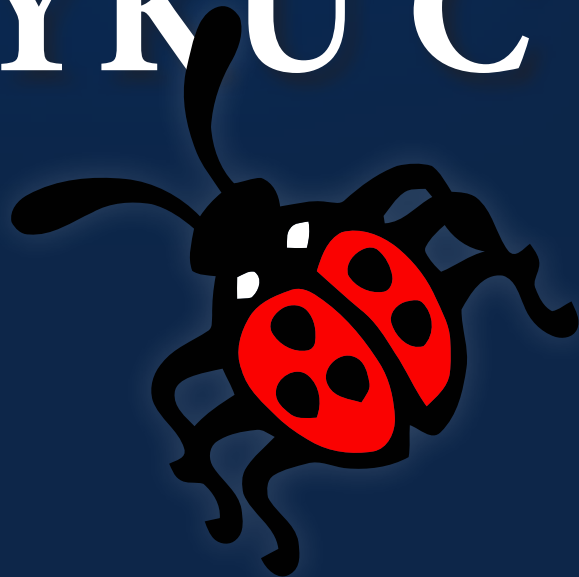


PROGRAMUJEME  
**AVR**  
V JAZYKU C



Bc. Ondrej Závodský

Zoznámte sa s MCU  
rady Atmel AVR



# PROGRAMUJEME AVR V JAZYKU C

**Bc. Ondrej Závodský**

**Túto knižku možno používať a rozširovať na nekomerčné účely bezplatne.**

Ak chcete prejaviť autorovi vďaku a uznanie za čas strávený napísaním tejto knižky, alebo prípadne usúdite, že vám táto knižka pomohla a ušetrila veľa práce a času, môžete prispieť ľubovoľnou čiastkou na číslo účtu: 0263746912/0900

Ďakujem!

© 2012 Bc. Ondrej Závodský - [zawin@svetelektro.com](mailto:zawin@svetelektro.com)

# OBSAH

Úvod.....	5
Mikropočítače Atmel AVR.....	6
Práca v AVR STUDIO 4 .....	7
Píšeme prvý program.....	10
Prerušená.....	16
Čítače/časovače.....	19
Čítače/Časovače - PWM .....	26
Práca so znakovým LCD displ.....	33
AD prevodníky .....	37
Rozhranie USART .....	42
Rozhranie SPI .....	49
Rozhranie TWI.....	53
Režimy spánku MCU.....	61
Použitá literatúra .....	65

# ÚVOD

Táto knižka vznikla spojením jednotlivých častí seriálu o programovaní AVR v jazyku C, ktoré publikujem na webe svetelektro.com. Knižka Vás najskôr zoznámi s mikropočítačmi Atmel AVR a programom AVR Studio 4, v ktorom budeme písať naše programy. V prvých kapitolách bude vysvetlená práca s registrami, I/O portami a obsluhou pamäti. V ďalších kapitolách budú bližšie vysvetlené jednotlivé periférie mikropočítača. Ku každej kapitole sa nachádza množstvo jednoduchých príkladov pre jednoduchšie pochopenie danej činnosti. Príklady sú testované na mikropočítači ATmega8, budú však fungovať aj na ATmega16 a 32. Schémy zapojenia sa však budú líšiť.

Knižka je orientovaná hlavne na popis práce s jednotlivými perifériami mikropočítača, nevenuje sa v nej pozornosť vysvetleniu programovacieho jazyka C. Preto predtým ako sa pustíte do čítania tejto knihy, odporúčam získať aspoň základné vedomosti o programovacom jazyku C. Na to Vám môže poslúžiť napr. kniha od Pavla Herouta - Učebnica jazyka C

## **Podakovanie:**

Chcem sa poďakovať všetkým, čo sa akýmkoľvek spôsobom pričínili o pomoc pri tvorbe článkov na webe svetelektro.com o programovaní AVR v jazyku C, pretože bez Vás by táto knižka nemohla existovať v takej podobe ako je teraz. Osobitne by som sa chcel poďakovať kamarátom Awen-ovi - ktorý mi každú časť seriálu podrobne skontroloval a opravil a Lubovi (luboss17) - ktorý mi pomohol s tvorbou knižníc.

# MIKROPOČÍTAČE ATMEL AVR

V roku 1997 uviedla na trh firma Atmel nové osembi-  
tové mikropočítače rady AVR.

Oproti predošlému jadrú 8051 nastalo viacero zmien. Mikropočítač Atmel AVR začal využívať architektúru RISC (Reduced instruction set computing), čo prinieslo značne zvýšenie výkonu. Taktiež oproti mikroprocesorom 8051 nastalo zväčšenie šírky inštrukčného slova na 16 bitov. To umožnilo zrýchlenie načítavania inštrukcií, kde až na niekoľko výnimiek dochádza k ich načítaniu v priebehu jedného strojového cyklu.

Mikropočítač AVR je optimalizovaný pre programovanie v jazyku C.

## Architektúra mikropočítača Atmel AVR

V tomto mikropočítači sa nachádzajú tri druhy pamäte:

- Pamäť FLASH – má šírku 16 bitov a využíva sa na uloženie programu, dá sa zmazať a opätovne zapísať, pričom výrobca garantuje až 10 000 cyklov zmazanie/zápis. Dá sa programovať sériovo, čo zjednodušilo nahrávanie programu z PC do mikropočítača.
- Pamäť RAM – má šírku 8 bitov, je rýchla a slúži na dočasné ukladanie výsledkov a dát. Po odpojení napätia sa zmaže.
- Pamäť EEPROM – slúži na ukladanie dát, ktoré sa uchovávajú aj po odpojení napájacieho napätia.

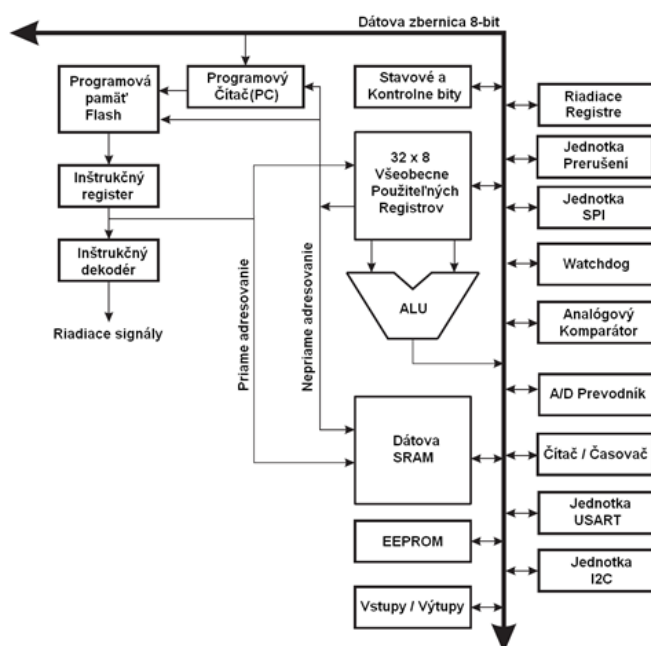
Mikropočítače AVR taktiež ponúkajú bohatú výbavu čo sa periférii týka sú to napríklad:

- Čítač/časovač s možnosťou PWM modulácie
- AD prevodník
- SPI, TWI, USART
- Watchdog
- A iné...

Týmto perifériám sa budeme podrobnejšie venovať v ďalších kapitolách knihy.

Mikropočítač podporuje taktiež aj viacero zdrojov hodinového signálu:

- externý kryštál / rezonátor
- externý nízko-frekvenčný kryštál (32,768 kHz)
- externý RC oscilátor
- interný kalibrovaný RC oscilátor



Obr 1.0 - bloková schéma mikropočítača ATmega8

# PRÁCA V AVR STUDIO 4

V tejto kapitole si ukážeme ako možno vytvoriť program v jazyku C a prostredí AVR Studio 4, v ktorom budeme písať všetky ďalšie programy. Upozorňujem čitateľov že je potrebná aspoň základná znalosť jazyka C.

## Inštalácia AVR Studia 4

Z oficiálnej stránky Atmel-u stiahneme súbory:

- AVR Studio 4
- AVR Toolchain 3.4 (pre 8-bitové MCU)

Odkaz:

<http://www.atmel.com/tools/studioarchive.aspx>

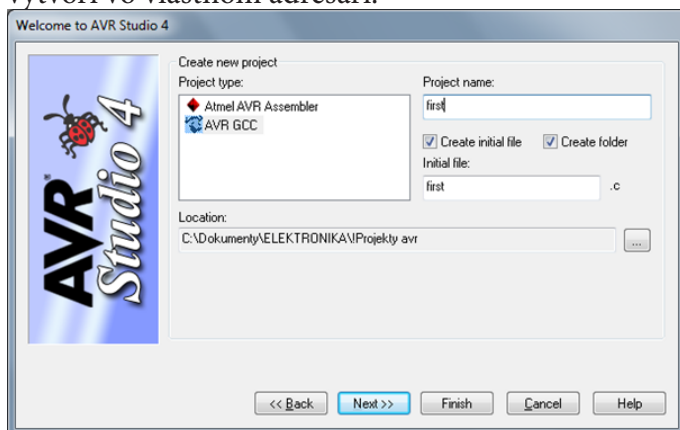
Pred sťahovaním treba vyplniť registračné údaje aby bolo možné súbory stiahnuť.

Nainštalujeme teda AVR Studio 4 a následne AVR Toolchain 3.4. AVR studio 4 je pracovné prostredie na vývoj programu, AVR toolchain 3.4 je compiler, linker a zjednodušená verzia stdlib jazyka C. Po nainštalovaní spustíme program AVR Studio a skúsime vytvoriť náš prvý projekt.

## Vytvorenie projektu

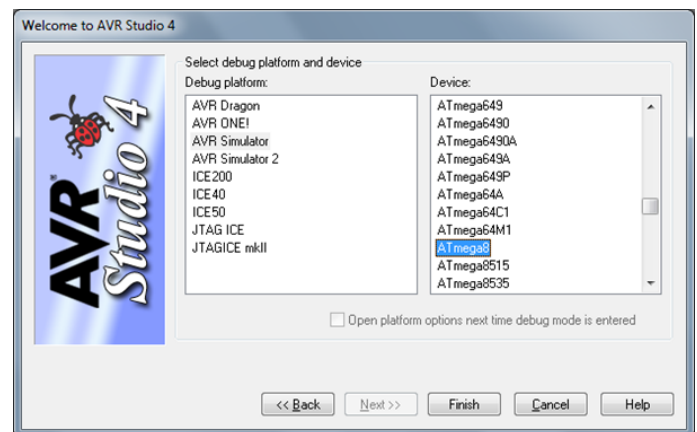
Po spustení programu AVR studio sa objaví welcome obrazovka kde zvolíme možnosť "Create new project".

Keďže programovať ideme v jazyku C, zvolíme možnosť AVR GCC. Napíšeme názov projektu a určíme pracovný adresár projektu. Odporúčam zaškrtnúť možnosť "Create folder", ktorá spôsobí že projekt sa vytvorí vo vlastnom adresári.



Obr 2.0 - Vytvorenie projektu - AVR Studio 4

Na ďalšej obrazovke si zvolíme debugovaciu platformu – čiže ako chceme program ladiť. Pokiaľ nemáte JTAG programátor zvolte možnosť AVR simulátor. Dôležité je vpravo vybrať správny mikropočítač. Nakoniec klikneme na tlačidlo "Finish" a tým vytvoríme nový projekt.



Obr 2.1 - Vytvorenie projektu - AVR Studio 4

## Popis pracovného prostredia

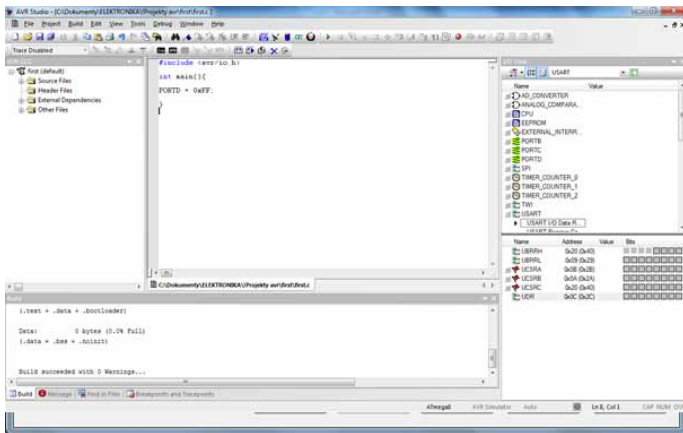
Po vytvorení projektu môžeme písať samotný projekt v jazyku C. Naľavo sa nachádza rozbaľovacia ponuka, kde si môžeme zvoliť zdrojové a hlavičkové súbory, ktoré následne cez direktívu #include môžeme pridať do projektu. Slúži to na lepší prehľad v projekte. Napr. jeden zdrojový súbor bude slúžiť na komunikáciu s UART, ďalší na komunikáciu s EEPROM a v hlavnom súbore ich potom pridáme.

V strede sa nachádza editor, v ktorom píšeme program. Nevýhodou je však to, že je to len holý editor s veľmi základným zvýraznením syntaxe, takže treba dbať na väčšiu pozornosť pri písaní programu.

Napravo sú ovládacie registre procesora, ktoré môžeme počas debugovania nastavovať alebo sledovať počas behu programu ako sa nastavili. Činnosť a funkcia registrov bude podrobnejšie vysvetlená v ďalšej kapitole.

Naspedu sa nachádza okno, ktoré nás informuje o úspešnom, alebo neúspešnom skompilovaní projektu.

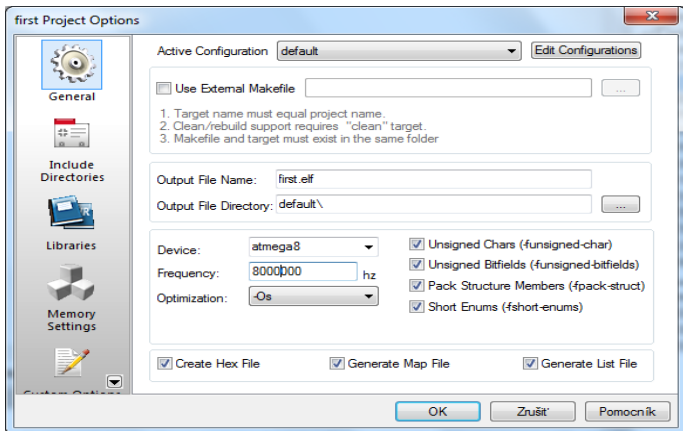
Vypíše chyby, alebo varovania, ktoré počas kompilovania nastali a zobrazí využitie pamäte FLASH a RAM pri použití nášho programu.



Obr. 2.2 - Prostredie programu AVR studio 4

## Nastavenie projektu

V hornom menu zvolíme možnosť: "Project – Configuration Options"



Obr. 2.3 - Nastavenie vlastností projektu

Tu nastavíme minimálne frekvenciu použitých hodín v Hz. Možnosť Optimization nastavuje ako chcem program optimalizovať na rýchlosť. Možné sú úrovne -O0 až -O3.

Ďalšími možnosťami sú -O0 (vypnutie všetkých optimalizácií) a -Os, ktorý povolí možnosti -O2 okrem tých, ktoré zväčšujú kód a pokúsi sa viac zmenšiť výsledný hex súbor. Všeobecne si vystačíme s voľbou -Os. Ďalšie možnosti zatiaľ nastavovať netreba, stačí potvrdiť tlačidlom OK.

## Kompilácia projektu

Väčšina netriviálnych projektov sa skladá z viac, ako jedného .c súboru. Je to spôsobené tým, že program je takto lepšie štruktúrovaný a niektoré jeho časti sa dajú opakovane použiť (napríklad podprogramy pre ovládanie periférií a podobne). Aby sa tieto súbory správne "zložili" do jediného výstupného .hex súboru, je potrebná spolupráca viacerých programov a súborov, preto popíšem postup kompilácie takéhoto projektu.

Ako príklad som si vybral projekt, ktorý generuje pulzy rôznej dĺžky nastaviteľnej z PC cez USB-VCP (virtual COM port) - UART prevodník.

Tento projekt sa skladá z nasledujúcich súborov :

- main.c – zdrojový súbor, obsahuje hlavný program
- ansiterm.c – zdrojový súbor, obsahuje rutiny pre obsluhu ANSI terminálu
- ansiterm.h – hlavičkový súbor, obsahuje deklarácie rutín z ansiterm.c
- Makefile – súbor, ktorý kontroluje preklad a napájanie projektu

Aby sa projekt mohol zostaviť, je najprv potrebné preložiť jednotlivé .c súbory. Prvé, čo sa v tejto chvíli stane je, že sa súbor otvorí a prejde cez preprocesor. To je program, ktorý odstráni komentáre, rôzne pripraví súbor, ale hlavne spracuje direktívy preprocesora (riadky začínajúce sa znakom #). Tam sa radia hlavne include guardy, makrá, include.

Napríklad pomocou direktívy #include sa vkladajú do .c súboru požadované .h súbory, ktoré obsahujú deklarácie. Tie oznamujú, aký tvar majú jednotlivé funkcie a tak podobne. Preto je napríklad pri kompilácii main.c projektu z príkladu potrebné vložiť ansiterm.h, ktorý obsahuje deklarácie funkcií pre ANSI terminál. Tento predspracovaný súbor ďalej spracuje kompilátor, ktorý pri bezchybnom preklade vygeneruje súbor .o. Tomuto súboru sa hovorí objektový súbor (nepiešť s OOP – objektovým programovaním). Tento súbor obsahuje preložené telá funkcií, definície premenných a tak podobne.

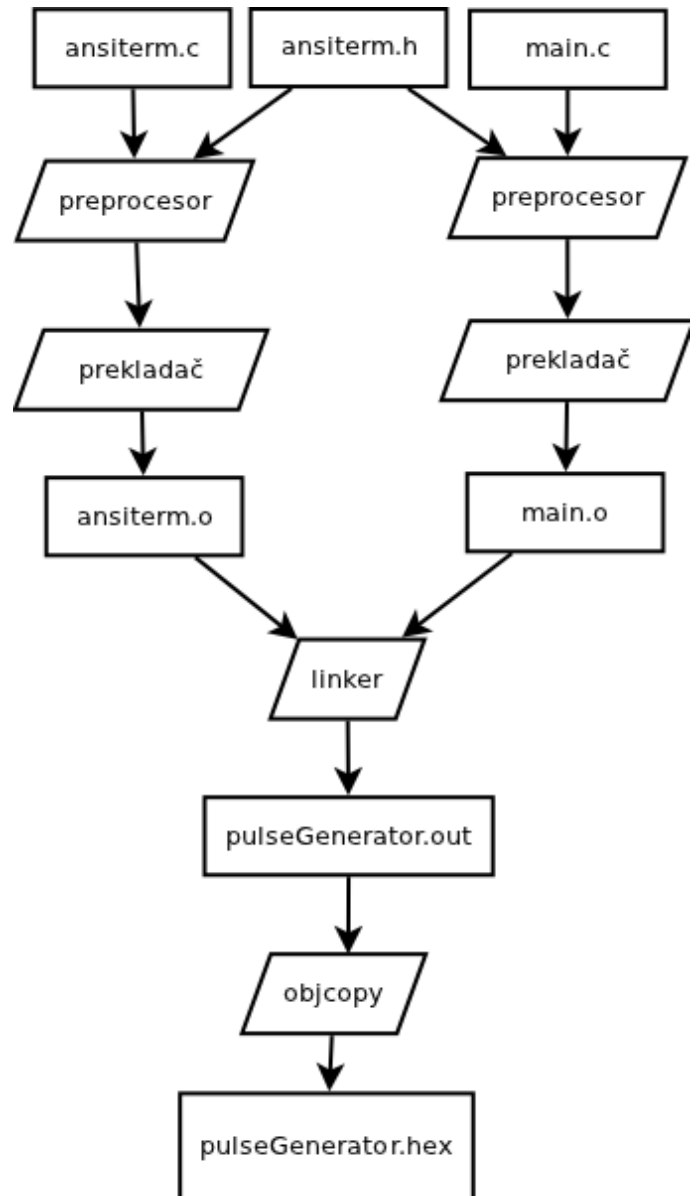
Po kompilácii potrebných .c súborov výsledné .o súbory idú na spracovanie do linkera, ktorý "pospája"



jednotlivé moduly – doplní adresy funkcií a premených z iných modulov a pod. Preto sú niektoré chyby zistené až v tejto fáze. To znamená, že ak linker píše, že niektorá funkcia nie je definovaná, je to práve kvôli chýbajúcemu objektovému, alebo knižničnému súboru. Keď prebehne v poriadku aj linkovanie, dostaneme výsledný binárny súbor, ktorý sa už len pretransformuje do vhodného .hex formátu a môže nasledovať vypálenie programu do MCU.

Celý tento proces je riadený programom make, ktorý spracováva súbor Makefile. Tento obsahuje popis úloh a pre každú úlohu závislosti. Na základe nich sa spracuje celý projekt. Napríklad výsledný súbor .hex závisí od skompilovanej binárky .out, tá od súborov .o a tie zasa od .c a .h súborov.

A týmto spôsobom sa kompiluje väčšina (ak nie všetky) programy v jazyku C, aj keď o tom programátor vďaka moderným prostrediam často ani nevie.



Obr 2.4 - Priebeh kompilácie projektu

# PÍŠEME PRVÝ PROGRAM

### Pravidlá písania kódu

1. Stiahnite si datasheet mikrokontroléra, s ktorým idete pracovať. Nachádzajú sa tam pre vás všetky potrebné informácie, stačí len správne hľadať!
2. Pri písaní kódu používajte komentáre pre lepšiu orientáciu v kóde.
3. Používajte štruktúrovaný zápis zátvoriek a správne odsadenie.
4. Časť kódu ktorá sa v programe viackrát opakuje je vhodnejšie napísať do funkcie.
5. Používajte výstižné názvy premenných, funkcií, typov...

### Využitie hlavičkových súborov

V hlavičkových súboroch sa nachádzajú definície konštánt, makrá a funkcie ktoré nám umožňujú prácu s mikropočítačom.

#### Najčastejšie používané hlavičkové súbory

`#include<avr/io.h>`

Umožňuje nám prácu s registrami mikrokontroléra a prístup k nim pomocou ich názvu. Bez pridania tohto hlavičkového súboru by zápis `PORTD |= (1 << PD4)` nebol možný!

`#include<avr/interrupt.h>`

Tento hlavičkový súbor vkladá funkcie a makrá na obsluhu prerušení.

`#include <avr/sleep.h>`

Využívame ho vtedy, keď chceme používať funkcie spánku.

`#include <util/delay.h>`

Veľmi často používaný. Vďaka tomuto hlavičkovému súboru môžeme využívať časovacie funkcie `_delay_us();` a `_delay_ms();`

`#include <stdlib.h>`

Funkcie na generovanie náhodných čísel, triedenie, hľadanie, celočíselná matematika, prevody reťazcov atd..

`#include <stdio.h>`

Rutiny pre prácu so vstupom a výstupom

### Registre

Naučiť sa správne pracovať s registrami patrí medzi znalosti, bez ktorých sa pri programovaní mikropočítačov ďalej nezaobídeme. Pomocou registrov nastavujeme a ovládame správanie mikrokontroléra a jeho periférií ako napríklad porty, čítač/časovač, A/D prevodník, UART a pod.

#### Rozlišujeme 3 typy registrov

1. Pracovné – je to 32 8-bitových registrov slúžiacich na ukladanie výsledkov a premenných pri behu programu. Pri programovaní v C tieto registre manažuje kompilér.
2. Príznakové – sú to registre, ktoré nás informujú o stave mikrokontroléra, periférií atď. Po ich prečítaní sa môžeme dozvedieť napr. stav portu.
3. Nastavovacie – týmto budeme venovať najväčšiu pozornosť. Pomocou týchto registrov môžeme nastavovať všetky parametre mikrokontroléra.

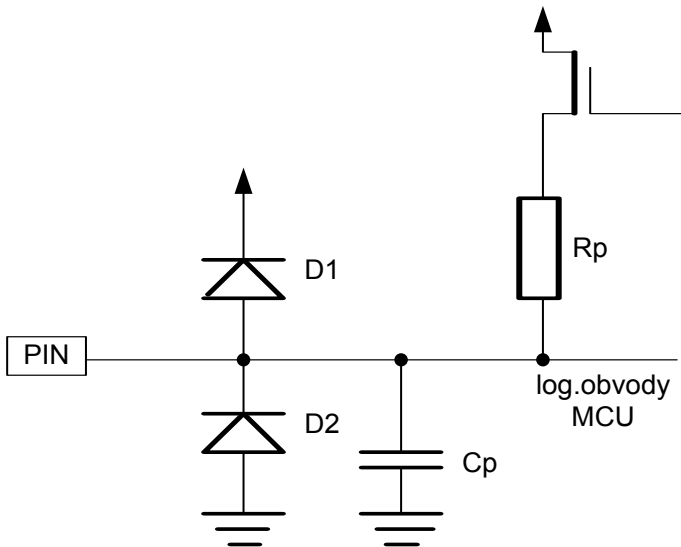
#### I/O Porty (Vstupno - výstupné porty)

Port predstavuje 8 bitov (tj. jeden bajt), ktoré sú fyzicky vyvedené na piny mikrokontroléra. Nie vždy je vyvedených všetkých 8 bitov (kvôli obmedzenému počtu pinov púzdra). Porty sa označujú abecedne – PORTA, PORTB, PORTC., ich počet závisí od typu mikrokontroléra.

Jednotlivé bity (piny) portu sú označené Pn0 - Pn7 (n – reprezentuje písmeno portu). Pn7 je prvý bit zľava.

#### Vlastnosti:

- Jednotlivé piny portu možno nastaviť ako vstupné alebo výstupné
- Piny portu môžu mať v závislosti od nastavenia registrov aj alternatívnu funkciu ako napr. UART, SPI, pripojenie externého kryštálu a pod.
- Piny sú chránené diódami voči VCC a GND
- Pin možno zaťažiť prúdom až 40mA a to v oboch stavoch log. 1 aj log. 0
- Pin môže mať aktivovaný pull-up rezistor voči VCC (cca 20 – 50 kΩ)
-



Obr 3.0 - Vnútroštruktúra I/O pinu MCU

**Každý port má 3 registre:**

- DDRx (data direction register) – nastavujeme ním jednotlivé piny portu ako vstupné alebo výstupné. Log. 1 na príslušnom bite nastavuje pin ako výstupný a log. 0 naopak ako vstupný.
- PORTx – nastavuje log. úrovně na výstupných pinoch portu. Pre piny, ktoré sú nastavené ako vstupné, sa zápisom log. 1 do príslušného bitu aktivuje pull-up rezistor na danom pine.
- PINx – určený len na čítanie. Register obsahuje stav logických úrovní na porte. Možno teda na port zvonka privádzať log. úrovně a pomocou tohto registra zisťovať logické hodnoty na jednotlivých pinoch portu.

**Príklady**

1.) Chceme nastaviť všetky piny portu PORTD ako výstupné, a zapísať na výstup bitovú informáciu 1111 0000. To znamená, že piny PD7 až PD4 budú v log. 1 a PD3 až PD0 v log. 0.

**Najskôr nastavíme piny portu ako výstupné:**

```
DDRD = 0b11111111; // bitový zápis, alebo:
DDRD = 0xFF; // hexadecimálny zápis
```

**Potom zapíšeme dáta:**

```
PORTD = 0b11110000; // bitový zápis, alebo:
PORTD = 0xF0; // hexadecimálny zápis, alebo:
PORTD = 240; // dekadický zápis
```

2.) Chceme nastaviť všetky piny portu PORTD ako vstupné, a niečo vykonať, ak je na všetky piny privedená log. 1.

```
DDRD = 0b00000000; // všetky piny ako vstupné
PORTD = 0b00000000; // pull-up rezistory neaktivované
if (PIND == 0b11111111) // podmienka ak stav portu rovná sa 1111 1111
{ /* akcia */ }
```

**Maskovanie**

Ukázali sme si teda ako nastavovať a čítať jednotlivé piny portu. Čo však v takom prípade, že na porte už máme zapísané nejaké hodnoty a chceme zmeniť len jeden pin (bit) v danom porte? Vtedy prichádza na rad maskovanie.

**Pri maskovaní využívame bitové log. operátory**

- & (bitový "and")
- | (bitový "or")
- ~ (bitovú negáciu)
- << bitový posun

A	B	O	A	B	O
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

**AND                      OR**

Obr 3.1 - Pravidlová tabuľka log. členov AND a OR

**Najčastejšie využívame takýto štýl zápisu:**

(pre názornosť zvolíme pin PD6 registra PORTD)

```
PORTD |= (1 << PD6);
- zápis log. 1 na pin 6 portu D

PORTD &= ~(1 << PD6);
- zápis log.0 na pin 6 portu D
```

## PÍŠEME PRVÝ PROGRAM

---

Tento zápis robí začiatčovníkom asi najväčší problém, preto si ho rozoberieme na jednotlivé časti.

V hlavičkovom súbore sú definované jednotlivé piny portu. Konkrétne ku PD6 je priradené číslo 6. Je to tak aj pre ďalšie piny napr. PD0=0, PD1=1, PD2=2 .....

Zápis (1 << PD6) možno nahradiť aj za zápis (1 << 6), teda log. 1 (hodnota 0b00000001) je bitovo posunutá o 6 bitov doľava - 0b01000000. Toto je teda naša maska ktorú sme vytvorili a budeme s ňou ďalej pracovať.

### Zápis log. 1 na pin 6 portu D:

Zápis `PORTD |= (1 << PD6);`

možno zapísať aj v tvare:

`PORTD = PORTD | (1 << PD6);`

Teda vykonáme log. operáciu OR (logický súčet) medzi jednotlivými bitmi aktuálnej hodnoty registra PORTD a masky, ktorú sme vytvorili, a tento výsledok potom naspäť zapíšeme do registra PORTD.

