převod na prvním kanálu v pořadí

WHILE (nekonečná smyčka)
END WHILE

ISR Timer Compare
       Zpracování výsledků A/D převodu
```

```
END ISR

ISR ADC
	Ulož do pole výsledek převodu
	Posuň se na další kanál
	Spusť převod
END ISR
```

- **Kód (pro překladač AVR-GCC):**

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define ADC_VREF_TYPE 0x00    // volba napetove reference interniho A/D prevodniku
                              // 0x00 - externi napetova reference na pinu AREF
                              // 0x01 - AVCC jako napetova reference
                              // 0x03 - interni napetova reference 2,56 V

#define ADC_VREF_MV           5000U  // hodnota referencniho napeti A/D prevodniku [mV]

// Cisla kanalu A/D prevodniku, ktere se budou cist
#define FIRST_ADC_INPUT             0
#define LAST_ADC_INPUT              5
#define ADC_INPUTS                  (LAST_ADC_INPUT-FIRST_ADC_INPUT+1)

// Globalni promenne
volatile unsigned short adc_data[ADC_INPUTS];            // pole s vysledky A/D prevodu,
index prvku = cislo kanalu
volatile unsigned short adc_data_copy[ADC_INPUTS];       // kopie pole s vysledky A/D
prevodu

int main(void)
{
	// Inicializace interniho A/D prevodniku
	ADMUX = (ADC_VREF_TYPE & 0x03)<<REFS0;     // nastaveni zdroje napetove reference,
vysledek prevodu zarovnan doprava
	ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADIE);      // povoleni A/D prevodniku,
taktovaci frekvence Fcpu/64, povoleni preruseni

	// Nastaveni casovace Timer/Counter 1 do rezimu CTC
	TCCR1A &= ~((1<<WGM10) | (1<<WGM11));      // bit WGM10 = 0, WGM11 = 0
	TCCR1B |= (1<<WGM12);        // bit WGM12 = 1
	TCCR1B &= ~(1<<WGM13);       // bit WGM13 = 0

	TIMSK |= (1<<OCIE1A); // povoleni preruseni pri komparaci
	sei(); // globalni povoleni preruseni

	OCR1A = 4607; // nastaveni komparacni hodnoty pro periodu 20 s (Fcpu = 14745600 Hz)
	TCNT1 = 0;            // vynulovani casovace
	TCCR1B |= (1<<CS11) | (1<<CS10);    // spusteni casovace s frekvenci Fcpu/64

	// Spusteni A/D prevodu na prvnim kanalu v poradi
	ADMUX |= FIRST_ADC_INPUT;    // vyber kanalu
	ADCSRA |= 1<<ADSC;    // spusteni prevodu

	while(1)
	{
	}
}

// Odmerovani periody vzorkovani
ISR(TIMER1_COMPA_vect)
{
	unsigned short i;
	unsigned char sreg_backup;
```

```
        // kopie dat do pracovniho pole, musi probehnout atomicky
        sreg_backup = SREG;
        cli();
        for (i=0;i<ADC_INPUTS;i++)
        {
                adc_data_copy[i] = adc_data[i];
        }
        //   nebo   misto   cyklu   for   lze   pouzit:   memcpy((void*)adc_data_copy,(const
void*)adc_data,sizeof(adc_data));
        SREG = sreg_backup;

        // zpracovani hodnot z A/D prevodniku
        /*
        teplota = PrevedNaTeplotu(adc_data_copy[0]);
        tlak = PrevedNaTlak(adc_data_copy[1]);
        atd..
        */
}

// ISR A/D prevod dokoncen
ISR(ADC_vect)
{
        static unsigned char input_index = 0; // cislo aktualniho kanalu

        // Vycteni vysledku prevodu
        adc_data[input_index] = ADCW;

        // Vyber dalsiho kanalu
        if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT)) input_index = 0;
        ADMUX &= ~0x1F;
        ADMUX |= FIRST_ADC_INPUT + input_index;

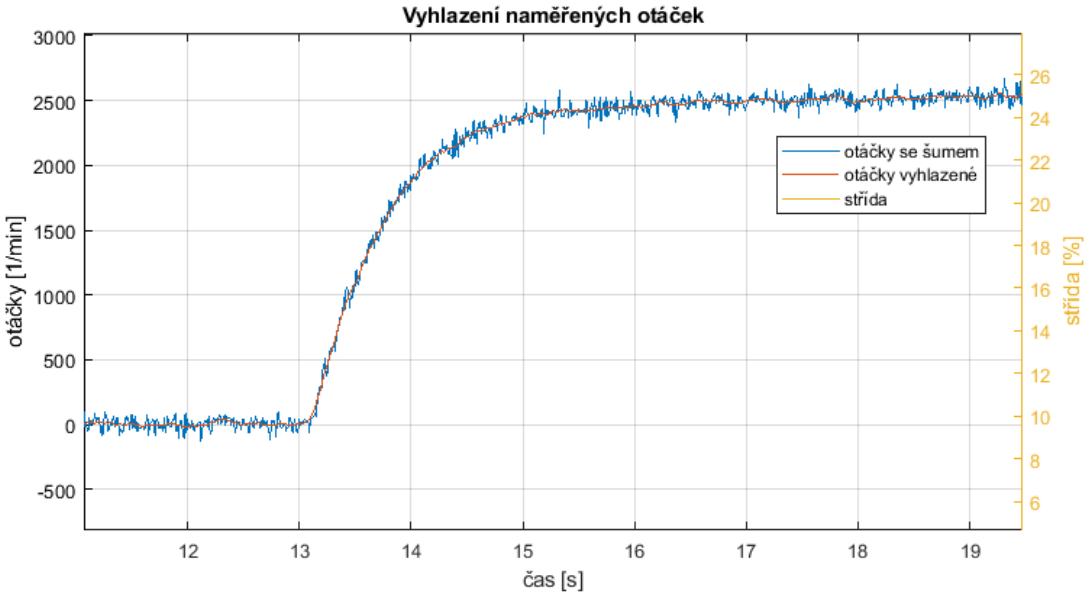
        // Spusteni prevodu
        ADCSRA |= 1<<ADSC;
}
```

- Připomínky k programu
  - Výsledky A/D převodu se ukládají do pole `adc_data[]`.
  - První převod je spuštěn ihned po dokončení inicializace A/D převodníku ve funkci `main()`.
  - V ISR „převod dokončen“ se vždy nejprve uloží aktuální výsledek převodu do pole, pak se vybere další kanál a spustí se převod.
  - Zpracování výsledků převodu se provádí periodicky každých 20 ms v ISR časovače.
    - Nejprve se zkopíruje pole `adc_data[]` do pole `adc_data_copy[]` a pracuje se s kopií.
    - Před kopírováním je globálně zakázáno přerušení - tím se zabraňuje zápisu nové hodnoty do pole `adc_data[]` v průběhu čtení hodnot z něj.
    - Po dokončení kopírování je přerušení globálně opět povoleno, resp. je obnoven původní obsah registru SREG – tedy i původní hodnota bitu I.

5 Vybrané programové techniky při použití A/D převodníku

5.1 Digitální filtrace

- Ve vzorcích signálu je většinou obsažen šum, jehož velikost je třeba omezit. Signál je třeba filtrovat.



Obrázek 94 Šum v měřeném signálu (modrý průběh)

- Existují různé typy digitálních filtrů podle oblasti použití:

FILTER USED FOR:	FILTER IMPLEMENTED BY: Convolution *Finite Impulse Response (FIR)*	Recursion *Infinite Impulse Response (IIR)*
Time Domain *(smoothing, DC removal)*	Moving average	Single pole
Frequency Domain *(separating frequencies)*	Windowed-sinc	Chebyshev
Custom *(Deconvolution)*	FIR custom	Iterative design

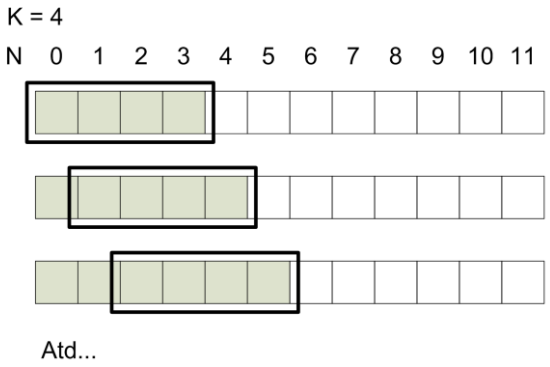
Obrázek 95 Typy digitálních filtrů pro různé oblasti použití

- **Time Domain filtry** použijeme tehdy, pokud je informace ukryta v časovém průběhu signálu (v jeho tvaru).
- **Frequency Domain filtry** použijeme tehdy, pokud je informace ukryta v amplitudě, frekvenci nebo fázi signálu. Cílem těchto filtrů je oddělit jedno pásmo frekvencí od jiného.

- V této kapitole budou ukázany dva typy filtrů spadající do oblasti Time Domain. Jedná se filtry založené na klouzavém průměrování (moving average) určené pro vyhlazení průběhu signálu v čase.
- Zájemce o hlubší studium digitálních filtrů odkazuji např. na příručku *The Scientist and Engineer's Guide to Digital Signal Processing, Steven W. Smith, Second Edition, California Technical Publishing , 1999, ISBN 0-9660176-7-6, ISBN 0-9660176-4-1, ISBN 0-9660176-6-8* dostupnou na https://www.analog.com/en/education/education-library/scientist_engineers_guide.html

5.1.1 Jednoduchý klouzavý průměr

- Simple Moving Average (SMA)
- Je to filtr typu FIR (Finite Impulse Response), dolní propust.
    - FIR = nový vzorek na výstupu filtru se počítá z aktuálního a několika minulých vstupních vzorků.
- Vzorky se ukládají do pole o `K` prvcích.
- **Po každém dokončeném převodu se vypočte průměr z posledních `K` prvků v poli.**
- Průměrovací okno široké `K` vzorků se po každém novém vzorku posune o jeden vzorek (nejstarší se zahodí a místo něj se použije nejnovější) a opět se vypočte průměr z `K` vzorků.



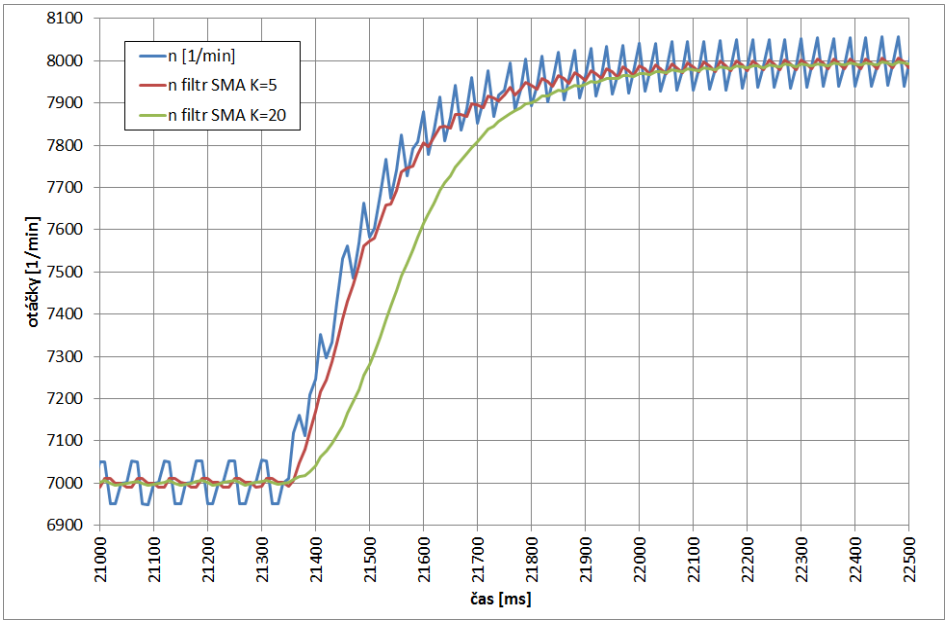
Obrázek 96 Princip filtru typu Simple Moving Average

- **Všechny vzorky mají stejnou váhu** – jedná se o nevážený klouzavý průměr. `N` značí nejnovější (`N`-tý) vzorek.
- **Čím je větší počet vzorků (`K`), které se průměrují, tím se signál více vyhlazuje.**
- Pro výpočet první hodnoty SMA musíme počkat na dostatečný počet vzorků (K).

- Např. průměr z posledních 4 prvků (K = 4):

N:	0	1	2	3	4	5	6	7
Hodnoty:	10	11	12	13	14	15	16	17

SMA pro N < 3 nelze vypočítat, musíme počkat na dostatečný počet (K) vzorků
SMA pro N = 3: (10 + 11 + 12 + 13) / 4 = 11,5
SMA pro N = 4: (11 + 12 + 13 + 14) / 4 = 12,5
SMA pro N = 5: (12 + 13 + 14 + 15) / 4 = 13,5
atd…



Obrázek 97 Filtr typu SMA

- **Přímý výpočet**

```
sum = 0.0;
for (i = N-K+1; i <= N; i++) sum += vzorek[i];
avg = sum/K;
```

  - Pouze znázornění principu – prakticky nerealizovatelné, protože bychom si museli pamatovat všechny vzorky (tolik paměti nemáme).
  - **Nevýhoda:** průměr se počítá ze všech vzorků – pro velké `K` časově náročné.

- **Rekurentní výpočet**

```
avg = (avg*K + vzorek[N] - vzorek[N-K]) / K
```

nový průměr — starý průměr — nový vzorek — nejstarší vzorek

- a po úpravě:

```
avg = avg + (1/K) * (vzorek[N] - vzorek[N-K])
```

  - kde `vzorek[N-K]` je nejstarší a `vzorek[N]` je nejnovější
  - Síla filtrace je rovna `1/K`.
    - Čím je `K` větší, tím více je signál vyhlazen.
    - `K` nabývá hodnot od 1 do nekonečna.
    - Při `K=1` je filtr vyřazen z činnosti (nefiltruje).
  - Musíme si pamatovat posledních `K` vzorků.
  - **Výhoda:** výpočet je rychlejší, protože se nemusí procházet celé pole vzorků, stačí použít nejnovější a nejstarší vzorek a minulou hodnotu průměru.
- **Praktická realizace**
  - Postupujeme podle vzorce pro rekurentní výpočet SMA.
  - Pamatujeme si posledních `K` vzorků.
  - Vzorky ukládáme do kruhového bufferu (pole) o konstantní délce `K` vzorků.
  - Pamatujeme si index nejstaršího prvku v poli.
  - Pamatujeme si průběžnou sumu `avg*K`.
  - Po každém příchodu nového vzorku:
    - Od průběžné sumy `avg*K` odečteme nejstarší vzorek a přičteme nejnovější vzorek, nejnovější vzorek uložíme na místo nejstaršího.
    - Inkrementujeme index nejstaršího prvku v poli, aby ukazoval na další prvk v pořadí, který bude nejstarším prvkem v příštím kole.
    - Pokud index přesáhne konec pole, nastavíme ho na začátek pole (index = 0) – princip kruhového bufferu.
- **Kód (pro překladač AVR-GCC)**
  - Níže uvedený kód implementuje filtr typu SMA.
  - Pro obsluhu filtru je vytovřena dvojice funkcí:
    - `sma()` – pro výpočet průměru
    - `smaInit()` – pro úvodní inicializaci filtru
  - Pomocná data pro management SMA filtru (index nejstaršího prvku, průběžná suma, počet využitých prvků v kruhovém bufferu) jsou zapouzdřena do struktury `SMA_t`.
  - Aby mohl uživatel filtr používat, musí v kódu vytvořit:
    - Pole pro uložení vzorků.
    - Proměnnou typu `SMA_t` pro uložení pomocných dat filtru.

- Pak stačí periodicky volat funkci `sma()`, která vrací průměr.
- Do funkce `sma()` je nutné předat tato data:
  - nový vzorek
  - data pro management filtru
  - kruhový buffer pro uložení vzorků
  - počet vzorků, ze kterých se dělá průměr
- Výhody této implementace:
  - Data jsou oddělena od logiky filtru.
  - V aplikaci je možné mít vytvořeno více filtrů s různou délkou okna.
- Omezení této konkrétní implementace:
  - Pracuje pouze s celými čísly.
  - Vzorky musí být typu `uint16_t`.
  - Průměr je také typu `uint16_t`.
  - Pokud jiný datový typ → změna hlavičky funkce `sma()`.

```
#include <avr/io.h>
#include <inttypes.h>

// struktura s daty pro obsluhu filtru typu Simple Moving Average
typedef struct
{
	uint8_t index;		// index nejstarsiho prvku
	int32_t sum;		// suma vsech vzorku v bufferu
	uint8_t count;		// vyuzity pocet vzorku v bufferu (pro rozjezd filtru)
} SMA_t;

// filtr pro 1. signal
#define K1    5			// sirka prumerovaciho okna
uint16_t buffer1[K1];		// kruhovy buffer pro vzorky
SMA_t filterSMA1;			// data pro management filtru SMA

// filtr pro 2. signal
#define K2    20			// sirka prumerovaciho okna
uint16_t buffer2[K2];		// kruhovy buffer pro vzorky
SMA_t filterSMA2;			// data pro management filtru SMA
// Spousti filtr typu Simple Moving Average
/*
parametry:    newSample            Novy vzorek
              filter               Data pro management filtru
              ringBuffer           Kruhovy buffer pro vzorky
              ringBufferLength     Sirka prumerovaciho okna (pocet vzorku)

vraci: Prumernou hodnotu za poslednich ringBufferLength vzorku.
*/
uint16_t sma(uint16_t newSample, SMA_t *filter, uint16_t ringBuffer[], uint8_t
ringBufferLength)
{
	filter->sum -= ringBuffer[filter->index];  // od sumy odecteme nejstarsi vzorek
	ringBuffer[filter->index] = newSample;     // nejstarsi vzorek nahradime nejnovejsim
	filter->sum += ringBuffer[filter->index];  // k sume pricteme nejnovejsi vzorek
	filter->index++; // posuneme ukazatel na dalsi prvek v poradi = nejstarsi prvek v
dalsim pruchodu
	filter->index = filter->index % ringBufferLength; // presun na zacatek pole
```

```
	if (filter->count < ringBufferLength) filter->count++;   //  zaplnenost  pole  (pro
rozjezd filtru)

	return (uint16_t)(filter->sum / filter->count);   //  prumer  ze  vsech  vzorku,  bez
desetinnych mist
}

// Inicializuje filtr typu Simple Moving Average
/*
Naplni filtr starymi vzorky - vsechny prvky kruhoveho bufferu
naplni hodnotou initialAverage.

parametry:	initialAverage		Vychozi prumerna hodnota
		filter			Data pro management filtru
		ringBuffer		Kruhovy buffer pro vzorky
		ringBufferLength	Sirka prumerovaciho okna (pocet vzorku)

vraci: nic
*/
void  smaInit(uint16_t  initialAverage,  SMA_t  *filter,  uint16_t  ringBuffer[],  uint8_t
ringBufferLength)
{
	uint8_t i;

	filter->count = ringBufferLength;
	filter->index = 0;
	filter->sum = (int32_t)initialAverage*ringBufferLength;

	for (i=0;i<ringBufferLength;i++)
	{
		ringBuffer[i] = initialAverage;
	}
}

int main(void)
{
	uint16_t avg1, avg2;  // filtrujeme 2 signaly

	// inicializace filtru (vychozi hodnota prumeru 0)
	smaInit(0,&filterSMA1,buffer1,K1);
	smaInit(0,&filterSMA2,buffer2,K2);

	while(1)
	{
		// naznak pouziti funkce sma()
		for (uint16_t i=0;i<100;i++)
		{
			avg1 = sma(i,&filterSMA1,buffer1,K1);	// filtrace 1. signalu
			avg2 = sma(i*2,&filterSMA2,buffer2,K2);	// filtrace 2. signalu
		}
	}
}
```

5.1.2 Klouzavý průměr s exponenciálním zapomínáním

- Exponential Moving Average (EMA)
- Chceme se zbavit nutnosti pamatovat si všech `K` vzorků. Proto nahradíme nejstarší vzorek jeho odhadem. Nejlepší odhad nejstaršího měření je poslední hodnota průměru.

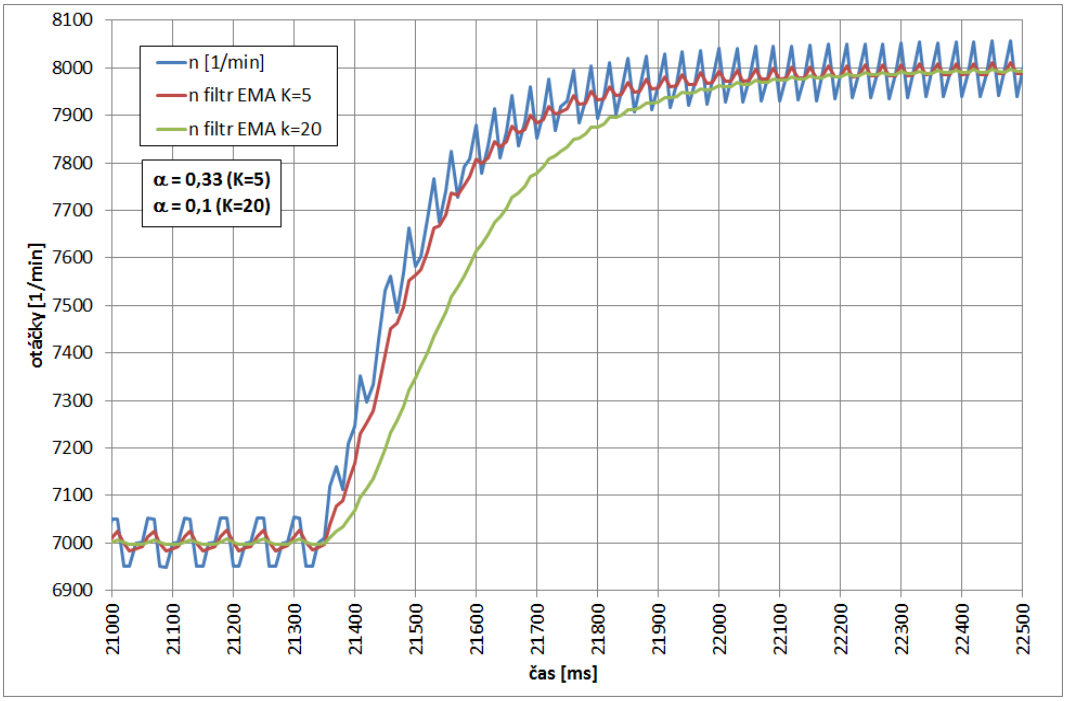
```
avg = avg + (2/(K+1)) * (vzorek[N] - avg)
```

nebo-li

```
avg = avg + α * (vzorek - avg)
```

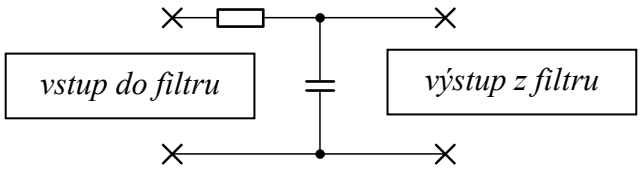
nový průměr starý průměr nový vzorek

  - Tím jsme se zbavili nutnosti uchovávat všechny vzorky, pro výpočet průměru stačí nejnovější vzorek a minulá hodnota průměru.
  - $\alpha = 2/(K+1)$ je váha filtru, čím je menší tím méně věříme novým hodnotám a tím je filtrace silnější, α nabývá hodnoty z intervalu <0;1>.
  - **Vzorky nemají stejnou váhu**.
    - Nejnovější vzorek má největší váhu.
    - Váha starších vzorků klesá exponenciálně (odtud název tohoto klouzavého průměru) dle vzorce $\left(\frac{K-1}{K}\right)^N$.

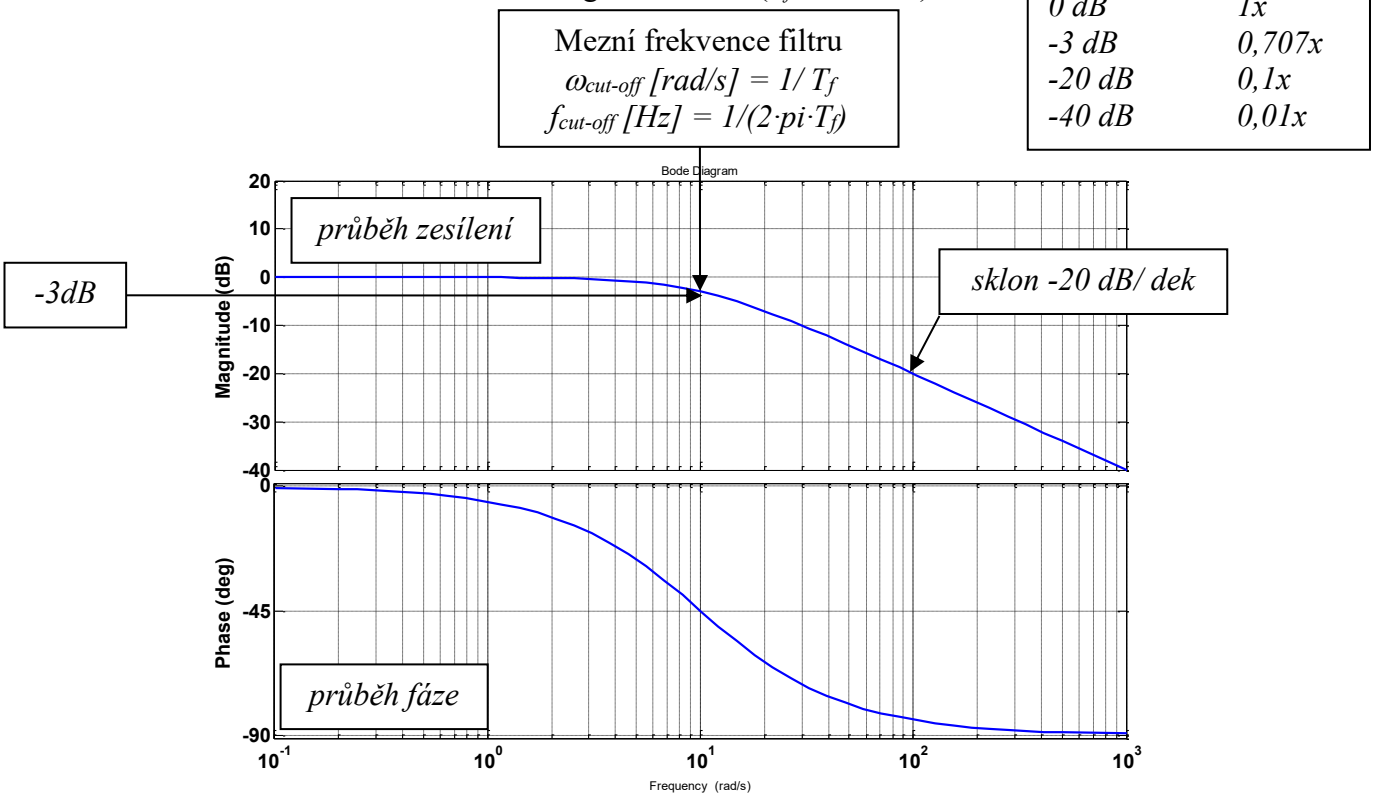


Obrázek 98 Filtr typu EMA

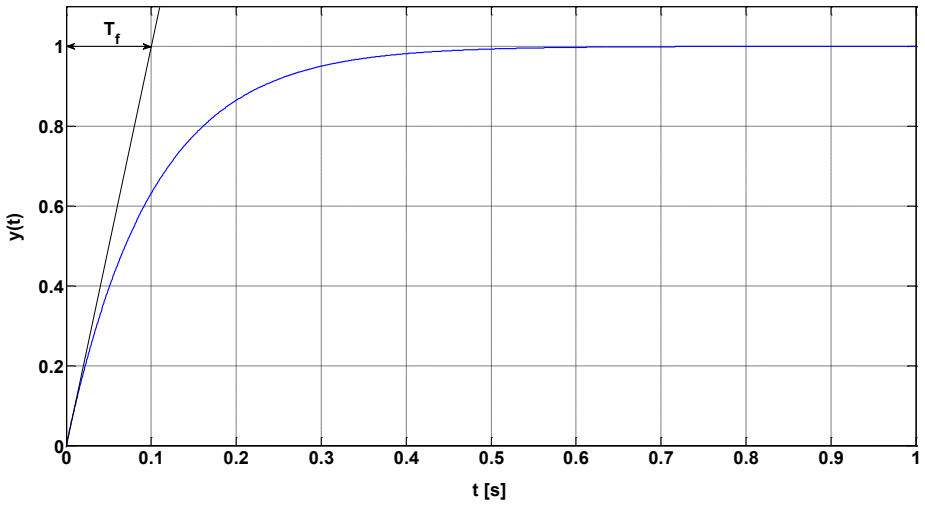
- **Filtr typu EMA je číslicový filtr 1. řádu typu IIR (Infinite Imuplse Response), dolní propust**.
    - Výše uvedený vzorec pro výpočet EMA je diferenční rovnice, kterou je filtr popsán.
    - Diferenční rovnice je ekvivalent rovnice diferenciální, kterou se popisují spojité systémy.
- Uvedenému číslicovému filtru odpovídá ve spojité oblasti **filtr analogový** popsaný diferenciální rovnicí $\tau_f y'(t) + y(t) = u(t)$, které odpovídá přenos v Laplaceově transformaci $G(s) = \frac{Y(s)}{U(s)} = \frac{1}{T_f s + 1}$.
    - Kde: T_f – časová konstanta filtru [s], která určuje mezní frekvenci filtru, což je frekvence, od které začíná filtr tlumit vstupní signál (přesněji: mezní frekvence filtru je frekvence, při které zesílení filtru nabývá hodnoty -3dB, tj. v lineárním měřítku zeslabení na úrovni 0,707x)
- Takovým filtrem je např. integrační článek s časovou konstantou $T_f = RC$.



- Frekvenční charakteristika analogového filtru (T_f = 100 ms):



- Odezva na jednotkový skok na vstupu analogového filtru (T_f = 100 ms):



  - Odezva filtru na jednotkový skok je v časové oblasti popsána rovnicí:

$$y(t) = 1 - e^{-\frac{t}{\tau_f}} \tag{63}$$

- Přepočet mezi váhou filtru a časovou konstantou T_f u číslicové varianty filtru je následující (aproximace):

$$\alpha = 1 - e^{-\frac{T_s}{T_f}} \tag{64}$$

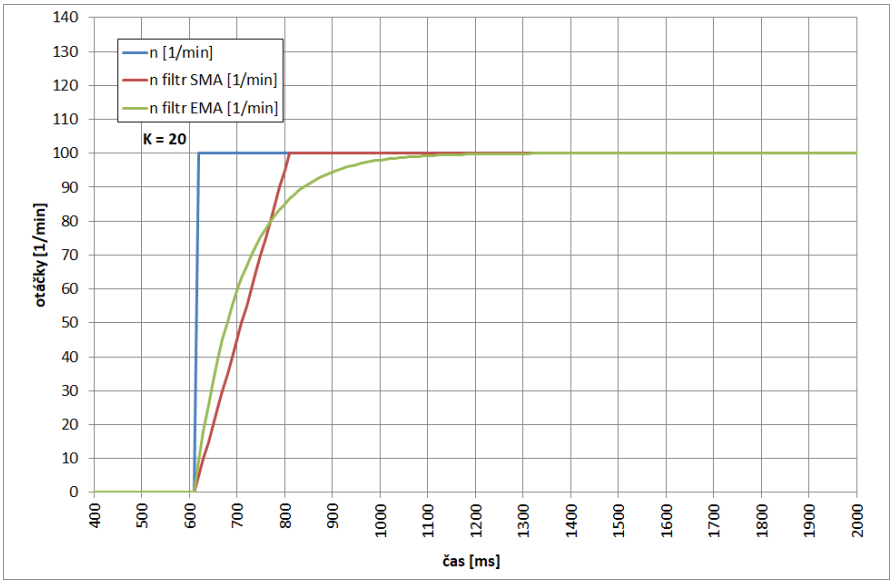
- Kde: T_s – perioda vzorkování [s]
  T_f – časová konstanta filtru [s]
- Síla filtrace α se volí jako kompromis mezi požadovaným vyhlazením signálu a jeho zpožděním – důležité zejména při zpětnovazební regulaci.
- Vztah (64) není úplně přesný, jedná se pouze o aproximaci. Poměrně přesně platí pro mezní frekvence filtru $F_{cut\text{-}off}$ (frekvence při níž je zesílení filtru -3 dB) menší nebo rovny hodnotě $0{,}1 \cdot F_{vz}$, kde F_{vz} je frekvence vzorkování. Pro hodnoty mezní frekvence větší než $0{,}1 \cdot F_{vz}$ chyba aproximace rychle roste.

$$F_s = \frac{1}{T_s} \tag{65}$$

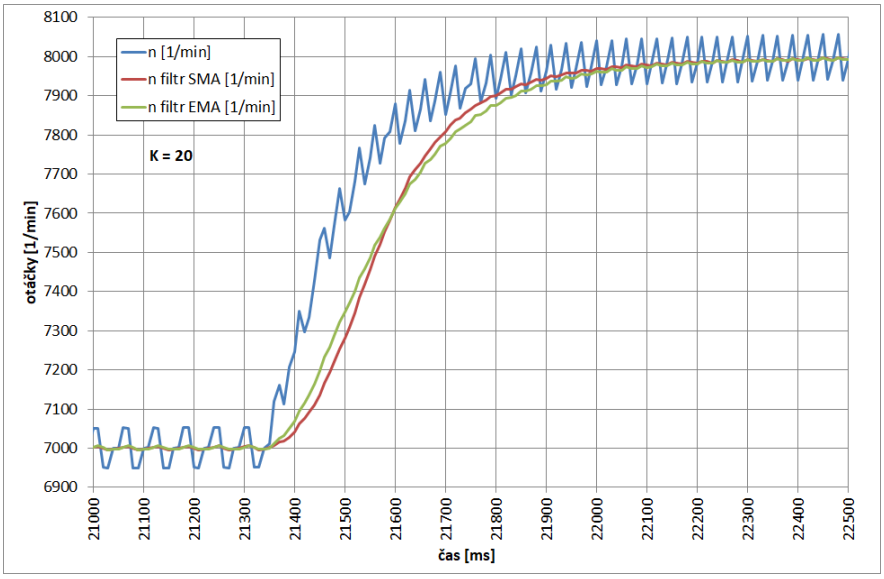
Kde: T_s – perioda vzorkování [s]
F_s – frekvence vzorkování [Hz]

5.1.3 Porovnání filtrů

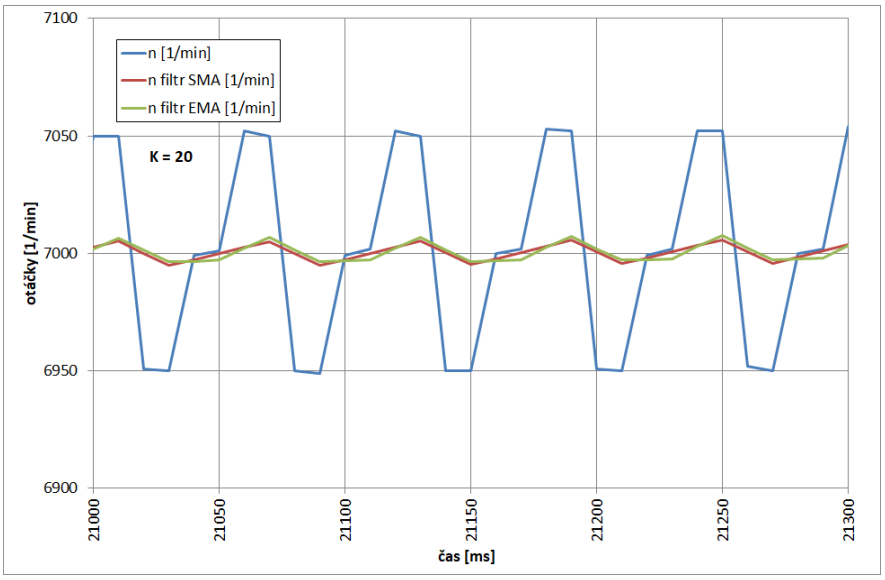
- Filtr typu EMA má menší zpoždění a reaguje na nové vzorky ochotněji než filtr typu SMA, protože staré vzorky mají menší váhu než novější.
- Filtr typu SMA zase poskytuje skutečnou průměrnou hodnotu signálu za několik vzorků dozadu. Filtr typu SMA je „línější“ proti filtru typu EMA.
- Filtr typu EMA je paměťově méně náročný, u něj si není třeba pamatovat všechny vzorky, ze kterých je průměr počítán (na rozdíl od filtru SMA).
- Nelze stanovit, který z filtrů je lepší, každý má jiné vlastnosti.
- **Porovnání filtru SMA a EMA pro stejné K, perioda vzorkování T_s = 10 ms:**



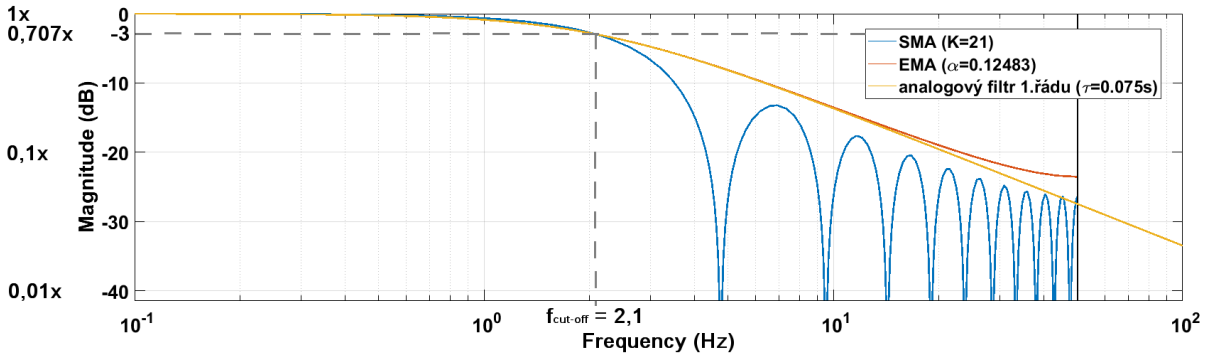
Obrázek 99 Odezva na skokovou změnu vstupního signálu (K = 20, T_s = 10 ms)



Obrázek 100 Vyhlazení signálu obsahujícího šum (K = 20, T_s = 10 ms)



Obrázek 101 Vyhlazení signálu obsahujícího šum (K = 20, T_s = 10 ms), detail



Obrázek 102 Amplitudová frekvenční charakteristika filtrů (K = 21, T_s = 10 ms)

5.1.4 Důležitá poznámka k implementaci filtru EMA

- Při implementaci do programu na AVR je nutné zvážit, zda použít plovoucí aritmetiku nebo celočíselnou.
- Velký rozdíl bude v rychlosti, aritmetika s plovoucí řádovou čárkou bude pomalejší.
- **Při použití celočíselné aritmetiky je nutné koeficient α škálovat** - vynásobit ho měřítkem alespoň 128.
    - Takže α může nabývat hodnot 0 až 128 a reprezentuje tak hodnotu α na intervalu <0;1>.
    - Výsledek je pak nutno měřítkem zpětně podělit.
    - Měřítko volíme jako mocninu 2^n (tj. 128, 256, 512, 1024 apod.), abychom následně mohli ve výpočtu nahradit dělení bitovým posunem o *n* bitů vpravo.
    - Čím vyšší měřítko použijeme, tím jemnější krok v nastavení koeficientu α budeme mít.
- Předpokládejme, že všechny hodnoty vstupující do výpočtu jsou celá čísla.

```
avg += α * (vzorek - avg)/128;
```

- α má po vyškálování x128 rozsah: <0;128>
- Používáme výhradně bitové posuny, především při dělení, takže:

```
avg += (α * (vzorek - avg))>>7;
```

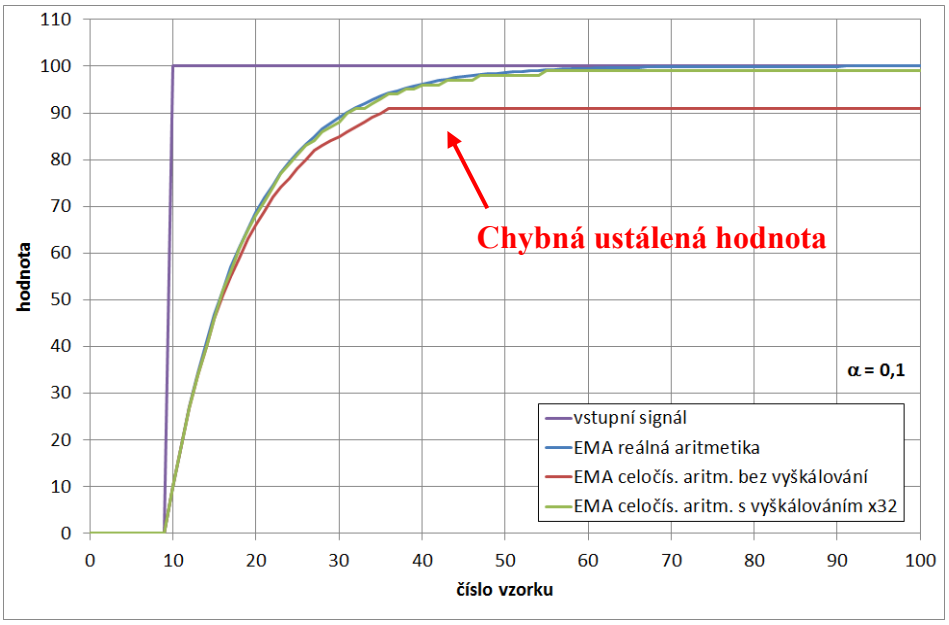
- **Výpočtem v celočíselné aritmetice vzniká nepřesnost** - výsledek je oříznut o případná desetinná místa.

  např.: (předpokládáme, že všechny proměnné jsou celočíselné)
  `avg = 583 + (50 * (956 - 583))>>7 = 583 + 145 = 728`

  ale ve skutečnosti by mělo být:

  `avg = 583 + (50 * (956 - 583))/128 = 583 + 145,703125 = 728,703125`

  Toto ořízntutí způsobuje kumulaci chyby výpočtu průměru `avg`, protože nová hodnota průměru `avg` je počítána z minulé hodnoty `avg`. **Tím se mění frekvenční charakteristika filtru a průběh odezvy v časové oblasti, což je patrné z následujícího obrázku.**



Obrázek 103 Odezva filtru na jednotkový skok signálu na jeho vstupu

- Vliv této chyby se projeví především při malých hodnotách koeficientu α.
- **Eliminace této chyby**
  - Aby se tato chyba alespoň částečně eliminovala, je nutné uměle rozšířit (vyškálovat) výpočet alespoň o jedno desetinné místo.
    - Již při vynásobění měřítkem 32 (rozšíření výpočtu o 5 bitů) dosáhneme dobrých výsledků.
  - Pro následnou reprezentaci výsledků je nutné mít na paměti toto rozšíření – skutečná hodnota výsledku je pak 32x menší.

    Př.: Vzorek násobím škálou 32. Výsledný průměr `avg` tedy bude také 32x větší.

    ```
    avg += (α * ((vzorek << 5) - avg))>>7;
    ```

  - Opět místo násobení používáme bitový posun, tentokrát vlevo, zde o 5 bitů (škálujeme x32).
  - Je nutné dávat pozor na možné přetečení při násobení velkých čísel.
    - Největší datový celočíselný typ je `long` dlouhý 4 B. Je nutné přetypovat operand `vzorek` na typ `long`, aby výpočty proběhly správně na 32 bitech a se znaménkem.

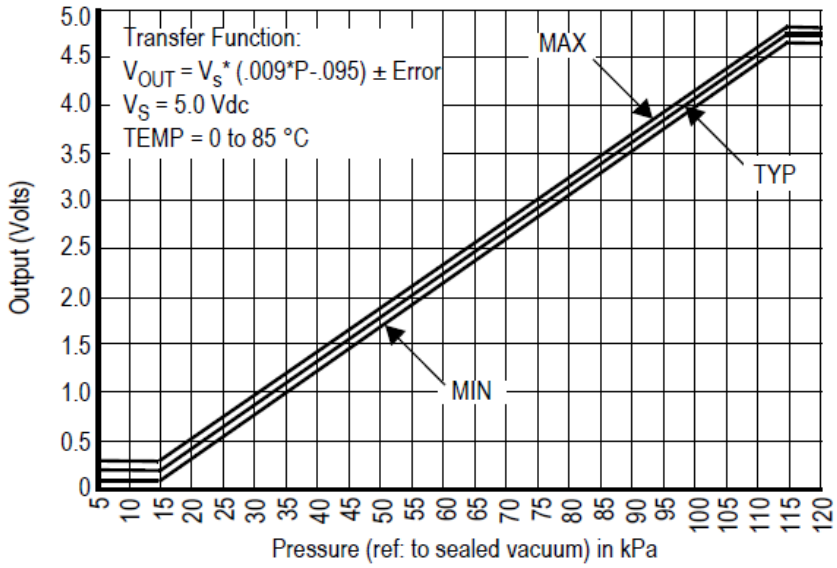
```
// definice promennych
unsigned char A = 64;   // koeficient filtrace (alfa), vyskalovany x128
unsigned short vzorek;  // napr. z A/D prevodniku
long avg = 0;
unsigned short vzorek_filt;

// filtrace
avg += (A * (((long)vzorek << 5) - avg))>>7;
vzorek_filt = avg >> 5; // vyfiltrovana hodnota zbavena meritka
```

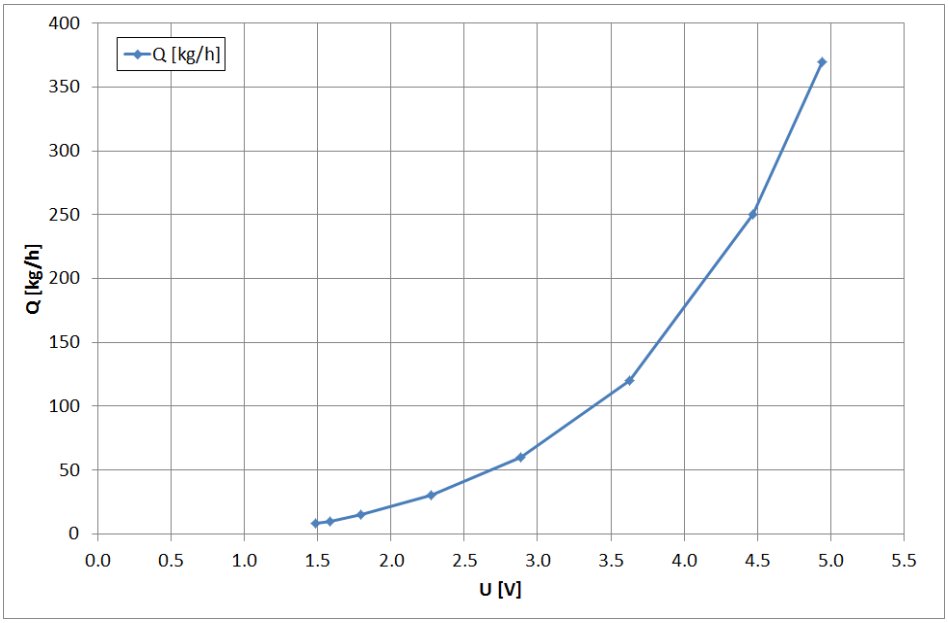
- Ve výše uvedeném příkladu pro interní výpočet závorky potřebuji 16b (vzorek) + 5b (rozšíření) + 7b koef. filtrace A, celkem tedy 28b což se vejde do proměnné typu `long`.
- Výpočet **v celočíselné aritmetice** trvá na 16 MHz cca **23 us**.
- Výpočet **s reálnou aritmetikou** cca **55 us**. (CodeVision, optimalizace Speed, úroveň Max).
  - Úspora času je tedy cca 50 %, což při větším počtu filtrovaných veličin již není zanedbatelná úspora.

5.2 Lineární interpolace

- Pro dopočet funkční hodnoty mezi dvěma známými body.
- Lineární interpolace = propojení dvou známých bodů přímkou.
- Využití:
  - Vyčítání hodnot z převodní charakteristiky (snímače, akčního členu apod.)
  - Převodní charakteristika je definovaná ve:
    - dvou krajních bodech (lineární převodní charakteristika)
    - více bodech (nelineární převodní charakteristika)

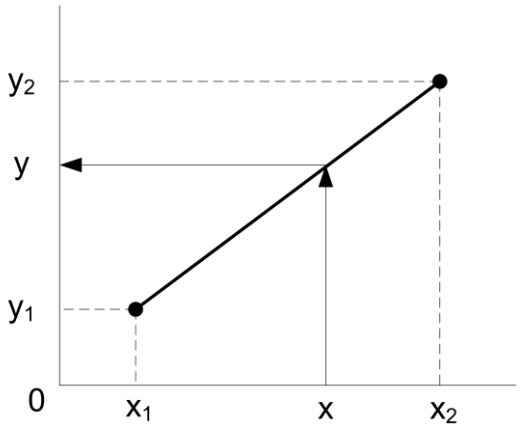


Obrázek 104 Lineární převodní charakteristika snímače tlaku



Obrázek 105 Nelineární převodní charakteristika snímače hmotnosti nasávaného vzduchu v autě

5.2.1 Princip lineární interpolace



Obrázek 106 Princip lineární interpolace

Funkční hodnota *y* mezi krajními body se vypočte jako:

$$y = (x - x_1) \cdot \frac{(y_2 - y_1)}{(x_2 - x_1)} + y_1 \tag{66}$$

Kde: $\frac{(y_2 - y_1)}{(x_2 - x_1)}$ je směrnice přímky

5.2.2 Implementace v programu

- Budou ukázány dvě implementace:
  - Výpočet vzorce (66) probíhá v reálné (float) aritmetice.
  - Výpočet vzorce (66) probíhá celočíselně.

- **Výpočet v reálné (float) aritmetice**

```
/// Provadi linearni interpolaci hodnoty y pro zadanou hodnotu x v zadanem intervalu.
/**
Doba vykonani: 1952 instrukcnich cyklu (AVR-GCC, vypnuta optimalizace -O0)

<PRE>
       y2     |_________________
              |                 |
       y      |___________      |
              |           |     |
       y1     |___        |     |
              |   |       |     |
              |   |       |     |
              ====|=======|====|====
              0   x1      x    x2

              y = [(x - x1)*(y2 - y1)/(x2 - x1)] + y1
</PRE>

Pokud je x1 rovno x2, tak metoda vraci hodnotu y1.

@param[in] x1 Leva hodnota intervalu ve vektoru X.
@param[in] x2 Prava hodnota intervalu ve vektoru X.
@param[in] y1 Leva hodnota intervalu ve vektoru Y.
@param[in] y2 Prava hodnota intervalu ve vektoru Y.
@param[in] x  Hodnota x, pro kterou se ma nalezt hodnota y.
@return       Hodnota y odpovidajici hodnote x.
*/
int32_t interp1D_flt(int32_t x1, int32_t x2, int32_t y1, int32_t y2, int32_t x)
{
       float temp;
       int32_t y;

       temp = (float)x2 - (float)x1;     // dx
       if (temp != 0)      // zabraneni deleni nulou
       {
              temp = ((float)y2 - (float)y1) / temp;   // dy/dx
              temp *= ((float)x - (float)x1);          // (dy/dx)*(x-x1)
              y = (int32_t)((float)y1 + temp);         // (dy/dx)*(x-x1) + y1
       }
       else y = y1;  // pokud dx==0, tak vraci y1
       return y;
}
```

- **Výpočet v celočíselné aritmetice**

```
/// Provadi linearni interpolaci hodnoty y pro zadanou hodnotu x v zadanem intervalu.
/**
Doba vykonani: 2074 instrukcnich cyklu (vypnuta optimalizace -O0)

<PRE>
       y2     |_________________
              |                 |
       y      |___________      |
              |           |     |
       y1     |___        |     |
              |   |       |     |
              |   |       |     |
              ====|=======|===|====
              0   x1      x   x2

              y = [(x - x1)*(y2 - y1)/(x2 - x1)] + y1
</PRE>

Parametry x1,x2,y1,y2 museji byt v rozsahu <-1073741824,1073741823> nebo-li
<-(2^32)/4),(2^32)/4 - 1>, aby vypocet probehl spravne. Rozsahy parametru
nejsou v metode kontrolovany, spravnost parametru musi zajistit volajici.

Pokud je x1 rovno x2, tak metoda vraci hodnotu y1.
Pokud je x mimo rozsah x1, x2, je y extrapolovano.

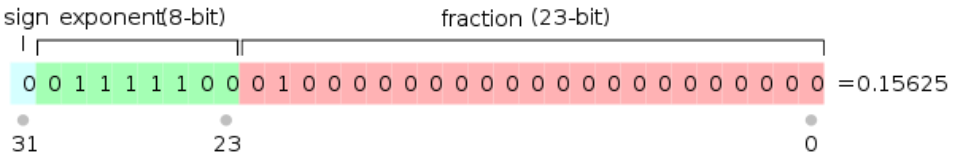
@param[in] x1 Leva hodnota intervalu ve vektoru X.
@param[in] x2 Prava hodnota intervalu ve vektoru X.
@param[in] y1 Leva hodnota intervalu ve vektoru Y.
@param[in] y2 Prava hodnota intervalu ve vektoru Y.
@param[in] x  Hodnota x, pro kterou se ma nalezt hodnota y.
@return       Hodnota y odpovidajici hodnote x.
*/
int32_t interp1D(int32_t x1, int32_t x2, int32_t y1, int32_t y2, int32_t x)
{
       int64_t dx, dy, itemp;
       int32_t y;

       dx = x2 - x1;
       if (dx != 0)  // zabraneni deleni nulou
       {
              dy = y2 - y1;
              itemp = x - x1;
              itemp *= dy;  // (x-x1)*dy
              itemp /= dx;  // ((x-x1)*dy)/dx
              y = (int32_t)((int64_t)y1 + itemp); // ((x-x1)*dy)/dx + y1
       }
       else y = y1;  // pokud dx==0, tak vraci y1
       return y;
}
```

  - Vlastnosti
    - Je důležité dodržet správný postup při násobení směrnicí přímky $\frac{dy}{dx} = \frac{(y_2 - y_1)}{(x_2 - x_1)}$.

- Takto je to dobře: $\frac{(x-x_1)\cdot(y_2-y_1)}{(x_2-x_1)}$
- Takto je to špatně: $(x-x_1)\cdot\frac{(y_2-y_1)}{(x_2-x_1)}$
- Nejprve musí proběhnout násobení v čitateli a teprve pak musí být mezivýsledek výsledek podělen jmenovatelem. Jinak dojde ke ztrátě přenosti, v extrémním případě vyjde 0, pokud bude *dy* < *dx*.
- Při výpočtu je $(x-x_1)\cdot(y_2-y_1)$ je třeba dávat pozor na přetečení mezivýsledku → proto je v příkladu použit datový typ `int64_t`, tedy 64-bit celé číslo se znaménkem. Násobení pak probíhá na 64-bitech. I přes to může dojít k přetečení mezivýsledku, pokud by uživatel využil plný rozsah parametrů předávaných do funkce. Z tohoto důvodu je v popisu funkce uživatel varován, že do funkce musí předávat parametry v rozsahu užším – parametry musí mít rozsah pouze `<INT32_MIN/2;INT32_MAX/2>`.

- **Porovnání obou zůsobů výpočtu**
  - **Výpočet ve float (32-bit single precision) aritmetice**
    - Na AVR je o trochu rychlejší než výpočet celočíselný, který využívá 64-bit celočíselné aritmetické operace.
    - Lze využít celý číslených rozsah vstupních aprametrů (do datového typu float lze uložit velmi velké číslo, až $3{,}4e^{38}$).
    - Ve výpočtu nehrozí přetečení.
    - Výpočet nebude tolik přesný jako při použití celočíselného výpočtu na `int64_t`, protože 32-bit datový typ `float` (jednoduchá přesnost) má jen 23-bit mantisu, ta udává přesnost čísla → cca 7 míst.



Obrázek 107 Čísla ve formátu float (single precision), IEEE 754

  - **Výpočet v celočíselné aritmetice**
    - Je třeba dát pozor na pořadí dílčích výpočtů.
    - Je třeba dát pozor na přetečení mezivýsledků → použití datového typu `int64_t` → časově náročné výpočty na 8-bit CPU.
    - Na AVR je výpočet o cca 6 % pomalejší než výpočet v reálné aritmetice.
    - Kompilátor musí podporovat datový typ `int64_t`.

- Omezení číselného rozsahu vstupních parametrů na polovinu, aby nedošlo k přetečení i na 64-bit.
- Výpočet bude zřejmě přesnější než při použití float aritmetiky s jednoduchou přesností (23-bit mantisa vs. 64-bit celé číslo).

- **Celý program**
  - Ukázka jak se funkce pro interpolaci používá.
  - Budeme vyčítat tlak [mBar] z převodní charakteristiky snímače tlaku v závislosti na napětí snímče [mV].

```
#include <avr/io.h>
#include <inttypes.h>

/// Provadi linearni interpolaci hodnoty y pro zadanou hodnotu x v zadanem intervalu.
/**
Pokud je x1 rovno x2, tak metoda vraci hodnotu y1.
Pokud je x mimo rozsah x1, x2, je y extrapolovano.

@param[in] x1   Leva hodnota intervalu ve vektoru X.
@param[in] x2   Prava hodnota intervalu ve vektoru X.
@param[in] y1   Leva hodnota intervalu ve vektoru Y.
@param[in] y2   Prava hodnota intervalu ve vektoru Y.
@param[in] x    Hodnota x, pro kterou se ma nalezt hodnota y.
@return         Hodnota y odpovidajici hodnote x.
*/
int32_t interp1D(int32_t x1, int32_t x2, int32_t y1, int32_t y2, int32_t x)
{
        int64_t dx, dy, itemp;
        int32_t y;

        dx = x2 - x1;
        if (dx != 0)    // zabraneni deleni nulou
        {
                dy = y2 - y1;
                itemp = x - x1;
                itemp *= dy;    // (x-x1)*dy
                itemp /= dx;    // ((x-x1)*dy)/dx
                y = (int32_t)(itemp + (int64_t)y1);     // ((x-x1)*dy)/dx + y1
        }
        else y = y1;    // pokud dx==0, tak vraci y1
        return y;
}

int main(void)
{
        // definice prevodni charakteristiky snimace: pressure_mBar = f(voltage_mV)
        int16_t voltage_mV[] = {200, 4700};             // vektor napeti [mV]
        int16_t press_mBar[] = {150, 1150};             // vektor tlaku [mBar]

        int16_t v_mV;           // merene napeti ze snimace [mV]
        int16_t p_mBar; // mereny tlak [mBar] - vysledek interpolace

        while(1)
        {
                // nejaky kod, ktery zmeri napeti na snimaci tlaku
                // v_mV = .....

                // prevod napeti snimace na tlak
                p_mBar = interp1D(voltage_mV[0],voltage_mV[1],press_mBar[0],press_mBar[1],v_mV);

                // nejaky dalsi kod, napr. vypis tlaku na LCD apod.
        }
}
```

- **Na Arduinu lze pro lineární interpolaci použít funkci map**
  - https://www.arduino.cc/reference/en/language/functions/math/map/
  - **Syntaxe:**

```
long map(long x, long in_min, long in_max, long out_min, long out_max)
{
   return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

  - **Význam parametrů**:

x:	the number to map.
in_min:	the lower bound of the value's current range.
in_max:	the upper bound of the value's current range.
out_min:	the lower bound of the value's target range.
out_max:	the upper bound of the value's target range.

  - **Příklad**:

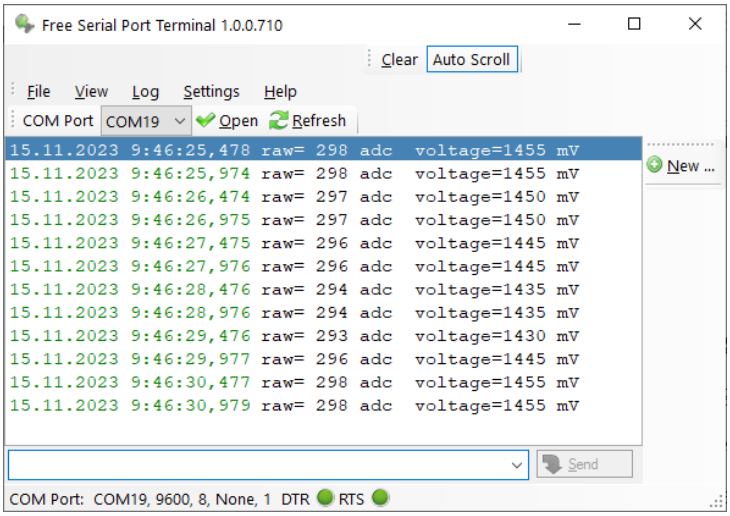
```
char text[33]; // retezec pro vypisy (32 znaku + zarazka NULL)

void setup()
{
       Serial.begin(9600);
}

void loop(void)
{
       // zmereni napeti pomoci interniho A/D prevodniku
       // cteni napeti na A0, vysledek prevodu je bezrozmerne cislo 0-1023
       int raw = analogRead(0);

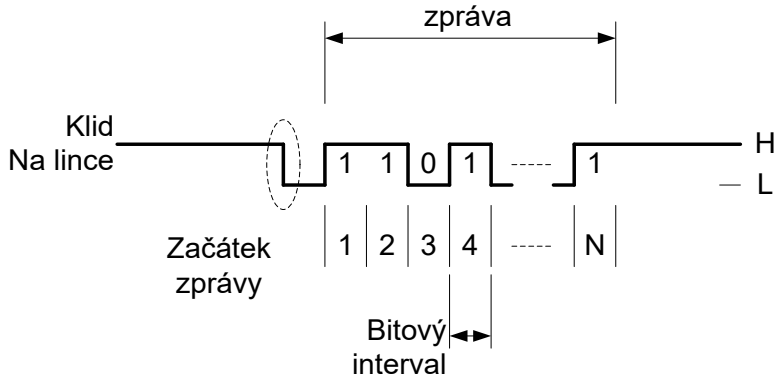
       // prevod na napeti v mV: 0-1024 -> 0-5000 mV
       int voltage = map(raw, 0, 1024, 0, 5000);

       // vypis hodnot pres USB
       snprintf(text,sizeof(text),"raw=%4d adc  voltage=%4d mV", raw, voltage);
       Serial.println(text);
       delay(500); // pauza 500 ms
}
```

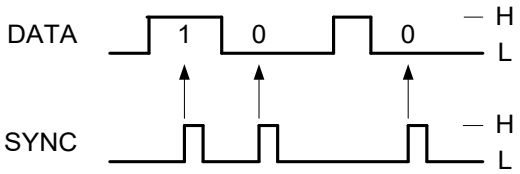


6 Sériový přenos dat

- Data se přenášejí postupně v čase bit po bitu.
- Úspora vodičů.
- Dle synchronizace
    - **asynchronní** – vysílač explicitně sdělí začátek a konec jednotlivých zpráv
        - není použit zvláštní vodič pro synchronizaci



    - **synchronní** – spolu s užitečnou informací jsou přenášeny i synchronizační informace:
        - zvláštním vodičem



        - smícháním synchronizační informace a užitečných dat
        - U synchronního přenosu dat lze obecně dosáhnout vyšších přenosových rychlostí než u asynchronní komunikace.

- Dle směru přenosu dat
    - **simplexní** – pouze jedním směrem
        - 1 datový vodič
    - **(plně) duplexní** – oběma směry zároveň
        - 2 datové vodiče (pro každý směr jeden)
    - **poloduplexní** – střídavě oběma směry s přepínáním směru komunikace
        - 1 datový vodič

7 Přístrojové sběrnice

- pro komunikaci součástek uvnitř jednoho přístroje – A/D, D/A převodníky, EEPROM, displeje, snímače, jednoúčelové IC atd…..
- nejsou určené pro přenos dat na dlouhé vzdálenosti
- obvykle komunikace mezi obvody na DPS
- nejznámější sběrnice: SPI, I2C, Microwire, Dallas
- více obvodů na jedné sběrnici – výrazná redukce počtu vodičů nutných pro připojení obvodu k MCU

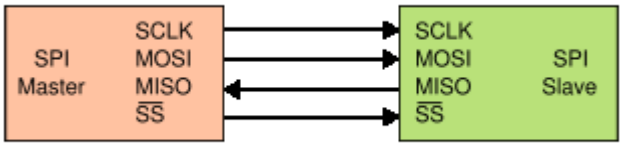


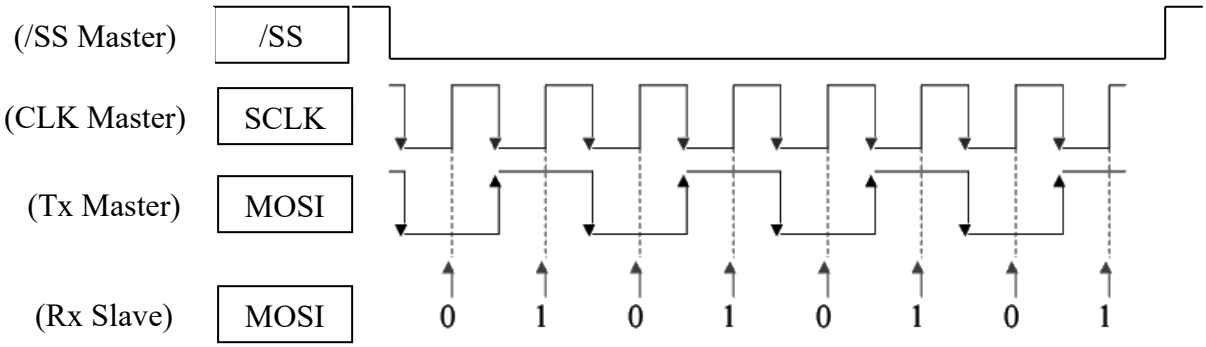
8 Sběrnice SPI (Serial Peripheral Interface)

- Umožňuje vysokorychlostní přenos dat (až desítky Mbps) mezi MCU a externími obvody nebo mezi dvěma MCU navzájem.
- SPI rozhraním je vybavena řada obovdů jako např.: externí EEPROM, A/D přev., D/A přev., budiče LED, LCD, hodiny reálného času, různé snímače apod.

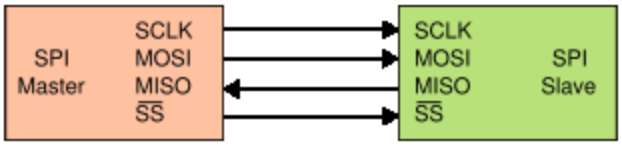
8.1 Vlastnosti SPI rozhraní

- Plně duplexní přenos, 3 vodičový synchronní přenos dat.
- Zařízení na sběrnici nemají adresu, zařízení jsou buď **Master** nebo **Slave**.
    - **Jedině Master může zahájit komunikaci**, Slave sám od sebe nemůže začít vysílat data.
    - **Master je zdrojem hodin CLK** (není CLK = nekomunikuje se).
    - Master aktivuje (vybere) daný Slave signálem **Slave Select (SS)** před zahájením komunikace stažením linky /SS do log. 0.
        - V jeden okamžik komunikuje Master jen s jedním Slave.

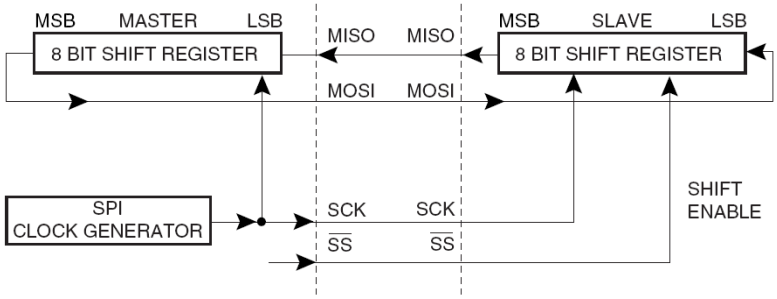




- Sběrnice je tvořena čtveřicí signálů:
    - **SCK** nebo **SCLK** (Serial Clock) - u Mastera je SCK výstupem, u Slave je SCK vstupem. Pouze Master generuje taktovací signál (hodiny) na SCK.
    - **MOSI** (Master Out Slave In) – přenos dat **ve směru Master → Slave**. Při zápisu do datového registru Mastera se začnou data vysouvat z posuvného registru přes tento pin a vstupují přes stejně označený pin do obvodu Slave.
    - **MISO** (Master In Slave Out) – přenos dat **ve směru Slave → Master**. U Mastera je to vstupní pin, u Slave výstupní pin.
    - **/SS** (Slave Select) – slouží k aktivaci (výběru) obvodu Slave (nastavením úrovně do log. 0) před zahájením komunikace. Tento vývod je také často nazýván /CS (Chip Select).



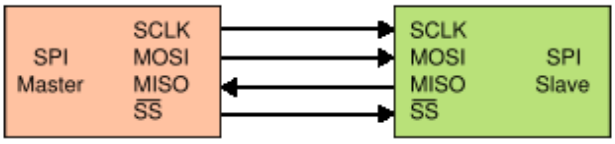
- Lze volit pořadí přenášených bitů (MSb až LSb nebo LSb až MSb).
- **Propojení obvodů Master a Slave na SPI sběrnici je následující:**
    - Master i Slave obsahují **posuvný registr** pro vysílání a příjem dat.



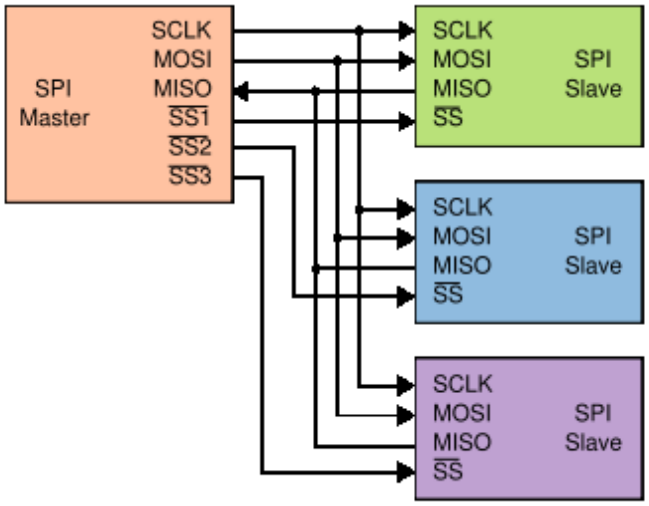
- **Při vysílání jsou data zároveň přijímána synchronně s CLK – společný Tx/Rx posuvný registr.**
- **Při přenosu dat se zároveň přesouvají data ve směru Master → Slave tak i Slave → Master. Dojde tedy vždy k vzájemné výměně dat mezi Master a Slave.**
- **Přenos dat zahajuje a časování na SCK generuje vždy Master. Slave sám nemůže začít přenášet data. Přenos dat ze Slave do Master je možný pouze, když Master posílá data do Slave.**
    - Pokud chce Master vyčíst data Slave, tak vysílá „libovolnou“ hodnotu jen za účelem generování taktovacího signálu (CLK) na pinu SCK, Slave přijmutá data na MOSI ignoruje a zároveň vysílá data na MISO do Mastera, Master přijímá data ze Slave.
- Délka přenášených dat není předepsána (8b až 32b). Délka přenášených dat je určena typem obvodu, se kterým MCU komunikuje.

8.2 Topologie sběrnice

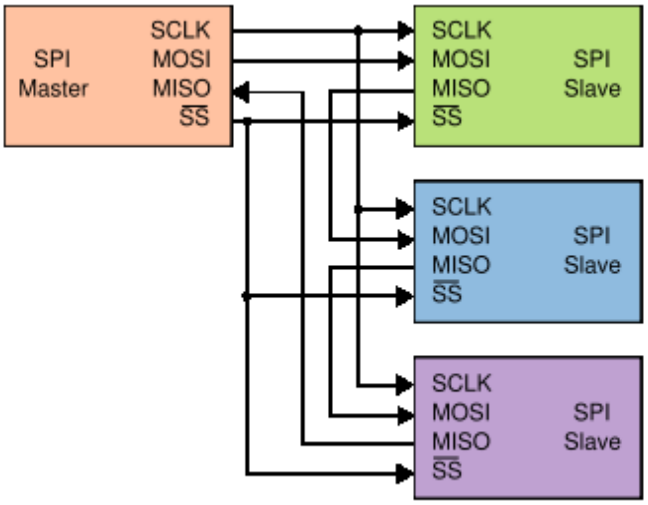
- 1 Master a 1 Slave



- 1 Master a více Slave (nezávislé propojení **Multi Slave**)



  - Není-li Slave vybrán (jeho /SS je v log.1), jsou jeho piny SCLK, MOSI a MISO ve stavu vysoké impedance, tudíž neovlivňují dění na sběrnici.
- 1 Master a více Slave (řetězové propojení **Daisy Chain**)
  - Obvody tento způsob propojení musí podporovat hardwarově !



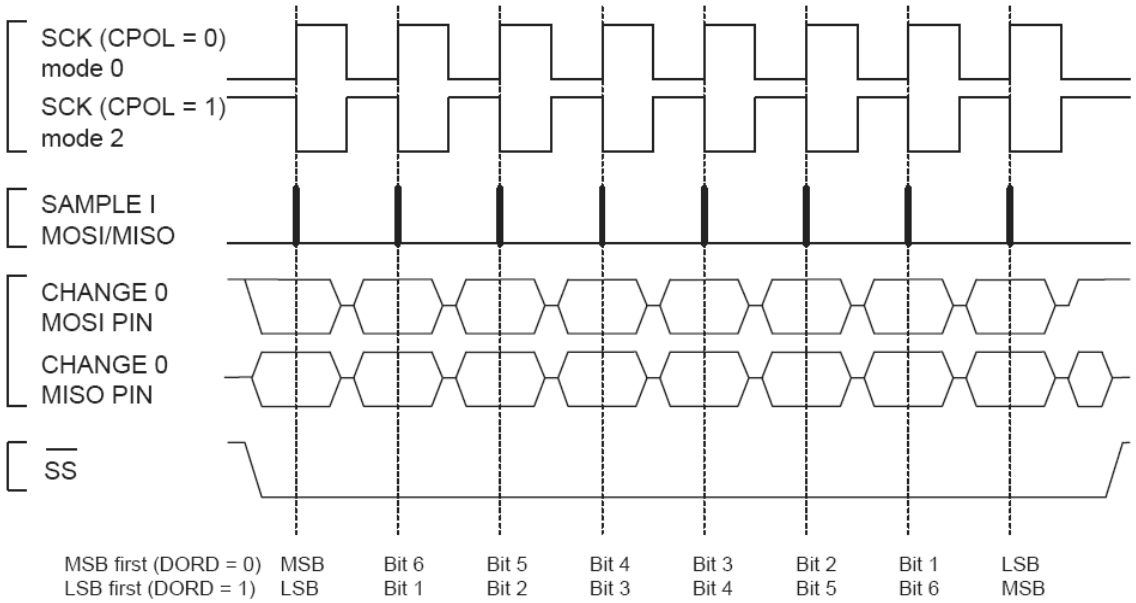
8.3 Přenos dat

- Postup:
  - Master nastaví pin /SS do 0 u vybraného Slave.
  - Zápis dat do posuvného registru Mastera, Master začne generovat CLK a přenášet data do Slave, zároveň se přenáší data ze Slave do Master.
  - Po skončení přenosu Master nastaví pin /SS do 1 a tím ukončí komunikaci s daným Slave.
- Celkem jsou možné **4 módy přenosu dat (mód 0 až 3)** v závislosti:
  - Na logické úrovni signálu SCK za klidu, když se nekomunikuje (určeno polaritou hodin CPOL).
  - Na kterou hranu signálu SCK jsou data platná (určeno fází hodin CPHA).
  - Nastavení příslušného módu v MCU se provede v závislosti na požadavcích připojené periferie (ne všechny Slave umí komunikovat ve stejném módu).
- Hodiny SCK začne řadič SPI generovat až v okamžiku, kdy zapíšeme nějaká data do posuvného registru.
- Pracovní módy SPI:

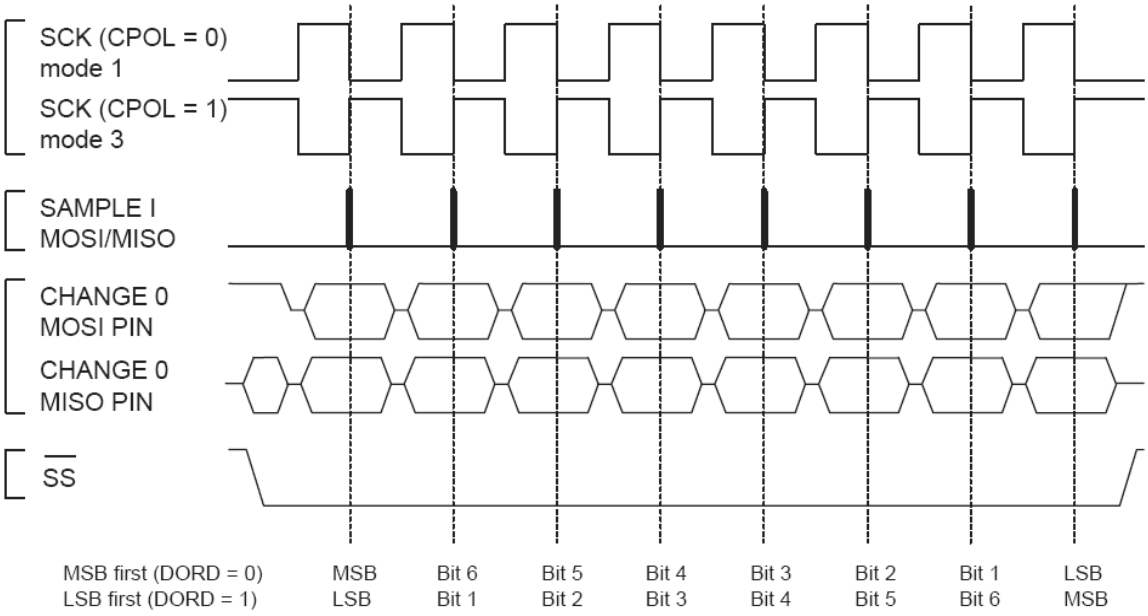
	Leading Edge	Trailing Edge	SPI Mode
CPOL = 0, CPHA = 0	Sample (Rising)	Setup (Falling)	0
CPOL = 0, CPHA = 1	Setup (Rising)	Sample (Falling)	1
CPOL = 1, CPHA = 0	Sample (Falling)	Setup (Rising)	2
CPOL = 1, CPHA = 1	Setup (Falling)	Sample (Rising)	3

Vysvětlivky: Sample (vzorkování logické úrovně dat), Setup (změna logické úrovně dat), Leading Edge (úvodní hrana), Trailing Edge (zadní hrana)

- **Módy 0 a 2** (CPHA = 0) – vzorkování dat na úvodní hranu hodin, změna dat na zadní hranu hodin, klidová logická úroveň hodin je 0 nebo 1 (dle módu)



- **Módy 1 a 3** (CPHA = 1) – vzorkování dat na zadní hranu hodin, změna dat na úvodní hranu hodin, klidová logická úroveň je 0 nebo 1 (dle módu)



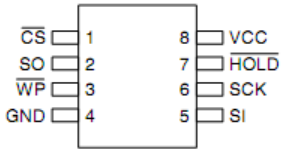
8.3.1 Příklad – čtení a zápis z/do sériové Flash EEPROM

- Typ AT25F5xxx
  - SPI rozhraní
  - velikost 512 kbit až 4 Mbit (64 kB až 512 kB), organizace po stránkách a sektorech
  - CLK až 50 MHz
  - SPI mód 0 nebo 3
  - Ochrana proti zápisu

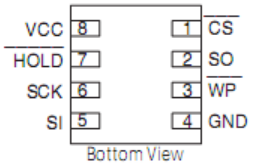
Table 0-1. Pin Configuration

Pin Name	Function
$\overline{CS}$	Chip Select
SCK	Serial Data Clock
SI	Serial Data Input
SO	Serial Data Output
GND	Ground
VCC	Power Supply
$\overline{WP}$	Write Protect
$\overline{HOLD}$	Suspends Serial Input

8-lead JEDEC SOIC



8-lead SAP



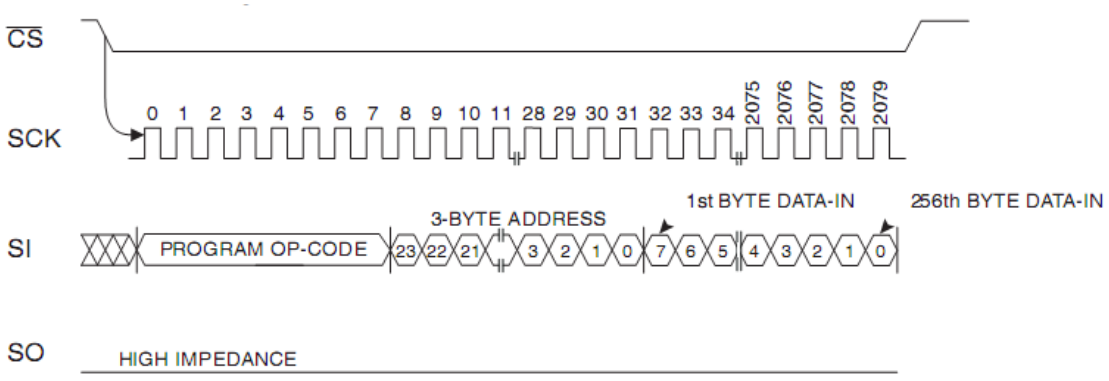
- **Komunikace s pamětí pomocí instrukcí**

Table 4-1. Instruction Set for the AT25FS010

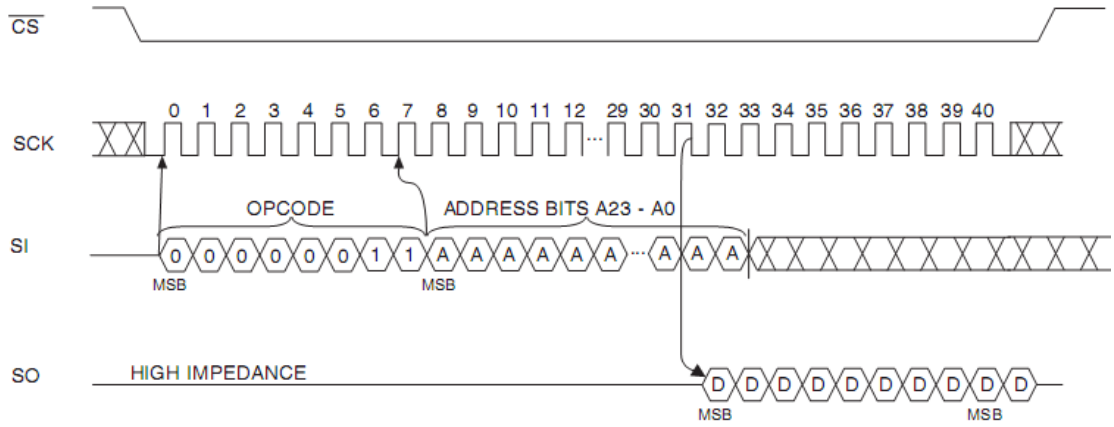
Instruction Name	One Byte OpCode		Operation
	Binary	Hex	
WREN	0000 X110	06	Set Write Enable Latch
WRDI	0000 X100	04	Reset Write Enable Latch
RDSR	0000 X101	05	Read Status Register
WRSR	0000 X001	01	Write Status Register
READ	0000 0011	03	Read Data from Memory Array
FAST READ	0000 1011	0B	Read Data from Memory Array (with dummy cycles)
PROGRAM	0000 X010	02	Program Data Into Memory Array
SECTOR ERASE (1)	0010 0000	20	Erase One 4kbyte Sector in Memory Array
	1101 0111	D7	
BLOCK ERASE(1)	0101 0010	52	Erase One 32kbyte Block in Memory Array
	1101 1000	D8	
CHIP ERASE(1)	0110 0000	60	Erase All Memory Array
	1100 0111	C7	
RDID(1)	1001 1111	9F	Read Manufacturer and Product ID
	1010 1011	AB	

  - Jako první se do EEPROM vždy přenáší příkaz (jednobajtová instrukce OpCode), která určuje operaci, která se bude následně vykonávat.

- **Zápis do EEPROM**



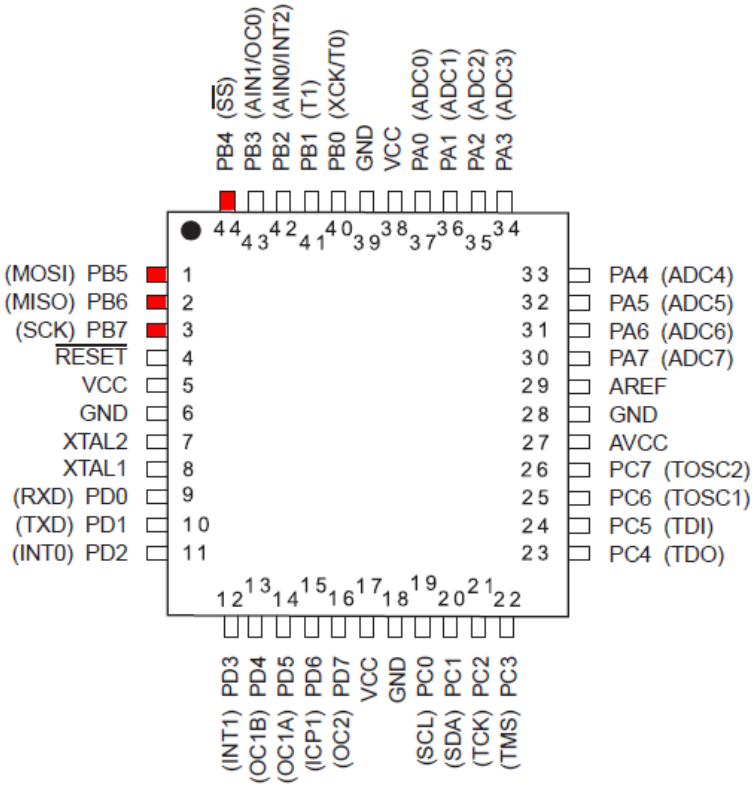
  - Postup při zápisu:
    - /CS = 0
    - Odeslání instrukce PROGRAM (1 bajt)
    - Odeslání adresy, na kterou se budou data zapisovat (3 bajty)
    - Odeslání dat (1 až 256 bajtů) – po zápisu každého bajtu automatická inkrementace adresy uvnitř EEPROM
    - /CS = 1
- **Čtení z EEPROM**



  - Postup při čtení:
    - /CS = 0
    - Odeslání instrukce READ (1 bajt)
    - Odeslání adresy, ze které se bude číst (3 bajty)
    - Odeslání 1 dummy bajtu (bajtu na jehož hodnotě nezáleží) – Master potřebuje generovat CLK a tím vyčíst data (1 bajt) z EEPROM
    - /CS = 1
    - V případě, že Master chce vyčíst více než 1 bajt dat, může po vyčtení 1. bajtu nechat /CS = 0 a generovat SCK signál tak dlouho, až vyčte potřebný počet bajtů (adresa se v EEPROM automaticky inkrementuje). Pak /CS = 1, čímž ukončíme přenos.

8.4 SPI u AVR

- Délka dat pevná (8 bit).
- Možno volit z 8-mi přenosových rychlostí $F_{osc}/128$ až $F_{osc}/2$ (takže max. rychlost SPI při F_{osc}= 16 MHz bude 8 MHz, ale doporučuje se max. $F_{osc}/4$ tedy 4 MHz).
- Možno volit pořadí bitů – MSB first nebo LSB first.
- MCU může být nakonfigurován jako Master nebo jako Slave.



Obrázek 108 Piny na MCU ATmega32 patřící rozhraní SPI (červeně označené)

8.4.1 Funkce pinu /SS na MCU AVR

- Pin /SS musí být nastaven buď jako výstup nebo jako vstup podle toho, zda má MCU na SPI sběrnici působit jako Master nebo jako Slave.

Overview of the $\overline{SS}$ pin functionality

Mode	$\overline{SS}$ Configuration	$\overline{SS}$ Pin-level	Description
Slave	Always Input	High	Slave deactivated (deselected)
		Low	Slave activated (selected)
Master	Input	High	Master activated (selected)
		Low	Master deactivated, switched to slave mode
	Output	High	Master activated (selected)
		Low	

Obrázek 109 Chování pinu /SS na MCU AVR

- **MCU jako Slave**
  - **Po nastvení SPI řadiče do Slave režimu je pin /SS automaticky SPI řadičem nastaven jako vstupní** a slouží pro aktivaci/deaktivaci komunikace – Master na sběrnici ovládá tento pin.
- **MCU jako Master**
  - **Programátor nastaví pin /SS výstupní**
    - V režimu Master není pin /SS SPI řadičem ovládán, programátor (program) ho musí ovládat ručně.
    - Pin /SS může, ale nemusí, být použit pro aktivaci Slave (stále mějte na paměti, že pin /SS v režimu SPI Master ovládá programátor, ne SPI řadič).
    - Pro aktivaci Slave z Mastera se může použít i kterýkoliv jiný digitální výstupní pin na MCU.
  - **Programátor nastaví pin /SS jako vstupní**
    - Je-li v log. 1, MCU zůstává v roli Mastera.
    - **Je-li uzemněn (log. 0), dojde k** automatickému **přepnutí SPI řadiče z režimu Master do režimu Slave** – bit **MSTR** v registru **SPCR** se nastaví do **0** a dále se nastaví příznak **SPIF** do **1**. Je-li povoleno přerušení od SPI, je vygenerováno.
    - Využití při meziprocesorové komunikaci (komunikaci mezi dvěma MCU, které se střídají v přenosu dat).

8.4.2 MCU jako Master

- Pin /SS na MCU nastavit jako výstupní (bezpodmínečně) a použít ho na ovládání některého ze Slave nebo jím ovládat něco jiného nebo pin na nic nepoužívat.
- Piny MOSI a SCK nastavit jako výstupní, pin MISO jako vstupní.
- Zvolit přenosovou rychlost (bity **SPR1** a **SPR0**), polaritu a fázi SCLK (bity **CPOL** a **CPHA**) – SPI Mode 0 až Mode 3. Bity jsou v registru **SPCR**.
- Přepnout do režimu Master (**MSTR = 1**) a povolit SPI (**SPE = 1**) v registru **SPCR**.
- Přenos je zahájen zápisem bajtu do SPI datového registru **SPDR**.
    - Master začne generovat hodinové pulsy na lince SCK a byte se přenáší přes linku MOSI z Mastera do Slave a současně přes linku MISO ze Slave do Mastera.
    - Po přesunutí jednoho byte se hodinový generátor zastaví a nastaví se příznak dokončeného přenosu **SPIF** v registru **SPSR** – bit lze testovat pollingem nebo událost obsloužit v přerušení.
    - Jestliže je povoleno přerušení (bit **SPIE=1** v registru **SPCR**), vygeneruje se.
    - Data současně přenesená ve směru Slave → Master se získají přečtením obsahu datového registru **SPDR** po dokončení přenosu (SPIF = 1).

8.4.3 MCU jako Slave

- Piny /SS, MOSI, MISO, SCK nastavit jako vstupní.
- Přepnout do režimu Slave (**MSTR = 0**) a povolit SPI (**SPE = 1**) v registru **SPCR**.
    - Po přepnutí do Slave režimu je pin /SS automaticky nakonfigurován jako vstupní.
- SPI interface ve Slave je neaktivní pokud je pin /SS v log. 1 (Master si nepřeje komunikovat).
    - Linka MISO ve Slave je v tomto případě automaticky ve stavu vysoké impedance, aby neovlivňovala komunikaci mezi Masterem a jiným Slave.
    - Předpřipravíme obsah datového registru **SPDR** pro přenos dat ze Slave do Master – **do registru SPDR zapíšeme byte**, který bude odeslán do Mastera až v okamžiku, kdy Master bude chtít komunikovat – až když Master nastaví pin /SS do 0 a začne generovat hodiny na lince SCK.
- Jakmile Master aktivuje /SS do log. 0 (chce komunikovat), začne Master generovat hodiny na SCK a v obou směrech se začnou přenášet data.
- Po přesunutí jednoho byte se nastaví příznak konce přenosu **SPIF.**
    - Přijmutý byte z Mastera získáme vyčtením registru **SPDR**.
    - Bit **SPIF** můžeme **testovat pollingem**, ale z důvodu zajištění minimální prodlevy je lepší použít **přerušení**, které se vyvolá po dokončeném přenosu (po nahození bitu **SPIF** do 1).
- Přerušení po dokončeném přenosu po SPI se povoluje bitem **SPIE** v registru **SPCR**.

8.5 Registry SPI v MCU AVR Atmega32

- Platí pro ATmega32
- **Registr SPCR (SPI Control Register)**

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **SPIE** (SPI Interrupt Enable) – povolení/zákaz přerušení od konce přenosu
  - **SPE** (SPI Enable) – povolení rozhraní SPI
  - **DORD** (Data Order) – určuje pořadí vysílaných bitů, nejdříve LSB (DORD = 1) nebo nejdříve MSB (DORD = 0)
  - **MSTR** (Master/Slave Select) – určuje režim Master (MSTR = 1) nebo Slave (MSTR = 0)
  - **CPOL** (Clock Polarity) – určuje polaritu hodin SCK, v klidu log.1 (CPOL = 1) nebo log. 0 (CPOL = 0)
  - **CPHA** (Clock Phase) – určuje na jakou fázi hodinového pulsu budou data platná
  - **SPR1 a SPR0** spolu s bitem SPI2X – výběr taktovacího kmitočtu SPI

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

- **Registr SPSR (SPI Status Register)**

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - **SPIF** (SPI Interrupt Flag) – příznak je nastaven po dokončení přenosu. Je-li povoleno přerušení, vygeneruje se. Příznak je nastaven ještě v případě, že v režimu Master je pin /SS nastaven jako vstupní a je shozen do log. 0. **Příznak je vynulován buď automaticky po vstupu do přerušovací rutiny nebo přečtením obsahu registru SPSR a následným přístupem k datovému registru SPDR.**
  - **WCOL** (Write Collision Flag) – příznak je nastaven, když je proveden zápis do datového registru SPDR při ještě nedokončeném přenosu. Příznak se vynuluje

ručně prvním přečtením obsahu registru SPSR a následným přístupem k datovému registru SPDR.

- **SPI2X** (Double SPI bit) – zdvojuje rychlost SPI, viz tabulka výše Pro rychlosti vyšší než Fosc / 4 není garantována funkčnost přenosu v režimu Slave.

- **Registr SPDR (SPI Data Register)**

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

  - Zápisem do registru se zahájí přenos. Čtením registru se přečtou příjmutá data z bufferu posuvného přijímacího registru.

- **Jeden vektor přerušení „SPI Transfer Complete“**

Table 18. Reset and Interrupt Vectors

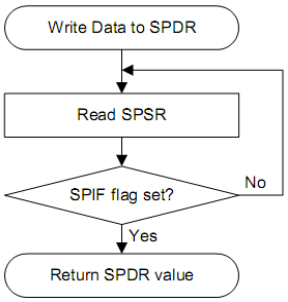
Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event
8	$00E	TIMER1 COMPA	Timer/Counter1 Compare Match A
9	$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

Obrázek 110 Vektor přerušení od SPI na Atmega32

8.6 Vysílání a příjem po SPI na AVR

8.6.1 Pomocí pollingu

- Periodicky kontrolujeme stav příznaku **SPIF** (odeslání/příjem dokončen).
- **MCU jako SPI Master**
  - Nejprve nastavit piny /SS, MOSI a SCK jako výstupní, MISO vstupní.
  - Zvolit přenosovou rychlost a polaritu a fázi SCLK (**CPOL** a **CPHA**).
  - Přepnout do režimu Master (**MSTR = 1**) a povolit SPI (**SPE = 1**) v registru **SPCR**.
  - **Vysílání/příjem (Master)**
    - Polling obvykle používáme, pokud se přenáší jen pár bajtů. Pak je čekání na odeslání dat krátké a neblokuje dlouho program.
      - Např. odeslání 1 B při F_{SPI} = 1 MHz trvá 8-bit/F_{SPI} = 8 μs.
    - Nejprve zapsat bajt do datového registru **SPDR**.
    - Čekat na odeslání neustálým kontrolováním příznaku **SPIF**.
    - Přečíst registr **SPDR** – v něm budou příjmutá data ze Slave a zároveň dojde k vynulování příznaku **SPIF**.



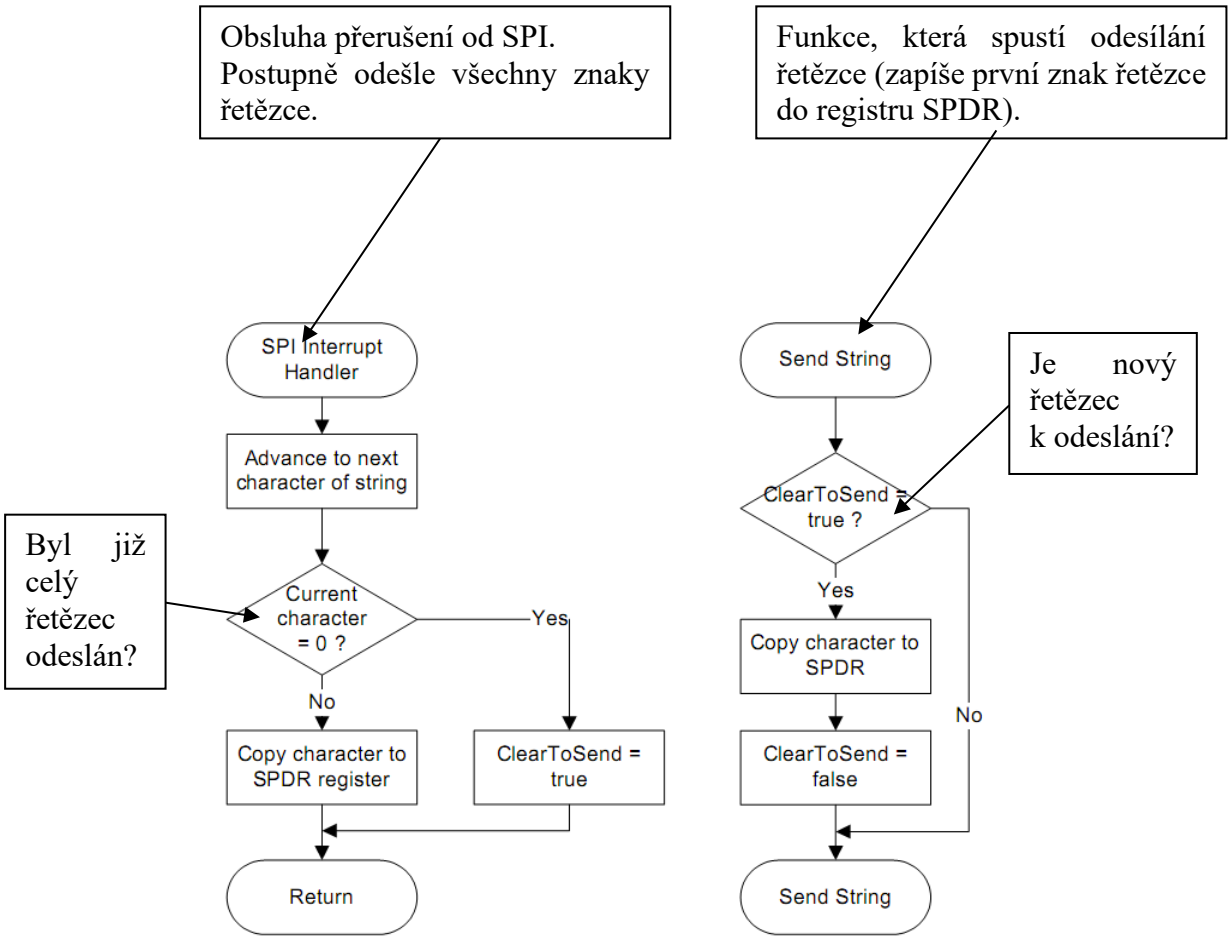
Obrázek 111 Vysílání po SPI jako Master (polling)

- **MCU jako Slave**
  - Nastavit piny /SS, MOSI, MISO, SCK jako vstupní.
  - Zvolit přenosovou rychlost a polaritu a fázi SCLK (**CPOL** a **CPHA**).
  - Přepnout do režimu Slave (**MSTR = 0**) a povolit SPI (**SPE = 1**) v registru **SPCR**.
  - **Příjem/vysílání (Slave)**
    - Pokud je /SS v log. 1, tak připravit byte pro odeslání do Mastera – zapsat ho do datového registru **SPDR** (k jeho odeslání nedojde dokud Master nezačne generovat hodiny na SCK).
    - Cyklicky kontrolovat příznak **SPIF**. Ten se ve Slave nastaví až po přenosu jednoho bajtu, což nastane až potom, co Master nastaví /SS do log. 0 a začne generovat SCK, čímž se začnou přenášet data oběma směry.

- Až bude **SPIF** == 1, tak:
  - Přečíst obsah **SPDR** – vyzvednutí přijmutého bajtu od Mastera, zároveň dojde k vynulování příznaku **SPIF**. Současně s příjmem došlo k odeslání předpřipraveného bajtu do Mastera.
  - Předpřipravit nový byte pro odeslání do Mastera (zapsat ho do registru SPDR).
- Opět cyklicky kontrolovat příznak **SPIF** – proces se opakuje.

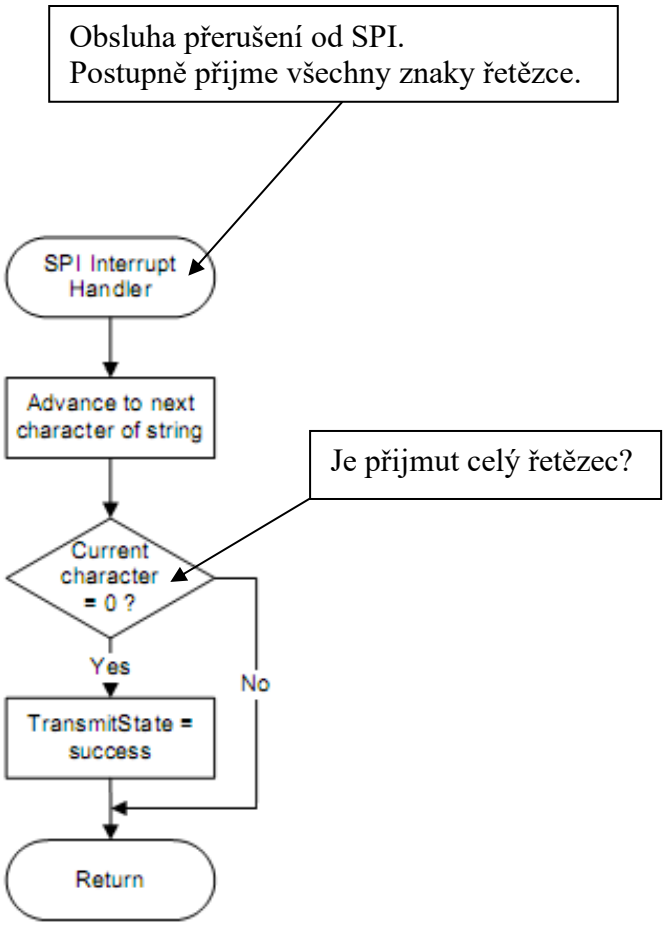
8.6.2 Pomocí přerušení

- **Konfigurace MCU jako SPI Master**
  - Stejná jako u pollingu + navíc povolit přerušení od SPI (bit **SPIE = 1**).
- **Vysílání/příjem (Master)**
  - Přerušení při vysílání použít při nízkých rychlostech SPI (F_{osc}/64 a méně), aby procesor zbytečně dlouho nečekal na odvysílání dat.
  - **V přerušovací rutině implementujeme kód, který zajistí odvysílání určitý počet bajtů.**
    - Nutný předpoklad je mít v programu vytvořen vysílací buffer, do kterého se budou ukládat hodnoty pro odeslání, a dále počítadlo zbývajících bajtů k odeslání.
    - Nebo v případě odesílání textových zpráv (řetězců) se využije znak zakončení řetězce (NULL). Jednotlivé znaky z řetězce se odesílají tak dlouho, až se narazí na zakončovací znak NULL. Tento způsob je zobrazen na následujícím obrázku.



Obrázek 112 Princip odeslání řetězce po SPI (s využitím přerušení)

- **Konfigurace MCU jako SPI Slave**
  - Stejná jako u pollingu + navíc povolit přerušení od SPI (bit **SPIE = 1**).
- **Příjem/vysílání (Slave)**
  - V rutinně přerušení „SPI přenos dokončen“ implementovat kód, který bude jednotlivé přijmuté bajty ukládat do programově vytvořeného přijímacího bufferu.
  - Po příjmu všech bajtů informovat hlavní program o úspěšném příjmu – hl. program dále zpracuje přijmutá data.
  - **Jak zjistit, že přišly všechny bajty?**
    - Buď vím dopředu kolik jich má přijít - pak stačí nastavit počítadlo na tuto hodnotu a při každé další obsluze přerušení počítadlo zmenšit o 1 a kontrolovat počítadlo na 0.
    - Nebo nevím kolik má přijít dat a v tom případě čekám na dohodnutý ukončovací znak – viz následující obrázek, kde je řešen příjem řetězce (ukončovací znak řetězce je NULL).



Obrázek 113 Princip příjmu dat po SPI v režimu Slave (s využitím přerušení)

8.7 SPI a kompilátor AVR-GCC

- Knihovna AVR-libc neobsahuje funkce pro obsluhu SPI rozhraní → vytvoříme si vlastní funkce pro obsluhu SPI.
- Obsluha SPI v režimu Master je poměrně jednoduchá, spočívá ve dvou krocích:
  - Nastavení řadiče SPI (mód, rychlost komunikace, Master).
  - Spuštění vysílání.
- **Kód (pro překladač AVR-GCC):**
  - Určen pro MCU AVR Atmega32.
  - Při odesílání bytu po SPI se používá polling, funkce pro odeslání bajtu je blokující.

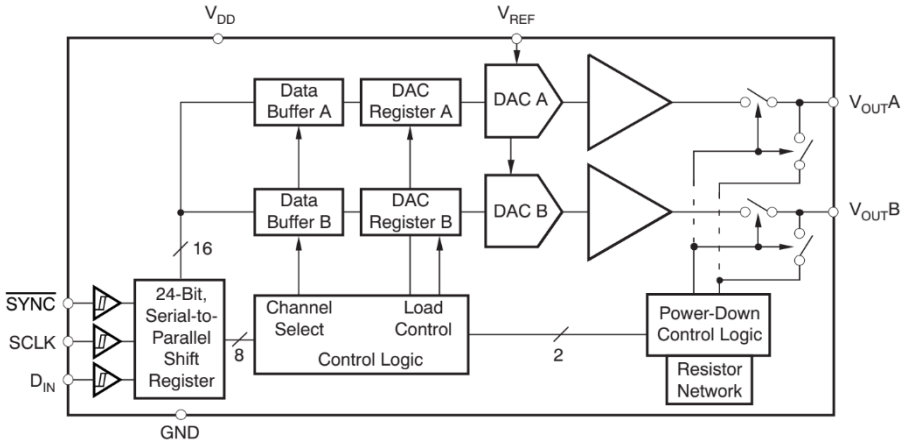
```
// Inicializace SPI rozhrani
void spiMasterInit(void)
{
	// Piny MOSI a SCK vystupni, pin MISO vstupni
	DDRB |= (1<<PB5)|(1<<PB7);
	DDRB &= ~(1<<PB6);

	// SPI Type: Master
	// SPI Clock Rate: 2*Fcpu/4
	// SPI Mode: 1
	// SPI Data Order: MSB First
	SPCR = (1<<SPE)|(1<<MSTR)|(1<<CPHA);
	SPSR |= (1<<SPI2X);
}

// Odesle byte po SPI
/*
Funkce je blokujici.
parametry:	data - byte k odeslani
vraci:		prijmuty byte
*/
unsigned char spiMasterSend(unsigned char data)
{
	SPDR = data;  // spusteni vysilani
	while(!(SPSR & (1<<SPIF)));  // cekame na dokonceni odeslani
	return SPDR;  // vracime prijmuty byte
}
```

8.7.1 Komunikace s 16-bit D/A převodníkem DAC8552 po SPI

- DAC8552 je dvoukanálový 16-bit D/A převodník.
- Rychlost D/A převodu je určena taktovacím signálem SCK na SPI sběrnici (max. 30 MHz).
- **Struktura DAC8552**



The input coding for each device is unipolar straight binary, so the ideal output voltage is given by:

$$V_{OUT}\,A,B = V_{REF} \times \frac{D}{65536} \qquad (1)$$

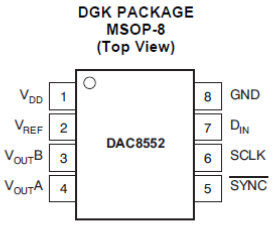
Where:

D = decimal equivalent of the binary code that is loaded to the DAC register; it can range from 0 to 65535.

V_{OUT}A,B refers to channel A or B.

- **Zapojení vývodů**

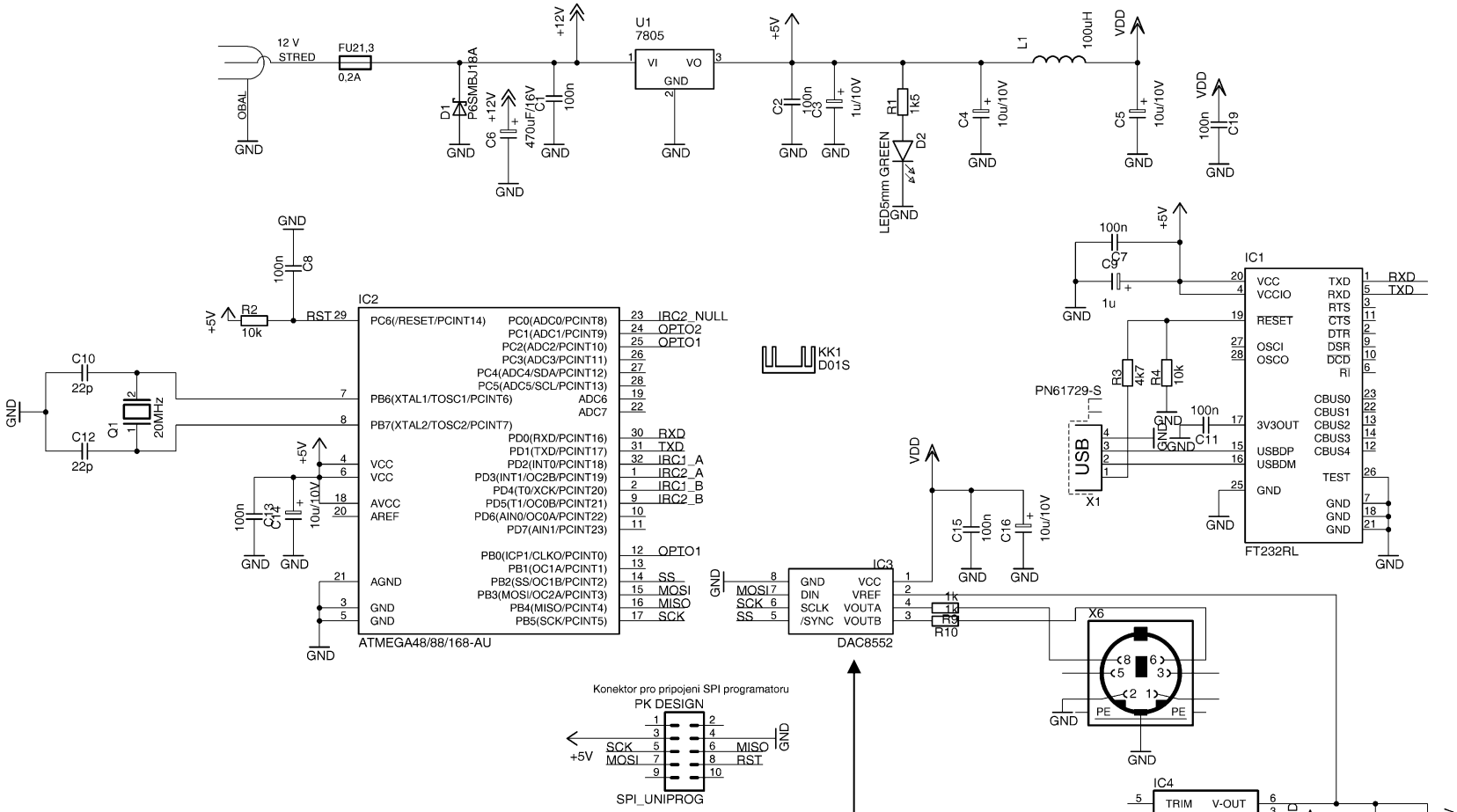
PIN CONFIGURATION



PIN DESCRIPTIONS

	PIN	NAME	FUNCTION
	1	V_{DD}	Power supply input, 2.7V to 5.5V
	2	V_{REF}	Reference voltage input
	3	$V_{OUT}B$	Analog output voltage from DAC B
	4	$V_{OUT}A$	Analog output voltage from DAC A
/SS	5	$\overline{SYNC}$	Level triggered $\overline{SYNC}$ input (active LOW). This is the frame synchronization signal for the input data. When $\overline{SYNC}$ goes LOW, it enables the input shift register and data is transferred on the falling edges of SCLK. The action specified by the 8-bit control byte and 16-bit data word is executed following the 24th falling SCLK clock edge (unless $\overline{SYNC}$ is taken HIGH before this edge, in which case the rising edge of $\overline{SYNC}$ acts as an interrupt and the write sequence is ignored by the DAC8552). Schmitt-Trigger logic input.
	6	SCLK	Serial Clock Input. Data can be transferred at rates up to 30MHz at 5V. Schmitt-Trigger logic input.
MOSI)	7	D_{IN}	Serial Data Input. Data is clocked into the 24-bit input shift register on the falling edge of the serial clock input. Schmitt-Trigger logic input.
	8	GND	Ground reference point for all circuitry on the part.

- **Zapojení v aplikaci**



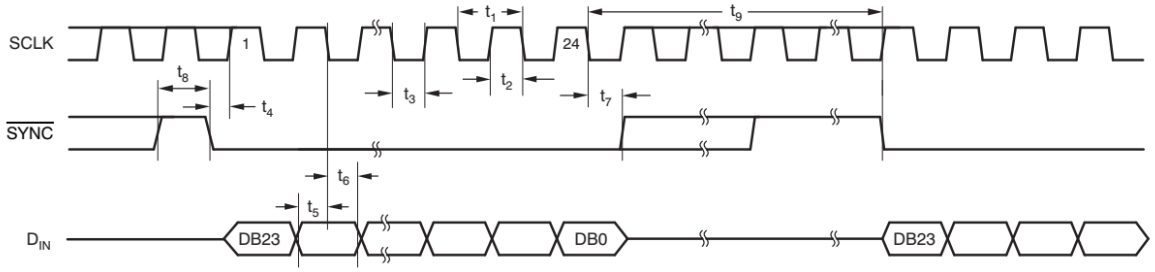
16-bit D/A převodník

SPI rozhraní na DAC8552 nemá pin MISO, protože D/A převodník nepotřebuje do MCU nic posílat, jedná se o jednosměrnou komunikaci MCU → D/A převodník přes pin MOSI.

přesná napěťová reference 4,096 V

- **Komunikace s DAC8552**
  - Přenáší se vždy 24 bitů (3 B).
  - 1. byte je příkaz (výběr kanálu, zap/vyp režimu s nízkou spotřebou).
  - 2. a 3. byte je 16-bit číslo (0-65535), které se zapíše do D/A převodníku a podle něj se na výstupu A nebo B objeví patřičné napětí.
  - SPI mode 1.
  - Data se přenášejí v pořadí MSB first.
  - Max. komunikační rychlost je 30 MHz.

SERIAL WRITE OPERATION



DB23											DB12
0	0	LDB	LDA	X	Buffer Select	PD1	PD0	D15	D14	D13	D12
DB11											**DB0**
D11	D10	D9	D8	D7	D6	D5	D5	D3	D2	D1	D0

Figure 43. DAC8552 Data Input Register Format

Table 1. Control Matrix

D23	D22	D21	D20	D19	D18	D17	D16	D15	D14	D13–D0	
Reserved	Reserved	Load B	Load A	Don't Care	Buffer Select	PD1	PD0	MSB	MSB-1	MSB-2... LSB	
(Always Write 0)					0 = A, 1 = B						DESCRIPTION
0	0	0	0	X	#	0	0	Data			WR Buffer # w/Data
0	0	0	0	X	#	See Table 2		X			WR Buffer # w/Power-down Command
0	0	0	1	X	#	0	0	Data			WR Buffer # w/Data and Load DAC A
0	0	0	1	X	0	See Table 2		X			WR Buffer A w/Power-Down Command and LOAD DAC A (DAC A Powered Down)
0	0	0	1	X	1	See Table 2		X			WR Buffer B w/Power-Down Command and LOAD DAC A
0	0	1	0	X	#	0	0	Data			WR Buffer # w/Data and Load DAC B
0	0	1	0	X	0	See Table 2		X			WR Buffer A w/Power-Down Command and LOAD DAC B
0	0	1	0	X	1	See Table 2		X			WR Buffer B w/Power-Down Command and LOAD DAC B (DAC B Powered Down)
0	0	1	1	X	#	0	0	Data			WR Buffer # w/Data and Load DACs A and B
0	0	1	1	X	0	See Table 2		X			WR Buffer A w/Power-Down Command and Load DACs A and B (DAC A Powered Down)
0	0	1	1	X	1	See Table 2		X			WR Buffer B w/Power-Down Command and Load DACs A and B (DAC B Powered Down)

Table 2. Power-Down Commands

D17	D16	
PD1	PD0	OUTPUT IMPEDANCE POWER DOWN COMMANDS
0	1	1kΩ
1	0	100kΩ
1	1	High Impedance

- **Kód pro komunikaci s DAC8552 (pro překladač AVR-GCC, ovládání SPI přes registry):**
  - Určen pro MCU AVR Atmega88.
  - F_{CPU} = 20 MHz
  - Ovládání SPI přes registry v MCU.

