

MCU

ATmega8

příklady a aplikace

Žilina 2006

Juraj MIČEK

Tento text vznikol len vďaka nízkej pracovnej zaťaženosti autora v priebehu roku 2005, kedy bola realizovaná väčšina uvádzaných príkladov. Text obsahuje mnoho chýb a nepresností. Je zrejmé, že príklady v texte je možné riešiť aj efektívnejšie (čo do dĺžky kódu, alebo rýchlosti spracovania), ale mojou snahou bolo uviesť typické riešenia a súčasne motivovať čitateľa k ďalšej práci tým, že po niekoľkých dňoch dokáže sám nájsť „krajšie“ riešenia než tie, ktoré uvádzam. Napriek môjmu presvedčeniu, že venovať sa popisu technických prostriedkov nemá veľký význam, sa týmto textom pokúšam dať dohromady izolované príklady a popis niektorých aplikácií. Architektúra AVR a vývojové prostredie AVR studio sú ešte natoľko jednoduché, že sa im nepodarilo vzdialiť sa od obvodových prostriedkov, tak aby sa stratili základné princípy práce počítača (inštrukčný cyklus, zreťazenie inštrukcií, prerušovací systém a jeho obsluha, zásobník a mnoho ďalších základných princípov a pojmov). Práve vďaka tejto jednoduchosti sa mi zdá, že architektúra AVR a podporné programové prostriedky sú ideálnou cestou na pochopenie práce počítača Harwardskej architektúry. Prepokladám, že do desiatich rokov budeme môcť prácu počítača popisovať výlučne teoreticky, čím stratíme ďalšiu cestu, ktorá nám pomáha učiť sa formou zábavy.

OBSAH

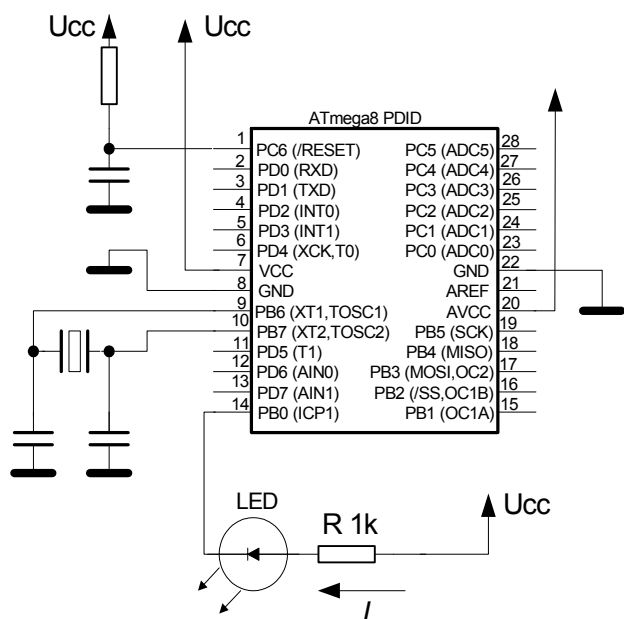
1. Indikácia a zobrazovanie údajov	4
2. Prerušovací podsystem	32
3. Univerzálny synchrónny/asynchrónny vysielateľ/prijímač USART	37
4. Časovače/čítače obvodu ATmega8	49
5. Analógový komparátor	78
6. Pamäť EEPROM	84
7. Analógový číslicový prevodník	90
8. A na záver	101

1. Indikácia a zobrazovanie údajov

Na zobrazovanie údajov, prípadne na indikovanie vybraných stavov MCU je možné využiť rôzne zobrazovacie prvky. Od najjednoduchších prvkov (LED dióda) až po zložité grafické LCD zobrazovacie jednotky. V nasledujúcich príkladoch sú uvedené niektoré jednoduché aplikácie slúžiace na komunikáciu MCU s užívateľom.

Príklad 1

Prvá úloha je veľmi jednoduchá. Je potrebné napísať program na rozsvietenie LED diódy pripojenej na vývod 0 portu B, (PB0) podľa obr.1.



Obr.1 Zapojenie LED diódy

Je zrejmé, že ak LED diódou bude pretekať dostatočný prúd I , LED dióda sa rozsvieti. Aby sme túto úlohu úspešne vyriešili je potrebné nastaviť smer vývodu PB0 na výstupný a nastaviť ho na hodnotu log.0. Potom na vývode PB0 bude napätie blízke potenciálu GND (0V) a pre prúd pretekajúci LED diódou bude platiť:

$$I = \frac{U_{cc} - U_{LED}}{R},$$

kde I je prúd pretekajúci diódou

U_{cc} je hodnota napájacieho napätia, 5V,

U_{LED} je úbytok napätia na LED dióde, približne 2V

Po dosadení hodnôt je možné určiť veľkosť prúdu pretekajúceho diódou LED, $I=3mA$.

Poznamenajme, že prúdová zaťažiteľnosť V/V vývodu obvodu ATmega8 je 20mA. Navrhované riešenie je preto z pohľadu dovolenej prúdovej zaťažiteľnosti V/V výstupu vyhovujúce.

Program napísaný v integrovanom vývojovom prostredí AVR Studio:

```

;program:LED1.asm zasvietenie LED diódy
;LED dióda je ovládaná vývodom PB0, ak je
;PB0 konfigurovaný ako výstup a do PORTB,0 zapíšeme
;log.1 dióda LED svieti pozn. zaťažiteľnosť V/V je 20 mA

.include "m8def.inc" ; definičný súbor ATmega8

.def temp=r16          ;definovanie symb. mena
                        ;na register r16 sa môžeme
                        ;odvolávať pomocou mena temp

.cseg                  ;segment programu
.org 0x0               ;hex kód uložiť od adresy 0

    rjmp RESET        ; skok na štart programu
                        ;nasleduje tabuľka vektorov prerušení, zatiaľ ich
                        ;využívať nebudeme, pretože budú globálne zakázané
                        ;prerušenia. Nič by sa nestalo, ak by sme program
                        ;písali od adr.0. Ale pre budúcu modifikáciu programu
                        ;necháme 18 voľných pozícií "nop" pre umiestnenie prer.
                        ;v poznámke sú uvedené skoky na príslušné obsluhy prer.
nop; rjmp EXT_INT0    ;IRQ handler
nop; rjmp EXT_INT1
nop; rjmp TIM2_COMP
nop; rjmp TIM2_OVF
nop; rjmp TIM1_CAPT
nop; rjmp TIM1_COMPA
nop; rjmp TIM1_COMPB
nop; rjmp TIM1_OVF
nop; rjmp TIM0_OVF
nop; rjmp SPI_STC
nop; rjmp USART_RXC
nop; rjmp USART_UDRE
nop; rjmp USART_TXC
nop; rjmp ADCC
nop; rjmp EE_RDY
nop; rjmp ANA_COMP
nop; rjmp TWSI
nop; rjmp SPM_RDY

RESET:
    ldi        temp,high(RAMEND) ;Nastavenie ukazovateľa zásobníka-SP
    out        SPH,temp          ;na koniec pamäte RAM
    ldi        temp,low(RAMEND)  ;poznamenajme,že zásobník zatiaľ
    out        SPL,temp          ;nepotrebuje
;***** Nastavenie smeru portu B *****
    ldi        temp,0x01
    out        DDRB,temp         ;do DDRB,0 log.1,PB,0 výstup

```

```

;*****
;
;          cbi          PORTB,0          ;nulovanie bitu PORTB,0
;                                     ;teraz by mala LED svietiť

STOP: rjmp  STOP          ;tradičné skončenie
;                                     ;jednoduchých programov
; MCU nastavil PB0 na log.1 a čaká v slučke „skok na návěstie STOP“

```

Pripomeňme, že bodkočiarka označuje začiatok komentárov. Text za bodkočiarkou až do konca príslušného riadku prekladač pokladá za komentár a pri preklade ho ignoruje.

Prvá časť programu obsahuje príkazy prekladača. Sú to:

.include - oznamuje prekladaču, aby pri preklade zaviedol súbor s názvom „*m8def.inc*“, ktorý obsahuje definície a priradenia symbolických mien príslušným fyzickým adresám.

.def – priradí symbolické meno „*temp*“ registru *r16*. Toto priradenie slúži len na prehľadnenie programu. V priebehu písania programu je možné využívať symbolické meno *temp*, a súčasne aj označenie registra *r16*. Vzhľadom k lepšej orientácii v programe však využívame len symbolické meno.

.cseg – oznamuje prekladaču, že nasleduje segment kódu programu. Táto voľba je v prekladači prednastavená.

.org – informuje prekladač od akej adresy má ukladať preložený kód programu. V našom prípade od adresy 0.

Prvá inštrukcia *rjmp RESET* spôsobí, že realizácia programu bude pokračovať inštrukciou na návěstí *RESET*. Tu sa nastaví obsah ukazovateľa zásobníka (*SPH,SPL*) na koniec pamäte RAM – (*RAMEND*). Hodnota *RAMEND* je definovaná v súbore *m8def.inc*. Zásobník slúži na uchovávanie návratových adries pri volaniach podprogramov a obsluhu prerušení, ako aj na uchovanie obsahov dôležitých registrov na základe rozhodnutia programátora. V uvádzanom programe zásobník nie je využívaný.

Pomocou nasledujúcich inštrukcií *ldi temp,0x01* a *out DDRB,temp* zapíšeme do registra určujúceho smer brány B – *DDRB* hodnotu 0b00000001. Takto je definovaný smer jednotlivých vývodov brány B. Vývod PB0 je výstupný, zvyšné vývody brány B sú nastavené ako vstupné. Nakoniec inštrukciou *cbi PORTB,0* nastavíme vývod PB0 na úroveň odpovedajúcu log.0, čo spôsobí rozsvietenie LED diódy. Po rozsvietení diódy program končí v slučke *STOP: rjmp STOP*.

Príklad 2:

Predchádzajúci príklad modifikujeme tak, aby LED dióda blikala. To je aby opakovane 1s svietila a ďalšiu bola zhasnutá.

```

;program:LED2.asm blikanie LED diódy
;LED dióda je ovládaná vývodom PB0, ak je
;PB0 konfigurovaný ako výstup a do PORTB,0 zapíšeme
;log.0 dióda LED svieti pri zápise log.1 zahasne

```

```
.include "m8def.inc" ; definíčný súbor ATmega8
```

```
.def temp=r16 ;definovanie symb. mena  
 ;na register r16 sa môžeme  
 ;odvolávať pomocou mena temp
```

```
.def temp1=r17
```

```
.def temp2=r18
```

```
.cseg ;segment programu  
.org 0x0 ;hex kód uložiť od adresy 0
```

```
 rjmp RESET ; skok na štart programu  
 ;nasleduje tabuľka vektorov prerušení, zatiaľ ich  
 ;využívať nebudeme, pretože budú globálne zakázané  
 ;prerušená. Nič by sa nestalo, ak by sme program  
 ;písali od adr.0. Teraz pre zmenu namiesto nop napíšeme na príslušné  
 ;adresy inštrukciu reti - návrat z podprogramu. V prípade, že  
 ;povolíme prerušená a niektoré z prerušení sa vyskytnú,  
 ;potom sa výkon programu prostredníctvom inštrukcie reti hneď vracia do hlavného  
 ;programu.
```

```
reti; rjmp EXT_INT0 ;IRQ handler  
reti; rjmp EXT_INT1  
reti; rjmp TIM2_COMP  
reti; rjmp TIM2_OVF  
reti; rjmp TIM1_CAPT  
reti; rjmp TIM1_COMPA  
reti; rjmp TIM1_COMPB  
reti; rjmp TIM1_OVF  
reti; rjmp TIM0_OVF  
reti; rjmp SPI_STC  
reti; rjmp USART_RXC  
reti; rjmp USART_UDRE  
reti; rjmp USART_TXC  
reti; rjmp ADCC  
reti; rjmp EE_RDY  
reti; rjmp ANA_COMP  
reti; rjmp TWSI  
reti; rjmp SPM_RDY
```

```
RESET:
```

```
 ldi temp,high(RAMEND) ;Nastavenie ukazovateľa zásobníka-SP  
 out SPH,temp ;na koniec pamäte RAM  
 ldi temp,low(RAMEND) ;poznajme, že zásobník zatiaľ  
 out SPL,temp ;nepotrebujeme  
 ;***** Nastavenie smeru portu B *****  
 ldi temp,0x01  
 out DDRB,temp ;do DDRB,0 log.1,PB,0 výstup
```

```

;*****
SKOK:      cbi          PORTB,0          ;nulovanie bitu PORTB,0
                                                ;teraz by mala LED svietit'

;*****čakacia slučka cca 82*255*255*3*62.5 ns 0.999759 s

C3:        ldi          temp,82          ;reg.temp naplníme hodnotou 82
C2:        ldi          temp1,0xff       ;reg.temp1 naplníme hodnotou 0xff=255
C1:        ldi          temp2,0xff       ;reg.temp2 naplníme hodnotou 0xff
          dec          temp2
          brne C1
          dec          temp1
          brne C2
          dec          temp
          brne C3
;*****koniec čakacej slucky*****

          sbi          PORTB,0          ;nastavenie bitu PORTB,0
                                                ;teraz by mala LED zhasnúť

;***** a zase počkáme približne sekundu
;musíme zmeniť označenie návěstí v čakacej slučke

C6:        ldi          temp,82          ;reg.temp naplníme hodnotou 82
C5:        ldi          temp1,0xff       ;reg.temp1 naplníme hodnotou 0xff=255
C4:        ldi          temp2,0xff       ;reg.temp2 naplníme hodnotou 0xff
          dec          temp2
          brne C4
          dec          temp1
          brne C5
          dec          temp
          brne C6
;*****koniec čakacej slucky*****

          rjmp SKOK          ;skok na návěstie SKOK

```

; MCU neustále vykonáva inštrukcie od návěstia SKOK po rjmp SKOK

Program z príkladu 1 je doplnený o dve čakacie slučky, ktoré spôsobia, že po zasvetení LED diódy procesor vykonáva cca 1s čakaciu slučku a potom diódu zhasne - inštrukcia *sbi PORTB,0*. Následne realizuje ďalšiu čakaciu slučku a pokračuje na návěstí *SKOK*, kde opätovne zasvieti LED diódu. Toto sa opakuje až do reštartu procesora, prípadne do odpojenia napájacieho napätia.

V uvedenom programe sa vyskytujú skoro identické časti programu – čakacie slučky. Aby sme ich nemuseli opakovane písať je možné využiť tzv. MAKRO. Využitie macra je ilustrované v nasledujúcom programe.


```
;program:LED3.asm blikanie LED diódy s využitím MAKRA
;LED dióda je ovládaná vývodom PB0, ak je
;PB0 konfigurovaný ako výstup a do PORTB,0 zapíšeme
;log.0 dióda LED svieti pri zápise log.1 zahasne
```

```
.include "m8def.inc" ; definíčný súbor ATmega8
```

```
.def temp=r16 ;definovanie symb. mena
 ;na register r16 sa môžeme
 ;odvolávať pomocou mena temp
.def temp1=r17
.def temp2=r18
```

```
.macro CAKAJ ;definujeme MAKRO bez parametrov
```

```
 ;*****čakacia slučka cca 82*255*255*3*62.5 ns 0.999759 s
```

```

C3:      ldi    temp,82          ;reg.temp naplníme hodnotou 82
        ldi    temp1,0xff       ;reg.temp1 naplníme hodnotou 0xff=255
C2:      ldi    temp2,0xff       ;reg.temp2 naplníme hodnotou 0xff
C1:      dec    temp2
        brne   C1
        dec    temp1
        brne   C2
        dec    temp
        brne   C3
```

```
.endmacro ;koniec MAKRA
```

```
.cseg ;segment programu
.org 0x0 ;hex kód uložiť od adresy 0
```

```

        rjmp   RESET           ; skok na štart programu
;nasleduje tabuľka vektorov prerušení, zatiaľ ich
;využívať nebudeme, pretože budú globálne zakázané
;prerušená. Nič by sa nestalo, ak by sme program
;písali od adr.0. Ale pre budúcu modifikáciu programu
;do 18. voľných pozícií teraz pre zmenu napíšeme rjmp PRER, čo pri
;výskyte prerušená spôsobí skok na návštevu PRER a odtiaľ
;návrat z obsluhy prerušená ( reti ).
```

```

        rjmp   PRER ;      rjmp   EXT_INT0 ;IRQ handler
        rjmp   PRER ;      rjmp   EXT_INT1
        rjmp   PRER ;      rjmp   TIM2_COMP
        rjmp   PRER ;      rjmp   TIM2_OVF
        rjmp   PRER ;      rjmp   TIM1_CAPT
        rjmp   PRER ;      rjmp   TIM1_COMPA
        rjmp   PRER ;      rjmp   TIM1_COMPB
        rjmp   PRER ;      rjmp   TIM1_OVF
        rjmp   PRER ;      rjmp   TIM0_OVF
```

```

rjmp PRER ;      rjmp SPI_STC
rjmp PRER ;      rjmp USART_RXC
rjmp PRER ;      rjmp USART_UDRE
rjmp PRER ;      rjmp USART_TXC
rjmp PRER ;      rjmp ADCC
rjmp PRER ;      rjmp EE_RDY
rjmp PRER ;      rjmp ANA_COMP
rjmp PRER ;      rjmp TWSI
rjmp PRER ;      rjmp SPM_RDY

```

RESET:

```

ldi temp,high(RAMEND) ;Nastavenie ukazovateľa zásobníka-SP
out SPH,temp          ;na koniec pamäte RAM
ldi temp,low(RAMEND)
out SPL,temp
;***** Nastavenie smeru portu B*****
ldi temp,0x01
out DDRB,temp          ;do DDRB,0 log.1,PB,0 výstup

;*****
SKOK: cbi PORTB,0          ;nulovanie bitu PORTB,0
                                ;teraz by mala LED svietiť

CAKAJ
sbi PORTB,0          ;nastavenie bitu PORTB,0
                                ;teraz by mala LED zhasnúť

CAKAJ
rjmp SKOK          ;skok na návěstie SKOK

; MCU neustále vykonáva inštrukcie od návěstia SKOK po rjmp SKOK

```

PRER: reti

Poznamenajme, že kód programu po preklade bude s výnimkou ďalšej modifikácie tabuľky vektorov prerušení identický s predchádzajúcim programom.

Uvedenú úlohu môžeme riešiť tak, že časť programu, ktorá odpovedá čakacej slučke napíšeme ako podprogram. Teraz ak v priebehu výkonu programu budeme musieť čakať, potom budeme volať podprogram *CAKAJ*. Uvedme výpis takto upraveného programu.

```

;program:LED4.asm blikanie LED diódy s využitím podprogramu CAKAJ
;LED dióda je ovládaná vývodom PB0, ak je
;PB0 konfigurovaný ako výstup a do PORTB,0 zapíšeme
;log.0 dióda LED svieti pri zápise log.1 zahasne

```

```

.include "m8def.inc" ;definičný súbor ATmega8

```

```

.def temp=r16          ;definovanie symb. mena

```

```

;na register r16 sa môžeme
;odvolávať pomocou mena temp

.def    temp1=r17
.def    temp2=r18

.cseg                                       ;segment programu
.org    0x0                                ;hex kód uložiť od adresy 0

        rjmp    RESET                    ; skok na štart programu
;nasleduje tabuľka vektorov prerušení, zatiaľ ich
;využívať nebudeme, pretože budú globálne zakázané
;prerušená. Nič by sa nestalo, ak by sme program
;písali od adr.0. Ale pre budúcu modifikáciu programu
;necháme 18 voľných pozícií "nop" pre umiestnenie prer.

        nop;    rjmp    EXT_INT0        ;IRQ handler
        nop;    rjmp    EXT_INT1
        nop;    rjmp    TIM2_COMP
        nop;    rjmp    TIM2_OVF
        nop;    rjmp    TIM1_CAPT
        nop;    rjmp    TIM1_COMPA
        nop;    rjmp    TIM1_COMPB
        nop;    rjmp    TIM1_OVF
        nop;    rjmp    TIM0_OVF
        nop;    rjmp    SPI_STC
        nop;    rjmp    USART_RXC
        nop;    rjmp    USART_UDRE
        nop;    rjmp    USART_TXC
        nop;    rjmp    ADCC
        nop;    rjmp    EE_RDY
        nop;    rjmp    ANA_COMP
        nop;    rjmp    TWSI
        nop;    rjmp    SPM_RDY

RESET:
        ldi     temp,high(RAMEND)        ;Nastavenie ukazovateľa zásobníka-SP
        out     SPH,temp                 ;na koniec pamäte RAM
        ldi     temp,low(RAMEND)
        out     SPL,temp
;***** Nastavenie smeru portu B*****
        ldi     temp,0x01
        out     DDRB,temp                ;do DDRB,0 log.1,PB,0 výstup

;*****
SKOK: cbi     PORTB,0                    ;nulovanie bitu PORTB,0
                                           ;teraz by mala LED svietiť
        rcall   CAKAJ                    ;volanie podprogramu
        sbi     PORTB,0                  ;nastavenie bitu PORTB,0
                                           ;teraz by mala LED zhasnúť
        rcall   CAKAJ                    ;volanie podprogramu

```

```

    rjmp    SKOK                ;skok na návěstie SKOK

; MCU neustále vykonáva inštrukcie od návěstia SKOK po rjmp SKOK

;podprogram CAKAJ
CAKAJ:
;*****čakacia slučka cca 82*255*255*3*62.5 ns 0.999759 s
    ldi     temp,82             ;reg.temp naplníme hodnotou 82
C3:  ldi     temp1,0xff          ;reg.temp1 naplníme hodnotou 0xff=255
C2:  ldi     temp2,0xff          ;reg.temp2 naplníme hodnotou 0xff
C1:  dec     temp2
    brne    C1
    dec     temp1
    brne    C2
    dec     temp
    brne    C3
    ret                        ;návrat z podprogramu
;*****koniec čakacej slučky*****

```

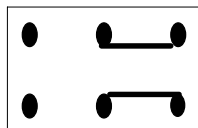
Uviedli sme tri modifikácie programu, ktoré riešia tú istú úlohu. Je nesporné, že uvedenú úlohu je možné úspešne vyriešiť pomocou iných programov, napr. s využitím integrovaných časovačov/čítačov a pod.

Poznamenajme, že posledný program s využitím podprogramu *CAKAJ* je z pohľadu dĺžky kódu úspornejší než predchádzajúce verzie. Pre zaujímavosť uvedme dĺžku preloženého kódu jednotlivých programov:

Program	Dĺžka kódu v pamäti FLASH [slov]
<i>LED2</i>	
<i>LED3</i>	
<i>LED4</i>	

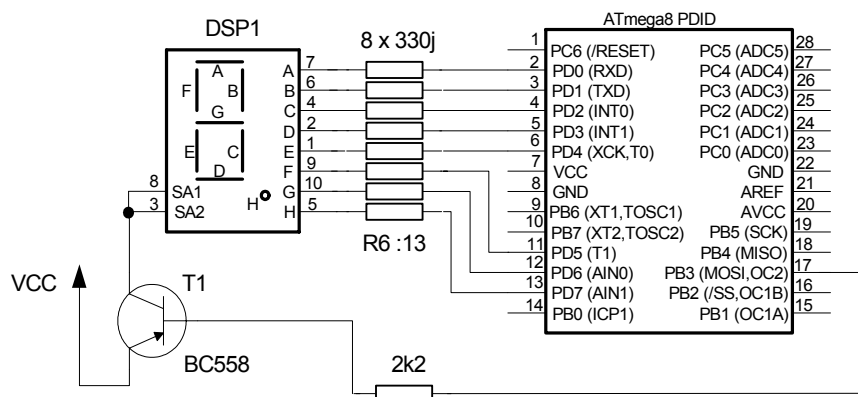
Príklad 3.

Na 7-segmentovej zobrazovacej jednotke DSP1 zobrazme obsah spodnej polovice registra *r20*. Pripomeňme, že vývody PD6 a PD7 sú na DSP pripojené prostredníctvom prepojky P5. Pre úspešné riešenie tejto úlohy musí byť prepojka P5 nastavená podľa obr.2.



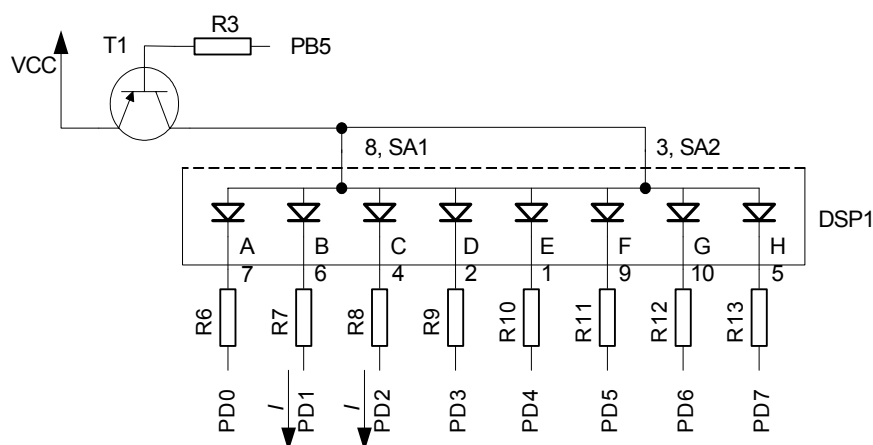
obr.2 Nastavenie prepojky P5

Schéma zapojenia DSP1 je uvedená na obr.3.



Obr.3 Zapojenie 7-segmentového zobrazovacieho prvku DSP1

Vnútna štruktúra a zapojenie zobrazovacieho prvku DSP1 je znázornená na obr.4.



Obr 4. Vnútna štruktúra a zapojenie DSP1

Ak chceme zobraziť napríklad symbol „1“, je potrebné rozsvietiť segmenty *B* a *C*, pričom všetky ostatné segmenty majú byť zhasnuté. Príslušné segmenty *B* a *C* budú svietiť vtedy, ak bude nimi pretekať dostatočne veľký prúd. Preto anódy DSP1 *SA* je potrebné pripojiť na napájacie napätie - v uvedenom prípade *VCC*, prostredníctvom tranzistora PNP, *T1*. Tranzistor *T1* bude zopnutý, ak na jeho báze je úroveň napätia odpovedajúca log.0. To znamená, že do bitu 5, *V/V* brány *B* (*PORTB*,5) je potrebné zapísať log.0. Katódy diód segmentov, ktoré budú svietiť musia byť na nízkom potenciáli. Preto do bitov *PORTD*,1 a *PORTD*,2 je potrebné zapísať log.0. Prúd prechádzajúci diódami *B* a *C* bude:

$$I = \frac{VCC - V_{LED}}{R1},$$

kde *VCC* je hodnota napájacieho napätia, (5V),
VLED je úbytok napätia na LED dióde (1.85V) a

RI je odpor zapojený medzi katódu diódy a príslušným výstupom MCU.

Prúd pretekajúci diódou za predpokladu je pri splnení predchádzajúcich podmienok je 9.5 mA.

Výpis programu:

```

;program:LED5.asm zobrazí na DSP1 obsah spodnej polovice
;registra r20
;na bázu tranzistora DSP je potrebné priviesť nízke
;napätie (log.0) aby príslušný DSP bol aktívny VCC na anódu
;na katódy log.0/log.1 podľa zobrazovaného znaku

.include "m8def.inc" ;definičný súbor ATmega8
.def    temp=r16      ;definovanie symbolického mena
.def    temp1=r17
.def    temp2=r18
;::::::::::::::::::TABUĽKA PRIRADENÍ
.equ    nula=0b11000000
.equ    jedna=0b11111001
.equ    dva=0b10100100
.equ    tri=0b10110000
.equ    styri=0b10011001
.equ    pat=0b10010010
.equ    sest=0b10000010
.equ    sedem=0b11111000
.equ    osem=0b10000000
.equ    devat=0b10010000
.equ    ahex=0b10001000
.equ    bhex=0b10000011
.equ    chex=0b10100111
.equ    dhex=0b10100001
.equ    ehex=0b10000110
.equ    fhex=0b10001110

.cseg                                ;segment programu
.org    0x0                          ;hex kód uložiť od adresy 0
    rjmp RESET                      ;skok na štart programu
;nasleduje tabuľka vektorov prerušení, zatiaľ ich
;využívať nebudeme, pretože prer. sú globálne zakázané
    reti;    rjmp EXT_INT0          ;IRQ handler
    reti;    rjmp EXT_INT1
    reti;    rjmp TIM2_COMP
    reti;    rjmp TIM2_OVF
    reti;    rjmp TIM1_CAPT
    reti;    rjmp TIM1_COMPA
    reti;    rjmp TIM1_COMPB
    reti;    rjmp TIM1_OVF
    reti;    rjmp TIM0_OVF
    reti;    rjmp SPI_STC

```

```

reti; rjmp USART_RXC
reti; rjmp USART_UDRE
reti; rjmp USART_TXC
reti; rjmp ADCC
reti; rjmp EE_RDY
reti; rjmp ANA_COMP
reti; rjmp TWSI
reti; rjmp SPM_RDY

```

RESET:

```

ldi temp,high(RAMEND) ;Nastavenie ukazovateľa zásobníka-SP
out SPH,temp ;na koniec pamäte RAM
ldi temp,low(RAMEND)
out SPL,temp
;***** Nastavenie smeru portu B*****
ldi temp,0b00111000
out DDRB,temp ;výstup je na PB5,PB4 a PB3
ser temp ;nastavenie temp na 0xff
out DDRD,temp ;celý PORTD bude výstupný
out PORTD,temp
out PORTB,temp ;zhasnute všetky DSP
ldi r20,0x01 ;chceme zobrazit "1"
rcall ZOBRAZ ;volanie podprogramu
cbi PORTB,3 ;svieti DSP1
; ak chceme zobrazit jednotku aj na DSP2, potom
; cbi PORTB,4
; ak chceme zobrazit jednotku aj na DSP3, potom
; cbi PORTB,5

```

KON: rjmp KON ;procesor čaká v cykle

;podprogram zobraz zobrazí hodnotu spodných 4 bitov r20

ZOBRAZ:

```

ldi r21,nula ;do r21 ulož hodnotu „nula“=0b11000000
andi r20,0x0f ;maskovanie horných štyroch bitov r20
breq Z1 ;skoč na návěstie Z1 ak výsledok bol 0
ldi r21,jedna ;do r21 ulož hodnotu „jedna“
dec r20 ;r20=r20 -1
breq Z1 ;ak výsledok predchádzajúcej operácie
;bol rovný 0, potom skoč na návěstie Z1
; atď.
ldi r21,dva
dec r20
breq Z1
ldi r21,tri
dec r20
breq Z1
ldi r21,styri

```

```

dec    r20
breq   Z1
ldi    r21,pat
dec    r20
breq   Z1
ldi    r21,sest
dec    r20
breq   Z1
ldi    r21,sedem
dec    r20
breq   Z1
ldi    r21,osem
dec    r20
breq   Z1
ldi    r21,devat
dec    r20
breq   Z1
ldi    r21,ahex
dec    r20
breq   Z1
ldi    r21,bhex
dec    r20
breq   Z1
ldi    r21,chex
dec    r20
breq   Z1
ldi    r21,dhex
dec    r20
breq   Z1
ldi    r21,ehex
dec    r20
breq   Z1
ldi    r21,fhex

```

```

Z1:    out    PORTD,r21        ;vyslanie obsahu r21 na PORTD
        ret                    ;návrat z podprogramu

```

Program *LED5.asm* na začiatku obsahuje príkazy prekladača (*.include*, *.def*, *.org*, *.cseg*), ktoré boli bližšie vysvetlené v predchádzajúcich príkladoch. Príkaz *.equ xx=zz* priradzuje symbolickému menu *xx* hodnotu *zz*. To znamená, že v priebehu písania programu môžeme využívať symbolické meno namiesto priradenej hodnoty. Potom výraz *ldi r21,nula* je po definovaní hodnoty (*.equ nula=0b11000000*) ekvivalentný výrazu *ldi r21,0b11000000*. V úvode programu sú 16-tim symbolickým menám *nula* až *fhex* priradené také hodnoty, ktoré po zapísaní do V/V brány D, (PORTD) spôsobia, že diódy odpovedajúce danému symbolu budú svietiť. To znamená, že napríklad pri symbole *nula* budú svietiť segmenty A, B, C, D, E, F a segmenty G a H budú zhasnuté. Každý segment je ovládaný práve jedným vývodom brány D, viď obr.x.

