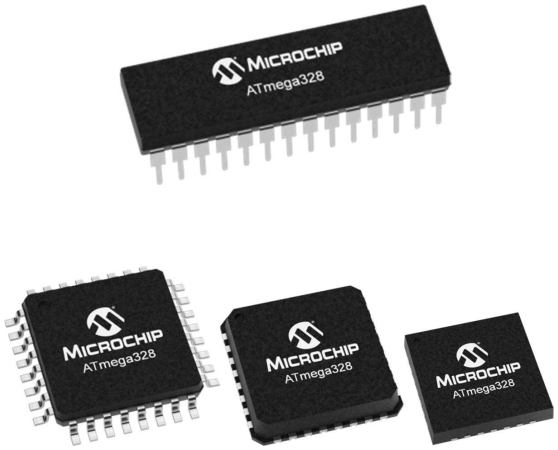
**TECHNICKÁ UNIVERZITA V KOŠICIACH**





| | |
|---|---|
| Názov | Mikroprocesorová technika ATmega328P |
| Autor | Ing. Livovský Ľubomír, PhD. |
| Vydavateľ | Technická univerzita v Košiciach |
| Rok | 2023 |
| Vydanie | prvé |
| Náklad | 50 ks |
| Rozsah | 139 strán |
| ISBN: | 978-80-553-4403-4 |
| Recenzenti: | prof. Ing. Pavol Galajda, PhD. |
| | doc. Ing. Ján Molnár, PhD. |


# MIKROPROCESOROVÁ TECHNIKA ATMEGA328P

ĽUBOMÍR LIVOVSKÝ

KOŠICE 2023

# Obsah

# Zoznam obrázkov

## Zoznam tabuliek







# Úvod

Táto učebnica pre vysoké školy je rozšírením učebnice „Základy mikroprocesorovej techniky“ a je určená ako doplnková literatúra pre štúdium predmetu „Základy mikroprocesorovej techniky“ a predmetu „Mikroprocesorová technika“ študentom Fakulty elektrotechniky a informatiky Technickej univerzity v Košiciach. Učebnica ja zameraná na hardvérový popis mikrokontroléra ATmega328P, ktorý je používaný pri praktických seminárach. Hlavnou náplňou učebnice je popis a vysvetlenie hardvéru mikrokontroléra a praktické ukážky používania jednotlivých hardvérových zariadení. Príklady softvérových funkcií sú riešené vo vývojovom prostredím *Microchip Studio*, ktoré obsahuje všetky potrebné súčasti na písanie zdrojového kódu, preklad, simuláciu kódu, nahrávanie kódu do mikrokontroléra. Pre potreby praktických seminárov sa odporúča vývojová doska Arduino UNO, alebo iná vývojová doska s jadrom AVR.

Táto učebnica vyšla s podporou projektu KEGA 011TUKE-4/2023 "**Synergia edukačného procesu, výskumu, inovácií a priemyselného sveta v kontexte prestavby študijného programu automobilová elektronika".**

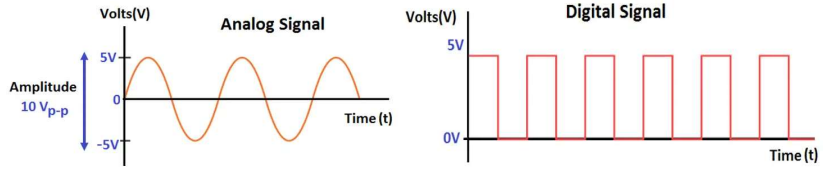
# 1. Úvod do problematiky digitálnej elektroniky

Elektronika je odvetvie fyziky, inžinierstva a technológie, ktoré sa zaoberá obvodmi pozostávajúcimi z komponentov, ktoré pracujú s elektrickým prúdom. Obvody a komponenty (súčiastky) možno rozdeliť do dvoch skupín: analógové a digitálne. Konkrétne zariadenie môže pozostávať z analógových obvodov, digitálnych obvodov alebo analógových a digitálnych obvodov. Digitálna elektronika alebo digitálne elektronické obvody pracujú s digitálnymi (číslicovými) signálmi. Na začiatku éry digitálnych obvodov bolo ich použitie zamerané na počítačové systémy. Teraz sa digitálna elektronika používa v širokej škále systémov, ako sú telekomunikačné systémy, vojenské systémy, lekárske systémy, riadiace systémy a spotrebná elektronika.

Elektronické systémy možno rozdeliť do dvoch širokých kategórií: digitálne a analógové. Digitálne obvody sú elektrické obvody, ktoré pracujú s digitálnymi signálmi, ktoré majú množstvo diskrétnych úrovní napätia. Pre väčšinu inžinierov sú pojmy „digitálny obvod“, „digitálny systém“ a „logika“ zameniteľné v kontexte digitálnych obvodov, zatiaľ čo analógové obvody zahŕňajú veličiny so spojitými hodnotami.

## 1.1. Analógový a digitálny signál

Analógová veličina je veličina, ktorá má spojité hodnoty v čase. Digitálna veličina je veličina, ktorá má diskrétny súbor hodnôt. V prirodzenom svete sú väčšiny fyzikálnych veličín, ako je teplota, tlak a napätie, analógové veličiny. Tieto fyzikálne veličiny môžu byť konvertované na spojité elektronické signály (napätie alebo prúd) pomocou senzora, takže môžu byť spracované pomocou elektrického obvodu. Analógový signál označuje signál, ktorý v priebehu času neustále mení svoju hodnotu. Typický analógový signál je sínusová vlna alebo zvuková vlna, ako je znázornené na obrázku Obr. 1. Termín analógový signál sa zvyčajne vzťahuje na elektronické signály, pričom mechanické, pneumatické, hydraulické, ľudská reč a iné systémy môžu tiež prenášať alebo byť považované za analógové signály.




Obr. 1 Porovnanie analógového a digitálneho signálu

Digitálny signál sa týka elektrického signálu, ktorý má sekvenciu diskrétnych hodnôt, teda v ľubovoľnom čase môže nadobudnúť iba jednu z konečného počtu hodnôt. To je v kontraste s





analógovým signálom, ktorý predstavuje spojité hodnoty, teda v akomkoľvek danom čase predstavuje reálne číslo v rámci súvislého rozsahu hodnôt. V digitálnom obvode je digitálny signál sled impulzov, to znamená sled elektrických impulzov s pevnou šírkou. Obr. 1 zobrazuje typický digitálny signál, ktorý sa mení medzi úrovňami nízkeho a vysokého napätia. Úroveň vysokého napätie predstavuje binárnu 1 a úroveň nízkeho napätia predstavuje binárnu 0. Tento druh digitálneho signálu sa tiež nazýva logický signál alebo binárny signál.

## 1.2. Binárne číslice

Väčšina digitálnych obvodov používa binárny systém, ktorý pozná iba dve číslice (1 a 0), ktoré môžu predstavovať dve úrovne napätia. Binárna číslica sa nazýva bit. Logická 0 bude často nižšie napätie a označuje sa ako úroveň LOW, zatiaľ čo logická 1 sa označuje ako úroveň HIGH. Toto priradenie napäťových úrovní sa nazýva pozitívna logika a bude používané v celej učebnici. Samozrejme, je možné priradiť logickú 0 úrovni HIGH a logickú 1 úrovni LOW a vtedy budeme hovoriť o negatívnej logike. V digitálnych systémoch sa kombinácia určitého počtu bitov 1 a 0 nazýva binárny kód, ktorý sa používa na reprezentáciu čísel, symbolov, abecedných znakov a iných typov informácií.

## 1.3. Logické úrovne

Napätie používané na vyjadrenie 1 alebo 0 sa nazýva logická úroveň. V praktickom digitálnom obvode môže byť HIGH úroveň akákoľvek úroveň napätia medzi špecifikovanou minimálnou hodnotou a špecifikovanou maximálnou hodnotou. Podobne LOW môže byť akákoľvek úroveň napätia medzi špecifikovaným minimom a špecifikovaným maximom. Rozsahy napäťových úrovní HIGH a LOW sa neprekrývajú.

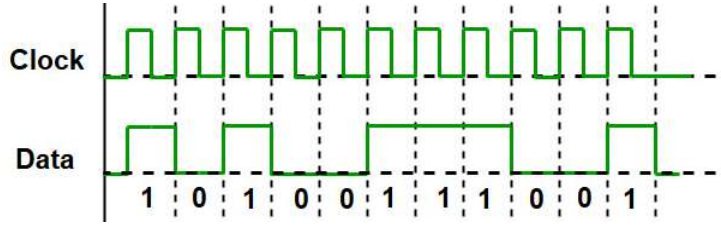
## 1.4. Tvar digitálneho signálu

Tvar digitálneho signálu pozostáva z úrovní napätia, ktoré sa medzi úrovňou HIGH a LOW. Ako je znázornené na obrázku (Obr. 2), kladný impulz sa generuje, keď napätie prejde z LOW úrovne na HIGH úroveň. Záporný impulz sa vytvorí, keď napätie prejde z HIGH úrovne na LOW úroveň.


Obr. 2 Tvar impulzu digitálneho signálu

Binárne informácie, s ktorými pracujú digitálne systémy, sa prejavujú ako digitálne priebehy. Digitálny priebeh sa skladá zo série impulzov nazývaný ako postupnosť impulzov. Keď je tvar vlny

HIGH, je prítomná binárna 1, keď je LOW, je prítomná binárna 0. Každý bit v sekvencii trvá definovaný časový interval. V digitálnych systémoch sú priebehy signálu synchronizované so základným časovým priebehom nazývaným taktovacie hodiny (Obr. 3). Hodinový signál je periodická postupnosť impulzov, ktorých perióda je rovná trvaniu jedného bitu. Binárne dáta sú indikované svojou úrovňou v postupnosti priebehu. Počas každej periódy hodinového signálu je priebeh binárnych dát buď HIGH alebo LOW, pričom úrovne HIGH a LOW predstavujú sekvenciu binárnych číslic (bitov). Binárne dáta sú reprezentované skupinou niekoľkých bitov.




Obr. 3 Synchronizovanie binárnych dát s hodinovým impulzom

## 1.5. Charakteristika digitálneho impulzu

Digitálny impulz má nasledujúcich päť dôležitých charakteristík (Obr. 4).

*Rise time* - čas vzostupu (nábehu) je čas potrebný na to, aby impulz prešiel z LOW úrovne na HIGH úroveň. V praxi je bežné merať čas nábehu od 10 % amplitúdy impulzu do 90 % amplitúdy impulzu.

*Fall time* - čas zostupu (pokles) je čas potrebný na to, aby impulz prešiel z HIGH úrovne na LOW úroveň. V praxi je bežné merať čas poklesu od 90 % amplitúdy impulzu do 10 % amplitúdy impulzu.

Možno poznamenať, že spodných 10 % a horných 10 % z amplitúdy impulzu nie sú zahrnuté v časoch nábehu a poklesu z dôvodu nelinearít v tvare vlny signálu v týchto oblastiach.

*Pulse width* – šírka impulzu je mierou trvania impulzu a často sa definuje ako časový interval medzi 50 % bodmi na vzostupnej a zostupnej hrane.

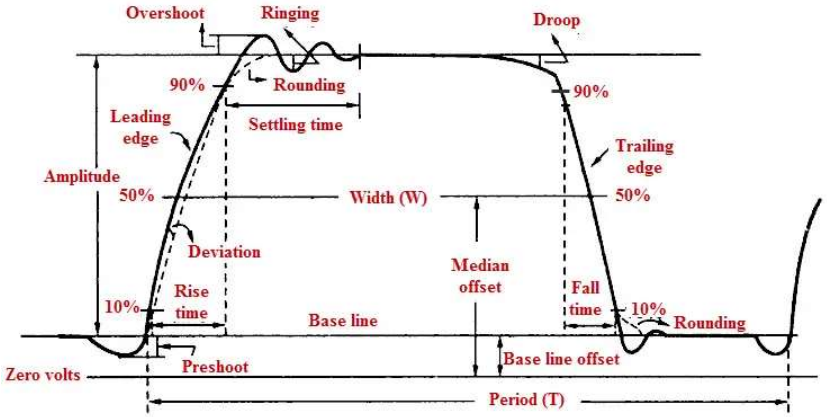
*Overshoot* – prekmit je nežiadúca vlastnosť číslicového impulzu. Prekročenie napäťovej úrovne môže byť pozitívne aj negatívne. Prekmit je spôsobený kapacitným efektom v obvode alebo meracom prístroji, ktorý vedie k tomu, že napätie na nábežnej a zostupnej hrane krátkodobo prekročí normálnu úroveň HIGH alebo LOW.

*Ringing* – zvlnenie je nežiadúca vlastnosť číslicového impulzu. Prekročenie napäťovej úrovne v oboch prípadoch môže byť pozitívne aj negatívne. Zvlnenie na vzostupnej a zostupnej hrane





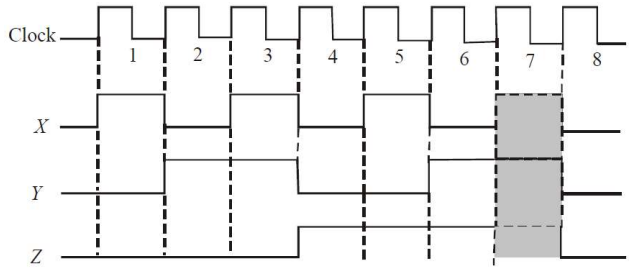
impulzu je vlastne oscilácia spôsobená kapacitou a indukčnosťou v obvode a po krátkom čase odznie.




Obr. 4 Parametre digitálneho impulzu

## 1.6. Časový diagram

Časový diagram je graf digitálnych priebehov, ktorý ukazuje časový vzťah všetkých priebehov a ako sa každý priebeh mení vo vzťahu k ostatným. Na Obr. 5 je príklad jednoduchého časového diagramu, ktorý ukazuje, ako súvisia hodinový priebeh a tvar signálu X, Y, Z. Časový diagram môže pozostávať z ľubovoľného počtu súvisiacich priebehov. Časový diagram zobrazuje stavy (HIGH a LOW) všetkých priebehov v akomkoľvek časovom bode a presný čas, kedy sa zmenil stav vzhľadom na ostatné priebehy.




Obr. 5 Príklad časového diagramu

Na časovom diagrame môžeme vidieť, že všetky tri priebehy *X, Y, Z* sú na úrovni HIGH iba počas 7. hodinového taktu a všetky sa zmenia späť na LOW na konci 7. hodinového taktu.

## 1.7. Charakteristika digitálnych obvodov

Digitálny obvod je obvod, ktorý pracuje s digitálnymi signálmi na vstupe, spracováva ich a nastavuje výstupné digitálne signály. Keďže digitálny signál je signál, v ktorom sa na reprezentáciu informácií používajú diskrétne úrovne (stavy), digitálne obvody majú jednu úroveň signálu, keď sú zapnuté, a inú úroveň signálu, keď sú vypnuté. Vo všeobecnosti je komponent v digitálnych obvodoch iba zapnutý alebo vypnutý. Napríklad tranzistory v digitálnych obvodoch pracujú buď v saturačnej oblasti alebo v medznej oblasti. Zatiaľ čo tranzistory v analógových obvodoch pracujú v aktívnej oblasti, ich výstupy sú citlivé na niekoľko faktorov, ako je teplota, napájacie napätie a starnutie komponentov. Preto výhodou digitálnych obvodov v porovnaní s analógovými obvodmi je, že signály reprezentované digitálne môžu byť prenášané bez degradácie v dôsledku šumu.

Digitálne obvody sú najbežnejšou fyzickou reprezentáciou Boolovej algebry. Návrh digitálnych obvodov je logický návrh, ktorý nevyžaduje od dizajnérov veľmi silné matematické zázemie, zatiaľ čo návrh analógového obvodu vyžaduje výpočet modelu, aby bolo možné pochopiť a preštudovať vnútorné charakteristiky a princíp činnosti obvodu. V digitálnom systéme je možné získať presnejšiu reprezentáciu signálu použitím viacerých binárnych číslic na jeho reprezentáciu. V analógovom systéme si dodatočné rozlíšenie vyžaduje zásadné vylepšenia linearity a šumových charakteristík každého kroku signálového reťazca.

Digitálne obvody sa ľahko integrujú, sú lacné a majú malé rozmery. Úroveň integrácie digitálnych obvodov je vo všeobecnosti vyššia ako úroveň analógových obvodov. Okrem toho sú digitálne obvody programovateľné. Počítačový jazyk možno použiť na navrhovanie niektorých digitálnych obvodov na dosiahnutie zodpovedajúcich logických funkcií. Počítačom riadené digitálne systémy je možné ovládať softvérom, čo umožňuje pridávanie nových funkcií bez zmeny hardvéru.

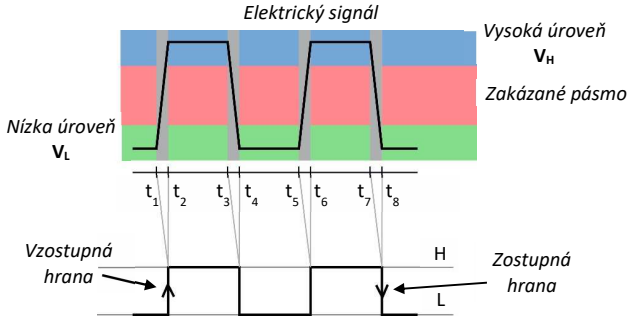
Digitálne elektronické obvody sa zvyčajne skladajú z logických brán, pomocou ktorých sa realizujú rôzne logické funkcie, ktoré v sebe zahŕňajú najmä kombinačné logické funkcie a sekvenčné logické funkcie. Všetky logické prvky a funkcie v digitálnom obvode sú dostupné vo forme integrovaného obvodu (IC). Monolitický IC je elektronický obvod, ktorý je celý skonštruovaný na jedinom malom čipe kremíka. Všetky komponenty, ktoré tvoria obvod – tranzistory, diódy, odpory a kondenzátory – sú neoddeliteľnou súčasťou tohto jediného čipu. Digitálne integrované obvody sú rozdelené do dvoch širokých kategórií: logika s pevnou funkciou a programovateľná logika.





## 1.8. Základné parametre digitálnych obvodov

Logická úroveň v číslicovej elektronike je reprezentácia logickej hodnoty určitým stavom elektrického obvodu, napr. hodnotou elektrického napätia. Rozlišujeme dve úrovne, vysokú úroveň HIGH a nízku úroveň LOW. Presné hodnoty napätí jednotlivých úrovní sa líšia podľa typu použitých logických obvodov (Tab. 1). LOW úroveň predstavuje logickú 0 a HIGH predstavuje logickú 1.




Obr. 6 Časový priebeh reálneho elektrického signálu (hore) a ideálneho signálu (dole)

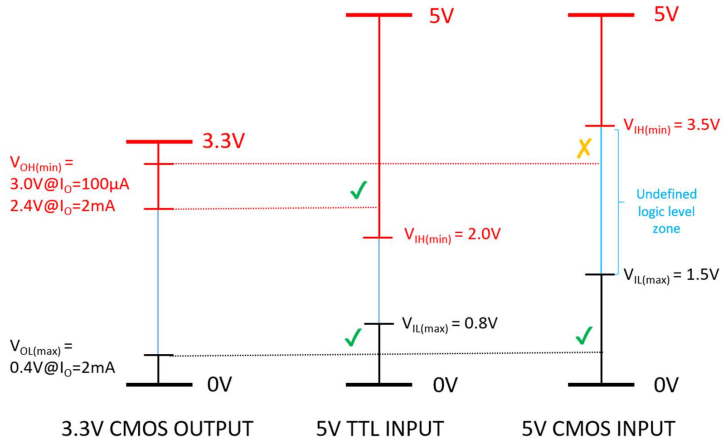
Na vstupoch logického obvodu hodnota $V_{IL}$ definuje maximálne napätie reprezentujúce LOW úroveň, hodnota $V_{IH}$ definuje minimálne napätie reprezentujúce HIGH úroveň. Medzi hodnotami $V_{IL}$ a $V_{IH}$ leží zakázané pásmo, v ktorom logická hodnota nie je jednoznačne určená (Obr. 7). Prechod medzi dvoma logickými úrovňami musí prebehnúť rýchlo, pretože prechádza zakázaným pásmom. Strmosť nárastu tohto prechodu sa označuje ako hrana signálu.

Tab. 1 Hodnoty napäťových úrovní niektorých technológií číslicových obvodov

| | **Vstupná úroveň $V_I$** | | **Výstupná úroveň $V_O$** | |
|---|---|---|---|---|
| Technológia | nízka ($V_{IL}$) | vysoká ($V_{IH}$) | nízka ($V_{OL}$) | vysoká ($V_{OH}$) |
| TTL 5V | ≤ 0,8 | ≥ 2,0 | ≤ 0,4 | ≥ 2,4 |
| CMOS 5V | ≤ 1,5 | ≥ 3,5 | ≤ 0,5 | ≥ 4,44 |
| LVTTL 3,3V | ≤ 0,8 | ≥ 2,0 | ≤ 0,4 | ≥ 2,4 |
| CMOS 2,5 V | ≤ 0,7 | ≥ 1,7 | ≤ 0,2 | ≥ 2,3 |
| CMOS 1,8 V | ≤ 0,7 | ≥ 1,17 | ≤ 0,45 | ≥ 1,2 |

Na výstupoch logického obvodu je zaručená vysoká úroveň HIGH napätím minimálne $V_{OH}$ a na vstupoch je to minimálne napätie $V_{IH}$. Výstupné napätie $V_{OH}$ je vždy väčšie ako vstupné napätie $V_{IH}$, rozdiel $V_{OH} - V_{IH}$ zabezpečuje prevádzkovú odolnosť obvodov voči rušeniu. Podobne pre nízku úroveň **L** rozdiel napätí $V_{IL} - V_{OL}$ zabezpečuje odolnosť voči rušeniu. Na Obr. 7 je zobrazený stav, ak sa ku 3,3V CMOC výstupu pripojí 5V TTL vstup a 5V CMOS vstup. Z hodnôt úrovní vstupných

a výstupných napätí je zrejmé, že kombinácia 3,3V CMOS výstup a 5V CMOS vstup nebude fungovať, lebo hodnota napätia pre 5V CMOS vstup je v zakázanom pásme.




Obr. 7 Napäťové úrovne: 3,3V CMOS výstup, 5V TTL vstup a 5V CMOS vstup.

**Statické parametre číslicových obvodov** vyjadrujú jednosmerné podmienky funkcie logických obvodov. Môžeme ich označiť aj ako jednosmerné parametre:

- **vstupné napätie** - $\mathbf{V_{IH}}$, je napätie na vstupe, pri ktorom je ešte zaručené HIGH,
- **vstupné napätie** - $\mathbf{V_{IL}}$, je napätie na vstupe, pri ktorom je ešte zaručené LOW,
- **vstupný prúd** - $\mathbf{I_{IH}}$, je prúd, ktorý môže prejsť vstupom, ak je vstup na úrovni HIGH,
- **vstupný prúd** - $\mathbf{I_{IL}}$, je prúd, ktorý môže prejsť vstupom, ak je vstup na úrovni LOW.

**Rozhodovacia úroveň** je napätie na vstupe logického obvodu, pri ktorom obvod prechádza z jedného stavu do druhého.

**Logický zisk na výstupe** vyjadruje maximálny počet vstupov nasledujúcich logických obvodov, ktoré je možné pripojiť na jeden výstup, aby boli zachované podmienky správanej činnosti logických obvodov. Ak logický zisku N = 10 znamená, že logický člen môže zo svojho výstupu napájať 10 logických vstupov.

**Odolnosť voči rušeniu** nazývame maximálne poruchové napätie, ktoré neovplyvní stav logického obvodu. Poruchové napätie môže vzniknúť vplyvom náhodných zmien napájacieho napätia.





**Stratový výkon** je výkon spotrebovaný jedným hradlom. Definuje sa pre určité podmienky, najčastejšie pri dynamickej činnosti hradla striedaním LOW a HIGH na vstupe logického obvodu pri určitej frekvencii.

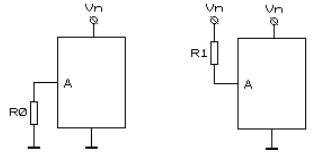
**Tolerancia napájacieho napätia** je rozptyl napájacieho napätia, pri ktorom nie je porušená správna činnosť logického obvodu.

**Rozsah pracovných teplôt** definuje rozsah pracovnej teploty, ktorom sú zaručené charakteristické parametre logického obvodu. Prekročenie tohto rozsahu môže ovplyvniť činnosť logických obvodov.

## 1.9. Zapojenia vstupov číslicových obvodov

Logický vstupný signál musí rešpektovať vstupné parametre typu logického obvodu a ďalej to, že nevyužitý vstup logického obvodu nesmie ostať nezapojený, pretože nie je presne definovaná jeho vstupná logická úroveň. Pri štandardných TTL obvodov sa síce nepripojený vstup najčastejšie správa ako by bol nastavený na úroveň H, ale má v tomto prípade veľmi nízku odolnosť voči rušeniu. V podstate je potrebné pripojiť nevyužitý vstup na zdroj napätia definovanej úrovne LOW alebo HIGH, tak aby nebola narušená logická funkcia obvodu. K dispozícii je niekoľko ošetrení nezapojených vstupov:

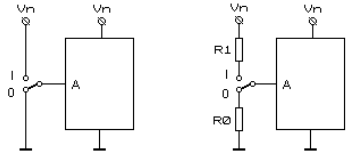
- Priame zapojenie nepoužitého vstupu na LOW (GND) alebo HIGH (VCC) – používa sa pre nevyužité vstupy dátových signálov, riadiacich signálov.
- Pripojenie nepoužitého vstupu cez rezistor na HIGH (*Pull-up*) alebo na LOW (*Pull-down*) – používa sa pre nevyužité vstupy dátových signálov a inicializačných vstupov. Hodnoty rezistorov je možné vypočítať:



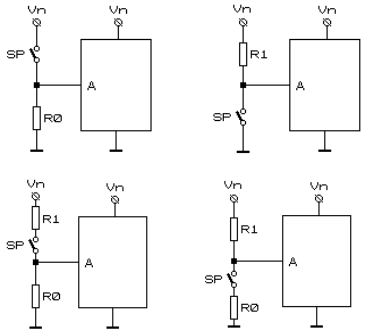

$$R_{0max} = \frac{V_{IL}}{I_{IL}} \qquad (pre\ TTL \approx 500\Omega)$$

$$R_{1max} = \frac{V_n - V_{IH}}{I_{IH}} \qquad (pre\ TTL \approx 75k\Omega)$$

- Prepínanie logických úrovní na vstupe




Nesprávne riešenie pomocou prepínača pretože vstup ostáva určitý čas nepripojený na definovanej napäťovej úrovni.

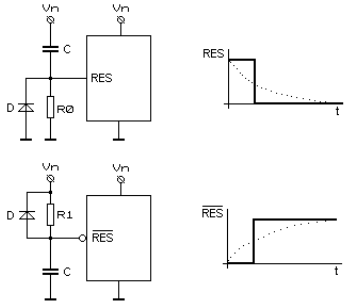



Správne riešenie pomocou spínača a jedného rezistora pretože vstup je stále pripojený na definovanú úroveň napätia.

Riešenie pomocou dvoch rezistorov. Ak $R_0 = 500\ \Omega$, potom $R_1 = 750\ \Omega$ podľa vzťahu:

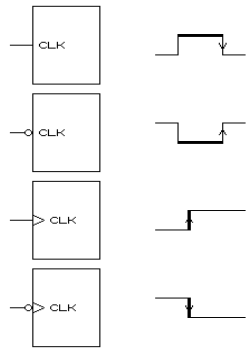
$$R_{1max} = R_0 \frac{V_n - V_{IH}}{V_{IH}}$$

- „Ošetrenie“ inicializačných vstupov – napr. „reset“ signálu




Nepoužíva sa spínač ani prepínač, ale vyžaduje sa automatická funkcia pomocného obvodu. Napr. pri zapnutí napájacieho napätia sa vygeneruje reset impulz na definovanie počiatočného stavu hlavne sekvenčných obvodov alebo mikroprocesorov.

Charakteristika a označovanie vstupov logických obvodov:




Hodinový vstup aktívny na úroveň HIGH, aktívna je hrana HIGH-LOW.

Hodinový vstup aktívny na úroveň LOW, aktívna je hrana LOW-HIGH.

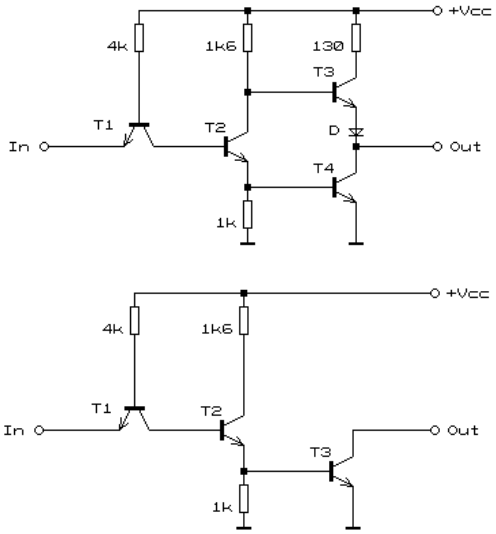
Hodinový vstup aktívny na vzostupnú hranu LOW-HIGH (čelo impulzu).

Hodinový vstup aktívny na zostupnú hranu HIGH-LOW (tylo impulzu).





## 1.10. Zapojenie výstupov číslicových obvodov



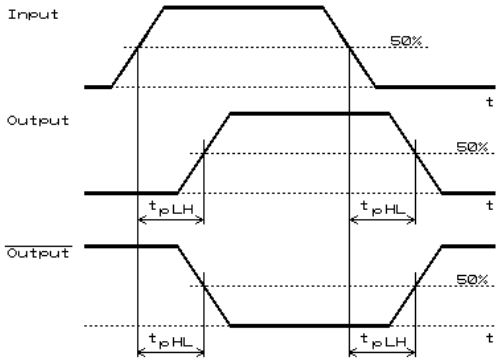

Výstupy logických obvodov môžu byť štandardné alebo výstup s otvoreným kolektorom.




Zapojenie s otvoreným kolektorom umožňuje pripojenie záťaže s rôznym napätím a výstupným prúdom.

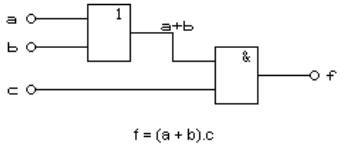
## 1.11. Rušenie číslicových obvodov

Pri prechode signálu každým elektronickým obvodom dochádza k oneskoreniu výstupného signálu voči vstupnému - **prechodové oneskorenie $t_p$** (*Propagation time*) (Obr. 8). Prechodové oneskorenie je definované pre 50 % úrovne vstupných a výstupných signálov logického obvodu a môže byť špecifikované aj s vyznačením zmyslu prechodu medzi logickými úrovňami ($t_{pLH}$, $t_{pHL}$).

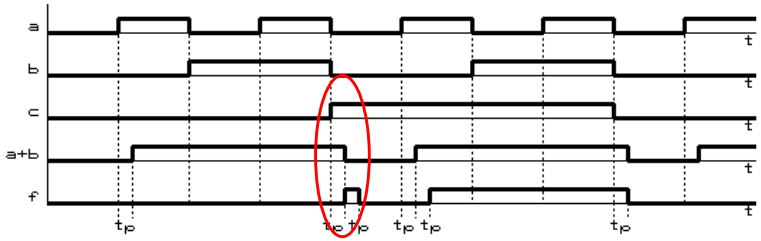



Obr. 8 Prechodové oneskorenie

Príčina vzniku hazardov je v oneskorení priechodu signálu medzi vstupom a výstupom logických obvodov. Z toho vyplýva, že najväčší predpoklad vzniku hazardov bude v systémoch, kde jeden vstupný signál logického systému prechádza veľkým počtom logických obvodov na rozdiel od iných signálov. Na (Obr. 9) je signál *a+b* oneskorený prechodom cez dvojvstupový OR na rozdiel od signál *c*. Toto oneskorenie sa prejaví, ako hazard prechodom cez dvojvstupový AND na signály *f* .




| c | b | a | f |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |



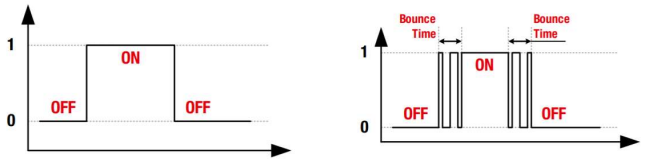

Obr. 9 Príklad vzniku hazardu prechodom signálov logickými obvodmi

## 1.12. Rušenie spínaním mechanických kontaktov na vstupe

Jedným z možných ovládaní vstupov digitálnych obvodov je použitie mechanického kontaktu ako sú tlačidla (*Buttons*), relé kontakty a iné. Mohli by sme si myslieť, že spojenie vytvárajúce prepínač je rýchle, priame a pevné. V skutočnosti typický mechanický spínač urobí niekoľko prechodov z vodivého do nevodivého stavu počas desiatok milisekúnd potrebných na rozpojenie alebo spojenie kontaktov v dôsledku vplyvov, ktoré zahŕňajú vek, prevádzkovú zotrvačnosť, mechanickú konštrukciu a mikroskopický stav kontaktných povrchov spínača (Obr. 10). Toto správanie, ktoré sa bežne nazýva zakmitanie kontaktov (*switch bounce)* a je neodmysliteľnou vlastnosťou všetkých mechanických spínačov a prepínačov.





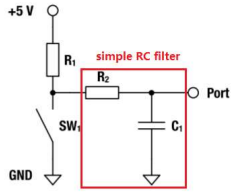



Obr. 10 Signál mechanického kontaktu: a) ideálny, b) reálny

Problém zakmitania kontaktov je možné riešiť hardvérovo alebo softvérovo.

Hardvérové riešenie ponúka tieto možnosti:

- Jednoduchý RC filter je jedným z najlacnejších a najjednoduchších spôsobov výroby dolnopriepustného filtra. Keď je spínač otvorený, kondenzátor sa nabíja cez R1+R2, čo spôsobuje pomalší nárast napätia. Keď je spínač zatvorený, kondenzátor sa vybíja cez R2 riadenou rýchlosťou.



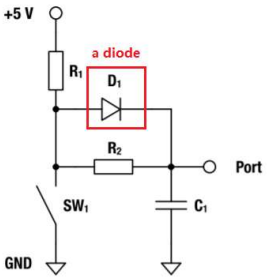

Obr. 11 Použitie RC filtra na eliminovanie zákmitov kontaktu

Ak súčiastky filtra vypočítané správne, zakmitanie kontaktu spínača je absorbované počas nabíjania a vybíjania počas plynulého prechodu. Na výpočet hodnoty kondenzátora a rezistorov je možné použiť nasledujúci vzorec:

$$t = (R_1 + R_2) * C_1 \quad [s, \Omega, \Omega, F] \tag{1.1}$$

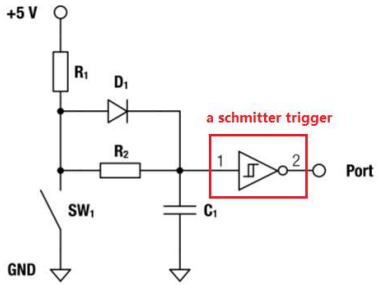
Napríklad, doba zákmitov kontaktu je t = 10ms, zvolíme $R_1$ = 1kΩ, $R_2$ = 10kΩ a $C1 = 1\mu F$ sa vypočíta zo vzťahu 1.1. Iné riešenie: $R_1$ = 1kΩ, $R_2$ = 47kΩ a $C1 = 220nF$.

- Pridanie diódy D1 paralelne k odporu R2 umožňuje samostatne ovládať čas nabíjania a vybíjania kondenzátora C1. Dióda D1 skratuje R2 a zrýchli sa nabíjanie kondenzátora, zatiaľ čo vybíjanie kondenzátore sa vykonáva cez odpor R2.



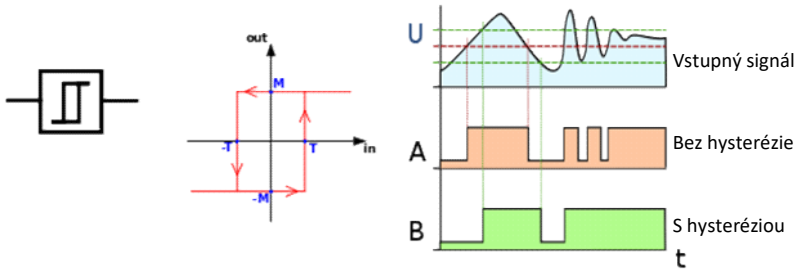

Obr. 12 Použitie RC filtra a diódy na eliminovanie zákmitov kontaktu

- V digitálnych elektronických obvodoch sú určité úrovne napätí nežiadúce (napríklad od 0,8 V do 2,4 V – zakázané pásmo TTL logiky), musí sa použiť elektronický obvod s hysteréziou napr. Schmittov obvod.




Obr. 13 Použitie obvodu s hysteréziou na eliminovanie zákmitov kontaktu

Schmittov obvod mení skokovo výstupné napätie so zmenou napätia na vstupe, pričom zmena výstupu pri náraste napätia na vstupe nastane pri vyššom napätí ako zmena výstupu pri poklese napätia na vstupe obvodu. Vďaka tejto hysterézii sú tieto obvody imúnne voči šumu na vstupe.

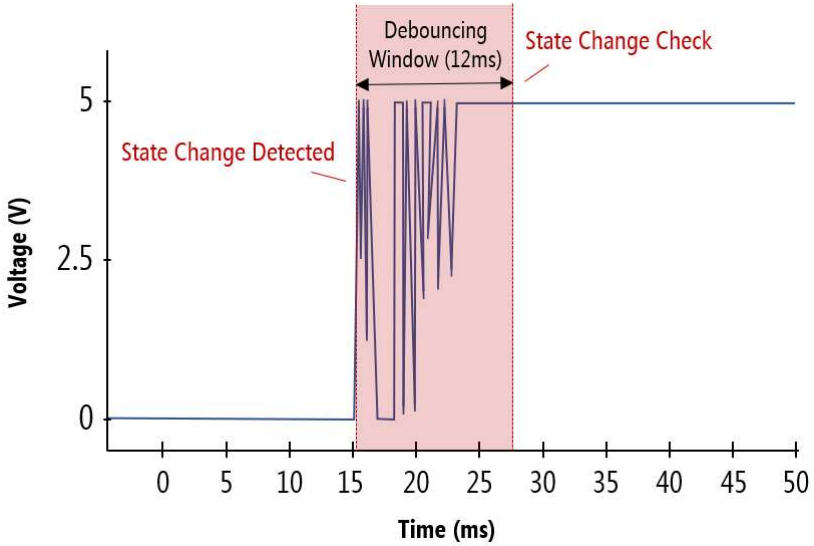



Obr. 14 Schmittov preklápací obvod





Softvérové riešenie je založené na sledovaní stavu mechanického kontaktu v závislosti na čase. Ak sa zistí zmenu stavu spínača (stav1), následne sa čaká určitý čas (časové okno) a potom sa opätovne skontrolovať skontroluje stav spínača (stav2). Ak sa počiatočný stav a stav po uplynutí času zhodujú (stav1 = stav2), potom prepínač prešiel z jedného ustáleného stavu do druhého (**Chyba! Nenašiel sa žiaden zdroj odkazov.**). Tento postup sa musí vyhodnocovať pri zopnutí a takisto aj pri rozopnutí kontaktu.




Obr. 15 Príklad softvérové riešenia zákmitov kontaktu pomocu časového okna

# 2. Číselné sústavy a kódovanie

Digitálne signály sú vo všeobecnosti reprezentované sériou binárnych číslic. Binárny číselný systém a digitálne kódy sú základom digitálnych obvodov a digitálnych systémov. Číselná sústava je spôsob, akým sú zapisované čísla pomocou znakov. Podľa spôsobu určenia hodnoty čísla z dané zápisu rozlišujeme dve hlavné číselne sústavy: pozičné číselné sústavy a nepozičné číselné sústavy.

Pozičné sústavy sú charakterizované tzv. základom alebo bázou, čo je obvykle kladné celé číslo definujúce maximálny počet číslic, ktoré sú v danej sústave k dispozícií. Číslo zapísané v pozičnej číselnej sústave je možné vyjadriť súčtom mocnín základu danej sústavy vynásobených príslušnými platnými číslicami. Medzi najčastejšie používané pozičné číselne sústavy patria: dvojková, osmičková, desiatková, šestnástková (hexadecimálna).

Tab. 2 Nepozičná číselná sústava

| Rímske číslo | Arabské číslo |
|---|---|
| I | 1 |
| V | 5 |
| X | 10 |
| L | 50 |
| C | 100 |
| D | 500 |
| M | 1000 |

Nepozičné číselné sústavy (Tab. 2) sú charakteristické tým, že význam číslice (symbolu) nie je určený pozíciou, ale usporiadaním týchto číslic (symbolov). Typickým príkladom sú rímske číslice (MCMXLVI = 1946 = 1000 + 1000-100 + 50-10 + 5 + 1).

## 2.1. Dvojková číselná sústava

Dvojková číselná sústava používa základ číslo 2. Binárne systém používa len dve číslice **0** a **1**. Dvojková sústava sa používa vo výpočtovej technike, pretože číslice 0 a 1 odpovedajú dvom jednoducho rozpoznateľným stavom elektrického obvodu (vypnutý a zapnutý), resp. nepravdivosti či pravdivosti výroku, t. j. tvrdenia, pri ktorom má zmysel sa opýtať, či je, alebo nie je pravdivý a môže nastať práve jedna z týchto možností.

Matematický výpočet hodnoty binárneho čísla, ktoré sa skladá z **k** číslic $x_0 x_1$ až $x_k$, každé s hodnotou 0 alebo 1 môžeme vypočítať pomocou:

$$x = \sum_{i=0}^{k} x_i * 2^{k-i} \qquad (1.1)$$





Postup prevodu dvojkového čísla $(11010110)_2$ do desiatkovej sústavy podľa predchádzajúceho vzťahu je nasledovný:

k=7; x <0, 1>; i - pozícia číslice

$$(11010110)_2 = \sum_{i=0}^{7} x_i * 2^{7-i} = 1*2^7 + 1*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 128 + 64 + 16 + 4 + 2 = (214)_{10} \qquad (1.2)$$

## 2.2. Šestnástková číselná sústava

Šestnástková (hexadecimálna) sústava používa ako základ číslo 16. Hexadecimálne čísla sa zapisujú pomocou číslic '0', '1', '2', '3', '4', '5', '6', '7', '8' a '9' a písmen 'A', 'B', 'C', 'D', 'E' a 'F', pričom písmena 'A'–'F' reprezentujú čísla s hodnotou 10 – 15. Čísla v tomto zápise sa obvykle označujú písmenom H pripojeným k číslu v dolnom indexe. Napr. $3F7_H$ reprezentuje hodnotu, ktorá v desiatkovej sústave odpovedá číslu $3 * 16^2 + 15 * 16^1 + 7 * 16^0 = 1015$. Vďaka jednoduchému vzájomnému prevodu medzi šestnástkovou a dvojkovou sústavou sa hexadecimálny zápis čísel často používa v oblasti informatiky, napríklad pre adresy v operačnej pamäti počítača (mikrokontroléra). Nakoľko základ hexadecimálnej sústavy je číslo 16, čo je $2^4$, jedna hexadecimálna číslica obsiahne 4 bity, čo sa označuje aj ako jeden N*ibble*. Napríklad všetky hodnoty obsiahnuté v jednom byte (8 bitov) môžeme vyjadriť dvoma šestnástkovými číslicami ($00_H$, $01_H$ , $02_H$ , .... , $FF_H$).