```
#include <avr/io.h>
#define F_CPU 20000000      // frekvence CPU [MHz]

// definice pripojeni signalu na piny portu
#define SS_PIN 2      // pin Slave Select

// ovladani pinu Slave Select - aktivace prenosu dat do DAC
#define SS_PIN_LOW    (PORTB &= ~(1<<SS_PIN))
#define SS_PIN_HIGH (PORTB |= (1<<SS_PIN))

// deklarace funkci
void spiMasterInit(void);
unsigned char spiMasterSend(unsigned char data);
void WriteToDAC(unsigned char channel, unsigned int DA_counts);


// Inicializace SPI rozhrani
void spiMasterInit(void)
{
       // Piny /SS,  MOSI, SCK vystupni, pin MISO vstupni
       DDRB |= (1<<SS_PIN)|(1<<PB3)|(1<<PB5);
       DDRB &= ~(1<<PB4);

       // SPI Type: Master
       // SPI Clock Rate: 2*5000,000 kHz
       // SPI Mode: 1
       // SPI Data Order: MSB First
       SPCR = (1<<SPE)|(1<<MSTR)|(1<<CPHA);
       SPSR |= (1<<SPI2X);
}

/*
Odesle byte po SPI.
Funkce je blokujici.
parametry:    data - byte k odeslani
vraci:        prijmuty byte
*/
unsigned char spiMasterSend(unsigned char data)
{
       SPDR = data;  // spusteni vysilani
       while(!(SPSR & (1<<SPIF)));  // cekame na dokonceni odeslani
       return SPDR;  // vracime prijmuty byte
       // priznakovy bit SPIF se smaze (nastavi do 0) automaticky po precteni SPDR
}
```

```
/*
Zapise novou 16-bit RAW hodnotu do DAC8552.
Parametry:    channel - cislo kanalu na DAC (0 – kanal A, 1 – kanal B)
              DA_counts – 16-bit RAW hodnota na vybranem kanalu v rozsahu 0 - 65535
Vraci:        nic
*/
void WriteToDAC(unsigned char channel, unsigned short DA_counts)
{
	unsigned char command = 0;

	// vytvoreni command bajtu podle cisla kanalu
	if (channel == 0)
		command = (1<<4);                  // LDA = 1, Buffer Select = A
	else if (channel == 1)
		command = (1<<5) | (1<<2);         // LDB = 1, Buffer Select = B


	// Odeslani dat do D/A prevodniku
	if (channel <= 1)
	{
		SS_PIN_LOW;                        // Slave Select do log. 0
		spiMasterSend(command);            // odeslani prikazu
		spiMasterSend(DA_counts >> 8);     // odeslani MSB dat
		spiMasterSend(DA_counts & 0xFF);   // odeslani LSB dat
		SS_PIN_HIGH;                       // Slave Select do log. 1
	}
}


int main(void)
{
	unsigned short output = 0;

	spiMasterInit();     // inicializace SPI

	// Nastav vychozi uroven napeti na obou vystupech D/A prevodniku na 0 V
	WriteToDAC(0,0);
	WriteToDAC(1,0);

	while(1)
	{
		// Postupne zvysovani napeti na obou vystupech D/A prevodniku a po
		// preteceni na 0 V stale dokola.
		WriteToDAC(0,output);
		WriteToDAC(1,output*2);
		_delay_us(100);  // aby se napeti nemenila prilis rychle
	}
}
```

8.8 SPI a Arduino

- Pro komfortní obsluhu sběrnice SPI je k dispozici knihovna **SPI**.
  - https://docs.arduino.cc/language-reference/en/functions/communication/SPI/
  - https://docs.arduino.cc/learn/communication/spi/
  - Na začátek programu je nutné vložit její hlavičkový soubor:
    ```
    #include <SPI.h>
    ```
  - Pak je možné použít globální objekt **SPI** pro komunikaci v režimu Master, např. odeslání bajtu s hodnotou `0x7F` a současný příjem jednoho bajtu `data`:
    ```
    SPI.begin();     // nastaví piny MOSI, SCK, SS výstupní, MISO vstupní
    SPI.beginTransaction(SPISettings(8000000, MSBFIRST,
    SPI_MODE1));     // nastavení komunikačních parametrů SPI řadiče
    digitalWrite(10,LOW);   // aktivace Slave (pin 10 je Chip Select)
    uint8_t data = SPI.transfer(0x7F);   // odeslání bajtu 0x7F a
    současný příjem bajtu data
    digitalWrite(10,HIGH); // deaktivace Slave (pin 10 je Chip Select)
    SPI.endTransaction();   // konec komunikace
    ```
  - Knihovna SPI používá při přenosu polling a umožňuje provoz pouze v režimu SPI Master, ne SPI Slave.
- Seznam Arduino funkcí pro obsluhu komunikace na SPI sběrnici:

Functions

- SPISettings
- begin()
- beginTransaction()
- endTransaction()
- end()
- setBitOrder()
- setClockDivider()
- setDataMode()
- transfer()
- usingInterrupt()

https://docs.arduino.cc/language-reference/en/functions/communication/SPI/

Zastaralé funkce, místo nich se má volat:

```
SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE1));
```

Kde parametry předávané do konstruktoru objektu třídy `SPISettings` nesoucí nastavení komunikace závisejí na konkrétním Slave.

- Jak používat objekt SPI pro komunikaci
  - **Inicializace:**
    - SPI.begin();
    - Pouze nastaví piny MOSI, SCK, /SS výstupní, MISO vstupní.
    - Stačí zavolat jednou, obvykle v `setup()`.
    - Pokud budeme používat SPI v rutině pro obsluhu přerušení (ISR) nějaké periferie, voláme:
      - SPI.begin();
      - SPI.usingInterrupt(255);
      - Tím se objektu SPI umožní zamezit konfliktu, když by chtěl náš kód v ISR používat SPI v okamžiku, kdy je SPI již používáno jinou částí programu mimo přerušení.
  - **Vysílání/příjem:**
    - `SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE1)); // nastavení komunikačních parametrů SPI řadiče`
    - `digitalWrite(10,LOW); // aktivace Slave (pin 10 je Chip Select)`
    - `uint8_t data = SPI.transfer(0x7F); // odeslání bajtu 0x7F a současný příjem bajtu data`
    - `digitalWrite(10,HIGH); // deaktivace Slave (pin 10 je Chip Select)`
    - `SPI.endTransaction(); // konec komunikace`
  - **Každé vysílání/příjem by mělo být ohraničené voláním funkcí:**
    - `SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE1));`
      - Nastavuje parametry komunikace:
        - Frekvence [Hz], nastaví se nejbližší nižší frekvence. U AVR je max. 8 MHz (16 MHz krystal). Nastavit dle toho, jakou frekvenci komunikace zvládá Slave.
        - Pořadí přenášených bitů v každém bajtu. `MSBFIRST` nebo `LSBFIRST`.
        - Polarita hodin SCK v klidovém stavu a typ hrany pro vzorkování bitů: `SPI_MODE0, SPI_MODE1, SPI_MODE2, SPI_MODE3`
      - Zabraňuje současnému použití SPI v různých částech programu v jeden okamžik (řeší přístup ke sdílenému HW):
        - Zakazuje globálně přerušení až do zavolání `SPI.endTransaction()`.
        - Aby jiná část programu nemohla v přerušení použít SPI během právě aktivní komunikace.

- `SPI.endTransaction();`
  - Uvolňuje objekt SPI pro použití v ostatních částech programu.
    - Opět povoluje globální přerušení, pokud bylo před `SPI.beginTransaction()` povoleno.
  - Nemění nastavení komunikace.

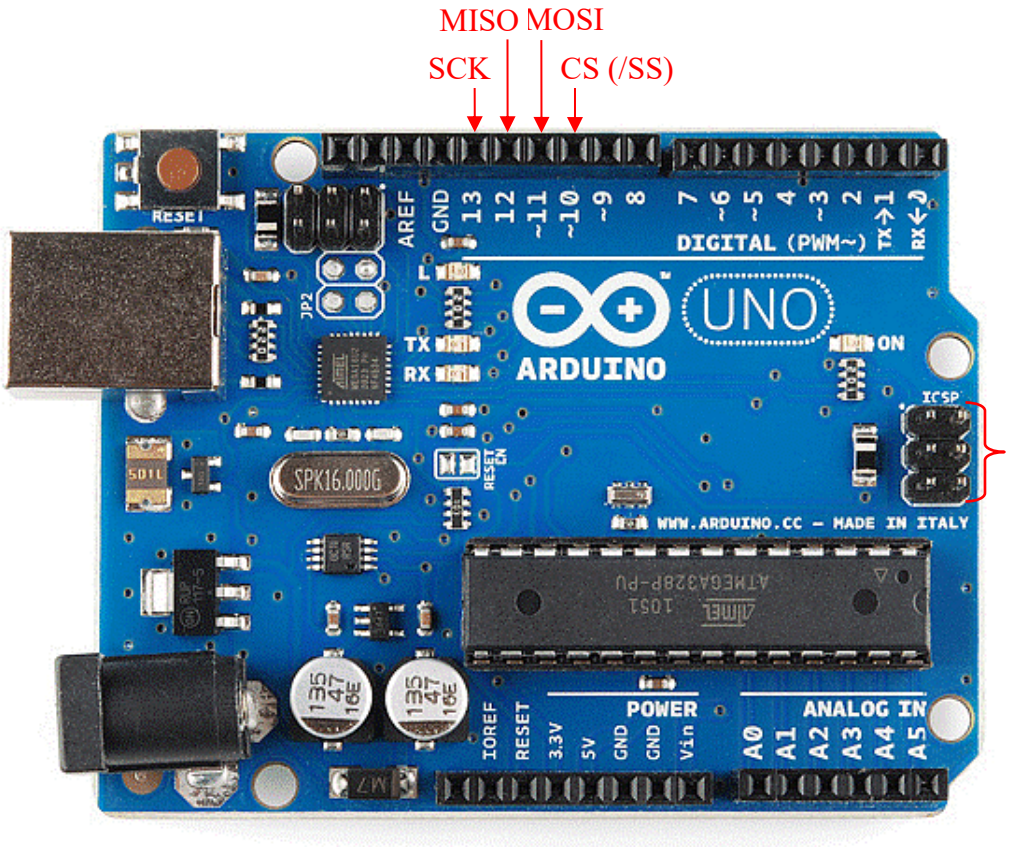
- Arduino = různé typy základních desek a použitých MCU.
  - Mapování linek SPI závisí na konkrétním typu Arduina.

Tabulka 27 Mapování pinů SPI rozhraní na Arduinu

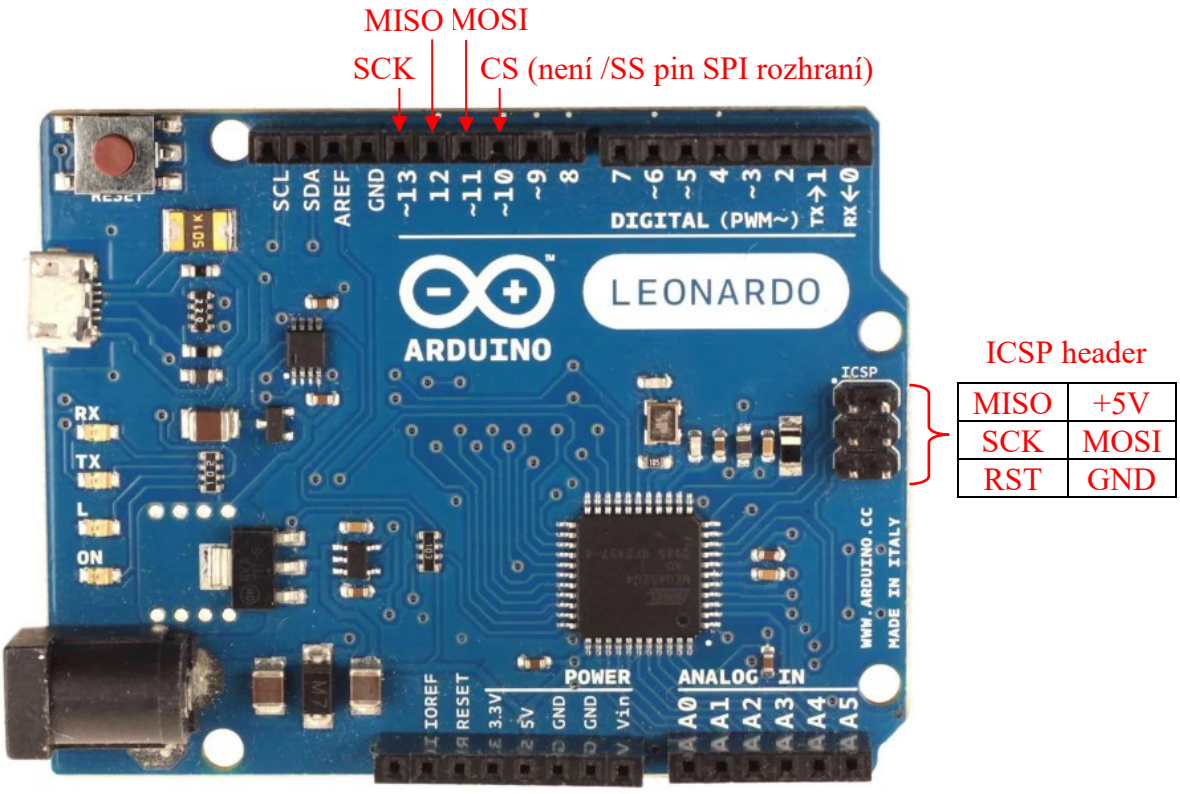
Boards	Default SPI Pins	Additional SPI Pins	Notes
UNO R3, UNO R3 SMD, UNO WiFi Rev2, UNO Mini LE	10(CS), 11(COPI), 12(CIPO), 13(SCK)		SPI pins available on ICSP header
UNO R4 Minima, UNO R4 WiFi	10(CS), 11(COPI), 12(CIPO), 13(SCK)		SPI pins available on ICSP header
Leonardo, Yún Rev2, Zero	10(CS), 11(COPI), 12(CIPO), 13(SCK)		SPI pins available on ICSP header
Micro	14(CIPO), 15(SCK), 16(COPI)		
Nano boards	11(COPI), 12(CIPO), 13(SCK)		
MKR boards	8(COPI), 9(SCK), 10(CIPO)		
Due	74(CIPO), 75(MOSI), 76(SCK)	SPI pins available on dedicated SPI header	
GIGA R1 WiFi	89(CIPO), 90(COPI), 91(SCK)	12(CIPO), 11(COPI), 13(SCK), 10(CS)	Note that pin 89, 90, 91 are located on the SPI header
Mega 2560 Rev3	50(CIPO), 51(COPI), 52(SCK), 53(CS)		SPI pins available on ICSP header

Poznámka: COPI = MOSI, CIPO = MISO

Pozor, na Arduino Leonardo není /SS pin SPI rozhraní vyveden na dutinkovou lištu s Dxx piny, ani na žádnou jinou lištu, není tedy pro uživatele přístupný. Pin /SS (PB0) je u Leonarda pevně připojen k LED RX indikující příjem dat na USB. Pin D10 (CS) uvedený výše v tabulce je u Leonarda standardní I/O digitální pin, který není nijak navázaný na SPI rozhraní. Arduino Leonardo tak není možné provozovat v režimu SPI Slave, ale pouze v režimu SPI Master. U Arduino UNO i MEGA2560 je pin CS uvedený v tabulce současně pinem /SS SPI rozhraní a je vyveden na dutinkovou lištu jako pin D10 (UNO) nebo D53 (MEGA2560). Hypoteticky je tedy možné UNO i MEGA2560 provozovat i jako SPI Slave.



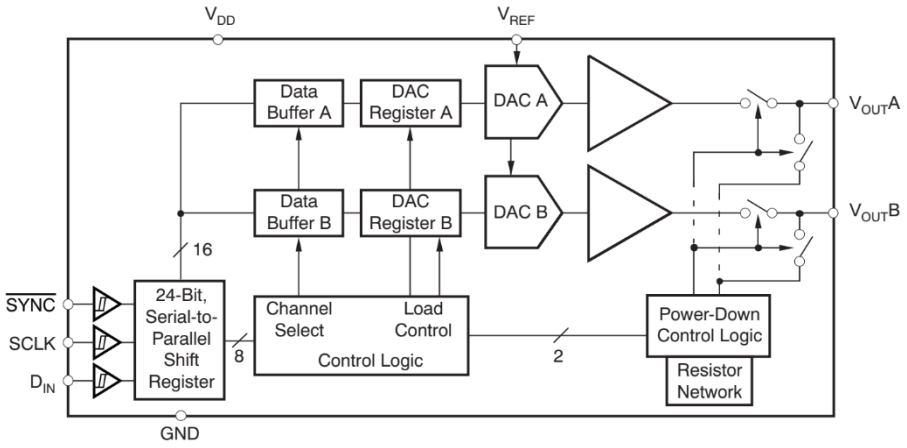
Obrázek 114 Piny SPI rozhraní na Arduino UNO



Obrázek 115 Piny SPI rozhraní na Arduino Leonardo

8.8.1 Komunikace s 16-bit D/A převodníkem DAC8552 po SPI

- DAC8552 je dvoukanálový 16-bit D/A převodník.
- Rychlost D/A převodu je určena taktovacím signálem SCK na SPI sběrnici (max. 30 MHz).
- **Struktura DAC8552**



The input coding for each device is unipolar straight binary, so the ideal output voltage is given by:

$$V_{OUT}\,A,B = V_{REF} \times \frac{D}{65536} \qquad (1)$$

Where:

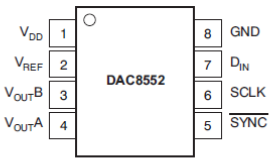
D = decimal equivalent of the binary code that is loaded to the DAC register; it can range from 0 to 65535.

V_{OUT}A,B refers to channel A or B.

- **Zapojení vývodů**

PIN CONFIGURATION

DGK PACKAGE
MSOP-8
(Top View)

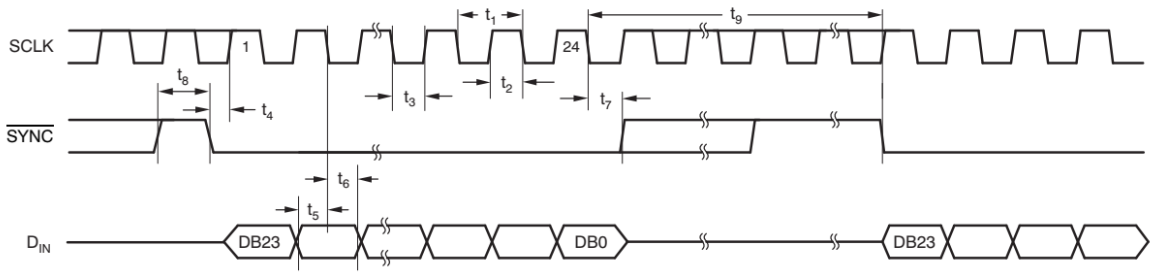


PIN DESCRIPTIONS

	PIN	NAME	FUNCTION
	1	V_{DD}	Power supply input, 2.7V to 5.5V
	2	V_{REF}	Reference voltage input
	3	V_{OUT}B	Analog output voltage from DAC B
	4	V_{OUT}A	Analog output voltage from DAC A
/SS	5	$\overline{SYNC}$	Level triggered $\overline{SYNC}$ input (active LOW). This is the frame synchronization signal for the input data. When $\overline{SYNC}$ goes LOW, it enables the input shift register and data is transferred on the falling edges of SCLK. The action specified by the 8-bit control byte and 16-bit data word is executed following the 24th falling SCLK clock edge (unless $\overline{SYNC}$ is taken HIGH before this edge, in which case the rising edge of $\overline{SYNC}$ acts as an interrupt and the write sequence is ignored by the DAC8552). Schmitt-Trigger logic input.
	6	SCLK	Serial Clock Input. Data can be transferred at rates up to 30MHz at 5V. Schmitt-Trigger logic input.
MOSI	7	D_{IN}	Serial Data Input. Data is clocked into the 24-bit input shift register on the falling edge of the serial clock input. Schmitt-Trigger logic input.
	8	GND	Ground reference point for all circuitry on the part.

- **Komunikace s DAC8552**
  - Přenáší se vždy 24 bitů (3 B).
  - 1. byte je příkaz (výběr kanálu, zap/vyp režimu s nízkou spotřebou).
  - 2. a 3. byte je 16-bit číslo (0-65535), které se zapíše do D/A převodníku a podle něj se na výstupu A nebo B objeví patřičné napětí.
  - SPI mode 1.
  - Data se přenášejí v pořadí MSB first.
  - Max. komunikační rychlost je 30 MHz.

SERIAL WRITE OPERATION



DB23											DB12
0	0	LDB	LDA	X	Buffer Select	PD1	PD0	D15	D14	D13	D12
DB11											**DB0**
D11	D10	D9	D8	D7	D6	D5	D5	D3	D2	D1	D0

Figure 43. DAC8552 Data Input Register Format

Table 1. Control Matrix

D23	D22	D21	D20	D19	D18	D17	D16	D15	D14	D13–D0	
Reserved	Reserved	Load B	Load A	Don't Care	Buffer Select	PD1	PD0	MSB	MSB-1	MSB-2... LSB	
(Always Write 0)					0 = A, 1 = B						DESCRIPTION
0	0	0	0	X	#	0	0	Data			WR Buffer # w/Data
0	0	0	0	X	#	See Table 2		X			WR Buffer # w/Power-down Command
0	0	0	1	X	#	0	0	Data			WR Buffer # w/Data and Load DAC A
0	0	0	1	X	0	See Table 2		X			WR Buffer A w/Power-Down Command and LOAD DAC A (DAC A Powered Down)
0	0	0	1	X	1	See Table 2		X			WR Buffer B w/Power-Down Command and LOAD DAC A
0	0	1	0	X	#	0	0	Data			WR Buffer # w/Data and Load DAC B
0	0	1	0	X	0	See Table 2		X			WR Buffer A w/Power-Down Command and LOAD DAC B
0	0	1	0	X	1	See Table 2		X			WR Buffer B w/Power-Down Command and LOAD DAC B (DAC B Powered Down)
0	0	1	1	X	#	0	0	Data			WR Buffer # w/Data and Load DACs A and B
0	0	1	1	X	0	See Table 2		X			WR Buffer A w/Power-Down Command and Load DACs A and B (DAC A Powered Down)
0	0	1	1	X	1	See Table 2		X			WR Buffer B w/Power-Down Command and Load DACs A and B (DAC B Powered Down)

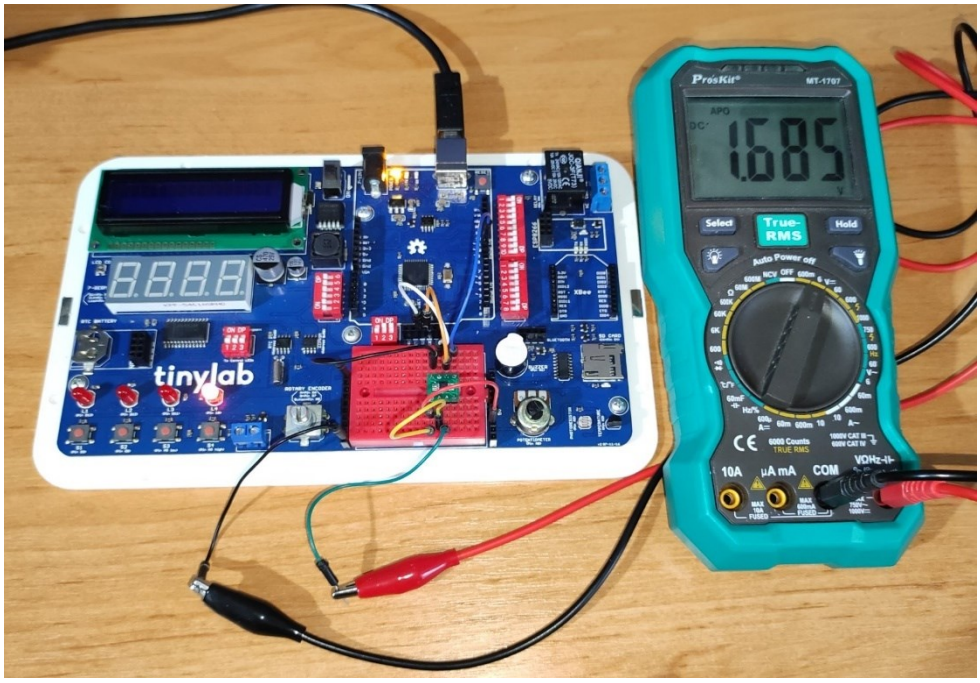
Table 2. Power-Down Commands

D17	D16	
PD1	PD0	OUTPUT IMPEDANCE POWER DOWN COMMANDS
0	1	1kΩ
1	0	100kΩ
1	1	High Impedance

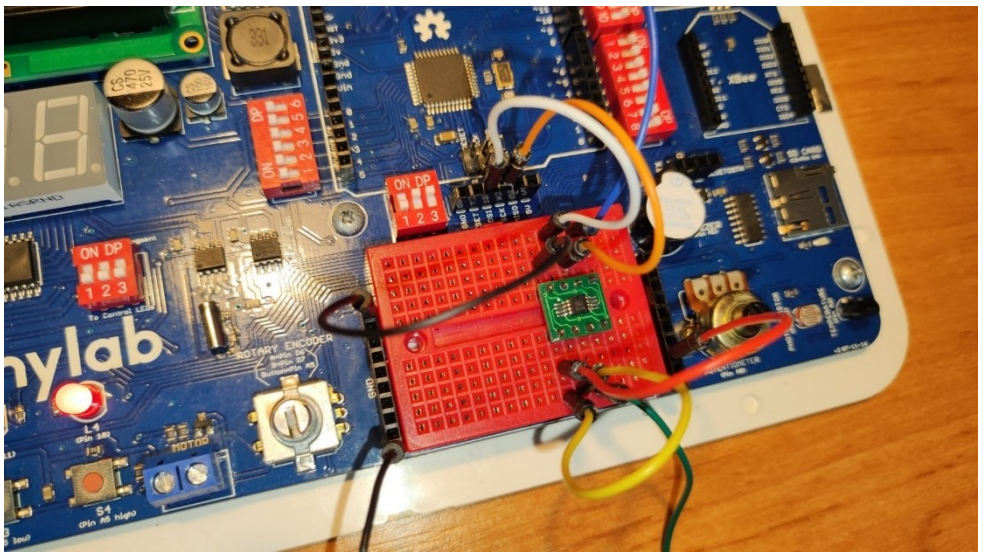
8.8.2 Propojení DAC8852 s kitem TinyLab

Tabulka 28 Propojení D/A převodníku DAC8552 s kitem TinyLab (Arduino Leonardo)

Pin na D/A DAC8552	Pin na TinyLab (Arduino Leoardo)
1 VDD	5V
2 VREF	5V
3 VOUT A	multimetr, osciloskop
4 VOUT B	nezapojeno
5 SYNC	D10 (CS)
6 SCLK	D13 (SCK)
7 DIN	D11 (MOSI)
8 GND	GND



Obrázek 116 Propojení D/A převodníku DAC8552 s kitem TinyLab



Obrázek 117 Propojení D/A převodníku DAC8552 s kitem TinyLab (detail)

- **Kód pro komunikaci s DAC8552 (pro Arduino)**
  - Arduino Leonardo
  - F_{CPU} = 16 MHz
  - Využití knihovny SPI.
  - Program na obou výstupech (kanálech) D/A převodníku postupně zvyšuje napětí z 0 V na maximum (cca 5 V), pak přeteče na 0 V a znovu se napětí na výstupech postupně zvyšuje z 0 V na maximum a tak stále dokola.

```
#include "Arduino.h"
#include <SPI.h>

#define SS_PIN 10   // D10 bude Slave Select (Chip Select) pin

// Makra pro ovladani Chip Select pinu SS
#define SS_PIN_LOW  digitalWrite(SS_PIN,LOW)
#define SS_PIN_HIGH digitalWrite(SS_PIN,HIGH)

// deklarace funkci
void WriteToDAC(uint8_t channel, uint16_t DA_counts);

void setup()
{
	// nastaveni urovne a smeru na pinu SS pro komunikaci s DAC8552
	digitalWrite(SS_PIN,HIGH);
	pinMode(SS_PIN,OUTPUT);

	// inicializace SPI sbernice
	SPI.begin();  // nastavi piny MOSI, SCK, SS na vystupni a pin MOSI na vstupni

	// Nastaveni 0 V na obou vystupech D/A prevodniku
	WriteToDAC(0,0);     // vystup (kanal) A
	WriteToDAC(1,0);     // vystup (kanal) B
}

void loop(void)
{
	static uint16_t output = 0;

	// Postupne zvysovani napeti na obou vystupech D/A prevodniku a po preteceni na
	// 0 V stale dokola.
	WriteToDAC(0,output);
	WriteToDAC(1,output*2);
	output++;
	delayMicroseconds(100); // aby se napeti nemenila prilis rychle
}
```

```
/*
Zapise novou 16-bit RAW hodnotu do DAC8552.
Parametry:    channel - cislo kanalu na DAC (0 – kanal A, 1 – kanal B)
              DA_counts – 16-bit RAW hodnota na vybranem kanalu v rozsahu 0 - 65535
Vraci:        nic
*/
void WriteToDAC(uint8_t channel, uint16_t DA_counts)
{
      uint8_t command = 0;

      // vytvoreni command bajtu podle cisla kanalu
      if (channel == 0)
            command = (1<<4);          // LDA = 1, Buffer Select = A, command = 0x10
      else if (channel == 1)
            command = (1<<5) | (1<<2); // LDB = 1, Buffer Select = B, command = 0x24

      // Odeslani dat do D/A prevodniku
      if (channel <= 1)
      {
            // nastaveni komunikacni rychlosti a rezimu prenosu
            // zde prenosova rychlost 1 MHz, Most Significant Bit first, SPI Mode 1,
            // max. rychlost SPI na AVR je 8 MHz (16 MHz krystal)
            SPI.beginTransaction(SPISettings(1000000, MSBFIRST, SPI_MODE1));

            SS_PIN_LOW;                      // Slave Select do log. 0
            SPI.transfer(command);           // odeslani prikazu (vyber kanalu)
            SPI.transfer(DA_counts>>8);      // odeslani horniho bajtu dat
            SPI.transfer(DA_counts&0xFF);    // odeslani dolniho bajtu dat
            SS_PIN_HIGH;                     // Slave Select do log. 1

            SPI.endTransaction();      // uvolneni SPI pro ostatni casti programu
      }
}
```

SPI.transfer(DA_counts>>8);

SPI.transfer(command);

SPI.transfer(DA_counts&0xFF);

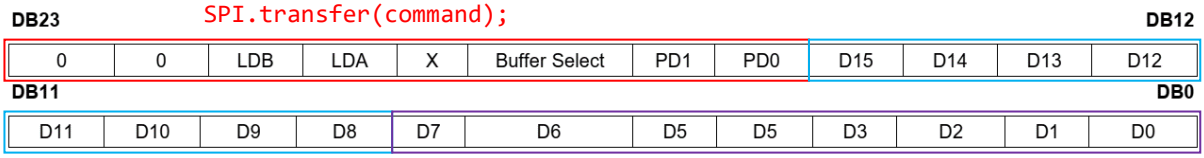
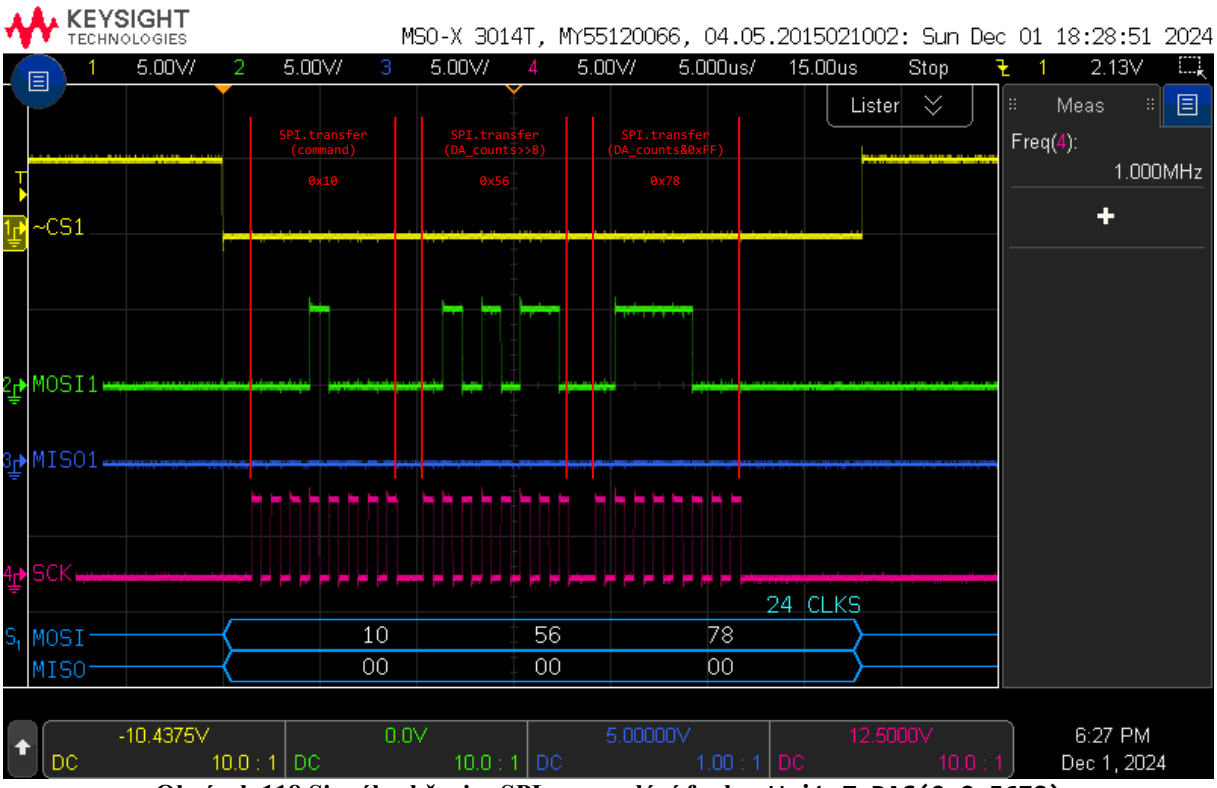
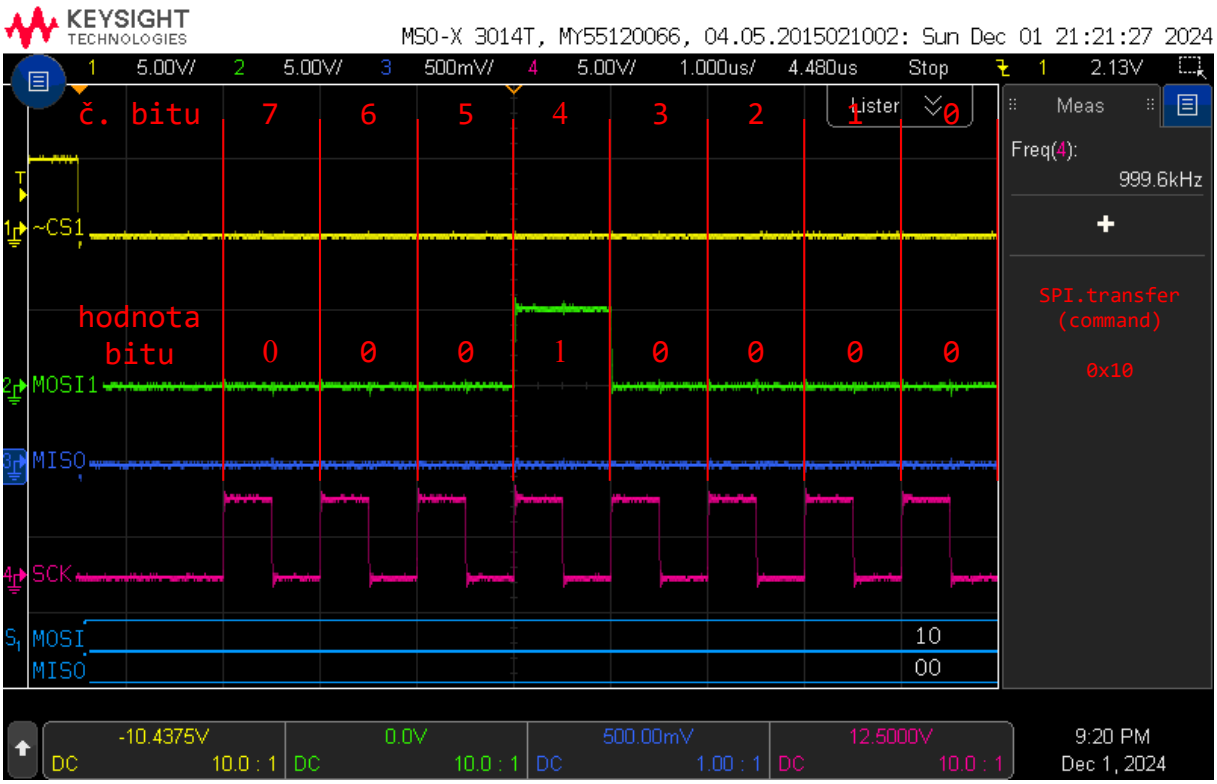


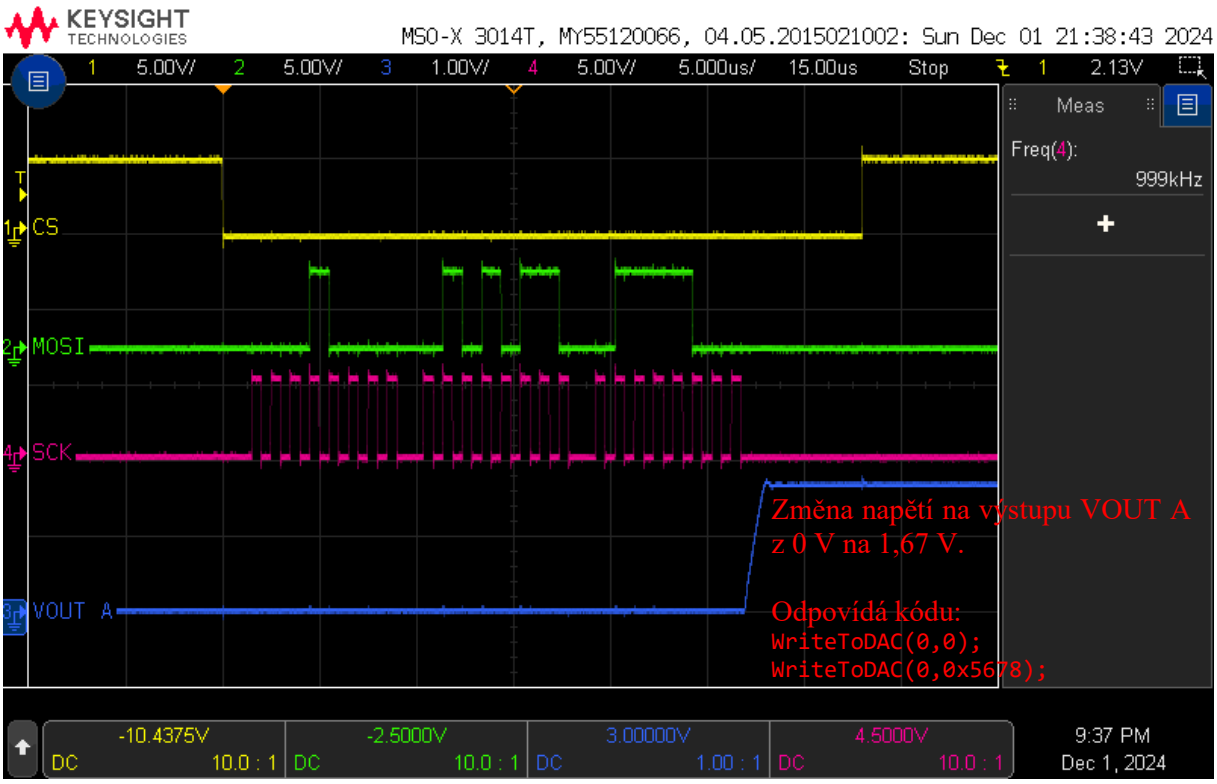
Figure 43. DAC8552 Data Input Register Format



Obrázek 118 Signály sběrnice SPI po zavolání funkce `WriteToDAC(0,0x5678)`



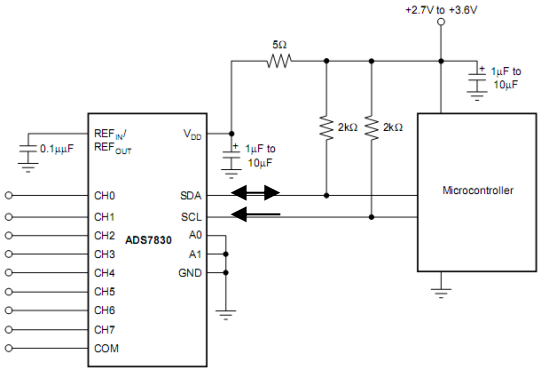
Obrázek 119 Detail 1. bajtu (command) odesílaného po SPI do D/A převodníku DAC8552



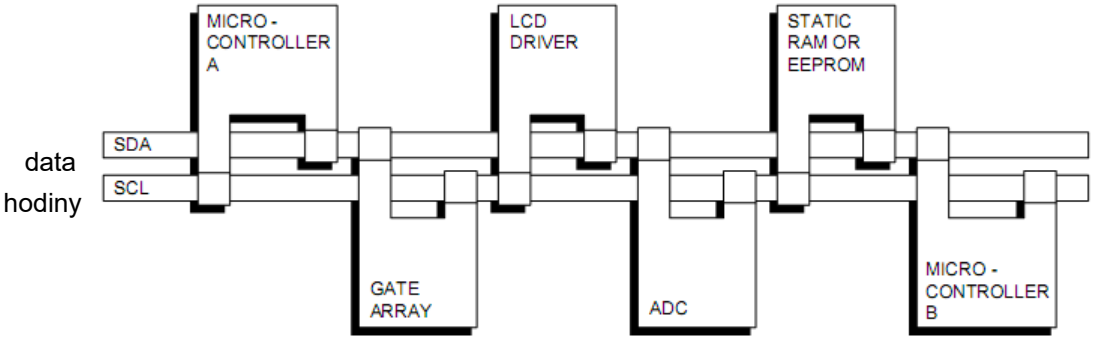
Obrázek 120 Viditelná změna napětí na výstupu VOUT A D/A převodníku DAC8552 po zavolání funkce `WriteToDAC(0,0x5678)`

9 Sběrnice I²C

- Je to dvouvodičová obousměrná polo-duplexní sériová sběrnice určená pro komunikaci MCU s podřízenými obvody na DPS.
- Přenos dat: sériový, 8-bit slova, obousměrný half-duplex
- Rychlosti:
  - 100 kbps (standard)
  - 400 kbps (fast)
  - 1 Mbps Fast mode+
  - 3,4 Mbps (high-speed mode)

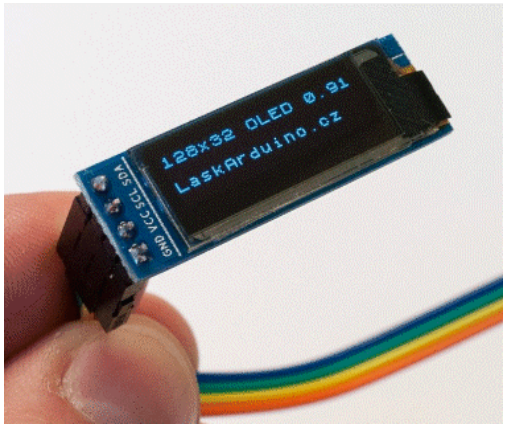


- Topologie: master/slave, multimaster
- **Všechny obvody jsou připojeny na jedné sběrnici a jsou vybírány pomocí adresy, která se přenáší společně s daty (není zde vodič /SS jako tomu je u sběrnice SPI).**
  - Adresa 7-bit (původní standard) nebo 10-bit (rozšířený standard)



- Počet zařízení na sběrnici – omezen pouze celkovou kapacitou sběrnice (max. 400 pF)
- **Existuje velké množství obvodů komunikujících po I2C sběrnici**. Např. paměti EEPROM, A/D a D/A převodníky, budiče LED, teploměry, I/O pro audio-vizuální techniku apod.

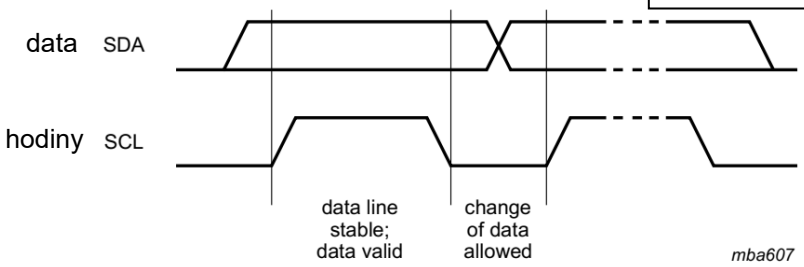
9.1 Fyzické uspořádání sběrnice



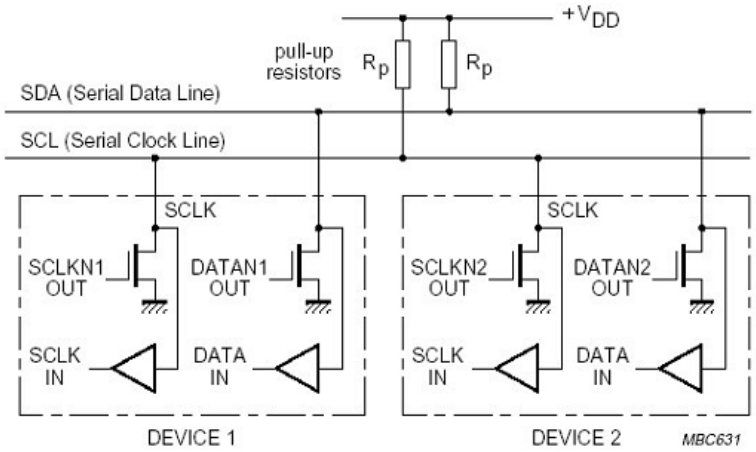
- 2 vodiče
  - SDA = serial data
  - SCL = serial clock
- Přenos bitů – závislý na úrovni CLK (ne hraně)
  - SCL = HIGH – platná data na SDA
  - Data se mění při SCL = LOW
  - 1 CLK pulz na 1 bit dat
- Délka sběrnice max. cca 30 cm (na 100 kHz).

- **Princip přenosu dat po I2C**

Data vysílače na lince SDA jsou platná při vysoké úrovni hodin (linka SCL = 1).



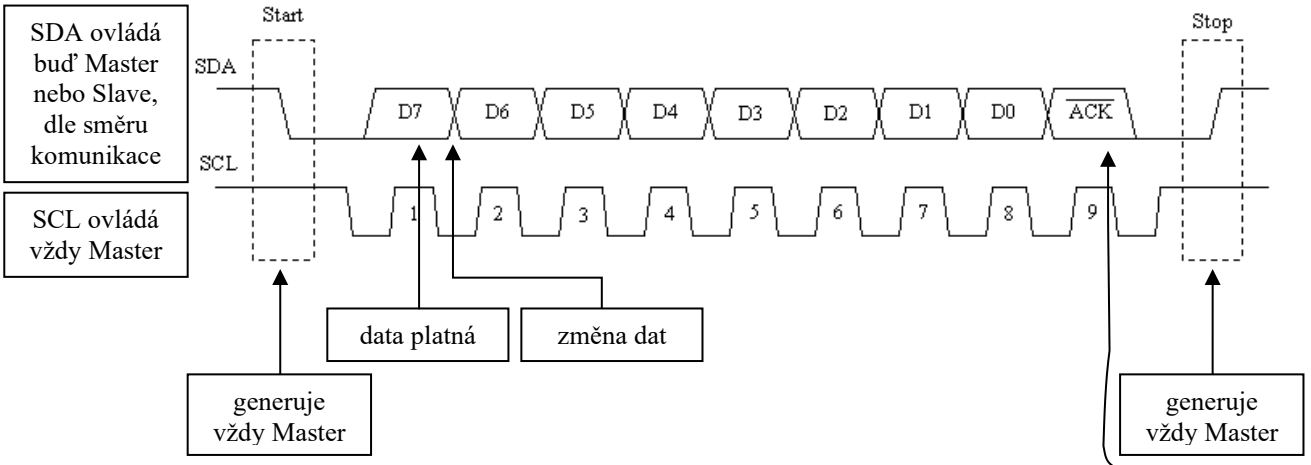
- Budiče jednotlivých zařízení na sběrnici realizují „wired-and“
  - **Výstupy vysílačů jsou typu otevřený kolektor** – na linku SDA a SCL je nutné přidat **externí pull-up rezistor** vůči VDD, který definuje logickou úroveň v klidovém stavu na sběrnici.
  - Hodnota pull-up rezistoru R_p je obvykle 2k2 nebo 4k7. Čím více obvodů je připojených na sběrnici, tím je potřeba menší pull-up rezistor, protože roste kapacita sběrnice. Podrobný postup výpočtu vhodné hodnoty pull-up rezistorů je uveden v kapitole 7 specifikace I2C (zde).



- **Klidový stav**
  - SDA i SCL v logické 1 (klidový stav je dán pull-up rezistorem na dané lince).
  - Napěťová úroveň na linkách je určena velikostí VDD.
- **Zdroje informací o sběrnici I2C**
  - Popis I2C v kostce: https://i2c.info/ https://www.ti.com/lit/slva704
  - Specifikace I2C od výrobce: https://www.pololu.com/file/0J435/UM10204.pdf
  - Manuál k I2C: https://www.nxp.com/docs/en/application-note/AN10216.pdf
  - I2C na Arduino:
    https://docs.arduino.cc/language-reference/en/functions/communication/wire/
    https://docs.arduino.cc/learn/communication/wire
    https://learn.adafruit.com/working-with-i2c-devices

9.2 Přenosový protokol

- Master (Controller) zahájí vysílání vygenerováním události Start - sestupná hrana na SDA při SCL = 1.

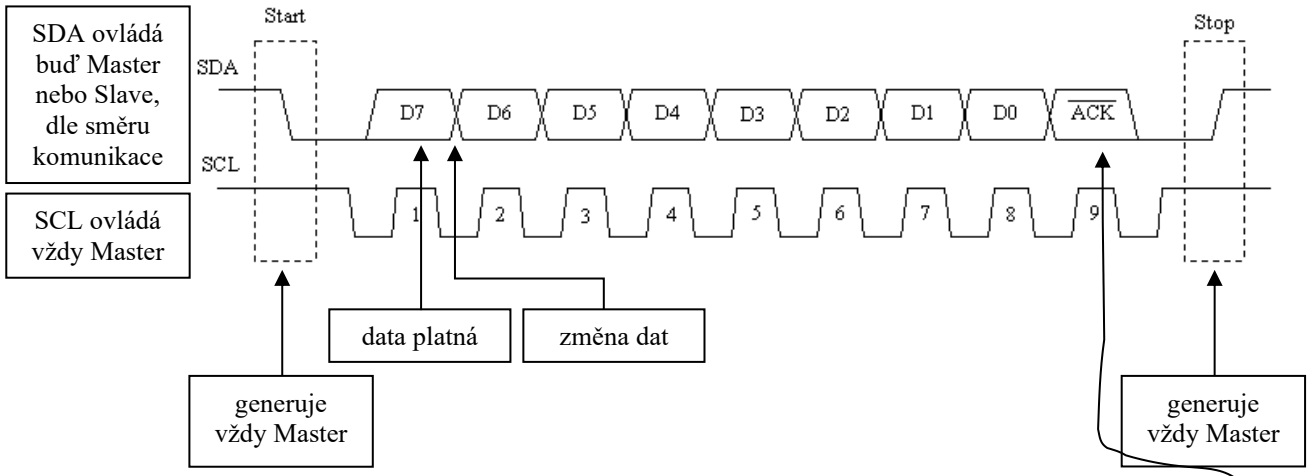


- **Master (Controller) přenese 8 bitů dat do Slave (Target Device)**
  - Data se přenáší od nejvyššího bitu k nejnižšímu (Most Significant Bit first).
  - Data vysílače na lince SDA jsou **platná při vysoké úrovni hodin** (linka SCL = 1).
  - Devátý datový bit je tzv. **ACK** – **potvrzovací (acknowledge) bit**.
    - **Vysílač**: uvolní linku SDA (pull-up rezistor zajistí log. úroveň 1)
    - **Přijímač:** linku SDA stáhne do 0 (**potvrdí přenos**)
    - Potvrzování pomocí ACK bitu se používá v obou směrech komunikace. Jak ve směru Master → Slave (Slave potvrzuje příjem) tak ve směru Slave → Master (Master potvrzuje příjem). Vysílačem dat na SDA tedy může být jak Master, tak Slave.
    - Co znamená **nepotvrzení příjmu** (ACK = 1 nebo-li **NACK**)
      - **Ve směru komunikace Master→Slave** se jedná o chybu na straně Slave (např. Slave není na sběrnici vůbec přítomen nebo má jinou adresu, než vysílá Master) a řešení této situace je na Masterovi - opakovaný start komunikace (Start) nebo stop (Stop).
      - **Ve směru komunikace Slave → Master** před ukončením komunikace (vygenerováním Stop) Master nepotvrzuje příjem posledního bajtu (NACK) ze Slave před Stop. To je správně.
- **Master ukončí komunikaci vygenerováním události Stop** - vzestupná hrana na SDA při SCL = 1.
- **Stavy sběrnice**
  - **Busy**: po Start a před dalším Stop
  - **Free**: po Stop a před dalším Start

- **Komunikaci po I2C lze realizovat**
  - **softwarově**
    - Tzv. „bit-bang“ (SW emulace protokolu I2C).
    - MCU nemusí být vybaven hardwarovým řadičem I2C.
    - Vyberou se 2 digitální piny na MCU (jeden bude sloužit jako SDA, druhý SCL) a logické úrovně na nich se ovládají programově.
  - **hardwarově**
    - MCU je vybaven řadičem I2C pro rychlou komunikaci.
    - Dovoluje použít i přerušení při vysílání a příjmu – zrychlení vykonávání programu, ten nemusí čekat na odeslání a příjem všech dat.

9.2.1 Formát rámce (zjednodušeně)

- **Master (Controller) zahájí vysílání vygenerováním události Start** - sestupná hrana na SDA při SCL = 1.

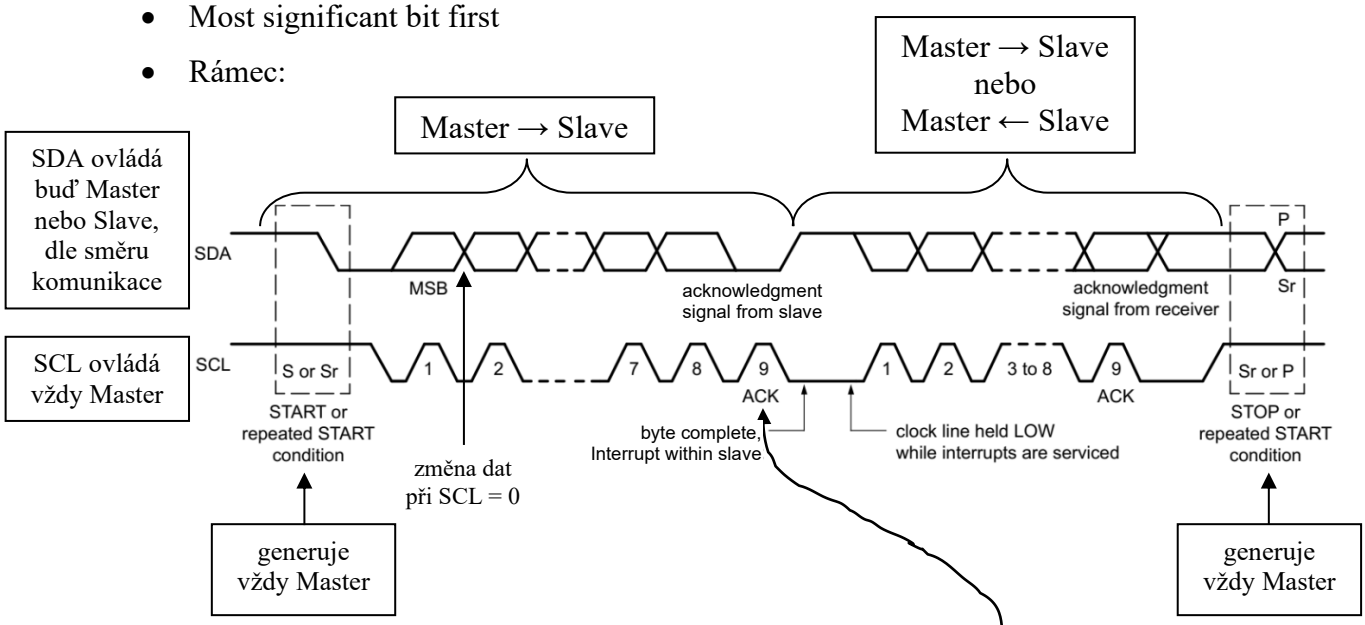


- **Master (Controller) přenese 8 bitů dat do Slave (Target Device)**
  - Data se přenáší od nejvyššího bitu k nejnižšímu (Most Significant Bit first).
  - Data vysílače na lince SDA jsou **platná při vysoké úrovni hodin** (linka SCL = 1).
  - Devátý datový bit je tzv. **ACK** – **potvrzovací (acknowledge) bit**.
    - **Vysílač**: uvolní linku SDA (pull-up rezistor zajistí log. úroveň 1)
    - **Přijímač:** linku SDA stáhne do 0 (potvrdí přenos)
    - Potvrzování pomocí ACK bitu se používá v obou směrech komunikace. Jak ve směru Master → Slave (Slave potvrzuje příjem) tak ve směru Slave → Master (Master potvrzuje příjem). Vysílačem dat na SDA tedy může být jak Master, tak Slave.
    - Co znamená **nepotvrzení příjmu** (ACK = 1 nebo-li **NACK**)
      - **Ve směru komunikace Master→Slave** se jedná o chybu na straně Slave (např. Slave není na sběrnici vůbec přítomen nebo má jinou adresu, než vysílá Master) a řešení této situace je na Masterovi - opakovaný start komunikace (Start) nebo stop (Stop).
      - **Ve směru komunikace Slave → Master** před ukončením komunikace (vygenerováním Stop) Master nepotvrzuje příjem posledního bajtu (NACK) ze Slave před Stop. To je správně.
- **Master ukončí komunikaci vygenerováním události Stop** - vzestupná hrana na SDA při SCL = 1.
- **Stavy sběrnice**
  - **Busy**: po Start a před dalším Stop
  - **Free**: po Stop a před dalším Start

- **Komunikaci po I2C lze realizovat**
  - **softwarově**
    - Tzv. „bit-bang“ (SW emulace protokolu I2C).
    - MCU nemusí být vybaven hardwarovým řadičem I2C.
    - Vyberou se 2 digitální piny na MCU (jeden bude sloužit jako SDA, druhý SCL) a logické úrovně na nich se ovládají programově.
  - **hardwarově**
    - MCU je vybaven řadičem I2C pro rychlou komunikaci.
    - Dovoluje použít i přerušení při vysílání a příjmu – zrychlení vykonávání programu, ten nemusí čekat na odeslání a příjem všech dat.

9.2.2 Formát rámce (podrobněji)

- Master zahajuje komunikaci vygenerováním události Start.
- Přenos po bajtech = 8b + 1 ACK bit.
- Most significant bit first
- Rámec:



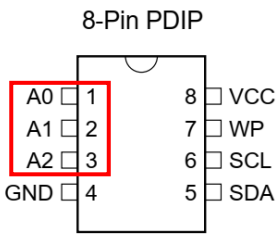
- Data platná při SCL = 1, změna dat na SDA při SCL = 0.
- **ACK** = 9. bit na SDA
  - Vysílač: uvolní SDA (= pull-up rezistor zajistí log. úroveň 1)
  - Přijímač: SDA stáhne do 0 (= potvrdí přenos)
  - Slave po vygenerování ACK uvolní linku SDA a umožní masterovi zakončit přenos (vygenerovat STOP condition).
- **NACK** = stav, kdy příjemce negeneruje ACK
  - Význam této události závisí na směru komunikace.
  - **Ve směru komunikace Master→Slave** se jedná o chybu na straně Slave (např. Slave není na sběrnici vůbec přítomen nebo má jinou adresu, než vysílá Master) a řešení této situace je na Masterovi - opakovaný start komunikace (Sr) nebo stop (P).
  - **Ve směru komunikace Slave → Master** před ukončením komunikace (vygenerováním Stop) Master nepotvrzuje příjem posledního bajtu (NACK) před Stop. To je správně.
- **SCL = 0 po ACK**
  - Slave může podržet SCL v 0 po ACK jako indikaci, že je zaneprázdněn zpracováním přijatých dat (např. u EEPROM chvíli trvá než dokončí zápis přijatého bajtu).

- V tomto případě Master musí čekat s přenosem dalších dat, než Slave zpracuje data – je znázorněno výše na obrázku („clock line held low while interrupts are serviced“).

9.2.3 Adresa

- Každý I2C obvod má na sběrnici svojí **jedinečnou adresu**, která je z části pevně daná („zadrátovaná v obvodu“) a z části je volitelná (připojením adresních vývodů součástky na GND nebo na VCC).
- **Pevná část** – určena výrobcem obvodu (konsorciem) dle typu součástky.
  https://learn.adafruit.com/i2c-addresses/the-list nebo https://i2cdevices.org/addresses
- **Volitelná část** – dovoluje provoz několika součástek stejného typu na sběrnici, určena log. úrovněmi na adresních pinech dané součástky.
- Adresa má délku 7-bit (původní standard – až 112 zařízení na sběrnici) nebo 10-bit (rozšířený standard – až 1024 zařízení na sběrnici). Každé zařízení musí mít na I2C sběrnici jedinečnou adresu.
- Ke sběrnici může být připojeno více obvodů stejného typu (např. několik pamětí EEPROM).
  - Jejich počet je však omezen počtem volitelných adres.
  - Pevnou část adresy budou mít např. všechny EEPROMky stejnou.
  - Má-li paměť např. 3 adresní vývody, tak mohu mít 2^3 = 8 různých adres a tedy max. 8 těchto (stejných) obvodů na jedné sběrnici.

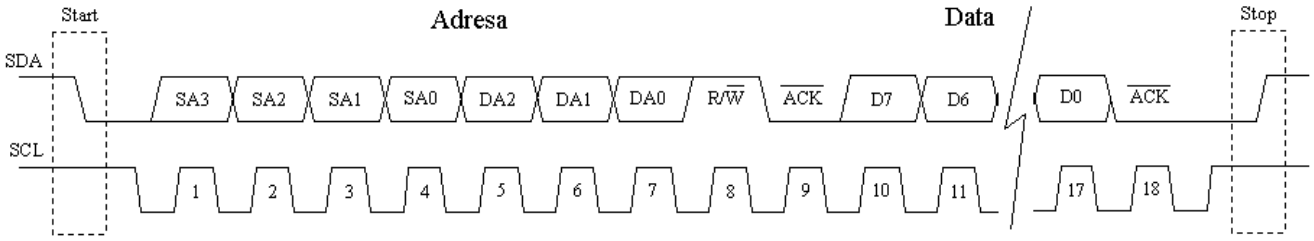
Pin Name	Function
A0 to A2	Address Inputs
SDA	Serial Data
SCL	Serial Clock Input
WP	Write Protect



Obrázek 121 Adresní piny na pouzdru externí I2C EEPROM AT24C32

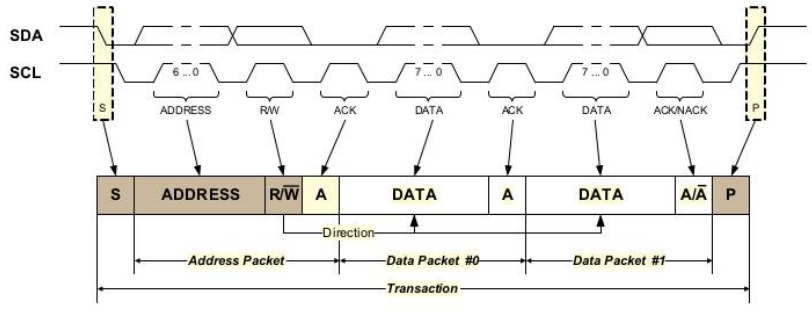
9.2.4 Přenos adresy a dat

- Komunikace vždy začíná přenosem adresy, po které následují datové bajty.
- Předpokládejme adresaci 7-bit.
- Prvních 7-bitů při vysílání = adresa
- 8. bit = směr přenosu (R/W bit):
  - 0 = master bude data zapisovat, stává se vysílačem
  - 1 = master bude data číst (po odvysílání adresy), stane se přijímačem
- Každé Slave zařízení porovná přijatou adresu se svou adresou, pokud shoda → nastavení směru komunikace dle R/W bitu.
- **Na následujícím obrázku je znázorněn typický příklad komunikace po I2C.**



bity SAx – pevná část adresy bity
bity DAx – volitelná část adresy

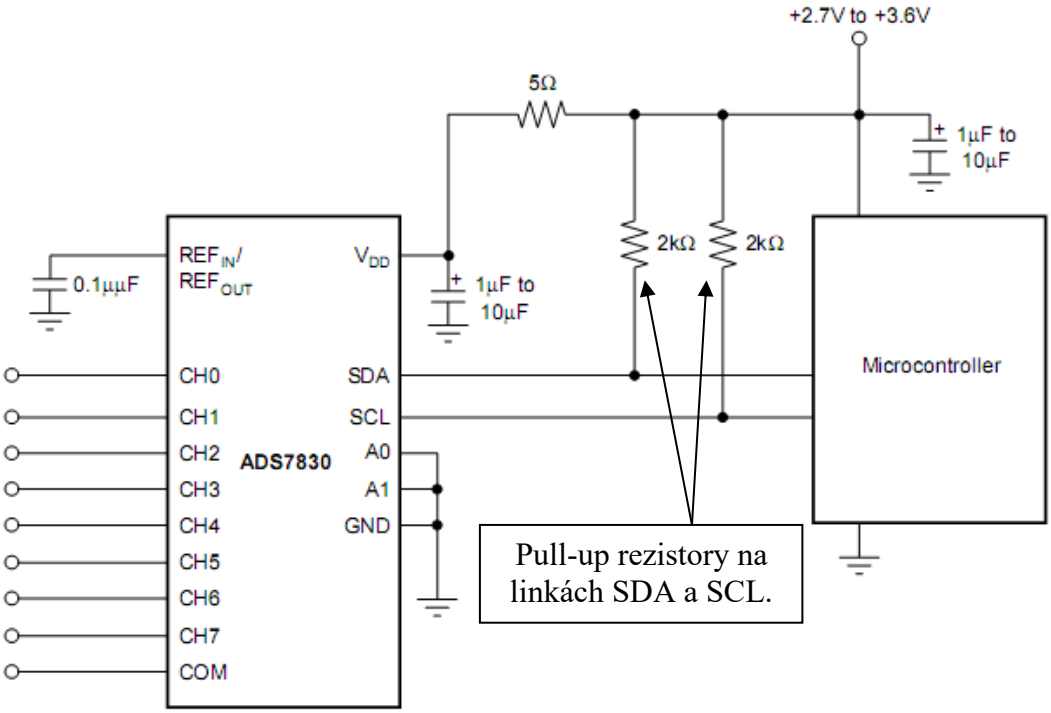
- Na obrázku jsou znázorněny 2 přenášené bajty:
  - Adresa
    - Bity SA3 až DA0 je určena 7-bitová adresa zařízení na sběrnici.
    - Bit R/W určuje směr přenosu (zda se jedná o čtení nebo o zápis).
      - Je-li R/W = 1 data se ze Slave budou vyčítat, je-li R/W = 0 data se do Slave budou zapisovat.
    - Přijímač (Slave) potvrdí přijatý bajt bitem ACK v log. 0.
  - Data
    - Po přenosu adresy následuje datový bajt (jeden nebo více), jehož obsah již závisí na typu zařízení, se kterým se komunikuje.
    - Po přenosu datového bajtu je spojení ukončeno (Master generuje Stop).
- Další příklad komunikace mezi Master a Slave (přenos 2 datových bajtů):



zdroj: adafruit.com

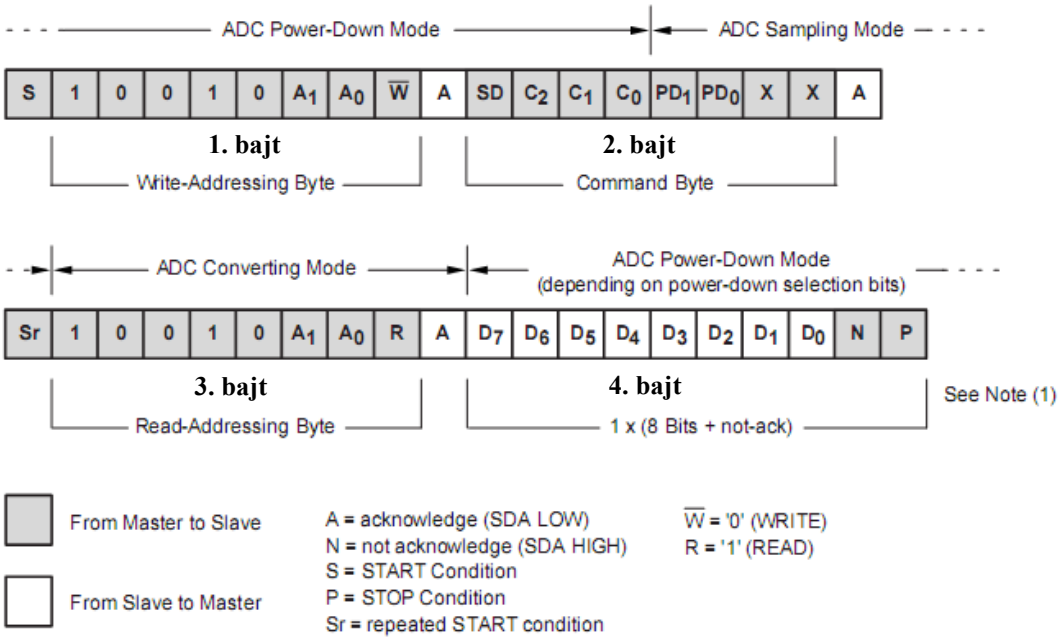
9.3 Příklad I²C obvodu – A/D převodník ADS7830

- 8-kanálový 8-bit A/D převodník s postupnou aproximací s rozhraním I2C.
- Rychlost komunikace po I2C : 100 kHz, 400 kHz, 3,4 MHz.
- 70 kHz maximální vzorkovací frekvence při komunikační rychlosti 3,4 MHz.
- **Příklad el. zapojení**



Obrázek 122 El. propojení 8-bit A/D převodníku ADS7830 s MCU

- **komunikace po I2C**
  - Spočívá v nastavení A/D převodníku, spuštění převodu, vyčtení výsledku.
  - To je realizováno přenosem 4 bajtů mezi MCU a A/D převodníkem.



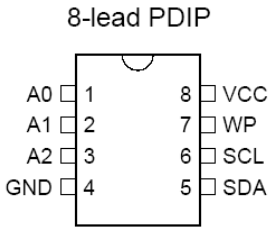
  - 1. bajt – „zapiš příkaz do A/D“ (Write-Addressing Byte)
    - adresa 0x48 + bity A1, A0
    - bude se zapisovat do ADC – bit /W = 0
    - ACK bit – MCU vysílá HIGH, ADC potvrzuje příjem stažením linky SDA do 0
  - 2. bajt – příkaz (Command Byte)
    - bit SD – nastavení vstupů A/D převodníku (unipolární/diferenciální)
    - bity C0, C1, C2 – výběr kanálu (č. 0 až 7)
    - bity PD0, PD1 – volba stavu A/D převodníku a napěťové reference
    - A/D převod se spouští automaticky po přijmutí 4. bitu příkazu (bit C0), po výběru kanálu
    - ACK bit – MCU vysílá HIGH, ADC potvrzuje příjem stažením linky SDA do 0
  - 3. bajt – „příkaz čti z A/D“ (Read-Addressing Byte)
    - adresa 0x48 + bity A1,A0
    - bude se číst výsledek převodu z ADC – bit R = 1
    - ACK bit – MCU vysílá HIGH, ADC potvrzuje příjem stažením linky SDA do 0

- 4. bajt – vyčtení 8-bit výsledku převodu z A/D převodníku
  - MCU generuje hodiny na SCL a ADC mu po SDA posílá převedené napětí (8-bit výsledek D7 až D0)
  - ACK bit – ADC vysílá HIGH, MCU nepotvrzuje příjem (ACK bit nechává v 1)
  - MCU ukončuje komunikaci vygenerováním STOP (bit označený P)
- **Frekvence vzorkování (perioda A/D převodu) je určena**
  - Periodou spouštění A/D převodu, tj. jak často se převod spouští povelem po I2C sběrnici.
  - Maximální vzorkovací frekvence závisí na použité komunikační rychlosti I2C sběrnice (přenos dat mezi MCU a ADC trvá dle výše uvedeného schematu cca 32 tiků hodin SCL, což je doba trvání jednoho A/D převodu).
  - Při komunikační frekvenci 100 kHz trvá jeden A/D převod cca 32x10 us = 320 us (bez započítání Start, Stop a ACK bitů). To znamená, že při kontinuální komunikaci bez prodlev je dosaženo frekvence vzorkování 1/320 us = 3,125 kHz.

9.4 Příklad I²C obvodu – EEPROM 24Cxx

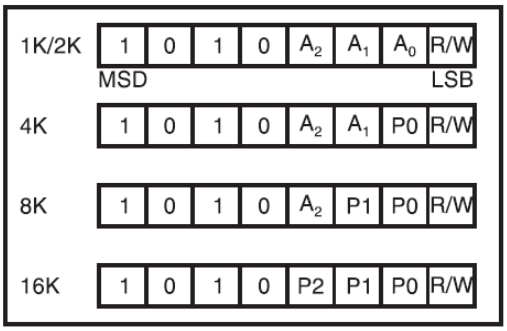
- Sériová paměť typu EEPROM firmy Atmel komunikující po sběrnici I2C.
- xx v názvu označuje velikost paměti – např. AT24C01 má velikost 1 kbit = 128 B
- Paměť je organizovaná po bajtech – např. 1 kbit = 128 * 8 bit
- Napájení 2,7 – 5,5 V. U Low-Power verzí až 1,8 V.
- Zápis jednoho slova trvá max. 10 ms.
- Životnost 1 M zápisových cyklů.
- Uspořádání a popis vývodů:

Pin Name	Function
A0 - A2	Address Inputs
SDA	Serial Data
SCL	Serial Clock Input
WP	Write Protect
NC	No Connect
GND	Ground
VCC	Power Supply



- **Adresní vývody A0 – A2** – pomocí nich lze určit volitelnou část adresy. U větších pamětí je funkčí např. pouze jeden vývod (A2), což limituje počet pamětí připojených na sběrnici na 2 kusy. Paměť 24C16 (16 kbit / 2 kB) lze připojit dokonce jenom jednu.
- **Pin WP** – připojením na Ucc aktivuje zákaz zápisu do paměti. Připojením ke GND je zápis povolen.

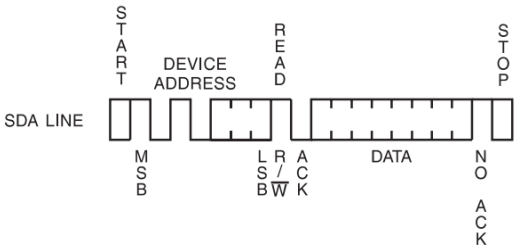
9.4.1 Adresa paměti



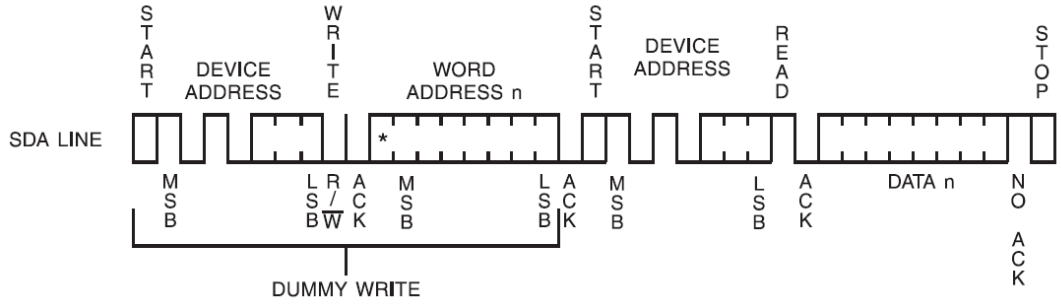
- V závislosti na velikosti paměti se mění počet volitelných adres a tím maximální počet obovdů připojených na jednu sběrnici. U pamětí 4 kbit (512 B) a větších jsou některé (nebo všechny) bity Ax nahrazeny částí adresy pro adresování paměťové buňky (bity Px). Např. u AT24C16 (2 kB) je nutné mít 11 bitovou adresu pro uadresování 2 kB, proto lze na sběrnici připojit pouze jednu paměť (chybí zde bity Ax).

9.4.2 Čtění z paměti

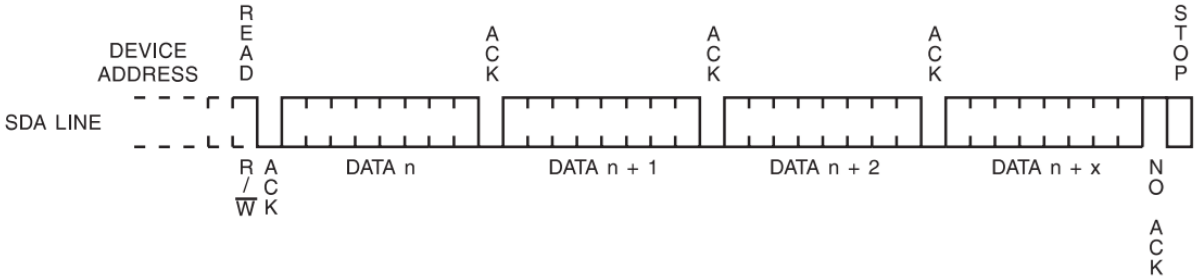
- **Čtení hodnoty na aktuální adrese (Current Address Read)** – vyčte bajt z paměti na adrese, která je dána aktuální hodnotou adresního čítače paměti. Ten je po každém přístupu inkrementován. Do paměti se odešle pouze fyzická adresa a příznak čtení. Přijmutá data se **nepotvrzují** (ACK = 1).



- **Vyčtení kteréhokoliv bajtu (Random Read)** – do paměti se nejprve odešle fyzická adresa a adresa datové buňky. Bit R/W je ovšem (nečekaně) nastaven na nulu, což značí zápis. Paměť potvrdí příjem těchto dvou bajtů (ACK = 0). Poté musí následovat opětovné vygenerování povelu START a odeslání fyzické adresy tentokrát ale s nastaveným bitem R/W pro čtení (R/W = 1). Paměť potvrdí příjem a vyšle obsah paměťové buňky. Příjem tohoto bajtu se ovšem **nepotvrzuje** (ACK = 1), pak se přenos ukoční STOP signálem.

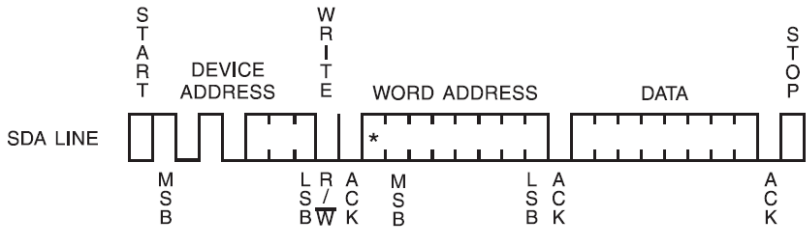


- **Sekvenční čtení** - je vyvoláno stejným způsobem jako Random Read, ale jednotlivé přijmuté bajty Master naopak **potvrzuje** (ACK = 0). Po každém povrzení se v paměti inkrementuje interní čítač adresy a odešle se další bajt s obsahem paměťové buňky. Data Slave vysílá tak dlouho dokud Master potvrzuje jejich příjem (ACK = 0). Příjem se ukončí stejně jako v předchozím případě – Master nepotvrdí příjem (ACK = 1) a ukončí přenos (vygeneruje STOP). Dosáhne-li se konce paměti, "přeteče" se na začátek – datovou adresu 0.

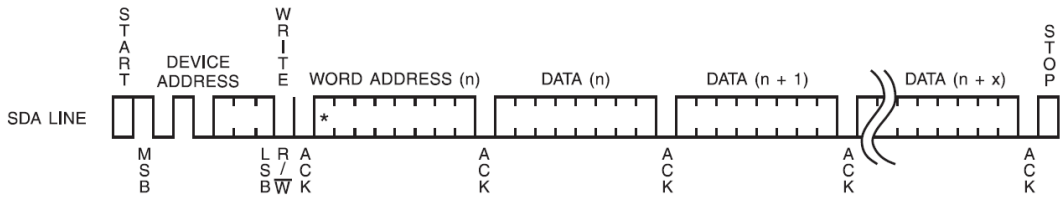


9.4.3 Zápis do paměti

- **Zápis jednoho bajtu** – se provede odesláním fyzické adresy s nastaveným příznakem zápisu (R/$\overline{W}$ = 0) následovaným bajtem s datovou adresou. U větších pamětí (24C02 a výše) je část datové adresy obsažena v bajtu s fyzickou adresou. Pak následuje odeslání datového bajtu s obsahem, který se má zapsat do paměti na danou adresu. Příjem každého bajtu pamět potvrzuje (ACK = 0). Po příjmu posledního ACK bitu musí Master vygenerovat STOP a ukončit tím přenos. Poté přejde paměť do módu zápisu a zapíše námi odeslaná data do příslušné paměťové buňky. Během zápisu paměť nereaguje na žádná data.

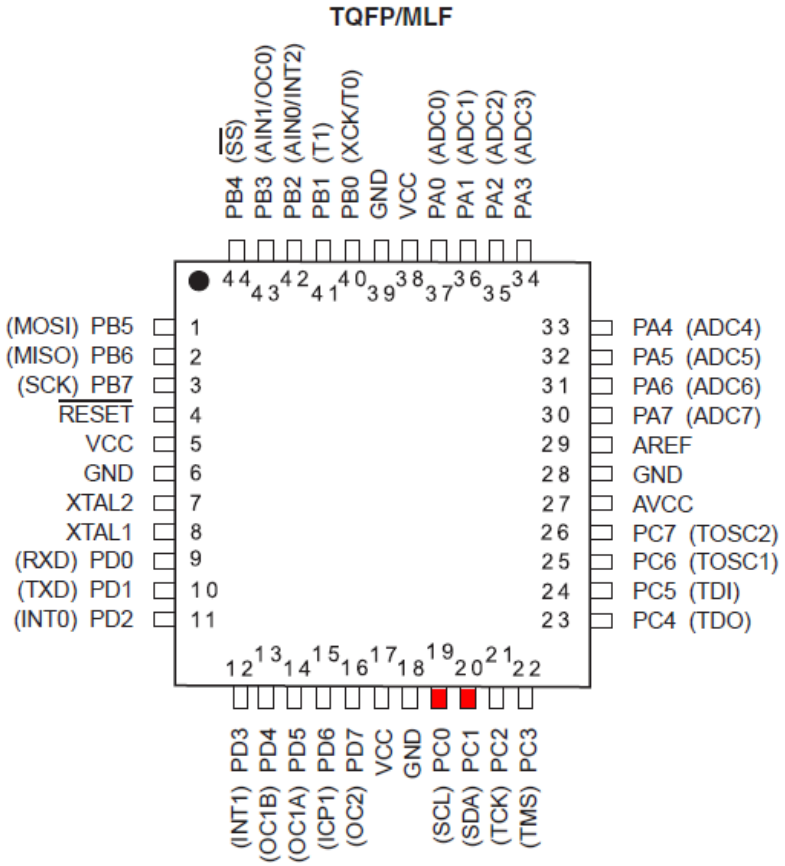


- **Sekvenční zápis** – protokol je stejný akorát Master po odeslání prvního datového bajtu nevygeneruje STOP, ale pokračuje v odesílání dalších datových bajtů. Maximální počet, který lze odeslat, závisí na velikosti paměti neboť interně je paměť rozdělena do několika stránek po 8-mi nebo 16-ti bajtech. Kdyby se odeslalo více dat než je maximum ve stránce, dojde k přepsání prvních bajtů v dané stránce (počítadlo "přeteče"). Přenos dat do paměti se ukončí tím, že Master vygeneruje STOP.



9.5 I²C u AVR

- Některé typy AVR obsahují HW řadič sběrnice I²C. Atmel toto rozhraní označuje jako **TWI** (**Two Wire Interface**). K tomuto rozhraní jsou přidruženy piny s alternativními funkcemi **SDA** a **SCL**.



Obrázek 123 Piny SDA a SCL u ATmega32

- Vlastnosti hardwarového I2C rozhraní u AVR
  - MCU se může chovat jako Master nebo Slave
  - 7-bit adresa
  - Max. přenosová rychlost 400 kHz
- Použití HW řadiče I2C viz datasheet.
- Pro obsluhu řadiče I2C v jazyku C lze využít knihovních funkcí dostupných v balíku kompilátoru (pokud existují) nebo knihovních funkcí třetích stran nebo si lze napsat funkce svoje vlastní (což bohužel není tak triviální jako u SPI, vzhledem k trochu komplikovanějšímu I2C protokolu ve srovnání s SPI).
- U AVR bez hardwarového I2C lze protokol I2C emulovat softwarově (tzv. „bit-bang“).
  - Nejjednodušeji použitím knihovny třetí strany.

- **Knihovna pro komunikaci na sběrnici I2C obvykle obsahuje funkce, které v roli Mastera umožňují**:
  - Zahájit přenos dat – generovat událost Start.
  - Odeslat byte z Master do Slave.
  - Přijmout byte ze Slave do Master s potvrzením (ACK) nebo bez potvrzení (NO ACK).
- Ukončit přenos dat – generovat událost Stop.

9.6 I²C a kompilátor CodeVision

- CV CodeVision má softwarovou podporu sběrnice I2C ve formě knihovních funkcí.
- Tyto funkce předpokládají, že AVR je Master a periferní obvody jsou Slave.
- Funkce jsou deklarovány v hlavičkovém souboru `i2c.h`, který je potřeba „nainkludovat“ na začátku programu.
- Jedná se o bit-bang implementaci I2C sběrnice (SW emulace). Před použitím těchto funkcí je nutné nadefinovat port a bity tohoto portu, které budou mít funkci linek SDA a SCL.