**Príklad:** Register PORTD má zapísanú hodnotu 0b00001111. Vykonáme operáciu bitový OR s maskou 0b01000000

PORTD	0b00001111
MASKA	0b01000000
Výsledok	0b01001111

Vidno teda, že predošlý obsah portu ostal zachovaný a zmenil sa len bit číslo 6, ktorý potrebujeme.

### Zápis log. 0 na pin 6 portu D:

`PORTD = PORTD & ~(1 << PD6);`

Majme teraz na PORTD zapísanú hodnotu 0b11110000.

Zápis `~(1 << PD6)` znamená to, že log. 1 bitovo posunieme o 6 bitov doľava (0b01000000) a následne vykonáme bitovú negáciu (0b10111111).

Teraz vykonáme operáciu AND (logický súčin) medzi bitmi registra PORTD a masky.

PORTD	0b11110000
MASKA	0b10111111
Výsledok	0b10110000

Zase vidíme že predošlý obsah portu ostal zachovaný a zmenil na log. 0 sa zmenil len šiesty bit.

V prípade že chceme zapísať viac bitov zápis je nasledovný:

`PORTB |= (1 << PB4) | (1 << PB2);`

pri zapísaní viac log. 1

`PORTB &= ~((1 << PB4) | (1 << PB2));`

pri zapísaní viac log. 0

Zoberte si teda papier a pero a overte, že tento zápis naozaj funguje :) !

### Maskovanie využívame aj pri čítaní:

Máme napr. podmienku v ktorej nás zaujíma či daný pin portu je zvonka nastavený na log. 1.

Pre názornosť si zvolme tento pin napr. PD4

```
if (PIND & (1 << PD4)) {  
/* ... príkazy po splnení podmienky... */  
}
```

Maska	0b00010000
PIND	0b00110011
Výsledok	0b00010000

Po operácií bitového AND je výsledok 0b00010000, teda ostatné piny portu sme "odfiltrovali" a overili sme stav len žiadaného pinu. Logická hodnota výrazu v podmienke je "pravda" (true), a teda podmienka je splnená, pretože výsledok operácie je nenulový (číselná hodnota 0, tj. 0b00000000 by bola vyhodnotená ako logická nepravda).

V ďalšom príklade majme opačnú situáciu a to overenie či daný pin PD4 je v log. 0

```
if ((PIND & (1 << PD4)) == 0) {  
/* ... príkazy po splnení podmienky... */  
}
```

Maska	0b00010000
PIND	0b10001111
Výsledok	0b00000000

Výsledok po operácií bitového AND bude = 0b00000000, a porovnanie s hodnotou 0 je pravda, čiže podmienka je splnená.

Ukončili sme teda časť softwarovú a teraz pristúpime k samotnému hardwaru. Ukážeme si ako zapojiť mikropočítač ATmega8, ako naprogramovať mikropočítač a jeho fuse-bity a nakoniec si predvedieme funkčnosť ukážkového programu.

## Zapojenie mikrokontroléra

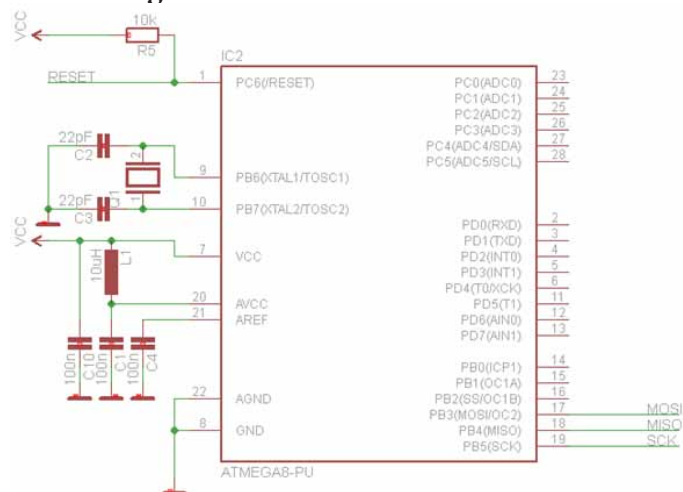
V ďalších kapitolách budem používať mikropočítač ATmega8. Vždy na konci kapitoly bude pár ukážkových programov, ktoré si môžete sami otestovať. Preto vám odporúčam zakúpiť tento mikropočítač a kontaktné pole, na ktorom si napísané programy jednoducho odskúšate.

Vačšina prezentovaných programov bude fungovať aj na mikropočítačoch ATmega16 a ATmega32. Rozdiel bude však v schémach, pretože tieto mikropočítače majú inak rozložené piny.

Pre tých čo to myslia s programovaním vážnejšie, dávam do pozornosti aj tieto vývojové dosky publikované na webe svetelektro.com :

- Vývojová doska s ATmega16/32 - luboss17
- Vývojová doska s ATmega 16/32 - bobo87

## Na Obr 3.2 vidno základné zapojenie mikrokontroléra ATmega8:



Obr 3.2 - Zapojenie MCU ATmega8

Reset signál je pripojený cez odpor 10k na VCC. Pripojením log. 0 na reset pin reštartujeme mikropočítač. Pokiaľ chceme použiť externý kryštál, tak ho zapojíme na piny PB6 a PB7. Pripojíme aj 22pF kondenzátory na tieto piny oproti zemi. Rozsah napájacieho napätia je 2,7 – 5,5V pre ATmega8L, a 4,5 - 5,5V pre ATmega8.

Čo najbližšie ku napájaciemu pinu zapojíme kondenzátor 100nF oproti zemi. Pokiaľ je napájacích pinov viac, dávame 100n kondenzátor pri každý z nich!

Pin AVCC je napájanie pre AD prevodník a piny 0 až 3 na porte C. Pri použití AD prevodníka je odporúčané tento pin pripájať cez LC filter, aby sme znížili rušenie z napájacej vetvy a dosiahli lepšie parametre AD prevodu. Ak AD prevodník nepoužívame, alebo LC filter nemusíme/nemôžeme/nechceme do obvodu zapojiť, pripojíme pin priamo na kladné napájanie.

Pin AREF je pin napäťovej referencie AD prevodníka. Pri použití internej referencie je naňho pripojené výstupné napätie z internej referencie, a preto na tento pin pripojíme blokovací kondenzátor 100nF. Ak používame externú napäťovú referenciu, pripájame ju na tento pin.

## Programovacie rozhranie mikrokontroléra

Najpoužívanejší spôsob programovania mikrokontroléra AVR je ISP (In System Programming), cez sériové rozhranie SPI. Mikrokontrolér sa dá naprogramovať týmto spôsobom priamo v aplikácii bez potreby vyberania mikrokontroléra, čo je veľká výhoda.

Na naprogramovanie sa využívajú 4 dátové vodiče – MOSI, MISO, SCK a RESET, a spoločná zem GND. Ako štandard na pripojenie programátora k mikropočítaču sa používa 6 alebo 10 pinový konektor. Niektoré mikropočítače rady AVR majú aj JTAG programovacie rozhranie, ktoré okrem programovania ponúka aj možnosť debugovania (ladenia) programu.

Pri programovaní cez ISP sa vždy držte pravidla že aplikácia napája programátor cez ISP a nie naopak!

## Štandard zapojenia ISP konektora



ISP Connectors: 6-pin & 10-pin

Obr 3.3 - Štandardné zapojenie ISP konektora

# PÍŠEME PRVÝ PROGRAM

## Výber programátora

V prvom rade sa treba rozhodnúť či chcete programovať mikropočítač v prostredí AVR Studio 4. Má to veľkú výhodu, pretože je všetko "pod jednou strechou". Pokiaľ sa rozhodnete pre túto možnosť budete si musieť zakúpiť originálny programátor alebo vyrobiť klon. Za celkom dobré ceny sa predávajú programátory AVR dragon, AVRISPMkI a AVRISPMkII. Na Slovensku sa dajú zakúpiť napr. v SOS alebo GME

Na internete je však aj množstvo návodov na výrobu ISP programátorov pre AVR:

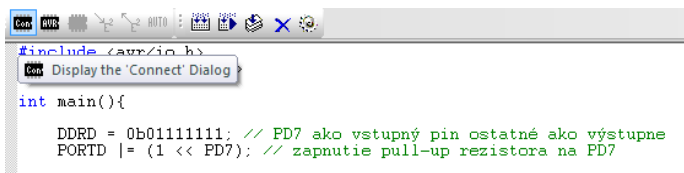
- Najjednoduchší programátor cez paralelný port (neodporúča sa)
- USBasp - USB programátor (pre AVRDUDE, odporúčaný)
- AVRISP-MKII clone - USB programátor (kompatibilný s AVR Studio, odporúčaný)
- biprog - RS232 programátor (kompatibilný s AVR Studio, odporúčaný)

Na stránkach sa dozviete aj aký programovací software treba použiť na naprogramovanie mikropočítača.

## Nahratie programu do mikrokontroléra v prostredí AVR studio 4:

Ďalej budem popisovať ako nahráť váš program pomocou programátora kompatibilného s prostredím AVR Studio 4. Pokiaľ sa rozhodnete pre iný programátor, treba si preštudovať spôsob programovania na stránke autora.

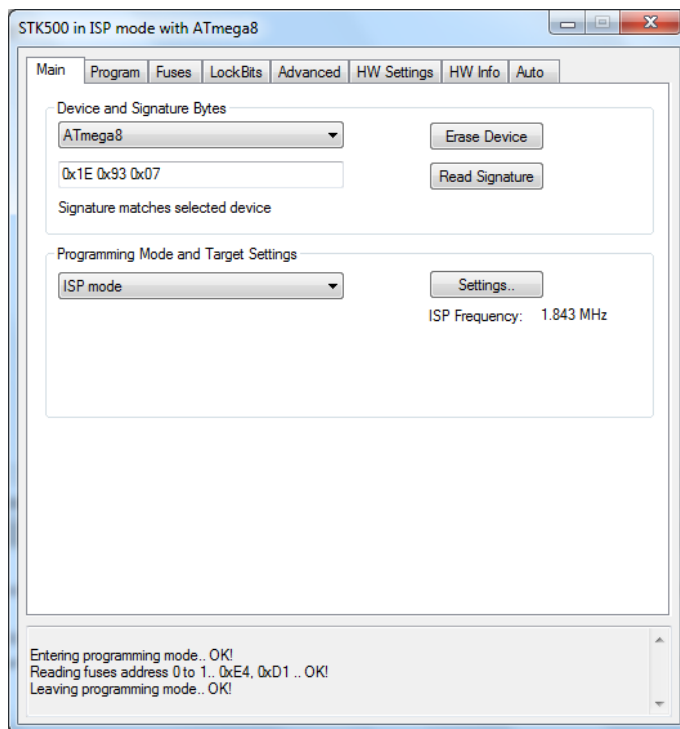
Po stlačení tlačidla "Connect" vyberieme typ programátora a port na ktorý je pripojený a potvrdíme tlačidlom "Connect".



```
#include <avr/io.h>
// Display the 'Connect' Dialog
int main(){
    DDRD = 0b01111111; // PD7 ako vstupný pin ostatné ako výstupné
    PORTD |= (1 << PD7); // zapnutie pull-up rezistora na PD7
```

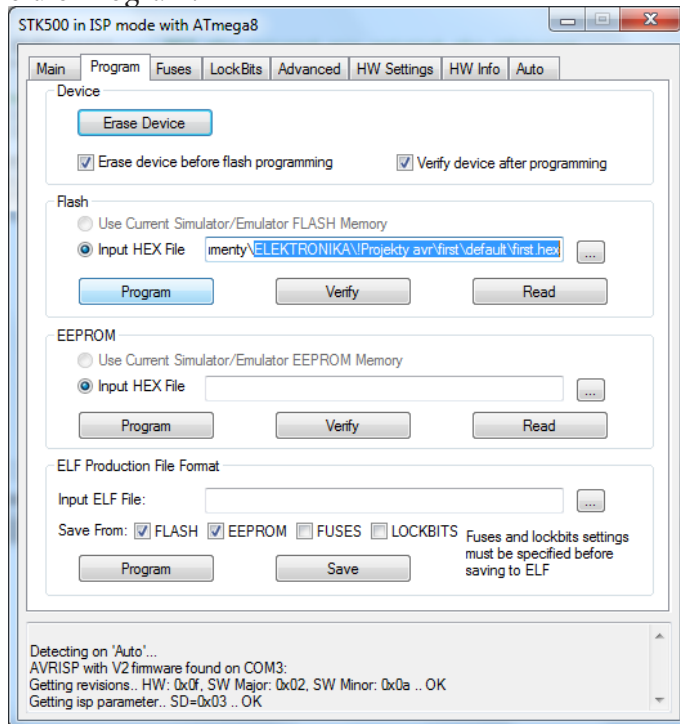
Obr. 3.4 - Pripojenie ku programátoru

Objaví sa nám nové okno s viacerými záložkami. Na prvej Main obrazovke možno vymazať obsah pamäte mikrokontroléra alebo prečítať Signatúru, ktorá nám hovorí o type mikrokontroléra. Tým môžeme zistiť aký mikropočítač sa nachádza v danom zapojení.



Obr 3.5 - Prvá záložka programovania MCU

Ďalšia záložka Program je najčastejšie využívaná. Slúži na naprogramovanie FLASH a EEPROM pamäte zo súboru, ktorý sme po skompilovaní dostali. Pre naprogramovanie nášho programu vyberieme skompilovaný \*.hex súbor z nášho projektu a stlačíme tlačidlo Program.



Obr 3.6 - Druhá záložka programovania MCU

## Záložka Fuses slúži na nastavenie mikrokontroléra:

- RSTDISBL – vypnutie reset signálu a využitie daného pinu ako vstupno-výstupný. Pozor – po nastavení tejto poistky sa nebude dať mikropočítač cez rozhranie ISP programovať!
- WTDON – zapnutie funkcie watchdog
- SPIEN – povolenie SPI rozhrania. Pozor – po vypnutí tejto poistky sa nebude dať mikropočítač cez rozhranie ISP programovať!
- EESAVE – poistka nastavuje či sa má EEPROM pamäť po prepísaní FLASH zmazať alebo zachovať
- BOOTSZ – určuje miesto vo FLASH pamäti kde sa nachádza bootloader
- BOOTRST – zapnutie funkcie bootloadera
- CKOPT – pri použití externého kryštálu táto voľba zvýši napätie oscilátora a robí ho tak menej náchylným v zarušenom prostredí. Zvyšuje spotrebu zariadenia
- BODLEVEL – nastavenie spodnej hranice napájacieho napätia pri ktorej sa mikropočítač reštartuje
- BODEN – zapnutie funkcie reštartu mikrokontroléra pri určenom nízkom stave napájacieho napätia
- SUT\_CKSEL – najpoužívanejšia poistka. Slúži na nastavenie zdroja hodín pre mikropočítač. Možno zvoliť napr. interný RC oscilátor, externý kryštál a pod...

## Náš prvý program

V našom prvom programe si vyskúšame prácu s IO Portami. Zapojenie zapojíme podľa schémy na Obr 3.2. Fuse bity nastavíme na interný RC oscilátor 8 MHz, takú istú frekvenciu nastavíme aj do projektu AVR Studio 4.

### Program funguje nasledovne:

LED bliká len v tom prípade ak je PD7 spojený s GND a teda je na ňom log. 0. Keď pin PD7 odpojíme od GND, interný pull-up rezistor privedie na pin PD7 napájacie napätie, čo zmení log. úroveň na pine. Tým pádom podmienka nie je splnená a LED neblinká.

## Využitie registrov:

- pomocou registra DDRD sme nastavili PD7 ako vstupný pin a ostatné piny portu ako výstupné
- pomocou registra PORTD sme nastavili pull-up rezistor na pin PD7 a prepíname pomocou neho log. úroveň na PD6 čím LED bliká.
- čítame register PIND a overujeme log. úroveň na PD6.

## Zdrojový kód:

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

int main()

    // PD7 ako vstupný pin ostatné ako výstupné
    DDRD = 0b01111111;
    PORTD |= (1 << PD7); // zapnutie pull-up rezistora na PD7

    // nekonečná slučka
    while(1){
        if((PIND & (1 << PD7)) == 0){ // ak je PD7 nulový

            PORTD |= (1 << PD6); // rozsvieť LED
            _delay_ms(200); // počkaj 200ms
            PORTD &= ~(1 << PD6); // zhasni LED
            _delay_ms(200); // počkaj 200ms
        }
    }

    return 0;
}
```

## Youtube video:

Test prvého programu - link

# PRERUŠENIA

## Prerušenie

Prerušovací podsystem umožňuje efektívnu obsluhu nepravidelných udalostí, na ktoré má mikrokontrolér reagovať. V prípade výskytu definovanej udalosti mikrokontrolér preruší beh programu a venuje sa programovej obsluhu udalosti, ktorá dané prerušenie vyvolala. Po skončení tejto programovej obsluhy mikrokontrolér pokračuje vo výkone hlavného programu. Význam prerušení je hlavne v zrýchlení a zefektívnení behu programu.

V jednej chvíli (t.j. v jednom strojovom cykle) sa môžu vyskytnúť viaceré prerušenia a mikrokontrolér musí rozhodnúť, ktorá žiadosť bude vykonaná ako prvá. Na to je potrebné definovať akési poradie dôležitosti – prioritu prerušení.

Na základe priority prerušení sa určí postupnosť spracovania žiadostí v tom istom strojovom cykle. Platí teda, že čím menšie číslo vektora prerušenia, tým ma prerušenie vyššiu prioritu. Z tabuľky teda vidíme že najväčšiu prioritu ma prerušenie od RESET a najmenšiu od SPM\_RDY.

Vector No.	Program Address <sup>(2)</sup>	Source	Interrupt Definition
1	0x000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow
10	0x009	TIMER0 OVF	Timer/Counter0 Overflow
11	0x00A	SPI, STC	Serial Transfer Complete
12	0x00B	USART, RXC	USART, Rx Complete
13	0x00C	USART, UDRE	USART Data Register Empty
14	0x00D	USART, TXC	USART, Tx Complete
15	0x00E	ADC	ADC Conversion Complete
16	0x00F	EE_RDY	EEPROM Ready
17	0x010	ANA_COMP	Analog Comparator
18	0x011	TWI	Two-wire Serial Interface
19	0x012	SPM_RDY	Store Program Memory Ready

Obr 4.1 - Tabuľka zdrojov prerušení

Vidíme teda že ATmega8 má celkovo 19 zdrojov prerušení. V nasledovných riadkoch sa budeme venovať externým prerušeniam - INT0, INT1.

Pred použitím akéhokoľvek prerušenia sa musia prerušenia najskôr globálne povoliť. To dosiahneme pomocou funkcie sei();

Naopak v niektorých kritických častiach programu chceme prerušenie globálne zakázať. Na to slúži funkcia cli();

## Zápis obsluhy prerušenia

Na začiatku programu pridáme hlavičkový súbor pomocou direktívy `#include <avr/interrupt.h>`, ktorý nám zabezpečí prácu s prerušeniami.

**Samotnú funkciu prerušenia potom deklarujeme takto:**

```
ISR (vektor prerušenia) {
// ... akcia prerušenia ... //
}
```

**Kompletný zoznam vektorov prerušenia pre C nájdete na adrese:**

[http://www.nongnu.org/avr-libc/user-manual/group\\_\\_avr\\_\\_interrupts.html](http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html)

**Program s prerušením môže teda vyzerať takto:**

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

```
// prerušenie od INT0
ISR (INT0_vect) {
// ...nejaká akcia... //
}
```

```
int main () {
sei(); //povolenie globálneho prerušenia
GICR |= (1 << INT0); // povolenie prerušenia od INT0
while(1); // nekonečná slučka
}
```

Podľa možnosti sa snažíme obslužnú funkciu prerušenia spraviť čo najkratšiu (zmeniť premennú a pod.). Robíme to preto, aby zbytočne dlhá obsluha prerušenia nebrzdila hlavný program alebo prípadne ďalšie prerušenia, ktoré môžu nastať.



**Zdielané premenné:**

Ak potrebujeme v obsluhu prerušenia meniť hodnotu globálnej premennej, musí byť táto premenná deklarovaná s kľúčovým slovom volatile. V opačnom prípade sa zmena hodnoty premennej vykonaná v obsluhu prerušenia neuchová po skončení obsluhy.  
Príklad: volatile uint8\_t abc;

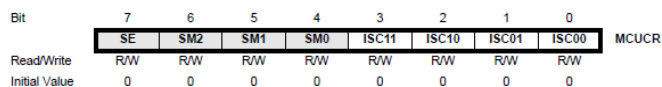
**Externé prerušenia**

Využívame ich vtedy, ak chceme prerušenie vyvolať zvonku (napr. stlačenie tlačidla). ATmega8 má dva externé zdroje prerušenia nazvané INT0 (na pine PD2) a INT1 (na pine PD3). Na ich obsluhu využívame dva registre (pozri datasheet MCU ATmega8 strana 66):

**Popis registrov:**

MCUCR - MCU Control Register

– pomocou bitov ISC11, ISC10, ISC01 a ISC00 v tomto registri nastavíme spôsob aktivácie externého prerušenia od INT0 a INT1.



Obr 4.2 - Register MCUCR

Na výber máme tieto možnosti:

- Reakcia na nízku úroveň napätia na pine
- Zmena logického stavu na pine
- Reakcia na nábežnú hranu (rising edge)
- Reakcia na dobežnú hranu (falling edge)

ISC01	ISC00	Popis
0	0	Žiadosť o prerušenie na vstupe INT0 je generovaná úrovňou odpovedajúcou log.0
0	1	Ľubovoľná logická zmena na vstupe INT0 generuje žiadosť o prerušenie
1	0	Zostupná hrana na vstupe INT0 generuje žiadosť o prerušenie
1	1	Nábežná hrana na vstupe INT0 generuje žiadosť o prerušenie

Tab 4.0 - Spôsob aktivácie ext. prerušenia INT0

ISC11	ISC10	Popis
0	0	Žiadosť o prerušenie na vstupe INT1 je generovaná úrovňou odpovedajúcou log.0
0	1	Ľubovoľná logická zmena na vstupe INT1 generuje žiadosť o prerušenie
1	0	Zostupná hrana na vstupe INT1 generuje žiadosť o prerušenie
1	1	Nábežná hrana na vstupe INT1 generuje žiadosť o prerušenie

Tab 4.1 - Spôsob aktivácie ext. prerušenia INT1

Príklad:

Nastavíme reakciu INT0 na nábežnú hranu a INT1 na dobežnú hranu:

```
// INT0 na nábežnú hranu
MCUCR |= (1 << ISC01) | (1 << ISC00);
```

```
//INT1 na dobežnú hranu
MCUCR |= (1 << ISC11);
```

GICR - pomocou bitov INT1 a INT0 v tomto registri povolíme prerušenia od INT0 a INT1.

Príklad:

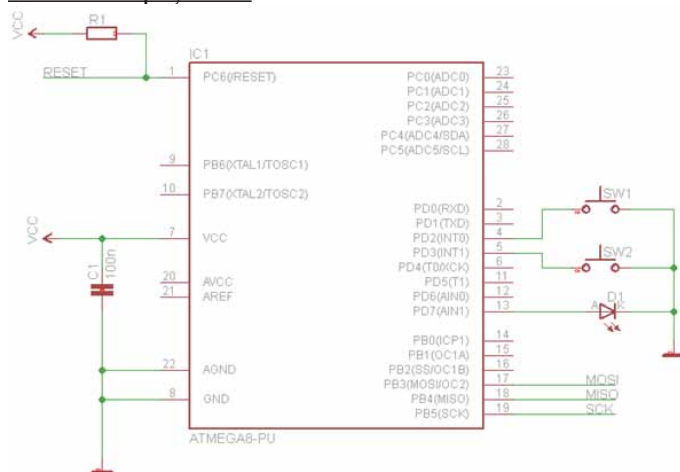
Povolenie prerušenia od INT1:

```
GICR |= (1 << INT1);
```

**Ukážkový program:**

Zapnutie LED na pine PD7 pomocou INT0 a vypnutie pomocou INT1. Reakcia prerušenia na dobežnú hranu (zmenu z log. 1 na log. 0).

Schéma zapojenia:



Obr 4.3 - Schéma zapojenia s ext. prerušením

## PRERUŠENIA

---

### Zdrojový kód:

```
#include <avr/io.h>
#include <avr/interrupt.h>
ISR(INT0_vect){
    PORTD |= (1 << PD7); // zapni LED
}

ISR(INT1_vect){
    PORTD &= ~(1 << PD7); // vypni LED
}

int main(){

    DDRD |= (1 << PD7); //PD7 ako výstupný (LED)

    // PD2,PD3 ako vstupné (ext. prerušenia)
    DDRD &= ~((1 << PD2) | (1 << PD3));

    // Zapnutie interného pull-up rezistora
    PORTD |= (1 << PD2) | (1 << PD3);

    MCUCR |= (1 << ISC01); // dobežná hrana INT0
    MCUCR |= (1 << ISC11); // dobežná hrana INT1

    // povol prerušenia od INT1 a INT0
    GICR |= (1 << INT1) | (1 << INT0);
    sei(); //povol globálne prerušenia

    while(1); // nekonečný cyklus

return 0;
}
```

### Zdrojový kód – použitie globálnej premennej:

To isté ako prvý ukázkový program, ale s použitím globálnej premennej.

Tip: vyskúšajte si program bez kľúčového slova "volatile".

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

volatile uint8_t led = 0;

ISR(INT0_vect) {
    led = 1;
}

ISR(INT1_vect) {
    led = 0;
}

int main() {

    DDRD |= (1 << PD7); //PD7 ako výstupný (LED)

    // PD2,PD3 ako vstupné (ext. prerušenia)
    DDRD &= ~((1 << PD2) | (1 << PD3));

    // Zapnutie interného pull-up rezistora
    PORTD |= (1 << PD2) | (1 << PD3);

    MCUCR |= (1 << ISC01); // dobežná hrana INT0
    MCUCR |= (1 << ISC11); // dobežná hrana INT1

    // povol prerušenia od INT1 a INT0
    GICR |= (1 << INT1) | (1 << INT0);
    sei(); // globálne povolenie prerušení

    while(1) // nekonečný cyklus
    {
        if (led)
            PORTD |= (1 << PD7); // zapni LED
        else
            PORTD &= ~(1 << PD7); // vypni LED
    }
    return 0;
}
```

### Youtube video:

Test externého prerušenia - link

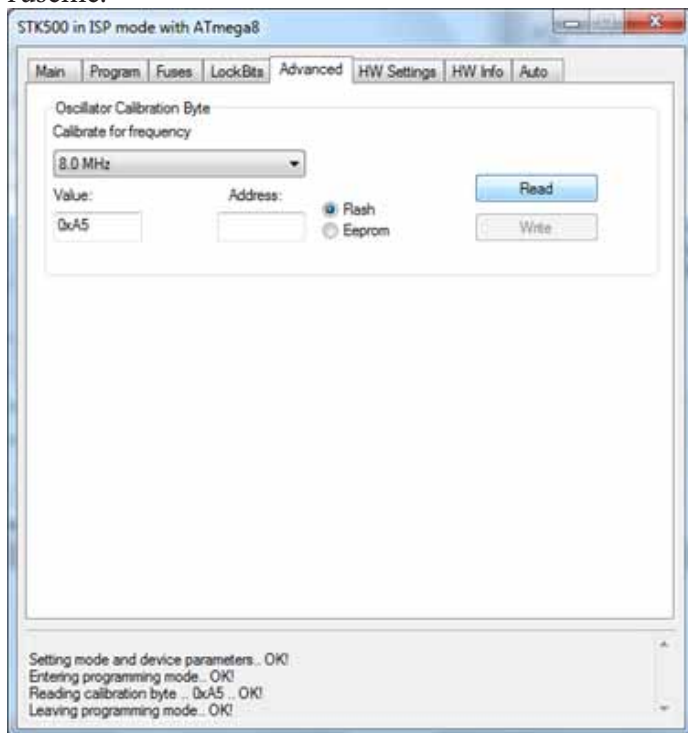
# ČÍTAČE/ČASOVAČE

## Úvod k čítačom/časovačom

Čítač/časovač patrí medzi najpoužívanejšie periférie čo sa mikrokontrolérov týka. Umožňuje presne časovať beh určitej časti programu, dokáže čítať vonkajšie alebo vnútorne impulzy, generovať signál PWM, alebo signál určitej frekvencie a pod. Ide o samostatnú časť, ktorá beží nezávisle na prebiehajúcich inštrukciách mikrokontroléra.

Mikrokontroléry rady AVR obsahujú niekoľko jednotiek čítača/časovača a označujú sa číslom (0, 1, 2 ...). Konkrétne ATmega8 obsahuje takéto jednotky tri. Každá z nich dokáže vykonávať základné činnosti (čítať/časovať) avšak niektoré z nich obsahujú aj ďalšie funkcie ako generovanie PWM signálu, externý vstup čítača atď...

Register ktorý ukladá aktuálnu hodnotu čítača sa označuje TCNTn, kde n predstavuje číslo čítača/časovača. Pri časovaní udalostí často využívame pretečenia TCNTn, pri ktorom vykonáme prerušenie. Pretečenie nastane vtedy, keď máme v čítači maximálnu hodnotu (napr. pri 8 bitovom je to hodnota 255) a túto hodnotu inkrementujeme. Vtedy sa čítač vynuluje a nastane prerušenie.



Obr 5.0 - Prečítanie kalibračnej konštanty MCU

Pri používaní interného RC oscilátora sa môže náš nastavený čas od skutočného mierne líšiť. Je to spôsobené odchýlkou oscilátora. Pri výrobe mikrokontroléra bol vnútorný RC oscilátor nakalibrovaný a výsledkom je kalibračná konštanta.