Matematický výpočet hodnoty šestnástkového čísla, ktoré sa skladá z **k** číslic $x_0 x_1$ až $x_k$, každé o hodnote 0 až F môžeme vypočítať pomocou:

$$x = \sum_{i=0}^{k} x_i * 16^{k-i} \qquad (1.3)$$

Postup prevodu šestnástkového čísla $(4B0E)_{16}$ do desiatkovej sústavy podľa predchádzajúceho vzťahu je nasledovný:

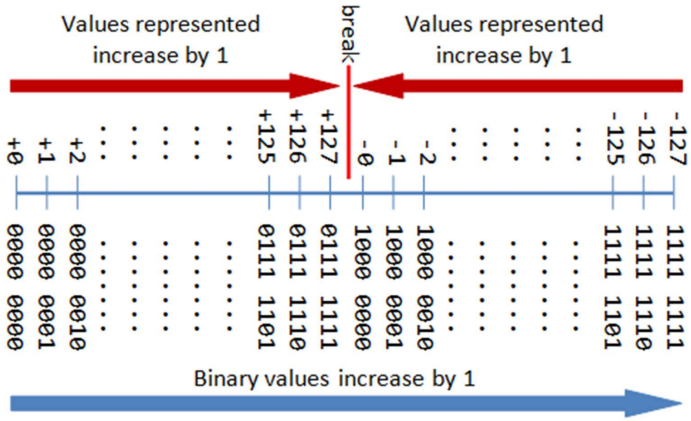
k=3; x <0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F>; i - pozícia číslice

$$(4B0E)_{16} = \sum_{i=0}^{3} x_i * 16^{3-i} = 4*16^3 + 11*16^2 + 0*16^1 + 14*16^0 = 16384 + 2816 + 0 + 14 = (19214)_{10} \qquad (1.4)$$

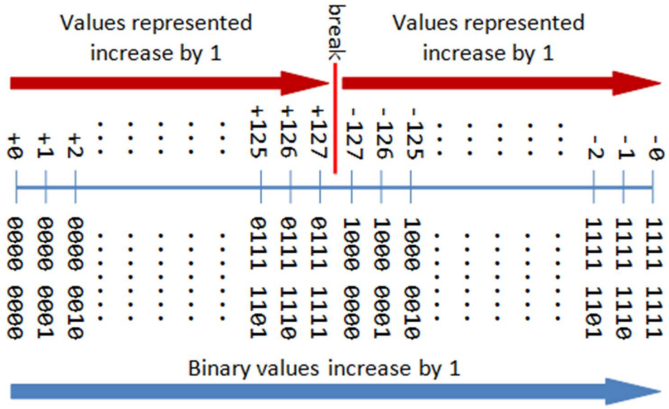
## 2.3. Reprezentácia záporných binárnych čísel

Binárne čísla používané v informatike sú kladné a záporné. Práca s kladnými binárnymi číslami zodpovedá jednoduchej interpretácií binárneho čísla. Práca so zápornými binárnymi číslami predstavuje komplikácie v počítačových systémoch s ich interpretáciou. Pre binárne čísla vo výpočtových systémoch sa používajú vo všeobecnosti tri formy reprezentácie: **znamienkovým bitom**, **jednotkovým doplnkom**, **dvojkovým doplnkom**.

Nevýhodou reprezentácie binárnych čísel znamienkovým bitom (Obr. 16) a jednotkovým doplnkom (Obr. 17) je, že číslo 0 je reprezentované ako kladná a súčasne záporná nula.




Obr. 16 Kódovanie binárnych čísel znamienkovým bitom




Obr. 17 Kódovanie binárnych čísel jednotkovým doplnkom





Najčastejšie používaný spôsob reprezentácie binárnych čísel je dvojkový doplnok (Obr. 18). **Dvojkový doplnok** je spôsob kódovania celých čísel kladných aj záporných v dvojkovej sústave, ktorý umožňuje zjednodušiť aritmeticko-logickú jednotku v procesore vďaka tomu, že sčítanie a odčítanie sa realizuje pre čísla so znamienkom rovnako ako pre čísla bez znamienka. Rozdiel je len v interpretácií pretečenia. Odčítanie je možné realizovať ako sčítanie prvého operandu s dvojkovým doplnkom druhého operandu.

*Príklad:* Ak N=00001101 je binárne vyjadrenie čísla 13, potom dvojkový doplnok (číslo −13) sa vypočíta ako:

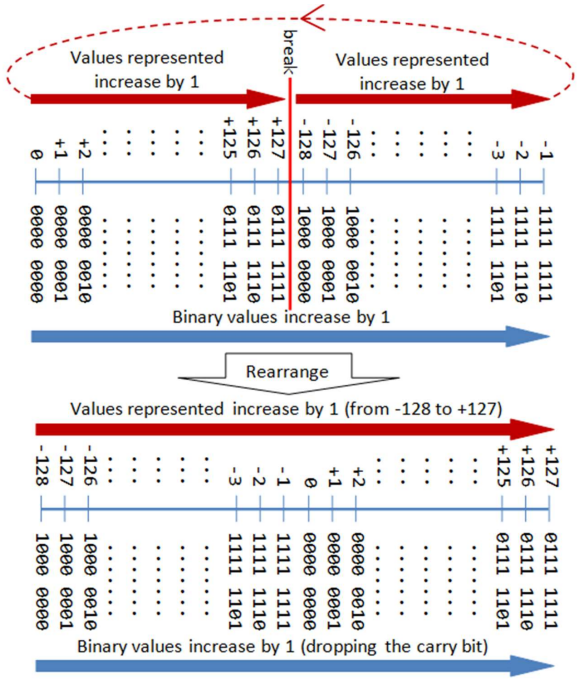
$$N_{2D} = 2^n - N = 2^8 - 13 = 100000000_2 - 00001101_2 = 11110011_2 = 243$$

Pokiaľ sa sčíta takto vyjadrené záporné číslo s iným záporným alebo väčším kladným číslom, dôjde k pretečeniu. Kód je ale zvolený tak, že po odrezaní bitu pretečenia dostaneme správny výsledok.

*Príklad:* Ak chceme vypočítať rozdiel 45 – 15, môžeme ho previesť na sčítanie 45 + $15_{2D}$

$$15_{2D} = 2^n - N = 2^8 - 15 = 100000000_2 - 00001111_2 = 11110001_2$$

$$45_{10} - 15_{10} = 45_{10} + (15)_{2D} = 00101101_2 + 11110001_2 = 1\,00011110_2 = 30_{10}$$



Obr. 18 Kódovanie binárnych čísel dvojkovým doplnkom

## 2.4. Bit, Byte

**Bit** (***Bi**nary digit***) je základnou jednotkou dát v číslicovej elektronike a v informatike. Označuje sa **b** a môže nadobúdať jednu z dvoch hodnôt **0** alebo **1**. Obe hodnoty môžeme chápať aj ako logické hodnoty (*true-false*, áno-nie), stavy (zapnutý-vypnutý).

Bit sa v číslicovej elektronike používa aj ako jednotka kapacity pamäti, tzn. jednotka množstva informácií, ktoré môžu byť v pamäti uložené (zapamätané).

Ak kapacitu podelíme časom, získame prenosovú rýchlosť, ktorej jednotkou je bit za sekundu (bit/s, bps – *bit per second*). Napríklad sériový prenos s prenosovou rýchlosťou 115200 kbit/s je schopný každú sekundu preniesť 115200 bitov.

**Byte** (bajt) je zoskupenie 8 bitov do jedného celku a označuje sa **B**. Jeden byte tvorený ôsmimi bitmi predstavuje $2^8$ = 256 celých čísel, teda čísla od 0 do 255. Tento interval čísel môžeme zapísať pomocou hexadecimálnych číslic ako $00_H$ až $FF_H$.

Rozsah čísel jedného bajtu sa od začiatku počítačovej techniky používal na kódovanie znakov klávesnice anglickej abecedy s rozlíšením veľkých aj malých písmen, vrátane číslic a základných interpunkčných znamienok. Spôsob kódovania znaku na číselnú hodnotu, popisuje kód **ASCII** (*American Standard Code for Information Interchange*). Tento kód (Tab. 4) však neobsahuje znaky s diakritickými znamienkami používané v iných jazykoch. Rozšírené verzie tohto kódu obsahujú aj rôzne znaky používané v jazykoch, ktoré píšu latinkou. 256 možností je ale málo pre potreby jazykov používaných pre inú abecedu (napr. ruština, čínština, japončina ...). Z tohto dôvodu vzniklo viacbajtové kódovanie znakov (*Unicode*).

## 2.5. BCD kód

Binárne kódované desiatkové číslo BCD (*Binary Coded Decimal*) je spôsob, ako vyjadriť každú číslicu z desiatkovej sústavy binárnym kódom. Keďže v systéme BCD existuje iba desať kódov, je veľmi jednoduché prevádzať medzi číslicami desiatkovej sústavy a BCD kódom (Tab. 3). Pretože radi čítame a píšeme v desiatkovej sústave, systém BCD poskytuje vynikajúce rozhranie pre binárne systémy. Príkladmi takýchto rozhraní sú vstupy z klávesnice a digitálne údaje. Systém BCD sa nazýva aj kód 8421. Binárne kódované desiatkové číslo znamená, že každá číslica 0 až 9, je reprezentovaná binárnym kódom so štyrmi bitmi. Označenie 8421 ($2^3$, $2^2$, $2^1$, $2^0$) označuje binárne váhy štyroch bitov.





Tab. 3 BCD kód číslic desiatkovej sústavy

| Decimal Digit | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| BCD | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 |

Tab. 4 Tabuľka ASCII znakov

| Base-16 | ASCII | Base-16 | ASCII | Base-16 | ASCII | Base-16 | ASCII |
|---|---|---|---|---|---|---|---|
| 00 | NULL | 20 | Space | 40 | @ | 60 | ‘ |
| 01 | SOH | 21 | ! | 41 | A | 61 | a |
| 02 | STX | 22 | " | 42 | B | 62 | b |
| 03 | ETX | 23 | # | 43 | C | 63 | c |
| 04 | EOT | 24 | $ | 44 | D | 64 | d |
| 05 | ENQ | 25 | % | 45 | E | 65 | e |
| 06 | ACK | 26 | & | 46 | F | 66 | f |
| 07 | BEL | 27 | ’ | 47 | G | 67 | g |
| 08 | BS | 28 | ( | 48 | H | 68 | h |
| 09 | HT | 29 | ) | 49 | I | 69 | i |
| 0A | LF | 2A | * | 4A | J | 6A | j |
| 0B | VT | 2B | + | 4B | K | 6B | k |
| 0C | FF | 2C | , | 4C | L | 6C | l |
| 0D | CR | 2D | - | 4D | M | 6D | m |
| 0E | SO | 2E | . | 4E | N | 6E | n |
| 0F | SI | 2F | / | 4F | O | 6F | o |
| 10 | DLE | 30 | 0 | 50 | P | 70 | p |
| 11 | DC1 | 31 | 1 | 51 | Q | 71 | q |
| 12 | DC2 | 32 | 2 | 52 | R | 72 | r |
| 13 | DC3 | 33 | 3 | 53 | S | 73 | s |
| 14 | DC4 | 34 | 4 | 54 | T | 74 | t |
| 15 | NAK | 35 | 5 | 55 | U | 75 | u |
| 16 | SYN | 36 | 6 | 56 | V | 76 | v |
| 17 | ETB | 37 | 7 | 57 | W | 77 | w |
| 18 | CAN | 38 | 8 | 58 | X | 78 | x |
| 19 | EM | 39 | 9 | 59 | Y | 79 | y |
| 1A | SUB | 3A | : | 5A | Z | 7A | z |
| 1B | ESC | 3B | ; | 5B | [ | 7B | { |
| 1C | FS | 3C | < | 5C | \ | 7C | \| |
| 1D | GS | 3D | = | 5D | ] | 7D | } |
| 1E | RS | 3E | > | 5E | ^ | 7E | ~ |
| 1F | US | 3F | ? | 5F | _ | 7F | DEL |

## 2.6. Grayov kód

Grayov kód je nevážený binárny kód, v ktorom sa dve po sebe nasledujúce hodnoty líšia iba o jeden bit (Tab. 5). To znamená, že k pozícii bitu nie sú priradené žiadne špecifické váhy. Keďže sa súčasne mení iba jeden bit, Grayov kód sa nazýva kód jednotkovej vzdialenosti a je tiež známy ako cyklický kód. Grayov kód sa nepoužíva na aritmetické operácie. Výhodou použitia Grayovho kódu je, že sa

dá vyhnúť prechodovým chybám. Počas procesu prechodu z jedného kódového čísla na nové sa rýchlosť prepínania každého bitu môže líšiť od ostatných. Napríklad proces prechodu z binárneho kódu 0111 na 1000 môže zaznamenať nasledujúce prechodné stavy: 0111→0011→1010→1000. V skutočnosti sú 0011 a 1010 prechodné stavy, o ktorých sa predpokladá, že sa neobjavia a nazývajú sa prechodnými chybami. Týmto prechodným chybám by sa malo v digitálnych systémoch vyhnúť.

Tab. 5 Grayov kód

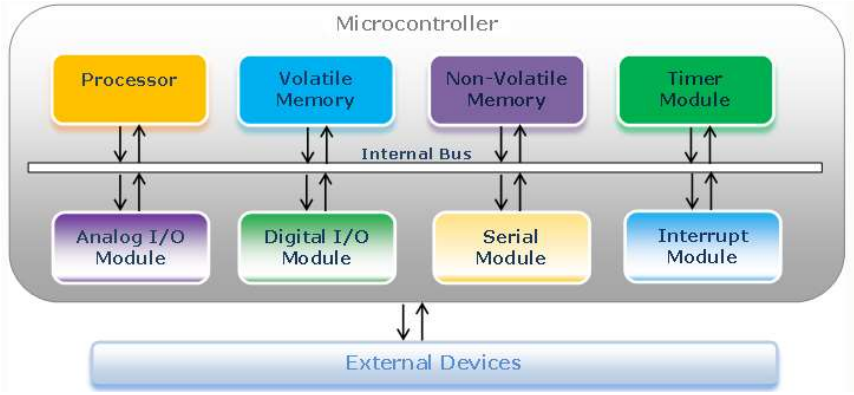
| Decimal | Binary | Gray | Decimal of Gray |
|---|---|---|---|
| 0 | 0000 | 0000 | 0 |
| 1 | 0001 | 0001 | 1 |
| 2 | 0010 | 0011 | 3 |
| 3 | 0011 | 0010 | 2 |
| 4 | 0100 | 0110 | 6 |
| 5 | 0101 | 0111 | 7 |
| 6 | 0110 | 0101 | 5 |
| 7 | 0111 | 0100 | 4 |
| 8 | 1000 | 1100 | 12 |
| 9 | 1001 | 1101 | 13 |
| 10 | 1010 | 1111 | 15 |
| 11 | 1011 | 1110 | 14 |
| 12 | 1100 | 1010 | 10 |
| 13 | 1101 | 1011 | 11 |
| 14 | 1110 | 1001 | 9 |
| 15 | 1111 | 1000 | 8 |





# 3. Charakteristika mikrokontrolérov AVR

Mikrokontrolér – MCU (*Micro Controller Unit*) združuje v jednom integrovanom obvode mikroprocesor, pamäť programu, dátovú pamäť a základné obvody pre komunikáciu s periférnymi zariadeniami (). MCU je Schopný samostatne fungovať bez prídavných obvodov, vyznačuje sa veľkou mierou spoľahlivosti a kompaktnosti, určený je pre jednoúčelové aplikácie ako je riadenie, regulácia, meranie a pod.



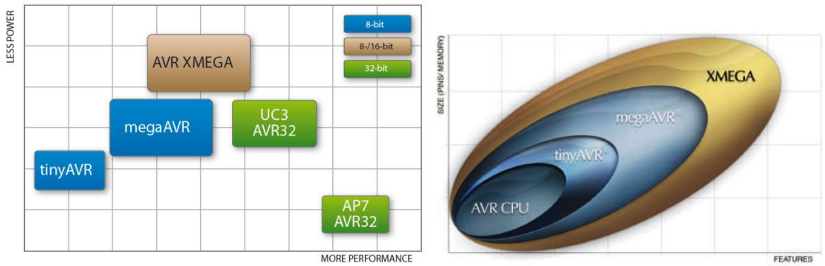

Obr. 19 Všeobecná bloková schéma mikrokontroléra

**AVR** je označenie rodiny 8-bitových a niektorých 32bitových mikrokontrolérom typu RISC s harvardskou architektúrou od firmy Atmel (dnes je súčasťou Microchip).

AVR mikrokontroléry sa delia do skupín (Obr. 20):

- tinyAVR: (ATtiny) 6-32 vývodové puzdro, 0,5 – 16 KB programovej pamäte, minimálne výbava podpory periférnych obvodov, 16 typov AVR,
- megaAVR: (ATmega) 28-100 vývodové puzdro, 4-256 KB programovej pamäte, rozšírený inštrukčný súbor, veľká podpora periférnych obvodov, 23 typov AVR,
- ASSP AVRs: megaAVR so špeciálnymi vlastnosťami, ktoré nie sú zastúpené u ostatných členov rodiny AVR, 8 typov AVR. Napr.: LCD radič, USB radič, CAN zbernica a pod. Oblasti použitia sú: *automotive*, riadenie motorov, manažment batérií,
- XMega: sú 8/16 bitové mikrokontroléry s nízkym príkonom, vysokým výkonom a bohatou ponukou periférnych zariadení. Sú rozšírením magaAVR.
- 32-bitové AVR: 32-bitovej architektúra nesúvisí s 8-bitovými AVR ale je to Atmel ekvivalent k procesorom s jadrom ARM. Má podporu funkcií pre spracovanie zvuku a videa. Inštrukčný

súbor je podobný ostatným RISC mikrokontrolérom ale nie je kompatibilný s pôvodným AVR (ani s ARM jadrami).




Obr. 20 Prehľad AVR mikrokontrolérov podľa výkonu

Hlavnou funkciou **jadra AVR** je zabezpečiť správne vykonávanie programu. CPU preto musí mať prístup k pamätiam, vykonávať výpočty, ovládať periférie a spracovávať prerušenia. Výkon jadra mikrokontrolérov AVR je garantovaný rozšíreným inštrukčným súbor a 32 univerzálnymi registrami. Všetky registre sú priamo pripojené s aritmeticko-logickou jednotkou (ALU) procesora, čo umožňuje súčasný prístup k dvom nezávislým registrom v jednom strojovom cykle. Architektúra AVR mikrokontrolérov je viac kódovo efektívna (priateľská k strojovému kódu), čím sa dosahuje až desaťkrát vyšší výkon ako u konvenčných CISC mikrokontrolérov. Aby sa maximalizoval výkon a paralelizmus, AVR používa Harvardskú architektúru so samostatnými pamäťami a zbernicami pre program a dáta. Inštrukcie v pamäti programu sa vykonávajú s jednoúrovňovým zreťazením (*pipeline*). Zatiaľ čo sa vykonáva jedna inštrukcia, ďalšia inštrukcia je načítava z pamäte programu. Tento koncept umožňuje vykonávať inštrukcie v každom taktovacom cykle.

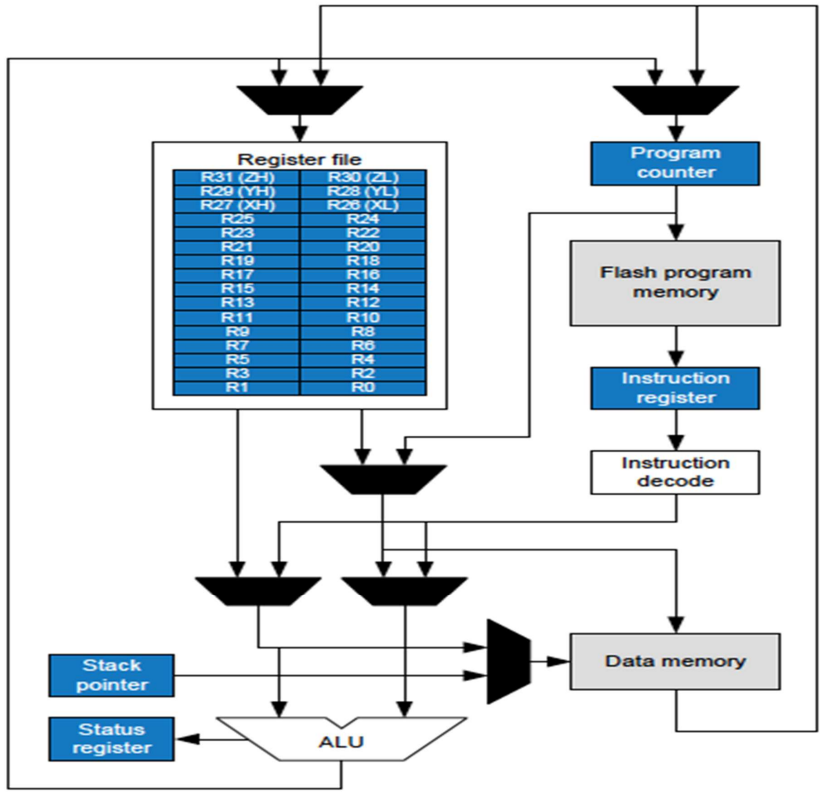
**Súbor registrov** s rýchlym prístupom obsahuje 32 x 8-bitových pracovných registrov s dobou prístupu jedného hodinového cyklu. Šesť z 32 registrov môže byť použitých ako tri 16-bitové nepriame adresové smerníky pre adresovanie dátového priestoru. Tieto 16-bitové registre sú označené X, Y a Z.

**ALU** vykonáva aritmetické a logické operácie medzi registrami alebo medzi konštantou (hodnotou z dátovej pamäte) a registrom. V typickej operácii ALU sú zo súboru registrov zvolené dva operandy, operácia sa vykoná a výsledok sa uloží späť do súboru registrov v jednom hodinovom cykle. Operácie s jedným registrom možno vykonávať aj v ALU.

Po aritmetickej operácii sa aktualizuje **Status** register (SREG), aby obsahoval informácie o výsledku každej operácie. Každý bit status registra má svoj význam a možno ich použiť na zmenu toku programu (vetvenie):








Obr. 21 Bloková schéma jadra AVR

I - *Global Interrupt Enable* – globálne povolenie prerušenia v MCU. Povolenia prerušení od jednotlivých zariadení sa vykonáva v samostatných riadiacich registroch daných zariadení. Ak sa zakáže globálne prerušenie, žiadne z prerušení sa nepovolí nezávisle od nastavení povolenia jednotlivých prerušení.

T - *Copy Storage* – zapamätanie kópie slúži na dočasné zapamätanie stavu ľubovoľného bitu zo súboru registrov. Do bitu T sa kopíruje pomocou inštrukciou BLD (*Bit Load*) a z T je možné skopírovať bit do bitu v súbore registrov inštrukciou BST (*Bit Store*).

H - *Half Carry Flag* – označuje polovičný prenos v niektorých aritmetických operáciách (BCD čísla).

S - *Sign Flag*, $S = N \oplus V$ - je *exclusive or* medzi N bitom a V bitom SREG registra.

V - *Two's Complement Overflow Flag* - príznak pretečenia dvojkového doplnku.

N - *Negativ Flag* - označuje záporný výsledok v aritmetickej alebo logickej operácií.

Z - *Zerro Flag* – označuje nulový výsledok v aritmetickej alebo logickej operácií.

C - *Carry Flag* – označuje prenos v aritmetickej alebo logickej operácií.

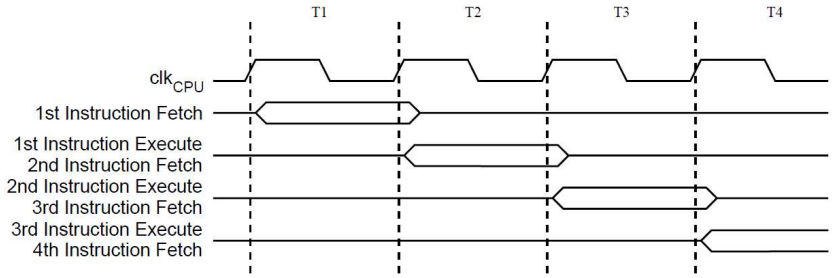
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x3F (0x5F) | I | T | H | S | V | N | Z | C | SREG |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Obr. 22 SREG - status register

Počas prerušení a volaní podprogramov je návratová adresa pre programové počítadlo PC (*Program Counter*) uložená v zásobníku. PC obsahuje adresu nasledujúcej inštrukcie v pamäti programu, ktorá sa bude vykonávať. Zásobník (*Stack*) sa používa hlavne na ukladanie dočasných údajov, lokálnych premenných a návratových adries po prerušeniach a volaniach podprogramov. Je implementovaný ako rastúci z vyšších na nižšie miesta pamäte (LIFO). Zásobník je alokovaný v dátovej SRAM pamäti, pričom veľkosť zásobníka je obmedzená len celkovou veľkosťou SRAM a jej využitím.

Všetky používateľské programy musia inicializovať **Stack Pointer** (SP) pred vykonaním podprogramov alebo prerušení. Register ukazovateľa zásobníka vždy ukazuje na vrch zásobníka, kde sa nachádzajú návratové adresy podprogramov a prerušení. Pri štarte programu sa musí vymedziť oblasť v SRAM pamäti pre zásobník a nastaví sa ukazovateľ zásobníka na vrchol zásobníka. Pretečenie zásobníka (do zásobníka sa zapíše viac hodnôt ako je jeho vyhradená veľkosť) je nežiadúce a dôsledok takejto situácie je pád programu.

CPU AVR je taktovaný hodinami **clkCPU**, ktoré sú priamo generované z vybraného zdroja hodín pre čip. Paralelné načítanie inštrukcií a vykonávanie inštrukcií (*Pipline*) umožňuje Harvardská architektúra a koncept CPU registrov s rýchlym prístupom. V jednom hodinovom cykle sa vykoná operácia aritmetickej logickej jednotky s použitím dvoch operandov zo súboru registrov CPU a výsledok sa uloží späť do cieľového registra (Obr. 23).




Obr. 23 Paralelné vykonávanie inštrukcií



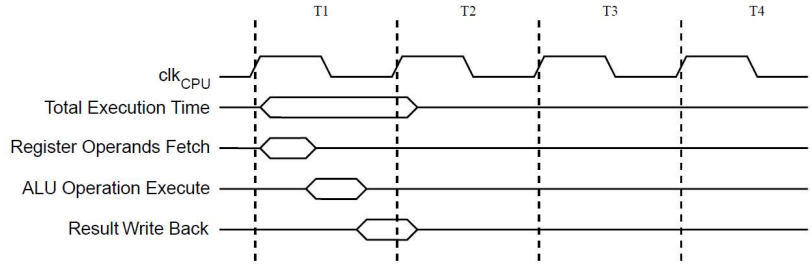


Činnosť v jednotlivých periódach je nasledovná:

- T1: vyzdvihnutie 1. inštrukcie z pamäte programu,
- T2: vykonanie 1. inštrukcie a vyzdvihnutie 2. inštrukcie z pamäte programu,
- T3: vykonanie 2. inštrukcie a vyzdvihnutie 3. inštrukcie z pamäte programu atď.

Obr. 24 znázorňuje časový priebeh operácie medzi pracovným registrom a aritmeticko-logickou jednotou. V jednom hodinovom cykle sa vyberie operand z registra (*Register Operands Fetch*), vykoná sa operácia v ALU (*ALU Operation Execute*) a výsledok sa uloží do cieľového registra (*Result Write Back*).

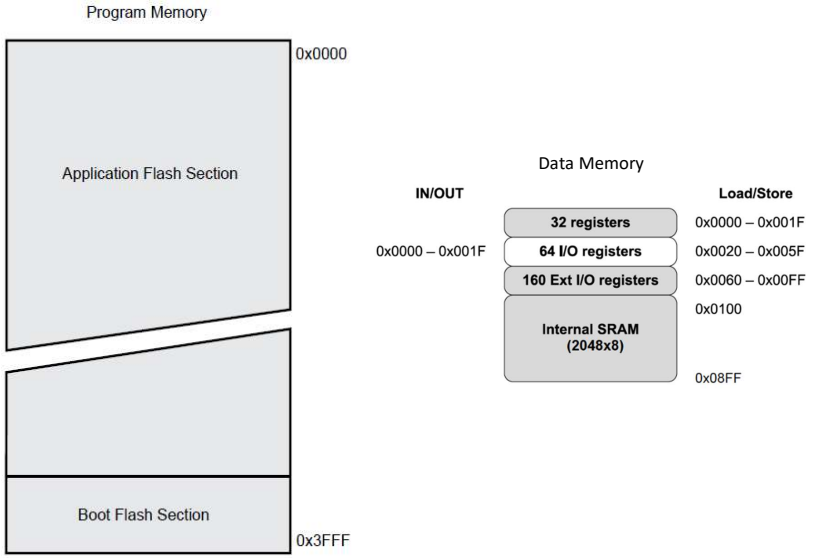
Väčšina inštrukcií trvá len jeden alebo dva hodinové cykly, čo robí AVR pomerne rýchlym medzi osembitovými MCU. Pretože všetky inštrukcie (okrem násobenia a 16-bitového sčítania / odčítania) na svoju činnosť potrebujú jeden strojový cyklus, AVR môžu vykonať až 1 milión inštrukcií za sekundu s 1 MHz taktovacou frekvenciou, teda procesor s taktovacou frekvenciou 8 MHz môže dosiahnuť výpočtový výkon 8 MIPS (*Million Instructions Per Second*). Načítanie a ukladanie do pamäte a vetvenie programu trvajú dva strojové cykly. Procesory AVR boli navrhnuté s ohľadom na efektívne spracovanie strojového kódu, ktorý je výsledkom prekladu z jazyka C. AVR bežne pracujú s taktovacou frekvenciou v rozsahu 0 až 20 MHz.




Obr. 24 Perióda vykonávania operácie v ALU

**Prerušenie** (*Interrupt*) je nástroj pre asynchrónnu obsluhu podnetov činnosti procesora. Procesor preruší vykonávanie postupnosti inštrukcií, vykoná obsluhu prerušenia a potom pokračuje v predchádzajúcej činnosti. Pri prerušení sa uložia potrebné dáta v registroch a v zásobníkové pamäti. Jednotlivé prerušenia majú priradenú prioritu, vykonávaná obsluha prerušenia môže byť opäť prerušená iba prerušením s vyššou prioritou. Pokiaľ nastane prerušenie s nižší prioritou, potom musí jeho obsluha počkať, až nebude prebiehať obsluha žiadneho prerušenia s vyššou prioritou.

Mikrokontroléry AVR obsahujú *On-chip In-System Reprogrammable Flash* pamäť na ukladanie programu. Keďže všetky inštrukcie AVR majú šírku 16 alebo 32 bitov, pamäť Flash má organizáciu 32K x 16b. Veľkosť programovej pamäte sa zvyčajne uvádza pri pomenovávaní samotného MCU (napr. ATmega64x má 64 KB Flash pamäte, zatiaľ čo ATmega32x má 32 KB Flash pamäte). Neexistuje možnosť rozšírenia programovej pamäti. Strojový kód programu musí byť umiestnený v *on-chip* Flash programovej pamäti AVR. Flash programovú pamäť AVR mikrokontrolérov je možné rozdeliť na *Application Flash Section* (časť pre aplikačný program) a *Boot Flash Section* (časť pre program na nahrávanie strojového kódu do aplikačnej časti) (Obr. 25).




Obr. 25 Adresový priestor programovej a dátovej pamäte MCU AVR

**Dátová pamäť** je prístupná cez štandardnú dátovú zbernicu. Dátovú pamäť tvorí: 32 pracovných registrov, vstupno-výstupné (I/O) registre, rozšírené I/O registre (závisí od zariadenie) a internej dátovej pamäte SRAM. Pracovné registre (R0-R31) zaberajú priestor prvých 32 adries 0x0000 - 0x001F, po ktorých nasleduje 64 vstupno-výstupných registrov 0x0020 - 0x005F. *V MCU s rozšíreným počtom periférnych zariadení nasleduje 160 registrov pre tieto zariadenia 0x0060 - 0x00FF*. Interná dátová pamäť začína na adrese 0x0060 alebo 0x0100 (MCU s rozšírenými perifériami) (Obr. 25)



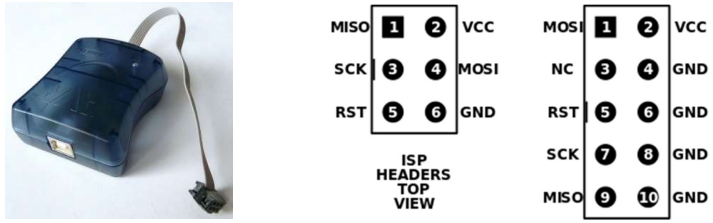


MCU AVR obsahujú aj **IN/OUT** dátovú zbernicu pre priamy prístup do 64-bajtovej I/O pamäte (64 I/O registrov) pomocou adresy 0x0000 až 0x003F. K tejto časti pamäti je možné pristupovať aj štandardnou dátovou zbernicou s použitím posunu adresy o 0x0020 (Obr. 25).

Takmer všetky mikrokontroléry AVR obsahujú vnútornú dátovú pamäť **EEPROM**. Elektricky zmazateľná pamäť je organizovaná ako samostatný dátový priestor, z ktorého je možné čítať a zapisovať. Do EEPROM je možné pristupovať iba spôsobom ako k integrovaným externým zariadeniam pomocou špeciálnych inštrukcií na čítanie a zápis, čím sa stáva prístup do EEPROM oveľa pomalší ako do internej pamäte SRAM. Vzhľadom na to, že počet zápisov do EEPROM je obmedzený (100 000 cyklov zápisov) preto, zapisovaniu do EEPROM by mal predchádzať test obsahu pamäťovej bunky s požadovaným obsahom zápisu a vykonať zápis iba vtedy, ak je potrebné zmeniť obsah.

Na uloženie (nahranie, *uploading*) strojového kódu programu do programovej pamäte na čipe AVR (*upload*) sa používajú tieto metódy.

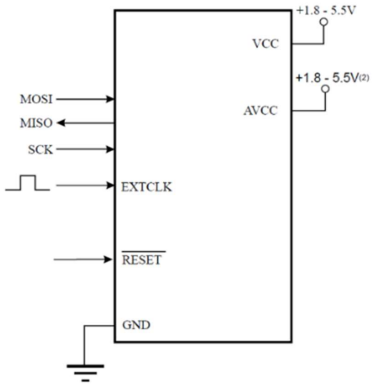
**ISP** (***I**n **S**ystem **P**rogramming*) – je metóda programovania pamäte (*uploading*) bez potreby vyberania MCU z dosky plošného spoja (DPS). Programovanie sa vykonávaná prostredníctvom **SPI** (***S**erial **P**eripheral **I**nterface*) komunikačnej zbernice a riadiaceho signálu Reset. Na programovanie sa môže používať programátor **Atmel AVRISP mkII**, ktorý sa pripája k USB počítača a pomocou 6 alebo 10 vývodového konektora k MCU na DPS (Obr. 26). Pomocou IDE vývojového prostredia *Atmel Studio* je možné nahrať dáta do pamäte mikrokontroléra. Toto je najbežnejší spôsob, vývoja aplikačného programu pre AVR.



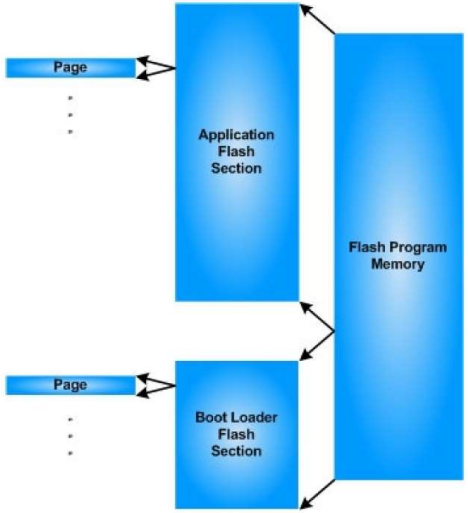

Obr. 26 Programátor AVRISP mkII a konektor so signálmi ISP

***Boot Loader*** je možnosť, ktorou disponujú mikroprocesory AVR, ktorá spočíva v tom, že je možné programovú pamäť rozdeliť na dve nezávislé časti (Obr. 28). Táto funkcia umožňuje flexibilné aktualizácie aplikačného softvéru riadené MCU pomocou rezidentného programu v *Boot Loader Flash Section*. Programový kód v sekcii *Boot Loader* má schopnosť zapisovať do celej pamäte Flash. *Boot Loader* sa môže upravovať sám a môže sa aj vymazať z pamäte, ak funkcia už nie je

potrebná. Pri zapnutí napájania alebo pri resetovaní procesora sa najprv vykonáva *Boot Loader*, ktorého úlohou je určiť, či existuje požiadavka o naprogramovanie strojového kódu do aplikačnej časti pamäte. Ak nie je požiadavka o naprogramovanie, spustí sa strojový kód v aplikačnej pamäti. Ak je požiadavka o *uploading*, uloží sa nový program do aplikačnej pamäte a následne sa spustí. Aplikačný program je možné naprogramovať prostredníctvom sériového rozhrania USART.




Obr. 27 Schéma zapojenia signálov pre sériové programovanie




Aplikačná časť – obsahuje hlavný strojový kód (program).
Boot Loader – obsahuje strojový kód (program) pre „uploading“.

Toto delenie umožňuje v AVR procesoroch mať prístup k dvom nezávislým aplikáciám. Obe sekcie majú vyhradené blokovacie bity na ochranu zápisu a čítania.

Obr. 28 Rozdelenie programovej Flash pamäte v AVR

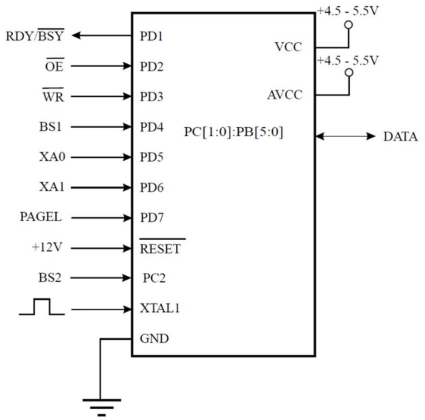




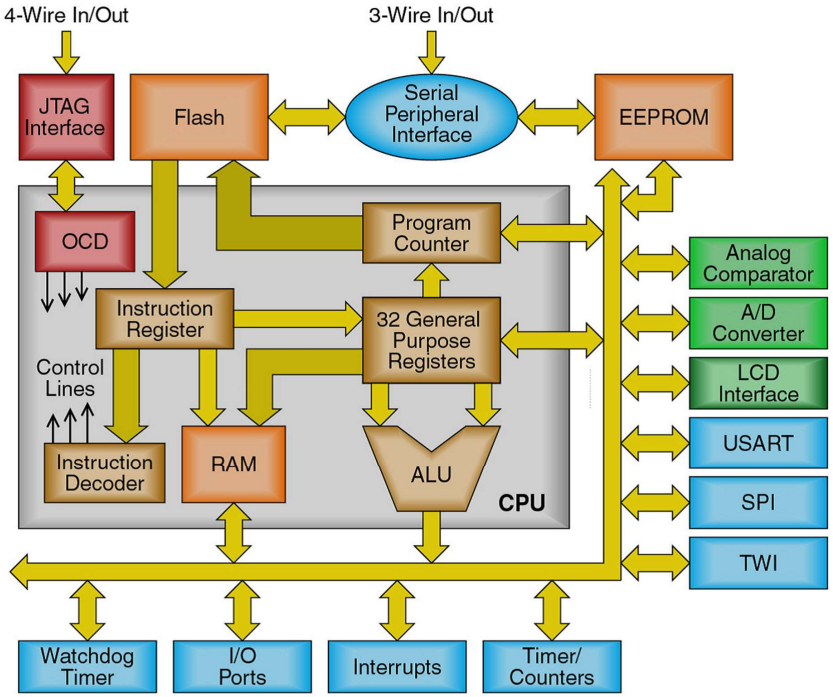
**JTAG** (*Joint Test Action Group*) je testovací štandard, ktorý poskytuje prístup k funkciám ladenia na čipoch procesorov, zatiaľ čo procesor je spusteným v užívateľskom systéme. JTAG umožňuje prístup k internej pamäti a registrom, zastavenie vykonávania strojového kódu na požadovanom mieste (break point), spustenie, zastavenie a sledovanie správania sa systému. Existuje niekoľko JTAG adaptérov (programátorov a ladiacich nástrojov) pre AVR (Obr. 29).

Obr. 29 ATMEL-ICE-BASIC hardvér na ladenie a programovanie MCU AVR a SAM

Väčšina MCU AVR Tiny a Mega obsahujú programovacie rozhranie pre **paralelné programovanie**, ktoré vyžaduje použitie „vysokého napätia“ 12 V privedeného na resetovací vývod MCU a vyžaduje sa prístup k väčšiemu počtu vývodov mikrokontroléra (Obr. 30). Z týchto dôvodov sa toto rozhranie primárne používa na výrobné programovanie MCU. Paralelné programovanie je vždy povolené, pretože ho nemožno neúmyselne vypnúť nastavením poistky alebo akciou používateľa. V praxi sa skoro vôbec nepoužíva, lebo MCU je spájkovaný na doske plošného spoja a vývody sú už pevne prepojené s elektronickými súčiastkami.




Obr. 30 Schéma zapojenia signálov pre paralelné programovanie




Obr. 31 Bloková schéma AVR mikrokontrolérov

Na Obr. 31 je bloková schéma mikrokontrolérov AVR, na ktorej sú k AVR jadru pripojené jednotlivé periférne zariadenia cez 8 bitovú dátovú zbernicu. Nastavovanie jednotlivých zariadení v MCU sa realizuje pomocou registrov v SRAM pamäti v časti *I/O registers* alebo *Ext I/O registers*. Komunikácia CPU s jednotlivými zariadeniami prebieha po 8-bitovej zbernici. Počet typov zariadení, ktoré MCU obsahuje je závislý od typu MCU. Pre účely predmetu Základy mikroprocesorovej techniky budeme pracovať s MCU ATmega328P na vývojovej doske Arduino UNO. Z tohto dôvodu je táto učebnica zameraná na tento typ AVR MVU. ATmega328P neobsahuje tieto zariadenie z blokovej schémy alebo sa nebudú používať (Obr. 31):

- JTAG Interface,
- LCD Interface,
- OCD (*On-Chip Debuger*) nebudeme používať. K jeho činnosti je potrebný hardvérový emulátor.





## 3.1. *Fuse* bity

AVR mikroprocesory obsahujú 3registre nastavovacích *Fuse* bitov (poistiek), ktoré bližšie špecifikujú vlastnosti MCU pre konkrétnu hardverovú aplikáciu. Zoznam týchto bitov a ich vplyv na funkciu MCU je v nasledujúcich tabuľkách (Tab. 6, Tab. 7, Tab. 8).