```
// I2C Bus functions
#asm
   .equ __i2c_port=0x1B ;PORTA
   .equ __sda_bit=0
   .equ __scl_bit=1
#endasm
#include                                                    <i2c.h>
```

- Mimo funkcí pro přímou obsluhu sběrnice nabízí CV i obslužné funkce pro řadu obvodů převážně od firmy Philips a Dallas (teplotní čidla, RTC apod.).

9.6.1 Funkce pro obsluhu sběrnice I2C

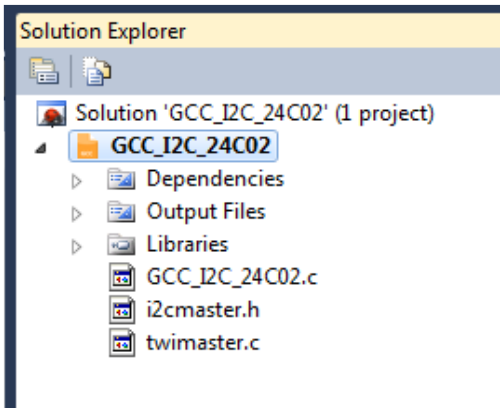
- `void i2c_init(void)` - Inicializace sběrnice, tato funkce musí zavolána jako první.
- `unsigned char i2c_start(void)` – Generuje START. Vrací 1, když je sběrnice volná, jinak vrací 0.
- `void i2c_stop(void)` – Generuje STOP. Ukončení přenosu, uvolnění sběrnice.
- `unsigned char i2c_read(unsigned char ack)` – Čte bajt ze sběrnice. Parametr `ack` udává, zda má být generováno potvrzení po přijmutí bajtu. (`ack = 1` generuj potvrzení, `ack = 0` nepotvrzuj).
- `unsigned char i2c_write(unsigned char data)` – Zapíše bajt na sběrnici. Vrací 1, když Slave potvrdil příjem, jinak vrací 0.

9.7 I²C a kompilátor AVR-GCC

- V knihovně AVR-libc bohužel nenalezneme funkce pro komfortní obsluhu komunikace po sběrnici I2C.
    - Zbývá si napsat funkce svoje vlastní nebo použít knihovnu třetí strany.

9.7.1 Knihovna TWI master

- Autor: Peter Fleury (http://homepage.hispeed.ch/peterfleury/)
- Vlastnosti knihovny:
    - Vyžaduje kompilátor AVG-GCC.
    - Verze se SW emulací I2C – použitelné na jakémkoliv AVR i bez HW rozhraní TWI, pro komunikaci lze použít jakékoliv 2 piny na MCU.
    - Verze s využitím HW rozhraní TWI – lze použít pouze u AVR s hardwarovým rozhraní TWI, použití pinů s funkcí SDA a SCL.
    - Používá se polling (čeká se na odvysílání a příjem).
    - AVR pouze ve funkci Master (AVR ve funkci Slave není podporováno).
- Knihovna se skládá z těchto souborů:
    - `i2cmaster.S` – implementace SW emulace I2C, napsáno v assembleru
    - `twimaster.c` – implementace funkcí pro obsluhu HW rozhraní TWI
    - `i2cmaster.h` – hlavičkový soubor s rozhraním knihovny
- Do projektu vkládáme vždy soubor `i2cmaster.h` a k němu soubor `i2cmaster.S` nebo `twimaster.c`, podle toho jakou implementaci komunikace I2C chceme použít.
    - Např. pokud chceme použít HW rozhraní TWI, tak do projektu vložíme soubory `i2cmaster.h` a `twimaster.c`.



Obrázek 124 Soubory pro komunikaci přes I2C sběrnici v projektu Atmel Studia 6.1

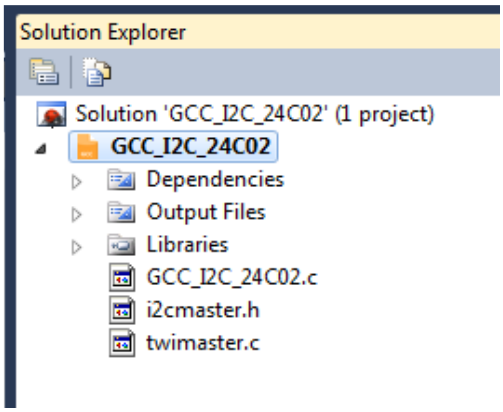
9.7.2 Funkce knihovny TWI master pro obsluhu sběrnice I2C

- `void i2c_init(void)`
  - Inicializace I2C rozhraní, tato funkce musí zavolána jako první.
- `unsigned char i2c_start(unsigned char addr)`
  - Generuje START a odešle adresu a směr komunikace.
  - Vrací 0, když je zařízení dostupné, jinak vrací 1.
- `unsigned char i2c_rep_start(unsigned char addr)`
  - Generuje opakovaný START a odešle adresu a směr komunikace. Vrací 0, když je zařízení dostupné, jinak vrací 1.
- `void i2c_start_wait(unsigned char addr)`
  - Generuje START a odešle adresu a směr komunikace. Pokud je zařízení BUSY, tak čeká až bude zařízení READY.
- `void i2c_stop(void)`
  - Generuje STOP. Ukončení přenosu, uvolnění sběrnice.
- `unsigned char i2c_readAck(void)`
  - Čte bajt ze sběrnice a potvrzuje ho.
- `unsigned char i2c_readNak(void)`
  - Čte bajt ze sběrnice a nepotvrzuje ho.
- `unsigned char i2c_read(unsigned char ack)`
  - Čte bajt ze sběrnice (přijme byte ze Slave).
  - Parametr `ack` udává, zda má být generováno potvrzení po přijmutí bajtu.
    - `ack` = 1 potvrď přijmutý byte
    - `ack` = 0 nepotvrzuj přijmutý byte
- `unsigned char i2c_write(unsigned char data)`
  - Zapíše bajt na sběrnici (odešle byte do Slave).
  - Vrací 0, když Slave potvrdil příjem, jinak vrací 1.

Žlutě jsou označeny nejdůležitější funkce, které obvykle postačují pro komunikaci.

9.7.3 Použití knihovny TWI master v programu

- Příklad je určen pro překladač AVR-GCC a MCU ATmega32.
- Příklad ukazuje komunikaci s externí sériovou EEPROM 24C02.
  - Zápis jednoho bajtu do EEPROM.
  - Čtení jednoho bajtu z EEPROM.
  - Sekvenční zápis více bajtů do EEPROM.
  - Sekvenční čtení více bajtů z EEPROM.
- Je použito HW rozhraní TWI.
- Do projektu je třeba vložit soubory:
  - `twimaster.c` – implementace funkcí pro obsluhu HW rozhraní TWI
  - `i2cmaster.h` – hlavičkový soubor s rozhraním knihovny

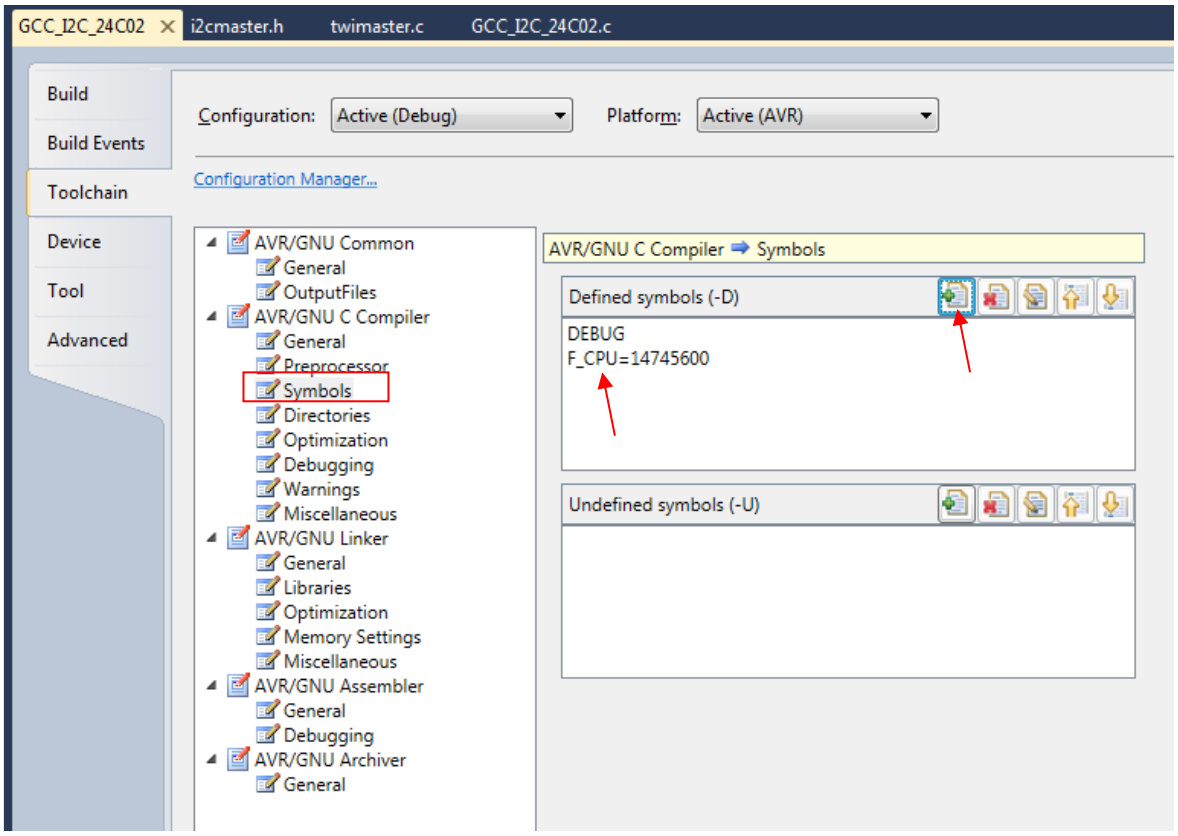


Obrázek 125 Soubory pro komunikaci přes I2C sběrnici v projektu Atmel Studia 6.1

- Dále je třeba:
  - Zadefinovat makro `F_CPU` s frekvencí MCU v Hz
    - Buď upravit makro `F_CPU` v souboru `twimaster.c`

```
/* define CPU frequency in hz here if not defined in Makefile */
#ifndef F_CPU
#define F_CPU 4000000UL
#endif
```

    - Nebo makro `F_CPU` vytvořit na úrovni projektu (v Atmel Studiu v menu Project – Properties – záložka Toolchain – AVR/GNU C Compiler – Symbols)



Obrázek 126 Přidání makra F_CPU do projektu

- Nastavit (změnit) komunikační rychlost I2C rozhraní
  - Standardně je nastaveno 100 kHz.
  - Změna se provede úpravou hodnoty makra v souboru `twimaster.c`

```
/* I2C clock in Hz */
#define SCL_CLOCK  100000L
```

  - Po zavolání funkce `i2c_init ()` dojde k nastavení předvolené komunikační rychlosti.

- **Kód (pro překladač AVR-GCC):**

```
#include <avr/io.h>
#include "i2cmaster.h"

#define Dev24C02  0xA2      // device address of EEPROM 24C02, see datasheet

int main(void)
{
       unsigned char ret;

       DDRA  = 0xff;                              // use all pins on port A for output
       PORTA = 0xff;                              // (active low LED's )

       i2c_init();                                // init I2C interface

       /* write 0x75 to eeprom address 0x05 (Byte Write) */
       ret = i2c_start(Dev24C02+I2C_WRITE);       // set device address and write mode
       if (ret)
       {
              /* failed to issue start condition, possibly no device found */
              i2c_stop();
              PORTA=0x00;                         // activate all 8 LED to show error */
       }
       else
       {
              /* issuing start condition ok, device accessible */
              i2c_write(0x05);   // write address 0x05
              i2c_write(0x75);   // write value 0x75, ret=0 -> Ok, ret=1 -> no ACK
              i2c_stop();        // set stop conditon = release bus

              /* write ok, read value back from eeprom address 0x05, wait until
              the device is no longer busy from the previous write operation */
              i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode
              i2c_write(0x05);                        // write address 0x05
              i2c_rep_start(Dev24C02+I2C_READ);       // set device address and read mode
              ret = i2c_readNak();                    // read one byte
              i2c_stop();

              PORTA = ~ret;                           // output byte on the LED's

              /* write 0x70,0x71,072,073 to eeprom address 0x00..0x03 (Page Write),
              wait until the device is no longer busy from the previous write operation */
              i2c_start_wait(Dev24C02+I2C_WRITE);     // set device address and write mode
              i2c_write(0x00);                        // write start address = 0
              i2c_write(0x70);                        // write data to address 0
              i2c_write(0x71);                        //    "    "   "    "    1
              i2c_write(0x72);                        //    "    "   "    "    2
              i2c_write(0x74);                        //    "    "   "    "    3
              i2c_stop();                             // set stop conditon = release bus

              /* write ok, read value back from eeprom address 0..3 (Sequencial Read),
              wait until the device is no longer busy from the previous write operation */
              i2c_start_wait(Dev24C02+I2C_WRITE);      // set device address and write mode
              i2c_write(0x00);                         // write address = 0
              i2c_rep_start(Dev24C02+I2C_READ);        // set device address and read mode
              ret = i2c_readAck();                     // read one byte form address 0
              ret = i2c_readAck();                     //  "    "    "    "     "    1
              ret = i2c_readAck();                     //  "    "    "    "     "    2
              ret = i2c_readNak();                     //  "    "    "    "     "    3
              i2c_stop();                              // set stop condition = release bus

              PORTA = ~ret;                            // output byte on the LED's
       }
       while (1);
}
```

9.8 I²C a Arduino

- Pro komfortní obsluhu sběrnice I2C je k dispozici knihovna **Wire**.
  - https://docs.arduino.cc/learn/communication/wire/#wire-library
  - Na začátek programu je nutné vložit její hlavičkový soubor:

    ```
    #include "Wire.h"
    ```
  - Pak je možné použít globální objekt **Wire** pro komunikaci, např. odeslání jednoho bajtu s hodnotou 0x7F do Slave na adrese 0x50:

    ```
    Wire.begin();
    Wire.beginTransmission(0x50);
    Wire.write(0x7F);
    Wire.endTransmission();
    ```
  - Funkcionalitu komunikace zajišťují ve skutečnosti dvě knihovny, které mezi sebou spolupracují:
    - **Wire**: `Wire.h`, `Wire.cpp` (user-friendly interface do programu)
    - **TWI**: `twi.h`, `twi.c` (low-level přístup na hardware)
      - Na AVR používá **T**wo **W**ire **I**nterface pro komunikaci na I2C.
    - Obě jsou součástí balíku zdrojových souborů Arduina pro dané MCU.
- Seznam Arduino funkcí pro obsluhu komunikace na I2C sběrnici:

To use this library:

```
#include <Wire.h>
```

Functions

- begin()
- end()
- requestFrom()
- beginTransmission()
- endTransmission()
- write()
- available()
- read()
- setClock()
- onReceive()
- onRequest()
- setWireTimeout()
- clearWireTimeoutFlag()
- getWireTimeoutFlag()

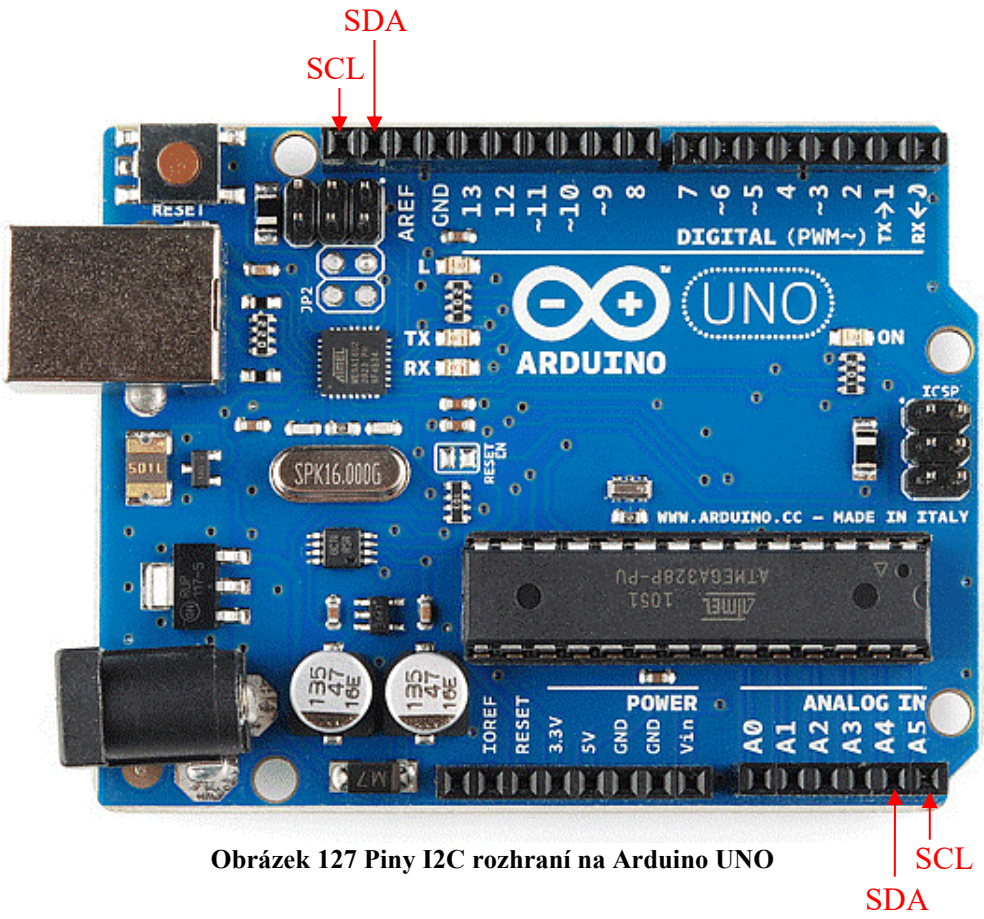
- Obvykle stačí používat jen několik funkcí
  - Inicializace:
    - `Wire.begin()`
    - Výchozí komunikační rychlost standardních 100 kbps.
  - Vysílání:
    - `Wire.beginTransmission()`
    - `Wire.write()`
    - `Wire.endTransmission()`
  - Příjem:
    - `Wire.requestFrom()`
    - `Wire.available()`
    - `Wire.read()`

- Arduino = různé typy základních desek a použitých MCU.
  - Mapování linek SDA a SCL závisí na konkrétním typu Arduina.
  - Některá Arduina mají více než jedno rozhraní I2C – ta je možné používat nezávisle na sobě.
  - Programové rozhraní zůstává stejné!

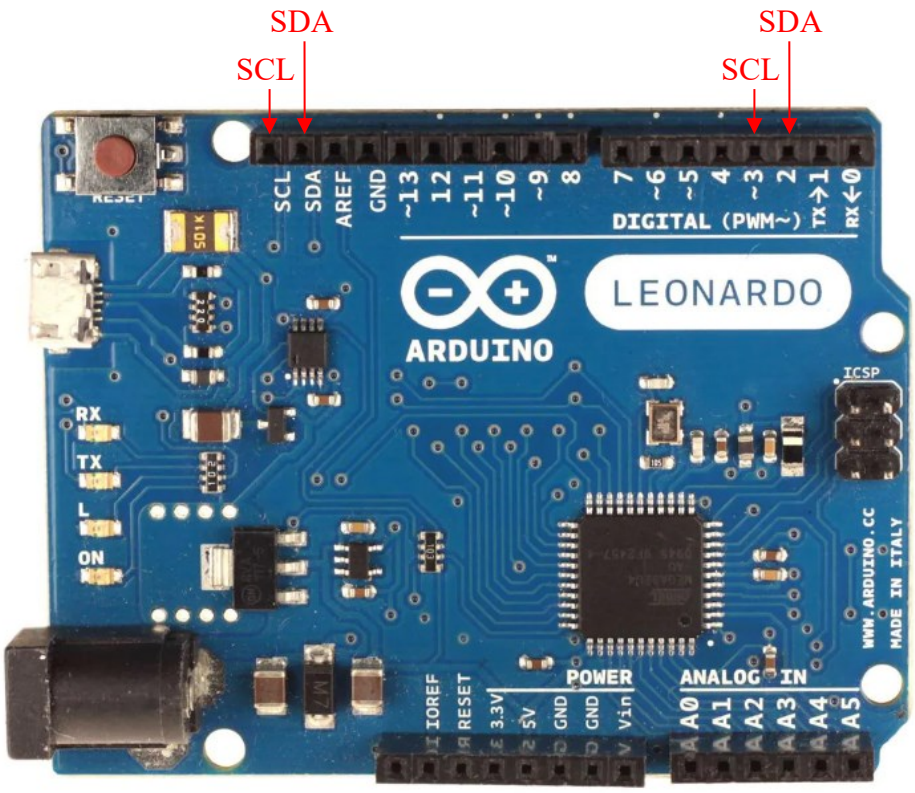
Tabulka 29 Mapování pinů SDA a SCL I2C rozhraní na Arduinu

Board	I2C Default	I2C1	I2C2	Notes
UNO R3, UNO R3 SMD, UNO Mini LE	A4(SDA), A5(SCL)			I2C also available on the SDA / SCL pins (digital header).
UNO R4 Minima, UNO R4 WiFi	A4(SDA), A5(SCL)	Qwiic: D27(SDA), D26(SCL)		I2C also available on the SDA / SCL pins (digital header).
UNO WiFi Rev2, Zero	20(SDA), 21(SCL)			
Leonardo, Micro, Yùn Rev2	D2(SDA), D3(SCL)			
Nano boards	A4(SDA), A5(SCL)			
MKR boards	D11(SDA), D12(SCL)			
GIGA R1 WiFi	20(SDA), 21(SCL)	D102(SDA1), D101 (SCL1)	D9(SDA2), D8 (SCL2)	Use `Wire1.begin()` for I2C1, and `Wire2.begin()` for I2C2.
Due	20(SDA), 21(SCL)	D70(SDA1), D71(SCL1)		Use `Wire1.begin()` for I2C1
Mega 2560 Rev3	D20(SDA), D21(SCL)			

- Arduino umí být MASTER nebo SLAVE
  - `Wire.begin()` – MASTER
  - `Wire.begin(0x63)` – SLAVE dostupný na I2C sběrnici pod adresou 0x63



Obrázek 127 Piny I2C rozhraní na Arduino UNO



Obrázek 128 Piny I2C rozhraní na Arduino Leonardo

9.8.1 Použití knihovny Wire v Arduino programu

- Příklady jsou určeny pro Arduino.
- Arduino je v roli Master.

Vysílání dat z Master do Slave

Master = Arduino
Slave = externí EEPROM 24LC256 (32 kByte) dostupná na sběrnici pod adresou 0x50

Zápis jednoho bajtu do EEPROM.

```
#include "Wire.h"

void setup()
{
        Wire.begin();        // inicializace I2C driveru, arduino bude I2C Master

        Serial.begin(9600);  // pro ladeni
}

void loop(void)
{
        char text[32];  // retezec pro tisk pres Serial

// zapis jednoho bajtu o hodnote 0x01 do externi EEPROM 24LC256 (32 kByte) na adresu 0x7FFF
// EEPROM je na sbernici I2C dostupna pod adresou 0x50

        Wire.beginTransmission(0x50);   // 0x50 = adresa Slave, pouze priprava Control Byte, Arduino fyzicky zatim nic na I2C nevysila

        Wire.write(0x7F);       // 0x7F = 1. datovy bajt k odeslani do Slave, pouze jeho ulozeni do vysilaciho bufferu Wire knihovny, Arduino fyzicky zatim nic na I2C nevysila

        Wire.write(0xFF);       // 0xFF = 2. datovy bajt k odeslani do Slave, pouze jeho ulozeni do vysilaciho bufferu Wire knihovny, Arduino fyzicky zatim nic na I2C nevysila

        Wire.write(0x01);       // 0x01 = 3. datovy bajt k odeslani do Slave, pouze jeho ulozeni do vysilaciho bufferu Wire knihovny, Arduino fyzicky zatim nic na I2C nevysila

        uint8_t status = Wire.endTransmission();// az tato funkce spusti fyzicke vysilani na I2C
                                                // mozne navratove hodnoty funkce:
                                                //      0 - uspech
                                                //      1 - vysilaci buffer je plny
                                                //      2 - NACK po odvysilani adresy
                                                //      3 - NACK po odvysilani dat, Slave potvrdil prijem adresy, ale nepotvrdil prijem dat
                                                //      4 - jina chyba

        // tisk uspechu/neuspechu pres USB
        snprintf(text,sizeof(text),"status=0x%02X\n",status);
        Serial.print(text);

        delay(1000);
}
```

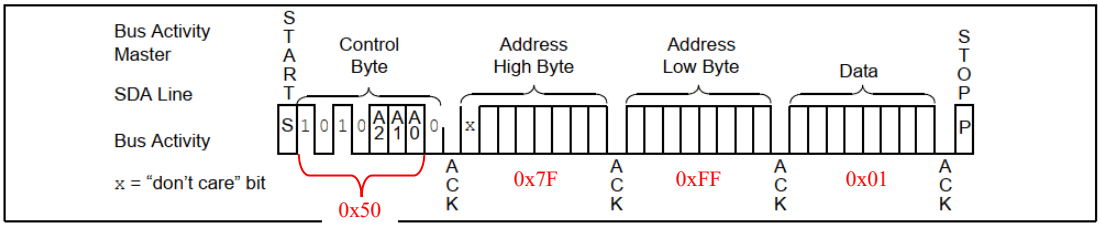
Control Byte

Address High Byte

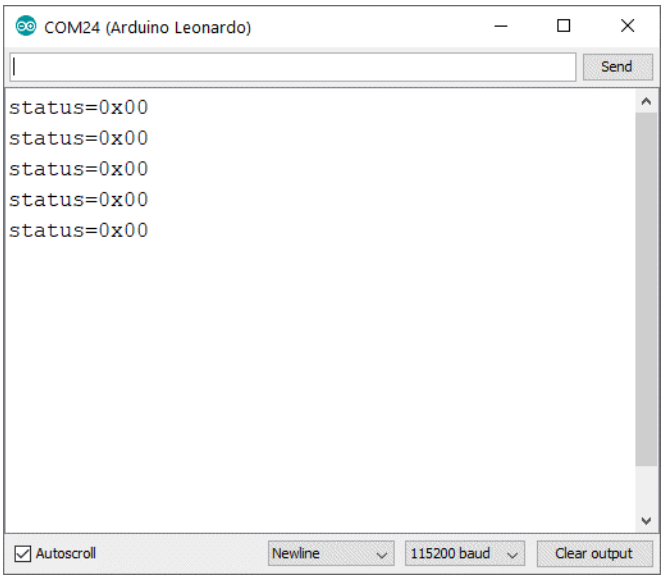
Address Low Byte

Data

FIGURE 6-1: BYTE WRITE



Obrázek 129 Zápis jednoho bajtu na vybranou adresu do I2C EEPROM 24LC256

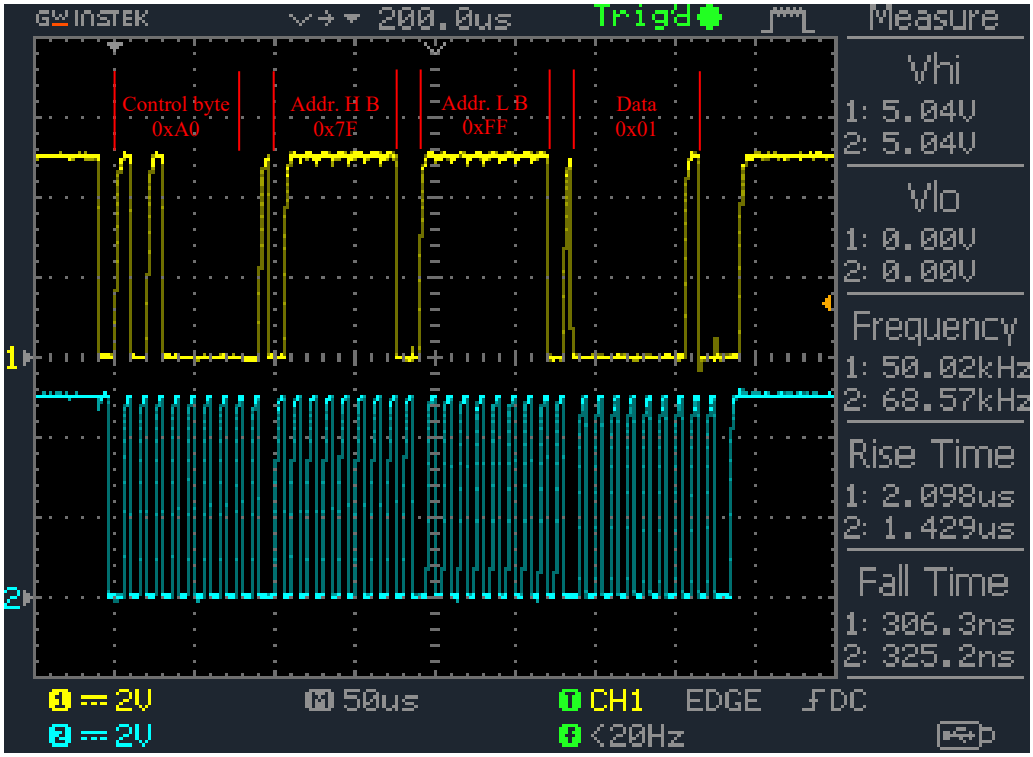


Obrázek 130 Výpis do terminálu – 5x za sebou úspěšný zápis dat přes I2C do externí EEPROM

Průběh komunikace na osciloskopu

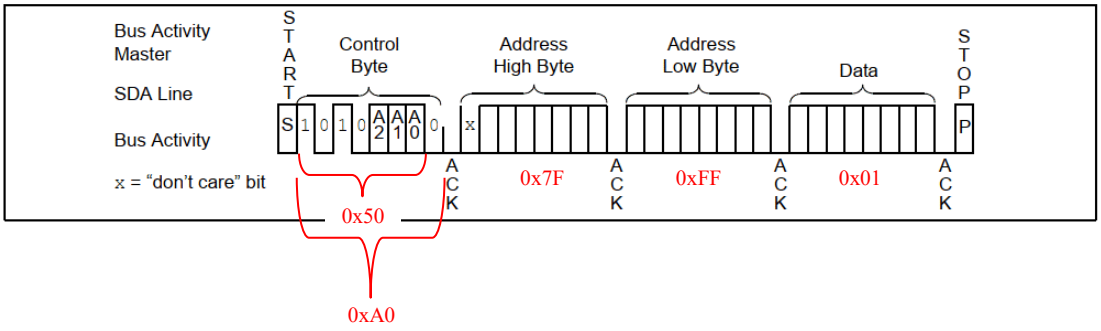
CH1 = SDA

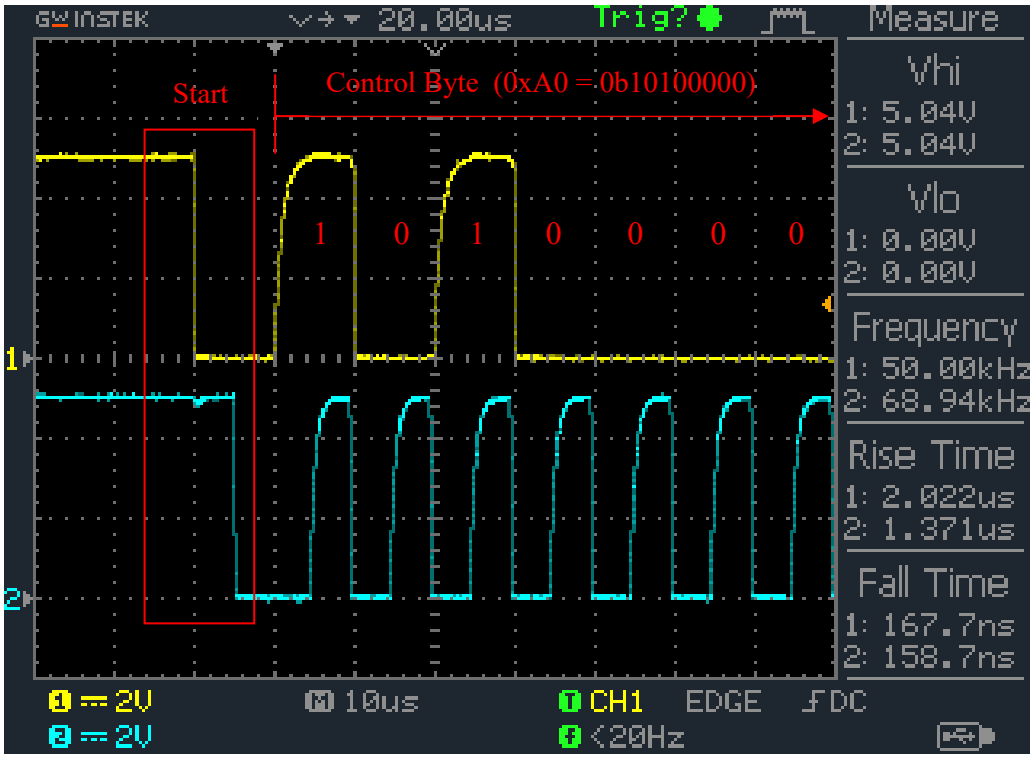
CH2 = SCL



Obrázek 131 Průběh komunikace na I2C sběrnici (CH1 = SDA, CH2 = SCL)

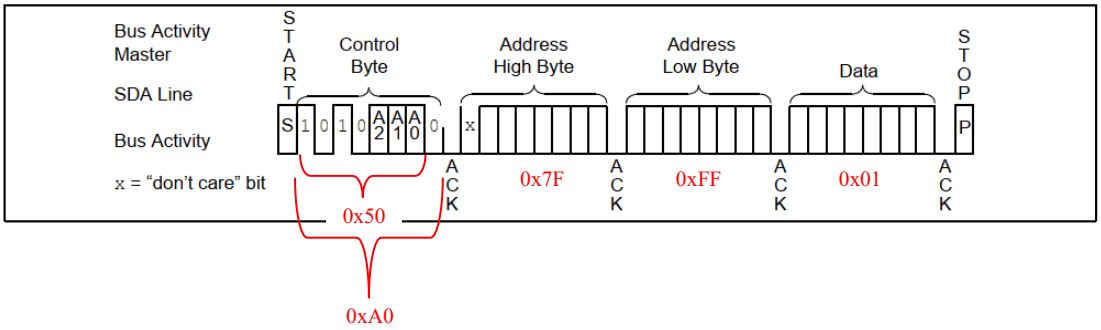
FIGURE 6-1: BYTE WRITE

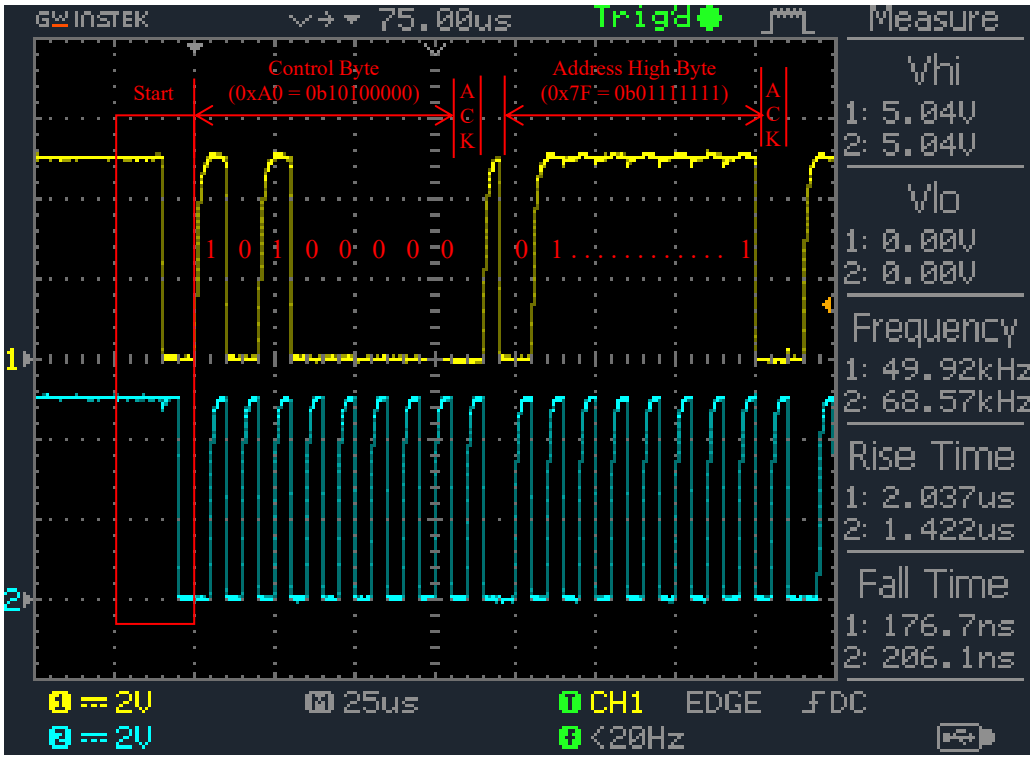




Obrázek 132 Detail začátku komunikace na I2C sběrnici

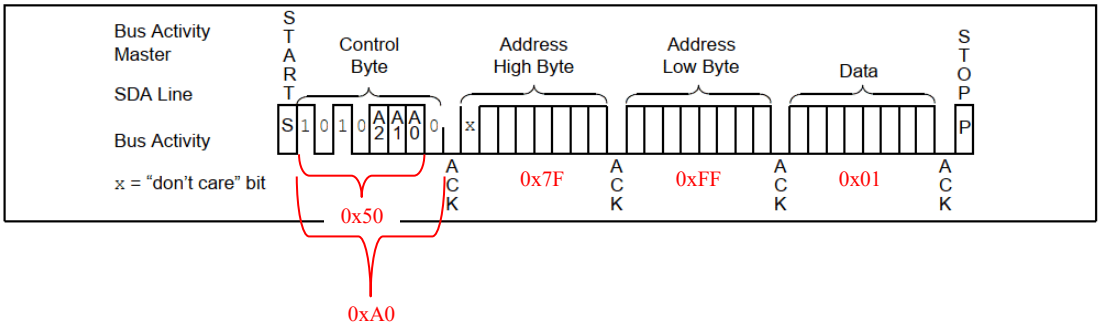
FIGURE 6-1: BYTE WRITE

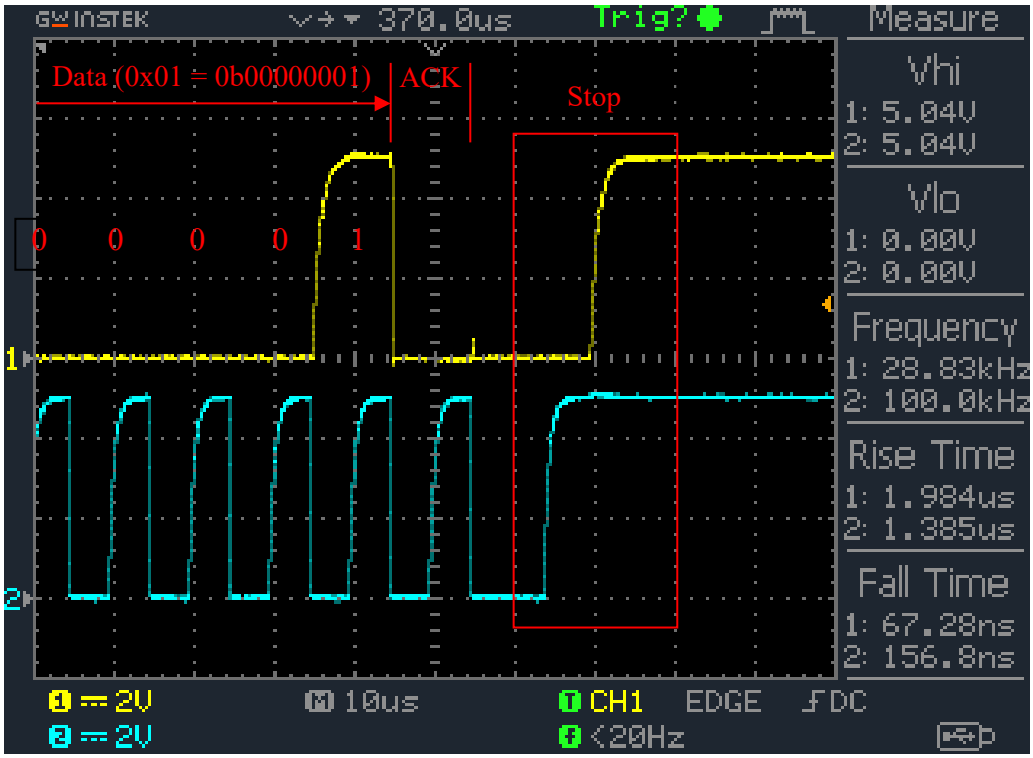




Obrázek 133 Detail prvních dvou bajtů na I2C sběrnici (Control Byte následovaný Address High Byte)

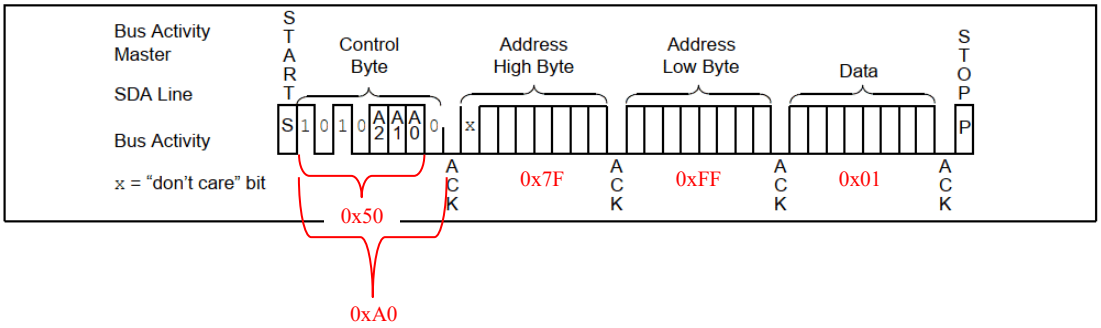
FIGURE 6-1: BYTE WRITE





Obrázek 134 Detail konce komunikace na I2C sběrnici

FIGURE 6-1: BYTE WRITE



Další pěkné vysvětlení souvislostí mezi knihovními funkcemi Wire a průběhy signálů na linkách SDA a SCL je např. zde: https://rheingoldheavy.com/arduino-wire-library/

Příjem dat ze Slave do Master

Master = Arduino
Slave = externí EEPROM 24LC256 (32 kByte) dostupná na sběrnici pod adresou 0x50

Vyčtení jednoho bajtu z EEPROM.

```
#include "Wire.h"

void setup()
{
        Wire.begin();        // inicializace I2C driveru, arduino bude I2C Master

        Serial.begin(9600);  // pro ladeni
}

void loop(void)
{
        char text[32];  // retezec pro tisk pres Serial

// vycteni jednoho bajtu z externi EEPROM 24LC256 (32 kByte) z adresy 0x7FFF

        // nejprve nastaveni adresy, ze ktere se ma vycist hodnota (zpusob cteni RANDOM READ, viz datasheet k EEPROMce)
        Wire.beginTransmission(0x50);   // 0x50 = adresa Slave, pouze priprava hlavicky, Arduino fyzicky zatim nic na I2C nevysila
        Wire.write(0x7F);       // 0x7F = horni byte adresy, ze ktere chci z EEPROM cist hodnotu
        Wire.write(0xFF);       // 0xFF = dolni byte adresy, ze ktere chci z EEPROM cist hodnotu

        Wire.endTransmission(); // az tato funkce spusti fyzicke vysilani na I2C


        // a az nyni vycteni hodnoty z EEPROM z prednastavene adresy

        uint8_t len = Wire.requestFrom(0x50, 1);// zadost na cteni ze Slave s adresou 0x50, 1 = pocet bajtu v odpovedi od Slave (Arduino pak generuje CLK pro vycteni vsech pozadovanych bajtu ze Slave), musi byt v rozmezi 1 az 32

        if (len == 1) // pro jistotu kontrola zda byl prijem uspesny
        {
                // vyzvednuti hodnoty z prijimaciho bufferu
                uint8_t data = (uint8_t)Wire.read();

                // tisk hodnoty vyctene z EEPROM
                snprintf(text,sizeof(text),"len=0x%02X data=0x%02X\n",len,data);
                Serial.print(text);
        }
        else // neco se pokazilo
        {
                snprintf(text,sizeof(text),"len=0x%02X chyba\n",len);
        }

        delay(1000);
}
```

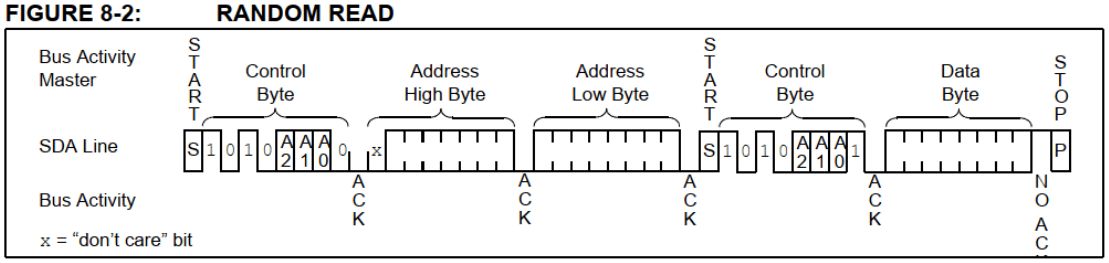
Control Byte

Address High Byte

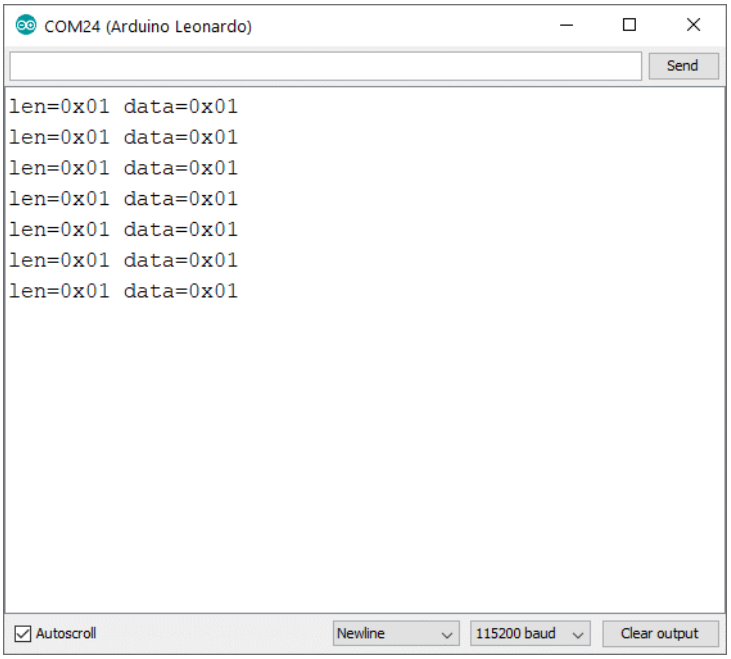
Address Low Byte

Data Byte

FIGURE 8-2: RANDOM READ



Obrázek 135 Čtení jednoho bajtu z vybrané adresy z I2C EEPROM 24LC256



Obrázek 136 Výpis do terminálu – 7x za sebou úspěšné vyčtení hodnoty z externí EEPROM

Ze Slave je možné vyčítat i více bajtů než 1.
Níže je příklad vyčtení 4 po sobě jdoucích bajtů z EEPROM od adresy 0x7FFC do 0x7FFF. Změny oproti předchozí verzi jsou žlutě označeny.

```
void loop(void)
{
        char text[32];  // retezec pro tisk pres Serial

// vycteni jednoho bajtu z externi EEPROM 24LC256 (32 kByte) z adresy 0x7FFF

        // nejprve nastaveni adresy, ze ktere se ma vycist hodnota (zpusob cteni RANDOM READ, viz datasheet k eepromce)
Control Byte
        Wire.beginTransmission(0x50);   // 0x50 = adresa Slave, pouze priprava hlavicky, Arduino fyzicky zatim nic na I2C nevysila
Address High Byte
        Wire.write(0x7F);       // 0x7F = horni byte adresy, ze ktere chci z EEPROM cist hodnotu
Address Low Byte
        Wire.write(0xFC);       // 0xFC = dolni byte adresy, ze ktere chci z EEPROM cist hodnotu

        Wire.endTransmission(); // az tato funkce spusti fyzicke vysilani na I2C

        // a az nyni vycteni hodnoty z EEPROM z prednastavene adresy

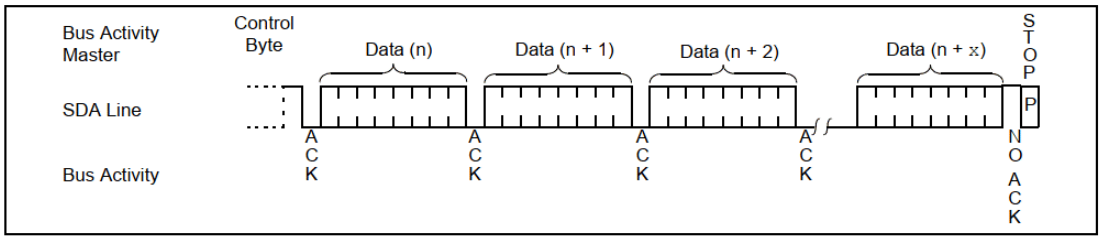
        uint8_t len = Wire.requestFrom(0x50, 4);// zadost na cteni ze Slave s adresou 0x50, 4 = pocet bajtu v odpovedi od Slave (Arduino pak generuje CLK pro vycteni vsech pozadovanych bajtu ze Slave), musi byt v rozmezi 1 az 32

        // vycteni vsech prijmutych dat z prijimaciho bufferu Arduina a jejich odeslani přes USB
        snprintf(text,sizeof(text),"len=%02d\n",len);
        Serial.print(text);
        for (uint8_t i=0; i<len; i++)
        {
Data(i)     int data = Wire.read(); // vraci -1 pokud je prijimaci buffer prazdny

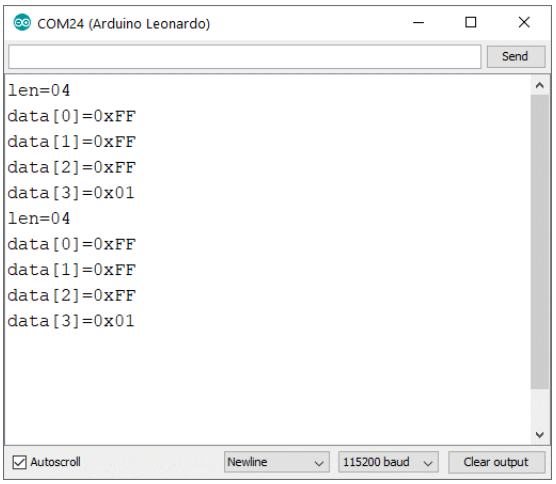
                if (data != -1) // radeji kontrola prazdnosti prijimaciho bufferu
                {
                        // zpracovani vycteneho bajtu (zde jen jeho tisk pres Serial)
                        snprintf(text,sizeof(text),"data[%d]=0x%02X\n",i,(uint8_t)data);
                        Serial.print(text);
                }
                else
                        break;  // predcasne ukonceni smycky, protoze prijimaci buffer je prazdny
        }

        delay(1000);
}
```

FIGURE 8-3: SEQUENTIAL READ



Obrázek 137 Čtení jednoho bajtu z vybrané adresy z I2C EEPROM 24LC256



Obrázek 138 Výpis do terminálu – 2x za sebou úspěšné vyčtení 4 hodnot z externí EEPROM

9.8.2 Shrnutí

- **Vysílání (Master → Slave)**
  - Skládá se z volání 3 funkcí.

```
Wire.beginTransmission(i2c_adresa_slave);

Wire.write(datovy_bajt);
.
.
.
Wire.write // zopakovat vícekrát, pokud chci do Slave přenášet více bajtů
.
.
.
uint8_t status = Wire.endTransmission(); // status == 0 znamena uspech
```

  - Pokud mám data pro odeslání uložena v poli, je snadnější použít tuto variantu funkce: `Wire.write(const uint8_t *data, size_t quantity)`

- **Příjem (Slave → Master)**
  - Skládá se z volání 2 až 3 funkcí.
  - **1. způsob**

```
uint8_t len = Wire.requestFrom(i2c_adresa_slave, pocet_prijimanych_bajtu);
// pocet_prijimanych_bajtu musi byt 1 az 32
// funkce vraci pocet prijmutych bajtu

// postupne vyzvednuti prijmutych bajtu z prijimaciho bufferu
for (uint8_t i=0; i<len; i++)
{
        int data = Wire.read(); // vraci -1 pokud je prijimaci buffer prazdny

        if (data != -1) // radeji kontrola prazdnosti prijimaciho bufferu
        {
                // buffer obsahuje data
                // zpracovani vycteneho bajtu „data“
        }
        else
         break; // predcasne ukonceni smycky, protoze prijimaci buffer je prazdny
}
```

- **2. způsob**

```
uint8_t len = Wire.requestFrom(i2c_adresa_slave, pocet_prijimanych_bajtu);
// pocet_prijimanych_bajtu musi byt 1 az 32
// funkce vraci pocet prijmutych bajtu

// postupne vyzvednuti prijmutych bajtu z prijimaciho bufferu
while (Wire.available()) // obsahuje prijimaci buffer data?
{
        int data = Wire.read(); // vraci -1 pokud je prijimaci buffer prazdny

        // zpracovani vycteneho bajtu „data“
}
```

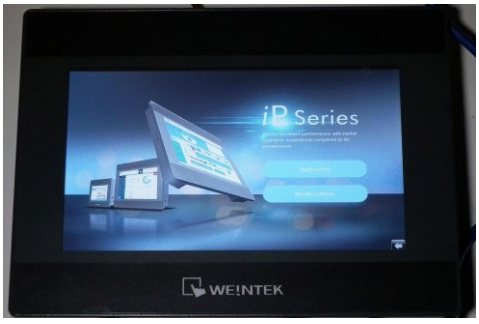
10 Sériová linka RS-232

- Standard EIA-232 (Electronic Industries Alliance Recommended Standard 232)
- Sériové rozhraní, plně duplexní, komunikace mezi zařízeními na delší vzdálenost (jednotky až desítky metrů).
- Pouze 2 zařízení na jedné komun. lince (nelze připojit více zařízení na jednu linku).
- Maximální komunikační rychlost obvykle 1-ky až 10-ky kB za sekundu (s rostoucí vzdáleností klesá).
- Standard EIA-232 definuje pouze fyzickou vrstvu = elektrické a mechanické parametry přenosu po sériovém asynchronním rozhraní.
    - elektrické parametry:
        - napěťové úrovně
        - impedanční poměry (odolnost proto zkratům, rušení, ...)
    - mechanické parametry:
        - specifikace konektorů
        - zapojení kabelů
- standard zavádí dva pojmy - **DTE** a **DCE**:
    - **DTE** (Data Terminal Equipment) = **počítač**
    - **DCE** (Data Circuit-terminating Equipment) = **koncové zařízení** (např. modem)
- Dříve (90. léta 20. století) běžné rozhraní na PC pro připojení myší a modemů.
    - Postupem času bylo rozhraní RS-232 vytlačeno rozhraním USB.
    - Dnes rozhraní RS-232 na PC nenajdete, pro komunikaci mezi PC a zařízením po RS-232 je nutné použít externí převodník USB/RS-232.



Obrázek 139 Převodník USB-RS232

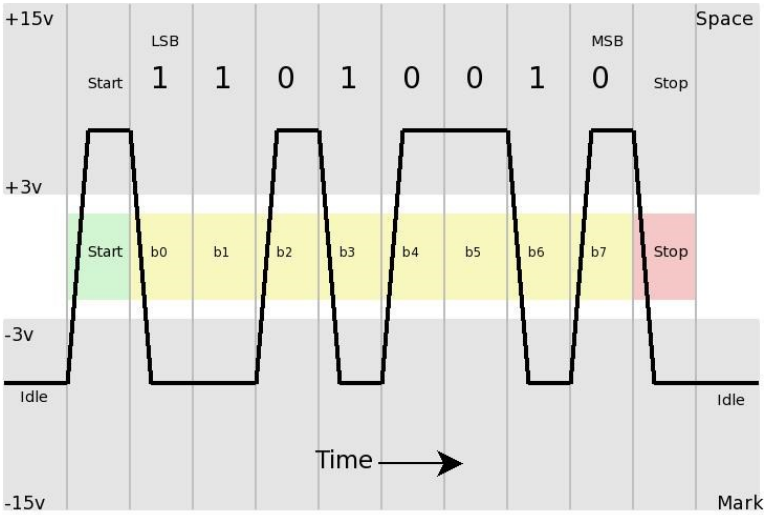
    - V průmyslu se rozhraní RS-232 stále používá.





10.1 Parametry fyzické vrstvy

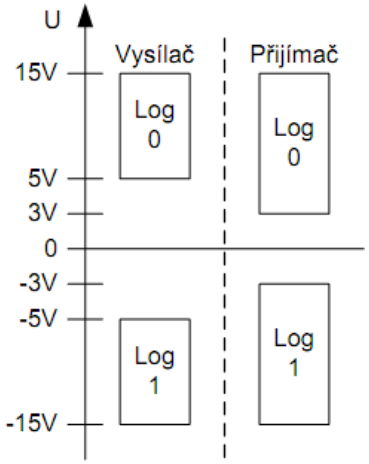
- data jsou přenášena jako posloupnost bitů



Obrázek 140 Přenos jednoho bajtu po sběrnici RS-232

- logické úrovně - negativní logika

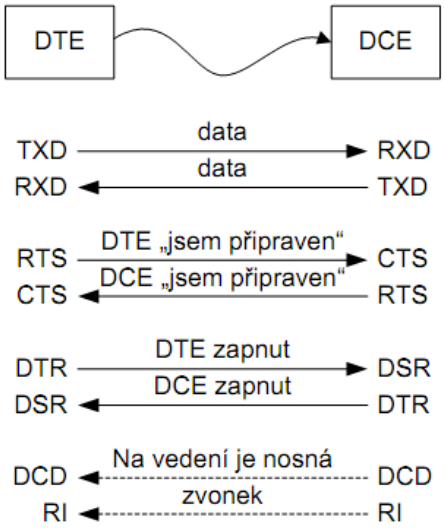
úroveň	označení	vysílač	přijímač
Log 0	Space	+5 až +15 V	+3 až +25 V
Log 1	Mark	–5 až –15 V	–3 až –25 V
nedefinováno			–3 až +3 V



  - vysílač běžně ±10 V (používá se napěťový zdvojovač 5 V a invertor)
- budiče jsou odolné proti zkratu (omezení proudu na 20 mA)
- impedance vedení Z = 300 Ω
- Max. délka vedení 15 m při přenosové rychlosti 19200 Bd (150 m při 9600 Bd), resp. nesmí být překročena max. celková kapacita vodiče 2500 pF.

10.2 Přenášené signály

- **GND** (vztažný potenciál)
- **TXD** (Transmitted Data) – vysílání dat
- **RXD** (Received Data) – příjem dat
- **RTS** (Request To Send) – log. 1 na tomto výstupu signalizuje, že DTE chce vysílat (u převodníků RS-232/RS-485 se tento signál používá pro přepínání směru komunikace)
- **CTS** (Clear To Send) – log. 1 na tomto vstupu DCE signalizuje, že DTE může vysílat data
- **DTR** (Data Terminal Ready) - log. 1 na tomto výstupu zařízení signalizuje protistraně svojí připravenost
- **DSR** (Data Set Ready) – log. 1 na tomto vstupu protistrana signalizuje, že je připravena
- **DCD** (Data Carrier Detect) – DCE (modem) signalizuje, že detekoval nosný signál na vedení
- **RI** (Ring Indicator) – signalizace příchozího hovoru (pokud je DCE modem)

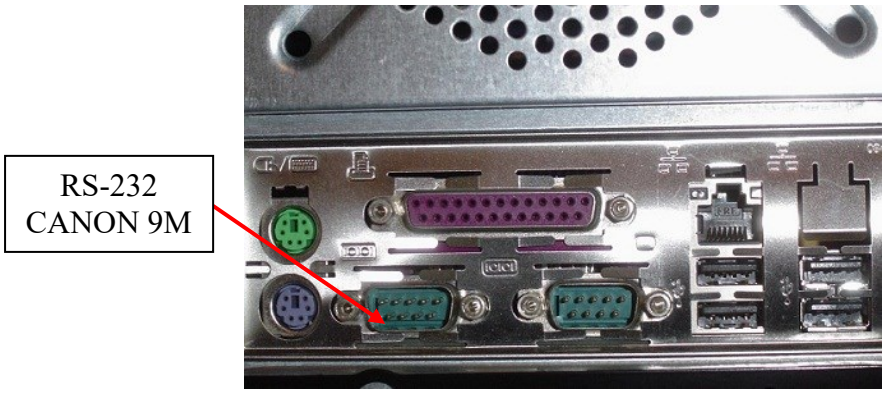


Obrázek 141 Propojení DTE s DCE u RS-232

- Signály RTS, CTS, DTR, DSR, DCD, RI se používají pro řízení přenosu (určují kdy se začnou data přenášet) a nemusejí být použity nebo mohou sloužit pro napájení zařízení.
- **Ve většině běžných případů (kdy není použito hardwarového řízení přenosu) se používají pouze vodiče TXD a RXD.**
- **Plně duplexní přenos dat (přenos dat oběma směry současně).**

10.3 Konektory

- cca 10 typů, nejznámější D-sub 9 a 25 (někdy nazývané Canon)
- DTE: samec (Male —○)
- DCE: samice (Female —( )
- přiřazení signálů na piny je standardizováno – viz níže kabely



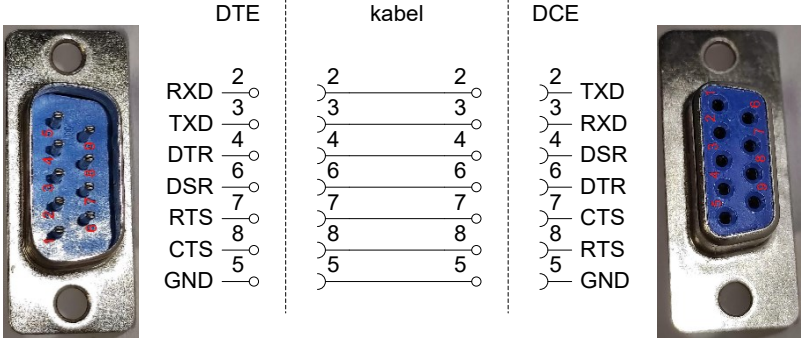
Obrázek 142 Rozhraní RS-232 na starším PC (konektor Canon 9M)

10.4 Kabely

- spousty variant dle osazených konektorů
- nemusí být vždy přítomny všechny signály (min. varianta RXD, TXD, GND)

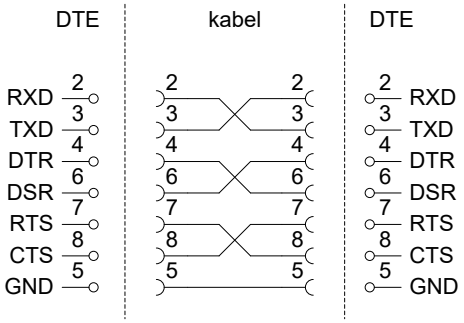


- **prodlužovací kabel („1 : 1“)**
  - spojení DTE – DCE (např. PC a nějaké zařízení)
  - kabel s konektory samec – samice (CANON 9M – CANON 9F)



Kabel je 1:1. Konektory na zařízeních jsou opačně zapojené pro realizaci správného křížení.

- **křížený kabel („cross cable“ nebo „null-modem“)**
  - spojení DTE – DTE (např. 2 PC mezi sebou)
  - kabel s konektory samice – samice (CANON 9F – CANON 9F)

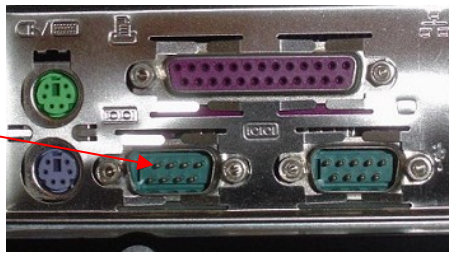


Kabel je křížený.
Konektory na zařízeních jsou shodně zapojené.

- Pozor: kabel CANON 9F – CANON 9F nemusí být vždy křížený!!!

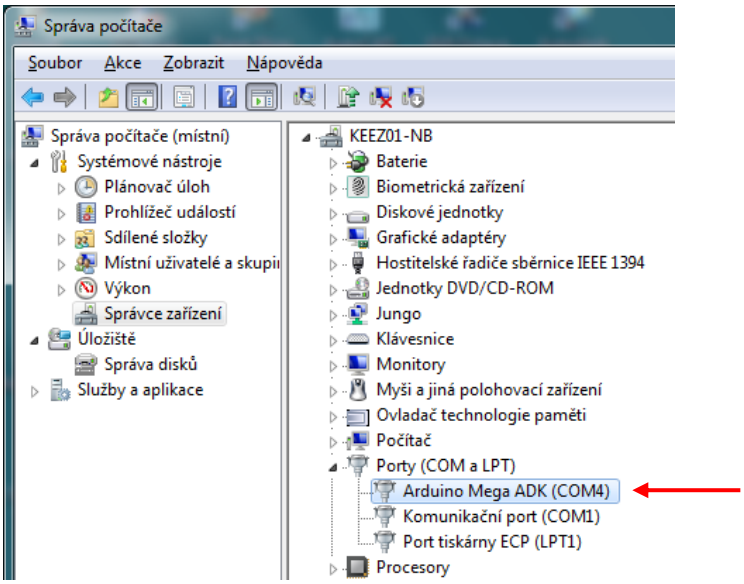
10.5 RS-232 na PC

- = v PC je řadič UART 16550 + konvertor úrovní TTL ↔ RS232
- Sériový port označen jako **COMx**, kde x = 1,2 , ...
- Vyveden na konektor CANON 9M.
- Dnes je RS-232 na PC vytlačeno rozhraním USB.
  - Nutnost použití převodníků USB / RS232.





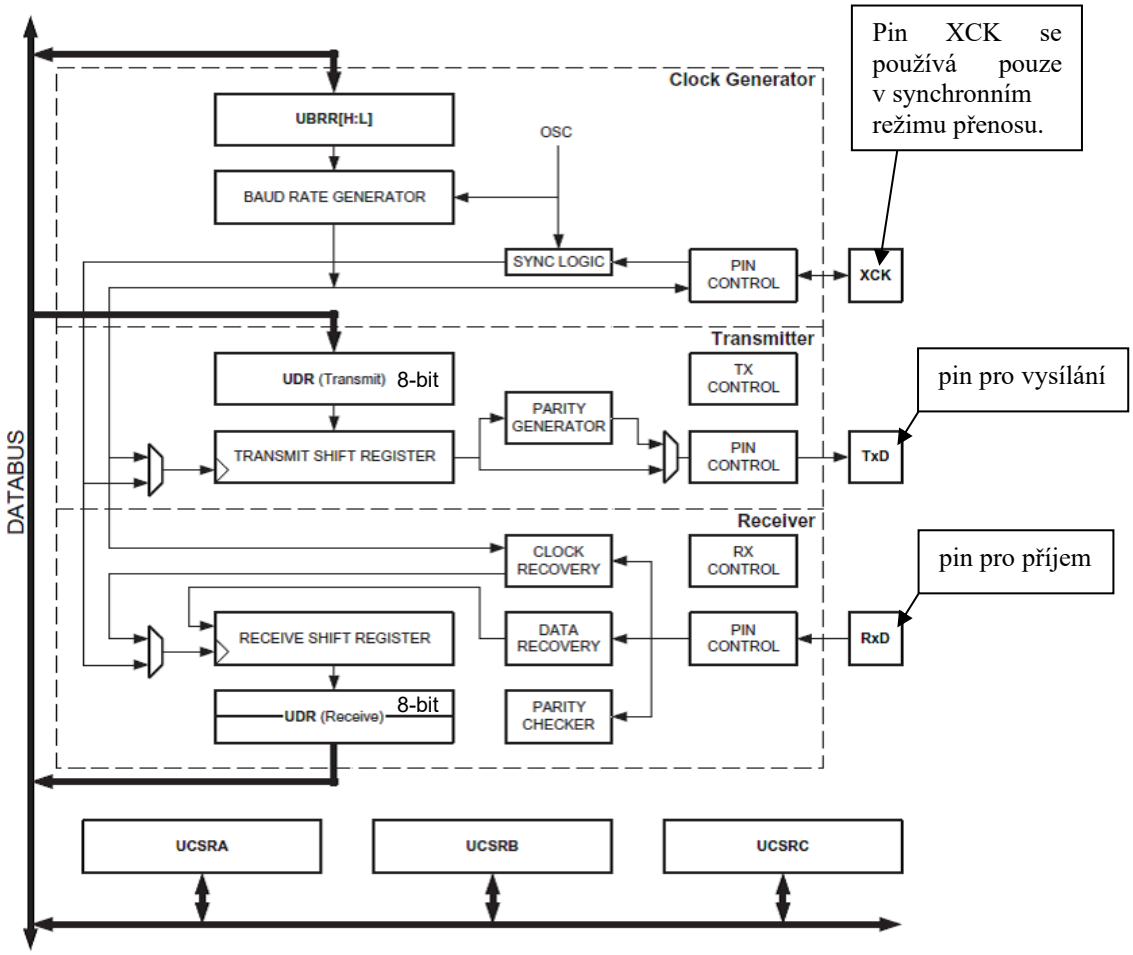
  - Tyto převodníky obvykle obsahují čipy od firem **Maxim, FTDI, Prolific** apod.
  - Po zasunutí převodníku do USB a se ve správci zařízení objeví nový **COM port** (virtuální COM port).



- Převodník je vůči aplikaci na PC naprosto transparentní. Aplikace, která uměla komunikovat pouze přes RS-232 i nadále funguje, akorát se data z PC posílají/přijímají fyzicky přes USB.
- Před prvním použitím převodníku je potřeba nainstalovat k němu určené drivery (Virtual Com Port).

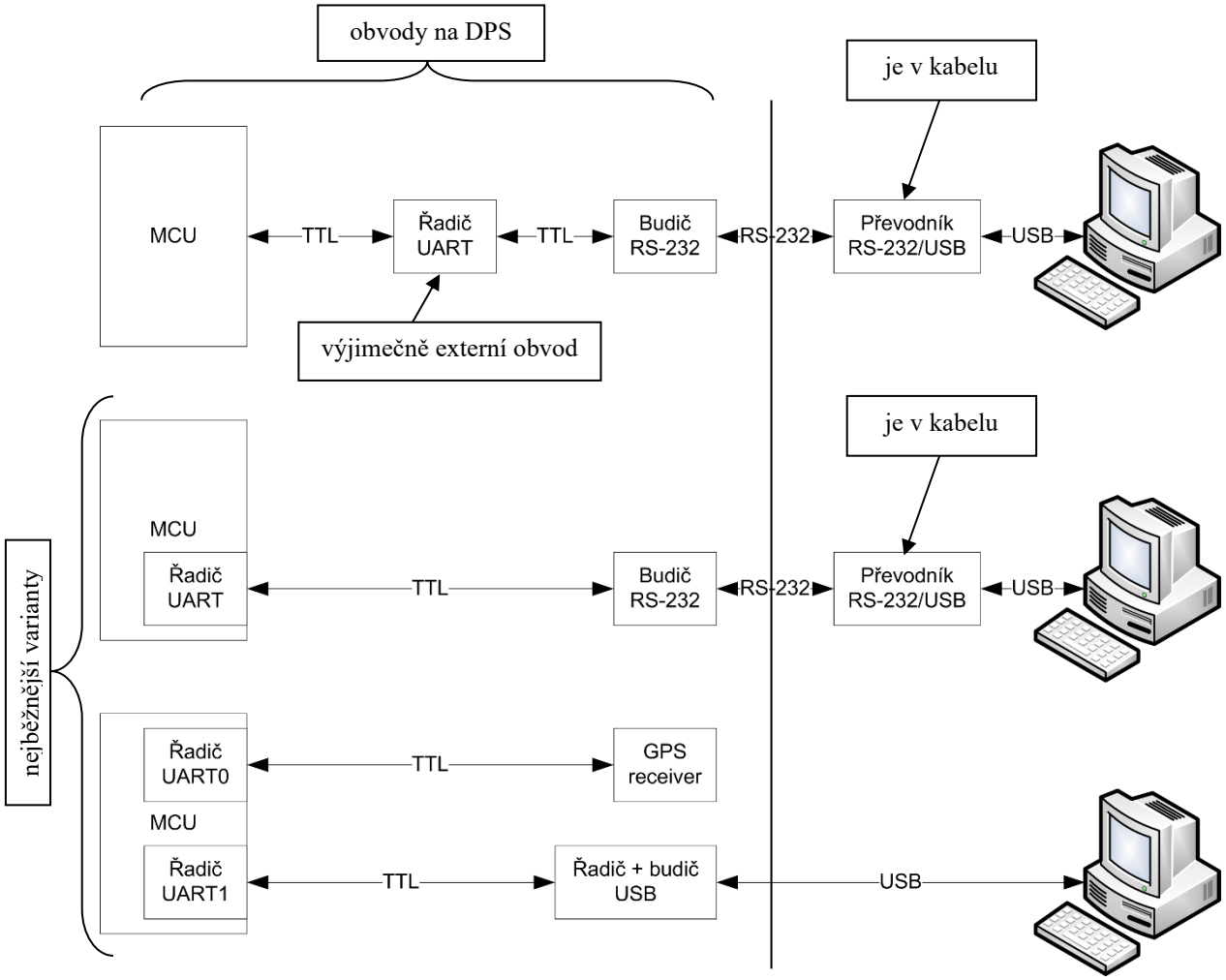
11 UART

- UART = Universal Asynchronous Receiver/Transmitter
- obvod převádějící 8-bit paralelní data do sériové formy (TX) a opačně (RX)
- linky TX (vysílání) a RX (příjem)



Obrázek 143 Blokové schéma obvodu UART v MCU AVR ATmega32

- **UART není RS-232**
  - **UART** = **definuje linkovou vrstvu**, tj. formát přenášených dat (tvar rámce), přenosovou rychlost, způsob přístupu na sběrnici apod.
    - Linkovou vrstvu realizuje **řadič** = integrovaný obvod.
    - Napěťové úrovně Tx, Rx obvykle TTL/CMOS.
  - **RS-232** = **definuje fyzickou vrstvu**, tj. kódování bitů, napěťové úrovně bitů, konektory, typ vedení.
    - Fyzickou vrstvu realizuje **budič** = integrovaný obvod.
    - Napěťové úrovně dle standardu RS-232.
  - Fyzická vrstva pod UART může být různá (RS-232, USB, ….).



Obrázek 144 UART, RS-232, USB

- Řadič UART bývá obvykle integrovaný v MCU (výjimečně mimo MCU).

- **Na sběrnici UART mohou být vždy pouze 2 zařízení.**
  - To je značná nevýhoda proti sběrnicím typu SPI nebo I2C.
  - Pokud je třeba komunikovat s více zařízeními, je třeba mít více linek UART nebo jednu linku hardwarově multiplexovat (což komplikuje návrh HW i SW).

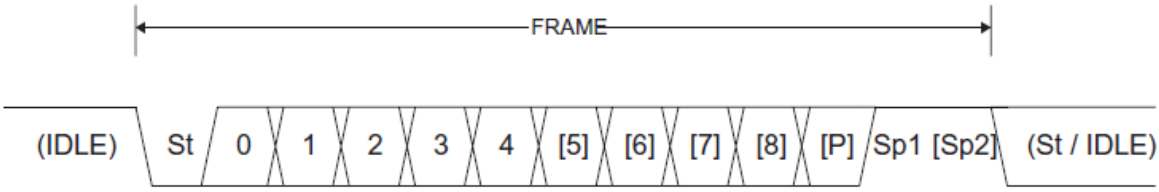
- Obvody komunikující přes UART:
  - různé snímače,
  - GPS přijímače,
  - různé komunikační převodníky (Bluetooth, Wifi apod.),
  - grafické a dotykové LCD,
  - apod. (portfolio je široké)





11.1 Formát rámce UART (frame format)

- = jak se přenáší data



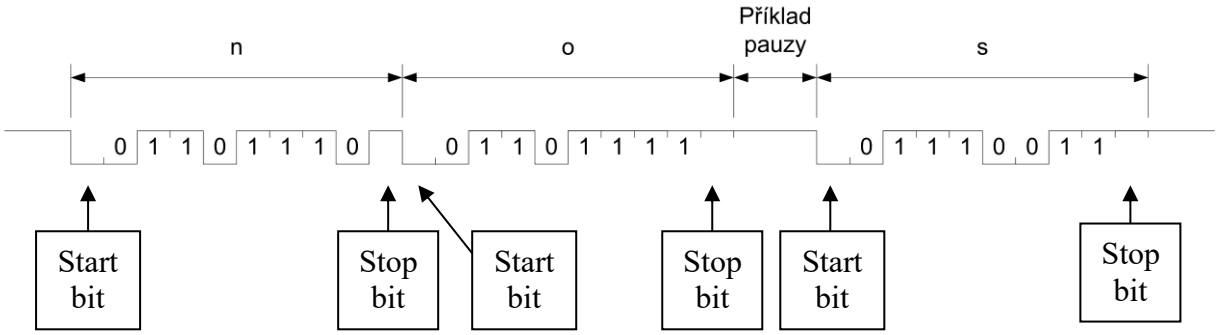
Obrázek 145 Formát rámce UART

- **IDLE** – klid na sběrnici (nedochází k přenosu dat) – vždy v log. 1
- **St** - start bit (log. 0) – zahájení přenosu
- **datové bity** b0÷ b8
  - vlastní užitečná informace
  - standardní pořadí vysílání bitů: LSB first
  - počet datových bitů 5 až 9
- **P** - paritní bit (volitelně)
  - jednoduchá forma zabezpečení přenosu
  - paritu je možné nastavit na lichou / sudou
  - při vysílání se sečte počet datových bitů v logické 1 a doplní se paritním bitem tak, aby byla zachována předem dohodnutá podmínka sudého nebo lichého počtu bitů v log. 1
  - přijímač spočítá paritu z přijatých dat a porovná jí s přijatým paritním bitem
  - lichá (odd): P = 1, je-li počet 1 v datových bitech + paritní bit = liché číslo
  - sudá (even): P = 1, je-li počet 1 v datových bitech + paritní bit = sudé číslo
  - řadič UART paritu počítá jako XOR datových bitů:

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

- **Sp** - stop bit (log. 1) – ukončení přenosu
  - délka 1; 1,5; 2 bitové intervaly (důvod prodloužení: více času na zpracování)
- **nejběžnější nastavení = 8N1 (8b, žádná P, 1 stop bit)**

- **Příklad:**
  přenos řetězce „nos“ (ASCII) v 8N1
  ‘n‘: 0b01101110 (110 dekadicky)
  ‘o‘: 0b01101111 (111 dekadicky)
  ‘s‘: 0b01110011 (155 dekadicky)



11.2 Přenosová rychlost (Baud Rate) v_B

- Odvozuje se od doby trvání jednoho bitu T_B

$$v_B = \frac{1}{T_B}\ [Bd; s]$$

- Jednotka: **Baud** („bód“) = [1/s]
- Standardizované rychlosti komunikace: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200 Bd
- Přenosová rychlost „užitečných“ dat je o trochu nižší, protože se v rámci přenášejí i neužitečné bity (start bit, stop bit, případně parita).
- V níže uvedené tabulce jsou uvedeny maximální dosažitelné rychlosti přenosu v kB/s a doba potřebná na odeslání jednoho rámce ve formátu 8N1:

Tabulka 30 Max. dosažitelné přenosové rychlosti (UART, rámec 8N1)

Baud Rate	Rychlost přenosu dat [kB/s]	Doba potřebná na odeslání jednoho rámce [μs]
9600	0,96	1042
38400	3,84	261
57600	5,76	174
115200	11,52	87

$$\text{doba odeslání rámce } [\mu s] = \frac{10}{\text{Baud Rate}} \cdot 10^6 \qquad (67)$$

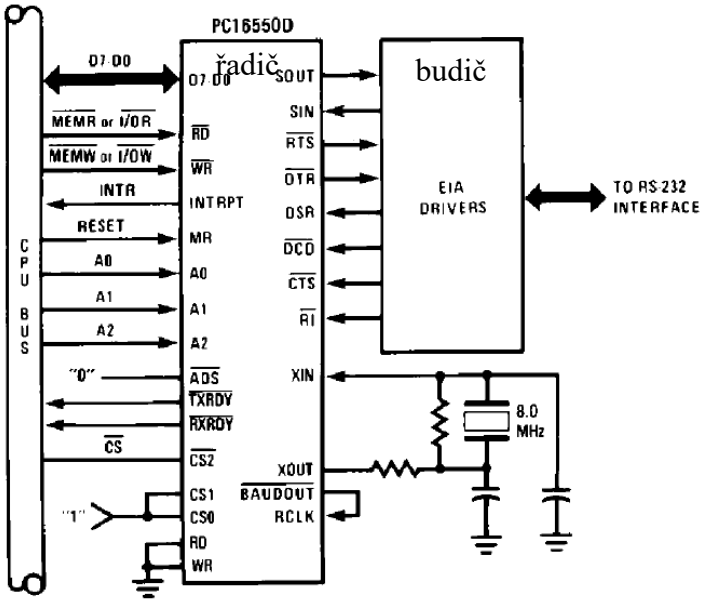
Kde: $10 = 8 + 2 \rightarrow$ 8 bitů data, 1 start bit, 1 stop bit

11.3 Handshaking

- = potvrzení příjmu a potvrzení začátku přenosu.
- Princip:
    - Vysílač dá přijímači na vědomí, že chce vysílat.
    - Přijímač potvrdí vysílači, že může začít vysílat.
- Hardwarový handshaking
    - Vyžaduje vodiče navíc (min. RTS, CTS).
    - RTS - vysílač dá přijímači na vědomí, že chce vysílat.
    - CTS - přijímač potvrdí vysílači, že může začít vysílat.
- Softwarový handshaking
    - Úspora vodičů (stačí RXD, TXD, GND).
    - Probíhá na úrovni komunikačních protokolů (ZMODEM, KERMIT...).
    - Pomocí běžného datového kanálu přijímač vysílači sdělí, zda je schopen data přijímat a zpracovávat.
    - Pro SW handshaking se použijí symboly XON/XOFF v ASCII tabulce (ASCII kód XON je 17, XOFF je 19). Je-li však potřeba v toku dat znaky XON/XOF vyslat, je nutné vyslat speciální sekvenci znaků, což samozřejmě přenos dat obsahujících převážně tyto znaky značně zpomalí.

11.4 UART v PC

- řešen jako externí integrovaný obvod připojený k datové sběrnici CPU
- 8250 – nejstarší
- 16450 – v PC-AT, vyšší v_B
- 16550 – navíc 16b vyrovnávací FIFO pro vysílání i příjem



Obrázek 146 Externí řadič UART 16550

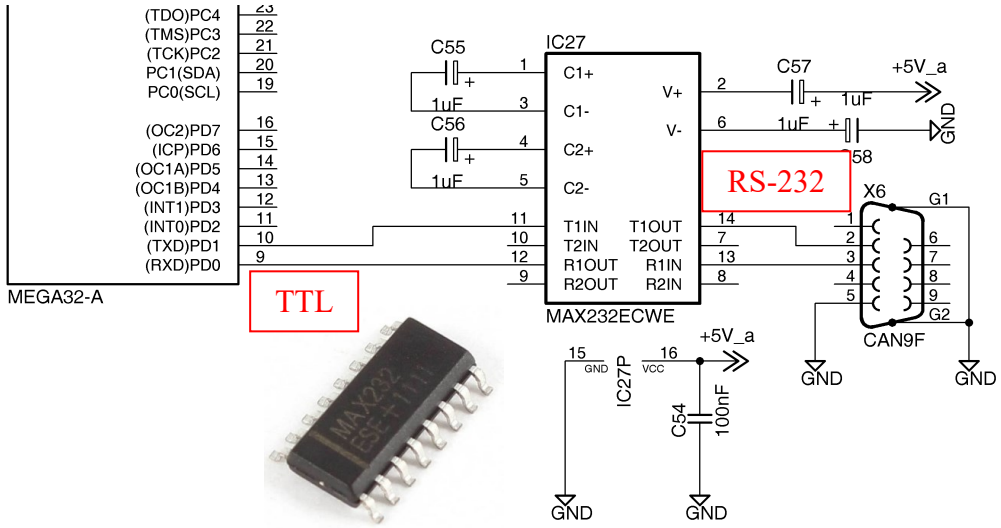
- Lze použít i s MCU, ale dnes již nemá význam (MCU mají integrovaný UART).

11.5 UART v MCU

- Většina MCU má integrovaný UART (není třeba používat externí integrovaný obvod).
- Větší MCU → více linek UART, např.:
  - ATmega32 - 1x UART
  - ATmega2560 – 4x UART
- Více UART znamená možnost komunikace s více zařízeními (z principu funkce UART nelze mít na jedné sběrnici více zařízení než 2).

11.6 Převodník UART / RS-232

- Pro konverzi úrovní UART (TTL) ↔ RS-232 je nutné doplnit externí převodník (**budič**) ve formě integrovaného obvodu.
- Často se používá obvod **MAX232** nebo **MAX3232** od firmy Maxim nebo podobný typ.

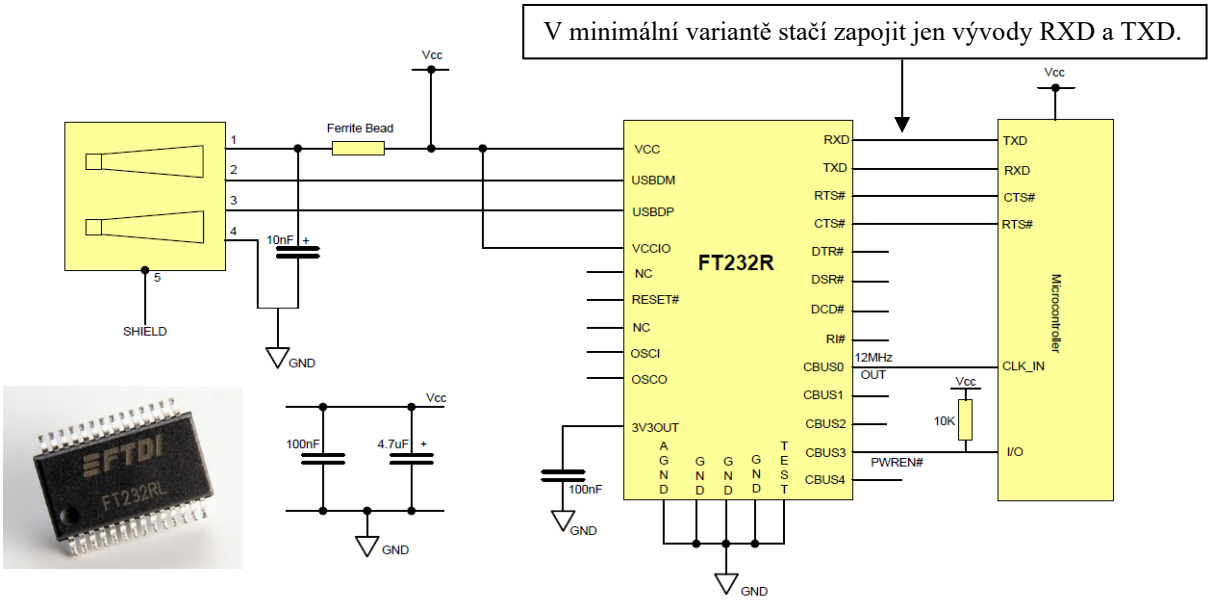


Obrázek 147 Konverze úrovní z TTL na RS-232

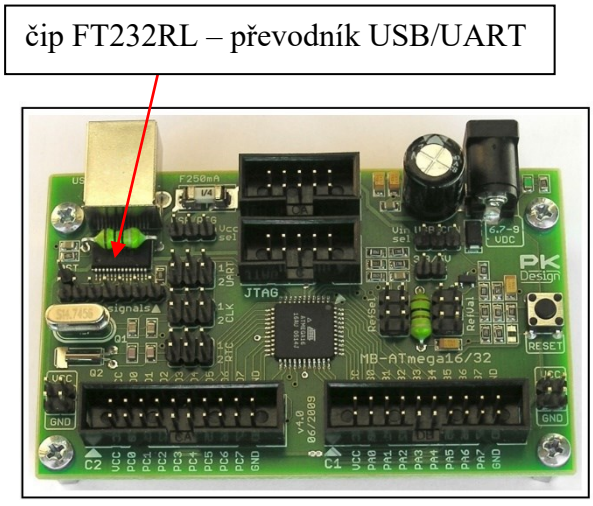
- Budiče MAXxxx (např. MAX232, MAX3232 apod.)
  - Různé typy podle:
    - Počtu kanálů (MAX232 4 kanály, 2x Tx, 2x Rx)
    - Max. komunikační rychlosti
    - Napájecího napětí 3 V, 5 V…
  - Mají integrovanou nábojovou pumpu pro vytvoření bipolárního napětí ±10V kvůli RS-232.

11.7 Převodník UART / USB

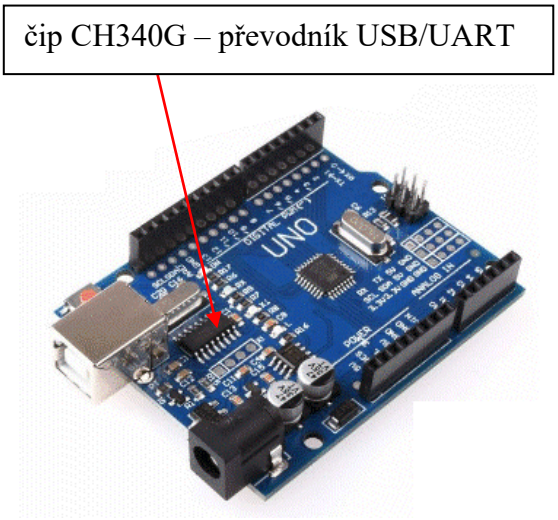
- Pro konverzi úrovní UART (TTL) ↔ USB je nutné doplnit externí převodník ve formě integrovaného obvodu.
- Často se používá obvod **FT232RL** od firmy **FTDI chip** nebo podobný typ.



Obrázek 148 Konverze úrovní z TTL na USB s převodníkem FT232R (zdroj: datasheet FT232R)



Obrázek 149 Základová deska ATmega32



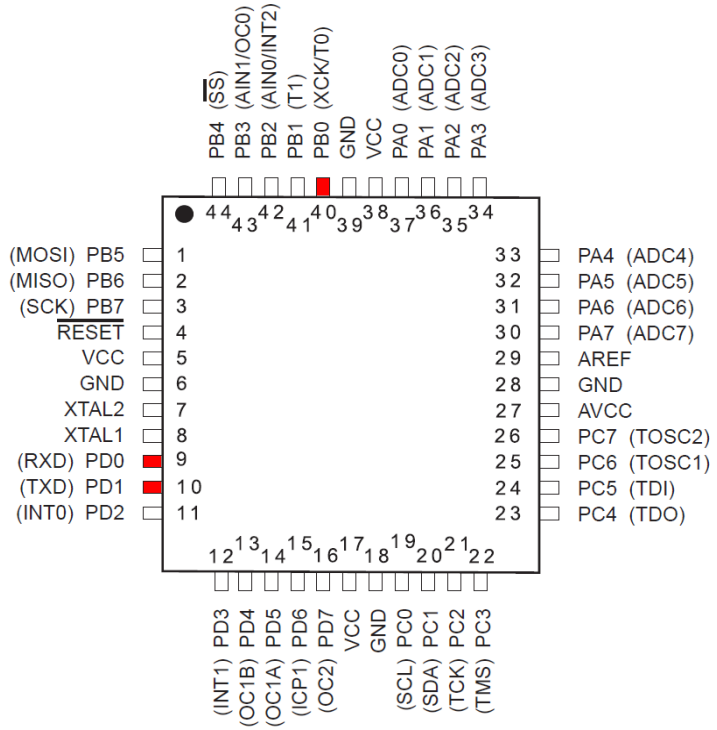
Obrázek 150 Arduino UNO

- **Výhody použití převodníků USB/UART**
  - Převodník obsahuje kompletní USB protokol, je to zároveň řadič i budič sběrnice USB.
  - MCU nemusí obsahovat řadič USB rozhraní (jehož obsluha je složitá), stačí, když má linku UART, pomocí které si vyměňuje data s převodníkem.
  - Převodník je pro MCU naprosto transparentní (jako kdyby tam nebyl).
    - Zařízení dříve komunikující po rozhraní RS-232 může pouhou výměnou HW komunikovat přes USB, není nutná žádná změna v programu na straně MCU, ani na druhé straně (např. PC).
  - Po připojení k PC přes USB se na PC objeví nový COM port, přes který probíhá komunikace se zařízením (na PC musí být nainstalovaný Virtual COM Port driver).
- **Různé způsoby zapojení převodníku**
  - Převodník i MCU jsou napájeny z USB rozhraní (Obrázek 148).
  - Převodník je napájen z USB, MCU je napájeno ze svého zdroje nezávislého na USB.
  - Převodník má vlastní generátor kmitočtu → může taktovat MCU, odpadá externí krystal pro MCU.
  - Lze realizovat různé převodníky z USB na jiný typ rozhraní:
    - USB / RS-232
    - USB / RS-422
    - USB / RS-485
    - USB / UART

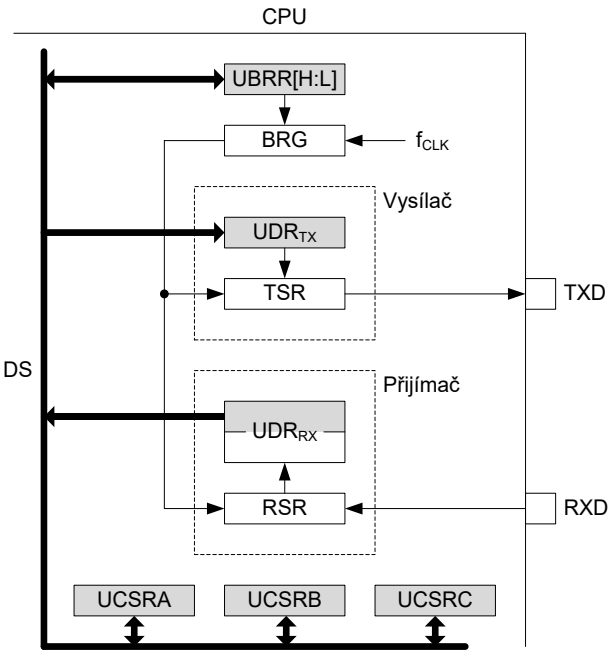
11.8 USART v AVR

- USART = Universal Synchronous and Asynchronous Receiver/Transmitter
- Je obsažen téměř ve všech typech AVR.
  - AVR s větším počtem pinů mají i více linek UART (až 4).
- Vlastnosti:
  - Plně duplexní sériový kanál.
  - Linky TXD a RXD, XCK.
  - Asynchronní nebo synchronní komunikace (dále se budeme zabývat pouze asynchronní komunikací).
    - U synchronní komunikace je třetí linka XCK (hodiny).
    - U asynchronní komunikace se linka XCK nepoužívá.
  - 5 až 9 datových bitů, 1 nebo 2 stop bity, parita žádná lichá nebo sudá.
  - Detekce falešného start bitu, chybného znaku a přetečení přijímacího bufferu.
  - Vlastní generátor přenosové rychlosti.
  - 3 samostatné vektory přerušení:
    - vysílání dokončeno
    - vysílací vyrovnávací registr prázdný
    - příjem dokončen

14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Register Empty
16	$01E	USART, TXC	USART, Tx Complete



Obrázek 151 Linka UART na AVR ATmega32



Obrázek 152 Zjednodušené blokové schéma linky UART na AVR

- **Baud Rate Generator** (BRG)
  - generuje signál pro posuvné registry dělením z fCLK
  - nastavení pomocí registru **UBRR** (16-bit, UBRRH: UBRRL)
- **UDR je datový registr pro vysílání i příjem**
  - Ve skutečnosti jsou to dvě fyzicky oddělená místa (na obrázku **UDRTX** a **UDRRX**).
  - Zápisem do **UDR** se zapíší data do vysílacího registru, čtením **UDR** se čtou data z přijímacího registru.
- **Vysílač**
  - data z registru **UDR** jsou zapsána do posuvného registru vysílače (**TSR** - Transmit Shift Register), ve kterém je sestaven UART rámec a bit po bitu odeslán na pin TXD
  - řízen signálem z BRG
  - dovoluje nepřetržité vysílání bez prodlev
- **Přijímač**
  - přijímaná data z pinu RXD jsou převedena v Reicever Shift Register (**RSR** – posuvný registr přijímače) do paralelní podoby a zapsána do registru **UDR** (bufferovaný)
  - umí detekovat chybu parity, rámce, přetečení dat v UDR
- **UCSRx** (Usart Control and Status Register x) – konfigurační a stavové registry
- Piny přidružené k sériovému portu jsou **TxD** (vysílaní), **RxD** (příjem) a **XCK** (hodiny – jen v případě synchronního přenosu)

11.8.1 Registry pro USART

- **UDR** – datový (společný pro vysílání i příjem)
- **UCSRA**, **UCSRB**, **UCSRC** – řídící a stavové
- **UBRRH**, **UBRRL** – nastavení generátoru přenosové rychlosti
- Viz datasheet.

11.8.2 Nastavení přenosové rychlosti

- Logika pro generování hodin sériového portu obsahuje čítač, který slouží pro generování taktovacího signálu a jehož obsah se každým taktem oscilátoru sníží o 1.
- Přednastavením tohoto čítače se určuje přenosová rychlost sériové linky. Registr **UBRR** (16-ti bitový) je předvolbou tohoto čítače.
- **Zápisem hodnoty do registru UBRR se nastaví přenosová rychlost**. Platí následující vztahy:

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

f_{OSC} System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

- Bitem **U2X** lze nastavit dvojnásobnou rychlost přenosu, čímž lze v některých situacích zmenšit chybu v generování přenosové rychlosti.
- Ne vždy (záleží na F_{OSC}) lze nastavit jakoukoli přenosovou rychlost neboť hodnota, vypočtená podle výše uvedených vztahů, nemusí být celé číslo.
  - Tím vzniká odchylka od správné hodnoty rychlosti.
  - Datasheet uvádí doporučené maximální chyby v nastavení přenosové rychlosti pro různé režimy komunikace.
  - Obecně se doporučuje, aby chyba nebyla vyšší než cca 1,5 % nominální přenosové rychlosti.
- V datasheetu jsou dále uvedeny hodnoty **UBRR** pro nejčastěji používané frekvence F_{OSC} a přenosové rychlosti včetně udání procentuálních chyb.

Table 70. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

1. UBRR = 0, Error = 0.0%

- **Registr UBRR** = UBRRH:UBRRL

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- **Přístup do registru UBRR**
  - Registr **UBRRH** sdílí stejnou adresu s registrem **UCSRC**.
  - **Zápis** do **UBRRH** nebo **UCSRC**:
    - Bitem **URSEL** v **UCSRC** (**UBRRH**) se volí přístup do registru **UBRR** nebo **UCSRC**:
      - bit **URSEL=1** → zápis do **UCSRC**
      - bit **URSEL=0** → zápis do **UBRRH**
  - **Čtení – je nutné dodržet následující postup:**
    - Prvním čtením se přečte **UBRRH**.
    - Následným druhým čtením se vyčte **UCSRA**.
    - Zřídkakdy je ovšem vůbec nutné tyto registry číst.

11.8.3 Inicializace sériového portu

- Nastavení přenosové rychlosti.
- Nastavení formátu přenosu (počet datových bitů, stop bitů, parita).
- Povolení vysílače a přijímače.
- **Příklad inicializace:**

```
UBRRH = 0;     // 38400 Bd, do MSB se musi zapisovat jako prvni
UBRRL = 25;

// TX on, Rx on, preruseni od RXC on
UCSRB = (1<<RXCIE) | (1<<RXEN) | (1<<TXEN);

// (URSEL musi byt 1 pri zapisu do UCSRC) 8b data, 1 stop bit,
// no parity, asynchronni prenos
UCSRC = (1<<URSEL) | (1<<UCSZ1) | (1<<UCSZ0);
```

11.8.4 Vysílání

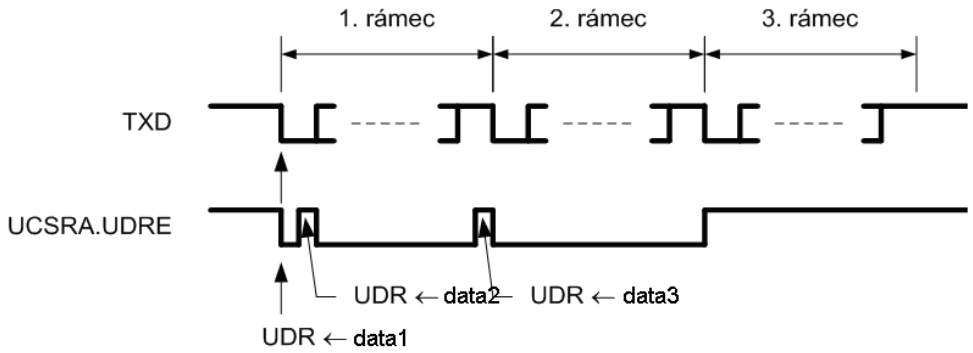
- Nejprve musíme vysílání povolit (bit **TXEN = 1** v registru **UCSRB**).
- Samotné vysílání zahájíme zápisem bajtu do datového registru UDR.
- Lze použít polling (čekáme na dokončení odeslání) nebo přerušení „vysílací buffer prázdný“ (USART, data register empty) nebo „vysílání dokončeno“ (USART, Tx complete).
- **Polling**
  - Komunikace je blokující – aplikace musí čekat než se data odešlou/přijmou.
    - Což může zejména u malých přenosových rychlostí (9600 Bd) a velkého počtu dat trvat poměrně dlouho a nevhodně blokovat vykonávání aplikace.
  - Kontroluje se příznak prázdného vysílacího bufferu **UDRE** („UDR Empty“) nebo příznak dokončeného vysílání **TXC** („Tx complete“).

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

  - **Příznak UDRE** – nastaví se tehdy, když se vyprázdní, tj. v okamžiku kdy se jeho obsah (1 bajt) přesune do posuvného vysílacího registru TSR (viz blokové schéma USARTu v AVR), data ještě nejsou fyzicky odeslána.
  - **Příznak TXC** – nastaví se až tehdy, pokud už došlo k fyzickému odesální celého bajtu (tj. vyprázdnění vysílacího posuvného registru TSR).
  - Následující funkce počká na vyprázdnění vysílacího bufferu a pak do něj zapíše novou hodnotu.
    - Výhoda použití příznaku **UDRE** – při odesílání většího množství dat není datový tok přerušován.

```
void USART_Transmit(unsigned char data)
{
        // Pockam na prazdny buffer – az se nastavi priznak UDRE
        while ( !( UCSRA & (1<<UDRE)) ) ;
        // Ulozi do bufferu nova data a zacne se odesilat
        UDR = data;
}
```

- Časový průběh vysílání:



- **Při devíti-bitové komunikaci** se musí nejprve zapsat 9. bit do bitu **TXB8** v registru **UCSRB** (ten právě slouží pro uložení 9. bitu při 9-ti bitovém přenosu) a pak datový registr **UDR** naplnit zbylými 8-mi bity.

- **Přerušení** (preferovaný způsob)
  - Příznaky **TXC** a **UDRE** mohou být využity pro generování přerušení.
  - Přerušení se povoluje pomocí bitů **TXCIE** respektive **UDRIE** v řídícím registru **UCSRB**.

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - Výhodou přerušení je to, že komunikace není blokující – aplikace nemusí čekat na odesílání/příjem dat.
  - Výhoda použití příznaku **UDRE** oproti **TXC** – při odesílání většího množství dat není datový tok přerušován.
  - **Příznak UDRE** (Data Register Empty) je **nastaven** vždy, když datový registr **UDR neobsahuje data**. **Nulový** je tehdy, když **UDR obsahuje data**, která ještě nebyla přesunuta do vysílacího posuvného registru.
  - **Příznak UDRE se nemaže automaticky po vstupu do přerušovací rutiny**, ale maže se automaticky až po zápisu nové hodnoty do UDR. Nedojde-li k tomu, pak je přerušení opět vyvoláno. Takže je nutno buď zapsat nová data do registru UDR nebo zakázat přerušení od UDRE.

- V praxi to může vypadat tak, že se toto přerušení použije např. pro postupné odvysílání nějakého pole hodnot, přičemž po odvysílání poslední hodnoty se v obsluze přerušení toto přerušení programově zakáže.
- Další vysílání se spustí pouhým povolením tohoto přerušení – nastavením bitu **UDRIE** do 1.
- **Viz ukázka programu (pro kompilátor AVR-GCC):**

```
volatile unsigned char tx_buf[64]; // vysilací buffer

// odeslání pole hodnot v přerušovací rutině
ISR(USART_UDRE_vect)
{
      UDR = tx_buf[tx_cnt];
      tx_cnt++;
      if (tx_cnt == tx_bytes) // vše odesláno?
      {
            tx_cnt = 0; tx_bytes = 0;
            UCSRB &= ~(1<<UDRIE); // zákaz přerušení
      }
}

.
.
.
.
// někde jinde v programu – zahájení odesílání
tx_bytes = sizeof(tx_buf);    // kolik hodnot se ma odeslat
UCSRB |= (1<<UDRIE);    // povoleni preruseni – zahajeni vysilani
```

Kde: `tx_buf` – pole bajtů, které se mají odeslat (vysílací buffer), v příkladu není ukázáno, kde se buffer plní daty (to zde není úplně podstatné)

`tx_cnt` – počítadlo odeslaných bajtů

`tx_bytes` – kolik bajtů celkem se bude odesílat (ve výše uvedeném příkladu se odešle vždy celý buffer, ale zpravidla tomu tak není vždy, počet odesílaných bajtů závisí na tom, jak je vysílací buffer zaplněn, a ten nemusí být zaplněn vždy celý)

- **Příznak TXC** (Transmit Complete) je nastaven, když data opustí vysílací posuvný registr **TSR** a pokud i datový registr **UDR** je prázdný. **Příznak se automaticky nuluje po vstupu do přerušovací rutiny.**
  - V praxi opět možno použít pro odvysílání několika hodnot po sobě. Vysílání je spuštěno zápisem první hodnoty do UDR. Následující hodnoty jsou již odesílány v přerušení.
  - **Viz ukázka programu (pro kompilátor AVR-GCC):**

```
// odeslání pole hodnot v přerušovací rutině
ISR(USART_TXC_vect)
{
      tx_cnt++;   // počítadlo odeslaných hodnot
      if (tx_cnt < tx_bytes) UDR = tx_buf[tx_cnt];
      else {tx_cnt = 0; tx_bytes = 0;} // vše odesláno
}

.
.
.
// někde jinde v programu – zahájení odesílání
tx_bytes = sizeof(tx_buf);    // kolik hodnot se má odeslat
UDR = tx_buf[0]; // nutno zapsat prvni hodnotu do UDR
```

11.8.5 Příjem

- Musí být nejprve povolen (bit **RXEN = 1** v registru **UCSRB**).

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

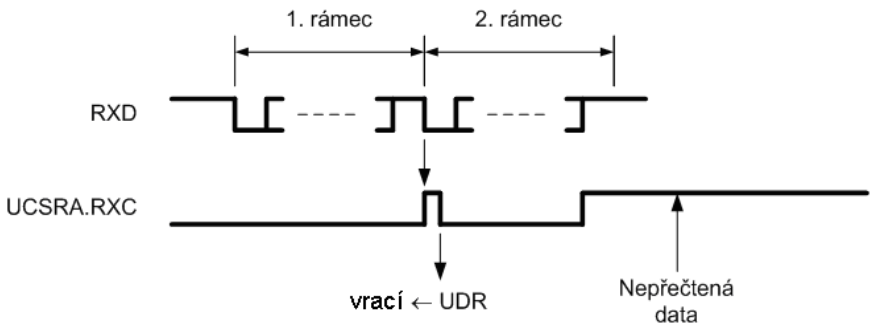
- Jakmile se celý rámec přijme do přijímacího posuvného registru (tozn. že je detekován STOP bit), dojde k přesunutí obsahu do registru **UDR**. Pak se nastaví příznak kompletního příjmu **RXC** v registru UCSRA.

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- Příznak **RXC** se vynuluje přečtením obsahu **UDR**.
- Opět lze využít polling (čekáme na příjem bajtu) nebo přerušení.
- **Polling**
  - Ručně stále dokola kontrolujeme příznak **RXC** v registru **UCSRA**.
  - Následující funkce vrací přijmutý bajt:

```
unsigned char USART_Receive(void)
{
      // čekání na dokončení příjmu
      while ( !(UCSRA & (1<<RXC)) ) ;
      // přečtení prijmutého bajtu a návrat
      return UDR;
}
```

  - Funkce je blokující – neustále čeká na příjem bajtu
    - Pokud nic nepřichází, tak program stojí!
  - Časový průběh:



- **Při devíti-bitové komunikaci** je nutné nejprve přečíst 9. bit z bitu **RXB8** v registru **UCSRB** (v něm je uložen přijmutý 9. bit) a pak přečíst zbylých 8 bitů z datového registru **UDR**.

- **Přerušení** (preferovaný způsob)
  - Příznak **RXC** je využit k vygenerování přerušení.
  - Příznak **RXC** je nastaven pokuď je v přijímacím bufferu nepřečtený byte.
  - Příznak se vynuluje přečtením obsahu **UDR**.
    - Nepřečtou-li se tato nová data, bude přerušení generováno neustále.
  - Neblokuje aplikaci - výhoda.

```
// obsluha přerušení při příjmu
ISR(USART_RXC_vect)
{
	unsigned char status, data;

	status = UCSRA;   // stav příjmu (pro kontrolu chyb)
	data = UDR;       // vyčtení dat z UDR

	// je-li příjem bez chyb, pokračuj
	if ((status & ((1<<FE)|(1<<DOR)|(1<<PE))) == 0)
	{
		// nějake zpracovaní přijmutého bajtu, obvykle jeho uložení
		do přijímací fronty a zpracování v jiné části programu,
		aby byla obsluha přerušení co nejkratší
	}
}
```

11.8.6 Příznaky chyb při příjmu

- Lze využít kontrolních mechanismů a zjišťovat zda nenastala při příjmu nějaká chyba:
  - **Framing Error** (chyba rámce – špatný stop bit)
  - **Data OverRun** (přetečení přijímacího bufferu) – pokud včas nevyčítáme UDR
  - **Parity Error** (chybná parita v přijatých datech, pokud je použita)
- Jedná se o příznaky (bity):
  - **FE**
  - **DOR**
  - **PE**
  - které jsou umístěny v registru **UCSRA**

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

  - Tyto příznaky negenerují žádná přerušení !

- Praxe je taková, že před vlastním čtením přijaté hodnoty kontrolujeme, zda nenastala nějaká z těchto chyb a dle toho se nějakým způsobem reaguje.
- Vždy je nutno dodržet toto pořadí – nejdřív číst příznaky chyb a pak teprve vlastní data, ne naopak. Viz úryvek kódu výše (nejdříve se čte registr **UCSRA** a pak teprve obsah registru **UDR**).
- **Modifikovaná funkce `USART_Receive` z příkladu, ve kterém byl využit polling:**
  - Přijatý bajt se předává odkazem jako parametr.
  - Funkce vrací informaci o chybě.