Túto konštantu sa dozvieme v programovacom okne v záložke "Advanced". Po vybratí želanej frekvencie a stlačení tlačidla Read sa dozvieme našu kalibračnú konštantu, ktorá je pre každý mikrokontrolér jedinečná. V mojom prípade je hodnota 0xA5. Keď chceme aplikovať kalibračnú konštantu v našom programe zapíšeme túto hodnotu do registra OSCCAL.

Zápis v mojom prípade bude vyzeráť takto :

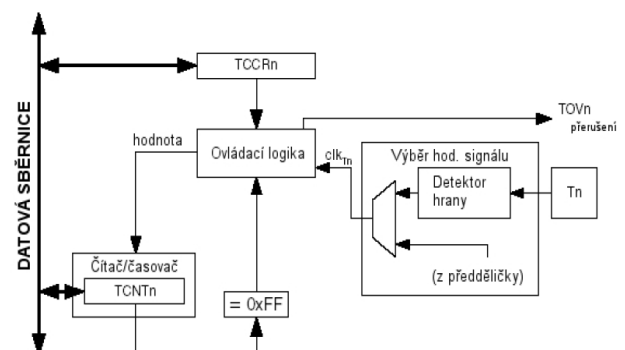
`OSCCAL = 0xA5;`

V nasledovných riadkoch si postupne rozoberieme tri čítače/časovače ktoré obsahuje mikrokontrolér ATmega8 a zoznámime sa s možnosťami ich použitia.

## Čítač/časovač 0 (viď datasheet obvodu Atmega8 na strane 67)

### Vlastnosti:

- 8 bitový čítač/časovač
- Čítač externých impulzov privádzaných na T0
- 10 bitová preddelička
- Prerušenie na akciu pretečenia TCNT0



Obr 5.1 - Bloková schéma čítača/časovača 0

## Popis registrov:

### TCCR0 (Timer/Counter 0 Control Register)

– slúži na riadenie funkcie čítača/časovača 0.

Bit	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R/W	R/W	R/W	TCCR0
Initial Value	0	0	0	0	0	0	0	0	

Obr 5.2 - Register TCCR0

V prípade čítača/časovača 0 slúži len na výber zdroja hodín a preddeličky pomocou bitov CS02, CS01, CS00. Z tabuľky vidíme, že obsah registra TCNT0 môžeme inkrementovať buď vnútorne (priamo alebo cez preddeličku), alebo externe cez vstupný pin T0. Ak máme frekvenciu hodín 8MHz, tak bez preddeličky sa bude obsah registra TCNT0 inkrementovať za čas  $T = 1/8\text{MHz} = 125\text{ns}$ . Pri preddeličke 64 bude perióda inkrementácie TCNT0 -  $T = 1/(8\text{MHz}/64) = 8\text{ms}$ .

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{\text{I/O}}$ /(No prescaling)
0	1	0	$\text{clk}_{\text{I/O}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{I/O}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{I/O}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (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

Obr 5.3 - Výber preddeličky

### TIMSK (Timer/counter Interrupt Mask Register)

Bit	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	TIMSK
Initial Value	0	0	0	0	0	0	0	0	

Obr 5.4 - Register TIMSK

Pomocou bitu TOIE0 (Timer/counter 0 Overflow Interrupt Enable) v tomto registri môžeme povoliť prerušenie na akciu pretečenia čítača/časovača 0.

## Príklady:

### Príklad č.1:

S využitím čítača/časovača 0 a frekvencií hodín 8 MHz realizujte blikanie LED s približnou dĺžkou zopnutia  $T_{on}=0,5\text{s}$  a dĺžkou rozopnutia  $T_{off} = 0,5\text{s}$ .

Zvolíme preddeličku 1024:

$$T = 1 / (8\text{MHz} / 1024) = 128\mu\text{s}$$

– perióda inkrementácie TCNT0.

Keďže čítač/časovač 0 je 8-bitový, tak pretečenie nastane vždy po 256 impulzoch, teda nastane každých  $256 \cdot 128\mu\text{s} = 32,768\text{ms}$ . Vypočítame teda koľko krát musí prerušenie nastať, aby sme dosiahli čas 0,5s.

$$N = 500\text{ms} / 32,768\text{ms} = 15,25x \text{ čo je zaokrúhlene } 15x, \text{ teda } 491,52\text{ms} \text{ čo predstavuje odchýlku } -1,7\%$$

## Zdrojový kód:

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

volatile unsigned char i;

// pretečenie počítadla TCNT0
//nastane každých 128us*256 = 32,768ms
ISR (TIMER0_OVF_vect){

    // 15*32.768 = 491,52ms
    if(i == 15){
        // finta s log. operáciou XOR - neguje len pin PD7
        PORTD ^= (1 << PD7);
        i=0;
    }

    i++;
}

int main(){
    DDRD |= (1 << PD7); // PD7 ako výstupný

    // preddelička /1024 (128us)
    TCCR0 |= (1 << CS02) | (1 << CS00);
    TIMSK |= (1 << TOIE0); // prerušenie pri pretečení TCNT0

    sei(); //povol globálne prerušenia

    while(1); //nekonenčná slučka

    return 0;
}
```

## Youtube video:

8-bitový čítač/časovač 0 - link

### Príklad č.2:

Z predchádzajúceho príkladu vidíme, že pri preddeličke 1024 a frekvencii hodín 8MHz nie je možné nastaviť čítač/časovač 0 na presne zadaný čas. Použijeme teda preddeličku 8 pri frekvencií hodín 8MHz, pri ktorom bude perióda inkrementácie čítača 1us.

S týmto časom už vieme pohodlnejšie pracovať.

Ukážeme si aj "fintu" s ručným nastavením registra TCNT0, ktorým nastavíme, aby nám pretečenie nastalo vždy po 100 impulzoch teda po 100us.

Zdrojový kód:

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

volatile unsigned int i;
void delay_ms(unsigned int time);

// pretečenie počítadla TCNT0 - nastane každých 100us
ISR (TIMER0_OVF_vect){
    // nastavenie počiatočnej hodnoty počítadla
    TCNT0 = 156;
    i++;
}

//vlastná čakacia funkcia s využitím čítača/časovača 0
void delay_ms(unsigned int time){

    TCNT0 = 156;
    i=0;
    // cakaj pokiaľ prebehne i krat pretecenie
    while(i != time*10);
}

int main(){

    DDRD |= (1 << PD7); // PD7 ako výstupný
    TCCR0 |= (1 << CS01); // preddelicka /1024 (1us)

    // prerušenie pri pretečení TCNT0
    TIMSK |= (1 << TOIE0);

    sei(); //povol globálne prerušenia

    while(1){
        PORTD ^= (1 << PD7); //neguj PD7

        // volanie nasej cakacej funkcie
        delay_ms(200);
    }

    return 0;
}
```

Youtube video:

8-bitový čítač/časovač 0 - čakacia funkcia

Príklad č.3:

Pomocou tlačidla privádzajte impulzy do externého vstupu T0 s reakciou na dobežnú hranu a napočítanú hodnotu (TCNT0) ,zobrazte pomocou 4 LED v binárnom tvare.

Zdrojový kód:

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

int main(){

    DDRC |= 0xFF; // PORTC ako výstupný

    DDRD &= ~(1 << PD4); // PD4 (T0) ako vstupný

    PORTD |= (1 << PD4); // PD4 (T0) do log 1.

    //dobežna hrana na T0
    TCCR0 |= (1 << CS02) | (1 << CS01);

    while(1){
        PORTC = TCNT0; // zobraz binárne napočítanú hodnotu

        if(TCNT0 > 15) TCNT0 = 0; // iba spodne 4 bity
    }

    return 0;
}
```

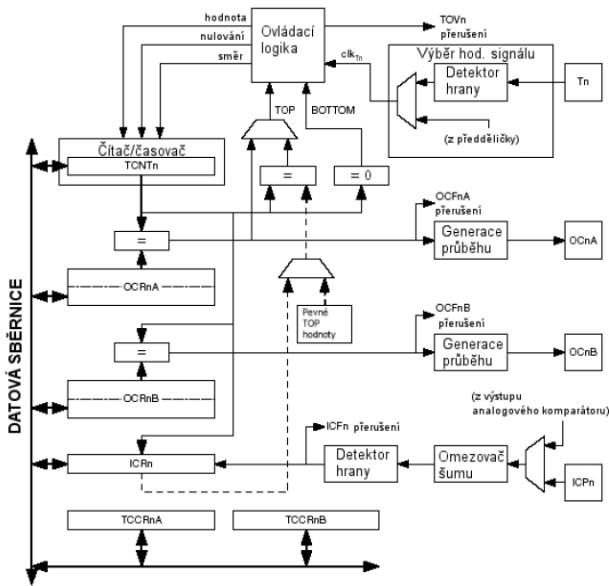
Youtube video:

8-bitový čítač/časovač 0 - externý zdroj hodín

**Čítač/časovač 1 (viď datasheet obvodu ATmega8 na strane 75)**

**Vlastnosti:**

- 16 bitový čítač/časovač
- 2 nezávislé porovnávacie jednotky OCRn
- Čítač vonkajších impulzov
- Generovanie PWM alebo frekvencie
- Zachytenie akt. hodnoty čítača/časovača 1 na udalosť Input Capture

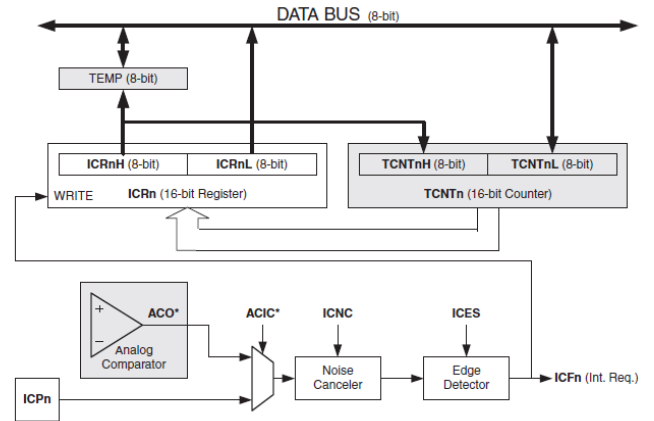


Obr 5.5 - Bloková schéma čítača/časovača 1

Čítač/časovač 1 obsahuje register TCNT1, do ktorého sa ukladá aktuálna hodnota čítača. Tento register je oproti TCNT0 16 bitový, čo znamená že počítadlo môže nadobúdať 65536 hodnôt. Znamená to teda, že bude pretekať pomalšie ako 8 bitový čítač/časovač, čo je pohodlnejšie z hľadiska generovania dlhších časov.

Obsah registra TCNT1 môžeme inkrementovať buď vnútorne (priamo alebo cez preddeličku), alebo externe cez vstupný pin T1. Ovládací register počítadla je TCCR1 je 16-bitový a preto je rozdelený na dva 8-bitové registre TCCR1A a TCCR1B. Obsah počítadla možno porovnávať s dvoma registrami OCR1A a OCR1B, ktoré sú tiež 16-bitové, čo sa využíva hlavne pri generovaní PWM signálu.

Čítač/časovač 1 má často využívanú funkciu uloženia stavu počítadla TCNT1 do registra ICR1 pri udalosti Input Capture (viď obr 5.6). Táto udalosť môže byť vyvolaná príchodom impulzu na pin ICP1 (Input Capture Pin) alebo výstupom z interného komparátora. Táto funkcia sa využíva hlavne na meranie dĺžky periódy externého signálu, frekvencie alebo striedy signálu.



Obr 5.6 - Bloková schéma Input Capture

## Popis registrov:

### TCCR1A (Timer/Counter 1 control register B)

– riadiaci register čítača/časovača 1  
Slúži na nastavenie režimu PWM, funkcia tohto registra bude vysvetlená v ďalšej kapitole.

### TCCR1B (Timer/Counter 1 control register B)

– riadiaci register čítača/časovača 1

Bit	7	6	5	4	3	2	1	0	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 5.7 - register TCCR1B

- ICNC1 (Input capture noise canceller) – pri povolení (log. 1) zapne obmedzovač šumu na pine ICP1
- ICES1 (Input Capture Edge Select) – nastaví na akú hranu vstupného signálu na pine ICP1 sa má vykonať zachytenie aktuálneho obsahu TCNT1 do registra ICR1. Pri zápise log. 1 je to reakcia na nábežnú hranu pri log. 0 na dobežnú hranu
- WGM13, WGM12 – slúži na nastavenie PWM módu
- CS12, CS11, CS10 – výber zdroja hodin a preddeličky (viď Obr 5.8)

CS12	CS11	CS10	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	clk <sub>IO</sub> /1 (No prescaling)
0	1	0	clk <sub>IO</sub> /8 (From prescaler)
0	1	1	clk <sub>IO</sub> /64 (From prescaler)
1	0	0	clk <sub>IO</sub> /256 (From prescaler)
1	0	1	clk <sub>IO</sub> /1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge
1	1	1	External clock source on T1 pin. Clock on rising edge

Obr 5.8 - výber preddeličky a zdroja hodín pre čítač/časovač 1

### TIMSK (Timer/counter Interrupt Mask Register)

– slúži na povolenie jednotlivých prerušení od čítača/časovača 1

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 5.9 - Register TIMSK

- TICIE1 (Timer/counter 1 Input Capture Interrupt Enable) – povolenie prerušenia po príchode impulzu na pin ICP1
- OCIE1A (Output compare A Interrupt Enable) – povolenie prerušenia pri zhode registra TCNT1 a OCR1A
- OCIE1B (Output compare B Interrupt Enable) – povolenie prerušenia pri zhode registra TCNT1 a OCR1B
- TOIE1 (Timer/counter 1 Overflow Interrupt Enable) – povolenie prerušenia pri pretečení alebo podtečený registra TCNT1

### Príklady

#### Príklad č.1:

Blikanie LED pomocou v intervale Ton=1s a Toff=1s ručným nastavením nastavením TCNT1 alebo využitím porovnávacieho registra OCR1A.

V programe sme zvolili preddeličku čítača/časovača 1 na hodnotu 256. To znamená, že register TCNT1 sa bude inkrementovať za čas 32us. Keď chceme aby prerušenie na udalosť pretečenia TCNT1 nastalo za 1sec, tak musí prísť 1s/32us = 31250 impulzov. Pretečenie nastane pri hodnote 65536 teda, keď maximálnu hodnotu čítača/časovača 1 (65535) inkrementujeme.

V prerušení teda nastavíme TCNT1 na hodnotu 65536 – 31250 = 34286

a pretečenie nastane vždy za čas 1s.

#### Zdrojový kód:

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

// pretečenie registra TCNT1
ISR (TIMER1_OVF_vect){

    // nastavenie počiatočnej hodnoty počítadla
    TCNT1 = 34286;
    PORTD ^= (1 << PD7); //neguj PD7
}

int main(){

    DDRD |= (1 << PD7); // PD7 ako výstupný

    TCCR1B |= (1 << CS12); //preddelička 256 (32us)

    // prerušenie pri pretečení TCNT1
    TIMSK |= (1 << TOIE1);

    OSCCAL = 0xA5; // nastavenie kalibracneho bajtu
    interneho RC oscilatora

    sei(); // povol globalne prerušenia

    while(1); // nekonečná slučka

    return 0;
}
```

#### Youtube video:

Čítač/časovač 1 - meranie času 1s

**Pozn.:** Podobný efekt môžeme dosiahnuť aj nastavením porovnávacieho registra OCR1A, ktorý nastavíme na hodnotu na 31250 – 1 = 31249. Pri zhode TCNT1 a OCR1A nastane prerušenie. V našom prípade tiež v 1s intervale.

#### Zdrojový kód:

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

// pretečenie pri zhode registrov OCR1A a TCNT1
ISR (TIMER1_COMPA_vect){
    TCNT1 = 0; // nuluj TCNT0
    PORTD ^= (1 << PD7); //neguj PD7
}

int main(){
    DDRD |= (1 << PD7); // PD7 ako výstupný

    TCCR1B |= (1 << CS12); //preddelička 256 (32us)
```

```
// prerušenie pri zhode OCR1A a TCNT1
TIMSK |= (1 << OCIE1A);
OCR1A = 31249; // TOP hodnota, pretečenie nastane
pri hodnote 31250

OSCCAL = 0xA5; // nastavenie kalibracneho bajtu
interneho RC oscilatora

sei(); // povol globalne prerušenia
while(1); // nekonečná slučka

return 0;
}
```

## Príklad č.2

Realizujte meranie času pomocou ICP1 pinu. Dobu v sekundách medzi dvoma stlačeniami, tlačidla bude reprezentovať počet bliknutí LED

### Zdrojový kód:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

volatile unsigned int namerane;

// akcia Input Capture
ISR (TIMER1_CAPT_vect){
    namerane = TCNT1;
    TCNT1 = 0;
}

int main(){
    unsigned char blik;

    DDRD |= (1 << PD7); // PD7 ako výstupný

    DDRB &= ~(1 << PB0); // PB0 (ICP1) ako vstupny
    PORTB |= (1 << PB0); //PB0 (ICP1) do log.1

    //preddelička 1024 (128us)
    TCCR1B |= (1 << CS12) | (1 << CS10);

    // prerušenie na udalosť Input Capture
    TIMSK |= (1 << TICIE1);

    // nastavenie kalibracneho bajtu int. RC oscilatora
    OSCCAL = 0xA5;

    sei(); // povol globalne prerušenia

    while(1){ // nekonečná slučka

        // ak sa stlačilo tlačidlo zablikaj LED
        if(namerane) {
```

```
// 7812 je počet impulzov TCNT1 za čas 1s
for(blik=0;blik < (namerane/7812);blik++){
    PORTD |= (1 << PD7);
    _delay_ms(200);
    PORTD &= ~(1 << PD7);
    _delay_ms(200);
}

namerane = 0;
}

return 0;
}
```

**Pozn:** Program bude korektne fungovať len do času menšieho ako pribl. 8 sec, nakoľko pri vyššom čase už pretečie obsah registra TCNT1

### Youtube video:

16 bitový čítač/časovač 1 - Meranie dĺžky času

## Príklad č.3:

Využitie čítača/časovača 0 a 1. Pomocou čítača/časovača 1 nastavte blikanie LED s frekvenciou pribl. 2Hz. Čítač/časovač 0 využite na moduláciu signálu pri zapnutí LED s frekvenciou pribl. 15Hz.

### Zdrojový kód:

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

volatile unsigned char blikaj;

// pretečenie TCNT0 za čas 32ms
ISR (TIMER0_OVF_vect){
    // ak blikaj tak blikaj :)
    if(blikaj) PORTD ^= (1 << PD7);
}

// pretečenie TCNT0 za čas 0,524288s
ISR (TIMER1_OVF_vect){
    blikaj ^= 1; // neguj premennu blikaj
}

int main(){

    DDRD |= (1 << PD7); // PD7 ako výstupný

    //preddelička č/č0 1024 (128us)
    TCCR0 |= (1 << CS02) | (1 << CS00);

    //preddelička č/č1 64 (8us)
    TCCR1B |= (1 << CS11) | (1 << CS10);
```



```
// prerušenie na udalost Input Capture
TIMSK |= (1 << TOIE0) |(1 << TOIE1);

// nastavenie kalibr. bajtu interneho RC oscilatora
OSCCAL = 0xA5;

sei(); // povol globalne prerušenia

while(1);

return 0;
}
```

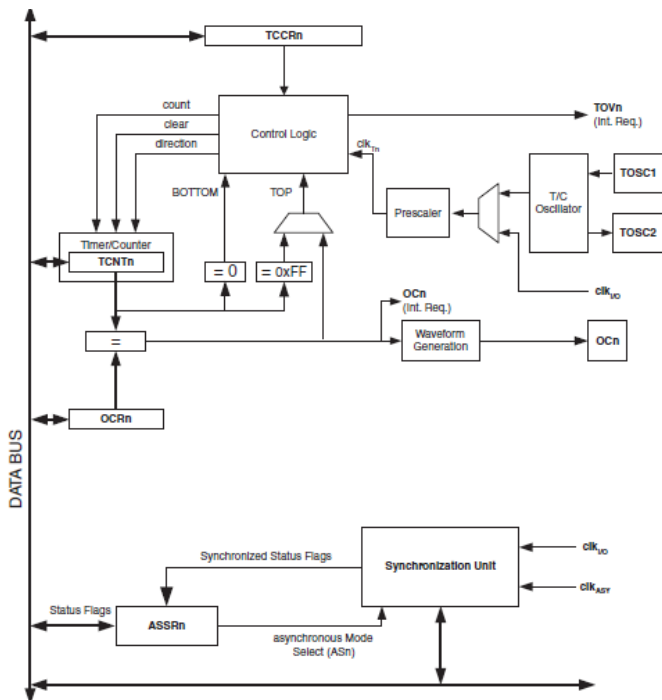
Youtube video:

Využitie čítača/časovača 0 a 1

## Čítač/časovač 2 (vid' datasheet obvodu ATmega 8 na strane 102)

### Vlastnosti:

- 8 bitový
- Pracuje ako čítač/časovač 0
- Možnosť taktovania asynchrónne z externého kryštálu 32,768 kHz pripojeného na piny TOSC1 a TOSC2. Využitie ako RTC (hodiny)
- Možnosť generovania PWM signálu alebo frekvencie



Obr 5.10 - bloková schéma čítača/časovača 2

Aktuálny stav čítača/časovača 2 sa ukladá do 8 bitového registra TCNT2. Tento čítač/časovač 2 má hlavne využitie ako RTC - zdroj reálneho času.

Keď potrebujeme prerušenie na udalosť pretečenia TCNT2 každú sekundu tak preddeličku nastavíme na 128, pripojíme na piny TOSC1 a TOSC2 externý oscilátor 32,768 kHz a prerušenie nastane každú sekundu pretože  $32,768 \text{ kHz} / (128 * 256) = 1\text{s}$

### Popis registrov :

#### TCCR2 (Timer/Counter 2 control register):

Bit	7	6	5	4	3	2	1	0	
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 5.11 - register TCCR2

Bity CS22,CS21,CS20 slúžia na výber výber preddeličky. Ostatné bity slúžia na nastavenie PWM.

Table 46. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$\text{clk}_{T2S}$ /(No prescaling)
0	1	0	$\text{clk}_{T2S}/8$ (From prescaler)
0	1	1	$\text{clk}_{T2S}/32$ (From prescaler)
1	0	0	$\text{clk}_{T2S}/64$ (From prescaler)
1	0	1	$\text{clk}_{T2S}/128$ (From prescaler)
1	1	0	$\text{clk}_{T2S}/256$ (From prescaler)
1	1	1	$\text{clk}_{T2S}/1024$ (From prescaler)

Obr 5.12 - Volba preddeličky čítača/časovača 2

#### ASSR2 (Asynchronous Status Register):

Bit	7	6	5	4	3	2	1	0	
	-	-	-	-	AS2	TCN2UB	OCR2UB	TCR2UB	ASSR
Read/Write	R	R	R	R	R/W	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Obr 5.13 - register ASSR2

- AS2 (Asynchronous Timer/Counter 2) – pri zápise log. 0 na tento bit je čítač/časovač taktovaný z frekvencie hodín. Pri zápise log. 1 je taktovaný z externého kryštálu 32,768 kHz pripojeného na piny TOSC1 a TOSC2.

#### TIMSK (Timer/counter Interrupt Mask Register):

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

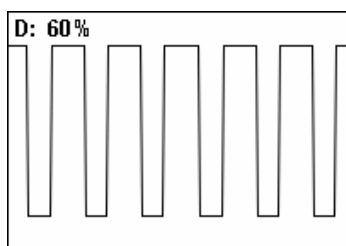
Obr 5.14 - register TIMSK

- OCIE2 (Timer/counter 2 Output compare Interrupt) – povolenie prerušenia pri zhode registrov TCNT2 a OCR2
- TOIE2 (Timer/counter 2 Overflow Interrupt) – povolenie prerušenia pri pretečení registra TCNT2

# ČÍTAČE/ČASOVAČE - PWM

## Úvod k impulzno-šírkovej modulácii (PWM)

Význam PWM na rozdiel od spojitej regulácie je hlavne v eliminovaní strát na regulátore. Pri PWM regulujeme dodávaný výkon do záťaže pomocou striedy. Strieda je pomer času zopnutia ku celkovej perióde spínacieho prvku.



Obr 6.0 - PWM signál

$$D = (t_{on} / t_p)$$

D- strieda

T<sub>on</sub> – čas zopnutia

T<sub>p</sub> - perióda

### Pre lepšie porozumenie uvediem príklad:

Nech máme napájacie napätie 12V a žiarovku s parametrami 6V/1A. Pri klasickej spojitej regulácii by sme na regulátore (tranzistore) museli vytvoriť úbytok napätia 6V pri prúde 1A, čo by spôsobilo tepelnú stratu na tranzistore 6W. Efektivita tejto regulácie by bola teda 50%, teda rovná polovica energie by sa premenila na teplo.

Pri PWM regulácii vytvoríme striedu D=0,5, teda tranzistor bude zopnutý polovicu času. Straty na tranzistore budú najvyššie pri zopínaní a rozopínaní tranzistora. Pri zopnutom tranzistore ním bude tiecť prúd, ale tranzistor bude v saturácii, takže sa na ňom vytvorí malý úbytok napätia, tým pádom bude aj výkonová strata malá. Pomocou PWM teda môžeme dosiahnuť vysokú účinnosť regulácie (obvykle nad 90%).

Pozn.: Pri takomto zapojení však nebude na výstupe konštantné napätie 6V! Na výstupe bude určitú časť periódy napájacie napätie 12V ale žiarovka sa neprepáli vďaka jej tepelnej zotrvačnosti. Polovodičové

prvky by však takéto pulzy asi neprežili, preto v týchto prípadoch treba na výstup zaradiť RC filter ktorý výstupne napätie vyhladí na hodnotu napätia danú veľkosťou striedy. Táto regulácia má v súčasnej dobe veľké využitie v elektronike. Využívame ju hlavne na reguláciu žiaroviek, motorov, v spínaných zdrojoch alebo dokonca v audio technike ako zosilňovač triedy D.

## Generovanie PWM pomocou AVR mikrokontroléra

Pri práci s PWM využívame registre TCCR1A a TCCR1B čítača/časovača 1, pomocou ktorých nastavíme režim PWM, pracovnú frekvenciu a správanie výstupných pinov OC1A a OC1B.

**Popis registrov (viď datasheet obvodu ATmega8 na strane 96):**

### TCCR1A (Timer/counter 1 control register A) - Využíva sa na nastavenie PWM:

Bit	7	6	5	4	3	2	1	0	TCCR1A
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 6.1 - Register TCCR1A

- COM1A1, COM1A0 – slúžia na nastavenie správania výstupného pinu OC1A
- COM1B1, COM1B0 – slúžia na nastavenie správania výstupného pinu OC1B

Nastavenie týchto bitov má odlišnú funkciu v závislosti od nastavenia režimu PWM.

### Normálny režim:

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level)
1	1	Set OC1A/OC1B on Compare Match (Set output to high level)

Obr 6.2 - Správanie výstupných pinov pri norm. režime

Pri normálnom režime (nie v režimoch PWM), môžeme nastaviť správanie pinov OC1A a OC1B pri zhode porovnávacieho registra OCR1n s registrom TCNT1 nasledovne:

- (0,0) Piny OC1A a OC1B sú odpojené od čítača
- (0,1) Zmena log. úrovne pri zhode registrov
- (1,0) Nastaví výstup OC1A/OC1B do log.0 pri zhode registrov
- (1,1) Nastaví výstup OC1A/OC1B do log.1 pri zhode registrov

### Režim rýchlej PWM (Fast PWM):

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM, (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM, (inverting mode)

Obr 6.3 - Správanie výstupných pinov Fast PWM

- (0,0) Piny OC1A a OC1B sú odpojené od čítača
- (0,1) Zmena log. úrovne na pine OC1A pri režime 15, OC1B odpojený. V ostatných režimoch sú OC1A a OC1B odpojené od čítača
- (1,0) Nastaví výstup OC1A/OC1B do log.0 pri zhode registrov a log.1 pri BOTTOM hodnote.
- (1,1) Nastaví výstup OC1A/OC1B do log.1 pri zhode registrov a log 0 pri BOTTOM hodnote.

### Režim Phase correct a Phase and Frequency correct PWM:

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 9 or 14: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match when up-counting. Set OC1A/OC1B on Compare Match when downcounting.
1	1	Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when downcounting.

Obr 6.4 - Správanie výstupných pinov pri Phase correct a frequency correct PWM

- (0,0) Piny OC1A a OC1B sú odpojené od čítača
- (0,1) Zmena log. úrovne na pine OC1A pri režime 9 a 14, OC1B odpojený od čítača. V ostatných režimoch sú OC1A a OC1B odpojené
- (1,0) Nastaví výstup OC1A/OC1B do log.0 pri zhode registrov počas inkrementácie a log. 1 po-

čas dekrementácie registra TCNT1

- (1,1) Nastaví výstup OC1A/OC1B do log.1 pri zhode registrov počas inkrementácie a log. 0 počas dekrementácie registra TCNT1

FOC1A, FOCA1B – pri zapísaní log.1 vyvolá compare match udalosť (FOC1A na OC1A a FOC1A na OC1B) s tým rozdielom že na tuto udalosť reaguje len Waveform Generation jednotka, ktorá riadi výstupné OCx piny, takže sa nenastaví flag OCFx a nevyvolá sa prerušenie a takisto sa nevykoná vynulovanie hodnoty čítača v CTC móde.

### WGM11,10 a WGM13,12 s registra TCCR1B (Waveform generation mode):

Z tabuľky vidno, že čítač/časovač 1 na ATmega8 dokáže pracovať v 15 režimoch v závislosti od nastavenia bitov WGM. Z toho sú 3 základné režimy PWM (Fast PWM, Phase correct PWM a Phase and Frequency correct PWM) a jeden režim pre generovanie frekvencie – CTC.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	--	--	--
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

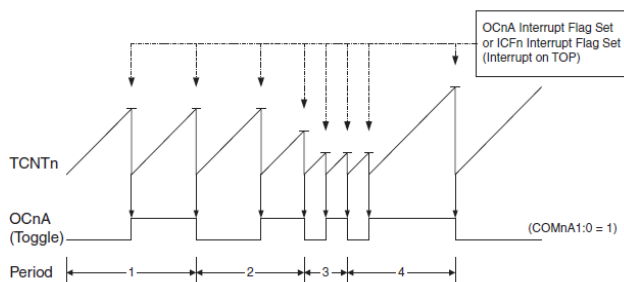
Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Obr 6.5 - PWM režimy MCU

## Režimy (viď datasheet obvodu ATmega8 na strane 87)

### 1) CTC režim (Clear timer on compare match)

Je charakterizovaný tým, že obsah čítača TCTN1 je vynulovaný, ak jeho hodnota dosiahne hodnotu uloženú v OCR1A (režim 4), alebo ICR1 (režim 12). Hodnota v OCR1A, alebo ICR1 udáva TOP, teda najvyššiu hodnotu čítača.



Obr 6.6 - Časový diagram CTC režimu

Žiadosť o prerušenie môže byť generovaná vždy, keď obsah čítača dosiahne hodnotu TOP definovanú obsahom OCR1A, alebo ICR1. Ak je prerušenie povolené, potom v rámci obslužnej rutiny môže byť zmenená príslušná hodnota TOP.

Výstupnú frekvenciu vypočítame podľa vzťahu:

$$f_{OCnA} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCRnA)}$$

, kde  $f_{clk}$  je frekvencia hodín, N je preddelička a OCRnA je hodnota porovnávacieho registra.

## Príklady

### Príklad č.1: Striedaj 5 periód s frekvenciou 1000 Hz a 500 Hz.

Využijeme CTC režim. Podľa vzorca vypočítame, že hodnota OCR1A registra bude 499 pre frekvenciu 1000Hz a 249 pre 2000Hz, pri nastavenej preddeličke 8 a frekvencií hodín 8 MHz. V prerušení na udalosť zhody OCR1A a TCNT1 registra po 5 periódach striedame hodnoty OCR1A registra, čím striedame aj frekvencie 1000 a 2000Hz.

### Zdrojový kód:

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