Tab. 6 *High Fuse* register, prednastavená hodnota: 0xD9

| ***High Fuse* register** | **Bit** | **Popis** | **Prednastavená hodnota** |
|---|---|---|---|
| **RSTDISBL** | 7 | Zakázanie externého *reset* vstupu | 1 nezakázané |
| **DWEN** | 6 | Povolenie *debugWire* | 1 nepovolené |
| **SPIEN** | 5 | Povolenie nahrávania programu a dát sériovým programovaním | 0 povolené SPI programovanie |
| **WDTON** | 4 | WDT vždy povolený | 1 nepovolený |
| **EESAVE** | 3 | Obsah EEPROM sa zachová pri mazaní čipu (*Chip Erase*) | 1 EEPROM sa nezachová |
| **BOOTSZ1** | 2 | Veľkosť pamäte pre *Boot Loader* | 0 *256, 512, 1024,* ***2048*** *words* |
| **BOOTSZ0** | 1 | Veľkosť pamäte pre *Boot Loader* | 0 *256, 512, 1024,* ***2048*** *words* |
| **BOOTRST** | 0 | Výber, povolenie *reset boot* vektora | 1 adresa 0x000 |

Tab. 7 *Low Fuse* register, prednastavená hodnota: 0x62

| ***Low Fuse* register** | **Bit** | **Popis** | **Prednastavená hodnota** |
|---|---|---|---|
| **CKDIV8** | 7 | Delenie frekvencie hodín 8 | 0 delenie frekvencie 8 |
| **CKOUT** | 6 | Povolenie výstupu hodinového signálu na vývode CLKO | 1 nepovolený |
| **SUT1** | 5 | Bit 1 voľby času nábehu MCU | 1 čas nábehu MCU 65ms |
| **SUT0** | 4 | Bit 0 voľby času nábehu MCU | 0 čas nábehu MCU 65ms |
| **CKSEL3** | 3 | Bit 3 výber zdroja hodinového signálu | 0 interný oscilátor 8MHz |
| **CKSEL2** | 2 | Bit 2 výber zdroja hodinového signálu | 0 interný oscilátor 8MHz |
| **CKSEL1** | 1 | Bit 1 výber zdroja hodinového signálu | 1 interný oscilátor 8MHz |
| **CKSEL0** | 0 | Bit 0 Výber zdroja hodinového signálu | 0 interný oscilátor 8MHz |

Bity CKSEL[3:0] sú prednastavené, tak že je zvolený interný 8MHz oscilátor. Ak zoberieme do úvahy nastavenie bitu CKDIV8, potom frekvencia hodinového signálu bude 8MHz / 8 = 1 MHz

Tab. 8 Extra *Fuse* register, prednastavená hodnota: 0xFF

| Extra *Fuse* register | Bit | Popis | Prednastavená hodnota |
|---|---|---|---|
| BODLEVEL2 | 2 | Bit 2 úroveň výpadku napätia 1,8V | 1 nefunkčné |
| BODLEVEL1 | 1 | Bit 1 úroveň výpadku napätia 2,7V | 1 nefunkčné |
| BODLEVEL0 | 0 | Bit 0 úoveň výpadku napätia 4,3V | 1 nefunkčné |

Bity BODLEVEL[2:0] určujú, pri akom napájacom napätí MCU zastaví svoju činnosť (reset stav) z dôvodu jeho poklesu. Význam týchto bitov je v Tab. 9.

Tab. 9 Význam BODLEVEL2 bitov

| BODLEVEL [2:0] Fuses | Min. $V_{BOT}$ | Typ. $V_{BOT}$ | Max $V_{BOT}$ | Units |
|---|---|---|---|---|
| 111 | BOD Disabled | | | |
| 110 | 1.7 | 1.8 | 2.0 | V |
| 101 | 2.5 | 2.7 | 2.9 | |
| 100 | 4.1 | 4.3 | 4.5 | |

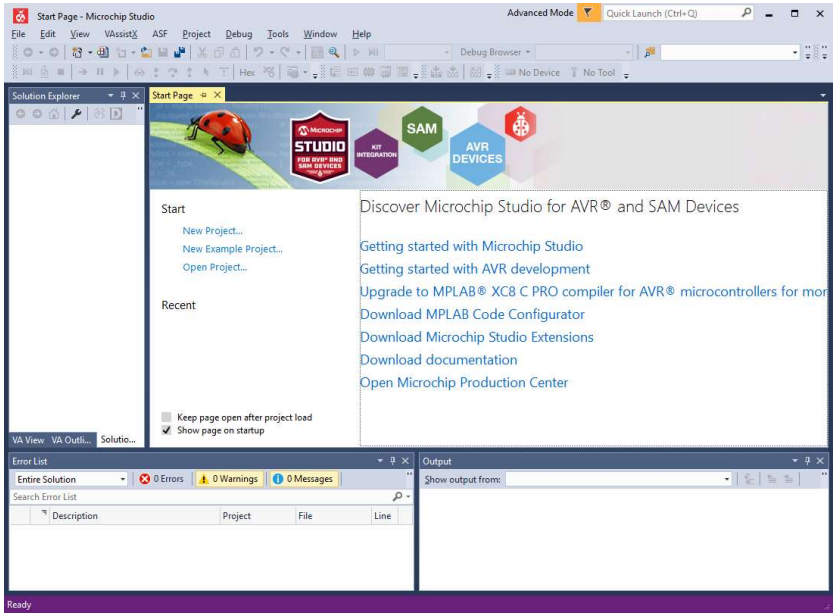
Nastavenie *Fuse* bitov je možné meniť napríklad v *Microchip Studio* pomocou externého programátora, alebo pomocou avrdude.exe a externého programátora.





# 4. Vývojové prostredie pre AVR mikrokontroléry

*Microchip Studio* je integrované vývojové prostredie (IDE - *Integrated Development Environment*) na písanie a ladenie aplikácií AVR v prostredí Windows [23]. *Microchip Studio* poskytuje nástroj na riadenie projektov, editor zdrojových súborov, simulátor, *assembler* (jazyk symbolických inštrukcií) a *front-end* pre C/C++, programovanie a ladenie na čipe. *Microchip Studio* má modulárnu architektúru, ktorá umožňuje interakciu s dodávateľmi softvéru tretích strán. Zásuvné moduly GUI a ďalšie moduly je možné napísať a pripojiť k systému. Umožňuje pripojenie k debuggerom, programátorom a vývojovému hardvéru, ktoré pracujú s AVR alebo SAM zariadeniami.



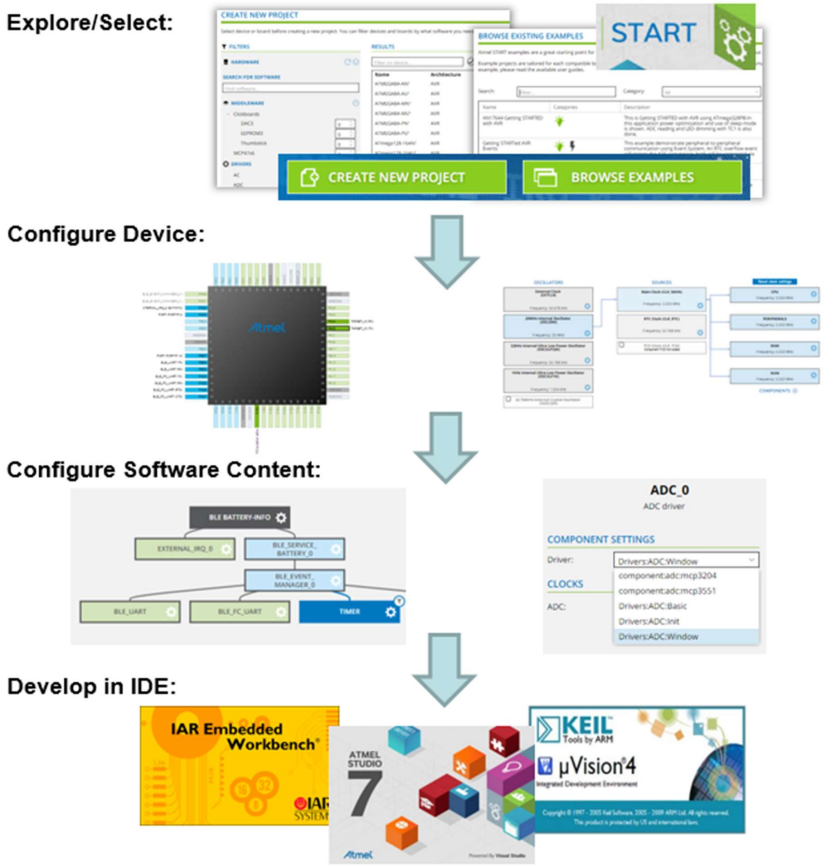

Obr. 32 IDE *Microchip Studio*

*Microchip Studio* poskytuje veľkú sadu funkcií pre vývoj a ladenie projektov:

- Editor kódu pre C/C++ a *Assembler* s rozšírením *Visual Assist Simulátor*,
- Simulátor s pokročilými funkciami ladenia,
- Vytváranie modulárnych aplikácií na akejkoľvek platforme AVR,
- Ladenie na skutočných zariadeniach pomocou nástrojov na ladenie ,
- SDK (*Software Development Kit*) umožňujúce integráciu zákazníckych doplnkov.

IDE je možné používať na vývoj aplikácie aj s využitím softvérových komponentov, ako sú ovládače a *middleware*, nakonfigurované a exportované z *Atmel START* alebo MCC (*MPLAB Code Configurator*).

Webový softvérový konfiguračný nástroj *Atmel START* pomáha s vývojom aplikácie pre ARM a SAM MCU [22]. Umožňuje vybrať a konfigurovať softvérové komponenty (z ASF4 a AVR kódu), aby sa prispôsobili konkrétnemu MCU v konkrétnej aplikácii. Po dokončení hardvérovej a softvérovej konfigurácie umožňuje vygenerovať projekt a otvoriť ho v IDE alebo vygenerovať *makefile*.




Obr. 33 Webový softvérový nástroj *Atmel START*

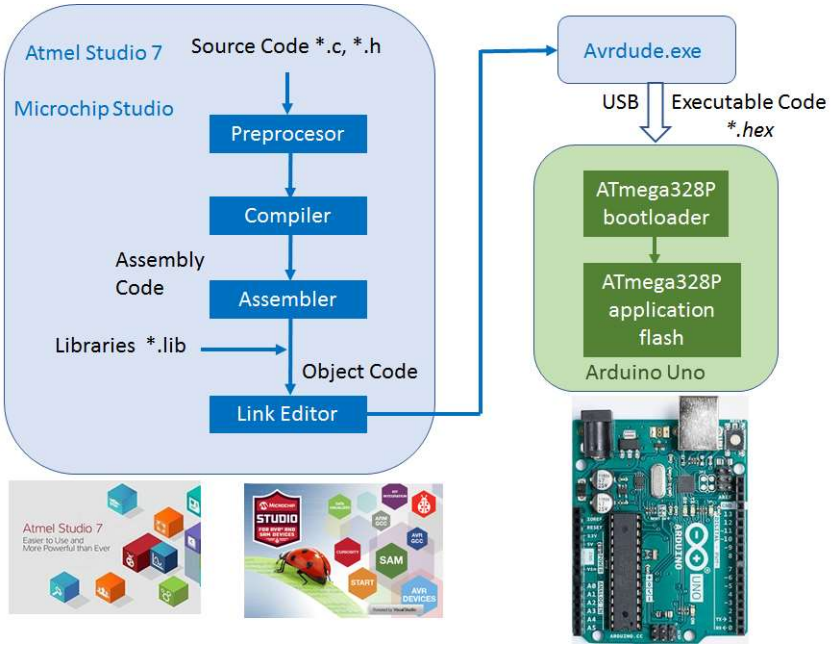




ASF (*Advanced Software Framework*) poskytuje bohatú sadu ovládačov a kódových modulov vyvinutých za účelom skrátenia času potrebného na návrh užívateľskej aplikácie. Zjednodušuje používanie mikrokontrolérov tým, že poskytuje abstrakciu hardvéru prostredníctvom ovládačov a vysokohodnotných *middleware* (softvér, ktorý pomáha vývojárom vytvárať a nasadzovať aplikácie efektívnejšie. Funguje ako prepojenie medzi aplikáciami, údajmi a používateľmi). ASF je bezplatná knižnica s otvoreným zdrojovým kódom navrhnutá na použitie vo fázach hodnotenia, vývoja prototypov, dizajnu a výroby.

ASF4 je štvrtou generáciou ASF a podporuje produktový rad mikrokontrolérov SAM. ASF4 sa musí používa v spojení s MCC (*MPLAB Code Configurator*), ktorý nahrádza ASF *Wizard* z ASF2 a ASF3.

*MPLAB Code Configurator* je grafické programovacie prostredie, ktoré generuje ľahko pochopiteľný kód v jazyku C pripravený pre vloženie do projektu. Pomocou intuitívneho rozhrania umožňuje konfigurovať periférie MCU pre konkrétnu aplikáciu. MCC [20] je vhodný aj pre *Microchip Studio*, ako rozšírenie a je to náhrada konfiguračného nástroja *Atmel START* pre nové projekty.



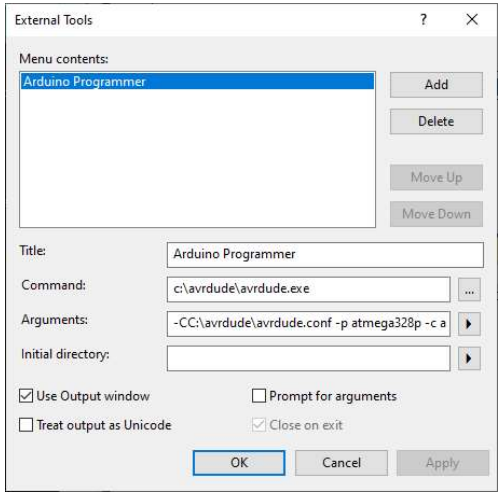

Obr. 34 Proces tvorby aplikácie pre ATmega328P v *Microchip Studio*

### 4.1. Softvérový nástroj AVRDUDE

Softvérový nástroj *AVRDUDE - AVR Downloader Uploader* - je program na sťahovanie a nahrávanie dát do pamätí na čipe mikrokontrolérov AVR [21]. Dokáže naprogramovať *Flash*, EEPROM pamäte, dokáže naprogramovať *Fuses* a blokovacie bity. *AVRDUDE* je možné použiť cez príkazový riadok alebo prostredníctvom interaktívneho (terminálneho) režimu. Použitie AVRDUDE z príkazového riadku funguje dobre na programovanie celej pamäte čipu z obsahu súboru, zatiaľ čo interaktívny režim je užitočný na skúmanie obsahu pamäte, programovanie Fuse bitov a blokovacích bitov. *AVRDUDE* podporuje tieto základné typy programátorov: Atmel STK500, Atmel AVRISP a AVRISP mkII zariadenia, Atmel STK600, Atmel JTAG ICE.

Na cvičeniach bude *AVRDUDE* používaní v spojení s vývojovým prostredím *Microchip Studio*. Na požiadavku nahrať strojový kód (*.hex súbor) do pamäte MCU vývojové prostredie zabezpečí spustenie programu *avrdude.exe* z príkazového riadku. Preto je potrebné do vývojového prostredia pridať externý nástroj na programovanie MCU podľa (Obr. 35). Predpokladom správneho fungovania je inštalácia súborov *avrdude.exe* a *avrdude.conf* do počítača. V argumentoch je použitý parameter *-P COM6* , ktorý definuje USB sériový port, ktoré prideľuje Windows po pripojení Arduino UNO k počítaču. U každého užívateľa môže mať inú hodnotu a preto je potrebné ho aktualizovať.

```
Title:      Arduino Programmer
Command:    c:\avrdude\avrdude.exe
Arguments: -CC:\avrdude\avrdude.conf -p atmega328p -c arduino -P COM6
           -b 115200 -U flash:w:"$(ProjectDir)Debug\$(TargetName).hex":i
```




Obr. 35 Definovanie externého programátora pre *Microchip Studio*





## 5. Mikrokontrolér ATmega328P

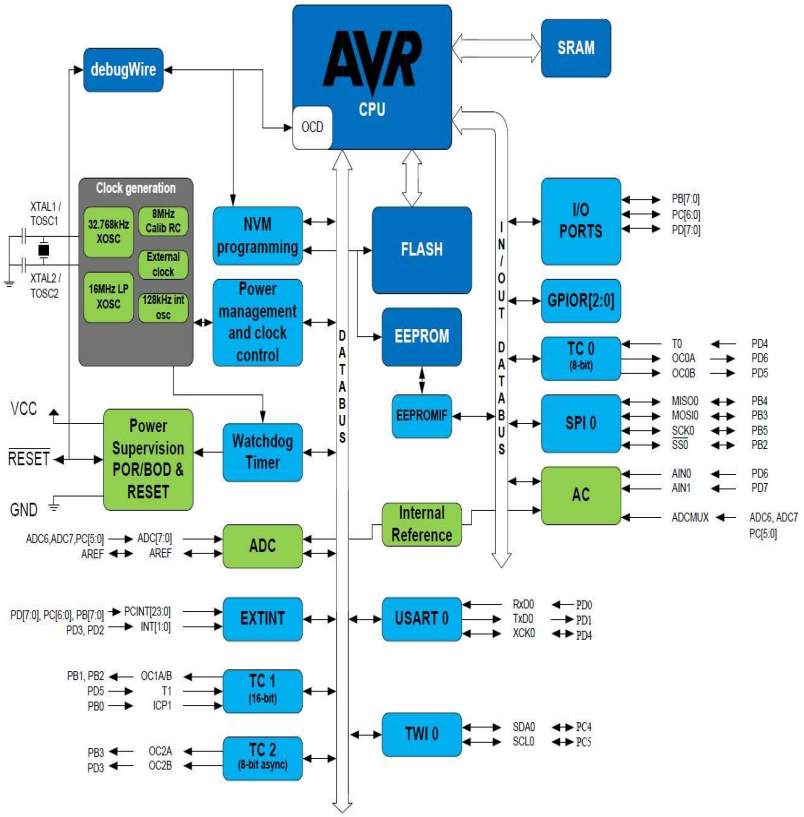
ATmega328 je CMOS 8-bitový mikrokontrolér s vylepšenou RISC AVR architektúrou. ATmega328 je charakteristický nasledujúcimi zariadeniami a funkciami (Obr. 36):

- **Rozšírená RISC architektúra**: 131 inštrukcií v inštrukčnom súbore, 32 univerzálnych pracovných registrov vykonávanie inštrukcie prevažne v 1 hodinovom cykle, taktovacia frekvencia od 0 do 20 MHz,
- **Integrované pamäte**: 32KByte programovateľná Flash pamäť, 1KByte EEPROM, 2KByte SRAM,
- **Periférne zariadenia**:
  - 2x 8-bitové časovače / počítadlá s režimami komparátora TC0, TC2,
  - 1x 16-bitový časovač / počítadlo s režimom komparátora a merania TC1,
  - RTC počítadlo (*Real Time Counter*) s oddeleným oscilátorom (*Clock generation*),
  - 6 PWM výstupných kanálov,
  - 6-kanálový (PDIP puzdro) alebo 8-kanálový (TQFP, QFN/MLF puzdro) 10-bit ADC prevodník, senzor teploty čipu,
  - 1x *Master*/*Slave* SPI sériový port: SPI0,
  - 1x programovateľný sériový port: USART0,
  - 2-vodičové sériové rozhranie (I2C): TWI0,
  - programovateľný časovač (*Watchdog Timer*) so samostatným vnútorným oscilátorom,
  - 1x analógový komparátor: AC,
  - modul prerušenia,
- **Špeciálne vlastnosti a funkcie**: *Power-on Reset* a programovateľný *Brown-out Detection*, vnútorný kalibrovaný oscilátor, externé a interné zdroje prerušení, šesť režimov šetrenia energie,
- **Digitálne vstupy a výstupy**: 23 programovateľných vstupno-výstupných vývodov PB[7:0], PC[6:0], PD[7:0],
- **Charakteristiky**: pracovné napätie 1,8V až 5,5V; (0 - 4MHz @ 1,8 – 5,5V; 0 - 10MHz @ 2,7 – 5,5V; 0 - 20MHz @ 4,5 – 5,5V), teplotný rozsah -40°C až 105°C.

Popis hardvéru v nasledujúcej časti učebnice sa bude vzťahovať na MCU ATmega328P. Tento MCU je použitý na vývojovej doske Arduino UNO.

# 6. Pamäťový priestor

Architektúra jadra AVR má dva hlavné pamäťové priestory, dátovú pamäť a pamäť programov. Okrem toho je zariadenie vybavené pamäťou EEPROM na ukladanie dát. Všetky pamäťové priestory sú lineárne a pravidelné.




Obr. 36 Bloková schéma ATmega328

## 6.1. *In-System Flash* programová pamäť

ATmega328P obsahuje integrovanú ISP re programovateľnú pamäť Flash s kapacitou 32KByte pre uloženie programu (Obr. 25). Keďže všetky inštrukcie AVR majú šírku 16 alebo 32 bitov, Flash pamäť je organizovaná ako 16KByte x 16bitov. Pamäťový priestor pamäte Flash je rozdelený na dve časti: sekciu *Boot Loader* a Aplikačný program. Pamäť Flash má životnosť 10 000 cyklov zápisu/vymazania.





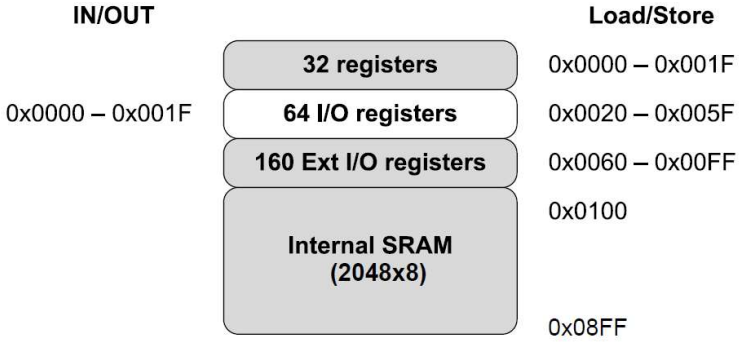
Programové počítadlo (*Program Counter*) ATmega328P má dĺžku 14 bitov, čo umožňuje adresovať 16K programových slov.

## 6.2. SRAM dátová pamäť

Mikrokontrolér s viacerými periférnymi jednotkami, než je možné podporovať v rámci 64 miest vyhradených v jazyku symbolických inštrukcií (*assembler*) pre inštrukcie IN a OUT. Pre rozšírený I/O priestor od 0x60 - 0xFF v SRAM možno použiť iba inštrukcie *Load* a *Store*.

SRAM pamäť dát v ATmega328 má veľkosť 2304 bytov, ktoré sú rozdelené nasledovne (Obr. 37):

- 32 registrov, ktoré využíva procesor pre prácu s ALU (0x0000-0x001F),
- 64 štandardných vstupno-výstupných registrov pre riadenie periférií (0x0000-0x003F), ku ktorým je možné pristupovať aj cez I/O príkazov IN a OUT v jazyku symbolických inštrukcií,
- 160 rozšírených vstupno-výstupných registrov pre riadenie rozšírených periférií (0x0060-0x00FF),
- 2KB (2048 bajtov) SRAM dátovej pamäte (0x0100-0x08FF).




Obr. 37 Adresový priestor dátovej pamäte ATmega328P

Procesor pristupuje k dátovej pamäti pomocou piatich režimov adresovania, ktoré sa používajú pri programovaný v jazyku symbolických inštrukcií:

- Priame adresovanie celého dátového priestoru,
- Nepriame relatívne adresovanie s posunutím maximálne o 63 adries od základnej adresy v registroch Y alebo Z,
- Nepriame s využitím registrov R26 až R31,
- Nepriame s využitím registrov R26 až R31 s predchádzajúcim dekrementovaním adresy,
- Nepriame s využitím registrov R26 až R31 s následným inkrementovaním adresy.

### 6.3. Integrovaná EEPROM pamäť dát

ATmega328 obsahuje integrovanú 1KB pamäť dát typu EEPROM. Je organizovaná ako samostatný dátový priestor, do ktorého je možné zapisovať aj z neho čítať jednotlivé bajty. EEPROM má životnosť najmenej 100 000 cyklov zápisu/vymazania. Procesor pracuje s pamäťou EEPROM ako s externou perifériou a preto sa komunikácia s touto pamäťou riadi pomocou jej riadiacich registrov:

- Registre adresy bunky v pamäti EEPROM EEARH (0x42), EEARL (0x41),
- Register dát EEDR (0x40) obsahuje dátový bajt, ktorý sa má zapísať alebo vyčítať z EEPROM,
- Riadiaci register EECR (0x3F).

Pomocou bitov EECR.EEPM[1:0] je možné zvoliť mód práce EEPROM pamäte a tým aj čas operácie s pamäťou (Tab. 10). Podľa tabuľky je čas potrebný na zapísanie dát do celej pamäte EEPROM rovný 1024 * 3,4ms = 3,48s. EEPROM pamäť má kapacitu 1024 bajtov, preto sa adresa skladá z dvoch registrov EEARH a EEARL, pričom z registra EEARH sú použité len prvé dva bity.

Tab. 10 Možnosti pracovného módu EEPROM pamäte

| EEPM[1:0] | Programming Time | Operation |
|---|---|---|
| 00 | 3.4ms | Erase and Write in one operation (Atomic Operation) |
| 01 | 1.8ms | Erase Only |
| 10 | 1.8ms | Write Only |
| 11 | - | Reserved for future use |

- Príklad funkcie na jednoduchý zápis do pamäte

```
//*********************************************************************
void EEPROM_write_byte (unsigned int Address, unsigned char Data)
{
  while (EECR & (1<<EEPE));    //Wait for completion of previous write
  EEAR = Address;              //Set up Address
  EEDR = Data;                 //Set up Data byte
  EECR |= (1<<EEMPE);          //Enable Master write
  EECR |= (1<<EEPE);           //Start eeprom write
}
```

- Príklad funkcie na jednoduché čítanie z pamäte

```
//*********************************************************************
unsigned char EEPROM_read_byte (unsigned int Address)
{
  while (EECR & (1<<EEPE));    //Wait for completion of previous write
  EEAR = Address;              //Set up Address
  EECR |= (1<<EERE);           //Start eeprom write
  return (EEDR);               //Return data
}
```





## 7. Generovanie hodinového signálu

Generátor hodinového signálu je neoddeliteľnou súčasťou každého mikroprocesora. ATmega328P môže generovať hodinový signál z interného alebo externého zdroja. Od hodnoty tejto frekvencie je závislá rýchlosť spracovania inštrukcií strojového kódu a taktiež komunikácie na internej zbernici. Čím bude väčšia frekvencia komunikácie na zbernici, tým rýchlejšie bude procesor vykonávať inštrukcie a tým bude vyšší výpočtový výkon. Čím je ale vyššia pracovná frekvencia procesoru a jeho periférií, tým väčšiu spotrebu elektrickej energie tieto obvody majú.

Na obrázku (Obr. 38) je znázornený systém generovania hodinových signálov a ich prepojenie s jednotlivými perifériami. Všetky hodinové signály nemusia byť aktívne. Aby sa znížila spotreba energie, môžu byť niektoré hodinové signály neaktívne, čím budú odpovedajúce periférie zastavené a uvedené do niektorého z režimov spánku. Frekvencia systémového hodinového signálu **$clk_{SYS}$** je frekvencia generovaná systémovým deličom *System Clock Prescaler*. Všetky hodinové signály generované riadiacou jednotkou *AVR Clock* majú rovnakú frekvenciu.

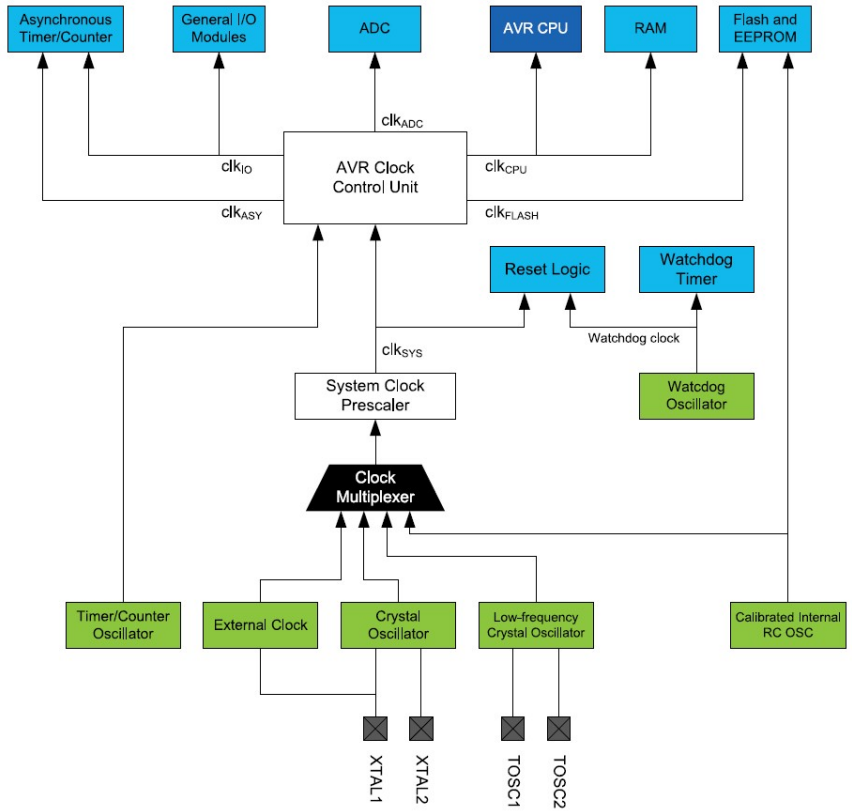
Hodinový signál CPU – **$clk_{CPU}$** je smerovaný do častí systému, ktoré zabezpečujú chod jadra AVR procesora, teda 32 pracovných registrov, register stavu (*Status register*) a dátová pamäť SRAM, v ktorej je uložený ukazovateľ zásobníka (*Stack Pointer*). Zastavením signálu **$clk_{CPU}$** sa prestanú vykonávať všeobecné operácie a výpočty.

Hodinový signál - **$clk_{IO}$** sa používa pre vstupno-výstupné moduly, ako časovače, počítadlá, SPI a USART a používa ho aj modul externého prerušenia.

Hodinový signál Flash – **$clk_{FLASH}$** používajú rozhrania Flash a EEPROM pamäte. Tento hodinový signál je zvyčajne aktívny súčasne s **$clk_{CPU}$**.

Hodinový signál asynchrónnych časovačov – **$clk_{ASY}$** umožňuje asynchrónny časovač-počítadlo taktovať priamo z externého hodinového signálu alebo z externého 32kHz kryštálu. Toto riešenie umožňuje používať časovač-počítadlo ako počítadlo reálneho času, aj keď je zariadenie v režime spánku.

Hodinový signál ADC – **$clk_{ADC}$** je určený iba pre ADC prevodník. To umožňuje zastaviť hodiny **$clk_{CPU}$** a **$clk_{IO}$** , aby sa znížil šum generovaný digitálnymi obvodmi. Týmto sa zabezpečia presnejšie výsledky ADC prevodníka.




Obr. 38 Zdroje a rozvod hodinového signálu

## 7.1. Zdroje hodinového signálu

ATmega328P má nasledujúce možnosti zdroja hodinových signálov, ktoré je možné voliť pomocou nastavovacích bitov (*Fuse bits*):

- *Low Power Crystal Oscilator* – je oscilátor s nízkym príkonom a zníženým kolísaním napätia na výstupe XTAL2. Poskytuje najnižšiu spotrebu energie, ale nie je schopný ovládať iné hodinové vstupy a môže byť viac citlivý na rušenie. Môže pracovať v troch rôznych režimoch, každý optimalizovaný pre určitý frekvenčný rozsah.





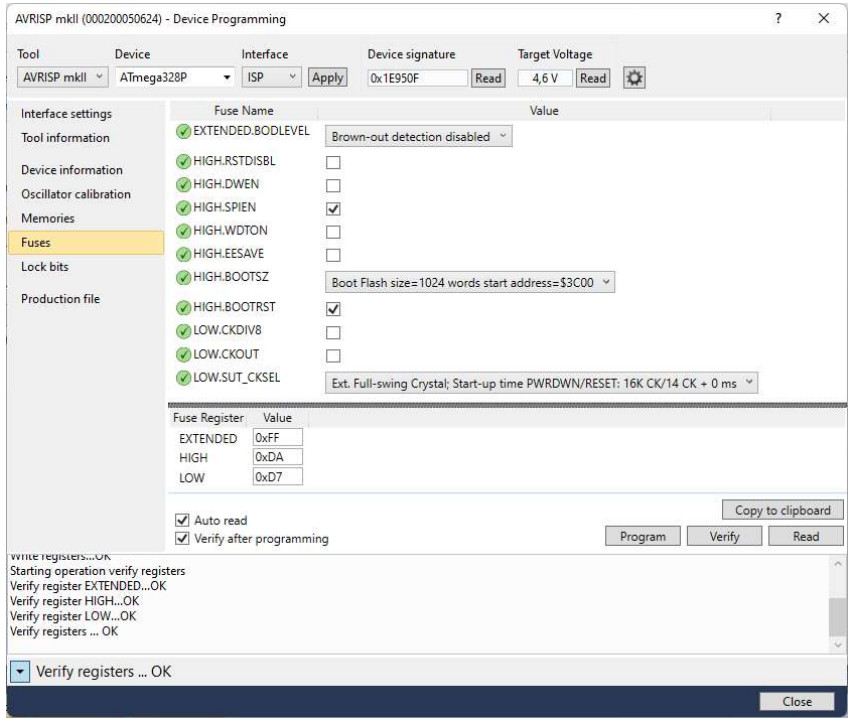
- *Full Swing Crystal Oscilator* – oscilátor je plnohodnotný oscilátor výstupe XTAL2. Je vhodný pre generovanie hodinových signálov v aplikáciách s možným rušením. Spotreba prúdu je vyššia ako pri nízko výkonnom oscilátore. Tento oscilátor bude pracovať len s VCC = 2,7-5,5V.
- *Low Frequency Crystal Oscilator* – Nízkofrekvenčný kryštálový oscilátor je optimalizovaný pre hodinový kryštál s frekvenciou 32,768 kHz. Oscilátor je optimalizovaný pre veľmi nízku spotrebu energie.
- *Internal 128 KHz RC Oscilator* - 128 kHz interný oscilátor je oscilátor s nízkym výkonom s hodinami 128 kHz. Frekvencia je nominálna pri 3V a 25°C. 128 kHz oscilátor nemá vysokú presnosť frekvencie.
- *Calibrated Internal RC Oscilator* – mikrokontrolér Atmega328 má tento 8,0 MHz RC oscilátor od výroby nastavený ako zdroj hodinového signálu s deliacim pomerom 8, čím sa vytvorí frekvencia hodinového signálu 1 MHz. Napriek tomu, že jeho frekvencia závisí na napätí a teplote, môže byť hodinový signál užívateľom presne kalibrovaný. Tento zdroj hodinového signálu nepotrebuje žiadne externé komponenty.
- *External Clock* - pri použití externého hodinového signálu je potrebné vyhnúť sa náhlym zmenám vo frekvencii, aby sa zabezpečila stabilná prevádzka MCU. Zmena frekvencie o hodnotu viac ako 2% môže viesť k nepredvídateľnému správaniu MCU. Ak je potrebné zmeniť frekvenciu o viac ako 2%, treba uviesť MCU do stavu Reset počas tejto zmeny.
- *Timer/Counter Oscilator* - Tento zdroj hodín pre časovač/počítadlo možno použiť len vtedy, keď je ako zdroj systémových hodín vybratý kalibrovaný interný RC oscilátor. Pre tento zdroj sú vývody oscilátora časovača/počítadla (TOSC1 a TOSC2) zdieľané s XTAL1 a XTAL2. Ak sa má použiť oscilátor časovača/počítadla, systémové hodiny musia mať štvornásobok frekvencie tohto oscilátora.

Každý zdroj hodín potrebuje dostatočné napätie VCC na spustenie kmitania a minimálny počet oscilačných cyklov predtým, než je možné frekvenciu považovať za stabilnú. Aby sa zaistil nábeh napájacieho napätia na hodnotu VCC po pripojení napájacieho napätia je potrebné generovať oneskorenie, ktoré udržiava MCU v reštarte, kým nie je napájanie nedosiahne minimálnu hodnotu VCC. Od oscilátora sa požaduje, aby vykonal minimálny počet kmitov pred tým, ako sa hodiny považujú za stabilné. Vnútorné počítadlo zvlnenia napájania monitoruje výstupné hodiny oscilátora a udržuje vnútorný reset MCU aktívny počas daného počtu hodinových cyklov. Reset sa potom uvoľní a MCU začne pracovať. Odporúčaný čas spustenia oscilátora závisí od zdroja hodín a pohybuje sa od 6 cyklov pre externe aplikované hodiny po 32 000 cyklov pre nízkofrekvenčný kryštál.

### 7.2. *System Clock Prescaler*

MCU obsahuje deličku systémových hodín, ktorú je možné použiť na zníženie frekvencie systémových hodín a tým aj spotreby energie. Deliaci faktor môže byť: 1, 2, 4, **8**, 16, 32, 64, 128, 256. Deliaci faktor nastavený výrobcom je 8, teda taktovacia frekvencia je od výroby 8MHz/8 = 1MHz.

Na účely predmetu sa používa vývojová doska Arduino UNO, ktorá obsahuje externý 16MHz kryštál a deliaci pomer je 1 preto, je frekvencia $\mathbf{clk_{SYS}}$ = 16MHz. Na (Obr. 39) sú zobrazené *Fuse* bity ATmega328P na doske Arduino UNO.




Obr. 39 Nastavenie *Fuse* bitov ATmega328P na doske Arduino UNO

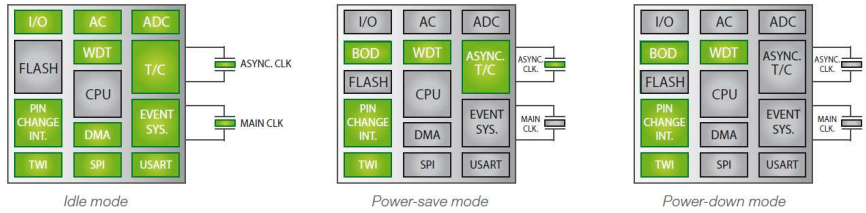




## 8. *Power* manažment

Správnou voľbou režimu činnosti AVR mikrokontroléra je možné upravovať spotrebu elektrickej energie. Zvolením vhodného režim spánku (*Sleep mode*) sa v MCU deaktivujú tie moduly, ktoré nie sú bezpodmienečne potrebné, čím sa šetrí energia. MCU poskytuje tieto režimy šetrenia energie:

- *Idle Mode* - režim nečinnosti zastaví procesor a súčasne povoľuje prácu s pamäťou SRAM, časovačmi, počítadlami, SPI, USART, I2C, analógovým komparátorom a systémom prerušenia. Tento režim spánku v podstate zastaví $clk_{CPU}$ a $clk_{FLASH}$, pričom necháva ostatné hodinové signály aktívne. Z režimu nečinnosti je možné MCU prebudiť pomocou externého a interného prerušenia, ako napríklad prerušenie časovača alebo USARTu.
- *ADC Noise Reduction Mode* - režim redukovania šumu pri ADC. Zastaví sa procesor a všetky vstupno-výstupné moduly okrem asynchrónneho časovača a ADC, aby sa minimalizoval šum, čo umožňuje realizovať AD prevod s vyšším rozlíšením. Tento režim spánku zastaví $clk_{IO}$, $clk_{CPU}$ a $clk_{FLASH}$, pričom ostatné hodinové signály sú funkčné.
- *Power-Down Mode* - režim vypnutia zablokuje všetky generátory hodinového signálu, čím pozastaví všetky zariadenia MCU okrem asynchrónnych modulov až do ďalšieho prerušenia alebo hardvérového resetovania.
- *Power-Save Mode* - režim šetrenia energie je totožný ako režim vypnutia s tým rozdielom, že ak je spustený počítač/časovač TC2, tak tento bude pokračovať.
- *Standby Mode* - pohotovostný režim je identický ako režim vypnutia s výnimkou, že generátor hodinového signálu je stále v prevádzke. Z pohotovostného režimu sa zariadenie prebudí v šiestich hodinových cykloch. To umožňuje veľmi rýchle spustenie v kombinácii s nízkou spotrebou energie.
- *Extended Standby Mode* - tento režim je identický s režimom šetrenia energie s výnimkou, že oscilátor je stále v prevádzke. Z rozšíreného pohotovostného režimu sa zariadenie prebudí v šiestich hodinových cykloch.

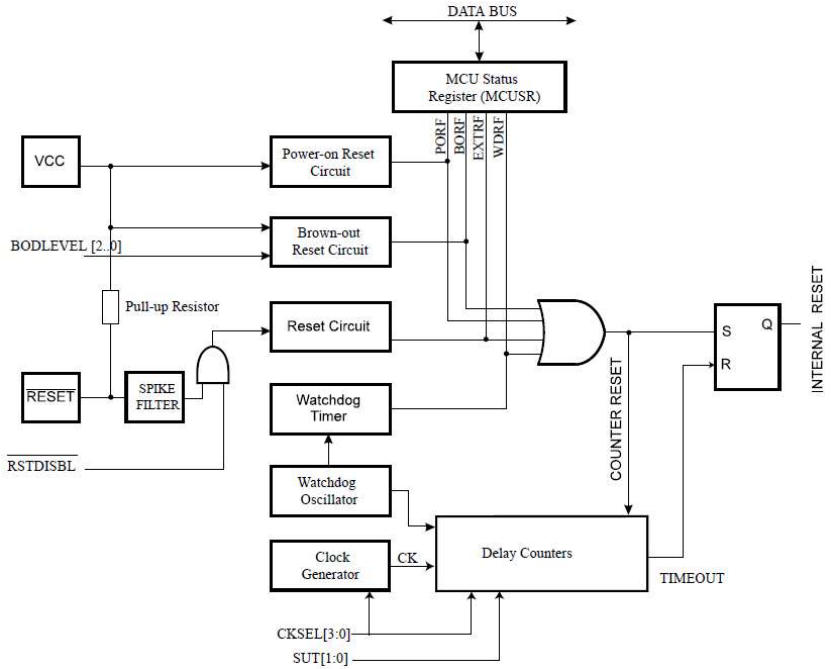



Obr. 40 Grafické znázornenie niektorých napájacích režimov MCU

Mikrokontrolér umožňuje znížiť spotrebu energie aj pomocou registra pre zníženie výkonu. V tomto registri je možné zastaviť jednotlivé hodinové signály a tým znížiť celkovú spotrebu MCU.

## 8.1. Systémový reset AVR

Počas resetovania sú všetky I/O registre nastavené na ich pôvodné hodnoty a program sa spustí z Reset vektora, ktorý má v tabuľke vektorov prerušenia najvyššiu prioritu. Inštrukcia umiestnená v Reset vektore musí byť inštrukcia absolútneho skoku (JMP) na obsluhu reset podprogramu. Potom, čo všetky zdroje resetovania prestanú byť aktívne, spustí sa počítadlo oneskorenia, ktoré predĺži interný reset mikrokontroléra. To riešenie umožňuje, aby výkon dosiahol stabilnú úroveň pred spustením normálnej prevádzky. Časový limit oneskorenia $\mathbf{t_{TOUT}}$ je definovaný užívateľom prostredníctvom poistiek SUT a CKSEL.