Nasleduje tabuľka vektorov prerušení a nastavenie ukazovateľa zásobníka. Ďalším krokom je nastavenie smeru príslušných vývodov vstupno/výstupných brán. PORTD bude mať všetky vývody definované ako výstupné, preto do registra DDRD zapíšeme hodnotu 0xFF,

(0b11111111). Z V/V brány B sú bity PB3, PB4 a PB5 využité na spínanie tranzistorov T1 a T3, preto sú nastavené ako výstupné. Do registra smeru DDRB zapíšeme hodnotu 0x38, (0b00111000).

Potom do všetkých bitov brány D a B, (PORTB a PORTD) zapíšeme hodnotu log.1, čo spôsobí, že žiadny segment na zobrazovacích prvkoch DSP1 až DSP3 nebude svietiť.

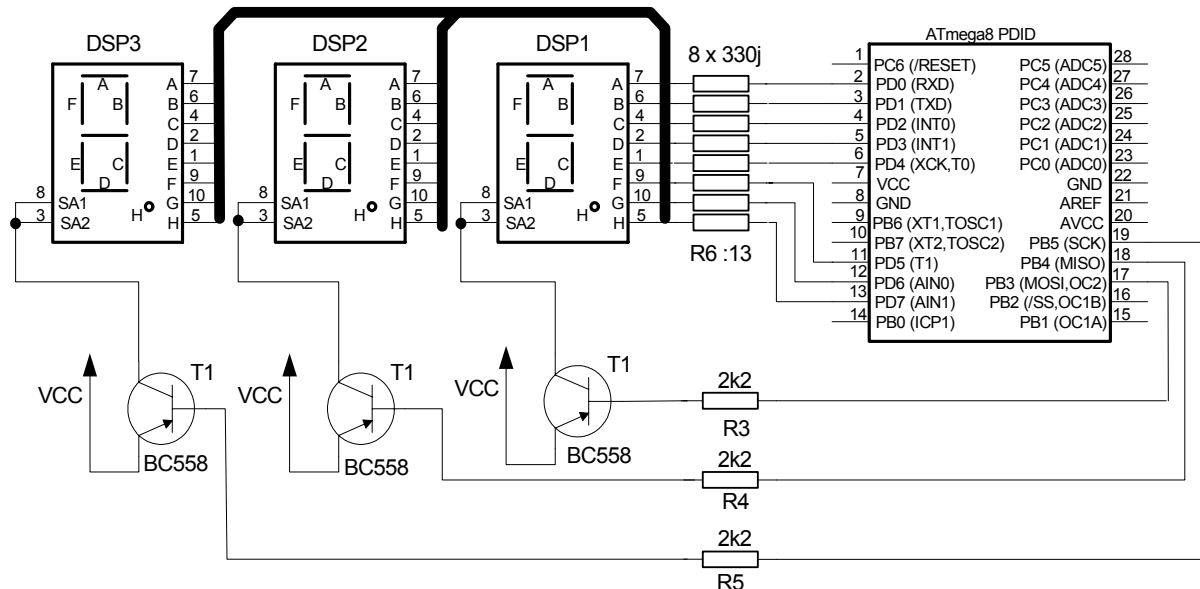
Do registra *r20* zapíšeme hodnotu „1“ a voláme podprogram ZOBRAZ. Podprogram ZOBRAZ zapíše na PORTD takú kombináciu núl a jednotiek, aby zodpovedala hodnote uloženej v dolných štyroch bitoch registra *r20*. Činnosť podprogramu ZOBRAZ je nasledovná:

1. Do registra *r21* sa uloží hodnota zodpovedajúca symbolickému menu *nula*.
2. Maskujú sa horné štyri bity registra *r20*, pretože na základe zadania nás zaujímajú len obsah dolných štyroch bitov registra *r20*.
3. Ak výsledok predchádzajúcej operácie bol rovný nule, t.j. ak obsah dolných štyroch bitov registra je rovný nule pokračujeme na návěstí *Z1*. Inštrukcia na návěstí *Z1* zapíše na PORTD obsah registra *r21*. Poznamenajme, že v registri *r21* je hodnota zodpovedajúca symbolu *nula*. A inštrukciou *ret* sa vraciame do hlavného programu.
4. Ak výsledok predchádzajúcej operácie nebol rovný nule, potom sa do registra *r21* zapíše hodnota odpovedajúca symbolu *jedna*. Obsah registra *r21* sa dekrementuje (zníži o hodnotu jedna) a opätovne sa testuje na hodnotu nula. Na základe hodnoty v *r21* sa zobrazí hodnota *jedna* a podprogram končí, alebo sa pokračuje v podprograme v súlade s vyššie popísanými krokmi až do okamžiku kedy test registra *r21* na hodnotu nula bude pozitívny.

Poznamenajme, že čas na realizáciu podprogramu ZOBRAZ je závislý na zobrazovanej hodnote *r20[3:0]*. V prípade, že obsah *r20[3:0]* je 0xf, potom podprogram ZOBRAZ trvá 49 strojových cyklov. V prípade zobrazenia nuly trvá len 7 strojových cyklov.

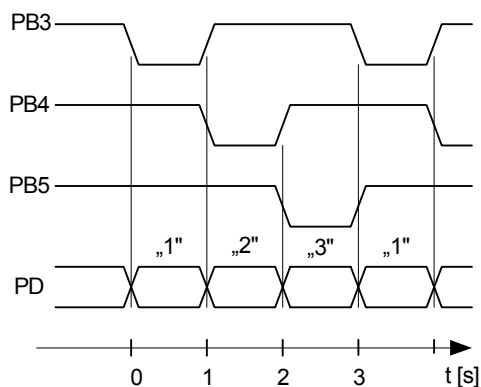
Realizácia podprogramu ZOBRAZ spôsobí, že na vývodoch V/V brány D je nastavená taká kombinácia núl a jednotiek, aby zodpovedala požadovanému symbolu. Aby sa príslušné diódy zobrazovacieho prvku DSP1 zasvietili je ešte potrebné priviesť na spoločnú anódu- SA napätie VCC. Toto sa realizuje výkonom inštrukcie, *cbi PORTB,3*. Teraz sa na prvku DSP1 zasvieti symbol zodpovedajúci jednotke a program pokračuje nekonečnou slučkou.

Upravme predchádzajúci príklad tak, aby na zobrazovacom prvku DSP1 bola prvú sekundu zobrazená hodnota „1“. Druhú sekundu zobražeme hodnotu dva na DSP2 a tretiu sekundu by bola na DSP3 zobrazená hodnota 3. Uvedený postup sa má opakovať. Zapojenie zobrazovacích prvkov je uvedené na obr.5.



Obr.5 Zapojenie zobrazovacích prvkov DSP1 až DSP3

Časový diagram prepínania hodnôt na vývodoch PB3 až PB5, ktoré ovládajú tranzistory T1, T2 a T3 je uvedený na obr.6.



Obr.6 Spínanie príslušných zobrazovacích prvkov DSP

Úprava predchádzajúceho programu je veľmi jednoduchá.

- Hlavný program je modifikovaný nasledovne:

```
ZOB: ldi    r20,0x01           ; chceme zobrazit "1"
      rcall ZOBRAZ             ; volanie podprogramu
```

```

cbi    PORTB,3           ; svieti DSP1
rcall  CAKAJ             ; 1s
sbi    PORTB,3           ; zhasne
ldi    r20,2             ; zobrazit' "2"
rcall  ZOBRAZ
cbi    PORTB,4           ; svieti DSP2
rcall  CAKAJ             ; 1s
sbi    PORTB,4           ; zhasne
ldi    r20,3             ; zobrazit' "3"
rcall  ZOBRAZ
cbi    PORTB,5           ; svieti DSP3
rcall  CAKAJ             ; 1s
sbi    PORTB,5           ; zhasne
rjmp   ZOB

```

- Pretože upravený hlavný program využíva podprogram „CAKAJ“ program „LED4“ je doplnený o podprogram „CAKAJ“.

;*****čakacia slučka cca $82 \cdot 255 \cdot 255 \cdot 3 \cdot 62.5 \text{ ns} = 0.999759 \text{ s}$

CAKAJ:

```

ldi    temp,82           ;reg.temp naplníme hodnotou 82
C3:    ldi    temp1,0xff   ;reg.temp1 naplníme hodnotou 0xff=255
C2:    ldi    temp2,0xff   ;reg.temp2 naplníme hodnotou 0xff
C1:    dec    temp2
      brne   C1
      dec    temp1
      brne   C2
      dec    temp
      brne   C3
      ret

```

Po úspešnom preklade a naprogramovaní obvodu môžeme sledovať, že na DSP1, DSP2 a DSP3 postupne zobrazujú hodnoty 1, 2 a 3.

Teraz sa pokúsime proces prepínania medzi jednotlivými zobrazovacími prvkami urýchliť. Skráťme čakaciu slučku približne štyridsaťnásobne, to je namiesto hodnoty 82 uložíme v slučke CAKAJ do registra temp hodnotu 2, (ldi temp,2).

Po naprogramovaní obvodu vidíme, že prepínanie medzi jednotlivými prvkami je podstatne rýchlejšie, ale stále vnímame isté blikanie segmentov na zobrazovacích prvkoch.

Pri ďalšom zvyšovaní rýchlosti prepínania (ldi temp,1, ldi temp1,0x80, ďalšie 4-násobné zvýšenie rýchlosti prepínania) je ťažké dosiahnuť takú rýchlosť prepínania segmentov, že nie sme schopní postrehnúť blikanie zobrazovacích prvkov. Pri uvedených hodnotách registrov temp a temp1 čakacia slučka bude trvať približne 1/164 s. V tomto prípade každý prvok svieti približne 6 ms a 12 ms je vypnutý. Frekvencia blikania prvkov je teraz približne 55 Hz. Je zrejmé, že frekvenciu spínania jednotlivých prvkov môžeme zvyšovať až do 1 MHz (bez použitia čakacej slučky), ale v praktických prípadoch nemá význam zvyšovať frekvenciu nad hodnotu 100Hz. Zvyšovanie frekvencie nad hodnotu 100Hz

nezlepšuje už vizuálny vnem, pričom zvyšuje zaťaženie procesora a generuje vyššiu úroveň rušenia. Uvedený program ukazuje jednu z možností, pomocou ktorej môžeme v časovom multiplexe zobrazovať obsahy viacerých registrov s využitím jednej 8-bitovej V/V brány, (PORTD).

Poznamenajme, že v uvedenom príklade bol strojový čas procesora neefektívne využitý na čkanie v slučke (cca $3 \cdot 128 \cdot 255 = 98\,000$ inštrukčných cyklov a len 14 až 20 cyklov bolo využitých na realizáciu hlavného programu a podprogramu *ZOBRAZ*). V praktických aplikáciách by sme zrejme na časovanie prepínania zobrazovacích prvkov využili interný časovač a prerušovací systém MCU čím by sme uvoľnili výpočtový výkon procesora pre spracovanie ďalších úloh. Naznačené riešenie bude uvedené v ďalších aplikačných príkladoch.

Zobrazovanie informácií na 7-segmentových zobrazovacích prvkoch má pomerne veľké obmedzenia. Tieto zobrazovacie prvky sú určené predovšetkým pre zobrazenie numerických údajov (číslíc 0 až 9), prípadne na zobrazenie údajov v hexa tvare (symboly 0 až f). V porovnaní s prvkami LCD majú vyššiu spotrebu. Z uvedených dôvodov sa vo viacerých aplikáciách stretáme so zobrazovacími jednotkami LCD. Vlastné zobrazovacie prvky LCD bývajú často doplnené o riadiacu časť (integrovaný radič, kontrolér), ktorá významne zjednodušuje použitie zobrazovacej jednotky v danej aplikácii.

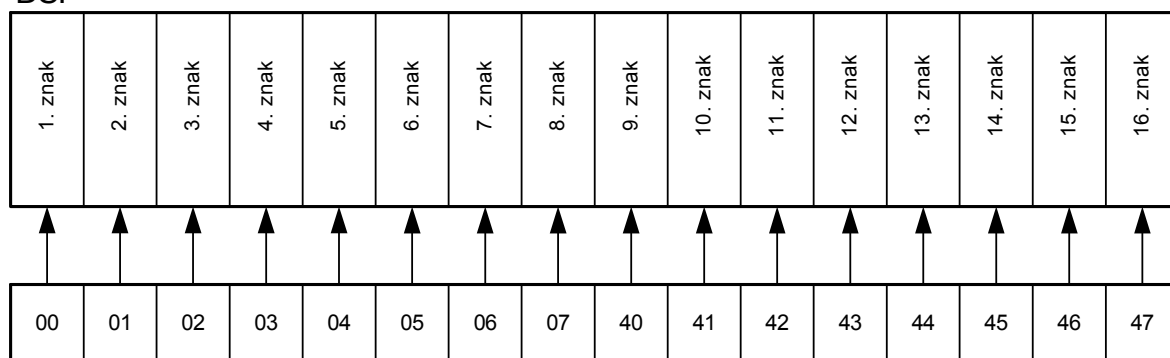
Aj modul ATmega8 je doplnený o jednoduchú zobrazovaciu jednotku na báze LCD prvku. V nasledujúcom príklade je uvedená komunikácia MCU s takouto zobrazovacou jednotkou.

Príklad 4

Navrhujeme jednoduchý program, ktorý umožní zobraziť ľubovoľný text na LCD jednotke, DEM 16101. Uvedená zobrazovacia jednotka LCD umožňuje zobraziť 16 alfanumerických znakov v jednom riadku. K aplikačnej doske sa pripája prostredníctvom 16-vývodového konektora (LCD konektor). Význam vývodov konektora LCD je uvedený v nasledujúcej tabuľke.

číslo vývodu	označenie	význam
1	Vss	Potenciál zeme (GND)
2	5V	Napájanie (5V)
3	VEE	Napájanie pre LCD
4	RS	Voľba dáta(H)/príkaz(L)
5	R/W	Voľba zápis/čítanie
6	E	Povolenie čítania/zápisu
7	DB0	Údaje, bit 0
8	DB1	Údaje, bit 1
9	DB2	Údaje, bit 2
10	DB3	Údaje, bit 3
11	DB4	Údaje, bit 4
12	DB5	Údaje, bit 5
13	DB6	Údaje, bit 6
14	DB7	Údaje, bit 7
15	NC1	Nevyužitý, pripravený pre podsvietenie
16	NC2	Nevyužitý, pripravený pre podsvietenie

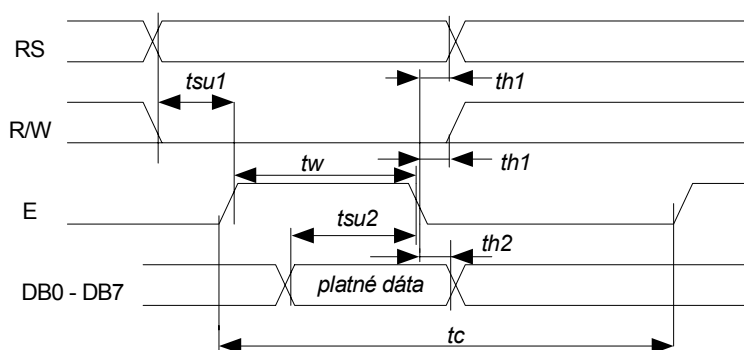
DSP



ADRESA RAM

Obr.8 Vzťah medzi pamäťou údajov LCD modulu a pozíciou znaku

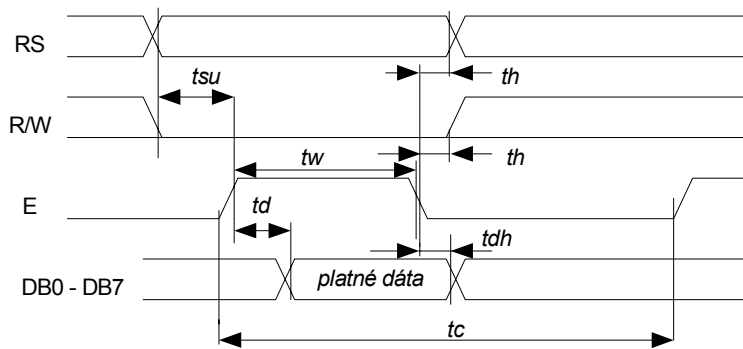
Časové priebehy riadiacich a údajových signálov v procese zápisu dát/príkazov do LCD modulu sú uvedené na obr.9.



<i>tc</i>	zápisový cyklus	<i>min</i>	500ns
<i>tw</i>	povoľovací signál	<i>min</i>	230ns
<i>tsu</i>	predstih RS a R/W pred E	<i>min</i>	40ns
<i>tsu1</i>	predstih RS a R/W pred E	<i>min</i>	40 ns
<i>th1</i>	čas oneskorenia	<i>min</i>	10ns
<i>tsu2</i>	setup dáta	<i>min</i>	80 ns
<i>th2</i>	presah dát	<i>min</i>	10 ns

Obr.9 Zápis dát

Proces čítania dát z LCD modulu DEM16101 je charakterizovaný časovými priebehmi uvedenými na obr.10.



<i>tc</i>	cyklus čítania	min	500ns
<i>tw</i>	povoľovací impulz	min	230ns
<i>tsu</i>	predstih RS a R/W pred E	min	40ns
<i>td</i>	oneskorenie dát	max	40 ns
<i>th</i>	čas oneskorenia	min	10ns

Obr.10 Čítanie dát

Pri zápise údajov do modulu LCD hodnota bitu RS - „L“ definuje príkaz a hodnota „H“ určuje že sa jedná o zápis dát. S rešpektovaním časového diagramu na obr.x je možné naprogramovať podprogramy pre zápis inštrukcií - *INST* a dát - *DATA*.
Poznamenajme, že zapisovaný údaj je uložený v registri *temp*.

```

INST:      push    temp1           ; uchovanie povodneho obsahu
            cbi     PORTB,5         ; RS = L zapis instr.
            cbi     PORTB,3         ; R/W = L
            ser     temp1
            out     DDRD,temp1      ; smer portD vystup
            out     PORTD,temp      ; obsah v temp
            sbi     PORTB,4         ; E LCD
            nop
            nop
            nop                     ; 4x62.5 = 250ns
            cbi     PORTB,4         ; strobovanie zapisu
            clr     temp1
            out     DDRD,temp1      ; PA vstup
            rcall   CAK3
            pop     temp1
            ret

```

Podprogram pre zápis dát je nasledovný:

```

DATA:     push    temp1
            sbi     PORTB,5         ; RS=H zapis dat
            cbi     PORTB,3         ; R/W = L
            ser     temp1
            out     DDRD,temp1      ; smer D, vystup
            out     PORTD,temp      ; obsah v temp

```

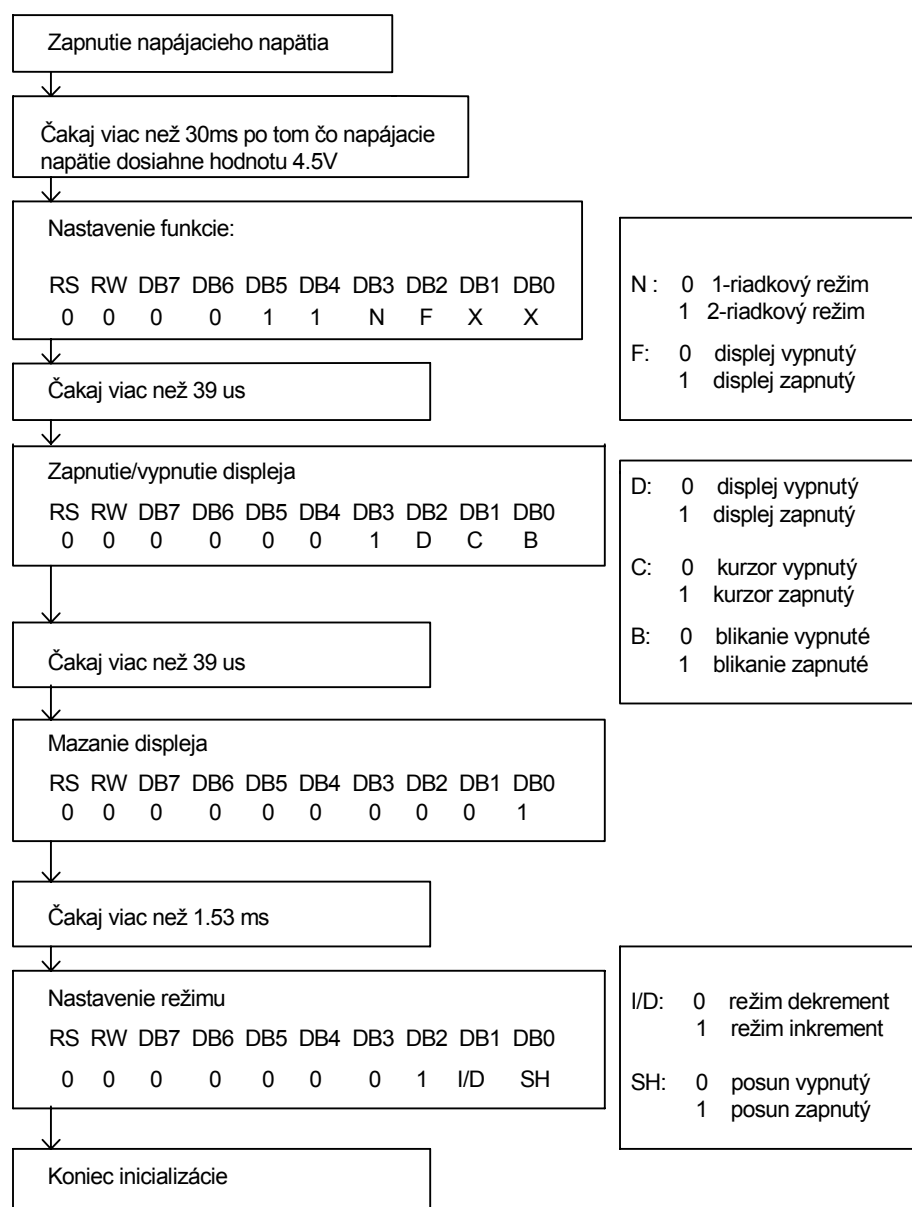
```

sbi    PORTB,4           ;E LCD
nop
nop
nop                               ;4x62.5 = 250ns
cbi    PORTB,4           ;strob zapisu
clr    temp1
out    DDRD,temp1        ;portD vstup
rcall  CAK3
pop    temp1
ret

```

Je zrejmé, že obidva podprogramy sú veľmi podobné. Jediný rozdiel je len v nastavení bitu *PORTB,5* – signál *RS*.

Pred vlastným zápisom dát do pamäte LCD modulu je potrebné modul inicializovať. Inicializácia pozostáva z nasledovnej postupnosti príkazov:



Obr. 11 Inicializácia LCD modulu

Ako príklad programovej obsluhy LCD modulu uveďme príklad, ktorý po pripojení napájacieho napätia zobrazí text „Vita Vas MCU AVR“.

```
;program „LCD.asm“ zapis textu na LCD DSP 16101
.include "m8def.inc"
```

```
.def    temp=r16
.def    temp1=r17
.def    temp2=r18
.def    temp3=r19
.cseg           ;segment programu
.org    0x0      ;ulozit od adresy 0

rjmp    RESET
reti           ;rjmp EXT_INT0    ;IRQ handler
reti           ;rjmp EXT_INT1
reti           ;rjmp TIM2_COMP
reti           ;jmp  TIM2_OVF
reti           ;rjmp TIM1_CAPT
reti           ;rjmp TIM1_COMPA
reti           ;rjmp TIM1_COMPB
reti           ;rjmp TIM1_OVF
reti           ;rjmp TIM0_OVF
reti           ;rjmp SPI_STC
reti           ;rjmp USART_RXC
reti           ;rjmp USART_UDRE
reti           ;rjmp USART_TXC
reti           ;rjmp ADCC
reti           ;rjmp EE_RDY
reti           ;rjmp ANA_COMP
reti           ;rjmp TWSI
reti           ;rjmp SPM_RDY
```

RESET:

```
ldi     temp,high(RAMEND) ;Nastavenie ukazovatela zasobnika
out     SPH,temp         ;na koniec pamate RAM
ldi     temp,low(RAMEND)
out     SPL,temp
***** Nastavenie portov*****
ldi     temp,0xff
out     DDRD,temp        ;PortD vystup
out     DDRB,temp        ;PortB vystup
clr     temp
out     DDRC,temp        ;Port C vstup
ser     temp
out     PORTD,temp       ;PortD do H
clr     temp
```

```

out    PORTB,temp          ;PortB do L

,*****
,***** D S P - INICIALIZACIA*****
    ldi    temp,3           ;cak slucka 36ms
    rcall  CAKAJ            ;min 30ms po nábehu Vcc
    ldi    temp,0x38        ;nastavenie funkcie DSP
    rcall  INST             ;vyslanie inštrukcie
    rcall  CAK3             ;0.46 ms
    ldi    temp,0x0c        ;riadenie on/off DSP a kurzora
    rcall  INST             ;vyslanie inštrukcie
    rcall  CAK3
    ldi    temp,0x01        ;mazanie DSP
    rcall  INST
    rcall  CAK3             ;cakaj 1.5 ms
    rcall  CAK3
    rcall  CAK3
    ldi    temp,0x06        ;nastavenie režimu DSP 06
    rcall  INST
,*****zapís textu na DSP *****

IV:    ldi    temp,0x80      ;adresa prvého zobraz. znaku, kurzor na začiatok
    rcall  INST             ;vyslanie inštrukcie
    ldi    ZH,high(TABLE<<1);nastav Z-pointer (r31:r30)
    ldi    ZL,low(TABLE<<1)
    ldi    temp3,4          ;8 znakov
II:    lpm    temp,Z+
    rcall  DATA
    lpm    temp,Z+
    rcall  DATA
    dec    temp3
    brne   II
    ldi    temp,0xc0        ;adresa 9.znaku to je zvláštnosť DSP
    rcall  INST
    ldi    temp3,4          ;zobraz 8 znakov
III:   lpm    temp,Z+
    rcall  DATA
    lpm    temp,Z+
    rcall  DATA
    dec    temp3
    brne   III
KON:   rjmp   KON
,*****koniec zapísu dat na DSP *****

,*****PODPROGAMY*****

CAKAJ:
    push   temp1
    push   temp2

```

```

C5:  ldi    temp1,0xff    ;caka 255x255x3x62.5ns x obsah temp
C3:  ldi    temp2,0xff    ;XTAL 16 MHz . . .st. cyklus 62.5ns
C4:  dec    temp2          ;to je cca 12.2ms x obsah temp
      brne  C4
      dec    temp1
      brne  C3
      dec    temp
      brne  C5
      pop   temp2
      pop   temp1
      ret

```

,*****

```

CAK3: push  temp1
      push  temp2

      ldi    temp1,0x96    ;255x96x3x62.5 ns
C6:  ldi    temp2,0xff    ; t.j. cca 0.46 ms
C7:  dec    temp2
      brne  C7
      dec    temp1
      brne  C6
      pop   temp2
      pop   temp1
      ret

```

,***** zápis inštrukcie do DSP

```

INST: push  temp1
      cbi    PORTB,5          ; RS = L zapis instr.
      cbi    PORTB,3          ; R/W = L
      ser    temp1
      out    DDRD,temp1      ; smer portD vystup
      out    PORTD,temp      ; obsah v temp
      sbi    PORTB,4          ; E LCD
      nop
      nop
      nop                    ; 4x62.5 = 250ns
      cbi    PORTB,4          ; strobovanie zapisu
      clr    temp1
      out    DDRD,temp1      ; PA vstup
      rcall  CAK3
      pop   temp1
      ret

```

,***** zápis dát do DSP

```

DATA:
      push  temp1

      sbi    PORTB,5          ; RS=H zapis dat
      cbi    PORTB,3          ; R/W = L
      ser    temp1
      out    DDRD,temp1      ; smer D, vystup

```

```

    out    PORTD,temp        ;obsah v temp
    sbi    PORTB,4           ;E LCD
    nop
    nop
    nop                      ;4x62.5 = 250ns
    cbi    PORTB,4           ;strob zapisu
    clr    temp1
    out    DDRD,temp1        ;portD vstup
    rcall  CAK3
    pop    temp1
    ret
;*****od adresy 200 v pamäti programu je ulozena tab. s textom
.org      200
TABLE:
.db       "Vita Vas MCU AVR"

```

Uvedený príklad je určený na ilustráciu programovej obsluhy LCD zobrazovacej jednotky DEM16101. Zobrazuje 16-znakov textu uloženého v pamäti programu od adresy 200.