```
// pocet period danej frekvencie
volatile unsigned char n;
```

```
//pretečenie TCNT1
ISR (TIMER1_COMPA_vect){
    // 5 period (dve zmeny)
    if(n == 10){
        // zmen hodnotu OCR1A po 5 period
        OCR1A = ((OCR1A == 499) ? 249 : 499);
        n=0;
    }
    n++;
}
```

```
int main(){
```

```
    DDRB |= (1 << PB1);    // OC1A ako výstupný
```

```
    // REZIM 4 (CTC), nastavenie zmeny log. úrovne
    // pri zhode OCR1A a TCNT1
    TCCR1A |= (1 << COM1A0);
```

```
    // preddelička 8 pri 8 MHz = (1us)
    TCCR1B |= (1 << WGM12) | (1 << CS11);
```

```
    OCR1A |= 499;
    //povol prerušenie pri zhode OCR1A a TCNT1
    TIMSK |= (1 << OCIE1A);
```

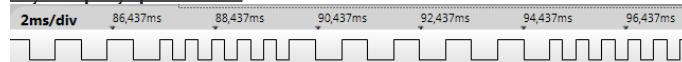
```
    OSCCAL = 0xA5;
    // nastavenie kalibr. bajtu interneho RC oscilatora
```

```
    sei(); // povol globalne prerušenia
```

```
    while(1); // nekonečná slučka
```

```
    return 0;
}
```

### Výstupný priebeh:

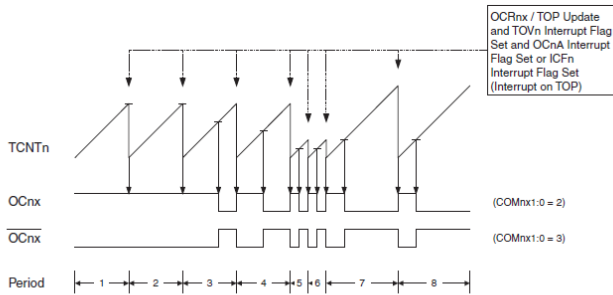


Obr 6.7 - Výstupný priebeh signálu s príkladu č.1

## 2) Režim " rýchla PWM" (Fast Pulse Width Modulation)

Režimy z tabuľky: 5, 6, 7, 14 alebo 15.

Generuje impulzný šírko modulovaný signál (PWM). V tomto režime sa obsah čítača zvyšuje od hodnoty BOTTOM do hodnoty TOP. V neinvertujúcom režime sa výstup OC1x nastaví pri zhode obsahov čítača TCNT1 a OCR1x a vynuluje sa pri dosiahnutí hodnoty TOP. Pri invertujúcom režime je to obrátene (viď časť registre).



Obr 6.8 - Časový diagram Fast PWM režimu

Rozlíšenie rýchlej PWM môže byť pevné, 8, 9, alebo 10 bitov, prípadne definované obsahom ICR1, alebo OCR1A. Minimálne povolené rozlíšenie je 2 bity (ICR1, alebo OCR1A je nastavené na hodnotu 0x0003). Maximálne rozlíšenie je 16 bitov (ICR1, alebo OCR1A je nastavené na hodnotu MAX).

Rozlíšenie PWM je dané nasledujúcim vzťahom:

$$R_{PWM} = \frac{\log(TOP + 1)}{\log(2)}$$

V režime rýchlej PWM sa čítač inkrementuje pokiaľ jeho hodnota nenadobudne jednu z definovaných hodnôt 0x00FF, 0x01FF, alebo 0x03FF (režimy 5, 6, alebo 7), hodnotu v ICR1 (režim 14), alebo hodnotu v OCR1A (režim 15). V nasledujúcom hodinovom cykle po dosiahnutí definovanej hodnoty sa jeho obsah vynuluje.

Príznak pretečenia sa nastaví, keď obsah čítača dosiahne hodnotu TOP. Ak je obsluha prerušenia povolená, potom pomocou obslužného programu prerušenia môžeme zmeniť hodnotu TOP, prípadne porovnávaciú hodnotu.

Frekvenciu PWM môžeme určiť na základe nasledujúceho vzťahu:

$$f_{OC1xPWM} = \frac{f_{clkIO}}{N(1 + TOP)}$$

, kde fclk je frekvencia hodín, N je preddelička a TOP je max. hodnota čítača/časovača.

## Príklad č.1: Plynule rozsvecovanie žiarovky

Použitie je 8-bitové PWM. Zmena striedy sa realizuje v nekonečnej slučke v 10ms intervaloch.

### Zdrojový kód:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

int main(){

    DDRB |= (1 << PB1); // OC1A ako výstupný

    // REZIM 5 - Fast PWM 8bit,
    TCCR1A |= (1 << COM1A1) | (1 << WGM10);

    // preddelička 8 pri 8 MHz = (1us)
    TCCR1B |= (1 << WGM12) | (1 << CS11);
    OCR1A = 0;

    // nastavenie kalibr. bajtu interneho RC oscilatora
    OSCCAL = 0xA5;

    sei(); // povol globalne prerušenia

    while(1){ // nekonečná slučka
        // rozsvecuj žiarovku zmenou striedy
        OCR1A++;
        if(OCR1A >= 255) OCR1A=0;
        _delay_ms(10);
    }
    return 0;
}
```

### Youtube video:

PWM - plynulé rozsvecovanie žiarovky - link

## Príklad č.2: Ovládanie jasu žiarovky pomocou 2 tlačidiel

Využili sme vedomosti z kapitoly venujúcej sa externému prerušeniu. Dve tlačidlá sú použité ako zdroj externého prerušenia piny (INT0 a INT1). Prvým tlačidlom pridávame hodnotu striedy, druhým uberáme. Použitie je 8-bitové PWM.

### Zdrojový kód

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

ISR(INT0_vect){
    if(OCR1A < 245) OCR1A+=10; //pridaj striedu
}
```

```
ISR(INT1_vect){
    if(OCR1A > 10) OCR1A-=10; // uber striedu
}

int main(){
    DDRB |= (1 << PB1);      // OC1A ako výstupný

    // REZIM 5 - Fast PWM 8bit,
    TCCR1A |= (1 << COM1A1) | (1 << WGM10);

    // predmelička 8 pri 8 MHz = (1us)
    TCCR1B |= (1 << WGM12) | (1 << CS11);

    // PD2,PD3 ako vstupné (ext. prerušenia)
    DDRD &= ~(1 << PD2) | (1 << PD3));

    // Zapnutie interného pull-up rezistora
    PORTD |= (1 << PD2) | (1 << PD3);

    MCUCR |= (1 << ISC01); // dobežná hrana INT0
    MCUCR |= (1 << ISC11); // dobežná hrana INT1
    // povol prerušenia od INT1 a INT0
    GICR |= (1 << INT1) | (1 << INT0);

    // nastavenie kalibr. bajtu interneho RC oscilatora
    OSCCAL = 0xA5;

    sei(); // povol globalne prerušenia
    while(1);

return 0;
}
```

## Youtube video:

Ovládanie jasú žiarovky pomocou dvoch tlačidiel

## Príklad č.3: Generovanie sínusu

Na začiatku vygenerujeme pole hodnôt cez funkciu `generate_sin()`; kde vstupným parametrom je želaná frekvencia. Sínusovka sa rozkúskuje na časti o dĺžke 32.768us, čo je vzorkovacia frekvencia a k týmto časťam sínusovky sa prideli hodnota striedy. Potom cez prerušenie na udalosť pretečenia TCNT1 sa postupne tieto hodnoty nahrávajú do registra OCR1.

Aby sme dostali sínusový priebeh musíme na výstup OC1A pripojiť dolnopriepustný RC filter, vypočítaný približne na dvojnásobnú frekvenciu zadaného signálu  $f = 1/(2 \cdot \pi \cdot R \cdot C)$ . S rastúcou frekvenciou klesá počet vzoriek na periódu. Pri frekvencii 1000Hz je to zhruba 30 vzoriek na periódu.

V príklade som generoval frekvenciu 1000Hz, teda hodnoty prvkov mi vyšli  $R=1k\Omega$ ,  $C=82nF$

## Zdrojový kód:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>

#define sample_rate 32.768e-6
volatile unsigned char i;
volatile unsigned char sine[256];

// vygeneruje pole s hodnotami sinus
void generate_sin(int freq){
    for(unsigned char x=0; x<255;x++){
        sine[x] = 128*(sin(2.0*M_PI*x*sample_rate*freq) + 1);
    }
}

//pretečenie nastane za čas 32,768us
ISR(TIMER1_OVF_vect){
    //generuj sinus
    OCR1A = sine[i];
    i++;
}

int main(){
    // generuj sinus o frekvencii 1000Hz
    generate_sin(1000);

    DDRB |= (1 << PB1);      // OC1A ako výstupný

    // REZIM 5 - Fast PWM 8bit, sample rate bude 32,768us
    TCCR1A |= (1 << COM1A1) | (1 << COM1A0)
              | (1 << WGM10);

    // predmelička 1 pri 8 MHz = (128ns)
    TCCR1B |= (1 << WGM12) | (1 << CS10);

    TIMSK |= (1 << TOIE1); //prerušenie na pretečenie

    OSCCAL = 0xA5;
    // nastavenie kalibr. bajtu interneho RC oscilatora

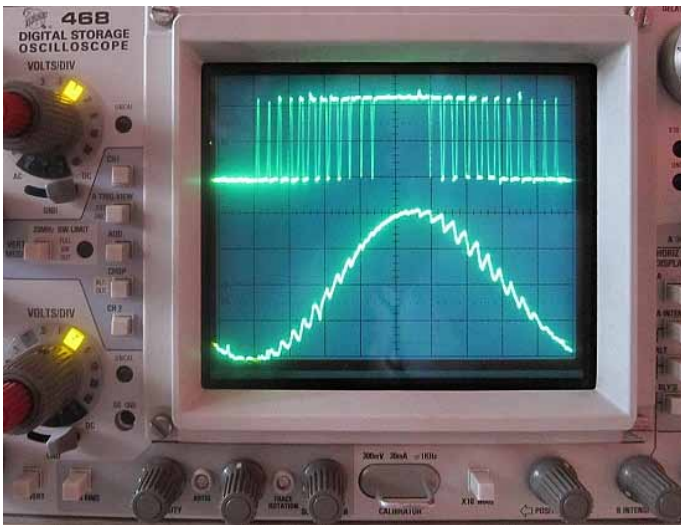
    sei(); // povol globalne prerušenia

    while(1);

return 0;
}
```

## Výstupný priebeh:

Horný priebeh je PWM signál na pine OC1A a spodný priebeh je sínus získaný z PWM cez dolnopriepustný RC filter.

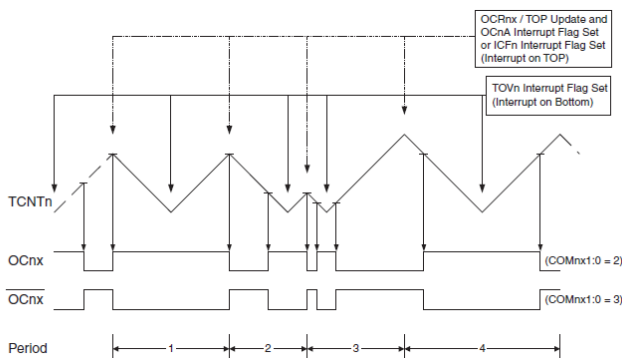


Obr 6.9 - výstupný priebeh PWM signálu s príkladu č. 3

### 3) fázovo korektná PWM (Phase correct PWM)

Režimy 1,2,3,10,11.

V tomto režime sa hodnota čítača zvyšuje (inkrementuje) od hodnoty BOTTOM (0x0000) až po hodnotu TOP a potom sa následne dekrementuje po hodnotu BOTTOM. V neinvertujúcom výstupnom porovnávacom režime je výstup OC1x vynulovaný v prípade rovnosti obsahov TCNT1 a OCR1x pri počítaní smerom nahor a nastavuje sa pri počítaní smerom nadol. V invertujúcom režime je situácia opačná. Maximálna dosiahnuteľná základná frekvencia fázovo korektnej PWM je nižšia než pri rýchlej PWM.



Obr 6.10 - Časový diagram režimu fázovo korektnej PWM

Tento režim je často používaný pri riadení motorov. Rozlíšenie PWM je pevné 8, 9, alebo 10 bitov, prípadne definované obsahom registrov ICR1, alebo OCR1A. Najmenšie použiteľné rozlíšenie je 2 bity (ICR1, alebo OCR1A je nastavené na hodnotu 0x0003). Maximál-

ne rozlíšenie je 16 bitov (ICR1, alebo OCR1A je nastavené na hodnotu MAX).

Rozlíšenie PWM je dané nasledujúcim vzťahom:

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

Ak je obsluha prerušenia povolená prerušenie môže v závislosti od nastavenia nastať pri hodnote TOP alebo BOTTOM. V obslužnom programe prerušenia môžeme potom zmeniť hodnotu TOP, prípadne porovnávaciu hodnotu.

Pri definovaní novej TOP hodnoty je potrebné zabezpečiť, aby bola väčšia, alebo rovná než hodnoty v porovnávacích registroch.

Frekvenciu fázovo korektnej PWM môžeme určiť na základe nasledujúceho vzťahu:

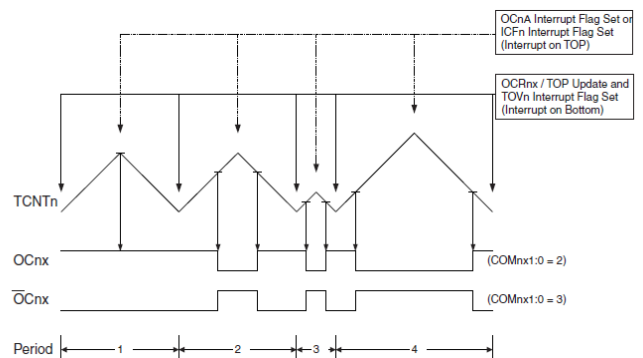
$$f_{OC1xPWM} = \frac{f_{clkIO}}{2NTOP}$$

, kde N je preddelička, TOP je max. hodnota čítača/časovača a fclk je frekvencia hodín.

### 4) fázovo a frekvenčne korektná PWM (Phase and frequency correct PWM)

Sú to režimy 8 a 9.

V tomto režime sa hodnota čítača inkrementuje od hodnoty BOTTOM (0x0000) až po hodnotu TOP a potom sa následne dekrementuje po hodnotu BOTTOM. V neinvertujúcom režime je pri počítaní smerom nahor výstup OC1x nulovaný v prípade rovnosti obsahov TCNT1 a OCR1x. Pri dekrementácii sa výstup nastaví v prípade zhody do log.1 V invertujúcom režime je situácia opačná.



Obr 6.11 - Časový diagram fáz. a frek. korektnej PWM

Maximálna dosiahnuteľná základná frekvencia fázovo a frekvenčne korektnej PWM je nižšia než pri rýchlejšej PWM. Tento režim je často používaný pri riadení motorov. Rozlíšenie PWM je definované obsahom registrov ICR1, alebo OCR1A. Najmenšie použiteľné rozlíšenie je 2 bity. Vtedy je ICR1, alebo OCR1A nastavené na hodnotu 0x0003. Maximálne rozlíšenie je 16 bitov, ak ICR1, alebo OCR1A je nastavené na hodnotu MAX.

Rozlíšenie PWM je dané nasledujúcim vzťahom:

$$R_{FPWM} = \frac{\log(TOP + 1)}{\log(2)}$$

V režime fázovo a frekvenčne korektnej PWM je obsah čítača inkrementovaný až do okamžiku, v ktorom nedosiahne jednu z hodnôt nastavených v registri ICR1 (režim 8), alebo v registri OCR1A (režim 9). Ak čítač dosiahne hodnotu TOP zmení smer počítania. Obsah čítača bude rovný hodnote TOP práve jeden hodinový cyklus.

Na definovanie hodnoty TOP použitý register OCR1A, alebo ICR1. Príznaky prerušenia môžu byť využité na generovanie prerušení v časových okamžikoch, v ktorých obsah TCNT1 dosiahne hodnotu TOP, alebo BOTTOM.

Frekvenciu fázovo a frekvenčne korektnej PWM môžeme určiť na základe nasledujúceho vzťahu:

$$f_{OCR1A/PWM} = \frac{f_{clk}}{2NTOP}$$

, kde fclk je frekvencia hodín, N je preddelička a TOP je max. hodnota čítača.

### Príklady

**Príklad č.1: Generujte PWM signál pomocou režimu 3 (fázovo aj frekvenčne korektné PWM)**

Jedná sa o 10-bitové PWM, takže TOP hodnota bude 1024. Obsah porovnávacieho registra OCR1A nesmie presiahnuť teda túto hodnotu. Pri frekvencii hodín a preddeličke 8 bude TCNT1 inkrementovaný za čas 1us. Pretečenie nastane za čas 2048us, pretože z max. hodnoty neprejde čítač/časovač do hodnoty 0 ale de-

krementuje sa, takže čas treba násobiť dvoma.

### Zdrojový kód:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

int i;

//pretečenie nastane za čas 2048us
ISR(TIMER1_OVF_vect){
    OCR1A = i;
    i+=10;
    if(i>1024) i=0;
}

int main(){

    DDRB |= (1 << PB1); // OC1A ako výstupný

    // REZIM 3 - Phase and Freq. correct PWM - 10bit
    TCCR1A |= (1 << COM1A1) | (1 << COM1A0)
    | (1 << WGM10) | (1 << WGM11);

    // preddelička 8 pri 8 MHz = (1us)
    TCCR1B |= (1 << CS11);

    TIMSK |= (1 << TOIE1); //prerušenie na pretečenie

    OSCCAL = 0xA5;
    // nastavenie kalibr. bajtu interneho RC oscilatora

    sei(); // povol globalne prerušenia

    while(1);

    return 0;
}
```

### Výstupný priebeh:



### Dodatok

Na záver dodám, že ATmega8 dokáže generovať PWM aj pomocou čítača/časovača 2, princíp ostáva rovnaký avšak nastavenie registrov je odlišné.



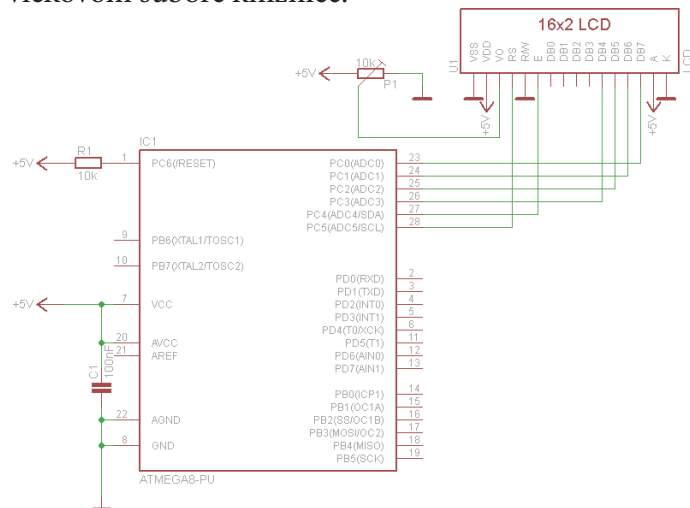
# PRÁCA SO ZNAKOVÝM LCD DISPL.

## Pripojenie znakového LCD displeja k MCU

Znakový LCD displej obsahuje obvod (nazývaný radič), ktorý nám uľahčuje prácu s displejom. Obsahuje komunikačné rozhranie, preddefinovanú sadu znakov atď. Nám potom stačí poslať cez komunikačné rozhranie niekoľko príkazov na vypísanie znakov na displej. Najčastejšie používaným radičom na znakové LCD displeje sa stal legendárny HD47780.

### Schéma zapojenia LCD displeja:

Dátové linky môžeme zapojiť na ktorýkoľvek PORT mikropočítača. Prepojenie potom definujeme v hlavičkovom súbore knižnice.



Obr 7.0 - Pripojenie displeja s radičom HD47780 ku MCU

### Popis pinov displeja:

- Pin č.1 (VSS) – GND
- Pin č.2 (VDD) – napájacie napätie v rozsahu 3,3 až 5V
- Pin č.3 (VO) – na tento pin pripájame trimer ktorým nastavujeme kontrast displeja.
- Pin č.4 (RS) – dátový pin (Register select), určuje či chceme cez dátové vodiče prenášať inštrukcie alebo dáta
- Pin č.5 (R/W) – nastavujeme ním či chceme do displeja zapisovať alebo z neho čítať. Zvyčajne využívame len zápis, preto tento pin pripájame na GND.

- Pin č.6 (E) – dátový pin (Enable), zapnutie/vypnutie LCD displeja
- Piny č. 7-14 (DB0-DB7) – obojsmerná dátová linka
- Pin č. 15 – Anóda (+) podsvietenia LCD displeja
- Pin č. 16 – Katóda (-) podsvietenia LCD displeja

## Práca s displejom pomocou MCU

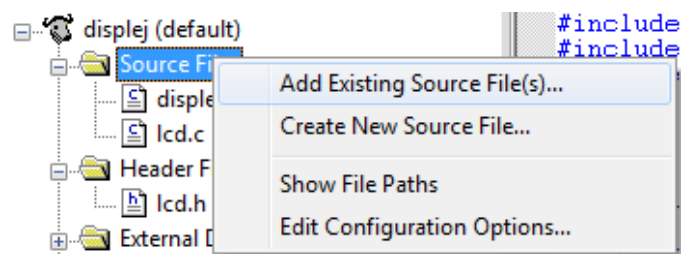
Nakoľko na internete existuje množstvo knižníc pre prácu so znakovými displejmi, teda jeho ovládanie oveľa jednoduchšie ako by ste možno čakali :)

Knižnica ktorú používam obsahuje dva súbory `lcd.c` a `lcd.h`, ktorá obsahuje funkcie na jednoduché ovládanie LCD displeja.

Tieto súbory si môžete stiahnuť s nasledovného linku: <http://svetelektro.com/Pictures/Microprocesory/avr/6/lcd.zip>

### Pre využívanie tejto knižnice v našom projekte treba spraviť nasledovné:

1. Vytvoriť nový projekt (viď 1. kapitola) alebo otvoriť existujúci projekt v AVR studio 4.
2. Do adresára kde sa nachádza náš projekt skopírujeme súbory `lcd.c` a `lcd.h`
3. Tieto súbory je treba pridať aj do nášho projektu v AVR studio 4. Klikneme pravým tlačidlom na položku 'Source Files' a zvolíme možnosť "Add existing source files" a vyberieme súbor `lcd.c`. Obdobným spôsobom pridáme aj hlavičkový súbor `lcd.h` položku "Header files"



Obr 7.1 - Pridanie súborov do projektu

4. Nakoniec pridáme hlavičkový súbor do hlavného programu pomocou direktívy `#include "lcd.h"`

# PRÁCA SO ZNAKOVÝM LCD DISPLEJOM

## Nastavenie hlavičkového súboru lcd.h:

Služi na nastavenie práce s LCD displejom. V mnohých prípadoch stačí nastaviť len počet znakov na riadok, počet riadkov a rozloženie pinov na ovládanie displeja. Pokiaľ máte LCD displej 2x16 znakov a je zapojený podľa hore uvedenej schémy, netreba v tomto súbore nič nastavovať.