Obr. 41 Logika resetovacieho signálu ATmega328P

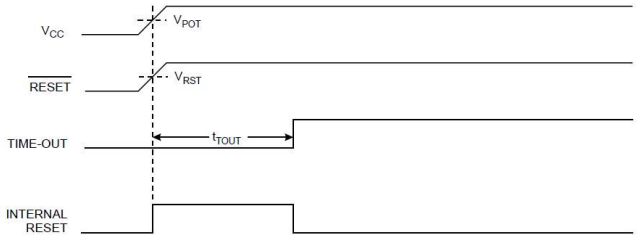
MCU AVR používa nasledujúce zdroje resetovania (obnovenia do počiatočného stavu) (Obr. 41):

- *Power-on Reset* – (POR) reset pri zapnutí napájania. MCU je v stave reset (nevykonáva činnosť), ak je napájacie napätie menšie ako hodnota $V_{POT}$. Reset signál je generovaný detekčným obvodom na čipe MCU vždy, keď je napätie VCC pod úrovňou detekcie (Obr. 42). Ak je



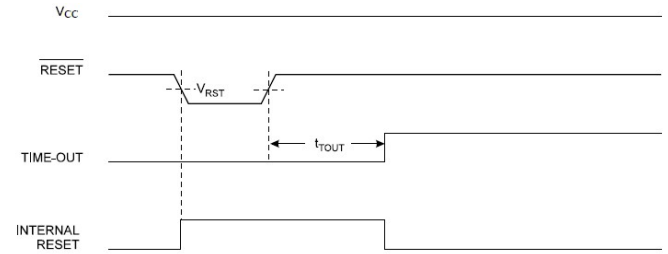


dosiahnuté prahového napätie, spustí sa počítadlo oneskorenia, ktoré určuje, ako dlho ($\mathbf{t_{TOUT}}$) sa zariadenie udržiava v režime interného resetu po dosiahnutí prahovej úrovne napätia VCC.




Obr. 42 Časový priebeh *Power-on Reset*

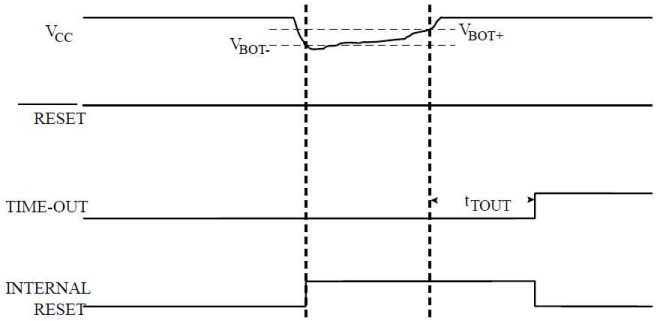
- *External Reset* - MCU sa resetuje, ak je na RESET vývode nastavená nízka úroveň, ktorá trvá dlhšia ako minimálna dĺžka impulzu. Ak je dosiahnuté prahového nábežnej hrany na RESET vývode, spustí sa počítadlo oneskorenia, ktoré určuje, ako dlho ($\mathbf{t_{TOUT}}$) sa zariadenie udržiava v režime interného resetu po dosiahnutí prahovej úrovne (Obr. 43). Externý reset môže byť zakázaný pomocou FUSE bitu RSTDISBL, potom už nie je možné resetovať MCU napr. externým tlačidlom.




Obr. 43 Časový priebeh *External Reset*

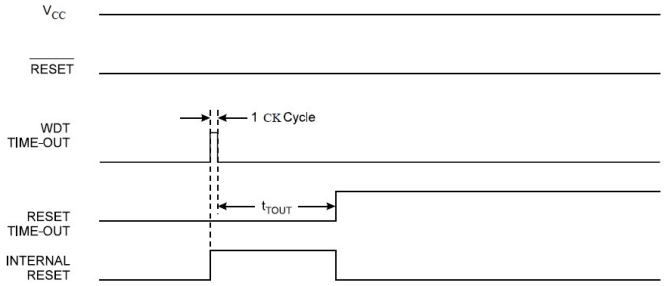
- *Brown-out Reset* - MCU sa resetuje, keď je napájacie napätie VCC menšie ako prahová hodnota vynulovania ($V_{BOT}$) a detektor BOD je zapnutý. MCU obsahuje na čipe obvod detekcie poklesu napájania (*Brown-Out Detection*). Ak je funkcia BOD zapnutá a hodnota VCC klesne pod hodnotu spúšťacej úrovne $V_{BOT}$-, *Brown-out Reset* sa okamžite aktivuje. Keď sa hodnota VCC

zvýši nad úroveň spúšťania $V_{BOT+}$ , spustí sa počítadlo oneskorenia a po uplynutí časového limitu $\mathbf{t_{TOUT}}$ MCU opúšťa interný reset stav (Obr. 44).




Obr. 44 Časový priebeh *Brown-out Reset*

- *WatchDog System Reset* - MCU sa resetuje, ak uplynie doba určená časovačom *Watchdog Timer* (WDT) a režim *Watchdog System Reset* je zapnutý. *WatchDog System Reset* resetuje MCU pri zlyhaní programu. K zlyhaniu systému môže dôjsť v dôsledku chyby v hardvéru alebo softvéru MCU. WDT vygeneruje prerušenie alebo systémový reset, keď počítadlo dosiahne nastavenú hodnotu počítadla (časového limitu). V normálnom prevádzkovom režime sa vyžaduje, aby sa v programe používala inštrukcia na nulovanie WDT počítadla pred uplynutím časového limitu. Ak MCU nereštartuje WDT, vygeneruje sa impulz, ktorý spustí prerušenie alebo interný reset MCU, ktorý sa ukončí po uplynutí časového limitu $\mathbf{t_{TOUT}}$ .



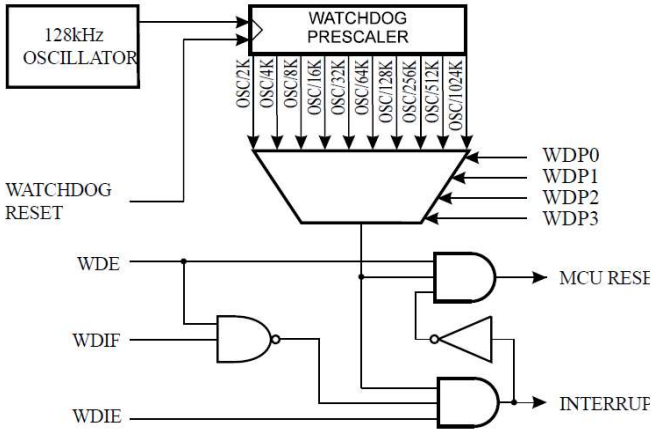

Obr. 45 Časový priebeh *WatchDog System Reset*





## 8.2. *Watchdog timer*

Pokiaľ sa *Watchdog timer* (WDT) v aplikácii nepoužíva, môže sa tento modul zakázať (pomocou *Fuse* bitu WDTON), čo by znamenalo zníženie spotreby energie. Ak je časovač sledovania povolený, je aktívny vo všetkých úsporných režimoch a preto bude vždy spotrebovávať energiu. V režimoch hlbšieho spánku to významne prispeje k celkovej spotrebe prúdu.




Obr. 46 Bloková schéma *WatchDog timer*

Tab. 11 Nastavenie *Watchdog Timer Prescaler*

| WDP[3] | WDP[2] | WDP[1] | WDP[0] | Number of WDT Oscillator (Cycles) | Oscillator |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 2K (2048) | 16ms |
| 0 | 0 | 0 | 1 | 4K (4096) | 32ms |
| 0 | 0 | 1 | 0 | 8K (8192) | 64ms |
| 0 | 0 | 1 | 1 | 16K (16384) | 0.125s |
| 0 | 1 | 0 | 0 | 32K (32768) | 0.25s |
| 0 | 1 | 0 | 1 | 64K (65536) | 0.5s |
| 0 | 1 | 1 | 0 | 128K (131072) | 1.0s |
| 0 | 1 | 1 | 1 | 256K (262144) | 2.0s |
| 1 | 0 | 0 | 0 | 512K (524288) | 4.0s |
| 1 | 0 | 0 | 1 | 1024K (1048576) | 8.0s |
| 1 | 0 | 1 | 0 | Reserved | |
| 1 | 0 | 1 | 1 | | |
| 1 | 1 | 0 | 0 | | |
| 1 | 1 | 0 | 1 | | |
| 1 | 1 | 1 | 0 | | |
| 1 | 1 | 1 | 1 | | |

WDT používa samostatný oscilátor s frekvenciou 128KHz (Obr. 46). Pomocou deličky je možné zvoliť 1 z 10 frekvencií oscilátora a tým nastaviť čas nulovania WDT počítadla. V normálnom prevádzkovom režime sa vyžaduje, aby sa programovo nulovalo WDT počítadlo pomocou inštrukcie v jazyku symbolických inštrukcií *watchdog reset* (WDR) pred dosiahnutím hodnoty časového limitu.

- Funkcia *WDT_Prescaler_Change()* nastavuje čas WDT pomocou bitov WDP3, WDP[2:0] podľa tabuľky Tab. 11.

```
//***************************************************************************
void WDT_Prescaler_Change(void)
{
   cli();                                  //Disable interrupt
   asm("wdr");                             //Assembler „watchdog_reset"
   WDTCSR |= (1<<WDCE)|(1<<WDE);           //Start timed equence
   WDTCSR = (1<<WDE)|(1<<WDP2)|(1<<WDP1);  //Set new time-out
   sei();                                  //Enable interrupt
}
```

- Funkcia *void WDT_off()* slúži na dočasné zakázanie WDT.

```
//***************************************************************************
void WDT_off (void)
{
   cli();                          //Disable interrupt
   asm("wdr");                     //Assembler „watchdog_reset"
   MCUSR &= ~(1<<WDRF);            //Clear WDRF in MCUSR
   WDTCSR |= (1<<WDCE)|(1<<WDE);   //Write logical one to WDCE and WDE
   WDTCSR = 0x00;                  //Turn off WDT
   sei();                          //Enable interrupt
}
```

Použitím knižnice <avr/wdt.h> je možné na nastavenie WDT použiť funkciu *wdt_enable(WDTO_500MS)* s parametrom WDT času a na nulovanie WDT požiť funkciu *wdt_reset()*.





# 9. Systém prerušení

ATmega328P disponuje niekoľkým zdrojmi prerušenia činnosti procesora (Tab. 12), ktoré môžeme rozdeliť na: **externé prerušenia** (vektor prerušenia 1 až 6) a **prerušenia od interných periférií**. Prerušenia majú vyhradené vektory prerušení (obsluhy prerušenia) v dátovej pamäti (v tabuľke vektorov prerušenia). Najnižšia adresa v tabuľke prerušení je štandardne vyhradená pre vektor prerušenia RESET. Priorita vykonávania prerušení je určená ich pozíciou v tabuľke prerušení, čím je adresa nižšia, tým vyššia je úroveň priority prerušenia.

Tab. 12 Tabuľka vektorov prerušení ATmega328

| VectorNo. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|
| 1 | 0x0000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset |
| 2 | 0x0002 | INT0 | External Interrupt Request 0 |
| 3 | 0x0004 | INT1 | External Interrupt Request 1 |
| 4 | 0x0006 | PCINT0 | Pin Change Interrupt Request 0 |
| 5 | 0x0008 | PCINT1 | Pin Change Interrupt Request 1 |
| 6 | 0x000A | PCINT2 | Pin Change Interrupt Request 2 |
| 7 | 0x000C | WDT | Watchdog Time-out Interrupt |
| 8 | 0x000E | TIMER2 COMPA | Timer/Counter2 Compare Match A |
| 9 | 0x0010 | TIMER2 COMPB | Timer/Counter2 Compare Match B |
| 10 | 0x0012 | TIMER2 OVF | Timer/Counter2 Overflow |
| 11 | 0x0014 | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 12 | 0x0016 | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 13 | 0x0018 | TIMER1 COMPB | Timer/Coutner1 Compare Match B |
| 14 | 0x001A | TIMER1 OVF | Timer/Counter1 Overflow |
| 15 | 0x001C | TIMER0 COMPA | Timer/Counter0 Compare Match A |
| 16 | 0x001E | TIMER0 COMPB | Timer/Counter0 Compare Match B |
| 17 | 0x0020 | TIMER0 OVF | Timer/Counter0 Overflow |
| 18 | 0x0022 | SPI, STC | SPI Serial Transfer Complete |
| 19 | 0x0024 | USART, RX | USART Rx Complete |
| 20 | 0x0026 | USART, UDRE | USART, Data Register Empty |
| 21 | 0x0028 | USART, TX | USART, Tx Complete |
| 22 | 0x002A | ADC | ADC Conversion Complete |
| 23 | 0x002C | EE READY | EEPROM Ready |
| 24 | 0x002E | ANALOG COMP | Analog Comparator |
| 25 | 0x0030 | TWI | 2-wire Serial Interface |
| 26 | 0x0032 | SPM READY | Store Program Memory Ready |

Každé prerušenie v tabuľke vektorov prerušení má priradený svoj povoľovací bit *(enable bit*), ktorým sa povoľuje alebo zakazuje jeho vykonávanie. Súčasne musí bit povolený aj bit I globálneho povolenia prerušení (*Global Interrupt Enable*), ktorý sa nachádza v status registri. Reakčná doba procesora na požiadavku prerušenia sú minimálne štyri hodinové takty. Po štyroch hodinových

taktoch sa vykoná skok na do programovej pamäte na adresu podľa tabuľky vektorov prerušení. Ak nastane požiadavka na obsluhu prerušeniu, keď je MCU v režime spánku, čas odozvy na prerušenia sa zvýši ešte o štyri hodinové takty, ktoré MCU potrebuje na „zobudenie“ z režimu spánku. Návrat z obsluhy prerušenia trvá tiež štyri hodinové cykly.

## 9.1. Externé prerušenia INT0, INT1

Externé prerušenia sa aktivujú pomocou vstupov INT0 alebo INT1 alebo akéhokoľvek vývodu PCINT. Prerušenie PCINT0 sa spustí, ak je aktívny ktorýkoľvek zo vstupov PCINT[7:0], podobne PCINT1 sa spustí, ak je aktívny ktorýkoľvek zo vstupov PCINT[14:8] a PCINT2 sa pustí pri aktivite ktoréhokoľvek vstupu PCINT[23:16]. V registroch PCMSK0, PCMSK1 a PCMSK2 je informácia, ktoré vstupy aktivovali prerušenie. Externé prerušenia môžu byť spustené zostupnou, vzostupnou hranou alebo nízkou úrovňou (LOW) alebo zmenou logickej úrovne signálu pripojeného na vstupy externého prerušenia. Ak sú externé prerušenia povolené a sú nakonfigurované na spúšťanie logickou úrovňou, prerušenia MCU budú trvať tak dlho, ako dlho je na vstupe logická úroveň LOW.

**Name:** EICRA *External Interrupt Control Register A*
**Offset:** 0x69
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | | | | | ISC11 | ISC10 | ISC01 | ISC00 |
| Access | | | | | R/W | R/W | R/W | R/W |
| Reset | | | | | 0 | 0 | 0 | 0 |

| Value | Description |
|---|---|
| 00 | The low level of INT1 generates an interrupt request. |
| 01 | Any logical change on INT1 generates an interrupt request. |
| 10 | The falling edge of INT1 generates an interrupt request. |
| 11 | The rising edge of INT1 generates an interrupt request. |

| Value | Description |
|---|---|
| 00 | The low level of INT0 generates an interrupt request. |
| 01 | Any logical change on INT0 generates an interrupt request. |
| 10 | The falling edge of INT0 generates an interrupt request. |
| 11 | The rising edge of INT0 generates an interrupt request. |

Obr. 47 Možnosti reagovania prerušenia INT0, INT1 na externý signál

- Príklad inicializácie externých prerušení. Pomocou bitov ISC11, ISC10, ISC01, ISC00 je nastavená zadná hrana externého signálu, definovanie HIGH na vstupoch INT0 a INT1 pomocou *Pull-up* rezistorov.

```
//*********************************************************************
void EXTINT_Init (void)
{
  EIMSK |= (1<<INT0) | (1<<INT1);       //Enable INT0, INT1
```





```
  EICRA |= (1<<ISC01) | (1<<ISC11);     //Trigger on failing edge
  EIFR  |= (1<<INTF0) | (1<<INTF1);
  PORTD |= (1<<PD2) | (1<<PD3);         //Pull-up on INT0, INT1
}
```

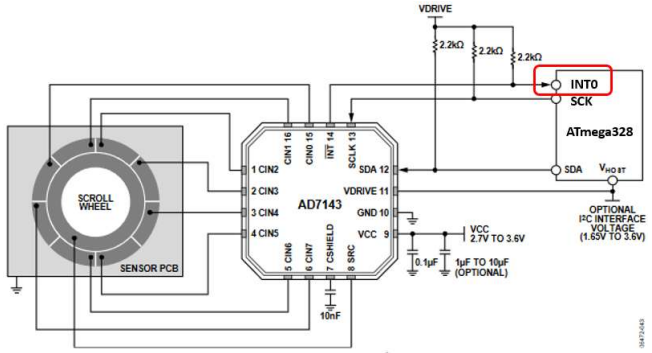
- Príklad obsluhy prerušenia INT0 nastavenie HIGH na vývode PB5

```
//*********************************************************************
ISR (INT0_vect)
{
  PORTB |= (1<<PB5);                    //HIGH, (Led On)
  EIFR |= 1<<INTF0;
}
```

- Príklad obsluhy prerušenia INT1 nastavenie LOW na vývode PB5

```
//*********************************************************************
ISR (INT1_vect)
{
  PORTB &= ~(1<<PB5);                   //LOW, (Led Off)
  EIFR |= 1<<INTF1;
}
```

Príklad použitia externého prerušenia:

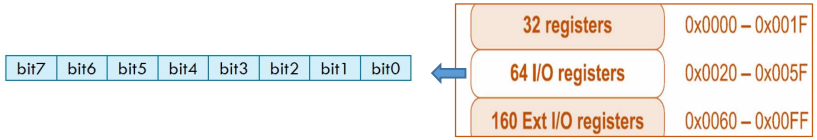



Obvodu AD7143 obsluhuje bezdotykovú kapacitnú klávesnicu pozostávajúcu z 8 senzorov. Po aktivácii ľubovoľného senzora vygeneruje AD7143 signál prerušenia, ktorý aktivuje (INT0) v MCU obsluhu tejto požiadavky. MCU okamžite reaguje na prerušenie a cez zbernicu I2C prečíta stav senzorov na klávesnici.

Použitím externého prerušenia nemusí MCU pravidelne v určitom časovom intervale komunikovať s AD7143 a zisťovať stav jednotlivých senzorov (Tento spôsob zisťovania stavu sa označuje *Polling*). Tento čas môže venovať inej činnosti.

# 10. I/O vývody

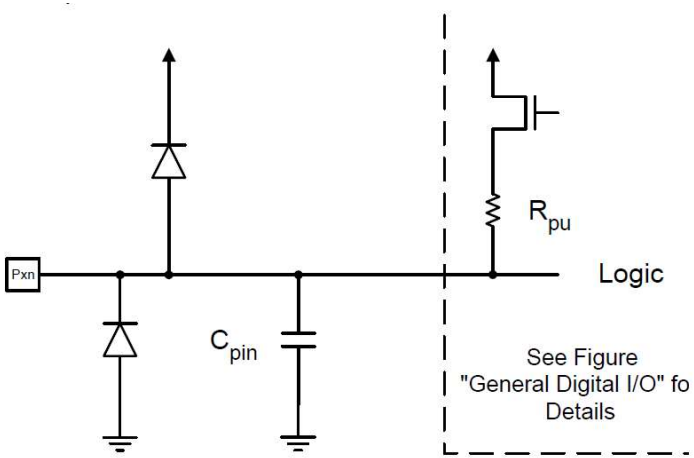
Často sa v problematike číslicovej elektroniky používajú pojmy **register** a **port**. Pod pojmom register rozumieme 8 bitové pamäťové miesto umiestnené na určitej adrese v SRAM pamäti. Používa sa ako rozhranie medzi „užívateľom-programátorom“ a elektronickým systémom (UART, SPI, ...). Register slúži ako elektronický prepínač, pomocou ktorého sa nastavujú vlastnosti zariadení podobne ako mechanické prepínače. Každý byt registra je určený na nastavenie určitej vlastnosti.




Obr. 48 Register ako elektronický prepínač zo súboru 64 I/O registrov

Port (brána) predstavuje rozhranie medzi MCU a okolitým svetom (externým signálom). Port je zoskupenie niekoľkých vstupno-výstupných vývodov MCU (väčšinou ôsmich) do jedného celku. Vstupno-výstupné porty sú jednou zo základných súčastí mikrokontrolérov.

MCU ATmega328 má k dispozícií 3 porty PB[7:0], PC[6:0], PD[7:0], spolu 23 vývodov, ktoré môžu pracovať v dvoch režimoch: vstupný alebo výstupný. Nastavenie smeru jednotlivých vývodov je nezávisle. Budiče jednotlivých vývodov sú dimenzované na priame napájanie LED displeja. Všetky vývody majú individuálne nastaviteľné *Pull-up* rezistory a každý vývod je chránený ochrannými diódami (Obr. 49).




Obr. 49 Schéma vstupno-výstupného vývodu





Pre správnu funkčnosť je nutné vstupno-výstupný port nakonfigurovať pomocou konfiguračných registrov. Každý port je možné nastaviť pomocou 3 konfiguračných registrov (Obr. 50):

- **PORTx** - dátový register výstupného portu. Nastavuje výstupnú hodnotu na vývodoch, ktoré sú nakonfigurované ako výstupy, alebo aktivuje/deaktivuje *Pull-up* rezistory na vývodoch, ktoré sú nakonfigurované ako vstupy,
- **DDRx** (*Data Direction Register*) - register smerovania dát, ktorý nastavuje smer vstupno/výstupných vývodov,
- **PINx** – (*Port Input Pins*) vstupný register, ktorý sa používa iba na čítanie logických hodnôt na vstupných vývodoch.

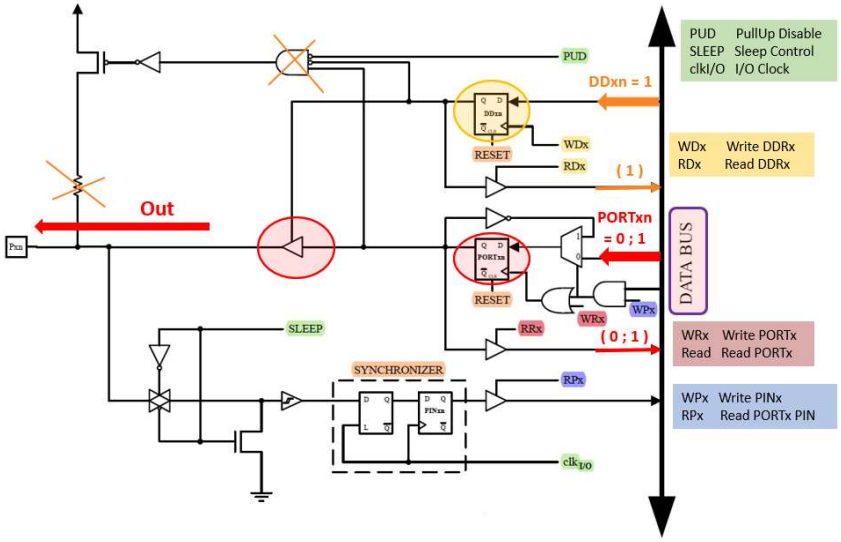
PINx je určený len na čítanie, PORTx a DDRx sú určené na čítanie aj zápis. Väčšina vývodov má alternatívne funkcie pre periférne funkcie zariadení v MCU. Povolenie alternatívnej funkcie niektorého z vývodov portu neovplyvní použitie ostatných vývodov v porte ako všeobecný digitálny I/O.


Obr. 50 Schematické znázornenie registrov I/O portov

Pre označovanie registrov sa používa všeobecné označenie PORTx, kde „x“ označuje B, C, D. Pre označenie bitov v jednotlivých portoch sa používa PORTxn, kde "x" označuje B, C, D a "n" označuje číslo bitu daného portu 0 - 7.

Každý I/O vývod Pxn je konfigurovaný pomocou troch bitov: DDxn, PORTxn a PINxn, ktoré prislúchajú jednotlivým portom: DDRx, PORTx a PINx.

Riadiaci obvod smeru I/O vývodu (žltá) zapína alebo vypína **trojstavový výstupný budič** (červená), ktorý prepája výstupný preklápací obvod (červená) **s** vývodom Pxn (Obr. 51). Ak je zvolený výstupný smer (DDxn=1), buffer nastaví logickú hodnotu bitu PORTXn (HIGH alebo LOW) na vývode Pxn. Zároveň je „odpojený“ *Pull-up* rezistor, pretože trojvstupové hradlo uzatvorí tranzistor, ktorý pripája k rezistoru napájacie napätie.




Obr. 51 Logika I/O vývodu v režime výstup (DDxn=1)

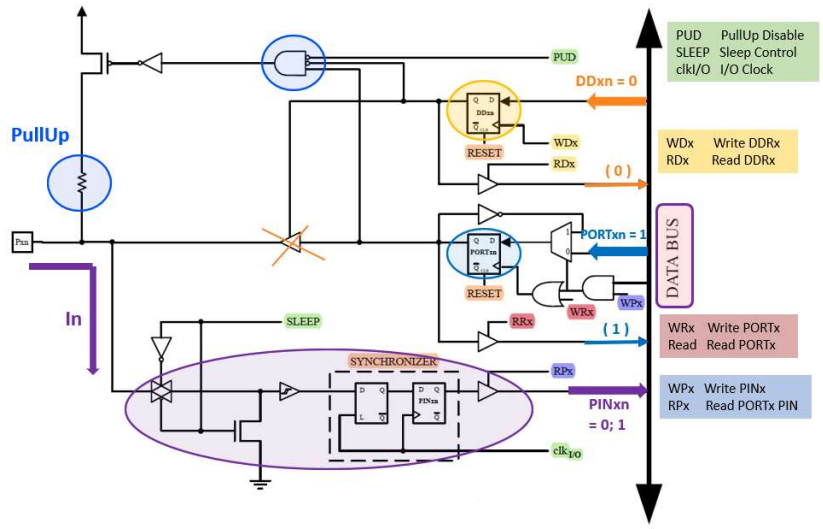
Ak DDxn bit definuje vývod ako vstupný (DDxn=0), výstupný budič je v stave vysokej impedancie čím je výstupný obvod (modrá) odpojený od vývodu Pxn (Obr. 52). Trojvstupové hradlo (modrá) aktivuje tranzistor, ktorý privedie na *Pull-up* rezistor napájacie napätie v prípade, že PORTxn=1 (modrá) , and PUD (*Pull-up disable*) nie je aktívny (LOW). Stav logickej úrovne na vývode Pxn je možné načítaním bitu PINxn zo vstupného obvodu (fialová) a bude vždy poskytovať okamžitú logickú hodnotu na vstupnom vývode.

Prípad na Obr. 53 je podobný ako na Obr. 52 s tým rozdielom, že *Pull-up* rezistor je „odpojený“ a nedefinuje na vstupe Pxn logickú úroveň HIGH. Ak tento vstup nebude pripojený k vonkajšiemu obvodu, tak bude mať nedefinovanú logickú hodnotu (bude „plávať“). V číslicovej elektronike je dobré sa takémuto nedefinovanému stavu vstupov logických obvodov vyhýbať, lebo môžu spôsobovať hazardné stavy.

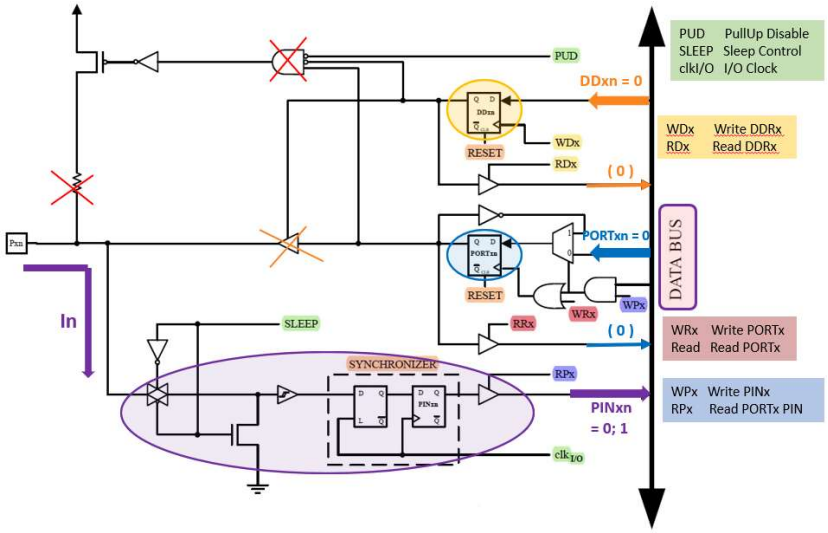
Ak sa niektoré vývody MCU nepoužívajú, je potrebné, aby na nich bola definovaná logická úroveň. Najjednoduchšia metóda na zabezpečenie definovanej úrovne je aktivovať vnútorné *Pull-up* rezistory. Pripojenie nepoužívaných vývodov priamo na VCC alebo GND sa neodporúča, pretože to môže spôsobiť nadmerný odber prúdu, ak je vývod náhodne nakonfigurovaný ako výstupný.








Obr. 52 Logika I/O vývodu v režime vstup so zapnutým *Pull-up* rezistorom (DDxn=0, PORTxn=1)




Obr. 53 Logika I/O vývodu v režime vstup bez *Pull-up* rezistora (DDxn=0, PORTxn=0)

Nezávisle od nastavenia bitu PORTxn je možné logický stav na vývode portu PINx prečítať cez bit registra PINxn. Ako je znázornené na (Obr. 52 a Obr. 53) stav I/O vývodu PXn v režime vstup prechádza cez D-preklápací obvod (fialová), ktorý môže obsahovať predchádzajúci stav a preto je potrebná synchronizácia pomocou jedného hodinového taktu. Z tohto dôvodu je potrebné do programu pre čítanie informácie zo vstupu vložiť inštrukciu NOP.

Bit DDxn z registra DDRx nastavuje režim vývodu:

- Ak je DDxn = 1, potom Pxn je nakonfigurovaný ako výstupný vývod,
- Ak je DDxn = 0, potom Pxn je nakonfigurovaný ako vstupný vývod.

Bit PORTxn nastavuje logickú hodnotu na výstupe Pxn:

- Ak je DDxn = 1 a súčasne PORTxn = 1, potom je na Pxn = 1,
- Ak je DDxn = 1 a súčasne PORTxn = 0, potom je na Pxn = 0.

Bit PORTxn aktivuje alebo deaktivuje *Pull-up* rezistor:

- Ak je DDxn = 0 a súčasne PORTxn = 1, potom je *Pull-up* rezistor aktivovaný,
- Ak je DDxn = 0 a súčasne PORTxn = 0, potom je *Pull-up* rezistor neaktívny,
- vypnutie *Pull-up* rezistora je možné urobiť aj nastavením DDxn = 1, teda Pxn je nastavený ako výstupný vývod.

Bit PINxn poskytuje logickú hodnotu na vstupe Pxn, ak je DDxn =0.

- Príklad nastavenia smeru vývodov PORTB pomocou DDRB. PB[7:4] vstupné vývody (po resete MCU), PB[3:0] výstupné vývody.

**Name:** DDRB
**Offset:** 0x24
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x04

| | Vstupy | | | | Výstup | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | DDRB7 | DDRB6 | DDRB5 | DDRB4 | DDRB3 | DDRB2 | DDRB1 | DDRB0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

```
//**************************************************************************
DDRB |= (1 << DDB3) | (1 << DDB2) | (1 << DDB1) | (1 << DDB0);
```

- Príklad nastavenia *Pull-up* rezistorov a logickej hodnoty na vývodoch PORTB. PB[7:6] *Pull-up* rezistory ON, PB[1:0] sú HIGH.





**Name:** PORTB
**Offset:** 0x25
**Reset:** 0x00
**Property:** When addressing as I/O Register: address offset is 0x05

| | *Pull-up* zapnuté | | *Pull- up* vypnuté | | Výstupy = LOW | | Výstupy = HIGH | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | PORTB7 | PORTB6 | PORTB5 | PORTB4 | PORTB3 | PORTB2 | PORTB1 | PORTB0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 1 | | | | | 1 | 1 |

```
//**************************************************************************
PORTB |= (1 << PB7) | (1 << PB6) | (1 << PB1) | (1 << PB0);
```

- Príklad načítania logických hodnôt na vstupoch PINB. PINB[7:2] sú LOW, PINB[1:0] sú HIGH, pretože PB[1:0] sú HIGH z predchádzajúceho príkladu.

**Name:** PINB
**Offset:** 0x23
**Reset:** N/A
**Property:** When addressing as I/O Register: address offset is 0x03

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 |
| Access | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| Reset | x | x | x | x | x | x | x | x |
| | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

```
//**************************************************************************
asm („nop“);          //Insert nop for synchronization
i = PINB;             //Read port pins
```

Vývojové prostredie *Microchip Studio* umožňuje použiť softvérový simulátor, v ktorom je možné odskúšať generovaný strojový kód ešte pred jeho nahraním do MCU. Obsahy jednotlivých registrov I/O portov je možné zobraziť aj v grafickom tvare.

| Name | Address | Value | Bits |
|---|---|---|---|
| I/O PINB | 0x23 | 0x03 | |
| I/O DDRB | 0x24 | 0x0F | |
| I/O PORTB | 0x25 | 0xC3 | |

Obr. 54 Registre portu PORTB, ich adresa v pamäti a hexa hodnota a stav jednotlivých bitov v simulátore *Microchip Studio*

# 11. Časovače a počítadla

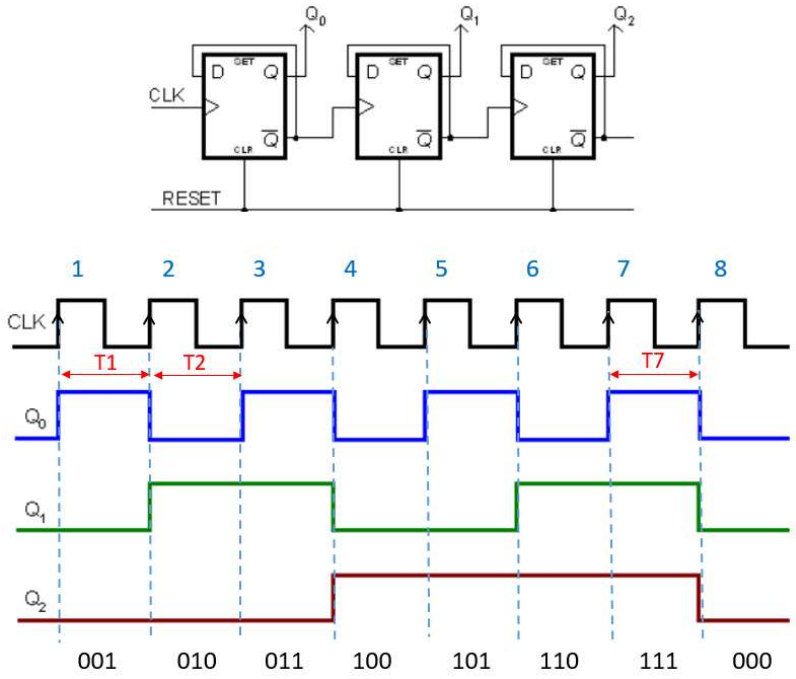
Časovače a počítadla tvoria dôležitú súčasť mikrokontroléra. Zjednodušene je možné povedať, že počítadlo predstavuje register, ku ktorému sa pripočíta jednotka na základe zistenia vzostupnej alebo zostupnej hrany signálu na vstupe počítadla. Počítadlo registruje zmenu signálu na vstupe (impulz), čo má za následok zvýšenie počítadla o jednotku.

Počítadlo = počet impulzov (n).

Časovač je počítadlo, ktoré počíta impulzy signálu (CLK) so známou frekvenciou (periódou), čo je možné použiť na vytvorenie časovej udalosti (určitého časového intervalu), generovanie PWM signálu. Ak sa počet impulzov počítadla vynásobí periódou impulzu dostaneme čas (Obr. 55).

Časovač = n * T, pričom T1 = T2 =....= Tn.

V ATmega328 sa nachádzajú tri samostatné časovače a počítadla.




Obr. 55 Trojbitové binárne počítadlo realizované pomocou D preklápacích obvodov



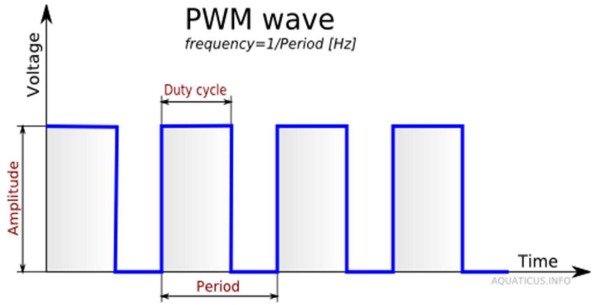


## 11.1. Impulzne šírková modulácia PWM

Impulzne šírková modulácia – PWM (*Pulse Width Modulation*) je diskrétna modulácia pre prenos analógového signálu pomocou dvojstavového signálu. PWM sa často využíva vo výkonovej elektronike za účelom riadenia veľkosti napätia alebo prúdu. Kombinácia PWM signálu s kondenzátorom sa často používa ako jednoduchá a lacná náhrada D/A prevodníka.

### 11.1.1. Perióda, strieda PWM signálu

Informácia v PWM signáli je prenášaná pomocou parametra striedy. U periodických signálov, ktoré počas periódy prechádzajú z jednej úrovne do druhej a naopak, znamená strieda (*Duty Cycle* – činiteľ plnenia) pomer časov, v ktorých je obdĺžnikový signál v jednotlivých úrovniach. Pokiaľ je strieda 1:1, znamená to, že obe úrovne signálu trvajú rovnako dlho. Pokiaľ je strieda uvedená v percentách, znamená to obvykle dobu trvania úrovne **H** voči celkovej perióde signálu. (0 % až 100 %, 50 % pre 1:1).



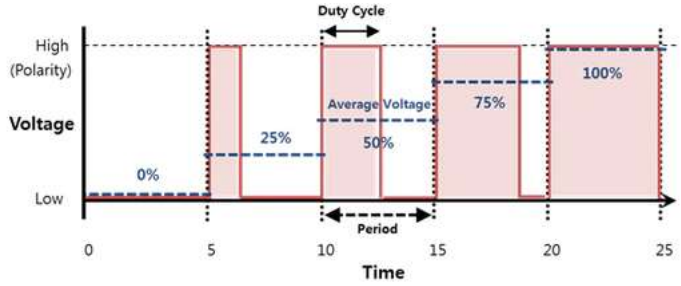

$$D = \frac{Duty}{Period}$$

$$DCL = \frac{Duty}{Period} * 100\%$$

Obr. 56 PWM pravouhlý periodický signál

### 11.1.2. Použitie impulzne šírkovej modulácie

Použitie PWM môžeme vidieť pri spínaní záťaže tranzistorom. Tranzistor má nízke straty v dvoch prevádzkových režimoch, kedy je naplno otvorený, alebo úplne zatvorený. Ak je polovodičový prechod nevodivý, je na ňom maximálne napätie a netečie ním žiadny prúd, takže v tomto stave je výkonová strata nulová. Ak je naplno otvorený, preteká ním prúd, pričom na polovodičovom prechode je saturačné napätie nízkej hodnoty, takže jeho stratový výkon je taktiež malý. Pretože pri každom zapnutí alebo vypnutí prechádza tranzistor cez oblasť vysokých strát, so stúpajúcou frekvenciou prepínania stavov výkonové straty rastú.




Obr. 57 Princíp generovania regulačného napätia PWM signálom

Skoro všetky DC/DC meniče (meniče jednosmerného napätia), meniče frekvencie alebo striedače využívajú formu impulzne šírkovej modulácie. PWM môže byť použitá na budenie jednosmerného elektromotoru (Obr. 57). Indukčnosť vinutia motoru sa potom správa ako nízkofrekvenčná priepusť, takže prúd a moment motoru je spojitý. Jedná sa prakticky o napäťové riadenie jednosmerného motoru.

## 11.2. TC0 - 8 bitový časovač / počítadlo s možnosťou PWM

TC0 je univerzálny 8-bitový časovač/počítadlo s dvoma nezávislými výstupnými porovnávacími jednotkami s podporou impulzne-šírkovej modulácie PWM. Umožňuje presné časovanie generovania priebehov a periodickú správu udalostí v čase (vykonávanie programu). TC0 má v SRAM pamäti vyhradené nasledujúce registre, ktoré potrebuje pre svoju činnosť:

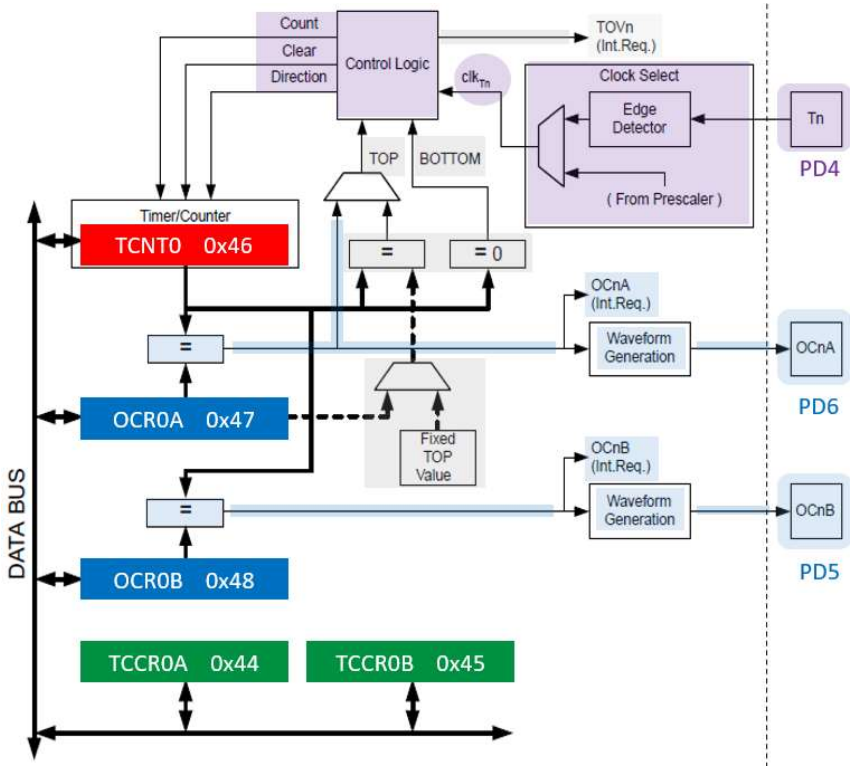
- TCCR0A (0x44), TCCR0B (0x45) – riadiace registre TC0,
- TIMSK0 (0x6E) – register maskovania prerušení generovaných TC0,
- TCTN0 (0x46) – register hodnoty počítania,
- OCR0A (0x47), OCR0B (0x48) – porovnávacie registre,
- TIFR0 (0x35) – register príznakov prerušení generovaných TC0.