```
// vraci 0 když byl příjem OK, jinak vrací 1
unsigned char USART_Receive(unsigned char *pData)
{
      while ( !(UCSRA & (1<<RXC)) ) ; // čekání na dokončení příjmu
      status = UCSRA;   // přečtení příznaku

      // kontrola bezchybného příjmu
      if ( status & ((1<<FE)|(1<<DOR)|(1<<PE)) )
      return 1;         // byla chyba

      *pData = UDR;     // příjem byl OK
      return 0;
}
```

11.8.7 Využití front pro vysílání a příjem

- Ve výše uvedených ukázkách kódu pro vysílání a příjem dat jsme nepoužívali fronty, maximálně jen jednoduché pole dat.
- Použití front se v případě nárazového odesílání/příjmu většího množství dat přímo nabízí a je výhodné.
- Dvě fronty:
  - Fronta pro vysílání (Tx queue).
  - Fronta pro příjem (Rx queue).
- Princip použití vysílací fronty:
  - Aplikace plní frontu daty, obecně kdykoliv a z různých míst v programu.
  - Obsah fronty se postupně odešle přes UART „na pozadí“ s použitím přerušení.
  - Aplikace může frontu plnit i během právě probíhajícího odesílání dat, aniž by to mělo na odesílání nějaký neblahý vliv.
  - Velikost fronty je třeba nastavit tak, aby nemohlo dojít k jejímu přeplnění (data z fronty se musejí stíhat odesílat rychleji, než se fronta plní).
- Princip použití přijímací fronty:
  - V obsluze přerušení při příjmu bajtu po UART se každý příjmutý byte uloží do fronty.

- Aplikace jednou za čas zkontroluje, zda jsou ve frontě nová data, pokud ano, tak je z fronty vyčte a zpracuje → zpracování dat je mimo obsluhu přerušení při příjmu, takže může být i časově náročnější a neztratíme nově přicházející bajty.
- Fronta se může plnit novými daty kdykoliv i v průběhu vyčítání dat z fronty.
- Velikost přijímací fronty je třeba stanovit podle množství přijímaných dat, komunikační rychlosti a frekvence vyčítání dat z fronty.

- Vytvoření programu využívajícího fronty v komunikaci je již trochu náročnější a vede de facto na vytvoření vlastní knihovny funkcí pro příjem a odesílání dat přes UART, znovupoužitelné v jakémkoliv dalším projektu.
  - Tomuto se můžeme vyhnout pokud:
    - Máme podporu na straně vývojového prostředí, které nám vygeneruje kód pro komunikaci nebo obsahuje knihovnu, kterou můžeme pro komunikaci využít.

      nebo

    - Použijeme knihovnu třetí strany (tzn. že někdo jiný už to naprogramoval za nás a mi jeho kód pouze použijeme).
  - V kapitole „**USART a kompilátor AVR-GCC**“ je ukázka použití knihovny třetí strany, která pro komunikaci po UART používá fronty.

11.9 USART a kompilátor CodeVision

- Lze si napsat své vlastní funkce pro TX a RX – viz výše.
- Nebo lze používat standardní vstupně výstupní funkce (`printf()`, `scanf()`, `puts()`, `gets()`, `putchar()`, `getchar()`), jejichž **vstup či výstup je přesměrován na sériový port USART**.
    - Předtím je ovšem nutné sériový port inicializovat a také vložit příslušný hlavičkový soubor (`#include <stdio.h>`).
- To, že jsou vstupy či výstupy přesměrovány na sériový port mají na svědomí funkce `putchar()` a `getchar()`. Tyto funkce jsou totiž volány z funkcí `puts()`, `putsf()`, `printf()`, `gets()`, `scanf()`.
    - Je klidně možné tyto dvě základní funkce (`putchar` a `getchar`) modifikovat a upravit si jejich chování dle libosti, tak lze třeba zajistit výstup či vstup někam jinam než na sériový port.
    - Podrobnosti viz help v CodeVision.
- Originální funkce **`putchar()`** a **`getchar()`** používají při přístupu na sériový port **polling** (čekají dokud se zpráva neodešle či nepřijme), jsou blokující. Pokud je třeba data odesílat a přijímat na pozadí (neblokovaně), je nutné použít přerušení. To znamená, napsat si vlastní oblsuhu přerušení nebo v CodeWizardovi zapnout podporu přerušení při příjmu resp. při odesílání po USART.
- **Příklad:**

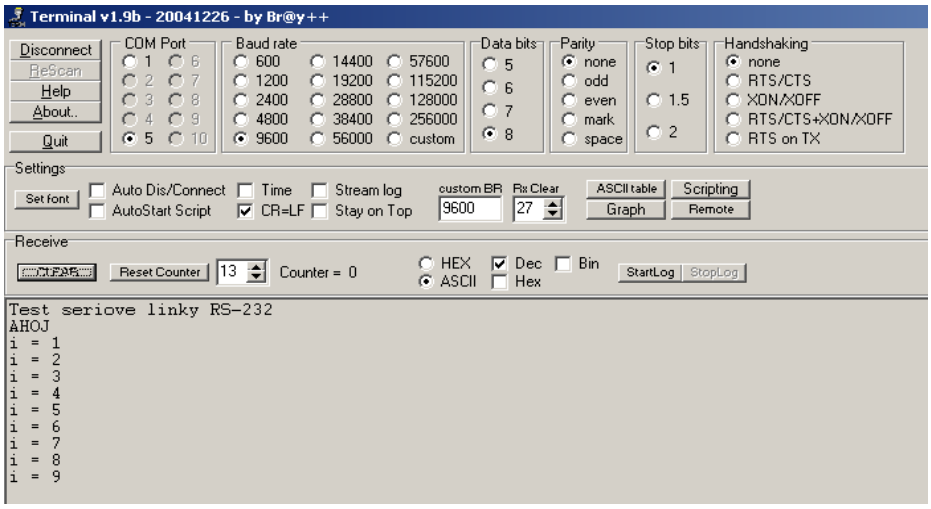
```
#include <mega32.h>
#include <delay.h>
#include <stdio.h>

void main(void)
{
      unsigned int i = 0;

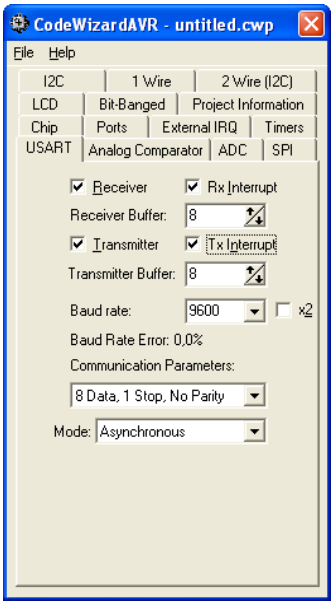
      // USART initialization
      // Communication Parameters: 8 Data, 1 Stop, No Parity
      // USART Receiver: On
      // USART Transmitter: On
      // USART Mode: Asynchronous
      // USART Baud Rate: 9600
      UCSRA=0x00;
      UCSRB=0x18;
      UCSRC=0x86;
      UBRRH=0x00;
      UBRRL=0x5F;

      // odesle uvodni text
      printf("Test seriove linky RS-232\n");
      // odesle AHOJ
      putchar('A');
      putchar('H');
      putchar('O');
      putchar('J');
      putchar('\n');
```

```
        // periodicky odesila hodnotu promenne i
        while (1)
        {
                i++;
                printf("i = %d\n", i);       // odesle hodnotu i
                delay_ms(1000);              // prodleva 1000 ms
        }
}
```



- Sériový port lze jednoduše nastavit pomocí CodeWizarda. Lze povolit i přerušení od TXC a RXC.
    - Lze určit velikost softwarového vysílacího a přijímacího bufferu. (implementovány jako fronty).
    - Následně dojde k vygenerování funkcí pro obsluhu přerušení včetně kompletního kódu pro příjem a odesílání hodnot za využití front.



Obrázek 153 CodeWizard (průvodce pro nastavení periferií v AVR) v CodeVision IDE

11.10 USART a kompilátor AVR-GCC

- Knihovna AVR-libcv toolchainu bohužel neposkytuje žádné funkce pro zjednodušení obsluhy linky USART na AVR.
- Musíme si napsat svoje vlastní funkce pro komfortní obsluhu linky USART (ukázky kódu byly uvedeny v předchozích kapitolách) nebo použít knihovnu třetí strany (jak bude ukázáno dále).

11.10.1 Knihovna UART

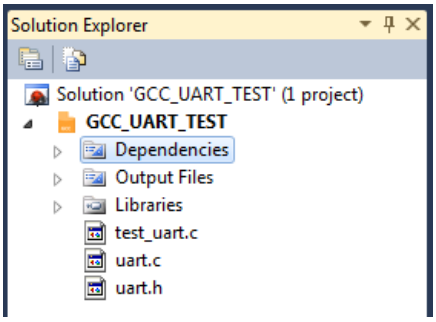
- Autor: Peter Fleury
  http://homepage.hispeed.ch/peterfleury/
  https://github.com/damadmai/pfleury (neoficiální)
- Vlastnosti knihovny:
  - Vyžaduje kompilátor AVG-GCC.
  - Pro příjem a vysílání používá přerušení a fronty implementované pomocí polí jako kruhové buffery.
    - V obsluze přerušení při příjmu je přijatý byte zapsán do přijímací fronty.
    - V obsluze přerušení při vysílání je postupně každý byte z vysílací fronty vyčten a odeslán.
  - Aplikace přistupuje na fronty pomocí funkcí (čtení, zápis).
  - Formát rámce 8N1 (jiný formát nelze nastavit pomocí funkce, jedině editací zdrojového kódu knihovny).
  - Nastavitelná rychlost komunikace.
  - Nastavitelná velikost přijímací a vysílací fronty.
  - U AVR s více UART umí obsluhovat UART0 a UART1.
  - Funkce pro:
    - Inicializaci UART.
    - Odeslání 1 bajtu.
    - Odeslání řetězce.
    - Čtení 1 bajtu.
- Knihovna se skládá z těchto souborů:
  - `uart.h` – hlavičkový soubor s rozhraním knihovny
  - `uart.c` – implementace

11.10.2 Funkce knihovny UART

- `void uart_init(unsigned int baudrate)`
  - Inicializace UART rozhraní, tato funkce musí zavolána jako první.
  - Jako parametr se do funkce předává číslo vypočtené pomocí makra `UART_BAUD_SELECT(baudRate,xtalCpu)`.
    - `baudRate` – požadovaná rychlost komunikace [Bd]
    - `xtalCpu` – frekvence CPU [Hz]
  - Makro je definováno v hlavičkovém souboru `uart.h`.
- `unsigned int uart_getc(void)`
  - Vyčte jeden byte z přijímaci fronty.
  - Dolní byte návratové hodnoty = přijmutý byte.
  - Horní byte návratové hodnoty = stav komunikace.
    - 0 – byte byl přijat v pořádku
    - `UART_NO_DATA` – pokud fronta neobsahuje žádná data
    - `UART_BUFFER_OVERFLOW` – pokud fronta přetekla
    - `UART_OVERRUN_ERROR` – pokud nebyl byte včas vyčten z registru UDR a byl přepsán nově příchozím
    - `UART_FRAME_ERROR` – chyba rámce UART
- `void uart_putc(unsigned char data)`
  - Zapíše byte do vysílací fronty a spustí vysílání.
- `void uart_puts(const char *s )`
  - Zapíše řetězec do vysílací fronty a spustí vysílání.
  - Pokud fronty nepojme celý řetězec, funkce čeká na uvolnění místa ve frontě.
- `void uart_puts_p(const char *s )`
  - Zapíše řetězec umístěný v paměti programu do vysílací fronty a spustí vysílání.
  - Pokud fronty nepojme celý řetězec, funkce čeká na uvolnění místa ve frontě.
- Knihovna obsahuje ty samé funkce pro obsluhu druhého řadiče UART (UART1), pokud jím je AVR vybaveno, tyto funkce mají názvy v tomto stylu:
  - `void uart1_init(unsigned int baudrate)`

11.10.3 Použití knihovny UART v programu

- Příklad je určen pro překladač AVR-GCC a MCU ATmega32.
- Příklad ukazuje:
  - Inicializaci USART.
  - Odeslání bajtu.
  - Odeslání řetězce.
  - Příjem bajtu.
  - Reakci na chybu při příjmu.
- Do projektu je třeba vložit soubory:
  - `uart.h` – hlavičkový soubor s rozhraním knihovny
  - `uart.c` – implementace



Obrázek 154 Soubory knihovny UART (Peter Fleury) v projektu Atmel Studia 6.1

- Dále je třeba:
  - Zadefinovat makro `F_CPU` s frekvencí MCU v Hz
    - Nejlépe na úrovni projektu (v Atmel Studiu v menu Project – Properties – záložka Toolchain – AVR/GNU C Compiler – Symbols)
  - Upravit velikosti Tx, Rx front, pokud výchozí nastavení (32 bajtů) nepostačuje.
    - Provede se úpravou hodnot maker v souboru `uart.h`.

```
/** @brief  Size of the circular receive buffer, must be power of 2
 *
 *  You may need to adapt this constant to your target and your application by adding
 *  CDEFS += -DUART_RX_BUFFER_SIZE=nn to your Makefile.
 */
#ifndef UART_RX_BUFFER_SIZE
#define UART_RX_BUFFER_SIZE 32
#endif

/** @brief  Size of the circular transmit buffer, must be power of 2
 *
 *  You may need to adapt this constant to your target and your application by adding
 *  CDEFS += -DUART_TX_BUFFER_SIZE=nn to your Makefile.
 */
#ifndef UART_TX_BUFFER_SIZE
#define UART_TX_BUFFER_SIZE 32
#endif
```

- **Kód (pro překladač AVR-GCC):**

```
/*************************************************************************
Title:    Example program for the Interrupt controlled UART library
Author:   Peter Fleury <pfleury@gmx.ch>   http://tinyurl.com/peterfleury
File:     $Id: test_uart.c,v 1.7 2015/01/31 17:46:31 peter Exp $
Software: AVR-GCC 4.x
Hardware: AVR with built-in UART/USART

DESCRIPTION:
          This example shows how to use the UART library uart.c

*************************************************************************/
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>

#include "uart.h"


/* define CPU frequency in Hz in Makefile */
#ifndef F_CPU
#error "F_CPU undefined, please define CPU frequency in Hz in Makefile"
#endif

/* Define UART buad rate here */
#define UART_BAUD_RATE      9600


int main(void)
{
    unsigned int c;   // received character, must be 16-bit integer!
    char buffer[7];   // transmitted string
    int  num=134;     // number to be converted to string


    /*
     *  Initialize UART library, pass baudrate and AVR cpu clock
     *  with the macro
     *  UART_BAUD_SELECT() (normal speed mode )
     *  or
     *  UART_BAUD_SELECT_DOUBLE_SPEED() ( double speed mode)
     */
    uart_init( UART_BAUD_SELECT(UART_BAUD_RATE,F_CPU) );

    /*
     * now enable interrupt, since UART library is interrupt controlled
     */
    sei();

    /*
     *  Transmit string to UART
     *  The string is buffered by the uart library in a circular buffer
     *  and one character at a time is transmitted to the UART using interrupts.
     *  uart_puts() blocks if it can not write the whole string to the circular
     *  buffer
     */
    uart_puts("String stored in SRAM\n");

    /*
     * Transmit string from program memory to UART
     */
    uart_puts_P("String stored in FLASH\n");
```

```
    /*
     * Use standard avr-libc functions to convert numbers into string
     * before transmitting via UART
     */
    itoa( num, buffer, 10);   // convert interger into string (decimal format)
    uart_puts(buffer);        // and transmit string to UART


    /*
     * Transmit single character to UART
     */
    uart_putc('\r');

    for(;;)
    {
        /*
         * Get received character from ringbuffer
         * uart_getc() returns in the lower byte the received character and
         * in the higher byte (bitmask) the last receive error
         * UART_NO_DATA is returned when no data is available.
         *
         */
        c = uart_getc();
        if ( c & UART_NO_DATA )
        {
            /*
             * no data available from UART
             */
        }
        else
        {
            /*
             * new data available from UART
             * check for Frame or Overrun error
             */
            if ( c & UART_FRAME_ERROR )
            {
                /* Framing Error detected, i.e no stop bit detected */
                uart_puts_P("UART Frame Error: ");
            }
            if ( c & UART_OVERRUN_ERROR )
            {
                /*
                 * Overrun, a character already present in the UART UDR register was
                 * not read by the interrupt handler before the next character arrived,
                 * one or more received characters have been dropped
                 */
                uart_puts_P("UART Overrun Error: ");
            }
            if ( c & UART_BUFFER_OVERFLOW )
            {
                /*
                 * We are not reading the receive buffer fast enough,
                 * one or more received character have been dropped
                 */
                uart_puts_P("Buffer overflow error: ");
            }

            /*
             * send received character back
             */
            uart_putc( (unsigned char)c );
        }
    }