```
/**
 * @name Definitions for Display Size
 * Change these definitions to adapt setting to your display
 */
#define LCD_LINES 2 /**< number of visible lines of the display
#define LCD_DISP_LENGTH 16 /**< visible characters per line of the di
#define LCD_LINE_LENGTH 0x40 /**< internal line length of the display
#define LCD_START_LINE1 0x00 /**< DDRAM address of first char of line 1
#define LCD_START_LINE2 0x40 /**< DDRAM address of first char of line 2
#define LCD_START_LINE3 0x14 /**< DDRAM address of first char of line 3
#define LCD_START_LINE4 0x54 /**< DDRAM address of first char of line 4
#define LCD_WRAP_LINES 0 /**< 0: no wrap, 1: wrap at end of visible
```

Obr 7.2 - Nastavenie hlavičkového súboru lcd.h

```
#define LCD_PORT PORTC /**< port for the LCD lines */
#define LCD_DATA0_PORT LCD_PORT /**< port for 4bit data bit 0 */
#define LCD_DATA1_PORT LCD_PORT /**< port for 4bit data bit 1 */
#define LCD_DATA2_PORT LCD_PORT /**< port for 4bit data bit 2 */
#define LCD_DATA3_PORT LCD_PORT /**< port for 4bit data bit 3 */
#define LCD_DATA0_PIN 3 /**< pin for 4bit data bit 0 */
#define LCD_DATA1_PIN 2 /**< pin for 4bit data bit 1 */
#define LCD_DATA2_PIN 1 /**< pin for 4bit data bit 2 */
#define LCD_DATA3_PIN 0 /**< pin for 4bit data bit 3 */
#define LCD_RS_PORT PORTC /**< port for RS line */
#define LCD_RS_PIN 5 /**< pin for RS line */
#define LCD_E_PORT PORTC /**< port for Enable line */
#define LCD_E_PIN 4 /**< pin for Enable line */
```

Obr 7.3 - Nastavenie hlavičkového súboru lcd.h

Knižnica obsahuje nasledovné funkcie na prácu s LCD displejom:

- `lcd_init(uint8_t dispAttr);` - inicializácia LCD displeja, ako parameter funkcie možno zvoliť nasledovné možnosti:
- `LCD_DISP_OFF` - displej vypnutý
- `LCD_DISP_ON` - displej zapnutý, kurzor vypnutý
- `LCD_DISP_ON_CURSOR` - displej zapnutý, kurzor zapnutý
- `LCD_DISP_ON_CURSOR_BLINK` - displej zapnutý, kurzor blinká
- `lcd_clrscr(void);` - vymazanie LCD displeja
- `lcd_home(void);` - nastaví kurzor na domovskú pozíciu
- `lcd_gotoxy(uint8_t x, uint8_t y);` - nastavenie kurzora na zadanú pozíciu x a y displeja (indexovanie je od 0)
- `lcd_putc(char c);` - zobraz znak
- `lcd_puts(const char *s);` - zobraz reťazec znakov
- `lcd_puts_p(const char *progmem_s);` - zobraz reťazec znakov umiestnený vo Flash pamäti
- `lcd_command(uint8_t cmd);` - vyšle príkaz na displej

## Príklady

### Príklad č.1 - prvý text:

Výpis textu na displej do dvoch riadkov

### Zdrojový kód:

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

int main(){

    lcd_init(LCD_DISP_ON); // inicializacia displeja
    lcd_puts("Hello World\nSvetelektro.com"); // vypis text

    return 0;
}
```



Obr 7.4 - Ukážka z príkladu č. 1

### Príklad č.2 – využitie funkcie gotoxy(x,y);

Pomocou funkcie `gotoxy(x,y);` posuň text na vhodnú pozíciu

### Zdrojový kód

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

int main(){

    lcd_init(LCD_DISP_ON); // inicializacia displeja
    lcd_gotoxy(4,0); // chod na poziciu x=4, y=0
    lcd_puts("Posunute"); // vypis text

    return 0;
}
```



Obr 7.5 - Ukážka z príkladu č. 2

## Výpis čísel na displej:

Pokiaľ chceme zobrazit' číslo na displeji, musíme ho skonvertovať na reťazec znakov. Pokiaľ ide o celé čísla využívame funkciu itoa() alebo sprintf().

Funkcia itoa ma nasledovné parametre -  
itoa(int value, char \*string, int radix);

- int value – premenná typu int v ktorej sa nachádza číslo ktoré chceme skonvertovať
- char \*string – ukazovateľ na reťazec znakov do ktorého sa zapíše naše číslo
- int radix – základ čísla, určíme v akej sústave má byť výsledné číslo

## Príklad č.1 – práca s číslami pomocou funkcie itoa();

```
#include <avr/io.h>
#include <stdlib.h>
#include "lcd.h"
```

```
int main(){

    char text[16]; //pole znakov pre vypis na displej

    int cele = 1234;

    lcd_init(LCD_DISP_ON); // inicializacia displeja

    // konvertuj cele cislo na znaky pri zaklade 10
    itoa(cele,text,10);

    lcd_puts(text); // vypis znaky na displej

    return 0;
}
```

Funkcia sprintf má nasledovne parametre -

sprintf ( char \* str, const char \* format, ... );  
, funguje podobne ako printf, s tým rozdielom že výsledok zapíše do textového reťazca.

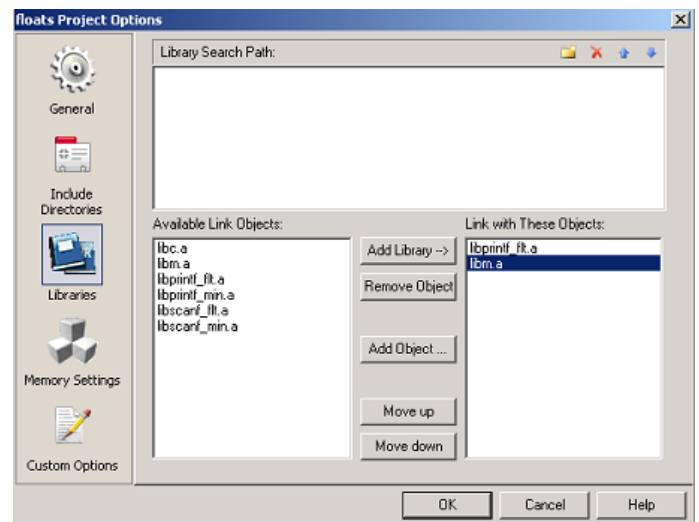
- char \* str – Ukazovateľ na pole znakov
- const char \* format – zápis reťazca

Problém nastane pokiaľ chceme vypísať desatinné číslo na displej pomocou funkcie sprintf();

Preto treba upraviť parametre projektu, aby sme s nimi mohli pracovať.

V menu teda vyberieme:

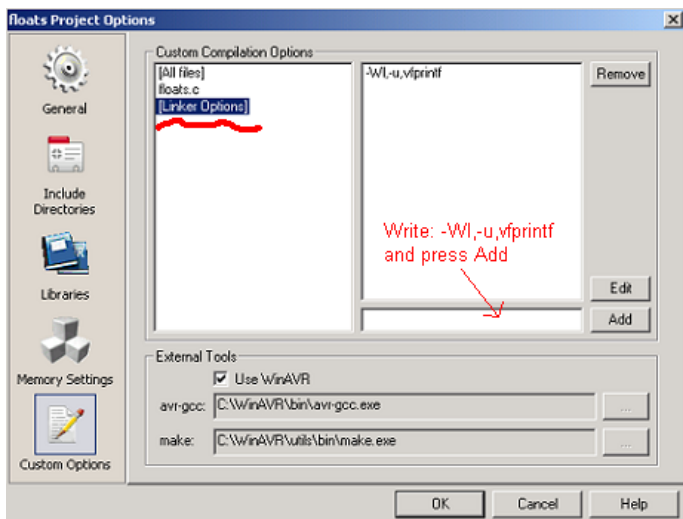
Project -> Configuration Options  
prejdeme na záložku Libraries kde pridáme súbory libprintf\_flt.a a libm.a



Obr 7.6 - Pridanie knižníc na prácu s desatinnými číslami

Potom prejdeme do záložky "Custom options", kde zvolíme "Linker Options", a pridáme prepínače -Wl,-u,vfprintf

# PRÁCA SO ZNAKOVÝM LCD DISPLEJOM



Obr 7.7 - pridanie prepínača na prácu s desatinnými číslami

Po tejto operácii môžete pracovať s desatinnými číslami typu float alebo double.

## Príklad č.2 – práca s desatinnými číslami pomocou funkcie sprintf()

### Zdrojový kód

```
#include <avr/io.h>
#include <stdlib.h>
#include <stdio.h>
#include "lcd.h"

int main(){

    float pi = 3.14;
    int cele = 1234;

    char text[32]; //pole znakov pre vypis na displej

    lcd_init(LCD_DISP_ON); // inicializacia displeja

    sprintf(text,"Cele cislo:%d\nDesatinne:%.2f",cele,pi);

    lcd_puts(text);

    while(1);

return 0;
}
```



Obr 7.8 - Ukážka z príkladu č. 2

# AD PREVODNÍKY

Analógovo-digitálny prevodník alebo analógovo-číslíkový prevodník (ADC z anglického Analog-to-Digital Converter) je elektronické zariadenie na prevod analógového signálu na digitálny signál. Prevod analógového signálu na digitálny je využívaný pomerne často, keďže signály sa skoro výlučne analyzujú a spracovávajú číslíkov. Príkladom konkrétnych aplikácií analógovo-digitálneho prevodu sú napr. meranie napätia, prúdu, neelektrických veličín...

## Základné parametre AD prevodníkov

### Rozlišovacia schopnosť (Resolution):

Je daná počtom rozlišiteľných úrovní analógového signálu. Mikrokontrolér ATmega8 má 10-bitový AD prevodník to znamená, že jeho rozlišovacia schopnosť je  $2^{10} = 1024$  úrovní.

### Rozsah prevodníka:

Je daný minimálnou a maximálnou hodnotou analógovej veličiny. Označuje sa zväčša symbolom FS (Full Scale). Rozsah AD prevodníka u ATmega8 môže byť 0-Vcc.

### Krok kvantovania:

Alebo citlivosť AD prevodníka je najmenšia rozlišiteľná veľkosť analógovej veličiny, tj. rozdiel susedných hodnôt analógovej veličiny pri ktorých sa zmení výstupný digitálny údaj. Označuje sa ako LSB (Least Significant Bit)

### Chyba kvantovania:

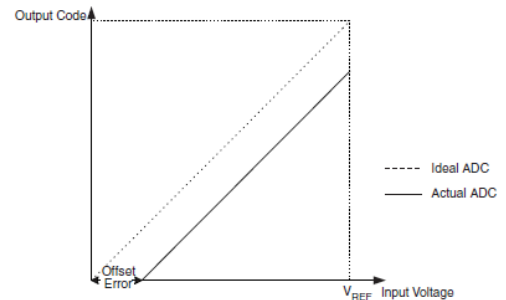
Chyba kvantovania predstavuje teoreticky maximálny rozdiel medzi hodnotou analógovej veličiny a jej maximálnou hodnotou zodpovedajúcou danému digitálnemu údaju. Obyčajne je to  $\frac{1}{2}$  LSB

### Rýchlosť prevodu (Conversion Time):

Rýchlosť prevodu je daná počtom prevodov, ktoré je schopný prevodník uskutočniť za jednotku času, alebo časom za ktorý vykoná prevodník jeden prevod. U ATmega8 môže byť čas prevodu v závislosti od predelčiky a zdroja hodín v rozsahu 13-260us.

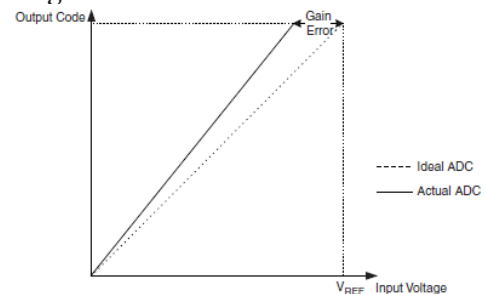
### Chybu prevodníka možno vyjadriť v zložkách:

Chyba offsetu (Offset error): je rovnaká pre celý rozsah AD prevodníka, vzniká posunutím nuly.



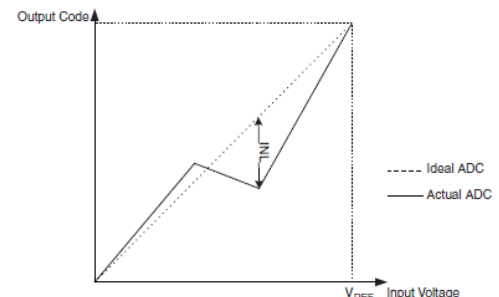
Obr 8.0 - Chyba offsetu AD prevodníka

Chyba zosilnenia (Gain Error): Závisí od hodnoty analógového signálu.



Obr 8.1 - Chyba zosilnenia AD prevodníka

Chyba nelineárnosti (Non – linearity error): Je spôsobená vtedy, ak závislosť vstupného analógového signálu nie je lineárne závislá od výstupného kódového slova.

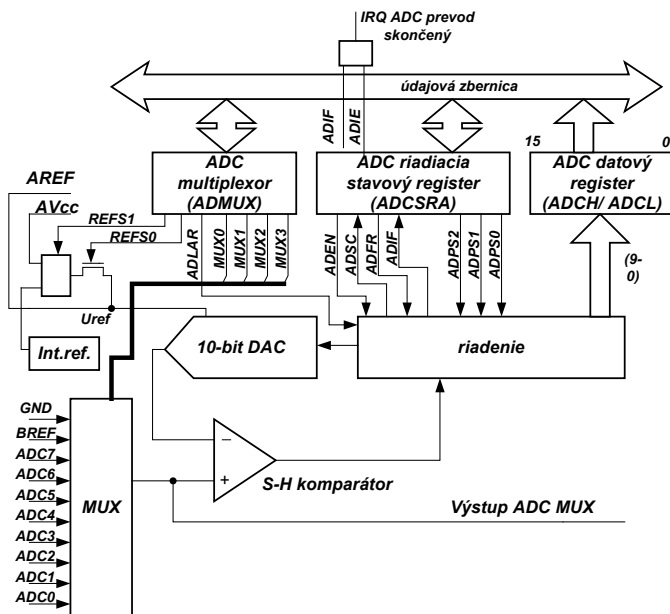


Obr 8.2 - Chyba nelineárnosti AD prevodníka

## AD prevodník mikrokontroléra ATmega8

### Základné vlastnosti:

- 10-bitové rozlíšenie
- Absolútna presnosť +2 LSB
- Integrálna nelinearita 0.5 LSB
- Čas prevodu 65 – 260us (do 15 ksps)
- 6 vstupných kanálov
- 2 ďalšie vstupné kanály, len v puzdrách TQFP a MLF
- Vstupný rozsah 0 až Vcc
- Voliteľné 2,56V interné referenčné napätie
- Opakovací, prípadne jednorazový režim
- Prerušenie po skončení prevodu
- Spiaci režim, potlačenie šumu



Obr 8.3 - Bloková schéma AD prevodníka

### Výpočet výstupnej digitálnej hodnoty pre AD prevodník:

$$ADC = (V_{in} * 2^n) / V_{ref}$$

(Vref - napätie referencie, Vin - vstupné napätie)

Pri Vin = 2,5V, Vref = 5V a rozlíšení AD prevodníka 10-bitov bude teda výstupná digitálna hodnota:

$$ADC = (2,5V * 2^{10}) / 5 = 512.$$

### Citlivosť AD prevodníka vypočítame ako:

$$LSB = V_{ref} / (2^n)$$

(Vref - napätie referencie, n - rozlíšenie AD prevod.)

Pri Vref= 5V a rozlíšení AD prevodníka 10-bitov bude teda citlivosť:

$$LSB = 5 / (2^{10}) = 4,9mV/bit$$

### Popis registrov AD prevodníka (vid' strana 199 v datasheete obvodu ATmega8)

#### Register ADMUX (ADC multiplexer selection register):

Bit	7	6	5	4	3	2	1	0	ADMUX
	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0	
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 8.4 - Register ADMUX

#### Bity REFS1 a REFS0 - výber zdroja referenčného napätia pre A/D prevodník

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V <sub>ref</sub> turned off
0	1	AV <sub>CC</sub> with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

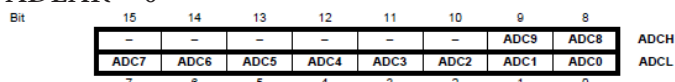
Obr 8.5 - Voľba referencie AD prevodníka

- (0, 0) – Referenčné napätie je privedené na pin AREF (povolený rozsah 2,0V - AVcc), interná referencia je vypnutá
- (0,1) – Referenciu predstavuje napájacie napätie na pine AVcc (vnútorne sa piny AVcc a AREF prepoja). Na pin AREF pripájame blokovací kondenzátor
- (1,0) – rezervované
- (1,1) – Vnútorná referencia 2,56V s pripojením blokovacím kondenzátorom na AREF pin. Pozor, presnosť vnútornej referencie má toleranciu až +-10%.

#### ADLAR - zarovnanie výsledku, ADC Left Adjust Result

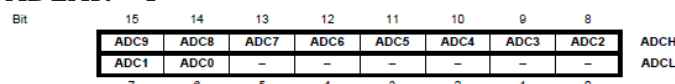
Pomocou obsahu bitu ADLAR je možné zvoliť zarovnanie výsledku prevodu v registroch ADCL a ADCH. Ak hodnota bitu ADLAR je log.1, potom výsledok prevodu je zarovnaný nasledovne: horných 8 bitov v registri ADCH a dva spodné bity v registri ADCL, bity 7 a 6. V opačnom prípade je spodných 8 bitov výsledku v registri ADCL a dva najvyššie bity v registri ADCH, v bitoch 0 a 1. Využíva sa to vtedy, ak nám dostačuje rozsah AD prevodníka iba na 8 bitov, teda v tom prípade zvolíme ADLAR=1 a prečítame len vrchný register ADCH.

ADLAR = 0



Obr 8.6 - Nastavenie ADLAR = 0

ADLAR = 1



Obr 8.7 - Nastavenie ADLAR = 1

### - Bity MUX3, MUX2, MUX1, MUX0 – Výber kanála, Analog Channel Selection

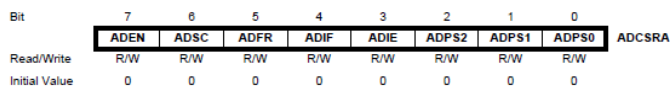
Hodnota uvedených bitov určuje, ktorý z analógových vstupných kanálov je pripojený na vstup A/D prevodníka.

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5

MUX3..0	Single Ended Input
0110	ADC6
0111	ADC7
1000	
1001	
1010	
1011	
1100	
1101	
1110	1.30V (V <sub>BG</sub> )
1111	0V (GND)

Obr 8.8 - Výber vstupného kanála

### Register ADCSRA (ADC Control and Status Register A) – riadiaci a stavový register



Obr 8.9 - Register ADCSRA

### - ADEN: povolenie činnosti ADC, ADC enable

Zápisom log.1 do bitu ADEN sa povoľuje činnosť prevodníka. Nulovaním uvedeného bitu sa prevodník vypína. Ak bit ADEN vynulujeme v priebehu prevodu, potom prevodník dokončí prebiehajúci prevod a vypne sa.

### - ADSC: štart prevodu, Start Conversion ADC

Ak prevodník pracuje v jednorazovom režime pred každým prevodom musí byť do bitu ADSC zapísaná log.1. Po ukončení prevodu sa pred vynulovaním bitu ADSC výsledok prevodu zapíše do údajových registrov ADC.

### - ADFR: výber opakovacieho režimu, ADC free run select

Ak nastavíme bit ADFR na log.1 prevodník pracuje v opakovacom režime. Vynulovaním bitu ADFR ukončíme opakovací režim.

### - ADIF: príznak prerušenia, ADC interrupt flag

Tento bit je nastavený na log.1 vtedy, keď prevod skončil a údajové registre ADCH a ADCL sú naplnené novým výsledkom.

### - ADIE: povolenie prerušenia, ADC interrupt enable

Ak bit ADIE je nastavený na log. 1 a je povolené globálne prerušenie vykoná sa prerušenie na akciu "dokončenie ADC prevodu".

### - ADPS2 – ADPS0: nastavenie preddeličky, ADC prescaler select bits

Uvedené bity určujú deliaci pomer medzi hodinovou frekvenciou (clkCPU) a frekvenciou vstupných hodinových impulzov do ADC. Pre dosiahnutie presnosti výsledku na 10 bitov musí byť frekvencia AD prevodníka 50-200kHz.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Obr 8.10 - Voľba preddeličky AD prevodníka

### Registre ADCH a ADCL:

- Výstupné registre, v ktorých sa nachádza výsledok AD prevodu. Tieto dva registre spoločne tvoria jeden 16-bitový register ADC.

## Techniky na potlačenie šumu:

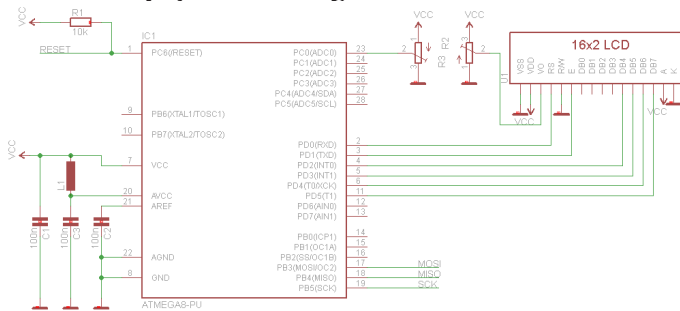
Digitálne obvody vo vnútri mikrokontroléra generujú elektromagnetický šum, ktorý môže mať negatívny vplyv na presnosť analógových meraní.

Ak je presnosť merania kritická, vplyv šumu môže byť čiastočne redukovaný nasledovným postupom:

1. Analógová časť mikrokontroléra a všetky analógové komponenty v aplikácii by mali mať pri návrhu PCB vlastnú analógovú zem. Analógová zem je s digitálnou spojená na PCB v jednom bode.
2. Minimalizujeme dĺžku analógových signálových vodičov. Analógové signálové vodiče umiestnime tak, aby prechádzali nad analógovou zemniacou plochou. Vedeime ich čo najďalej od digitálnych vodičov s rýchlo sa meniacimi signálmi.
3. AVCC vývod spojíme s napájaním digitálnej časti VCC cez filtračný LC článok.
4. Použijeme funkciu potlačenia šumu CPU.
5. Ak sú piny 0-3 portu C sú použité ako digitálne výstupy nemeníme ich pokiaľ prebieha prevod, pretože napájanie týchto pinov je spoločné s napájaním AD prevodníka.

## Práca s AD prevodníkom

### Schéma zapojenia ATmega8



Obr 8.11 - Schéma zapojenia MCU na prácu s AD prevodníkom

Na Obr 8.11 je štandardné zapojenie mikrokontroléra pre prácu s AD prevodníkom. Túto schému budeme využívať v nasledovných príkladoch. LC člen na pine AVCC slúži na potlačenie šumu, pre náš prípad stačí pripojiť pin AVCC priamo na VCC napájanie. Čo sa týka displeja, možno ho pripojiť na ľubovoľné piny mikrokontroléra okrem PORTC, pretože ten budeme využívať ako vstup pre AD prevodník. Piny treba následne definovať v hlavičkovom súbore lcd.h (viď predchádzajúca kapitola).

## Príklady:

### Príklad č.1: využitie prerušenia na akciu skončenia AD prevodu

V prvom prípade nastavíme AD prevodník na free running mode s vyvolaním prerušenia pri skončení prevodu. V prerušení zapíšeme obsah registra ADC do globálnej premennej adc. V nekonečnej slučke hlavného programu prepočítavame hodnotu adc na napätie a tieto dve hodnoty zobrazujeme na displeji.

### Zdrojový kód:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include "lcd.h"

//globalna premenna adc
volatile unsigned int adc;

// prerušenie na akciu skončenia AD prevodu
ISR(ADC_vect){
    adc=ADC; // zapis vysledok prevodu
}

int main(){

    char text[32];
    long u;

    ADMUX |= (1 << REFS0); //referencia 5V

    // zapnutie AD prevodnika, Free runing,
    // preddelicka 128 pri frek. hodin 8MHz
    ADCSRA |= (1 << ADEN) | (1 << ADSC) | (1 << ADFR)
    | (1 << ADIE) | (1 <<ADPS2) | (1 <<ADPS1) | (1 <<ADPS0);

    sei(); //povol globalne prerusenias

    lcd_init(LCD_DISP_ON);

    while(1){
        // vypočet napätia v mV
        u = ((long)adc*5000/1024);

        sprintf(text,"ADC:%d \nU:%ldmV",adc,u);
        lcd_clrscr();
        lcd_gotoxy(0,0);
        lcd_puts(text);
        _delay_ms(500);
    }
}
```



### Príklad č. 2: Vytvorenie funkcie na čítanie kanála s AD prevodníka

Vytvorená funkcia:

unsigned int Read\_ADC(unsigned char channel);  
 má parameter channel, kde napíšeme číslo kanála, ktorý chceme použiť ako vstup pre AD prevodník (viď Obr 8.8 - ADMUX). Návratovou hodnotou tejto funkcie je výsledok AD prevodu. Na zistenie skončenia prevodu využívame príznak na bite ADSC v registri ADCSRA. Túto hodnotu v nekonečnej slučke spolu s prepočítaním napätím zobrazujeme na displeji.

Pozn: Pokiaľ by sme chceli zobrazovať na displeji napätie ako desatinné číslo, treba nastaviť kompilier, aby prácu s desatinnými číslami umožňoval – viď predošlá kapitola

#### Zdrojový kód:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#include <stdio.h>
#include "lcd.h"

unsigned int Read_ADC(unsigned char channel){

    // vloz cislo kanala, vymaskuj nepouzite bity
    ADMUX &= 0xF0;
    ADMUX |= channel & 0x0F;

    // start conversion
    ADCSRA |= (1 << ADSC);

    // cakaj na priznak skoncenia konverzie
    while(ADCSRA & (1 << ADSC));

    // navratova hodnota - vysledok ad prevodu
    return ADC;
}

int main(){

    char text[32];
    unsigned int u,adc;

    ADMUX |= (1 << REFS0); //referencia 5V

    // povolenie AD prevodnika, Interrupt flag
    //128 pri frekvencii hodin 8MHz
    ADCSRA |= (1 << ADEN) | (1 << ADIF) | (1 <<ADPS2)
| (1 <<ADPS1) | (1 <<ADPS0);
```

```
sei(); // povol globalne prerusenie

lcd_init(LCD_DISP_ON);

while(1){

    adc = Read_ADC(0);
    // vypocet napatia v mV
    u = ((long)adc*5000/1024);

    sprintf(text,"ADC:%d \nU:%dmV",adc,u);
    lcd_clrscr();
    lcd_gotoxy(0,0);
    lcd_puts(text);

    _delay_ms(500);