Bloková schéma TC0 pozostáva z blokov:

- Riadenie počítania a zdroj hodinových impulzov $clk_{TC0}$. Buď interný zdroj $clk_{IO}$ alebo externý zdroj hodinových impulzov na vstupe PD4 nastavuje sa v TCCR0B.CS0[2:0] (Fialový blok). Počítanie impulzov môže byť smerom hore (inkrementovanie TCNT0) alebo smerom dole (dekrementovanie TCNT0). Riadiaca jednotka vyhodnocuje obsah TCNT0 a podľa zvoleného módu činnosti TC0 generuje prerušenie pretečenia počítadla 17 (TIMER0_OVF).
- TCNT0 - obojsmerné počítadlo impulzov $clk_{TC0}$. CPU môže kedykoľvek čítať obsah TCNT0 alebo hodnotu v TCNT0 zmeniť.








Obr. 58 Bloková schéma TC0 – 8 bitový časovač / počítadlo

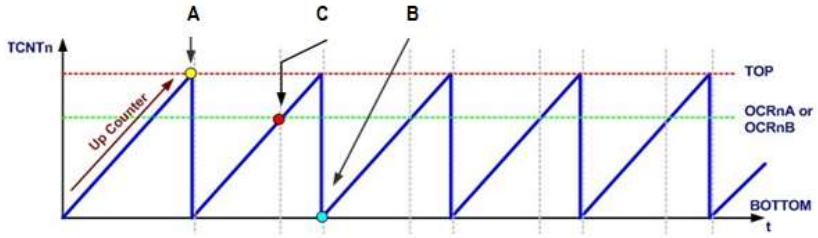
- Porovnávacia jednotka A porovnáva obsahy TCNT0 a OCR0A (*Output Compare Register*). Ak je zhoda, nastaví sa príznak TIFR0.OCF0A s možnosťou generovania prerušenia 15 (TIMER0 COMPA). Výsledok porovnávania umožňuje generovať časové impulzy s programovateľnou frekvenciou a striedou na vývode PD6.
- Porovnávacia jednotka B porovnáva obsahy TCNT0 a OCR0B. Ak je zhoda, nastaví sa príznak TIFR0.OCF0B s možnosťou generovania prerušenia 16 (TIMER0 COMPB). Výsledok porovnávania umožňuje generovať časové impulzy s programovateľnou frekvenciou a striedou na vývode PD5.
- Riadiace registre TCCR0A, TCCR0B, ktoré určujú režim činnosti TC0 a spravujú povolenia prerušení.

V jednotlivých módoch činnosti časovača-počítadla existujú hraničné stavy na základe, ktorých sa riadi jeho činnosť (Obr. 59):

- Stav počítadla **BOTTOM** - počítadlo dosiahne hodnotu nula (0x00).
- Stav počítadla **MAX** - počítadlo dosiahne hodnotu (0xFF).
- Stav počítadla **TOP** – počítadlo dosiahne tento stav, keď bude obsahovať najvyššiu definovanú hodnotu v postupnosti počítania. **TOP** hodnota môže byť **MAX**, alebo môže byť hodnota z intervalu (0x00 až 0xFF) uložená v registri OCR0A.

Počítanie môže byť v týchto smeroch:

A. Počítanie smerom **hore** pokiaľ obsah TCNT0 nie je rovný 0xFF (MAX),
B. Počítanie smerom **dole** pokiaľ obsah TCNT0 nie je rovný 0x00 (BOTTOM),
C. Počítanie smerom **hore** pokiaľ obsah TCNT0 nie je rovný hodnote **TOP**. Hodnota **TOP** môže byť rovná **MAX**, alebo je rovná hodnote uloženej v **OCR0A**.




Obr. 59 Stavy počítadla TCNT0 a smer počítania

Nastavením bitov v riadiacich registroch TCCR0B.WGM02, TCCR0A.WGM0[1:0]) sa vyberá režim činnosti počítadla-časovača a definuje sa správanie OC0A, OC0B a jednotky porovnávania výstupu (Tab. 13). K dispozícií sú módy 0 a 2, ktoré neumožňujú generovanie PWM signálu, len zmenu výstupu OC0A, OC0B. Ďalšie módy 1, 3, 5 a 7 sú režimy, ktoré umožňujú generovať PWM signál na vývodoch OC0A, OC0B. Funkciu vývodov OC0A a OC0B určujú bity TCCR0A.COM0A[1:0], TCCR0A.COM0B[1:0] v závislosti od zvoleného režimu (Tab. 14).

Tab. 13 Módy činnosti TC0

| Mode | WGM02 | WGM01 | WGM00 | Timer/Counter Mode of Operation | TOP | Update of OCR0x at | TOV Flag Set on[1)][2)] |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Normal | 0xFF | Immediate | MAX |
| 1 | 0 | 0 | 1 | PWM, Phase Correct | 0xFF | TOP | BOTTOM |
| 2 | 0 | 1 | 0 | CTC | OCRA | Immediate | MAX |
| 3 | 0 | 1 | 1 | Fast PWM | 0xFF | BOTTOM | MAX |
| 4 | 1 | 0 | 0 | Reserved | - | - | - |
| 5 | 1 | 0 | 1 | PWM, Phase Correct | OCRA | TOP | BOTTOM |
| 6 | 1 | 1 | 0 | Reserved | - | - | - |
| 7 | 1 | 1 | 1 | Fast PWM | OCRA | BOTTOM | TOP |





Bity TCCR0A.COM0A[1:0] určujú správanie výstupného vývodu OC0A Pre Normálny alebo CTC režim (režimy bez funkcie PWM) za predpokladu že, bit DDRD.DD6 určuje nastavenie vývodu PD6 ako výstupný a funkciu nad týmto vývodom preberá TC0. Jednotlivé možnosti správania sa vývodu OC0A (podobne OC0B) sú v Tab. 14.

Tab. 14 Režimy vývodu OC0A pre Normálny mód a CTC mód bez PWM

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | Toggle OC0A on Compare Match. |
| 1 | 0 | Clear OC0A on Compare Match. |
| 1 | 1 | Set OC0A on Compare Match . |

Módy TC0, ktoré podporujú generovanie PWM signálu umožňujú nastaviť správanie sa vývodu OC0A (OC0B) podľa tabuliek Tab. 15 a Tab. 16.

Tab. 15 Režimy vývodu OC0A pre módy 3, 7 *Fast* PWM

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | WGM02 = 0: Normal Port Operation, OC0A Disconnected<br>WGM02 = 1: Toggle OC0A on Compare Match |
| 1 | 0 | Clear OC0A on Compare Match, set OC0A at BOTTOM (non-inverting mode) |
| 1 | 1 | Set OC0A on Compare Match, clear OC0A at BOTTOM (inverting mode) |

Tab. 16 Režimy vývodu OC0A pre módy 1, 5 *Phase Correct* PWM

| COM0A1 | COM0A0 | Description |
|---|---|---|
| 0 | 0 | Normal port operation, OC0A disconnected. |
| 0 | 1 | WGM02 = 0: Normal Port Operation, OC0A Disconnected.<br>WGM02 = 1: Toggle OC0A on Compare Match. |
| 1 | 0 | Clear OC0A on Compare Match when up-counting. Set OC0A on Compare Match when down-counting. |
| 1 | 1 | Set OC0A on Compare Match when up-counting. Clear OC0A on Compare Match when down-counting. |

To isté platí aj pre vývod OC0B (TCCR0A.COM0B[1:0], DDR.DD5).

Generovanie signálov na výstupoch OC0A a OC0B je závislé od frekvencie taktovacích hodín TC0. K dispozícií je interný zdroj frekvencie s možnosťou jej zníženia deliacim pomerom alebo externý

zdroj (Tab. 17). Podľa tabuľky existuje možnosť zastavenia činnosti TC0, ak sa nevyberie žiadny zdroj hodinových impulzov.

Tab. 17 Výber zdroja hodinových impulzov

| CA02 | CA01 | CS00 | 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 | clkI/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. |

### 11.2.1. TC0 mód 0 - Normálny režim

Je najjednoduchší režim prevádzky počítadla TC0. V tomto režime je smer počítania vždy smer nahor. Po dosiahnutí maximálneho stavu počítadla TCNT0 = 0xFF, počítanie automaticky pokračuje znova od hodnoty 0x00. V normálnom režime sa nastavuje indikátor pretečenia počítadla TIFR0.TOV0 v tom istom časovom cykle, v ktorom bude hodnota TCNT0 vynuluje. V tomto prípade sa príznak TIFR0.TOV0 správa ako deviaty bit, ktorý sa iba nastavuje ale nenuluje. Obsluha prerušenia pretečenia počítadla automaticky nuluje príznak TIFR0.TOV0 a môže sa softvérovo zvýšiť rozlíšenie počítadla z 8 na 9 bitov. Frekvencia generovaného priebehu v normálnom režime sa vypočíta podľa vzťahu:

$$f_{TC0} = \frac{f_{clk\_IO}}{N*(256-TCNT0)}, \quad \text{kde N je hodnota deliaceho faktora (1, 8, 64, 256, 1024)} \qquad (3.1)$$

- Príklad 4ms prerušenie TOV0.

$f_{clk\_IO}$ = 16MHz, N = 256, perióda $T_{TOV0}$ = 4ms -> $f_{TOV0}$ = 250Hz.

$$TCNT0 = 256 - \frac{16\,Mhz}{256*250\,Hz} = 6$$

Aby sa vygenerovalo prerušenie TOV0, musí TCNT0 obsahovať hodnotu 256. TCNT0 musí napočítať 250 impulzov, aby bol dodržaný čas 4ms. Preto je potrebné prednastaviť nastaviť TCNT0 na hodnotu 6.

- Príklad 1ms prerušenie TOV0.

$f_{clk\_IO}$ = 16MHz, N = 256, perióda $T_{TOV}$ = 1ms -> $f_{TOV0}$ = 1000Hz.

$$TCNT0 = 256 - \frac{16\,Mhz}{256*1000\,Hz} = 193{,}5;$$ buď TCNT0 = 193, TCNT0 = 194.





Ak TCNT0 = 193, potom bude $f_{TOV0}$ = 992,06Hz -> $T_{TOV0}$ = 1,008ms.
Ak TCNT0 = 194, potom bude $f_{TOV0}$ = 1008,06Hz -> $T_{TOV0}$ = 0,992ms.

- Príklad 5ms prerušenie TOV0.

$f_{clk\_IO}$ = 16MHz, N = 256, perióda $T_{TOV}$ = 5ms -> $f_{TOV0}$ = 200Hz.

$$TCNT0 = 256 - \frac{16\,Mhz}{256*200\,Hz} = -56{,}5$$ -> N = 1025

$$TCNT0 = 256 - \frac{16\,Mhz}{1024*200\,Hz} = 177{,}775$$ -> TCNT0 = 178

Ak TCNT0 = 178, potom bude $f_{TOV0}$ = 200,32Hz -> $T_{TOV0}$ = 4,992ms.

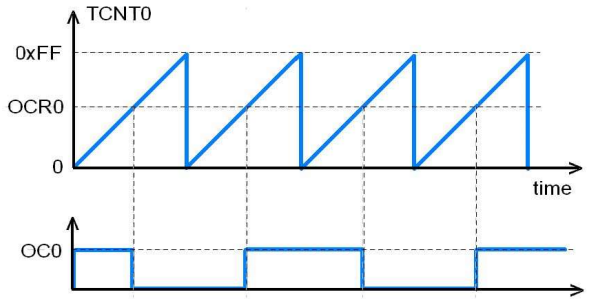
- Príklad inicializácie TC0 v normálnom móde TOV0 = 4ms a generovanie 125Hz signálu na výstupe OC0A

```
//**************************************************************************
TCCR0B |= (1<<CS02);                //Set prescaler 256
TIFR0  |= (1<<TOV0);                //Clear TOV0 flag
TIMSK0 |= (1<<TOIE0);               //Enable Timer0 overflow interrupt
TCNT0 = 6;                          //Set TCNT0 on 4ms
TCCR0A |= (1<<COM0A0);              //Toggle OC0A on Compare Match
DDRD |= (1<<DDD6);                  //Enable OC0A (PD6) Output
OCR0A = 125;
```

- Príklad obsluhy prerušenia TOV0. Led 100ms ON / 900ms OFF

```
//**************************************************************************
ISR (TIMER0_OVF_vect)
{
  ticled++;                                   //Led time = ticled * 4ms
  if (ticled < 25) PORTB |= (1<<PB5);                    //Led ON
  if (ticled > 25 && ticled < 250) PORTB &= ~(1<<PB5);   //Led OFF
  if (ticled > 250) ticled = 0;               //Clear Led time
  TCNT0 = 6;                                  //Set TCNT0 on 4ms
}
```

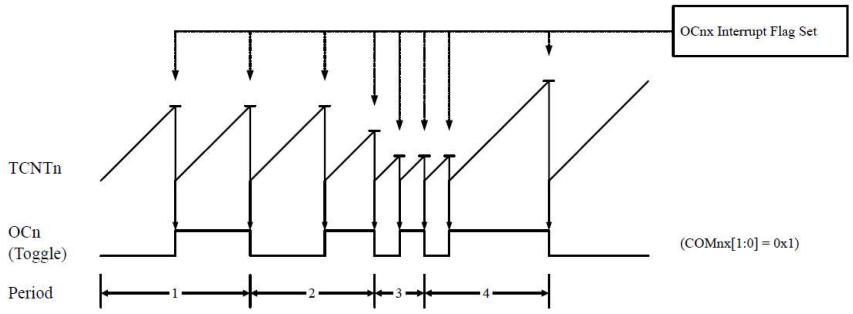
Priebeh generovaného výstupu v móde 0 na vývode OC0A je možné vidieť na (Obr. 60)Obr. 60 Priebeh signálu na vývode OC0A v normálnom móde TC0. Počítadlo TCNT0 počíta od 0x06, pri zhode TCNT0 a OCR0A dôjde k zmene stavu na výstupe OC0A, pričom počítanie v TCNT0 pokračuje, kým nedosiahne hodnotu 0xFF.




Obr. 60 Priebeh signálu na vývode OC0A v normálnom móde TC0

### 11.2.2. TC0 mód 2 – CTC (*Clear Timer on Compare Match*)

Rozdiel medzi Normálnym módom a CTC módom je, že pri zhode TCNT0 s obsahom registra OCR0A sa počítadlo TCNT0 vynuluje (Obr. 61). OCR0A definuje TOP hodnotu počítadla, ktorú môže pri počítaní dosiahnuť. Na obrázku (Obr. 61) je vidieť časový priebeh zmeny počítadla TCNT0 pokiaľ nie je zhoda TCNT0 a OCR0A (index n=0) s následným nulovaní TCNT0. Prerušenie je generované vždy, keď hodnota TCNT0 dosiahne hodnotu TOP nastavením príznaku TIFR0.OCFA. Ak je povolená obsluha tohto prerušenia, potom sa môže aktualizovať (meniť) hodnota OCR0A teda TOP. Ak je zhoda TCNT0 = OCR0A nastane zmena výstupného signálu na vývode 0C0A.




Obr. 61 Časový priebeh zmeny výstupu OCn v režime CTC časovača

Frekvencia generovaného priebehu v režime CTC na výstupe OC0A sa vypočíta podľa vzťahu:

$$f_{OC0A} = \frac{f_{clk\_IO}}{2*N*(1+OCR0A)}$$ , kde N je hodnota deliaceho faktora (1, 8, 64, 256, 1024) (3.2)

Maximálna frekvencia signálu na výstupe OCOA sa dosiahne, ak hodnota OCR0A = 0x00 a N = 1. Tak ako pri normálnom móde nastavuje sa príznak pretečenia počítadla TOV0, keď dochádza k zmene obsahuj počítadla TCNT0 z hodnoty MAX na 0x00.

- Príklad inicializácie TC0 v CTC móde, generovanie signálu na výstupe OC0A (Obr. 62)

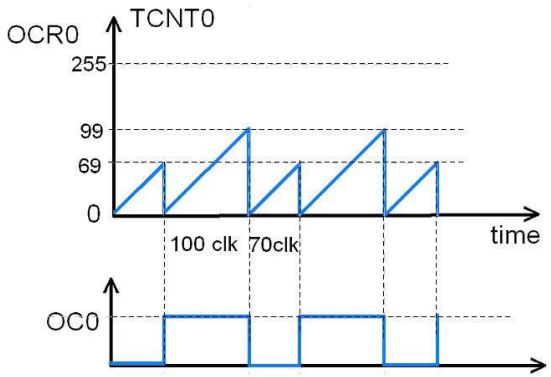
```
//**********************************************************************
TCCR0B |= (1<<CS02);               //Set prescaler 256
TCCR0A |= (1<<WGM01);              //Set CTC mode
TIMSK0 |= (1<<OCIEA0);             //Enable Output Compare Interrupt
TIFR0  |= (1<<OCFA0);              //Clear OCFA flag
TCCR0A |= (1<<COM0A0);             //Toggle OC0A on Compare Match
DDRD |= (1<<DDD6);                 //Enable OC0A (PD6) Output
OCR0A = 69;                        //Set 70  fOC0A=446,42Hz
```





- Príklad obsluhy prerušenia TOV0. Led 500ms ON/500ms OFF

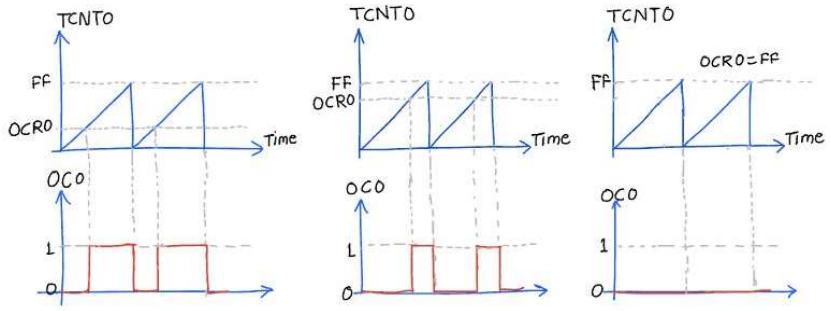
```
//**********************************************************************
ISR (TIMER0_COMPA_vect)
{
  if (OCR0A == 69) OCR0A = 99;      //Set 100 TOC0A1 = 1,6ms
  else OCR0A = 69;                  //Set 70  TOC0A2 = 1,12ms
}
```



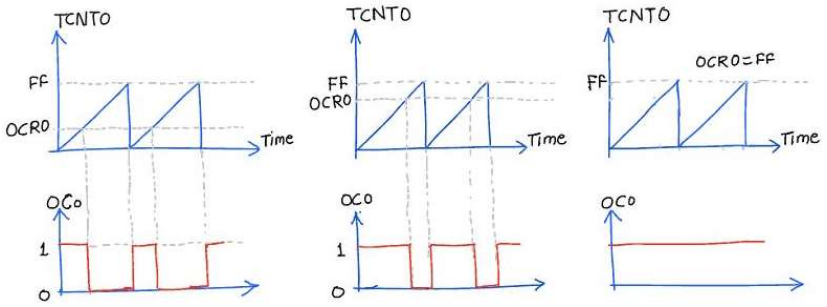

Obr. 62 Príklad priebehu signálu na vývode OC0A v CTC móde TC0

### 11.2.3. TC0 Mód 3, 7 – *Fast* PWM

Poskytuje možnosť generovania vysokofrekvenčného signálu. Režim rýchlej PWM sa líši od ostatných režimov PWM tým, že pracuje s „jedným sklonom“ (pílovitý tvar priebehu). Počítadlo počíta od BOTTOM po TOP a potom sa reštartuje. Na obrázkoch Obr. 63 a Obr. 64 sú názorné zobrazené princípy *Fast P*WM pre invertujúci a neinvertujúci PWM signál.

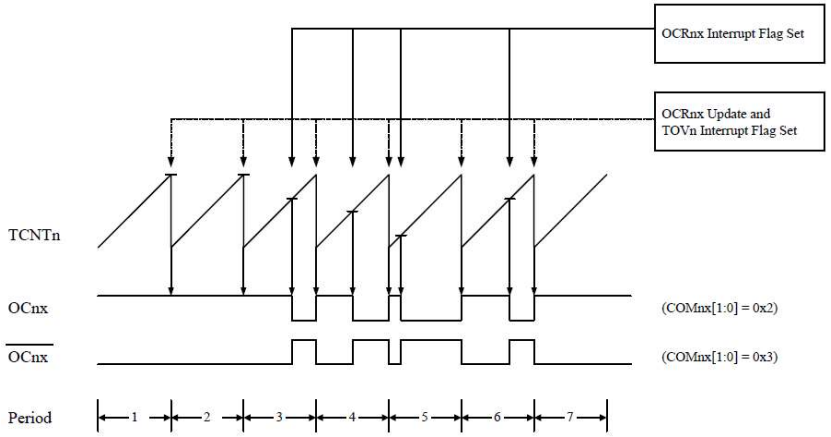



Obr. 63 Jednoduché znázornenie invertujúceho *Fast* PWM signálu




Obr. 64 Jednoduché znázornenie neinvertujúceho *Fast* PWM signálu

TOP hodnota môže byť definovaná ako 0xFF, alebo môže byť definovaná v OCR0A. Pretože sa pracuje s „jedným sklonom“ môže byť pracovná frekvencia režimu PWM 2x vyššia, ako je to pri fázovo korektnom režime PWM, ktorý využíva operáciu s dvojitým sklonom (trojuholníkový priebeh). V režime rýchlej PWM sa počítadlo TCNT0 zvyšuje dovtedy, pokiaľ sa hodnota počítadla nezhoduje s hodnotou TOP. Počítadlo sa potom v nasledujúcom cykle časovača vynuluje.




Obr. 65 Časový priebeh zmeny OCn v režime rýchlej PWM

Hodnoty TCNT0 sú v časovom diagrame znázornené ako vektory za účelom ilustrácie generovania hrán výstupného signálu na OC0A. Malé horizontálne značky na priebehu TCNT0 predstavujú zhodu medzi OCR0A a TCNT0. V čase ak TCNT0 = TOP nastavuje sa výstup OC0A na logická1, ak nastane





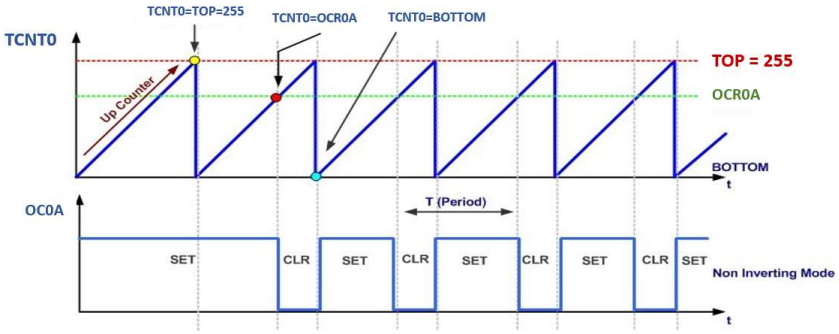
zhoda TCNT0 s OCR0A na výstupe OC0A sa nastaví LOW. Ak sa bude meniť hodnota OCR0A, bude sa meniť strieda generovaného PWM signálu na výstupe OC0A. Frekvenciu PWM výstupu OC0x môžeme vypočítať podľa nasledujúcej rovnice:

$$f_{OC0x\_PWM} = \frac{f_{clk\_IO}}{N*256}, \quad \text{kde N je hodnota deliaceho faktora (1, 8, 64, 256, 1024)} \qquad (3.3)$$

Príznak pretečenia počítadla TIFR0.TOV0 sa nastavuje vždy, keď počítadlo dosiahne hodnotu TOP. Ak je prerušenie povolené, v obsluhe prerušenia sa môže aktualizovať hodnota porovnávania OCR0x. Režim rýchlej PWM umožňuje generovanie PWM priebehov na vývodoch OC0x.

- Príklad inicializácie mód 3 (*Fast* PWM), generovanie PWM na OC0A, ($f_{OC0x\_PWM} = \frac{f_{clk\_IO}}{N*256} = \frac{16000000}{256*256} = 244{,}14Hz;\ T = \frac{1}{f_{OC0x_{PWM}}} = 4{,}096ms$) (Obr. 66)

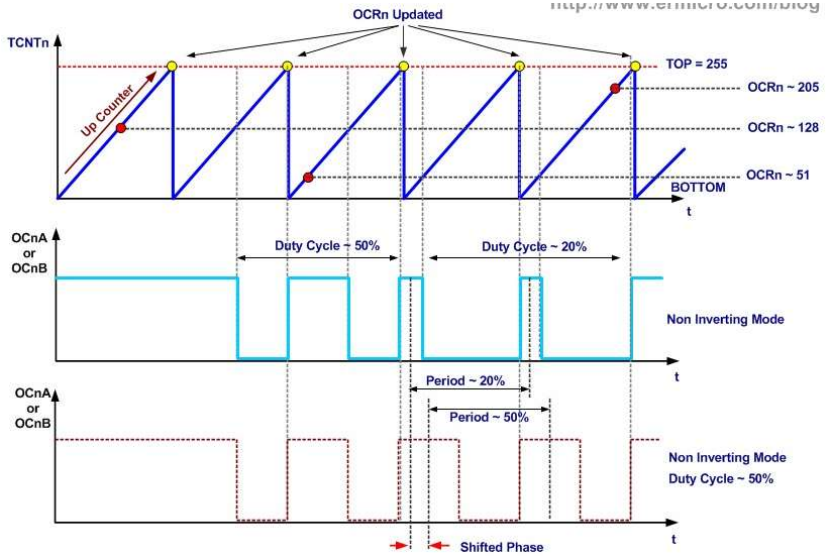
```
//**************************************************************************
TCCR0A |= (1<<WGM01)|(1<<WGM00);    //Set Fast PWM - mode3
TCCR0A |= (1<<COM0A1);              //Set Non Inverting Mode
TCCR0B |= (1<<CS02);                //Set prescaler 256 (Tpwm = 4,096ms)
TCNT0 = 0;
OCR0A = 170;                        //tSET = 170 * (4,096ms/256) = 2,72ms
```




Obr. 66 Príklad časového priebehu zmeny výstupu OC0A v režime *Fast* PWM

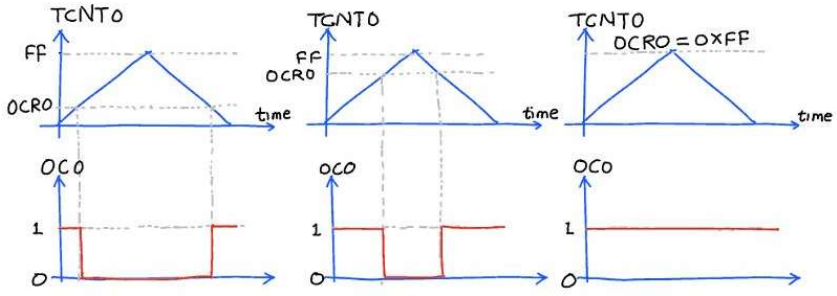
### 11.2.4. TC0 mód 1, 5 - PWM *Phase Correct*

Jednou z nevýhod používania režimu *Fast* PWM na generovanie PWM signálu je posunutie fázy PWM, vždy keď sa mení pracovný cyklus PWM. Je to preto, že keď register počítania TCNT0 dosiahne hodnotu TOP a začína počítať od BOTTOM, vždy nastaví výstupy (OC0A alebo OC0B) napriek hodnote výstupného porovnávacieho registra OCR0A a OCR0N (Obr. 67). Preto režim *Fast* PWM nie je vhodný pre presné riadenie otáčok motora. Lepšie je použiť režim *Phase Correct* PWM.




Obr. 67 Princíp vzniku posunutia fázy PWM signálu v režime *Fast* PWM pri zmene *Duty Cycle*

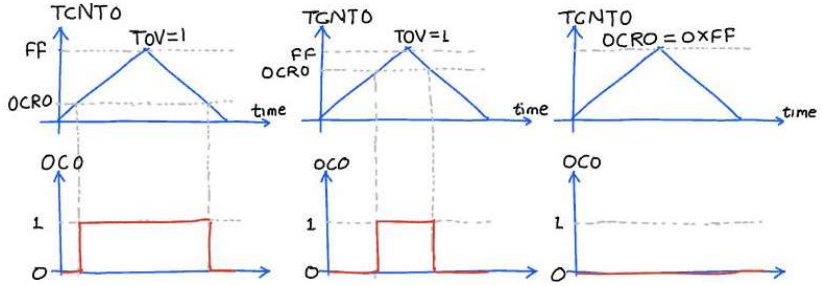
Základom tohto módu je počítanie s dvojitým sklonom z hodnoty BOTTOM k hodnote TOP a potom od TOP k BOTTOM, čím sa umožňuje generovanie priebehu s rozlíšením fázy PWM signálu. TOP hodnota môže byť definovaná ako 0xFF, alebo môže byť definovaná v OCR0A. Generovaný PWM signál má nižšiu maximálnu frekvenciu v porovnaní s *Fast* PWM. Vzhľadom na symetrickú vlastnosť režimov PWM s „dvojitým sklonom" sú tieto režimy uprednostňované pri aplikáciách riadenia motora. Počítadlo TCNT0 sa zvyšuje dovtedy, kým sa hodnota počítadla nezhoduje s hodnotou TOP. Keď počítadlo dosiahne hodnotu TOP, zmení sa smer počítania. Vždy keď TCNTO = BOTTOM, nastaví sa príznak prerušenia TOV0.




Obr. 68 Jednoduché znázornenie neinvertujúceho *Phase Correct* PWM signálu

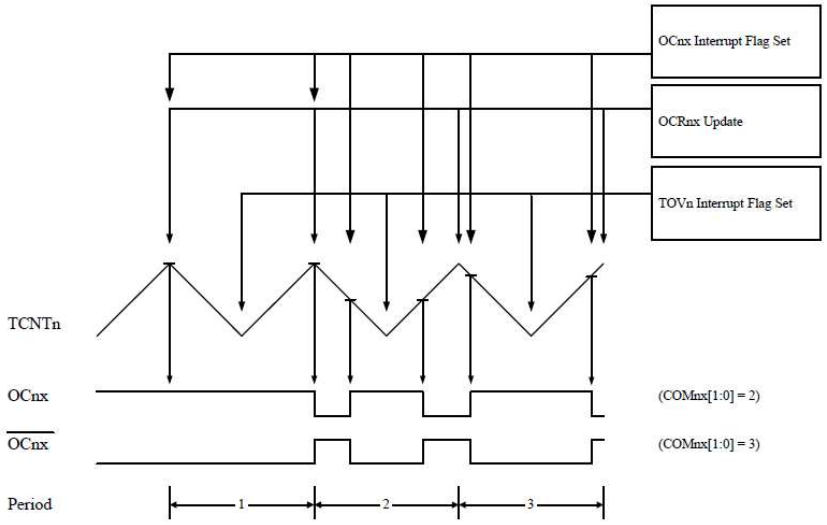







Obr. 69 Jednoduché znázornenie invertujúceho *Phase Correct* PWM signálu

Na obrázkoch Obr. 68 a Obr. 69 sú názorné zobrazené princípy *Phase Correct* PWM pre invertujúci a neinvertujúci PWM signál.



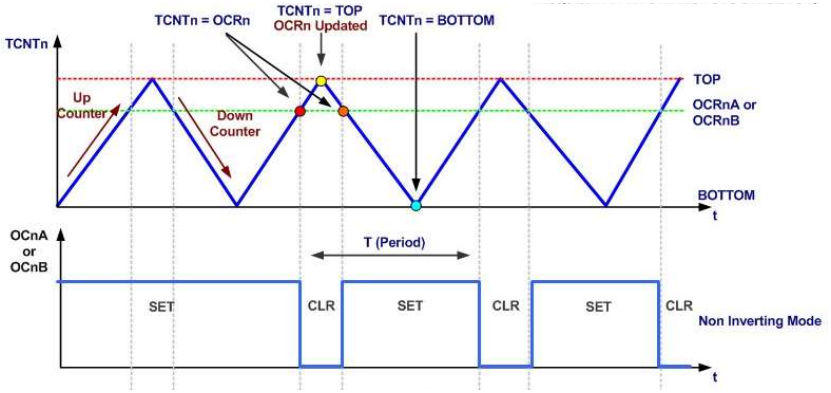

Obr. 70 Časový priebeh zmeny OCn v režime fázovo korektnej PWM

PWM signál na výstupe OC0A je generovaný nastavením LOW, ak platí TCNT0 = TOP, alebo pri počítaní smerom nahor platí TCNT0 = 0CR0A. PWM signál bude mať hodnotu HIGH, ak pri počítaní smerom nadol bude platiť TCNT0 = OCR0A. Frekvencia PWM signálu na výstupe OC0x môže byť vypočítaná podľa nasledujúcej rovnice:

$$f_{OCnxPCPWM} = \frac{f_{clk\_IO}}{N*510}$$ , kde N je hodnota deliaceho faktora (1, 8, 64, 256, 1024) (3.4)

- Príklad inicializácie mód 1 (*Phase Correct* PWM), generovanie PWM na OC0A, ($f_{OC0x\_PWM} = \frac{f_{clk\_IO}}{N*510} = \frac{16000000}{256*510} = 122{,}55Hz;\ T = \frac{1}{f_{OC0x_{PWM}}} = 8{,}16ms$) (Obr. 71)

```
//*********************************************************************
TCCR0A |= (1<<WGM00);               //Set Fast PWM - mode1
TCCR0A |= (1<<COM0A1);              //Set Non Inverting Mode
TCCR0B |= (1<<CS02);                //Set prescaler 256 (Tpwm = 8,16ms)
TCNT0 = 0;
OCR0A = 190;                        //tSET = 190*2*(8,16ms/510) = 6,08ms
```




Obr. 71 Príklad časového priebehu zmeny výstupu OC0A v režime *Phase Correct* PWM

## 11.3. TC1 – 16-bit časovač / počítadlo s možnosťou PWM

TC1 je univerzálny 16-bitový časovač – počítač umožňuje presné časovanie programu, generovanie signálov a meranie dĺžky impulzov. Obsahuje dve nezávislé výstupné porovnávacie jednotky s podporou PWM. TC1 má v SRAM pamäti vyhradené nasledujúce registre, ktoré potrebuje pre svoju činnosť:

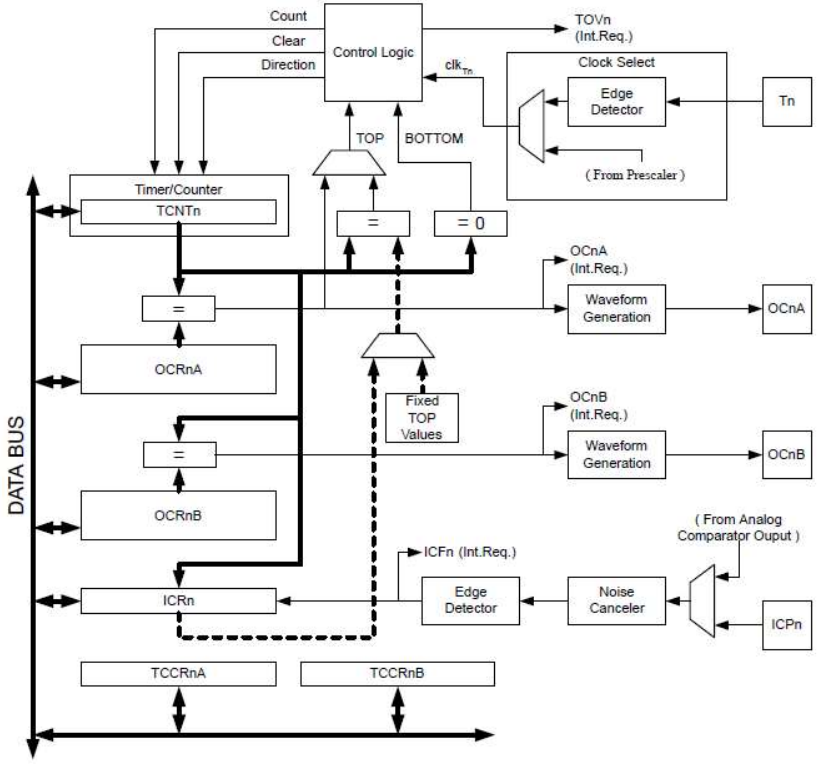
- TCCR1A (0x80), TCCR0B (0x81), TCCR1C (0x82) – 8 bitové riadiace registre TC1,
- TCTN1L (0x84), TCTN1H (0x85) – 16 bitový register hodnoty počítania, BOTTOM – TCNT1 = 0x0000, MAX – TCNT1 = 0xFFFF, TOP – hodnota TCNT1 sa rovná hodnote OCR1A,
- ICR1L (0x86), ICR1H (0x87) – 16 bitový register zmeny vstupného signálu,
- TIMSK1 (0x6F) – 8 bitový register maskovania prerušení generovaných TC1,
- OCR1AL (0x88), OCR1AH (0x98), OCR1BL (0x8A), OCR1BH (0x8B) – dva výstupné 16 bitové porovnávacie registre,
- TIFR1 (0x36) – 8 bitový register príznakov prerušení generovaných TC1.





Módy činnosti TC1 sú rovnaká ako pri TC0, rozdiel je len vtom, že TC1 pracuje so 16 bitovými registrami. Podstatnejší rozdiel v porovnaní s TC0 je v tom, že TC1 vie merať parametre externých vstupných signálov.

TC1 obsahuje hardvérovú podporu na „zachytávanie“ zmien (*Input Capture*) na vstupoch mikrokontroléra, ktorá dokáže zachytiť externé udalosti a zaznamenať čas vzniku tejto zmeny. Externý signál sa pripája na vývod ICP1 alebo alternatívne na vstup analógového komparátora. Zo zaznamenaných časov zmien signálu na vstupe ICP1 je možné vypočítať frekvenciu vstupného signálu, periódu signálu ako aj striedu signálu.

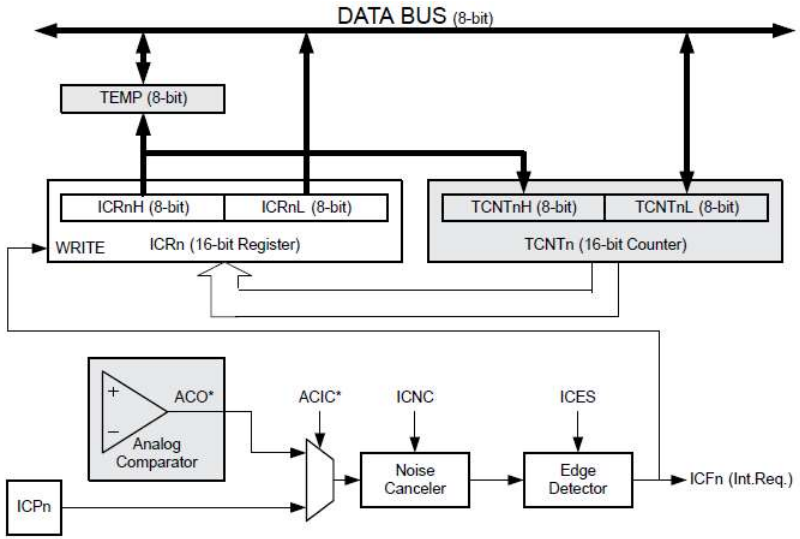



Obr. 72 Bloková schéma 16 bitového časovača – počítača TC1

Keď nastane zmena logickej úrovne (udalosti) na vstupe ICP1 alebo alternatívne na výstupe analógového komparátora (ACO) a táto zmena súhlasí nastavením detektora zmeny hrany signálu (vzostupná, zostupná hrana), aktivuje sa snímanie t. j. 16-bitová hodnota počítadla TCNT1 sa zapíše do registra vstupu zachytenia ICR1. Súčasne v rovnakom okamihu je nastavený aj príznak zmeny

vstupného signálu ICF. Ak je povolené vykonanie prerušenia TIMSK1.ICIE, prerušenie sa vykoná čím sa automaticky nuluje príznak prerušenia. Ak prerušenie nie je povolené, príznak ICF je možné nulovať softvérovo.

Súčasťou vstupných obvodov detekovania hrán impulzov je obvod na potlačenie šumu, ktorý využíva jednoduchý digitálny filter. Obvod šumovej imunity monitoruje štyri po sebe idúce vzorky a všetky musia mať rovnakú úroveň, ktorá sa potom použije detektorom hrany impulzu. Funkcia obvodu na potlačenie šumu sa povoľuje nastavením bitu riadiacom registri B (TCCR1B.ICNC). Ak je táto funkcia zapnutá, aktualizácia registra ICR1 nastane až po uplynutí štyroch cyklov systémových hodín po zmene na vstupe ICP1.




Obr. 73 Hardvérové riešenie „zachytávania“ hrany signálu na vstupe (n=1)

Hlavnou úlohou pri používaní jednotky *Input Capture* je zabezpečiť dostatočný výkon procesora na zvládnutie prichádzajúcich udalostí (zmien hrán vstupného signálu). Čas medzi dvoma zmenami nesmie byť kratší ako je čas, ktorý procesor potrebuje na prečítanie zaznamenanej hodnoty v registri ICR1. V opačnom prípade sa obsah v registri ICR1 prepíše novou hodnotou. AK sa použije na vyčítavanie obsahu registra ICR1 prerušenie, vyčítanie obsahu ICR1 v obsluhe prerušenia by malo byť čo najskôr. Napriek tomu, že prerušenie (TIMER CAPT) na vstupe ICP1 má relatívne vysokú prioritu, maximálna doba odozvy na prerušenie závisí od maximálneho počtu cyklov hodín, ktoré sú potrebné na zvládnutie akýchkoľvek iných požiadaviek na prerušenie. Meranie striedy externého signálu vyžaduje, aby sa po každom zachytení menilo nastavenie hrany, na ktorú má obvod





zachytenia reagovať (vzostupná hrana, zostupná hrana). Zmena snímania hrán sa musí vykonať čo najskôr po prečítaní obsahu registra ICR1. Po zmene snímanej hrany, sa musí nulovať softvérovo príznak (ICF). Pri meraní frekvencie signálu, nie je potrebné nulovať ICF softvérovo.