}
```

```
/*
** high byte error return code of uart_getc()
*/
/* Framing Error by UART       */
#define UART_FRAME_ERROR      0x1000
/* Overrun condition by UART   */
#define UART_OVERRUN_ERROR    0x0800
/* Parity Error by UART        */
#define UART_PARITY_ERROR     0x0400
/* receive ringbuffer overflow */
#define UART_BUFFER_OVERFLOW  0x0200
/* no receive data available   */
#define UART_NO_DATA          0x0100
```

Nekonečná smyčka.

Periodická kontrola, zda jsou nějaká data k dispozici.

Reakce na chyby při příjmu.

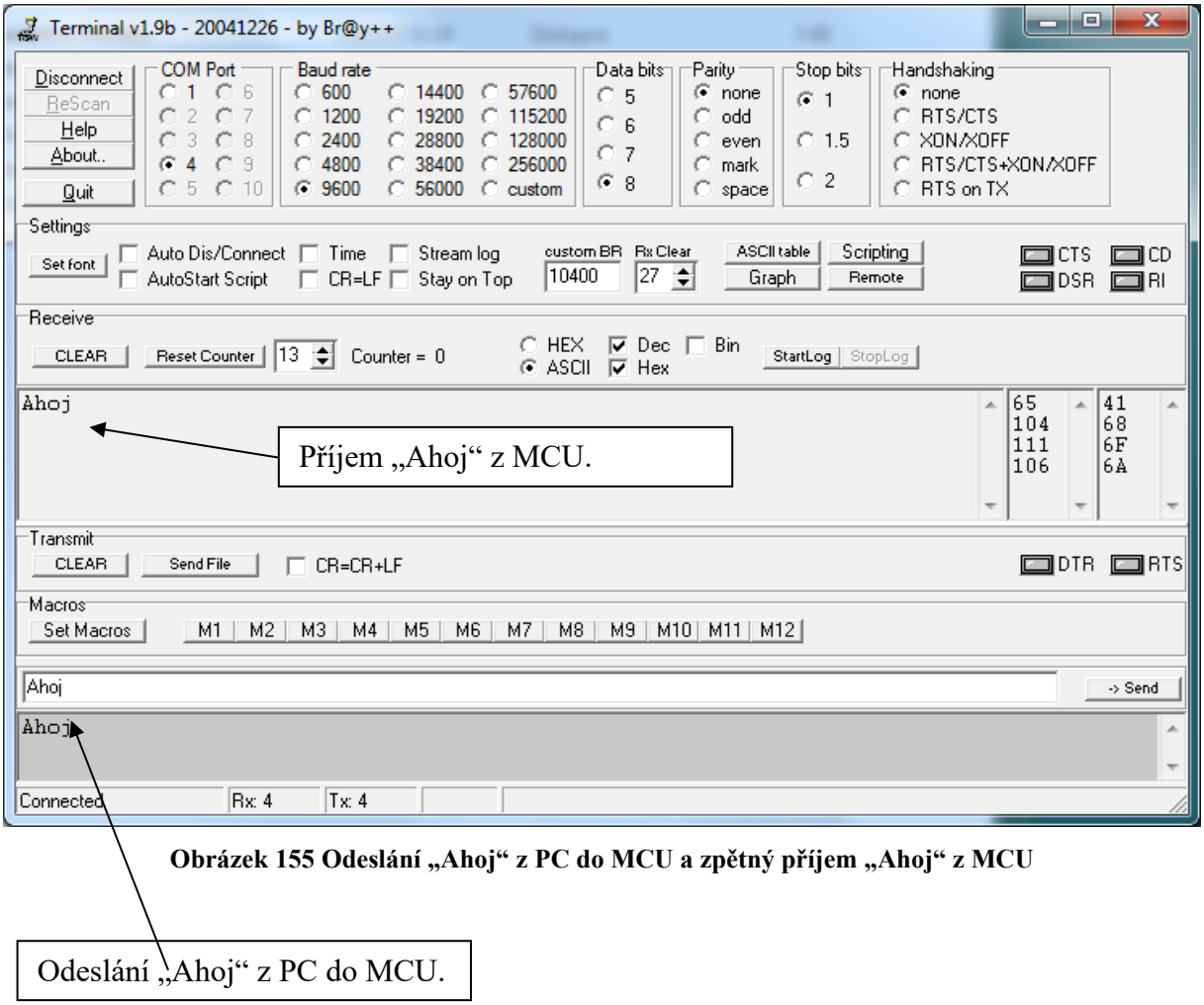
Kontrola chyb není povinná, je ale vhodná.

Zpracování přijatého bajtu.
Každý přijatý bajt odešle zpět po UART.

`(unsigned char)c` zajistí vyzvednutí jen užitečného dolního bajtu z proměnné `c`. Horní byte proměnné `c` obsahuje jen příznaky chyb (nenese užitečná data).

Ověření funkce programu

Pokud spojíme MCU s PC pomocí převodníku UART/RS-232 nebo UART/USB a na PC otevřeme terminál, tak můžeme do MCU posílat znaky a MCU nám bude odpovídat (to co jsme do něj poslali, to nám pošle zpátky).

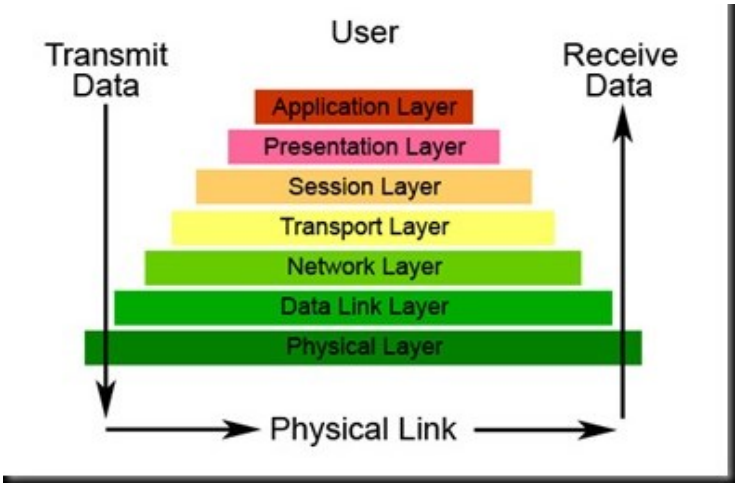


Obrázek 155 Odeslání „Ahoj“ z PC do MCU a zpětný příjem „Ahoj“ z MCU

12 Komunikační protokoly

12.1 ISO/OSI komunikační model a sběrnice

- Abstraktní popis hladinově orientovaného komunikačního kanálu.
- Standardní definice modelu.

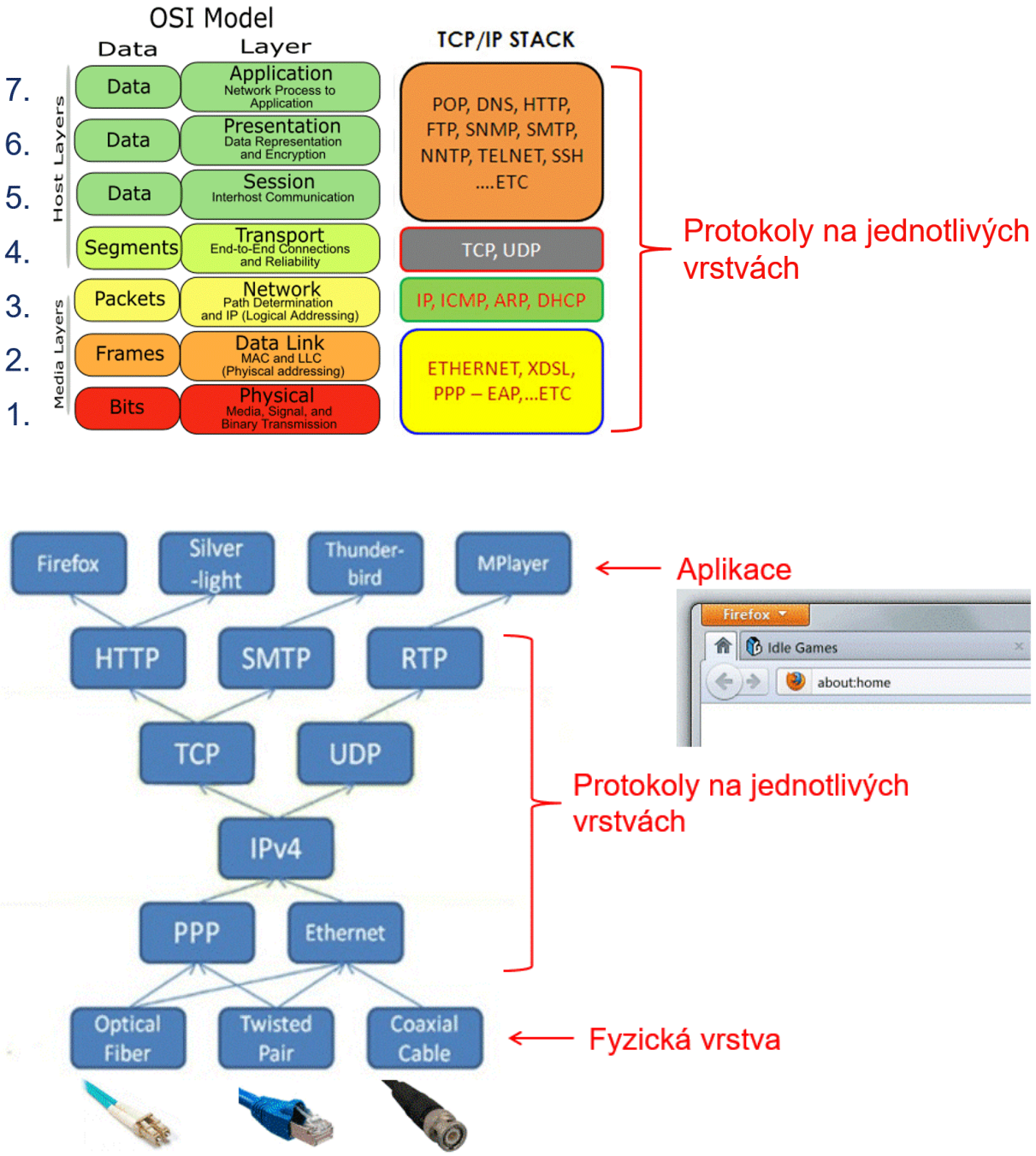


Obrázek 156 Vrstvový ISO/OSI komunikační model

- **Fyzická vrstva** – elektrické a mechanické specifikace zařízení (napětí = reprezentace log úrovní, impedance, kabely, konektory, ...)
  - RS232, RS485, proudová smyčka, I2C („wired and“, TTL úrovně) ...
- **Linková vrstva** – kódování dat, tvorba slov, rámců, časování
  - UART (start bit, časování ...), I2C (Start, Sr, Stop, ACK bity)
- **Síťová vrstva** – jak dostat data od jednoho účastníka ke druhému
  - topologie sběrnice, přístup k médiu (arbitrace, ...)
  - Master – Slave, token ring, Aloha
  - I2C (adresování, arbitrace)
- **Aplikační vrstva** – přenášená data (obsah), přístup aplikace k síti
- Ne všechny vrstvy musejí být v komunikaci použity.

Příklad: TCP/IP stack

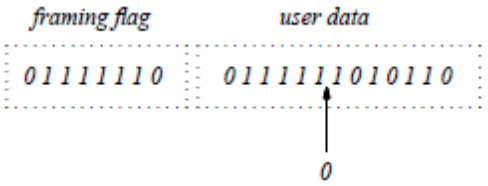
Rodina protokolů TCP/IP (anglicky Transmission Control Protocol/Internet Protocol – „primární přenosový protokol/protokol síťové vrstvy“) obsahuje sadu protokolů pro komunikaci v počítačové síti a je hlavním protokolem celosvětové sítě Internet. [Wikipedie]



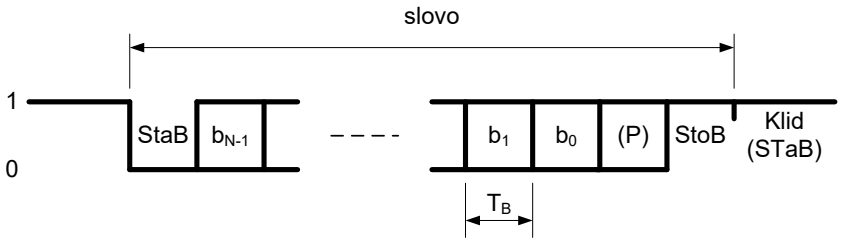
Obrázek 157 Internetové protokoly v ISO/OSI komunikačním modelu

12.2 Úvod do problematiky komunikačních protokolů

- Komunikační protokol = linková vrstva
- **rámec** = základní komunikační jednotka pro přenos jedné nebo více zpráv
- obsah rámce: (začátek) – data – (konec)
  - **začátek (konec)**: synchronizace přijímače (rámcová synchronizace) – přijímač pozná, kdy rámec začíná a kdy končí
  - **data**: obsah zprávy
- **Bitově orientované komunikační protokoly**
  - příklad:



    prokládání bitů (bit stuffing) – odlišení rámcové synchronizace od dat
  - **framing flag** – označuje začátek rámce, dohodnutá kombinace 0 a 1
  - **user data** – užitečná data, nesmí obsahovat kombinaci 0 a 1, která by se dala zaměnit za „framing flag“ → nutné použít prokládání znaků v datech
  - výhoda rámcové synchronizace:
    - při rozsynchronizování vysílače a přijímače (např. vlivem rušení, chvilkovou poruchou apod.) přijímač pozná v přijímaných datech začátek rámce a zasynchronizuje se
- **Znakově orientované komunikační protokoly**
  - **rámec**
    - základní komunikační jednotka
    - dohodnutá sekvence slov
    - např.: `začátek - data - data - data - konec`
  - **slovo** - posloupnost bitů, např. ve formátu UART



  - **dle povahy** (formátu přenášených dat)
    - textově orientované – přenáší se ASCII kódy (123 = 0x31 0x32 0x33)
    - binárně orientované – přenáší se datový typ (123 = 123)

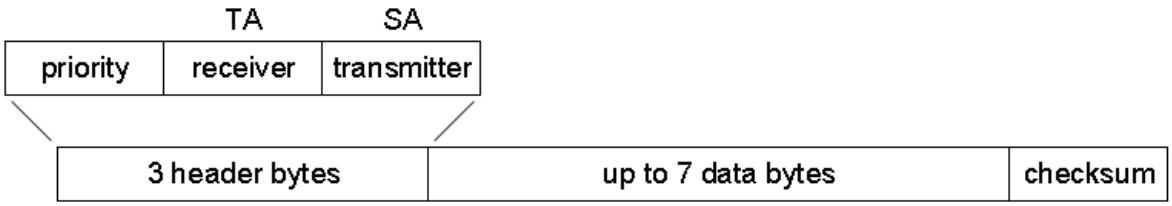
- **dle určení délky rámce**
  - s udanou délkou zprávy: `začátek – délka dat – data`
  - s neudanou délkou zprávy: `začátek – data – konec`
  - U textově orientované komunikace se jako konec zprávy často používá znak odřádkování Carriage Return <CR> (ASCII kód 13) nebo Line Feed <LF> (ASCII kód 10), případně jejich kombinace <CR><LF> (dva znaky). Při spojení s PC to ulehčuje odeslání zprávy z PC (znaky zapsané do okna terminálu stačí potvrdit klávesou Enter a dojde k jejich odeslání a automatickému doplnění znaku odřádkování).

Rámec

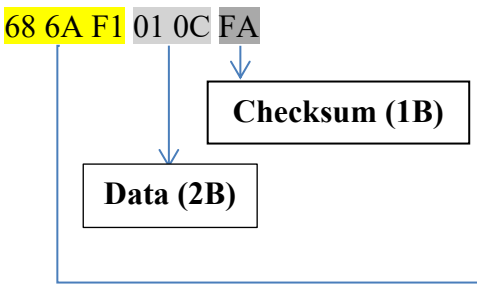
- **Např.:** začátek – hlavička – data – zabezpečení
- **hlavička** (header) – určuje obsah zbytku rámce
  - obsahuje např.: adresu příjemce, identifikátor zprávy, délku zprávy…
- **zabezpečení**
  - = možnost určit/opravit poškození zprávy během přenosu
  - zvýšení bezpečnosti, spolehlivosti
  - jednoduché: XOR mezi všemi bajty rámce

    `bajt zabezpečení = B0 xor B1 xor B2 xor …Bn`
  - složitější: výpočet CRC (8-bit, 16-bit nebo 32-bit CRC)

Příklad:
Zpráva protokolu ISO 9141-2 přenášená mezi ECU vozidla a OBD diagnostickým testerem.



TA – target address (1 bajt)
SA – source addres (1 bajt)
checksum – zabezpečení zprávy (1 bajt)



3 header bytes

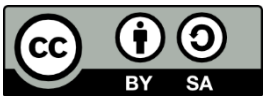
68 – format byte:
0b011110000
Priority P2,P1,P0: 011 (000 je nejvyšší)
Header byte H: 0 (3 byte form)
Interframe response K: 1 (not allowed)
Addressing mode Y: 0 (functional)
Specific. Msg type ZZ: 00
Message type (KYZZ): 0000 (Function)

6A – adresa funkce
F1 – fyzická adresa testeru

UNIVERZITA PARDUBICE
DOPRAVNÍ FAKULTA
JANA PERNERA

Vytvořeno v rámci projektu **Green Deal UPCE**, reg. č. NPO_UPCE_MSMT-2142/2024-4.

Toto dílo podléhá licenci Creative Commons BY-SA 4.0. Pro zobrazení licenčních podmínek navštivte https://creativecommons.org/licenses/by-sa/4.0/.







NÁRODNÍ PLÁN OBNOVY



Financováno Evropskou unií
NextGenerationEU

Univerzita Pardubice
Dopravní fakulta Jana Pernera

Kurz celoživotního vzdělávání

Elektrotechnické systémy pro environmentálně příznivou dopravu

Předmět

Pokročilé systémy digitálního řízení a komunikace v dopravní technice

Studijní materiál

Komunikační sběrnice CAN

Ing. Zdeněk Mašek, Ph.D.



Obsah

1. Komunikační sběrnice v automobilech
2. Příklad použití CAN BUS v osobním vozidle
3. Vlastnosti CAN BUS a způsob komunikace
4. Síťový model CAN BUS
5. Linková vrstva CAN BUS
6. Fyzická vrstva CAN BUS

Počet snímků: 79

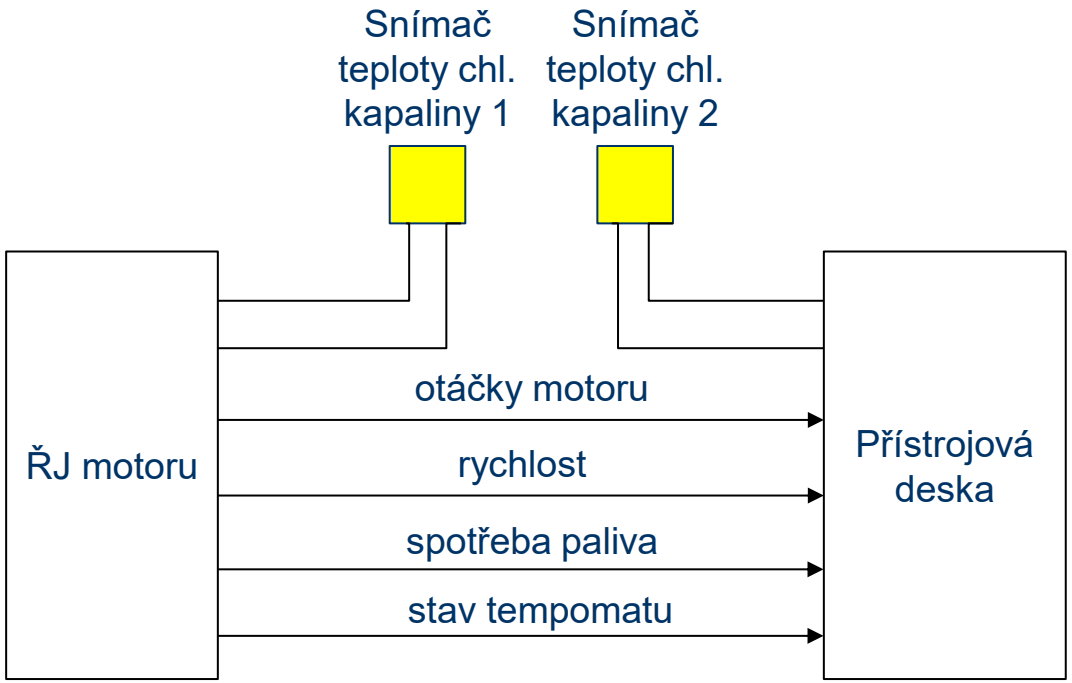
Komunikační sběrnice v automobilech

Sběrnice v automobilech

- **Motivace zavedení sběrnic**
  - Sdílení informací mezi více jednotkami.
  - Redukce kabeláže.

Varianta bez sběrnice

Každá informace je přenášena po vlastním vedení.

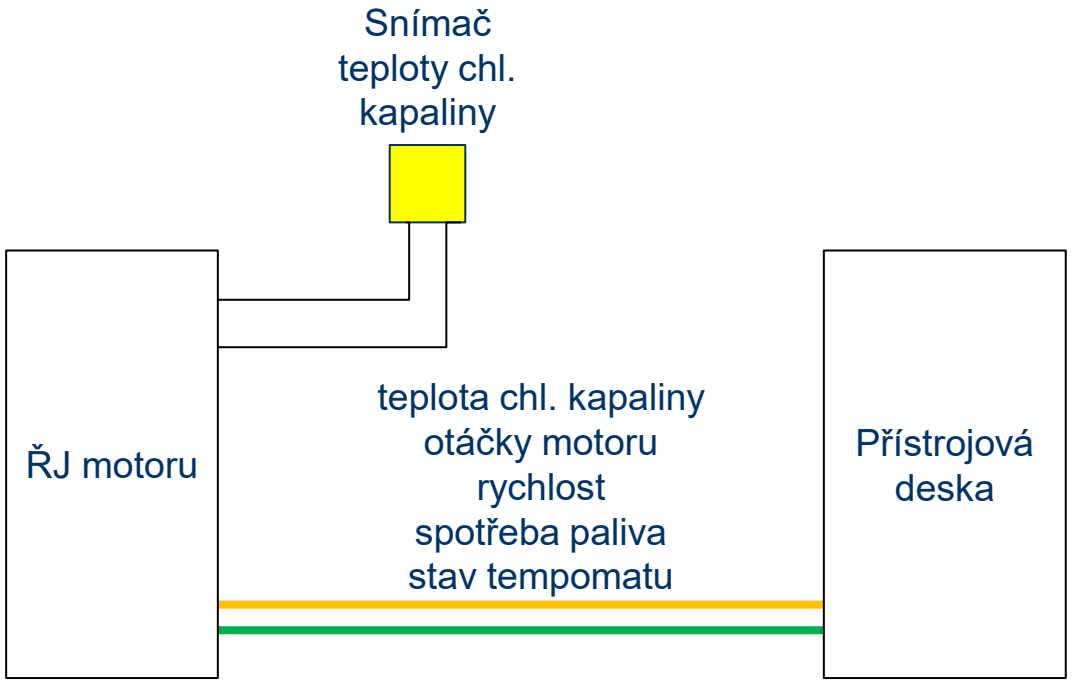


Sběrnice v automobilech

- **Motivace zavedení sběrnic**
  - Sdílení informací mezi více jednotkami.
  - Redukce kabeláže.

Varianta se sběrnicí

Všechny informace jsou přenášeny po sběrnici (u CAN BUS jeden pár vodičů).



Sběrnice v automobilech

- **Vlastnosti komunikačních sběrnic v (osobních) automobilech**
  - Přenosová rychlost (od 20 kBit/s do 1000 Mbit/s)
  - Množství přenášených dat v jedné zprávě
  - Míra zabezpečení přenášených dat (úroveň detekce chyb v přenášené zprávě)
  - Typ přenosového média (metalické vedení, optické vlákno, počet vodičů)
  - Délka sběrnice
  - Topologie sběrnice (strom, hvězda, kruh)
  - Max. počet jednotek připojených ke sběrnici
  - Způsob přenosu dat (kolizní/bezkolizní přístup na sběrnici, simplex, half-duplex, full-duplex)
  - Cena vybudování sběrnice

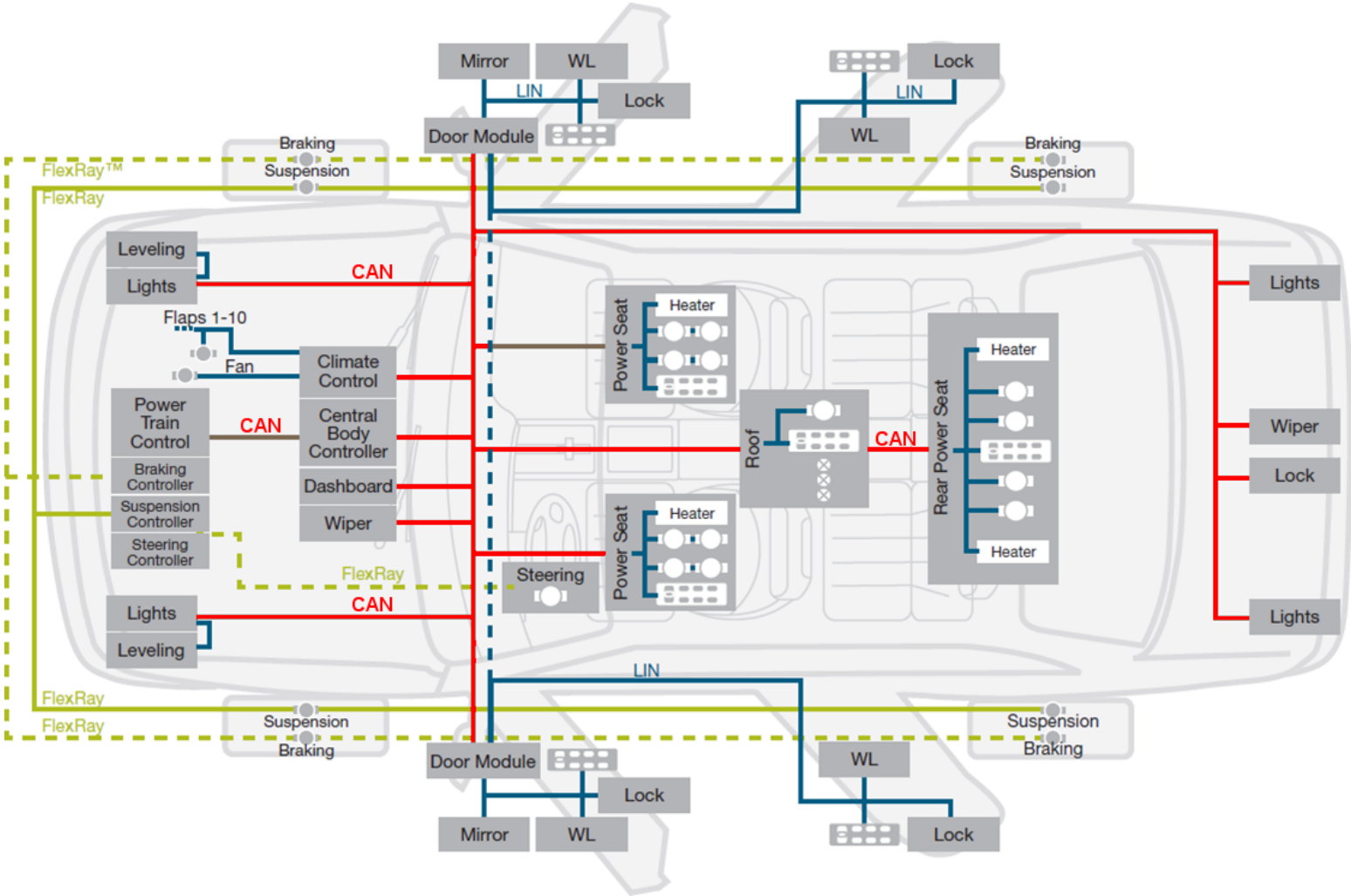
Sběrnice v automobilech

- **Síťové domény**
  - **Drivetrain (řízení pohonu)**
    - Motor
    - Převodovka
  - **Chassis (podvozek)**
    - Řízení
    - Brzdové systémy ABS, ESP
    - Adaptivní odpružení
    - Pohon 4x4
  - **Body (karoserie)**
    - Vnitřní, vnější osvětlení
    - Dveře, okna, zrcátka, zamykání
    - Stírače
    - Vyhřívání sedaček
    - a další …
  - **Telematika (infotainment)**
    - Audio, video - autorádio, navigace, hands-free…
    - Konektivita
  - **Asistenční systémy řidiče (ADAS)**
    - Hlídání mrtvého úhlu
    - Adaptivní tempomat
    - Vedení v jízdních pruzích
    - Brake assist (nouzové brždění)
    - Park assist (pomoc při parkování, automatické parkování)

Seskupování řídících jednotek do sítí podle oblastí, ve kterých řídící jednotky působí.

Sběrnice v automobilech

- **Sběrnice v moderním osobním automobilu**

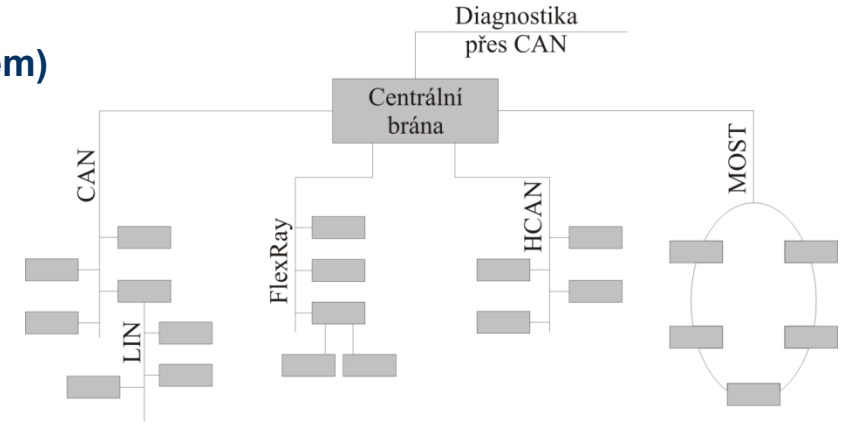


zdroj: [4]

Sběrnice v automobilech

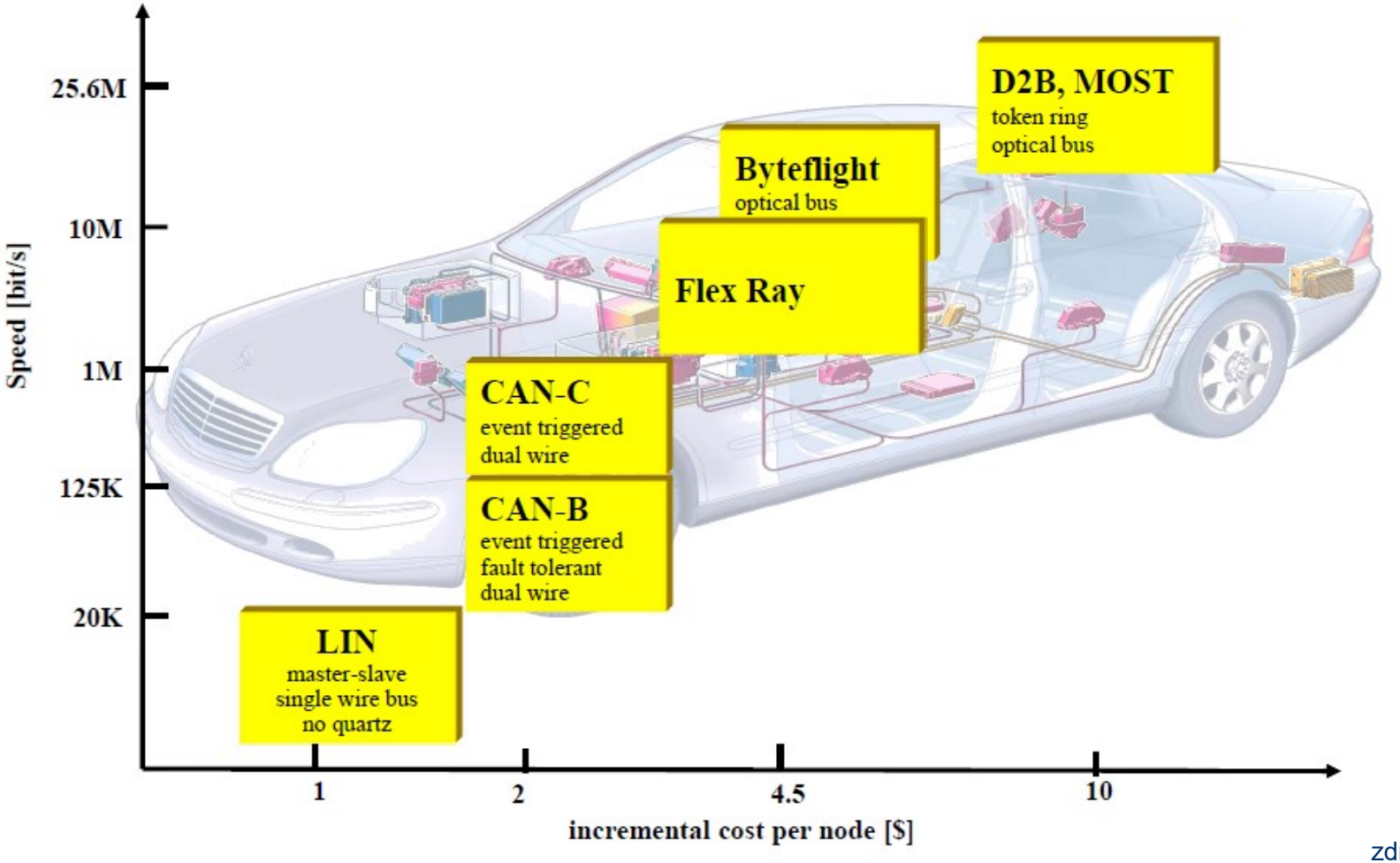
- **Používané komunikační sběrnice v (osobních) automobilech**
  - **LIN**
    - Do 20 kBit/s
    - Chytré snímače a akční členy
    - Stahování oken, zamykání, stěrače, vnitřní osvětlení, panely spínačů apod.
    - Jednovodičová sběrnice pro nenáročný přenos dat, Master-Slave komunikace
  - **CAN**
    - Do 1 Mbit/s (CAN FD do 5 Mbit/s)
    - Kolizní přístup na sběrnici (CSMA) – vyhrává zpráva s nejvyšší prioritou (nejnižším ID)
    - Dobrá detekce chyb v přenášených zprávách
    - Kroucená dvoulinka
    - Převažující sběrnice, která propojuje ŘJ ve vozidle
  - **FlexRay**
    - Do 10 Mbit/s
    - Určeno pro systémy x-by-wire (steer-by-wire, brake-by-wire, adaptivní podvozek)
    - Časový multiplex (TDMA) – bezkolizní přístup na sběrnici, doručení zprávy v zaručeném čase
    - Redundance (2 kanály) – zvýšená bezpečnost
  - **MOST (Media Oriented Transport System)**
    - 150 Mbit/s
    - Audio/Video (rádio, obrazovka, audio zesilovač)
    - Kruhová topologie
    - Kroucená dvoulinka nebo optické vlákno
  - **Automotive Ethernet**
    - 10, 100, 1000 Mbit/s, 1 Gbit/s
    - Asistenční systémy řidiče (kamery, radar/lidar)
    - Diagnostická zásuvka (rychlé flashování ŘJ)

množství přenášených dat



Sběrnice v automobilech

- **Náklady na vybudování sběrnice**

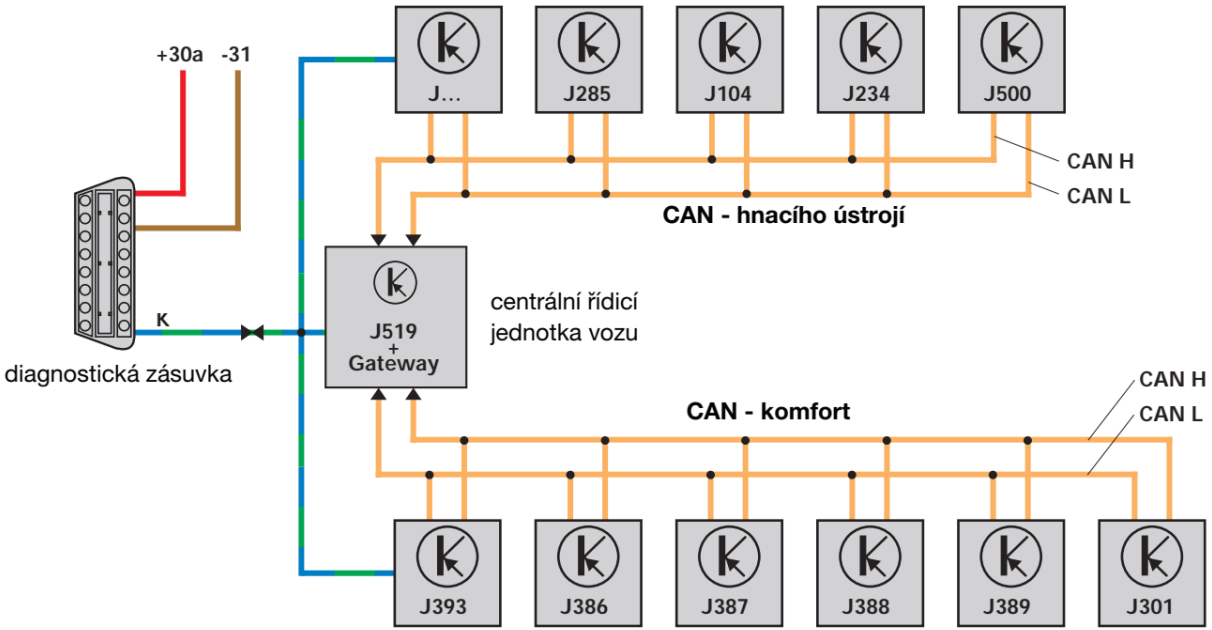


zdroj: [7]

Sběrnice v automobilech



- **Sběrnice ve Škoda Fabia I**
  - **2xCAN**
  - **Diagnostika přes K-line**



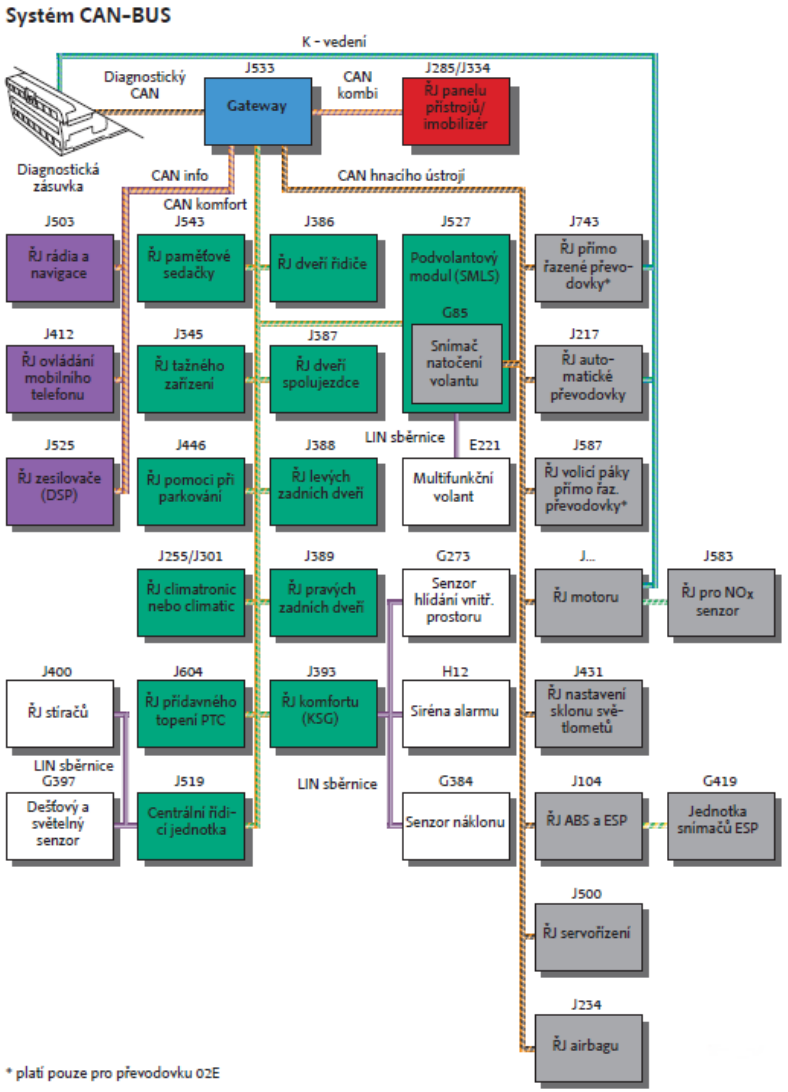
J… ŘJ motoru
J285 ŘJ panelu přístrojů
J104 ŘJ ABS
J234 ŘJ airbagu
J500 ŘJ servořízení
J519 ŘJ centrální ŘJ vozu
J393 centrální ŘJ komfortní elektriky
J386 ŘJ dveří řidiče
J387 ŘJ dveří spolujezdce
J388 ŘJ levých zadních dveří
J389 ŘJ pravých zadních dveří
J301 ŘJ klimatizace

zdroj: [9]

Sběrnice v automobilech



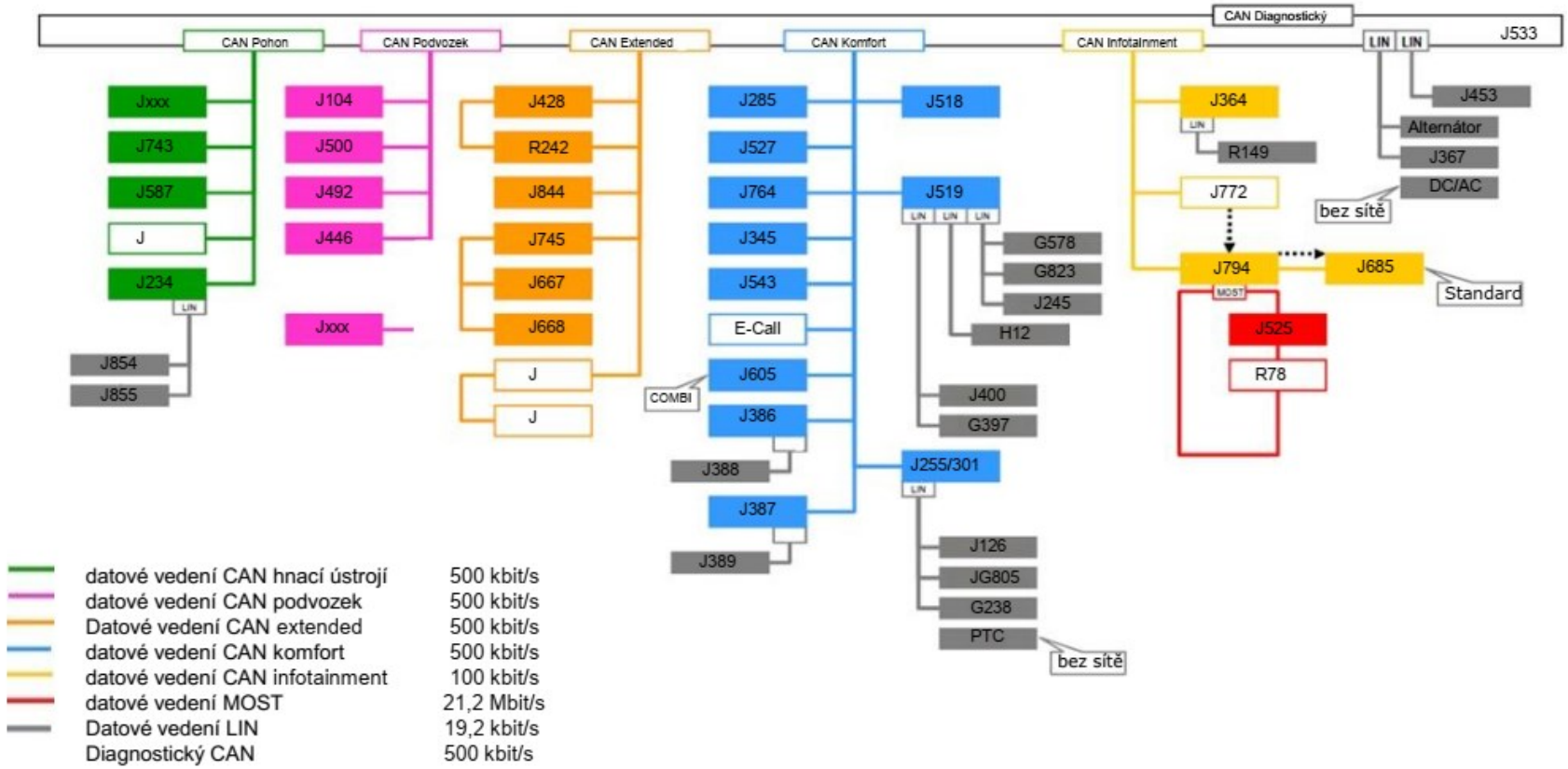
- **Sběrnice ve Škoda Octavia II**
  - **5xCAN, 3xLIN**
  - **Diagnostika přes CAN a K-line**



Sběrnice v automobilech



- **Sběrnice ve Škoda Octavia III**
  - **6xCAN, 10xLIN, 1xMOST**
  - **Diagnostika přes CAN**

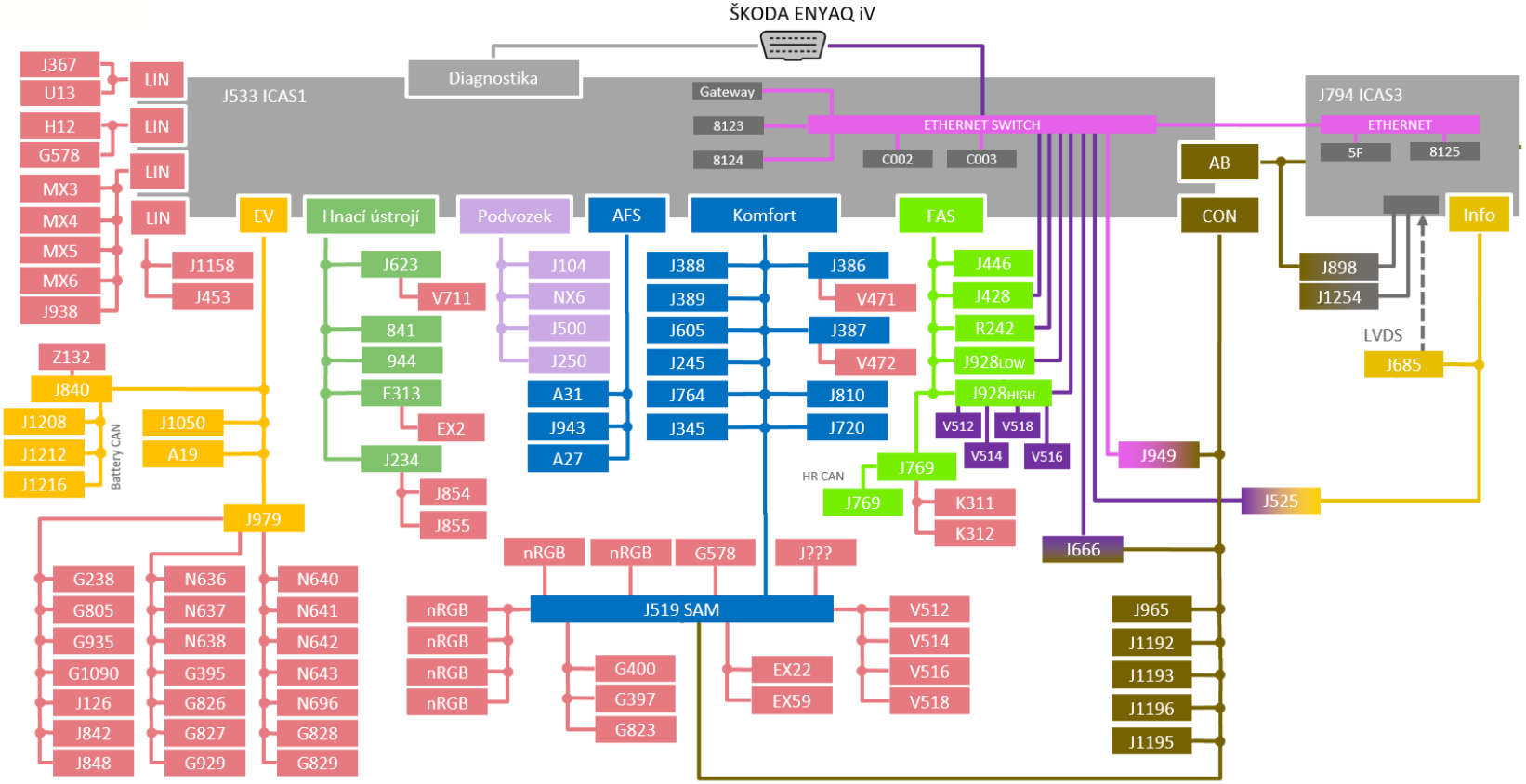


zdroj: [8]

Sběrnice v automobilech



- **Sběrnice ve Škoda ENYAQ iV**
  - 4xCAN, 6xCAN FD, ?xLIN, ETH 100 Mbit, ETH 1 Gbit
  - Diagnostika přes CAN a ETH

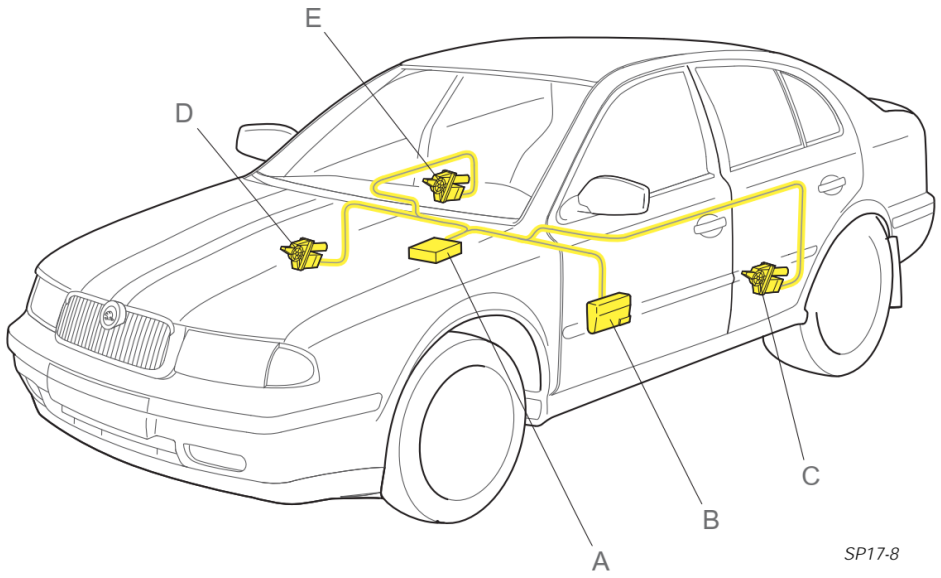


zdroj: [14]

Příklad použití sběrnice CAN v osobním vozidle

Centrální zamykání, ovládání oken a el. zrcátek ve Škoda Octavia I

Systém komfortní elektroniky



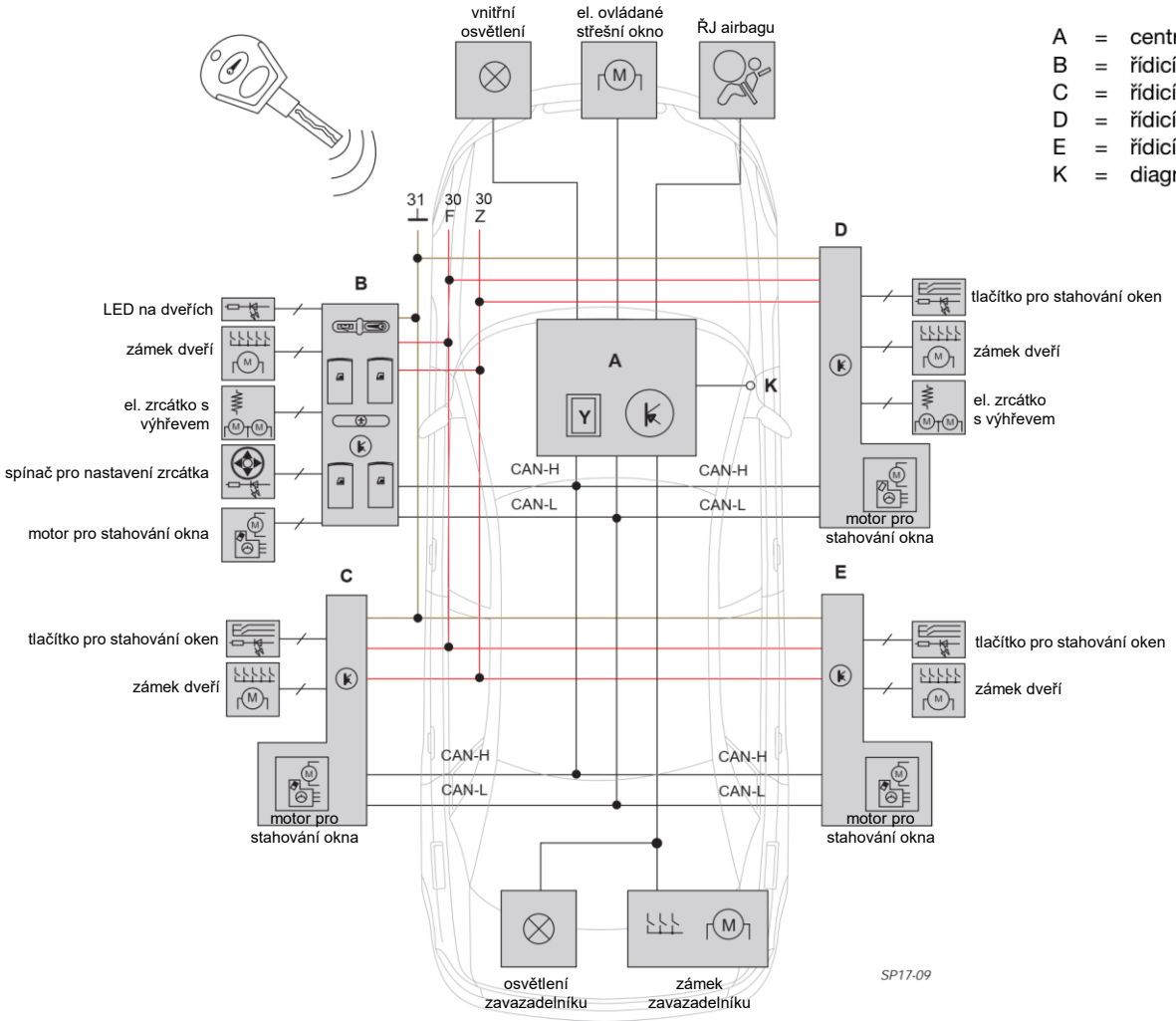
Řídicí jednotky systému komfortní elektroniky

A = centrální řídicí jednotka komfortní elektriky J393
B = řídicí jednotka komfortní elektriky dveří řidiče J386
C = řídicí jednotka komfortní elektriky levých zadních dveří J388
D = řídicí jednotka komfortní elektriky dveří spolujezdce J387
E = řídicí jednotka komfortní elektriky pravých zadních dveří J389

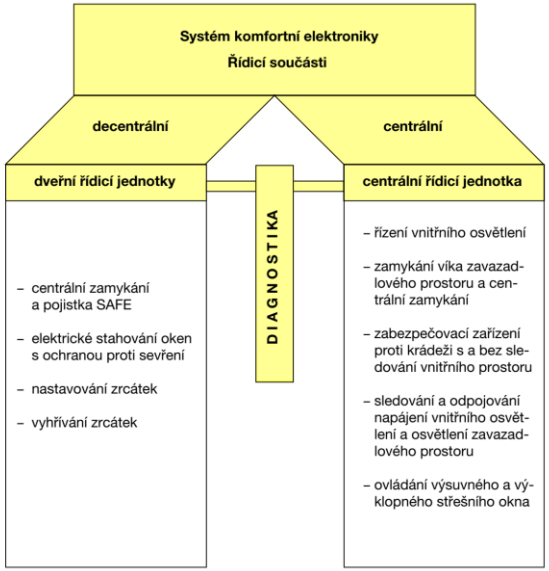
- 1 centrální jednotka a 4 decentrální (ve dveřích)
- Každá dveřní ŘJ pracuje samostatně, centrální jednotka nemá žádnou master-slave funkci.
- Jednotky jsou datově propojeny sběrnicí CAN – značná úspora vodičů.
- Do dveří jde jen 5 vodičů
  - kostra, trvalé + pro stahování oken, trvalé + pro zamykání, datové vodiče CAN-H a CAN-L.
- Diagnostika dveřních jednotek skrz centrální jednotku J393 přes K-line (J393 funguje jako bridge mezi K-line a CAN).

zdroj: [17]

Centrální zamykání, ovládání oken a el. zrcátek ve Škoda Octavia I



A = centrální řídicí jednotka komfortní elektriky J393 s anténou
B = řídicí jednotka komfortní elektriky dveří řidiče J386
C = řídicí jednotka komfortní elektriky levých zadních dveří J388
D = řídicí jednotka komfortní elektriky dveří spolujezdce J387
E = řídicí jednotka komfortní elektriky pravých zadních dveří J389
K = diagnostická přípojka vedení-K

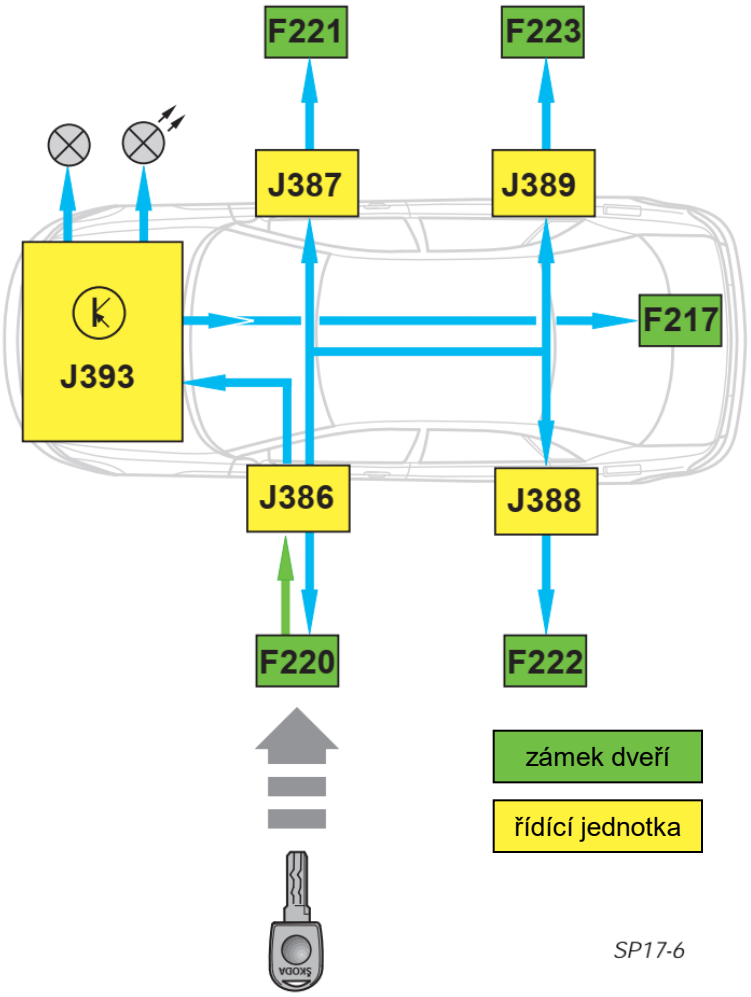


zdroj: [17]

Centrální zamykání, ovládání oken a el. zrcátek ve Škoda Octavia I

Proces zamknutí vozidla klíčem

- Klíč v zámku F220 v poloze „zamknout“.
- Předání informace do J386 (stav mikrospínače).
- V J386 kontrola podmínek zamknutí dveří (nejsou dveře řidiče otevřeny?)
- Aktivace motorku v zámku dveří řidiče – zamknout.
- J386 odešle přes CAN BUS povel „zamknout“ do ostatních dveří a do ŘJ komfortní elektriky J393.
- Všechny dveřní jednotky samostatně zamknou dveře do stavu SAFE.
- ŘJ komfortní elektriky J393
  - zamkne víko zavazadlového prostoru (F217)
  - vypne vnitřní osvětlení
  - aktivuje varovné zařízení proti krádeži (hlídání vnitřního prostoru, náklonu vozidla).
- Všechny ŘJ ve vozidle do několika sekund „usnou“ (přejdou do režimu SLEEP), odběr proudu z baterie klesne na jednotky mA.

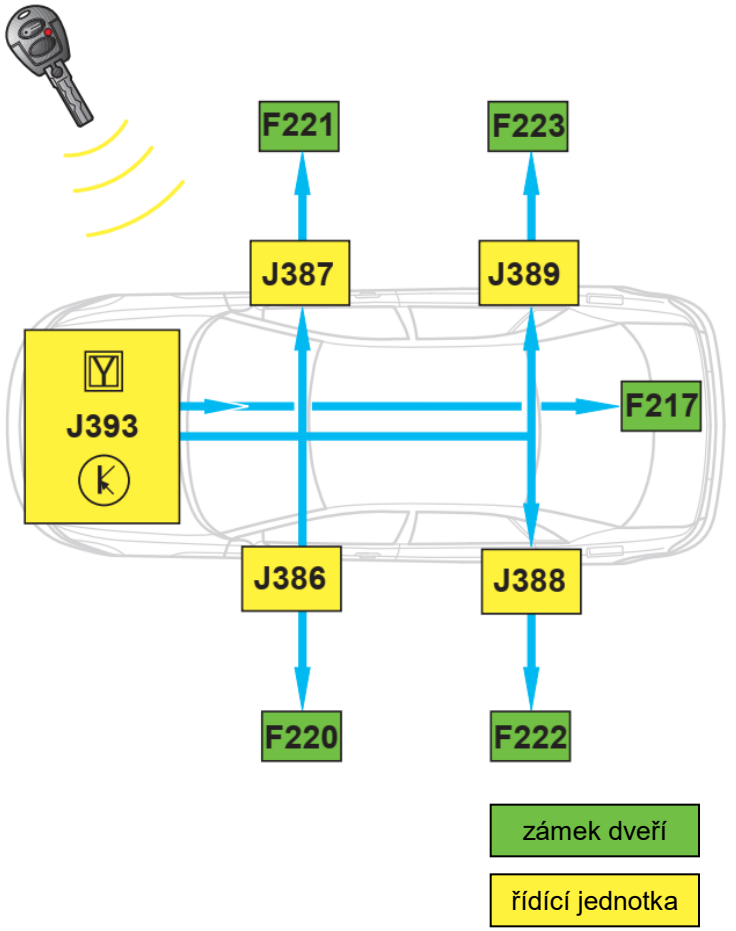


zdroj: [17]

Centrální zamykání, ovládání oken a el. zrcátek ve Škoda Octavia I

Proces odemknutí vozidla pomocí DO

- Stisk tlačítka „odemknout“ na rádiovém DO.
- Příjem signálu anténou do ŘJ komfortní elektriky J393.
- J393 odešle na CAN BUS povel „odemknout“ do všech dveřních jednotek.
- Dveřní jednotky odemknou dveře.
- J393 přímo aktivuje motorek ve víku zavazadelníku.
- J393 aktivuje vnitřní osvětlení vozidla (plynule rozsvítí, plynule zhasne po zavření dveří a zapnutí klíčku zapalování nebo do 30 s).

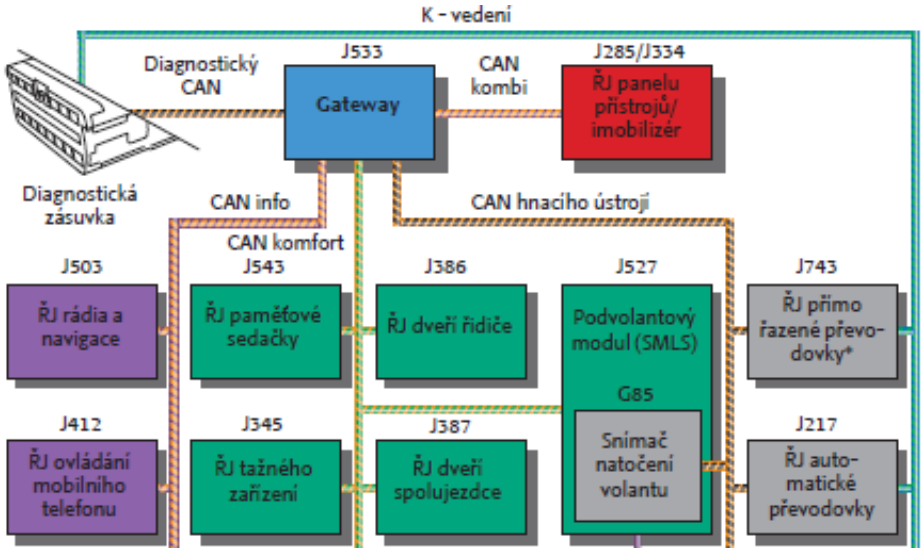


zdroj: [17]

Způsob komunikace na CAN BUS

Funkce Gateway

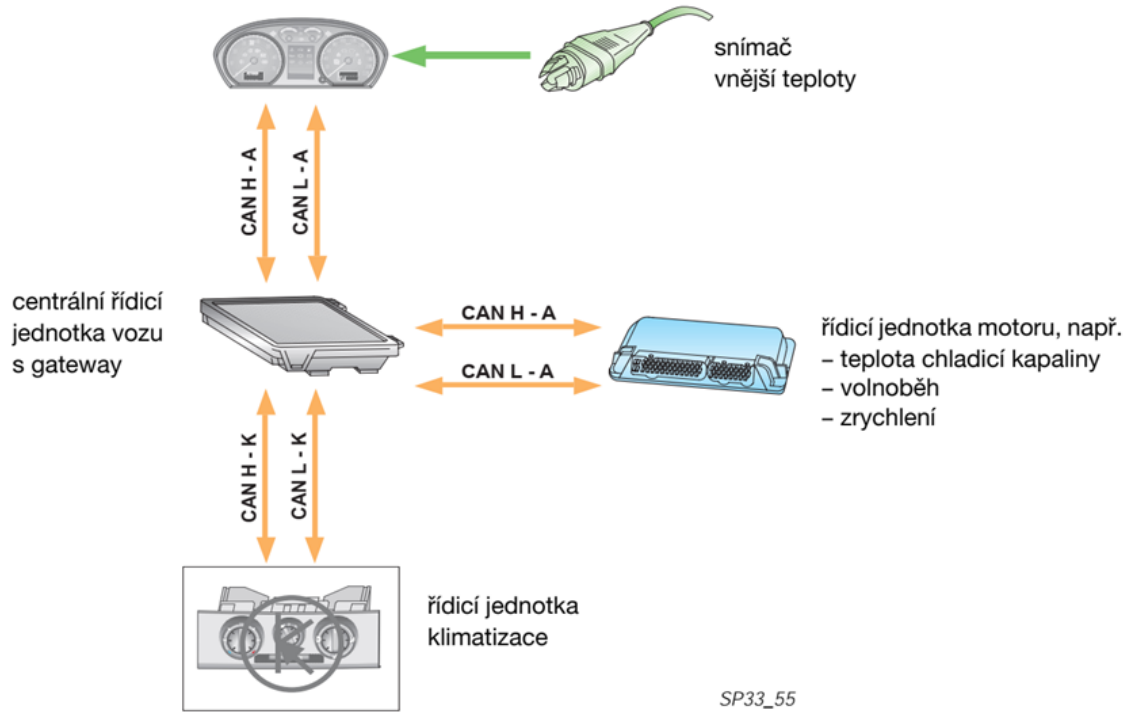
- Elektrické oddělení sběrnic
  - Snížení komunikačního zatížení jednotlivých sběrnic.
  - Zvýšení spolehlivosti (porucha na jedné sběrnici neovlivní funkci ostatních).
- Předávání dat z jedné sběrnice na jinou.
- Ochrana proti hackingu
  - Diagnostická zásuvka oddělena přes Gateway.
  - Není přímé spojení diag. testeru s některou z vnitřních sběrnic vozu.
- Řízení režimu spánku (Sleep mode) – zastavení komunikace a přechod vozidla do režimu se sníženou spotřebou.

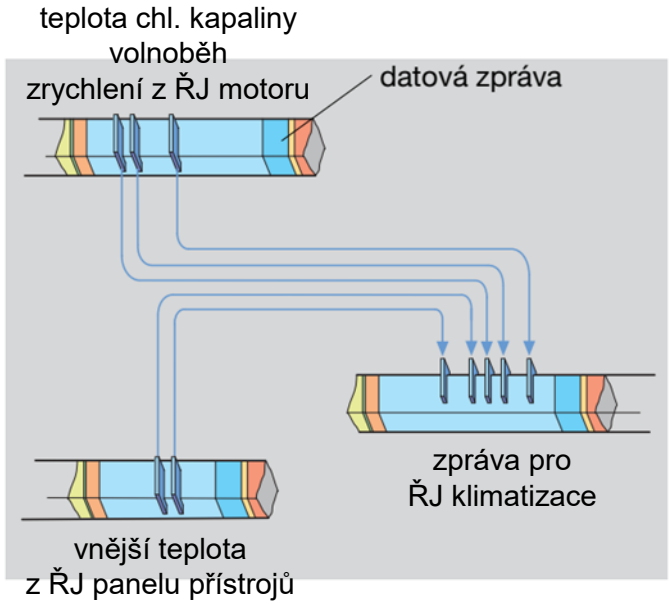


Způsob komunikace na CAN BUS

Princip předávání dat přes Gateway

- Předávání dat z panelu přístrojů a ŘJ motoru (CAN pohonu) do ŘJ klimatizace (CAN komfort).





zdroj: [9]

Způsob komunikace na CAN BUS

Umístění ŘJ Gateway ve vozidlech Škoda

	Panel přístrojů	Centrální ŘJ	Samostatně
Octavia I, Superb I	X		
Octavia II, Superb II, Yeti			X
Fabia I, II bez start-stop		X	
Fabia I, II start-stop			X
Fabia III, IV		X	
Citigo		X	
Scala, Kamiq			X
Octavia III, IV, Superb III, Kodiaq, Karoq			X
Enyaq iV			ICAS 1

zdroj: [14]

Vlastnosti CAN BUS a způsob komunikace

Co je sběrnice CAN ?

- **Controller Area Network** (**CAN**) je sériová komunikační sběrnice pro aplikace běžící v reálném čase
- Vyvinuta firmou BOSCH v roce 1986 pro automobilní techniku
- První nasazení CAN v r. 1992 ve voze Mercedes Benz



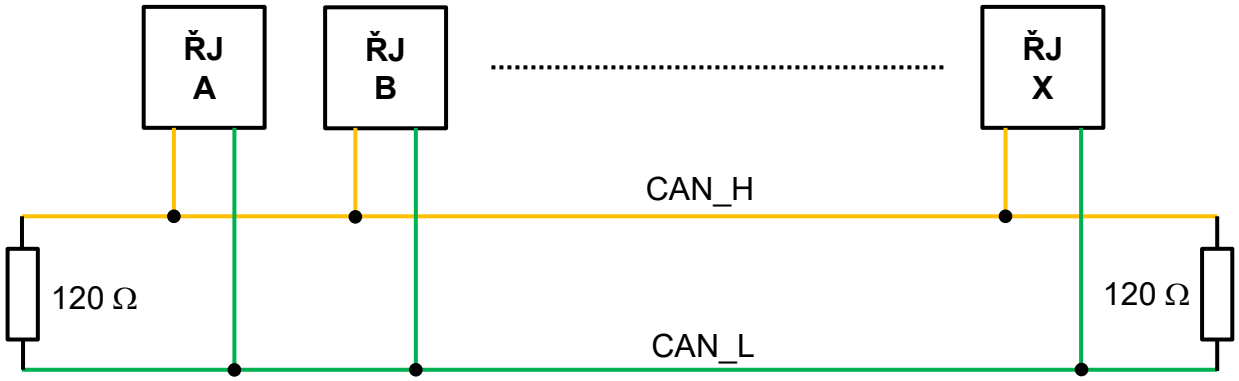
- Rozšíření CAN do více oblastí
  - Nákladní vozidla, autobusy, traktory, pracovní stroje
  - Kolejová vozidla
  - Letectví a kosmonautika
  - Průmyslová automatizace (robotika, pohony)
  - Automatizace budov, vybavení budov (osvětlení, dveře, klimatizace, výtahy)
  - Zdravotní technika
- Hlavním požadavky a důvody, které zapříčinily vývoj a vznik technologie CAN:
  - Sdílení informací mezi více řídícími jednotkami
  - Relativně rychlý přenos dat (komunikace v reálném čase)
  - Důraz na dobrou detekci chyb během přenosu zpráv
  - Úspora kabeláže
  - Nízká cena





Základní vlastnosti sběrnice CAN

- Mezinárodní standard **ISO 11898**
- Dvouvodičová sběrnice
  - Vodiče označené CAN-HIGH a CAN-LOW tvoří kroucený pár.
  - Rozdíl napětí mezi vodiči určuje logickou úroveň (0 nebo 1) na sběrnici.

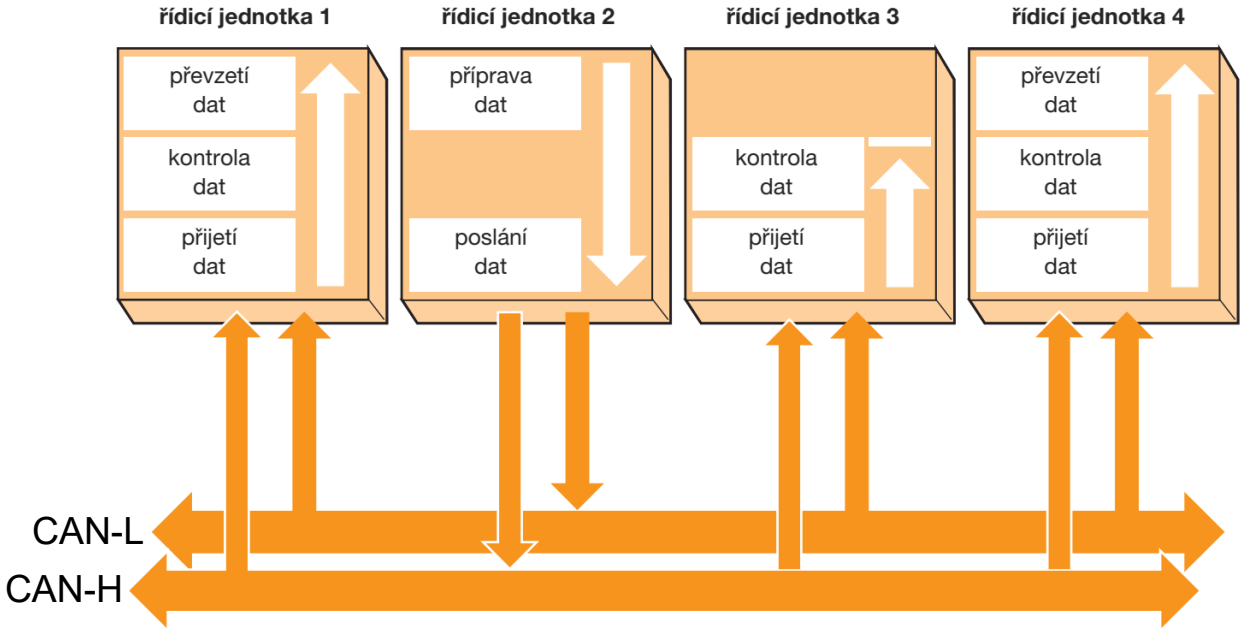




- Data se mezi řídícími jednotkami na sběrnici přenášejí ve zprávách.
- Krátké zprávy – **0 až 8 bajtů dat** v každé zprávě.
  - Novější verze CAN FD má 0 až 64 bajtů dat v každé zprávě.
- Max. přenosová rychlost **1 Mbit** na vzdálenost 40 m.
  - Teoreticky se po sběrnici přenese až cca 9000 zpráv za sekundu při rychlosti 1 Mbit.
  - U novější verze CAN FD je max. přenosová rychlost až 5 Mbit.

Způsob komunikace na CAN BUS

- V jeden okamžik „mluví“ vždy jen jedna řídící jednotka (ŘJ).
- Řídící jednotky se v „mluvení“ neustále střídají.
- Jedna řídící jednotka vysílá a ostatní její zprávu přijímají (princip producent – konzument).
- Vysílačem vyslanou zprávu přijímají všechny ostatní řídící jednotky, ale na zprávu reagují pouze ty, které její obsah k něčemu využijí.
  - ŘJ 2 vysílá, ostatní ŘJ přijímají
  - ŘJ 1 a ŘJ 4 přijmutá data využijí (zpracují), ŘJ 3 nikoliv (na přijatou zprávu nereaguje)

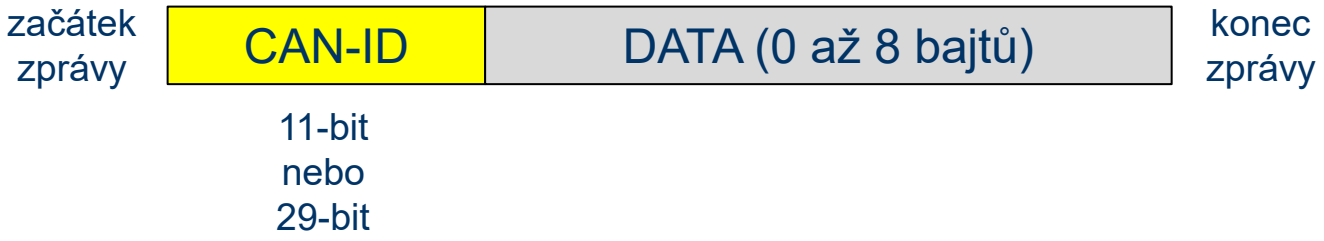


zdroj: [11]

Způsob komunikace na CAN BUS

- Přístup na sběrnici a priorita zpráv
  - **Libovolná řídící jednotka (ŘJ) může odeslat zprávu kdykoliv, když vidí, že na sběrnici nikdo jiný nevysílá (sběrnice je volná)**.
  - Pokud se dvě a více ŘJ rozhodne začít vysílat v jeden a ten samý okamžik, **dojde ke kolizi**.
  - **Kolize je rozpoznána a zprávu odvysílá ta ŘJ, jejíž zpráva má nejvyšší prioritu**. Ostatní ŘJ se přepnou do příjmu, tuto zprávu přijmou, po uvolnění sběrnice se pokusí svojí zprávu odvysílat znovu.
  - **Priorita zprávy** je dána číselným **identifikátorem zprávy** (**CAN-ID**).
  - **Pole CAN-ID jednoznačně určuje význam zprávy a její obsah.**
    - 11-bit CAN-ID ⇒ 2048 různých zpráv se může vyskytovat na jedné CAN sběrnici
    - 29-bit CAN-ID ⇒ 536 870 912 různých zpráv se může vyskytovat na jedné CAN sběrnici

Velmi zjednodušená struktura datové zprávy



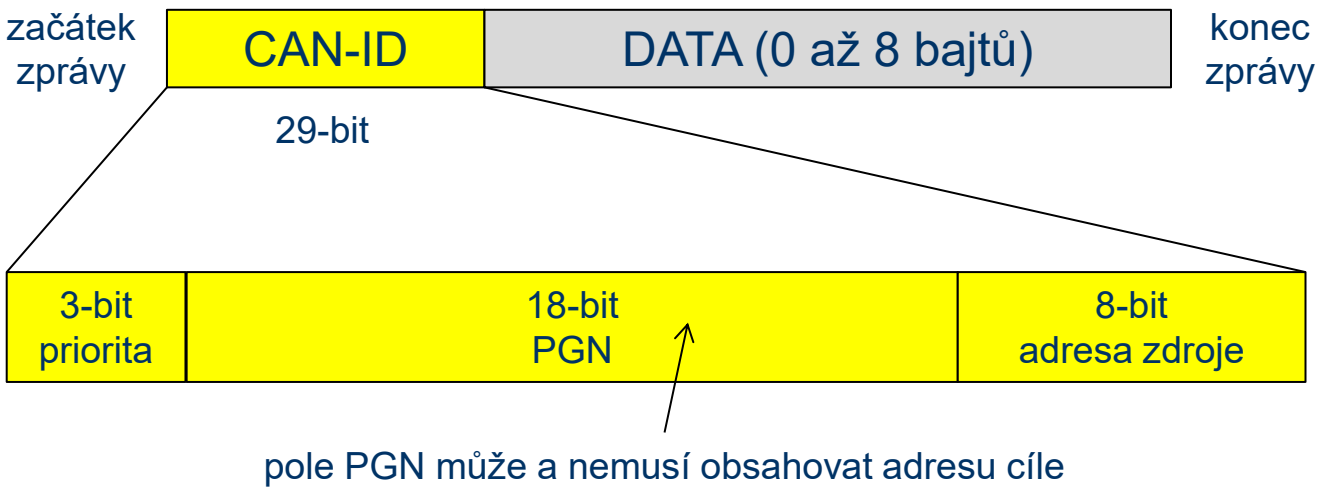


1 B (bajt) má 8 b (bitů)

Způsob komunikace na CAN BUS

- Adresování jednotek na sběrnici
  - **Řídící jednotky (ŘJ) mohou ale nemusí mít adresu na sběrnici**. Závisí to na použitém komunikačním protokolu.
  - Adresa slouží pro jednoznačné určení zdroje a cíle zprávy (která ŘJ zprávu odeslala a pro kterou ŘJ je zpráva určena).
  - **Protokol SAE J1939 používá adresování** (autobusy, nákl. vozidla, traktory, pracovní stroje), **adresa zdroje a cíle je ukryta v CAN-ID každé zprávy**.
  - Pokud není použit žádný komunikační protokol, pak ŘJ adresy nemají a každá ŘJ má svůj seznam zpráv (seznam CAN-ID), které vysílá a které přijímá.

Adresa zdroje a cíle zakódovaná do CAN-ID u protokolu SAE J1939



Způsob komunikace na CAN BUS

Co se ve zprávách přenáší?

- Stavy (např. stavy svorek, spínačů, kontrolek, vnitřní stavy ŘJ apod.)
- Měřené hodnoty (např. ze snímačů, vypočítané hodnoty apod.)
- Povely (např. pro ovládání akčních členů, jiných ŘJ apod.)

Kdy se zprávy přenášejí?

- Opakované vysílání zpráv – perioda od 10 ms do jednotek sekund.
- Při změně stavu – např. stisk tlačítka pro stahování okna (úspora přenosové kapacity sběrnice – zpráva se vysílá jen když je potřeba, ne neustále).
- Na vyžádání – jedna ŘJ odešle požadavek, druhá ŘJ na něj odpoví.
- Jednotky se ve vysílání střídají – chvíli vysílá jedna, pak zase druhá, atd…

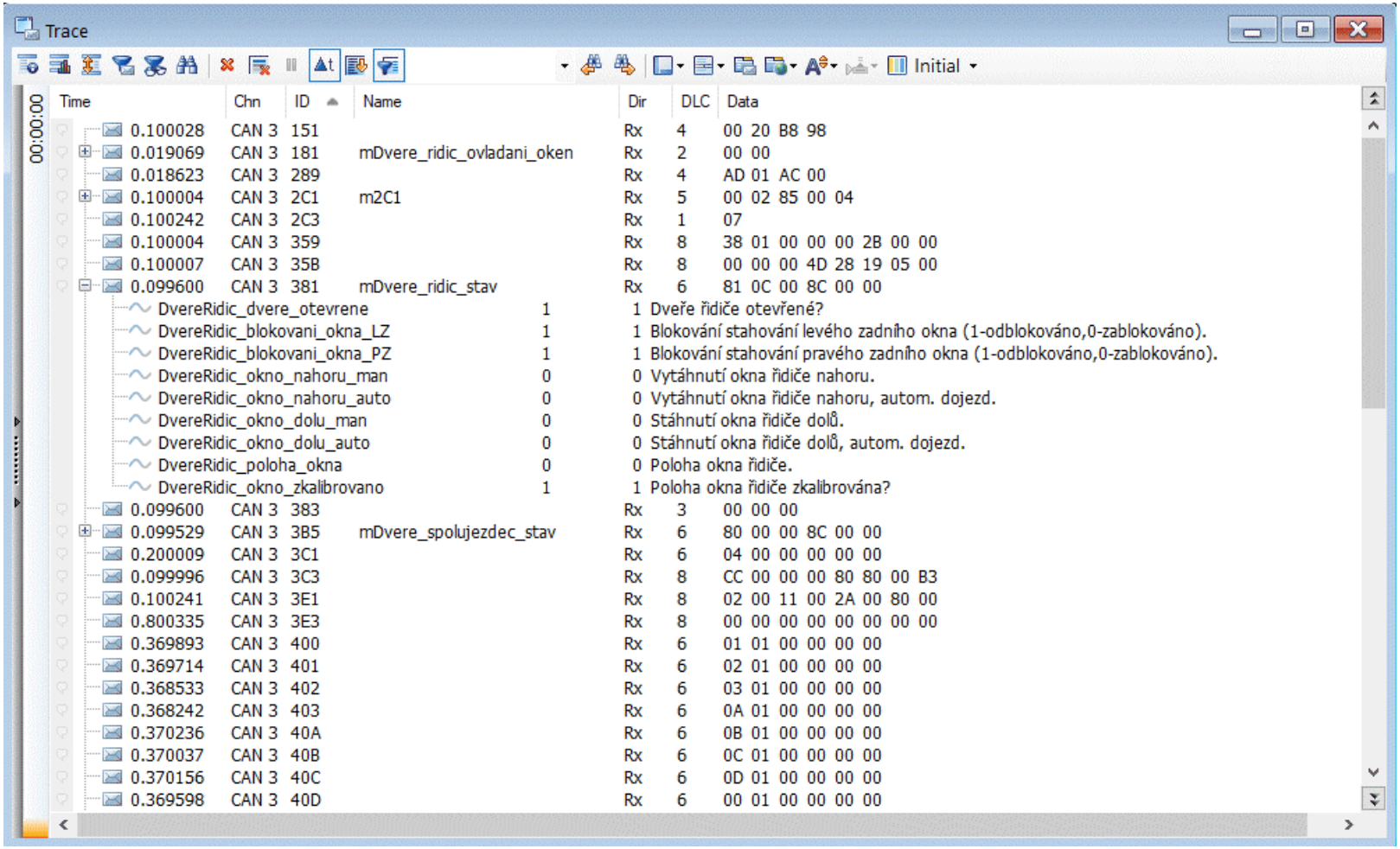
Jak jednotka pozná, že je zpráva přenášená po sběrnici určena pro ní?

- Z identifikátoru zprávy (CAN-ID), který jednoznačně určuje význam a obsah zprávy.

Způsob komunikace na CAN BUS

Zprávy na sběrnici CAN komfort Škoda Octavia II (100 kbit/s)

- Po sběrnici se za jednu sekundu přenese cca 346 zpráv (vytížení sběrnice na 35 % kapacity).

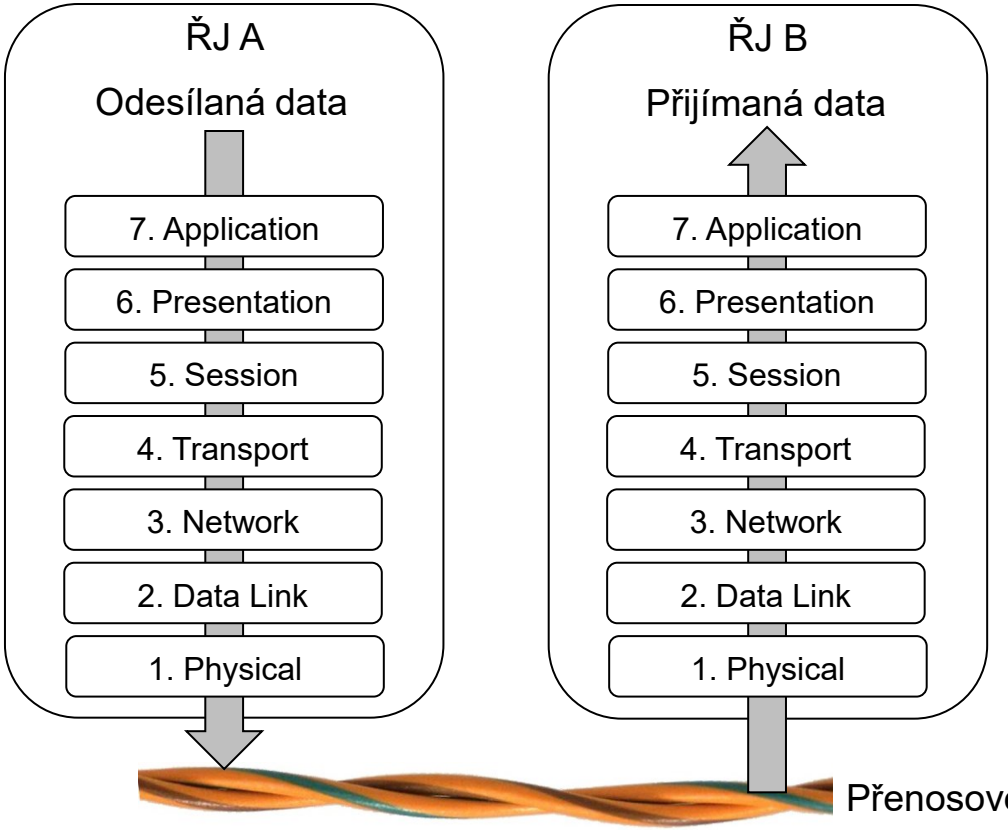


Time	Chn	ID	Name	Dir	DLC	Data
0.100028	CAN 3	151		Rx	4	00 20 B8 98
0.019069	CAN 3	181	mDvere_ridic_ovladani_oken	Rx	2	00 00
0.018623	CAN 3	289		Rx	4	AD 01 AC 00
0.100004	CAN 3	2C1	m2C1	Rx	5	00 02 85 00 04
0.100242	CAN 3	2C3		Rx	1	07
0.100004	CAN 3	359		Rx	8	38 01 00 00 00 2B 00 00
0.100007	CAN 3	35B		Rx	8	00 00 00 4D 28 19 05 00
0.099600	CAN 3	381	mDvere_ridic_stav	Rx	6	81 0C 00 8C 00 00
DvereRidic_dvere_otevrene			1		1	Dveře řidiče otevřené?
DvereRidic_blokovani_okna_LZ			1		1	Blokování stahování levého zadního okna (1-odblokováno,0-zablokováno).
DvereRidic_blokovani_okna_PZ			1		1	Blokování stahování pravého zadního okna (1-odblokováno,0-zablokováno).
DvereRidic_okno_nahoru_man			0		0	Vytáhnutí okna řidiče nahoru.
DvereRidic_okno_nahoru_auto			0		0	Vytáhnutí okna řidiče nahoru, autom. dojezd.
DvereRidic_okno_dolu_man			0		0	Stáhnutí okna řidiče dolů.
DvereRidic_okno_dolu_auto			0		0	Stáhnutí okna řidiče dolů, autom. dojezd.
DvereRidic_poloha_okna			0		0	Poloha okna řidiče.
DvereRidic_okno_zkalibrovano			1		1	Poloha okna řidiče zkalibrována?
0.099600	CAN 3	383		Rx	3	00 00 00
0.099529	CAN 3	3B5	mDvere_spolujezdec_stav	Rx	6	80 00 00 8C 00 00
0.200009	CAN 3	3C1		Rx	6	04 00 00 00 00 00
0.099996	CAN 3	3C3		Rx	8	CC 00 00 00 80 80 00 B3
0.100241	CAN 3	3E1		Rx	8	02 00 11 00 2A 00 80 00
0.800335	CAN 3	3E3		Rx	8	00 00 00 00 00 00 00 00
0.369893	CAN 3	400		Rx	6	01 01 00 00 00 00
0.369714	CAN 3	401		Rx	6	02 01 00 00 00 00
0.368533	CAN 3	402		Rx	6	03 01 00 00 00 00
0.368242	CAN 3	403		Rx	6	0A 01 00 00 00 00
0.370236	CAN 3	40A		Rx	6	0B 01 00 00 00 00
0.370037	CAN 3	40B		Rx	6	0C 01 00 00 00 00
0.370156	CAN 3	40C		Rx	6	0D 01 00 00 00 00
0.369598	CAN 3	40D		Rx	6	00 01 00 00 00 00

CAN
síťový model

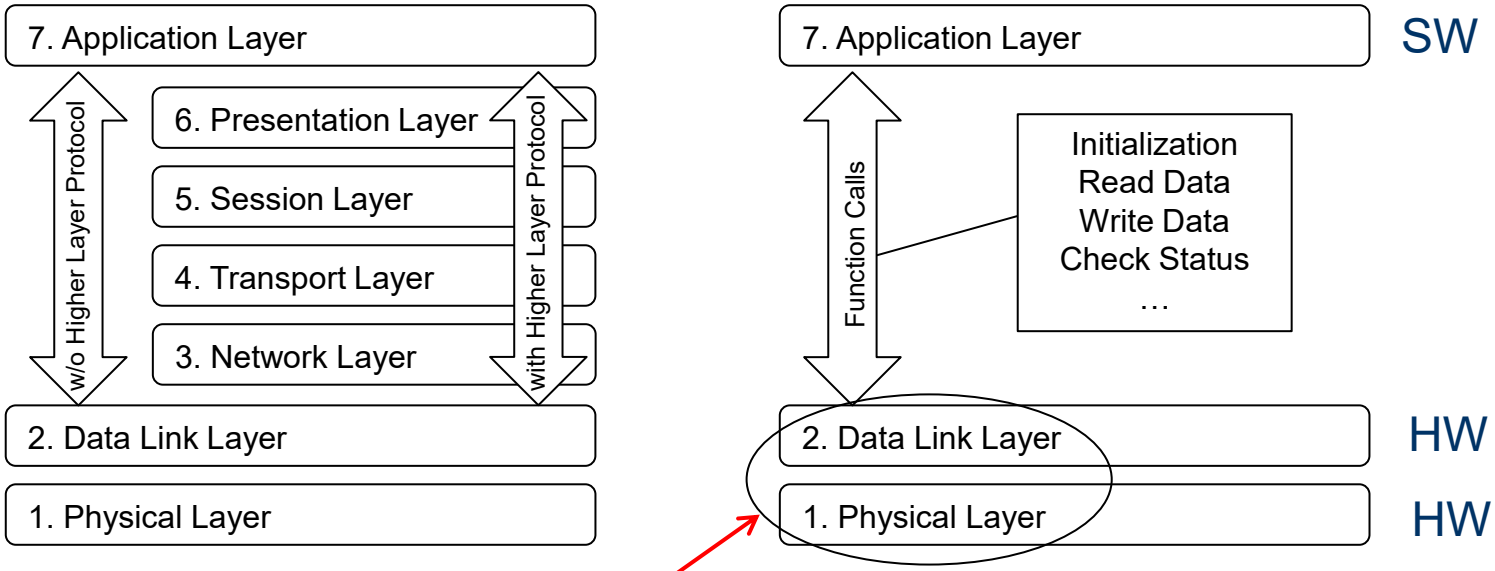
ISO/OSI síťový model

- **OSI síťový model** = definuje vrstvy komunikace
  - Je základnou pro komunikační protokoly, které definují způsoby propojování systémů mezi sebou. Je standardizován v ISO (tzv. ISO/OSI síťový model).
- **Komunikační protokol** = způsob komunikace (přenosu dat) mezi dvěma uzly
  - Protokol působí na jedné nebo více vrstvách OSI modelu.



Daná komunikační vrstva je realizována buď jako program (SW) nebo jako čip (HW) na desce plošného spoje.

CAN síťový model



- ISO 11898 (CAN)
  - definuje pouze linkovou (ISO 11898-1) a fyzickou vrstvu CAN (ISO 11898-2)
  - obě vrstvy jsou implementované v HW (řadič a budič sběrnice CAN)



- Vyšší vrstvy (3. až 7.) jsou využívány až protokoly vyšších vrstev (SW)
  - průmyslové protokoly – CANopen, DeviceNet
  - automotive protokoly – KWP2000, UDS, J1939, ISOBUS

zdroj: [2], web

CAN síťový model – význam vrstev

7. Aplikační – Je realizovaná jako program. Poskytuje rozhraní pro aplikaci, která je nad ní a která komunikuje po síti (vrstva předává data aplikaci a přebírá data od aplikace). Definuje význam dat přenášených po síti – tj. definuje jaká data se budou po síti přenášet (teplota motoru, otáčky…).

2. Linková

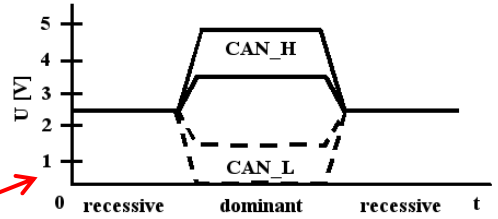
- Definuje strukturu (formát) zpráv přenášených po sběrnici.
- Zajišťuje vysílání a příjem zpráv.
- Vložení a vyjmutí dat do/z zprávy (Data Field v CAN zprávě níže na obrázku).



- Generování přenosové rychlosti (taktu s jakým se vysílají/přijímají bity zprávy).
- Filtrování zpráv - rozhoduje o tom, které zprávy přijaté z fyzické vrstvy mají být zpracovány.
- Detekce, kontrola a signalizace chyb ve zprávě.
- Řízení přístupu řídící jednotky k přenosovému médiu – rozhoduje kdy je sběrnice volná pro nový přenos dat či naopak pro jejich příjem.

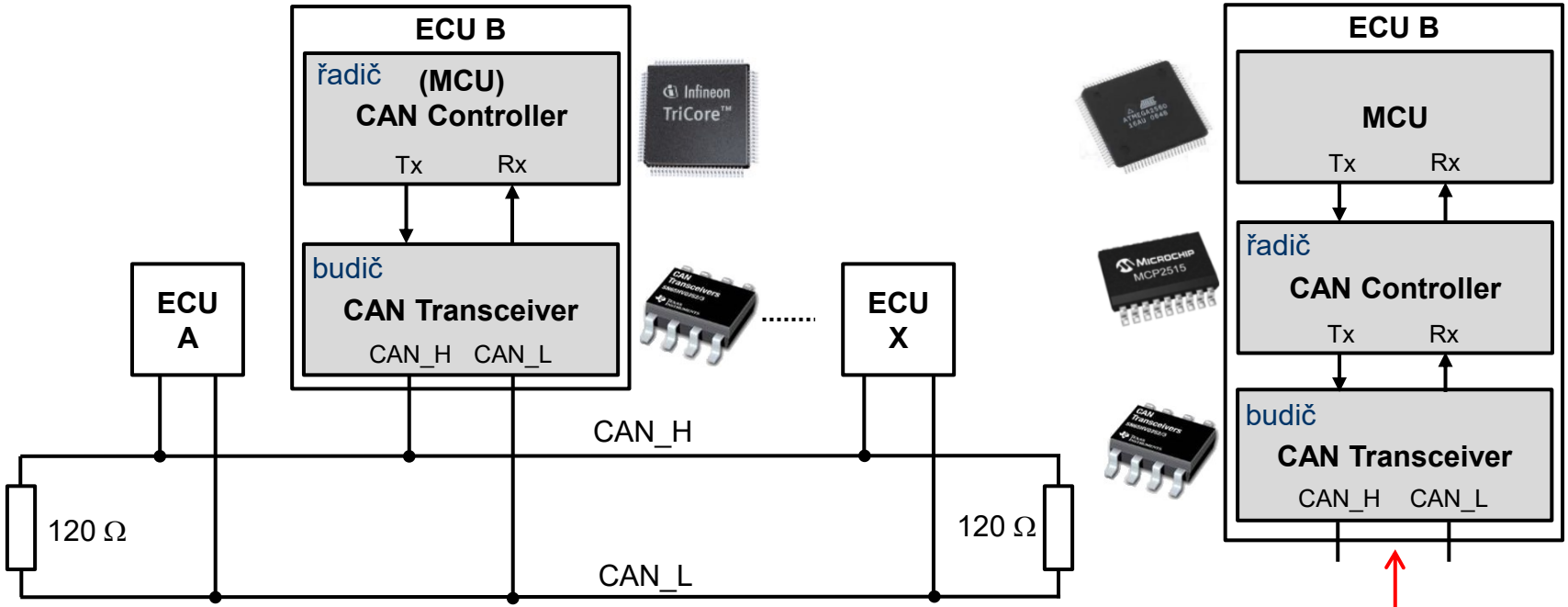
1. Fyzická

- Definuje způsob fyzického spojení mezi uzly v síti.
- Definuje napěťové úrovně signálu a reprezentaci jednotlivých bitů.
- Definuje typ přenosového média (typy metalického vedení, typy konektorů apod.).



zdroj: [2]

Struktura řídící jednotky na CAN BUS



- **Node** – řídící jednotka připojená ke sběrnici CAN
- **MCU** – microcontroller (mikrokontrolér) – mozek ŘJ se SW (7. vrstva)
- **CAN Controller** – **řadič** CAN – realizuje linkovou vrstvu (2. vrstva v OSI modelu), bývá integrován do MCU, ale může být i zvlášť
- **CAN Transceiver** – **budič** (přijímač/vysílač) – realizuje fyzickou vrstvu (1. vrstva v OSI modelu), je to vždy samostatný čip na desce plošného spoje

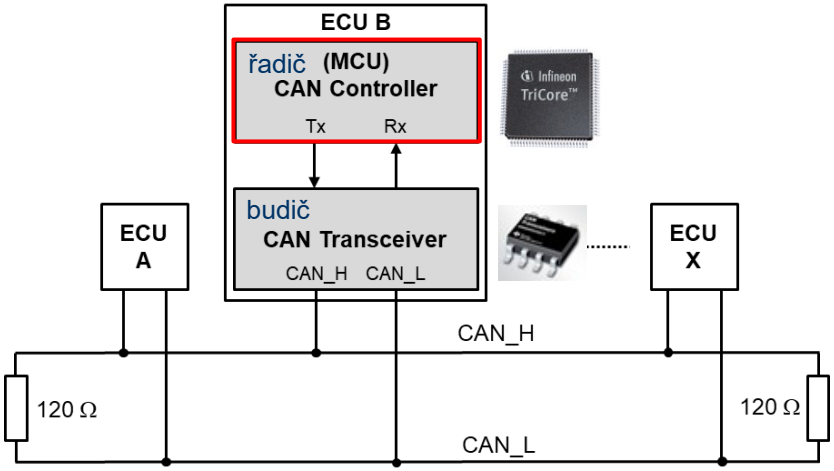
zdroj: [2]

CAN BUS
Linková vrstva



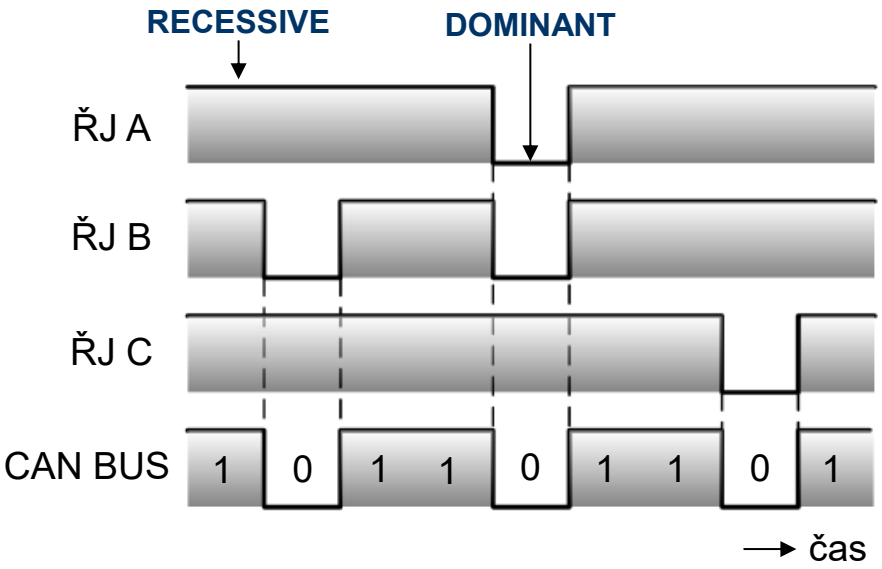
Funkce linkové vrstvy

- **Funkce linkové vrstvy**
  - Definuje strukturu zpráv přenášených po sběrnici.
  - Vložení a vyjmutí dat do/ze zprávy.
  - Generování přenosové rychlosti a synchronizace přijímače s vysílačem.
  - Řízení přístupu uzlu k přenosovému médiu – rozhoduje kdy je sběrnice volná pro nové vysílání dat či naopak pro jejich příjem.
  - Filtrování zpráv - rozhoduje o tom, které zprávy, přijaté fyzickou vrstvou, mají být zpracovány (projdou dál do programu v mikrokontroléru).
  - Detekce a signalizace chyb na sběrnici.
- Linková vrstva je implementována v HW - **řadič CAN** (**CAN Controller**)

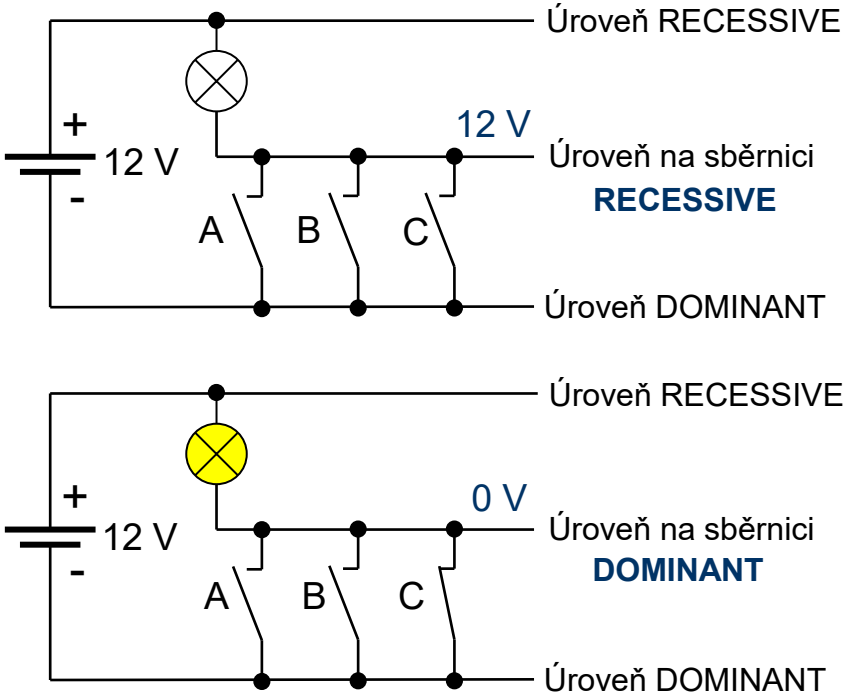


Logické úrovně na sběrnici

- Logické úrovně na sběrnici:
  - **DOMINANT** – aktivní stav – „logická 0“
  - **RECESSIVE** – pasivní stav – „logická 1“



Přirovnání k obvodu se žárovkou a spínači





Úroveň DOMINANT vždy „přebije“ úroveň RECESSIVE. Díky tomu je možné rozlišovat prioritu zpráv a umožnit přednostně odeslat zprávu s vyšší prioritou (s číselně menším CAN-ID).

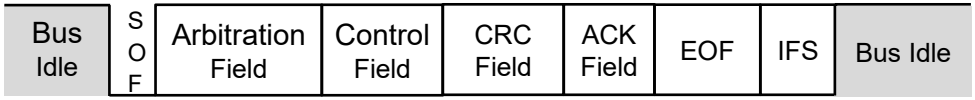
Typy zpráv „Classic CAN“ dle ISO 11898-1

- **Datová zpráva (Data Frame)** - pro běžný přenos dat po sběrnici



  - **ID zprávy** – definuje význam přenášených dat ve zprávě a prioritu (11-bit nebo 29-bit)
  - **Data zprávy** – užitečná data přenášená uvnitř zprávy (0 až 64 bit = 0 až 8 byte)
  - **Ostatní pole** – nutný „balast“ pro zajištění správného přenosu zprávy po sběrnici
  - Hlavní typ zprávy na sběrnici.

- **Žádost o data (Remote Frame)** – jedna ŘJ žádá jinou, aby jí zaslala data



zpráva nepřenáší žádná data (chybí Data Field)

  - často se nepoužívá (je doporučeno jí nepoužívat pro zamezení kolizí při více odpovědích)

- **Chybová zpráva (Error Frame)** – signalizace chyb na sběrnici
  - Vysílá jí každá ŘJ, která při vysílání nebo příjmu detekuje chybu ve zprávě/na sběrnici.
  - Všechny ŘJ na sběrnici se tak dozví, že přijatá zpráva není platná a budou jí ignorovat.
  - Pro zajištění, aby žádná ŘJ na sběrnici nepřijmula zprávu s vadným obsahem.

zdroj: [2]

Provoz na sběrnici CAN

- **Škoda Octavia II, CAN komfort, význam dat neznámý**

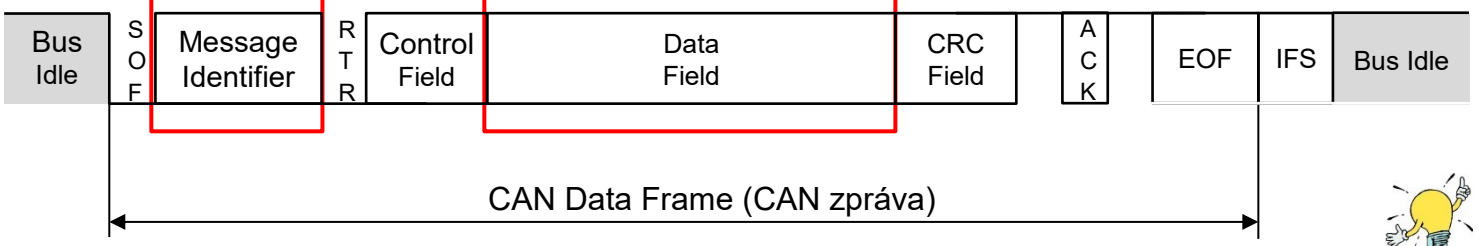
USB-CAN adapter V4.9 - Device Present

File Edit Control Config Help Info

Messages | Signals | Trends | Records | CANopen

Line	Arrival Time [ms]	ID [Hex]	Dir	Data [0, 1, 2, 3, 4, 5, 6, 7]	Description
1	10:19:23.346	3B5	Rx	83 00 4F 80 00 00	
2	10:19:23.349	2C1	Rx	00 00 80 00 04	
3	10:19:24.049	3B5	Rx	83 00 4F 80 00 00	
4	10:19:24.054	381	Rx	81 0C 00 8C 00 00	
5	10:19:24.056	2C1	Rx	00 00 80 00 04	
6	10:19:24.059	601	Rx	00	
7	10:19:24.148	3B5	Rx	83 00 4F 80 00 00	
8	10:19:24.153	381	Rx	81 0C 00 8C 00 00	
9	10:19:24.156	2C1	Rx	00 00 80 00 04	

Datové bajty = užitečná informace přenášená uvnitř zprávy





1 B (bajt) má 8 b (bitů)

Poznámka: Hodnoty v polích ID a Data v obrázku jsou v šestnáctkové soustavě (HEX).

Provoz na sběrnici CAN

- **DAF 105.460, aplikační vrstva CAN SAE J1939**

dvoubajtová informace

jednobajtová informace

Zpráva	Jméno	P...	PGN...	Zdroj	Cíl	Měř.per.	Data
EEC2	Electronic Engine Controller #2	F003	61443	00 Motor		50	F1 00 12 FF FF FF FF FF
EEC1	Electronic Engine Controller #1	F004	61444	00 Motor		19	F0 7D 87 34 11 00 F4 FF
TCO1	Tachograph #1	FE6C	65132	EE Tachograf		50	08 10 00 C0 00 00 00 00
FE	Fuel Economy	FEF2	65266	00 Motor		98	3B 00 00 00 FF FF 00 FF
CCVS	Cruise Control/Vehicle Speed	FEF1	65265	00 Motor		100	F7 00 00 04 01 00 E0 FF
	ECU_vozidla?					1000	FF 0F FF FF FF FF FF FF
ENTEMP	Engine Temperature	FEEE	65262	00 Motor		1001	52 4E 00 27 FF FF FF FF
HRVD	High Resolution Vehicle Distance	FEC1	65217	EE Tachograf		1000	66 99 67 08 C5 DF 84 00
EHR	Engine Hours, Revolutions	FEE5	65253	00 Motor		900	D9 47 03 00 56 18 0C 00
mFEEC_EEbam	Proprietary BAM	ECFF	60671	EE Tachograf			20 12 00 03 FF EC FE 00
mFEEC_EEpck	Multipacket	EBFF	60415	EE Tachograf		100	58 4C 52 54 45 34 37 4D 53

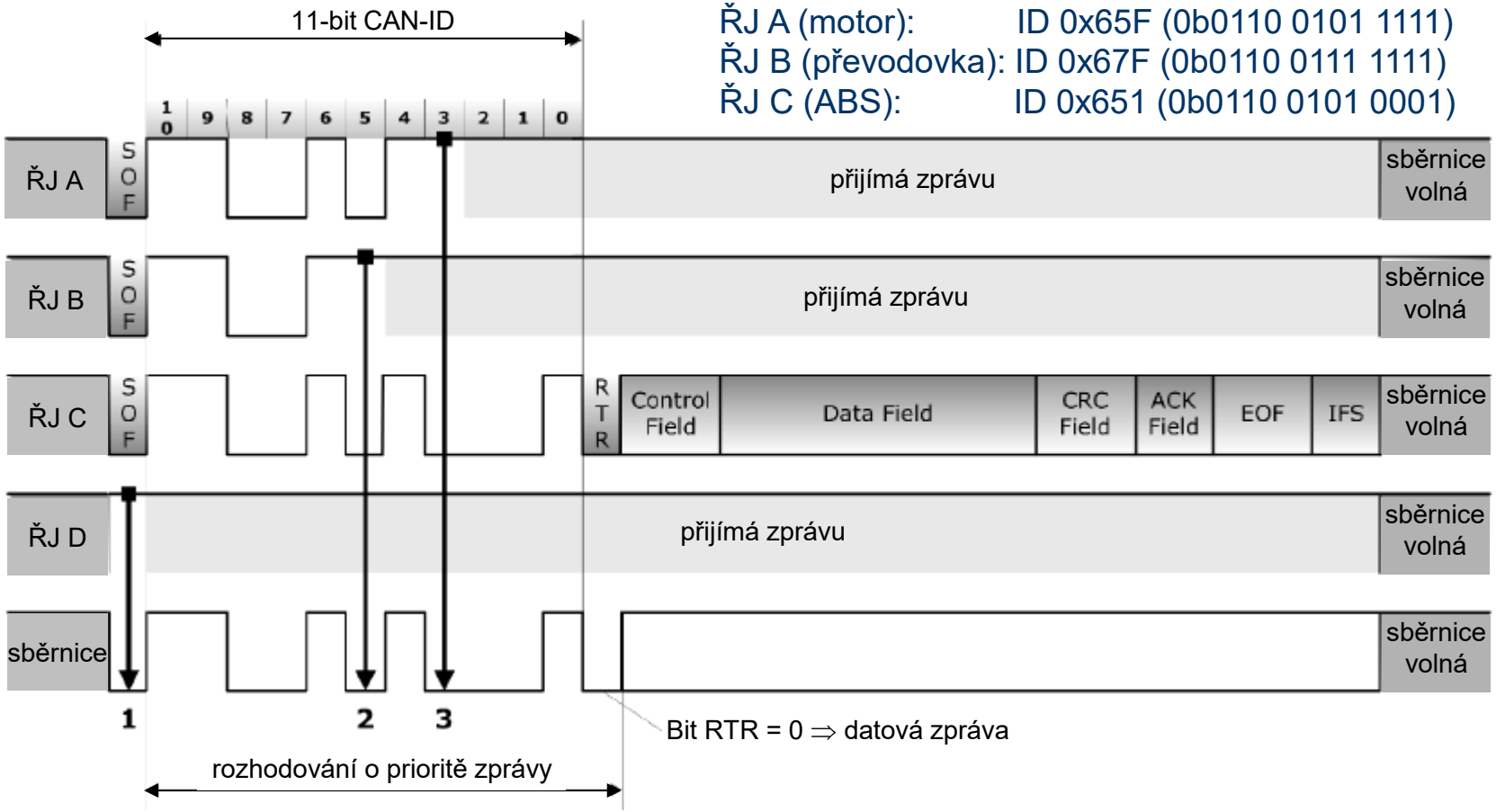
Veličina	Aktuální	Rozsah	Jedn.	Zpráva	SA:
Accelerator Pedal 1 Low Idle Switch			[]	EEC2: 1/1-2	00 Motor
Accelerator Pedal Kickdown Switch			[]	EEC2: 1/3-4	00 Motor
Accelerator Pedal Position 1	0,0	0,0 - 0,0	%	EEC2: 2	00 Motor
Percent Load at Current Speed	16	0 - 125	%	EEC2: 3	00 Motor
Actual Engine - % Torque	9	0 - 49	%	EEC1: 3	00 Motor
Engine Speed	551,3	0,0 - 728,5	rpm	EEC1: 4-5	00 Motor
Tachograph Vehicle Speed	0,0	0,0 - 0,0	km/h	TCO1: 7-8	EE Tachograf
Fuel Rate	2,60	0,00 - 11...	l/hod	FE: 1-2	00 Motor
Engine Coolant Temperature	42	41 - 42	°C	ENTEMP: 1	00 Motor
Fuel Temperature			°C	ENTEMP: 2	00 Motor
Engine Oil Temperature	39	39 - 39	°C	ENTEMP: ...	00 Motor
Total Vehicle Distance			km	HRVD: 1-4	EE Tachograf
Trip Distance			km	HRVD: 5-8	EE Tachograf
Total Engine Hours	10750,1	10750,0 ...	hr	EHR: 1-4	00 Motor



Priorita vysílaných zpráv

- Pokud chce v jeden okamžik vysílat svojí zprávu více ŘJ současně, dostane přednost zpráva s vyšší prioritou (nižším ID).

ŘJ A (motor): ID 0x65F (0b0110 0101 1111)
ŘJ B (převodovka): ID 0x67F (0b0110 0111 1111)
ŘJ C (ABS): ID 0x651 (0b0110 0101 0001)



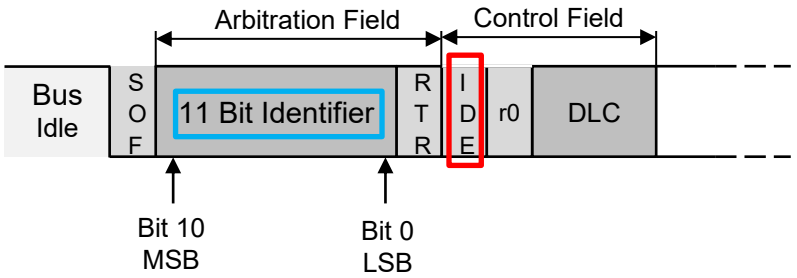
Přístup ke sběrnici dostává ŘJ C – může odvysílat svojí zprávu.

zdroj: [2]

Rozšířená datová zpráva (Extended Data Frame)

- **Dva typy datových rámců (zpráv)** – dle délky identifikátoru
  - **CAN 2.0A – standard frame (11 bit ID)** – počet různých ID je 2^{11} = 2048
  - **CAN 2.0B – extended frame (29 bit ID)** – počet různých ID je 2^{29} = 536 milionů
- Oba typy rámců mohou být provozovány na jedné sběrnici současně.

- **Standardní rámec CAN 2.0A (11 bit ID)**

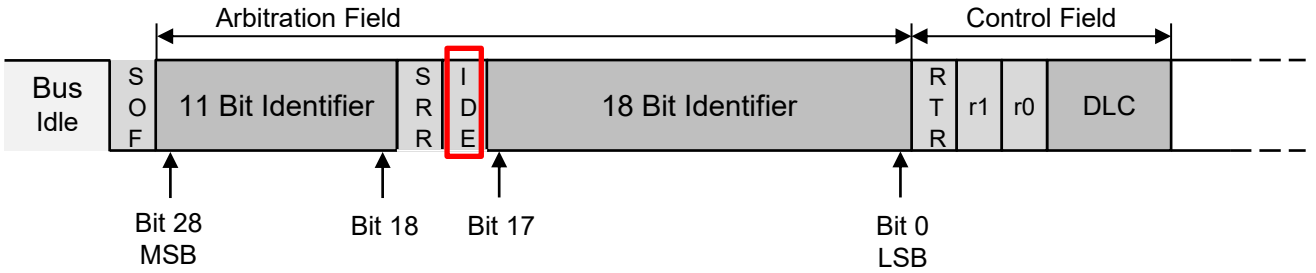


IDE = 0



LSB – nejméně význačný bit
MSB – nejvíce význačný bit

- **Rozšířený rámec CAN 2.0B (29 bit ID)**



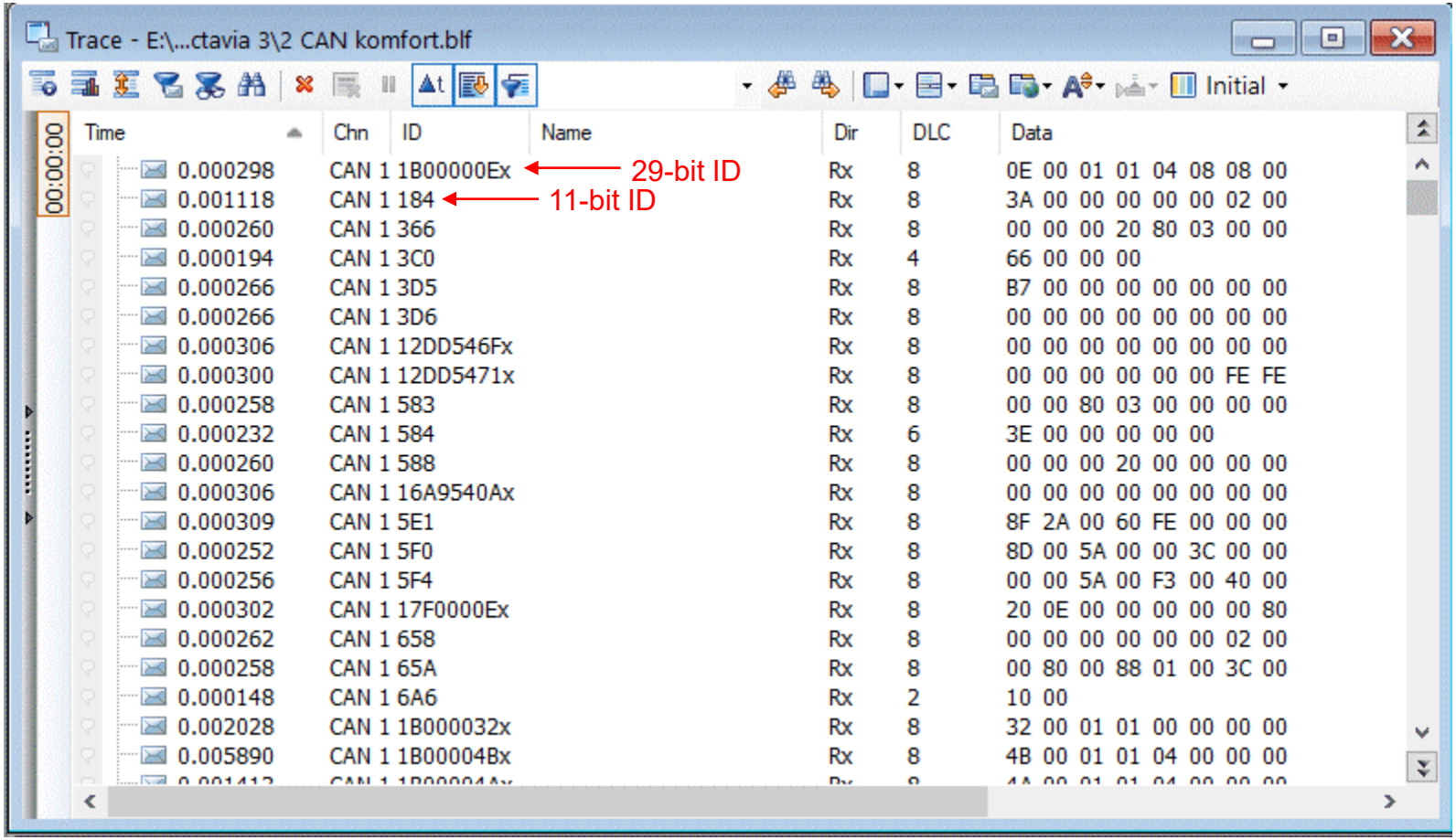
IDE = 1

- **Bit IDE rozlišuje mezi typy rámců**

zdroj: [2]

Smíšený provoz na sběrnici CAN

- Zprávy s 11-bit a 29-bit ID na jedné CAN sběrnici.
- Škoda Octavia III, CAN komfort



Time	Chn	ID	Name	Dir	DLC	Data
0.000298	CAN 1	1B00000Ex		Rx	8	0E 00 01 01 04 08 08 00
0.001118	CAN 1	184		Rx	8	3A 00 00 00 00 00 02 00
0.000260	CAN 1	366		Rx	8	00 00 00 20 80 03 00 00
0.000194	CAN 1	3C0		Rx	4	66 00 00 00
0.000266	CAN 1	3D5		Rx	8	B7 00 00 00 00 00 00 00
0.000266	CAN 1	3D6		Rx	8	00 00 00 00 00 00 00 00
0.000306	CAN 1	12DD546Fx		Rx	8	00 00 00 00 00 00 00 00
0.000300	CAN 1	12DD5471x		Rx	8	00 00 00 00 00 00 FE FE
0.000258	CAN 1	583		Rx	8	00 00 80 03 00 00 00 00
0.000232	CAN 1	584		Rx	6	3E 00 00 00 00 00
0.000260	CAN 1	588		Rx	8	00 00 00 20 00 00 00 00
0.000306	CAN 1	16A9540Ax		Rx	8	00 00 00 00 00 00 00 00
0.000309	CAN 1	5E1		Rx	8	8F 2A 00 60 FE 00 00 00
0.000252	CAN 1	5F0		Rx	8	8D 00 5A 00 00 3C 00 00
0.000256	CAN 1	5F4		Rx	8	00 00 5A 00 F3 00 40 00
0.000302	CAN 1	17F0000Ex		Rx	8	20 0E 00 00 00 00 00 80
0.000262	CAN 1	658		Rx	8	00 00 00 00 00 00 02 00
0.000258	CAN 1	65A		Rx	8	00 80 00 88 01 00 3C 00
0.000148	CAN 1	6A6		Rx	2	10 00
0.002028	CAN 1	1B000032x		Rx	8	32 00 01 01 00 00 00 00
0.005890	CAN 1	1B00004Bx		Rx	8	4B 00 01 01 04 00 00 00

CAN FD (Flexible Data rate)

- **CAN FD (ISO 11898-1 od r. 2015)**
  - Rok uvedení 2012. Rozšíření původního standardu linkové vrstvy ISO 11898-1.
  - **Pro rychlejší přenos většího množství dat – využití např. pro aktualizaci SW v řídících jednotkách vozidla (zkrácení času aktualizace)**.
  - **Až 64 bajtů užitečných dat** v jedné zprávě (oproti 8 bajtům u klasického CAN).
  - Zvýšená max. rychlost přenosu užitečných dat až 8x (**max. 5 Mbit/s**).
  - Přepínání komunikační rychlosti v průběhu vysílání zprávy.
  - Lepší zabezpečení dat ve zprávě (spolehlivější rozpoznání chyby ve zprávě).

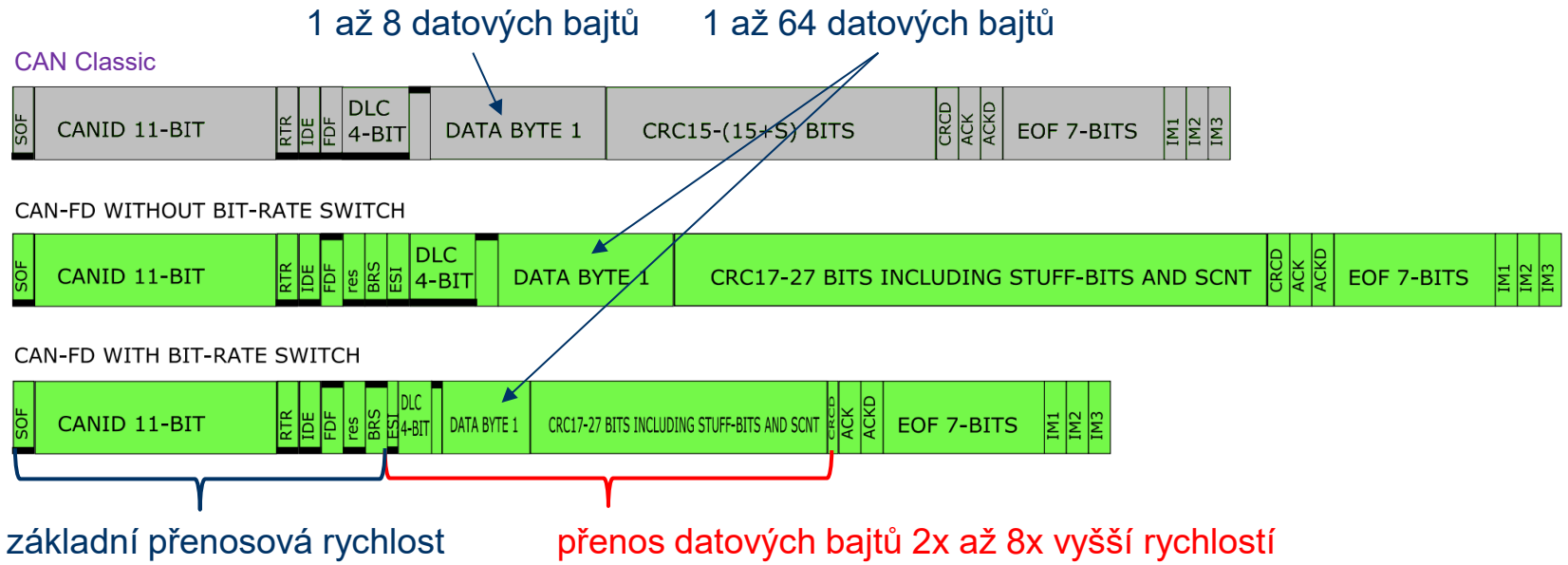
Doba potřebná na odvysílání jedné CAN zprávy – porovnání CAN Classic s CAN FD

CAN Classic, 8 B dat	základní přenosová rychlost
CAN FD, 64 B dat	základní přenosová rychlost, 8x větší počet dat
CAN FD, 64 B dat	8x vyšší přenosová rychlost, 8x větší počet dat

zdroj: [16]

CAN FD (Flexible Data rate)

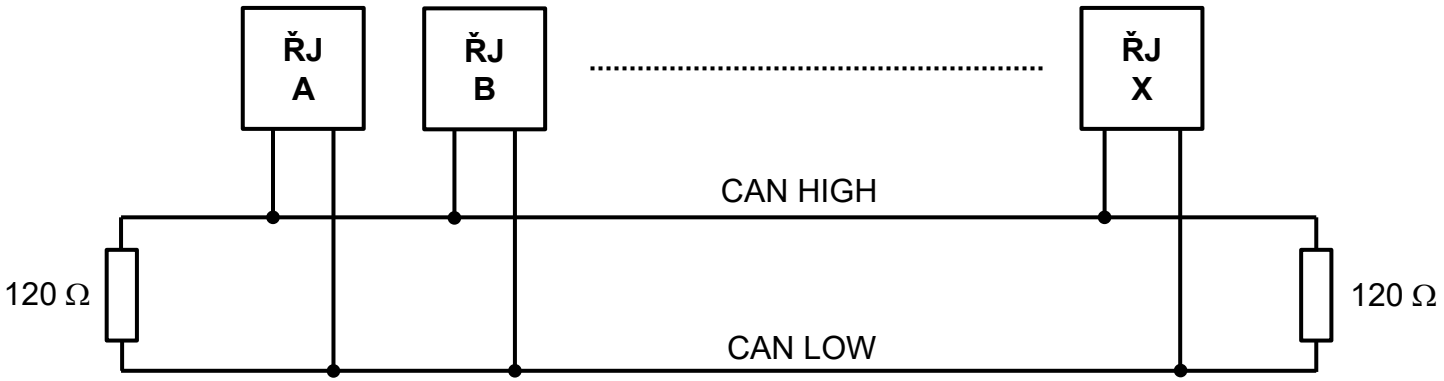
- **CAN FD (ISO 11898-1 od r. 2015)**
  - CAN FD datová zpráva má jinou strukturu než CAN Classic zpráva.



  - **Existují dva typy CAN řadičů (CAN Controller)**
    - **Classic CAN** - umí jen klasický formát datové zprávy dle původního standardu
    - **CAN FD** - umí klasický formát datové zprávy a FD formát datové zprávy
  - CAN FD vyžaduje jiný řadič (CAN Controller) a rychlejší budič (CAN Transceiver) v řídící jednotce oproti původnímu standardu CAN Classic.

zdroj: [16]

CAN BUS
Fyzická vrstva



Varianty fyzické vrstvy CAN BUS

V automobilech se lze setkat s těmito druhy fyzické vrstvy sběrnice CAN:

- CAN High Speed (ISO 11898-2)
  - Rok uvedení 1993 (od počátku vzniku CAN).
  - Max. rychlost 1 Mbit/s.
  - **Zakončovací rezistory 120 Ohm na koncích sběrnice.**
  - Motor, převodovka, brzdy, airbag, OBD zásuvka a další.

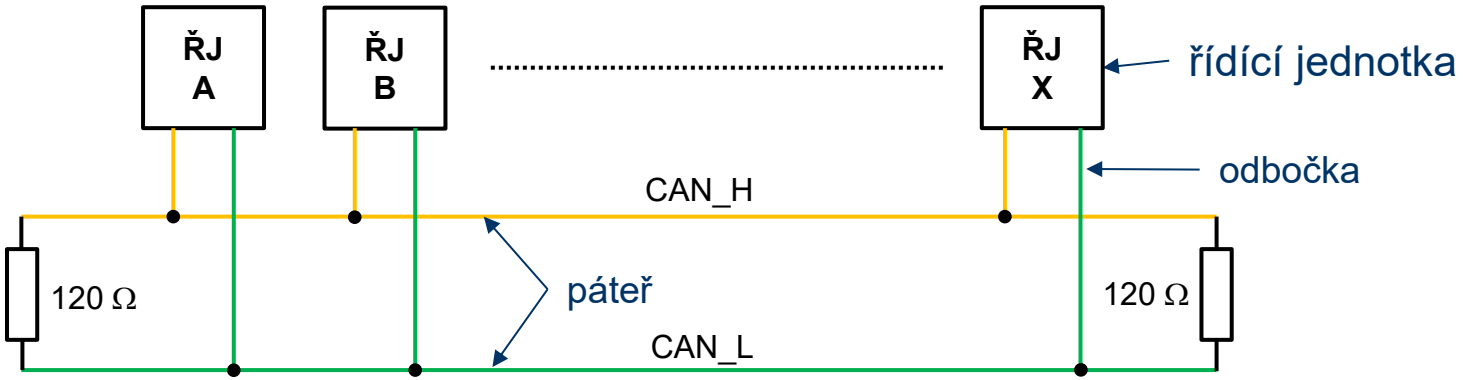
nejvíce používaná fyzická vrstva

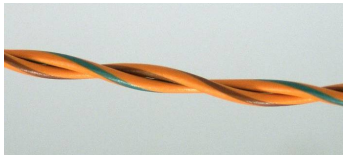
- CAN Low Speed, Fault Tolerant (ISO 11898-3)
  - Rok uvedení 2006.
  - Sběrnice odolná vůči el. závadám na vedení (přerušení, zkrat jednoho vodiče), přechod na jednovodičovou komunikaci při poruše jednoho z vodičů.
  - Max. 125 kbit/s (odolnost vůči el. závadám na vedení vykoupena nižší max. přenosovou rychlostí).
  - **Nemá zakončovací rezistory 120 Ohm na koncích sběrnice!**
  - Použití v okruhu komfortní elektriky (osvětlení, zamykání, okna, klimatizace, apod.).
- Single-wire CAN (SAE J2411)
  - Rok uvedení 2000 ?
  - **Jednovodičová sběrnice.**
  - Rychlost 33,333 nebo 83,333 kbit/s.
  - Používaná ve vozidlech koncernu GM (GM-LAN, SWCAN) v okruhu komfortní elektriky.
- CAN FD (ISO 11898-2 od r. 2016)
  - Zpětně kompatibilní s CAN High Speed.
  - Budič je rychlejší, aby uměl vysílat a přijímat CAN FD zprávy přenášené vyšší rychlostí (až 5 Mbit/s).

Navzájem nekompatibilní !

Topologie sběrnice CAN high speed

Topologie sběrnice CAN dle ISO 11898-2 (CAN high speed)





- **Metalické vedení** – **kroucená dvoulinka** (vodič **CAN_H** a vodič **CAN_L**)
  - Nestíněná (unshielded) – ve vozidlech převažuje
  - Stíněná (shielded – kabel má kovové opletení, které se ukostřuje)
- **Diferenciální sběrnice** – logická hodnota bitu je dána rozdílem napětí mezi vodiči **CAN_H** a **CAN_L** – odolnost proti rušení
- **Zakončovací rezistory 120 Ohm** – až na koncích vedení (eliminace odrazů signálu na vedení)
  - Umístěné buď na kabelu (v konektorech) nebo v krajních uzlech (na DPS uvnitř ŘJ).
  - Bez nich komunikace nefunguje!

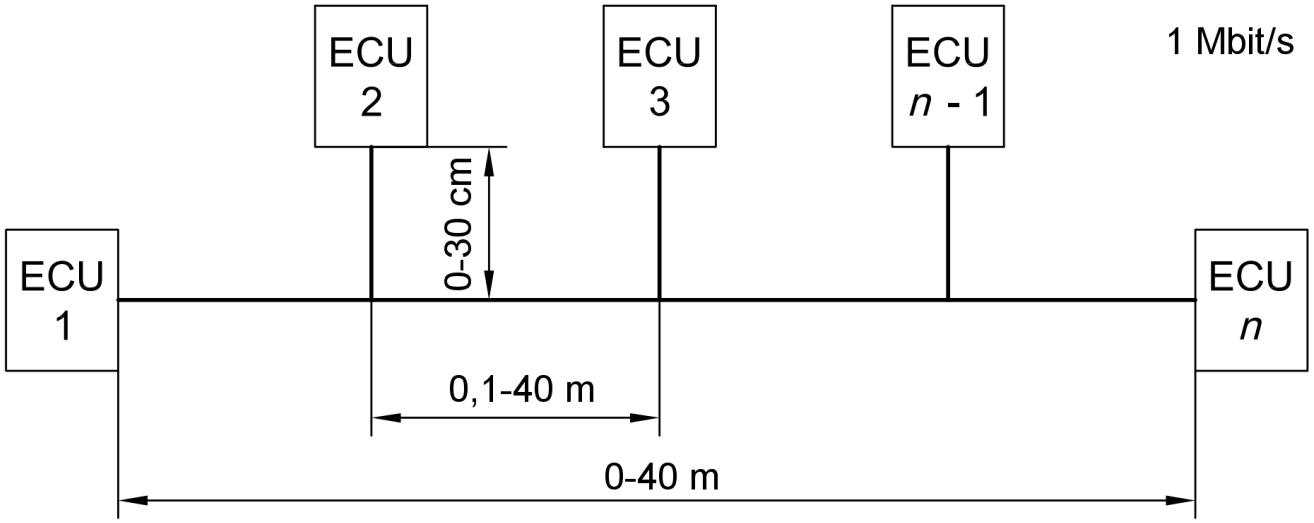
Při vypnutém napájení všech uzlů se multimetrem naměří odpor 60 Ohm mezi CAN_H a CAN_L.

zdroj: [1],[2]

Topologie sběrnice CAN high speed

Doporučené délky vedení dle ISO 11898-2 (CAN high speed)

- Průběžné (páteřní) vedení s co nejkratšími odbočkami za účelem omezení odrazů na vedení. **Sběrnicová „bus“ topologie, nazývaná též „linear“ topologie**.
- Max. délka páteřního vedení sběrnice je 40 m při komunikační rychlosti 1 Mbit/s.
  - Při menších rychlostech je možné vzdálenosti prodloužit.
- Max. délka odbočky z páteře k ŘJ je 30 cm při 1 Mbit/s, 3 m při 250 kbit/s.
- Max. počet uzlů na sběrnici není normou definován. Závisí na komunikační rychlosti (délka trvání jednoho bitu), délce sběrnice a kvalitě el. provedení jednotlivých uzlů. Obvykle max. malé desítky uzlů na jedné sběrnici (např. 30).

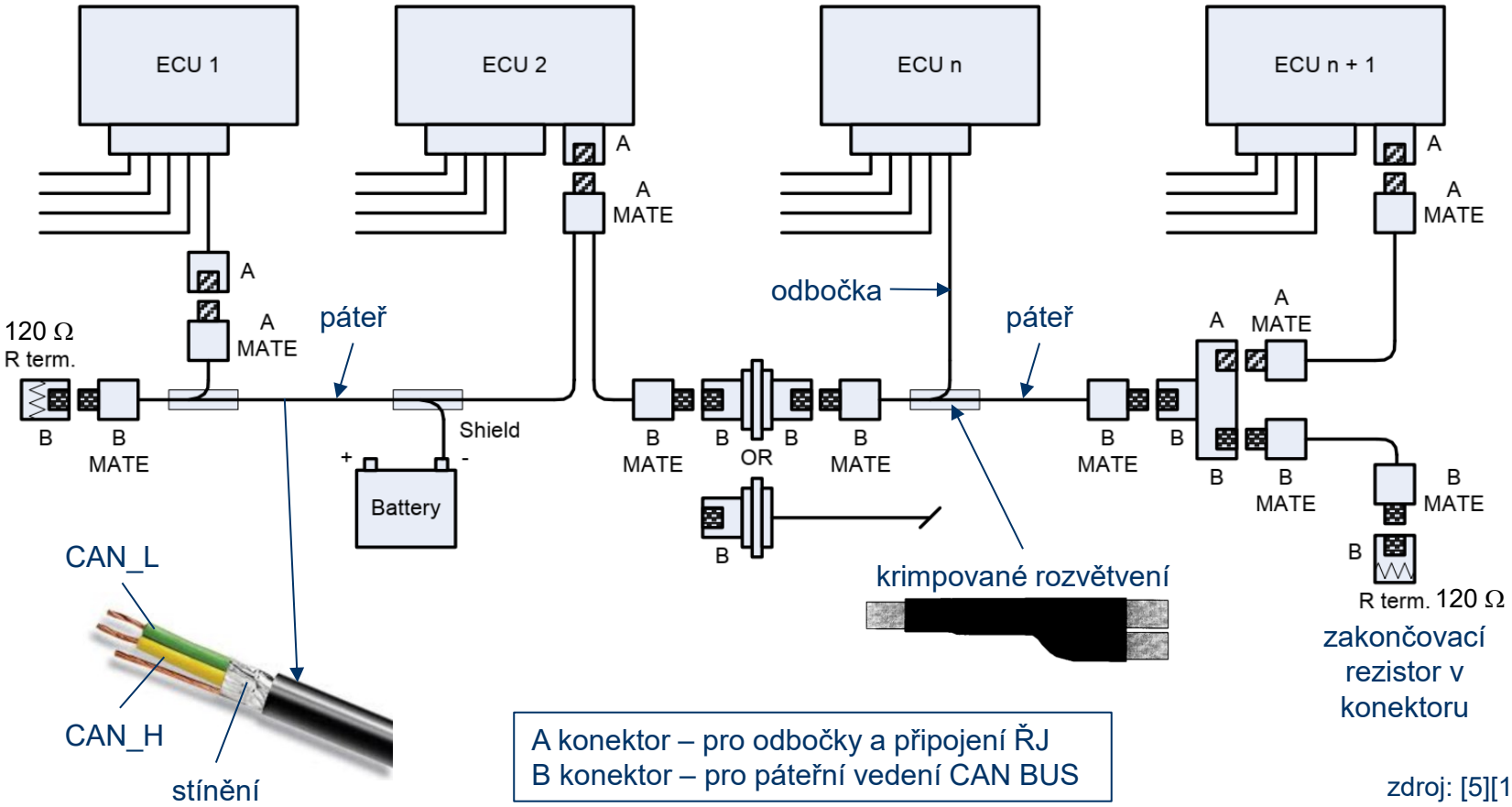


zdroj: [12]

Topologie sběrnice CAN high speed

Doporučené použití konektorů pro připojení ŘJ na CAN BUS

- Dle normy SAE J1939-11 (fyzická vrstva, 250 kbit/s, stíněná kroucená dvoulinka).
- Nákladní automobily, autobusy, traktory, pracovní stroje.



A konektor – pro odbočky a připojení ŘJ
B konektor – pro páteřní vedení CAN BUS

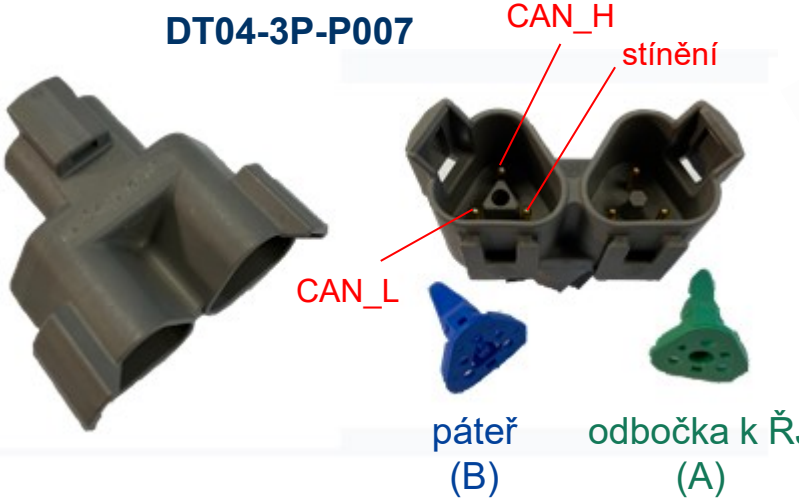
zdroj: [5][15]

Topologie sběrnice CAN high speed

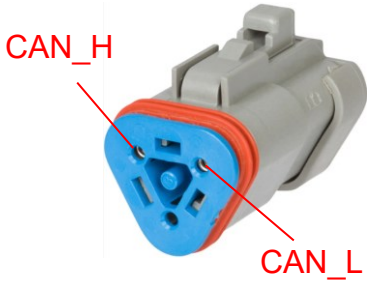
Doporučené použití konektorů pro připojení ŘJ na CAN BUS

- Dle normy SAE J1939-11 (fyzická vrstva, 250 kbit/s, stíněná kroucená dvoulinka).
- DEUTSCH 3-pinové konektory řady DT04 (samec) a DT06 (samice).

Rozbočovač
DT04-3P-P007

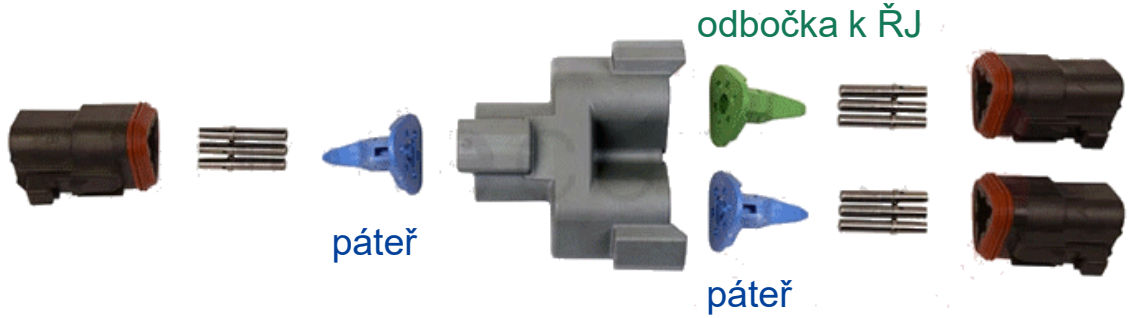


Zakončovací rezistor 120 Ω
DT06-3S-P006



Zakončovací rezistor 120 Ω
DT04-3P-P006





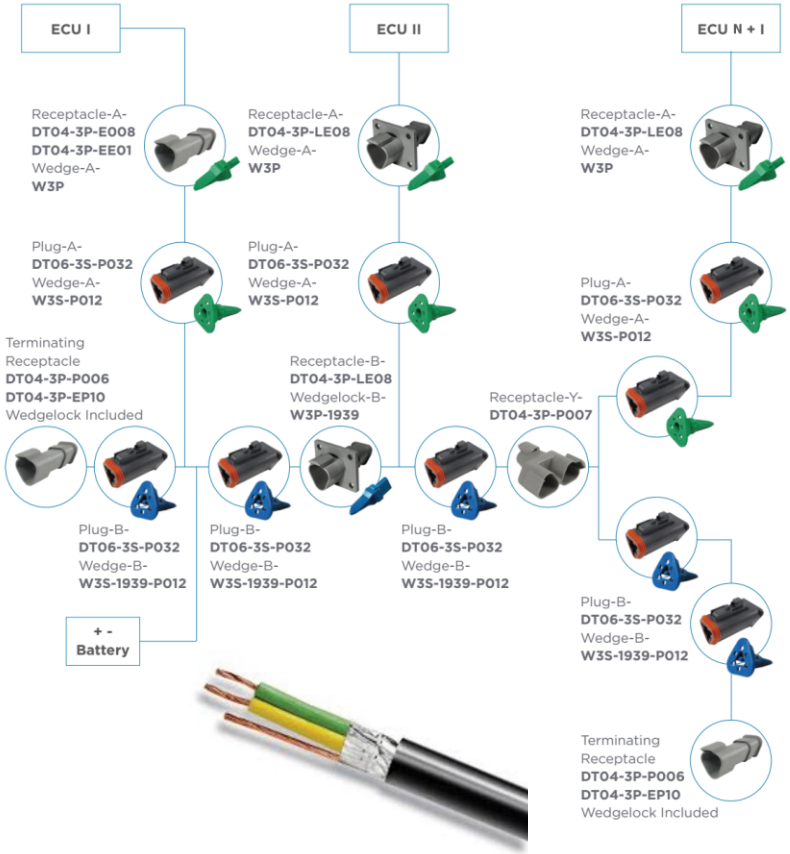
zdroj: [5]

Topologie sběrnice CAN high speed

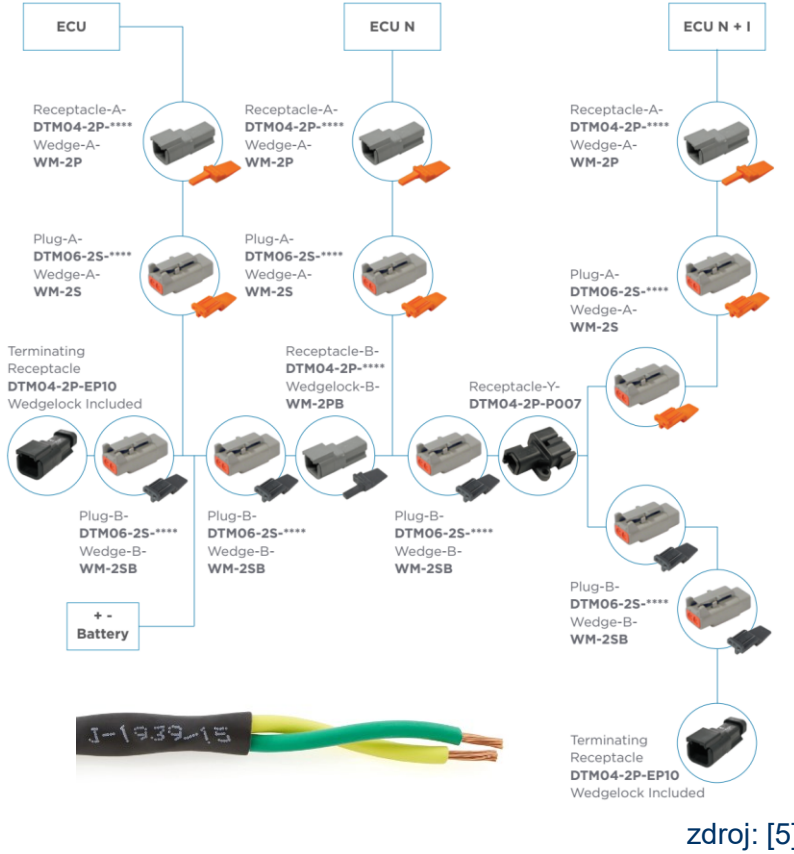
Systém konektorů pro stíněnou a nestíněnou kroucenou dvojlinku

- portfolio výrobce TE Connectivity pro kabeláž CAN dle normy SAE J1939-11 a -15

Třívodičový systém (stíněná kroucená dvoulinka)
3-pinové konektory DEUTSCH DT (SAE J1939-11)



Dvouvodičový systém (nestíněná kroucená dvoulinka)
2-pinové konektory DEUTSCH DTM (SAE J1939-15)



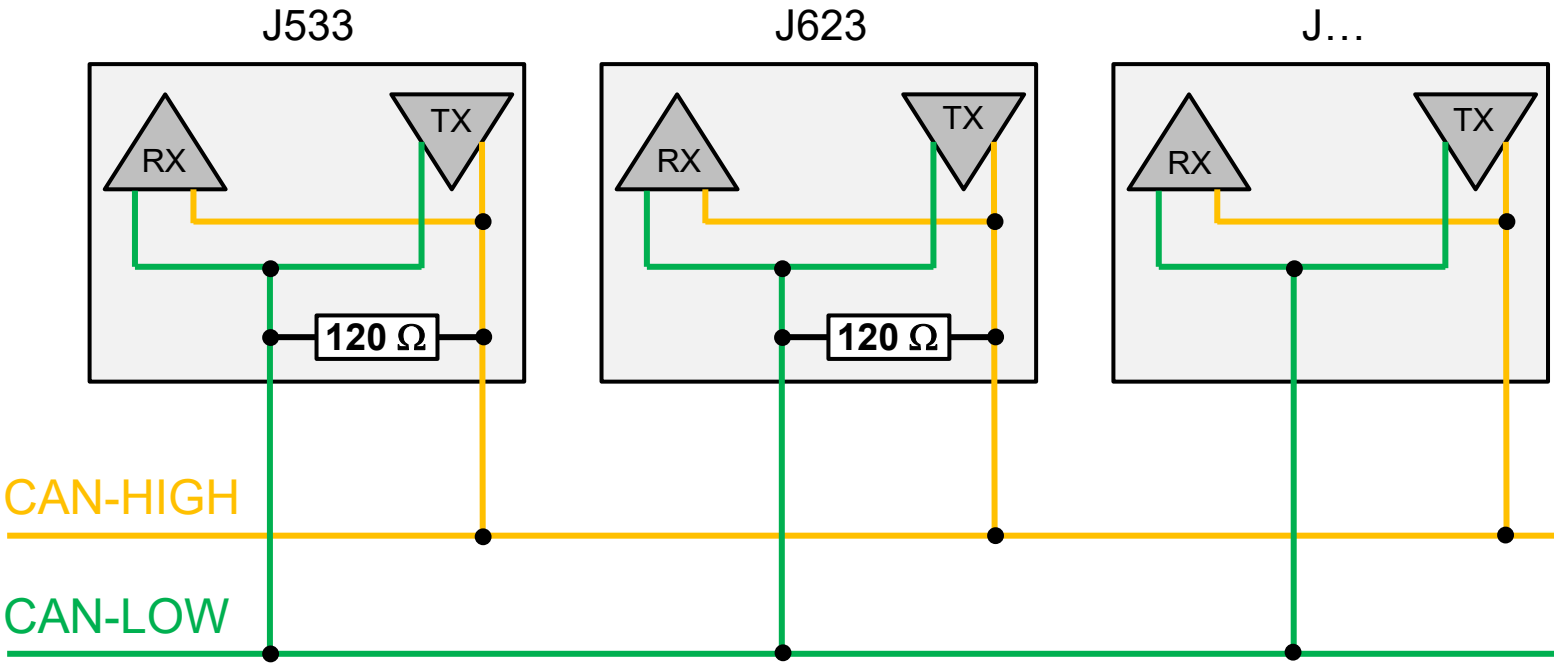
zdroj: [5]

Topologie sběrnice CAN high speed

Osobní vozidlo – koncern VW platforma MQB (Škoda Octavia III a novější)

- CAN high speed.
- Zakončovací odpory vždy v řídících jednotkách, ne externí.
- Jeden je vždy v ŘJ Gateway J533, druhý v jiné jednotce na daném CANu.
- Celkový odpor mezi vodiči CAN HIGH a CAN LOW je 60 Ohm.

Jxxx - ŘJ
RX - přijímač
TX - vysílač



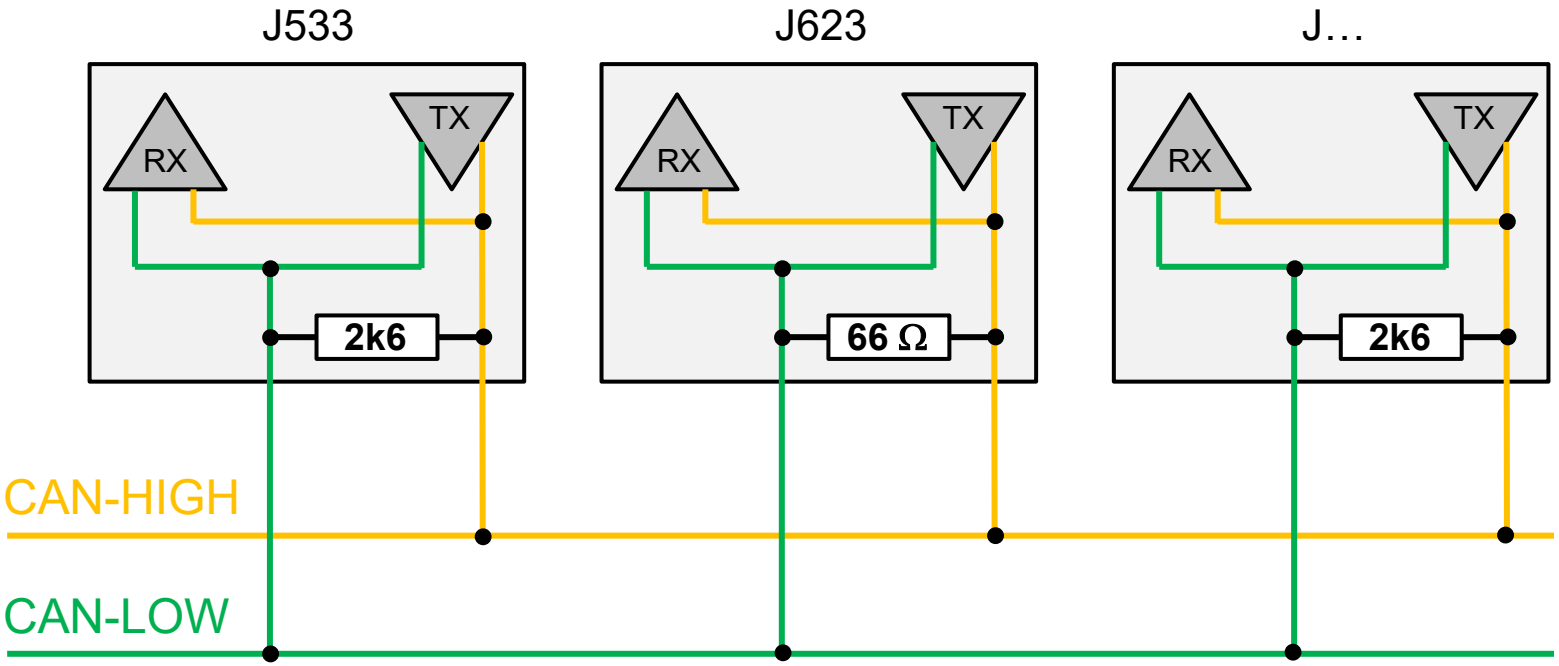
zdroj: [12]

Topologie sběrnice CAN high speed

Před zavedením platformy MQB (Škoda Octavia II a starší)

- Řídicí jednotka motoru má zakončovací odpor 66 Ohm, všechny ostatní řídicí jednotky na téže sběrnici mají každá zakončovací odpor 2,6 kOhm.
- Celkový odpor mezi CAN HIGH a CAN LOW je cca 60 Ohm.
- Týká se Škoda Fabia I, Škoda Octavia I, II.

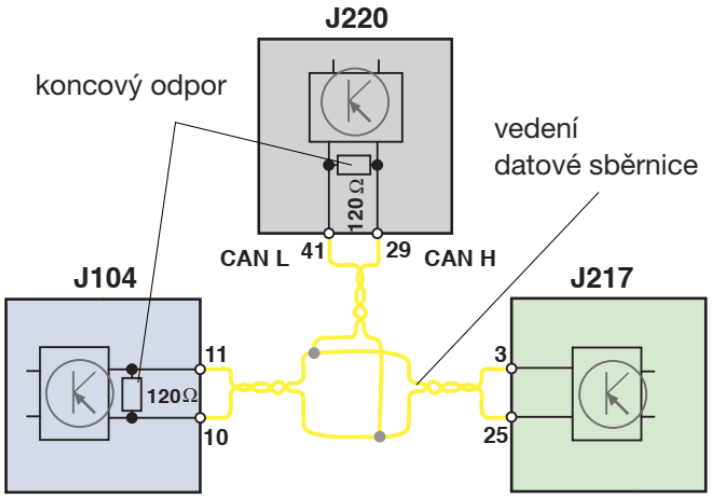
Jxxx - ŘJ
RX - přijímač
TX - vysílač



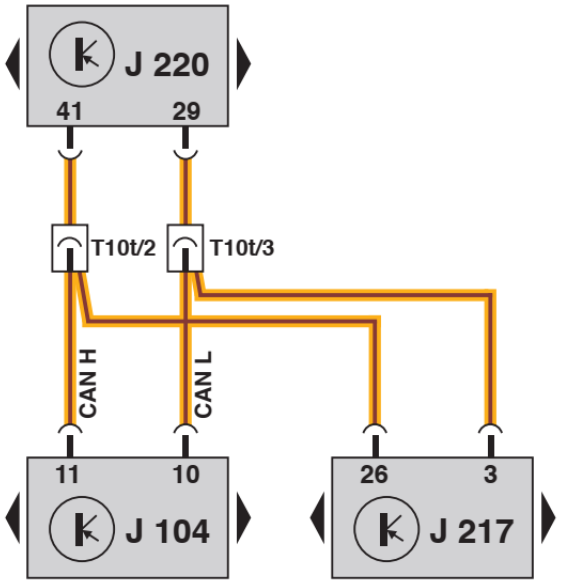
zdroj: [12]

Topologie sběrnice CAN high speed

- **Používá se i topologie „do hvězdy“ („star“).**
- Spojení vodičů CAN LOW, CAN HIGH v jednom bodě a následné rozvětvení z tohoto bodu k jednotlivým řídícím jednotkám.
- Při přerušení vodiče přestane komunikovat jen jedna ŘJ, ostatní komunikují dál.
- Je ale větší riziko nechtěných odrazů signálu na vedení a porušení bitů ve zprávě.



J104 ŘJ ABS
J217 ŘJ automatické převodovky
J220 ŘJ motoru



zdroj: [11]

Rozpoznání CAN vodičů na vozidle

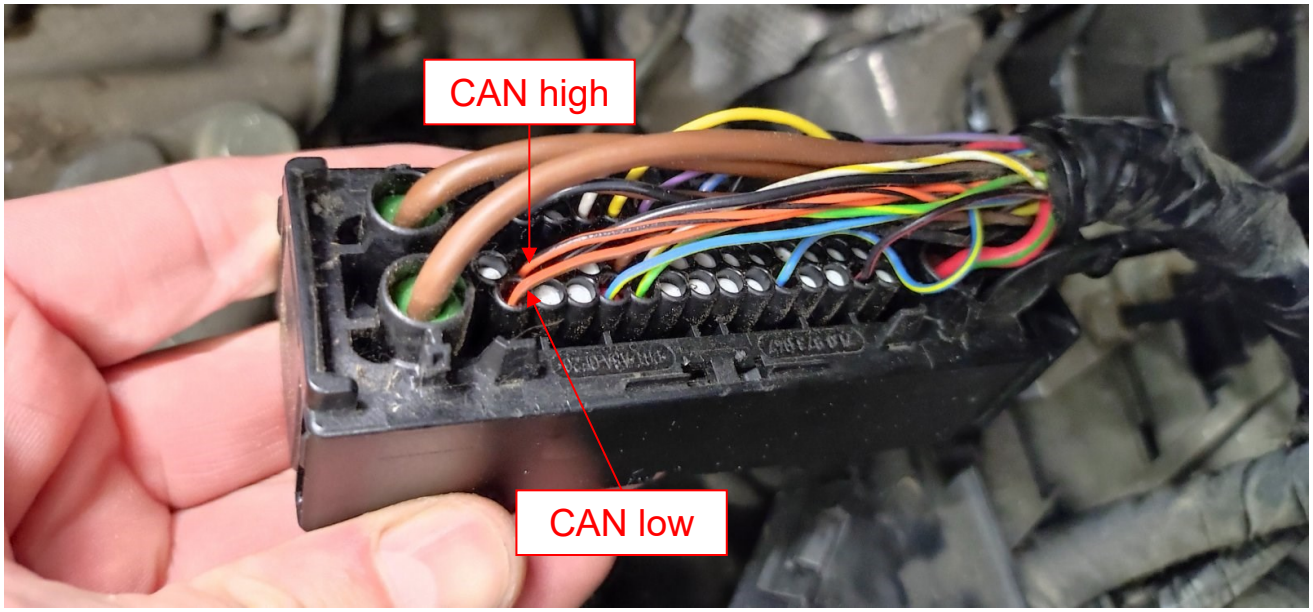
- **Barevné značení vodičů CAN v osobních vozidlech koncernu VW.**

CAN_H, pohon	oranžový/černý
CAN_H, komfort	oranžový/zelený
CAN_H, infotainment	oranžový/fialový
CAN_L, všechny	oranžový/hnědý

Pozor, uvedené barevné značení nemusí platit u novějších platforem VW MQB a MEB.

- Průřez vodičů CAN_L, CAN_H buď 0,35 mm^2 nebo 0,5 mm^2.

konektor do ŘJ ABS

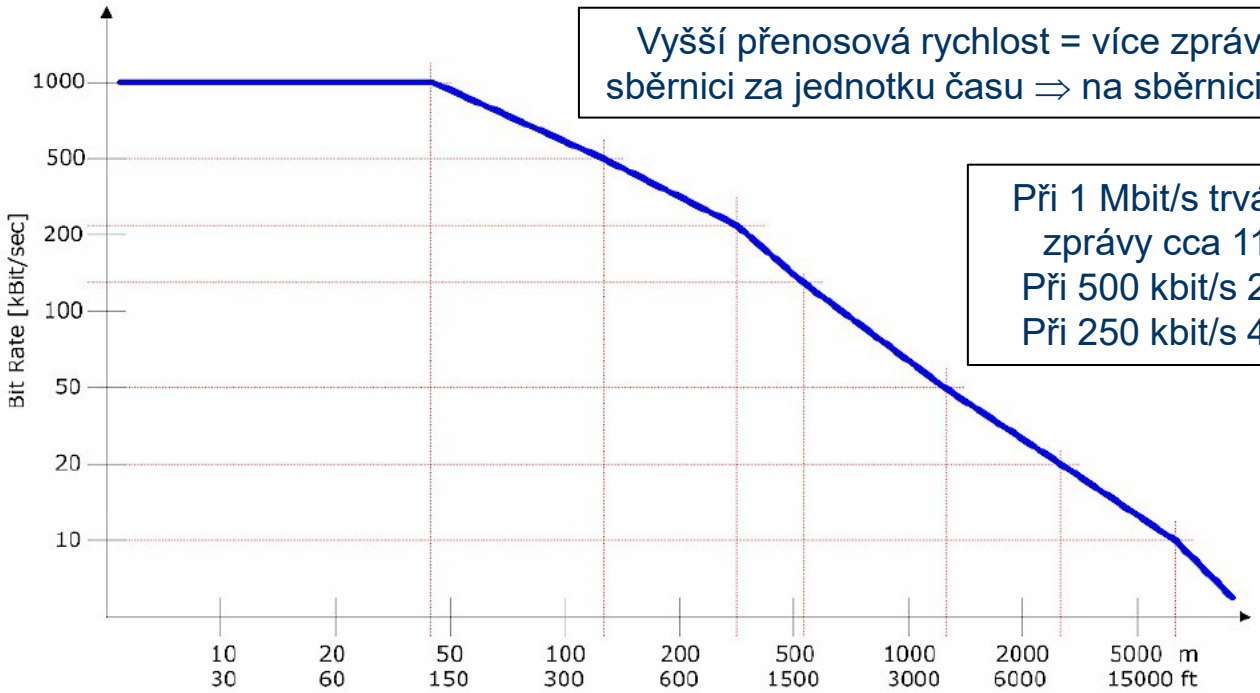




Přenosová rychlost

CAN dle ISO 11898-2 (CAN high speed)

- **Přenosová rychlost** – max.1 Mbit/s při délce vedení do 40 m (s rostoucí vzdáleností klesá)



Vyšší přenosová rychlost = více zpráv přenesených po sběrnici za jednotku času ⇒ na sběrnici může být více ŘJ.

Při 1 Mbit/s trvá odvysílání jedné zprávy cca 111 mikrosekund.
Při 500 kbit/s 222 mikrosekund.
Při 250 kbit/s 444 mikrosekund.

- **Standardní přenosové rychlosti:** 1 Mbit/s, 500 kbit/s, 250 kbit/s, 125 kbit/s, 100 kbit/s, 50 kbit/s, 20 kbit/s, 10 kbit/s



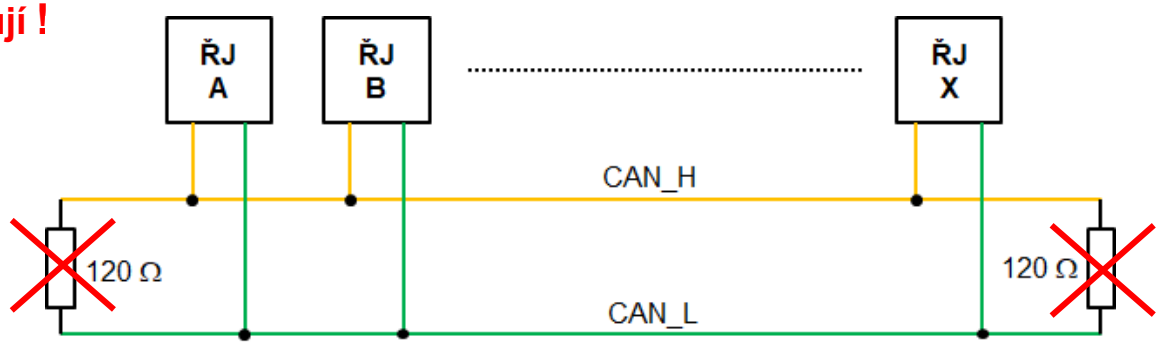
B = byte (bajt)
b = bit
1 B = 8 b (1 byte obsahuje 8 bitů)

zdroj: [2]

Topologie sběrnice CAN low speed

Topologie sběrnice CAN dle ISO 11898-3 (CAN low speed, fault tolerant)