}
}
```

#### Youtube video:

Skúška AD prevodníka

# ROZHRIANIE USART

V nasledujúcich kapitolách budú vysvetlené štandardy určené na obojsmerný prenos dát. V tejto kapitole sa bližšie pozrieme na univerzálnu asynchrónnu/synchrónnu sériovú linku alebo skrátene USART. USART patrí k základnej výbave takmer všetkých mikropočítačov rady AVR, to znamená že je hardvérovo implementovaný ako periféria mikropočítača.

## Popis asynchrónnej/synchrónnej sériovej linky

Sériová linka je známa už mnoho rokov. Na počítači sa vyskytuje pod štandardom RS-232 a je vyvedená z počítača pomocou DB-9 konektora. Rozhranie RS-232 však nemôžeme priamo pripojiť s rozhraním USART na mikropočítači, pretože RS-232 ma odlišne definované napäťové úrovne logických hodnôt. V dnešnej dobe sa už čoraz menej vyskytuje sériová linka na počítačoch či notebookoch, pretože je nahradzaná rozhraním USB. Na to aby sme mohli prenášať dáta z mikropočítača cez USART do počítača potrebujeme prevodník na rozhranie RS-232 alebo USB.

Návod na jednoduchý prevodník na USB nájdete v tomto článku:

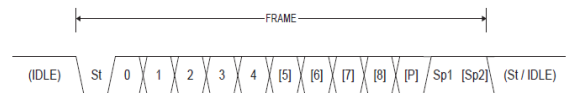
<http://svetelektro.com/clanky/jednoduchy-prevodnik-z-usb-na-uart-a-rs232-503.html>

Dáta z mikropočítača možno teda prenášať buď asynchrónne alebo synchrónne. Rozdiel je v tom, že pri synchrónnom prenose potrebujeme ďalší vodič, ktorý bude synchronizovať prenos dát hodinovým signálom. Pri synchrónnom prenose môžeme dosahovať vyššie rýchlosti v porovnaní s asynchrónnym. Avšak asynchrónny prenos dát sa využíva oveľa častejšie, najmä z toho dôvodu, že rozhranie RS-232 na počítači dokáže komunikovať len asynchrónne.

Pri asynchrónnom prenose je dátový rámec definovaný nasledovne:

- Start bit – začiatok prenosu, slúži na zosynchronizovanie vysielateľa a prijímateľa, stav linky je v log. 0
- Dátové bity – môže ich byť 5 až 9, začína sa bitom najnižšej váhy (LSB)

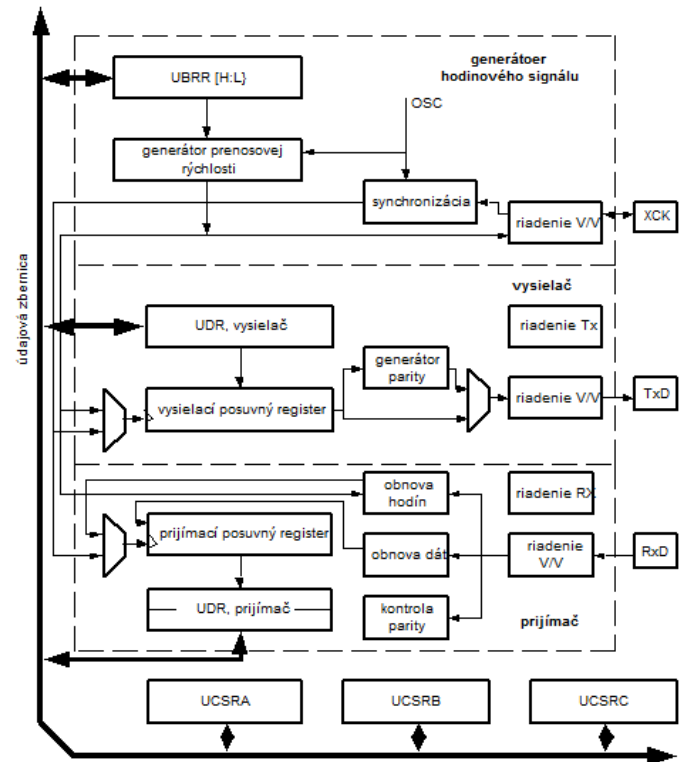
- Parity bit – slúži ako kontrolná suma pri prenose dát
- Stop bity (1 alebo 2) – ukončenie prenosu, stav linky je v log. 1
- Stav nečinnosti – ak sa žiadne dáta po linke neodosielajú ostáva v log. 1.



- St Start bit, always low
- (n) Data bits (0 to 8)
- P Parity bit. Can be odd or even
- Sp Stop bit, always high
- IDLE No transfers on the communication line (Rx/D or Tx/D). An IDLE line must be high

Obr 10.0 - Dátový rámec UART

## Bloková schéma USART



Obr 10.1 - Bloková schéma periférie USART MCU

Jednotka USART pozostáva z troch základných blokov.

Sú to: generátor hodinového signálu, vysielateľ a prijímač.

Generátor hodinového signálu je zložený z generátora prenosovej rýchlosti a synchronizačných obvodov pre externý vstup hodinového signálu, ktorý je využívaný v prípade, ak jednotka pracuje v synchrónnom "SLAVE" režime. Vývod XCK sa využíva len ak jednotka pracuje v synchrónnom prenosovom režime.

Vysielateľ je zložený z vyrovnávacieho registra, sériového posuvného registra, generátora parity a riadiacich obvodov pre prenos rôznych formátov údajov. Vyrovnávací register pre zápis dovoľuje vysielateľ údaje bez oneskorenia medzi jednotlivými čiastkovými prenosmi.

Prijímač je vďaka jednotkám obnovy údajov a hodinového signálu najzložitejšou časťou modulu USART. Jednotky obnovy sa využívajú pri asynchrónnom prijímaní dát. Prijímač obsahuje tiež obvod na kontrolu parity, riadiacu jednotku, posuvný register a dvojúrovňový prijímací vyrovnávací register (UDR). Prijímač podporuje všetky formáty prenosu dát vysielateľa a môže detekovať chybu príjmu, pretečenie údajov a chybu parity.

### Popis registrov

Modul USART mikropočítača Atmega8 obsahuje štyri 8-bitové registre:

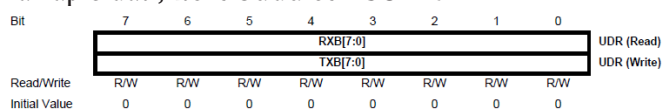
- údajový register UDR,
- riadiaci a stavový register A, UCSRA
- riadiaci a stavový register B, UCSRB
- riadiaci a stavový register C, UCSRC

Jeden 16-bitový register:

- register prenosovej rýchlosti UBRB.

### Údajový register UDR:

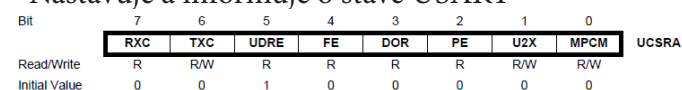
- Služi na ukladanie prijatých dát cez USART a taktiež na zápis dát, ktoré budú cez USART



Obr 10.2 - Register UDR

### Riadiaci a stavový register A, UCSRA - USART Control and Status Register A:

- Nastavuje a informuje o stave USART



Obr 10.3 - Register USCRA

### Bit 7 – RXC: USART Recieve Complete, Ukončený príjem znaku

Tento príznakový bit je nastavený na hodnotu log.1, ak v prijímací zásobník obsahuje neprečítané dáta. Bit RXC je vynulovaný, ak prijímací zásobník je prázdny. Ak sa zakáže činnosť prijímača, potom sa vynuluje zásobník a bit RXC bude obsahovať hodnotu log.0. Príznakový bit RXC môže byť využitý na generovanie prerušenia.

### Bit 6 – TXC: USART Transmit Complete, ukončené vysielanie znaku

Tento príznakový bit sa nastaví na hodnotu log.1, keď rámec vložený do vysielacieho posuvného registra je kompletne vyslaný a nové údaje nie sú zapísané do vysielacieho vyrovnávacieho registra (UDR). TXC príznak sa automaticky vynuluje, keď sa vykoná príslušná obsluha prerušenia, alebo zápisom log.1 do tejto V/V lokácie. Príznak TXC môže byť využitý na generovanie odpovedajúceho prerušenia.

### Bit 5 – UDRE: USART Data Register Empty, údajový register prázdny

Príznakový bit UDRE indikuje, že vysielací vyrovnávací register je pripravený prijať nové dáta. Ak bit UDRE má hodnotu log.1 vyrovnávací register je prázdny a pripravený na zápis nového znaku. Príznak UDRE môže byť využitý na generovanie odpovedajúceho prerušenia.

### Bit 4 – FE: Frame Error, chyba rámca

Tento bit sa nastaví na hodnotu log.1, ak znak v prijímacom vyrovnávacom registri obsahoval chybu rámca (prvý stop bit prijatého znaku bol rovný log.0). Hodnota bitu FE je platná pokiaľ sa neprečíta obsah prijímacieho vyrovnávacieho registra UDR. Bit FE je rovný nule, ak stop bit prijatého rámca má hodnotu jedna. Zápisom do registra UCSRA sa uvedený bit vynuluje.

### Bit 3 – DOR: Data OverRun, pretečenie dát

Tento bit sa nastaví na hodnotu log.1, ak dôjde k pretečeniu pri prijímaní dát. Pretečenie je charakterizované nasledovnými podmienkami: Prijímací zásobník je plný-(dva znaky), nový znak je v prijímacom posuvnom registri a je prijatý ďalší štart bit. Hodnota bitu DOR je platná pokiaľ sa neprečíta obsah prijímacieho vyrovnávacieho registra UDR. Zápisom do registra UCSRA sa uvedený bit vynuluje.

### Bit 2 – PE: Parity Error, chyba parity

Bit PE sa nastaví na hodnotu log.1, ak znak v prijímacom vyrovnávacom registri obsahoval chybu parity a kontrola parity bola povolená (UPM1=1). Hodnota bitu PE je platná, pokiaľ sa neprečíta obsah prijímacieho vyrovnávacieho registra UDR. Zápisom do registra UCSRA sa uvedený bit vynuluje.

### Bit 1 – U2X: Double the USART Transmission speed, dvojnásobná prenosová rýchlosť

Bit U2X má význam len pri asynchrónnom prenose. Ak je zvolený synchronný prenos, do bitu U2X zapíšeme hodnotu log.0. Pri asynchrónnej komunikácii sa zápisom logickej jednotky do bitu U2X prenosová rýchlosť zdvojnásobí.

### Bit 0 – MPCM: Multi-processor Communication Mode

Pomocou bitu MPCM povoľujeme režim multiprocessorovej komunikácie. Keď bit MPCM obsahuje log.1, potom všetky prijímané rámce, ktoré neobsahujú informáciu o adrese sa prijímačom USART ignorujú. Poznamenajme, že hodnota bitu MPCM nemá vplyv na činnosť vysielateľa modulu USART.

### Riadiaci a stavový register B, UCSRB - USART Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 10.4 - Register UCSRB

### Bit 7– RXCIE: RX Complete Interrupt Enable, povolenie prerušenia od RXC

Zápisom log.1 do bitu RXCIE povoľujeme prerušenie od príznaku RXC. Prerušenie RXC bude generované,

ak bit RXCIE obsahuje hodnotu log.1, je globálne povolená obsluha prerušení (bit I v SREG má hodnotu log.1) a súčasne sa bit RXC v registri UCSRA nastaví na hodnotu log.1.

### Bit 6 – TXCIE: TX Complete Interrupt Enable, povolenie prerušenia od TXC

Zápisom log.1 do bitu TXCIE povoľujeme prerušenie od príznaku TXC. Prerušenie TXC bude generované, ak bit TXCIE obsahuje hodnotu log.1, je globálne povolená obsluha prerušení (bit I v SREG má hodnotu log.1) a súčasne sa bit TXC v registri UCSRA nastaví na hodnotu log.1.

### Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable, povolenie prerušenia od UDRE

Zápisom log.1 do bitu UDRIE sa povolí prerušenie od príznaku UDRE. Prerušenie UDRE bude generované, ak bit UDRIE obsahuje hodnotu log.1, je globálne povolená obsluha prerušení (bit I v SREG má hodnotu log.1) a súčasne sa bit UDRE v registri UCSRA nastaví na hodnotu log.1.

### Bit 4 – RXEN: Receiver Enable, povolenie činnosti prijímača

Zápis log.1 do bitu RXEN sa povolí činnosť prijímača jednotky USART. Režim V/V vývodu RxD sa nastaví na vstup do prijímača USART. Ak sa zakáže činnosť prijímača (RXEN = 0), potom sa zničí obsah prijímacieho zásobníka a príznakov FE, DOR a PE.

### Bit 3 – TXEN: Transmitter Enable, povolenie činnosti vysielateľa

Zápis log.1 do bitu TXEN povoľujeme činnosť vysielateľa jednotky USART. Aktuálny režim V/V vývodu TxD sa nastaví na výstup z vysielateľa USART. Ak sa zakáže činnosť prijímača (TXEN = 0), potom sa vyšlú všetky údaje z posuvného vysielacieho registra a z vyrovnávacieho registra, zakáže sa činnosť vysielateľa a vývod TxD sa vráti do pôvodného režimu.

### Bit 2 – UCSZ2: Character Size, dĺžka znaku

Bit UCSZ2 spolu s bitmi UCSZ1:0 v registri UCSRC určuje počet údajových bitov prenášaného znaku.

### Bit 1 – RXB8: Receive Data Bit 8, prijatý údajový bit 8

Ak je nastavená dĺžka prenášaných znakov na deväť bitov, potom v bite RXB8 je uchovaná hodnota prijatého

tého deviateho údajového bitu. Jeho hodnota musí byť prečítaná pred čítaním obsahu registra UDR.

**Bit 0 – TXB8: Transmit Data Bit 8, vysielaný údajový bit 8**

Ak je nastavená dĺžka prenášaných znakov na deväť bitov, potom TXB8 reprezentuje deviaty vysielaný údajový bit. Deviaty údajový bit musí byť zapísaný pred zápisom spodných ôsmich bitov do registra UDR.

**Riadiaci a stavový register C, UCSRC - USART Control and Status Register C**

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

Obr 10.5 - Register UCSRC

**Bit 7 – URSEL: Register Select, výber registra**

Na základe hodnoty bitu URSEL je možné voliť medzi prístupom k registru UCSRC, alebo registru UBRRH. Pri čítaní UCSRC je hodnota tohto bitu log.1. Ak zapisujeme do registra UCSRC musí URSEL obsahovať hodnotu log.1.

**Bit 6 – UMSEL: USART Mode Select, výber režimu USART**

Pomocou bitu UMSEL volíme režim činnosti jednotky USART. Ak bit UMSEL má hodnotu nula, potom USART pracuje v asynchrónnom prenosovom režime. Ak bit UMSEL má hodnotu jedna, potom USART pracuje v synchrónnom prenosovom režime.

**Bit 5:4 – UPM1:0: Parity Mode, režim parity**

Tieto bity povoľujú a nastavujú typ generovania a kontroly parity. Ak je generovanie a kontrola parity povolená, potom vysielateľ bude v každom rámci generovať a vysielateľ paritný bit. Prijímač bude na základe prichádzajúcich dát generovať hodnotu paritného bitu a porovnávať ho s prichádzajúcim paritným bitom v súlade s nastavením bitu UPM0. Pri chybe parity sa nastaví príznak PE v registri UCSRA.

UPM1	UPM0	Režim parity
0	0	Žiadna
0	1	Nevyužité
1	0	Párna parita
1	1	Nepárna parita

Tab 10.0 - Nastavenie parity

**Bit 3 – USBS: Stop Bit Selct, výber stop bitu**

Pomocou bitu USBS sa volí počet stop bitov, ktoré sú vložené do vysielaného rámca. Prijímač modulu USART ignoruje nastavenie bitu USBS. Ak bit USBS má hodnotu nula, potom vysielaný rámec obsahuje jeden stop bit. Ak bit USBS má hodnotu jedna, potom vysielaný rámec bude obsahovať dva stop bity.

**Bit 2:1 – UCSZ1:0: Character Size, dĺžka znaku**

Pomocou bitov UCSZ1:0 a bitu UCSZ2 v registri UCSRB je možné nastaviť počet dátových bitov (dĺžku znaku) vo vysielanom a prijímanom rámci.

UCSZ2	UCSZ1	UCSZ0	Dĺžka znaku
0	0	0	5 bitov
0	0	1	6 bitov
0	1	0	7 bitov
0	1	1	8 bitov
1	0	0	Nevyužité
1	0	1	Nevyužité
1	1	0	Nevyužité
1	1	1	9 bitov

Tab 10.1 - Nastavenie dĺžky dátovej časti rámca

**Bit 0 – UCPOL: Clock Polarity, polarita hodinových impulzov**

Bit UCPOL je využitý len v prípade synchrónneho prenosového režimu. Ak je používaný asynchrónny režim do bitu UCPOL zapisujeme nulu. Pomocou bitu UCPOL definujeme vzťah medzi hodinovým signálom a vzorkovaním vstupného signálu, respektíve okamžikom zmeny výstupného signálu.

UCPOL	Zmena vysielaných dát,(TxD vývod)	Vzorkovanie prijímaných dát, (RxD vývod)
0	Vzostupná XCK hrana	Zostupná XCK hrana
1	Zostupná XCK hrana	Vzostupná XCK hrana

Tab 10.2 - Nastavenie synchronizačnej hrany XCK

## Registre prenosovej rýchlosti, UBRRL a UBRRH - USART Baud Rate Registers

Bit	15	14	13	12	11	10	9	8	
	URSEL	-	-	UBRR[11:8]					UBRRH
	UBRR[7:0]								UBRRL
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0	0

Obr 10.6 - Registre UBRRL a UBRRH

### Bit 15 – URSEL: Register Select, výber registra

Na základe hodnoty bitu URSEL je možné voliť medzi prístupom k registru UCSRC, alebo k registru UBRRH. Pri čítaní UBRRH je hodnota tohto bitu log.0. Ak zapisujeme do registra UBRRH musí URSEL obsahovať log 0.

### Bity 14:12 – nevyužitú (rezervované) bity

Pri zápise do UBRRH musia byť tieto bity rovné nule.

### Bity 11:0 – UBRR11:0: USART Baud Rate Register, register prenosovej rýchlosti

12-bitový register UBRR obsahuje údaj o prenosovej rýchlosti jednotky USART. V registri UBRRH sú obsiahnuté štyri bity najvyššej váhy a register UBRRL obsahuje osem bitov nižšej váhy. Zmena obsahu registra UBRR v priebehu prenosu spôsobí stratu prijímaných a vysielaných dát. Zápisom do registra UBRRL sa bezprostredne zmení prenosová rýchlosť.

### Výpočet prenosovej rýchlosti v Baudoch alebo výpočet obsahu registra UBRR na základe prenosovej rýchlosti

Operating Mode	Equation for Calculating Baud Rate <sup>(1)</sup>	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

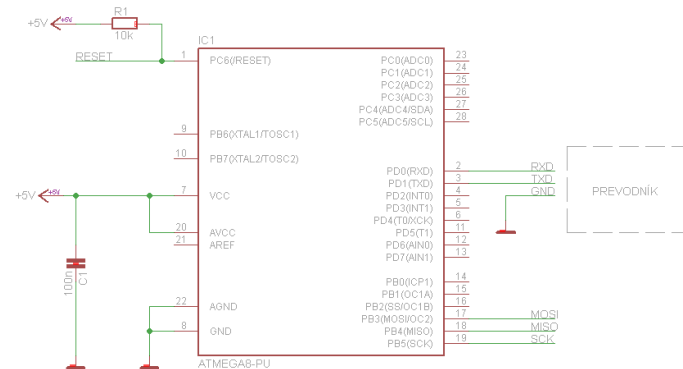
Obr 10.7 - Nastavenie prenosovej rýchlosti USART

**Pozn.:** Časté problémy s nefunkčnosťou UART na mikropočítači bývajú spôsobené odchýlkou prenosovej rýchlosti od žiadanej, nakoľko výpočet hodnoty UBRR nevyjde vždy celé číslo. Na dosiahnutie štandardných prenosových rýchlostí bez odchýlky sa po-

užívajú externé kryštálové oscilátory na frekvenciách 1,8432 MHz, 3,6864 MHz, 7,3728 MHz, 11,0592 MHz a 14,7456MHz. Pokiaľ používame vnútorný RC oscilátor nezabudneme nastaviť register OSCCAL, ktorý kalibruje RC oscilátor. Nastavenie registra UBRR a odchýlky pri rôznych prenosových rýchlostiach v závislosti od frekvencie hodín sa dočítate v datasheete ATmega8 od strany 153.

## Praktická realizácia prenosu dát cez UART Zapojenie mikropočítača ATmega8 pre prenos dát cez UART

Nato aby sme mohli otestovať priložené programy musíme najskôr prepojiť mikropočítač s počítačom cez prevodník. Prevodník môžeme realizovať buď na rozhranie RS-232 alebo USB. Zapojenie ilustruje aj nasledovná schéma:



Obr 10.8 - Schéma zapojenia MCU pre test USART

### Práca s knižnicou na obsluhu sériovej linky

Pre jednoduchšiu prácu s rozhraním UART na mikropočítači ATmega8 som sa rozhodol v spolupráci s kamarátom luboss17 naprogramovať knižnicu na obsluhu sériovej linky. Pomocou nej si ukážeme ako inicializovať UART a ako prenášať dáta.

Knižnicu určenú pre mikropočítač ATmega8,16,32 si môžete stiahnuť z nasledovného odkazu:

<http://svetelektro.com/Pictures/Microprocesory/avr/8/uart.zip>

V AVR studio 4 si vytvoríme nový projekt, následne vo vytvorenom adresári projektu pridáme súbory knižnice (uart.c, uart.h). Tieto súbory taktiež pridáme aj v našom projekte v AVR studio. Ak sa nám to podarilo môžeme otestovať náš prvý program.

## Príklady

### Príklad č.1: cyklické odosielanie reťazca znakov

Na začiatku programu inicializujeme UART mikropočítača cez funkciu `uart_init()`; pričom parametrom tejto funkcie je prenosová rýchlosť, ktorú musíme zadať. Štandardne sa používa rýchlosť 9600 baudov, preto som túto rýchlosť zvolil aj ja v nasledovnom programe. Pre správne nastavenie prenosovej rýchlosti musíme mať korektné nastavenú pracovnú frekvenciu mikropočítača v nastaveniach projektu.

Funkcia `uart_init()`; ďalej pomocou registra UCSRB povolí prijímač a vysielateľ a registrom UCSRC nastaví parametre prenosu – 8 dátových bitov, 1 stop bit, parita vypnutá.

Po inicializovaní UART-u môžeme začať odosielať alebo prijímať dáta. Na začiatok teda v programe odosieme v nekonečnej slučke reťazec znakov pomocou funkcie `uart_puts()`;

Pozn. 1: na odoslanie jedného znaku slúži funkcia `uart_putc()`;

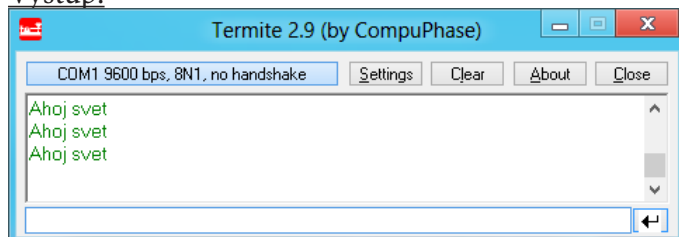
Pozn. 2: Pri zápise reťazca znakov používame úvodzovky napr. `uart_puts("Ahoj");` pri zápise jedného znaku apostrofy `uart_putc('A');`

### Zdrojový kód:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "uart.h"

int main(){
    // inicializacia uart na rychlost 9600bd
    uart_init(9600);
    while(1){
        // posli retazec znakov
        uart_puts("Ahoj svet\n");
        _delay_ms(1000);    // cakaj 1s
    }
    return 0;
}
```

### Výstup:



Obr 10.9 - Výstup programu č.1 v termináli

### Príklad č. 2: vrátenie odoslaného reťazca znakov

V druhom programe budeme už posilať aj dáta do mikropočítača. Dáta sú mikropočítačom prijímané do buffera (zásobníka) s nastavenou dĺžkou 128 bajtov. Pri prijímaní nového znaku nastane prerušenie a znaky sa ukladajú do poľa pričom sa čaká na ukončovací znak `"\r"` alebo `"\n"`. Ak ukončovací znak dorazí reťazec znakov sa ukončí a nastaví sa príznak informujúci že príjem reťazca znakov je ukončený. Pomocou funkcie `uart_gets()`; môžeme tieto dáta vybrať na spracovanie. Návrátovou hodnotou funkcie je príznak či bol prijatý reťazec znakov alebo nie. Do parametra funkcie zadáme ukazovateľ na pole do ktorého sa má prijatý reťazec znakov uložiť.

### Zdrojový kód:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#include "uart.h"

int main(){

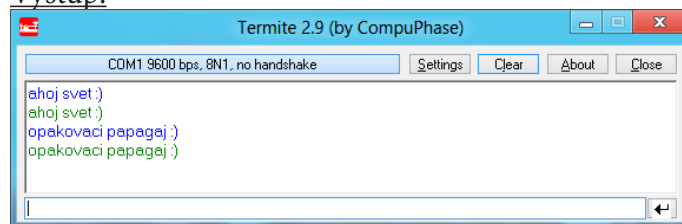
    char pole[128];

    // inicializacia uart na rychlost 9600bd
    uart_init(9600);

    // povol prerusenia
    sei();

    // nekonecna slucka
    while(1){
        //ak prisiel retazec znakov
        if(uart_gets(pole)){
            // posli spat retazec znakov
            uart_puts(pole);
        }
    }
    return 0;
}
```

### Výstup:



Obr 10.10 - Výstup programu č.2 v termináli

## ROZHRANIE USART

### Program č. 3: sčítavanie čísel

V tretom ukážkovom programe si ukážeme jednoduchý program, ktorý sčítava prijaté čísla cez sériovú linku a naspäť posiela výsledok sčítania.

V nekonečnom cykle sa testuje či prišiel nový reťazec znakov. Ak áno, vykoná sa funkcia `zisti_cislo()`; ktorá prijatý znak čísla premení na číslo. Ak číslo je nenulové pričíta sa ku výsledku a následne sa výsledok pošle naspäť cez UART. Program pracuje len s jednocifernými číslami, nič vám však nebráni môj ukážkový program vylepšiť :)

#### Zdrojový kód:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdio.h>

#include "uart.h"

unsigned char zisti_cislo(char *str){

    unsigned char cislo;

    //zisti zo znaku cislo
    //len jednociferne cisla
    switch(str[0]){
        case ,0': cislo = 0;break;
        case ,1': cislo = 1;break;
        case ,2': cislo = 2;break;
        case ,3': cislo = 3;break;
        case ,4': cislo = 4;break;
        case ,5': cislo = 5;break;
        case ,6': cislo = 6;break;
        case ,7': cislo = 7;break;
        case ,8': cislo = 8;break;
        case ,9': cislo = 9;break;
        default: cislo = 0; break;
    }

    return cislo;
}

int main(){

    char pole[10],text[16];
    unsigned char cislo=0,vysledok=0;

    // inicializacia uart na rychlost 9600bd
    uart_init(9600);

    // povol prerusenien
    sei();
```

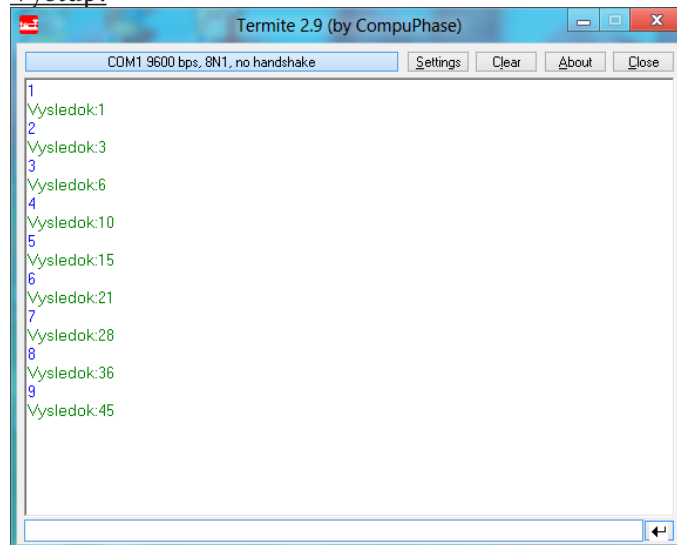
```
// nekonecna slucka
while(1){
    //cakaj na cislo
    do{
        // ak prislo cislo cez uart tak ho zisti
        if(uart_gets(pole)) cislo = zisti_cislo(pole);
    }while(cislo == 0);

    // pricitaj cislo ku vysledku
    vysledok += cislo;
    //zobraz vysledok
    sprintf(text,"Vysledok:%d",vysledok);
    // posli vysledok
    uart_puts(text);

    //nuluj premennu
    cislo=0;
}

return 0;
}
```

#### Výstup:



```
COM1 9600 bps, 8N1, no handshake | Settings | Clear | About | Close
1
Vysledok:1
2
Vysledok:3
3
Vysledok:6
4
Vysledok:10
5
Vysledok:15
6
Vysledok:21
7
Vysledok:28
8
Vysledok:36
9
Vysledok:45
```

Obr 10.11 - Výstup programu č.3 v termináli



# ROZHRAKIE SPI

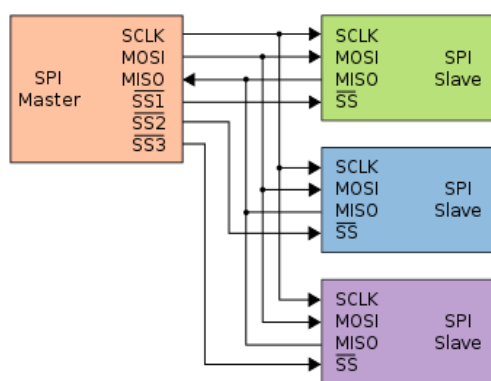
## Popis rozhrania SPI (sériové periférne rozhranie)

Najčastejšie sa používa na komunikáciu mikropočítača s ostatnými perifériami (AD-prevodník, EEPROM, displeje a pod.). V systéme využívajúcom rozhranie SPI môže byť zapojených dva alebo viac obvodov, pričom jeden z nich je označovaný ako Master (riadi komunikáciu), ostatné sú označované ako Slave.

Komunikácia prebieha pomocou štyroch vodičov MOSI, MISO, SCK, SS nasledovne:

- Dátový výstup Master obvodu MOSI (Master Out, Slave In), je pripojený na vstupy MOSI všetkých obvodov Slave
- Dátový vstup Master obvodu MISO (Master In, Slave Out), je pripojený na výstupy MISO všetkých obvodov Slave
- Hodinový signál SCK, generovaný Master obvodom, je pripojený na vstupy SCK všetkých obvodov Slave
- Každý obvod Slave má vstup SS (Slave select) alebo označovaný aj ako CS (Chip Select). Pokiaľ je vstup SS v neaktívnom stave (log. 1), rozhranie SPI daného obvodu je neaktívne a jeho výstup MISO je v stave vysokej impedancie. Vstupy SS jednotlivých obvodov sú samostatnými vodičmi pripojené k obvodu Master. Pomocou týchto vodičov teda možno jednoducho vyberať obvod, s ktorým máme v danom okamihu komunikovať.

Ukážka komunikácie MCU s viacerými perifériami pomocou rozhrania SPI:



Obr 11.0 - Komunikácia prostredníctvom SPI

Priebeh komunikácie rozhrania SPI:

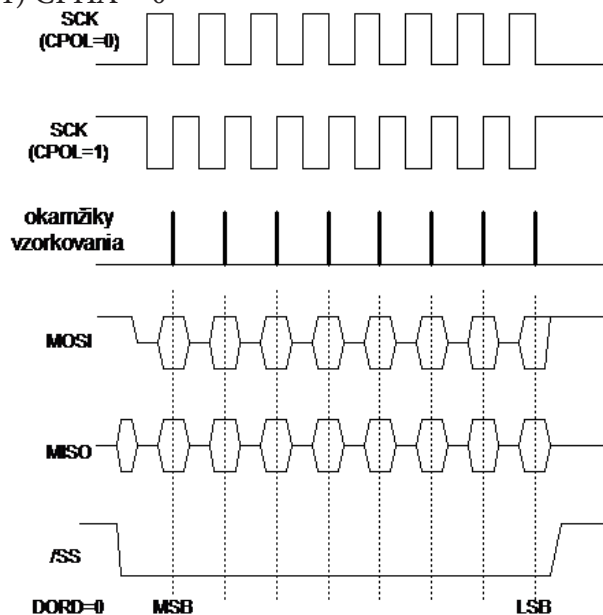
- Na začiatku komunikácie nastaví Master pin SS do log. 0 na zariadení, s ktorým chce komunikovať.
- Následne začne generovať hodinový signál SCK a v tej chvíli pošlú obe zariadenia (Master aj Slave) svoje dáta. Master cez pin MOSI, Slave cez pin MISO.
- Akonáhle sú dáta vyslané môže komunikácia ďalej prebiehať pokiaľ Master dodáva hodinový signál a hodnota SS je stále v log. 0.
- Master ukončí komunikáciu uvedením SS do stavu log. 1.

Polarita a fáza hodinového signálu

Vzťah medzi hodinovým signálom a dátami sa určuje pomocou dvoch konfiguračných bitov CPOL a CPHA. Ich význam je nasledovný:

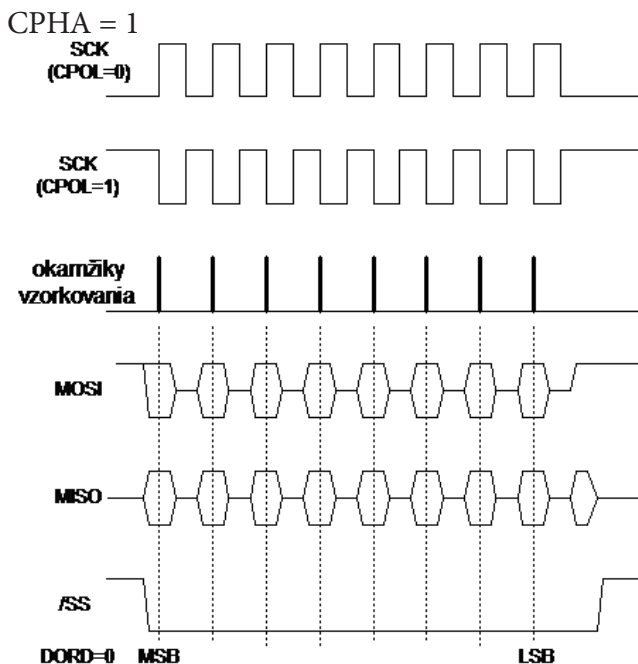
- CPOL = 0, hodnota hodinového signálu v stave nečinnosti je log. 0
- CPOL = 1, hodnota hodinového signálu v stave nečinnosti je log. 1
- CPHA = 0, dáta sú snímané nábežnou hranou hodinového signálu
- CPHA = 1, dáta sú snímané dobežnou hranou hodinového signálu

1) CPHA = 0



Obr 11.1 - Polarita a fáza hodinového signálu

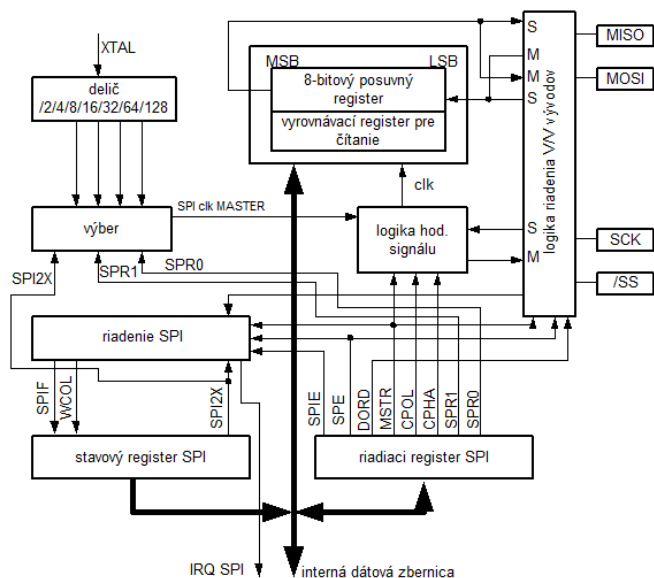
## ROZHRIANIE SPI



Obr 11.2 - Polarita a fáza hodinového signálu

**Pozn.:** Správne nastavenie bitov CPOL a CPHA býva zväčša uvedené v datasheete obvodu s ktorým chceme komunikovať. Preto si vždy overte toto nastavenie, aby ste sa vyhli prípadným problémom.