- Príklad merania periódy signálu na vstupe ICP1. Výsledný počet impulzov periódy je potrebné vynásobiť jednotkou času.

```
//***************************************************************************
#define F_CPU 16000000UL
int main ( )
{
  unsigned int t;
  char buffer[35];

  TCCR1B |= (1<<ICES1)|(1<<CS10);       //capture on rising edge

  while (1){
    TCNT1 = 0;
    TIFR1 = (1<<ICF1);                  //clear capture flag
    while ((TIFR1 & (1<<ICF1)) == 0);   //monitor for capture
    t = ICR1;
    TIFR1 = (1<<ICF1);                  //clear capture flag
    while ((TIFR1 & (1<<ICF1)) == 0);   //monitor for capture
    t = ICR1 - t;                       //period=recent-previous

    sprintf (buffer, "period=%u\n\r", (int)(t/F_CPU));
    USART_Transmit_string(buffer);
    _delay_ms (1000);
  }
}
```

- Príklad merania frekvencie a striedy na vstupe ICP1.

```
//***************************************************************************
#define F_CPU 16000000UL
int main ( )
{
  unsigned int  a, b, c, high, period;
  char          buffer[35];
  unsigned long freq;
  float         duty;

  while (1){
    TCNT1 = 0;
    TCCR1B = (1<<ICES1)|(1<<CS10);      //rising edge no prescaler
    TIFR1 = (1<<ICF1);                  //clear capture flag
    while ((TIFR1 & (1<<ICF1)) == 0);   //monitor for capture
    a = ICR1;

    TCCR1B = (1<<CS10);                 //falling edge no prescaler
    TIFR1 = (1<<ICF1);                  //clear capture flag
    while ((TIFR1 & (1<<ICF1)) == 0);   //monitor for capture
    b = ICR1;

    TCCR1B = (1<<ICES1)|(1<<CS10);      //rising edge no prescaler
    TIFR1 = (1<<ICF1);                  //clear capture flag
    while ((TIFR1 & (1<<ICF1)) == 0);   //monitor for capture
    c = ICR1;

    if ((a < b) && (b < c))               //check for valid condition
    {
```

```
        high = b - a;
        period = c - a;
        freq = F_CPU/period;
        duty = ((float)high / (float)period)*100;
      }
      sprintf (buffer, "freq=%luHz, duty =%3.1f%%\n\r", freq, duty);
      USART_Transmit_string(buffer);
      _delay_ms (1000);
    }
}
```

### 11.4. TC2 – 8-bit časovač / počítadlo s PWM a asynchrónnym režimom

TC2 je univerzálny 8-bitový časovač - počítač s dvoma nezávislými výstupnými porovnávacími jednotkami s podporou PWM. Umožňuje časovanie vykonávania programu (správy udalostí) a generovanie priebehov. TC2 má v SDRAM pamäti vyhradené nasledujúce registre, ktoré potrebuje pre svoju činnosť:

- TCTN2 (0xB2) – register hodnoty počítania, BOTTOM – TCNT2 = 0x00, MAX – TCNT2 = 0xFF, TOP – hodnota TCNT2 sa rovná hodnote OCR2A,
- TCCR2A (0xB0), TCCR2B (0xB1) – riadiace registre TC2,
- OCR2A (0xB3), OCR2B (0xB4) – výstupný porovnávací register,
- TIMSK2 (0x70) – register maskovania prerušení generovaných TC2,
- TIFR2 (0x37) – register príznakov prerušení generovaných TC2,
- ASSR (0xB6) – register stavu asynchrónneho módu,

TC2 je v podstate zhodný s TC0 (umožňuje všetky režimy činnosti), s tým rozdielom že TC2 podporuje ešte aj asynchrónnu komunikáciu.

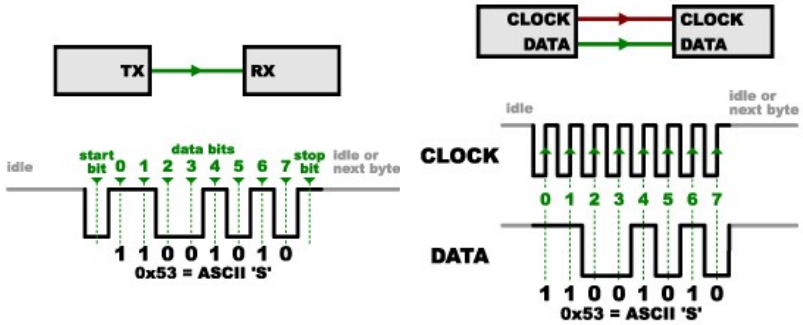




## 12. Sériové komunikačné prostriedky

Sériová komunikácia alebo sériový prenos v elektronických číslicových obvodoch je proces prenosu dát postupne po jednotlivých bitoch (t. j. sekvenčne) pomocou komunikačnej zbernice. Sériová komunikácia je v priamom protiklade s paralelnou komunikáciou, kde sa naraz prenáša niekoľko bitov paralelnými vodičmi. Sériová komunikácia sa v procesorových aplikáciách používa aj preto, lebo odstraňuje problémy paralelnej komunikácie (parazitné kapacity, presluchy medzi paralelnými vodičmi a pod.). Sériová zbernica používa pre prenos dát jeden alebo dva dátové vodiče, na riadenie činnosti na zbernici sa používajú riadiace vodiče alebo riadiace bity. Formát prenášaných dát, časovanie prenosu, riadenie prenosu atď. popisuje protokol zbernice. Štandard zbernice určuje aj elektrické a mechanické (konektory) parametre zbernice. Dátová informácia sa prenáša buď zmenou elektrického napätia alebo zmenou elektrického prúdu. Realizácia zmenou napätia je jednoduchšia, zmenou prúdu je zložitejšia, ale má väčšiu odolnosť voči elektromagnetickému rušeniu.

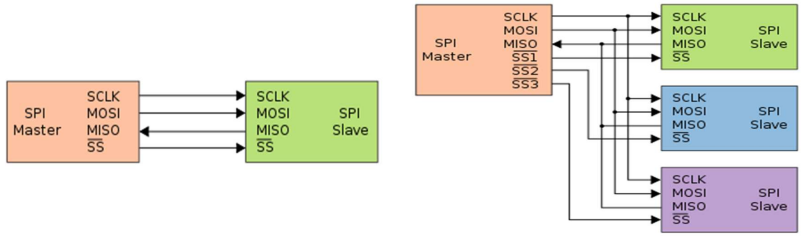
Fyzický prenos komunikácie môžeme rozdeliť na **synchrónnu** a **asynchrónnu** komunikáciu. O synchrónnej komunikácií hovoríme vtedy, keď je prenos jednotlivých bitov z hľadiska vysielača a prijímača synchronizovaný jednotným riadiacim (synchronizačným) signálom. Asynchrónna komunikácia nepoužíva žiadny riadiaci signál, vysielač aj prijímač musia pred prenosom poznať časové parametre prenosu (prenosovú rýchlosť). Prijímač asynchrónneho komunikačného kanálu musí vedieť identifikovať začiatok a koniec dátového prenosu. K tomuto účelu sa využívajú riadiace bity: štart bit a stop bit, ktoré určujú prechod dátového signálu z neaktívnej do aktívnej úrovne a naopak.




Obr. 74 Časový priebeh asynchrónnej komunikácie(vľavo) a synchrónnej komunikácie (vpravo)

## 12.1. Sériové periférne rozhranie - SPI

Sériové periférne rozhranie (SPI) umožňuje vysokorýchlostný synchrónny prenos dát medzi mikrokontrolérom a perifémymi zariadeniami. Komunikácia je plne duplexná typu *Master – Slave* realizovaná pomocou štvorvodičovej zbernice (*Master* - hlavný, ten ktorý riadi komunikáciu, *Slave* - podriadený, ten ktorý sa prispôsobuje komunikácií). Môže byť tvorená jedným *Master* zariadením a viacero *Slave* zariadeniami. Adresovanie *Slave* zariadení sa realizuje pomocou signálových vodičov s označením SS (*Slave Select*) alebo CS (*Chip Select*), ktoré sú aktívne pri LOW a aktivujú komunikáciu so zvoleným zariadením. Prepojenie medzi MCU v režime *Master* a *Slave* zariadeniami je znázornené na obrázku (Obr. 75). *Master* iniciuje komunikáciu aktivovaním signálu /SS požadovaného *Slave* zariadenia. *Master* vysiela svoje dáta cez MOSI (*Master Output Slave Input*) signálový vodič a prijíma dáta od cez MISO (*Master Input Slave Output*) signálový vodič. Jednotlivé bity sa posielajú obojsmerne (súčasne) podľa hodinového signálu SCK (*Serial Clock*).

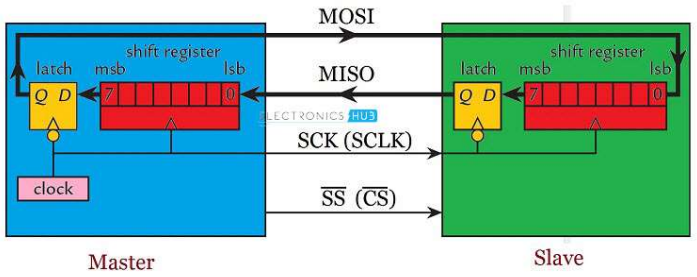



Obr. 75 Zapojenie SPI zbernice (vľavo jedno SPI zariadenie, vpravo viacero SPI zariadení)

Komunikácia cez SPI rozhranie prebieha nasledovne: *Master* vyberie *Slave* zariadenie pomocou /SS signálu. Počas každého cyklu hodín SCK dochádza k plne duplexnému prenosu dát (Obr. 76). *Master* posiela bity cez MOSI vodič a *Slave* ich číta a súčasne *Slave* posiela bity cez MISO vodič a *Master* ich číta. Tento postup je zachovaný aj keď je použitý jednosmerný prenos dát (najprv posiela *Master* a potom posiela *Slave*). Obojsmerný dátový prenos tvoria dva osem bitové posuvné registre, jeden v *Master* a jeden v *Slave* zariadení, ktoré sú spojené vo virtuálnej kruhovej topológii. Dáta sú zvyčajne posielané s MSB bitom ako prvým. Hranou hodinového signálu sa zabezpečuje prenos jednotlivých bitov cez dátové vodiče (MOSI a MISO), ktoré prepájajú obe zariadenia. Po ôsmich hodinových taktoch je jeden bajt prenesený z *Master* zariadenia do *Slave* zariadenia a súčasne zo *Slave* zariadenia do *Master* zariadenia.





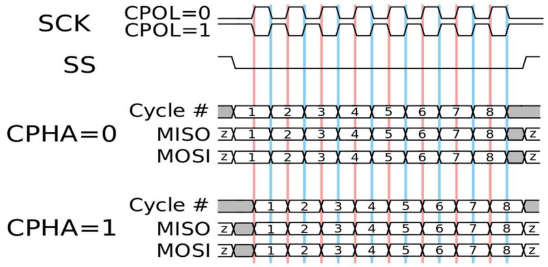



Obr. 76 Hardvérové riešenie duplexného prenosu SPI

Okrem generovania hodinového signálu SCK musí *Master* nakonfigurovať aj polaritu hodín CPOL a fázu CPHA vzhľadom k platnosti dát na signálových vodičoch. CPOL=0 znamená, že predná hrana SCK je stúpajúca hrana a zadná hrana je klesajúca. CPOL=1 znamená, že predná hrana je klesajúca a zadná hrana SCK signálu je stúpajúca. CPHA určuje časovanie (fázu) dátových bitov vzhľadom na hodinové impulzy. CPHA=0 znamená, že predná hrana SCK určuje platnosť dátových bitov na MISO a MOSI vodičoch a v čase zadnej hrany dochádza k zmene dátových bitov. CPHA=1 znamená, že zadná hrana SCK určuje platnosť dátových bitov na MISO a MOSI vodičoch a v čase prednej hrany dochádza k zmene dátových bitov (Obr. 77).

SPI rozhranie v ATmega328P možno charakterizovať:

- Plne duplexný režim prenosu pomocou trojvodičovej zbernice (MISO, MOSI, SCK),
- Režim činnosti *Master* alebo *Slave*, nastaviteľná rýchlosť prenosu, generovanie prerušenia na konci prenosu.
- SPI má v SRAM pamäti vyhradené tieto registre: SPCR0 (0x4C) – SPI riadiaci register, SPSR0 (0x4D) – SPI status register, SPDR0 (0x4E) – SPI dátový register.




Obr. 77 Časový diagram polarity a fázy hodinového signálu SPI prenosu.

Ak je MCU nakonfigurovaný ako *Master*, rozhranie SPI neovláda automaticky signál SS. Tento musí byť riadený softvérovo a aktivovaný skôr, ako začne komunikácia. Ak je SS aktívny, potom zapísaním bajtu do dátového registra SPDR0 sa spustí generátor hodín SPI a hardvér zabezpečí odvysielanie ôsmich bitov do *Slave* zariadenia. Po vyslaní bajtu sa generátor hodín SPI zastaví a nastaví sa príznak konca prenosu SPIF. Ak je nastavený bit prerušenia SPIE, vykoná sa prerušenie, v ktorom môže *Master* pokračovať vo vysielaní ďalšieho bajtu tým, že ho zapíše do SPDR alebo signalizovať *Slave* koniec prenosu deaktivovaním SS signálu. Posledný prichádzajúci bajt bude uložený v registri vyrovnávacej pamäte pre neskoršie použitie.

Ak je MCU nakonfigurovaný ako *Slave*, rozhranie SPI je v pohotovostnom režime s MISO signálom v treťom stave (stav vysokej impedancie) tak dlho, pokiaľ je signál /SS neaktívny. V tomto stave, program môže aktualizovať obsah dátového registra SPDR0, tento bajt sa bude vysielať počas prichádzajúcich hodinových impulzov na vývode SCK. Keď bol vyslaný bajt, nastaví sa príznak ukončenia vysielania SPIF. Ak je povolené prerušenie SPIE, vykoná sa prerušenie. *Slave* môže pokračovať v umiestňovaní nových dát do dátového registra SPDR0 po načítaní prichádzajúcich dát.

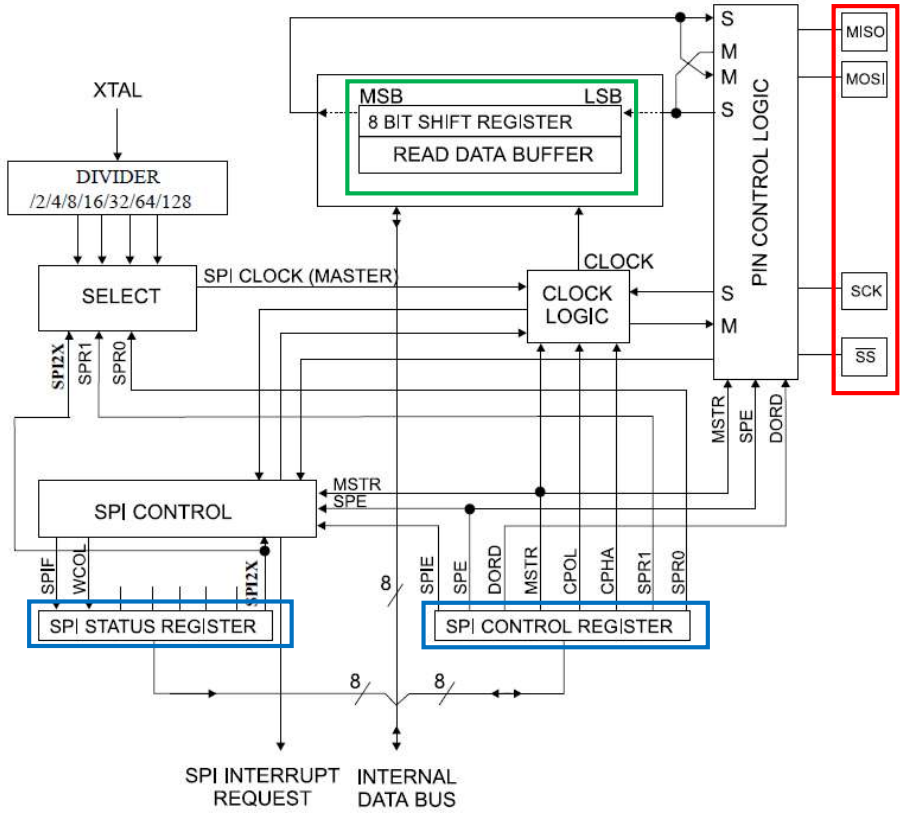
Tab. 18 Zmena smeru vývodov MCU podľa režimu SPI prenosu.

| Pin | Direction, Master SPI | Direction, Slave SPI |
|---|---|---|
| MOSI | User Defined | Input |
| MISO | Input | User Defined |
| SCK | User Defined | Input |
| $\overline{SS}$ | User Defined | Input |

Ak je povolené SPI podľa zvoleného režimu prenosu MCU sa smer vývodov signálov MOSI, MISO, SCK a /SS sa riadi podľa tabuľky (Tab. 18). Výstupné vývody sú ovládané programom, vstupné vývody ovláda externé zariadenie, ktoré sa zúčastňuje komunikácie.








Obr. 78 Bloková schéma SPI v AVR

- Príklad inicializácie SPI ako *Master*.

```
//**************************************************************************
void SPI_MasterInit(void)
{
  //Set MOSI and SCK output, all other input
  DDRB |= (1<<DDB3)|(1<<DDB5);
  //Enable SPI, Master, set clock rate fck/16
  SPCR0 = (1<<SPE0)|(1<<MSTR0)|(1<<SPR00);
}
```

- Príklad vysielania jedného bajtu.

```
//**************************************************************************
void SPI_MasterTransmit(char cData)
{
  SPDR0 = cData;                    //Start transmision
  while(!(SPSR0 & (1<<SPIF0)));     //Wait for transmission complete
}
```

- Príklad inicializácie SPI ako *Slave*.

```
//**********************************************************************
void SPI_SlaveInit(void)
{
  DDRB |= (1<<DDB4)|(1<<DDB5);      //Set MISO output
  SPCR0 = (1<<SPE0);                //Enable SPI
}
```

- Príklad príjmu jedného bajtu.

```
//**********************************************************************
char SPI_SlaveReceive(void)
{
  while(!(SPSR0 & (1<<SPIF0)));     //Wait for reception complete
  return SPDR0;                     //Return Data Register
}
```

## 12.2. Univerzálny synchrónny-asynchrónny prijímač vysielač - USART

**USART** (*Universal Synchronous Asynchronous Receiver Transceiver*) je flexibilné sériové komunikačné rozhranie, ktoré môže pracovať v plne duplexnom režime (súčasne prijíma a vysiela dáta). Môže pracovať v dvoch režimoch:

- asynchrónny režim, ktorý môžeme označiť UART,
- synchrónny režim, v ktorom sa z USART správa ako SPI.

USART má v I/O SRAM pamäti vyhradené nasledujúce registre, ktoré používa pri svojej činnosti:

- UDR0 (0xC6) – dátový register,
- UCSR0A (0xC0), UCSR0B (0xC1), UCSR0C (0xC2) – riadiace a status registre,
- UBRR0L (0xC4), UBRR0H (0xC5) – registre prenosovej rýchlosti.

Zariadenie USART sa skladá z troch blokov (Obr. 79):

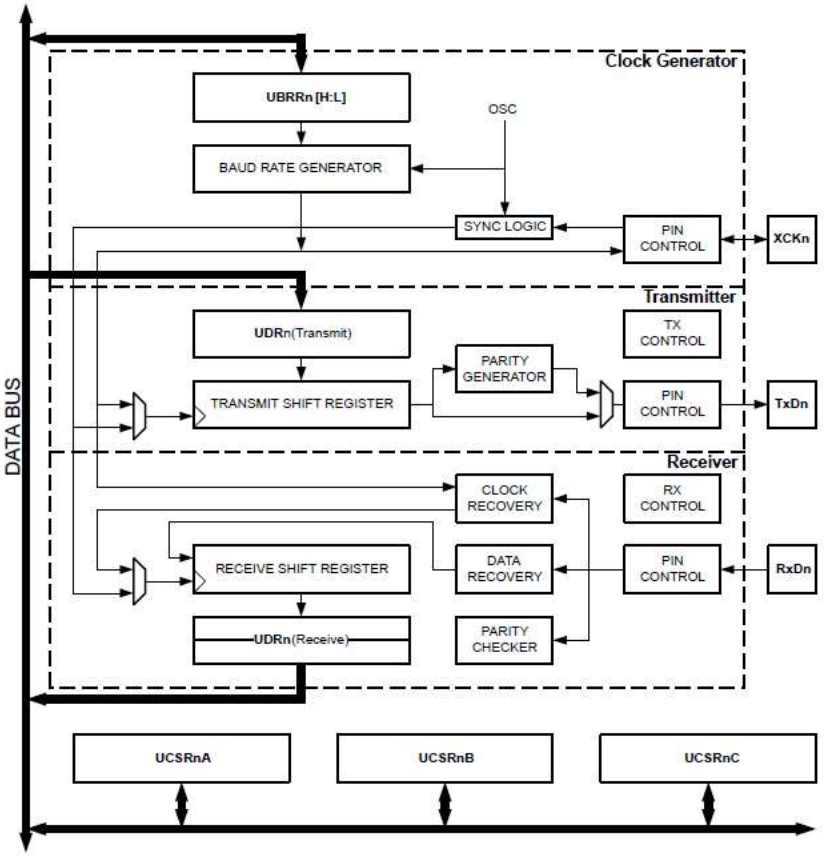
- Obvody generátora hodinového signálu (*Clock Generator*), ktorý generuje hodinový signál pre vysielač aj prijímač.
- Obvody vysielača sériového signálu (*Transmitter*).
- Obvody prijímača sériového signálu (*Receiver*).

USART podporuje štyri režimy prevádzky, ktoré sa nastavujú pomocou bitov UCSR0C.UMSEL0[1:0], UCSR0A.U2X0 :

- asynchrónny normálny,
- asynchrónny s dvojnásobnou rýchlosťou,
- synchrónny mód,
- *Master* SPI.








Obr. 79 Bloková schéma USARTu

Register UBRR0 prenosovej rýchlosti USARTu obsahuje hodnotu pre deličku generátora prenosovej rýchlosti. Prenosová rýchlosť USARTu pre normálny asynchrónny režim sa vypočíta podľa vzťahu:

$$BAUD = \frac{f_{osc}}{16 * (UBRR0 + 1)} \tag{3.4}$$

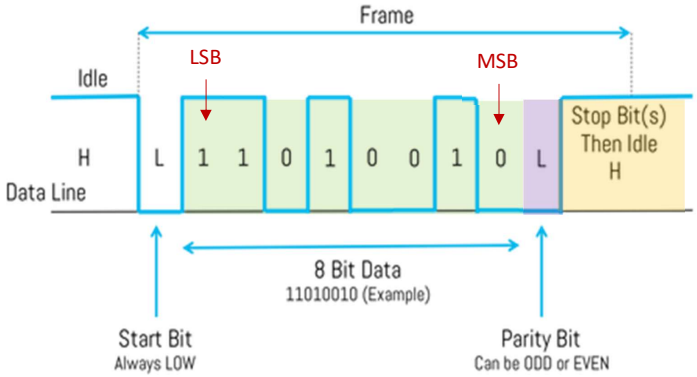
Prenosová rýchlosť pre synchrónny režim sa vypočíta:

$$BAUD = \frac{f_{osc}}{2 * (UBRR0 + 1)} \tag{3.5}$$

Prenos dát prostredníctvom USARTu sa realizuje vysielaním resp. prijímaním *frame* (prenosový rámec). Predpokladajme, že budeme používať USART len v asynchrónnom režime a preto, vysielač aj prijímač musia mať nastavenú tú istú prenosovou rýchlosťou.

*Frame* je definovaný ako blok bitov, ktorý je tvorený: dátovými bitmi, synchronizačnými bitmi (štart a stop) a voliteľným paritným bitom na kontrolu integrity *frame* (Obr. 80). USART v ATmega328 akceptuje všetkých 30 kombinácií formátov:

- 1 štartovací bit (vždy LOW),
- 5, 6, 7, 8 alebo 9 dátových bitov, počet sa nastavuje pomocou bitov UCSR0C.UCSZ0[1:0],
- žiadny, párny alebo nepárny paritný bit, nastavuje sa pomocou bitov UCSR0C.UPM0[1:0], 1 alebo 2 stop bity (HIGH), nastavuje sa pomocou bitu UCSR0C.USBS0.




Obr. 80 Formát dát sériového prenosu - *Frame format*

Príklad *frame* na Obr. 80 prenáša bajt 0b01001011 teda 0x4B a paritný bit je 0 lebo je zvolená párna parita a je prenesený párny počet jednotkových bitov.

*Frame* vždy začína štartovacím bitom, po ktorom nasledujú dátové bity (5 až 9 dátových bitov). Ako prvý sa prenáša najmenej významový dátový bit LSB (*Last Significant Bit*), potom ďalšie bity a ako posledný sa prenáša najviac významový bit MSB (*Most Significant Bit*). Ak je povolený paritný bit, ten nasleduje po dátových bitoch. Ako posledné sa prenášajú jeden alebo dva stop bity. Po prenose celého *frame* môže nasledovať prenos ďalšieho *frame* alebo môže byť komunikačná linka nastavená do vyčkávacieho *idle* stavu.

Parita je jednoduchý kontrolný súčet určený pre ochranu integrity jedného *frame*. Parita je bit, ktorý sa pripája k dátovému bajtu a vyjadruje, či je počet logických jednotiek v dátových bitoch párny alebo nepárny. Ak je zvolená párna parita, paritný bit je nastavený na HIGH, ak je počet jednotiek v dátovom bajte nepárny a na LOW, ak je počet bitov párny. Ak zvolená nepárna parita, paritný bit





nastavený na HIGH, ak je počet jednotiek v dátovom bajte párny a na LOW, ak je počet bitov nepárny.

Tab. 19 Hodnoty prenosovej rýchlosti pre USART

| Baud Rate [bps] | $f_{osc}$ = 16.0000MHz | | | | $f_{osc}$ = 18.4320MHz | | | | $f_{osc}$ = 20.0000MHz | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | | U2Xn = 0 | | U2Xn = 1 | |
| | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error | UBRRn | Error |
| 2400 | 416 | -0.1% | 832 | 0.0% | 479 | 0.0% | 959 | 0.0% | 520 | 0.0% | 1041 | 0.0% |
| 4800 | 207 | 0.2% | 416 | -0.1% | 239 | 0.0% | 479 | 0.0% | 259 | 0.2% | 520 | 0.0% |
| 9600 | 103 | 0.2% | 207 | 0.2% | 119 | 0.0% | 239 | 0.0% | 129 | 0.2% | 259 | 0.2% |
| 14.4k | 68 | 0.6% | 138 | -0.1% | 79 | 0.0% | 159 | 0.0% | 86 | -0.2% | 173 | -0.2% |
| 19.2k | 51 | 0.2% | 103 | 0.2% | 59 | 0.0% | 119 | 0.0% | 64 | 0.2% | 129 | 0.2% |
| 28.8k | 34 | -0.8% | 68 | 0.6% | 39 | 0.0% | 79 | 0.0% | 42 | 0.9% | 86 | -0.2% |
| 38.4k | 25 | 0.2% | 51 | 0.2% | 29 | 0.0% | 59 | 0.0% | 32 | -1.4% | 64 | 0.2% |
| 57.6k | 16 | 2.1% | 34 | -0.8% | 19 | 0.0% | 39 | 0.0% | 21 | -1.4% | 42 | 0.9% |
| 76.8k | 12 | 0.2% | 25 | 0.2% | 14 | 0.0% | 29 | 0.0% | 15 | 1.7% | 32 | -1.4% |
| 115.2k | 8 | -3.5% | 16 | 2.1% | 9 | 0.0% | 19 | 0.0% | 10 | -1.4% | 21 | -1.4% |
| 230.4k | 3 | 8.5% | 8 | -3.5% | 4 | 0.0% | 9 | 0.0% | 4 | 8.5% | 10 | -1.4% |
| 250k | 3 | 0.0% | 7 | 0.0% | 4 | -7.8% | 8 | 2.4% | 4 | 0.0% | 9 | 0.0% |
| 0.5M | 1 | 0.0% | 3 | 0.0% | – | – | 4 | -7.8% | – | – | 4 | 0.0% |
| 1M | 0 | 0.0% | 1 | 0.0% | – | – | – | – | – | – | – | – |
| Max.(1) | 1Mbps | | 2Mbps | | 1.152Mbps | | 2.304Mbps | | 1.25Mbps | | 2.5Mbps | |

USART sa musí inicializovať skôr, ako sa môže uskutočniť akákoľvek komunikácia. Proces inicializácie zvyčajne pozostáva z nastavenia prenosovej rýchlosti, nastavenia formátu prenosu a povolenie vysielača alebo prijímača v závislosti od ich použitia:

1. nastaviť prenosovú rýchlosť v registri UBRR0 v bitoch za sekundu (*baud*), dôležitá je frekvencia $f_{osc}$, pretože má vplyv na generátor prenosovej rýchlosti. Prenosovú rýchlosť volíme tak, aby percentuálna chybovosť mala čo najnižšiu hodnotu pre konkrétnu hodnotu frekvencie oscilátora (Tab. 19).
2. povoliť vysielanie dát TxD a (alebo) prijímanie dát RxD v UCSR0B,
3. nastaviť *frame* formát v UCSR0C,
4. povoliť (zakázať) prerušenie od činnosti USARTu v UCSR0B, v prípade riadenia komunikácie cez prerušenia. K dispozícii sú tri vektory prerušenia *USART Rx Complete, USART Data Register Empty, USART Tx Complete*.

```
• Príklad inicializácie USARTu, prenos dát bez použitia prerušenia.
#define   F_CPU 16000000UL
#define   BAUD 9600
#define   UBRR F_CPU/16/BAUD-1
//**********************************************************************
```

```
void USART_Init (unsigned int ubrr)
{
  // set baud rate
  UBRR0H = (unsigned char)(ubrr>>8);
  UBRR0L = (unsigned char)ubrr;
  // Enable receiver and transmitter;
  UCSR0B |= (1 << RXEN0) | (1 << TXEN0);
  //Set Frame format: 8data 1stop
  UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);
}
//**********************************************************************
void main(void)
{
......

  USART_Init(UBRR);

......
}
```

- Príklad inicializácie USARTu, prenos dát s použitím prerušení.

```
#define   F_CPU 16000000UL
#define   BAUD 9600
#define   UBRR F_CPU/16/BAUD-1
//**********************************************************************
void USART_Init_isr (unsigned int ubrr)
{
  //Set baud rate
  UBRR0H = (unsigned char)(ubrr>>8);
  UBRR0L = (unsigned char)ubrr;
  //Enable receiver and transmitter, interrupt RxD, interrupt TxD;
  UCSR0B |= (1<<RXEN0)|(1<<TXEN0)|(1<<RXCIE0)|(1<<TXCIE0);
  //Set Frame format: 8data 1stop
  UCSR0C |= (1 << UCSZ01) | (1 << UCSZ00);
}
//**********************************************************************
void ISR_USART_TX (USART_TX_vect)
{
  ...
}
//**********************************************************************
void ISR_USART_RX (USART_RX_vect)
{
  ...
}

//**********************************************************************
void main(void)
{
  ...

  USART_Init(UBRR);

  ...
}
```

### 12.2.1. USART vysielač

Je povolený nastavením bitu vysielania UCSR0B.TXEN0. Ak je povolený USART vysielač, funkcia vstupno-výstupného vývodu PD1 je automaticky prepojená na funkciu sériového výstupu vysielača





TxD. Prenos dát cez vývod TxD sa začína zápisom vysielaného dátového bajtu do dátového registra UDR0. Hardvér zabezpečí odvysielanie jednotlivých bitov podľa nastavených parametrov v inicializácií USARTu.

- Príklad funkcie pre vyslanie bajtu cez USART

```
//**********************************************************************
void USART_Transmit (char dataout)
{
  while(!(UCSR0A & (1<<TXC0)));     //Wait for transmit complete
  //while(!(UCSR0A & (1<<UDRE0)));    //Wait for empty UDR0
  UDR0 = dataout;                   //Put data into UDR0, send data
}
```

Vysielanie dát cez USART je možné sledovať pomocou dvoch príznakov, ktoré charakterizujú stav procesu vysielania: prázdny dátový register - UCSR0A.UDRE0 a dátový prenos kompletný - UCSR0A.TXC0. Obidva príznaky môžu byť použité na generovanie prerušenia. Príznak UDRE0 signalizuje, či je dátový register pripravený na prijímanie nových dát. Príznak TXC0 je nastavený, keď bol celý *frame* odvysielaný a vo vyrovnávacej pamäti nie sú prítomné nové dáta. Príznak TXC0 sa používa v polo duplexných komunikačných rozhraniach (ako je štandard RS-485), kde musí vysielacia aplikácia prejsť do režimu prijímania a komunikačnú zbernicu uvoľniť ihneď po dokončení vysielania.

- Príklad funkcie pre vyslanie viacerých bajtov cez USART

```
//**********************************************************************
void USART_Transmit_Text (char *text) {

  unsigned char len, i;

  len = strlen(text);               //Length of text
  for (i=0; i<len; i++)             //Repeat
    USART_Transmit (text[i]);       //Send data
}
```

### 12.2.2. USART prijímač

Je povolený nastavením bitu vysielania UCSR0B.RXEN. Ak je povolený USART prijímač, funkcia vstupno-výstupného vývodu PD0 je automaticky prepojená na funkciu sériového vstupu vysielača RxD. Prijímač spustí príjem dát, keď zistí platný štartovací bit na vstupe RxD. Každý bit, ktorý nasleduje po štartovom bite sa bude snímať prenosovou rýchlosťou alebo hodinami na vstupe XCK (PD4) a presunie sa do posuvného registra prijímača, kým sa neprijme prvý stopový bit. Druhý stop bit (ak je použitý) prijímač ignoruje. Prijatie stop bitu signalizuje, že v posuvnom registri sú zapamätané všetky dátové bity a obsah posuvného registra sa presunie do dátového registra UDR0. Následne je možné načítať prijatý bajt čítaním obsahu UDR0 buď jednoduchou funkciou v hlavnej slučke programu, alebo cez prerušenie *USART Rx Complete*.

- Príklad funkcie pre príjem bajtu cez USART

```
//*****************************************************************************
unsigned char USART_Receive(void)
{
  while(!(UCSR0A & (1<<RXC0)));     //Wait for data to be received
  return UDR0;                      //Get and return received data
}
```

Prijímač USART má len jeden príznak UCSR0A.RXC0, ktorý označuje stav prijímača. Príznak dokončenia príjmu RXC0 označuje, či je v dátovom registri UDR0 neprečítaný bajt. Ak príznak má hodnotu HIGH znamená to, že UDR0 obsahuje neprečítaný bajt, ak je LOW, UDR0 neobsahuje dáta. Ak je povolené prerušenie po ukončení príjmu jedného *frame* UCSR0B.RXC0IE, obsluha prerušenia musí najprv vyčítať obsah registra UDR0, čím sa vynuluje príznak RXC0 a prijímač je pripravený na príjem ďalšieho *frame*. Počas príjmu *frame* môže dochádzať k vzniku chýb, ktoré vie USART detegovať a signalizovať pomocou trojice príznakov, ktoré sú zaznamenané v riadiacom registri UCSR0A:

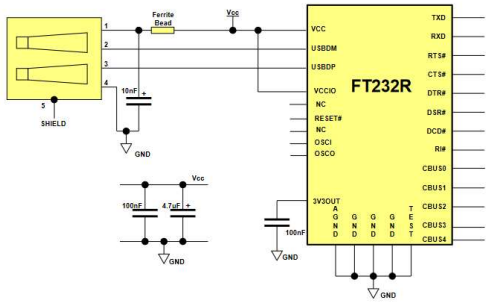
- Chyba *frame* UCSR0A.FE0 - signalizuje, že prijatý stop bit je LOW a nie HIGH,
- Pretečenie dát UCSR0A.DOR0 – signalizuje, že UDR0 má nevyčítaný bajt a bol prijatý nový štartovací bit,
- Chyba parity UCSR0A.UPE0 – signalizuje chybnú paritu prijatého bajtu.

### 12.2.3. Použitie USARTu na vývojovej doske Arduino *UNO*

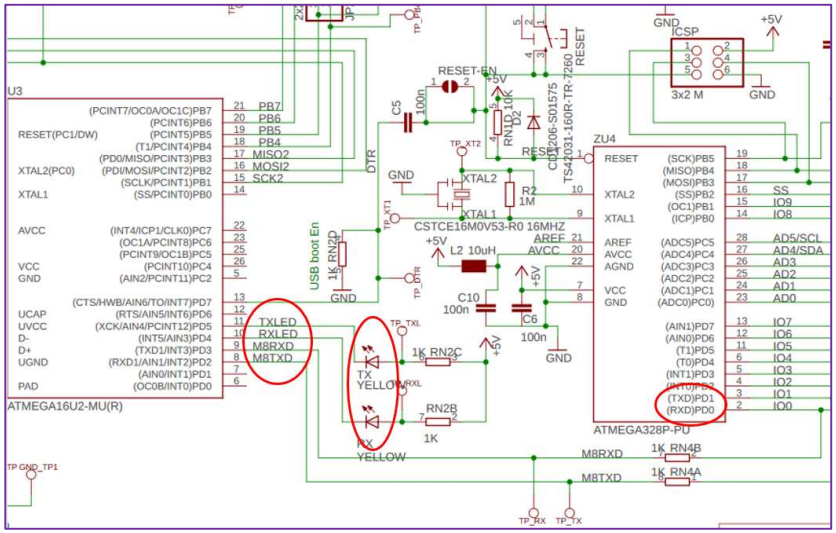
Asynchrónna sériová komunikácia mikrokontroléra ATmega328P sa realizuje pomocou vývodov PD1-TXD a PD0-RXD. Na vývojovej doske Arduino *UNO* sú tieto vývody prepojené cez rezistory RN4A = 1kΩ a RN4B = 1kΩ s prevodníkom *Serial/USB* realizovaným obvodom FT232 alebo ATmega16, ktorý zabezpečuje obojstrannú komunikáciu MCU s počítačom cez USB zbernicu. Súčasne je možné sledovať aktivitu na týchto dvoch vodičoch pomocou dvoch led diód RX a TX (Obr. 82). Toto prepojenie slúži na komunikáciu počítača s ATmega328P a nie je možné ho použiť na komunikáciu s iným hardvérom pripojeným k USARTu. Ak je potrebné používať USART komunikáciu s iným hardvérom, potom je potrebné odstrániť odpory RN4A a RN4B.








Obr. 81 Obvod FT232R používaný na prevod *Serial/USB* na doskách Arduino *UNO*




Obr. 82 Zapojenie vývodov RXD,TXD (ATmega328P) s prevodníkom *Serial/USB* (ATmega16)

### 12.2.4. Multiprocesorový komunikačný mód

Nastavením bitu režimu komunikácie s viacerými procesormi UCSR0A.MPCM0 sa použije funkcia filtrovania *Frames* prijatých prijímačom USART. *Frames*, ktoré neobsahujú informácie o adrese, budú ignorované a neuložia sa do prijímacej vyrovnávacej pamäte. To efektívne znižuje počet prichádzajúcich rámcov, ktoré musí spracovávať CPU v systéme s viacerými MCU, ktoré komunikujú cez rovnakú sériovú zbernicu. Vysielač nie je ovplyvnený nastavením multiprocesorovej

komunikácie, ale musí sa používať s ohľadom na systém využívajúci režim multiprocesorovej komunikácie.

Ak je prijímač nastavený na príjem rámcov, ktoré obsahujú 5 až 8 dátových bitov, potom prvý stop bit určuje, či rámec obsahuje dáta alebo informácie o adrese. Ak je prijímač nastavený na rámce s 9 dátovými bitmi, potom sa deviaty bit (RXB8) používa na identifikáciu adresových a dátových rámcov. Ak je bit určujúci typ rámca (prvý stop alebo deviaty bit) HIGH, rámec obsahuje adresu. Ak je bit typu rámca LOW, potom rámec obsahuje dáta.

Režim komunikácie s viacerými procesormi umožňuje niekoľkým podriadeným MCU prijímať dáta z hlavného MCU. To sa vykoná tak, že sa najprv dekóduje adresový rámec, aby sa zistilo, ktorý MCU bol adresovaný. Ak bola adresa pltná pre konkrétny MCU, pokračuje MCU v príjme dátových rámcov, zatiaľ čo ostatné MCU budú ignorovať prijaté rámce, kým sa neprijme ďalší adresový rámec.

Na výmenu údajov vo viacprocesorovom komunikačnom režime by sa mal použiť nasledujúci postup:

1. Všetky *Slave* MCU sú v režime multiprocesorovej komunikácie UCSR0A.MPCM0=1.

2. Hlavný MCU odošle adresový rámec a všetky podriadené jednotky tento rámec prijmú a čítajú. V podriadených MCU sa príznak RXC v UCSR0A nastaví ako normálny.

3. Každý *Slave* MCU načíta register UDR0 a určí, či bol vybraný. Ak áno, vymaže bit MPCM0 (UCSR0A.MCPM0=0), inak čaká na ďalší bajt adresy a zachová nastavenie MPCM0.

4. Adresovaný MCU bude prijímať všetky dátové rámce, kým nebude prijatý nový adresový rámec. Ostatné *Slave* MCU, ktoré majú stále nastavený bit MPCM0, budú ignorovať dátové rámce.

5. Keď adresovaná MCU prijme posledný dátový rámec, adresovaná MCU nastaví bit MPCM0 a čaká na nový adresový rámec od *Master* MCU. Proces sa potom opakuje od kroku 2.

Použitie ktoréhokoľvek z 5 až 8 bitových formátov rámca je možné, ale nepraktické, pretože prijímač musí prepínať medzi formátmi rámca znakov n a n+1. To sťažuje plne duplexnú prevádzku, pretože vysielač a prijímač používajú rovnaké nastavenie veľkosti znakov. Ak sa používajú 5 až 8 bitové znakové rámce, vysielač musí byť nastavený na používanie dvoch stop bitov (UCSR0C.USBS0=1), pretože prvý stop bit sa používa na označenie typu rámca.

#### 12.2.5. USART v móde SPI

Univerzálny synchrónny a asynchrónny sériový prijímač a vysielač (USART) možno nastaviť na režim prevádzky kompatibilný s hlavným SPI, nastavením bitov UCSR0C.UMSEL0[1:0] na HIGH. V tomto





režime hlavná riadiaca logika SPI preberá priamu kontrolu nad prostriedkami USART. Tieto prostriedky zahŕňajú posuvný register, vyrovnávacie pamäte vysielača a prijímača a generátor prenosovej rýchlosti. Generátor a kontrola parity, logika dát a hodín, logika riadenia RX a TX sú vypnuté. Logika riadenia vývodov a logika generovania prerušenia je identická v oboch režimoch prevádzky. Umiestnenia I/O registrov sú v oboch režimoch rovnaké. Niektoré funkcie riadiacich registrov sa však pri používaní MSPIM (*Master SPI Mode*) menia.