- Použití u „pomalejších systémů“ typu osvětlení, centrální zamykání, stahování oken, klimatizace, apod.
- Max. komunikační rychlost 125 kbit/s. Max. délka sběrnice 40 m (při 125 kbit/s).
- Max. počet uzlů min. 20 při délce sběrnice 40 m a komunikační rychlosti 125 kbit/s.
- Při nižší komunikační rychlosti může být sběrnice delší a může mít více uzlů.
- Odolnost proti el. závadám na vodičích CAN_L a CAN_H (zkrat, přerušení 1 vodiče).
  - Při detekci el. závady automatický přechod na jednovodičovou komunikaci. Komunikace může pokračovat dál se sníženou odolností proti rušení (může být častější chybovost v přenosu dat).
- Není kompatibilní s CAN high speed (ISO 11898-2).
  - **CAN low speed, fault tolerant** má jiný typ budičů než **CAN high speed**.
  - Na jedné sběrnici nemohou existovat jednotky s budiči CAN high speed a CAN low speed.
- **U CAN low speed se zakončovací rezistory 120 Ohm na vedení mezi CAN_H a CAN_L neosazují !**

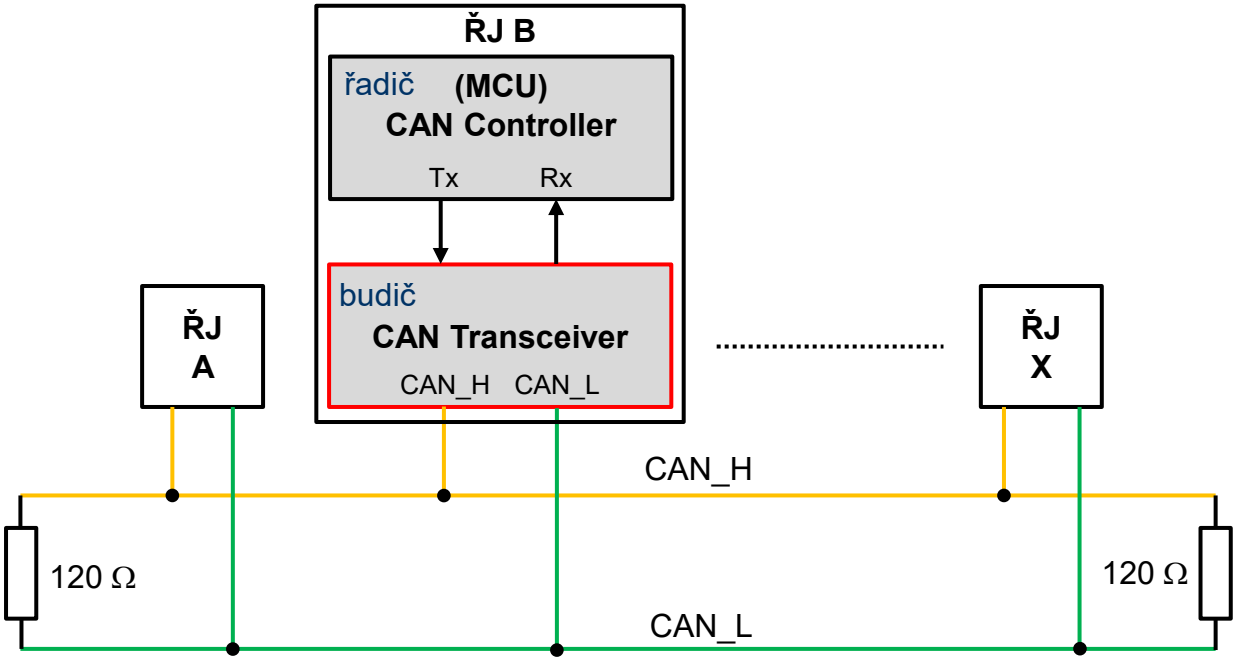


Při vypnutém napájení všech ŘJ se multimetrem naměří nekonečný odpor mezi vodiči CAN_H a CAN_L.

zdroj: [12]

Napěťové úrovně na sběrnici

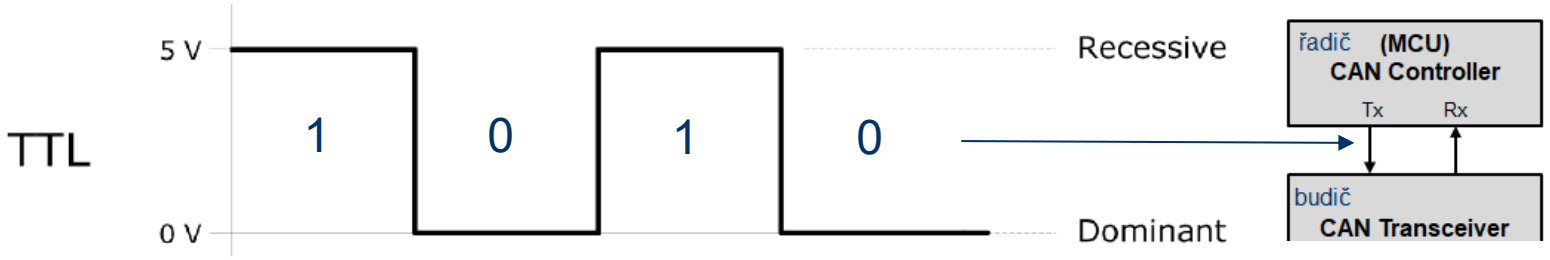
- Napěťové úrovně závisejí na typu budiče (CAN Transceiver)

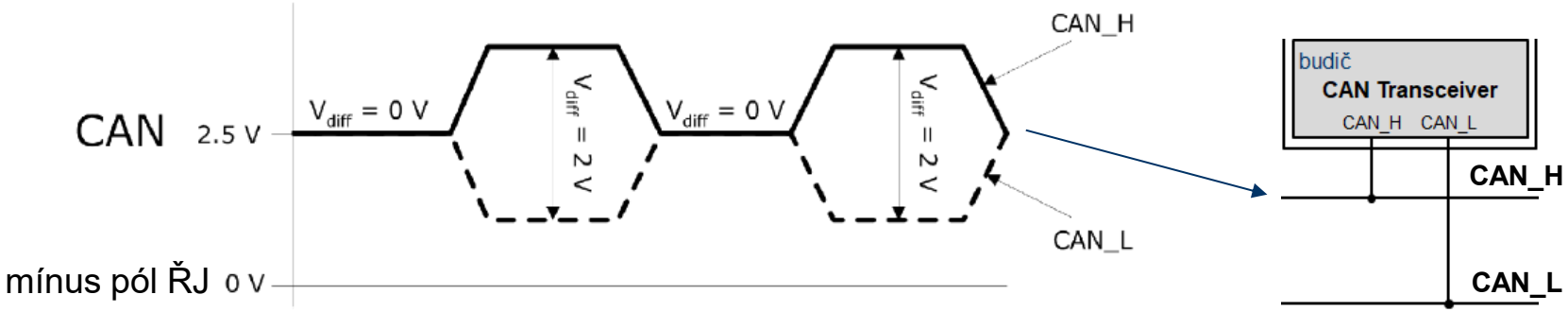


- **ISO 11898-2 CAN high speed**
- **ISO 11898-3 CAN low speed, fault tolerant**
- Oba typy budičů pracují s 12 V i 24 V palubní sítí vozidla.
- Řídící jednotky s různými typy budičů nelze připojit na jednu síť!

Napěťové úrovně na sběrnici

- **ISO 11898-2 (High Speed CAN)**
  - Rychlost do 1 Mbit/s.
  - Při el. chybě (zkrat, přerušení) na vodičích CAN_L a CAN_H nelze komunikovat.





Stav na sběrnici	U_{diff} [V]	U_{CAN_H} [V]	U_{CAN_L} [V]
DOMINANT	2,0	3,5	1,5
RECESSIVE	0,0	2,5	2,5

$$U_{diff} = U_{CAN_H} - U_{CAN_L}$$

zdroj: [2]

Napěťové úrovně na sběrnici

- **ISO 11898-2 (High Speed CAN)**
  - Napětí na vodičích CAN_H a CAN_L vůči GND a rozdílové napětí U_{diff} mají povolené tolerance.

stav DOMINANT			
	Min.	**Nom.**	**Max.**
U_{CAN_H} [V]	2,75	3,5	4,5
U_{CAN_L} [V]	0,5	1,5	2,25
U_{diff} [V]	1,5	2,0	3,0

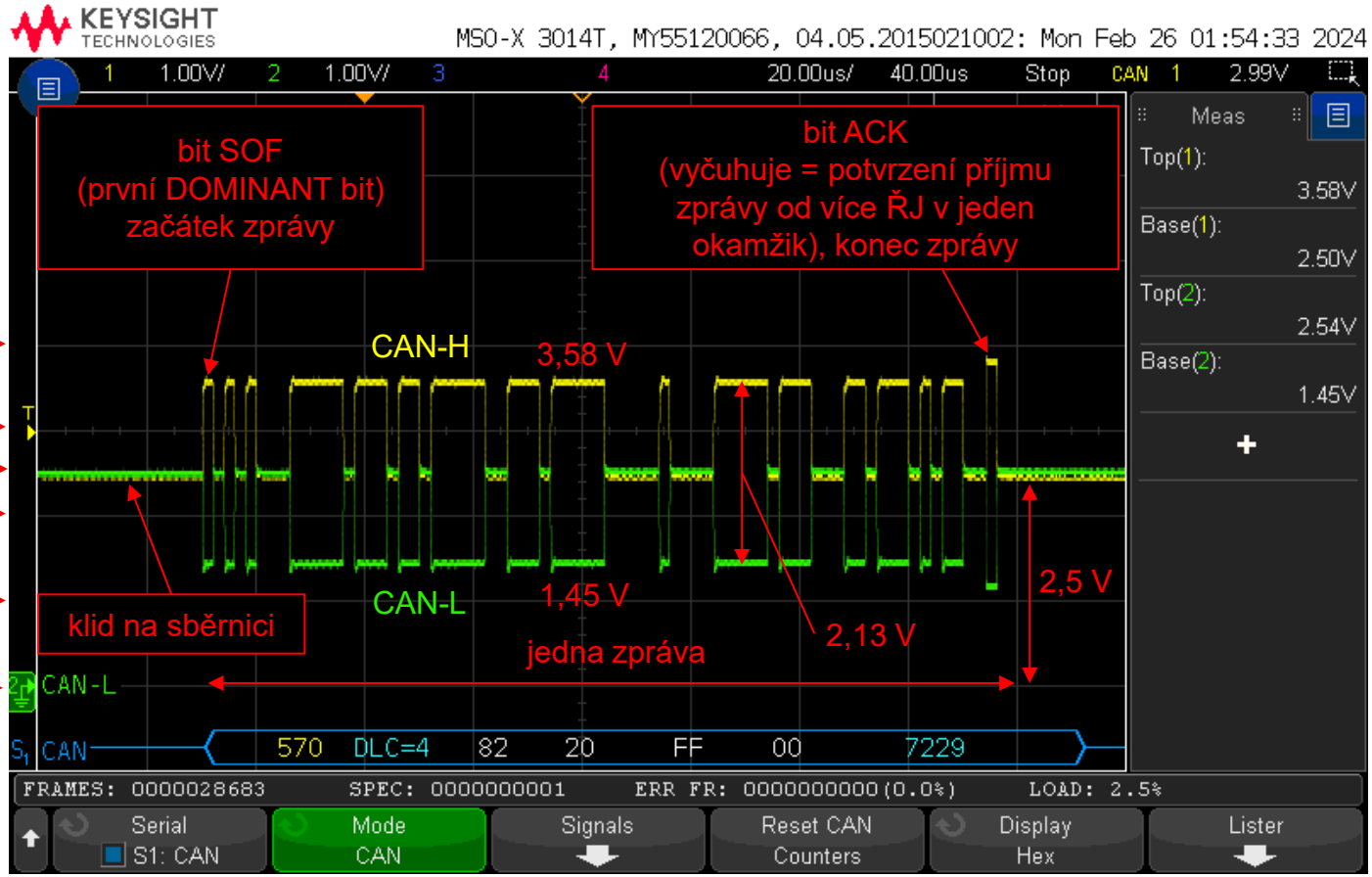
stav RECESSIVE			
	Min.	**Nom.**	**Max.**
U_{CAN_H} [V]	2,0	2,5	3,0
U_{CAN_L} [V]	2,0	2,5	3,0
U_{diff} [V]	-0,5	0,0	0,05

$$U_{diff} = U_{CAN_H} - U_{CAN_L}$$

zdroj: [12]

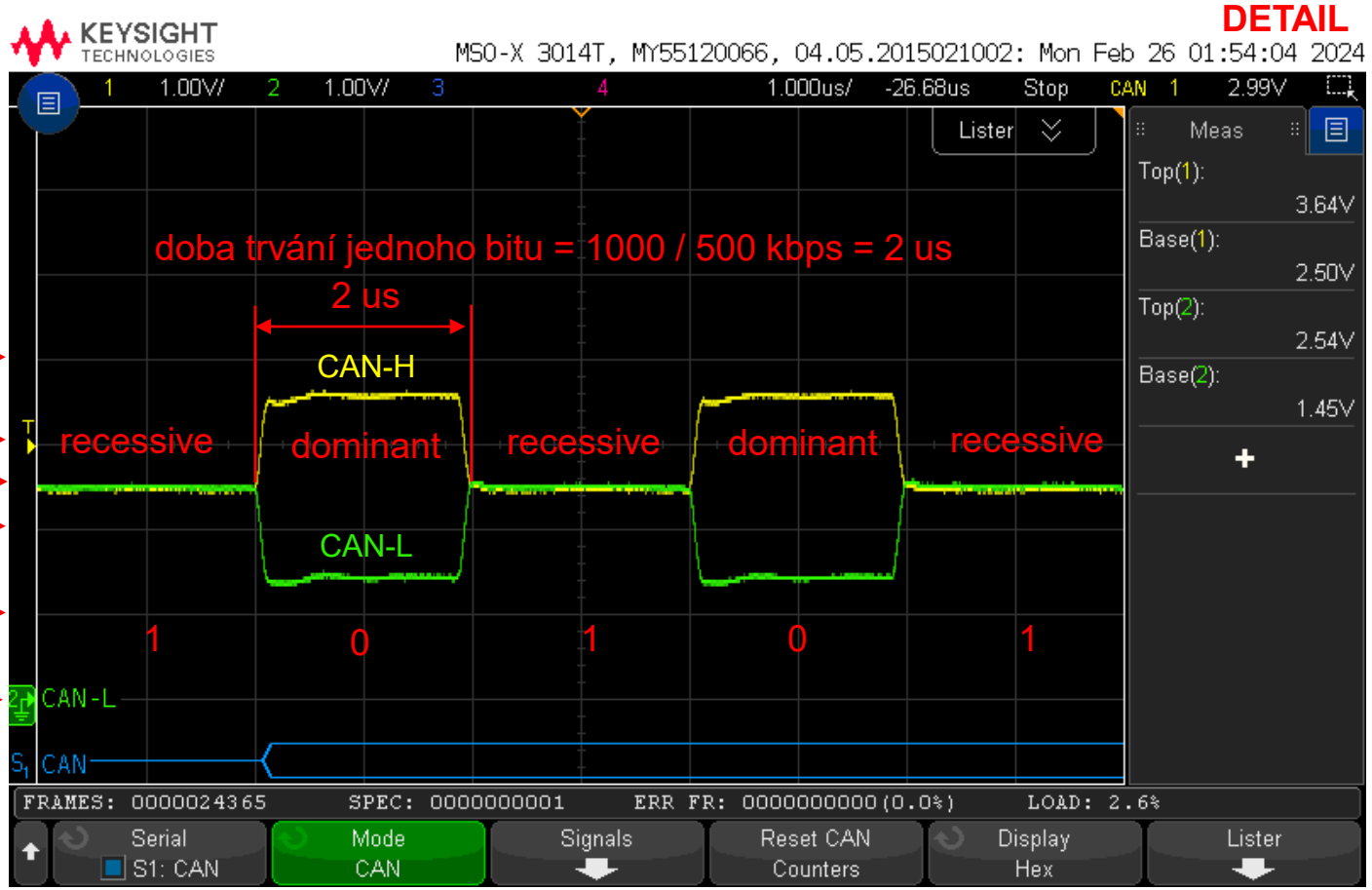
Napěťové úrovně na sběrnici

- **Oscilogram s průběhy napětí na vodičích CAN_H a CAN_L**
  - **ISO 11898-2 (High Speed CAN)**. Komunikační rychlost 500 kbit/s.
  - Sběrnice zakončená rezistory 120 Ohm.



Napěťové úrovně na sběrnici

- **Oscilogram s průběhy napětí na vodičích CAN_H a CAN_L**
  - **ISO 11898-2 (High Speed CAN)**. Komunikační rychlost 500 kbit/s.
  - Sběrnice zakončená rezistory 120 Ohm.



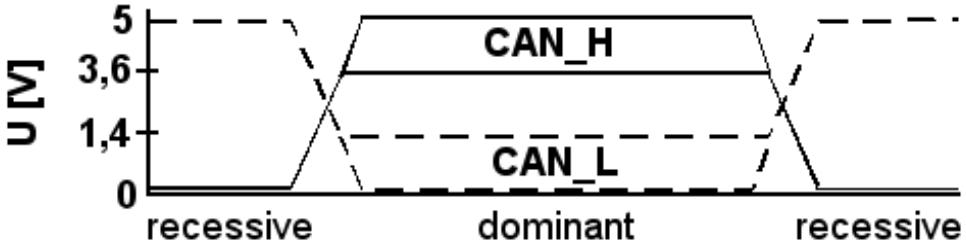
Chování sběrnice při el. chybách na vodičích CAN_H a CAN_L

- **ISO 11898-2 (High Speed CAN)**
  - **Přerušení CAN_H** – rozdělí síť na dvě poloviny, v každé polovině lze dál komunikovat mezi ŘJ, mezi polovinami sítě komunikovat nelze
  - **Přerušení CAN_L** – rozdělí síť na dvě poloviny, v každé polovině lze dál komunikovat mezi ŘJ, mezi polovinami sítě komunikovat nelze
  - **Zkrat CAN_H na BAT+** – komunikace může ale nemusí být možná (spíše ne)
  - **Zkrat CAN_H na BAT-** – komunikace není možná
  - **Zkrat CAN_L na BAT+** – komunikace není možná
  - **Zkrat CAN_L na BAT-** – komunikace je možná
  - **Spojení CAN_H s CAN_L** – komunikace není možná
  - Nepřítomnost pouze jednoho zakončovacího rezistoru 120 Ω - komunikace může ale nemusí být možná
  - Ani jeden zakončovací rezistor 120 Ohm není přítomen – komunikace není možná

Vysvětlivky:
BAT+ – plus pól baterie (12 V, 24 V)
BAT- – kostra (záporný pól baterie)

Napěťové úrovně na sběrnici

- **ISO 11898-3 (Low Speed, Fault Tolerant CAN)**
  - Komunikační rychlost do 125 kbit/s.
  - Komunikace funguje i při el. chybách na vodičích CAN_H a CAN_L
    - Zkrat na jednom vodiči na B+ nebo na kostru.
    - Zkrat vodičů CAN_L, CAN_H mezi sebou.
    - Přerušení jednoho vodiče.
    - Při výskytu el. závady na 1 vodiči přechod na jednovodičový provoz – komunikace zachována.
  - Jiné napěťové úrovně než budiče CAN High Speed, na jedné sběrnici nelze mít současně uzly s budiči Low Speed a High Speed !

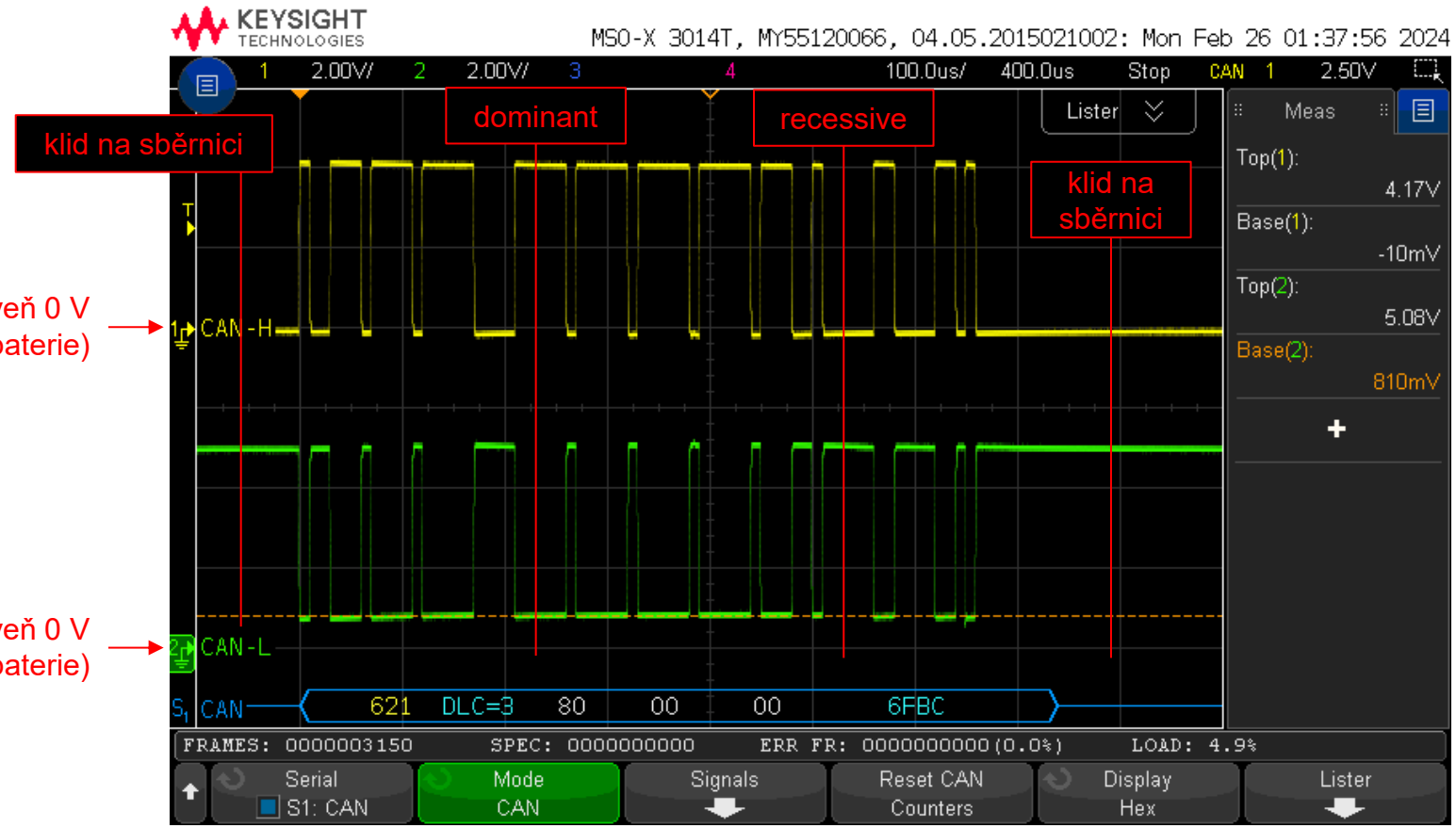


Stav na sběrnici	U_{diff} [V]	U_{CAN_H} [V]	U_{CAN_L} [V]
DOMINANT	min. 2,2 max. 5,0	min. 3,6	max. 1,4
RECESSIVE	min. -5,0 max. -4,4	max. 0,3	min. 4,7

$$U_{diff} = U_{CAN_H} - U_{CAN_L}$$

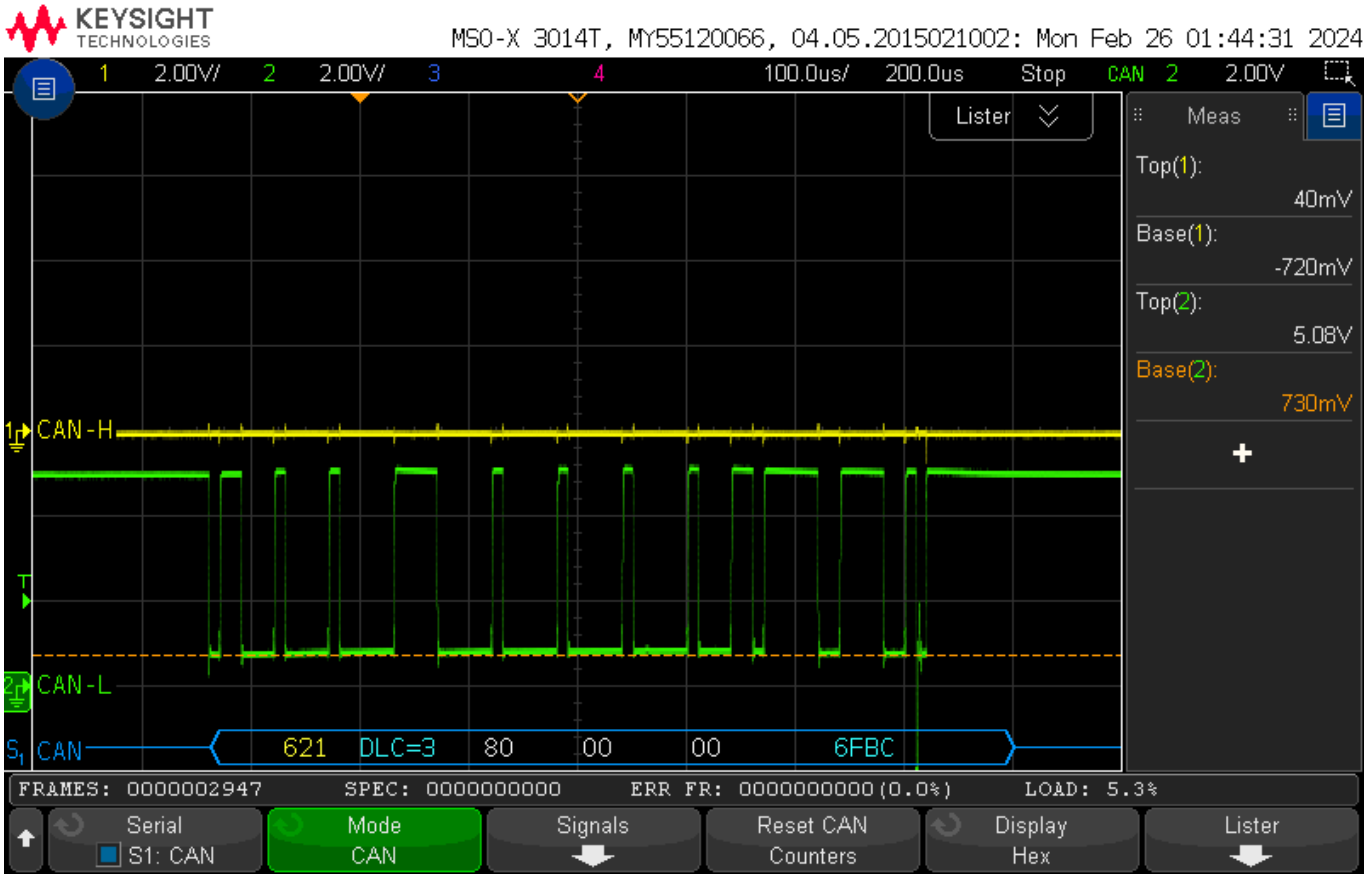
Napěťové úrovně na sběrnici

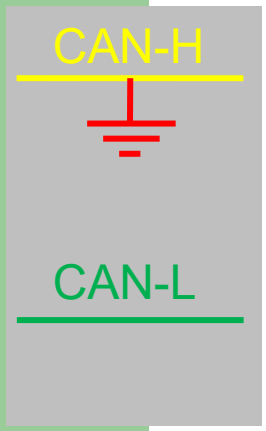
- **Oscilogram s průběhy napětí na vodičích CAN_H a CAN_L**
  - **ISO 11898-3 (Low Speed, Fault Tolerant)**. Komunikační rychlost 100 kbit/s.
  - Normální provoz po dvou vodičích (žádné el. chyby na vodičích).



Napěťové úrovně na sběrnici

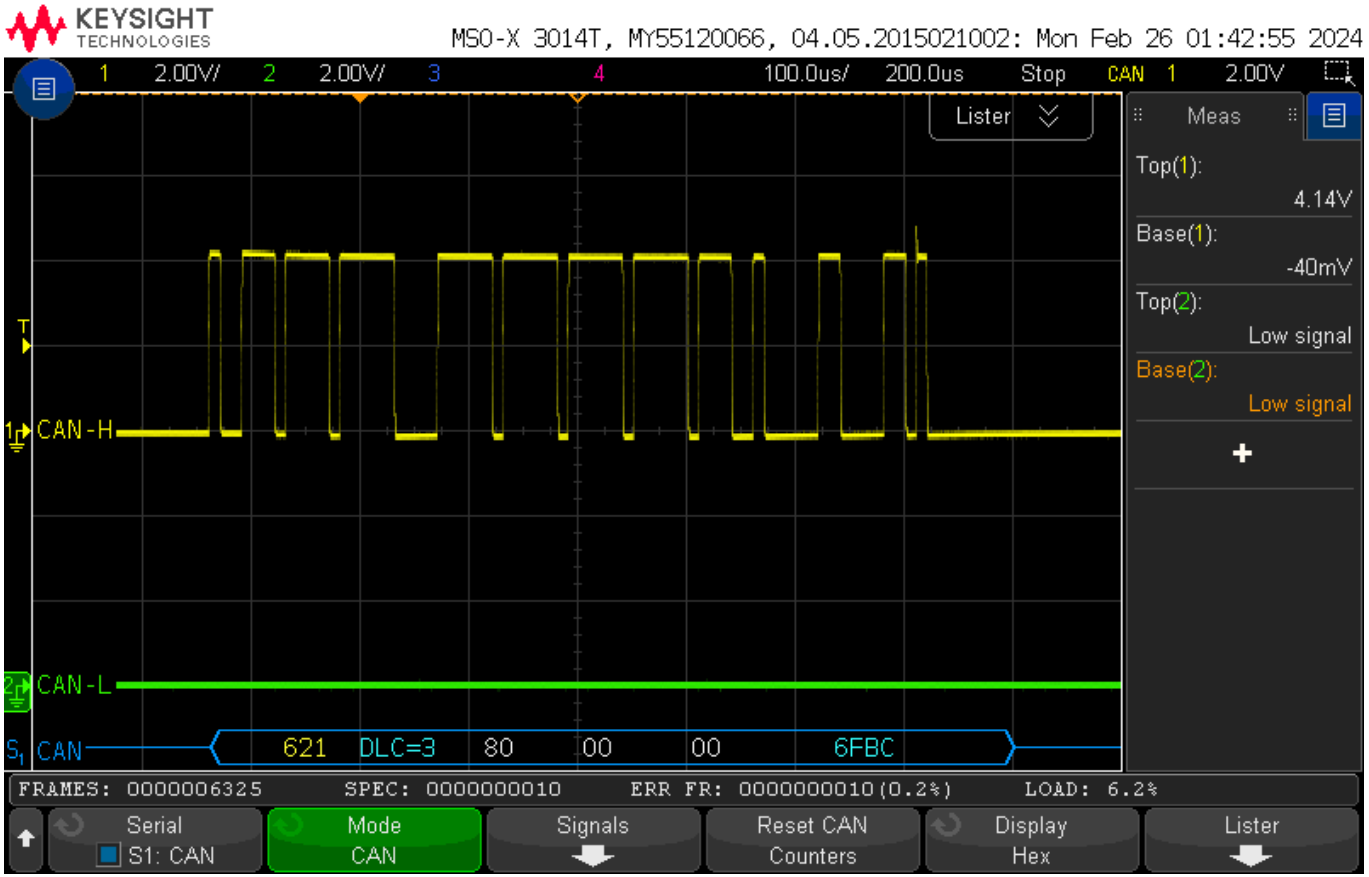
- **Oscilogram s průběhy napětí na vodičích CAN_H a CAN_L**
  - **ISO 11898-3 (Low Speed, Fault Tolerant)**. Komunikační rychlost 100 kbit/s.
  - **Zkrat CAN_H na kostru**. Přechod na jednovodičový provoz (komunikuje dál jen přes vodič CAN_L).

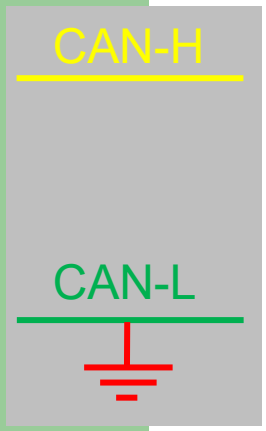




Napěťové úrovně na sběrnici

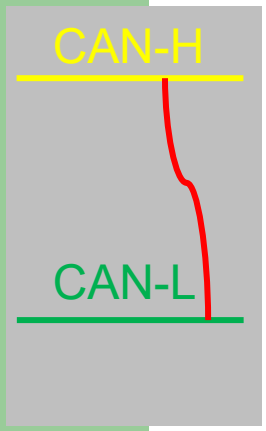
- **Oscilogram s průběhy napětí na vodičích CAN_H a CAN_L**
  - **ISO 11898-3 (Low Speed, Fault Tolerant)**. Komunikační rychlost 100 kbit/s.
  - **Zkrat CAN_L na kostru**. Přechod na jednovodičový provoz (komunikuje dál jen přes vodič CAN_H).

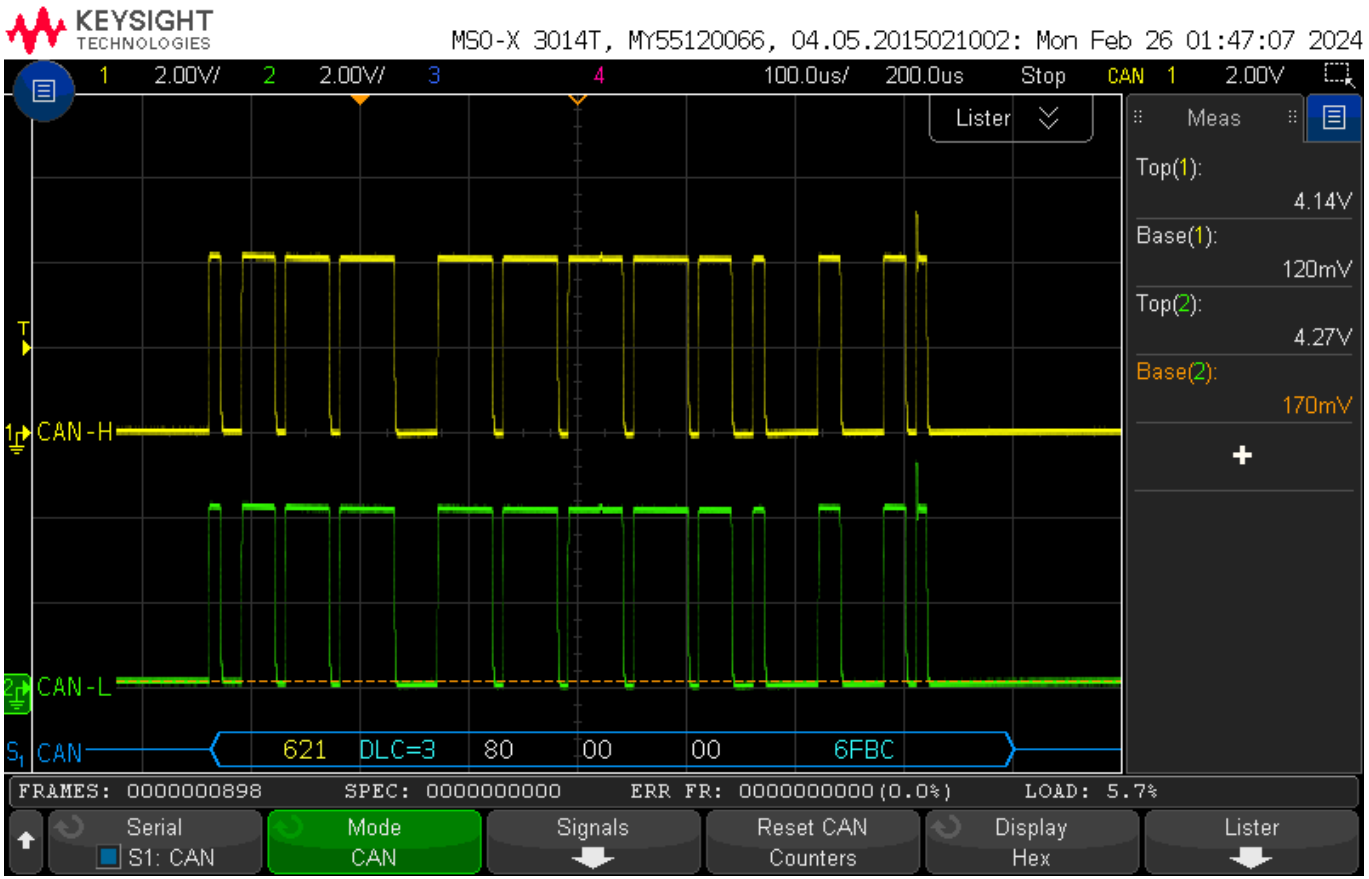




Napěťové úrovně na sběrnici

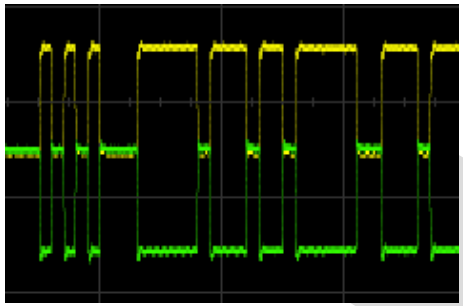
- **Oscilogram s průběhy napětí na vodičích CAN_H a CAN_L**
  - **ISO 11898-3 (Low Speed, Fault Tolerant)**. Komunikační rychlost 100 kbit/s.
  - **Zkrat CAN_H a CAN_L mezi sebou**. Přechod na jednovodičový provoz.

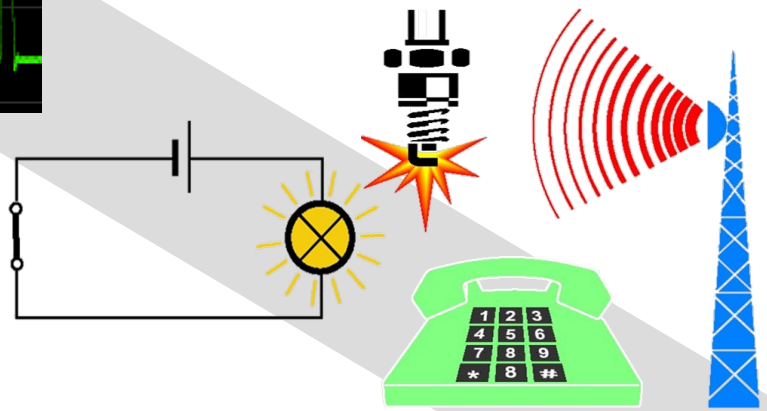


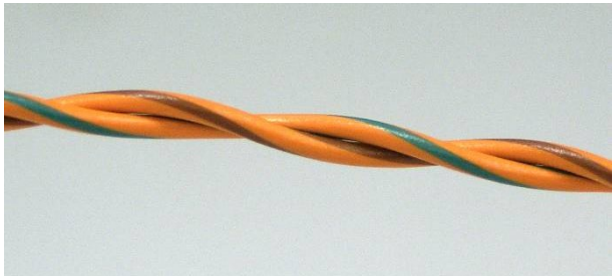


Ochrana před rušením

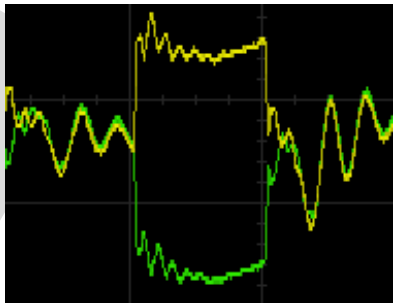
signály bez rušení







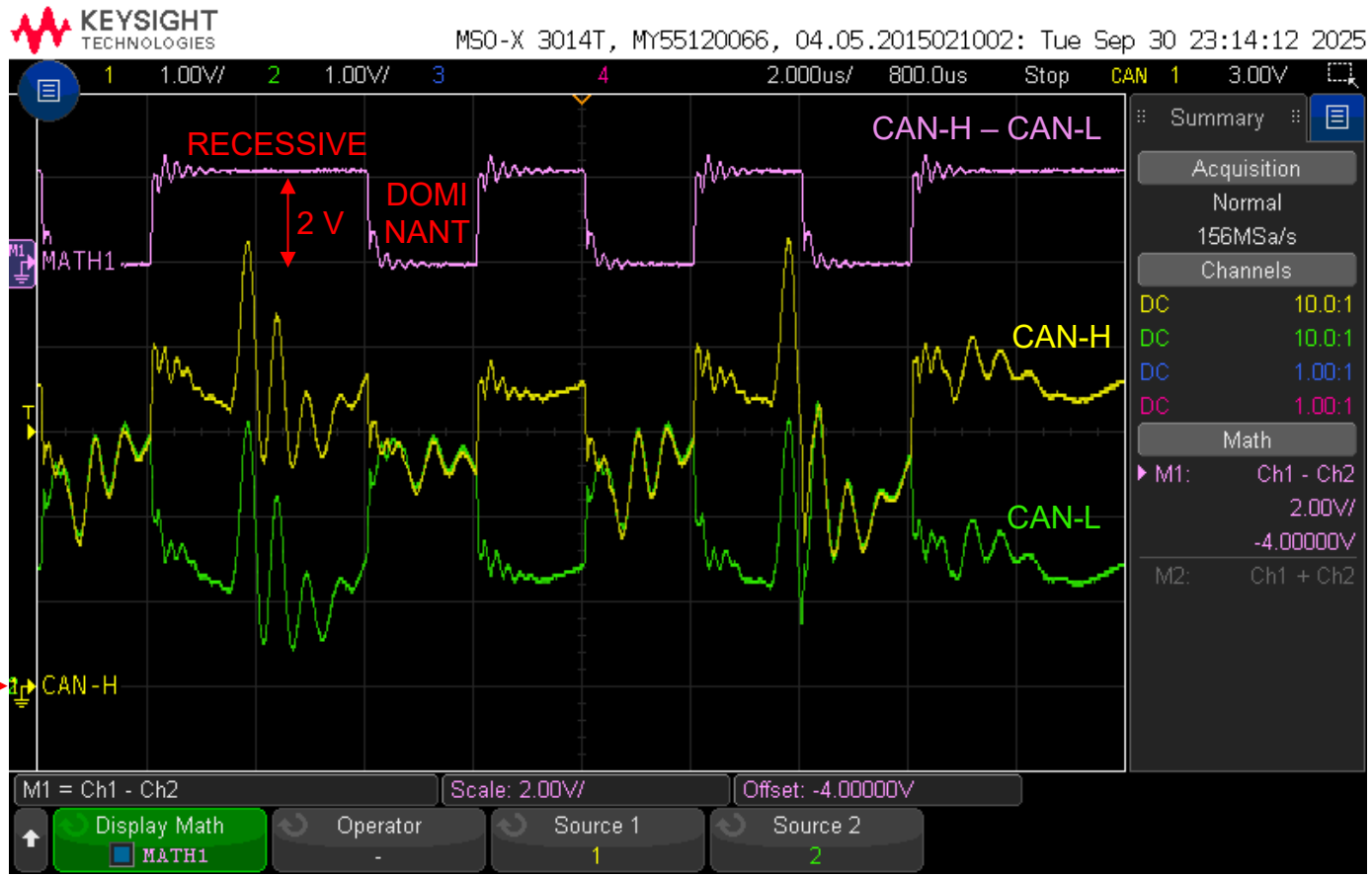
signály s rušením



zdroj: [14]

Ochrana před rušením

- Rušení působící na oba dva CAN vodiče stejnou měrou nepoškodí data přenášená na sběrnici.
- Rozdíl napětí mezi vodiči CAN-H a CAN-L není ovlivněn rušením.



úroveň 0 V
(- pól baterie)

Druhy el. závad CAN sběrnice

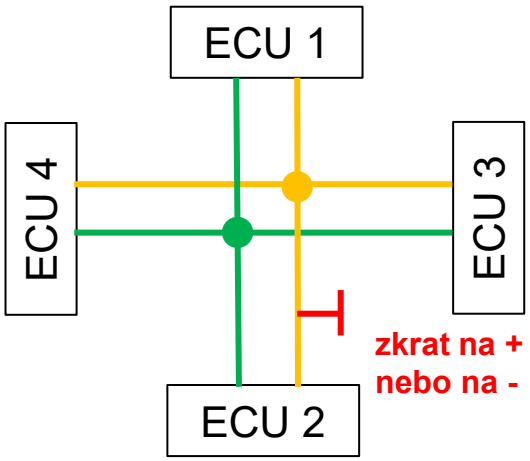
Globální závada

- Zkrat jednoho komunikačního vodiče na mínus nebo na plus pól baterie.
- U CAN high speed může vyřadit komunikaci na celé sběrnici (dle druhu zkratu a na jakém vodiči se vyskytl).
- U CAN low speed fault tolerant komunikace na celé síti funguje dál po druhém vodiči.

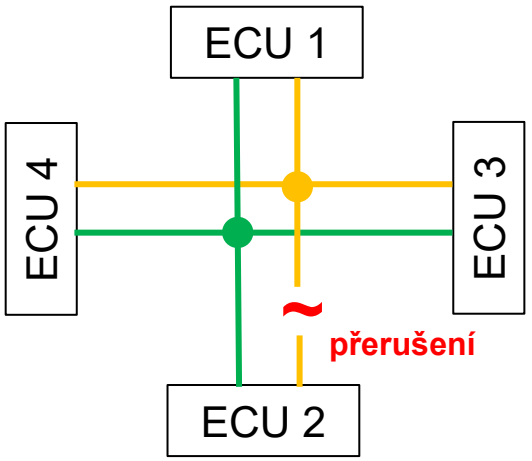
Lokální závada

- Přerušení jednoho komunikačního vodiče.
- Omezuje jen jednu ŘJ, u které se přerušení nachází.
- U CAN high speed zcela vyřadí komunikaci u dané ŘJ, zbytek sítě funguje dál.
- U CAN low speed fault tolerant komunikace u dané ŘJ funguje dál po druhém vodiči.

globální závada



lokální závada



zdroj: [14]

Sleep režim a klidový proud

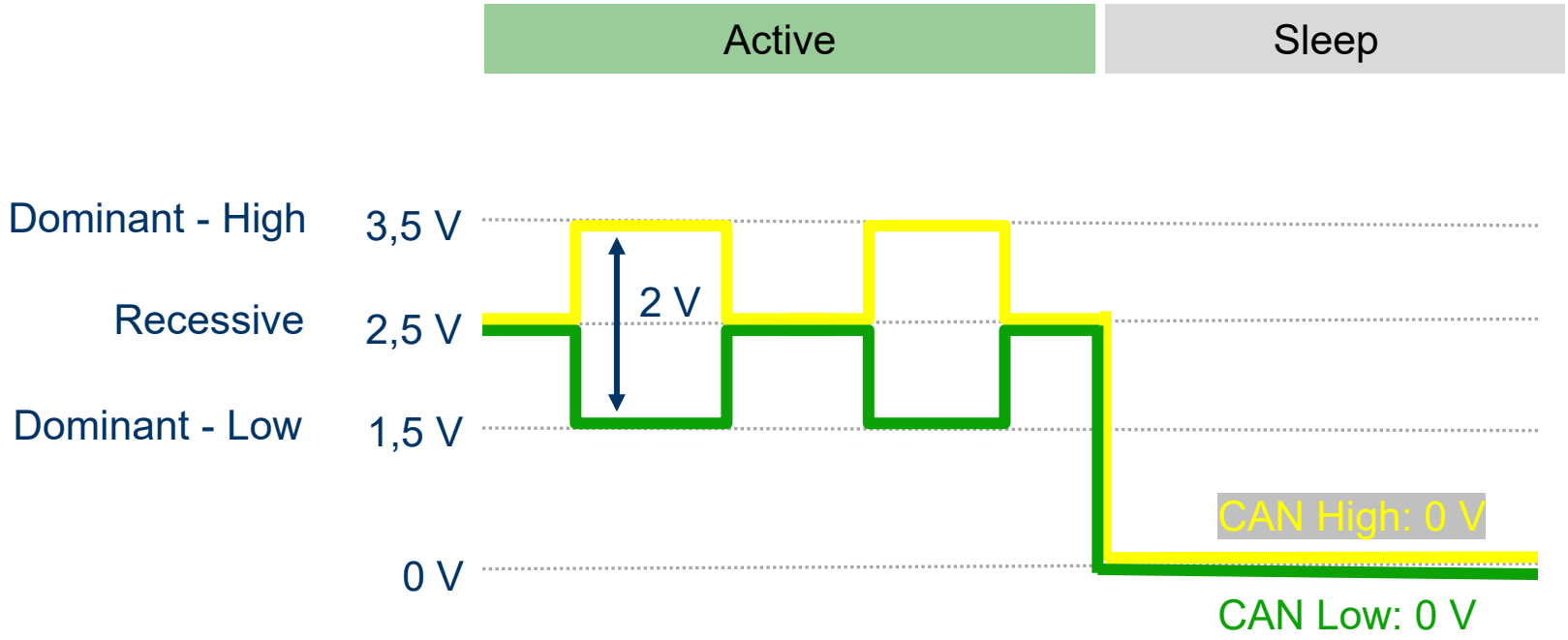
- Po opuštění vozidla celé vozidlo do určité doby (20 min až 2 h) „usne“ – všechny ŘJ přejdou do režimu se sníženou spotřebou proudu („sleep režim“).
  - Většina ŘJ je trvale napájena ze svorky 30 (+ pól baterie), aby mohly provádět některé své funkce i po opuštění vozidla a aby mohly reagovat na události typu odemknutí vozidla, otevření dveří, zasunutí klíčku, zapnutí světel bez klíčku apod.
- Proud odebíraný z 12 V baterie do vozidla klesne ve sleep režimu pod 50 mA.
- Ve sleep režimu neprobíhá komunikace mezi ŘJ – na sběrnicích je klid.
  - Není komunikace na sběrnici = ŘJ přešly do sleep režimu.
  - U osobních vozidel koncernu VW řídí usínání vozidla ŘJ Gateway.
- ŘJ se ze sleep režimu probudí (wake-up) při aktivitě na sběrnici (jiná ŘJ odešle zprávu a probudí ostatní ŘJ na sběrnici).

- Možné problémy
  - Některá ŘJ nepřejde do sleep režimu a neustále na sběrnici vysílá.
  - Díky tomu ostatní ŘJ drží „probuzené“.
  - Je zvýšený klidový odběr proudu z 12 V baterie – hrozí její vybití pokud se vozidlo několik dní nepoužívá.

- Měřením napětí na vodičích sběrnice běžným multimetrem lze zjistit, zda na sběrnici probíhá komunikace (jednotky nespí) nebo ne (jednotky spí).

Napětí na CAN-H, CAN-L ve sleep režimu

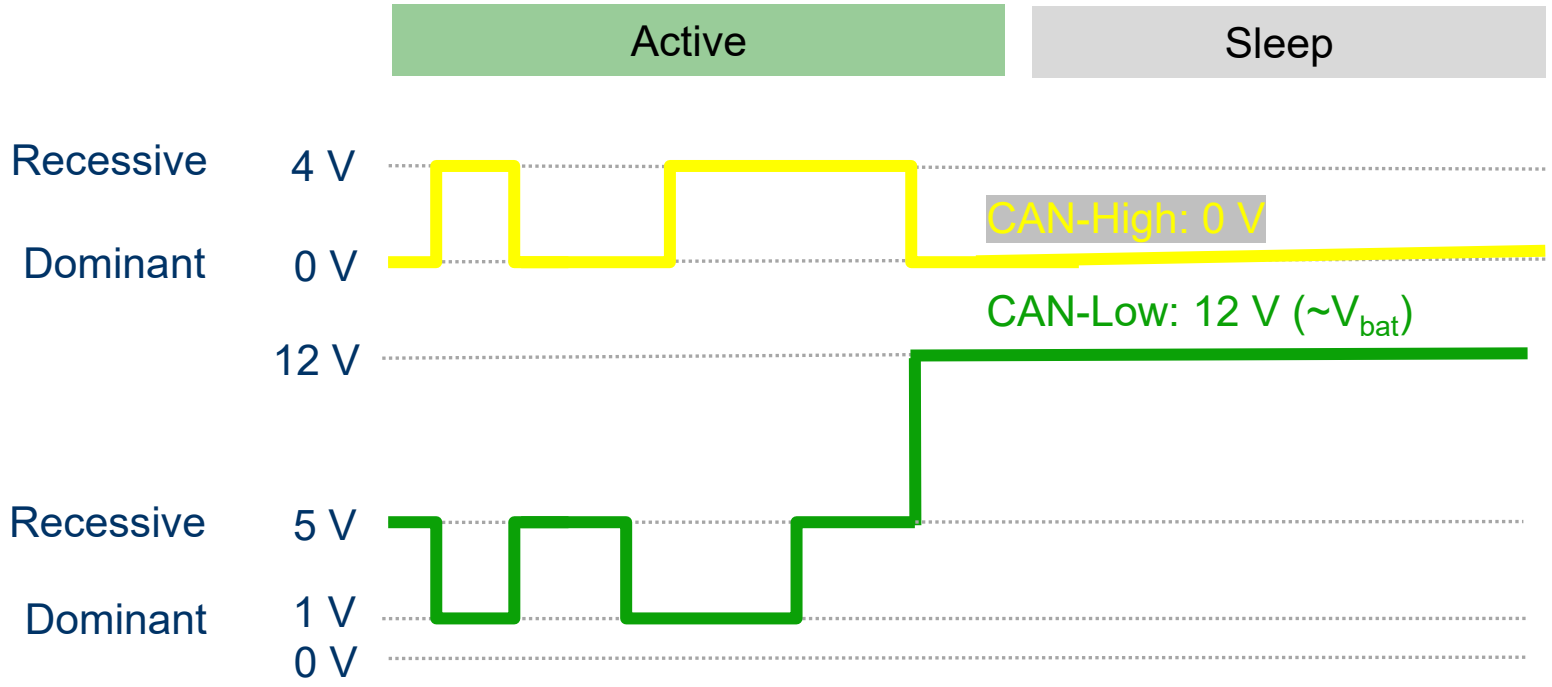
- Platí pro fyzickou vrstvu CAN High Speed a CAN FD.



zdroj: [14]

Napětí na CAN-H, CAN-L ve sleep režimu

- Platí pro fyzickou vrstvu CAN Low Speed, Fault Tolerant.



zdroj: [14]

KONEC

Děkuji za pozornost

Literatura

[1] Voss, W. *A Comprehensive Guide To Controller Area Network*. Copperhill Media Corporation, pp. 164, 2005, ISBN 0976511606.

[2] Voss, W. *Controller Area Network*. ESD electronics, Inc. Power Point prezentace.

[3] Hajný, A. *Aplikační protokoly CAN pro dieselelektrické lokomotivy*. UniControls, Power Point prezentace.

[4] Loka, P. *Moderní datové sběrnice používané ve vozidlech*. Univerzita Pardubice, bakalářská práce, 2013.

[5] www.google.com

[6] www.can-cia.org

[7] Mohamed AbdElHay. *Automotive embedded systems.* Power Point prezentace.

[8] Škoda. Dílenská učební pomůcka č. 98, 2012.

[9] Škoda. Dílenská učební pomůcka č. 33.

[10] Škoda. Dílenská učební pomůcka č. 17.

[11] Škoda. Dílenská učební pomůcka č. 24.

[12] ISO 11898-2:2003.

[13] ISO 11898-3:2006.

[14] Škoda Auto. Školení Datové sběrnice. 2024.

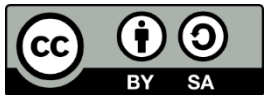
[15] SAE J1939-11:2016.

[16] Lennartsson, K. *Comparing CAN FD with Classical CAN*. Kvaser, 2016.

UNIVERZITA PARDUBICE
DOPRAVNÍ FAKULTA JANA PERNERA

Vytvořeno v rámci projektu **Green Deal UPCE**, reg. č. NPO_UPCE_MSMT-2142/2024-4.

Toto dílo podléhá licenci Creative Commons BY-SA 4.0. Pro zobrazení licenčních podmínek navštivte https://creativecommons.org/licenses/by-sa/4.0/.



NÁRODNÍ PLÁN OBNOVY

Financováno Evropskou unií
NextGenerationEU

Univerzita Pardubice
Dopravní fakulta Jana Pernera

Kurz celoživotního vzdělávání

Elektrotechnické systémy pro environmentálně příznivou dopravu

Předmět

Pokročilé systémy digitálního řízení a komunikace v dopravní technice

Studijní materiál

Komunikační protokol CANopen

Ing. Zdeněk Mašek, Ph.D.



Obsah přednášky

1. Základní vlastnosti a použití
2. Komunikační model
3. Slovník objektů
4. SDO a PDO
5. Síťový management - NMT služby
6. Konfigurace zařízení

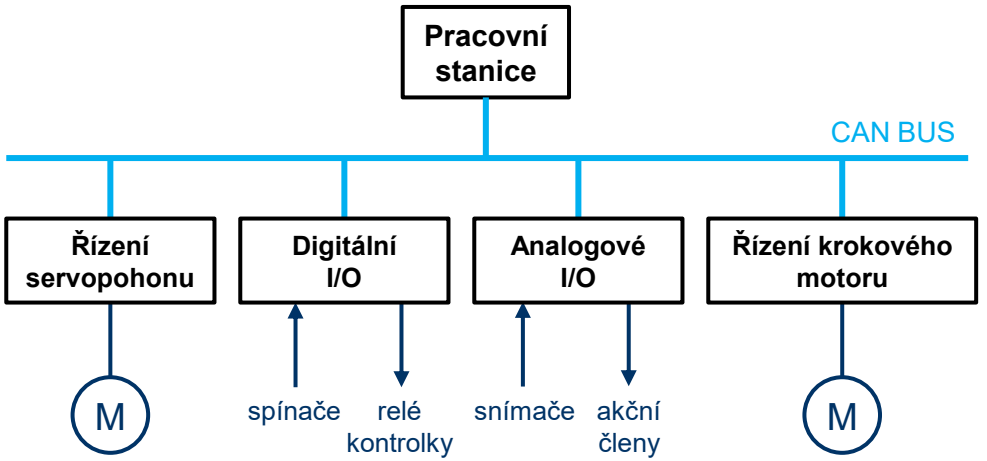
Počet snímků: 71

1. Základní vlastnosti a použití

Co je CANopen ?

- Protokol na vyšších vrstvách založený na sběrnici CAN.
- Původně vyvinut pro řízení různých strojů a pohonů.
- Sjednocení komunikace – všechna CANopen zařízení „hovoří stejným jazykem“.
- Cíl – nezávislost na výrobci zařízení, zařízení jsou zaměnitelná.
- Různé typy zařízení – digitální I/O, analogové I/O, sensory, akční členy, servopohony, PLC, ….
- Komunikace typu Master/Slave, dovoluje i přímou komunikaci mezi Slave.

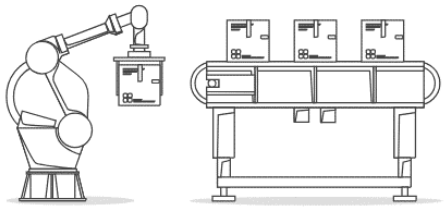


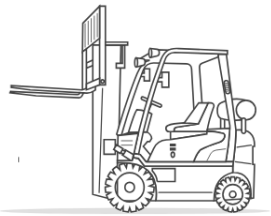


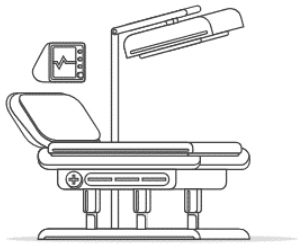
zdroj: [5]

Použití CANopen

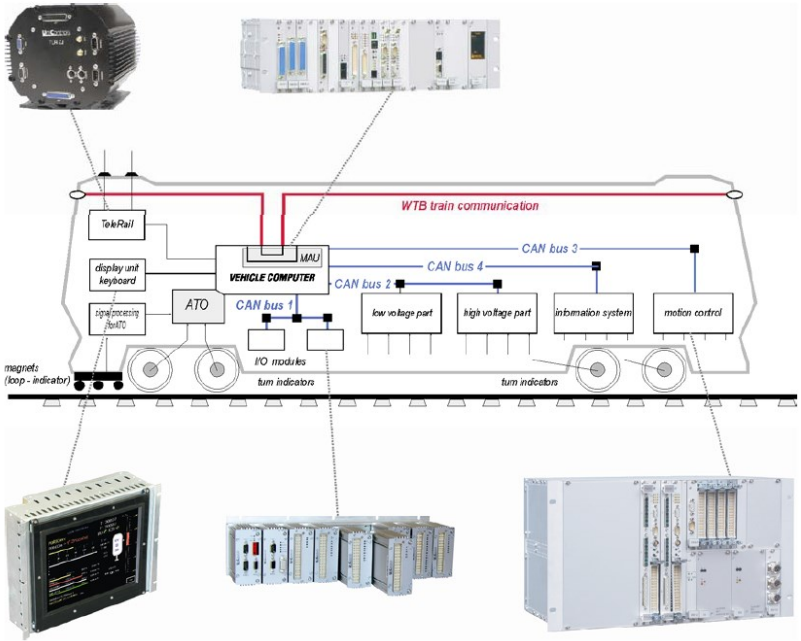
- Průmysl – robotika, manipulátory, dopravníky (servopohony).
- Pracovní stroje, v zemědělství, lesnictví, v dolech.
- Železnice – komunikace mezi systémy v rámci vozidla.
- Letecký a kosmický průmysl.
- Technologie budov (výtahy, osvětlení, dveřní systémy, snímače, …)
- Energetika – rozvodny, větrné elektrárny.
- Zdravotnictví (rentgen, CT, dialýza…).

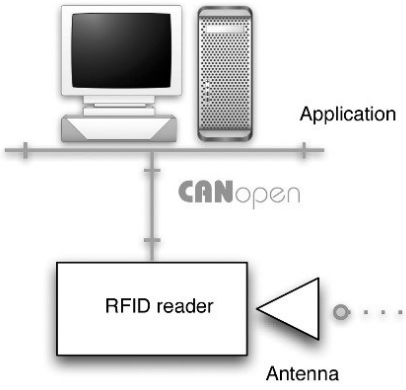










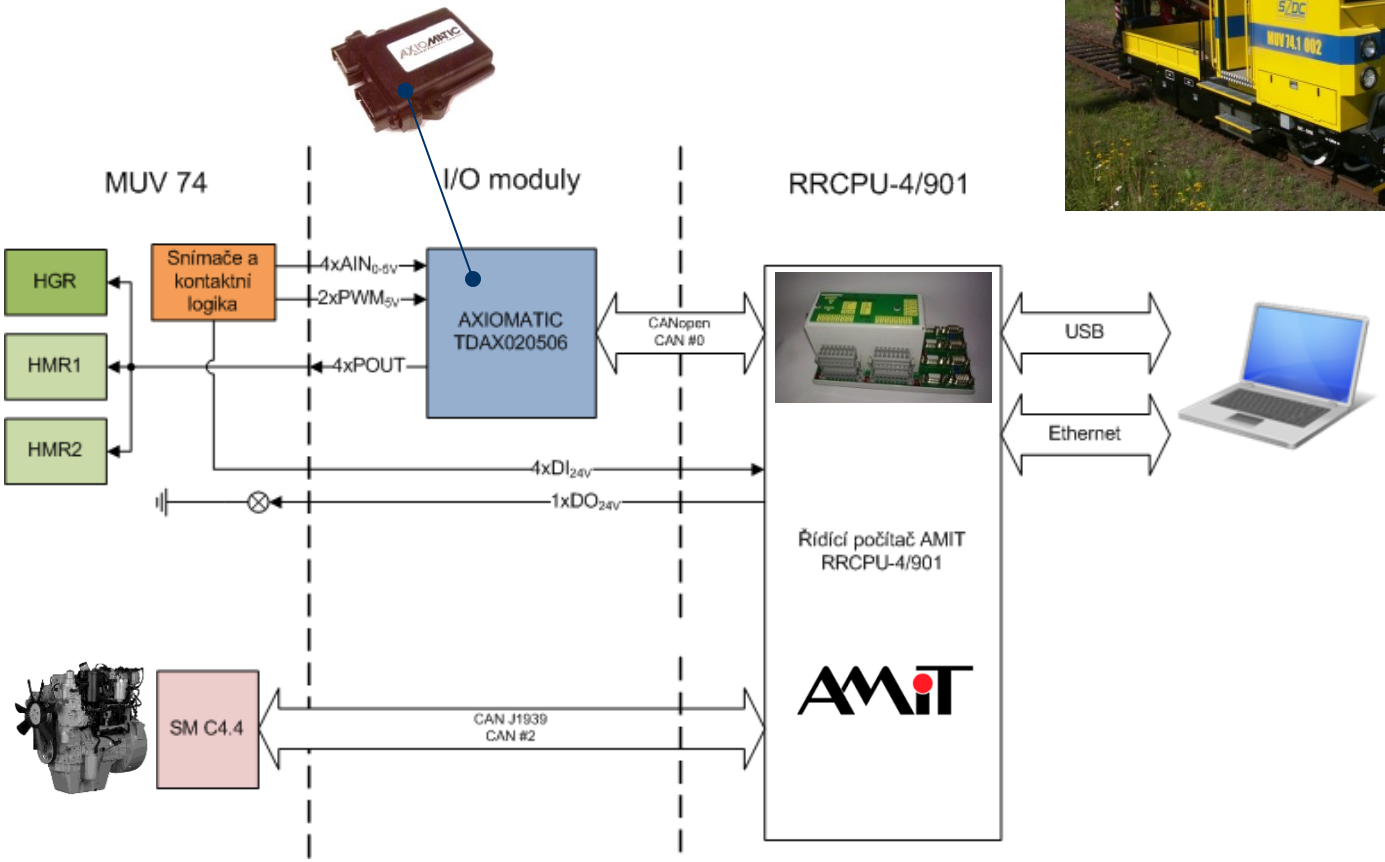


zdroj: [4],[9]

Použití CANopen

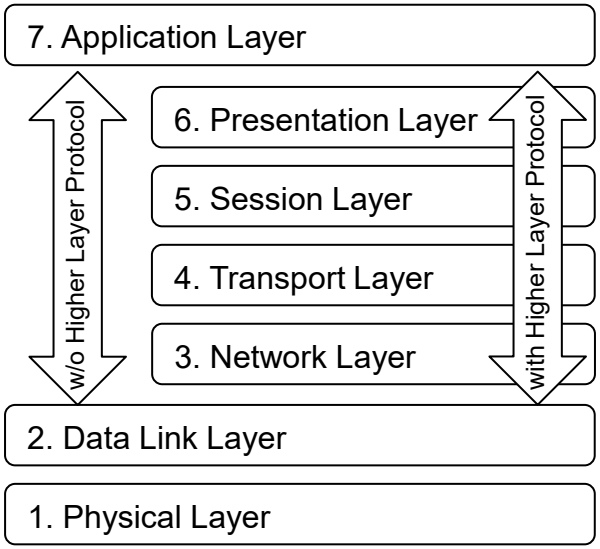
- Řídící systém speciálního drážního vozidla
  - Kombinace protokolu J1939 a CANopen.





Protokoly vyšších vrstev

- K čemu jsou dobré protokoly vyšších vrstev ?



- Přenos většího počtu dat než pouhých 8 bajtů (limit CAN zprávy).
- Network Management (inicializace sítě po startu, monitoring uzlů na síti, synchronizace uzlů na síti, …).
- Vytvoření komunikačního modelu typu Master/Slave.
- Standardizace – sjednocení způsobu komunikace mezi zařízeními různých výrobců, podpůrné prostředky, …

CAN in Automation (CiA) group

- CANopen byl vytvořen a je spravován organizací CiA (CAN in Automation Group)
  - Vyvíjí a spravuje standardy týkající se high-level protokolů na CAN BUS.
  - Seskupuje výrobce a uživatele zařízení.
  - Vydává CAN Newsletter.
  - www.can-cia.org



zdroj: [6]

Profile	Device
DS-401	Generic I/O modules
DS-402	Drives and motion control
DS-403	Not allocated
DS-404	Measuring devices and closed lo
DS-405	IEC 61131-3 programmable dev
DS-406	Encoders
DS-407	Public transportation - Passenge
DS-408	Fluid Power Technology - Hydrau
DS-409	Vehicle door control
DS-410	Declinometers

Historie CANopen

1983	Bosch začíná pracovat na vývoji vozidlové komunikační sběrnice.
1986	Uvedení CAN komunikačního protokolu.
1987	První CAN řadič od firmy Intel a Philips.
1991	Bosch uvádí CAN ve verzi 2.0.
1992	Založeno sdružení CAN in Automation (CiA).
1992	CiA představilo CAN Application Layer (CAL).
1992	První automobil vybavený sběrnicí CAN (Mercedes Benz).
1993	CAN standardizován pod ISO 11898.
1994	První mezinárodní CAN konference (iCC).
1994	Allen Bradley představuje průmyslový komunikační protokol DeviceNet na CAN.
1995	Doplnění standardu ISO 11898 (CAN) o 29-bit CAN-ID.
1995	Představen CANopen komunikační protokol.

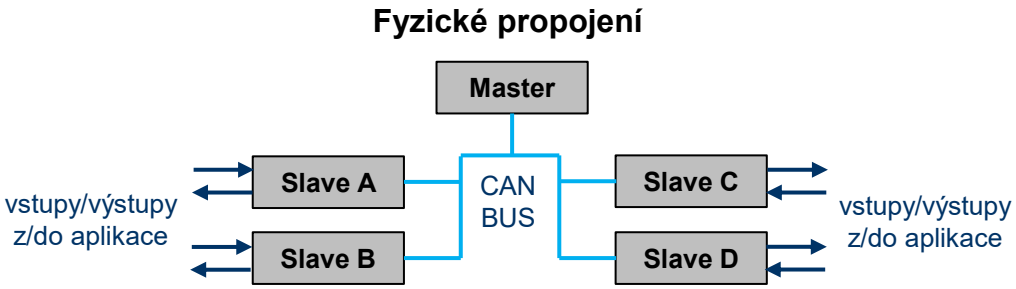
Další vlastnosti CANopen

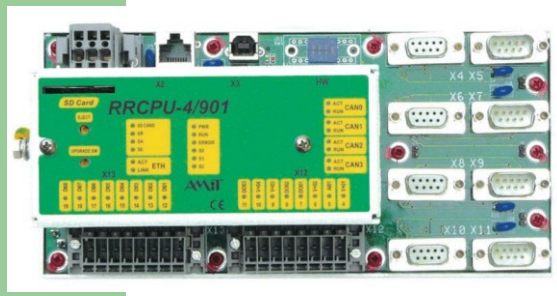
- Komunikace typu Master/Slave, Multi-Master a přímá komunikace mezi Slave bez účasti Mastera.
- Každé zařízení má svojí adresu (NODE ID) – max. 127 zařízení na jedné síti.
- Kontrola stavu zařízení (Node Guarding, HeartBeat, EMCY).
- Centrální konfigurace sítě, konfiguraci lze měnit za běhu.
- Konfigurace zařízení uložená v txt souborech (EDS, DCF).
- Synchronizace zařízení pomocí SYNC zpráv (vyslání dat ze všech Slave v jeden okamžik) .
- Protokol je vhodný pro embedded aplikace – malé nároky na paměť a zdroje MCU.
  - CANopen Slave: 10 až 20 kB ROM, < 1kB RAM
  - CANopen Master: 20 až 30 kB ROM, 1 kB RAM
  - Dostupné komunikační stacky pro řadu platforem.

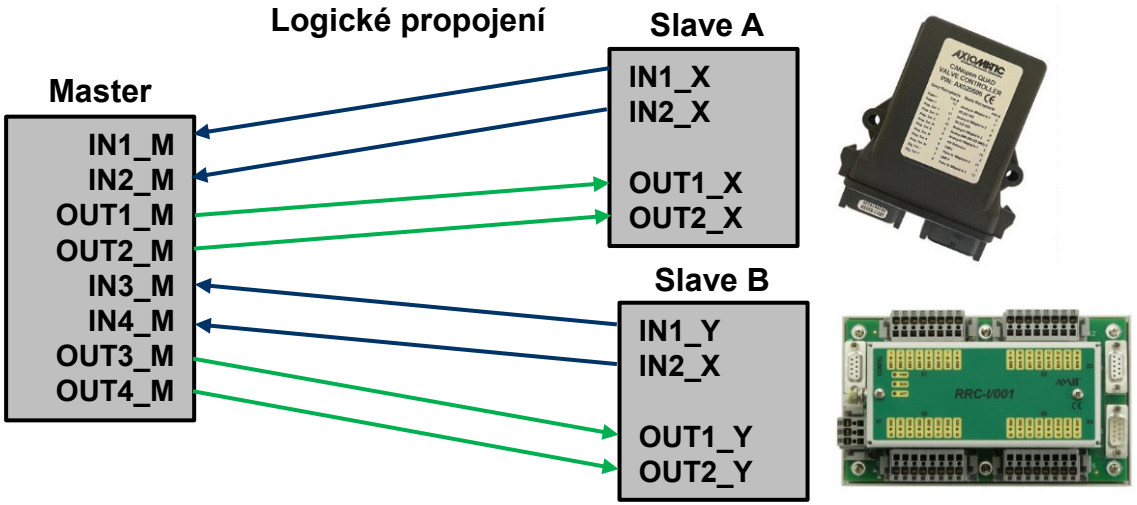
2. Komunikační model

Komunikační modely

- **Master/Slave komunikace**
  - Master načítá hodnoty vstupů ze Slave, vykonává řídící algoritmy a nastavuje výstupy prostřednictvím výstupních Slave.





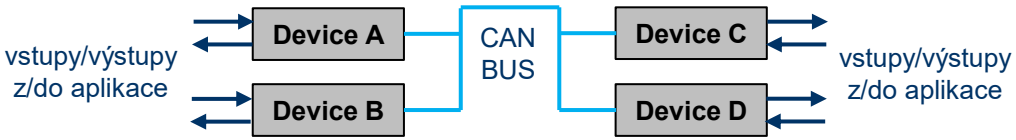


zdroj: [1]

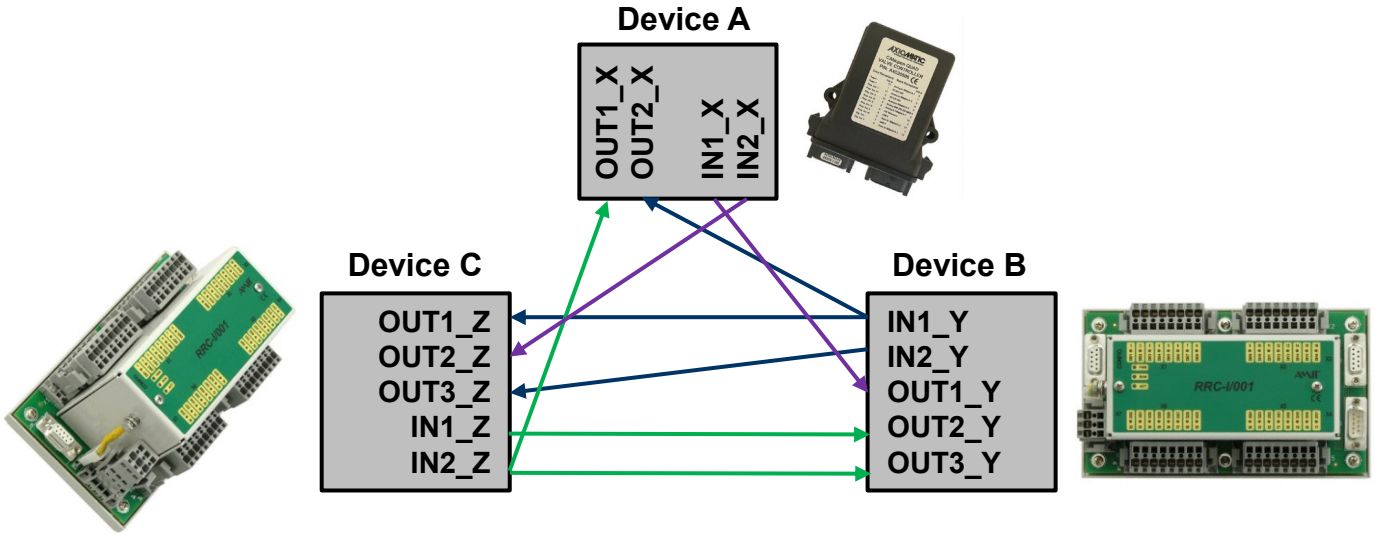
Komunikační modely

- **Přímá komunikace – bez účasti Mastera**
  - Není potřeba Master, zařízení samo ví jak data zpracovat a jakému dalšímu zařízení odeslat požadavek na nastavení výstupů.

Fyzické propojení

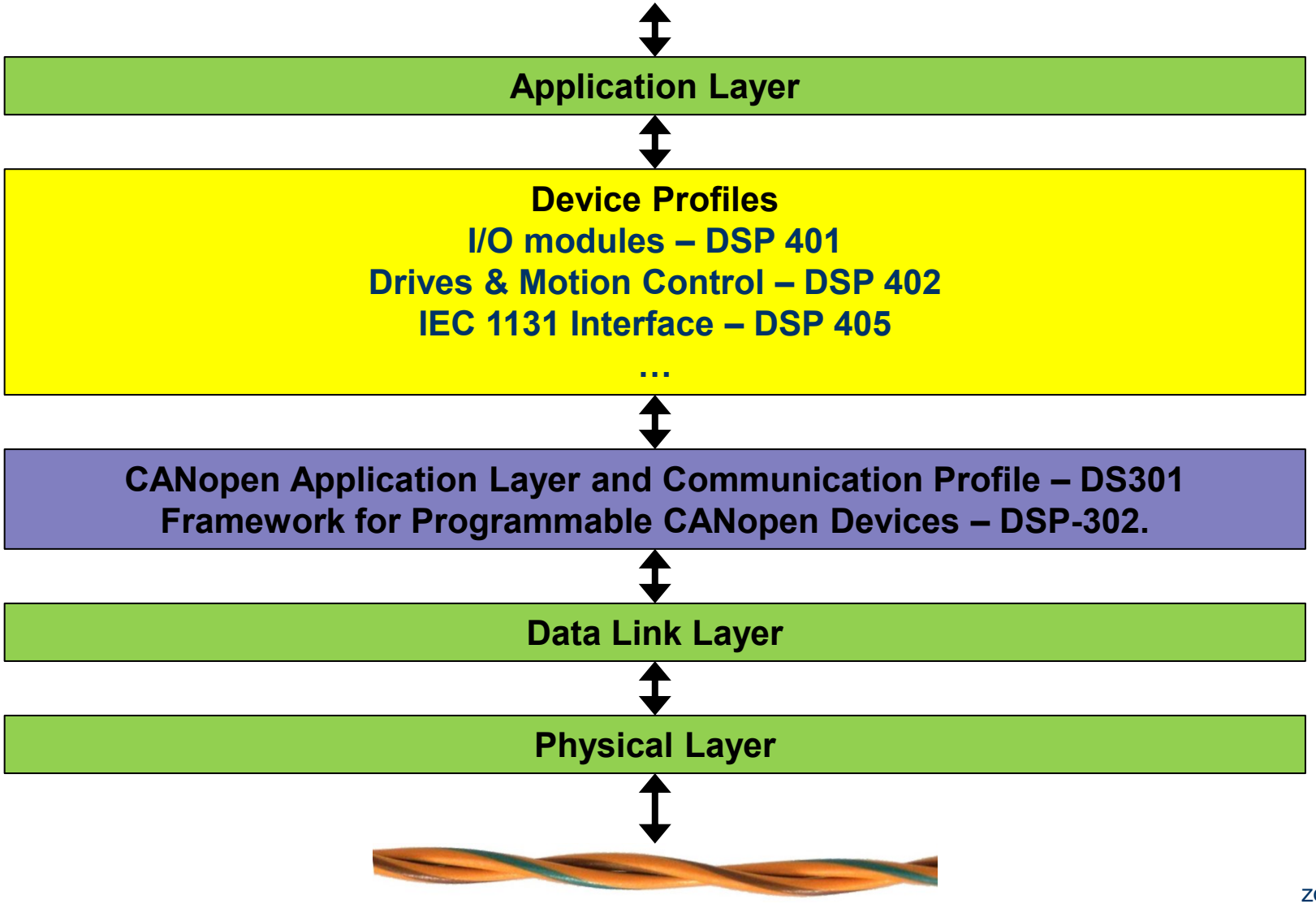


Logické propojení



zdroj: [1]

Síťový model CANopen



zdroj: [2]

Profily zařízení

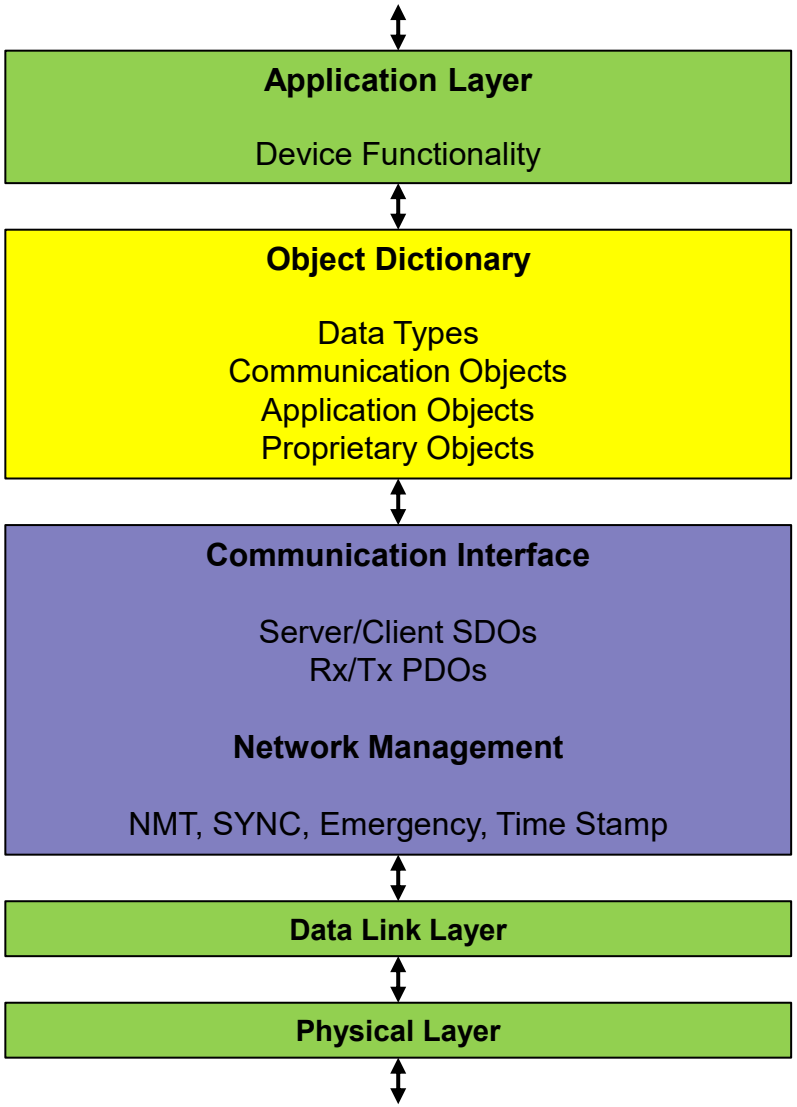
- Rozšíření základních funkcí protokolu nad DS-301.
- Specifické funkce pro daný typ zařízení, profil říká:
  - Jaké jsou dostupné komunikační parametry.
  - Jaké položky obsahuje slovník objektů.

Profile	Device
DS-404	Generic I/O modules
DS-402	Drives and motion control
DS-403	Not allocated
DS-404	Measuring devices and closed loop controllers
DS-405	IEC 6113-3 programmable devices
DS-406	Encoders
DS-407	Public transportation – Passenger Information Systems
DS-408	Fluid Power Technology – Hydraulic drives and proportional valves
DS-409	Vehicle door control
DS-410	Declinometers

- Pozn.: DS = Draft Specification (pevný standard, u nějž je předpoklad, že se nebude v budoucnu výrazně měnit).

zdroj: [6]

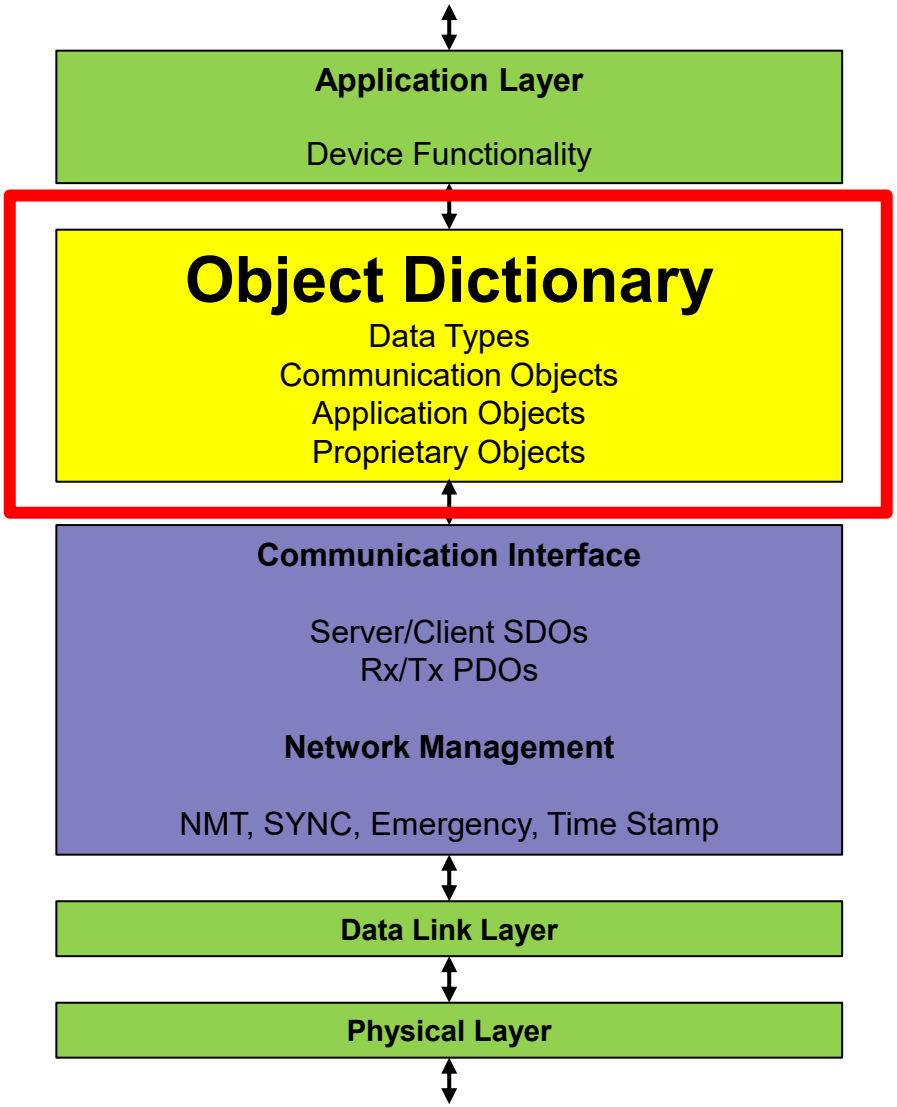
Model zařízení



zdroj: [2]

3. Slovník objektů

Model zařízení



zdroj: [2]

Object dictionary (slovník objektů)

- „Tabulka“ s položkami, které jsou přístupné po síti
  - Položky ve slovníku mohou číst nebo do nich zapisovat jiná zařízení na síti.
- Obsahuje:
  - Nastavení komunikace po CANopen daného zařízení.
  - Nastavení specifických funkcí daného zařízení (např. filtrace analogových vstupů, perioda měření, převodní charakteristika apod.).
  - Procesní data (měřené, žádané hodnoty).

Index	Object	
0000h	Not Used	Common to any Device
0001h – 025Fh	Data Type	
0260h – 0FFFh	Reserved	
1000h – 1FFFh	Communication Profile Area	
2000h – 5FFFh	Manufacturer Specific Profile Area	Device Specific
6000h – 9FFFh	Standardized Device Profile Area	
A000h - AFFFh	Network Variables	
B000h - FFFFh	Reserved for Future Use	

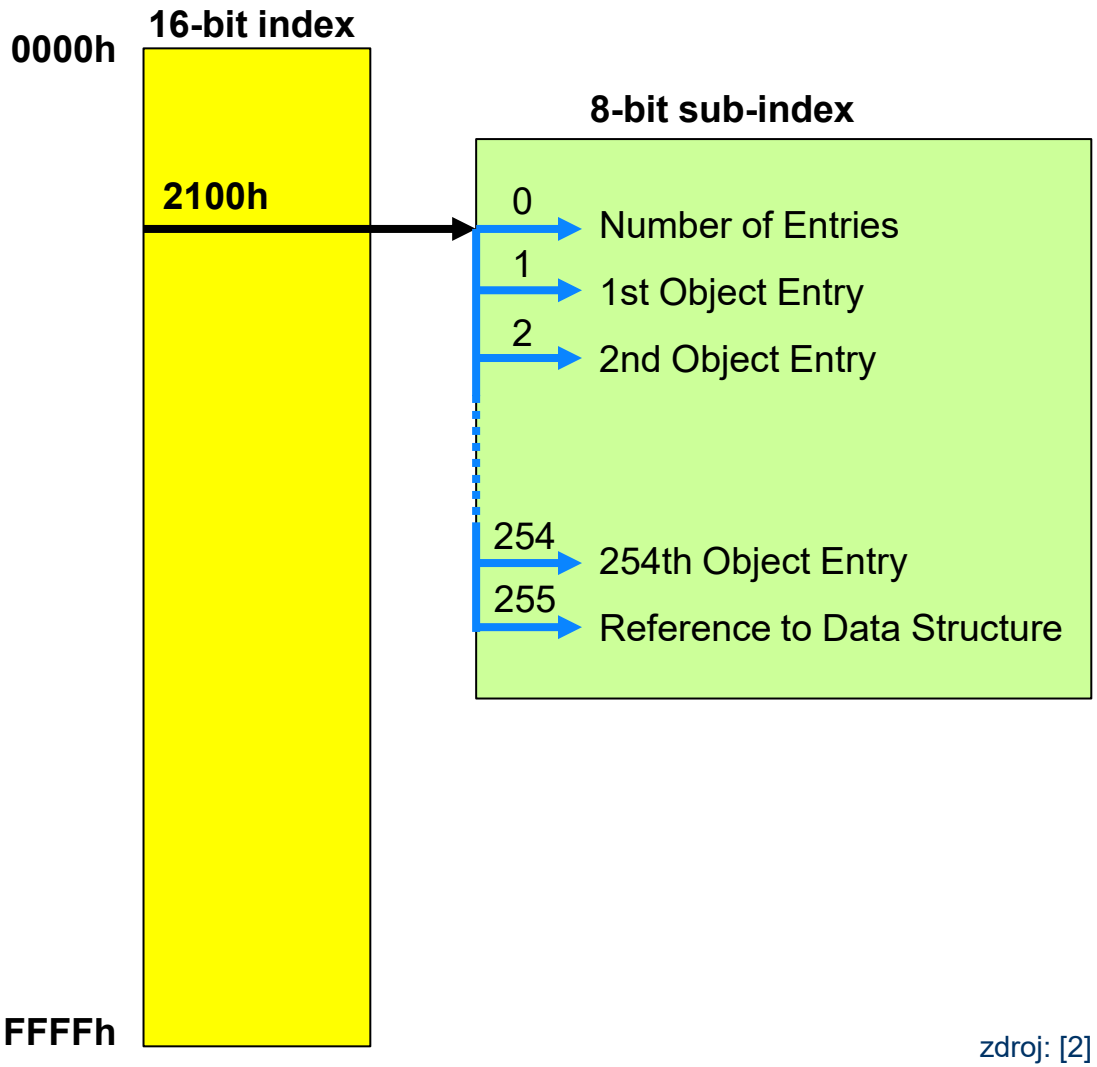
zdroj: [2]

Organizace slovníku objektů

- Max. 65536 položek (Entries)
  - 16-bit index
- Max. 256 podpoložek (Subentries)
  - 8 bit index

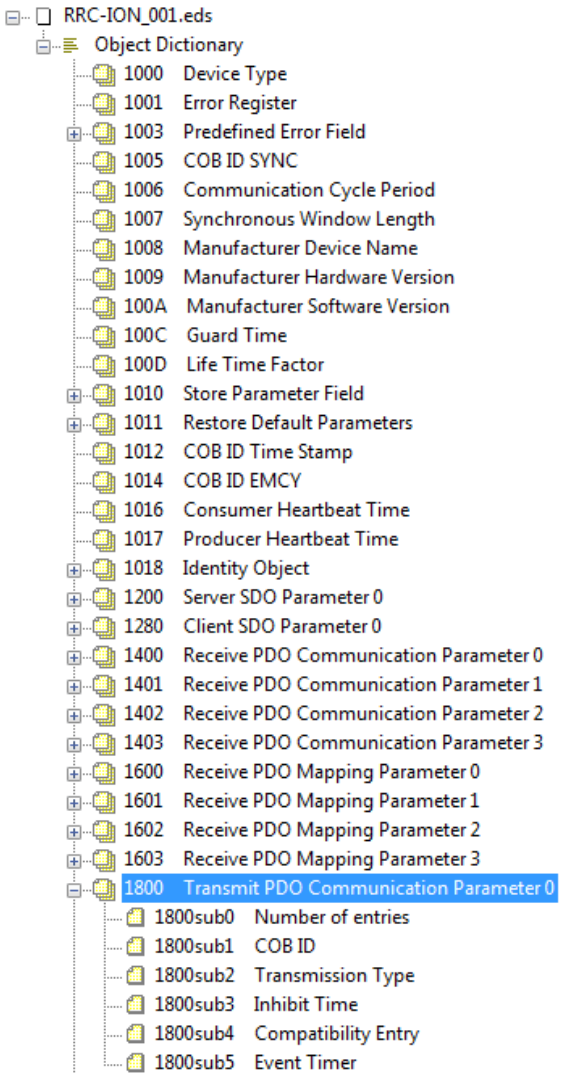
Např.: [2100h,01h]

index sub-index



zdroj: [2]

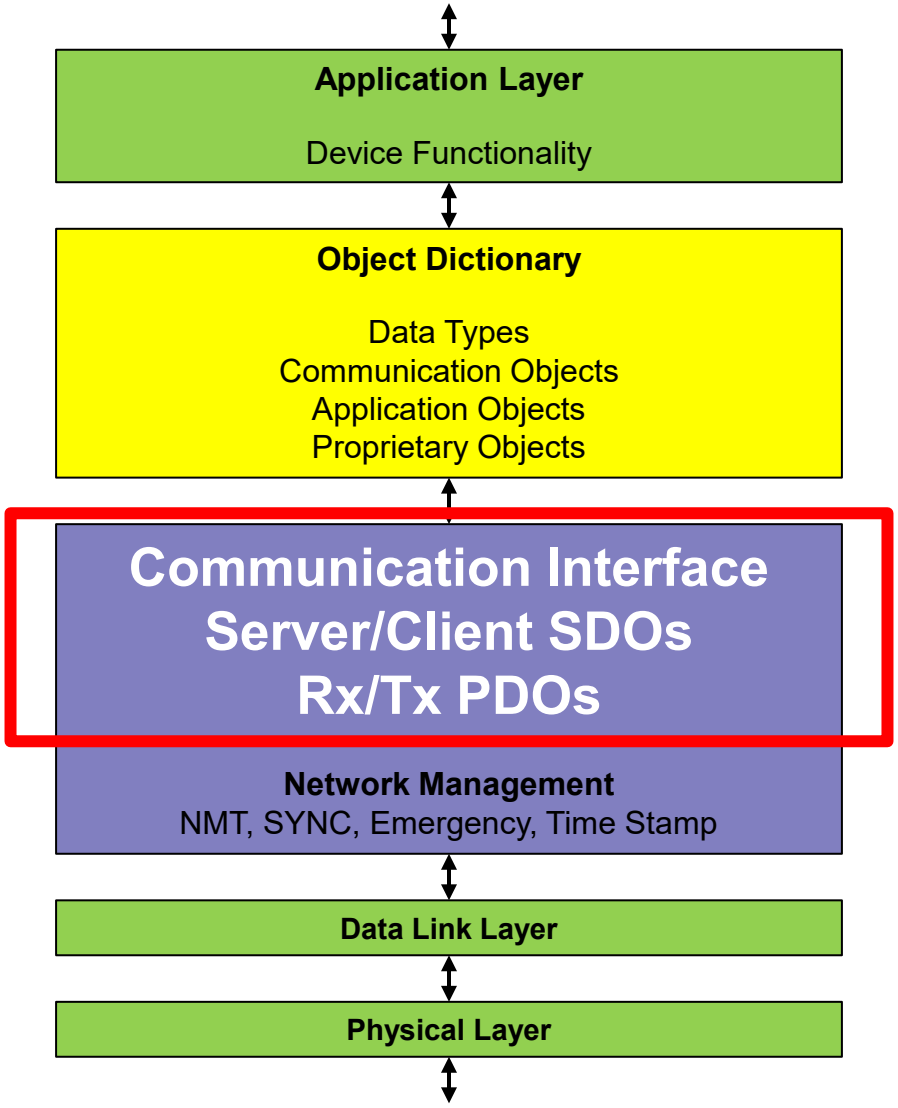
Organizace slovníku objektů



Index	Sub-Index	Name	Type
1800	00h	Number of entries	UNSIGNED8
	01h	COB ID	UNSIGNED32
	02h	Transmission Type	UNSIGNED8
	03h	Inhibit Time	UNSIGNED16
	04h	Reserved	UNSIGNED8
	05h	Event Timer	UNSIGNED16

Počet položek – dle typu zařízení
Položky – povinné, nepovinné
Přístup k položkám – RO, WO, RW

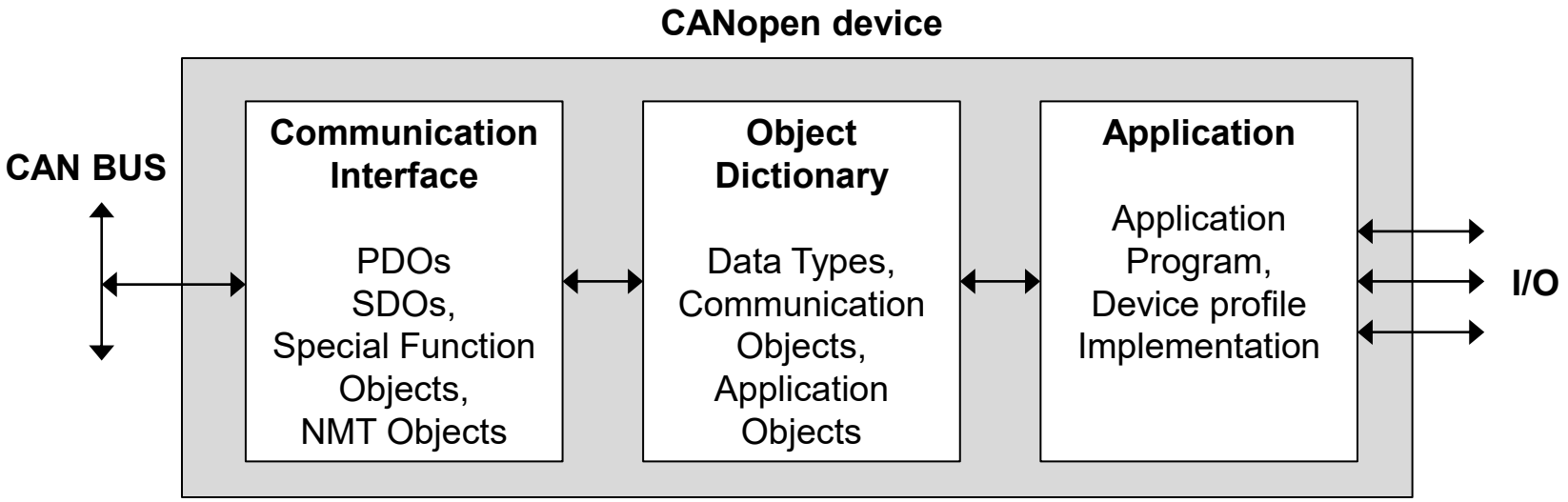
Komunikační interface



zdroj: [2]

Komunikační interface

- Komunikace mezi zařízeními probíhá pomocí těchto zpráv:
  - SDO (Service Data Object)
  - PDO (Process Data Object)
  - Administrativní zprávy - zprávy síťového managementu (Network Management)
    - NMT
    - SYNC
    - Time Stamp
    - Emergency



zdroj: [3]

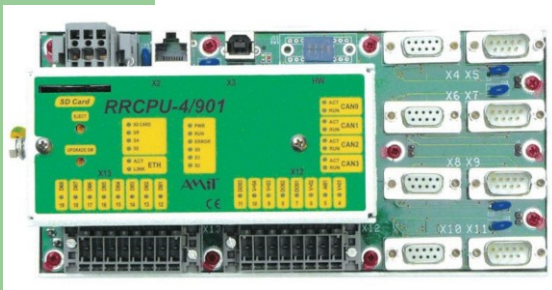
4. SDO a PDO

SDO

- **SDO = „Service Data Object“**
- Pro přístup (čtení/zápis) ke všem objektům ve slovníku daného zařízení.
- **SDO server** – zařízení, které zpracovává požadavky na čtení/zápis ze/do slovníku objektů.
- **SDO client** – zařízení (Master nebo nástroj pro diagnostiku), které posílá požadavky na čtení nebo zápis z/do slovníku objektů druhého zařízení.
- Potvrzovaná komunikace (žádost → odpověď).
- Podpora přenosu dat delších než jedna CAN zpráva.
- Používá se výlučně pro konfiguraci zařízení.
- Pro přenos procesních dat (měřené, žádané hodnoty apod.) se používají **PDO zprávy**.

CAN ID použité pro SDO zprávy

- Identifikátory (CAN ID) použité pro SDO zprávy
  - SDO client → server: **600h + Node ID**
  - SDO server → client (odpověď): **580h + Node ID**



Master SDO Tx

Master SDO Rx

NMT Master
(client)

Sends SDO request by using message ID:
600h + Node ID

Expects reply in message ID:
580h + Node ID

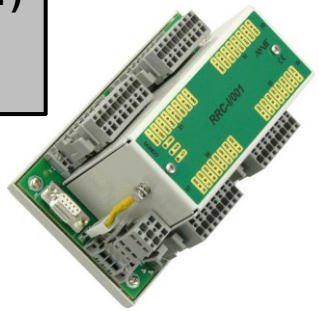
Node 1 (server)
Tx SDO: 581h
Rx SDO: 601h



Node 2 (server)
Tx SDO: 582h
Rx SDO: 602h



Node 3 (server)
Tx SDO: 582h
Rx SDO: 602h



- V jeden okamžik může s daným zařízením (Node) komunikovat pouze 1 SDO klient.

zdroj: [1]

Typy přenosu SDO zpráv

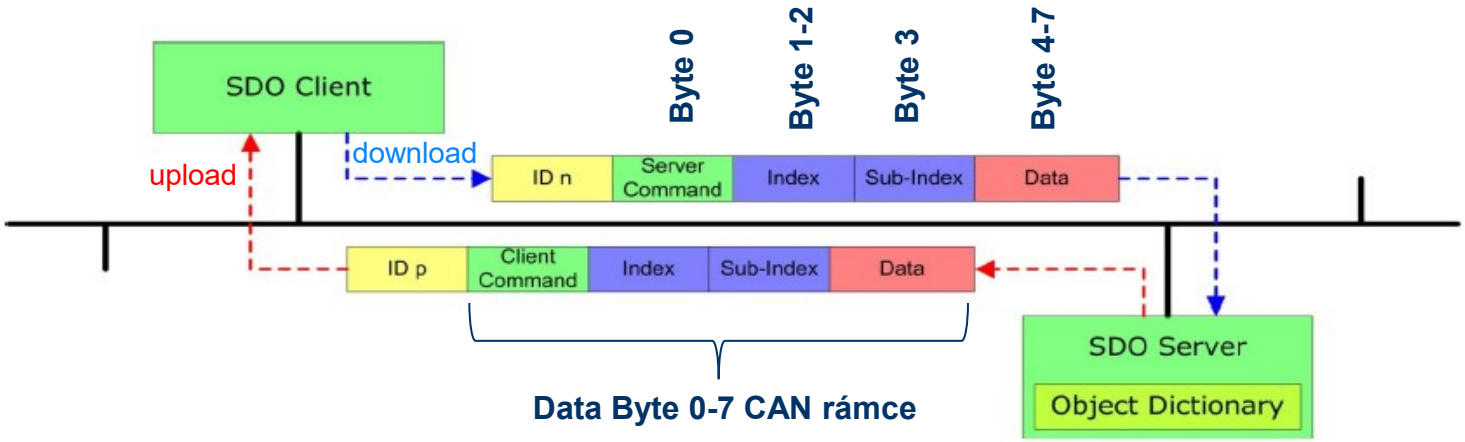
- Typ přenosu dle množství přenášených dat.

- **Expedited Transfer**
  - Pro přenos max. 4 bajtů dat.
  - Stačí 1 SDO zpráva.

- **Segmented Transfer, Block Transfer**
  - Jakékoliv množství dat.
  - Data jsou rozdělena do více SDO zpráv.

Struktura SDO zpráv

- **Struktura SDO zprávy typu expedited transfer (4B dat)**



- **Význam polí**
  - ID n(p) = CAN-ID zprávy ve směru SDO client→server (server → client)
  - Server (Client) Command
    - Download / Upload
    - Request / Response
    - Typ přenosu (segmented, expedited)
    - Počet přenášených datových bajtů
    - Toggle bit
  - Index, Sub-Index = objekt ve slovníku
  - Data = až 4 bajty dat

Download – zápis do slovníku objektů
Upload – čtení ze slovníku objektů

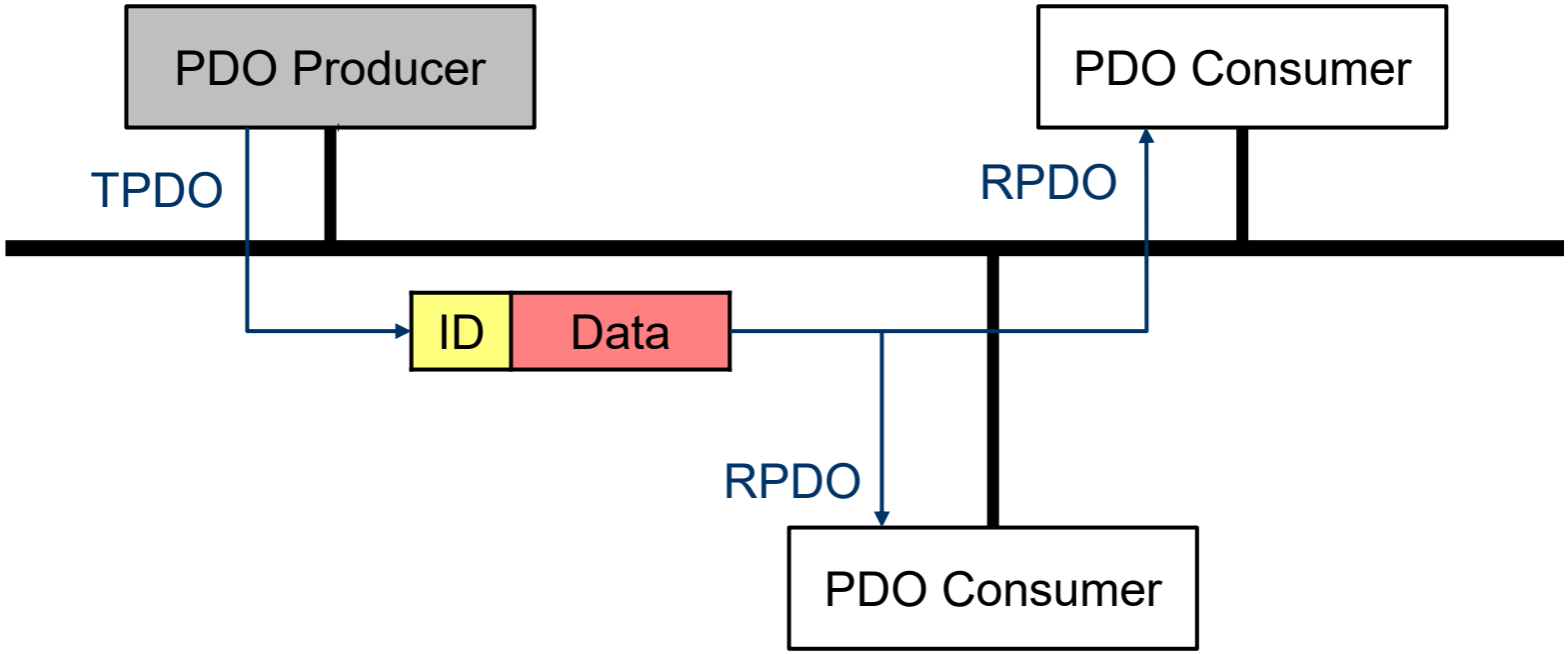
zdroj: [2]

PDO

- **PDO = „Process Data Object“**
- Pro přenos procesních dat (žádané, měřené hodnoty apod.) v reálné čase.
- Nepotvrzovaný přenos dat.
- Princip Producer/Consumer
  - Producer = ten, kdo vysílá zprávu
  - Consumer = ten, který jí přijímá a zpracovává
- Dva typy PDO
  - TPDO = Transmit PDO
  - RPDO = Receive PDO
- 1 PDO = 1 CAN zpráva = 1 unikátní CAN ID
- Počet použitých TPDO/RPDO u daného zařízení závisí na konkrétní aplikaci (max. 512 TPDO a 512 RPDO).
- Max. počet přenášených dat v 1 PDO je 8 bajtů.

PDO – princip přenosu

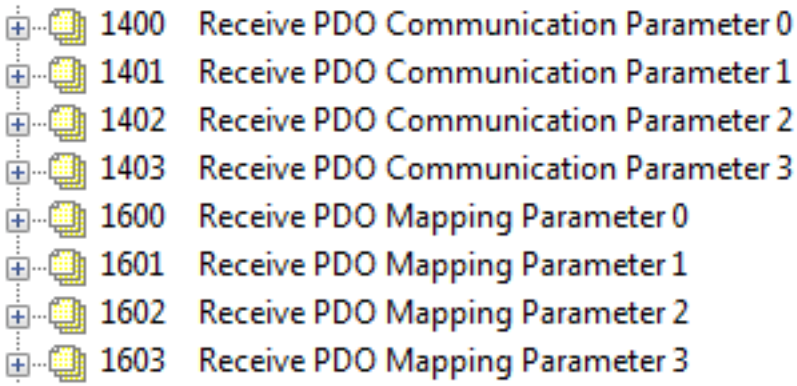
- Producer/Consumer princip
  - Producer (odesílatel) PDO nebo Consumer (příjemce) PDO může být Master i Slave.
  - TPDO = Transmit PDO
  - RPDO = Receive PDO



zdroj: [2]

PDO – parametry

- Pro každé xPDO se nastavují následující parametry.
- **Communication parameters**
  - Jaké CAN-ID zprávy bude použito pro přenos PDO.
  - Kdy bude PDO odesláno (jaká spouštěcí událost vyvolá odeslání, jak často se bude odesílat).
  - TPDO a RPDO communication parameters se významově mírně liší.
- **Mapping parameters**
  - Definuje obsah PDO, tj. co se v PDO bude přenášet (formou odkazu na položku ze slovníku objektů).



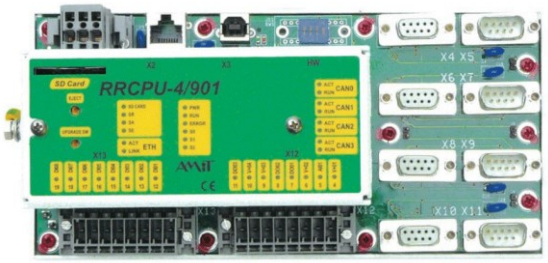


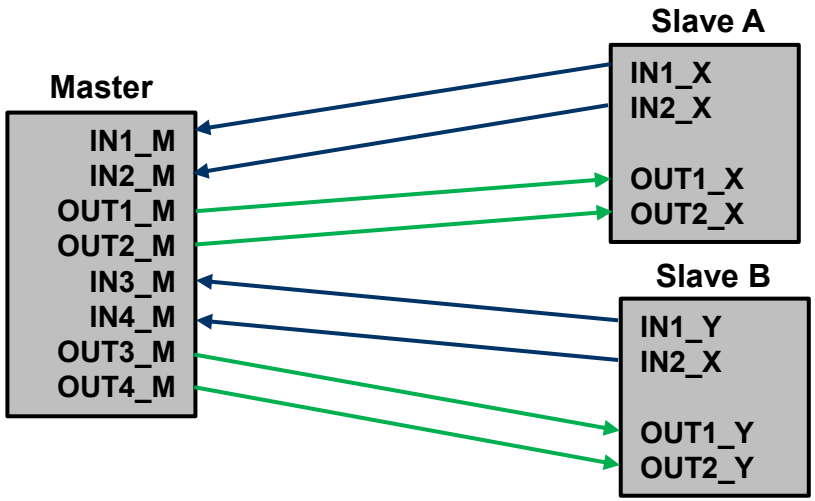
PDO – linking

- Určení, které TPDO bude dané zařízení vysílat, a které RPDO bude přijímat (ostatní ignoruje).
- PDO linking může být konfigurován Masterem ve fázi inicializace sítě nebo jsou zařízení nakonfigurována před spuštěním sítě.
- Obecně existují dva způsoby propojení (PDO linking):
  - Master/Slave model.
  - Přímé propojení mezi zařízení.

PDO – linking

- **Defaultně Master/Slave model**
  - Master načte vstupní data ze Slave, tato data zpracuje (vykoná řídící algoritmus), nově vypočtené hodnoty odešle do Slave, který zajistí např. aktivaci akčního členu.





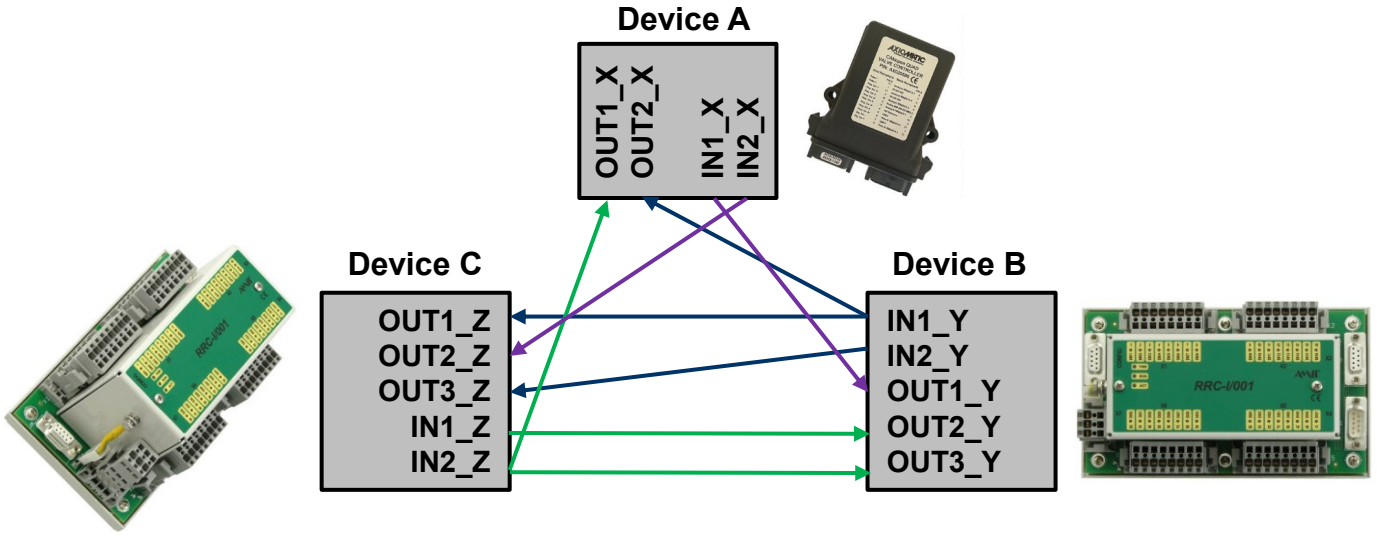




zdroj: [1]

PDO – linking

- **Přímé propojení zařízení mezi sebou**
  - Není potřeba Master, zařízení samo ví jak data zpracovat a jakému dalšímu zařízení odeslat požadavek na nastavení výstupů.



zdroj: [1]

PDO – linking

- Každé zařízení umožňuje defaultní nastavení PDO („pre-defined connection set“), které odpovídá modelu **Master/Slave**, kdy neexistuje přímé propojení TPDO a RPDO mezi Slave zařízeními navzájem.
  - Jedině Master může přijímat/vysílat PDO ze/do všech zařízení
- V tomto případě je CAN ID každého xPDO na síti unikátní
  - Např. zařízení se síťovou adresou 5:

    NODE ID = 5
    CAN ID TPDO0 (vysílání) = 180h + NODE ID = 185h
    CAN ID RPDO0 (příjem) = 200h + NODE ID = 205h
    atd. pro další TPDO a RPDO
  - Obdobně pro ostatní zařízení s jinou síťovou adresou
- V případě **přímého propojení** zařízení
  - Např. zařízení se síťovou adresou 5 chce ve svém RPDO3 přijímat PDO, které vysílá zařízení s adresou 2 ve svém TPDO2:

    SLAVE 2 (vysílač): CAN ID TPDO2 = 380h + 2h = 382h
    SLAVE 5 (přijímač): CAN ID RPDO3 = CAN ID TPDO2 = 382h

PDO – linking

- Defaultní nastavení CAN ID u TPDO a RPDO.

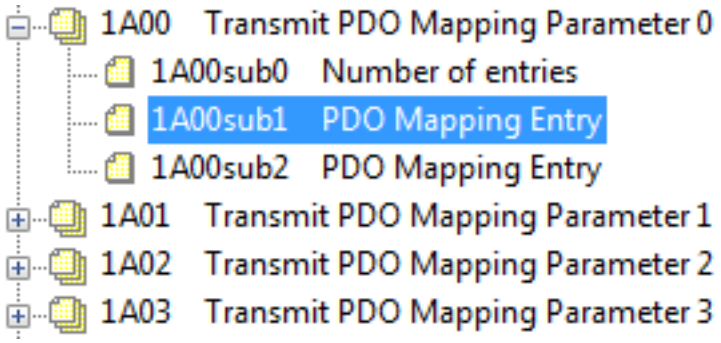
Rostoucí priorita zprávy

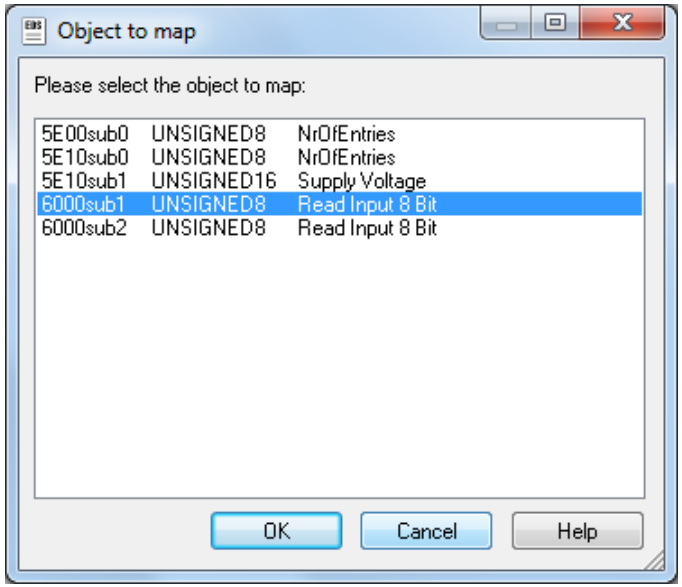
CAN ID			
From	**To**	**Communication Object**	**Comment**
0h	-	NMT Service	From NMT Master
80h	-	SYNC Message	From SYNC Producer
81h	FFh	Emergency Message	From timestamp producer
100h	-	Time Stamp Message	From nodes 1 to 127
181h	1FFh	1st Transmit PDO	For nodes 1 to 127
201h	27Fh	1st Receive PDO	From nodes 1 to 127
281h	2FFh	2nd Transmit PDO	For nodes 1 to 127
301h	37Fh	2nd Receive PDO	From nodes 1 to 127
381h	2FFh	3rd Transmit PDO	For nodes 1 to 127
401h	47Fh	3rd Receive PDO	From nodes 1 to 127
481h	4FFh	4th Transmit PDO	For nodes 1 to 127
501h	57Fh	4th Receive PDO	From nodes 1 to 127
581h	5FFh	Transmit SDO	For nodes 1 to 127
601h	67Fh	Receive SDO	From nodes 1 to 127
701h	77Fh	NMT Error Control	For nodes 1 to 127

zdroj: [1]

PDO – mapování

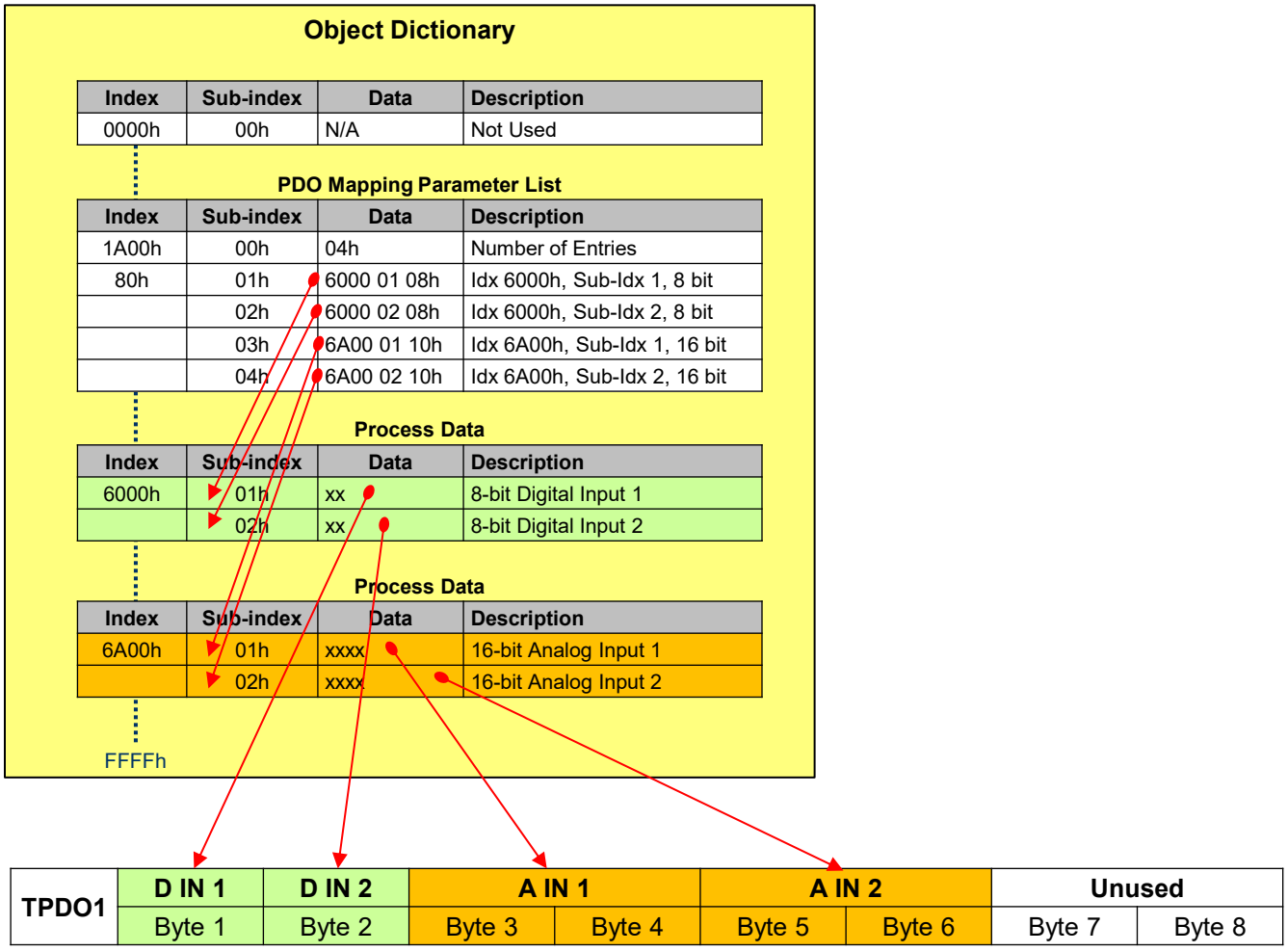
- **Určení datového obsahu PDO**
- TPDO – jaká data se budou odesílat
- RPDO – jaká data se budou přijímat
- Mapování procesních dat objektu ve slovníku objektů do PDO
  - Pomocí „Mapping Parameter List“.
  - Ve formě Index, Sub-Index, počet obsazených bitů.
  - Každé PDO má svůj vlastní „Mapping Parameter List“.





PDO – mapování

- **Příklad**



zdroj: [2]

PDO – mapování



- Která PDO jsou defaultně předdefinována a jaké je defaultní mapování závisí na profilu zařízení a výrobci zařízení.

TPDO1: default ID 0x180 + node ID

Subindex	Value	Object
0	4	Number of mapped application objects in PDO
1	0x71000110	AI Input Field Value for AIN1
2	0x71000210	AI Input Field Value for AIN2
3	0x71000310	AI Input Field Value for AIN3
4	0x71000410	AI Input Field Value for AIN4

TPDO2: default ID 0x280 + node ID

Subindex	Value	Object
0	2	Number of mapped application object
1	0x71000510	AI Input Field Value for FIN1
2	0x71000610	AI Input Field Value for FIN2
3	0	
4	0	

TPDO3: default ID 0x380 + node ID

Subindex	Value	Object
0	2	Number of mapped application object
1	0x60000108	DI Read State 8 Input Lines
2	0x300C0108	DO Read State 8 Output Lines
3	0	
4	0	

TPDO4: default ID 0x480 + node ID

Subindex	Value	Object
0	4	Number of mapped application object
1	0x30C00110	AO Measured Current for POUT1
2	0x30C00210	AO Measured Current for POUT2
3	0x30C00310	AO Measured Current for POUT3
4	0x30C00410	AO Measured Current for POUT4

RPDO1: default ID 0x200 + node ID

Subindex	Value	Object
0	4	Number of mapped application objects in PDO
1	0x73300110	AO Output Field Value for POUT1
2	0x73300210	AO Output Field Value for POUT2
3	0x73300310	AO Output Field Value for POUT3
4	0x73300410	AO Output Field Value for POUT4

RPDO2: default ID 0x300 + node ID

Subindex	Value	Object
0	1	Number of mapped application objects in PDO
1	0x62000108	DO Write State 8 Output Lines
2	0	
3	0	
4	0	

RPDO3: default ID 0x400 + node ID

Subindex	Value	Object
0	0	Number of mapped application objects in PDO
1	0	
2	0	
3	0	
4	0	

RPDO4: default ID 0x500 + node ID

Subindex	Value	Object
0	0	Number of mapped application objects in PDO
1	0	
2	0	
3	0	
4	0	

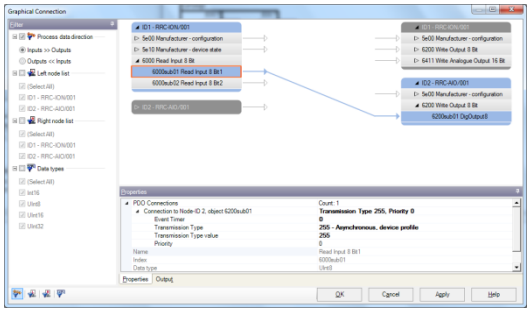
zdroj: [7]

PDO – mapování

- **Kdy lze nastavit mapování ?**
  - Nelze měnit, je pevně dáno z výroby – **statické mapování**
    - Nevýhoda – obsah PDO nelze optimalizovat (např. mít DI a AI v jednom PDO).
  - Lze za běhu zařízení – **dynamické mapování**
    - Zařízení je ve stavu Operational = normálně vysílá a přijímá (nejen) PDO.
    - PDO se zakáže, změní se mapování a následně se PDO opět povolí.
  - Lze pouze v určitém stavu zařízení – **variabilní mapování**
    - Zařízení musí být ve stavu Pre-Operational = vysílání a příjem PDO je zastaveno.
    - Zabránění dočasné nekonzistence mezi TPDO a RPDO.

- **Jak nastavit mapování ?**
  - Master může nakonfigurovat mapování PDO ve Slave při startu sítě.
  - Pomocí konfiguračního nástroje (HW + SW, např. CAN analyzátor a vhodný SW).

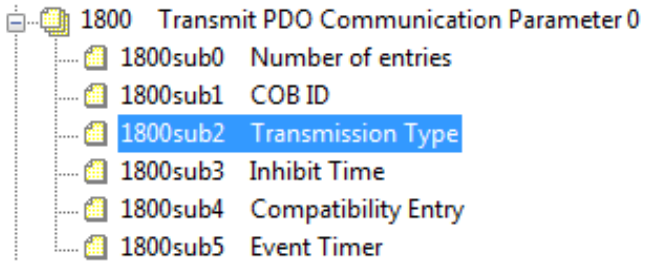




zdroj: [4]

PDO – typy přenosu

- Určení, kdy se bude PDO přenášet
- Nastavuje se u každého PDO



Transmission Type	Condition to trigger PDO (B = both needed, O = one or both)			PDO Transmission
	SYNC*	RTR*	Event*	
0	B	-	B	Sync, acyclic
1 – 240	O	-	-	Sync, cyclic
241 – 251	-	-	-	Reserved
252	B	B	-	Sync, after RTR
253	-	O	-	Async, after RTR
254	-	O	O	Async, manufacturer specific event
255	-	O	O	Async, device profile specific event

*SYNC = SYNC-object oriented
*RTR = Remote Transmission Request received (*Remote Frame* CAN message type)
*Event = e.g. change of value or timer interrupt

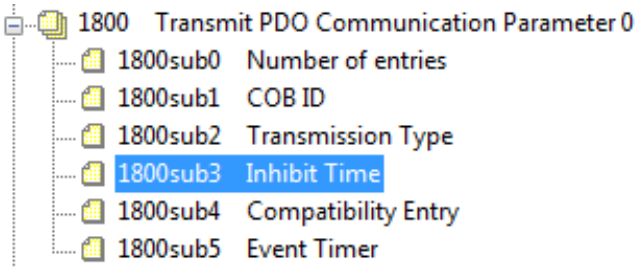
zdroj: [3]

PDO – typy přenosu

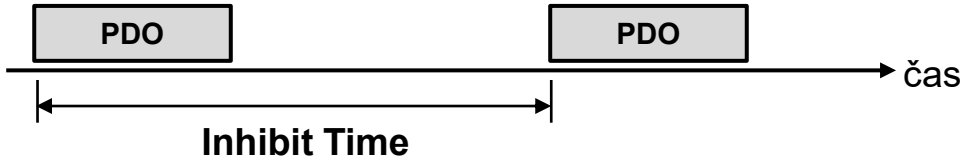
- **Event driven**
  - PDO je odesláno při výskytu události, např. změně logické úrovně na DI vstupu, dosažení limitu apod. (závisí na profilu zařízení).
  - Výhoda: rychlá odezva
  - Nevýhoda: neustálé změny na vstupu mohou zahltit sběrnici → zavedení tzv. **Inhibit Timer**
  - **Inhibit Timer** – čas, který musí vypršet předtím, než může být to samé PDO opět odesláno. Lze jím určit nejhorší scénář, který může nastat, tj. maximální frekvenci odesílání daného PDO.

Bez Inhibit Timeru může zařízení odeslat PDO po každém výskytu události.





S Inhibit Timerem může zařízení znovu odeslat PDO až po vypršení času.



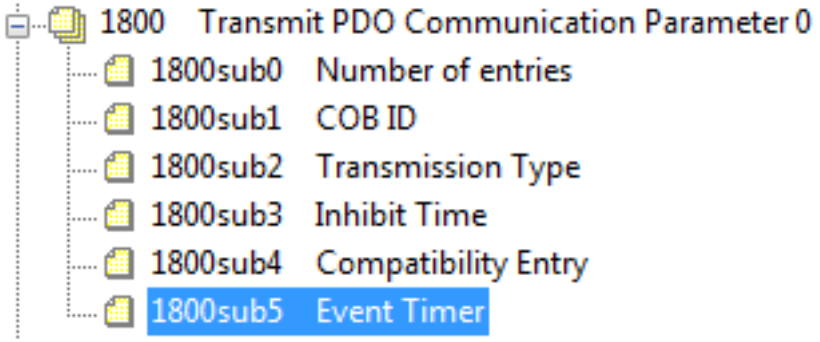


Inhibit Time *se nastavuje v kroku 100 mikrosekund.*

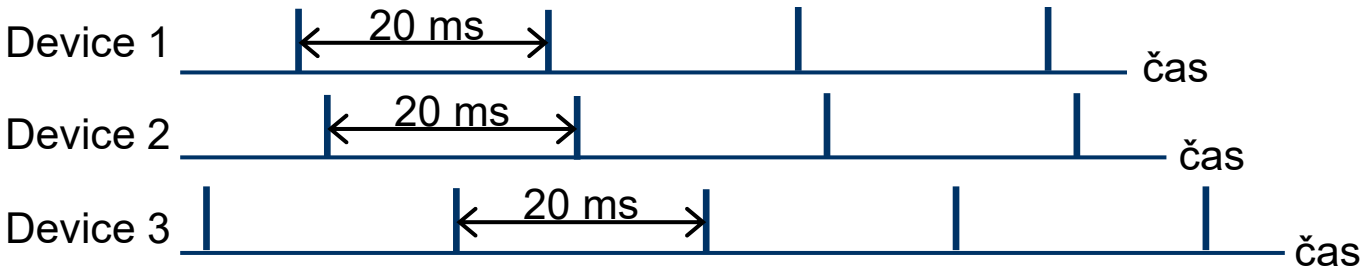
zdroj: [1]

PDO – typy přenosu

- **Time driven**
  - TPDO je odesíláno periodicky dle hodnoty nastavené v **Event Timer**.
  - Perioda se nastavuje s krokem 1 ms.
  - Každé zařízení má časovač pro každé PDO, po jehož vypršení je **TPDO** odesláno, u **RPDO** má význam **watchdogu** při příjmu.



  - Časovače mezi různými zařízeními nejsou navzájem synchronizované, okamžiky odeslání dat z různých zařízení jsou časově posunuté

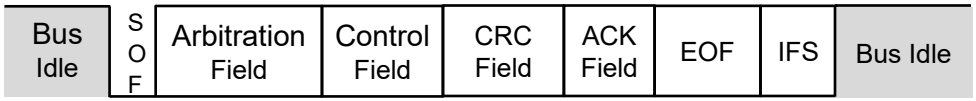


PDO – typy přenosu

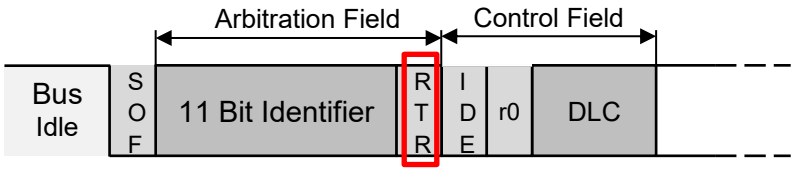
- **Individual polling**
  - TPDO je odesláno až na vyžádání, které přijde od jiného zařízení.
  - Za tímto účelem se používá **Remote Request Frame** (bit RTR v CAN rámci), který je odeslán zařízením, které požaduje data.
  - Tento způsob není doporučován z důvodu různé implementace **Remote Request Frame** v CAN řadičích různých výrobců, místo toho se doporučuje použití SYNC signálu (viz dále).
  - Připomenutí:
    - CAN datový rámec:



    - CAN rámec „Žádost o data“ (Remote Request Frame):



Chybí Data Field a bit RTR = 1



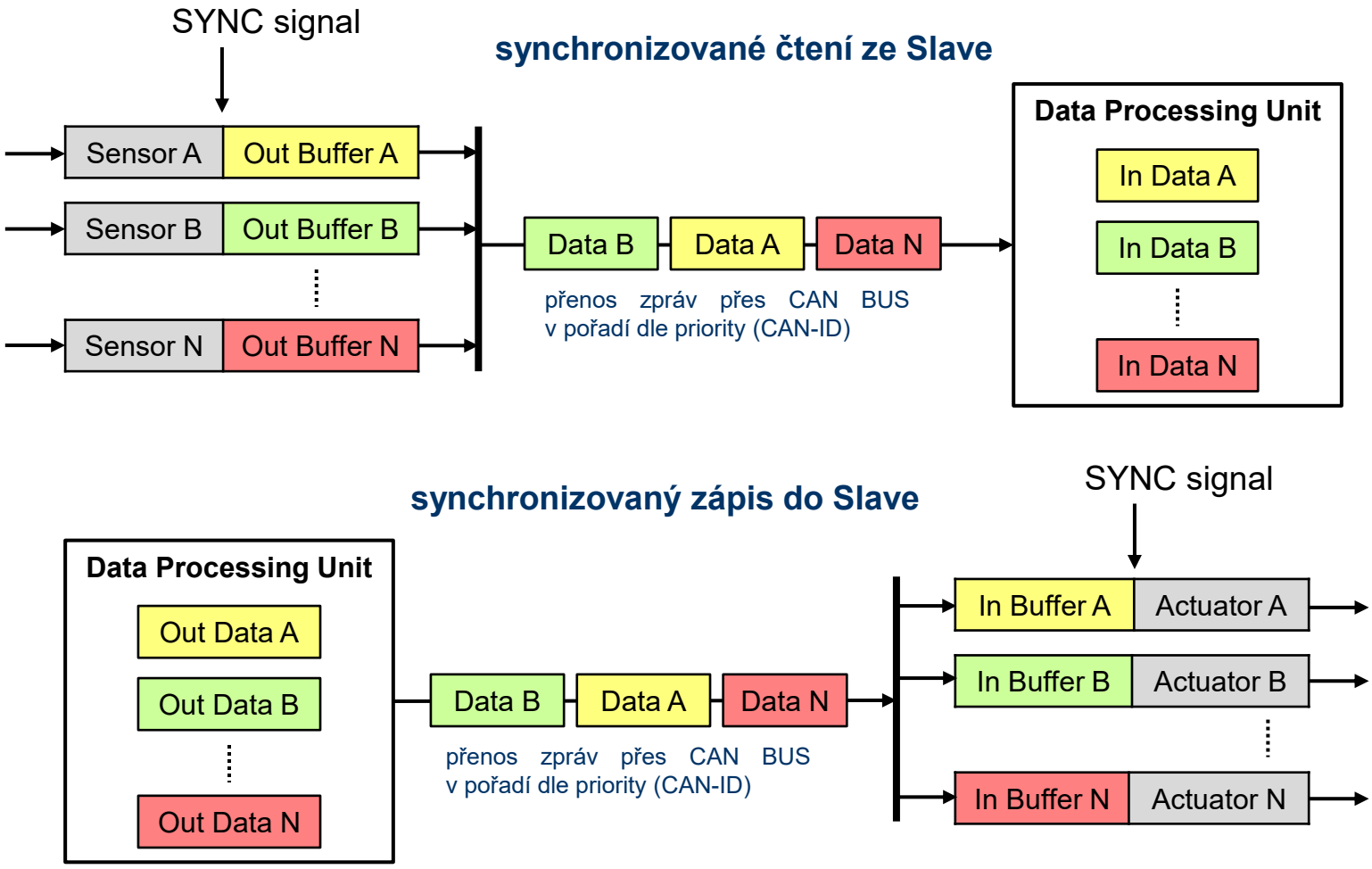
zdroj: [1]

PDO – typy přenosu

- **Synchronized or grouped polling**
  - Hlavní myšlenka:
    - hodnoty vstupů získat ze všech zařízení v jeden okamžik
    - hodnoty výstupů změnit u všech zařízení v jeden okamžik
  - TPDO je odesláno až při příjmu SYNC zprávy.
  - Data příjmutá v RPDO jsou z RPDO vyzvednuta a zpracována až při příjmu SYNC zprávy.
  - SYNC zpráva se vysílá periodicky – jedná se vlastně o **globální Event Timer společný pro všechna zařízení**.
  - Zdrojem SYNC zprávy může být libovolné zařízení (ne jen Master).
  - CAN-ID SYNC zprávy je 80h (vysoká priorita) a zpráva neobsahuje žádná data.

PDO – typy přenosu

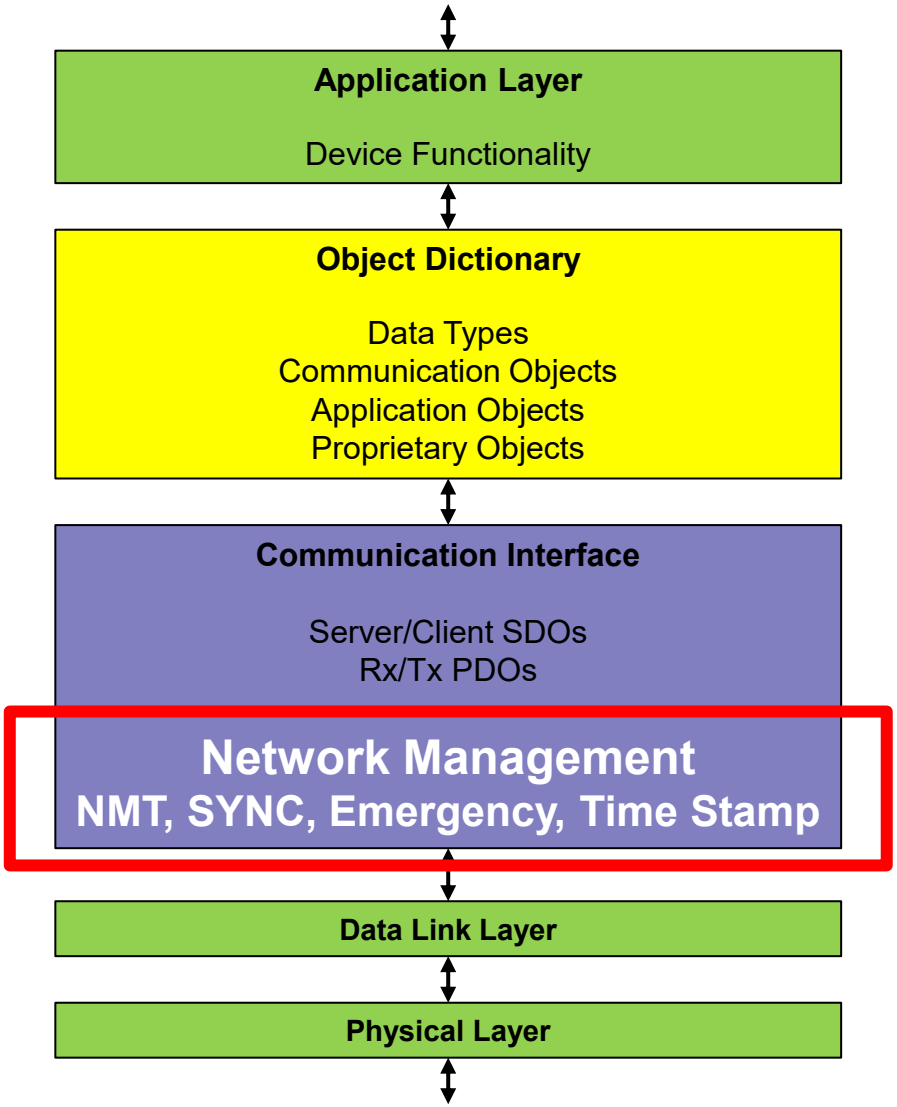
- **Synchronized or grouped polling**



zdroj: [1]

5. Síťový management

Služby síťového managementu



zdroj: [2]

Služby síťového managementu

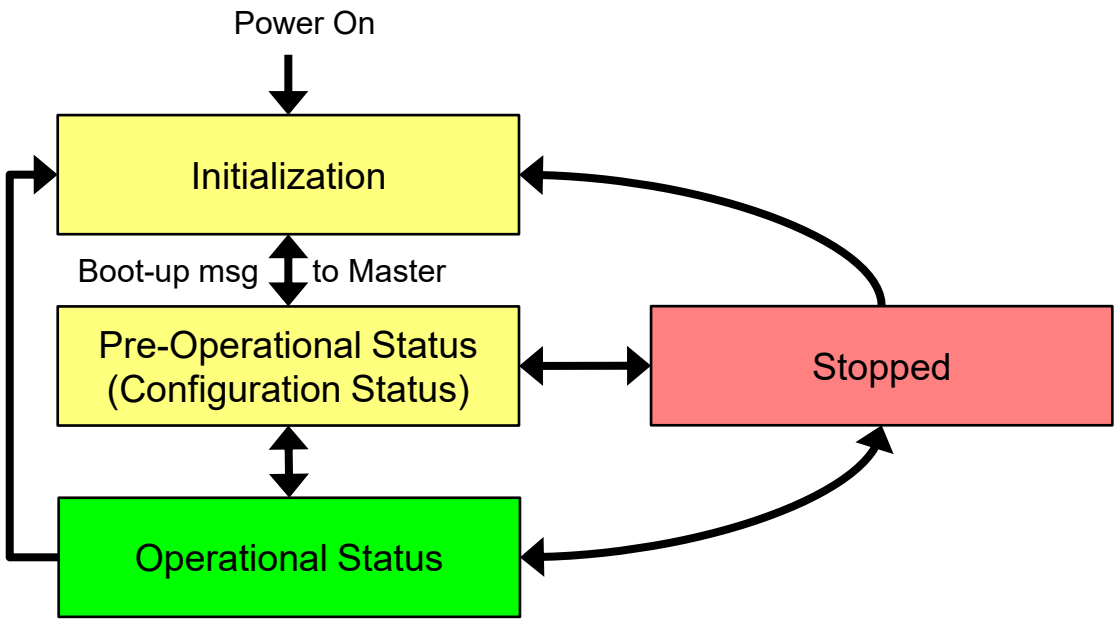
- Network management (NMT) je založen na principu Master/Slave komunikace
  - NMT Master poveluje ostatní Slave zařízení.

- Úkoly NMT Mastera
  - Řídí spuštění sítě.
  - Download konfigurace do Slave.
  - Hlídání stavu Slave.

- NMT služby
  - **NMT module control** – řízení stavu Slave.
  - **SYNC** – synchronizace PDO mezi více Slave.
  - **Time Stamp** – synchronizace reálného času na síti.
  - **Node Guarding/Heartbeat** – hlídání stavu Slave.
  - **Emergency** – zpráva o chybě Slave.

Stavy Slave zařízení

- Slave se může nacházet v jednom z několika stavů
- NMT Master může přepínat Slave zařízení do těchto stavů:
  - Pre-operational
  - Operational
  - Stopped



- Aktivita Slave:
  - *Power-On Reset* → *Initilization* → *Pre-Operational* → odeslání *Boot-Up message* do Mastera a čeká na povel z Mastera, který dovolí přechod do stavu *Operational*

zdroj: [2]

Stavy Slave zařízení

- Na stavu Slave závisí možnosti komunikace.
- Ne vždy je povolena veškerá komunikace.

	Initializing	Pre-operational	Operational	Stopped
Boot-up	X			
SDO		X	X	
Emergency		X	X	
SYNC/TIME		X	X	
Nodeguard/Heartbeat		X	X	X
PDO			X	

- Pouze ve stavu **Operational** zařízení plně vykonává své funkce.

NMT module control message

- Zpráva, kterou Master řídí stav Slave
  - CAN-ID (COB-ID) = 000h (zpráva s nejvyšší prioritou)

NMT-Master ⇒ NMT-Slave(s)

COB-ID	Byte 0	Byte 1
000h	CS	Node-ID

Zpráva může být adresována pouze jednomu Slave (Node-ID je adresa daného Slave) nebo všem Slave (Node-ID = 0)

Command Specifier	NMT Service
1	Start Remote Node
2	Stop Remote Node
128	Enter Pre-Operational State
129	Reset Node
130	Reset Communication

zdroj: [3]

Hlídání stavu Slave

- **Nodeguarding**
  - NMT Master se dotazuje každého Slave na jejich aktuální stav.
  - Pokud Slave neodpoví do stanoveného času, je považován za vadný a Master rozhodne co dál (např. se pokusí znovu inicializovat Slave nebo zastaví celý systém apod.).
  - Kontrola funguje i opačně – Slave může kontrolovat, zda se ho Master pravidelně dotazuje (tzv. **Lifeguarding**).
  - Zpráva z Mastera do Slave (**Remote Frame**, bit RTR = 1):

NMT-Master ⇒ NMT-Slave

COB-ID
700h + Node-ID

  - Odpověď od Slave:

NMT-Master ⇐ NMT-Slave

COB-ID	Byte 0
700h + Node-ID	bit 7: *toggle*, bit 6-0: *state*

state	Meaning
0	Boot-up
4	Stopped
5	Operational
127	Pre-Operational

zdroj: [3]

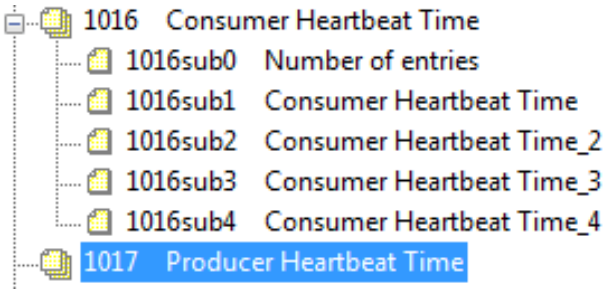
Hlídání stavu Slave

- **Heartbeat**
  - Každý Slave periodicky posílá svůj aktuální stav.
  - Perioda je nastavitelná (**Producer HeartBeat Time** [ms]).
  - Výhodnější oproti **Nodeguarding**
    - Poloviční zatížení sběrnice (odpadá dotazování).
    - Navíc Slave může sledovat i **Heartbeat**y okolních Slave – typicky svých přímých komunikačních partnerů (**Consumer Heartbeat Time**).
  - Slave implementuje buď **Nodeguarding** nebo **Heartbeat**, ne oboje.
  - **Struktura Heartbeat zprávy**:

HEARTBEAT Producer ⇒ Consumer(s)

COB-ID	Byte 0
700h + Node-ID	*state*

state	Meaning
0	Boot-up
4	Stopped
5	Operational
127	Pre-Operational



zdroj: [3]

Emergency object

- **Emergency zpráva (EMCY)**
  - Zařízení jí odesílá pokud detekuje interní chybu.
  - Zpráva má vysokou prioritu (nízké CAN ID).
  - Zpráva je odeslána pouze jednou při prvním výskytu dané chyby (uvažuje se, že chyba přetrvává), při odeznění chyby je EMCY zpráva odeslána znovu, ale s příznakem resetu chyby.
  - Např. při detekci napětí na AI vstupu mimo povolený rozsah je odeslána EMCY zpráva, při návratu napětí zpět do pracovního rozsahu je odeslána EMCY zpráva o odeznění chyby.
  - Chyby se zároveň ukládají do **Error Register** [1001h,00h] ve slovníku objektů (8 bitů, jejich význam viz tabulka).
  - **Struktura EMCY zprávy:**

Emergency sender ⇒ Emergency receiver(s)

COB-ID	Byte 0-1	Byte 2	Byte 3-7
80h + Node-ID	**Emergency Error Code**	**Error Register** (Object 1001h)	Manufacturer specific error field

Bit	Error type
0	Generic
1	Current
2	Voltage
3	Temperature
4	Communication
5	Device profile specific
6	Reserved (=0)
7	Manufacturer specific

zdroj: [3]

Emergency object

- **Emergency Error Code**
  - Standardizované kódy chyb dle DS-301 a profilu zařízení DS-4xx

Emergency sender ⇒ Emergency receiver(s)

COB-ID	Byte 0-1	Byte 2	Byte 3-7
80h + Node-ID	**Emergency Error Code**	**Error Register** (Object 1001h)	Manufacturer specific error field

Emergency Error Code	Meaning
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	current, device input side
22xx	current, inside the device
23xx	current, device output side
30xx	Voltage
31xx	mains voltage
32xx	voltage inside the device
33xx	output voltage
40xx	Temperature
41xx	ambient temperature
42xx	device temperature
50xx	Device hardware
60xx	Device software



Supported EMCY Codes

EMCY Error Code (hex)	Meaning
0000	Error Reset or No Error
2320	Short Circuit at Output
3300	Open Circuit at Output
8130	Life Guard or Heartbeat Error
FF00	Out of Range Low at Input
FF01	Out of Range High at Input
FF02	Error at Digital Output (DOUT only)

xx – dle profilu zařízení

zdroj: [3],[7]

Emergency object

Emergency sender ⇒ Emergency receiver(s)

COB-ID	Byte 0-1	Byte 2	Byte 3-7
80h + Node-ID	**Emergency Error Code**	**Error Register** (Object 1001h)	Manufacturer specific error field

- **Manufacturer specific error field**
  - Kódy chyb definované výrobcem zařízení
  - Např.: Quad Controller TDAX020506 fy Axiomatic



Supported Error Descriptions

Description (hex)	Meaning
01	Fault at an analog input
02	Fault at a digital output
04	Fault at a proportional output
08	Lost Heartbeat or Life Guard from a node

Supported Channel Numbers

Description (hex)	Meaning
01	AIN1 or POUT1 (depending on description)
02	AIN2 or POUT2 (depending on description)
03	AIN3 or POUT3 (depending on description)
04	AIN4 or POUT4 (depending on description)
05	FIN1 or DOUT (depending on description)
06	FIN2

16-bit kód upřesňující chybu

For example, the Quad detects a short circuit on proportional output channel 3.

EMCY Code = 0x2320 current, device output side
Additional Information = 0x0403 Fault at proportional output, POUT3
Resulting Code in 1003h = 0x04032320

MSB			LSB
Error Description	Channel	EMCY Error Code	
x04	0x03	0x2320	

zdroj: [7]

Profil zařízení DS-401

- **DS-401 Generic IO modules**
  - Moduly s DI, AI vstupy a DO, AO výstupy.

RRC-ION/001 je jednotka vzdálených V/V s protokolem CANopen DS 401. Jednotka je konstruována a certifikována pro použití na kolejových vozidlech.

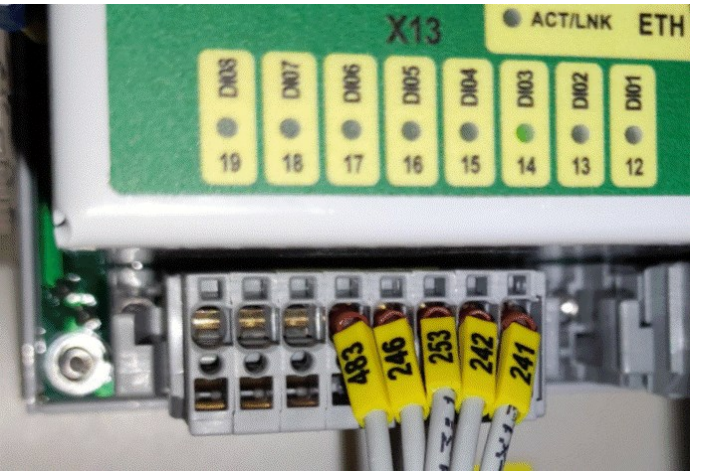
Základní vlastnosti

- 12 číslicových vstupů bez GO
- 8 číslicových výstupů bez GO
  - samostatná napájecí svorka pro každý výstup
  - odolnost vůči zkratu
  - odolnost vůči přepólování
- 4 analogové výstupy bez GO
  - proudový rozsah 4 mA až 20 mA
  - externí zátěž k zemi
- 1 skrytý analogový vstup
  - měření napájecího napětí
- 1 rozhraní CAN s GO
  - zřetězení konektorů
- Napájecí napětí 24 V ss., -30 % až +25 %
- Montáž na DIN lištu 35 mm
- Konstrukce dle ČSN EN 50155 ed. 3



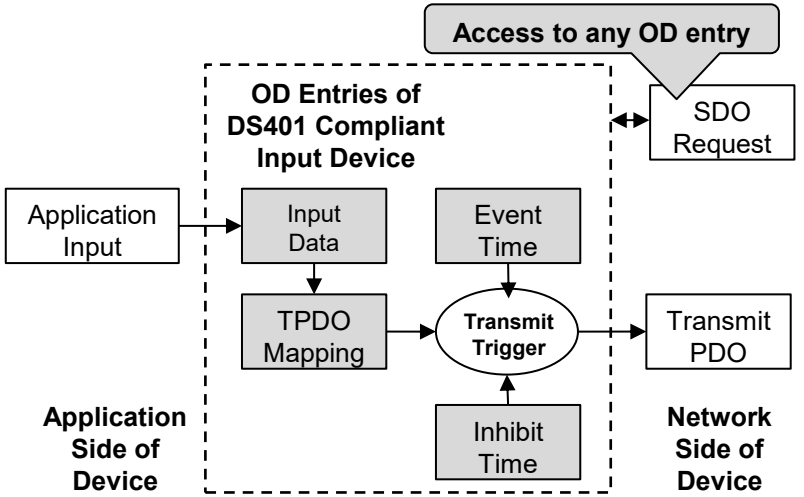
zdroj: [5]

Profil zařízení DS-401

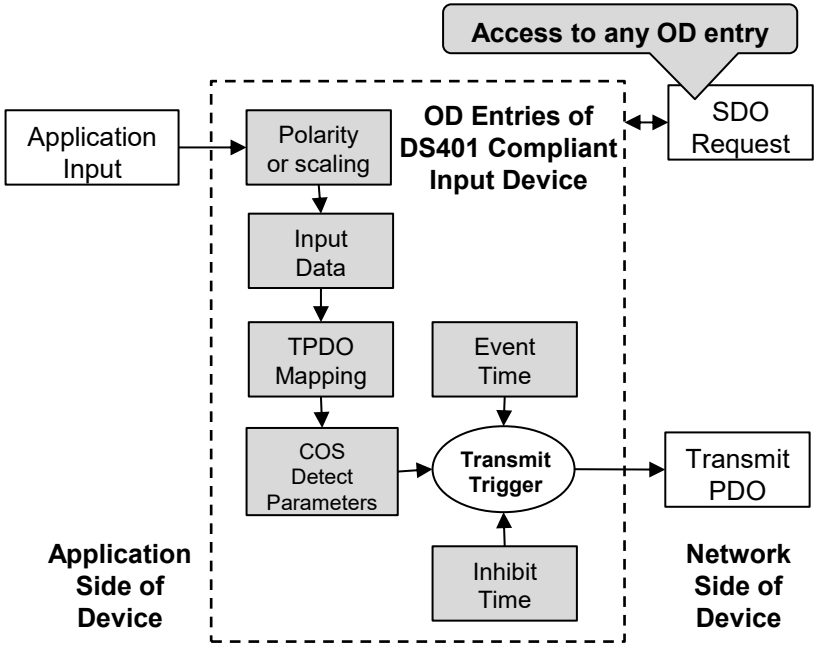


- **Zpracování signálů z DI nebo AI**
  - Relevantní položky slovníku objektů, které ovlivňují zpracování signálů z DI nebo AI.

Jednodušší náhled



Detailnější náhled

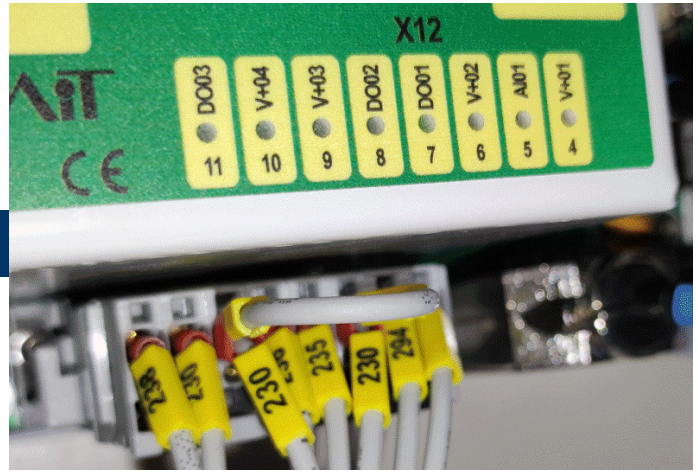


DI – digital input (digitální vstup)
AI – analog input (analogový vstup)
OD – Object Dictionary (slovník objektů)

COS = Change Of State

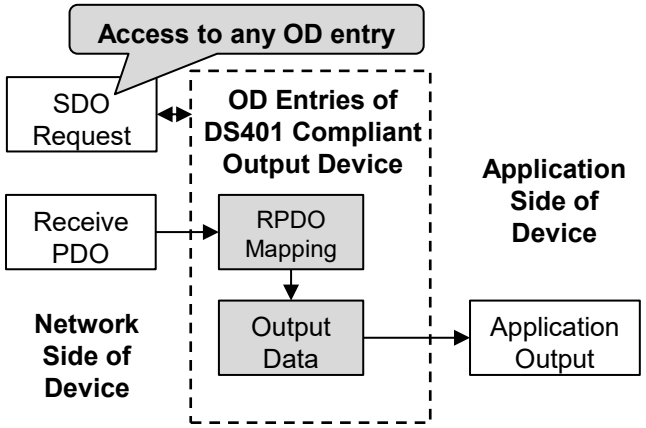
zdroj: [1]

Profil zařízení DS-401

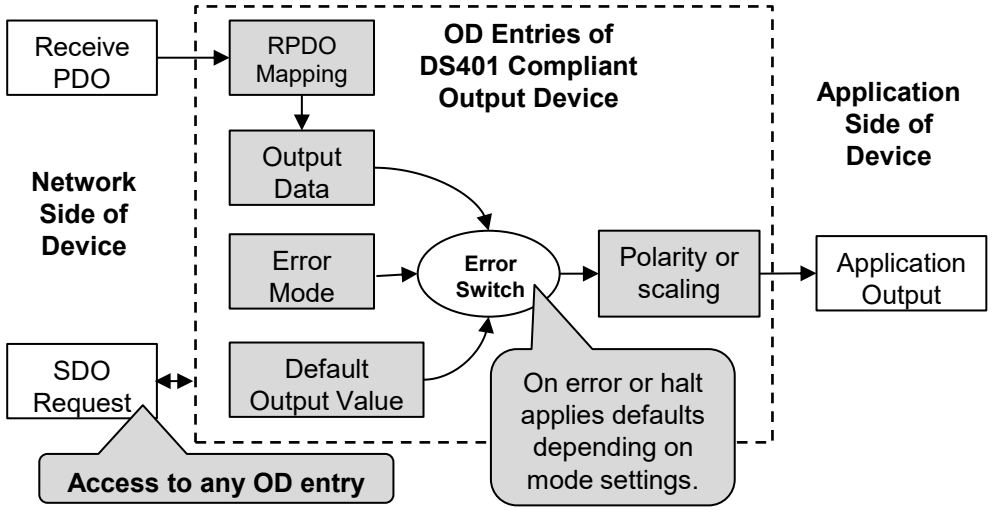


- **Nastavení DO nebo AO**
  - Relevantní položky slovníku objektů, které ovlivňují nastavení hodnoty na DO nebo AO

Jednodušší náhled



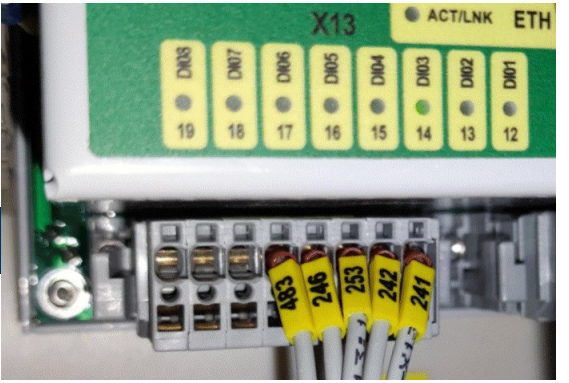
Detailnější náhled



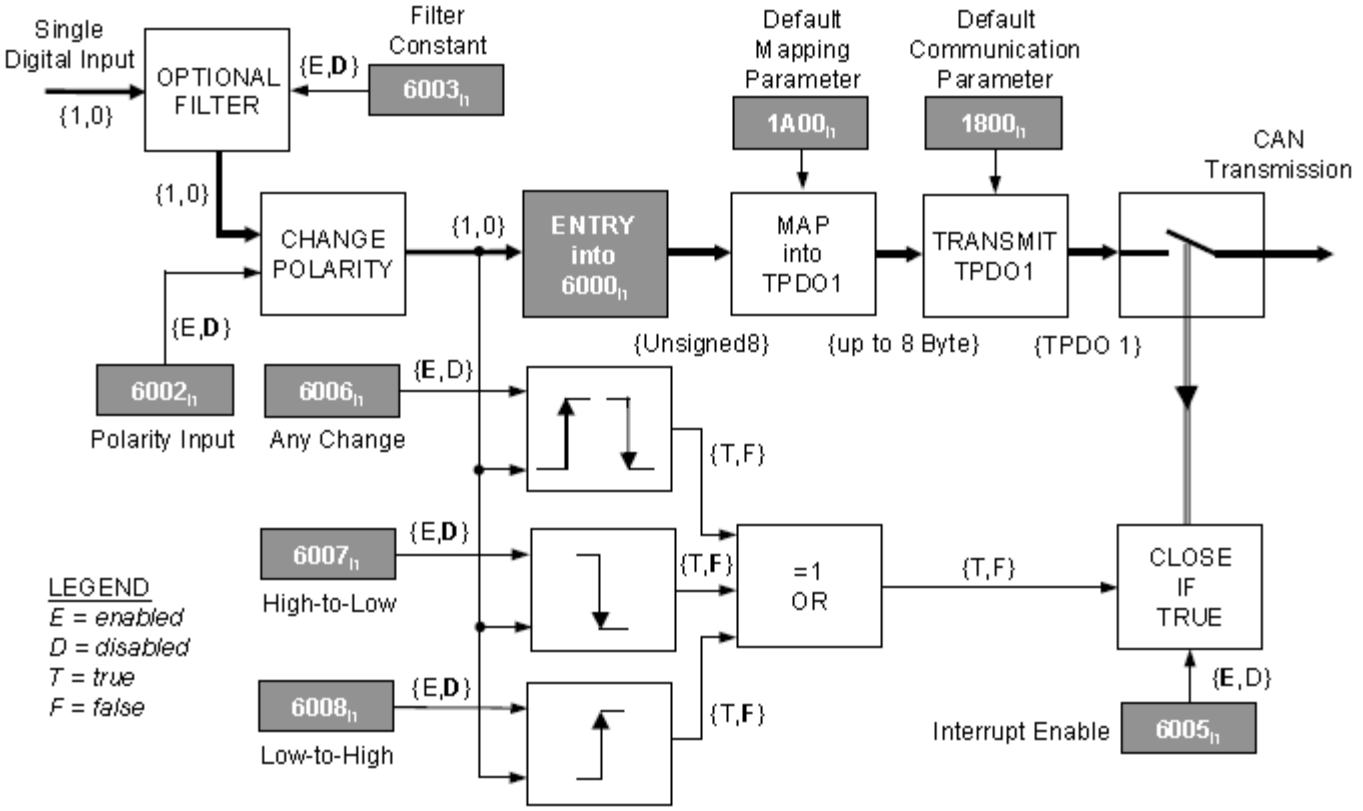
DO – digital output (digitální výstup)
AO – analog output (analogový výstup)
OD – Object Dictionary (slovník objektů)

zdroj: [1]

Profil zařízení DS-401



- Podrobnější blokové schéma zpracování DI



zdroj: [7]

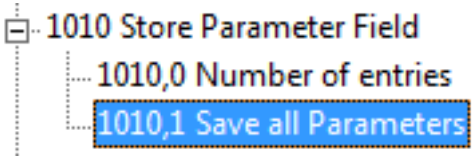
6. Konfigurace zařízení

Konfigurace zařízení

- **Nastavení slovníku objektů u Slave**
  - **Automatická konfigurace Slave po zapnutí napájení**
    - Nastavení slovníku každého Slave je uloženo v Masteru, po zapnutí napájení Master pomocí SDO zpráv nastaví slovník objektů každého Slave.
    - **Výhoda**: Nastavení slovníku Slave je automatické – snadné přidání nového zařízení do sítě nebo výměna zařízení.

NEBO

  - **Slave je nakonfigurován již předem** před přidáním do sítě, nastavení má uloženo v nonvolatilní paměti (EEPROM), nastavení si nahraje po každém startu sám.
    - Slovník Slave je nakonfigurován pomocí externího nástroje (např. SW **ProCANopen** fy Vector) a odeslán pomocí SDO zpráv do Slave. Následně je odeslán pokyn „**Store Parameters**“ pro uložení nastavení do nonvolatilní paměti.
    - **Nevýhoda**: Každý Slave musí být ručně nakonfigurován před přidáním do sítě.



Konfigurace zařízení

- **Uložení nastavení slovníku objektů zařízení**
  - V **EDS** a **DCF** souborech.
  - Textové soubory (podobnost s *.ini* soubory)
  - Popisují slovník objektů daného zařízení
    - Jaké položky jsou dostupné.
    - Jaké mají nastavení.
  - Oba soubory mají shodný formát, ale odlišné použití.
  - **EDS** (**Electronic Data Sheet**) – od výrobce zařízení (defaultní nastavení zařízení).
  - **DCF** (**Device Configuration File**) – modifikovaný EDS dle potřeb konkrétního zařízení.
  - Konfiguraci lze pomocí SW nástroje odeslat ze souboru do zařízení nebo jí načíst a uložit do souboru.