Jednotlivé segmenty uvedeného programu (podprogramy *INST*, *DATA*, *CAKAJ*, *CAK3*, prípadne inicializáciu LCD) je možné využiť v ďalších programoch, v ktorých budeme na module LCD zobrazovať vybranú informáciu. Z tohto dôvodu je vhodné časť programu pre inicializáciu LCD prepísať ako podprogram s názvom *INIT* a spolu s ďalšími podprogramami (*DATA*, *INST*, *CAKAJ*, *CAK3*) uložiť s názvom napríklad *LCD*. Pri tvorbe ďalších aplikácií bude potom možné ich prilinkovať k ľubovoľnému aplikačnému programu. Aplikačný program môže využívať (volať) všetky prilinkované podprogramy. Upozorníme, že aplikačný program nesmie používať tie isté označenia návští, ktoré sú využité v prilinkovaných podprogramoch (napríklad *C3*, *C4*, *C5*, *C6*, *C7*).

Ako príklad súboru, ktorý obsahuje podprogramy na obsluhu LCD modulu uveďme program *LCDX.asm*.

```

; súbor LCDX.asm
;LCD utility pre LCD DSP 16101
;v uvedených podprogramoch nie su vyuzivane
;symbolicke mena registrov okrem registra temp,
;ktory sluzi na komunikaciu s aplikacnym programom
;prostrednictvom registra temp aplikacia komunikuje s podprogramami
;obsahy dalsich pouzivanych registrov su nezmenene
;UPOZORNENIE aplikacny program nesmie priradit registrom r17, alebo r18 meno temp

; inicializacia modulu DEM16101

```

```

INI:  push    temp
      ldi     temp,3           ;cak slucka 30ms
      rcall  CAKAJ            ;min 30ms po nábehu Vcc
      ldi     temp,0x38       ;nastavenie funkcie DSP
      rcall  INST             ;vyslanie inštrukcie
      rcall  CAK3             ;0.46 ms
      ldi     temp,0x0c       ;riadenie on/off DSP a kurzora
      rcall  INST             ;vyslanie inštrukcie

```

```

rcall  CAK3
ldi    temp,0x01          ;mazanie DSP
rcall  INST
rcall  CAK3                ;cakaj 1.5 ms
rcall  CAK3
rcall  CAK3
ldi    temp,0x06          ;nastavenie režimu DSP 06
rcall  INST
pop    temp
ret

```

;******Zapis prikazu (instrukcie) ******

```

INST: push  r17
      cbi   PORTB,5          ; RS = L zapis instr.
      cbi   PORTB,3          ; R/W = L
      ser   r17
      out   DDRD,r17         ; smer portD vystup
      out   PORTD,temp       ; obsah v temp
      sbi   PORTB,4          ; E LCD
      nop
      nop
      nop                   ; 4x62.5 = 250ns
      cbi   PORTB,4          ; strobovanie zapisu
      clr   r17
      out   DDRD,r17         ; PA vstup
      rcall CAK3
      pop   r17
      ret

```

;******zapis dát do DSP*

```

DATA: push  r17

      sbi   PORTB,5          ; RS=H zapis dat
      cbi   PORTB,3          ; R/W = L
      ser   r17
      out   DDRD,r16         ; smer D, vystup
      out   PORTD,temp       ; obsah v temp
      sbi   PORTB,4          ; E LCD
      nop
      nop
      nop                   ; 4x62.5 = 250ns
      cbi   PORTB,4          ; strob zapisu
      clr   r17
      out   DDRD,r17         ; portD vstup
      rcall CAK3
      pop   r17
      ret

```

;******cakacia slucka*

```

CAKAJ:
      push  r17
      push  r18

```

```

C5:  ldi    r17,0xff      ;caka 255x255x3x62.5ns x obsah temp
C3:  ldi    r18,0xff      ;XTAL 16 MHz . . .st. cyklus 62.5ns
C4:  dec    r18            ;to je cca 12.2ms x obsah temp
      brne  C4
      dec    r17
      brne  C3
      dec    temp
      brne  C5
      pop    r18
      pop    r17
      ret
;*****cakacia slucka
CAK3: push  r17
      push  r18
      ldi    r17,0x96      ;255x96x3x62.5 ns
C6:  ldi    r18,0xff      ; t.j. cca 0.46 ms
C7:  dec    r18
      brne  C7
      dec    r17
      brne  C6
      pop    r18
      pop    r17
      ret

```

Predchádzajúci súbor uložíme pod názvom *LCDX.asm*. Za účelom zjednodušenia a skrátenia zápisu aplikačného programu vytvoríme aj súbor s názvom *PRER8.asm*, ktorý obsahuje tabuľku vektorov prerušení.

```

; súbor PRER.asm
.cseg
.org 0x0
rjmp RESET
reti          ;rjmp EXT_INT0    ;IRQ handler
reti          ;rjmp EXT_INT1
reti          ;rjmp TIM2_COMP
reti          ;jmp  TIM2_OVF
reti          ;rjmp TIM1_CAPT
reti          ;rjmp TIM1_COMPA
reti          ;rjmp TIM1_COMPB
reti          ;rjmp TIM1_OVF
reti          ;rjmp TIM0_OVF
reti          ;rjmp SPI_STC
reti          ;rjmp USART_RXC
reti          ;rjmp USART_UDRE
reti          ;rjmp USART_TXC
reti          ;rjmp ADCC
reti          ;rjmp EE_RDY
reti          ;rjmp ANA_COMP
reti          ;rjmp TWSI
reti          ;rjmp SPM_RDY

```

Ak sme vytvorili a uložili súbory *PRER8.asm* a *LCDX.asm*, potom predchádzajúci program na zápis textu na LCD modul môže byť v nasledovnom tvare.

;program LCD2.asm príklad zápis textu na LCD DSP 16101

.include "m8def.inc"

.def temp=r16

.def temp1=r17

.def temp2=r18

.def temp3=r19

.include "PRER8.asm" ; prílinkovanie suboru PRER8.asm

RESET:

ldi temp,high(RAMEND) ;Nastavenie ukazovatela zasobníka

out SPH,temp ;na koniec pamate RAM

ldi temp,low(RAMEND)

out SPL,temp

****** Nastavenie portov******

ldi temp,0xff

out DDRD,temp ;PortD vystup

out DDRB,temp ;PortB vystup

clr temp

out DDRC,temp ;Port C vstup

ser temp

out PORTD,temp ;PortD do H

clr temp

out PORTB,temp ;PortB do L

rcall INIT

IV: ldi temp,0x80 ;adresa prvého zobraz.znaku kurzor na začiatok

rcall INST ;vyslanie inštrukcie

ldi ZH,high(TABLE<<1);nastav Z-pointer (r31:r30)

ldi ZL,low(TABLE<<1)

ldi temp3,4 ;8 znakov

II: lpm temp,Z+

rcall DATA

lpm temp,Z+

rcall DATA

dec temp3

brne II

ldi temp,0xc0 ;adresa 9.znaku to je zvláštnosť DSP

rcall INST

ldi temp3,4 ;zobraz 8 znakov

III: lpm temp,Z+

```
rcall DATA
lpm temp,Z+
rcall DATA
dec temp3
brne III
```

```
KON: rjmp KON
;*****koniec zapisu dat na DSP *****
```

```
.include "LCDX.asm" ; prilinkovanie suboru LCDX.asm, ktory obsahuje
; všetky potrebné podprogramy
TABLE:.db "**** MCU AVR ****" ; text bude nasledovať bezprostredne po podprogramoch
```

Je zrejmé, že s využitím už odladených častí programu (*PRER8.asm*) a podprogramov (*LCDX.asm*) aplikačný program *LCD2.asm* napíšeme podstatne rýchlejšie, ako jeho predchádzajúcu verziu „*LCD.asm*“.

2. Prerušovací podsystem

Prerušovací podsystem umožňuje efektívnu obsluhu asynchrónnych udalostí, na ktoré musí procesor reagovať. V prípade výskytu definovanej udalosti procesor preruší beh programu a venuje sa programovej obsluhu udalosti, ktorá dané prerušenie vyvolala. Po skončení tejto programovej obsluhy procesor pokračuje vo výkone hlavného programu. Je zrejmé, že v tom istom okamžiku (t.j. v jednom strojovom cykle) sa môžu vyskytnúť viaceré žiadosti o prerušenie – od rôznych zdrojov. Aby bolo možné rozhodnúť, ktorá žiadosť o prerušenie bude obslužená ako prvá je potrebné definovať akési poradie dôležitosti – priority prerušení. Na základe priority prerušení sa určí postupnosť spracovania žiadostí o prerušenie, ktoré sa vyskytli v tom istom strojovom cykle. Pripomeňme, že MCU mega8 má definovaných 19 zdrojov prerušení. Tabuľka zdrojov prerušení je uvedená na str.46. Priorita je daná pozíciou v tabuľke prerušení. (RESET má najvyššiu prioritu a SPM_RDY najnižšiu.) Pripomeňme, že nastavením riadiaceho bitu prerušenia, bit I v S- registri povolíme prerušovací systém. Vynulovaním I-bitu (*cli*) zakážeme procesoru reagovať na žiadosti o prerušenie. Jednotlivé zdroje prerušení môžeme individuálne povoliť/zakázať nastavením/nulovaním príslušných maskovacích bitov.

Popíšme správanie sa procesora pri výskyte žiadosti o prerušenie napríklad od zdroja INT0, (v module ATmega8 žiadosť INT0 odpovedá stlačeniu tlačidla TL2).

1. Nech sa po reštarte obvodu programový čítač naplní hodnotou 0x0000. Poznamenajme, že programovaním FUSE-bitu BOOTRST môžeme túto hodnotu zmeniť na adresu začiatku zavádzacej sekcie. V ďalšom kroku procesor pokračuje spracovávaním inštrukcie na adrese danej návěstím RESET, (adr. 19).
2. Inštrukcie na adresách 19 až 22 nastaví ukazovateľ zásobníka na koniec pamäte SRAM (0x045F).
3. Nastavením obsahu registra MUCRC definujeme udalosť, ktorá vyvolá prerušenie, (0...úroveň L).
4. Nastavením obsahu registra GICR povolíme obsluhu prerušenia od zdroja INT0.
5. Inštrukciou na adrese 27 – *sbi PORTD,2* sa zapne interný pull-up odpor.
6. Inštrukcia *sei* nastaví I-bit v S-registri, čím sa povolí prerušovací systém MCU.

7. Procesor pokračuje spracovaním nasledujúcich inštrukcií programu.

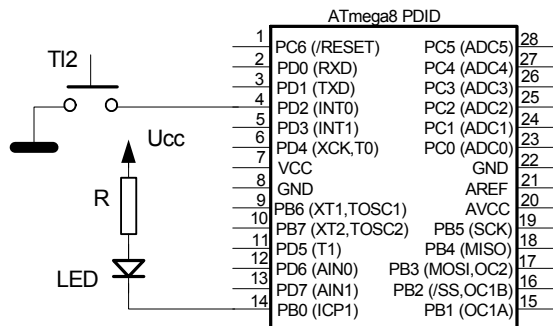
Adresa	Inštrukcia	Ukazovateľ zásobníka		Adr.	0x45C	0x45D	0x45E	0x45F
		SPH	SPL					
0	rjmp RESET	xx	xx	xx	xx	xx	xx	xx
1	rjmp INT0	xx	xx	xx	xx	0	51	
2	reti ;rjmp INT1	04	5D					
-	-							
-	-							
RESET: 19	ldi temp, hight(RAMEND)	xx	xx	xx	xx	xx	xx	xx
20	out SPH, temp	04	xx	xx	xx	xx	xx	xx
21	ldi temp, low(RAMEND)	04	xx	xx	xx	xx	xx	xx
22	out SPL, temp	04	5F	xx	xx	xx	xx	xx
23	ldi temp, 0b00000010	04	5F	xx	xx	xx	xx	xx
24	out MCUCR, temp	04	5F	xx	xx	xx	xx	xx
25	ldi temp, 0b01000000	04	5F	xx	xx	xx	xx	xx
26	out GICR, temp	04	5F	xx	xx	xx	xx	xx
27	sbi PORTD, 2	04	5F	xx	xx	xx	xx	xx
28	cli	04	5F	xx	xx	xx	xx	xx
-	-							
-	- prerušenie							
50	xxx ← INT0	04	5F	xx	xx	xx	xx	xx
51	xxx	04	5F	xx	xx	0	51	
100	xxx	04	5F	xx	xx	0	51	
INT0: 101	xxx	04	5D	xx	xx	0	51	
-	-							
-	-							
106	reti	04	5D	xx	xx	0	51	

- Nech v priebehu výkonu programu, pri realizácii inštrukcie na adrese 50 sa vyskytne žiadosť o prerušenie, INT0.
- Dokončí sa práve vykonávaná operácia, do zásobníka sa uloží adresa nasledujúcej inštrukcie (0 51) a obsah ukazovateľa zásobníka sa po každom prístupe do pamäte SRAM dekrementuje na hodnotu 0x045D. Programový čítač sa naplní adresou, ktorá odpovedá konkrétnej žiadosti o prerušenie - (adr.1).
- Výkon programu preto ďalej pokračuje na adrese 1, kde je uložená inštrukcia relatívneho skoku na adresu obslužného programu prerušenia – návstievie INT0 na adrese 101.
- Od adresy 101 je uložená obsluha prerušenia, ktorá končí inštrukciou návratu z prerušenia- *reti*.
- Realizáciou inštrukcie *reti* sa dekrementuje ukazovateľ zásobníka a programový čítač sa naplní hodnotou uloženou v horných dvoch bytoch zásobníka, adresou 051.
- Procesor ďalej pokračuje spracovaním inštrukcií od adresy 51.

Príklad 2.1

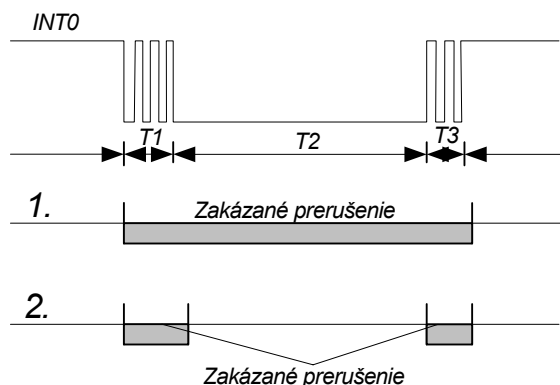
Navrhujeme program, ktorý pri stlačení tlačidla TL2 zasvieti LED diódu, ak bola zhasnutá, alebo zhasne LED, ak bola zasvietená. Na riešenie danej úlohy sa v prvom kroku pokúsime využiť prerušovací systém MCU.

Tlačidlo TL2 a LED dióda sú zapojené podľa obr.12. V demonštračnom module je vývod PD2/INT2 súčasne využitý aj na ovládanie 7-segmentových LED zobrazovacích prvkov, preto pri stlačení tlačidla TL2 sa v prípade, ak vývody PB3 až PB5 majú úroveň H rozsvietia segmenty c zobrazovačov DSP. V uvedenom príklade sú výstupy PB3 až PB5 v úrovni H, preto popísanú udalosť nespozorujeme.



Obr.12. Zapojenie tlačidla TL2 a svetelnej diódy

Poznamenajme, že pri stlačení tlačidla TL2 sa generuje veľký počet impulzov, ktoré môžu byť nesprávne interpretované ako ďalšie stlačenia tlačidla TL2, obr.13.



Obr.13 Stlačenie tlačidla TL2

Pri stlačení tlačidla TL2, v čase T1 a T3 sa generuje veľký počet náhodných impulzov, ktoré sú procesorom nesprávne interpretované, ako séria nezávislých stlačení. Z uvedeného dôvodu je potrebné programovými prostriedkami znázornený jav eliminovať.

Jeden z možných spôsobov je zakázať obsluhu prerušenia INT0 po príchode prvého impulzu na čas dlhší než je predpokladané stlačenie tlačidla, (čas T1+T2+T3). Programové riešenie je dokumentované nasledovným výpisom programu.

```

;program "tlacidlo.asm"
;pri stlaceni tlacidla TI2 sa zmeni stav LED diody, svieti/nesvieti
;vyuzitie prerusenia INT0

.include "m8def.inc"

.def    temp=r16
.def    temp1=r17
.def    temp2=r18

.cseg                      ;segment programu
.org    0x0                 ;ulozit od adresy 0

        rjmp    RESET
        rjmp    EXT_INT0    ;      IRQ handler tlacidla TI2
        reti    ;          rjmp    EXT_INT1
        reti    ;          rjmp    TIM2_COMP
        reti    ;          rjmp    TIM2_OVF
        reti    ;          rjmp    TIM1_CAPT
        reti    ;          rjmp    TIM1_COMPA
        reti    ;          rjmp    TIM1_COMPB
        reti    ;          rjmp    TIM1_OVF
        reti    ;          rjmp    TIM0_OVF
        reti    ;          rjmp    SPI_STC
        reti    ;          rjmp    USART_RXC
        reti    ;          rjmp    USART_UDRE
        reti    ;          rjmp    USART_TXC
        reti    ;          rjmp    ADCC
        reti    ;          rjmp    EE_RDY
        reti    ;          rjmp    ANA_COMP
        reti    ;          rjmp    TWSI
        reti    ;          rjmp    SPM_RDY

RESET:
        ldi      temp,high(RAMEND) ;Nastavenie ukazovateľa zásobníka
        out      SPH,temp          ;na koniec pamäte RAM
        ldi      temp,low(RAMEND)
        out      SPL,temp

;***** Nastavenie portov*****
        clr      temp
        out      DDRD,temp        ;PortD vstup
        ser      temp
        out      DDRB,temp        ;PortB vystup
        out      PORTB,temp       ;PortB do H
        out      PORTD,temp       ;zapnute pull-up odpory
;***** Nastavenie režimu a povolenie INT0
        ldi      temp,0x0         ;prerušenie INT0 na úroveň L
        out      MCUCR,temp
        ldi      temp,0x40        ;povolenie prer.INT0
        out      GICR,temp
        sei      ;Globalne povolenie prerusení

KON:    rjmp     KON              ;MCU čaka na výskyt prerusení

;*****obsluha prerusení INT0 *****

```

EXT_INT0:

```

        in      temp,SREG      ;uchovanie obsahu SREG
        push   temp
        sbis    PORTB,0        ;preskoc ak PORTB,0 je nastaveny,H
        rjmp    NASTAV
        cbi     PORTB,0        ; LED svieti
        rjmp    KONINT
NASTAV:  sbi     PORTB,0        ; LED zhasne

```

KONINT:

```

        ldi     temp2,0x20      ; cakacia slucka (0.25s), koja
E3:      ldi     temp,0xff       ; trva dlhsie nez stlacenie
E1:      ldi     temp1,0xff      ; tlacidla tl2
E2:      dec     temp1
        brne    E2
        dec     temp
        brne    E1
        dec     temp2
        brne    E3
        pop     temp
        out     SREG,temp       ;navrat obsahu SREG
        reti

```

Nastavenie registra MCUCR definuje udalosť, ktorá vyvolá prerušenie. Prerušenie je vyvolané v prípade, ak na vstupe INT0 sa objaví napätie odpovedajúce úrovni L. Nastavenie registra GICR povoľuje akceptovanie prerušenia od externého zdroja INT0. Pretože tlačidlo T12, pripojené na vývod INT0, pri stlačení pripojí INT0 k potenciálu GND bude sa generovať žiadosť o prerušenie. Pri pustení tlačidla sa vďaka zopnutému internému pull-up odporu na vstupe INT0 objaví napätie odpovedajúce úrovni H.

V uvedenom programe je čakacia slučka súčasťou obsluhy prerušenia *EXT_INT0*. Poznamenajme, že v prípade ak tlačidlo T12 bude stlačené dlhšie než doba čakacej slučky (v uvedenom príklade cca. 0.25s) bude po skončení obsluhy prvého prerušenia akceptované ďalšie prerušenie a bude sa vykonávať jeho obsluha. Táto skutočnosť sa v niektorých aplikáciách využíva na automatické zvyšovanie počtu stlačení pri držaní klávesy.

Uvedené riešenie nie je z hľadiska efektívneho využívania výpočtového výkonu MCU vhodné. Veľký výpočtový výkon je obetovaný na neefektívne čakanie na pustenie tlačidla. Je zrejmé, že v diskutovanom príklade je možné stlačením tlačidla úplne zablokovat' činnosť MCU.

Ďalšie možné riešenie eliminácie vplyvu prechodových javov pri stačení/pustení tlačidla vychádza z myšlienky zakázať obsluhu prerušenia INT0 len na čas pokiaľ neskončí prechodový proces (T1, príp. T3). V tomto prípade je vhodné aby udalosť vyvolávajúca prerušenie bola ľubovoľná zmena na vstupe INT0 (H do L, alebo L do H). Potom nastavenie registra MCUCR bude 0x01. Je zrejmé, že v tomto prípade bude vyvolané prerušenie pri stlačení aj pri pustení tlačidla TL2. Preto je potrebné aby procesor bol schopný zistiť či sa jedná o zatlačenie, alebo pustenie tlačidla. Jeden z možných prístupov je ilustrovaný nasledujúcim výpisom obsluhy prerušenia *EXT_INT0*.

EXT_INT0:

```

        in      temp,SREG      ;uchovanie obsahu SREG
        push   temp

```

```

E1:      ldi    temp,0x40          ; cakacia slucka, ktora trva
E2:      ldi    temp1,0xff        ; dlhsie nez prechodovy dej pri
      dec     temp1              ; stlaceni/pusteni tlacidla tl2
      brne    E2                 ; zavisí od vlastnosti TL2
      dec     temp                ; v príklade cca 2ms
      brne    E1

      sbic    PIND,2              ; preskoc ak INT0 je L (TL2 je stlacene)
      rjmp    KONINT             ; koniec ak INT0 je H

      sbis    PORTB,0             ; preskoc ak PORTB,0 je nastaveny,H
      rjmp    NASTAV
      cbi     PORTB,0             ; LED svieti
      rjmp    KONINT
NASTAV:  sbi     PORTB,0           ; LED zhasne

KONINT:
      pop     temp
      out     SREG,temp           ; navrat obsahu SREG
      reti

```

V obsluhu prerušenia sme uchovávali obsah stavového registra v zásobníku (*in temp,SREG push temp*). Pred návratom z prerušenia sme pôvodný obsah stavového registra vybrali zo zásobníka (*pop temp*) a uložili späť do registra stavu (*out SREG,temp*). V uvádzaných demonštračných príkladoch by programy pracovali korektne aj bez uchovania obsahu registra stavu, pretože hlavný program - slučka *KON: rjmp KON*, nevyužíva obsah stavového registra. Vo väčšine aplikácií však hlavný program vykonáva operácie v súvislosti s riešením definovaných úloh, pričom modifikácia obsahu stavového slova v priebehu obsluhy prerušenia by mohla spôsobiť nekorektný beh programu.

Poznamenajme, že je možné nájsť mnoho ďalších spôsobov eliminácie zákmitov tlačidla TL2. Uvedené príklady slúžia len ako ukážka dvoch možných riešení a nie vždy optimálnych riešení.

S využitím prerušovacieho systému MCU sa ešte stretneme v ďalšom popise aplikácií interných modulov MCU.

3. Univerzálny synchrónny/asynchrónny vysielač/prijímač USART

Modul ASART integrovaný v obvode MCU súži na sériovú komunikáciu medzi MCU a okolím. Jej základné charakteristiky sú nasledovné:

- Plne obojsmerná činnosť,
- asynchrónny, alebo synchrónny komunikačný režim,
- generátor prenosovej rýchlosti s vysokou rozlišovacou schopnosťou,
- synchrónna činnosť v režimoch Master, alebo Slave,
- podporuje prenos dát dĺžky 5, 6, 7, 8, alebo 9 dátových bitov a 1, alebo 2 stop bity,
- generátor párnej, alebo nepárnej parity a kontrola paritného bitu,
- detekcia pretečenia dát,
- detekcia chyby prenosu,

filtrácia šumu,
tri oddelené prerušenia, (Tx kompletne, Tx dátový register prázdny a Rx kompletne),
režim multiprocessorovej komunikácie,
asynchrónny komunikačný režim s dvojnásobnou rýchlosťou.

V aplikačných príkladoch uvedieme len základné možnosti jednotky USART. Pre nastavenie Sériový komunikačný modul USART využíva nasledovné údajové, stavové a riadiace registre:

V/V údajový register, UDR - USART I/O Data Register

UDR:

Bit	7	6	5	4	3	2	1	0
Symbol	RXB7	RXB6	RXB5	RXB4	RXB3	RXB2	RXB1	RXB0
Symbol	TXB7	TXB6	TXB5	TXB4	TXB3	TXB2	TXB1	TXB0
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
p. hod.	0	0	0	0	0	0	0	0

Prijímací i vysielač vyrovnávací register modulu USART zdieľajú rovnakú V/V adresu symbolicky označenú ako UDR. Zapisované údaje do UDR sa zapíšu do vysielačieho vyrovnávacieho registra TXB. Pri čítaní UDR sa prečíta obsah prijímacieho vyrovnávacieho registra RXB. Pri prenose 5, 6, alebo 7 bitových znakov sa pri čítaní horné bity nastaví na hodnotu log. 0. Pri vysielaní sa horné bity ignorujú.

Vysielač vyrovnávací register môže byť prepisovaný len ak je UDRE príznak v UCSRA registri nastavený na hodnotu log.1. V prípade, že UDRE príznak má hodnotu log.0

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

UCSRA:

Bit	7	6	5	4	3	2	1	0
Symbol	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
Prístup	R	R/W	R	R	R	R	R/W	R/W
P.hod.	0	0	1	0	0	0	0	0

Bit 7 – RXC: USART Receive Complete, Ukončený príjem znaku

Bit 6 – TXC: USART Transmit Complete, ukončené vysielačie znaku

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

Bit 4 – FE: Frame Error, chyba rámca

Bit 3 – DOR: Data OverRun, pretečenie dát

Bit 2 – PE: Parity Error, chyba parity

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

Bit 0 – MPCM: Multi-processor Communication Mode

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

UCSRB:

Bit	7	6	5	4	3	2	1	0
Symbol	RXCIE	TXCIE	UDREI	RXEN	TXEN	UCSZ2	RXB8	TXB8

Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
Poč. hodnota	0	0	0	0	0	0	0	0

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

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

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

Bit 4 – RXEN: Receiver Enable, povolenie činnosti prijímača

Bit 3 – TXEN: Transmitter Enable, povolenie činnosti vysieláča

Bit 2 – UCSZ2: Character Size, dĺžka znaku

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

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

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

UCSRC:

Bit	7	6	5	4	3	2	1	0
Symbol	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P.hod.	1	0	0	0	0	1	1	0

Register UCSRC zdieľa tú istú V/V lokáciu ako register UBRRH. Prístup k uvedeným registrom je popísaný v predchádzajúcich častiach.

Bit 7 – URSEL: Register Select, výber registra

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

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

UPM1	UPM0	Režim parity
0	0	Zakázaná
0	1	Nevyužitá
1	0	Povolená, párna parita
1	1	Povolená, nepárna parita

Tab.3 Nastavenie režimu generovania a kontroly parity

Bit 3 – USBS: Stop Bit Select, výber stop bitu

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.4 Nastavenie dĺžky vysielaného znaku

Bit 0 – UCPOL: Clock Polarity, polarita hodinových impulzov

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.5 Nastavenie bitu UCPOL

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

UBRR:

Bit	15	14	13	12	11	10	9	8
Symbol	URSEL	-	-	-	UBRR11	UBRR10	UBRR9	UBRR8
Prístup	R/W	R	R	R	R/W	R/W	R/W	R/W
P.hod.	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
Symbol	UBRR7	UBRR6	UBRR5	UBRR4	UBRR3	UBRR2	UBRR1	UBRR0
P.hod.	0	0	0	0	0	0	0	0

Bit 15 – URSEL: Register Select, výber registra

Bity 14:12 – nevyužité (rezervované) bity

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

12-bitový register UBRR obsahuje údaj o prenosovej rýchlosti jednotky USART.

Pren. Rýchlosť [b/s]	UBRR pri fosc 4 MHz	UBRR pri fosc 8 MHz	UBRR pri fosc 16 MHz
2400	103	207	416
4800	51	103	207
9600	25	51	103
14.4k	16	34	68
19.2k	12	25	51
28.8k	8	16	34
38.4k	6	12	25
57.6k	3	8	16

76.8k	2	6	12
115.2k	1	3	8
230.4k	0	1	3

Tab. 6 Nastavenie prenosovej rýchlosti modulu USART

Príklad 3.1

Vytvorme jednoduchú aplikáciu, pomocou ktorej vyšleme text „Pozdravuje Ta AVR“ na sériovú linku. Definujme základné parametre prenosu.

1. Asynchrónna komunikácia,
2. prenosová rýchlosť nech je 9600b/s,
3. dĺžka slova 8 bitov,
4. jeden stop bit,
5. bez parity,

Poznamenajme že, v prvom prípade nebudeme v priebehu vysielania využívať prerušovací systém. Uvoľnenie registra údajov, UDR budeme testovať pomocou programových prostriedkov.

V prvom kroku je potrebné nastaviť parametre prenosu tak, aby boli splnené podmienky zadania:

1. Nastavíme prenosovú rýchlosť:
Zapíšeme odpovedajúce hodnoty do registrov UBRH a UBRL. Podľa tabuľky x3 pri $f_{osc}=16\text{ MHz}$, hodnota zapísaná do registrov UBRR bude 103.
2. Povolíme vysielanie jednotky USART. Tretí bit registra UCSRB nastavíme na hodnotu log.1.
3. Definujeme formát vysielaných dát. V zmysle zadania 8 bitov, párna parita, jeden stop bit. Register UCSRC: bity UCSZ2=0, UCSZ1=1, UCSZ0=1, UPM1=1, UPM0=0, USBS=0.
4. Do registra UDR postupne zapisujeme jednotlivé vysielané údaje pričom kontrolujeme bit UDRE registra UCSRA, dovoľujúci korektný zápis nových dát.