- USART MSPIM Initialization

```
//**********************************************************************
{
  UBRR0 = 0;
  DDR_XCK |= (1<<XCK);         //Set XCK as output, enables master mode
  //Set MSPI mode of operation and SPI data mode 0
  UCSR0C = (1<<UMSEL01)|(1<<UMSEL00)|(0<<UCPHA0)|(0<<UCPOL0);
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);  //Enable receiver and transmitter
  //The Baud Rate must be set after the transmitter is enabled
  UBRR0 = baud;
}
```

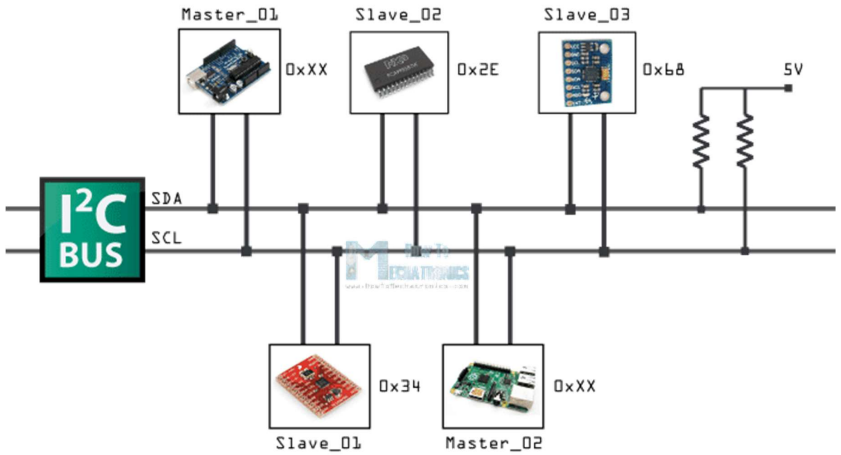
- USART MSPIM Data Transfer

```
//**********************************************************************
unsigned char MSPI_Transfer (unsigned char data)
{
  while (!(UCSR0A & (1<<UDRE0)));  //Wait for empty transmit buffer
  UDR0 = data;                     //Put data into UDR0, sends the data
  while (!(UCSR0A & (1<<RXC0)));   //Wait for data to be received
  return UDR0;                     //Get and return received data

}
```

### 12.3. Dvojvodičová sériová zbernica - TWI

Dvojvodičové sériové rozhranie TWI (*Two-Wire Interface*) je ideálne riešenie pre prepojenie mikrokontroléra s viacerými zariadeniami. Protokol TWI umožňuje pripojiť k mikrokontroléru až 128 rôznych zariadení pomocou dvoch vodičov: jeden hodinový signál SCL (*Serial Clock Line*) a jeden obojsmerný dátový signál SDA (Serial Data Line). TWI je implementácia sériovej I2C zbernice v mikrokontroléroch AVR.




Obr. 83 Príklad pripojenia zariadení k I2C zbernici

Oba vodiče zbernice sú pripojené cez *Pull-up* rezistory ku kladnému napájaciemu napätiu. I2C zariadenia pripojené ku zbernici môžu byť v režime *Master* (nadriadený) alebo v režime *Slave* (podriadený). Každé I2C zariadenie má svoju jedinečnú *Slave* adresu. Ak *Master* chce komunikovať so *Slave*, vyšle na I2C zbernicu adresu *Slave* zariadenia. Všetky *Slave* zariadenia príjmu túto adresu, ale v komunikácií pokračuje iba to zariadenie, ktorého adresa bola na zbernicu vyslaná. I2C zbernica podporuje tieto rýchlosti prenosu:

- 100Kbit/s v štandardnom móde,
- 400Kbit/s v rýchlom móde,
- 3,4Mbit/s vo veľmi rýchlom móde (TWI nepodporuje túto rýchlosť).

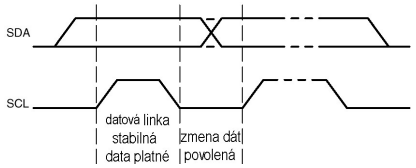
Tab. 20 Základné pojmy pre I2C zbernicu.

| | |
|---|---|
| ***Master*** | Zariadenie, ktoré iniciuje prenos, generuje hodinový signál a ukončuje prenos. |
| ***Slave*** | Zariadenie adresované *Master* zariadením. |
| **Vysielač** | Zariadenie, ktoré vysiela dáta na zbernicu. |
| **Prijímač** | Zariadenie, ktoré prijíma dáta zo zbernice. |

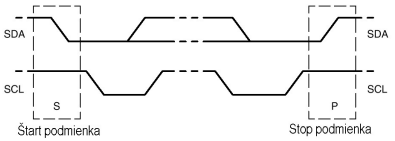
Prenášaný bit po dátovom SDA vodiči sa nastavuje v čase, keď na hodinovom SCL vodiči je stav LOW. Dátový vodič SDA musí byť stabilný (nastavená hodnota prenášaného bitu), keď je hodinový signál v stave HIGH (Obr. 84).





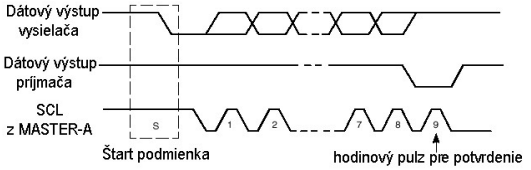



Obr. 84 Funkcia hodinového signálu




Obr. 85 Štart a stop stav na I2C zbernici

Počas komunikácie vznikajú na I2C zbernici stavy, ktoré sú definované ako START a STOP stavy (Obr. 85). START a STOP stavy sú vždy generované zariadením typu *Master*. Po vyslaný START podmienky sa zbernica považuje za obsadenú. Po vyslaní STOP podmienky je zbernica považovaná opäť za voľnú.




Obr. 86 Potvrdenie príjmu dát ACK zariadením na I2C zbernici

Každý bajt vyslaný na SDA vodiči musí pozostávať z 8 bitov. Za každým bajtom musí nasledovať bit potvrdenia ACK. Zariadenie ktoré prijíma dáta musí potvrdiť prijatie tak, že uvedie SDA vodič do LOW počas potvrdzujúceho hodinového impulzu na SCL vodiči (Obr. 86).

Ako prvý sa vysiela najviac významový bit MSB. Počet bajtov prenesených v rámci jedného prenosu nie je obmedzený. Ak prijímač nemôže prijať ďalší bajt (napríklad obsluhuje interné prerušenie), podrží SCL vodič na úrovni LOW, čím donúti vysielacie zariadenie k prechodu do čakacieho stavu. Dátový prenos pokračuje až vtedy, keď je prijímač pripravený prijať bajt, čo signalizuje uvoľnením hodinového SCL vodiča.

Dátový prenos začína *Master* vyslaním *Slave* adresy zariadenia na I2C zbernicu. Adresu tvorí 7 bitov, pričom posledný 8-mi bit určuje, či sa bude do zariadenia zapisovať, alebo sa bude z neho čítať.

Pre svoju činnosť TWI využíva v pamäti SDRAM vyhradené nasledujúce registre:

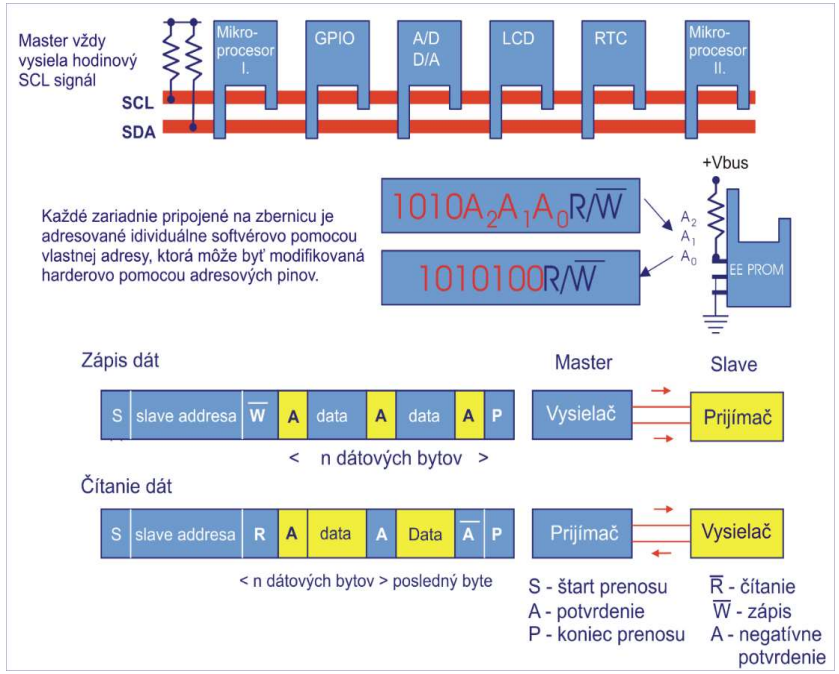
- TWBR (0xB8) – register prenosovej rýchlosti,
- TWSR (0xB9) – status register,
- TWAR (0xBA) – register adresy *Slave*, ak je MCU v režime *Slave*
- TWDR (0xBB) – dátový register,

- TWCR (0xBC) – riadiaci register,
- TWAMR (0xBD) – register *Slave* masky.

Na Obr. 87 je znázornený princíp komunikácie pre zápis dát na zbernicu a čítanie dát zo zbernice. Každé *Slave* zariadenie pripojené na zbernicu má svoju jedinečnú adresu, ktorá sa vo väčšine prípadov skladá z pevnej časti, ktorú nastavuje výrobca obvodu a voliteľnej časti, ktorú hardvérovo nastavuje užívateľ. Adresa je doplnená bitom R/W, ktorý určuje, ako sa so zariadením bude komunikovať (čítanie/zápis). Podľa Obr. 87 je adresa EEPROM pamäti tvorená pevnými bitmi 1010, ktoré sú nastavené výrobcom, voliteľnou častou adresy bity A2, A1, A0 a bitom R/W:

- SLAVE_Adr_R = 0b10101001 = 0xA9,
- SLAVE_Adr_W = 0b10101000 = 0xA8.

Ako je vidieť z príkladu, tak *Slave* zariadenie s obojsmernou komunikáciou má dve adresy, jednu pre zápis a druhú pre čítanie.




Obr. 87 Princíp komunikácie na I2C zbernici

Vývody SCL a SDA v MCU môžu využiť interné *Pull-up* rezistory a nemusia sa použiť externé rezistory, ako to je to znázornené na (Obr. 87).





Generátor prenosovej rýchlosti generuje hodinový signál na vodiči SCL, ak je MCU v režime *Master*. Frekvencia hodinového signálu SCL sa nastavuje bitmi registra TWBR.TWBR[7:0] a bitmi status registra TWSR.TWPS[1:0]. Ak je MCU v režime *Slave* frekvencia hodín CPU musí byť aspoň 16-krát vyššia ako frekvencia SCL vodiča.

$$f_{SCL} = \frac{f_{osc}}{16 + 2(TWBR) * (PrescalerValue)} \quad (3.6)$$

TWBR je hodnota z registra pre nastavenie prenosovej rýchlosti, *PrescalerValue* - môže nadobúdať hodnoty: 1, 4, 16 alebo 64.

Register TWDR obsahuje *Slave* adresu alebo dátový bajt, ktorý sa má vysielať, alebo prijatý dátový bajt. K 8-bitovému registru TWDR je pridružený bit ACK/NACK, ktorý sa vysiela alebo prijíma podľa toho, či je potrebné potvrdiť prijaté dáta alebo zistiť prijatie dát *Slave* zariadením.

### 12.3.1. Postup vyslania bajtu na zbernicu v režime *Master*

Ukážka vyslania jedného bajtu na zbernicu v režime *Master*. Ako prvé je potrebné inicializovať I2C zbernicu.

- Príklad inicializácie TWI zbernice v Master móde

```
//*********************************************************************
void TWI_Init (void)
{
  TWBR = 0x48;              // Set bit rate register. 100KHz CLK
  TWDR = 0xFF;             // Default content = SDA released.
  TWCR |= (1<<TWEN);       // Enable TWI-interface and release TWI pins.
}
```

Po inicializácií I2C zbernice sa prenos začína vyslaním START podmienky.

- Príklad funkcie na vyslanie START podmienky na zbernicu

```
//*********************************************************************
void StartTWI (void)
{
  TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWSTA);    //TWI Enable, Interrupt Flag
  TWCR |= (1<<TWSTA);                        //START Condition
  while (!(TWCR & (1<<TWINT)));              //Wait while TWINT = 1
}
```

Aplikačný softvér môže otestovať hodnotu status registra TWSR, aby si potvrdil, že START podmienka bola úspešne vyslaná na zbernicu. Ak obsah TWSR je iný ako očakávaný, môže aplikačný softvér zavolať chybovú obsluhu.

- Príklad funkcie na testovanie vyslania START podmienky na zbernicu

```
//*********************************************************************
void TestStart (void)
{
  if ((TWSR & 0xF8)!=0x08) Error();     //START has been not transmitted
}
```

Po vyslaní START podmienky, musí aplikácia nastaviť *Slave* adresu + bit zápisu (SLA + W) do dátového registra TWDR. Následne sa do riadiaceho registra TWCR zapíše pokyn na vyslanie *Slave* adresy na zbernicu.

- Príklad funkcie na vyslanie *Slave* adresy na zbernicu *data* = SLA+W

```
//*********************************************************************
void SendToTWI(char data)
{
  TWDR = data;                              //TWDR = SLA+W
  TWCR = (1<<TWINT)|(1<<TWEN);              //Transmit TWDR
  while (!(TWCR & (1<<TWINT)));             //Wait while TWINT = 1
}
```

Aplikačný softvér testuje hodnotu status registra TWSR, aby si potvrdil, že *Slave* adresa bola odoslaná a že hodnota bitu ACK bola podľa očakávania. Ak obsah TWSR0 je iný ako očakávaný, môže aplikačný softvér zavolať chybovú rutinu.

- Príklad funkcie na testovanie ACK potvrdenia od *Slave*

```
//*********************************************************************
void TestACK (void)
{
  if ((TWSR & 0xF8)!=0x18) Error();     //ACK has been not received
}
```

Ak je stav podľa očakávania, aplikácia zapíše dátový bajt do TWDR, ktorý sa má vyslať na zbernicu. Následne sa do riadiaceho registra TWCR zapíše pokyn na vyslanie bajtu z TWDR na TWI zbernicu.

- Príklad funkcie na vyslanie *data byte* na zbernicu

```
//*********************************************************************
void SendToTWI(char data)
{
  TWDR = data;                              //TWDR = data byte
  TWCR = (1<<TWINT)|(1<<TWEN);              //Transmit TWDR
  while (!(TWCR & (1<<TWINT)));             //Wait while TWINT = 1
}
```

Aplikačný softvér testuje hodnotu status registra TWSR, aby si potvrdil, že bajt bol odoslaný a že hodnota bitu ACK bola podľa očakávania. Ak obsah TWSR je iný ako očakávaný, môže aplikačný softvér zavolať chybovú rutinu.

- Príklad funkcie na testovanie ACK potvrdenia od *Slave*

```
//*********************************************************************
void TestACK (void)
{
  if ((TWSR & 0xF8)!=0x28) Error();       //ACK has been not received
}
```

Ak stav je podľa očakávania, do riadiaceho registra TWCR sa zapíše pokyn na STOP prenosu a ukončenie komunikácie.

- Príklad funkcie na ukončenie prenosu na zbernici

```
//*********************************************************************
```





```
void StopTWI (void)
{
  TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);       //STOP condition
}
```

### 12.3.2. Postup načítania bajtu zo zbernice v režime *Master*

Po inicializácií I2C zbernice sa prenos začína vyslaním START podmienky.

- Príklad funkcie na vyslanie START podmienky na zbernicu

```
//*********************************************************************
void StartTWI (void)
{
  TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWSTA);    //TWI Enable, Interrupt Flag
  TWCR |= (1<<TWSTA);                        //START Condition
  while (!(TWCR & (1<<TWINT)));              //Wait while TWINT = 1
}
```

Aplikačný softvér môže otestovať hodnotu status registra TWSR, aby si potvrdil, že START podmienka bola úspešne vyslaná na zbernicu. Ak obsah TWSR je iný ako očakávaný, môže aplikačný softvér zavolať chybovú obsluhu.

- Príklad funkcie na testovanie vyslania START podmienky na zbernicu

```
//*********************************************************************
void TestStart (void)
{
  if ((TWSR & 0xF8)!=0x08) Error();     //START has been not transmitted
}
```

Po vyslaní START podmienky, musí aplikácia nastaviť *Slave* adresu + bit zápisu (SLA + R) do dátového registra TWDR. Následne sa do riadiaceho registra TWCR zapíše pokyn na vyslanie *Slave* adresy na zbernicu.

- Príklad funkcie na vyslanie *Slave* adresy na zbernicu *data* = SLA+R

```
//*********************************************************************
void SendToTWI(char data)
{
  TWDR = data;                              //TWDR = SLA+R
  TWCR = (1<<TWINT)|(1<<TWEN);              //Transmit TWDR
  while (!(TWCR & (1<<TWINT)));             //Wait while TWINT = 1
}
```

Aplikačný softvér testuje hodnotu status registra TWSR, aby si potvrdil, že *Slave* adresa bola odoslaná a že hodnota bitu ACK bola podľa očakávania. Ak obsah TWSR0 je iný ako očakávaný, môže aplikačný softvér zavolať chybovú rutinu.

- Príklad funkcie na testovanie ACK potvrdenia od *Slave*

```
//*********************************************************************
void TestACK (void)
{
  if ((TWSR & 0xF8)!=0x40) Error();     //ACK has been not received
}
```

Ak je stav podľa očakávania, aplikácia načíta jeden bajt od Slave, ktorý bude v registri TWDR. Následne sa do riadiaceho registra TWCR zapíše pokyn na ukončenie načítavania zo Slave NACK.

- Príklad funkcie na načítanie jedného *data byte* od Slave s NACK

```
//*********************************************************************
char ReceiveTWI_Nack(void)
{
  TWCR = (1<<TWINT)|(1<<TWEN);
  while (!(TWCR & (1<<TWINT)));           //Wait while TWINT = 1
  return (TWDR);                          //Received byte
}
```

- Príklad funkcie na opakované načítanie *data byte* od Slave s ACK

```
//*********************************************************************
char ReceiveTWI_Ack(void)
{
  TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);  //ACK
  while (!(TWCR & (1<<TWINT)));           //Wait while TWINT = 1
  return (TWDR);                          //Received byte
}
```

- Príklad funkcie na testovanie NACK potvrdenia od *Master*

```
//*********************************************************************
void TestACK (void)
{
  if ((TWSR & 0xF8)!=0x58) Error();     //NACK has been not transmitted
}
```

- Príklad funkcie na testovanie ACK potvrdenia od *Master*

```
//*********************************************************************
void TestACK (void)
{
  if ((TWSR & 0xF8)!=0x50) Error();     //ACK has been not transmitted
}
```

- Príklad funkcie na ukončenie prenosu na zbernici

```
//*********************************************************************
void StopTWI (void)
{
  TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWSTO);      //STOP condition
}
```





# 13. Analógovo digitálny prevodník

Analógovo – digitálny (A/D) prevodník je elektronické zariadenie, ktoré realizuje digitalizáciu signálu a používa sa na meranie elektrických analógových veličín. A/D prevodník je charakterizovaný týmito parametrami:

- Napäťový rozsah – je rozsah napätia privedeného na vstup prevodníka, ktoré je schopný transformovať do číslicového tvaru,
- Rozlíšenie A/D prevodníka – je určené počtom rozlišovacích úrovní (v bitoch) analógového signálu,
- Čas prevodu – je určený počtom prevodov za sekundu, ktoré je A/D prevodník realizovať.

Súčasťou mikrokontroléra AVR je 10-bitovým A/D prevodníkom s postupnou aproximáciou. Vstup A/D prevodníka je pripojený k výstupu 8-kanálovému (6-kanalovému v PDIP puzdre) analógového multiplexeru, ktorý umožňuje pripojiť k A/D prevodníku 8 resp. 6 napäťových vstupov s 0V (GND) referenčným potenciálom. Taktiež je možné zvoliť na výstupe multiplexora interný teplotný senzor, interné referenčné napätie 1,1 V alebo nulové napätie GND (Obr. 88).

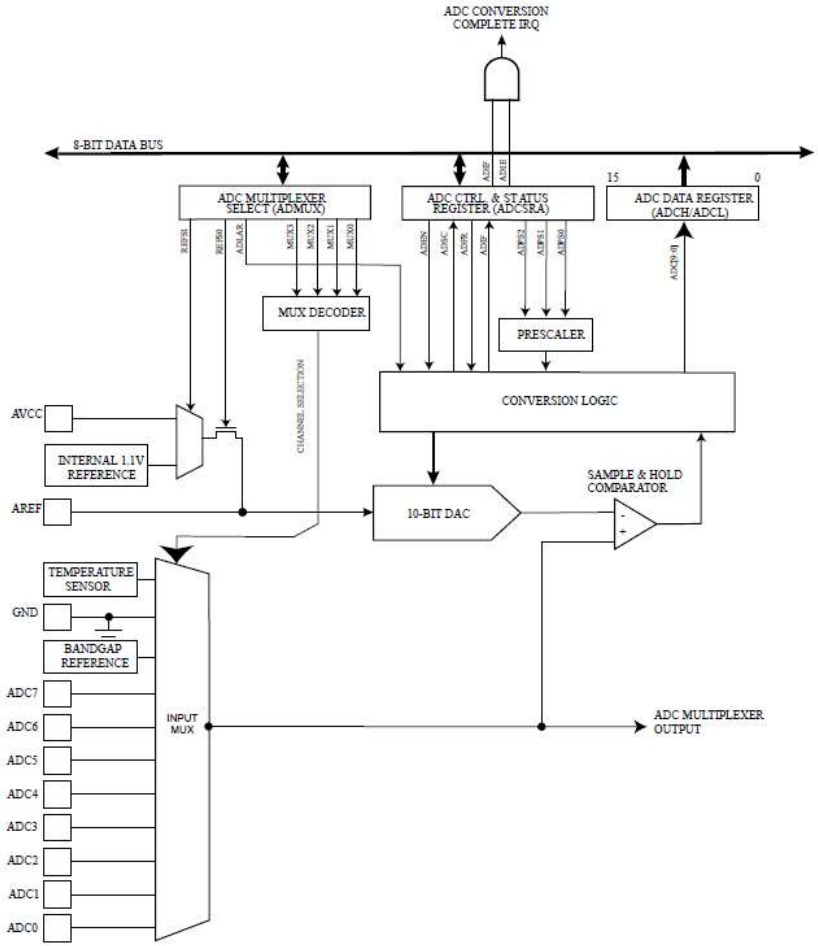
A/D prevodník obsahuje na vstupe obvod *Sample and Hold*, ktorý zabezpečí, že napätie na vstupe prevodníka sa nebude meniť počas doby konverzie na číslicový tvar. Ako referenčné napätie pre A/D prevod môže byť použité AVCC napätie, externé referenčné napätie pripojené na vstupe AREF, resp. je možné použiť vnútorné referenčné napätie 1,1V. Analógové napätie AVCC by sa nemalo líšiť od VCC o viac ako ± 0.3V (Obr. 88).

ADC má v SRAM pamäti vyhradené nasledujúce registre, ktoré potrebuje pre svoju činnosť:

- ADMUX (0x7C) – register výberu vstupu A/D prevodníka,
- ADCSRA (0x7A) – riadiaci a status register A prevodníka,
- ADCL (0x78), ADCH (0x79) - 16 bitový dátový register obsahuje výsledok A/D prevodníka,
- ADCSRB (0x7B) - riadiaci a status register B prevodníka,
- DIDR0 (0x7E) – register zákazu digitálnych vstupov.
- Výsledok A/D prevodu sa nachádza v registroch ADCH a ADCL. Ak sa požaduje 8-bitová presnosť stačí vyčítať obsah registra ADCH. Ak je požadovaná 10-bitová presnosť je potrebné najprv vyčítať obsah ADCL a potom ADCH, aby sa zabezpečilo, že sa obsah dátových registrov neprepíše výsledkom novej prevodu. Po načítaní obsahu ADCL je prístup A/D prevodníku do dátových registrov zablokovaný. To znamená, že ak sa prečíta ADCL a druhá konverzia skončí pred čítaním ADCH, žiadny register sa neaktualizuje a výsledok druhej konverzie sa stratí. Keď sa vyčíta ADCH, prístup A/D prevodníka do ADCH a ADCL registrov je znovu povolený. A/D

prevodník generuje prerušenie, ktoré sa môže použiť na vyčítanie dátových registrov okamžite po skončení konverzie.

- A/D prevod sa môže vykonať spustením z aplikácie alebo A/D prevodník môže byť v kontinuálnom režime (*Free Running mode*) (Tab. 21), nová konverzia sa spustí automaticky po skončení predchádzajúcej konverzie, pričom dochádza k prepisovaniu obsahu ADCH ADCL registrov.




Obr. 88 Bloková schéma A/D prevodníka





Tab. 21 Výber spúšťania A/D prevodu (ADCSRB.ADTS[2:0])

| ADTS[2:0] | Trigger Source |
|---|---|
| 000 | Free Running mode |
| 001 | Analog Comparator |
| 010 | External Interrupt Request 0 |
| 011 | Timer/Counter0 Compare Match A |
| 100 | Timer/Counter0 Overflow |
| 101 | Timer/Counter1 Compare Match B |
| 110 | Timer/Counter1 Overflow |
| 111 | Timer/Counter1 Capture Event |

Bežný A/D prevod trvá 13 ADC cyklov. Prvý A/D prevod po aktivovaní A/D prevodu trvá 25 cyklov ADC hodín, aby sa inicializovali príslušné analógové obvody. Ak nie sú napäťové pomery v analógových obvodoch stabilizované, hodnota vyčítaná po prvom A/D prevode môže byť nesprávna. Po dokončení prevodu sa výsledok zapíše do ADCL a ADCH registrov a nastaví sa príznak A/D prevodu ADCSRA.ADIF.

Tab. 22 Čas konverzie A/D prevodníka

| Condition | Sample & Hold (Cycles from Start of Conversion) | Conversion Time (Cycles) |
|---|---|---|
| First conversion | 13.5 | 25 |
| Normal conversions, single ended | 1.5 | 13 |
| Auto Triggered conversions | 2 | 13.5 |

Referenčné napätie určuje veľkosť napätia, ktoré je schopný A/D prevodník snímať a transformovať v rozsahu 10 bitov, teda čísla 0 do 1023. Ako referenčné napätie je možné zvoliť napätie AVCC, interný referenčný zdroj 1,1 V, alebo ako externý referenčné napätie pripojené na vývod AREF (

Tab. 23). AVCC je pripojené k A/D prevodníku prostredníctvom pasívneho prepínača, vnútorná referencia 1,1 V sa generuje prostredníctvom vnútorného zosilňovača. V oboch prípadoch je vhodné pripojiť k vývodu AREF kondenzátor za účelom zväčšenia šumovej imunity AREF a GNDA.

**Name:** ADMUX
**Offset:** 0x7C
**Reset:** 0x00
**Property:** -

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | REFS1 | REFS0 | ADLAR | | MUX3 | MUX2 | MUX1 | MUX0 |
| Access | R/W | R/W | R/W | | R/W | R/W | R/W | R/W |
| Reset | 0 | 0 | 0 | | 0 | 0 | 0 | 0 |

Tab. 23 Výber referenčného napätia a vstupného kanálu pre A/D prevodník (ADMUX.REFS[7:6], ADMUX.MUX[3:0])

| REFS[1:0] | Voltage Reference Selection |
|---|---|
| 00 | AREF, Internal $V_{ref}$ turned off |
| 01 | $AV_{CC}$ with external capacitor at AREF pin |
| 10 | Reserved |
| 11 | Internal 1.1V Voltage Reference with external capacitor at AREF pin |

| MUX[3:0] | Single Ended Input |
|---|---|
| 0000 | ADC0 |
| 0001 | ADC1 |
| 0010 | ADC2 |
| 0011 | ADC3 |
| 0100 | ADC4 |
| 0101 | ADC5 |
| 0110 | ADC6 |
| 0111 | ADC7 |
| 1000 | Temperature sensor |
| 1001 | Reserved |
| 1010 | Reserved |
| 1011 | Reserved |
| 1100 | Reserved |
| 1101 | Reserved |
| 1110 | 1.1V ($V_{BG}$) |
| 1111 | 0V (GND) |

Ak sa používa externý zdroj referenčného napätia na vývode AREF, nesmú sa už povoliť iné možnosti referenčného napätia, pretože sa skratuje s externým napätím. Ak nie je na vývod AREF pripojené žiadne externé napätie, môže sa prepínať medzi AVCC a 1,1 V ako zdrojom referenčného napätia.

- Funkcia *void Init_ADC()* inicializuje A/D prevodník s referenčný napätím VCC, možnosťou zadania čísla vstupu ADC a zakázaním digitálneho vstupu na ADC vstupoch multiplexera v režime *Free Running mode*.

```
//*************************************************************************
void Init_ADC (unsigned char channel)
{
  //Enable ADC, Division factor 128 16Mhz/128 = 125kHz
  ADCSRA |= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
  //AVcc Reference
  ADMUX = (1<<REFS0);
  //Set ADC Channel
  ADMUX |= channel;
  //Disable Digital Input on ADC Channel to reduce power consumption
  DIDR0 = 0x3F;
}
```

- Funkcia *void Init_ADC_isr()* inicializuje A/D prevodník s referenčný napätím VCC, možnosťou zadania čísla vstupu ADC a zakázaním digitálneho vstupu na ADC vstupoch multiplexera so softvérovým púšťaním prevodu a vyčítaním prevodu cez prerušenie.

```
//*************************************************************************
void Init_ADC_isr (unsigned char channel)
{
  ADCSRA |= (1<<ADEN);              //Enable ADC
  ADCSRA |= (1<<ADIE);              //Enable ADC Interrupt
```





```
  ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0) // Division factor 128
  ADMUX = (1<<REFS0);                        //AVcc Reference
  ADMUX |= channel;                          //Set ADC Channel
  DIDR0 = 0x3F;                   //Disable Digital Input on ADC Channel
}
```

- Jednoduchý príklad funkcie na spustenie A/D prevodu v programe.

```
//*************************************************************************
void Start_ADC (void)
{
  ADCSRA |= (1<<ADSC);
}
```

- Príklad obsluhy prerušenia A/D prevodníka po dokončení prevodu.

```
//*************************************************************************
void ISR_ADC (ADC_vect)
{
  napatie = (ADC * 5) / 1024;    //ADC * Vref / 1024
}
```

Prvý výsledok A/D prevodu po prepnutí zdroja referenčného napätia môže byť nepresný a je potrebné ho ignorovať. Výpočet A/D prevodu je možné urobiť podľa vzťahu:

$$U_{IN} = \frac{[(ADCH \ll 8) \mid ADCL] * U_{REF}}{1024} \tag{3.7}$$

Meranie teploty čipu je možné pomocou interného senzora teploty. Senzor teploty sa zvolí nastavením ADMUX.MUX[3:0] na hodnotu 0b1001. Ako referenčné napätie pre A/D prevod musí byť zvolené interné referenčné napätie 1,1 V. Namerané napätie má lineárnu závislosť na teplote približne 1mV / °C s presnosťou ± 10 ° C.

Tab. 24 Závislosť veľkosti napätia od teploty čipu ATmega328

| Temperature | -45°C | +25°C | +85°C |
|---|---|---|---|
| Voltage | 242mV | 314mV | 380mV |

Hodnoty v tabuľke (Tab. 24) sú typické hodnoty. Avšak kvôli odchýlkam vo výrobnom procese sa výstupné napätie senzora teploty líši od jedného čipu k druhému. Aby bolo možné dosiahnuť presnejšie výsledky, meranie teploty musí byť kalibrované v aplikačnom softvéri. Kalibrácia sa môže vykonať podľa nasledujúceho vzťahu:

$$T = \frac{[(ADCH \ll 8) \mid ADCL] - T_{OS}}{k} \tag{3.8}$$

- Príklad funkcie void *Init_ADC_temperature()*, ktorá inicializuje A/D prevodník s referenčný interným 1,1V

```
//*************************************************************************
```

```
void Init_ADC_temperature (void)
{
  //Enable ADC, Division factor 128 16Mhz/128 = 125kHz
  ADCSRA |= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
  //Internal 1.1V Reference
  ADMUX |= (1<<REFS1)|(1<<REFS0);
}
```

- Príklad funkcie *unsigned int Init_ADC_read_temperature(), ktorá* vráti 10 bitovú hodnotu teploty čipu

```
//*********************************************************************
unsigned int ADC_read_temperature (void)
{
  ADMUX &= 0xF0;
  ADMUX |= 8;
  ADCSRA |= (1<<ADSC);          //Start single A/D
  while(!(ADCSRA & (1<<ADIF))); //Wait for conversion to complete
  ADCSRA &= ~(1<<ADIF);         //Clear flag AD conversion completed
  return (ADC);                 //ADC = ADCH + ADCL
}
//*********************************************************************
```

- Príklad funkcie *unsigned int Init_ADC_mV_temperature(), ktorá* vráti hodnotu teploty čipu v mV

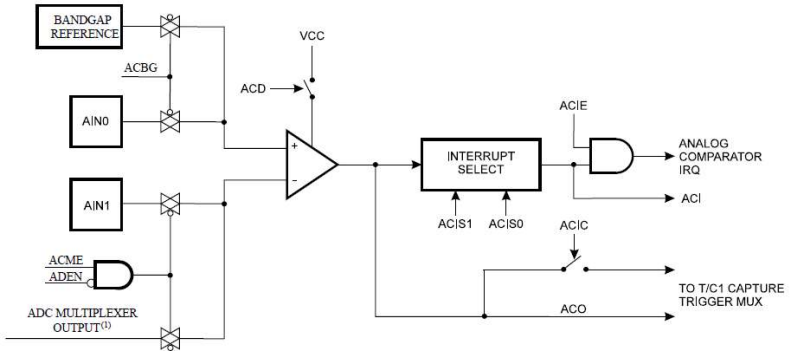
```
//*********************************************************************
unsigned int Init_ADC_mV_temperature()
{
  ADMUX &= 0xF0;
  ADMUX |= 8;
  ADCSRA |= (1<<ADSC);          //Start single A/D
  while(!(ADCSRA & (1<<ADIF))); //Wait for conversion to complete
  ADCSRA &= ~(1<<ADIF);         //Clear flag AD conversion completed
  return (ADC * 1100 / 1024);   //ADC * Vref / 1024 [mV]
}
```

## 13.1. Analógový komparátor

Analógový komparátor porovnáva vstupné hodnoty na kladnom vývode AIN0 a zápornom vývode AIN1 (Obr. 89) Keď je napätie na vývode AIN0 vyššie ako napätie na AIN1, nastaví sa výstup analógového komparátora ACO. Výstup komparátora možno nastaviť tak, aby spustil funkciu *Timer/Counter1 Input Capture*. Okrem toho môže komparátor spustiť samostatné prerušenie. Prerušenie môže byť generované pri vzostupe, poklese alebo prepnutí výstupu komparátora.








Obr. 89 Bloková schéma logiky AC komparátora

Záporný vstup komparátora AIN1 je možné nahradiť ľubovoľným z vývodov ADC[7:0]. Aby bolo možné túto náhradu realizovať, musí byť A/D prevodník vypnutý (ADCSRA.ADEN=0) a povolený bit aktivácie analógového komparačného multiplexora (ADCSRB.ACME=1). Tri najmenej významné bity výberu analógového kanála (ADMUX.MUX[2:0]) vyberajú vstupný vývod, ktorý nahradí záporný vstup analógového komparátora (Tab. 25). Ak ADCSRB.ACME=0 alebo ADCSRA.ADEN=1, na záporný vstup analógového komparátora sa použije AIN1.

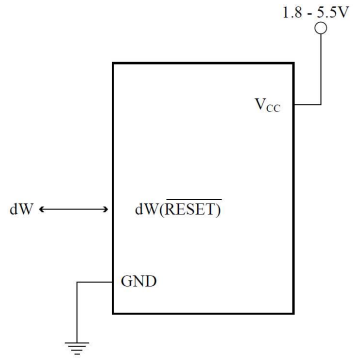
Tab. 25 Výber záporného vstupu AC komparátora (ADCSRB.ACME6, ADCSRA.ADEN7, ADMUX.MUX[2:0])

| ACME | ADEN | MUX[2:0] | Analog Comparator Negative Input |
|---|---|---|---|
| 0 | x | xxx | AIN1 |
| 1 | 1 | xxx | AIN1 |
| 1 | 0 | 000 | ADC0 |
| 1 | 0 | 001 | ADC1 |
| 1 | 0 | 010 | ADC2 |
| 1 | 0 | 011 | ADC3 |
| 1 | 0 | 100 | ADC4 |
| 1 | 0 | 101 | ADC5 |
| 1 | 0 | 110 | ADC6 |
| 1 | 0 | 111 | ADC7 |

# 14. *debugWIRE* – Systém ladenia programu na čipe MCU

*debugWIRE On-chip* ladiaci systém používa jeden vodič s obojsmerným rozhraním na riadenie chodu programu a vykonávanie inštrukcií AVR v CPU a na programovanie rôznych energeticky nezávislých pamätí. *debugWIRE* je navrhnutý ako jednoduchšia alternatíva k JTAG, zameraná na mikrokontroléry s obmedzenými zdrojmi (vývodmi). Podporujú ho klasické *ATtiny* a niektoré menšie *ATmega* MCU, ako napríklad ATmega328. *debugWIRE* umožňuje plný prístup na čítanie a zápis do celej pamäte a plnú kontrolu nad priebehom vykonávania programu. Podporuje jednokrokové, *Run-to-cursor*, *Step-out* inštrukcie pre prerušenie programu.

Ak je *FUSE* bit *debugWIRE Enable* (DWEN) nastavený na 0 a blokovacie *FUSE* bity sú neaktívne (1), v mikrokontroléry sa aktivuje systém *debugWIRE* a vývod /RESET je nakonfigurovaný ako obojsmerný I/O vývod s povoleným *Pull-up* rezistorom a stáva sa komunikačnou bránou medzi MCU a emulátorom (ATMEL-ICE).




Obr. 90 Schéma prepojenia /RESET vývodu MCU na *debugWIRE*

Pri navrhovaní systému, kde sa bude používať *debugWIRE*, je potrebné vziať do úvahy nasledujúce informácie:

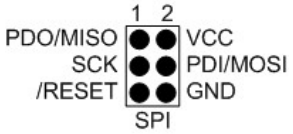
- *Pull-up* rezistory na vývode dW/(RESET) nesmú byť menšie ako 10kΩ. *Pull-up* rezistor nie je potrebný pre funkčnosť *debugWIRE*.
- Priame pripojenie vývodu /RESET k VCC nebude funkčné.
- Kondenzátory pripojené na vývod /RESET musia byť pri použití *debugWIRE* odpojené.
- Všetky externé zdroje resetovania musia byť odpojené.
- Obmedzený počet (10000) garantovaných cyklov zápisu do FLASH pamäte zariadenia.



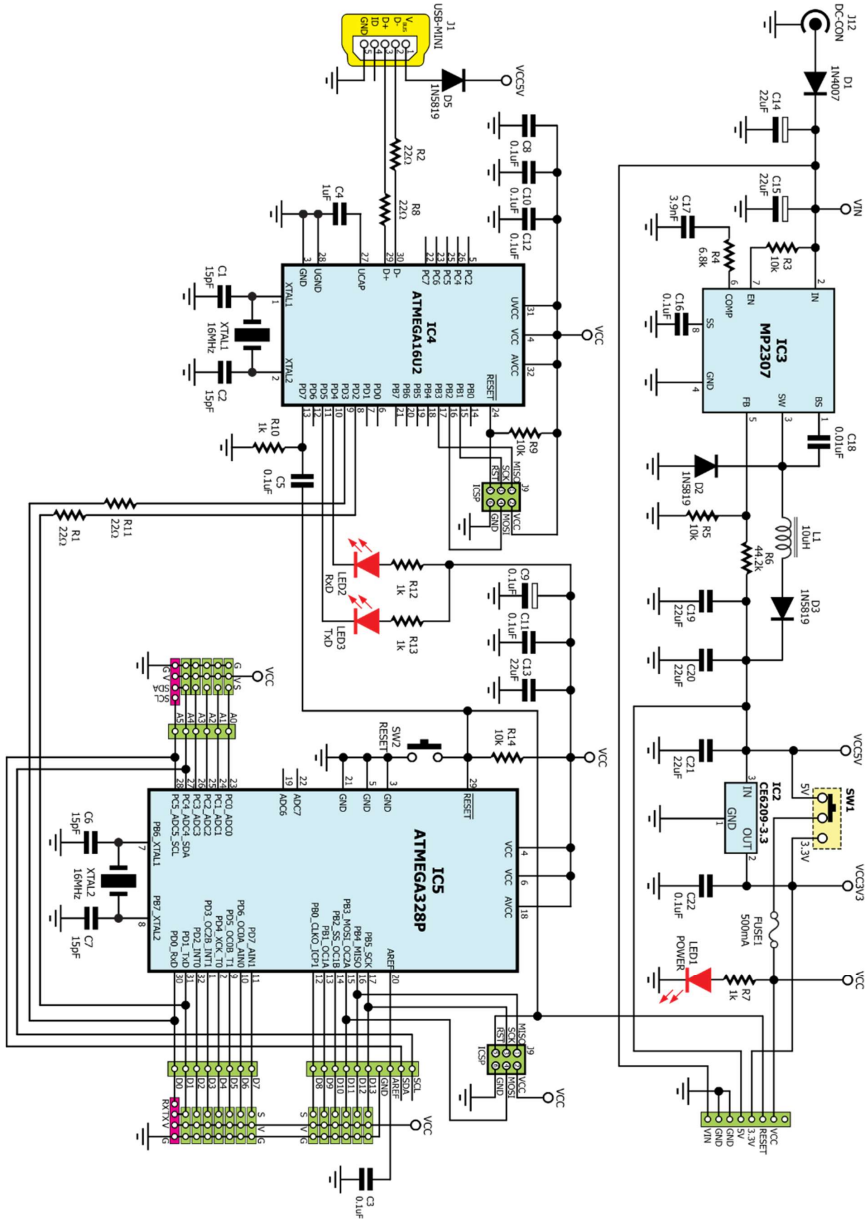


*debugWIRE* podporuje funkciu *Break Points* v pamäti programu pomocou inštrukcie AVR *Break*. Nastavením bodu prerušenia v *Microchip Studio* sa do pamäte programu vloží inštrukcia BREAK. Inštrukcia nahradená inštrukciou BREAK bude uložená. Keď pokračuje vykonávanie programu, uložená inštrukcia sa vykoná pred pokračovaním z pamäte programu. Prerušenie programu (*Break Point*) je možné vložiť manuálne vložením inštrukcie BREAK do programu. Programová *Flash* pamäť sa musí preprogramovať vždy, keď sa zmení *Break Point*. Toto automaticky rieši *Microchip Studio* cez rozhranie *debugWIRE*.

Vývod *debugWIRE* je fyzicky umiestnený na rovnakom vývode ako externý reset (/RESET). Zdroj externého resetovania preto nie je podporovaný, keď je povolený *debugWIRE*. Naprogramovaný *FUSE* bit DWEN umožňuje, aby niektoré časti systému hodín bežali vo všetkých režimoch spánku MCU. Tým sa zvýši spotreba energie v režime spánku. Preto by mala byť *FUSE* bit DWEN deaktivovaný, keď sa *debugWIRE* nepoužíva.