```
PDOMapping=0

[1018]
SubNumber=5
ParameterName=Identity
ObjectType=9

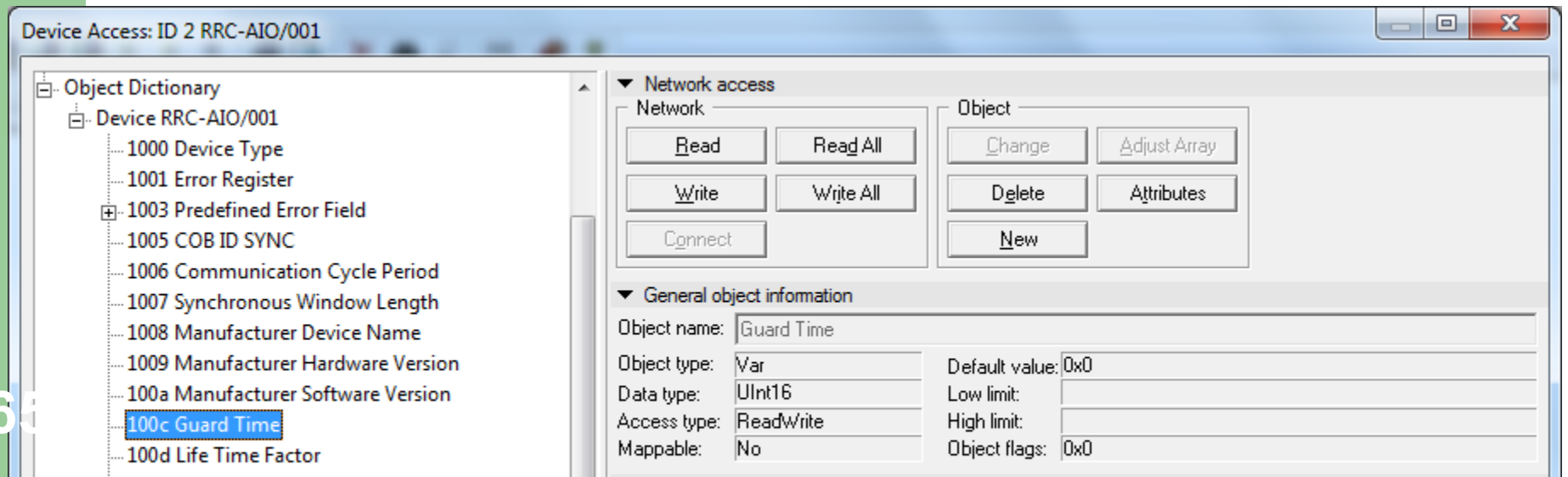
[1018sub0]
ParameterName=Number of Entries
ObjectType=7
DataType=0x0005
AccessType=ro
DefaultValue=4
PDOMapping=0

[1018sub1]
ParameterName=Vendor ID
ObjectType=7
DataType=0x0007
AccessType=ro
DefaultValue=0x00000055
PDOMapping=0