Nasledovný výpis programu ilustruje uvedenú postupnosť krokov.

```
;program príklad 3.1 Seriova komunikacia  
; J Micek 10.4.2005
```

```
.include "m8def.inc"
```

```
.def    temp=r16  
.def    temp1=r17
```

```
.equ    PRENRL=103           ;definovanie prenosovej rýchlosti
```

```

.equ    PRENRH=0

.include "PRER8.asm"    ; prilinkovanie suboru PRER8.asm vektory prerusenja

RESET:
    ldi    temp,high(RAMEND)    ;Nastavenie ukazovateľa zásobníka
    out    SPH,temp            ;na koniec pamäte RAM
    ldi    temp,low(RAMEND)
    out    SPL,temp
,*****USART*****
    ldi    temp,PRENRH    ;nastavenie prenosovej rýchlosti
    ldi    temp1,PRENRL
    out    UBRRH,temp
    out    UBRL,temp1

    ldi    temp,0b00001000 ;povolenie vysielania
    out    UCSRB,temp

    ldi    temp,0b10000110 ;nastavenie formátu
    out    UCSRC,temp
;    Koniec inicializácie USARTu

    ldi    ZH,high(TABLE<<1)    ;nastav Z-pointer (r31:r30)
    ldi    ZL,low(TABLE<<1)    ;na začiatok tabuľky vys.znakov

;    Vysielanie
    ldi    temp1,17            ;pocet vysielaných znakov
I:    lpm    temp,Z+

TEST:    sbis    UCSRA,UDRE    ;test pripravenosti
        rjmp    TEST
        out    UDR,temp
        dec    temp1
        brne    I

KON:    rjmp    KON

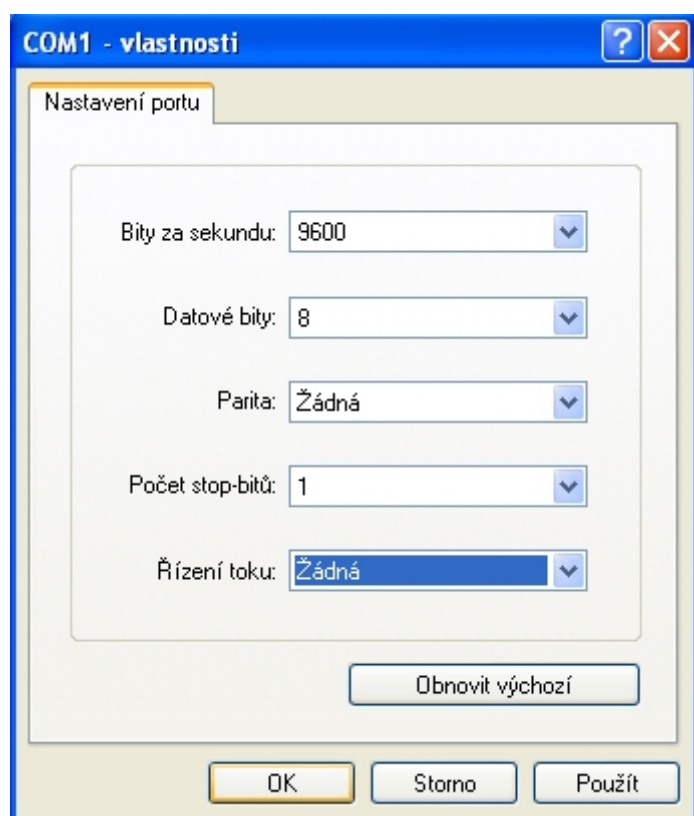
,*****koniec*****

TABLE: .db    "Pozdravuje Ta AVR"    ;text

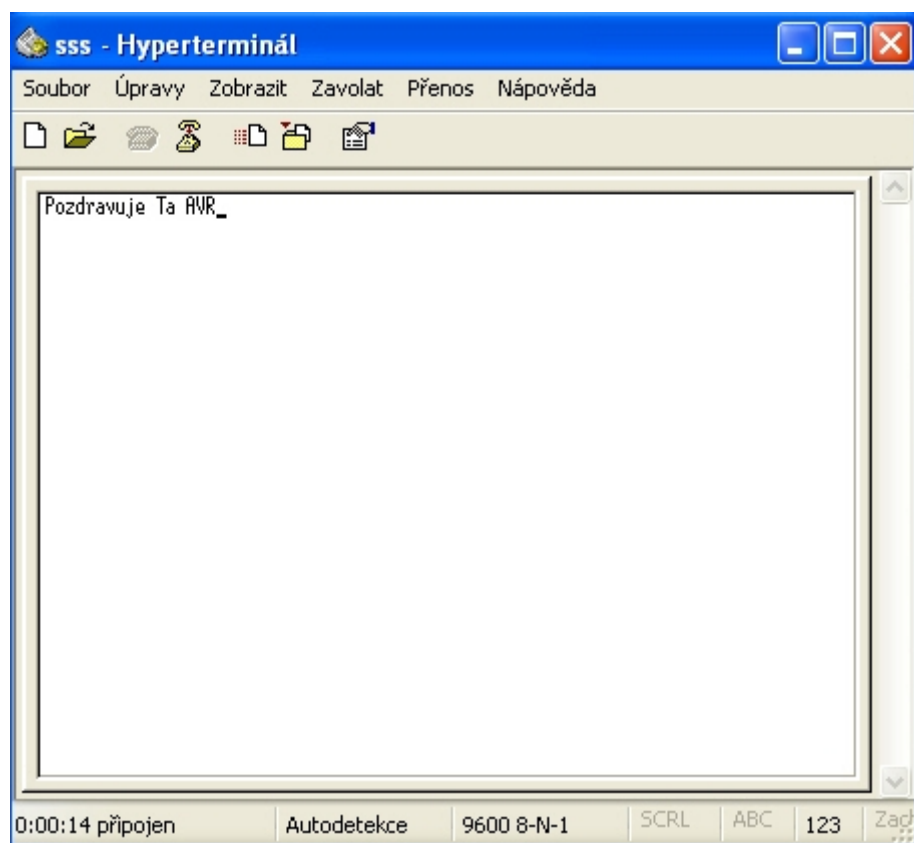
```

Funkčnosť vytvoreného programu môžeme odskúšať pomocou komunikácie s aplikáciou „Hyperterminál“. Poznamenajme, že v aplikácii Hyperterminál je potrebné nastaviť rovnaké parametre prenosu ako v module USART MCU a v ponuke riadenie toku zvoliť: „žiadne“. Modul Atmega8 je prepojkou P0 konfigurovaný v režime programovania. Pre overenie správnosti komunikácie je potrebné prepojkou P0 nastaviť tak, aby výstup sériovej linky MCU ATmega8 bol pripojený na budiaci obvod RS232. Prepojka P0 musí prepájať vývody 3-2.

Po spustení a správnom nastavení Hyperterminálu a po pripojení modulu Atmega8 na napájacie napätie, alebo reštartovaní (tlačidlo T11) sa na obrazovke objaví vysielaný text.



Obr.14 Nastavenie parametrov sériového prenosu



Obr.15 Pracovné okno Hyperterminálu

Príklad 3.2

Predchádzajúci program upravme tak, aby MCU prijímal znaky vysielané z Hyperterminálu a prijatý znak bol vysielaný späť na sériovú linku (echo). Uvedený program má zaistiť také správanie Hyperterminálu, aby znak ktorý stlačíme na klávesnici sa zobrazil v okne aplikácie Hyperterminál. Parametre prenosu sú 9600b/s, 8 bitov, jeden stop bit, bez parity, bez riadenia toku.

Výpis programu:

```
;program príklad 3.2 seriova komunikacia
; J Micek 10.4.2005

.include "m8def.inc"

.def    temp=r16
.def    temp1=r17

.equ    PRENRL=103          ;definovanie prenosovej rýchlosti
.equ    PRENRH=0

.include "PRER8.asm"      ;prilinkovanie suboru PRER8.asm

RESET:

    ldi    temp,high(RAMEND)    ;Nastavenie ukazovateľa zásobníka
    out    SPH,temp            ;na koniec pamäte RAM
    ldi    temp,low(RAMEND)
    out    SPL,temp
;*****USART*****
    ldi    temp,PENRH          ;nastavenie prenosovej rýchlosti
    ldi    temp1,PENRL
    out    UBRRH,temp
    out    UBRRL,temp1

    ldi    temp,0b00011000      ;povolenie prijmu a vysielania
    out    UCSRB,temp

    ldi    temp,0b10000110      ;nastavenie formátu
    out    UCSRC,temp
;    Koniec inicializácie USARTu

;----- Prijem znaku-----
TESTR:    sbis    UCSRA,RXC          ;test pripravenosti
          rjmp    TESTR
          in      temp,UDR

;----- Vysielanie znaku -----
TEST:     sbis    UCSRA,UDRE          ;test pripravenosti
          rjmp    TEST
          out     UDR,temp

          rjmp    TESTR              ;skok na prijem ďalšieho znaku
;*****koniec*****
```

Príklad 3.3

Upravme predchádzajúci program tak, aby sme na identifikáciu príjmu znaku využili prerušenie. Teraz sme uvoľnili procesor, ktorý nemusí testovať bit RXC registra UCSRA a môže byť využitý na iné účely.

Úprava je veľmi jednoduchá a spočíva v nasledujúcich krokoch:

1. Upraviť tabuľku vektorov prerušení, tak aby na pozícii, ktorá odpovedá vektoru prerušenia od udalosti RXC, sme umiestnili skok na obslužný podprogram príjmu znaku s návěstím `USART_RXC`.

```
.org    0x0                ;ulozit od adresy 0

rjmp    RESET
reti;   rjmp    EXT_INT0    ;IRQ handler
reti;   rjmp    EXT_INT1
reti;   rjmp    TIM2_COMP
reti;   rjmp    TIM2_OVF
reti;   rjmp    TIM1_CAPT
reti;   rjmp    TIM1_COMPA
reti;   rjmp    TIM1_COMPB
reti;   rjmp    TIM1_OVF
reti;   rjmp    TIM0_OVF
reti;   rjmp    SPI_STC
rjmp    USART_RXC
reti;   rjmp    USART_UDRE
reti;   rjmp    USART_TXC
reti;   rjmp    ADCC
reti;   rjmp    EE_RDY
reti;   rjmp    ANA_COMP
reti;   rjmp    TWSI
reti;   rjmp    SPM_RDY
```

2. Povolit prerušenie od udalosti RXC, zapísaním hodnoty jedna do bitu RXCIE registra UCSRB.

```
ldi      temp,0b10011000 ;povolenie prijmu, vysielania a prerusenia od RXC
out      UCSRB,temp
```

3. Napísať obslužnú rutinu prerušenia, napríklad v tvare:

```
USART_RXC:                ; obsluha prerusenia
    in     temp,SREG        ; uchovanie stavoveho slova
    push  temp
    in     temp,UDR         ; citanie znaku
; vyslanie znaku
TEST:  sbis   UCSRA,UDRE    ; test pripravenosti
    rjmp   TEST
    out    UDR,temp

    pop    temp
    out    SREG,temp        ; navrat stavoveho slova
    reti   ; navrat z prerusenja
```

4. Povolit prerušovací systém inštrukciou `sei`.

;program príklad 3.3 seriová komunikácia
; J Micek 10.4.2005

.include "m8def.inc"

```
.def    temp=r16
.def    temp1=r17

.equ    PRENRL=103          ;definovanie prenosovej rýchlosti
.equ    PRENRH=0
```

```
rjmp    RESET
reti;   rjmp    EXT_INT0      ;IRQ handler
reti;   rjmp    EXT_INT1
reti;   rjmp    TIM2_COMP
reti;   rjmp    TIM2_OVF
reti;   rjmp    TIM1_CAPT
reti;   rjmp    TIM1_COMPA
reti;   rjmp    TIM1_COMPB
reti;   rjmp    TIM1_OVF
reti;   rjmp    TIM0_OVF
reti;   rjmp    SPI_STC
rjmp    USART_RXC
reti;   rjmp    USART_UDRE
reti;   rjmp    USART_TXC
reti;   rjmp    ADCC
reti;   rjmp    EE_RDY
reti;   rjmp    ANA_COMP
reti;   rjmp    TWSI
reti;   rjmp    SPM_RDY
```

RESET:

```
ldi      temp,high(RAMEND)    ;Nastavenie ukazovateľa zásobníka
out      SPH,temp            ;na koniec pamäte RAM
ldi      temp,low(RAMEND)
out      SPL,temp
*****USART
ldi      temp,PRENRH        ;nastavenie prenosovej rýchlosti
ldi      temp1,PRENRL
out      UBRRH,temp
out      UBRRL,temp1

ldi      temp,0b10011000 ;povolenie prijmu, vysielania a prer
out      UCSRB,temp

ldi      temp,0b10000110 ;nastavenie formátu
out      UCSRC,temp
;      Koniec inicializácie USARTu
```

```
sei ;globalne povolenie prerušení
```

```
KON:   rjmp    KON          ;v tejto aplikácii tu stojíme, ale
                               ;inak môže procesor vykonávať inú činnosť
```

******OBSLUŽNY PODPROGRAM PRERUŠENIA*

```
USART_RXC: ; obsluha prerušenia
in      temp,SREG          ;uchovanie stavového slova
push    temp
```

```

        in      temp,UDR      ;citanie znaku
;vyslanie znaku
TEST:   sbis    UCSRA,UDRE    ;test pripravenosti
        rjmp   TEST
        out    UDR,temp

        pop    temp
        out    SREG,temp     ;navrat stavoveho slova
        reti   ;navrat z prerusenja

,*****koniec*****

```

Poznámka: Zmenené časti kódu sú zvýraznené.

Príklad 3.4

Upravme predchádzajúci program tak, aby znak, ktorý vyšleme prostredníctvom sériovej linky sa zobrazil na LCD module a súčasne vyslal späť na sériovú linku. Potrebné úpravy sú trochu komplikovanejšie, pretože vývody MCU RXD a TXD sú využívané aj na riadenie komunikácie s LCD zobrazovacím modulom.

Potrebné úpravy pozostávajú z nasledovných krokov:

- definovanie symbolického mena temp2, ktoré využívajú podprogramy LCDX,
- nastavenie V/V portov B a D, potrebné pre komunikáciu s LCD
- inicializácia LCD, podprogram INIT, ktorý je súčasťou súboru LCDX.asm,
- nastavenie adresy znaku zobrazovaného na LCD
- v obsluhu prerušenia USART_RXC prestavenie alternatívnych funkcií vývodov RXD a TXD, prípadne i zákaz prerušenia od RXC,
- volanie podprogramu DATA, zobrazenie obsahu registra temp,
- volanie podprogramu INST, nastavenie adresy zobrazovania ďalšieho znaku,
- znovu nastavenie alternatívnych funkcií vývodov RXD a TXD,
- prilinkovanie súboru LCDX ktorý obsahuje volané podprogramy.

Výpis upraveného podprogramu:

```

;program priklad 3.4 seriova komunikacia +LCD
; J Micek 10.4.2005

```

```

.include "m8def.inc"

```

```

.def    temp=r16
.def    temp1=r17
.def    temp2=r18
.equ    PRENRL=103      ;definovanie prenosovej rýchlosti
.equ    PRENRH=0

rjmp    RESET
reti;   rjmp    EXT_INT0    ;IRQ handler
reti;   rjmp    EXT_INT1
reti;   rjmp    TIM2_COMP
reti;   rjmp    TIM2_OVF
reti;   rjmp    TIM1_CAPT
reti;   rjmp    TIM1_COMPA
reti;   rjmp    TIM1_COMPB
reti;   rjmp    TIM1_OVF

```

```

    reti;    rjmp    TIM0_OVF
    reti;    rjmp    SPI_STC
    rjmp     USART_RXC
    reti;    rjmp    USART_UDRE
    reti;    rjmp    USART_TXC
    reti;    rjmp    ADCC
    reti;    rjmp    EE_RDY
    reti;    rjmp    ANA_COMP
    reti;    rjmp    TWSI
    reti;    rjmp    SPM_RDY

RESET:
    ldi      temp,high(RAMEND)    ;Nastavenie ukazovateľa zásobníka
    out      SPH,temp            ;na koniec pamäte RAM
    ldi      temp,low(RAMEND)
    out      SPL,temp
;*****USART
    ldi      temp,0xff
    out      DDRD,temp           ;PortD výstup
    out      DDRB,temp           ;PortB výstup
    clr      temp
    out      DDRC,temp           ;Port C vstup
    ser      temp
    out      PORTD,temp          ;PortD do H
    clr      temp
    out      PORTB,temp          ;PortB do L

    ldi      temp,PENRHL         ;nastavenie prenosovej rýchlosti
    ldi      temp1,PENRLL
    out      UBRRH,temp
    out      UBRL,temp1

    rcall    INIT                ;inicializácia LCD
    ldi      temp,0x80            ;adresa,pozícia znaku na DSP
    rcall    INST

    ldi      temp,0b10011000      ;povolenie prijmu,vysielania a prer
    out      UCSRB,temp
    ldi      temp,0b10000110      ;nastavenie formátu
    out      UCSRC,temp
;    Koniec inicializácie USARTu

    sei                                ;globalne povolenie prerušení

KON:    rjmp    KON              ;v tejto aplikácii tu stojíme, ale
                                ;inak môže procesor vykonávať inú činnosť

;*****OBSLUŽNY PODPROGRAM PRERUŠENIA

USART_RXC:
    in       temp,SREG            ;obsluha prerušenia
    push     temp                ;uchovanie stavového slova
    in       temp,UDR            ;čítanie znaku
;vyslanie znaku
TEST:    sbis     UCSRA,UDRE      ;test pripravenosti
    rjmp     TEST

```



```

out    UDR,temp

ldi    temp1,0b00000000 ;zakazanie prijmu,vysielania a prer
out    UCSRB,temp1

rcall   DATA           ;zapis dat LCD

ldi    temp,0x80         ;adresa,pozicia znaku na DSP
rcall   INST

ldi    temp1,0b10011000 ;povolenie prijmu,vysielania a prer
out    UCSRB,temp1

pop     temp
out     SREG,temp       ;navrat stavoveho slova
reti   ;navrat z prerusenja

;*****koniec *****

.include "LCDX.asm"

```

Integrovaná jednotka MCU, USART je výkonným a flexibilným prostriedkom umožňujúcim sériovú komunikáciu. Je zrejmé, že uvedené aplikačné príklady využívajú len malú časť ponúkaných možností. Nie sú tu uvedené zaujímavé aplikácie „multimaster“ komunikácie, synchronného prenosu a mnohé ďalšie. Na obmedzenom priestore však nie je možné sa podrobne venovať všetkým možnostiam, ktoré uvedený obvod užívateľom ponúka.

Príklad: Pri programovaní obvodu zapíšete 3 miestne číslo do pamäte EEPROM (adr. 00az 02). Po pripojení na napájacie napätie zobrazte obsah prvých troch bytov EEPROM na LED DSP.

Pri každom zapnutí inkrementujte obsah prvých troch bytov pamäte EEPROM o hodnotu 1. (počítadlo počtu zapnutí.)

4.Časovače/čítače obvodu ATmega8

Jednotky časovačov/čítačov sú jednými z najčastejšie používanými modulmi mikrokontrolérov. Pre ich všestranné možnosti využitia sa s nimi stretneme vo väčšine aplikácií MCU. Sú využívané na synchronizáciu výpočtov, generovanie definovaných časových úsekov, ako počítadlá (čítače) externých udalostí, na generovanie priebehov a podobne. MCU ATmega8 obsahuje 3 nasledovné jednotky časovačov/čítačov:

- 8-bitový časovač/čítač 0
- 16-bitový časovač/čítač 1
- 16-bitový časovač/čítač 2

a. Časovač/čítač 0

Časovač/čítač 0 je tvorený modulom univerzálneho, jednokanálového 8-bitového časovača/čítača. Jeho základné charakteristiky sú nasledovné:

- jednokanálový čítač
- frekvenčný generátor
- čítač externých udalostí
- 10-bitová preddelička hodinového signálu

Časovač/čítač 0 (TCNT0) je tvorený 8-bitovým registrom. Žiadosti o prerušenie sú uchované v registri príznakov prerušenia časovača (TIFR). Všetky prerušenia sú individuálne maskované pomocou registra masiek prerušenia časovača (TIMSK).

Časovač/čítač môže byť budený internými obvodmi, výstupom preddeličky, alebo externým zdrojom hodinového signálu, prostredníctvom vstupu T0. Blok výberu zdroja hodinových impulzov určuje zdroj hodinového signálu a hranu, na ktorú má byť obsah čítača modifikovaný. Časovač/čítač nie je aktívny, ak nie je vybraný žiadny zdroj hodinového signálu.

Základná časť časovača/čítača je programovateľná jednotka čítača. Smer počítania je vždy nahor (t.j. inkrement). Čítač pretečie, ak jeho obsah prekročí hodnotu MAX=0xFF, v tomto prípade sa jeho obsah (nastaví na hodnotu 0x00). Ak čítač pracuje v normálnom režime, príznak prerušenia TOV0 sa nastaví v tom istom hodinovom cykle, v ktorom sa obsah čítača nastaví na hodnotu BOTTOM=0x00. V tomto prípade sa príznak TOV0 správa ako deviaty bit 8-bitového čítača.

Jednotka časovača/čítača 0 obsahuje nasledovné, užívateľovi prístupné registre:

Riadiaci register časovača/čítača 0, TCCR0**TCCR0**

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	-	CS02	CS01	CS00
Prístup	R	R	R	R	R	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Tri bity CS00, CS01 a CS02 sa používajú na výber zdroja hodinového signálu. Ich význam je uvedený v tab.7.

CS02	CS01	CS00	Popis
0	0	0	Žiadny zdroj hodinového signálu, časovač/čítač je zastavený
0	0	1	Clk _{IO} (bez preddelenia)
0	1	0	Clk _{IO} /8 (z preddeličky)
0	1	1	Clk _{IO} /64
1	0	0	Clk _{IO} /256
1	0	1	Clk _{IO} /1024
1	1	0	Externý zdroj, vývod T0, zostupná hrana hodinových impulzov
1	1	1	Externý zdroj, vývod T0, nábežná hrana hodinových impulzov

Tab.7 Výber zdroja hodinového signálu

Register časovača/čítača, TCNT0**TCNT0**

Bit	7	6	5	4	3	2	1	0
Symbol	TCNT07	TCNT00
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Register masiek prerušení, TIMSK**TIMSK**

Bit	7	6	5	4	3	2	1	0
Symbol	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Bit 0 – TOIE0: povolenie prerušenia pri pretečení časovača/čítača0, Timer/Counter0 Overflow Interrupt Enable

Ak bit TOIE0 má hodnotu log.1 a súčasne je globálne povolené prerušenie (bit I v SREG má hodnotu log.1), potom je povolené prerušenie pri pretečení TCNT0.

Register príznakov prerušenia od časovačov/čítačov, TIFR**TIFR**

Bit	7	6	5	4	3	2	1	0
Symbol	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Bit 0 – TOV0: príznak pretečenia registra TCNT0, Timer/Counter0 Overflow Flag

Bit TOV0 sa nastaví na hodnotu log.1 v prípade pretečenia registra TCNT0. Bit TOV0 je automaticky nulovaný v priebehu realizácie obsluhy prerušenia. Obsah bitu TOV0 môže užívateľ vynulovať zápisom log.1.

V/V register špeciálnych funkcií SFIOR**SFIOR**

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	ACME	PUD	PSR2	PSR10
Prístup	R	R	R	R	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Bit 0 – PSR10: Reštart predmeličky TCNT1 a TCNT2, Prescaler Reset TCNT1 a TCNT2

Zápisom log.1 do bitu PSR10 sa reštaruje predmelička. Bit PSR10 sa po vykonaní operácie automaticky vynuluje.

Príklad 5.1.1

Na ukážku využitia časovača/čítača 0 uveďme nasledujúci príklad. Navrhujeme program, ktorý bude po reštarte procesora každú sekundu inkrementovať obsah registra temp. Po 10 sekundách rozsvieti LED diódu. Po uplynutí ďalších 10 sekúnd LED dióda zhasne atď. To znamená, že dióda bude blikať s intervalom 10 s. Pri realizácii uvedeného programu využijeme časovač/čítač 0 a prerušovací systém.

- Časový úsek, 1 s, budeme generovať pomocou čítača 0 s využitím interného zdroja hodinového signálu (Clk_{IO}) s preddeličkou.

Frekvencia hodinového signálu Clk_{IO} je 16 MHz. Výstup preddeličky nech je rovný $\text{Clk}_{\text{IO}}/1024 = 15.625 \text{ kHz}$. V prípade, že využijeme plný rozsah 8-bitového čítača, maximálny dosiahnuteľný časový úsek, medzi dvoma pretečeniami je:

$$1/(15.625) * 256 = 16.384 \text{ ms}$$

Je zrejmé, že maximálny časový úsek generovaný 8-bitovým časovačom pri využití hodinového signálu Clk_{IO} = 16 MHz je 16.384 ms. Z uvedeného vyplýva, že na riešenie danej úlohy (generovanie času 1s) musíme použiť pomocné počítadlo počtu prerušení. Počítadlo počtu prerušení budeme v každom prerušení inkrementovať a pri dosiahnutí hodnoty $1/0.016384 = 61.035156$ je potrebné inkrementovať register počítania sekúnd, temp. Pretože obsah registra môžeme inkrementovať len s krokom 1, už pri dosiahnutí hodnoty 61 budeme zvyšovať obsah registra temp. Je zrejmé, že sa v tomto prípade dopustíme chyby spôsobenej zanedbaním desatinnej časti (0.035156), t.j. $0.035156/61.035156 = 0.5759 \text{ ms}$. Namiesto 1s budeme register temp inkrementovať každých 999.4241 milisekúnd.

Výpis jadra programu:

```

.
.
rjmp    TIM0_OVF                ;prer od C/C 0
.
.
ldi     temp,0x5                ;nastavenie deliaceho pomeru 1:1024
out     TCCR0,temp
ldi     temp,0x1                ;nastavenie masky prer.
out     TIMSK,temp

sei                                           ;povolenie preruseni
ldi     temp,0                  ;pociatocna hodnota pocitadla
ldi     temp1,0                 ;temp1 ...pocitadlo sekund
KON:    rjmp    KON             ;tu stoji a caka na prerusenie

;*****obsluha prerusenia od C/C 0
TIM0_OVF:
    push    temp2                ;ulozenie/zachrana dat
    in      temp2,SREG
    push    temp2

    inc     temp
    cpi     temp,61

```

```

        brne      KK
        clr       temp
        inc       temp1          ;inkrement pocitadla sekund
        cpi       temp1,10      ;preslo uz pozadovanych 10 sek?
        brne      KK
        clr       temp1
        sbic      PORTB,0       ; zasvietenie diody
        rjmp      ZASVIET
        sbi       PORTB,0       ;zhasni LED
        rjmp      KK
ZASVIET:
        cbi       PORTB,0       ;zasviet LED

KK:
        pop       temp2          ;navrat obsahov SREG a temp2
        out       SREG,temp2
        pop       temp2
        reti
;*****KONIEC obsluhy prerusenien od C/C 0

```

V prípade, že deliaci pomer predделиčky nastavíme na 1: 256, potom výstup predделиčky bude mať frekvenciu $16000/256 = 62.5$ kHz. To je každých 0.016 ms bude inkrementovaný obsah registra čítača 0, TCNT0. Žiadosť o prerušenie bude potom generovaná, pri prechode obsahu z FF na 0 každých $0.016 * 256 = 4.096$ ms. Pri skrátení cyklu čítača na 250, bude generovaná žiadosť o prerušenie každé 4 ms. V tomto prípade, sa počítadlo počtu prerušení bude inkrementovať presne po hodnotu 250, týmto spôsobom generujeme časový úsek $250 * 4$ ms.

V tomto prípade je výpis programu v tvare:

```

.
.
rjmp    TIM0_OVF          ;prer od C/C 0
.
.
.
ldi     temp,0x4          ;nastavenie deliaceho pomeru 1:256
out     TCCR0,temp
ldi     temp,0x1          ;nastavenie masky prer.
out     TIMSK,temp
sei     ;povolenie preruseni
ldi     temp,6            ;pociatocna hodnota citaca
out     TCNT0,temp

        ldi     temp,0     ;pociatocna hodnota pocitadla
        ldi     temp1,0    ;temp1 ...pocitadlo sekund
KON:    rjmp     KON        ;tu stoji a caka na prerusenie

;*****obsluha prerusenien od C/C 0
TIM0_OVF:
        push    temp2      ;ulozenie/zachrana dat
        in      temp2,SREG
        push    temp2

        ldi     temp2,6    ;skratenie cyklu citaca
        out     TCNT0,temp2 ; inc citaca od hodnoty 6

        inc     temp

```

```

        cpi    temp,250                ;kontrola poctu preruseni
        brne   KK
        clr    temp
        inc    temp1                  ;inkrement pocitadla sekund
        cpi    temp1,10                ;preslo uz pozadovanych 10 sek?
        brne   KK
        clr    temp1
        sbic    PORTB,0                ; zasvietenie diody
        rjmp   ZASVIET
        sbi     PORTB,0                ;zhasni LED
        rjmp   KK
ZASVIET:
        cbi     PORTB,0                ;zasviet LED

KK:
        pop    temp2                  ;navrat obsahov SREG a temp2
        out    SREG,temp2
        pop    temp2
        reti
;*****KONIEC obsluhy prerusenania od C/C 0

```

V predchádzajúcich dvoch príkladoch boli uvedené veľmi jednoduché aplikácie časovača/čítača 0. V uvedených príkladoch hlavný program čakal v slučke (*KON:* *rjmp* *KON*) na výskyt prerušenia od časovača 0. Uvedené aplikácie nie sú preto typické. Uvedme preto príklad, ktorý dokumentuje praktické využitie časovača.