Obr. 91 Rozloženie signálov na SPI konektore (v prípade ladenia /RESET=dW)




Obr. 92 Elektrická schéma zapojenia vývojovej dosky Arduino UNO





# 15. Príklad 1

Príklad používania externých prerušení, komunikácie cez UART, nastavenie PWM na vývodoch OC0A a OC0B. Fungovanie programu je možné odskúšať pomocou programu *Terinalv1.9b.exe* alebo je možné použiť *Data Visualizer* z IDE *Microsoft Studio.*

```
#include <avr/io.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include <string.h>
#include <stdio.h>

#define BAUD        9600
#define BUFLEN      15
#define STX         2
#define ETX         3
#define CR          0x0D
#define LF          0x0A
#define TCNT0_SET   0x06
#define AVCC        4.8

char                    RxB[15], iRx, status;
volatile unsigned int   ticled, loop;

#define USARTRX         (status & (1<<0))       //Status USART receive
#define SET_USARTRX     (status |= (1<<0))
#define CLR_USARTRX     (status &= ~(1<<0))
#define BLINK           (status & (1<<1))       //Status LED
#define SET_BLINK       (status |= (1<<1))
#define CLR_BLINK       (status &= ~(1<<1))
#define OK              (status & (1<<2))       //Status USART command
#define SET_OK          (status |= (1<<2))
#define CLR_OK          (status &= ~(1<<2))
```

- Funkcia prerušenia sa vykonáva každé 4ms, blikanie led diódy, ak je Požadované a bude v 500ms intervaloch (4*125=500)

```
//***************************************************************************
ISR (TIMER0_OVF_vect) {
  loop++;
  if (BLINK) {
    if (++ticled == 125) {
      ticled = 0; PORTB ^= (1<<PB5); //TOG LED
    }
  }
  TCNT0 = TCNT0_SET;                  //4ms
}
```

- Funkcia prerušenia sa vykonáva po aktivácií INT0

```
//***************************************************************************
ISR (INT0_vect){
  PORTB |= (1<<PB5);                  //LED_ON;
}
```

- Funkcia prerušenia sa vykonáva po aktivácií INT1

```
//***************************************************************************
ISR (INT1_vect){
```

```
  PORTB &= ~(1<<PB5);                //LED_OFF;
}
```

- Funkcia prerušenia sa vykonáva po prijatí bajtu v UDR0 registry ukladá ho do buffera *RxB*. Súčasťou je kontrola riadiacich znakov (STX a ETX)a kontrola obsadenosti RxB buffera. Ak je prijatý znak STX, vynuluje sa index buffera iRx. Ak je prijatý znak ETX, znamená to ukončenie príjmu, čo sa signalizuje nastavením bitu SET_USARTRX.

```
//*****************************************************************************
ISR (USART_RX_vect){
  char temp;
  temp = UDR0;
  if (temp == STX) {iRx = 0; return;}
  if (iRx < BUFLEN) RxB[iRx++] = temp;
  if (temp == ETX) {RxB[--iRx]='\0'; SET_USARTRX;}
  return;
}
//*****************************************************************************
void USART_Transmit (char dataout)
{
  while(!(UCSR0A & (1 << UDRE0)));
  UDR0 = dataout;
}
```

- Vyslanie znakov cez USART. Znaky sú pripravené v bufferi *text*

```
//*****************************************************************************
void USART_Transmit_Text (char *text) {
  unsigned char i;
  for (i=0; i<strlen(text); i++) USART_Transmit (text[i]);
}
```

- Funkcia na čítanie výsledku prevodu AD prevodníka. V premennej *channel* je číslo vstupu AD prevodníka, *vref* obsahuje zdroj referenčného napätia (0:AREF, 1:AVCC, 3:1,1V).

```
//*****************************************************************************
unsigned int adc (unsigned char Channel, unsigned char Vref)
{
  ADMUX = (0x40 * Vref) + Channel;
  ADCSRA &= ~(1<<ADIF);
  ADCSRA |= (1<<ADSC);             //Start single A/D
  while(!(ADCSRA & (1<<ADIF)));
  return (ADC);
}
```

- Havná funkcia príkladu

```
//*****************************************************************************
int main (void) {

  unsigned int  n=0, bri;
  float         t=0.0, vref=0.0;
  char          text[50], duty;
  unsigned int  ubrr=F_CPU/16/BAUD-1;


  DDRB |= (1<<DDB5);               //Set output LED diode
  PORTB &= ~(1<<PB5);              //LED OFF
  DDRD |= (1<<DDD6) | (1<<DDD5);   //Set output PWM OC0A, OC0B
  PORTD |= (1<<PD2) | (1<<PD3);    //Pull-up on INT0, INT1

  EICRA = (1<<ISC01) | (1<<ISC11); // Trigger INT0, INT1 on failing edge
  EIFR = (1<<INTF0) | (1<<INTF1);  // Clear External INT0, INT1 Flag
```





```
  EIMSK = (1<<INT0) | (1<<INT1);   // Enable INT0, INT1

  UBRR0H = (unsigned char)(ubrr>>8);
  UBRR0L = (unsigned char) ubrr;
  UCSR0B = (1 << RXEN0) | (1 << TXEN0)  | (1 << RXCIE0);
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);

  TIFR0 |= 1<<TOV0;
  TIMSK0 |= 1<<TOIE0;
  TCNT0 = TCNT0_SET;
  TCCR0A |= (1<<COM0A1)|(1<<COM0A0);  //Set inverted mode OCR0A
  TCCR0A |= (1<<WGM01)|(1<<WGM00);    //Set fast PWM
  TCCR0A |= (1<<COM0B1)|(1<<COM0B0);  //Set inverted mode OCR0B
  TCCR0B |= (1<<CS02);                //Set prescaler to 256

  ADCSRA |= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
  DIDR0 = 0x3F;

  sei();
  wdt_enable(WDTO_500MS);             //WDT = 500ms, enable
  USART_Transmit_Text("\n\rPriklad 1\n\r");

  while(1) {
    wdt_reset();
    if (USARTRX) {
      CLR_USARTRX; CLR_OK;
      if (!strcmp (RxB, "ON")) {
        PORTB |= (1<<PB5); CLR_BLINK; SET_OK;
      }
      if (!strcmp (RxB, "OFF")) {
        PORTB &= ~(1<<PB5); CLR_BLINK; SET_OK;
      }
      if (!strcmp (RxB, "BLINK")) {
        if (BLINK) {CLR_BLINK; PORTB &= ~(1<<PB5);}
        else SET_BLINK;
        SET_OK;
      }
      //Set up PWM on output OC0B
      if ((RxB[0]=='P')&&(RxB[1]=='W')&&(RxB[2]=='M')&&(RxB[3]=='_')&&
          &&(RxB[4]=='S')) {
        SET_OK;
        duty = 100-((RxB[5]-'0')*100+(RxB[6]-'0')*10+RxB[7]-'0');
        OCR0B = TCNT0_SET + (255-TCNT0_SET) * duty / 100;
        sprintf (text, "\n\rOCR0B = 0x%02X %03d", OCR0B, duty);
        USART_Transmit_Text(text);
      }
      //Execute ADC on input (0-channel0, 8-temperature, R-reference,
      //G-Ground). Result in napatie
      if ((RxB[0]=='N')&&(RxB[1]=='a')&&(RxB[2]=='p')&&(RxB[3]==' ')) {
        SET_OK;
        switch (RxB[4]) {
          case '0':   n = adc(0, 1); vref=AVCC; break;
          case '8':   n = adc(8, 3); vref=1.1; break;
          case 'R':   n = adc(14, 3); vref=1.1; break;
          case 'G':   n = adc(15, 3); vref=1.1; break;
        }
        //Formatted output in text buffer
        sprintf (text, "\n\r%c: %04d %1.3fV", RxB[4],n,n*vref/1024.0);
        USART_Transmit_Text(text);
        if (RxB[4] == '8') {
          t = (float)n * 1100.0 / 1024.0;
```

```
            sprintf(text,"\n\rChip temp: %3.2fmV %2.1f°C ", t, t-363.0);
            USART_Transmit_Text(text);
          }
        }
        if (OK) USART_Transmit('+');
        else {
          USART_Transmit_Text("\n\r");
          USART_Transmit_Text(RxB);
          USART_Transmit('-');
        }
      }
      //LED brightness adjustment according to voltage of each 80ms
      if ((loop > 20) && (RxB[4] == '0')) {
        loop = 0;
        n = adc(0, 1);
        bri = (1023 - n)/10;
        if (bri > 100) bri = 100;
        if (bri < 1) bri = 1;
        OCR0A = TCNT0_SET + (255-TCNT0_SET) * bri / 100 ;
      }
    }
}
//*****************************************************************************
```



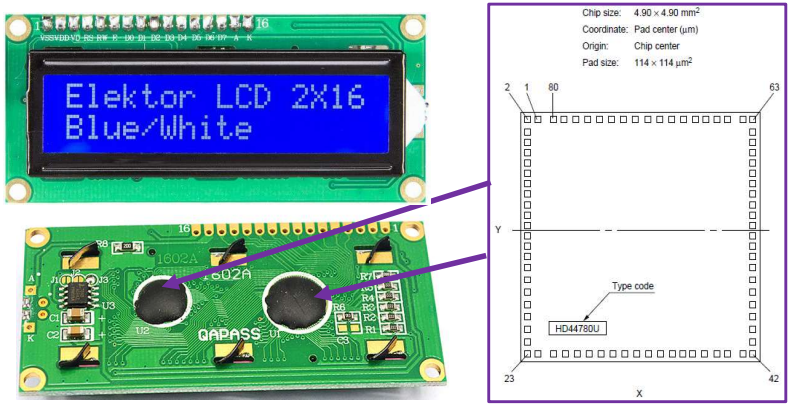


# 16. Príklad 2

Praktický príklad je zameraný na komunikáciu cez I2C zbernicu so znakovým displejom LCD s dvoma radičmi (každý pre jeden riadok). Moduly displejov tohto typu môžu obsahovať niektorý z týchto radičov HD44780U, AiP31066L, ST7066, KS0066 alebo iné. Radič obsahuje tabuľku kódov znakov (ASCII), ktoré vykresľuje z matice 5x8 bodov pre každý znak. Ovládanie modulu displeja je cez I2C I/O *Expander* PCF8574.

## 16.1. Modul displeja 16 znakov 2 riadky

Modul displeja sa skladá z 2-riadkového 16-znakového LCD maticového displeja, led podsvietenia displeja a dvoch radičov, každý pre jeden riadok. Radiče displeja a pomocné súčiastky sú umiestnené na spodnej strane modulu (Obr. 93).

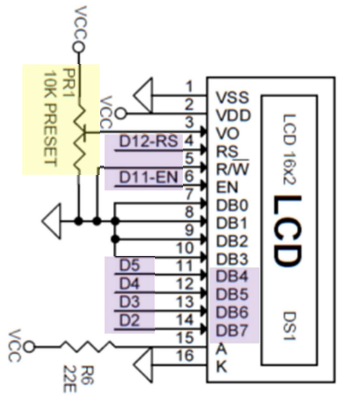



Obr. 93 LCD znakový modul displeja s 2x16 znakov

Modul displeja má 16 vývodový konektor so signálmi, ktoré sú potrebné pre komunikáciu s radičmi displeja (Obr. 93).

## 16.2. Dot Matrix LCD Controller/Driver HD44780U

Maticový ovládač LCD HD44780U môže byť nakonfigurovaný tak, aby s MCU komunikoval pomocou 4- alebo 8-vodičov. Ak má MCU k dispozícií dostatočný počet I/O vodičov je vhodné zvoliť 8-vodičovú komunikáciu, inak sa volí 4-vodičovú. Všetky funkcie, ako je RAM pamäť znakov displeja, generátor znakov a ovládač tekutých kryštálov, sú integrované na jednom čipe, nie sú potrebné ďalšie obvody. Jeden HD44780U môže zobraziť až jeden 8-znakový riadok alebo dva 8-znakové riadky. ROM generátor znakov HD44780U umožňuje generovať 208 5x8 bodových znakov a 32 5x10 bodových.




| | |
|---|---|
| VSS | *0V* |
| VDD | *+5V* |
| VO | *LCD contrast adjust* |
| RS | *Register Select* |
| R/W | *Read/Write Signal* |
| EN | *Enable Signal* |
| DB0 | --- |
| DB1 | --- |
| DB2 | --- |
| DB3 | --- |
| DB4 | *Data bit4* |
| DB5 | *Data bit5* |
| DB6 | *Data bit6* |
| DB7 | *Data bit7* |
| A | *Back Light +* |
| K | *Back Light -* |

Obr. 94 Popis signálov konektora LCD znakového modulu displeja s 2x16 znakov

Na module displeja sú použité dva obvody HD44780U, pričom 2x8 znakov vytvára jeden 16 znakový riadok, teda spolu sú k dispozícií 2 riadky po 16 znakov. Na komunikáciu sú použité 4 dátové vodiče a 3 riadiace signály. Bloková schéma radiča je na Obr. 96. Signály COM1 až COM16 označujú znak v riadku, SEG1 až SEG40 vytvárajú 5x8 maticu znaku.

**HD44780U**

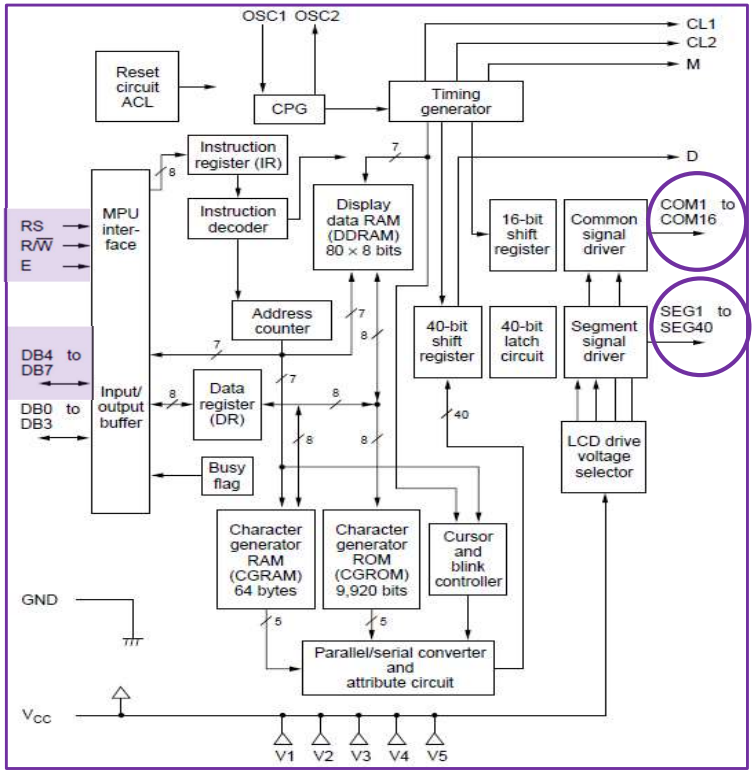
**Pin Functions**

| Signal | No. of Lines | I/O | Device Interfaced with | Function |
|---|---|---|---|---|
| RS | 1 | I | MPU | Selects registers.<br>0: Instruction register (for write) Busy flag: address counter (for read)<br>1: Data register (for write and read) |
| R/$\overline{W}$ | 1 | I | MPU | Selects read or write.<br>0: Write<br>1: Read |
| E | 1 | I | MPU | Starts data read/write. |
| DB4 to DB7 | 4 | I/O | MPU | Four high order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U. DB7 can be used as a busy flag. |
| DB0 to DB3 | 4 | I/O | MPU | Four low order bidirectional tristate data bus pins. Used for data transfer and receive between the MPU and the HD44780U.<br>These pins are not used during 4-bit operation. |

Obr. 95 Popis riadiacich signálov radiča HD44780U



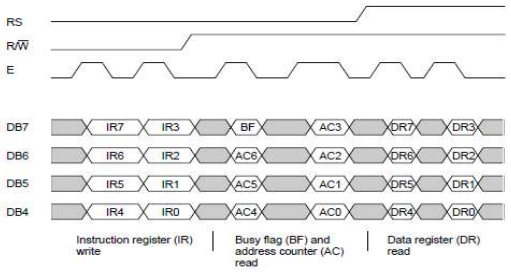





Obr. 96 Bloková schéma radiča HD44780U

Radič HD44780U používa dva 8-bitové registre: inštrukčný register IR a dátový register DR. Do IR sa ukladajú kódy inštrukcií, ako je vymazanie displeja, posun kurzora, adrese pre pamäť RAM displeja (DDRAM) a RAM generátora znakov (CGRAM). Do DR sa dočasne ukladajú dáta na zápis do DDRAM alebo CGRAM a dočasne sa ukladajú dáta na čítanie dát z DDRAM alebo CGRAM. Údaje zapísané do DR z mikroprocesora sa internou operáciou automaticky prepíšu do DDRAM alebo CGRAM. Najprv je potrebné definovať adresu DDRAM alebo CGRAM do IR a potom sa dáta postupne zapisujú cez DR do DDRAM alebo CGRAM.

Na Obr. 97 je zobrazený časový priebeh signálov pre 4-bitovú komunikáciu, pričom sa na prenos používajú iba štyri bity (DB4 až DB7). Prenos dát medzi HD44780U a MPU sa vykoná dvojnásobným prenosom 4-bitov. Ako prvé sa prenášajú bity vyššieho rádu (D4 až D7) a potom sa prenesú bity nižšieho rádu (D0 až D3). Po dvojnásobnom prenose 4-bitových dát je dobré skontrolovať *Busy Flag* (zaneprázdnenosť radiča).




Obr. 97 Časový diagram signálov pre 4-bitovú komunikáciu s radičom HD44780U

| Instruction | Code RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | Description | Execution Time (max) (when $f_{op}$ or $f_{osc}$ is 270 kHz) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Clear display | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Clears entire display and sets DDRAM address 0 in address counter. | |
| Return home | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | — | Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged. | 1.52 ms |
| Entry mode set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | Sets cursor move direction and specifies display shift. These operations are performed during data write and read. | 37 µs |
| Display on/off control | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B). | 37 µs |
| Cursor or display shift | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | — | — | Moves cursor and shifts display without changing DDRAM contents. | 37 µs |
| Function set | 0 | 0 | 0 | 0 | 1 | DL | N | F | — | — | Sets interface data length (DL), number of display lines (N), and character font (F). | 37 µs |
| Set CGRAM address | 0 | 0 | 0 | 1 | ACG | ACG | ACG | ACG | ACG | ACG | Sets CGRAM address. CGRAM data is sent and received after this setting. | 37 µs |
| Set DDRAM address | 0 | 0 | 1 | ADD | ADD | ADD | ADD | ADD | ADD | ADD | Sets DDRAM address. DDRAM data is sent and received after this setting. | 37 µs |
| Read busy flag & address | 0 | 1 | BF | AC | AC | AC | AC | AC | AC | AC | Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents. | 0 µs |
| Write data to CG or DDRAM | 1 | 0 | Write data | | | | | | | | Writes data into DDRAM or CGRAM. | 37 µs<br>$t_{ADD}$ = 4 µs* |
| Read data from CG or DDRAM | 1 | 1 | Read data | | | | | | | | Reads data from DDRAM or CGRAM. | 37 µs<br>$t_{ADD}$ = 4 µs* |
| | I/D = 1: Increment<br>I/D = 0: Decrement<br>S = 1: Accompanies display shift<br>S/C = 1: Display shift<br>S/C = 0: Cursor move<br>R/L = 1: Shift to the right<br>R/L = 0: Shift to the left<br>DL = 1: 8 bits, DL = 0: 4 bits<br>N = 1: 2 lines, N = 0: 1 line<br>F = 1: 5 × 10 dots, F = 0: 5 × 8 dots<br>BF = 1: Internally operating<br>BF = 0: Instructions acceptable | | | | | | | | | | DDRAM: Display data RAM<br>CGRAM: Character generator RAM<br>ACG: CGRAM address<br>ADD: DDRAM address (corresponds to cursor address)<br>AC: Address counter used for both DD and CGRAM addresses | Execution time changes when frequency changes<br>Example:<br>When $f_{op}$ or $f_{osc}$ is 250 kHz,<br>37 µs × 270/250 = 40 µs |

Note: — indicates no effect.
* After execution of the CGRAM/DDRAM data write or read instruction, the RAM address counter is incremented or decremented by 1. The RAM address counter is updated after the busy flag turns off. In Figure 10, $t_{ADD}$ is the time elapsed after the busy flag turns off until the address counter is updated.

Obr. 98 Inštrukcie radiča displeja HD44780 a čas ich vykonania

Interný resetovací obvod automaticky inicializuje HD44780U po zapnutí napájania. Počas inicializácie sa vykonajú nasledujúce pokyny. Príznak zaneprázdnenia (BF=1) je aktívny dovtedy,

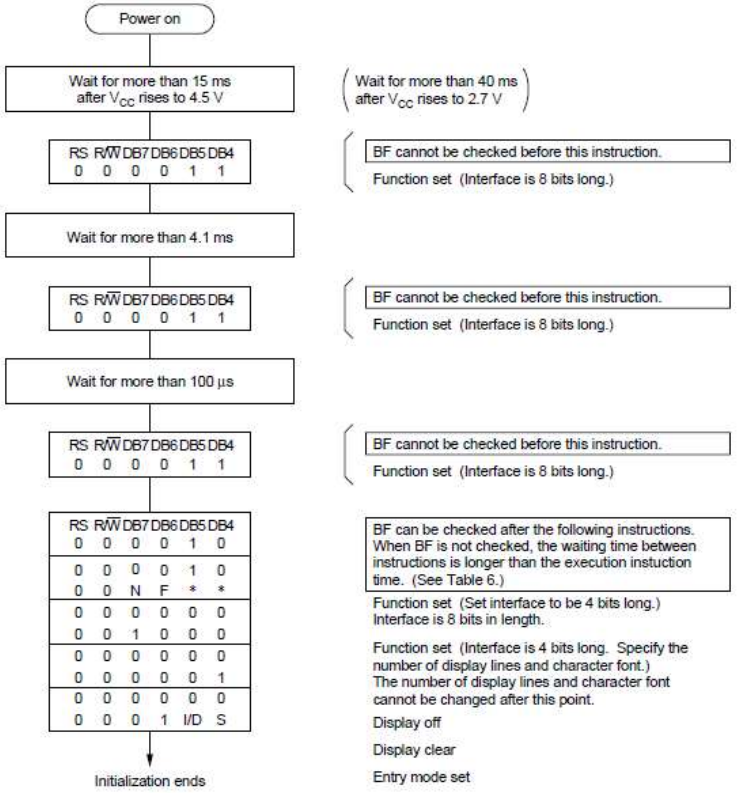




pokiaľ sa neskončí inicializácia. Tento stav trvá 10 ms potom, čo VCC stúpne na 4,5V. Inicializácia pozostáva z výkonu týchto inštrukcií:

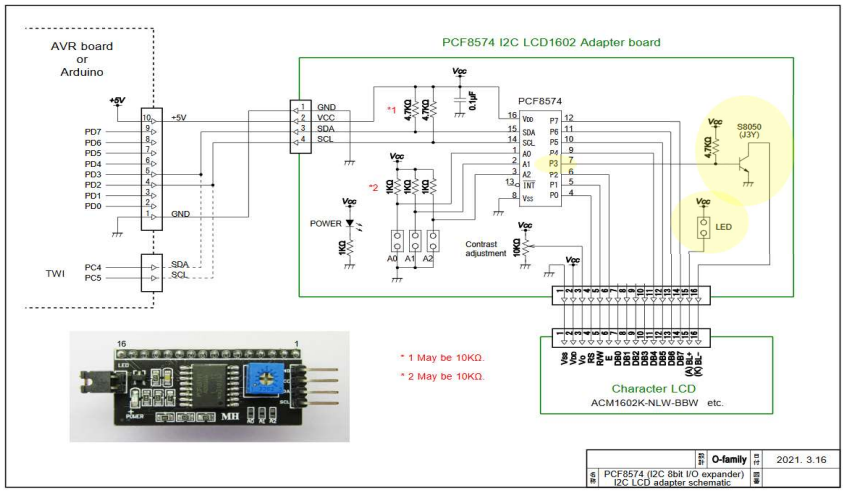
- *Clear display* - Vymazanie displeja,
- *Function set:* DL = 1 8-bitová komunikácia; N = 0 jeden riadok displeja aktívny; F = 0 veľkosť znaku 5x8 bodov
- *Display on/oFF control:* D = 0 Displej vypnutý; C = 0 Kurzor vypnutý; B = 0 Blikanie vypnuté
- *Entry mode set:* I/D = 1 Automatické inkrementovanie adresy pamäti o 1; S = 0 Žiaden posun

Pred používaním modulu displeja je potrebné vykonať inicializačnú sekvenciu na prepnutie radiča do komunikačného módu so 4 bitmi (D7, D6, D5, D4) (Obr. 99).




Obr. 99 Postup inicializácie 4-bitového komunikačného prenosu

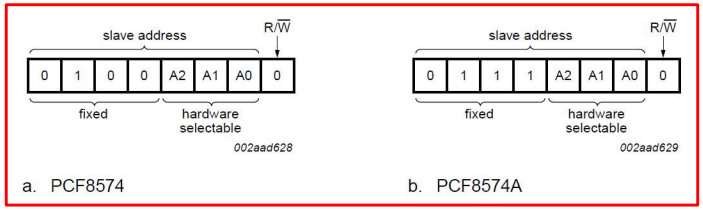
Ak je potrebné ešte znížiť počet vodičov pre pripojenie modulu displeja, je vhodné použiť I2C zbernicu a obvod PCF8574, ktorý poskytuje 8 I/O vodičov. Teda zo siedmich vodičov (D4,D5,D6,D7, RS,R/W,E) sa prepojenie zmenší na 2 vodiče (SCL, SDA), pričom PCF8574 bude vykonávať reálne prepojenie s displejom pomocou 7 vodičov (P0,P1,P2,P4,P5,P6,P7). Vodič I/O výstup P3 je použitý na zapínanie/vypínanie podsvietenia displeja (Obr. 100).




Obr. 100 Zapojenie I2C modulu s PCF8574 s modulom displeja

## 16.3. PCF8574/PCF8574A

Obvod PCF8574/74A realizuje rozšírenie I/O vývodov cez I2C zbernicu. Označenie vývodov obvodu a ich význam je zrejmý z blokovej schémy na Obr. 102. Obvody s označením PCF8574 a PCF8574A sú identické, s výnimkou odlišnej pevnej časti *Slave* adresy, ktorú nastavuje výrobca (Obr. 101). Užívateľ má k dispozícii 3 bity hardvérovej adresy umožňuje pripojenie ôsmych obvodov na I2C zbernicu, teda spolu môže byť na rovnakej zbernici I2C môže byť spolu až 16 I/O expandérov PCF8574/74A s podporou až 128 I/O vývodov.

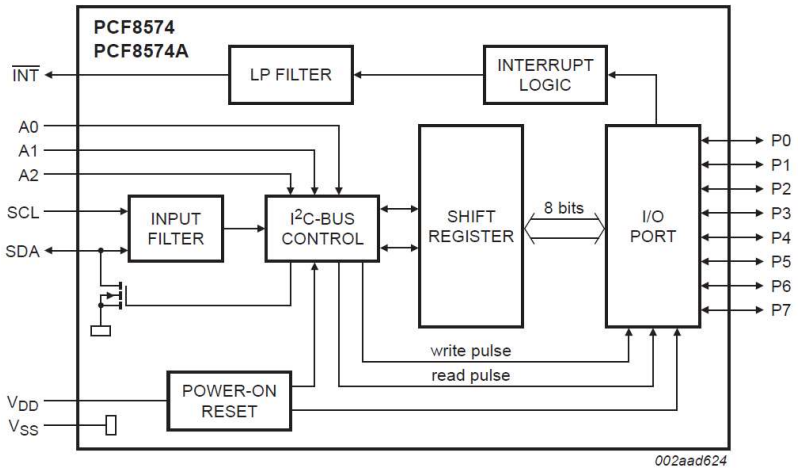



Obr. 101 Slave adresa obvodov PCF8574/74A



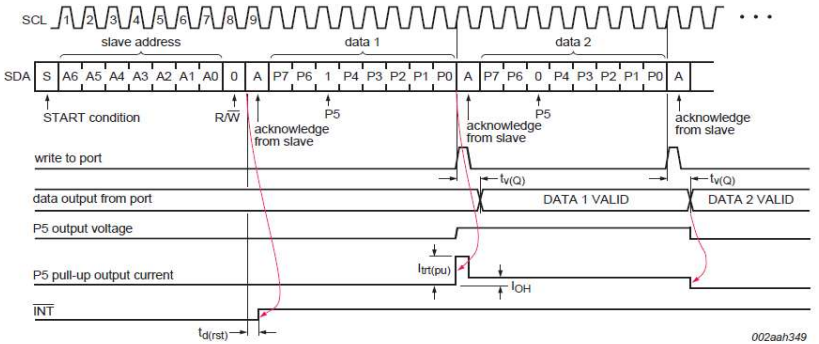


Modul expandéra má adresové bity nastavené: A2 = 1, A1 = 1, A0 = 1.




Obr. 102 Bloková schéma PCF8574/74A

Na nasledujúcom obrázku je zobrazený časový priebeh signálov obvodu a schéma komunikácie po I2C zbernici s PCF8574 v režime zápisu dát do obvodu. Režim čítanie z obvodu nie je uvedený, lebo do PCF8574 sa budú dáta len zapisovať a budú nastavovať I/O vývody expandéra tak, aby plnili funkciu rozhrania medzi I2C zbernicou a 4-bitovou komunikáciou s radičom HD44780U.




**Write mode (output)**

Simple code WRITE mode:

<S> <slave address + write> <**ACK**> <data out> <**ACK**> <data out> <**ACK**> ...
<data out> <**ACK**> <P>

Obr. 103 Časový diagram a schéma I2C komunikácie pre zápis do PCF8574/74A

Z prepojenia vývodov modulu displeja a I2c modulu s PCF8574 na Obr. 100 sú zrejmé nasledovné prepojenia signálov: RS = P0; R/W=P1; E=P2; DB4=P4; DB5=P5; DB6=P6; DB7=P7.

Vývody P0 až P7 obvodu PCF8574 simulujú signály 4-bitovej komunikácie. Úlohou programu pre MCU je zapisovať po I2C zbernici do PCF8574 dáta, ktoré budú ovládať modul displeja (HD44780U).

- Príklad zápisu bajtu do PCF8574/74A

```
#define PCF8574    0x4E //PCF8574A  0x7E
//*************************************************************************
void DataToPCF8574 (char data)
{
  starti2c();
  sendtoi2c(PCF8574); sendtoi2c(data);
  stopi2c();
}
```

- Príklad inicializácie HD44780U na 4-bitovú komunikáciu. Význam jednotlivých bitov: D7 D6 D5 D4 BL E RW RS. Zápisom bajtu do PCF8574 sa nastavujú komunikačné bity a súčasne riadiace signály.

```
//*************************************************************************
#define CLEAR_DISP    0x01      //Definitions of instruction codes HD44780U
#define RETURN_H      0x02
#define MODE_SET      0x04
#define DISP_ON_OFF   0x08
#define CUR_DISP      0x10
#define FUN_SET       0x20
#define SET_CGRAM     0x40
#define SET_DDRAM     0x80
void Lcd1602_Init(void)
{
  DataToPCF8574(0x00);
  _delay_ms(40);
  DataToPCF8574(0b00110100);   //D7=0,D6=0,D5=1,D4=1,BL=0,EN=1,RW=0,RS=0
  _delay_us(5);
  DataToPCF8574(0b00110000);   //D7=0,D6=0,D5=1,D4=1,BL=0,EN=0,RW=0,RS=0
  _delay_ms(5);
  DataToPCF8574(0b00110100);   //D7=0,D6=0,D5=1,D4=1,BL=0,EN=1,RW=0,RS=0
  _delay_us(5);
  DataToPCF8574(0b00110000);   //D7=0,D6=0,D5=1,D4=1,BL=0,EN=0,RW=0,RS=0
  _delay_us(100);
  DataToPCF8574(0b00110100);   //D7=0,D6=0,D5=1,D4=1,BL=0,EN=1,RW=0,RS=0
  _delay_us(5);
  DataToPCF8574(0b00110000);   //D7=0,D6=0,D5=1,D4=1,BL=0,EN=0,RW=0,RS=0
  _delay_us(100);
  DataToPCF8574(0b00100100);   //D7=0,D6=0,D5=1,D4=0,BL=0,EN=1,RW=0,RS=0
  _delay_us(5);
  DataToPCF8574(0b00100000);   //D7=0,D6=0,D5=1,D4=0,BL=0,EN=0,RW=0,RS=0
  _delay_us(50);

  Lcd_Cmd(FUN_SET|0x08);       //DL=0,N=1,F=0
  _delay_us(50);
  Lcd_Cmd(DISP_ON_OFF|0x04);   //D=1,C=1,B=0
  _delay_us(50);
  Lcd_Cmd(CLEAR_DISP);
  _delay_us(2000);
  Lcd_Cmd(MODE_SET|0x02);      //I/D=1
}
```





- Funkcia na vyslanie príkazu 4-bitovou komunikáciou do HD44780U

```
#define RS_H   0b00000001       //Signal edges generated by the software
#define RS_L   0b11111110
#define RW_H   0b00000010
#define RW_L   0b11111101
#define EN_H   0b00000100
#define EN_L   0b11111011
#define BL_H   0b00001000
#define BL_L   0b11110111
//*************************************************************************
void Lcd_Cmd (unsigned char cmd)
{
  unsigned char lcd_data;

  lcd_data = (cmd & 0xF0);          //Set D7,D6,D5,D4 bits of command
  lcd_data &= RS_L;                 //Set RS=LOW,
  lcd_data |= (EN_H | BL_H);        //Set EN=HIGH, BL_H=HIGH
  lcd_data &= RW_L;                 //Set RW=LOW,
  DataToPCF8574 (lcd_data);         //Write lcd_data to PCF8574
  lcd_data &= EN_L;                 //Set EN=LOW
  DataToPCF8574 (lcd_data);         //Write lcd_data to PCF8574

  lcd_data = ((cmd & 0x0F)*0x10);   //Set D3,D2,D1,D0 bits of command
  lcd_data &= RS_L;
  lcd_data |= (EN_H | BL_H);
  lcd_data &= RW_L;
  DataToPCF8574(lcd_data);
  lcd_data &= EN_L;
  DataToPCF8574(lcd_data);
}
```

- Funkcia na vyslanie znaku 4-bitovou komunikáciou do HD44780U

```
//*************************************************************************
void Lcd_Char(unsigned char chr)
{
  unsigned char lcd_data;

  lcd_data = (chr & 0xF0);          //Set D7,D6,D5,D4 bits of chr
  lcd_data |= (RS_H | EN_H | BL_H);//Set RS=HIGH, EN=HIGH, BL_H=HIGH
  lcd_data &= RW_L;                 //Set RW=LOW,
  DataToPCF8574(lcd_data);          //Write lcd_data to PCF8574
  lcd_data &= EN_L;                 //Set EN=LOW;
  DataToPCF8574(lcd_data);          //Write lcd_data to PCF8574

  lcd_data = ((chr & 0x0F)*0x10);   //Set D3,D2,D1,D0 bits of chr
  lcd_data |= (RS_H | EN_H | BL_H);//Set RS=HIGH, EN=HIGH, BL_H=HIGH
  lcd_data &= RW_L;                 //Set RW=LOW,
  DataToPCF8574(lcd_data);          //Write lcd_data to PCF8574
  lcd_data &= EN_L;                 //Set EN=LOW;
  DataToPCF8574(lcd_data);          //Write lcd_data to PCF8574
}
```

- Funkcia na zápis textu na displej do riadku (1,2) a stĺpca (0 až 15)

```
//*************************************************************************
void Lcd1602_Out(unsigned char row, unsigned char column, char *str)
{
  unsigned char i;

  switch(row)
  {
    case 1:   row = 0x00; break;        //Set address 1 row
```

```
    case 2:   row = 0x40; break;        //Set address 2 row
  }
  Lcd_Cmd(SET_DDRAM | row | column);    //Set DDRAM address

  for (i=0; i< strlen(str); i++) {
    Lcd_Char(str[i]);                   //Write char to HD44780U
  }
}
```

- Hlavný program príkladu 2

```
//*****************************************************************************
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>
#include <string.h>
#include <stdio.h>
#include "I2CUtility.h"

#define BAUD       9600
#define RXBUFLEN   25
#define STX        2
#define ETX        3
#define HODINY     0
#define ASCII      1
#define COMMAND    7
#define PCF8574    0x4E //0x7E

char                    buffer[50], znak;
unsigned char           status;
volatile unsigned char  i;
volatile unsigned int   msec4, count = 0;

typedef struct{
  unsigned char second;
  unsigned char minute;
  unsigned char hour;
  unsigned char date;
  unsigned char month;
  unsigned char weekday;
}time;
time t;

void Lcd1602_Init (void);
void Lcd1602_Out (unsigned char row, unsigned char column, char *str);
void USART_Transmit_Text(char *text);

//*****************************************************************************
void Hodiny (void)
{
  if(++t.second==60) {
    t.second=0; if (++t.minute==60) {
      t.minute=0; if (++t.hour==24) {
        t.hour=0;
        t.weekday++;
        if(++t.date==32) {
          t.month++;
          t.date=1;
        }
        else if(t.date==31) {
```





```
          if((t.month==4)||(t.month==6)||(t.month==9)||(t.month==11)) {
            t.month++;
            t.date=1;
          }
        }
        else if(t.date==30) {
          if(t.month==2) {
            t.month++;
            t.date=1;
          }
        }
        else if(t.date==29) {
          if(t.month==2) {
            t.month++;
            t.date=1;
          }
        }
        if(t.month==13) t.month=1;
      }
    }
  }
}

//*****************************************************************************
ISR (TIMER0_OVF_vect) { //4ms
  msec4++;
  if (msec4 == 250) {
    msec4 = 0;
    Hodiny();
    PORTB ^= (1<<PB5);
    if (status & 0x01) {
      sprintf(buffer,"%02d.%02d. %02d:%02d:%02d ", t.date, t.month,
                     t.hour, t.minute, t.second);
    }
    if (status & 0x02) {
      sprintf(buffer,"%03d  %c%c%c     ", znak, znak, znak, znak);
      znak++;
    }
    if (status & 0x03) Lcd1602_Out(1,0,buffer);
    ADCSRA |= (1<<ADSC);           //Start single A/D
    while(!(ADCSRA & (1<<ADIF)));
    ADCSRA &= ~(1<<ADIF);
    sprintf (buffer, "%2.1f°C", ((float)ADC*1100.0/1024.0) -353.0);
    Lcd1602_Out(2,12,buffer);
  }
  TCNT0 = 6;
}

//*****************************************************************************
ISR (USART_RX_vect)
{
  char temp, RxBuf[RXBUFLEN];

  temp = UDR0;
  if (temp == STX) {i = 0; return;}
  if (i < (RXBUFLEN-1)) RxBuf[i++] = temp;
  if (temp == ETX) {
    RxBuf[--i]='\0';
    USART_Transmit_Text(RxBuf);
    status |= (1<<COMMAND);
    switch (RxBuf[0]) {
```

```
      case 'h':   status |= (1<<HODINY);
                  status &= ~(1<<ASCII);
                  break;
      case 'a':   status |= (1<<ASCII);
                  status &= ~(1<<HODINY);
                  break;
      case 's':   t.date = (RxBuf[1]-'0')*10 + RxBuf[2]-'0';
                  t.month = (RxBuf[4]-'0')*10 + RxBuf[5]-'0';
                  t.hour = (RxBuf[7]-'0')*10 + RxBuf[8]-'0';
                  t.minute = (RxBuf[10]-'0')*10 + RxBuf[11]-'0';
                  t.second = (RxBuf[13]-'0')*10 + RxBuf[14]-'0';
                  break;
      default:    status &= ~(1<<COMMAND); break;
    }
    if (status & 0x80) strcpy (buffer, "+\n\r");
    else  strcpy (buffer, "?\n\r");
    USART_Transmit_Text(buffer);
  }
  return;
}

//***************************************************************************
ISR (INT0_vect)
{
  count++;
  EIFR |= 1<<INTF0;
}

//***************************************************************************
ISR (INT1_vect)
{
  count--;
  EIFR |= 1<<INTF1;
}

//***************************************************************************
int main (void)
{
  unsigned int     ubrr = F_CPU/16/BAUD-1, count_temp=0;

  DDRB |= 1<<DDB5;

  UBRR0H = (unsigned char)(ubrr>>8);
  UBRR0L = (unsigned char) ubrr;
  UCSR0B = (1 << RXEN0) | (1 << TXEN0)  | (1 << RXCIE0);
  UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);

  TIFR0 |= 1<<TOV0;
  TIMSK0 |= 1<<TOIE0;
  TCCR0B |= (1<<CS02);
  TCNT0 = 6;                //Init TOV0 interrupt 4ms

  TWBR = 0x48;             //Set bit rate register. 100KHz CLK
  TWDR = 0xFF;             //Default content = SDA released.
  TWCR |= (1<<TWEN);       //Enable TWI-interface and release TWI pins

  ADMUX = 8;               //Chip temeperature
  ADMUX |= 0xC0;           //1.1V
  ADCSRA |= (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);

  EIMSK |= (1<<INT0) | (1<<INT1);       //Enable INT0, INT1
```





```
  EICRA |= (1<<ISC01) | (1<<ISC11);     //Trigger on failing edge
  EIFR  |= (1<<INTF0) | (1<<INTF1);
  PORTD |= (1<<PD2) | (1<<PD3);         //Pull-up on INT0, INT1

  sei();

  Lcd1602_Init();
  wdt_enable(WDTO_2S);
  status = 1<<HODINY;

  while(1) {
    wdt_reset();
    if (count != count_temp) {
      count_temp = count;
      sprintf (buffer, "%04d", count);
      Lcd1602_Out(2,0,buffer);
    }
  }
}
//***************************************************************************
```

## Zoznam použitej literatúry


[1]. Dell, J. A.: Digital interface design and application. ISBN:9781118974322, 2015.

[2]. Barrett, S. F.: Embedded Systems Design with the Atmel AVR Microcontroller Part I. ISBN: 9781608451289, 2010.

[3]. Barrett, S. F.: Embedded Systems Design with the Atmel AVR Microcontroller Part II. ISBN: 9781608453200, 2010.

[4]. Russell, D. J.: Introduction to Embedded Systems Using ANSI C and the Arduino Development Environment. ISBN:978160845990, 2010.

[5]. Prauzek, M. :Číslicová a mikroprocesorová technika. ISBN: 9788024814970, 2007.

[6]. Kesl, J.: Elektronika III, číslicová technika. ISBN: 8073001829, 2005.

[7]. Shuqin Lou, Chunling Yang: Digital Electronic Circuits

[8]. Galajda, P.; Lukáč, R.: Elektronické obvody. ISBN:8089061591, 2002.

[9]. Drutarovský, M.: Jednočípové mikropočítače a jazyk C. ISBN: 9788055325392, 2016.

[10]. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-microcontroller-ATmega328-328P_Datasheet.pdf.

[11]. http://physics.mff.cuni.cz/kfpp/skripta/elektronika/

[12]. http://www.kiwiki.info/index.php/Architekt%C3%BAra_mikrokontrol%C3%A9ra

[13]. https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/INTPAM.HTML

[14]. http://www.elektromys.eu/clanky/avr_timer1/clanek.html.

[15]. http://microchipdeveloper.com/8avr:avrtimer

[16]. http://www.avr-tutorials.com/

[17]. http://www.ermicro.com/blog/?p=1971

[18]. https://matika.elf.stuba.sk/KMAT/LogickeSystemy/ParalelkaC?action=AttachFile&do=get&target=LSprednaskyMika.doc.

[19]. https://www.electronicwings.com/avr-atmega/atmega1632-clear-timer-on-compare-match-ctc-mode

[20]. https://www.microchip.com/en-us/tools-resources/configure/mplab-code-configurator

[21]. https://github.com/avrdudes/avrdude

[22]. https://start.atmel.com/

[23]. https://www.microchip.com/en-us/tools-resources/develop/microchip-studio

[24]. https://www.electronicwings.com/