[1018sub2]
ParameterName=Product Code
ObjectType=7
DataType=0x0007
AccessType=ro
DefaultValue=0xAA020506
PDOMapping=0
```

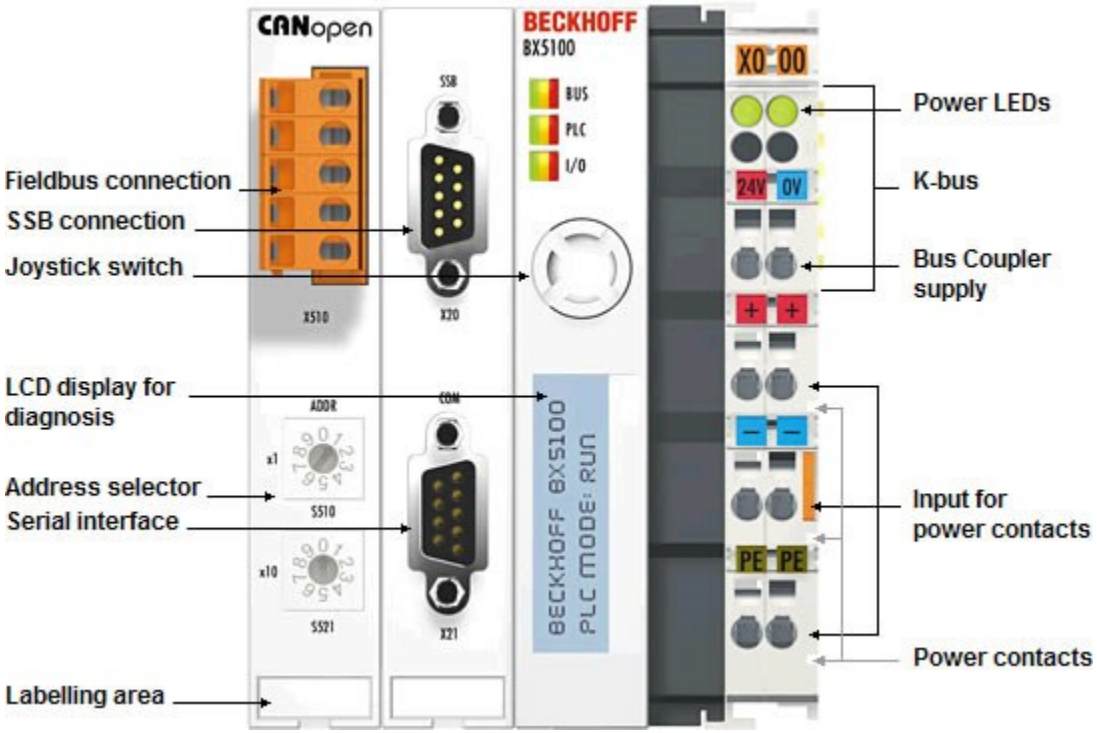
Konfigurace zařízení

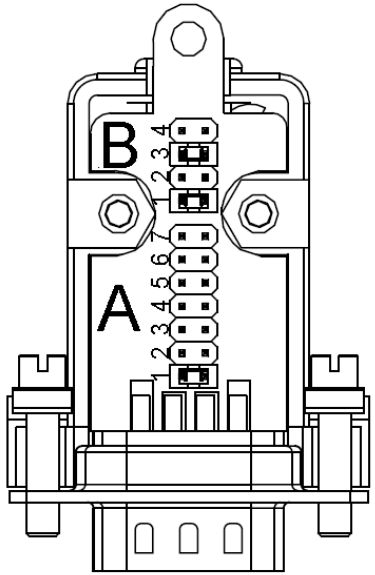
- Manipulace s těmito soubory
  - Ruční editace – nepraktické.
  - Pomocí SW nástroje pro konfiguraci zařízení, konfiguraci sítě.
    - Editace položek.
    - Vytváření/mazání položek.
    - Sestavení EDS/DCF souboru vyčtením slovníku z neznámého zařízení, ke kterému chybí EDS nebo DCF soubor.
- Použití souborů
  - CANopen konfigurační nástroje – zavedení obsahu slovníku do zařízení, konfigurace sítě.
  - CANopen analyzátory – přiřazení významu zachycených zpráv a jejich přenášených dat (asociace položek a názvu symbolů ve slovníku s daty ve zprávách).



Konfigurace zařízení

- **Nastavení adresy a přenosové rychlosti**
  - Každý uzel musí mít unikátní NODE-ID a všechny uzly stejnou přenosovou rychlost (Baud Rate).
  - **Mechanicky**
    - Pomocí jumperů, DIP switchů, rotačních přepínačů apod. na každém zařízení.





zdroj: [4],[5]

Konfigurace zařízení

- **Nastavení adresy a přenosové rychlosti**
  - **Softwarově**
    - Přes CANopen sběrnici pomocí LSS protokolu
    - **LSS** = **Layer Setting Services** (DS-306)
    - Jak ?
      - Spojení 1:1 mezi nástrojem a zařízením
      - Odeslání několika zpráv do zařízení
      - Formát:
        - Nástroj → Zařízení: COB-ID 0x7E5, délka dat 8 B
        - Zařízení → Nástroj: COB-ID 0x7E4, délka dat 8 B
        - Nepoužité datové bajty mají vždy hodnotu 0
      - Zařízení potvrzuje přijímané příkazy



zdroj: [4]

Př.: nastavení adresy

	ID [Hex]	Data [0, 1, 2, 3, 4, 5, 6, 7]
1.	7E5	04 01 00 00 00 00 00 00
2.	7E5	5E 00 00 00 00 00 00 00
3.	7E5	11 01 00 00 00 00 00 00
4.	7E5	17 00 00 00 00 00 00 00
5.	7E5	04 00 00 00 00 00 00 00

04, 01 – přepnutí zařízení do režimu konfigurace

5E - zjištění aktuálního NODE-ID

11 – příkaz „Nastav NODE-ID“
01 – nové NODE-ID

17 – „Ulož nastavení“

04, 00 - přepnutí zařízení zpět do normálního režimu

Význam kontrolek na zařízení

- **Význam kontrolek je uveden v dokumentu DS-303**
- **Error LED**
  - Indikace stavu fyzické vrstvy (Active, Passive, Bus-off) a chybějících zpráv (SYNC, Guard nebo Heartbeat).
- **Run LED**
  - Indikace stavu, ve kterém se zařízení nachází (Pre-operational, Operational, Stopped).

CAN Run LED	State	Description	Category
Flickering	AutoBitrate/LSS	The auto-bitrate detection is in progress or LSS services are in progress (alternately flickering with error LED)	Optional
Blinking	PRE-OPERATIONAL	The device is in state PRE-OPERATIONAL	Mandatory
Single flash	STOPPED	T	
Double flash		F	
Triple flash	Program/ Firmware download	A d	
On	OPERATIONAL	T	

ERR LED	State	Description	Category
Off	No error	The device is in working condition	Mandatory
Flickering	AutoBitrate/ LSS	The auto-bitrate detection is in progress or LSS services are in progress (alternately flickering with run LED)[1]	Optional
Blinking	Invalid Configuration	General configuration error	Optional
Single flash	Warning limit reached	At least one of the error counters of the CAN controller has reached or exceeded the warning level (too	Mandatory

zdroj: [6]

KONEC

Děkuji za pozornost

Literatura

[1] Pfeiffer, O., Ayre, A., Keydel, Ch. *Embedded Networking with CAN and CANopen*. Copperhill Technologies Corp. 2008.

[2] Voss, W. *CANopen: Higher Level Protocol based ond Controller Area Network (CAN)*. ESD electronics, Inc. Power Point prezentace.

[3] Boterenbrood, H. *CANopen: high-level protocol for CAN-bus*. NIKHEF Amsterdam. 2000.

[4] www.google.com

[5] www.amit.cz

[6] www.can-cia.org

[7] AXIOMATIC. *Quad Controller with CANopen: user manual*. January 07, 2011. Version 2.00

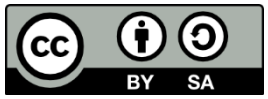
[8] Video seminář CANopen Basics

[9] https://www.csselectronics.com/pages/canopen-tutorial-simple-intro

UNIVERZITA PARDUBICE
DOPRAVNÍ FAKULTA JANA PERNERA

Vytvořeno v rámci projektu **Green Deal UPCE**, reg. č. NPO_UPCE_MSMT-2142/2024-4.

Toto dílo podléhá licenci Creative Commons BY-SA 4.0. Pro zobrazení licenčních podmínek navštivte https://creativecommons.org/licenses/by-sa/4.0/.



NÁRODNÍ PLÁN OBNOVY

Financováno Evropskou unií
NextGenerationEU

Univerzita Pardubice
Dopravní fakulta Jana Pernera

Kurz celoživotního vzdělávání

Elektrotechnické systémy pro environmentálně příznivou dopravu

Předmět

Pokročilé systémy digitálního řízení a komunikace v dopravní technice

Studijní materiál

Komunikační protokol SAE J1939

Ing. Zdeněk Mašek, Ph.D.



Obsah přednášky

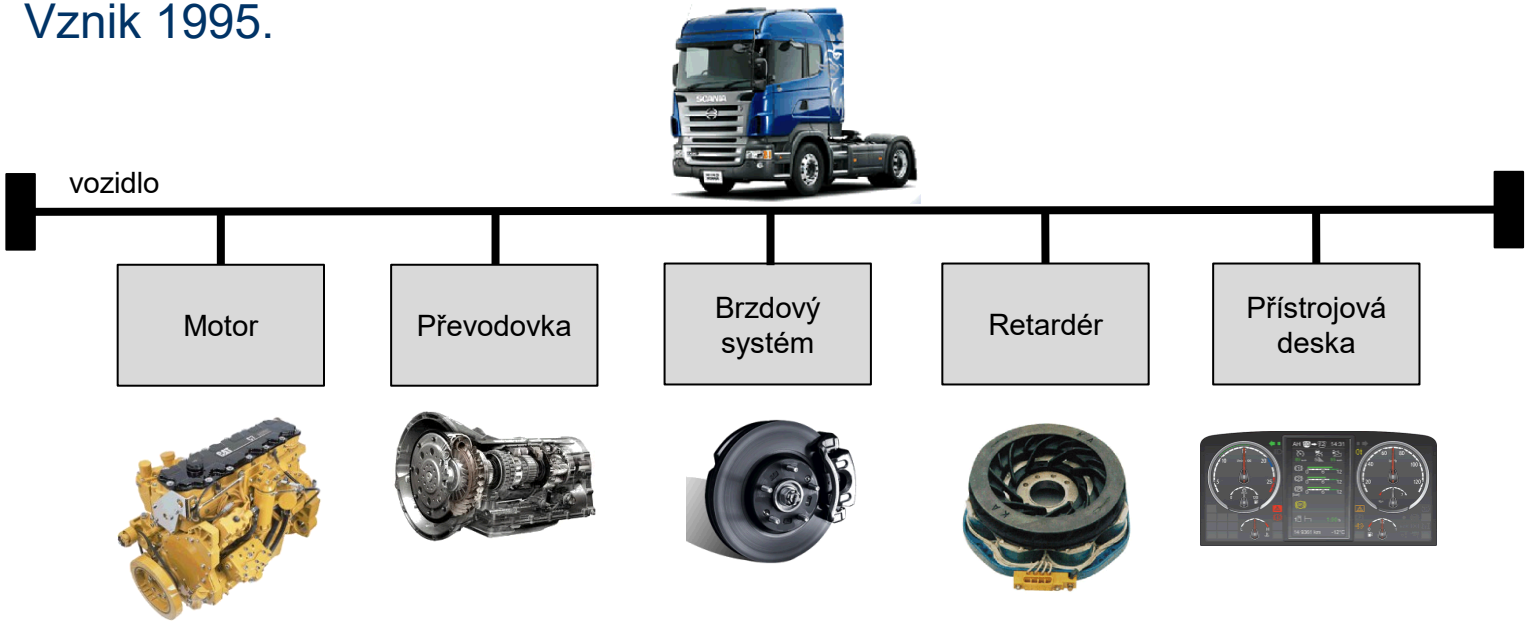
1. Základní vlastnosti a použití
2. Fyzická vrstva
3. Aplikační vrstva - PGN a SPN
4. Linková vrstva - formát zprávy
5. Transportní protokol
6. Síťový management
7. Diagnostika
8. Vybrané PGN vysílané spal. motorem
9. Nástroje pro analýzu komunikace

Počet snímků: 81

1. Základní vlastnosti a použití

Co je SAE J1939 ?

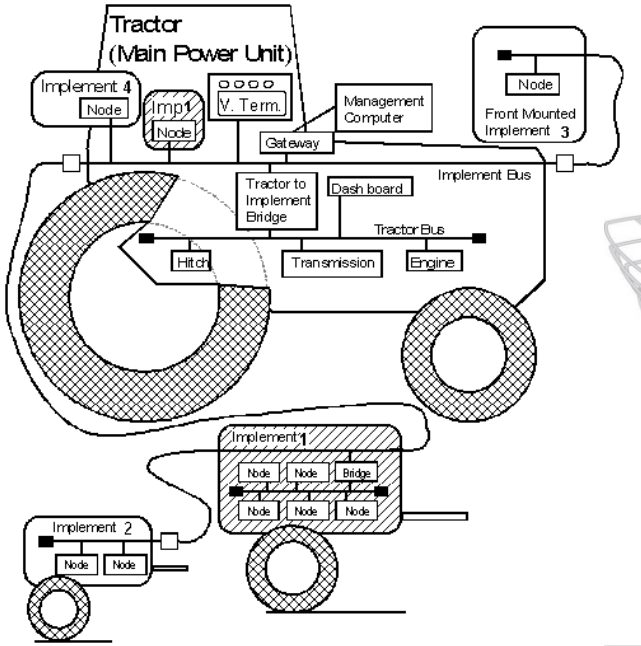
- Komunikační protokol na založený na sběrnici CAN.
- Sériová komunikace mezi ECU v „heavy-duty“ vozidlech (nákl. vozidla, autobusy, zemědělské stroje, stavební stroje, …) i stacionárních strojích.
- Standardizovaný význam dat napříč různými výrobci.
- Sjednocené diagnostické rozhraní (diagnostický protokol).
- Plug and Play připojení ECU do sítě.
- Vyvinut a standardizován skupinou SAE (= dokumentace není zdarma).
- Vznik 1995.

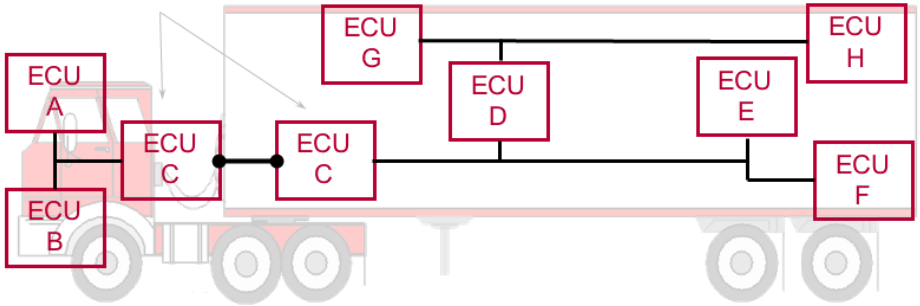


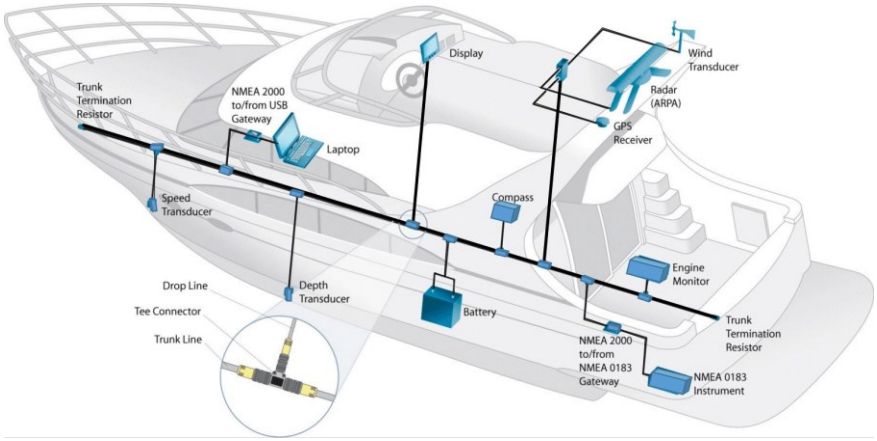
zdroj: [11]

Použití SAE J1939

- Komunikace uvnitř vozidla – nákl. automobily, autobusy apod.
- Mezivozová komunikace – např. tahač + návěs
- Lokomotivy se vznětovými motory
- Zemědělské, stavební, lesní stroje (ISOBUS)
- Vojenská vozidla (MiLCAN)
- Lodě (NMEA2000)
- Stacionární stroje (generátory)





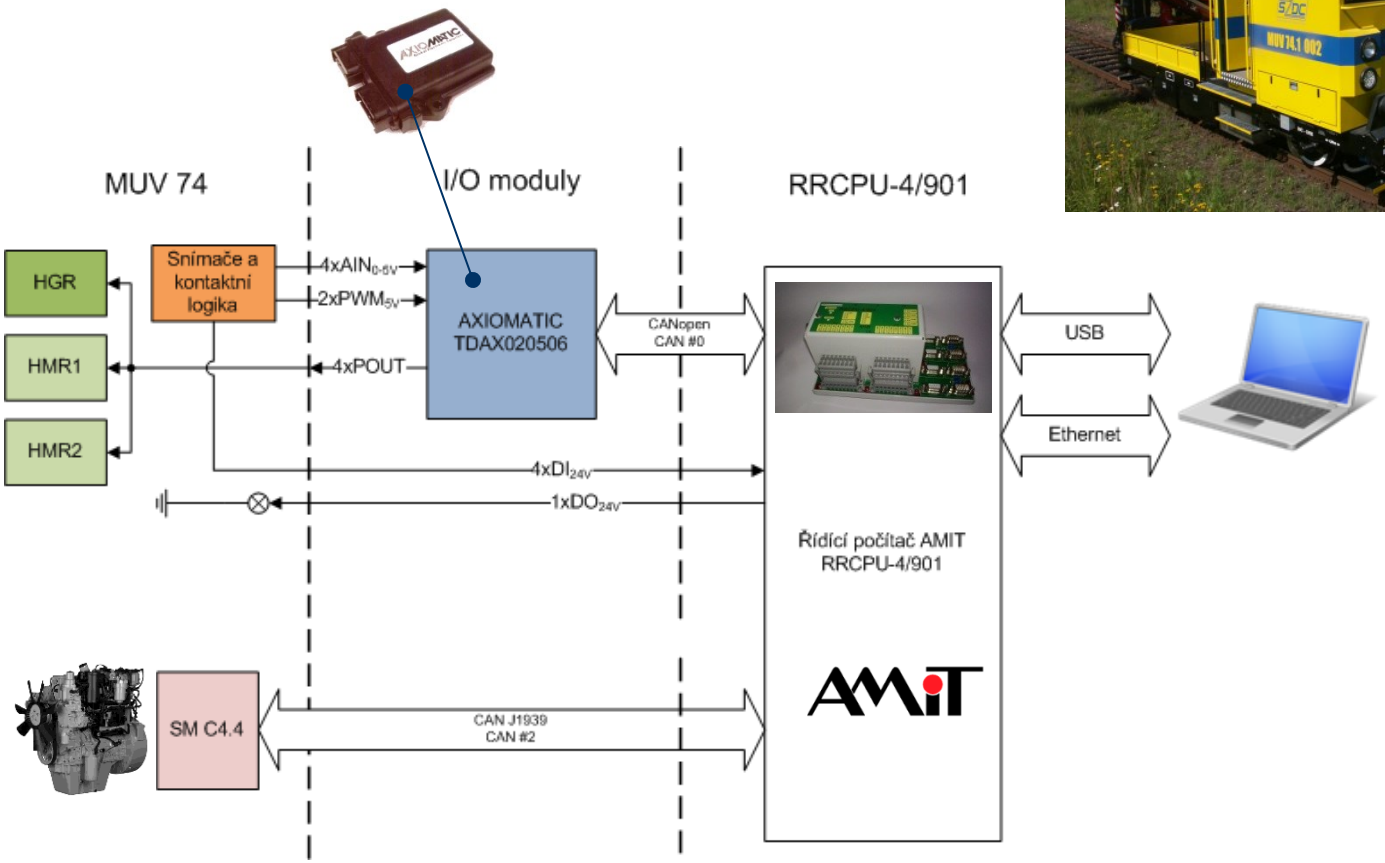


zdroj: [11]

Použití J1939

- Řídící systém speciálního drážního vozidla
  - Kombinace protokolu J1939 a CANopen



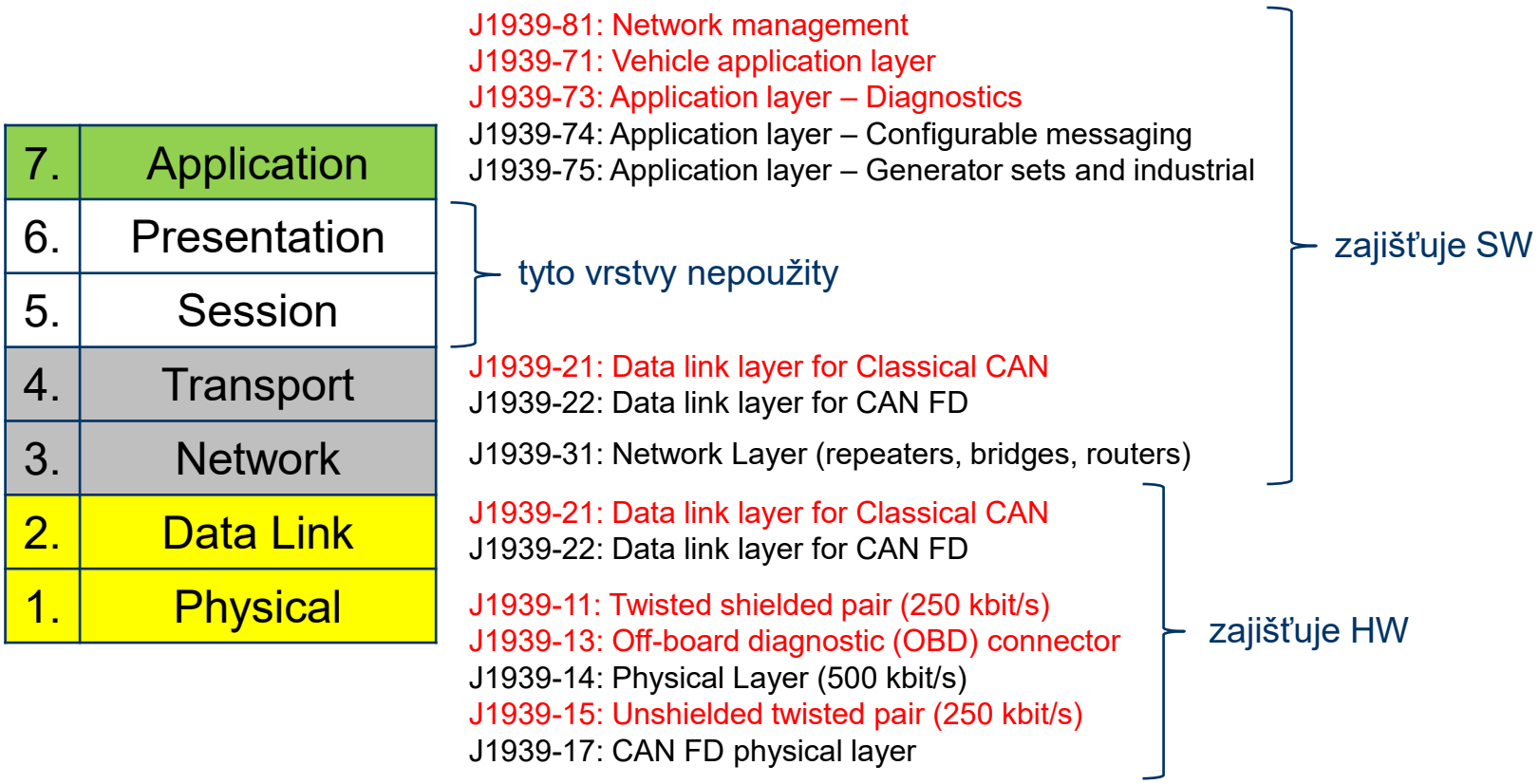


Další vlastnosti SAE J1939

- Komunikace typu Multi-Master (Master/Slave nebo Client/Server není podporováno) – všechny ECU jsou si rovnocenné, ECU vysílá kdy chce.
- Rozdělení dat do více zpráv pokud se data nevejdou do jedné CAN zprávy, max. 1785 B dat u **CAN Classic** a 15 kB až 16 MB dat u **CAN FD**.
- Plug and Play připojování ECU do sítě za chodu, automatické přiřazení adresy ECU při startu sítě.
- Standard definuje:
  - Aplikační použití
  - Způsob komunikace mezi ECU
  - Formát dat
  - Sjednocené diagnostické rozhraní
- Používá **29-bit identifikátor zpráv** (CAN 2.0B).
- Komunikační rychlost **250 kbit** nebo **500 kbit** u **CAN Classic** nebo **500/2000 kbit** u **CAN FD**.
- Přenos signálů v tzv. skupinách parametrů (**PGN** – Parameter Group Number).
- Signály standardizovány, vedeny pod čísly **SPN** (Suspect Param. Num.).

Vrstvy protokolu SAE J1939

- Vrstvový model komunikace J1939 je založen na OSI modelu.
- Standard J1939 je rozdělen do řady dokumentů - číslování dokumentů cca odpovídá číslování vrstev v OSI modelu.



2. Fyzická vrstva

Fyzická vrstva

- Vychází z ISO 11898-2 (CAN High Speed)
- Přenosové médium
  - Stíněná kroucená dvoulinka STP (J1939-11)
  - Nestíněná kroucená dvoulinka UTP (J1939-15)

J1939 specifikace pro vodiče			
	min.	jmen.	max.
Impedance (Ω)	108	120	132
Kapacita mezi vodiči (pF/m)	0	40	75
Kapacita mezi vodiči a stíněním (pF/m)	0	70	110
Zpoždění linky (ns/m)		5	

STP



UTP



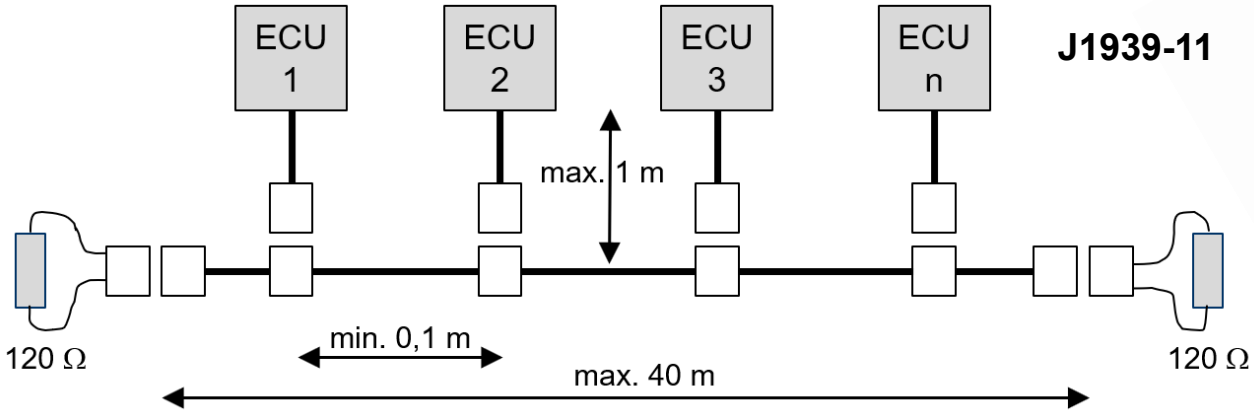
- Přenosová rychlost, počet ECU a délka sběrnice dle verze fyz. vrstvy:

	J1939-11	J1939-15	J1939-14
Kroucená dvoulinka	Stíněná	Nestíněná	Stíněná nebo nestíněná
Přenosová rychlost	250 kbit/s	250 kbit/s	500 kbit/s
Max. počet ECU	30	10	30
Délka sběrnice	Max. 40 m	Max. 40 m	Max. 40,0 m – 56,4 m

zdroj: [11]

Fyzická vrstva

- Topologie s páteřní linkou a krátkými odbočkami.
- Pro omezení odrazů signálu na vedení.
- Stínění STP kabelu má být ukostřeno pouze v jednom bodě.



J1939-11

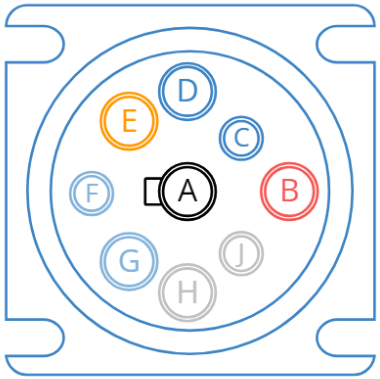


	J1939-11	J1939-15	J1939-14
Kroucená dvoulinka	Stíněná	Nestíněná	Stíněná nebo nestíněná
Přenosová rychlost	250 kbit/s	250 kbit/s	500 kbit/s
Max. počet ECU na sběrnici	30	10	30
Max. délka sběrnice	40 m	40 m	40 až 56,4 m
Délka odbočky	Max. 1 m	Max. 3 m	Max. 1,67 m
Vzdálenost mezi odbočkami	Min. 0,1 m	Min. 0,1 m	Min. 0,3 m
Umístění zakončovacích rezistorů 120 Ω	mimo ECU	mimo ECU nebo v ECU	mimo ECU nebo v ECU

zdroj: [11]

Fyzická vrstva

- Diagnostický konektor (dle SAE J1939/13)
  - 9-pin kulatý konektor Deutsch HD10-9-1939
  - Na voze je zásuvka (male)
  - Type 1
    - černý, starší (250 kbit/s)
    - nelze zastrčit do zásuvky Type 2
  - Type 2
    - zelený, novější (500 kbit/s)
    - podpora SAE J1939/14 (500 kbit/s)
    - lze zastrčit do zásuvky Type 1



A	Ground
B	Battery power
C	CAN 1 H
D	CAN 1 L
E	CAN shield
F	J1708 (+) / CAN 2 H
G	J1708 (-) / CAN 2 L
H	OEM specific
J	OEM specific

Type 1 (black): CAN 1 = 250K
Type 2 (green): CAN 1 = 500K



zdroj: [11,14]

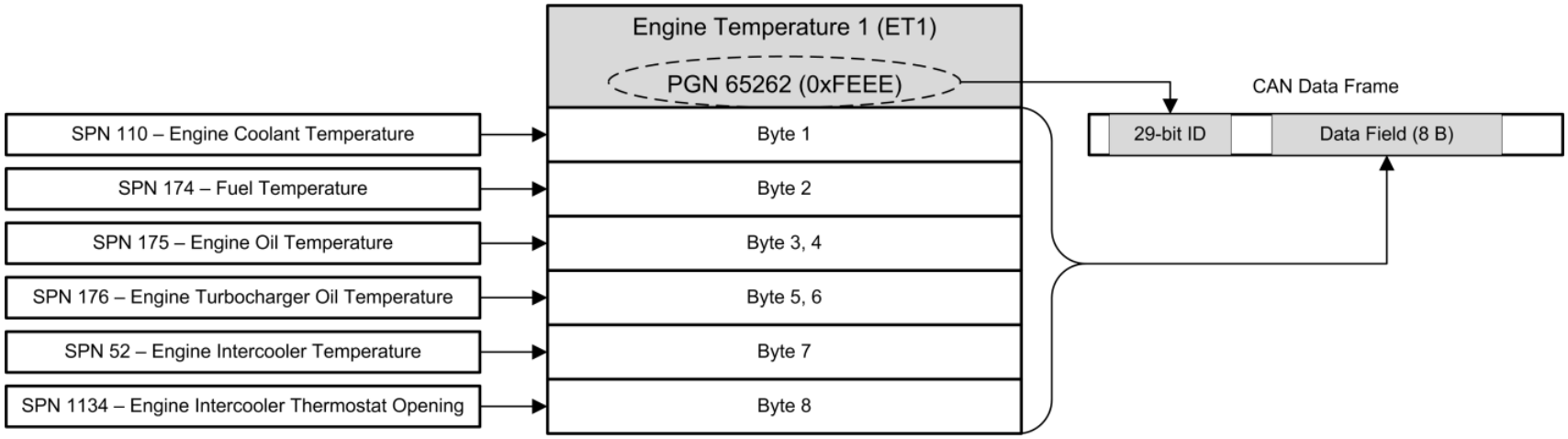
3. Aplikační vrstva

Aplikační vrstva (J1939/71)

- **J1939-71 Vehicle application layer**
  - Definuje signály přenášené v CAN zprávách
  - Každý přenášený signál má definovány všechny potřebné atributy
    - **SPN** = Suspect Parameter Number (ID signálu dohledatelné v normách J1939-xx)
    - Délka (v bitech)
    - Typ veličiny (stavová/měřená)
    - Fyzické rozlišení (způsob přepočtu na fyzikální hodnotu)
    - Rozsah platnosti (min. a max. hodnota)
  - Signály jsou sdruženy do skupin (**PGN** = Parameter Group Number)
    - Pro lepší využití přenosové kapacity sběrnice.
    - V jedné zprávě jsou veličiny jednoho typu, např. teploty, tlaky nebo mohou být namixovány, např. otáčky, zatížení, poloha plyn. pedálu nebo veličiny spjaté s převodovkou apod…
    - Pod jedním PGN se skrývá několik SPN.
    - Seznamy PGN lze nalézt v normách J1939-xx.
    - **PGN je 18-bit číslo, které je součástí 29-bit identifikátoru CAN zprávy**.

Aplikační vrstva (J1939/71)

- **SPN a PGN**



- **Pořadí bajtů (byte order) vícebajtových hodnot je Little-endian**
  - Nejméně významný byte se ve zprávě přenáší jako první v pořadí (Intel format).

	Data Field (8 B)							
Byte č.	1	2	3	4	5	6	7	8
Hodnota			0x34	0x12				

16-bit číslo přenášené ve zprávě má hodnotu: 0x1234 (4660 v desítkové soustavě)

Aplikační vrstva (J1939/71)

- **Ukázka PGN**

PGN 65262 ***Engine Temperature 1*** ***ET1***

Transmission Repetition Rate:	1 s
Data Length:	8
Extended Data Page:	0
Data Page:	0
PDU Format:	254
PDU Specific:	238
Default Priority:	6
Parameter Group Number:	65262 (0x00FEEE)

PGN Supporting Information:

Start Position	Length	Parameter Name	SPN
1	1 byte	Engine Coolant Temperature	110
2	1 byte	Engine Fuel Temperature 1	174
3-4	2 bytes	Engine Oil Temperature 1	175
5-6	2 bytes	Engine Turbocharger Oil Temperature	176
7	1 byte	Engine Intercooler Temperature	52
8	1 byte	Engine Intercooler Thermostat Opening	1134

- Číslo PGN se skládá z:
  - Extended Data Page
  - Data Page
  - PDU Format
  - PDU Specific

PGN = EDP * 131072 + DP * 65536 + PDU Format * 256 + PDU Specific
PGN = 0*131072 + 0*65536 + 254*256 + 238 = 65262 (0xFEEE)

zdroj: [7]

Aplikační vrstva (J1939/71)

- **Ukázka SPN**

SPN 110 ***Engine Coolant Temperature***

Temperature of liquid found in engine cooling system.

Data Length:	1 byte	
Resolution:	1 deg C/bit, -40 deg C offset	
Data Range:	-40 to 210 deg C	Operational Range: same as data range
Type:	Measured	
Supporting Information:		
PGN reference:	65262	

  - Přepočet surové hodnoty přenášené ve zprávě na fyzikální hodnotu:

Fyzikální hodnota = Surová hodnota * Rozlišení + Offset

  - Pro konkrétní SPN 110, např.:
    - Surová hodnota = 30 (0x1E) – toto číslo se přenáší ve zprávě
    - Rozlišení (Resolution) = 1 °C/bit
    - Offset = -40 °C
    - **Teplota chladící kapaliny [°C] = 30 * 1 – 40 = -10 °C**
  - Hodnoty jsou ve zprávě přenášeny vždy jako **celá čísla**.

zdroj: [7]

Aplikační vrstva (J1939/71)

- **DAF 105.460, aplikační vrstva CAN SAE J1939**

dvoubajtový signál

jednobajtový signál

Time	Chn	ID	PGN	Name	Src	Dest	Sender Node	Receive Node	Prio	DLC	Data
0.050027	CAN 2	CF00300x	F003p	EEC2	00	all	Engine1		3	8	F0 C0 2C FF FF 03 FF FF
0.019923	CAN 2	CF00400x	F004p	EEC1	00	all	Engine1		3	8	FE C6 A6 2C 2C 00 04 FF
DemandPercentTorque							<Not avail.>	FF Engine Demand – Percent Torque....			
StarterMode							starter inhibited due to engine alrea...	4 Engine Starter Mode....			
SAOfCtrlDeviceForEngCtrl							0	0 Source Address of Controlling Device for Engine Control....			
Speed							1413.5000 rpm	2C2C Engine Speed....			
ActualEnginePercentTorque							41 %	A6 Actual Engine - Percent Torque....			
DriversDemandEnginePercentTorque							73 %	C6 Driver's Demand Engine - Percent Torque....			
ActualEngPercentTrqFractional							1.8750 %	F Actual Engine - Percent Torque (Fractional)....			
TorqueMode							14	E Engine Torque Mode....			
0.048751	CAN 2	CFE6CEEx	FE6Cp	TCO1	EE	all	ECU 238		3	8	08 10 00 C0 00 00 00 00
0.998172	CAN 2	18FEC027x	FEC0p	SERV	27	all	ManagementComputer		6	8	FF DB 6F FF FF FF FF FF
1.000628	CAN 2	18FEC1EEx	FEC1p	VDHR	EE	all	ECU 238		6	8	4A DC F9 04 17 EC 97 00
1.021240	CAN 2	18FEE500x	FEE5p	HOURS	00	all	Engine1		6	8	76 19 02 00 87 EC 06 00
0.998404	CAN 2	18FEE900x	FEE9p	LFC1	00	all	Engine1		6	8	74 01 00 00 54 84 04 00
0.999929	CAN 2	18FEEE00x	FEEEp	ET1	00	all	Engine1		6	8	4A 35 A3 24 FF FF FF FF
0.099962	CAN 2	18FEF100x	FEF1p	CCVS1	00	all	Engine1		6	8	F7 00 00 04 01 00 E0 FF
0.097858	CAN 2	18FEF200x	FEF2p	LFE1	00	all	Engine1		6	8	3E 02 00 00 FF FF C9 FF
0.998808	CAN 2	18FEFC27x	FEFCp	DD1	27	all	ManagementComputer		6	8	FF FA FF FF FF FF FF FF
10.200812	CAN 2	18ECFFEEx	EC00p	TPCM	EE	all	ECU 238		6	8	20 12 00 03 FF EC FE 00
0.100082	CAN 2	18EBFFEEx	EB00p	TPDT	EE	all	ECU 238		6	8	03 38 38 35 2A FF FF FF
10.200880	CAN 2	18FEECEEx	FEECp	VI	EE	all	ECU 238		6	18	58 4C 52 54 45 34 37 4D 53 ...
9.999980	CAN 2	1CFDD127x	FDD1p	FMS	27	all	ManagementComputer		7	8	F0 30 30 31 30 FF FF FF



Aplikační vrstva (J1939/71)

- **DAF 105.460, aplikační vrstva CAN SAE J1939**

dvoubajtový signál

jednobajtový signál

Zpráva	Jméno	P...	PGN...	Zdroj	Cíl	Měř.per.	Data
EEC2	Electronic Engine Controller #2	F003	61443	00 Motor		50	F1 00 12 FF FF FF FF FF
EEC1	Electronic Engine Controller #1	F004	61444	00 Motor		19	F0 7D 87 34 11 00 F4 FF
TCO1	Tachograph #1	FE6C	65132	EE Tachograf		50	08 10 00 C0 00 00 00 00
FE	Fuel Economy	FEF2	65266	00 Motor		98	3B 00 00 00 FF FF 00 FF
CCVS	Cruise Control/Vehicle Speed	FEF1	65265	00 Motor		100	F7 00 00 04 01 00 E0 FF
	ECU_vozidla?					1000	FF 0F FF FF FF FF FF FF
ENTEMP	Engine Temperature	FEEE	65262	00 Motor		1001	52 4E 00 27 FF FF FF FF
HRVD	High Resolution Vehicle Distance	FEC1	65217	EE Tachograf		1000	66 99 67 08 C5 DF 84 00
EHR	Engine Hours, Revolutions	FEE5	65253	00 Motor		900	D9 47 03 00 56 18 0C 00
mFEEC_EEbam	Proprietary BAM	ECFF	60671	EE Tachograf			20 12 00 03 FF EC FE 00
mFEEC_EEpck	Multipacket	EBFF	60415	EE Tachograf		100	58 4C 52 54 45 34 37 4D 53

Veličina	Aktuální	Rozsah	Jedn.	Zpráva	SA:
Accelerator Pedal 1 Low Idle Switch			[]	EEC2: 1/1-2	00 Motor
Accelerator Pedal Kickdown Switch			[]	EEC2: 1/3-4	00 Motor
Accelerator Pedal Position 1	0,0	0,0 - 0,0	%	EEC2: 2	00 Motor
Percent Load at Current Speed	16	0 - 125	%	EEC2: 3	00 Motor
Actual Engine - % Torque	9	0 - 49	%	EEC1: 3	00 Motor
Engine Speed	551,3	0,0 - 728,5	rpm	EEC1: 4-5	00 Motor
Tachograph Vehicle Speed	0,0	0,0 - 0,0	km/h	TCO1: 7-8	EE Tachograf
Fuel Rate	2,60	0,00 - 11...	l/hod	FE: 1-2	00 Motor
Engine Coolant Temperature	42	41 - 42	°C	ENTEMP: 1	00 Motor
Fuel Temperature			°C	ENTEMP: 2	00 Motor
Engine Oil Temperature	39	39 - 39	°C	ENTEMP: ...	00 Motor
Total Vehicle Distance			km	HRVD: 1-4	EE Tachograf
Trip Distance			km	HRVD: 5-8	EE Tachograf
Total Engine Hours	10750,1	10750,0 ...	hr	EHR: 1-4	00 Motor



zdroj: [11]

4. Linková vrstva

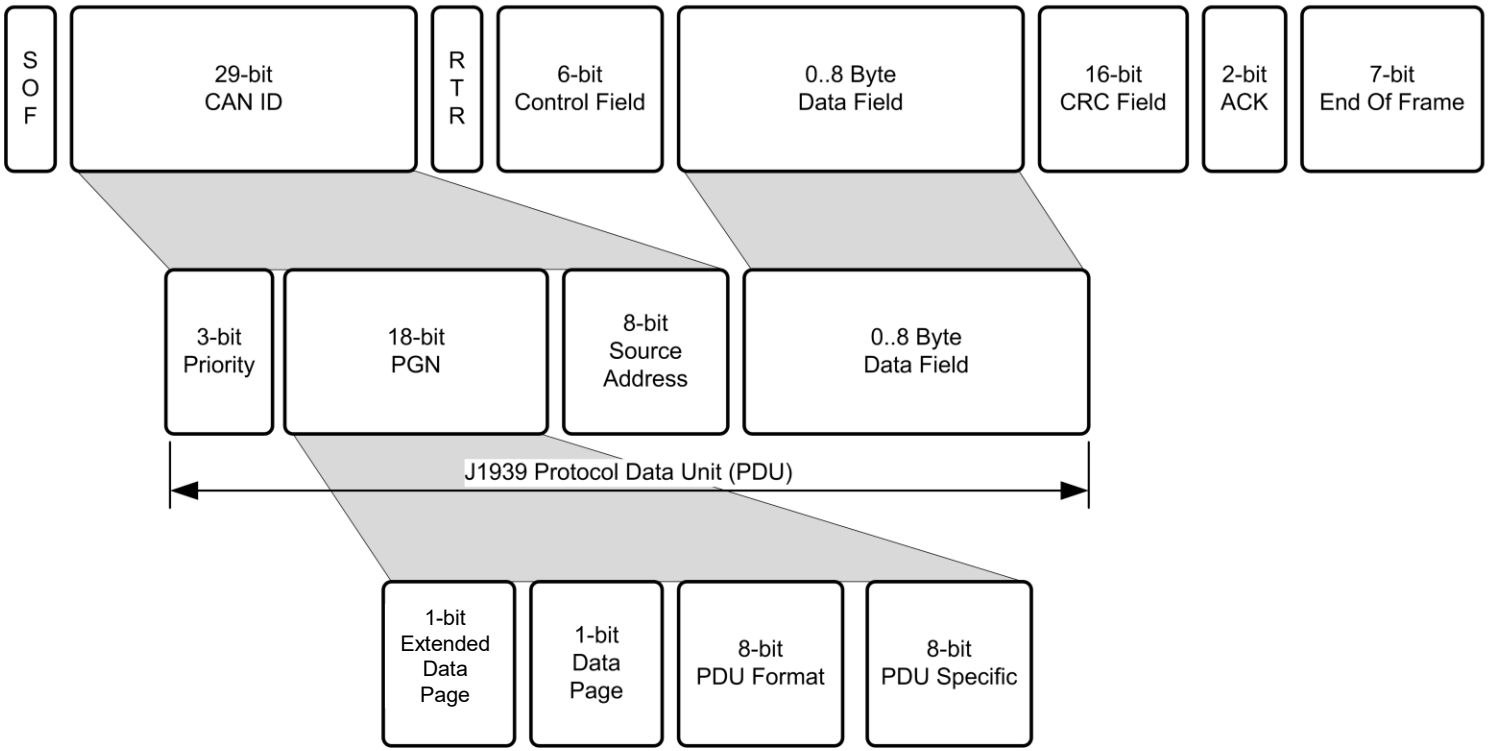
Linková vrstva (J1939/21)

- Definuje
  - Jak vypadá CAN datová zpráva (CAN data frame).
  - Transportní protokol pro přenos dat, kterých je více než se vejde do jedné CAN zprávy (data jsou rozdělena do více zpráv).

- Formát zprávy na J1939
  - Používá se 29-bit identifikátor zprávy (CAN 2.0B) jehož význam je rozšířen, obsahuje tyto informace:
    - Priorita zprávy
    - Typ zprávy (peer-to-peer, broadcast, transportní protokol, network management)
    - Adresa zdroje
    - Adresa cíle (pouze v peer-to-peer komunikaci)
  - 8 B dat u CAN Classic nebo až 60 B dat u CAN FD
    - Obsah PGN.
    - Co se vejde do jedné CAN zprávy.
    - Přenos většího počtu dat, než se vejde do jedné CAN zprávy, řeší transportní protokol.

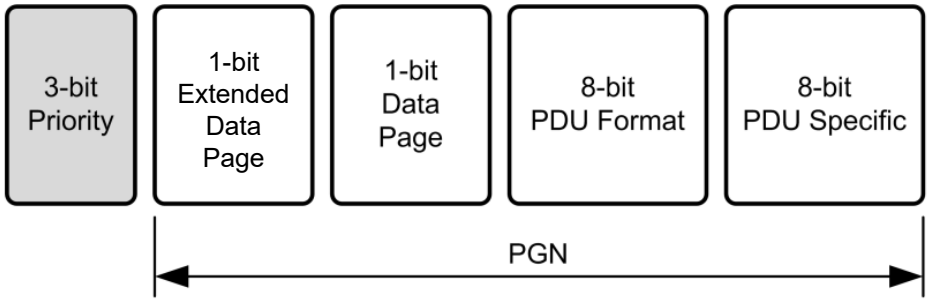
Formát zprávy na J1939

- **PDU = Protocol Data Unit**



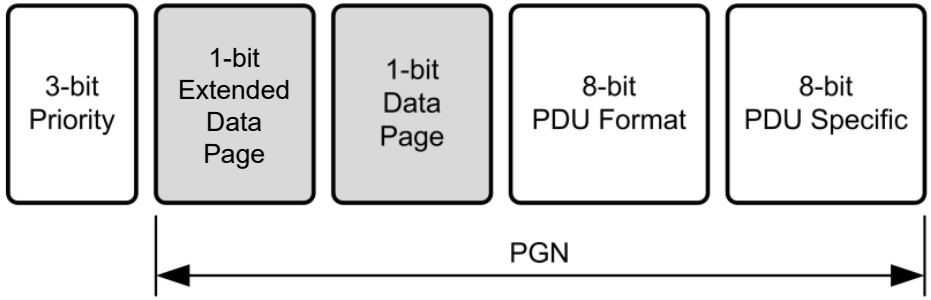
  - Počet datových bajtů (Data Field) je, až na několik výjimek, vždy 8 (nevyužité datové bajty mají hodnotu 0xFF „not available“).

Formát zprávy na J1939



- **Priority**
  - 3 bity (jsou mimo PGN)
  - 8 úrovní priority přenášené zprávy
    - 0 (0b000) nejvyšší priorita
    - 7 (0b111) nejnižší priorita
  - Vysoká priorita je přiřazena časově kritickým datům jako např. zprávy z převodovky pro řízení momentu spalovacího motoru.
  - Nízká priorita je přiřazena časově nenáročným datům jako např. atmosférický tlak nebo konfigurační data motoru.

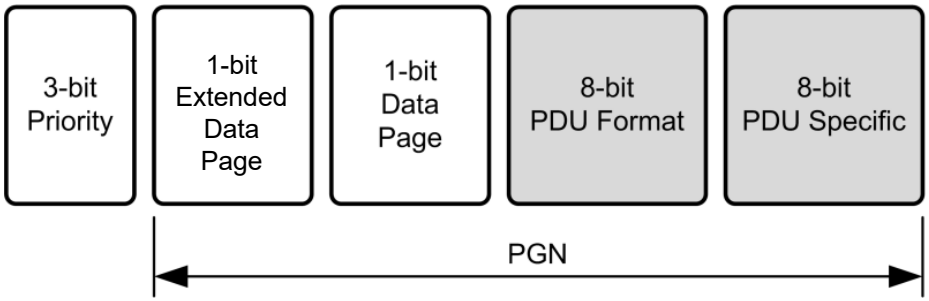
Formát zprávy na J1939



- **Extended Data Page**
  - 1 bit
- **Data Page**
  - 1 bit
- Tyto 2 bity vybírají stránku pro následné pole PDU Format

Extended Data Page	Data Page	Popis
0	0	SAE J1939 a ISO 11783 (ISOBUS)
0	1	SAE J1939 NMEA 2000
1	0	SAE J1939 rezervováno
1	1	ISO 11992 Diagnostika

Formát zprávy na J1939



- **PDU Format a PDU Specific**
  - **PDU Format je v rozsahu 0 až 239 (PDU1)** → pole PDU Specific má význam adresy cílového zařízení, pro které je zpráva určena
    - PDU Format určuje význam přenášených dat (8-bit)
    - PDU Specific určuje cílovou adresu zprávy (8-bit)
  - **PDU Format je v rozsahu 240 až 255 (PDU2)** → pole PDU Specific rozšiřuje pole PDU Format na šířku 16-bit
    - PDU Format + PDU Specific (16-bit) určují význam přenášených dat

Formát zprávy na J1939

Příklad

- CAN ID: 0x18FEEE00
  - Priorita: 6
  - Extended Data Page: 0
  - Data Page: 0
  - PDU Format: 254 ≥ 240 ⇒ Formát PDU2
  - PDU Specific: 238
  - Adresa zdroje zprávy: 00 (spalovací motor)

PGN = EDP * 131072 + DP * 65536 + PDU Format * 256 + PDU Specific
PGN = 0*131072 + 0*65536 + 254*256 + 238 = 65262 (0x00FEEE)

Time	Chn	ID	Prio	Data Page	PGN	Name	Src	Dest	DLC	Data
1.000016	CAN 2	18FEEE00x	6	0	FEEEp	ET1	00	FF	8	58 48 E0 29 FF FF FF FF

Adresa cíle neexistuje (resp. je FF – „pro všechny“), protože se jedná o broadcast zprávu vysílanou spalovacím motorem (adresa zdroje 00) a přijímanou ostatními řídícími jednotkami na sběrnici.

Formát zprávy na J1939

Příklad

- CAN ID: 0x18FEEE00
  - Co se skrývá pod PGN 65262 ?

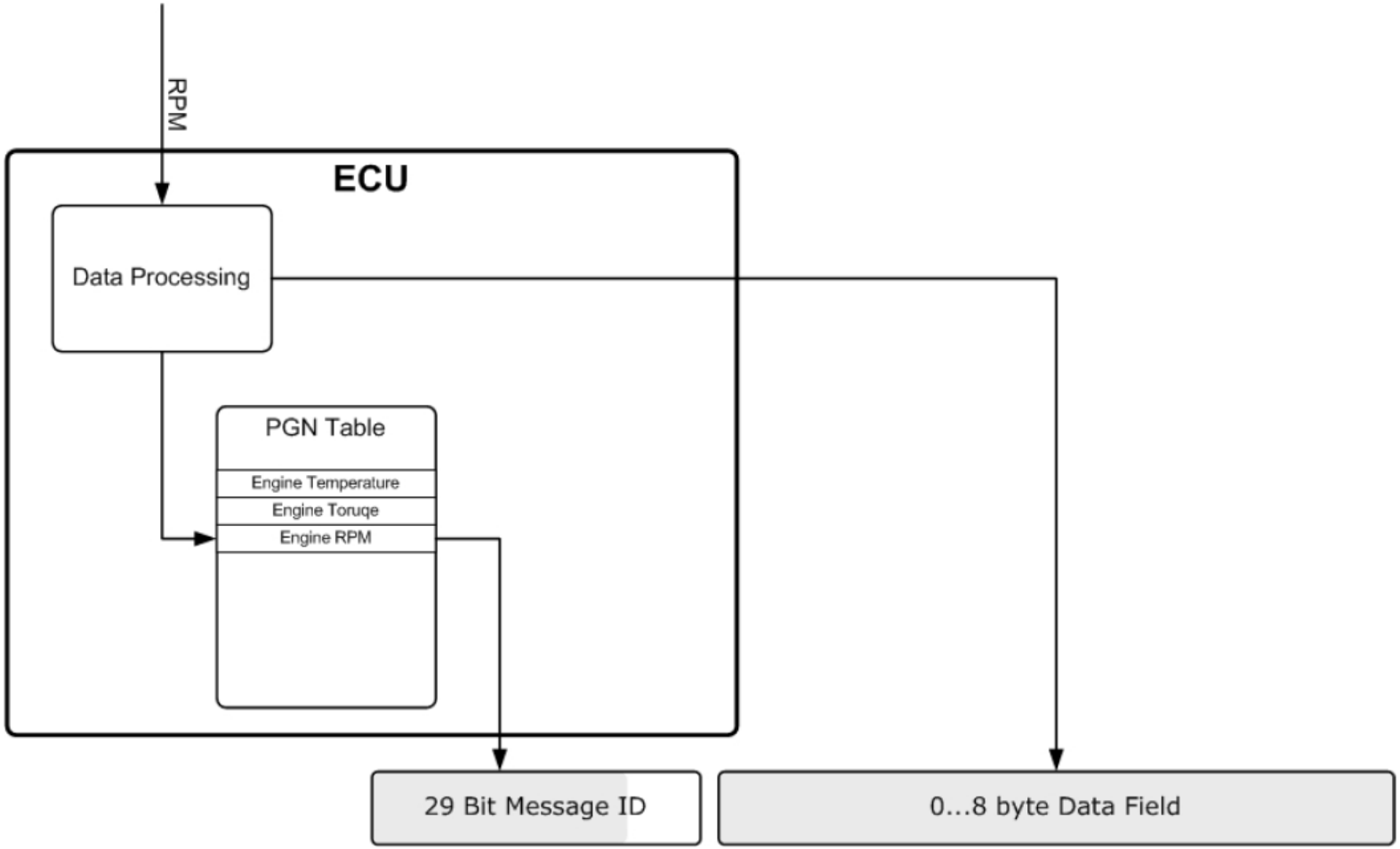
PGN 65262 ***Engine Temperature 1***

Transmission Repetition Rate:	1 s		
Data Length:	8		
Extended Data Page:	0		
Data Page:	0		
PDU Format:	254		
PDU Specific:	238	PGN Supporting Information:	
Default Priority:	6		
Parameter Group Number:	65262 (0x00FEEE)		

Start Position	Length	Parameter Name	SPN
1	1 byte	Engine Coolant Temperature	110
2	1 byte	Engine Fuel Temperature 1	174
3-4	2 bytes	Engine Oil Temperature 1	175
5-6	2 bytes	Engine Turbocharger Oil Temperature	176
7	1 byte	Engine Intercooler Temperature	52
8	1 byte	Engine Intercooler Thermostat Opening	1134

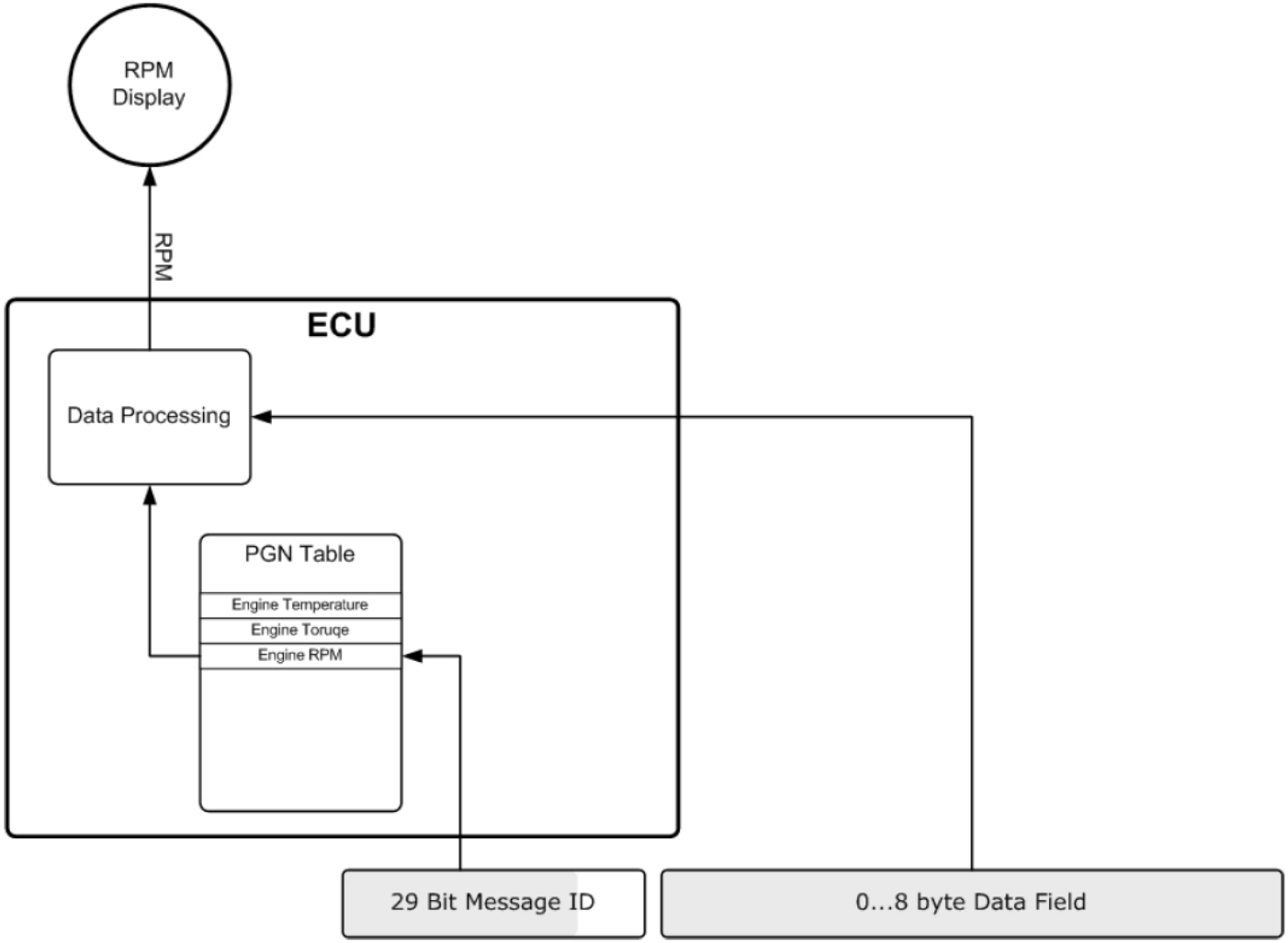
zdroj: [7]

Ukázka – odeslání otáček motoru



zdroj: [2]

Ukázka – příjem otáček motoru



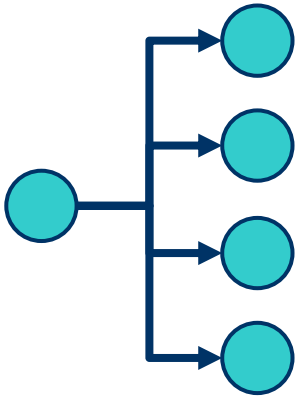
zdroj: [2]

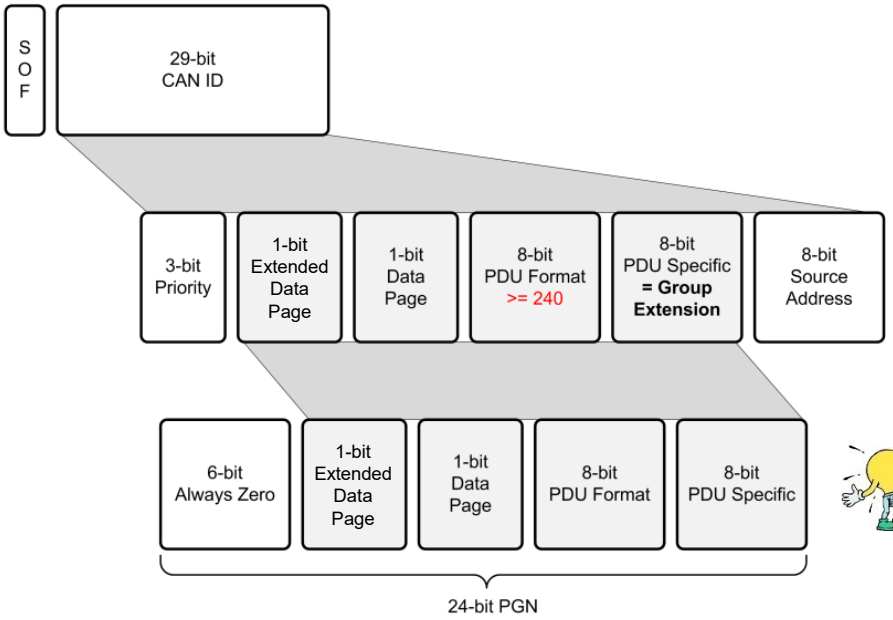
Formát zprávy na J1939

- **Broadcast zprávy (Global PGN)**
  - Zpráva **určená všem ECU** na sběrnici.
    - Pole PDU Format ≥ 240 (≥ 0xF0) ⇒ PDU2
    - Pole PDU Specific má význam Group Extension (rozšiřuje pole PDU Format).
    - Jedná se o všechna PGN v rozmezí 0x00F000 až 0x00FFFF (61440 až 65535).
    - **Celkový počet těchto PGN je 4096**.
    - Převažují v komunikaci.



zdroj: [3]



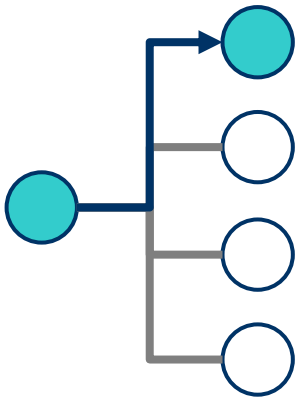


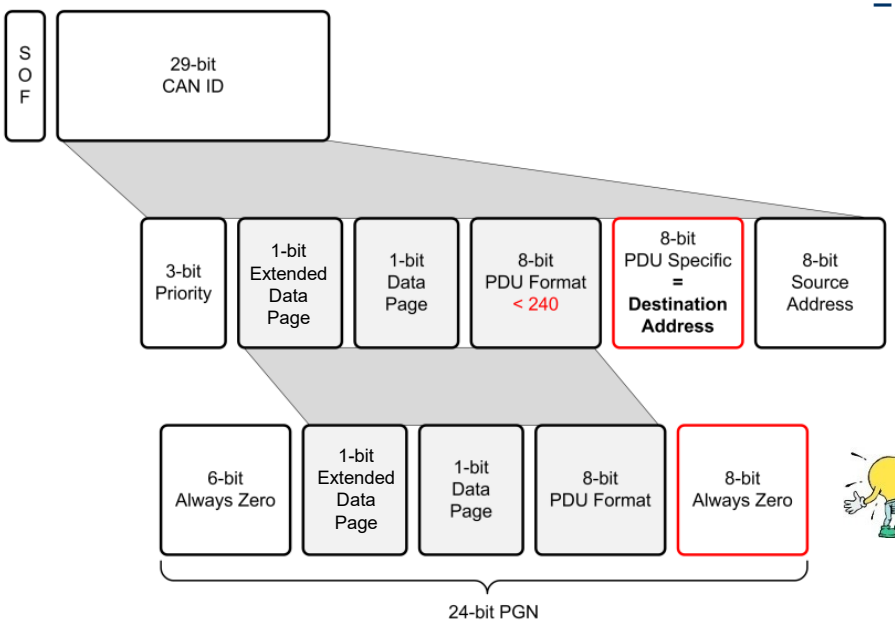


V programu běžícím v ECU je PGN uloženo jako 24-bitové číslo.

Formát zprávy na J1939

- **Peer-to-peer zprávy (Specific PGN)**
  - Zpráva **určená pro jednu konkrétní ECU**.
    - Pole PDU Format < 240 (< 0xF0) ⇒ PDU1
    - Pole PDU Specific má význam cílové adresy (a proto je v PGN toto pole vždy vynulováno, cílová adresa není součástí PGN).
    - Jedná se o všechna PGN v rozmezí 0x000000 až 0x00EF00 (0 až 61184).
    - **Celkový počet těchto PGN je jen 240**.
    - I v peer-to-peer komunikaci lze odesílat broadcast zprávy, když Destination Address = 255 (0xFF)







zdroj: [3]



V programu běžícím v ECU je PGN uloženo jako 24-bitové číslo.

Formát zprávy na J1939

- **Broadcast** a **Peer-to-Peer** zprávy

pro koho je určená
kdo zprávu odeslal

pro koho je určená
kdo zprávu odeslal

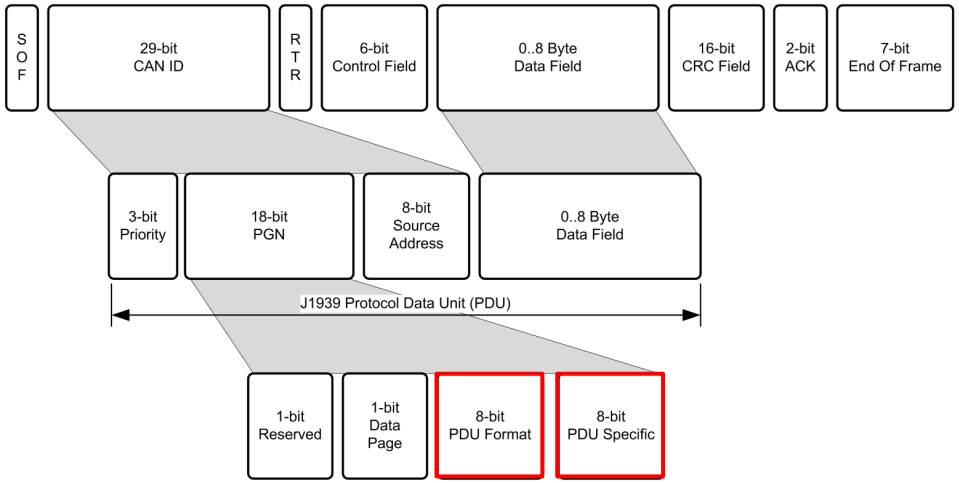
Time	Chn	ID	PGN	Name	Src	Dest	Sender Node	Receive Node	Prio	DLC	Data
29.557434	CAN 2	18EA0003x	EA00p	RQST	03	00	Transmission1	Engine1	6	3	C1 FE 00
29.898648	CAN 2	18EA00F2x	EA00p	RQST	F2	00	ECU 242	Engine1	6	3	E5 FE 00
30.119828	CAN 2	C00000Bx	0p	TSC1	0B	00	BrakesSystemController	Engine1	3	8	FC FF FA FA FF FF FF FF
30.137475	CAN 2	C000003x	0p	TSC1	03	00	Transmission1	Engine1	3	8	EC FF FF FA FF FF FF FF
30.177544	CAN 2	C000F03x	0p	TSC1	03	0F	Transmission1	RetarderEngine	3	8	EF FF FF 7D FF FF FF FF
30.146178	CAN 2	C000F0Bx	0p	TSC1	0B	0F	BrakesSystemController	RetarderEngine	3	8	FC FF FA 00 FF FF FF FF
30.140867	CAN 2	C00100Bx	0p	TSC1	0B	10	BrakesSystemController	RetarderDriveline	3	8	FC FF FA 00 FF FF FF FF
30.151474	CAN 2	C00290Bx	0p	TSC1	0B	29	BrakesSystemController	RetarderExhaustEngine1	3	8	FC FF FA 00 FF FF FF FF
29.427715	CAN 2	18DF6566x	DF00p	DM13	66	65	ECU 102	ECU 101	6	8	D4 03 61 03 0A FF FF FF
30.218065	CAN 2	FF012Fx	FF01p		2F	all	SuspensionSystemController1		0	8	B3 76 A2 76 7F 76 FF FF
30.194515	CAN 2	FF022Fx	FF02p		2F	all	SuspensionSystemController1		0	8	FF FF FF FF FF FF FF FF
30.195683	CAN 2	FF032Fx	FF03p		2F	all	SuspensionSystemController1		0	8	00 20 03 05 FB FF C0 00
28.032397	CAN 2	18FFECEEx	FFECp		EE	all	ECU 238		6	17	30 30 30 30 30 30 30 30 30 ...
29.871789	CAN 2	18FEF500x	FEF5p	AMB	00	all	Engine1		6	8	C6 FF FF FF FF FF FF FF
30.192799	CAN 2	CFE5A2Fx	FE5Ap	ASC1	2F	all	SuspensionSystemController1		3	8	77 00 00 0F FF F5 FF F0
30.193903	CAN 2	18FE592Fx	FE59p	ASC3	2F	all	SuspensionSystemController1		6	8	FF FF 91 76 7F 76 77 76
30.195099	CAN 2	18FE582Fx	FE58p	ASC4	2F	all	SuspensionSystemController1		6	8	FF FF FF FF FF FF FF FF
29.872377	CAN 2	18FE5600x	FE56p	AT1T1I	00	all	Engine1		6	8	FF FF FF FF FF FF FF FF
30.198371	CAN 2	18FED900x	FED9p	AUXIO1	00	all	Engine1		6	8	00 00 00 FF FF FF FF FF
30.200102	CAN 2	18FEF100x	FEF1p	CCVS	00	all	Engine1		6	8	F0 00 00 C0 00 00 1F F0
29.510013	CAN 2	18FEFC65x	FEFCp	DD	65	all	ECU 101		6	8	FF EE FF FF FF FF FF 1A

all – znamená „pro všechny“ (broadcast)

Formát zprávy na J1939

- Shrnutí významu polí **PDU Format** a **PDU Specific**

	PDU Format	PDU Specific	Způsob komunikace	Počet PGN
PDU1 Format	0 – 239 (0x00 – 0xEF)	Destination Address 0 – 255 (0x00 – 0xFF)	Peer-to-Peer	240
PDU2 Format	240 – 255 (0xF0 – 0xFF)	Group Extension 0 – 255 (0x00 – 0xFF)	Broadcasting	4096



Formát zprávy na J1939

- PGN jsou
  - Standardizované (uvedené v příslušných dokumentech SAE J1939-xx).
  - Definované výrobcem ECU (Manufacturer Specific - MF).

	EDP	DP	Rozsah PGN (hex)	Počet PGN	SAE nebo MF	Způsob komunikace
SAE J1939	0	0	000000 - 00EE00	239	SAE	Peer-to-Peer
	0	0	00EF00	1	MF	Peer-to-Peer
	0	0	00F000 - 00FEFF	3840	SAE	Broadcasting
	0	0	00FF00 - 00FFFF	256	MF	Broadcasting
NMEA2000	0	1	010000 - 01EE00	239	SAE	Peer-to-Peer
	0	1	01EF00	1	MF	Peer-to-Peer
	0	1	01F000 - 01FEFF	3840	SAE	Broadcasting
	0	1	01FF00 – 01FFFF	256	MF	Broadcasting

  - **Data Page** bit plní funkci přepínače stránek pro pole PDU Format – celkový počet různých PGN je díky tomu dvojnásobný: 2x(240 + 4096) = 8672

Formát zprávy na J1939

SW Vector CANalyzer

SA = Source Address (adresa ECU, která zprávu odeslala)

manufacturer specific PGN (CAN-ID xxFFxxSA) – jejich význam zná jen výrobce

Time	Chn	ID	PGN	Name	Src	Dest	Sender Node	Receive Node	Prio	DLC	Data
9.749424	CAN 2	CFF6752x	FF67p		52	all	ECU 82		3	14	17 05 00 00 00 00 00 D8 03 ...
9.838290	CAN 2	CFF1900x	FF19p		00	all	Engine1		3	8	10 0E 00 00 00 00 A6 3D
9.905125	CAN 2	CFF1752x	FF17p		52	all	ECU 82		3	8	FF FF FF FF FF FF FF FF
9.878009	CAN 2	CFF1500x	FF15p		00	all	Engine1		3	8	00 00 C0 12 F0 FE FC 0F
9.881693	CAN 2	CFF1400x	FF14p		00	all	Engine1		3	8	00 FF FF FF FF FF FF FF
9.889177	CAN 2	CFF113Dx	FF11p		3D	all	ExhaustEmissionController		3	8	50 06 80 33 9E 00 00 FF
9.887365	CAN 2	CFF103Dx	FF10p		3D	all	ExhaustEmissionController		3	8	FF FE 84 AC BA 25 40 05
9.769904	CAN 2	18FEF600x	FEF6p	IC1	00	all	Engine1		6	8	FF 00 62 32 00 80 33 FF
9.272133	CAN 2	18FEF500x	FEF5p	AMB	00	all	Engine1		6	8	C5 FF FF 66 24 FE FF FF
9.834758	CAN 2	18FEF100x	FEF1p	CCVS	00	all	Engine1		6	8	FF 00 00 FF FF FF FF FF
9.268126	CAN 2	18FEEE00x	FEEEp	ET1	00	all	Engine1		6	8	7B 7B BD 2D FF FF FF FF
9.143696	CAN 2	18FECA3Dx	FECAp	DM1	3D	all	ExhaustEmissionController		6	8	00 FF 18 F0 E5 01 FF FF
9.276357	CAN 2	18FEA400x	FEA4p	ET2	00	all	Engine1		6	8	FE FF FF FF FF FF FF FF
9.768128	CAN 2	18FD6E00x	FD6Ep	SCR1	00	all	Engine1		6	8	0E 00 0C 00 FF FF FF FF
9.908101	CAN 2	CF00400x	F004p	EEC1	00	all	Engine1		3	8	F0 7D 82 C0 12 00 F3 FF
9.879813	CAN 2	CF00300x	F003p	EEC2	00	all	Engine1		3	8	DD 00 09 FF FF FF FF FF
9.646334	CAN 2	CECFF52x	EC00p	TPCM	52	all	ECU 82		3	8	20 0E 00 02 FF 67 FF 00

standardizované PGN – uvedené v dokumentech SAE J1939-xx

Formát zprávy na J1939

- **SW ELBAS Ucint Suite**

SA = Source Address (adresa ECU, která zprávu odeslala)

manufacturer specific PGN (CAN-ID xxFFxxSA) – jejich význam zná jen výrobce

CANTrees DEMO

Vymazat | Zdroje | Nástroje | Grafy | Konec

Dávka | 23528 | 1 | – PAUZA –

Stromy | Zprávy | Veličiny

Arb.ID	E.	Prior	Page	res.	Zpráva	Jméno	PGN hex	PGN dec	Zdroj	Cíl	Četnost	Mě...	Data
18FD6E00	x	6				Motor?						0	0D 00 0E 00 FF FF FF FF
0CFF1D03	x	3				Převodovka?						10	F0 FF FA FF FA FF FF FF
0CFF1C03	x	3				Převodovka?						10	FF FF 00 00 FF FF FF FF
0CFF1B03	x	3				Převodovka?						11	C0 FF FA E0 C4 FF FF 00
0CFF1900	x	3				Motor?						100	10 0E B6 11 3C 00 67 58
0CFF1800	x	3				Motor?						9	36 91 1A 00 00 FF 25 0A
0CFF1752	x	3				52?						20	FF FF FF FF FF FF FF FF
0CFF1500	x	3				Motor?						49	00 00 AA 1A F5 FE FC 0F
0CFF1400	x	3				Motor?						49	00 FF FF FF FF FF FF FF
0CFF113D	x	3				SCR/NOx?						52	FA 1A 64 3C 0E 00 00 FF
0CFF103D	x	3				SCR/NOx?						52	FF FE 8E A1 64 24 01 58
1CEBFF3D	x	7			mFECA_3Dpck	Multipacket	EBFF	60415	3D SCR/NOx			52	00 FF 18 F0 E5 01 17 F0 EE 01 FF FF FF FF
18EBFF00	x	6			mFECA_00pck	Multipacket	EBFF	60415	00 Motor			60	40 3F EB 0B 01 01 FA 0F 10 7F FF FF FF FF
0CEBFF52	x	3			mFF67_52pck	Multipacket	EBFF	60415	52 52			51	5E 04 FC 10 00 00 00 EF 03 01 10 4D A0 C8
1CECFF3D	x	7			mFECA_3Dbam	DM1 BAM	ECFF	60671	3D SCR/NOx			1000	20 0A 00 02 FF CA FE 00
18ECFF00	x	6			mFECA_00bam	DM1 BAM	ECFF	60671	00 Motor			960	20 0A 00 02 FF CA FE 00
0CECFF52	x	3			mFF67_52bam	Proprietary BAM	ECFF	60671	52 52			1000	20 0E 00 02 FF 67 FF 00
0CF00300	x	3			EEC2	Electronic Engine Controller #2	F003	61443	00 Motor			49	D1 00 00 FF FF FF FF FF
0CF00400	x	3			EEC1	Electronic Engine Controller #1	F004	61444	00 Motor			20	F0 7D 7D 91 1A 00 F3 FF
18FEEE00	x	6			ENTEMP	Engine Temperature	FEEE	65262	00 Motor			1000	5E 5E 5B 29 FF FF FF FF
18FEF100	x	6			CCVS	Cruise Control/Vehicle Speed	FEF1	65265	00 Motor			100	FF 99 2B FF FF FF FF FF
18FEF500	x	6			AMCOND	Ambient conditions	FEF5	65269	00 Motor			1000	C6 60 23 70 24 FE FF FF
18FEF600	x	6			INCOND	Inlet/Exhaust conditions	FEF6	65270	00 Motor			501	FF 05 44 36 00 FE FF FF

standardizované PGN – uvedené v dokumentech SAE J1939-xx

Rozsahy signálů: validní, chybný, nedostupný

CAN-ID	Datové bajty	
0x18FEDF00	83 00 13 FF FF FF FF FF	nedostupné (chybějící) signály

Bitů	Validní signál	Chybný signál	Signál není dostupný
2	0x0 – 0x1	0x2	0x3
8	0x00 – 0xFA	0xFE	0xFF
16	0x0000 – 0xFAFF	0xFE00 – 0xFEFF	0xFF00 – 0xFFFF
24	0x000000 – 0xFAFFFF	0xFE0000 – 0xFEFFFF	0xFF0000 – 0xFFFFFF
32	0x00000000 – 0xFAFFFFFF	0xFE000000 – 0xFEFFFFFF	0xFF000000 – 0xFFFFFFFF

- Jedná se o surové hodnoty signálů přenášených ve zprávě! Nejsou to hodnoty signálů po přepočtu na fyzikální rozměr!
- Podle surové hodnoty signálu příjemce zprávy pozná, zda je signál validní, chybný nebo nedostupný.

Rozsahy signálů: validní, chybný, nedostupný

CAN-ID	Datové bajty	
0x18FEDF00	84 E8 1C FF FF FF FF FF	nedostupné (chybějící) signály

Time	Chn	ID	PGN	Name	Src	Dest	Prio	DLC	Data
3.516136	CAN 2	18FEDF00x	FEDFp	EEC3	00	all	6	8	84 E8 1C FF FF FF FF FF
EngsDsrdOprtngSpdAsymmtryAdjstmnt				<Not avail.>			FF		This byte is utilized in transmission g
Aftrtratment1ExhaustGasMassFlow				<Not avail.>			FFFF		Measured/calculated exhaust gas m
Aftertreatment2IntakeDewPoint				NotAvailable			3		Indicates that the temperature on
Aftertreatment2ExhaustDewPoint				NotAvailable			3		Indicates that the temperature on
Aftertreatment1IntakeDewPoint				NotAvailable			3		Indicates that the temperature on
Aftertreatment1ExhaustDewPoint				NotAvailable			3		Indicates that the temperature on
EstEngPrsticLossesPercentTorque				<Not avail.>			FF		The calculated torque that indicate
EngsDesiredOperatingSpeed				925.0000	rpm		1CE8		An indication by the engine of the
NominalFrictionPercentTorque				7	%		84		The calculated torque that indicate

nedostupné signály

fyzikální hodnota signálu po přepočtu

surová hodnota signálu

Rozsahy signálů: validní, chybný, nedostupný

CAN-ID	Datové bajty	
0x18FEDF00	84 E8 1C FF FF FF FF FF	nedostupné (chybějící) signály
	1. 2. 3. 4. 5. 6. 7. 8.	

PGN 65247 ***Electronic Engine Controller 3*** ***EEC3***

Transmission Repetition Rate:	250 msec (preferred) or Engine Speed Dependent (if required by application)	
Data Length:	8	
Extended Data Page:	0	
Data Page:	0	
PDU Format:	254	
PDU Specific:	223	PGN Supporting Information:
Default Priority:	6	
Parameter Group Number:	65247 (0x00FEDF)	

Start Position	Length	Parameter Name	SPN
1	1 byte	Nominal Friction - Percent Torque	514
2-3	2 bytes	Engine's Desired Operating Speed	515
4	1 byte	Engine's Desired Operating Speed Asymmetry Adjustment	519
5	1 byte	Estimated Engine Parasitic Losses - Percent Torque	2978
6-7	2 bytes	Aftertreatment 1 Exhaust Gas Mass Flow	3236
8.1	2 bits	Aftertreatment 1 Intake Dew Point	3237
8.3	2 bits	Aftertreatment 1 Exhaust Dew Point	3238
8.5	2 bits	Aftertreatment 2 Intake Dew Point	3239
8.7	2 bits	Aftertreatment 2 Exhaust Dew Point	3240

nedostupné (chybějící) signály

zdroj: [7]

Rozsahy signálů: validní, chybný, nedostupný

CAN-ID	Datové bajty
0x18FEDF00	84 E8 1C FF FF FF FF FF
	1. 2. 3. 4. 5. 6. 7. 8.

nedostupné (chybějící) signály

11 11 (2)

Dva dvoustavové signály, každý přenášený pomocí 2 bitů.

SPN 3239 ***Aftertreatment 2 Intake Dew Point***

Indicates that the temperature on the intake side of the aftertreatment system has exceeded the dew point, as estimated by the ECM in exhaust bank 2.

00 - Not exceeded the dew point
01 - Exceeded the dew point
10 - Error
11 - Not available

dvoubitová informace

Zde jsou oba signály „Not available“.

Data Length:	2 bits
Resolution:	4 states/2 bit, 0 offset
Data Range:	0 to 3
Type:	Status
Supporting Information:	
PGN reference:	65247

Operational Range: same as data range

SPN 3240 ***Aftertreatment 2 Exhaust Dew Point***

Indicates that the temperature on the exhaust side of the aftertreatment has exceeded the dew point, as estimated by the ECM in exhaust bank 2.

00 - Not exceeded the dew point
01 - Exceeded the dew point
10 - Error
11 - Not available

dvoubitová informace

Data Length:	2 bits
Resolution:	4 states/2 bit, 0 offset
Data Range:	0 to 3
Type:	Status
Supporting Information:	
PGN reference:	65247

Operational Range: same as data range

zdroj: [7]

Speciální PGN

- Při analýze komunikace na sběrnici se lze setkat se speciálními typy PGN.
- Tyto speciální PGN jsou definovány v dokumentech:
  - SAE J1939-21 *Data Link Layer*
  - SAE J1939-81 *Network Management*
  - SAE J1939-83 *Application Layer - Diagnostics*

Název PGN	Zkratka	PGN	Význam
Request	**RQST**	0x00EA00	Žádost o zaslání nějakého PGN.
Acknowledgment	**ACKM**	0x00E800	Odpověď na RQST zprávu.
Transport Protocol - Connection Management	**TP.CM**	0x00EC00	PGN transportní vrstvy. Pro přenos dat, které se nevejdou do 1 zprávy. První zpráva zahajující přenos multipaketu.
Transport Protocol - Data Transfer	**TP.DT**	0x00EB00	PGN transportní vrstvy. Pro přenos dat, které se nevejdou do 1 zprávy. Přenos vlastních dat multipaketu.
Address Claimed / Cannot Claim	**ACL**	0x00EE00	Používá se v Network Managementu (J1939-81) ve fázi získávání adresy ECU po připojení ECU k síti.
Commanded Address	**CA**	0x00FED8	Používá se v Network Managementu (J1939-81) za účelem změny adresy ECU na síti.
Diagnostic message 1	**DM1**	0x00FECA	Vyčtení paměti aktivních závad a zjištění stavu kontrolek závad. DM zpráv je celkem 60 (DM1 až DM60).

Speciální PGN: RQST a ACKM podrobněji

- **Request parameter group**
  - **RQST (PGN 0x00EA00)** (peer-to-peer)
  - **Žádost o zaslání nějakého PGN**
  - Zpráva je odeslána pro konkrétní nebo všechny ECU (specific vs. global request).
  - RQST zpráva obsahuje číslo požadovaného PGN (přenáší se jako data zprávy).
  - Pokud příjemce žádosti nemůže odpovědět (např. požadované PGN nemá nebo je chráněno přístupem nebo je ECU zaneprázdněná), odešle jako odpověď NACK (Negative Acknowledgment).
  - Pokud příjemce žádosti může odpovědět, zašle požadované PGN.

Typ zprávy	PGN	DA	SA	Data 1	Data 2	Data 3
Global request	0xEA00	0xFF	Requester	PGN (LSB)	PGN	PGN (MSB)
Specific request	0xEA00	Responder	Requester	PGN (LSB)	PGN	PGN (MSB)

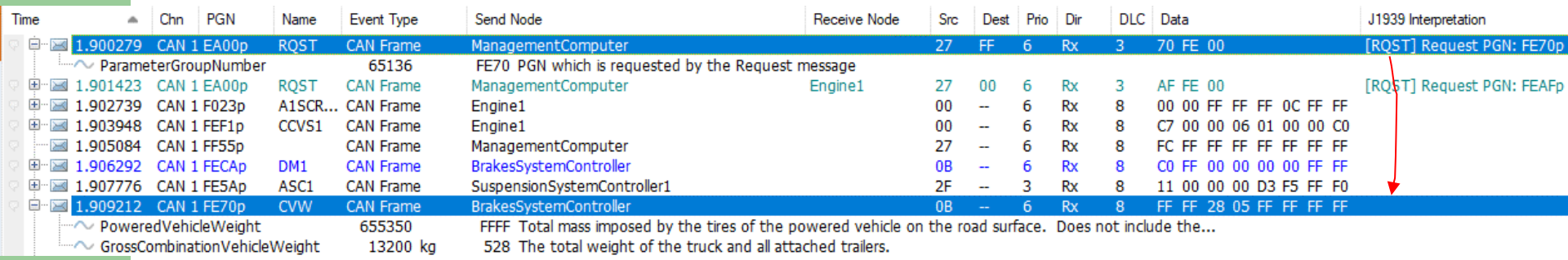
DA – Destination Address
SA – Source Address
Requester – adresa odesílatele žádosti
Responder – adresa pro koho je žádost určena

- **Acknowledgement parameter group**
  - **ACKM (PGN 0x00E800)** (peer-to-peer)
  - Slouží pro odeslání kladné nebo záporné odpovědi na RQST zprávu.

Speciální PGN: RQST a ACKM podrobněji

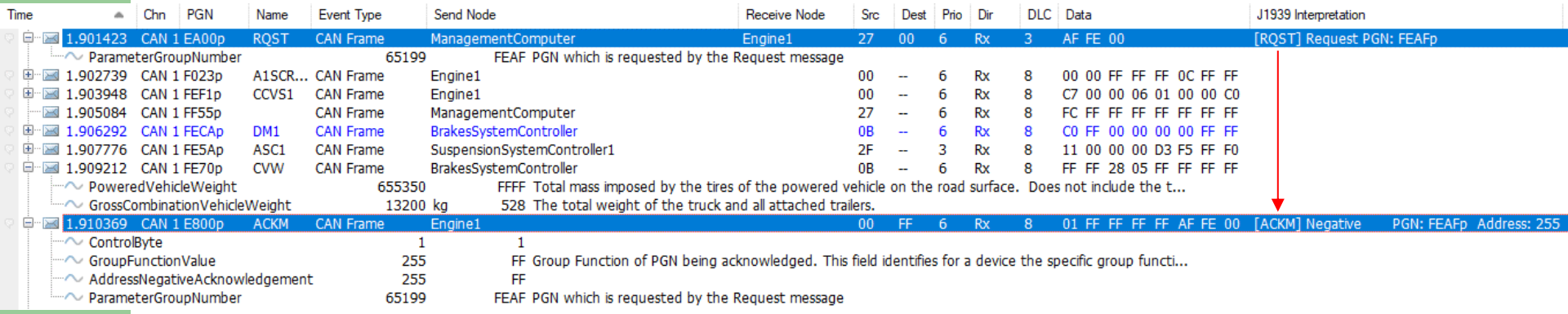
Příklad

- ECU vozidla (SA=0x27) žádá zaslání PGN 0xFE70 od kohokoliv (DA=0xFF)
- PGN 0xFE70 posílá ECU ABS (SA=0x0B)



Time	Chn	PGN	Name	Event Type	Send Node	Receive Node	Src	Dest	Prio	Dir	DLC	Data	J1939 Interpretation
1.900279	CAN 1	EA00p	RQST	CAN Frame	ManagementComputer		27	FF	6	Rx	3	70 FE 00	[RQST] Request PGN: FE70p
ParameterGroupNumber				65136	FE70 PGN which is requested by the Request message								
1.901423	CAN 1	EA00p	RQST	CAN Frame	ManagementComputer	Engine1	27	00	6	Rx	3	AF FE 00	[RQST] Request PGN: FEAFp
1.902739	CAN 1	F023p	A1SCR...	CAN Frame	Engine1		00	--	6	Rx	8	00 00 FF FF FF 0C FF FF	
1.903948	CAN 1	FEF1p	CCVS1	CAN Frame	Engine1		00	--	6	Rx	8	C7 00 00 06 01 00 00 C0	
1.905084	CAN 1	FF55p		CAN Frame	ManagementComputer		27	--	6	Rx	8	FC FF FF FF FF FF FF FF	
1.906292	CAN 1	FECAp	DM1	CAN Frame	BrakesSystemController		0B	--	6	Rx	8	C0 FF 00 00 00 00 FF FF	
1.907776	CAN 1	FE5Ap	ASC1	CAN Frame	SuspensionSystemController1		2F	--	3	Rx	8	11 00 00 00 D3 F5 FF F0	
1.909212	CAN 1	FE70p	CVW	CAN Frame	BrakesSystemController		0B	--	6	Rx	8	FF FF 28 05 FF FF FF FF	
PoweredVehicleWeight				655350	FFFF Total mass imposed by the tires of the powered vehicle on the road surface. Does not include the...								
GrossCombinationVehicleWeight				13200 kg	528 The total weight of the truck and all attached trailers.								

- ECU vozidla (SA=0x27) žádá zaslání PGN 0xFEAF od ECU motoru (DA=0x00)
- ECU motoru (SA=0x00) toto PGN nemá, odpovídá **Negative Acknowledgment**



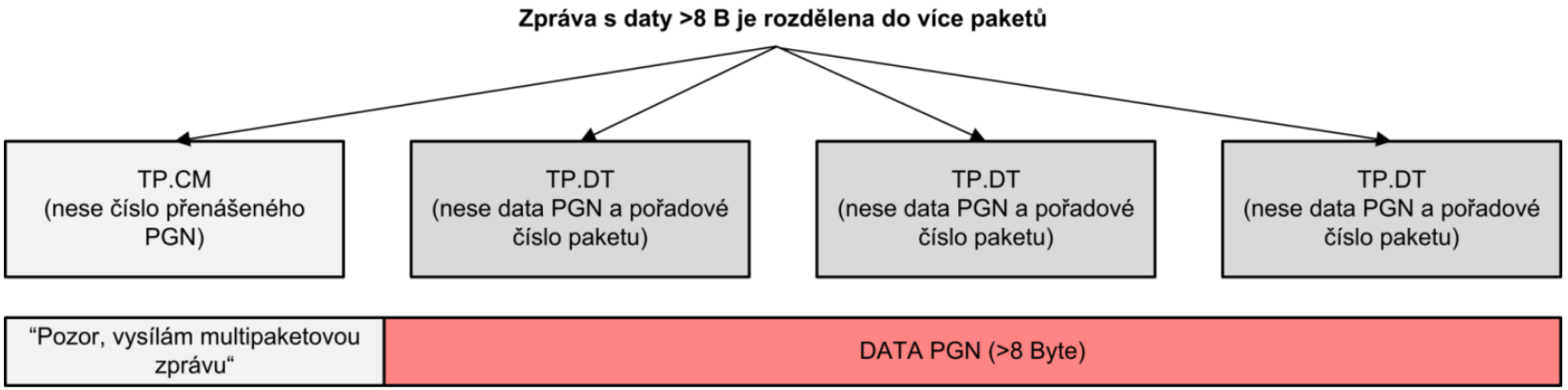
Time	Chn	PGN	Name	Event Type	Send Node	Receive Node	Src	Dest	Prio	Dir	DLC	Data	J1939 Interpretation
1.901423	CAN 1	EA00p	RQST	CAN Frame	ManagementComputer	Engine1	27	00	6	Rx	3	AF FE 00	[RQST] Request PGN: FEAFp
ParameterGroupNumber				65199	FEAF PGN which is requested by the Request message								
1.902739	CAN 1	F023p	A1SCR...	CAN Frame	Engine1		00	--	6	Rx	8	00 00 FF FF FF 0C FF FF	
1.903948	CAN 1	FEF1p	CCVS1	CAN Frame	Engine1		00	--	6	Rx	8	C7 00 00 06 01 00 00 C0	
1.905084	CAN 1	FF55p		CAN Frame	ManagementComputer		27	--	6	Rx	8	FC FF FF FF FF FF FF FF	
1.906292	CAN 1	FECAp	DM1	CAN Frame	BrakesSystemController		0B	--	6	Rx	8	C0 FF 00 00 00 00 FF FF	
1.907776	CAN 1	FE5Ap	ASC1	CAN Frame	SuspensionSystemController1		2F	--	3	Rx	8	11 00 00 00 D3 F5 FF F0	
1.909212	CAN 1	FE70p	CVW	CAN Frame	BrakesSystemController		0B	--	6	Rx	8	FF FF 28 05 FF FF FF FF	
PoweredVehicleWeight				655350	FFFF Total mass imposed by the tires of the powered vehicle on the road surface. Does not include the t...								
GrossCombinationVehicleWeight				13200 kg	528 The total weight of the truck and all attached trailers.								
1.910369	CAN 1	E800p	ACKM	CAN Frame	Engine1		00	FF	6	Rx	8	01 FF FF FF FF AF FE 00	[ACKM] Negative PGN: FEAFp Address: 255
ControlByte				1	1								
GroupFunctionValue				255	FF Group Function of PGN being acknowledged. This field identifies for a device the specific group functi...								
AddressNegativeAcknowledgement				255	FF								
ParameterGroupNumber				65199	FEAF PGN which is requested by the Request message								

5. Transportní protokol

Transportní protokol

Uvažujme CAN Classic, nikoliv CAN FD.

- **Zajišťuje přenos PGN, jejichž obsah je delší než 8 B (nevejde se do 1 CAN zprávy).**
- Lze přenést max. 1785 B.
- Data jsou rozdělena do více zpráv (tzv. paketů).



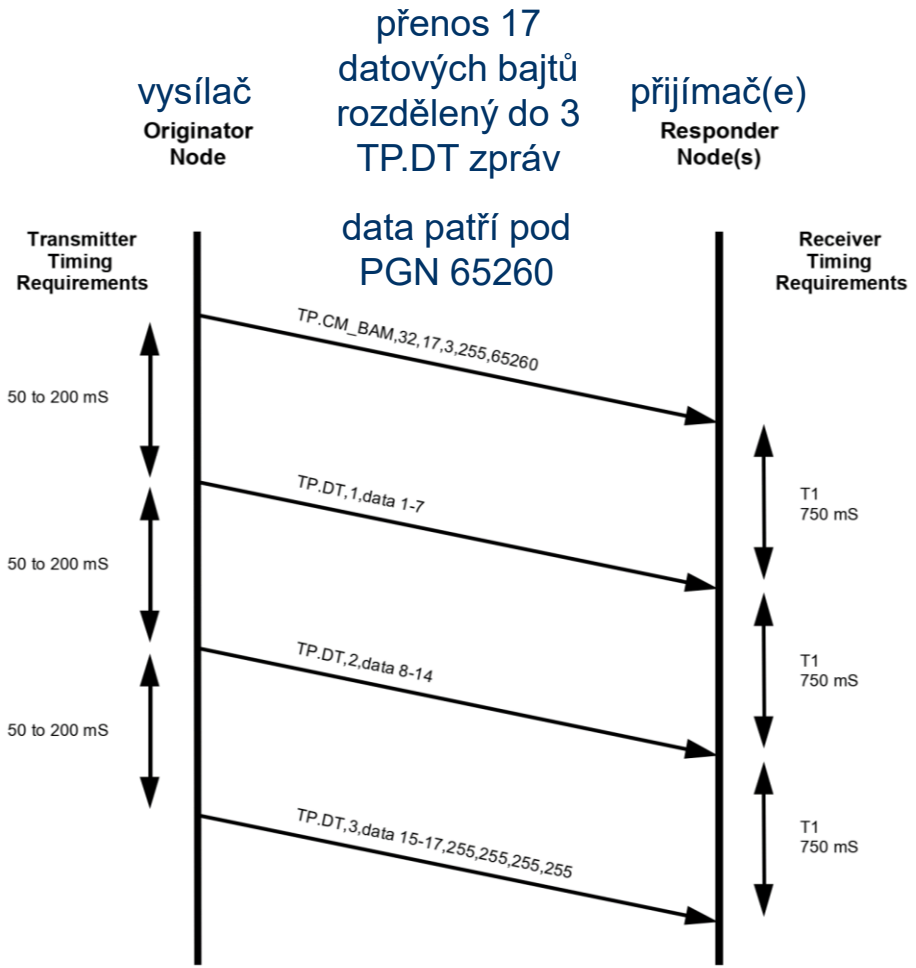
- Transportní protokol používá za tímto účelem speciální PGN
  - **TP.CM** (**Connection Management**) – pro navázání a udržení spojení
  - **TP.DT** (**Data Transfer**) – pro přenos samotných dat
- Existují dva různé transportní protokoly dle typu spojení
  - **Broadcast** (zpráva je určena pro všechny, bez potvrzování příjmu zprávy)
  - **Peer-to-peer** (komun. pouze mezi dvěma ECU, s potvrzováním příjmu zprávy)

Transportní protokol

- **Broadcast**
  - Pro zahájení spojení se používá **TP.CM_BAM** (**Broadcast Announcement Message**) zpráva (PGN 0x00EC00)
    - **Význam: „pozor začínám vysílat multipaketovou zprávu“**
  - Poté následují pakety **TP.DT** (PGN 0x00EB00), které již přenášejí samotná data v min. intervalech 50 ms.
  - Bez potvrzování příjmu.



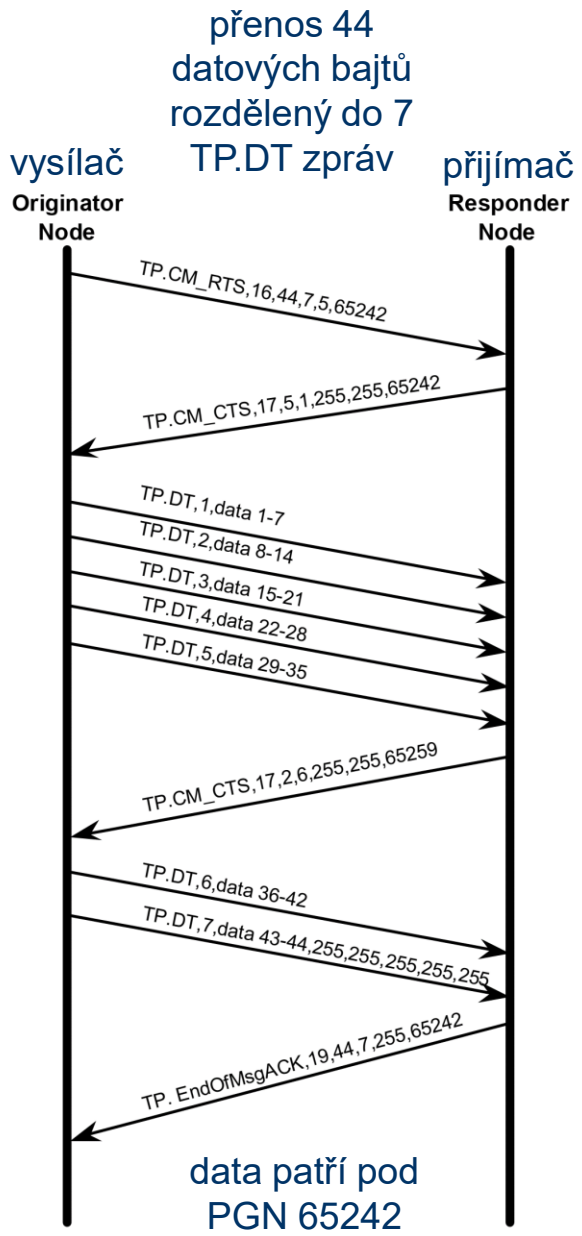
V úvodní **TP.CM_BAM** zprávě je uveden celkový počet paketů. V 1. datovém bajtu každého **TP.DT** paketu je vždy uvedeno číslo paketu. Příjemce tak může zkontrolovat zda přijmul zprávu celou.



zdroj: [8]

Transportní protokol

- **Peer-To-Peer**
  - Přenos dat pouze **mezi dvěma ECU.**
  - **Potvrzovaná komunikace.**
  - Vysílač zahájí spojení odesláním zprávy **TP.CM_RTS** (**request to send**).
    - **Význam: „pozor začínám vysílat multipaketovou zprávu“**
  - Přijímač odpovídá zprávou **TP.CM_CTS** (**clear to send**).
    - **Význam: „pošli mi další paket“**
  - Poté následují pakety **TP.DT**, které již přenášejí samotná data v min. intervalech 50 ms.
  - Po dokončeném příjmu všech paketů přijímač ukončí přenos zprávou **TP.CM_EndOfMsgACK**.

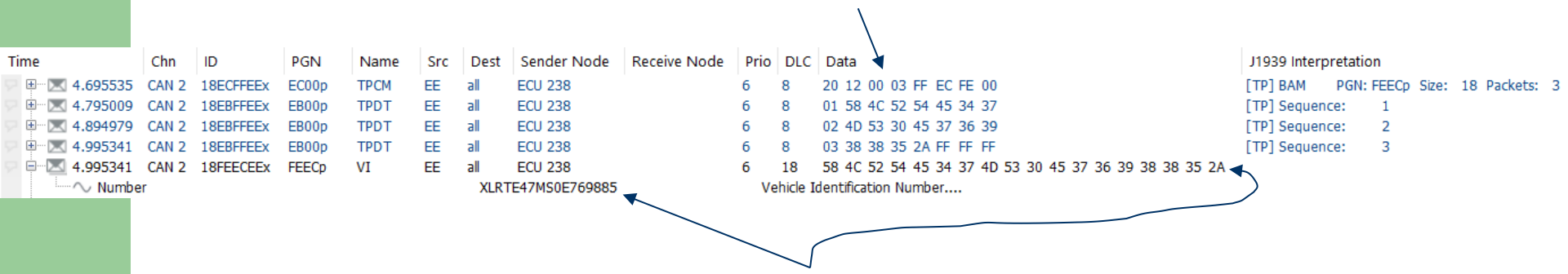


zdroj: [8]

Transportní protokol

- **Příklad**
  - ECU 238 (0xEE) vysílá VIN kód vozidla (PGN 0x00FEEC), který je dlouhý 18 bajtů (17 znaků + poslední znak ‘*‘ jako oddělovač).
  - VIN kód se nevejde do jedné CAN zprávy (8 bajtů), proto je odeslán ve 3 zprávách (paketech) pomocí transportního protokolu.
  - Přenos je realizován jako broadcast.

modře jsou TRACE okně označeny zprávy TP.BAM a TP.DT transportního protokolu



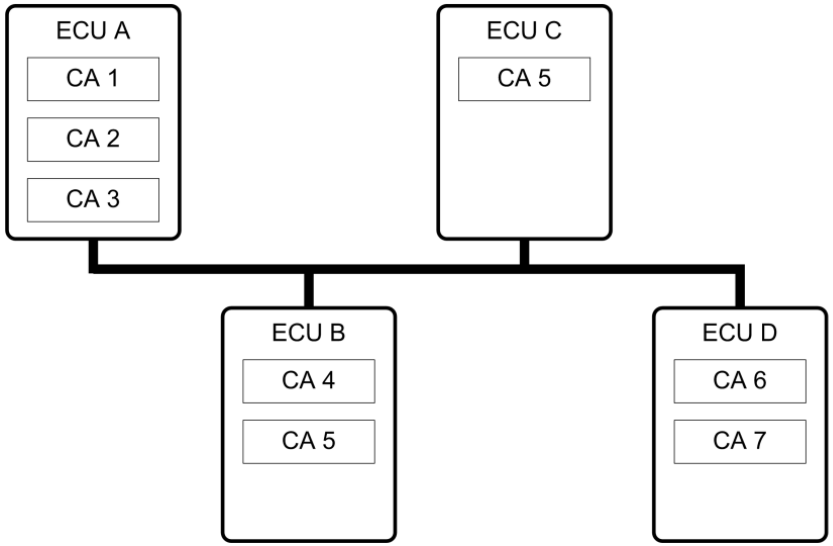
Time	Chn	ID	PGN	Name	Src	Dest	Sender Node	Receive Node	Prio	DLC	Data	J1939 Interpretation
4.695535	CAN 2	18ECFFEEx	EC00p	TPCM	EE	all	ECU 238		6	8	20 12 00 03 FF EC FE 00	[TP] BAM PGN: FEECp Size: 18 Packets: 3
4.795009	CAN 2	18EBFFEEx	EB00p	TPDT	EE	all	ECU 238		6	8	01 58 4C 52 54 45 34 37	[TP] Sequence: 1
4.894979	CAN 2	18EBFFEEx	EB00p	TPDT	EE	all	ECU 238		6	8	02 4D 53 30 45 37 36 39	[TP] Sequence: 2
4.995341	CAN 2	18EBFFEEx	EB00p	TPDT	EE	all	ECU 238		6	8	03 38 38 35 2A FF FF FF	[TP] Sequence: 3
4.995341	CAN 2	18FEECEEx	FEECp	VI	EE	all	ECU 238		6	18	58 4C 52 54 45 34 37 4D 53 30 45 37 36 39 38 38 35 2A	
Number							XLRTE47MS0E769885				Vehicle Identification Number....	

SW CANalyzer automaticky skládá přijímané pakety transportního protokolu do jednoho výsledného, takže v TRACE okně vidíme výsledné PGN se všemi daty (tj. celý VIN kód).

6. Síťový management

Network Management (J1939-81)

- **Pro dynamické přidělování adres zařízením (Plug and Play)**
  - Před tím, než zařízení může používat svojí adresu, jí musí na sběrnici registrovat (informovat, že jí chce používat) a případně ji musí změnit, pokud jí používá někdo jiný ⇒ **Address Claiming Procedure**
- Tato funkce není povinná. Na vozidle mají všechny ECU (CA) obvykle pevné adresy, známé již v době návrhu sítě (nehrozí konflikt adres).
- Dynamické přidělování adres je vhodné použít u zařízení, které se do sítě připojují/odpojují později (přípojná vozidla, diagnostické a servisní přístroje).
- **Jedna funkce = jedna CA** (**Controller Application**)
  - Každá **CA** má svojí adresu na síti (SA = Source Address).
- **Jedna ECU může plnit více funkcí (má více CA)**
  - Např. SA 0x00: Engine #1
    SA 0x0F: Engine retarder



Network Management (J1939/81)

- **Identifikace CA na síti**
  - Adresa – pro zajištění unikátnosti CAN-ID zpráv na síti.
  - Jméno (NAME) – pro jednoznačné určení funkce CA na síti.

- **Adresa**
  - Dle účelu CA (Controller Application).
  - CA má přidělenu defaultní (preferovanou) adresu, kterou se pokusí získat.
    - Pokud se jí to nepodaří, musí se pokusit získat jinou (volnou) adresu.
    - Pokud jiná adresa není volná, tak CA (ECU) musí přestat na síti vysílat.
  - Preferované adresy jsou standardizovány SAE.
    - Např. motor má adresu 0, převodovka adresu 3, atd…
  - Max. 254 adres v rozsahu 0 až 253.
    - Rezervované adresy:
      - 254 (0xFE) je NULL (používá se v proceduře získávání adresy)
      - 255 (0xFF) je broadcast (zpráva určená pro všechny CA (ECU))
    - Adresy přidělované při konfliktu adres:
      - 128 (0x80) až 247 (0xF7)

Network Management (J1939-81)

- **Adresa**
  - **Preferované adresy a jejich rozsahy**

Industry Group	Preferred Address Range
Global (platí pro všechny Industry Groups)	0 – 84 Assigned 85 – 127 Reserved 248, 252 – 255 Reserved
Industry group #1 – On-highway Equipment	128 – 160 Dynamic 161 – 247 Assigned
Industry group #2 – Agricultural and Forestry Equipment	128 – 207 Dynamic 208 – 248 Reserved
Industry group #3 – Construction Equipment	128 – 207 Dynamic 208 – 247 Reserved
Industry group #4 – Marine Equipment	128 – 207 Dynamic 208 – 247 Reserved
Industry group #5 – Industrial, Process Control, Stationary Equipment	128 – 207 Dynamic 208 – 247 Reserved

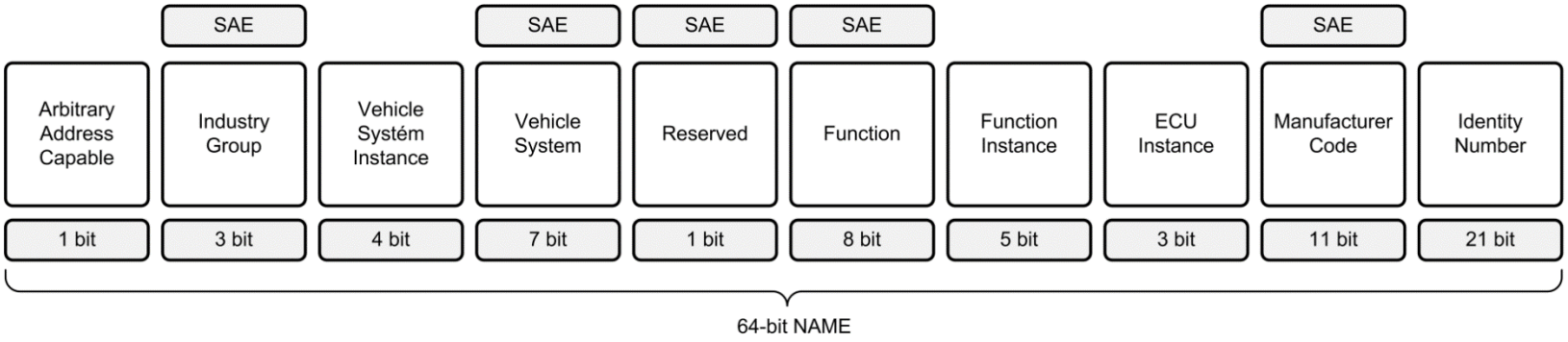


Adresy z rozsahu „Dynamic“ jsou určeny pro přidělování adres při konfliktu adres.

Address	Controller
0 (0x00)	Engine #1
1 (0x01)	Engine #2
2 (0x02)	Turbocharger
3 (0x03)	Transmission #1
4 (0x04)	Transmission #2
5 (0x05)	Shift Console – Primary
6 (0x06)	Shift Console – Secondary
7 (0x07)	Power Take Off
8 (0x08)	Axle - Steering
9 (0x09)	Axle - Drive #1
10 (0x0A)	Axle - Drive #2
11 (0x0B)	Brakes – System Controller
…	
15 (0x0F)	Retarder – Engine
16 (0x10)	Retarder – Driveline
…	
249 (0xF9)	Off Board Diagnostic-Service Tool #1

Network Management (J1939/81)

- **NAME**
  - 64-bit číslo pro jednoznačnou identifikaci CA (každé CA má jiné NAME).
  - Definuje hlavní účel, který plní CA na dané adrese.
    - Typ vozidla, např. tahač, návěs, secí stroj apod.
    - Funkce, např. řízení motoru, brzd apod.
    - Atd…
  - Skládá se z 10-ti položek z nichž některé definuje SAE a některé určuje výrobce.
  - **Mimo definice hlavní funkce CA se NAME používá v situaci, kdy více CA soupeří o jednu adresu** → adresu získá CA s nižší číselnou hodnotou NAME (tedy s vyšší prioritou).

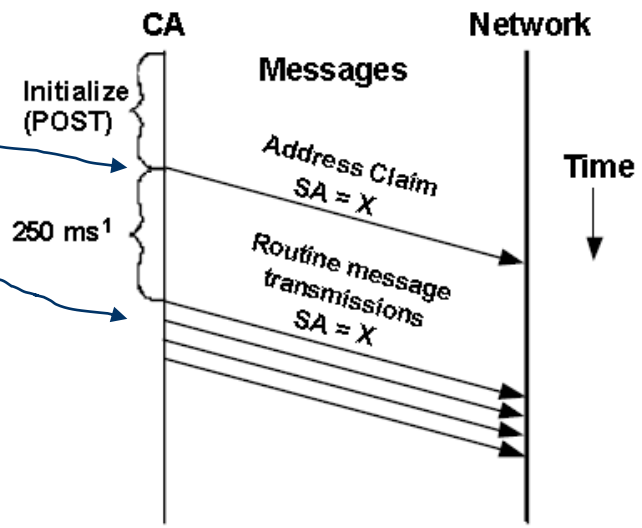


Network Management (J1939-81)

- **Address Claim Procedure**
  - Postup při získávání adresy
  - Probíhá při připojení ECU k síti
  - **Průběh**
    - CA v ECU odešle po zapnutí zprávu **Address Claimed message** se žádanou adresou (SA=0x00..0xFD) a svým NAME jako broadcast (DA=0xFF) a čeká definovaný čas (250 ms).
    - Pokud nezjistí konflikt s jinou CA, tak může adresu použít a zahájit běžnou komunikaci po sběrnici.

ACL (PGN 0x00EE00) (peer-to-peer)

Address Claimed message	
CAN ID	8 bajtů dat
0x18EEFFSA	64-bit NAME odesílatele



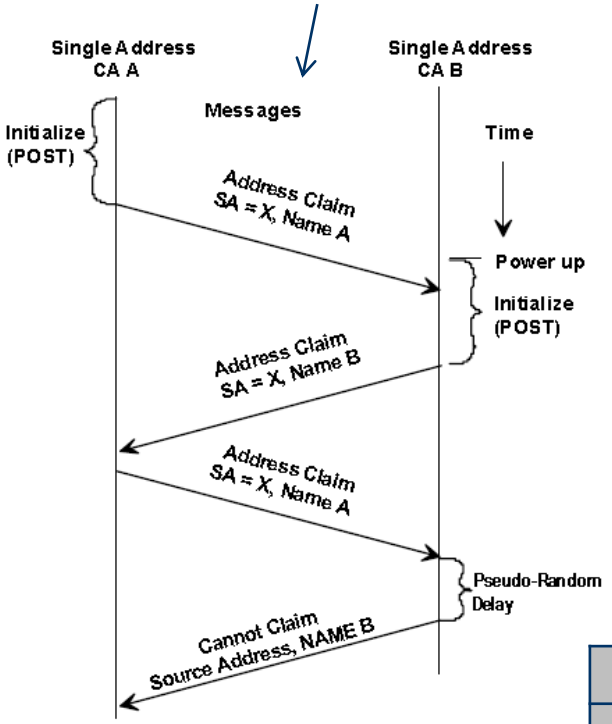


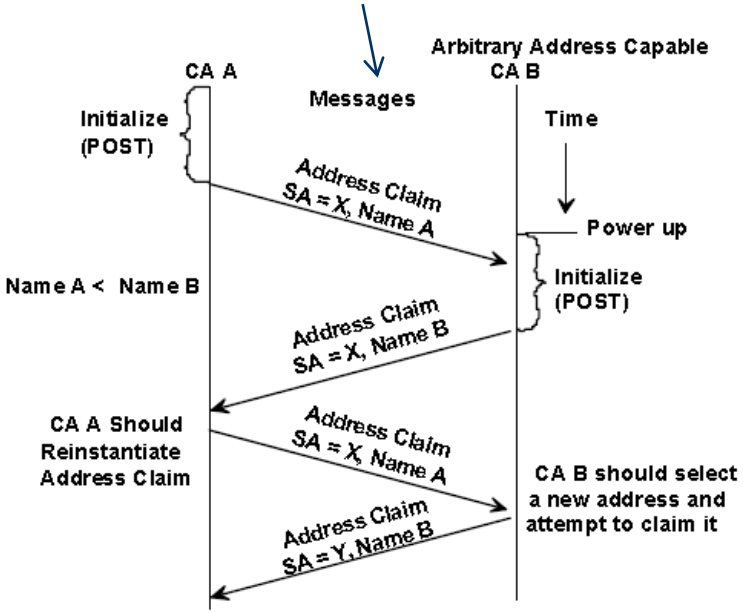
SA = Source Address
DA = Destination Address

zdroj: [9]

Network Management (J1939-81)

- Pokud další CA již používá tuto adresu, tak nastane konflikt.
- **CA s vyšší prioritou NAME získá adresu**.
- Druhá CA musí odeslat zprávu **Cannot Claim Address** se zdrojovou adresou SA=0xFE (NULL) pokud není schopna si zažádat o další adresu (obr. vlevo) nebo si zažádá o další adresu (obr. vpravo).





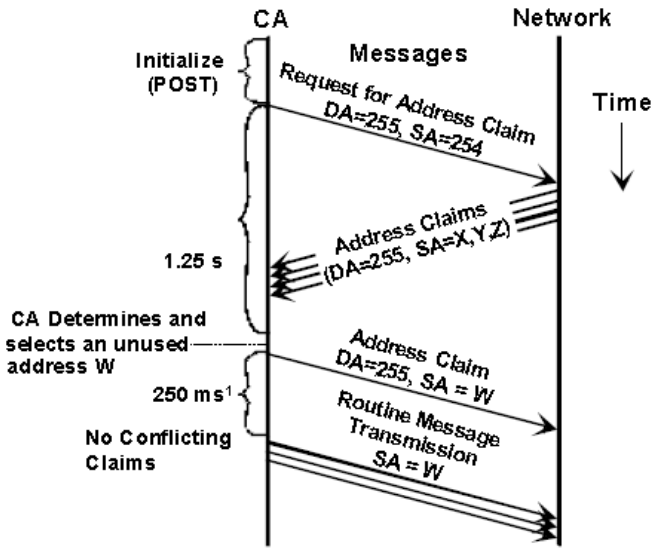
Cannot Claim Address	
CAN ID	8 bajtů dat
0x18EEFFFE	64-bit NAME odesílatele

zdroj: [9]

Network Management (J1939-81)

- **Request for Address Claimed**
  - Vyžádání adresy a NAME jakékoliv CA nebo všech CA na síti.
  - Lze použít na
    - Zjištění, zda je adresa dostupná.
    - Zjištění přítomnosti nějaké konkrétní nebo všech ECU (resp. CA) na síti.
  - Používají typicky ECU (CA), které se připojují k síti až později (např. návěs, diag. tester).
  - Po obdržení této zprávy dotyčná CA odešle svojí adresu a NAME.

Time	Chn	PGN	Name	Event Type	Send Node	Receive Node	Src	Dest	Prio	Dir	DLC	Data
9.815...	CAN 1	EA00p	RQST	CAN Frame	OffBoardDiagnosticTool1		F9	FF	6	Tx	3	00 EE 00
ParameterGroupNumber		60928		EE00 PGN which is requested by the Request message								





V obrázku je znázorněno zjištění adres všech CA na síti.

SA=254 (0xFE) znamená, že CA nemá zatím žádnou svojí validní adresu

DA=255 (0xFF) znamená broadcast (zprávu určenou pro všechny CA na síti)

zdroj: [9]

7. Diagnostika

Diagnostika (J1939-73)

- **J1939-73 Application layer - Diagnostics**
  - Hlavní funkce
    - Zajištění legislativních požadavků na palubní diagnostiku (OBD).
    - Sledování stavu komponent a hlášení chyb.
    - Vyčtení a mazání pamětí závad.
    - Flashování ECU.
  - Diagnostický konektor
    - 9-pin kulatý konektor Deutsch HD10-9-1939



  - Způsob přenosu diagnostických dat
    - Pomocí **DM zpráv** (**Diagnostic Messages**), **DM1** až **DM60**.
    - Ne všechny jsou povinné a tudíž implementované v každé ECU.
  - Kódy závad (**DTC** = **Diagnostic Trouble Codes**)
    - Číslo parametru (např. poloha plynového pedálu – SPN91).
    - Typ závady (např. napětí nad horní mezí).
    - Počet výskytů závady.

zdroj: [10]

Diagnostika (J1939-73)

- **Kód chyby (DTC = Diagnostic Trouble Code)**
  - Skládá se ze 4 částí
    - Suspect Parameter Number (**SPN**) – **co se porouchalo** (19 bit)
    - Failure Mode Identifier (**FMI**) – **jak se to porouchalo** (5 bit)
    - Occurrence Count (**OC**) – **počet výskytů chyby** (7 bit)
    - SPN Conversion Method (**CM**) – určuje pořadí bajtů SPN ve zprávě (1 bit)

SPN	FMI	OC	SPN: FMI text
172	3	5	Air Inlet Temperature: Napětí nad normálem nebo zkrat na plus
157	16	126	Engine Injector Metering Rail 1 Pressure: Data platná ale běžným rozsahem
3242	3	1	???: Napětí nad normálem nebo zkrat na plus
4783	9	4	???: Neobvyklá perioda (obnovení)
5576	8	1	???: Neobvyklá frekvence nebo šířka pulsu

J1939 SPN	*J1939 FMI*	*DIAGNOSTIC CODE DESCRIPTION*
27	2	EGR POSITION SENSOR CIRCUIT DATA INCORRECT
27	3	EGR VALVE POSITION CIRCUIT SHORTED HIGH
27	4	EGR VALVE POSITION CIRCUIT SHORTED LOW

zdroj: [10]

Diagnostika (J1939-73)

- **FMI kód**
  - Upřesňuje typ chyby

FMI Code	Description
0	Data Valid but above Normal Operating Range
1	Data Valid but below Normal Operating Range
2	Data Erratic, Intermittent or Incorrect
3	Voltage above Normal or Shorted to High Source
4	Voltage below Normal or Shorted to Low Source
5	Current below Normal or Open Circuit
6	Current above Normal or Grounded Circuit
7	Mechanical System Not Responding or out of adjustment
8	Abnormal frequency or pulse width or period
9	Abnormal Update Rate
10	Abnormal Rate of Change
11	Root Cause Not Know
12	Bad Intelligent Device or Component
13	Out of Calibration
14	Special Instructions
15	Data Valid But Above Normal Operating Range (Least Severe Level)
16	Data Valid But Above Normal Operating Range (Moderate Sever Level)
17	Data Valid But Below Normal Operating Range (Least Severe Level)
18	Data Valid But Below Normal Operating Range (Moderate Severe Level)

Uvedený výčet FMI není kompletní, je jich celkem 32 (0-31) a část z nich není použita.

zdroj: [10]

Diagnostika (J1939-73)

- **Kód chyby (DTC = Diagnostic Trouble Code)**
  - Uložení DTC v datové části DM zprávy

DTC																													
Byte 3 (Least Significant Byte of SPN)								Byte 4 (Second Byte of SPN)								Byte 5 (3 Most Significant Bits of SPN)					Byte 6							
SPN																		FMI					CM	OC					
8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
1	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	1	0

SPN 1208 : Pre-filter oil pressure

SPN = 000 00000100 10111000 = 0x0040B8 = 1208
FMI = 00011 = 0x3 = 3
OC = 0001010 = 0xA = 10
CM = 0

  - Bit CM určuje pořadí pořadí bajtů SPN

zdroj: [4]

Diagnostika (J1939-73)

Kontrolky závad – jejich stav se také přenáší v DM zprávách



– Malfunction Indicator Lamp

- Svítí pokud je aktivní DTC, které se vztahuje k emisím výf. spalin.



– Red Stop Lamp

- Svítí pokud je závada natolik závažná, že by měl řidič zastavit a nepokračovat dál v jízdě.
- Např. přehřátí motoru, porucha brzd.



zdroj: [6]



– Amber Warning Lamp

- Svítí pokud se vyskytne nějaká závada, ale vozidlo (motor) kvůli ní není třeba okamžitě zastavit.



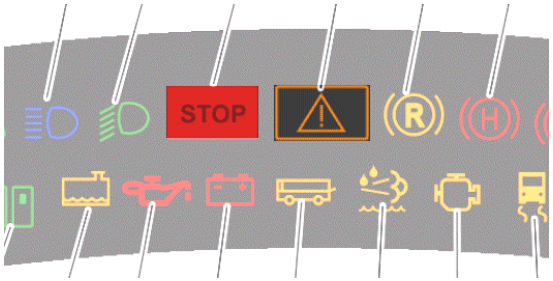
- Bliká pokud je závada závažná (může závažně poškodit vozidlo).

Každá z kontrolek může trvale svítit nebo blikat.
Každá ECU (CA) má své vlastní kontrolky.
Jedná se o SW kontrolky, které mohou/nemusí ovládat reálné HW kontrolky na přístrojové desce.

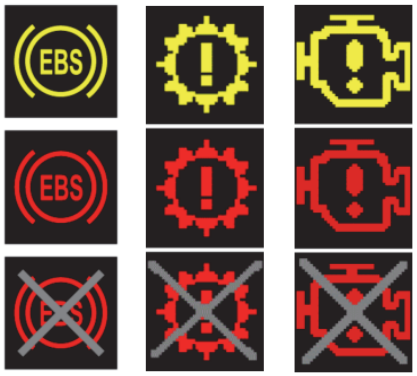
Diagnostika (J1939-73)



Kontrolky závad



Závada motoru



Aktivní - **varování**

Aktivní - **STOP**

Aktivní - **nekomunikuje**



Svítí při požadavku na rozsvícení červené kontrolky DM1

- STOP motor
- STOP převodovka
- STOP ABS - EBS
- STOP retardér
- STOP ECAS
- STOP HORTON
- STOP TIRE
- STOP EAPC
- STOP FIRE

Varování - svítí při rozsvícení signalizace:



- Varovná světla
- Motor, převodovka
- EBS
- ECAS
- TIRE
- TCO
- EAPC
- Horton
- CAN_FIRE

Kontrolky diagnostiky svou barvou indikují úroveň poruchy:

1. Lehká, střední závada - žluté Varování
2. Těžká závada - červené STOP - **Vozidlo nesmí pokračovat v jízdě.**
3. Nekomunikuje - červená přeškrtnutá - **Řídicí jednotka zařízení nereaguje. Pokud je to možné, odstavte vozidlo - vozidlo nemůže pokračovat v jízdě!**

Výňatek z návodu k obsluze k autobusu SOR NB18.

zdroj: [16]

Diagnostika (J1939-73)

- **DM zprávy**
  - Obdoba diagnostických módů EOBD u osobních vozidel.
  - **DM1** - Active Diagnostic Trouble Codes
  - **DM11** - Diagnostic Data Clear/Reset for Active DTCs
  - **DM2** - Previously Active Diagnostic Trouble Codes
  - **DM3** - Diagnostic Data Clear/Reset for Previously Active DTCs
  - **DM4** - Freeze Frame Parameters →
    - Pro každou chybu jeden Freeze Frame.

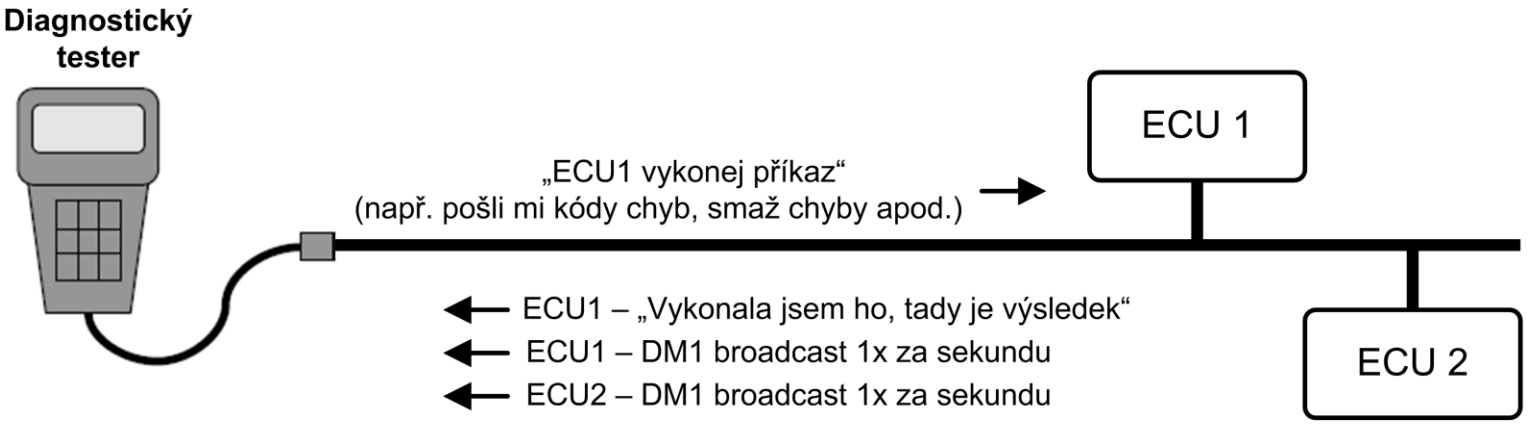
Byte:	6	Engine Torque Mode (SPN 899)
Byte:	7	Boost (SPN 102)
Byte:	8	Engine Speed (SPN 190) (LSB)
Byte:	9	(MSB)
Byte:	10	Engine % Load (SPN 92)
Byte:	11	Engine coolant temperature (SPN 110)
Byte:	12	Vehicle Speed (SPN 86) (LSB)
Byte:	13	(MSB)
Byte:	14-n	Manufacturer Specific information

  - **DM5** - Diagnostic Readiness 1
    - Stav sledovaných systémů vozidla
  - atd…
  - Každá DM zpráva má své PGN
    - **Broadcast** (např. DM1, DM2, DM4 – vysílány při prvním výskytu chyby a následně periodicky např. 1x za sekundu)
    - **Peer-To-Peer** (např. DM11, DM3, DM5 – na vyžádání)
  - Každá ECU (CA) může posílat své DM zprávy
    - Protože každá CA má unikátní adresu.

zdroj: [4]

Diagnostika (J1939-73)

- **Komunikační model**



- ECU odesílá broadcastem diagnostické informace a odpovídá na přijatý příkaz
  - Naslouchá žádostem, které vysílá diagnostický tester.
  - Periodicky odesílá diagnostická data (např. DM1).
  - Může sama odeslat žádost na diagnostické informace do jiné ECU.
- Diagnostický tester
  - Žádá o diagnostické informace, pokud nejsou z ECU (CA) zasílány jako broadcast.
  - Musí znát adresu ECU (CA), ze které chce získat data.
  - ECU (CA) poskytne požadovaná data.
  - Přijímá broadcast diagnostické zprávy ze sítě (např. DM1).

zdroj: [11]

Diagnostika (J1939-73)

- **DM1 - Active Diagnostic Trouble Codes (PGN 65226 / 0x00FECA)**
  - Pomocí této zprávy se přenáší seznam všech aktuálních závad.
  - Obdoba paměti závad u osobních vozidel.
  - Každá ECU (CA) na síti reportuje aktuální závady.
  - Při výskytu více závad je zpráva moc dlouhá a pro přenos se používá transportní protokol – data jsou rozdělena do více paketů.
  - Obsah zprávy
    - Stav kontrolek závad
      - Malfunction Indicator Lamp
      - Red Stop Lamp
      - Amber Warning Lamp
      - Protect Lamp
    - DTC (chybový kód)
      - SPN, FMI, OC
  - Kdy a jak často DM1 zprávu každá ECU odesílá?
    - Při prvním výskytu závady a následně periodicky 1x za sekundu.

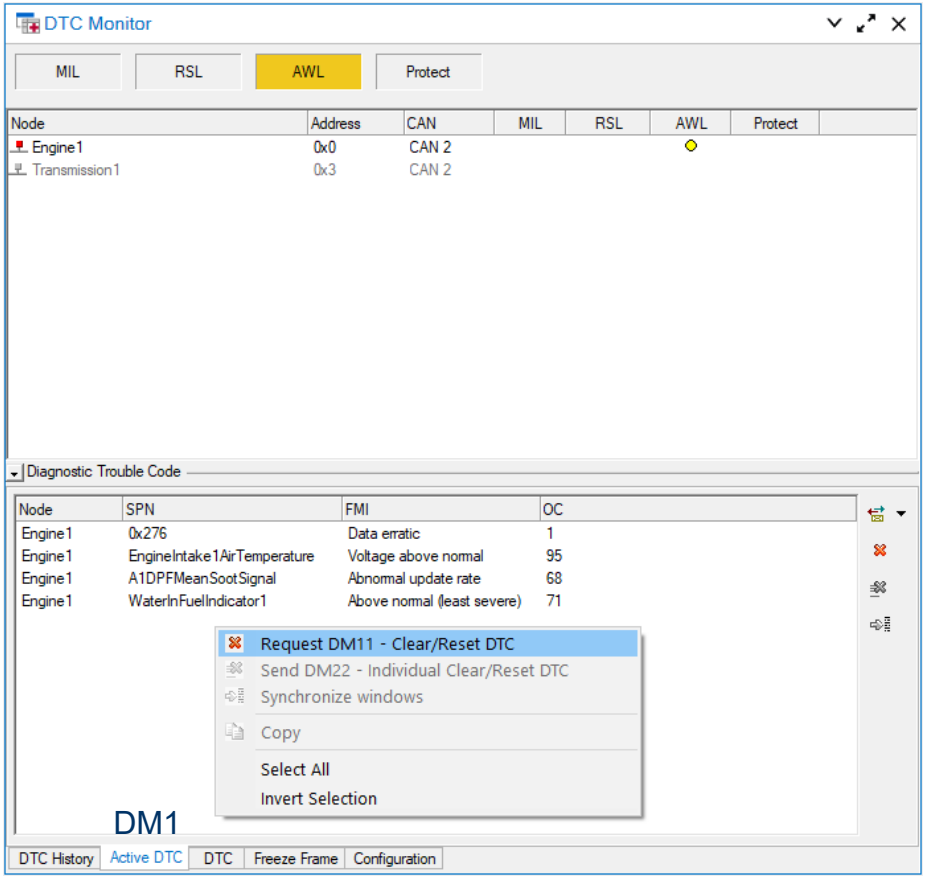
Diagnostika (J1939-73)

- **Zpráva DM1 v TRACE okně SW CANalyzer**
  - Na motoru jsou 4 aktuální závady. Přenáší se jako multipaket (18 B).
  - Pro každou závadu: SPN, FMI, OC, stav kontrolek

Time	Chn	ID	PGN	Name	Src	Dest	Sender Node	Receive Node	Prio	DLC	Data
216.901...	CAN 2	18FECA00x	FECAp	DM01	00	all	Engine1		6	18	04 FF 76 02 02 01 AC 00 03 5F AF 12 09 44 61 00 0F 47
SPN_High4							0	0	3 most significant bits of SPN (most significant at bit 8)		
SPN_High3							0	0	3 most significant bits of SPN (most significant at bit 8)		
SPN_High2							0	0	3 most significant bits of SPN (most significant at bit 8)		
SPN_High1							0	0	3 most significant bits of SPN (most significant at bit 8)		
SpnConversionMethod1							0	0	If set to '0', the document is converted according to version 4 of the J1939-73 doc		
SpnConversionMethod4							0	0	If set to '0', the document is converted according to version 4 of the J1939-73 doc		
SpnConversionMethod3							0	0	If set to '0', the document is converted according to version 4 of the J1939-73 doc		
SpnConversionMethod2							0	0	If set to '0', the document is converted according to version 4 of the J1939-73 doc		
OccurenceCount4							71	47	Contains the number of times a fault has been independently detected.		
OccurenceCount3							68	44	Contains the number of times a fault has been independently detected.		
OccurenceCount2							95	5F	Contains the number of times a fault has been independently detected.		
OccurenceCount1							1	1	Contains the number of times a fault has been independently detected.		
FMI4							Above normal (least severe)	F	The FMI defines the type of failure detected in the subsystem identified by an SPN.		
FMI3							Abnormal update rate	9	The FMI defines the type of failure detected in the subsystem identified by an SPN.		
FMI2							Voltage above normal	3	The FMI defines the type of failure detected in the subsystem identified by an SPN.		
FMI1							Data erratic	2	The FMI defines the type of failure detected in the subsystem identified by an SPN.		
FlashRedStopLamp							Unavailable / Do Not Flash	3	This parameter provides the capability to flash the RSL...		
FlashProtectLamp							Unavailable / Do Not Flash	3	This parameter provides the capability to flash the engine protect lamp...		
FlashMalfuncIndicatorLamp							Unavailable / Do Not Flash	3	This parameter provides the capability to flash the MIL...		
FlashAmberWarningLamp							Unavailable / Do Not Flash	3	This parameter provides the capability to flash the AWL...		
MalfunctionIndLampStatus							Lamp Off	0	A lamp used to relay only emissions-related trouble code information. This lamp is on		
AmberWarningLampStatus							Lamp On	1	This lamp is used to relay trouble code information that is reporting a problem with t		
RedStopLampState							Lamp Off	0	This lamp is used to relay trouble code information that is of a severe enough condit		
ProtectLampStatus							Lamp Off	0	This lamp is used to relay trouble code information that is reporting a problem with a		
SPN4							SPN 0x61 (WaterInFuelIndicator1)	61	16 least significant bits of SPN		
SPN3							SPN 0x12AF (A1DPFMeanSootSignal)	12AF	16 least significant bits of SPN		
SPN2							SPN 0xAC (EngineIntake1AirTemperature)	AC	16 least significant bits of SPN		
SPN1							SPN 0x276	276	16 least significant bits of SPN		

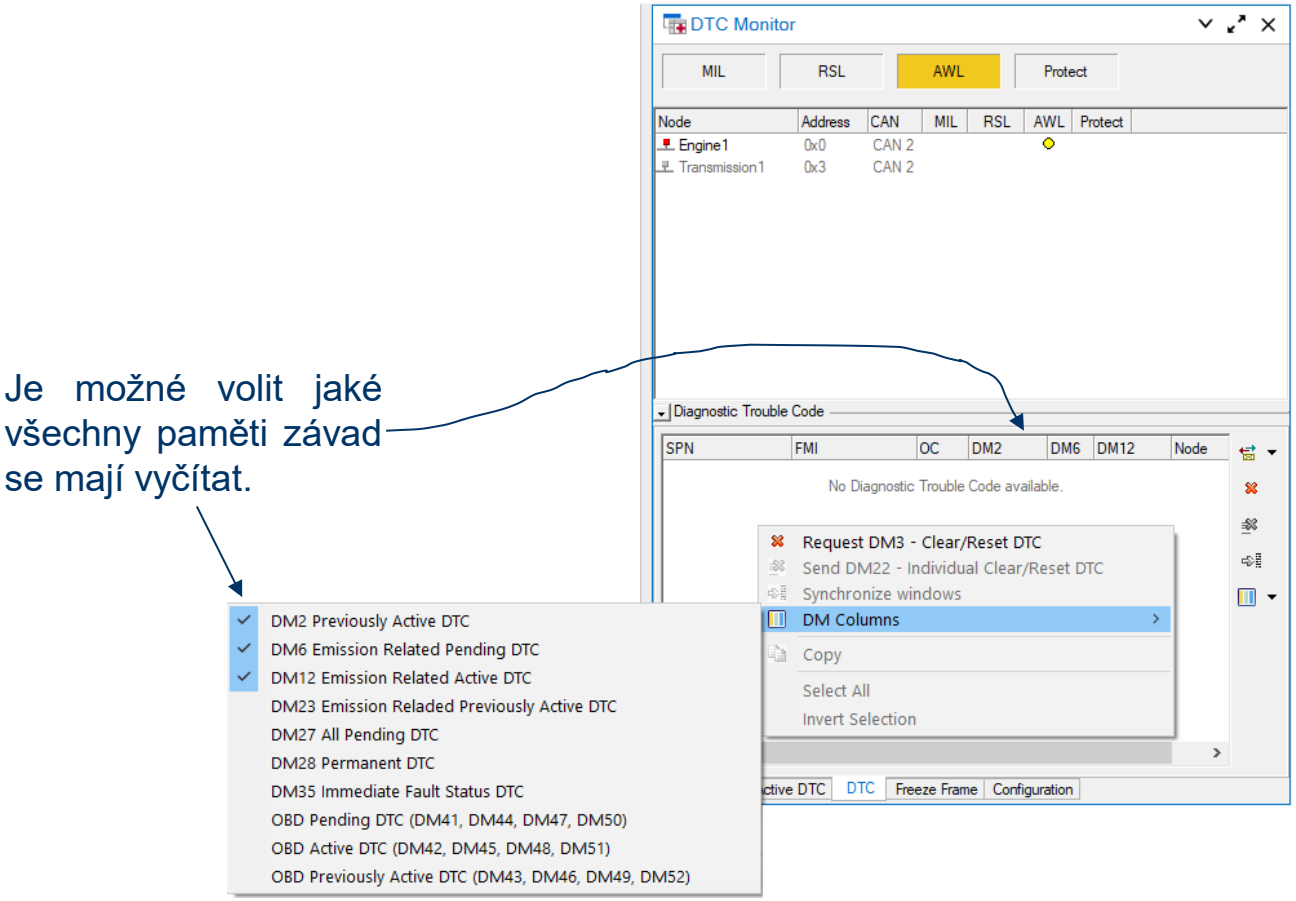
Diagnostika (J1939-73)

- **DTC monitor v SW CANalyzer**
  - Přehled o závadách v jednotlivých ECU (CA) na síti J1939.
  - Čtení a mazání závad.



Diagnostika (J1939-73)

- **DTC monitor v SW CANalyzer**
  - Přehled o závadách v jednotlivých ECU (CA) na síti J1939.
  - Čtení a mazání závad.

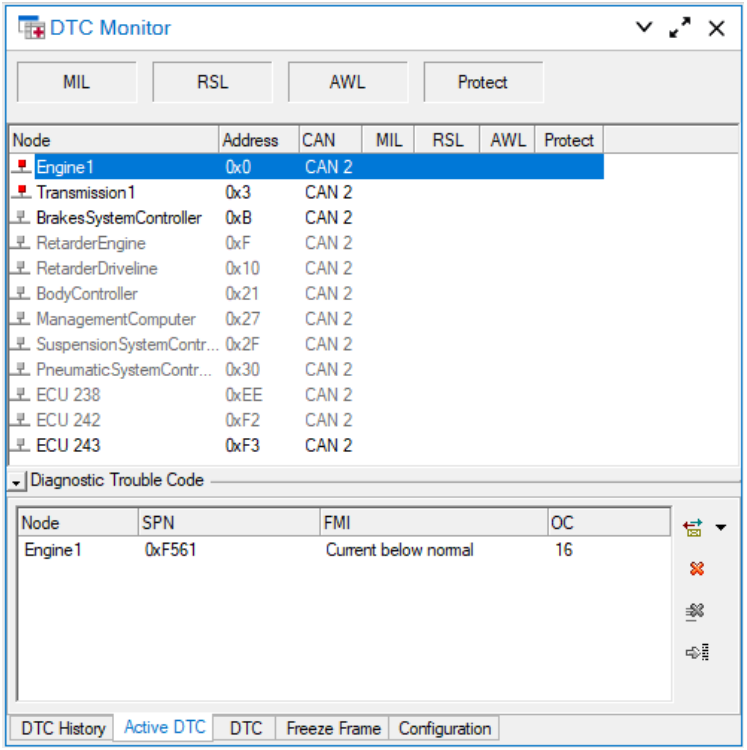


Diagnostika (J1939-73)

- **Více DM1 zpráv na sběrnici od různých ECU (CA)**
  - Každá ECU (CA) vysílá DM1 zprávu periodicky každou 1 sekundu jako broadcast.

Time	Chn	PGN	Name	Event Type	Sender Node	Receive Node	Src	Dest	Prio	Dir	DLC	Data length	Data
0.964884	CAN 2	FECAp	DM01	CAN Frame	Engine1		00	all	6	Rx	8	8	C3 FF 61 F5 05 10 FF FF
0.966968	CAN 2	FECAp	DM01	CAN Frame	Transmission1		03	all	6	Rx	8	8	C0 FF 20 15 0E 03 FF FF
0.965489	CAN 2	FECAp	DM01	CAN Frame	BrakesSystemController		0B	all	6	Rx	8	8	C0 FF 00 00 00 00 FF FF
0.960197	CAN 2	FECAp	DM01	CAN Frame	ECU 243		F3	all	6	Rx	8	8	C0 FF 00 00 00 00 FF FF

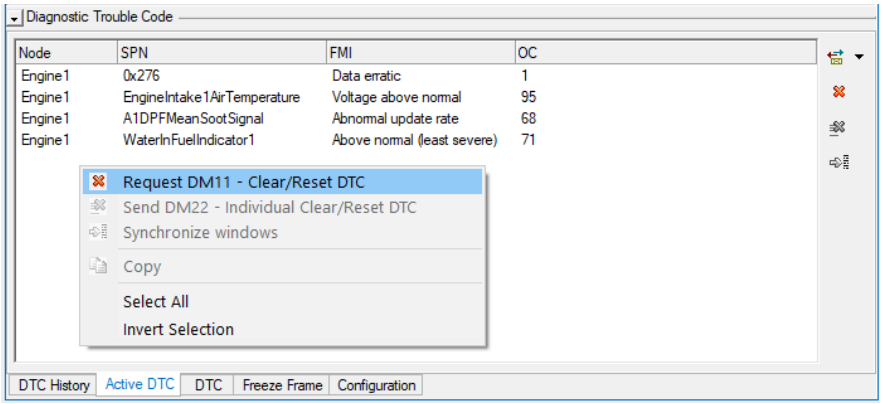
perioda [s]



Diagnostika (J1939-73)

- **DM11 – Clear Active Diagnostic Trouble Codes (PGN 65235 / 0x00FED3)**
  - Smazání právě aktivních DTC (mazání pamětí závad pod DM1).
  - Obdoba smazání paměti závad u osobních vozidel.
  - Posílá diagnostický tester do ECU jako žádost – pomocí REQUEST zprávy linkové vrstvy (PGN 59904, 0x00EA00).

Time	Chn	ID	PGN	Name	Src	Dest	Sender Node	Receive Node	Prio	DLC	Data	J1939 Interpretation
36.470440	CAN 2	18EA00F9x	EA00p	RQST	F9	00	OffBoardDiagnosticTool1	Engine1	6	3	D3 FE 00	[RQST] Request PGN: FED3p
ParameterGroupNumber	DM11	FED3	Whenever it is necessary to identify a Parameter Group in the data field of a CAN data frame, it ...									



  - ECU potvrdí dokončení operace – pomocí ACKNOWLEDGMENT zprávy linkové vrstvy (PGN 59392 (0xE800))
    - Výmaz proveden (Positive Acknowledge) / neproveden – různé důvody

9. Nástroje pro analýzu komunikace

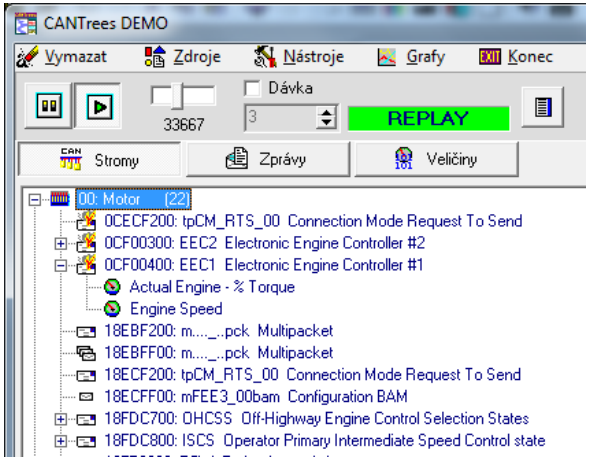
Nástroje pro analýzu komunikace

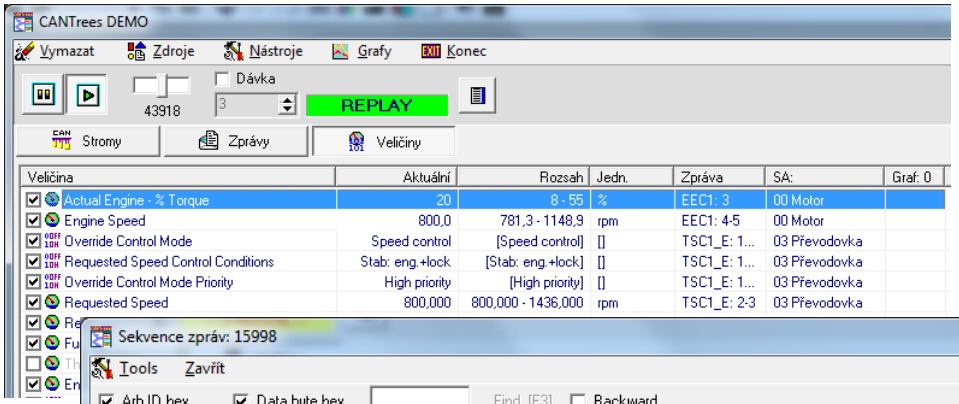
- **CAN analyzátor J1939 fy ELBAS s.r.o.**
  - Balík 10-ti programů pod názvem CAN Suite 6
  - Podpora 5-ti převodníků CAN-USB od různých firem
  - Funkce
    - On-line analýza – zachytávání zpráv, jejich logování
    - Off-line analýza uloženého záznamu
    - Automatická identifikace typu a obsahu zprávy, převod do fyzikálních jednotek, záznam mezních hodnot
    - Zobrazení zpráv do stromu, do seznamu, do grafu
    - Vyhodnocení diagnostických zpráv, textová interpretace SPN a FMI, mazání chyb v ECU
    - Uživatelsky editovatelná databanka PGN, SPN, šablony pro zobrazení veličin do grafů (zjednodušení rutinních operací při servisu)
    - Export záznamu do textového souboru a do MS Excel
    - Odeslání vybraného PGN – jednorázové, periodické
    - Zpětné přehrání nalogovaných zpráv (jejich odeslání do sítě se stejnými časovými odstupy jako v logu) – např. pro účely vývoje a testování komunikace ECU
    - A další…

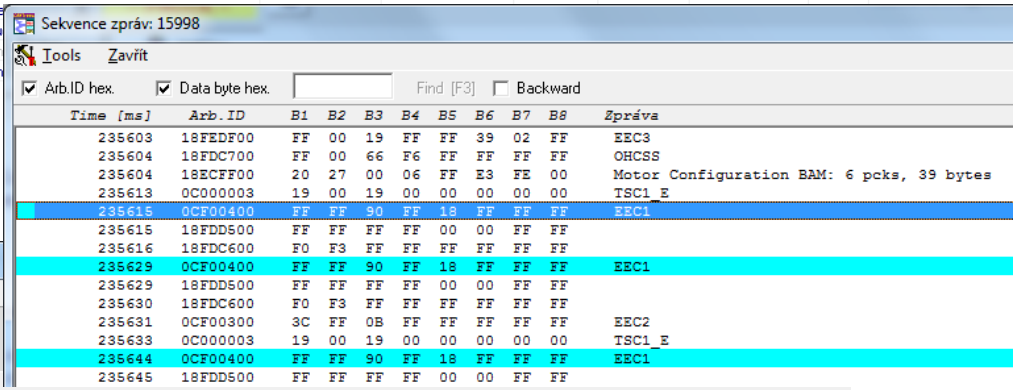


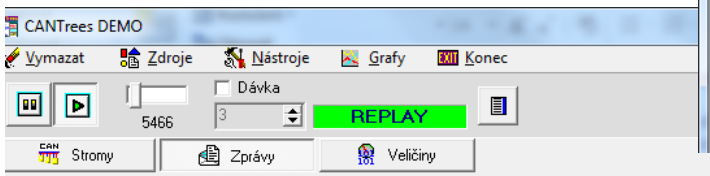
Nástroje pro analýzu komunikace

- **UCINT Suite 5 (předchůdce CAN Suite 6)**





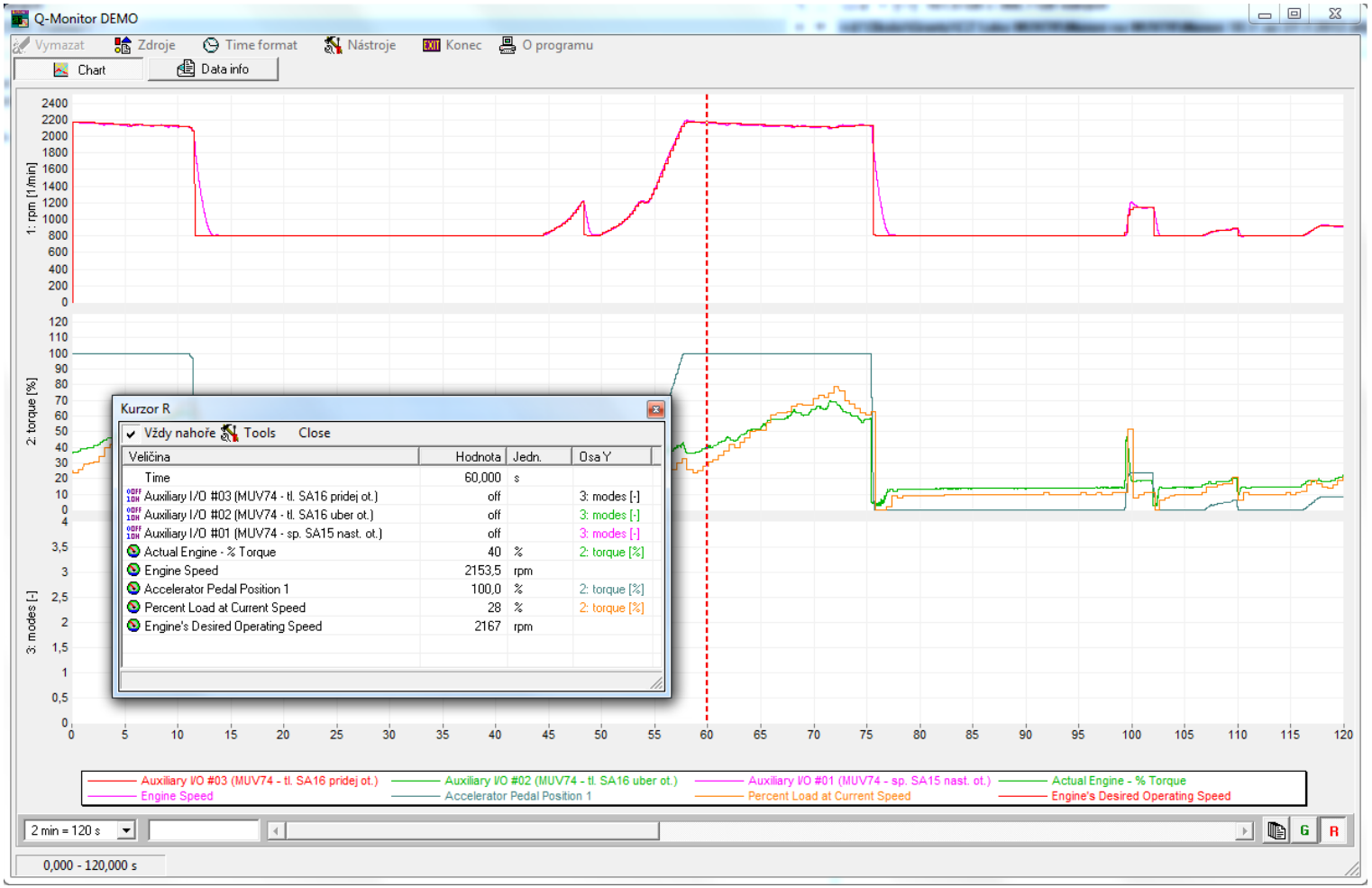




Arb.ID	E..	Prior	Page	res.	Zpráva	Jméno	P...	PGN...	Zdroj	Cíl	Č.	Měř.per.	Data
0C000003	x	3			TSC1_E	Torque/Speed Control (10-50 ms)	0000	0	03 Převodovka	00 Motor		20	19 00 19 00 00 00 00 00
0CF00300	x	3			EEC2	Electronic Engine Controller #2	F003	61443	00 Motor			60	3C FF 0D FF FF FF FF FF
0CF00400	x	3			EEC1	Electronic Engine Controller #1	F004	61444	00 Motor			15	FF FF 92 FA 18 FF FF FF
18EA00F2	x	6			RqtPGN	Request PGN	EA00	59904	F2 F2	00 Motor		3001	CB FE 00 7A 1B FF FF FF
18EA03FE	x	6			RqtPGN	Request PGN	EA03	59907	FE FE	03 Př...		5040	00 EE 00 15 63 00 00 00
18EA27FE	x	6			RqtPGN	Request PGN	EA27	59943	FE FE	27 EC...		5040	00 EE 00 BE 37 00 00 00
18EBF200	x	6			tpCM_RTSpck	Multipacket	EBF2	60402	00 Motor	F2 F2		16	00 FF AC 00 03 05 9D 00 10
18EBFF00	x	6			mFEE3_00pck	Multipacket	EBFF	60415	00 Motor			57	00 19 B7 C0 44 CD 40 1F C4
18EC00F2	x	6			tpCM_End_F2	End of Message Acknowledgment	EC00	60416	F2 F2	00 Motor		113	13 16 00 04 FF CB FE 00
18ECF200	x	6			tpCM_RTS_00	Connection Mode Request To Send	ECF2	60658	00 Motor	F2 F2		3000	10 16 00 04 FF CB FE 00
18ECFF00	x	6			mFEE3_00bam	Configuration BAM	ECFF	60671	00 Motor			510	20 27 00 06 FF E3 FE 00

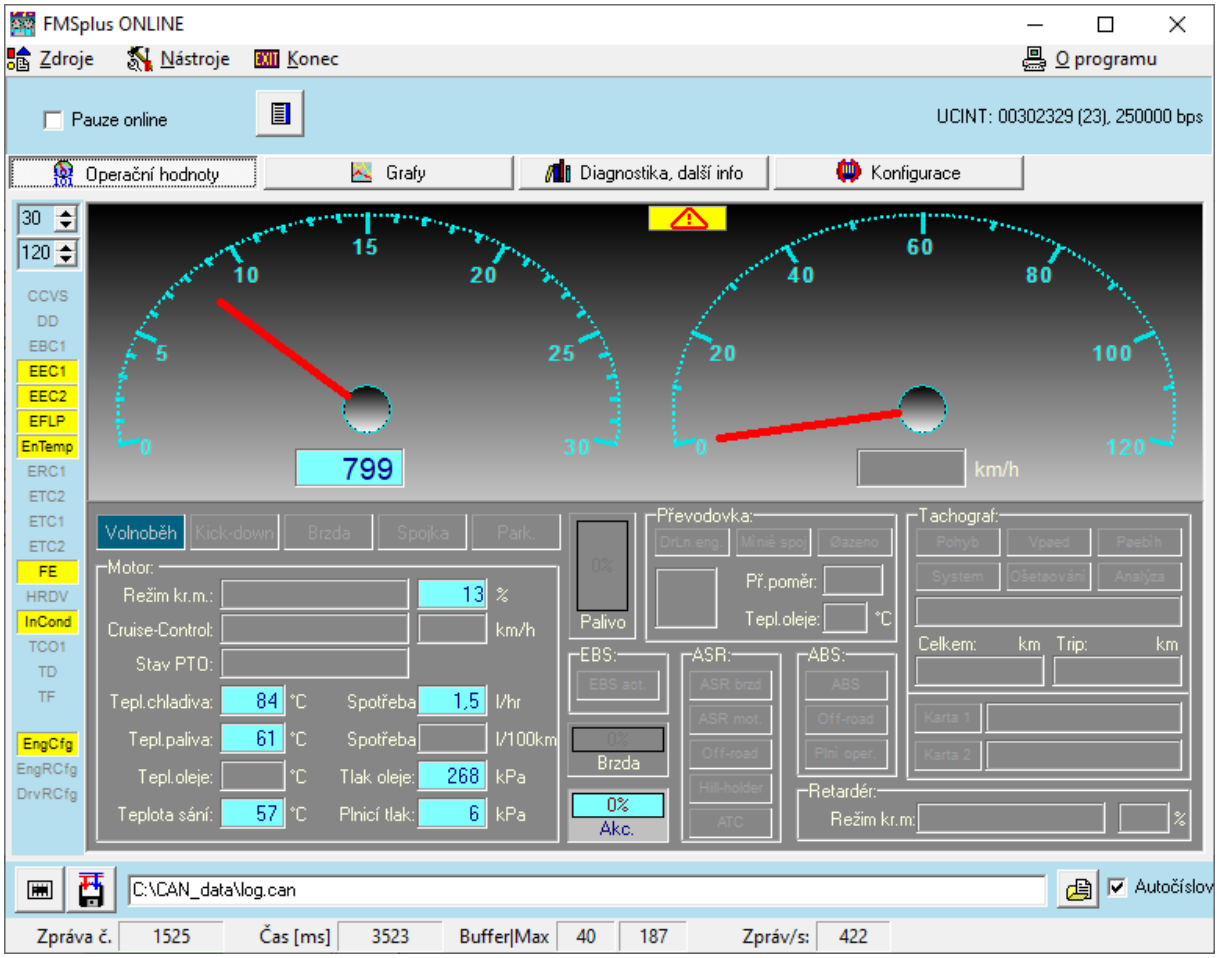
Nástroje pro analýzu komunikace

- Různé možnosti zobrazení (starší UCINT Suite 5)



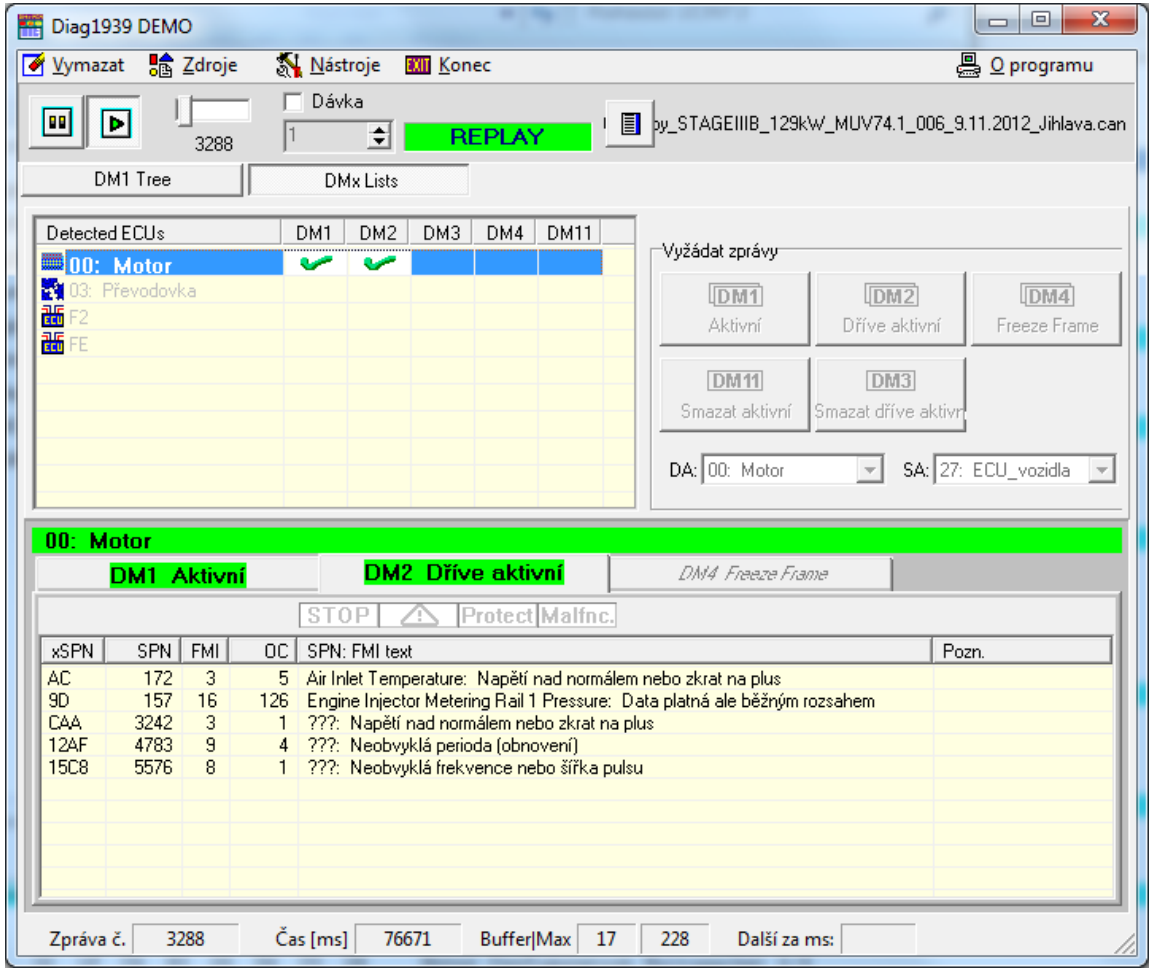
Nástroje pro analýzu komunikace

- Různé možnosti zobrazení (starší UCINT Suite 5)



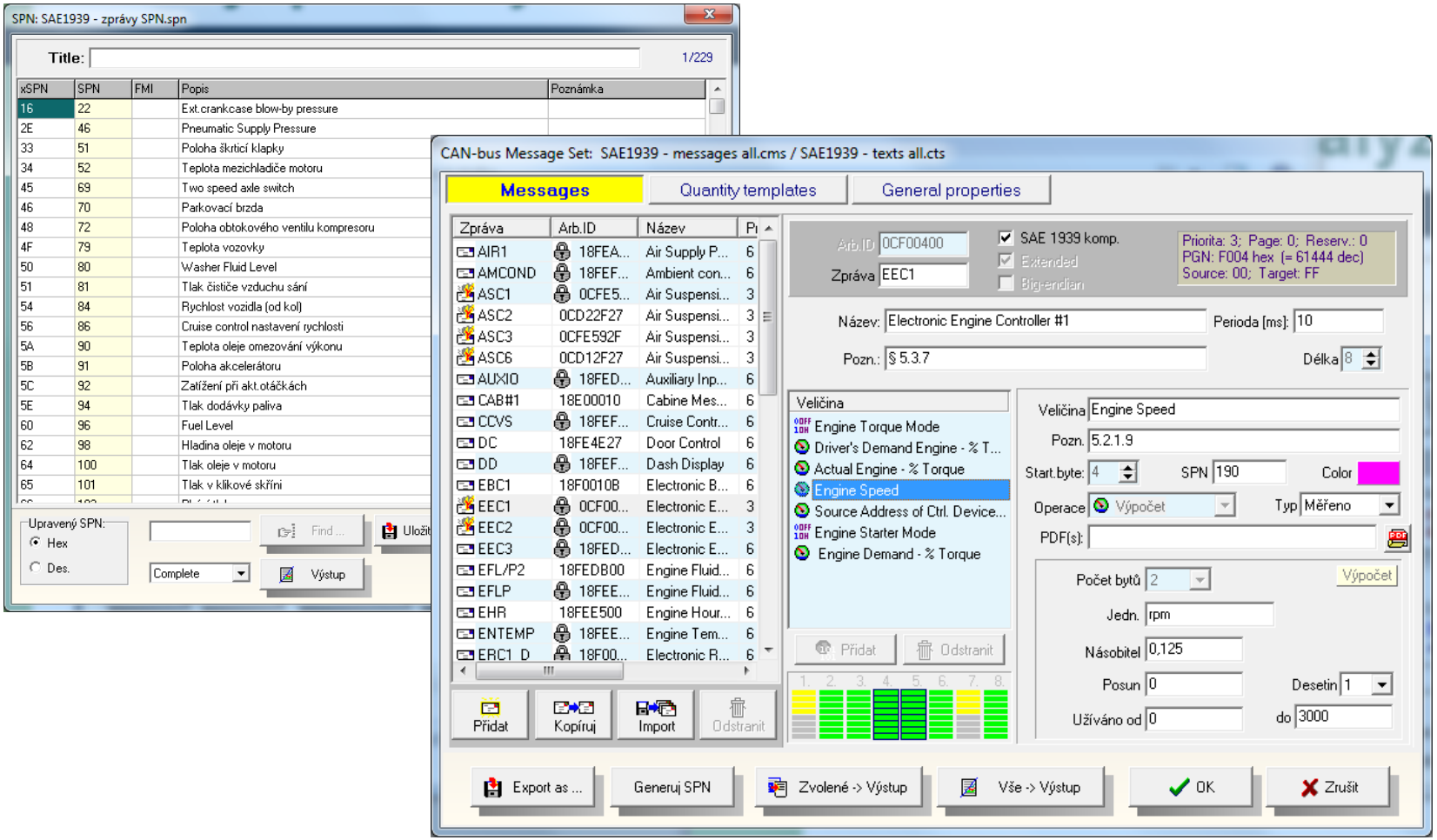
Nástroje pro analýzu komunikace

- **Diagnostika (starší UCINT Suite 5)**



Nástroje pro analýzu komunikace

- **Databanka SPN a PGN (starší UCINT Suite 5)**



KONEC

Děkuji za pozornost

Literatura

[1] Voss, W. *A Comprehensible Guide to J1939*. Copperhill Technologies Corp. 2008. ISBN 978-0-9765116-3-2.

[2] Voss, W. *SAE J1939: Serial Control and Communications Vehicle Network*. ESD electronics, Inc. Power Point prezentace.

[3] Vector. *Introduction to J1939*. Application Note AN-ION-1-3100. 2010.

[4] *ISO 9141-2 and J1939 Protocols on OBDII.* Power Point prezentace

[5] Hajný, A. *Aplikační protokoly CAN pro dieselelektrické lokomotivy*. Unicontrols. PDF prezentace.

[6] Caterpillar. *C4.4 ACERT – C.18 ACERT Electronic Applications & Installation Guide*. 2012.

[7] Dokumentace SAE J1939-71, rev. Januar-2008.

[8] Dokumentace SAE J1939-21, rev. April-2001.

[9] Dokumentace SAE J1939-81, rev. May-2003.

[10] Dokumentace SAE J1939-73, rev. September-2006.

[11] www.google.com

[12] Video seminář Fundamentals of the J1939 Protocol

[13] Video seminář A Brief Introduction to SAE J1939

[14] https://www.csselectronics.com/pages/j1939-explained-simple-intro-tutorial#j1939-connector

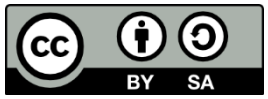
[15] https://www.vector.com/at/en/know-how/protocols/sae-j1939/#c150199

[16] SOR Libchavy spol. s r.o. *SOR NB18 Euro VI: Návod k obsluze*. 2019.

UNIVERZITA PARDUBICE
DOPRAVNÍ FAKULTA JANA PERNERA

Vytvořeno v rámci projektu **Green Deal UPCE**, reg. č. NPO_UPCE_MSMT-2142/2024-4.

Toto dílo podléhá licenci Creative Commons BY-SA 4.0. Pro zobrazení licenčních podmínek navštivte https://creativecommons.org/licenses/by-sa/4.0/.



NÁRODNÍ PLÁN OBNOVY

Financováno Evropskou unií
NextGenerationEU