Príklad 5.1.2

Navrhujeme programový modul, ktorý bude realizovať obsluhu modulu zobrazovacej jednotky LED. Na zobrazovacích prvkoch DSP1 a DSP2 chceme zobrazovať obsah registra temp a na DSP3 obsah dolných 4 bitov registra temp1.

```

;program: citac2.asm zobrazí na LED DSP obsah temp a obsah
;4-roch dolnych bitov temp1

```

```

.include "m8def.inc"      ; definičný súbor ATmega8

```

```

.def    temp=r16
.def    temp1=r17
.def    temp2=r18
.def    temp3=r19
;::::::::::::::::::TABUĽKA PRIRADENÍ
.equ    nula=0b11000000
.equ    jedna=0b11111001
.equ    dva=0b10100100
.equ    tri=0b10110000
.equ    styri=0b10011001
.equ    pat=0b10010010
.equ    sest=0b10000010
.equ    sedem=0b11111000
.equ    osem=0b10000000
.equ    devat=0b10010000
.equ    ahex=0b10001000
.equ    bhex=0b10000011
.equ    chex=0b10100111
.equ    dhex=0b10100001
.equ    ehex=0b10000110
.equ    fhex=0b10001110

```

```

.cseg                                ;segment programu
.org    0x0                          ;hex kód uloži od adresy 0

    rjmp    RESET                    ; skok na štart programu
;nasleduje tabuľka vektorov prerušení, zatiaľ ich
;využívať nebudeme, pretože prer. sú globálne zakázané

    reti;   rjmp    EXT_INT0        ;IRQ handler
    reti;   rjmp    EXT_INT1
    reti;   rjmp    TIM2_COMP
    reti;   rjmp    TIM2_OVF
    reti;   rjmp    TIM1_CAPT
    reti;   rjmp    TIM1_COMPA
    reti;   rjmp    TIM1_COMPB
    reti;   rjmp    TIM1_OVF
    rjmp    TIM0_OVF                ; prer od C/C 0
    reti;   rjmp    SPI_STC
    reti;   rjmp    USART_RXC
    reti;   rjmp    USART_UDRE
    reti;   rjmp    USART_TXC
    reti;   rjmp    ADCC
    reti;   rjmp    EE_RDY
    reti;   rjmp    ANA_COMP
    reti;   rjmp    TWSI
    reti;   rjmp    SPM_RDY

RESET:
    ldi     temp,high(RAMEND)        ;Nastavenie ukazovateľa zásobníka-SP
    out     SPH,temp                ;na koniec pamäte RAM
    ldi     temp,low(RAMEND)
    out     SPL,temp
;***** Nastavenie smeru portu B*****
    ldi     temp,0b00111001
    out     DDRB,temp                ;vystup je na PB5,PB4 a PB3
    ser     temp                    ;nastavenie temp na 0xff
    out     DDRD,temp                ;celý PORTD bude výstupný
    out     PORTD,temp
    out     PORTB,temp                ;zhasnute všetky LED
;***** Nastavenie C/C 0

    ldi     temp,0x5                  ;nastavenie deliaceho pomeru 1:1024
    out     TCCR0,temp
    ldi     temp,0x1                  ;nastavenie masky prer.
    out     TIMSK,temp

    sei                                     ;povolenie prerušení

    clr     temp3                    ;temp3 ...pocítadlo počtu prerušení

    ldi     temp,0x21                  ;hodnoty, ktoré zobrazíme na LED DSP
    ldi     temp1,0x03

KON:   rjmp    KON                    ;tu stojí a čaka na prerušenie

;obsluha prerušenia od C/C 0
TIM0_OVF:

```

```

        push    temp2                ;ulozenie/zachrana dat
        in      temp2,SREG
        push    temp2
        sbi     PORTB,3
        sbi     PORTB,4
        sbi     PORTB,5
        cpi     temp3,0
        brne    T1
        cbi     PORTB,3                ;zapnuty DSP 1
        inc     temp3
        mov     r20,temp
        rcall   ZOBRAZ
        rjmp    KK

T1:      cpi     temp3,1
        brne    T2
        cbi     PORTB,4                ;zapnuty DSP 2
        mov     r20,temp
        swap    r20
        inc     temp3
        rcall   ZOBRAZ
        rjmp    KK

T2:      mov     r20,temp1
        cbi     PORTB,5                ;zapnuty DSP 3
        clr     temp3
        rcall   ZOBRAZ

KK:      pop     temp2                ;navrat obsahov SREG a temp2
        out     SREG,temp2
        pop     temp2
        reti
;*****KONIEC obsluhy prerusenja od C/C 0

```

;podprogram zobraz zobrazi (vysle na PORTD) hodnotu spodných 4 bitov r20

ZOBRAZ:

```

        push    r21
        ldi     r21,nula
        andi    r20,0x0f
        breq    Z1
        ldi     r21,jedna
        dec     r20
        breq    Z1
        ldi     r21,dva
        dec     r20
        breq    Z1
        ldi     r21,tri
        dec     r20
        breq    Z1
        ldi     r21,styri
        dec     r20
        breq    Z1
        ldi     r21,pat
        dec     r20
        breq    Z1
        ldi     r21,sest

```



```

    dec    r20
    breq   Z1
    ldi    r21,sedem
    dec    r20
    breq   Z1
    ldi    r21,osem
    dec    r20
    breq   Z1
    ldi    r21,devat
    dec    r20
    breq   Z1
    ldi    r21,ahex
    dec    r20
    breq   Z1
    ldi    r21,bhex
    dec    r20
    breq   Z1
    ldi    r21,ches
    dec    r20
    breq   Z1
    ldi    r21,dhex
    dec    r20
    breq   Z1
    ldi    r21,ehex
    dec    r20
    breq   Z1
    ldi    r21,fhex
Z1:   out   PORTD,r21
    pop    r21
    ret

```

Výpis programu obsahuje podprogram „ZOBRAZ“, ktorý bol uvedený v predchádzajúcom texte, preto ho nebudeme komentovať. Približne každých 16 ms je vyvolané prerušenie od časovača 0 a vykoná sa obslužný program prerušenia. V obslužnom programe prerušenia sa na základe hodnoty registra temp3 vyhodnotí, ktorý z prvkov DSP1 až 3 bude v danej perióde aktívny. Príslušným obsahom temp, resp. temp1 sa naplní register r20 a volá sa podprogram ZOBRAZ, ktorý realizuje vyslanie odpovedajúcej kombinácie na PORTD. Po naprogramovaní obvodu môžeme vidieť, že na DSP1 až 3 sa zobrazí číslo 321, ktorým sme naplnili registre temp1:temp. Súčasne môžeme vidieť, že prerušenie je vyvolávané s pomerne dlhou periódou, pretože blikanie LED zobrazovacej jednotky je možné zreteľne pozorovať. Najjednoduchšia úprava programu spočíva v znížení deliaceho pomeru preddeličky na hodnotu 1:256.

```
ldi    temp,0x4                ;nastavenie deliaceho pomeru 1:256
```

Ak uvedený modul budeme využívať v neskorších aplikáciách musíme si uvedomiť, že v aplikácii je potrebné:

1. Definovať mená pomocných registrov temp, temp1 a temp2

```

.def    temp=r16
.def    temp1=r17
.def    temp2=r18

```

2. Nastaviť smer vstupno-výstupných portov:

```

ldi    temp,0b00111001
out    DDRB,temp                ;vystup je na PB5,PB4 a PB3

```

```

ser    temp                ;nastavenie temp na 0xff
out    DDRD,temp          ;celý PORTD bude výstupný
out    PORTD,temp
out    PORTB,temp         ;zhasnute všetky LED

```

3. Nastaviť časovač/čítač 0:

```

ldi    temp,0x5           ;nastavenie deliaceho pomeru 1:1024
out    TCCR0,temp
ldi    temp,0x1           ;nastavenie masky prer.
out    TIMSK,temp

```

4. Nastaviť vektor prerušenia a povoliť prerušovací systém:

```

rjmp    TIM0_OVF          ;prer od C/C 0

sei                      ;povolenie preruseni

```

5. Nastaviť obsah pomocného registra temp3 na počiatočnú hodnotu 0.

```

clr     temp3             ;temp3 ...pocitadlo poctu preruseni

```

6. Prilinkovať modul

```

.include "ZOBRAZX.asm"

```

Programový modul ZOBRAZX.asm bude v tvare:

```

;Programový modul ZOBRAZX.asm
;.....:TABULKA PRIRADENÍ
.equ    nula=0b11000000
.equ    jedna=0b11111001
.equ    dva=0b10100100
.equ    tri=0b10110000
.equ    styri=0b10011001
.equ    pat=0b10010010
.equ    sest=0b10000010
.equ    sedem=0b11111000
.equ    osem=0b10000000
.equ    devat=0b10010000
.equ    ahex=0b10001000
.equ    bhex=0b10000011
.equ    chex=0b10100111
.equ    dhex=0b10100001
.equ    ehex=0b10000110
.equ    fhex=0b10001110

TIM0_OVF:
    push    r20
    push    temp2          ;ulozenie/zachrana dat
    in      temp2,SREG
    push    temp2
    sbi     PORTB,3
    sbi     PORTB,4
    sbi     PORTB,5
    cpi     temp3,0
    brne    T1

```

```

        cbi     PORTB,3                ;zapnuty DSP 1
        inc     temp3
        mov     r20,temp
        rcall   ZOBRAZ
        rjmp    KK

T1:     cpi     temp3,1
        brne    T2
        cbi     PORTB,4                ;zapnuty DSP 2
        mov     r20,temp
        swap    r20
        inc     temp3
        rcall   ZOBRAZ
        rjmp    KK

T2:     mov     r20,temp1
        cbi     PORTB,5                ;zapnuty DSP 3
        clr     temp3
        rcall   ZOBRAZ

KK:     pop     temp2                  ;navrat obsahov SREG a temp2
        out     SREG,temp2
        pop     temp2
        pop     r20
        reti
;*****KONIEC obsluhy prerusenja od C/C 0

```

;podprogram zobraz zobrazi (vysle na PORTD) hodnotu spodných 4 bitov r20

```

ZOBRAZ:
        push    r21
        ldi     r21,nula
        andi    r20,0x0f
        breq    Z1
        ldi     r21,jedna
        dec     r20
        breq    Z1
        ldi     r21,dva
        dec     r20
        breq    Z1
        ldi     r21,tri
        dec     r20
        breq    Z1
        ldi     r21,styri
        dec     r20
        breq    Z1
        ldi     r21,pat
        dec     r20
        breq    Z1
        ldi     r21,sest
        dec     r20
        breq    Z1
        ldi     r21,sedem
        dec     r20
        breq    Z1
        ldi     r21,osem

```

```

    dec    r20
    breq   Z1
    ldi    r21,devat
    dec    r20
    breq   Z1
    ldi    r21,ahex
    dec    r20
    breq   Z1
    ldi    r21,bhex
    dec    r20
    breq   Z1
    ldi    r21,ches
    dec    r20
    breq   Z1
    ldi    r21,dhex
    dec    r20
    breq   Z1
    ldi    r21,ehex
    dec    r20
    breq   Z1
    ldi    r21,fhex
Z1:    out    PORTD,r21
    pop    r21
    ret

```

Pri využívaní programového modulu ZOBRAZX.asm musíme mať na pamäti že:

- je povolený prerušovací systém MCU,
- zobrazujeme obsah registra temp a spodnej časti registra temp1,
- je využívaný register temp3,
- v aplikácii nesmieme využívať použité návestia: *TIM0_OVF*, *T1*, *T2*, *KK*, *ZOBRAZ*, *Z1*,
- aplikácia nemôže využívať časovač/čítač0.

Ak dodržíme všetky uvedené pravidlá môžeme v ďalších aplikáciách využívať uvedený modul na zobrazovanie výsledkov prostredníctvom zobrazovacieho modulu LED DSP. Do aktuálnej aplikácie ho jednoducho prilinkujeme.

Na ukážku je uvedený program, ktorý približne každú sekundu zvýši obsah registrov temp1:temp o jednotku a výsledok zobrazuje na LED DSP. Poznávam, že uvedený program je možné napísať úspornejšie, čo sa týka dĺžky kódu, alebo počtu použitých registrov.

```

.include "m8def.inc"                ; definovaný súbor ATmega8

.def    temp=r16                    ; definovanie symbolického mena
.def    temp1=r17
.def    temp2=r18
.def    temp3=r19
.def    temp4=r20
.def    temp5=r21
.def    temp6=r22
.cseg                                  ; segment programu
.org    0x0                          ; hex kód uložiť od adresy 0

rjmp    RESET                        ; skok na štart programu

reti;    rjmp    EXT_INT0            ; IRQ handler

```

```

reti; rjmp EXT_INT1
reti; rjmp TIM2_COMP
reti; rjmp TIM2_OVF
reti; rjmp TIM1_CAPT
reti; rjmp TIM1_COMPA
reti; rjmp TIM1_COMPB
reti; rjmp TIM1_OVF
rjmp TIM0_OVF ; prer od C/C 0
reti; rjmp SPI_STC
reti; rjmp USART_RXC
reti; rjmp USART_UDRE
reti; rjmp USART_TXC
reti; rjmp ADCC
reti; rjmp EE_RDY
reti; rjmp ANA_COMP
reti; rjmp TWSI
reti; rjmp SPM_RDY

RESET:
    ldi temp,high(RAMEND) ;Nastavenie ukazovateľa zásobníka-SP
    out SPH,temp ;na koniec pamäte RAM
    ldi temp,low(RAMEND)
    out SPL,temp
,***** Nastavenie smeru portu B*****
    ldi temp,0b00111001
    out DDRB,temp ;vystup je na PB5,PB4 a PB3

    ser temp ;nastavenie temp na 0xff
    out DDRD,temp ;celý PORTD bude výstupný
    out PORTD,temp
    out PORTB,temp ;zhasnute všetky LED

,***** Nastavenie C/C 0
    ldi temp,0x4 ;nastavenie deliaceho pomeru 1:256
    out TCCR0,temp
    ldi temp,0x1 ;nastavenie masky prer.
    out TIMSK,temp

    sei ;povolenie preruseni
    clr temp3 ;temp3 ...pocítadlo počtu preruseni

POK1:    ldi temp,0x00 ;poc. hodnota LED DSP
        ldi temp1,0x00

POK:    inc temp
        brne PP
        inc temp1
        cpi temp1,0xff ; pocita do hodnoty 0xff00
        breq POK1

PP:    rcall CAKAJ
        rjmp POK

,*****čakacia slučka
CAKAJ:
        ldi temp4,82 ;reg.temp4 naplníme hodnotou 82
C3:    ldi temp5,0xff ;reg.temp5 naplníme hodnotou 0xff=255
C2:    ldi temp6,0xff ;reg.temp6 naplníme hodnotou 0xff
C1:    dec temp6
        brne C1

```

```

dec      temp5
brne     C2
dec      temp4
brne     C3
ret

```

```
.include "ZOBRAZX.asm"
```

b. Časovač/čítač 1

16-bitový časovač/čítač1 umožňuje užívateľovi časovať vykonávanie programu, realizovať generátor priebehov, meranie časových parametrov signálov a pod.. Jeho základné charakteristiky sú nasledovné:

- 16-bitová štruktúra
- dve nezávislé porovnávacie jednotky
- výstupné porovnávacie registre s pomocným registrom
- vstupnú záchytnú jednotku
- jednotku na potlačenie šumu
- automatické nulovanie časovača pri zhode
- fázovo korektná PWM
- možnosť voľby periódy PWM
- frekvenčný generátor
- čítač externých udalostí
- štyri nezávislé zdroje prerušenia (TOV1, OCF1A, OCF1B, ICF1)

Modul časovača/čítača1 využíva nasledovné riadiace a stavové registre:

Riadiaci register A časovača/čítača 1, TCCR1A

TCCR1A

Bit	7	6	5	4	3	2	1	0
Symbol	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Prístup	R	R	R	R	R	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Bity 6:7 - COM1A1:0: Výstupný porovnávací režim pre kanál A, Compare Output Mode for channel A

Bity 5:4 - COM1B1:0: Výstupný porovnávací režim pre kanál B, Compare Output Mode for channel B

Pomocou obsahu bitov COM1A1:0, prípadne COM1B1:0 je určené správanie výstupu OC1A, prípadne OC1B. Tabuľka 41 popisuje funkciu bitov COM1x1:0, za predpokladu, že bity WGM13:0 sú nastavené tak, aby časovač/čítač pracoval v normálnom režime, prípadne v režime CTC (nie v režimoch PWM).

COM1A1/ COM1B1	COM1A0/ COM1B0	Popis
0	0	Normálna V/V funkcia, OC1A/OC1B odpojené

0	1	Zmena výstupu OC1A/OC1B na zhodu pri porovnávaní
1	0	Nulovanie (nízka úroveň) OC1A/OC1B na zhodu pri porovnávaní
1	1	Nastavenie (vysoká úroveň) OC1A/OC1B na zhodu pri porovnávaní

Tab.8 Porovnávací výstupný režim

Tabuľka 9 popisuje funkciu bitov COM1x1:0 za predpokladu, že bity WGM13:0 sú nastavené tak, aby časovač/čítač pracoval v režime rýchlej PWM.

COM1A1/ COM1B1	COM1A0/ COM1B0	Popis
0	0	Normálna V/V funkcia, OC1A/OC1B odpojené
0	1	WGM13:0 =15, zmena výstupu OC1A pri zhode, výstup OC1B je odpojený (normálna V/V funkcia). Pri iných nastaveniach WGM1 je OC1A/OC1B odpojené.
1	0	Nulovanie (nízka úroveň) OC1A/OC1B na zhodu, nastavenie OC1A/OC1B pri dosiahnutí hodnoty TOP.
1	1	Nastavenie (vysoká úroveň) OC1A/OC1B na zhodu, nulovanie OC1A/OC1B pri dosiahnutí hodnoty TOP.

Tab.9 Porovnávací výstupný režim v režime rýchlej PWM

Tabuľka 10 popisuje funkciu bitov COM1x1:0 za predpokladu, že bity WGM13:0 sú nastavené tak, aby časovač/čítač pracoval v režime fázovo korektnej a fázovo a frekvenčne korektnej PWM.

COM1A1/ COM1B1	COM1A0/ COM1B0	Popis
0	0	Normálna V/V funkcia, OC1A/OC1B odpojené
0	1	WGM13:0 =9, alebo 14, zmena výstupu OC1A pri zhode, výstup OC1B je odpojený (normálna V/V funkcia). Pri iných nastaveniach WGM1 je OC1A/OC1B odpojené.
1	0	Nulovanie (nízka úroveň) OC1A/OC1B na zhodu pri zvyšovaní obsahu TCNT1, nastavenie OC1A/OC1B na zhodu pri znižovaní obsahu TCNT1.
1	1	Nastavenie (vysoká úroveň) OC1A/OC1B na zhodu pri zvyšovaní obsahu TCNT1, nulovanie OC1A/OC1B na zhodu pri znižovaní obsahu TCNT1.

Tab.10 Porovnávací výstupný režim v režime fázovo korektnej a fázovo a frekvenčne korektnej PWM

Bit 3 – FOC1A: Vnútenie výstupu porovnania na kanál A, Force Output Compare for channel A

Bit 2 – FOC1B: Vnútenie výstupu porovnania na kanál B, Force Output Compare for channel B

Bity FOC1A/FOC1B sú aktívne len v normálnom a CTC režime, nie v PWM režimoch. V režimoch PWM musia byť nastavené na hodnotu log.0. Ak v FOC1A/FOC1B bitoch je zapísaná logická 1, potom výstupom porovnávacej jednotky je priamo ovládaný generátor priebehov.

Bit 1:0 – WGM11:0: Režim generovania priebehov, Waveform Generation Mode

Bity WGM11:0 spolu s bitmi WGM13:2 v registri TCCR1B riadia počítacie sekvencie čítača, určujú zdroj pre hodnotu TOP a typ generovania priebehov. Režimy činnosti časovača/čítača sú nasledovné:

Normálny režim (čítač),
nulovanie čítača na zhodu pri porovnaní (CTC režim),
tri režimy PWM (rýchla, fázovo korektná a frekvenčne a fázovo korektná).
Tabuľka 11 popisuje jednotlivé režimy.

Rež.	WGM 13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Režim činnosti Č/Č	TOP	Prepis OCR1x	TOV1 nastavenie na
0	0	0	0	0	Normálny	0xFFFF	Hned'	MAX
1	0	0	0	1	PWM, fáz. korektný, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, fáz. korektný, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, fáz. korektný, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Hned'	MAX
5	0	1	0	1	PWM rýchla, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	PWM rýchla, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	PWM rýchla, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM fázovo a frekvenčne korekt.	ICR1	BOTT.	BOOTOM
9	1	0	0	1	PWM fázovo a frekvenčne korekt.	OCR1A	BOTT.	BOTTOM
10	1	0	1	0	PWM fázovo korektná	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM fázovo	OCR1A	TOP	BOTTOM

					korektná			
12	1	1	0	0	CTC	ICR1	Hneď	MAX
13	-	-	-	-	Nevyužíte	-	-	-
14	1	1	1	0	PWM rýchla	ICR1	TOP	TOP
15	1	1	1	1	PWM rýchla	OCR1A	TOP	TOP

Tab.11 Režimy generovania priebehov

Riadiaci register B časovača/čítača 1, TCCR1B**TCCR1B:**

Bit	7	6	5	4	3	2	1	0
Symbol	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
Prístup	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Bit 7 - ICNC1: Potlačenie šumu na záchytnom vstupe, Input Capture Noise Canceler

Nastavením bitu ICNC1 na hodnotu log.1 sa aktivuje blok potlačenia šumu na záchytnom vstupe ICP1. Funkcia filtrácie vyžaduje na zmenu výstupnej hodnoty štyri rovnaké, po sebe idúce vzorky vstupného signálu na vstupe ICP1. Blok potlačenia šumu spôsobuje oneskorenie signálu o 4 periódy hodinového signálu.

Bit 6 - ICES1: Voľba hrany signálu na záchytnom vstupe, Input Capture Edge Select

Pomocou bitu ICES1 je možné zvoliť aktívnu hranu signálu na vstupe ICP1. Ak bit ICES1 obsahuje hodnotu log.0, potom je aktívna zostupná hrana signálu. Ak bit ICES1 obsahuje hodnotu log.1, potom je aktívna nábežná hrana signálu. Keď sa na vstupe ICP1 objaví vybraná hrana signálu, potom aktuálna hodnota čítača sa prepíše do záchytného registra ICR1 a súčasne sa nastaví príznak ICF1. Ak je register ICR1 využívaný na definovanie hodnoty TOP, potom je vstup ICP1 odpojený.

Bit 5 - : Nevyužitý (rezervovaný) bit**Bit 4 - WGM13:2: Režim generovania priebehov, Waveform Generation Mode**

Význam jednotlivých bitov je uvedený v tabuľke 43.

Bit 2:0 - CS12:0: Voľba hodinového signálu, Clock Select

Pomocou bitov CS12:0 je možné zvoliť zdroj hodinového signálu pre vstup časovača/čítača podľa tabuľky 12.

CS12	CS11	CS10	Popis
0	0	0	Nie je vybraný žiadny zdroj hodinového signálu (STOP)
0	0	1	clk _{IO}
0	1	0	Clk _{IO} /8 (výstup preddeličky)
0	1	1	Clk _{IO} /64 (výstup preddeličky)
1	0	0	Clk _{IO} /256 (výstup preddeličky)

1	0	1	Clk _{IO} /1024 (výstup preddeličky)
1	1	0	Externý vstup na vývode T1, záverná hrana signálu
1	1	1	Externý vstup na vývode T1, nábežná hrana signálu

Tab.12 Výber zdroja hodinového signálu

Ak je zvolený externý zdroj, potom aj v prípade, že vývod T1 je konfigurovaný ako výstup definovaná zmena jeho hodnoty, bude interpretovaná ako vstupný hodinový signál. Táto vlastnosť dovoľuje programové riadenie zvyšovania/znižovania obsahu čítača.

Časovač/čítač1 – TCNT1H a TCNT1L

TCNT1:

Bit	7	6	5	4	3	2	1	0
Symbol	TCTN1 15:8 (TCTN1H)							
Symbol	TCTN1 7:0 (TCTN1L)							
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Dve V/V pamäťové miesta (TCTN1L a TCTN1H) ponúkajú možnosť čítať/zapisovať obsah 16-bitového čítača TCNT1. Aby bolo možné obsah čítača čítať/zapisovať simultánne je prístup do časti TCNT1H realizovaný prostredníctvom pomocného 8-bitového registra TEMP. Poznamenajme, že tento pomocný register využívajú aj ďalšie 16-bitové registre v bloku čítača/časovača1.

Výstupný porovnávací register 1A – OCR1AH a OCR1AL

OCR1A

Bit	7	6	5	4	3	2	1	0
Symbol	OCR1A 15:8 (OCR1AH)							
Symbol	OCR1A 7:0 (OCR1AL)							
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Výstupný porovnávací register 1B – OCR1BH a OCR1BL

OCR1B:

Bit	7	6	5	4	3	2	1	0
Symbol	OCR1B 15:8 (OCR1BH)							
Symbol	OCR1B 7:0 (OCR1BL)							
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Výstupné porovnávacie registre obsahujú hodnoty, ktoré sú priebežne porovnávané s obsahom čítača TCNT1. Zhoda obsahov môže byť využitá na generovanie žiadosti o prerušenie, prípadne v procese generovania priebehov na výstupe OC1x. Na zaistenie simultánneho prístupu k obidvom 8-bitovým registrom (OCR1BL a OCR1BH) sa využíva pomocný register TEMP.

Vstupný záchytný register 1 – ICR1H a ICR1L

ICR1:

Bit	7	6	5	4	3	2	1	0
Symbol	ICR1 15:8 (ICR1H)							
Symbol	ICR1 7:0 (ICR1L)							
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Obsah vstupného záchytného registra je obnovovaný (prepísovaný) obsahom TCNT1 vždy, ak sa na vstupe ICP1, alebo na analógovom komparátore vyskytne definovaná udalosť. Register ICR1 môže byť tiež využitý na definovanie hodnoty TOP. Na zaistenie simultánneho prístupu k obom 8-bitovým registrom (OCR1BL a OCR1BH) sa využíva pomocný register TEMP.

Maskovací register prerušení od časovača/čítača, TIMSK

TIMSK:

Bit	7	6	5	4	3	2	1	0
Symbol	OCIE2	TOIE2	TICE1	OCIE1A	OCIE1B	TOIE1	-	TOIE0
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
P. hod.	0	0	0	0	0	0	0	0

Bit 5 - TICE1: Povolenie prerušenia od vstupnej záchytnéj udalosti, Input Capture Interrupt Enable

Ak bit TICE1 je nastavený na hodnotu log.1, potom v prípade, že aj bit I v stavovom registri je nastavený na hodnotu log.1 (globálne povolenie prerušení), je povolené prerušenie pri výskyte udalosti, ktorá spôsobí prepis obsahu TCTN1 do ICR1.

Bit 4 - OCIE1A: Povolenie prerušenia na zhodu obsahov TCTN1 a OCR1A, Timer/Counter1, Output Compare A Match Interrupt Enable

Ak bit OCIE1A je nastavený na hodnotu log.1, potom v prípade že sú globálne povolené prerušenia, je povolené prerušenie pri zhode obsahu časovača/čítača1 s obsahom výstupného porovnávacieho registra OCR1A.

Bit 3 - OCIE1B: Povolenie prerušenia na zhodu obsahov TCTN1 a OCR1B, Timer/Counter1, Output Compare B Match Interrupt Enable

Ak bit OCIE1B je nastavený na hodnotu log.1 a sú globálne povolené prerušenia, potom je povolené prerušenie pri zhode obsahu časovača/čítača1 s obsahom výstupného porovnávacieho registra OCR1B.

Bit 2 - TOIE1: Povolenie prerušenia pri pretečení TCTN1, Overflow Interrupt Enable

Ak bit TOIE1 je nastavený na hodnotu log.1 a súčasne sú globálne povolené prerušenia, potom je povolené prerušenie pri pretečení časovača/čítača1.

Register príznakov prerušení od časovača/čítača1, TIFR

TIFR:

Bit	7	6	5	4	3	2	1	0
Symbol	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	-	TOV0
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W
P. hod.	0	0	0	0	0	0	0	0

Bit 5 - ICF1: Časovač/čítač1, vstupný záchytný príznak, Timer/Counter1, Input Capture Flag

Bit ICF1 sa nastaví na hodnotu log.1 pri výskyte definovanej udalosti na vstupe ICP1. Ak je register ICR1 využitý na definovanie hodnoty TOP, príznak ICF1 sa nastaví na hodnotu log.1 vtedy, keď obsah čítača dosiahne hodnotu TOP. Bit ICF1 je automaticky vynulovaný pri výkone odpovedajúcej obsluhy prerušenia, alebo zápisom hodnoty log.1.

Bit 4 - OCF1A: Časovač/čítač1, príznak zhody obsahov TCTN1 a OCR1A, Timer/Counter1, Output Compare A Match Flag

Bit OCF1A sa nastaví na hodnotu log.1 v hodinovom cykle po dosiahnutí rovnosti obsahov registrov TCNT1 a OCR1A. Bit OCF1A sa automaticky vynuluje pri výkone odpovedajúcej prerušovacej rutiny, alebo zápisom hodnoty log.1.

Bit 3 - OCF1B: Časovač/čítač1, príznak zhody obsahov TCTN1 a OCR1B, Timer/Counter1, Output Compare B Match Flag

Bit OCF1B sa nastaví na hodnotu log.1 v hodinovom cykle po dosiahnutí rovnosti obsahov registrov TCNT1 a OCR1B. Bit OCF1B sa automaticky vynuluje pri výkone odpovedajúcej prerušovacej rutiny, alebo zápisom hodnoty log.1.

Bit 2 - TOV1: Časovač/čítač1, príznak pretečenia, Timer/Counter1, Overflow Flag

V normálnom a CTC režime sa bit TOV1 nastaví na hodnotu log.1 pri pretečení čítača. Správanie sa príznaku TOV1 v iných režimoch bolo uvedené v tab.43. Bit TOV1 sa automaticky vynuluje pri výkone odpovedajúcej prerušovacej rutiny, alebo zápisom hodnoty log.1.

Príklad 1.2.1

Na ilustráciu využitia časovača/čítača1 uveďme nasledujúci príklad. Navrhujeme program, ktorý bude po reštarte procesora každú sekundu inkrementovať obsah 16-bitového registra temp:temp1. Obsah registra budeme zobrazovať na LED zobrazovači (DSP1, DSP2 a DSP3). Pri realizácii uvedeného programu využijeme programový modul ZOBRAZX z predchádzajúceho príkladu.

- Časový úsek, 1 s, budeme generovať pomocou čítača 1 s využitím interného zdroja hodinového signálu (Clk_{IO}) s preddeličkou.

Frekvencia hodinového signálu Clk_{IO} je 16 MHz. Výstup preddeličky pre budenie čítača/časovača1 nech je rovný $Clk_{IO}/256=62.500\text{kHz}$. V prípade, že využijeme plný rozsah 16-bitového čítača1, maximálny dosiahnuteľný časový úsek, medzi dvoma pretečeniami bude:

$$1/(62500)*65536 = 1.048576 \text{ s}$$

Je zrejmé, že maximálny časový úsek generovaný 16-bitovým časovačom pri využití hodinového signálu $Clk_{IO} = 16 \text{ MHz}$ s preddeličkou 1:256 je 1.048576 s. Ak chceme aby prerušenie bolo vyvolané každú sekundu, tak výstupný porovnávací register OCR1A nastavíme na hodnotu 62500. Potom keď register TCNT1 dosiahne hodnotu 62500 nastaví sa príznak OCF1A a v prípade ak je povolené odpovedajúce prerušenie bude generovaná žiadosť o prerušenie.