### Rozhranie SPI mikropočítača ATmega8



Obr 11.3 - Bloková schéma SPI rozhrania MCU

### Parametre:

- Obojsmerný (Full-duplex) synchronný prenos prostredníctvom troch vodičov.
- Režim "MASTER" alebo "SLAVE"
- Prenos začína LSB, alebo MSB
- Sedem programovateľných prenosových rýchlostí
- Príznak prerušenia pri ukončení prenosu
- Nastavenie príznaku pri kolízii
- Aktivácia procesora z "Idle" režimu
- Dvojnásobná rýchlosť (Ck/2)

V jednoduchosti systém pozostáva z dvoch posuvných registrov (jeden pre vysielanie, druhý pre príjem) a generátora hodinového signálu. Systém SPI sa vyznačuje jednoduchou vyrovnávacou pamäťou v smere vysielania a dvojitou pri prijímaní dát. To znamená, že byty, ktoré sú vysielané nemôžeme zapisovať do SPI dátového registra pred tým, než je kompletne vyslaný predchádzajúci bajt. Pri prijímaní údajov môžeme predchádzajúci bajt čítať v priebehu príjmu nasledujúceho bajtu. Predchádzajúci bajt musíme prečítať pred tým než je kompletne prijatý ďalší bajt.

V režime Slave riadiace obvody vzorkujú prichádzajúci hodinový signál na vstupe SCK. Frekvencia hodinového signálu SPI nesmie prekročiť hodnotu  $f_{OSC}/4$ .

### Funkcia pinu SS mikropočítača

#### a) v Master móde:

Ak je pin nastavený ako výstupný nepreberá automaticky v režime Master riadenie na vodiči SS. Toto riadenie musí byť realizované pomocou užívateľského programu počas komunikácie.

Ak je pin SS nastavený ako vstupný v režime Master, musíme ho udržiavať v log. 1 pokiaľ požadujeme Master mód. Pri privedení pinu SS do stavu log. 0 sa rozhranie SPI prepne do Slave modu a čaká na prijatie dát.

#### b) v Slave móde:

Ak je rozhranie SPI nastavené na režim SLAVE, potom pin SS bude vždy vstupný. Pokiaľ privedieme pin SS do stavu log. 0, aktivujeme SPI rozhranie. V opačnom prípade je rozhranie SPI neaktívne a nie je možné prijať žiadne dáta.

## Popis registrov

### Riadiaci register SPCR – SPI control register

- Riadiaci register SPCR je využitý na riadenie činnosti jednotky SPI.

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 11.4 - Register SPCR

### Bit 7 – SPIE: Povolenie prerušenia od SPI, SPI Interrupt Enable

Ak bit SPIE obsahuje hodnotu log.1 a súčasne sú globálne povolené prerušenia, potom pri nastavení bitu SPIF bude generovaná žiadosť o obsluhu prerušenia.

### Bit 6 – SPE: Povolenie činnosti jednotky SPI, SPI Enable

Ak bit SPE obsahuje hodnotu log.1, potom je povolená činnosť jednotky SPI.

### Bit 5 – DORD: Poradie dát, Data Order

Ak bit DORD obsahuje hodnotu log.1, potom prenos dát začína od LSB (najmenej významného bitu). Ak bit DORD obsahuje hodnotu log.0, potom prenos dát začína od MSB (najvýznamnejšieho bitu).

### Bit 4 – MSTR: Výber režimu MASTER/SLAVE, Master/Slave select

Pomocou bitu MSTR volíme režim jednotky SPI. Ak bit MSTR obsahuje hodnotu log.1 bude jednotka SPI v režime MASTER. V opačnom prípade bude SPI v režime SLAVE. Ak je vývod /SS konfigurovaný ako vstupný a je naň privedená log. 0, pričom je bit MSTR log.1, potom sa bit MSTR vynuluje a príznak SPIF sa nastaví na hodnotu log.1.

### Bit 3 – CPOL: Nastavenie polaritu hodinového signálu, Clock Polarity

Ak je bit CPOL nastavený na hodnotu log. 1 bude hodinový signál SCK v stave nečinnosti v log. 1 v opačnom prípade v log. 0. (viď obr. 11.1 a 11.2)

### Bit 2 – CPHA: Nastavenie fázy hodinového signálu, Clock Phase

Ak je bit nastavený do log. 1, budú dáta vzorkované na dobežnú hranu hodinového signálu. V opačnom prípade na nábežnú hranu. (viď obr. 11.1 a 11.2)

### Bity 1, 0 – SPR1, SPR0: Nastavenie generátora hodín, SPI Clock Rate Select 1 a 0

Ak SPI je v režime MASTER, potom pomocou obsahu bitov SPR1 a SPR0 je možné nastaviť frekvenciu hodinového signálu SCK. Vzťah medzi signálom SCK a frekvenciou oscilátora  $f_{osc}$  je uvedený v nasledujúcej tabuľke 11.0.

SPI2X	SPR1	SPR0	Frekvencia SCK
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Tab 11.0 - Nastavenie prenosovej rýchlosti SPI

### Stavový register jednotky SPI – SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	-	-	-	-	-	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 11.5 - Register SPSR

### Bit 7 – SPIF: Príznak prerušenia od SPI, SPI Interrupt Flag

Po skončení sériového prenosu dát sa príznak SPI nastaví na hodnotu log. 1. V prípade, že je povolené prerušenie bude generovaná aj žiadosť o obsluhu prerušenia. Aj v prípade, ak na vývod /SS, ktorý je v režime MASTER konfigurovaný ako vstup privedieme log. 0, sa nastaví príznak SPIF. Príznakový bit SPIF je nulovaný automaticky po skončení zodpovedajúcej obsluhy prerušenia. Príznakový bit SPIF je tiež nulovaný čítaním SPI stavového registra, prípadne prístupom k SPI dátovému registru.

### Bit 6 – WCOL: Príznak kolízie pri zápise, Write Collision Flag

Bit WCOL sa nastaví na hodnotu log.1 ak do SPI dátového registra sa zapisuje počas prenosu dát. Bit WCOL je nulovaný čítaním SPI stavového registra, prípadne prístupom do SPI dátového registra.

## ROZHRAINIE SPI

Bity 5...1: Nevyužitie bity

Bit 0 – SPI2X: Bit dvojnásobnej rýchlosti SPI, Double SPI Speed Bit

Keď bit SPI2X má hodnotu log.1, potom v prípade, ak jednotka SPI je v režime MASTER bude frekvencia hodinového signálu SCK dvojnásobná.

### Údajový register SPI – SPDR

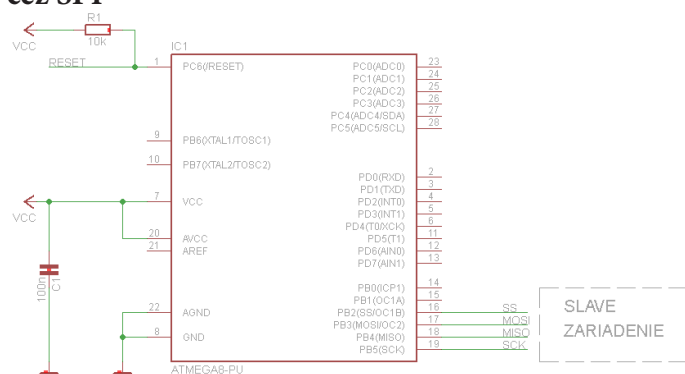
Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	-	-	-	-	-	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 11.6 - Register SPDR

Údajový register SPDR sa využíva na prenos dát medzi polom univerzálnych registrov a posuvným registrom SPI. Zápisom do registra SPDR inicializujeme prenos dát. Čítanie registra spôsobí prečítanie obsahu posuvného registra SPI.

### Praktická realizácia prenosu dát cez SPI

Zapojenie mikropočítača ATmega8 pre prenos dát cez SPI



Obr 11.7 - Schéma zapojenia MCU pre test SPI

Obvod pripojíme cez rozhranie SPI s mikropočítačom pomocou 4 vodičov SS, MOSI, MISO, SCK (viď popis na začiatku kapitoly). Pokiaľ chceme pripojiť viacero obvodov cez SPI ku mikropočítaču, využijeme ďalšie voľné piny mikropočítača pre riadiace signály SS pre ďalšie zariadenia.

### Práca s knižnicou na obsluhu rozhrania SPI

Pre jednoduchšiu prácu s rozhraním SPI na mikropočítači ATmega8 som sa rozhodol naprogramovať knižnicu na jeho obsluhu. Pomocou nej si ukážeme ako inicializovať SPI a ako prenášať dáta.

Knižnicu určenú pre mikropočítač ATmega8,16,32 si môžete stiahnuť tu:

[http://svetelektro.com/Pictures/Microprocesory/avr/9/clanok\\_spi.zip](http://svetelektro.com/Pictures/Microprocesory/avr/9/clanok_spi.zip)

V AVR studio 4 vytvoríme nový projekt, následne vo vytvorenom adresári projektu pridáme súbory knižnice (spi.c, spi.h). Tieto súbory taktiež pridáme aj do nášho projektu v AVR studio. Ak sa nám to podarilo môžeme otestovať náš prvý program.

Pre správnu činnosť nezabudnite správne nastaviť bity CPOL a CPHA. Tieto bity možno nastaviť vo funkcii InitSPI(void), konkrétne zápisom do registra SPCR (viď popis registrov vyššie). V hlavičkovom súbore sa nachádzajú okrem definície portov rozhrania aj makrá select(); a deselect(); ktorými riadime pin SS.

### Príklady

#### Príklad č. 1:

Cez SPI rozhranie MCU zapíš do Slave zariadenia hodnotu 0x21h. Následne prečítaj odpoveď slave zariadenia.

#### Zdrojový kód:

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "spi.h"

int main(){
    unsigned char data;

    //inicializacia SPI
    InitSPI();

    //zapis dat
    SELECT(); // vyber SS Slave zariadenia
    WriteByteSPI(0x21); //zapis data cez SPI
    // zrus vyber SS Slave zariadenia
    DESELECT();

    //citanie dat
    SELECT(); // vyber SS Slave zariadenia
    data = ReadByteSPI(); //citaj data cez SPI
    // zrus vyber SS Slave zariadenia
    DESELECT();

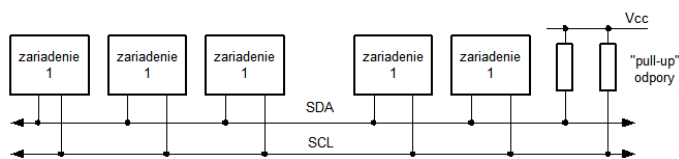
    return 0;
}
```

# ROZHRIANIE TWI

## Popis rozhrania I2C

Rozhranie I2C bolo vyvinuté firmou Philips pre komunikáciu zariadení na krátke vzdialenosti, typicky na doske plošných spojov. Rozhranie I2C je u mikropočítačov AVR nazývané ako dvojvodičové sériové rozhranie (TWI).

Dvojvodičové sériové rozhranie (TWI) je ideálne rozhranie pre mnohé aplikácie mikropočítačov. Protokol rozhrania TWI dovoľuje užívateľovi vzájomne prepojiť až 128 rôznych zariadení pomocou dvojvodičovej obojsmernej zbernice. Jeden vodič slúži na prenos hodinového signálu (SCL), druhý na prenos dát (SDA). Všetky externé komponenty pozostávajú z dvoch pull-up odporov (typicky hodnoty 4k7). Všetky zariadenia pripojené na zbernicu majú samostatné adresy a obsahujú mechanizmus pripájania na zbernicu, ktorý je súčasťou TWI protokolu.



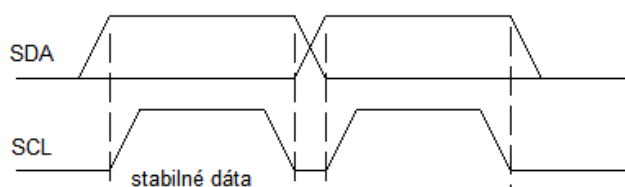
Obr 12.0 - Pripojenie zariadení na rozhranie TWI

Ako je uvedené na Obr 12.0 obidva vodiče zbernice sú pripojené na napätie Vcc prostredníctvom "pull-up" odporov zväčša hodnoty 4k7. Výstupné obvody TWI kompatibilných zariadení sú typu "open-drain", teda pracujú s otvoreným kolektorom. Z toho vyplýva, že úroveň napätia zodpovedajúca log.0 sa vyskytne na vodiči TWI zbernice vtedy, ak jeden, alebo viac zariadení bude generovať na svojom výstupe napätovú úroveň log.0. V prípade, že všetky TWI zariadenia budú mať výstupy v stave vysokej impedancie, bude vďaka "pull-up" odporom úroveň napätia na príslušnom vodiči zodpovedať log.1.

Počet spolupracujúcich zariadení je obmedzený kapacitou zbernice (max 400pF) a tiež 7-bitovým adresným priestorom. Detailnú špecifikáciu elektrických parametrov zbernice TWI je možné nájsť v datasheete MCU ATmega8 na strane 238.

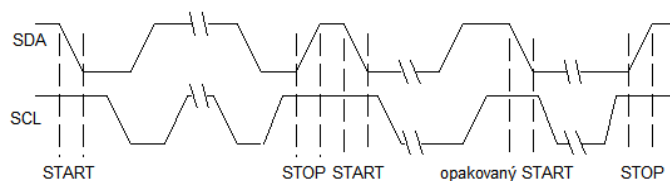
## Prenos dát a formát rámca (súbor údajov prenášaný v jednom bloku)

Každý údajový bit prenesený prostredníctvom TWI zbernice je sprevádzaný hodinovým impulzom na vodiči SCL. Napätová úroveň na dátovom vodiči, SDA musí byť počas vysokej úrovne SCL stabilná, (viď Obr 12.1). Výnimkou je len generovanie štartovacej a ukončovacej podmienky.



Obr 12.1 - Vzorkovanie dát na TWI zbernici

Zariadenie typu MASTER inicializuje a ukončuje prenos dát. Prenos je iniciovaný zariadením MASTER, ktorý na zbernicu generuje podmienku štartu (START). Prenos ukončuje MASTER pomocou ukončovacej podmienky (STOP). Medzi podmienkami START a STOP je zbernica obsadená a žiadne iné zariadenie typu MASTER nemôže prevziať riadenie zbernice. V špeciálnom prípade sa môžeme stretnúť s generovaním novej podmienky START medzi podmienkami START a STOP. Tento prípad sa nazýva opakovaný START a využíva sa v prípadoch, keď MASTER chce iniciovať nový prenos, bez opustenia riadenia zbernice. Po opakovanom štarte zbernica ostáva obsadená až po nasledujúcu podmienku STOP. Ako je znázornené na Obr. 12.2, START a STOP podmienky sú charakterizované zmenou úrovne na SDA vodiči počas doby, keď na SCL vodiči je vysoká úroveň napätia.



Obr 12.2 - Vyslanie Štart a Stop podmienky

## ROZHRANIE TWI

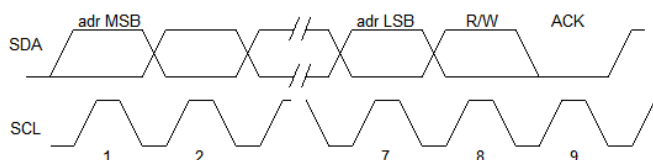
Adresné bloky prenášané prostredníctvom zbernice TWI majú 9 bitov (viď Obr 12.3).

Pozostávajú zo:

- siedmych adresných bitov
- jedného riadiaceho bitu - ČÍTANIE/ZÁPIS (R/W)
- bitu potvrdenia - (ACK)

Ak R/W bit má hodnotu log.1, bude vykonaná operácia čítania, inak sa vykoná operácia zápisu. Ak ľubovoľné zariadenie typu SLAVE rozpozná svoju adresu v deviatom takte hodín generuje na SDA vodič úroveň log.0. Ak je adresované zariadenie SLAVE zaneprázdnené a nemôže odpovedať na požiadavku zariadenia MASTER v deviatom cykle hodín (cyklus ACK) bude úroveň na vodiči SDA odpovedať log.1. Potom zariadenie typu MASTER môže vyslať podmienku STOP, alebo opakovaný START na iniciovanie nového prenosu. Adresný blok (paket) pozostáva teda z adresy a čítacieho/zápisového bitu, ktorý sa označuje SLA+R, alebo SLA+W.

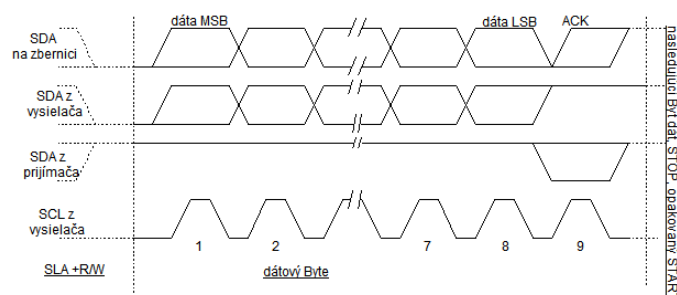
MSB adresa sa vysiela ako prvá. V priebehu návrhu môže užívateľ voľne pridelovať adresy jednotlivým zariadeniam, ale adresa 0000000b je rezervovaná pre všeobecné výzvy. Keď zariadenie typu MASTER vyšle všeobecnú výzvu, všetky zariadenia budú v cykle ACK generovať na SDA úroveň log.0. Všeobecná výzva sa používa v prípade, ak zariadenie MASTER chce všetkým zariadeniam SLAVE vyslať tú istú správu. V prípade, že vo všeobecnej výzve je bit R/W rovný log.0, potom nasledujúci dátový paket bude prijímaný všetkými zariadeniami SLAVE, ktoré reagovali na všeobecnú výzvu. Poznamenajme, že všeobecná výzva s R/W bitom rovným log.1 nemá prakticky význam. V tomto prípade by začalo súčasne vysielať viacero zariadení rôzne údaje.



Obr 12.3 - Adresný paket odoslaný cez TWI

Všetky dátové pakety prenášané prostredníctvom zbernice TWI majú deväť bitov (viď obr 12.4). Obsahujú 8 dátových bitov a jeden potvrdzovací bit. Počas

prenosu dát zariadenie typu MASTER generuje hodiiny a podmienky START a STOP, pričom prijímač je zodpovedný za generovanie potvrdenia príjmu ACK. Ako náhle prijímač ponechá vodič SDA počas deviateho cyklu hodín v stave vysokej úrovne (NACK), táto skutočnosť je interpretovaná ako neschopnosť prijímať ďalšie dáta.



Obr 12.4 - Dátový paket odoslaný cez TWI

Všeobecne je prenos zložený z podmienky START a SLA+R/W (adresa prijímača + R/W bit), jeden, alebo viac údajových paketov a z podmienky STOP. Nie je povolený prenos prázdnej správy (podmienka START a bezprostredne za ňou podmienka STOP). Poznajte, že zariadenie SLAVE môže podľa potreby predĺžiť nízku úroveň na vodiči SCL. Túto skutočnosť je možné využiť v prípade, ak rýchlosť SCL daná zariadením MASTER je pre daný prijímač príliš vysoká. Predĺženie času nízkej úrovne SCL nemá vplyv na čas trvania vysokej úrovne. Dôsledkom uvedeného je, že zariadenie SLAVE môže meniť prenosovú rýchlosť na zbernici TWI.

### Rozhranie TWI mikropočítača ATmega8

Modul TWI MCU ATmega8 obsahuje niekoľko funkčných blokov podľa Obr. 12.5. Všetky registre znázornené hrubou čiarou sú prístupné prostredníctvom údajovej zbernice procesora.

Vývody SDA a SCL spolupracujú s internými časťami MCU. Výstupné budiace obvody obsahujú obmedzovač strmosti hrán signálu (SRC), podľa špecifikácie rozhrania zbernice TWI. Vstupy obsahujú obvody potlačenia špičiek signálu (SF), ktoré sú kratšie než 50 ns. Interné "pull-up" odpory môžu byť použité nastavením príslušných bitov registra PORTC, v opačnom prípade treba použiť externé "pull-up" rezistory.

V prípade, že MCU pracuje v režime MASTER, jed-

notka generovania prenosovej rýchlosti riadi periódu hodinového signálu SCL. Frekvencia signálu SCL je určená obsahom bitov preddeličky v registri TWSR a obsahom registra TWBR. Ak MCU pracuje v režime SLAVE prenosová rýchlosť nezávisí od nastavenia registrov TWSR a TWBR, musí byť však minimálne 16 krát nižšia než je frekvencia hodín CPU.

Frekvencia generovaného signálu SCL je daná nasledovným vzťahom:

$$f_{SCL} = \frac{CLK_{CPU}}{16 + 2(TWBR)4^{TWPS}}$$

, kde TWBR je hodnota uchovaná v registri TWBR a TWPS je hodnota uchovaná v príslušných bitoch preddeličky v registri TWSR.

Jednotka rozhrania TWI obsahuje údajový a adresný posuvný register (TWDR), obvody riadenia a generovania START/STOP podmienok a výberové a detekčné obvody. Register TWDR obsahuje bajt dát/adresy, ktorý má byť vyslaný, prípadne bajt prijatých dát/adresy. Ako doplnok k registru TWDR sa využíva bit (N)ACK, ktorý bude vysielaný, alebo bol prijatý. Tento bit nie je priamo prístupný užívateľskému programu, iba prostredníctvom registra TWICR. Obvody riadenia START/STOP podmienok generujú a detekujú podmienky START, opakovaný START a STOP. Obvody riadenia START/STOP podmienok sú schopné rozpoznať podmienky START a STOP aj v prípade, že MCU je v niektorom z úsporných režimov. Ak je zariadenie adresované potom aktivujú MCU.

Ak TWI chce vysielat dáta v režime MASTER výberové a detekčné obvody neustále monitorujú prenos dát na zbernici. Ak v procese výberu je zariadenie neúspešné potom výberové a detekčné obvody informujú riadiacu jednotku TWI.

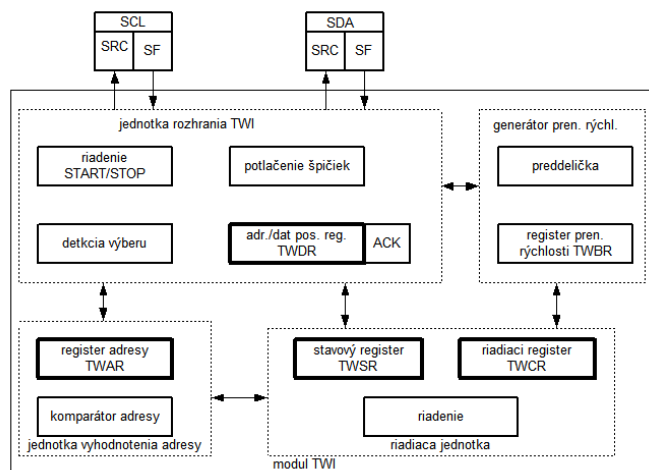
Jednotka vyhodnotenia adres kontroluje prijímané adresné bajty a porovnáva ich zhodu s obsahom 7-bitového adresného registra TWAR. Ak bit TWGCE v registri TWAR je nastavený na hodnotu log.1, potom prijatý adresný bajt je porovnávaný s adresou určenou pre všeobecnú výzvu. Ako už bolo spomenuté, jednotka vyhodnotenia adres je schopná vyhodnocovať adresy aj v prípade, že MCU je v úspornom režime. V prípade, že MCU je adresovaná zariadením typu

MASTER uvedie sa do aktívneho režimu.

Riadiaca jednotka monitoruje TWI zbernicu a generuje odozvu v súlade s nastavením riadiaceho registra TWCR. Ak sa na zbernici vyskytne taká udalosť, ktorá vyžaduje zásah aplikačného programu nastaví sa príznak prerušenia TWINT v registri TWCR. V nasledujúcom hodinovom cykle sa prepíše obsah stavového registra TWSR, pomocou ktorého je možné identifikovať druh udalosti. Register TWSR obsahuje relevantnú informáciu len po nastavení príznaku TWIN. Inak obsah registra TWSR pomocou špeciálneho stavového kódu informuje CPU, že relevantná informácia nie je prístupná. Po celý čas nastavenia príznaku TWINT je vodič SCL prostredníctvom výstupu držaný na nízkej úrovni. To dovoľuje aplikačnému programu reagovať na vzniknutú udalosť pred povolením ďalšieho prenosu prostredníctvom TWI zbernice.

TWIN príznak je nastavený v nasledujúcich prípadoch:

- Po tom, čo TWI vyslala podmienku START, alebo opakovaný START.
- Po tom, čo TWI vyslala SLA+R/W.
- Po tom, čo TWI vyslala adresný bajt.
- Po tom, čo TWI vypadla z procesu arbitráže (výber zariadenia typu MASTER).
- Po tom, čo bola adresovaná (vlastnou SLAVE adresou, alebo všeobecnou výzvou).
- Po tom, čo prijala bajt dát.
- Po tom, čo prijala STOP, alebo opakovaný START pokiaľ bola adresovaná ako SLAVE.
- Po tom, keď bola identifikovaná chyba zbernice (nepripustné START a STOP podmienky).



Obr 12.5 - Bloková schéma TWI rozhrania

## Popis registrov

### Register prenosovej rýchlosti – TWBR

Bit	7	6	5	4	3	2	1	0	TWBR
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 12.6 - Register TWBR

### Bity 7 až 0 - Bity registra prenosovej rýchlosti TWI, TWI Bit Rate Register

Pomocou obsahu registra TWBR sa určuje deliaci pomer generátora prenosovej rýchlosti. Generátor prenosovej rýchlosti je delič frekvencie slúžiaci na generovanie hodinového signálu SCL, v prípade, že MCU pracuje v režime MASTER. Prenosová rýchlosť je určená vzťahom uvedeným na predchádzajúcej strane.

### Riadiaci register TWI – TWCR

Bit	7	6	5	4	3	2	1	0	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 12.7 - Register TWCR

Riadiaci register TWCR je využitý na riadenie činnosti jednotky TWI.

### Bit 7 – Príznak prerušenia od TWI, TWI Interrupt Flag

Bit TWINT sa automaticky nastaví, keď TWI jednotka ukončí práve prebiehajúcu úlohu a vyžaduje odozvu (zásah) aplikačného programu. Ak sú globálne povolené prerušenia a súčasne je bit TWIE v registri TWCR nastavený na hodnotu log.1, potom pri nastavení bitu TWINT bude CPU pokračovať vo výkone programu na adrese prerušovacieho vektora TWI. Pokiaľ je bit TWINT nastavený na hodnotu log.1 výstup SCL je na nízkej úrovni. Bit TWINT musí byť vynulovaný pomocou programu, zápisom log.1. Poznamenajme, že nulovanie príznaku TWINT povolí činnosť na zbernici TWI. Preto obsahy všetkých registrov (stavový, dátový a adresný) musia byť prečítané/zapísané pred vynulovaním príznakového bitu TWINT.

vení bitu TWINT bude CPU pokračovať vo výkone programu na adrese prerušovacieho vektora TWI. Pokiaľ je bit TWINT nastavený na hodnotu log.1 výstup SCL je na nízkej úrovni. Bit TWINT musí byť vynulovaný pomocou programu, zápisom log.1. Poznamenajme, že nulovanie príznaku TWINT povolí činnosť na zbernici TWI. Preto obsahy všetkých registrov (stavový, dátový a adresný) musia byť prečítané/zapísané pred vynulovaním príznakového bitu TWINT.

### Bit 6 – Povolenie potvrdzovania, TWI Enable Acknowledge Bit

Bit TWEA riadi generovanie potvrdenia príjmu impulzom ACK. Ak bit TWEA je nastavený na hodnotu log.1, potom je impulz ACK generovaný na zbernici TWI pri splnení nasledujúcich podmienok:

- Ak zariadenie je v režime SLAVE a prijalo vlastnú adresu.
- Po prijatí všeobecnej výzvy, za predpokladu, že bit TWGCE v registri TWAR je nastavený na hodnotu log.1.
- Po prijatí dátového bajte v režime MASTER prijímač, alebo SLAVE prijímač.

Zápisom log.0 do bitu TWEA bude zariadenie virtuálne, dočasne odpojené od zbernice TWI. Rozpoznanie adresy môže znova nastaviť bit TWEA na hodnotu log.1.

### Bit 5 – Bit podmienky START, TWI START Condition Bit

Ak zariadenie chce na zbernici zaujať postavenie MASTER, potom aplikačný program zapíše do bitu TWSTA log.1. Obvodové prostriedky jednotky TWI kontrolujú zbernici a ak je zbernica voľná generujú podmienku START. Ak nie je zbernica uvoľnená TWI jednotka čaká pokiaľ sa na zbernici nevyskytne podmienka STOP a následne generuje podmienku START. Po vyslaní podmienky START bit TWSTA musí byť vynulovaný programom

### Bit 4 – Bit podmienky STOP, TWI Stop Condition Bit

Ak TWI jednotka je v režime MASER, potom zápisom log.1 do bitu TWSTO bude na TWI zbernici generovaná podmienka STOP. Po vyslaní podmienky STOP sa bit TWSTO automaticky vynuluje. V režime SLAVE sa nastavenie bitu TWSTO používa na zotavenie po podmienke ERROR. V tomto prípade sa ne-



generuje STOP podmienka, ale zariadenie sa vráti do neadresovaného režimu SLAVE. Jeho výstupy budú v stave vysokej impedancie.

**Bit 3 – Príznak kolízie pri zápise, TWI Write Collision Flag**

Bit TWWC sa nastaví, keď sa pokúšame zapísať dáta do TWI údajového registra TWDR, a príznak TWINT má hodnotu log.0. Tento príznak je nulovaný zápisom do registra TWDR, keď bit TWINT má hodnotu log.1.

**Bit 2 – Bit povolenia TWI, TWI Enable Bit**

Bit TWEN povoľuje činnosť jednotky TWI. Keď do bitu TWEN zapíšeme hodnotu log.1 jednotka TWI preberá riadenie V/V vývodov, priradí im funkciu SCL a SDA, povolí činnosť obmedzovača strmosti hrán signálu a filtra. Ak do bitu TWEN zapíšeme hodnotu log.0 jednotka TWI sa vypína a prenos sa ukončí.

**Bit 1 – Nevyužitý bit, Reserved Bit**

**Bit 0 – Povolenie prerušenia od TWI, TWI Interrupt Enable**

Ak bit TWIE je nastavený na hodnotu log.1 a súčasne sú globálne povolené prerušenia TWI jednotka bude generovať žiadosť o prerušenie vždy, ak bude nastavený príznak TWINT.

**Stavový register TWI – TWSR**

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	-	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Obr 12.8 - Register TWSR

**Bity 7..3 – TWS, Stav TWI, TWI Status**

Týchto 5 bitov charakterizuje stav jednotky TWI i samotnej zbernice.

**Bit 2 – Nevyužitý bit, Reserved Bit**

**Bity 1..0 –TWPS, Bity preddeličky, TWI Prescaler Bits**  
Obsah bitov TWPS určuje použitý deliaci pomer preddeličky podľa tabuľky 12.1.

TWPS1	TWPS0	Deliaci pomer
0	0	1
0	1	4
1	0	16
1	1	64

Tab.12.1 Hodnoty deliaceho pomeru

**Údajový register TWI – TWDR**

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

Obr 12.9 - Register TWDR

Ak je jednotka TWI vo vysielacom režime, potom register TWDR obsahuje nasledujúci bajt, ktorý bude vysielaný. Ak je jednotka TWI v režime príjmu, potom register TWDR obsahuje posledný prijatý bajt. Do registra TWDR sa nemôže zapisovať, pokiaľ sa ešte posúva vysielaný bajt. Ak sa nastavil príznak TWINT je možné pristupovať k registru TWDR. Poznamenajme, že k registru TWDR nemôžeme pristupovať pred prvým výskytom prerušenia. Údaje v registri TWDR ostávajú stabilné pokiaľ príznak TWINT má hodnotu log.1. Pokiaľ sú údaje posúvané z registra TWDR na zbernicu, tak aktuálne údaje zo zbernice sú posúvané do registra. Register TWDR preto vždy obsahuje posledný bajt, ktorý bol na zbernici, s výnimkou prechodu z úsporného režimu po prerušení od jednotky TWI. V tomto prípade je obsah registra TWDR nedefinovaný. Bit potvrdenia príjmu ACK je nastavovaný automaticky, vlastná CPU nemôže k nemu pristupovať.

**Bity 7:0 - Bity údajového registra, TWI Data Register**

Tieto bity tvoria nasledujúci údajový bajt, ktorý má byť vyslaný, alebo posledný údajový bajt, ktorý bol prijatý z dvojvodičovej (TWI) zbernice.

**Adresový register TWI – TWAR**

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCEN	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

Obr 12.10 - Register TWAR

Register TWAR obsahuje 7-bitovú adresu, na ktorú bude TWI reagovať vždy, ak bude v režime SLAVE vysielateľ, alebo prijímač. V režime MASTER nie je potrebná. V systémoch s viacerými zariadeniami typu MASTER musí byť register TWAR nastavený v tých zariadeniach, ktoré môžu byť adresované ako zariadenia SLAVE druhými zariadeniami typu MASTER. Bit TWGCE je využitý na povolenie rozpoznávania všeobecnej výzvy (adresa 0x00).

### Bity 7:1 - Bity adresového registra, TWI (Slave) Adress Register

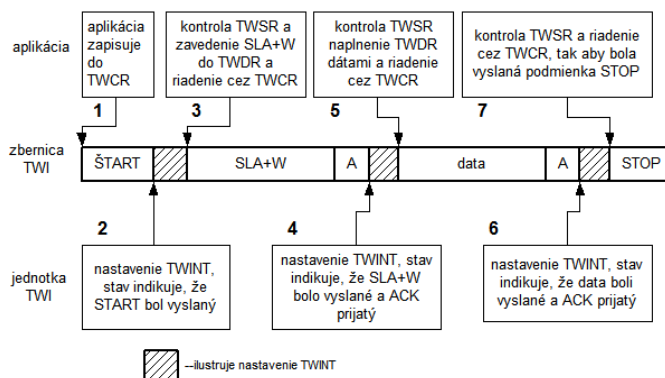
Tieto bity obsahujú adresu TWI jednotky v režime SLAVE.

### Bit 0 – Povolenie rozpoznania všeobecnej výzvy, TWI General Call Recognition Enable Bit

Ak bit TWGCE je nastavený na hodnotu log.1, potom je povolené rozpoznávanie všeobecnej výzvy vyskytujúcej sa na TW sériovej zbernici.

### **Prenos údajov prostredníctvom TWI**

Jednotka AVR TWI je bajtovo orientovaná s plným využitím prerušení. Prerušenia sú generované po všetkých udalostiach na zbernici. Pretože jednotka TWI využíva prerušovací systém, CPU môže v priebehu prenosu jedného bajtu vykonávať aplikačný program. Ak nie je povolené prerušenie od TWI jednotky, prípadne sú globálne zakázané prerušenia, CPU musí programovými prostriedkami testovať nastavenie príznaku TWINT. Príznak TWINT sa nastaví vždy po ukončení operácie na zbernici. TWI jednotka potom očakáva odozvu od aplikačného programu. V tomto prípade stavový register TWSR obsahuje hodnotu, ktorá charakterizuje aktuálny stav TWI zbernice. Aplikačný program na základe tohoto stavu rozhodne ako sa bude jednotka TWI chovať v nasledujúcom zbernicovom cykle. Správanie jednotky TWI určuje aplikačný program prostredníctvom vhodného nastavenia registrov TWCR a TWDR. Na nasledujúcom obrázku je znázornená spolupráca aplikačného programu s jednotkou TWI. V uvedenom príklade zariadenie typu "MASTER" prenáša jeden bajt do zariadenia typu "SLAVE".



Obr 12.11 - Časový diagram prenosu dát cez TWI

1. V prvom kroku je potrebné vyslať podmienku START. Aplikácia to zaisťuje zapisom definovanej hodnoty do registra TWCR. Takto iniciuje jednotku TWI na prenos START podmienky. Týmto zapisom sa príznak TWINT nastaví na hodnotu log.1. Zapisom log.1 do TWINT sa príznak TWINT vynuluje. Jednotka TWI nezahájí vlastný prenos pokiaľ TWINT obsahuje hodnotu log.1. Bezprostredne po vynulovaní bitu TWINT jednotka zahájí vlastný prenos podmienky START.
2. Po vyslaní START podmienky sa príznak TWINT v registri TWCR nastaví na hodnotu log.1 a obnoví sa obsah TWSR, ktorý indikuje že podmienka bola úspešne vyslaná.
3. Aplikačný program testuje obsah TWSR a zisťuje či podmienka START bola úspešne vyslaná. Ak obsah TWSR nezodpovedá úspešnému vyslaniu START podmienky potom aplikácia musí reagovať napríklad volaním chybovej rutiny. V prípade, že obsah TWSR odpovedá úspešnému vyslaniu podmienky START, potom aplikácia naplní register TWDR hodnotou SLA+W a do registra TWCR zapíše definovanú hodnotu, ktorá informuje TWI jednotku, že má zahájiť prenos SLA+W z registra TWDR. Zapisom hodnoty log.1 do bitu TWINT sa nuluje príznak TWINT. Bezprostredne po vynulovaní bitu TWINT jednotka zahájí vlastný prenos adresovacieho paketu.
4. Ak bol adresovací paket úspešne prenesený nastaví sa príznak TWINT v registri TWCR a súčasne sa obnoví obsah registra TWSR tak, aby odpovedal úspešnému prenosu adresy. Obsah registra obsahuje aj informáciu či adresované zariadenie potvrdilo, alebo nepotvrdilo príjem paketu.

5. Aplikáčny program musí testovať obsah TWSR a zistiť, či bol adresovací paket úspešne prenesený a či bol potvrdený jeho príjem. Ak obsah TWSR nezodpovedá úspešnému vyslaniu a potvrdeniu príjmu adresovacieho paketu, potom aplikácia musí reagovať, napríklad volaním chybovej rutiny. V prípade, že obsah TWSR odpovedá úspešnému vyslaniu adresovacieho paketu, potom aplikácia naplní register TWDR hodnotou dátového bajtu. Do registra TWRC zapíše definovanú hodnotu, ktorá informuje TWI jednotku, že má zahájiť prenos dátového bajtu z registra TWDR. Zápisom hodnoty log.1 TWINT sa nuluje príznak TWINT. Bezprostredne po vynulovaní bitu TWINT jednotka zahájí vlastný prenos údajového paketu.
6. Po prenose údajového paketu sa nastaví príznak TWINT v registri TWCR a súčasne sa obnoví obsah registra TWSR tak, aby odpovedal úspešnému prenosu dátového paketu. Obsah registra obsahuje aj informáciu či adresované zariadenie potvrdilo, alebo nepotvrdilo príjem paketu.
7. Aplikáčny program testuje obsah TWSR a zisťuje, či bol dátový paket úspešne prenesený a či bol potvrdený jeho príjem. Ak obsah TWSR nezodpovedá úspešnému vyslaniu a potvrdeniu príjmu dátového paketu, potom aplikácia musí reagovať, napríklad volaním chybovej rutiny. V prípade, že obsah TWSR odpovedá úspešnému vyslaniu dátového paketu, potom aplikácia zapíše do registra TWRC definovanú hodnotu, ktorá informuje TWI jednotku, že má zahájiť prenos podmienky STOP. Zápisom hodnoty log.1 TWINT sa nuluje príznak TWINT. Bezprostredne po vynulovaní bitu TWINT jednotka zahájí vlastný prenos STOP podmienky. Poznamenajme, že po prenose podmienky STOP sa už príznak TWINT nenastaví.

Uvedený príklad mal za úlohu poukázať na základné princípy prenosu prostredníctvom zbernice TWI.

Tieto princípy je možné zhrnúť nasledovne:

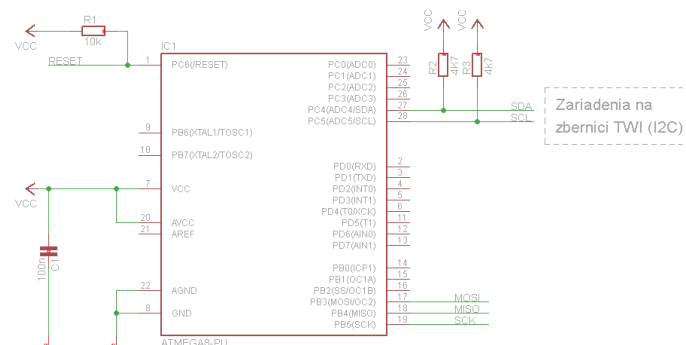
- Ak jednotka TWI ukončí ľubovoľnú operáciu a očakáva odozvu aplikácie nastaví príznak TWINT. Pokiaľ príznak TWINT je nastavený, bude na vodiči SCL TWI zbernice úroveň logickej nuly.
- Pokiaľ TWINT má hodnotu log.0, musí užívateľ obnoviť obsah registrov TWI jednotky, tak, aby

boli relevantné pre nasledujúci zbernicový cyklus.

- Po obnovení obsahu TWI registrov aplikáčny program zapíše vhodný obsah do TWCR registra. Zápisom logickej jednotky do bitu TWINT sa príznak TWINT vynuluje. Jednotka TWI zahájí operáciu odpovedajúcu obsahu registra TWCR.

## Praktická realizácia prenosu dát cez TWI

### Zapojenie mikropočítača ATmega8 pre prenos dát cez TWI



Obr 12.12 - Schéma zapojenia MCU pre test TWI

V praktickej časti si ukážeme ako čítať a zapisovať dáta zo Slave zariadenia pripojeného ku mikropočítaču, ktorý bude v režime Master.

Naše Slave zariadenie (napr. EEPROM pamäť, senzor, displej a pod) pripojíme pomocou vodičov SDA a SCL ku mikropočítaču. Na linky SDA a SCL pripojíme "pull-up" rezistor s hodnotou 4k7.

### Práca s knižnicou na obsluhu rozhrania TWI

Pre jednoduchšiu prácu s rozhraním TWI na mikropočítači ATmega8 môžete použiť knižnicu i2cmaster, ktorú naprogramoval kolega luboss17.

Knižnicu určenú pre mikropočítač ATmega8 si môžete stiahnuť tu:

<http://svetelektro.com/Pictures/Microprocesory/avr/10/twi.zip>

V AVR studio 4 vytvoríme nový projekt, následne vo vytvorenom adresári projektu prekopírujeme súbory knižnice (i2c\_master.c, i2c\_master.h). Tieto súbory taktiež pridáme aj do nášho projektu v AVR Studio.

## ROZHRAINIE TWI

---

Pred použitím knižnice je vhodné nastaviť v hlavičkovom súbore `i2c_master.h` frekvenciu prenosu – konštantu `SCL_CLOCK` (do 400 kHz).

Knižnica obsahuje všetky potrebné funkcie pre prácu s rozhraním TWI v režime Master.

Na začiatku nášho programu musíme najskôr TWI rozhranie inicializovať pomocou funkcie:

```
void i2c_init(void);
```

Následne môžeme vyslať štart podmienku pomocou funkcie:

```
unsigned char i2c_start(unsigned char address, unsigned char dir);
```

, kde prvý parameter je adresa zariadenia (už posunutá o bit doľava t.j. posledný bit je voľný) a druhý parameter je riadenie smeru prenosu, kde môžeme využiť definované konštanty `I2C_READ` alebo `I2C_WRITE`. Po bezchybnom priebehu zápis vráti táto funkcia nulu. Po neúspešnom štarte vráti 1 a ak SLAVE nepotvrdí dáta Ack bitom vráti 2.

Dáta na zbernicu vyšleme funkciou (po štart podmienke s adresou a `I2C_WRITE`):

```
unsigned char i2c_write(unsigned char data);
```

, kde parametrom funkcie je vysielaný bajt. Po úspešnom vyslaní bajtu vráti funkcia 0 ak nie vráti 1.

Bajt zo zbernice prečítame funkciou:

```
unsigned char i2c_read_nAck(void);
```

V prípade, že chceme naraz prijať viac bajtov za sebou použijeme funkciu:

```
unsigned char i2c_read_Ack(void);
```

, to znamená, že za prijatým bajtom MASTER generuje ACK bit a čaká na ďalší bajt pričom pri poslednom bajte použijeme funkciu:

```
unsigned char i2c_read_nAck(void);
```

Stop podmienka sa vykonáva funkciou:

```
unsigned char i2c_stop(void);
```

po úspešnom priebehu vráti 0 po neúspešnom vráti 1.

### Príklady

#### Príklad č. 1:

Zapíš do Slave zariadenia s adresou `0x55h` hodnotu `0x33h`. Následne prečítaj 4 bajty dát z tohto zariadenia.

#### Zdrojový kód:

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

#include "i2c_master.h"

int main(void){

    unsigned char a,b,c,d;

    // inicializacia i2c rozhrania
    i2c_init();

    // zapis data 0x33h na Slave adresu zariadenia 0x55
    i2c_start(0x55,I2C_WRITE);
        i2c_write(0x33);
        i2c_stop();

    // citaj 4 bajty zo Slave zariadenia
    i2c_start(0x55,I2C_READ);
        unsigned char a = i2c_read_Ack();
        unsigned char b = i2c_read_Ack();
        unsigned char c = i2c_read_Ack();
        unsigned char d = i2c_read_Ack();
        i2c_stop();

}
```

# REŽIMY SPÁNKU MCU

## Režimy spánku

Režimy so zníženou spotrebou umožňujú vypnúť práve nepoužívané periférie mikrokontroléra a tým znížiť spotrebu zariadenia. Mikrokontrolér ponúka viacero režimov so zníženou spotrebou a tak umožňuje užívateľovi redukovať spotrebu energie v závislosti na požiadavkách aplikácie.

Režimy spánku využijeme hlavne v aplikáciách, pri ktorých je mikrokontrolér napájaný z batérie. Tam môžeme využitím vhodného režimu spánku niekoľkonásobne zvýšiť výdrž batérie.

## Popis registrov

Ak chceme využiť ľubovoľný z piatich úsporných režimov, potom bit SE v MCUCR registri musí byť nastavený na hodnotu 1. Výkonom inštrukcie SLEEP prejde MCU do zvoleného úsporného režimu. Hodnoty bitov SM2, SM1 a SM0 v registri MCUCR definujú, ktorý z piatich možných režimov bude aktivovaný. Ak sa v priebehu úsporného režimu vyskytne povolené prerušenie MCU prejde do aktívneho stavu. V tomto prípade MCU bude na 4 hodinové cykly zastavená a po nábehovom čase sa realizuje príslušná obsluha prerušenia. Po obsluhu prerušenia bude MCU pokračovať vo výkone programu, inštrukciou nasledujúcou za inštrukciou SLEEP. Obsah pola univerzálnych registrov a pamäte SRAM ostáva nezmenený. Register MCUCR obsahuje riadiace bity pre riadenie spotreby.

### Register MCUCR:

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Obr 13.0 - Register MCUCR

### Bit 7 – SE: Sleep Enable

Ak bit SE je nastavený na hodnotu 1, potom je povolené vykonanie inštrukcie SLEEP. Vhodné je nastaviť bit SE práve pred inštrukciou SLEEP.

### Bity 6, 4 – SM2 . . 0 Sleep Mode Select 2, 1 a 0

Pomocou bitov SM2, SM1 a SM0 je možné zvoliť jeden z piatich úsporných režimov MCU podľa Tab. 1

SM2	SM1	SM0	Úsporný režim
0	0	0	Idle
0	0	1	ADC noise reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Rezervované
1	0	1	Rezervované
1	1	0	Standby(1)

(1) Standby režim je možné využiť len s externým kryštálom, alebo rezonátorom

Tab 13.0 Režimy spánku MCU

## Popis režimov spánku

### Idle mode, režim nečinnosti

Ak bity SM2..0 sú nastavené na hodnotu 000, potom realizácia inštrukcie SLEEP spôsobí, že MCU prejde do režimu nečinnosti. V tomto režime je zastavená činnosť CPU, ale SPI, USART, analógový komparátor, ADC, TWSI, čítače/časovače, watchdog a prerušovací systém pokračujú v činnosti. Je zrejmé, že v tomto režime je zastavený hodinový signál clkCPU a clkFLASH. Z režimu nečinnosti sa MCU dostane na základe výskytu žiadosti o prerušenie od povolených interných a externých zdrojov prerušenia.

### ADC noise reduction mode, režim redukcie šumu

Ak sú bity SM0..2 nastavené na hodnotu 001, potom realizácia inštrukcie SLEEP spôsobí, že MCU prejde do režimu redukcie šumu. Tento režim zastaví CPU a niektoré I/O obvody. V režime redukcie šumu budú zastavené nasledovné hodinové signály: clkIO, clkCPU a clkFLASH. Toto spôsobí zníženie úrovne šumu generovaného digitálnymi obvodmi a zvýši sa presnosť A/D prevodu. Z režimu redukcie šumu sa MCU dostane na základe výskytu žiadosti o prerušenie od nasledovných povolených zdrojov: koniec ADC prevodu, externý reštart, watchdog reštart, Brown-out reštart, TWSI zhoda adresy, čítač/časovač2, SPM a EEPROM pripravené a externé prerušenia na úrovni INT0 a INT1.

## Power-down mode, režim so zníženou spotrebou

Ak sú bity SM0..2 nastavené na hodnotu 010, potom realizácia inštrukcie SLEEP spôsobí, že MCU prejde do režimu so zníženou spotrebou. V tomto režime je zastavený externý oscilátor. Externé prerušenia, TWSI a watch-dog pokračujú v činnosti. Z režimu so zníženou spotrebou sa MCU dostane na základe výskytu žiadosti o prerušenie od nasledovných povolených zdrojov: externý reštart, watchdog reštart, Brown-out reštart, TWSI zhoda adresy a externé prerušenia na úroveň INT0 a INT1. V tomto režime sú pozastavené generátory hodinových signálov a v činnosti zostávajú len asynchrónne pracujúce moduly.

## Power-save mode, režim s úspornou spotrebou

Ak sú bity SM0..2 nastavené na hodnotu 011, potom realizácia inštrukcie SLEEP spôsobí, že MCU prejde do režimu s úspornou spotrebou. Tento režim je identický s predchádzajúcim režimom so zníženou spotrebou s nasledovnou výnimkou: Ak čítač/časovač 2 pracuje asynchrónne (bit AS2 v ASSR je nastavený), bude v tomto režime aktívny a v prípade výskytu žiadosti o prerušenie prejde MCU do aktívneho stavu. Ak nie je povolená asynchrónna činnosť čítača/časovača2 odporúča sa používať režim so zníženou spotrebou. V tomto režime sú pozastavené všetky hodinové signály s výnimkou clkASY.

## Standby mode, pohotovostný režim

Ak sú bity SM0..2 nastavené na hodnotu 110 a CPU využíva ako zdroj hodinového signálu generátor s externým kryštálom/rezonátorom, potom realizácia inštrukcie SLEEP spôsobí, že MCU prejde do pohotovostného režimu. Tento režim je identický s režimom so zníženou spotrebou s výnimkou, že oscilátor zostáva v činnosti. Z tohto režimu MCU prechádza do aktívneho režimu v priebehu 6 hodinových periód.

Sleep Mode	Active Clock Domains					Oscillators		Wake-up Sources					
	clkCPU	clkFLASH	clkIO	clkADC	clkASY	Main Clock Source Enabled	Timer Osc. Enabled	INT1 INTO	TWI Address Match	Timer 2	SPM/EEPROM Ready	ADC	Other I/O
Idle			X	X	X	X	X <sup>(2)</sup>	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X <sup>(2)</sup>	X <sup>(3)</sup>	X	X	X	X	X
Power Down								X <sup>(3)</sup>	X				
Power Save					X <sup>(2)</sup>		X <sup>(2)</sup>	X <sup>(3)</sup>	X	X <sup>(2)</sup>			
Standby <sup>(1)</sup>						X		X <sup>(3)</sup>	X				

Notes: 1. External Crystal or resonator selected as clock source  
2. If AS2 bit in ASSR is set  
3. Only level interrupt INT1 and INTO

Obr 13.1 - Popis jednotlivých režimov spánku

## Minimalizácia spotreby

Pri minimalizovaní spotreby mikropočítača sa snažíme v maximálnej miere využiť možnosti režimov spánku, ktoré nám mikropočítač ponúka. Niektoré periférie mikropočítača však potrebujú osobitné nastavenie, ak požadujeme znížiť spotrebu do maximálnej možnej miery.

Ďalšie možnosti zníženia spotreby teda sú:

- Vypnúť AD prevodník (bit ADEN v ADCSRA)
- Vypnúť Analógový komparátor (bit ACD v registri ACSR)
- Vypnúť Brown-out Detect (nastavenie BODEN v poistkách)
- Vypnúť Watch-dog timer (bit WDE v registri WDTCR)

Na spotrebu mikropočítača nielen v režimoch spánku má najväčší vplyv frekvencia hodinového signálu a pracovné napätie. Preto sa snažíme voliť kompromis medzi výkonom a spotrebou mikropočítača.

V tabuľke č. 13.1 som meraním zisťoval spotrebu mikropočítača ATmega8 v rôznych režimoch činnosti. Ako zdroj hodín som využíval interný RC oscilátor s rôznou frekvenciou. Počas merania bol vypnutý Brown-out detektor, AD prevodník, Watchdog-timer a Analógový komparátor.

Najväčší vplyv na spotrebu vo všetkých režimoch činnosti mal AD prevodník, ktorý odoberal po zapnutí prúd 0,3mA.

Režim spánku	Napätie	Oscilátor	Spotreba
Bez spánku	5V	1MHz	3,4mA
Bez spánku	5V	8MHz	11,4mA
Bez spánku	3,3V	1MHz	1,63mA
Bez spánku	3,3V	8MHz	5,3mA
Idle	5V	1MHz	2,5mA
Idle	5V	8MHz	6,8mA
Idle	3,3V	1MHz	0,6mA
Idle	3,3V	8MHz	2,9mA
Power Down	5V	8MHz	0,7uA
Power Save	5V	8MHz	0,7uA

Tab. 13.1 - Meranie spotreby MCU ATmega8 v rôznych režimoch činnosti

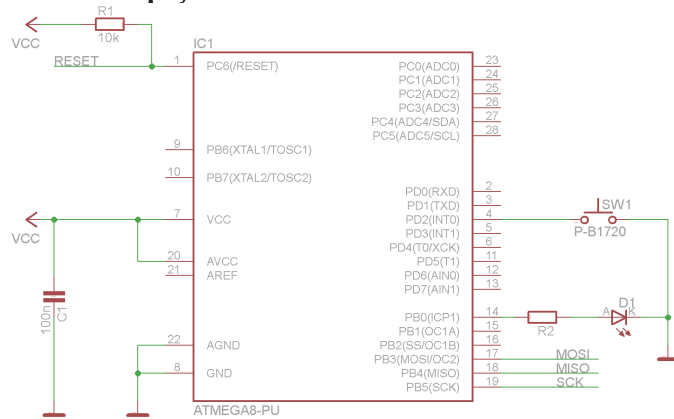
Vidíme teda, že najmenšiu spotrebu dosiahneme s režimami Power Down alebo Power Save.

## Praktická realizácia režimov spánku u mikropočítača

Priamo na prácu s režimami spánku slúži knižnica (`avr/sleep.h`), ktorú som využíval aj v nasledovných príkladoch.

Pomocou funkcie `set_sleep_mode(<mode>)`; nastavíme cez parameter funkcie požadovaný režim spánku a funkciou `sleep_mode(void)`; daný režim spánku aktivujeme.

### Schéma zapojenia:



Obr 13.2 - Schéma zapojenia MCU pre test režimov spánku

### Príklady:

#### Príklad č. 1:

Nastav externé prerušenie generované dobežnou hrannou signálom a prejdí do režimu spánku – Idle mód. Po zobudení rozsvieť po 2sec LED.

#### Zdrojový kód:

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/sleep.h>
```

```
int main(void){
    //nastavenie vyst. portu pre LED
    DDRB |= (1 << PB0);
    PORTB &= ~(1 << PB0);

    // nastavenie ext. prerusenia, dobezna hrana signalu
    DDRD &= ~(1 << PD2);
    PORTD |= (1 << PD2);
    GICR |= (1 << INT0);
```

```
MCUCR |= (1 << ISC01);
sei();
```

```
//nastav sleep mod
set_sleep_mode(SLEEP_MODE_IDLE);
sleep_mode();
```

```
//po prebudení
_delay_ms(2000);
PORTB |= (1 << PB0);
```

```
return 0;
}
```

```
ISR(INT0_vect) {
}
```

### Youtube video:

Test Idle režimu spánku

#### Príklad č. 2:

Nastav externé prerušenie generované nízkou úrovňou signálu, prejdí do režimu spánku – Power Down. Po zobudení rozsvieť po 2sec LED.

**Pozn.:** Pri režimoch spánku, okrem režimu Idle, je možné použiť externé prerušenie jedine v režime, keď je generované nízkou úrovňou signálu. Preto po zobudení musíme zakázať externé prerušenia, nakoľko inak by mikropočítač po zobudení neustále generoval prerušenia pokiaľ by bol stav nízkej úrovne signálu.

#### Zdrojový kód:

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <util/delay.h>
#include <avr/sleep.h>
```

```
int main(void){
    // nastavenie vyst. portu pre LED
    DDRB |= (1 << PB0);
    PORTB &= ~(1 << PB0);

    // nastavenie ext. prerusenia, nizka uroven napatia
    DDRD &= ~(1 << PD2);
    PORTD |= (1 << PD2);
    GICR |= (1 << INT0);
    sei();

    // nastav režim spánku
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);
    sleep_mode();
```

## REŽIMY SPÁNKU MCU

---

```
// zrus externe prerusenien  
cli();  
  
//pockaj 2s a potom rozsviet LED  
_delay_ms(2000);  
PORTB |= (1 << PB0);  
  
return 0;  
}  
  
ISR(INT0_vect)  
{  
  
}
```

Youtube video:

Test režimu Power-down



# POUŽITÁ LITERATÚRA

**DOC. ING. JURAJ MIČEK PHD.:** "MIKROKONTROLERY ATMEL AVR, ATMEGA8, POPIS, PROGRAMOVANIE, APLIKACIE", Žilinská univerzita 2004

**ATMEL:** "KATALÓGOVÝ LIST OBVODU ATMEGA8", [ONLINE]  
<http://www.atmel.com/images/doc2486.pdf>

**WIKIPEDIA:** "SERIAL PERIPHERAL INTERFACE BUS" [ONLINE]  
[http://en.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface\\_Bus](http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus)

**WIKIPEDIA:** "IMPULZOVÁ ŠÍRKOVÁ MODULÁCIA" [ONLINE]  
<http://sk.wikipedia.org/wiki/PWM>

**WINAVR.SCIENCEPROG.COM:** "PRÁCA S DESATINNÝMI ČÍSLAMI POMOCOU FUNKCIE SPRINTF V PROGRAME AVR STUDIO 4" [ONLINE]  
<http://winavr.scienceprog.com/avr-gcc-tutorial/using-sprintf-function-for-float-numbers-in-avr-gcc.html>





© 2012 Bc. Ondrej Závodský