Programová realizácia daného príkladu pozostáva s:

- Povolenie prerušenia na zhodu OCR1A a TCNT1:

```
rjmp    TIM1_COMPA
```

- Nastavenie riadiacich registrov časovača čítača1:

Riadiaci register TCCR1A pre základný režim čítača/časovača1 naplníme hodnotou 0.

Riadiaci register TCCR1B nastavíme na hodnotu 0x04.

Register masky TIMSK prerušenia nastavíme na hodnotu 0x11. Povolené prerušenie na zhodu OCR1A a TCNT1 a pri pretečení časovača0 - (používa modul ZOBRAZX).

- Porovnávací register OCR1A nastavíme na hodnotu 62 500 (0xf424). (OCR1AH nastavíme na hodnotu 0xf4 a OCR1AL na hodnotu 0x24).

- Vytvoríme obslužný program prerušenia TIM1_COMPA. V rámci obsluhy prerušenia inkrementujeme obsah 16-bitového registra temp1:temp a vynulujeme obsah registra TCNT1.

- Prilinkujeme programový modul ZOBRAZX.asm.

Výpis programu:

```
;program: citac11.asm pripocitava kazdu sekundu
;k obsahu registra temp1:temp jednotku a obsah zobrazi na LED DSP.
;vyuziva prerusenien TIM0_OVF a TIM1_COMPA
```

```
.include "m8def.inc"      ; definčný súbor ATmega8
```

```
.def      temp=r16          ;definovanie symbolického mena
                        ;na register r16 sa môžeme
                        ;odvoláva pomocou mena "temp"
```

```
.def      temp1=r17
.def      temp2=r18
```

```

.def    temp3=r19
.def    temp4=r20
.def    temp5=r21
.def    temp6=r22
.cseg
.org    0x0                                ;segment programu
                                           ;hex kód uloži od adresy 0

        rjmp    RESET                    ; skok na štart programu
;nasleduje tabulka vektorov prerušení, zatiaľ ich
;využívať nebudeme, pretože prer. sú globálne zakázané

        reti;   rjmp    EXT_INT0          ;IRQ handler
        reti;   rjmp    EXT_INT1
        reti;   rjmp    TIM2_COMP
        reti;   rjmp    TIM2_OVF
        reti;   rjmp    TIM1_CAPT
        rjmp    TIM1_COMPA                ;pre. od C/C 1
        reti;   rjmp    TIM1_COMPB
        reti;   rjmp    TIM1_OVF
        rjmp    TIM0_OVF                ; prer od C/C 0
        reti;   rjmp    SPI_STC
        reti;   rjmp    USART_RXC
        reti;   rjmp    USART_UDRE
        reti;   rjmp    USART_TXC
        reti;   rjmp    ADSC
        reti;   rjmp    EE_RDY
        reti;   rjmp    ANA_COMP
        reti;   rjmp    TWSI
        reti;   rjmp    SPM_RDY

RESET:
        ldi      temp,high(RAMEND)        ;Nastavenie ukazovateľa zásobníka-SP
        out      SPH,temp                ;na koniec pamäte RAM
        ldi      temp,low(RAMEND)
        out      SPL,temp
;***** Nastavenie smeru portu B*****
        ldi      temp,0b00111001
        out      DDRB,temp                ;výstup je na PB5,PB4 a PB3

        ser      temp                    ;nastavenie temp na 0xff
        out      DDRD,temp                ;celý PORTD bude výstupný
        out      PORTD,temp
        out      PORTB,temp                ;zhasnúť všetky LED

;***** Nastavenie C/C 0   táto časť by mohla byť aj súčasťou
                                           ;programového modulu

ZOBRAZ
        ldi      temp,0x4                ;nastavenie deliaceho pomeru 1:256
        out      TCCR0,temp
        ldi      temp,0x1                ;nastavenie masky prer. len pre C/C0
        out      TIMSK,temp
;*****

        ldi      temp,0                  ; riadiace slovo TCCR1A
        out      TCCR1A,temp
        ldi      temp,0x04                ;riadiace slovo TCCR1B
        out      TCCR1B,temp
        ldi      temp,0x11                ;maska prerušení, povolené
        out      TIMSK,temp                ;TIM0_OVF a TIM1_COMPA
;naplnenie porovnávacieho registra

```

```

        ldi        temp1,0xf4                ;obsah OCR1AH
        ldi        temp,0x24                ;obsah OCR1AL
        out        OCR1AH,temp1
        out        OCR1AL,temp
;*****
        clr        temp
        clr        temp1
        sei                        ;povolenie preruseni
KON:    rjmp       KON

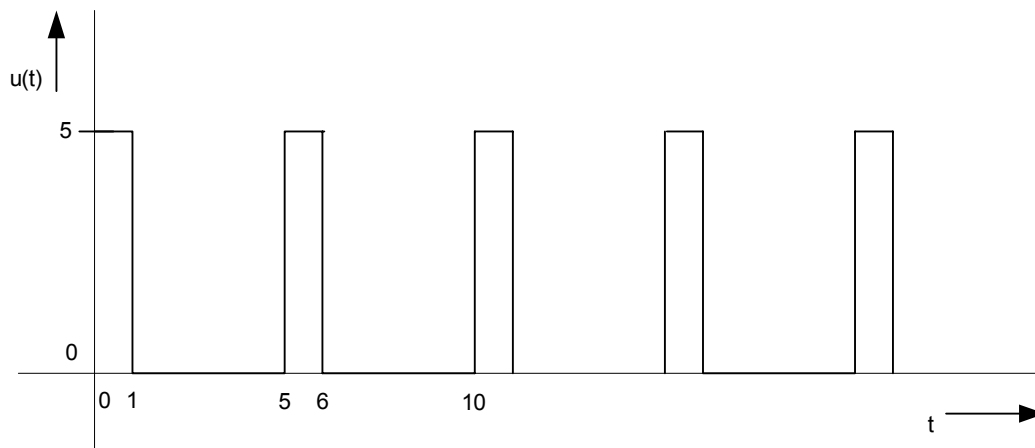
;obslužny podprogram prerusenia na zhodu obsahov registrov TCNT1 a OCR1A
TIM1_COMPA:
        in         temp2,SREG                ;uchovanie obsahu stavoveho slova
        push      temp2
        inc        temp
        brne       POKRACUJ
        inc        temp1
POKRACUJ:
        clr        temp2                    ;nulovanie obsahu TCNT1
        out        TCNT1H,temp2
        out        TCNT1L,temp2
        pop        temp2                    ;navrat obsahu SREG
        out        SREG,temp2
        reti

.include "ZOBRAZX.asm"

```

Časovač/čítač1 vo funkcii PWM

Pomocou čítača/časovača1 generujte na výstupe MCU, OC1A napätie so strednou hodnotou 1V. Použite 10-bitovú PWM. Napätie je možné merať na prepojke P4. Predpokladáme, že úroveň napätia pri logickej hodnote H je cca. 5V a pri logickej hodnote L 0V. Na základe uvedených predpokladov je potrebné na výstupe OC1A generovať priebeh podľa obr.16.



Obr.16 Očakávaný priebeh napätia na výstupe OC1A

Zvolíme režim rychlen 10-bitovej PWM.

Nastavenie registrov:

Riadiaci register čítača/časovača1 TCCR1A nastavíme na hodnotu 0x83.

Riadiaci register čítača/časovača1 TCCR1B nastavíme na hodnotu 0x0a.

Nastavenie porovnávacích registrov: Čítač TCNT1 bude periodicky zvyšovať svoju hodnotu od hodnoty 0 do hodnoty 1023. Pokiaľ jeho hodnota bude nižšia než hodnota uložená v registri OCR1A bude na výstupe OC1A vysoká úroveň. V prípade, že hodnota v registri TCNT1 bude vyššia než v registri OCR1A bude na výstupe OC1A nízka úroveň.

Na základe uvedeného je vhodné, aby v registri OCR1A bola hodnota 1024/5 t.j. približne hodnota 205.

OCR1AH ..0

OCR1AL ..205

RESET:

```
ldi    temp,high(RAMEND)    ;Nastavenie ukazovateľa zásobníka
out    SPH,temp              ;na koniec pamäte RAM
ldi    temp,low(RAMEND)
out    SPL,temp
```

,***** nastavenie používaných V/V

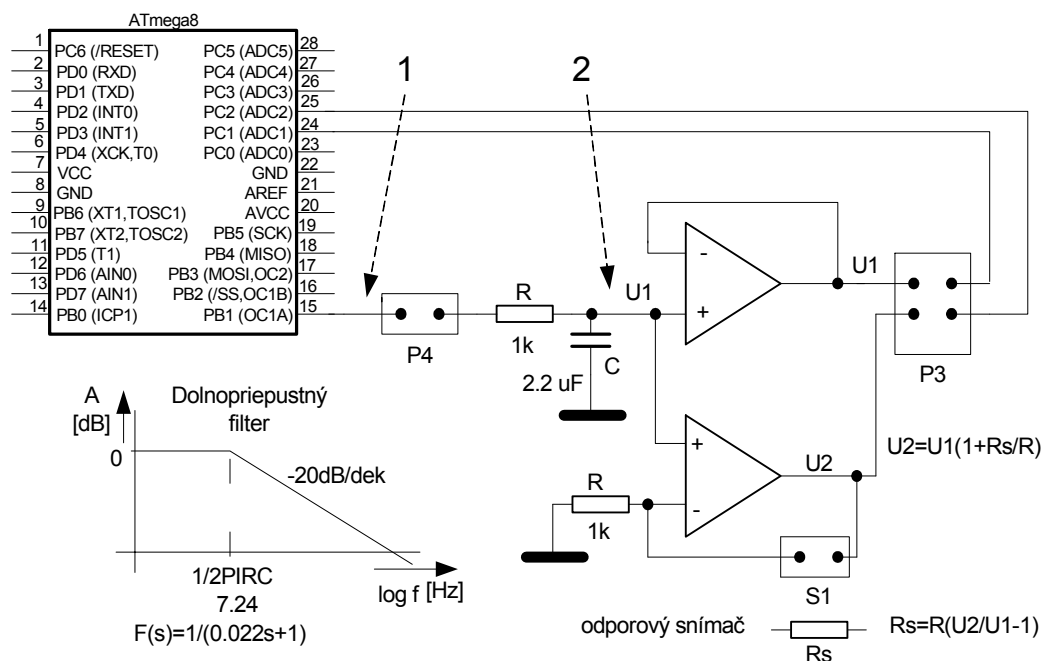
```
ldi    temp,0x83
out    TCCR1A,temp
ldi    temp,0xa
out    TCCR1B,temp
ldi    temp,0                ;naplnenie porovnávacieho registra
ldi    temp1,205
out    OCR1AH,temp
out    OCR1AL,temp1
```

KON: rjmp KON

Určíme opakovaciu frekvenciu PWM:

Frekvencia hodinových impulzov CLKIO je 16 MHz. Nastavenie preddeľičky v riadiacom registri TCCR1B je 8. Použitý je 10-bitový generátor PWM. Potom opakovacia frekvencia PWM bude:

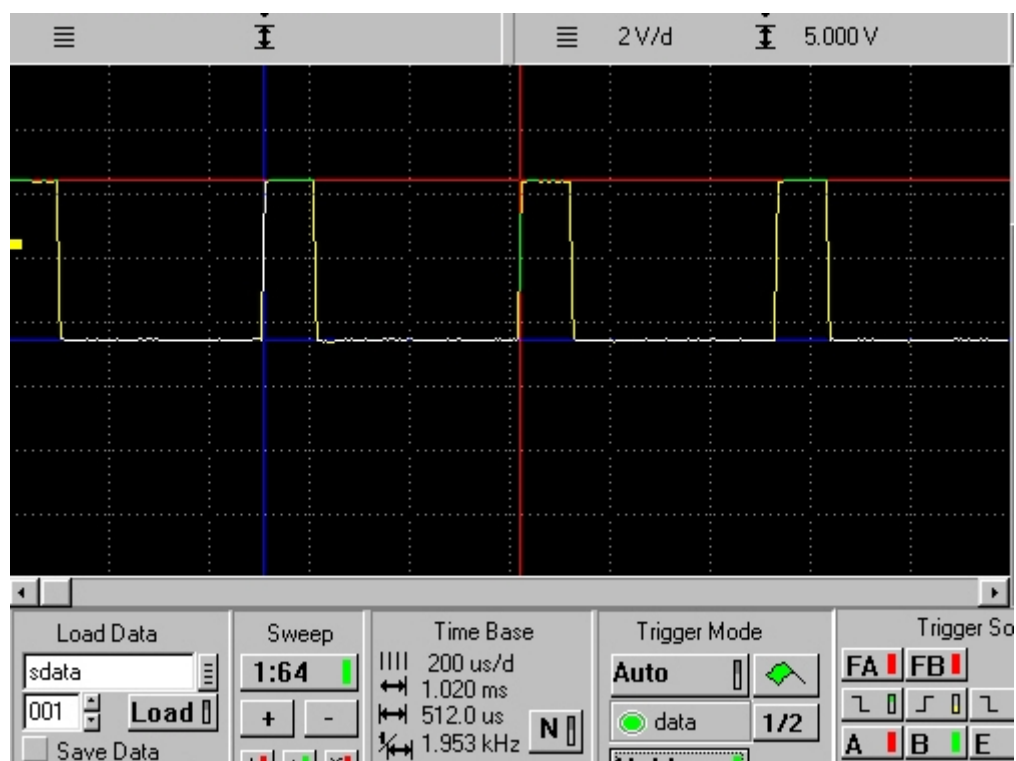
$$16000/8/1024 = 1.953125 \text{ kHz}$$



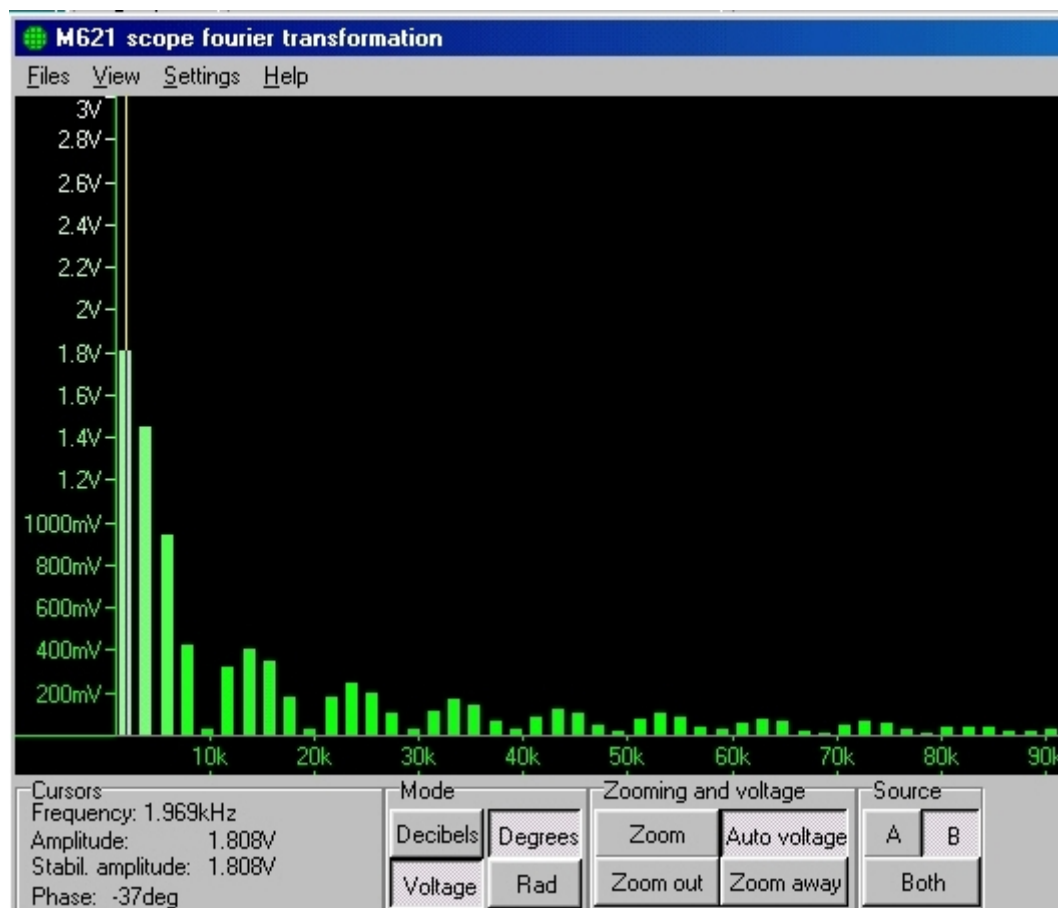
Obr.17 Obvody PWM

Poznamenajme, že zapojenie na obr.17 slúži len na demonštráciu niektorých vlastností PWM signálu. Vzhľadom k unipolárnemu napájaniu použitého OZ bude presnosť filtrovaného signálu, najmä v oblasti malých napätí, nedostačujúca.

Na obr. 18 je uvedený priebeh generovaného napätia v bode 1. Frekvenčné spektrum generovaného napätia je uvedené na obr. 19.

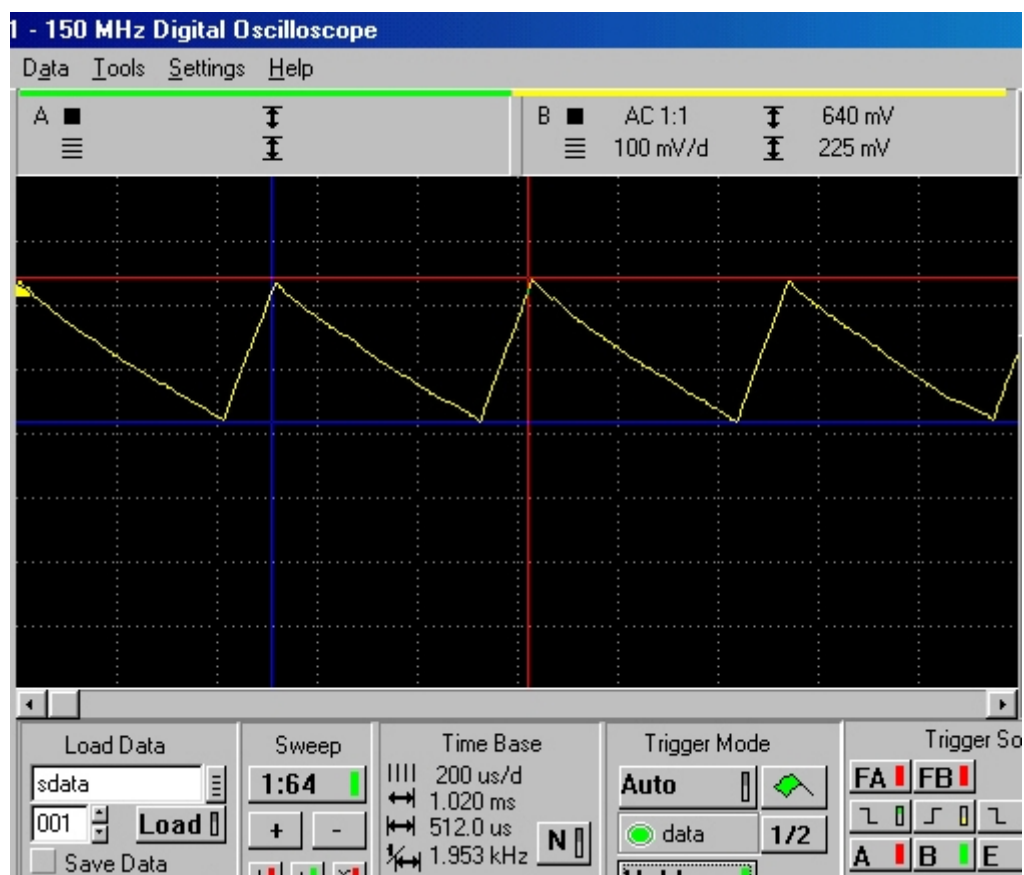


Obr.18 Priebeh generovaného napätia na výstupe PWM, (merací bod 1)

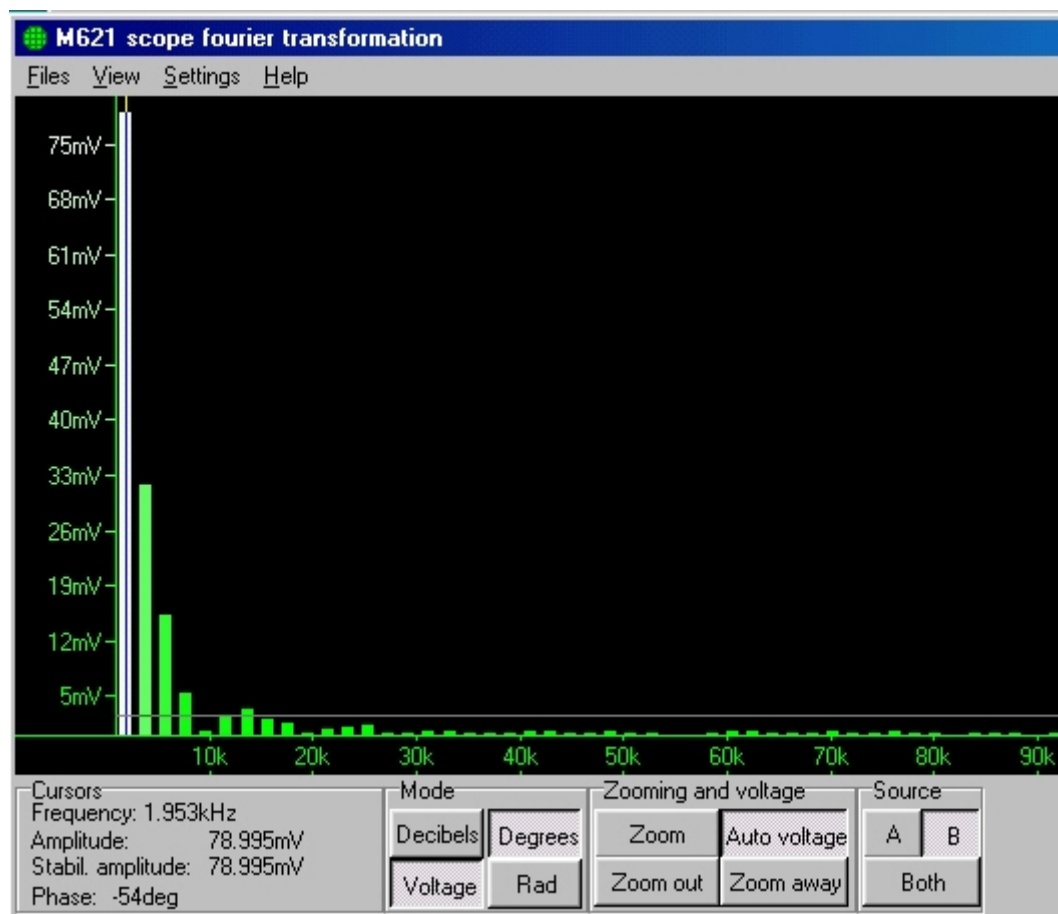


Obr.19 Frekvenčné spektrum signálu z obr.18

Na výstupe dolnopriepustného filtra, (bod.2) dostaneme napätie U_1 , ktorého priebeh je uvedený na obr.19. Frekvenčné spektrum signálu U_1 je uvedené na obr.20.

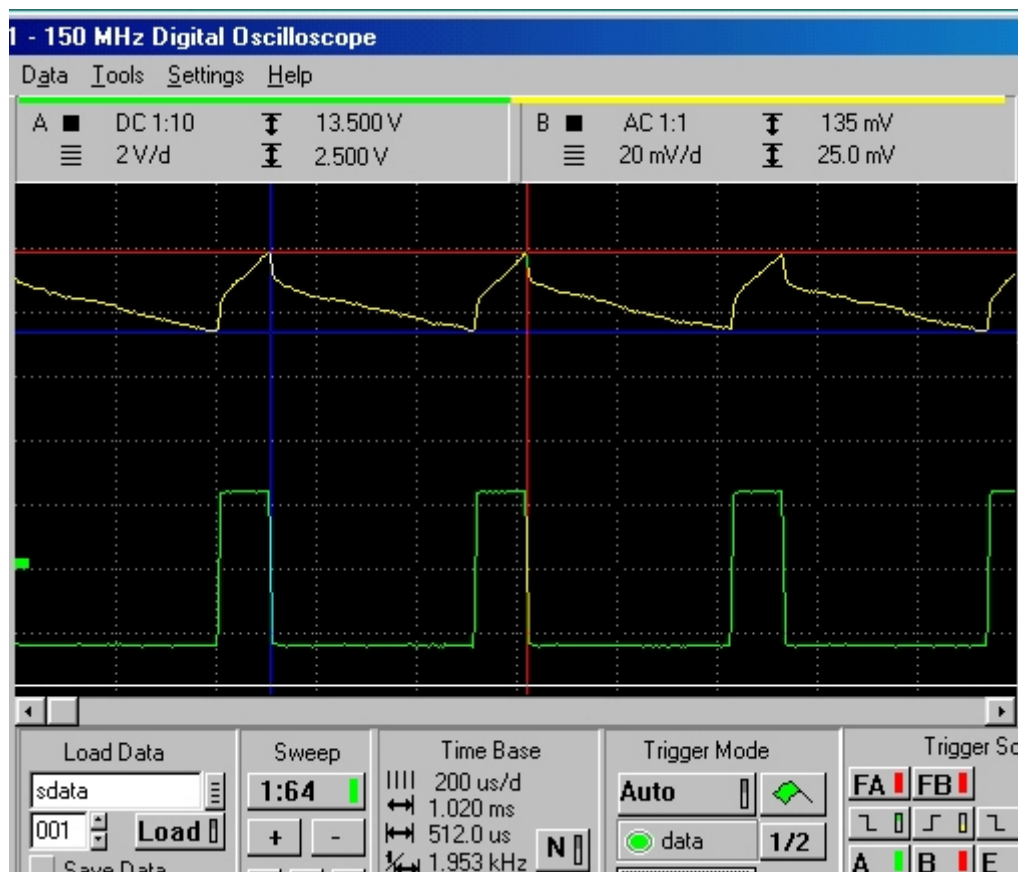


Obr.19 Priebeh napätia na výstupe dolnopiepušného filtra, (merací bod 2)

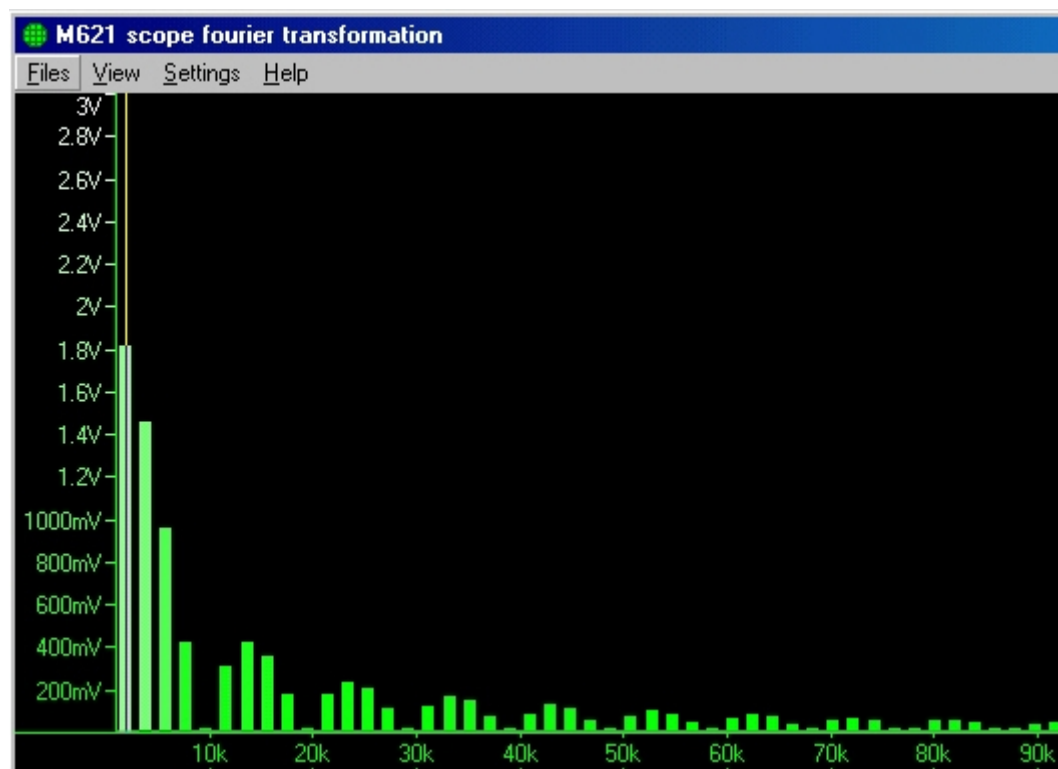


Obr.20 Frekvenčné spektrum signálu z obr.19

Kapacitu kondenzátora C v DPF zväčšíme z pôvodnej hodnoty $2.2\mu\text{F}$ na hodnotu $33\mu\text{F}$. Namerané priebehy napätia PWM a napätia na výstupe filtra sú uvedené na obr.21. Poznamenajme, že rozkmit napätia na výstupe filtra sa zmenšil z pôvodných 220 mV na hodnotu cca. 25mV .



Obr.21 Pribeh napätia na výstupe PWM a dolnopiepušného filtra

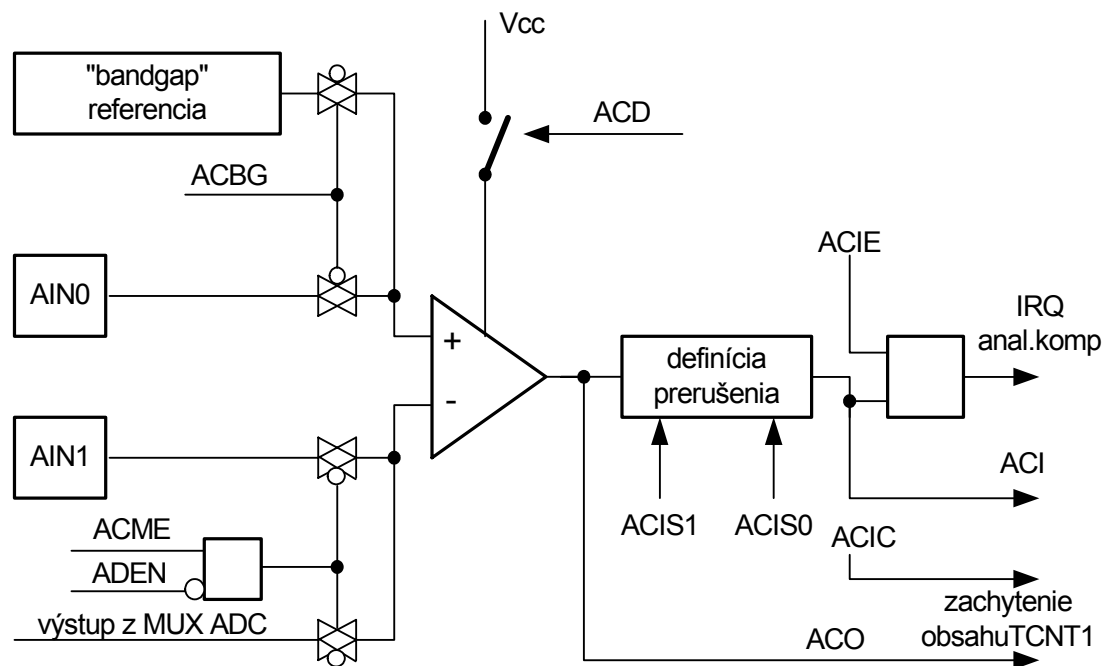


Obr.22 Frekvenčné spektrum signálu z obr.21 (výstup PWM)

5. Analógový komparátor

Jednou z najpoužívanějších integrovaných periférií mikrokontrolérov je analógový komparátor. Analógový komparátor slúži na porovnávanie úrovne napätí privedených na invertujúci a neinvertujúci vstup. Komparátor sa často využíva na kontrolu úrovne napájacieho napätia vlastného mikrokontroléra. Pri náhodnom výpadku napájania – sa pri poklese napájacieho napätia pod definovanú úroveň - vyvolá prerušenie, v rámci ktorého sa potrebné údaje uložia do pamäte typu EEPROM. Po obnovení napájacieho napätia môže beh programu pokračovať v definovanom bode. Použitie komparátora je rôzne a je závislé od konkrétnej aplikácie. Obvod ATmega8 obsahuje jeden analógový komparátor.

Analógový komparátor porovnáva hodnoty napätí na neinvertujúcom vstupe AIN0 a na invertujúcom vstupe AIN1. Ak napätie na neinvertujúcom vstupe je väčšie než napätie na invertujúcom, potom výstup analógového komparátora ACO nadobúda hodnotu log.1. Výstup analógového komparátora môže byť využitý ako strobovací signál pre časovač/čítač1. Výstup komparátora môže generovať žiadosť o prerušenie na nábežnú hranu, zostupnú hranu, prípadne zmenu logickej úrovne na výstupe komparátora. Bloková schéma obvodov komparátora je uvedená na obr.23.



Obr.23 Obvody analógového komparátora

2.8.1 Popis registrov

V/V register špeciálnych funkcií, SFIOR

SFior:

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	ACME	PUD	PSR2	PSR10

Prístup	R	R	R	R	R/W	R/W	R/W	R/W
P. hod.	0	0	0	0	0	0	0	0

Bit 3 – ACME: povolenie multiplexora analógového komparátora, Analog Comparator Multiplexer Enable

Ak bit ACME obsahuje log.1 a A/D prevodník (ADC) je vypnutý (bit ADEN = 0), potom invertujúci vstup analógového komparátora je pripojený na výstup ADC multiplexora. Ak bit ACME obsahuje log.0, potom invertujúci vstup je pripojený na vstup AIN1.

Riadiaci a stavový register analógového komparátora, ACSR

ACSR:

Bit	7	6	5	4	3	2	1	0
Symbol	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
Prístup	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
P. hod.	0	0	N/A	0	0	0	0	0

Bit 7 – ACD: Zákaz analógového komparátora, Analog Comparator Disable

Keď bit ACD obsahuje log.1 je vypnuté napájanie analógového komparátora. Analógový komparátor je možné kedykoľvek zapnúť zápisom log.0 do bitu ACD. Takéto ovládanie komparátora redukuje spotrebu MCU. Pri zmene obsahu bitu ACD musíme zakázať prerušenie od analógového komparátora zapísaním log.0 do bitu ACIE v registri ACSR.

Bit 6 – ACBG: Voľba „bandgap“, Analog Comparator Bandgap Select

Keď bit ACBG obsahuje log.1, potom „bandgap“ referenčné napätie je pripojené na neinvertujúci vstup analógového komparátora. Ak bit ACBG obsahuje log.0, potom neinvertujúci vstup analógového komparátora je pripojený na vstup AIN0.

Bit 5 – ACO: Výstup analógového komparátora, Analog Comparator Output

Výstup analógového komparátora je po synchronizácii privedený priamo na bit ACO. Proces synchronizácie spôsobuje oneskorenie o 1 až 2 hodinové cykly.

Bit 4 – ACI: Príznak prerušenia od analógového komparátora, Analog Comparator Interrupt Flag

Bit ACI sa nastavuje automaticky, keď na výstupe analógového komparátora sa vyskytne udalosť definovaná bitmi ACIS0 a ACIS1. Obsluha prerušenia sa vykoná v prípade, ak sú nastavené na log.1 bity I v SREG a ACIE v ACSR. Bit ACI sa automaticky nastaví na hodnotu log.0 v priebehu realizácie odpovedajúceho prerušovacieho vektora. Bit ACI môže byť vynulovaný aj zápisom log.1.

Bit 3 – ACIE: Povolenie prerušenia od analógového komparátora, Analog Comparator Interrupt Enable

Ak bit ACIE a súčasne aj I-bit v SREG obsahujú hodnotu log.1 prerušenie od analógového komparátora je povolené. V prípade, že bit ACIE obsahuje hodnotu log.0 prerušenie od analógového komparátora bude zakázané.

Bit 2 – ACIC: Povolenie záchytnej funkcie TCNT1 od analógového komparátora, Analog Comparator Input Capture Enable

Ak bit ACIC obsahuje log.1 povoľuje aby záchytná funkcia obsahu časovača/čítača1 bola ovládaná výstupom analógového komparátora. V tomto prípade je výstup analógového komparátora priamo pripojený na vstup záchytnej logiky. Ak bit ACIC obsahuje log.0, potom neexistuje spojenie výstupu analógového komparátora a záchytnej logiky časovača/čítača1.

Bity 1, 0 – ACIS1 a ACIS0: Výber režimu prerušenia od analógového komparátora, Analog Comparator Interrupt Enable

Obsah bitov ACIS1 a ACIS0 určuje aká udalosť na výstupe analógového komparátora môže vyvolať prerušenie, tab.13.

ACIS1	ACIS0	Popis
0	0	Prerušenie na zmenu výstupu komparátora
0	1	Nevyužité
1	0	Prerušenie na zostupnú hranu signálu na výstupe komparátora
1	1	Prerušenie na nábežnú hranu signálu na výstupe komparátora

Tab. 13 Nastavenie bitov ACIS1 a ACIS0

Keď sa mení nastavenie bitov ACIS1 a ACIS0 je potrebné zakázať prerušenie vynulovaním bitu ACIE. V opačnom prípade, môže zmena obsahu bitov ACIS1,0 vyvolať neželanú žiadosť o prerušenie.

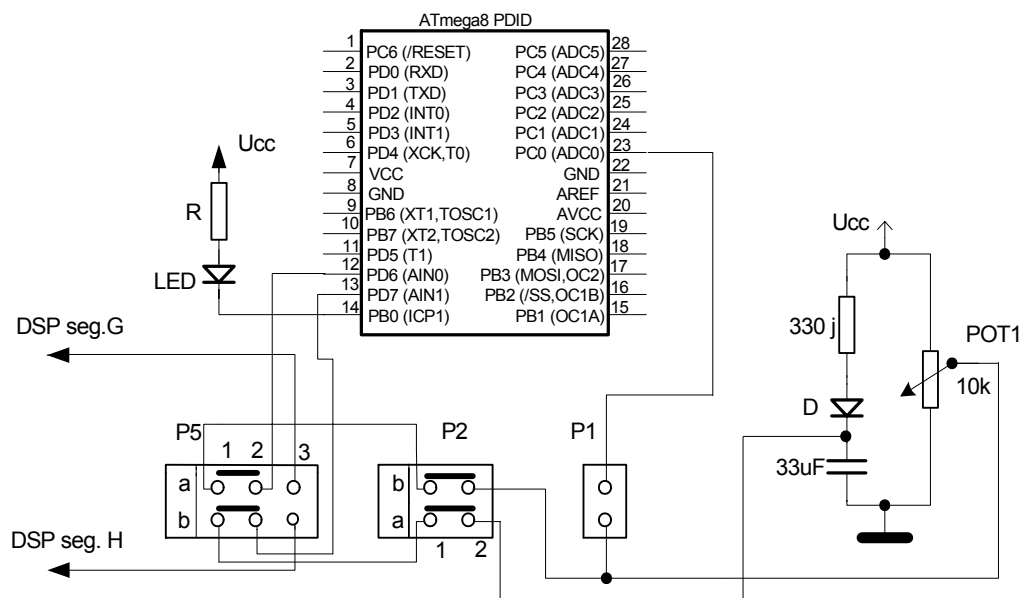
Ako bolo uvedené vyššie, na invertujúci vstup analógového komparátora môžeme pripojiť ľubovoľný vstup analógového multiplexora. Ak je analógový prevodník vypnutý (bit ADEN v ADCSRA nastavený na log.0) a súčasne je povolený vstup od multiplexora (bit ACME v SFIOR nastavený na log.0), potom na invertujúci vstup analógového komparátora je pripojený jeden zo vstupov ADC0 až ADC7. Výber konkrétneho vstupu je realizovaný prostredníctvom bitov MUX2..0 v registri ADMUX. Ak hodnota bitu ACME je log.0, alebo bit ADEN obsahuje hodnotu log.1, potom je na invertujúci vstup pripojený vývod AIN1. V tab. 14 sú uvedené jednotlivé možnosti prepojenia invertujúceho vstupu analógového komparátora.

ACME	ADEN	MUX2..0	Pripojenie invertujúceho vstupu
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6 (len v púzdre TQFP a MLF)
1	0	111	ADC7 (len v púzdre TQFP a MLF)

Tab.14 Pripojenie invertujúceho vstupu analógového komparátora

Príklad 1.

Navrhните program, ktorý zabezpečí, že ak napätie na invertujúcom vstupe bude väčšie než na neinvertujúcom bude svietiť LED dióda. V opačnom prípade svetelná dióda zhasne. Schéma zapojenia vstupov komparátora je uvedená na obr.24.

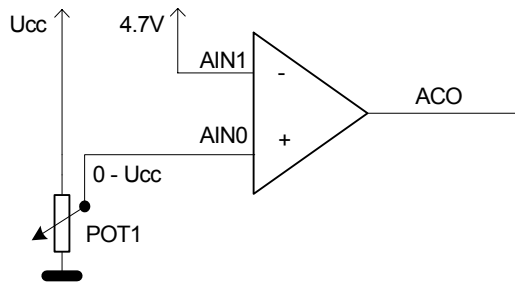


Obr.24 Zapojenie vstupov interného komparátora

Vývody mikrokontroléra PD6 a PD7 sú využité na budenie zobrazovacej jednotky DSP (ovládanie segmentov G a H), prípadne sa využíva ich alternatívna funkcia - vstupy do analógového komparátora (AIN0, AIN1). Prepojky P5 a P2 určujú či je na uvedené vývody privedené napätie z bežca potenciometra POT1- AIN0 a katódy diódy D-AIN1, alebo ich prostredníctvom ovládame príslušné segmenty DSP. V prípade, že budeme využívať interný analógový komparátor je vhodné na jeho vstupy pripojiť analógové napätie. Z uvedeného dôvodu je potrebné modifikovať prepojky P1 a P5 podľa obr.2. Poznamenajme, že uvedené zapojenie vstupov komparátora nie je pre definovanú úlohu najvhodnejšie. Zapojenie je navrhnuté na kontrolu úrovne napájacieho napätia UCC. Táto aplikácia bude popísaná v ďalších príkladoch.

Pre efektívne riešenie zadanej úlohy je vhodné:

- Vstup AIN0 pripojiť na interný zdroj referenčného napätia „bandgap“ typicky cca 1.23 V. Bit ACBG registra ACSR nastavíme na log. 1.
- Vstup komparátora AIN1 pripojiť na bežec potenciometra POT1. Prostredníctvom multiplexera analógových veličín ADMUX. Cez prepojkú P1 privedieme požadované napätie na vstup multiplexera - ADC0. Výstup multiplexera bude pripojený na vstup komparátora AIN1.



Obr.25 Pripojenie vstupov komparátora

V prípade, že napätie na vstupe komparátora AIN1 bude väčšie než na vstupe AIN0 zasvietime svetelnú diódu, PB0 = L. V opačnom prípade PB0=H.

Výpis programu:

Uvedený program je mimoriadne jednoduchý, nevyužíva prerušovací systém. V programovej slučke sa nepretržite testuje hodnota výstup komparátora – ACO.

*;program príklad 4.1 analogovy komparator
; J Micek 19.11.2005*

.include "m8def.inc"

.def temp=r16

*.cseg ;segment programu
.org 0x0 ;ulozit od adresy 0*

```
rjmp RESET
reti; rjmp EXT_INT0 ;IRQ handler
reti; rjmp EXT_INT1
reti; rjmp TIM2_COMP
reti; rjmp TIM2_OVF
reti; rjmp TIM1_CAPT
reti; rjmp TIM1_COMPA
reti; rjmp TIM1_COMPB
reti; rjmp TIM1_OVF
reti; rjmp TIM0_OVF
reti; rjmp SPI_STC
reti; rjmp USART_RXC
reti; rjmp USART_UDRE
reti; rjmp USART_TXC
reti; rjmp ADCC
reti; rjmp EE_RDY
reti; rjmp ANA_COMP
reti; rjmp TWSI
reti; rjmp SPM_RDY
```

RESET:

```
ldi temp,high(RAMEND) ;Nastavenie ukazovateľa zasobníka
out SPH,temp ;na koniec pamäte RAM
ldi temp,low(RAMEND)
out SPL,temp
```

*,***** nastavenie pouzivanych V/V*

```

sbi      DDRB,0      ;nastavenie smeru V/V PB0
sbi      PORTB,0     ;LED zhasne

```

```

;*****test vystupu komparatora

```

```

TEST:  sbis      ACSR,ACO
        cbi      PORTB,0      ;zasviet LED
        sbic     ACSR,ACO
        sbi      PORTB,0      ;zhasni LED
        rjmp     TEST

```

Uvedenú úlohu je možné riešiť s využitím prerušovacieho systému. V tomto prípade je potrebné:

- definovať, ktorá udalosť na výstupe komparátora vyvolá prerušenie,
- vytvoriť obslužný program prerušenia od komparátora (ANA_COMP),
- povoliť prerušenie od komparátora,
- nastaviť globálne povolenie prerušení. Úpravy predchádzajúceho programu sú vyznačené hrubým písmom.

```

;program príklad 4.1b analogovy komparator
; J Micek 19.11.2005

```

```

.include "m8def.inc"

```

```

.def      temp=r16
.cseg      ;segment programu
.org      0x0      ;ulozit od adresy 0

rjmp      RESET
reti;     rjmp      EXT_INT0      ;IRQ handler
reti;     rjmp      EXT_INT1
reti;     rjmp      TIM2_COMP
reti;     rjmp      TIM2_OVF
reti;     rjmp      TIM1_CAPT
reti;     rjmp      TIM1_COMPA
reti;     rjmp      TIM1_COMPB
reti;     rjmp      TIM1_OVF
reti;     rjmp      TIM0_OVF
reti;     rjmp      SPI_STC
reti;     rjmp      USART_RXC
reti;     rjmp      USART_UDRE
reti;     rjmp      USART_TXC
reti;     rjmp      ADCC
reti;     rjmp      EE_RDY
rjmp      ANA_COMP
reti;     rjmp      TWSI
reti;     rjmp      SPM_RDY

```

```

RESET:
ldi      temp,high(RAMEND)      ;Nastavenie ukazovatela zasobnika
out      SPH,temp      ;na koniec pamate RAM
ldi      temp,low(RAMEND)
out      SPL,temp

```

```

;*****nastavenie pouzivanych V/V

```

```

sbi    DDRB,0          ;nastavenie smeru V/V PB0
sbi    PORTB,0         ;LED zhasne
                        ;preruschenie na kazdu zmenu vystupu
                        ;komparatora - default
sbi    ACSR,ACIE       ;povolenie prerusenania od komp.

sei    ;globalne povolenie prerruseni
;*****test vystupu komparatora*****

KON:   rjmp    KON

;*****OBSLUZNY PROGRAM PRERUSENIA*****
ANA_COMP:
    in     temp,SREG    ;uschovanie obsahu SREG
    push  temp
    sbis   ACSR,ACO
    cbi    PORTB,0      ;zasviet LED
    sbic   ACSR,ACO
    sbi    PORTB,0      ;zhasni LED
    pop    temp
    out    SREG,temp
    reti

```

Ďalšie aplikácie využitia analógového komparátora budú uvedené neskôr.

6. Pamäť EEPROM

MCU Atmega8 obsahuje 512 bytov pamäte EEPROM. EEPROM pamäť má oddelený adresný priestor od pamäte SRAM. Pamäť EEPROM umožňuje vykonať až 100 000 zápisových a mazacích cyklov. Pamäť EEPROM je možné programovať v procese programovania obvodu, napr. programátorom AVR ISP voľbou „Program“ v položke „EEPROM“. Tento postup predpokladá, že máte pripravený súbor (štandardne s príponou eep), ktorý chcete umiestniť do pamäte EEPROM. Ak v priebehu písania programu použijete príkaz .ESEG prekladač pre preklade automaticky vytvorí súbor s príponou eep s dátami uvedenými za príkazom. V uvedenom príklade so šiestimi bytami údajov, ktoré pri programovaní pamäte EEPROM uloží od adr.0. Poznamenajme že, riadiť ukladanie dát na príslušné adresy môžeme pomocou známej direktívy .ORG XX.

```

.ESEG
NAVEST:
.DW 0x75ff,1245,0b1010101010000000

```

Je zrejmé, že obsah pamäte EEPROM je tiež možné čítať/zapisovať z aplikačného programu. Prístup procesora k pamäti EEPROM je zabezpečený prostredníctvom vyhradených registrov. Sú to EEPROM adresovacie registre, EEPROM dátový register a EEPROM riadiaci register. Tieto EEPROM registre sú prístupné vo V/V adresnom priestore.

Keď čítame obsah EEPROM, tak CPU je pozastavená na 4 hodinové cykly. Pri zápise do pamäte EEPROM je CPU pozastavená na dva hodinové cykly.

Adresovacie registre EEPROM

EEARH register:

Bit	15	14	13	12	11	10	9	8
Symb.	-	-	-	-	-	-	-	EEAR8
Prístup	R	R	R	R	R	R	R	R/W
Poč.hod.	0	0	0	0	0	0	0	X

EEARL register:

Bit	7	6	5	4	3	2	1	0
Symb.	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Poč.hod.	X	X	X	X	X	X	X	X

Bity 15 až 9

Bity 15 až 9 nie sú v MCU Atmega8 využité.

Bity 8 až 0 – EEAR8...0, EEPROM Address

EEAR register pozostáva z dvoch 8-bitových registrov EEARH a EEARL. Adresa pamäťového miesta je uložená v bitoch EEAR0 až EEAR7. Počiatočná hodnota bitov EEAR8..0 nie je definovaná.

EEPROM údajový register

EEDR:

Bit	7	6	5	4	3	2	1	0
Symb	MSB							LSB
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Poč.hod.	0	0	0	0	0	0	0	0

Bity 7 až 0 – EEDR7..0 EEPROM Data

Pri zápise do pamäte EEPROM register EEDR obsahuje údaje, ktoré majú byť zapísané do pamäťového miesta adresovaného obsahom registrov EEAR. Pri čítaní obsahu EEPROM register EEDR obsahuje údaje, ktoré boli prečítané z pamäťového miesta adresovaného obsahom registrov EEAR.

EEPROM riadiaci register

EECR:

Bit	7	6	5	4	3	2	1	0
Symb.	-	-	-	-	EERIE	EEMWE	EEWE	EERE
Prístup	R	R	R	R	R/W	R/W	R/W	R/W
Poč.hod.	0	0	0	0	0	0	X	0

Bity 7 až 4

Bity 7 až 4 nie sú v MCU Atmega8 využité.

Bit 3 – EERIE, EEPROM Ready Interrupt Enable

Ak je hodnota bitu EEMWE logická 1 a súčasne je nastavený I-bit v SREG, potom je povolené prerušenie EEPROM Ready.

Bit 2 – EEMWE, EEPROM Master Write Enable

Ak bit EEMWE má hodnotu log.1, potom nastavenie bitu EEMWE spôsobí, že v priebehu štyroch hodinových cyklov, budú na vybranú adresu zapísané dáta. Ak EEMWE bude obsahovať log.0, potom nastavenie EEMWE nemá žiadny vplyv. Ak programovými prostriedkami nastavíme bit EEMWE na hod log.1, v priebehu nasledujúcich štyroch hodinových cyklov bude automaticky tento bit vynulovaný.

Bit 1 – EEMWE, EEPROM Write Enable

Bit EEMWE má funkciu strobovacieho signálu pre zápis údajov do pamäte EEPROM. Ak sú korektne nastavené údajový a adresovací registre, potom na zápis obsahu registra EEDR do pamäti EEPROM je potrebné nastaviť bit EEMWE. Pre zápis do pamäti EEPROM musíme vykonať nasledujúcu procedúru:

1. Čakaj pokiaľ EEMWE nemá hodnotu log.0.
2. Čakaj pokiaľ bit SPMEN v registri SPMCR nemá hodnotu log.0.
3. Zapiš hodnotu adresy EEPROM do registrov EEAR.
4. Zapiš údaj do registra EEDR.
5. Zapiš log.1 do bitu EEMWE, ak bit EEMWE v registri EECR je log.0.
6. V priebehu štyroch hodinových cyklov po zápise do EEMWE zapiš log.1 do bitu EEMWE v registri EECR.

Pamäť EEPROM nemôže byť programovaná, pokiaľ CPU zapisuje dáta do pamäte FLASH. Užívateľ musí zabezpečiť aby programovanie pamäte FLASH sa skončilo pred zápisom do pamäte EEPROM. Krok 2 je nutný len v tom prípade, ak užívateľský program používa zavádzací program, ktorý umožňuje CPU programovať pamäť FLASH.

Výskyt prerušenia medzi krokmi 5 a 6 spôsobí, že zápis do pamäte EEPROM neprebehne korektne. Preto sa doporučuje v priebehu zápisu do pamäte EEPROM (kroky 3 až 6) globálne zakázať obsluhu prerušení. Poznamenajme, že po zápise do EEPROM sa bit EEMWE automaticky vynuluje. Užívateľský program môže hodnotu bitu EEMWE testovať a s novým zápisom čakať až pokiaľ bude jeho obsah rovný log.0. Po nastavení bitu EEMWE CPU pozastaví svoju činnosť po dobu dvoch cyklov a pokračuje realizáciou nasledujúcej inštrukcie.

Bit 0 – EERE, EEPROM Read Enable

Bit EERE má funkciu strobovacieho signálu pre čítanie údajov z pamäte EEPROM. Ak je korektne nastavená adresa v registroch EEAR, potom čítanie obsahu adresovanej lokácie pamäte EEPROM je vykonané nastavením bitu EERE. Pri čítaní pamäte EEPROM sa CPU pozastaví po dobu štyroch hodinových cyklov. Potom pokračuje vo výkone nasledujúcej inštrukcie. Užívateľský program by mal pred čítaním EEPROM pamäte kontrolovať hodnotu bitu EEMWE, pretože nie je možné meniť obsah registrov EEAR ani čítať EEPROM pamäť pred skončením operácie zápisu.

Na časovanie prístupu k pamäti EEPROM sa používa kalibrovaný RC oscilátor. Typický programovací čas pamäte EEPROM je rovný 8448 periód hodinového signálu RC oscilátora s frekvenciou 1 MHz, čo je približne 8.5ms.

Zápis Bytu do pamäte EEPROM

Poznamenajme, že adresa EEPROM pamäte na ktorú budeme zapisovať je uložená v dvojici registrov *adrh:adrl*. Zapisovaný údaj je uložený v registri *temp*.

ZAPIS:

```
sbic      EECR,EWE      ; test pripravenosti
rjmp     ZAPIS
out      EEARH,adrh     ; naplnenie adresných registrov
out      EEARL,adrl
out      EEDR,temp      ;naplnenie datoveho registra
cli
sbi      EECR,EEMWE
sbi      EECR,EWE      ;start zapisu
sei
ret
```

Čítanie Bytu z pamäte EEPROM

Poznamenajme, že adresa EEPROM pamäte z ktorej budeme čítať je v dvojici registrov *adrh:adrl*. Čítaný údaj bude uložený v registri *temp*.

CITAJ:

```
sbic      EECR,EWE      ;test pripravenosti
rjmp     CITAJ
out      EEARH,adrh     ; naplnenie adresných registrov
out      EEARL,adrl
sbi      EECR,EERE      ;citaj adresovany Byt pamate EEPROM
in       temp,EEDR
ret
```

Príklad.

Vytvorte program, ktorý bude na DSP zobrazovať počet zapnutí zariadenia. Pri každom zapnutí prečítajte a na DSP zobrazte hodnotu vybraného pamäťového miesta EEPROM pamäte. Inkrementujte jeho hodnotu a uložte znova do pamäte EEPROM. Poznamenajme, že rovnaký efekt ako vypnutie zariadenia bude mať aj reštartovanie systému.

Nenaprogramovaná pamäť EEPROM obsahuje vo všetkých pamäťových miestach hodnotu 0xff. Z toho dôvodu je potrebné prečítaný obsah pred zobrazením upraviť pomocou inštrukcie jednotkový doplnok, pričom namiesto zvyšovania obsahu o hodnotu 1 sa obsah dekrementuje. Uvedený problém je možné riešiť aj takým spôsobom, že pri programovaní obvodu naprogramujeme aj obsah príslušného pamäťového miesta (adr.0) počiatočnou hodnotou, 0x00.

Výpis programu:

```
;program príklad 4.1b zapis a citanie obsahu pamate EEPROM
;po kazdom vypnuti zariadenia, alebo restarte sa inkrementuje
;zobrazovany obsah, a dekrementuje obsah pamate EEPROM na adr.0
; vyuziva rutinu ZOBRAZX
; J Micek 26.11.2005
```

```
.include "m8def.inc"
```

```
.def    temp=r16
.def    temp1=r17
.def    temp2=r18
.def    temp3=r19
.def    adrl=r21
.def    adrh=r22

.cseg                                ;segment programu
.org    0x0                          ;ulozit od adresy 0

rjmp    RESET
reti;   rjmp    EXT_INT0             ;IRQ handler
reti;   rjmp    EXT_INT1
reti;   rjmp    TIM2_COMP
reti;   rjmp    TIM2_OVF
reti;   rjmp    TIM1_CAPT
reti;   rjmp    TIM1_COMPA
reti;   rjmp    TIM1_COMPB
reti;   rjmp    TIM1_OVF
rjmp    TIM0_OVF                    ;Zobrazovanie, obsluha DSP
reti;   rjmp    SPI_STC
reti;   rjmp    USART_RXC
reti;   rjmp    USART_UDRE
reti;   rjmp    USART_TXC
reti;   rjmp    ADCC
reti;   rjmp    EE_RDY
rjmp    ANA_COMP
reti;   rjmp    TWSI
reti;   rjmp    SPM_RDY
```

RESET:

```
ldi     temp,high(RAMEND)           ;Nastavenie ukazovateľa zásobníka
out     SPH,temp                    ;na koniec pamäte RAM
ldi     temp,low(RAMEND)
out     SPL,temp

,***** nastavenie pouzivaných V/V
ldi     temp,0xff                    ;DSP / PORTD...out
out     DDRD,temp
sbi     DDRB,5
sbi     DDRB,4
sbi     DDRB,3

,***** nastavenie časovacia 0
ldi     temp,0x4                     ;nastavenie deliaceho pomeru 1:256
out     TCCR0,temp
ldi     temp,0x1                     ;nastavenie masky prer.
out     TIMSK,temp
ldi     temp1,0
ldi     adrl,0                       ;adresa EEPROM
ldi     adrh,0
```



```

    rcall    CITAJ                ;citaj obsah adr,0 EEPROM
    dec      temp
    rcall    ZAPIS
    inc      temp                ; obnovit precitany obsah
;obsah nenaprogramovanej pamati EEPROM je 0xff, preto vyuzijeme instrukciu com
    com      temp
    sei                      ;globalne povolenie preruseni
                        ;zobrazuj temp, tremp1 na DSP
KON:    rjmp    KON

;***** citanie adr.0 EEPROM do temp

CITAJ:
    sbic     EECR,EEWE           ;test pripravenosti
    rjmp     CITAJ
    out      EEARH,adrh          ; naplnenie adresných registrov adr.0
    out      EEARL,adrl
    sbi      EECR,EERE           ;citaj adresovany Byt pamate EEPROM
    in       temp,EEDR
    ret
;***** zapis obsahu temp do EEPROM na adr.0
ZAPIS:
    sbic     EECR,EEWE           ; test pripravenosti
    rjmp     ZAPIS
    out      EEARH,adrh          ; naplnenie adresných registrov
    out      EEARL,adrl
    out      EEDR,temp           ;naplnenie datoveho registra
    cli
    sbi      EECR,EEMWE
    sbi      EECR,EEWE           ;start zapisu
    sei
    ret

.include "ZOBRAZX.asm"

```

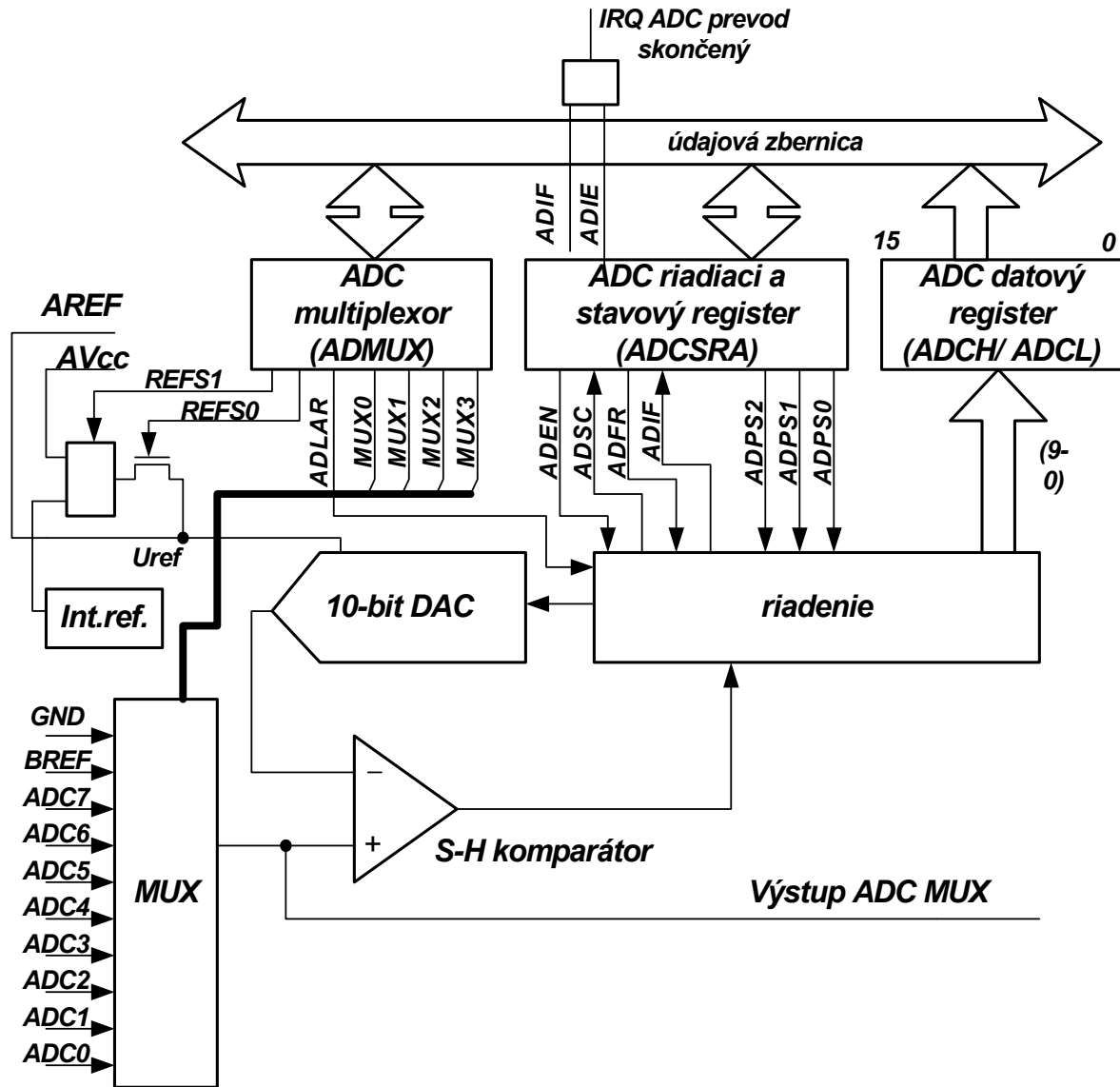
Ďalší príklad: Pri programovaní obvodu zapíšte 3 miestne číslo do pamäte EEPROM (adr. 00az 02). Po pripojení na napájacie napätie zobrazte obsah prvých troch bytov EEPROM na LED DSP.

Pri každom zapnutí inkrementujte obsah prvých troch bytov pamäte EEPROM o hodnotu 1. (počítadlo počtu zapnutí.) Pozor! Programátor bude hlásiť pri verifikácii zápisu do pamäte EEPROM chybové hlásenie. Prečo?

7. Analógovo číslicový prevodník

Často využívanou integrovanou perifériou mikrokontrolérov je analógovo číslicový prevodník. Zabezpečuje styk číslicového systému so spojitým okolím. Mikrokontrolér

ATmega8 obsahuje integrovaný 10-bitový analógovo číslicový prevodník s postupnou aproximáciou a s 8-kanálovým analógovým multiplexorom. Súčasťou prevodníka je vzorkovací zosilňovač, ktorý zabezpečí konštantné napätie na vstupe prevodníka počas doby prevodu. Bloková schéma prevodníka je uvedená na nasledujúcom obrázku.



Obr. 26 Bloková schéma A/D prevodníka

Analógovo číslicový prevodník konvertuje analógové vstupné napätie na 10-bitový digitálny údaj. Minimálna hodnota (0x000) reprezentuje napätie medzi vybraným vstupom a AGND 0V. Maximálna hodnota (0x3FF) je rovná napätiu na vývode AREF mínus 1LSB. Na vývod AREF môže byť pripojené napätie AVCC, alebo interné referenčné napätie 2.56V, pomocou vhodného nastavenia bitov REFS1:0 v registri ADMUX. Pre zvýšenie šumovej imunity je vhodné AREF blokovat' pomocou externej kapacity.

Činnosť A/D prevodníka sa povoľuje nastavením povoľovacieho bitu ADEN v registri ADCSRA.

A/D prevodník generuje 10-bitový výsledok prevodu, ktorý je uložený v údajových registroch ADCH a ADCL. Výsledok prevodu môže byť prečítaný až po ukončení prevodu. Pri čítaní údajov sa musí ako prvý prečítať obsah registra ADCL. Interné ochranné obvody zabezpečia že nedôjde ku zmene obsahu ADCH pokiaľ jeho obsah nebude následne prečítaný. Je zrejmé, že prístup prevodníka k vlastným údajovým registrom je blokovaný, až do okamžiku prečítania obsahu registra ADCH. S touto skutočnosťou je potrebné počítať pri tvorbe obslužných programov.

10-bitový výsledok prevodu môže byť v registroch ADCH a ADCL uložený dvoma nasledovnými spôsobmi:

- dolných 8 bitov výsledku je v registri ADCL a horné 2 bity v dolných 2 bitoch registra ADCH, (počiatočné nastavenie),
- horných 8-bitov výsledku je v registri ADCH a spodné 2 bity v 2 horných bitoch registra ADCL. Táto možnosť sa využíva v prípade, že v danej aplikácii postačí 8-bitová rozlišovacia schopnosť prevodníka. Potom stačí čítať len obsah registra ADCH.

Jeden z uvedených dvoch režimov je možné zvoliť prostredníctvom obsahu bitu ADLAR v registri ADMUX.

Prevodník má dva samostatné vývody na pripojenie napájacieho napätia – AVCC, AGND. Vývod AGND musí byť spojený s GND a úroveň napätia na vývode AVCC sa nesmie líšiť o viac než 0.3V od VCC. Ak je použité externé referenčné napätie toto musí byť v rozsahu AGND až AVCC.

Prevodník môže pracovať v dvoch rôznych režimoch – v režime opakovaný prevod a v režime jednorazový prevod. V jednorazovom režime je každý prevod inicializovaný užívateľským programom. V režime opakovaný prevod je s definovanou periódou vzorkovania automaticky štartovaný prevod, pričom výsledok sa prepisuje do ADC registrov. Pomocou obsahu bitu ADFR v registri ADCSRA je možné voliť jeden z popísaných režimov. Prevod začína zápisom log.1 do štartovacieho bitu ADSC. Bit ADSC bude mať hodnotu log.1, pokiaľ prevod neskončí. Po ukončení prevodu bude automaticky vynulovaný. Ak v priebehu prevodu je prostredníctvom zmeny obsahu registra ADMUX zvolený nový kanál, prevod bude dokončený s pôvodným kanálom a až po jeho skončení dôjde k prepnutiu kanálu. Po ukončení prevodu má prevodník možnosť generovať žiadosť o prerušenie procesora.

Popis registrov

ADC multiplexer select register ADMUX:

ADMUX:

Bit	7	6	5	4	3	2	1	0
Symbol	REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
Prístup	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
P.hod.	0	0	0	0	0	0	0	0

Bit 7:6- REFS1:0: bity pre výber referencie, Reference Selection Bits

Obsah bitov REFS1:0 určuje zdroj referenčného napätia pre A/D prevodník. Ich význam je uvedený v tabuľke 15.

REFS1	REFS0	Zdroj referenčného napätia
0	0	AREF, interné zdroje vypnuté
0	1	AVCC s externou kapacitou
1	0	rezervované
1	1	Interná ref. 2.56V s externou kapacitou

Tab.15 Voľba zdroja referenčného napätia

Bit 5 – ADLAR: umiestnenie výsledku, ADC Left Adjust Result

Pomocou obsahu bitu ADLAR je možné zvoliť umiestnenie výsledku prevodu v registroch ADCL a ADCH. Ak hodnota bitu ADLAR je log.1, potom výsledok prevodu je umiestnený nasledovne: horných 8 bitov v registri ADCH a dva spodné bity v registri ADCL, bity b7,b6. V opačnom prípade je spodných 8 bitov výsledku v registri ADCL a dva najvyššie bity v registri ADCH, v bitoch b0 a b1.

Bity 3:0 – MUX3:0: Výber kanálu, Analog Channel Selection

Hodnota uvedených bitov určuje, ktorý z analógových vstupných kanálov je pripojený na vstup A/D prevodníka, podľa tabuľky 16.

MUX3....0	Vstup
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6, nie pre MCU v puzdre DIP
0111	ADC7, nie pre MCU v puzdre DIP
1000	-
1001	-
1010	-
1011	-
1100	-
1101	-
1110	1.23V (VBG)
1111	0V (GND)

Tab.16 Výber vstupných kanálov A/D prevodníka

ADC Control and Status RegisterA – ADCSRA, riadiaci a stavový register**ADCSRA:**

Bit	7	6	5	4	3	2	1	0
Symbol	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Pristup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P.hod.	0	0	0	0	0	0	0	0

Bit7- ADEN: povolenie činnosti ADC, ADC enable

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

Bit6- 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.

Bit 5-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.

Bit 4-ADIF: príznak prerušenia, ADC interrupt flag

Tento bit je nastavený na log.1 vtedy, keď prevod skončil a údajové registre sú naplnené novým výsledkom. Prerušenie od konca ADC prevodu sa vykoná, ak ADIE bit a I bit stavového registra sú nastavené na hodnotu log.1. Bit ADIF je vynulovaný, keď sa vykoná odpovedajúca obsluha prerušenia. ADIF môže byť tiež vynulovaný zápisom log. 1.

Bit3 – ADIE: povolenie prerušenia, ADC interrupt enable

Ak bit ADIE je nastavený – log. 1 a I bit v S-registri je taktiež nastavený bude povolené prerušenie „ADC prevod ukončený“.

Bit2: 0- ADPS2 – ADPS0: nastavenie preddeličky, ADC prescaler select bits

Uvedené bity určujú deliaci pomer medzi hodinovou frekvenciou (XTAL) a frekvenciou vstupných hodinových impulzov do ADC.

ADSP2	ADPS1	ADPS0	deliaci pomer
0	0	1	1
0	0	0	2
0	1	1	4
0	1	0	8
1	0	1	16
1	0	0	32
1	1	1	64
1	1	0	128

Tab.17 Deliaci pomer

ADC Data Register ADCL a ADCH, údajové registre A/D prevodníka

ADCH: (ADLAR=0)

Bit	7	6	5	4	3	2	1	0
Symbol	-	-	-	-	-	-	ADC9	ADC8
Prístup	R	R	R	R	R	R	R	R
P.hod.	0	0	0	0	0	0	0	0

ADCL: (ADLAR=0)

Bit	7	6	5	4	3	2	1	0
Symbol	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
Prístup	R	R	R	R	R	R	R	R
P.hod.	0	0	0	0	0	0	0	0

ADCH: (ADLAR=1)

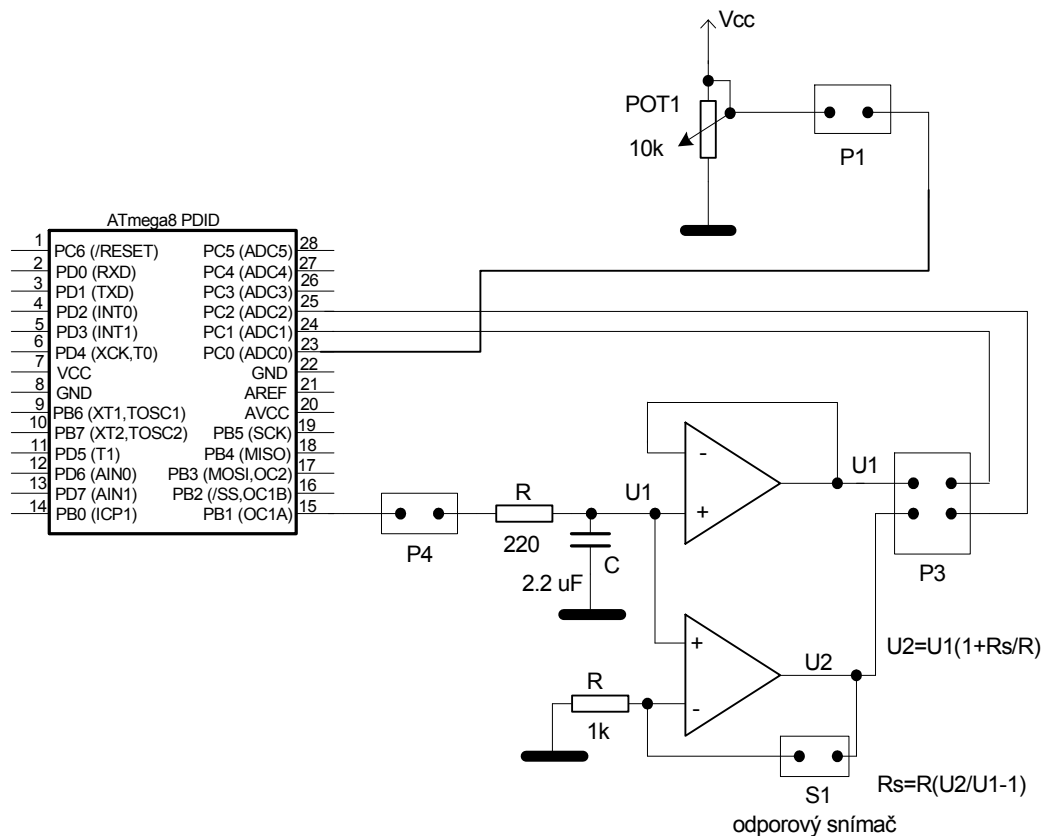
Bit	7	6	5	4	3	2	1	0
Symbol	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P.hod.	0	0	0	0	0	0	0	0

ADCH: (ADLAR=1)

Bit	7	6	5	4	3	2	1	0
Symbol	ADC1	ADC0	-	-	-	-	-	-
Prístup	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
P.hod.	0	0	0	0	0	0	0	0

Po skončení prevodu sa výsledok nachádza v týchto dvoch registroch. V opakovacom režime je dôležité priebežne zaistiť čítanie týchto dvoch registrov. Register ADCL čítame skôr než register ADCH.

Pripojenie vstupov analógovo číslcového prevodníka na doske ATmega8 je uvedené na obrázku 27.



Obr. 27 Pripojenie vstupov analógovo číslicového prevodníka

Poznamenajme, že pri prepojenej prepojke P1 môžeme prostredníctvom POT1 meniť napätie na vstupe ADC0 v rozsahu 0 až V_{cc} , pričom pri využívaní interného zdroja referenčného napätie bude vstupný rozsah 0 až V_{ref} (2.56V).

Príklad:

Vytvorte program, pomocou ktorého budeme každú sekundu realizovať analógovo číslicový prevod napätia na vstupe ADC0 a jeho výsledok v hexa-tvare zobrazovať na DSP. Identifikáciu konca prevodu budeme vykonávať testovaním bitu ADSC v registri ADCSRA. Zdroj hodinových impulzov pre ADC nastavíme pomocou preddeličky na hodnotu cca 200kHz. Režim prevodníka bude nastavený na jednorázový prevod.

Nastavenie registrov:

Register multiplexera ADMUX :

Bity:	b7,6	REF1:0	0 1	V_{cc} ako referenčné napätie
			1 1	interná referencia 2.56V
	b5	ADLAR	0	klasické usporiadanie výsledku
	b3:0	MUX3:0	0000	vybraný kanál ADC0

Riadiaci a stavový register prevodníka ADCSRA:

Bity:	b7	ADEN	1	povolená činnosť ADC
	b6	ADSC	0-1	štart prevodu
	b5	ADFR	0	jednorázový prevod
	b4	ADIF	x	príznak konca prevodu
	b3	ADIE	0	prerušenie zakázané
	b2:0	ADPS2:0	1 1 1	fclk = 16000/64 = 250kHz, nastavenie frekvencie

Výpis programu:

```

;program príklad AD prevodník
;prevádza napätie zo vstupu ADC0 a jeho hodnotu periodicky
;zobrazuje na DSP
; vyuziva rutinu ZOBRAZX
; J Micek 26.11.2005

.include "m8def.inc"

.def    temp=r16
.def    temp1=r17
.def    temp2=r18
.def    temp3=r19
.def    temp4=r21
.def    temp5=r22

.cseg                                ;segment programu
.org    0x0                          ;ulozit od adresy 0

rjmp    RESET
reti;   rjmp    EXT_INT0             ;IRQ handler
reti;   rjmp    EXT_INT1
reti;   rjmp    TIM2_COMP
reti;   rjmp    TIM2_OVF
reti;   rjmp    TIM1_CAPT
reti;   rjmp    TIM1_COMPA
reti;   rjmp    TIM1_COMPB
reti;   rjmp    TIM1_OVF
rjmp    TIM0_OVF                    ;Zobrazovanie, obsluha DSP
reti;   rjmp    SPI_STC
reti;   rjmp    USART_RXC
reti;   rjmp    USART_UDRE
reti;   rjmp    USART_TXC
reti;   rjmp    ADCC
reti;   rjmp    EE_RDY
reti;   rjmp    ANA_COMP
reti;   rjmp    TWSI
reti;   rjmp    SPM_RDY

RESET:
ldi      temp,high(RAMEND)           ;Nastavenie ukazovateľa zásobníka
out      SPH,temp                   ;na koniec pamäte RAM
ldi      temp,low(RAMEND)
out      SPL,temp

;***** nastavenie pouzivanych V/V
ldi      temp,0xff                   ;DSP / PORTD...out
out      DDRD,temp

```



```

sbi        DDRB,5
sbi        DDRB,4
sbi        DDRB,3

sbi        DDRB,0                ;nastavenie smeru V/V PB0

;***** nastavenie casovaca 0, pre zobrazovanie

ldi        temp,0x4                ;nastavenie deliaceho pomeru 1:256
out        TCCR0,temp
ldi        temp,0x1                ;nastavenie masky prer.
out        TIMSK,temp

ldi        temp,0x40                ;nastavenie registra ADMUX
out        ADMUX,temp

sei                    ;globalne povolenie preruseni
                    ;zobrazuje temp, temp1 na DSP
ldi        temp,0x87                ;nastavenie riadiaceho registra
out        ADCSRA,temp

KON:
rcall      PREV
rcall      CAKAJ
rjmp       KON

;***** prevod

PREV:
sbi        ADCSRA,ADSC                ;start prevodu
TEST:
sbic       ADCSRA,ADSC                ;test konca prevodu
rjmp       TEST
in         temp,ADCL                ;citanie vysledku
in         temp1,ADCH
ret

;***** zapis obsahu temp do EEPROM na adr.0

CAKAJ:
ldi        temp5,0x70                ;cca 1s
C3: ldi        temp4,0xff
C2: ldi        temp2,0xff
C1: dec       temp2
brne       C1
dec        temp4
brne       C2
dec        temp5
brne       C3
ret

.include "ZOBRAZX.asm"

```

Príklad 2

Každú sekundu zmerajme hodnotu odporu pripojeného na svorkovnicu S1. Odmeranú hodnotu je potrebné zobrazit' na DSP. Zapojenie meracieho systému je uvedené na obr.3.

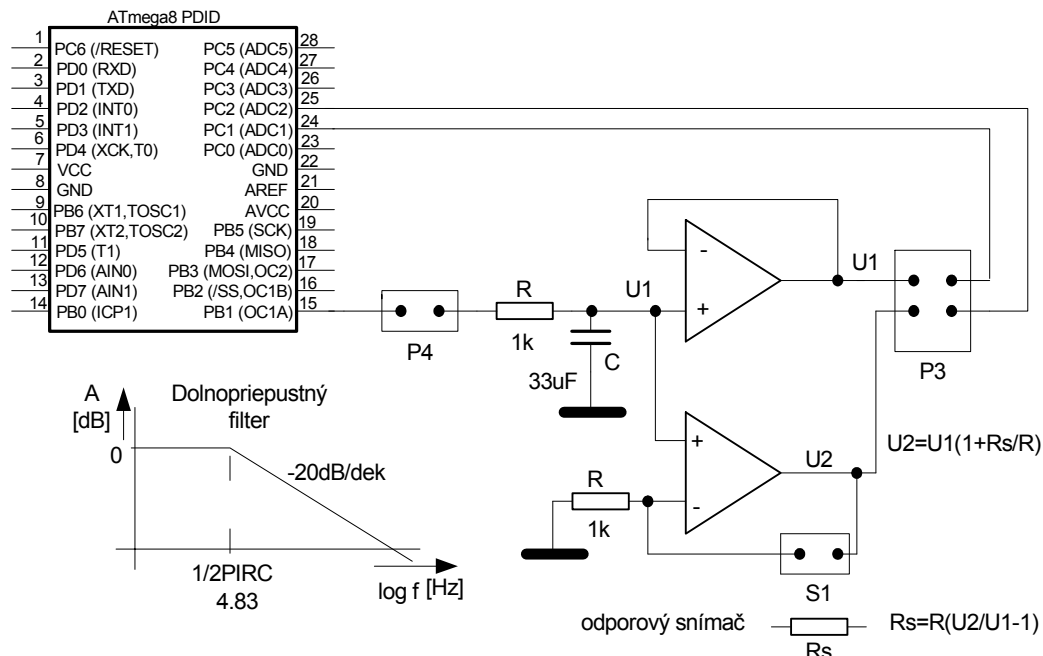
Pretože vstupný rozsah prevodníka a súčasne aj napájanie OZ je ohraničené hodnotou AV_{CC} je potrebné určiť hodnotu napätia U_1 vo vzťahu k predpokladanej hornej hranici meraných odporov. Pre jednoduchosť predpokladajme, že maximálna hodnota meraného odporu bude 7 kohmov. Potom generované napätie U_1 určíme na základe vzťahu:

$$U_1 \leq \frac{U_2}{1 + R_{s_{\max}} / R}$$

Ak predpokladáme, že vstupný rozsah prevodníka je rovný AV_{CC} , t.j. 5V a R_s je maximálne 7 k Ω , potom pre U_1 platí:

$$U_1 \leq 0.625$$

Pre hodnotu U_1 zvolíme hornú hranicu 0.625V. Z uvedeného vyplýva, že pri použití 10-bitovej PWM bude hodnota v registroch OCR1AH:OCR1AL rovná 1024/8, alebo 10248(0.625/5)=128, (0x80). Pri generovaní rýchlej PWM použijeme najvyššiu možnú opakovaciu frekvenciu, čím minimalizujeme zvlnenie napätia U_1 , na výstupe filtra.



Obr.28 Meranie hodnoty odporu odporového snímača R_s .

Hodnotu meraného odporu R_s vypočítame na základe vzťahu:

$$R_s = R(U_2 / U_1 - 1),$$

kde:

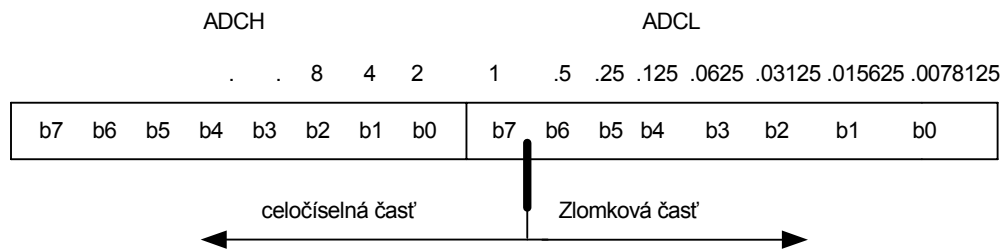
R je odpor 1k Ω

U_1 je generované napätie charakterizované binárnym ekvivalentom 0b10000000, (128).

U_2 je odmerané napätie v rozsahu 0 až 5V, s výsledkom prevodu v rozsahu 0 až 1023.

Ak ekvivalent napätia U_1 je rovný celočíselnej mocnine 2, ($128 = 2^7$), je operácia delenia jednoducho realizovateľná posuvom binárneho ekvivalentu U_2 o 7 miest do prava. Potom

celočíselnú časť výsledku charakterizujú tri horné bity a desatinnú časť obsah spodných siedmich bitov, obr. 29.



Obr.29 Interpretácia výsledku prevodu

Od celočíselnej časti je potrebné podľa vzťahu na výpočet R_s odčítať hodnotu 1. a zobrazit' na ju DSP3, pritom nezabudnime aktivovať príslušnú desatinnú čiarku-segment H na DSP3. V ďalšom kroku je potrebné navrhnuť algoritmus prevodu zlomkovej časti na výsledku na desatinnú časť v kóde BCD. Príklad možného riešenia je nasledovný:

```
cli
clc
rol      temp      ;uprava vysledku
rol      temp1     ; v temp1 su horne 3 bity vysledku
dec      temp1     ; temp1 = temp1-1
,*****uprava zlomkovej casti
clr      temp3
clr      temp4
sbrcl    temp,7
ldi      temp4,50
add      temp3,temp4
clr      temp4
sbrcl    temp,6
ldi      temp4,25
add      temp3,temp4
clr      temp4

sbrcl    temp,5
ldi      temp4,13
add      temp3,temp4
clr      temp4

sbrcl    temp,4
ldi      temp4,6
add      temp3,temp4
clr      temp4

sbrcl    temp,3
ldi      temp4,3
add      temp3,temp4
clr      temp4

sbrcl    temp,2
ldi      temp4,2
add      temp3,temp4
clr      temp4
```

```

        sbrc          temp,1
        ldi           temp4,1
        add           temp3,temp4
        clr           temp4

        cpi           temp3,100
        brne         P1
        inc           temp1
        clr           temp3

P1:      rcall         BCD          ;prevod zlomkovej casti na BCD

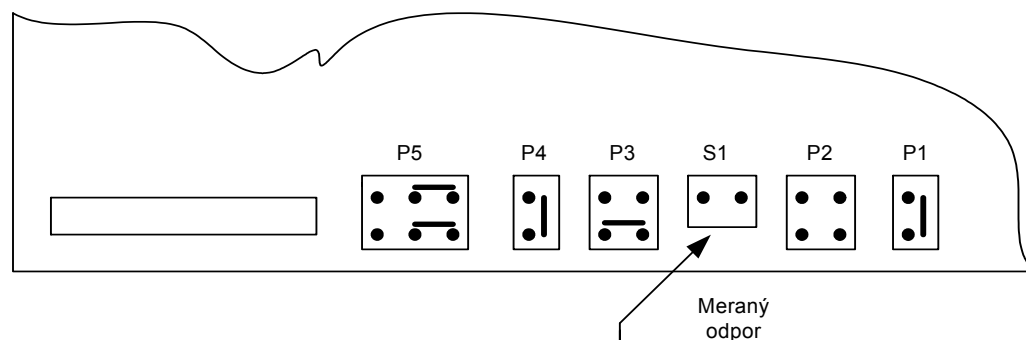
        sei

;*****prevod bin BCD vstup temp3, vystup temp (0 az 99)
BCD:
        clr          temp4
        clr          temp
BCD3:   cpi           temp3,0
        breq         BCD1
        inc          temp
        cpi          temp,10
        brne         BCD2
        clr          temp
        inc          temp4
BCD2:   dec          temp3
        rjmp         BCD3
BCD1:   andi          temp4,0x0f
        swap         temp4
        or            temp,temp4
        ret

```

Poznamenajme, že zobrazovaný výsledok bude teoreticky v rozsahu 0.00 až 7.00 KΩ. Vo vývojovom module sú použité bežné operačné zosilňovače LM358. Nemajú vlastnosti OZ „rail-to-rail“, preto sa v tejto jednoduchšej aplikácii značne obmedzí použiteľný rozsah meraných hodnôt odporu R_s na rozsah cca do 4.5 kΩ.

Modifikácia prepojení prvkov pre riešenie príkladu 2 je uvedená na obr.30.



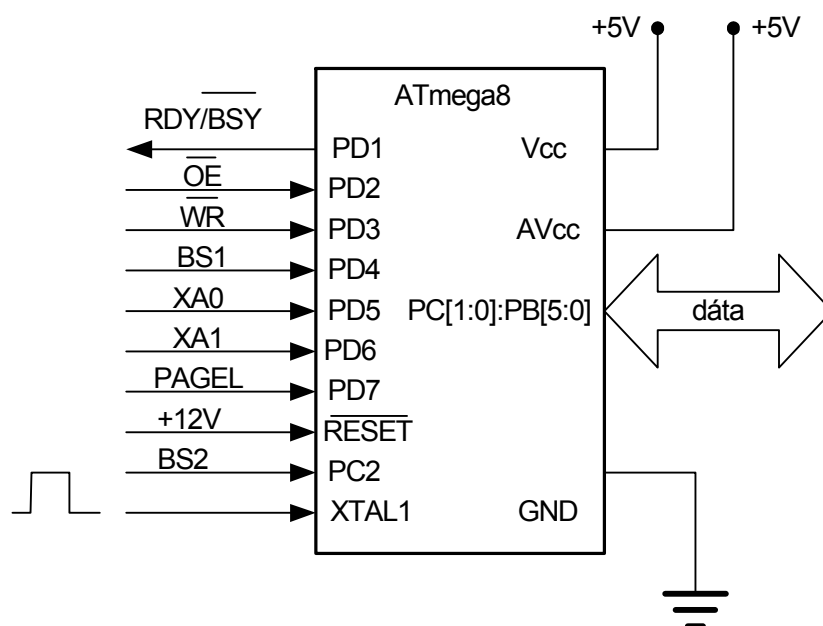
Obr.30 Modifikácia prepojení prvkov modulu ATmega8

8. A na záver

Ak v priebehu experimentovania programovania sa neopatrnosťou ATmega 8 podarilo zakázať RESET, prípadne ISP nezostáva vám nič iného len MCU uviesť do pôvodného stavu pomocou paralelného programátora (to nie je ani programátor) , ktorého najjednoduchšia verzia je uvedená v nasledujúcom texte obr.32.

Paralelné programovanie MCU

V tejto časti sú uvedené možnosti paralelného programovania FLASH pamäte programu, EEPROM pamäte dát, „Lock“ bitov a „Fuse“ bitov mikrokontroléra ATmega8. Na obr.31 sú znázornené vývody MCU, ktoré sú využívané v procese paralelného programovania.



Obr.31 Paralelné programovanie MCU

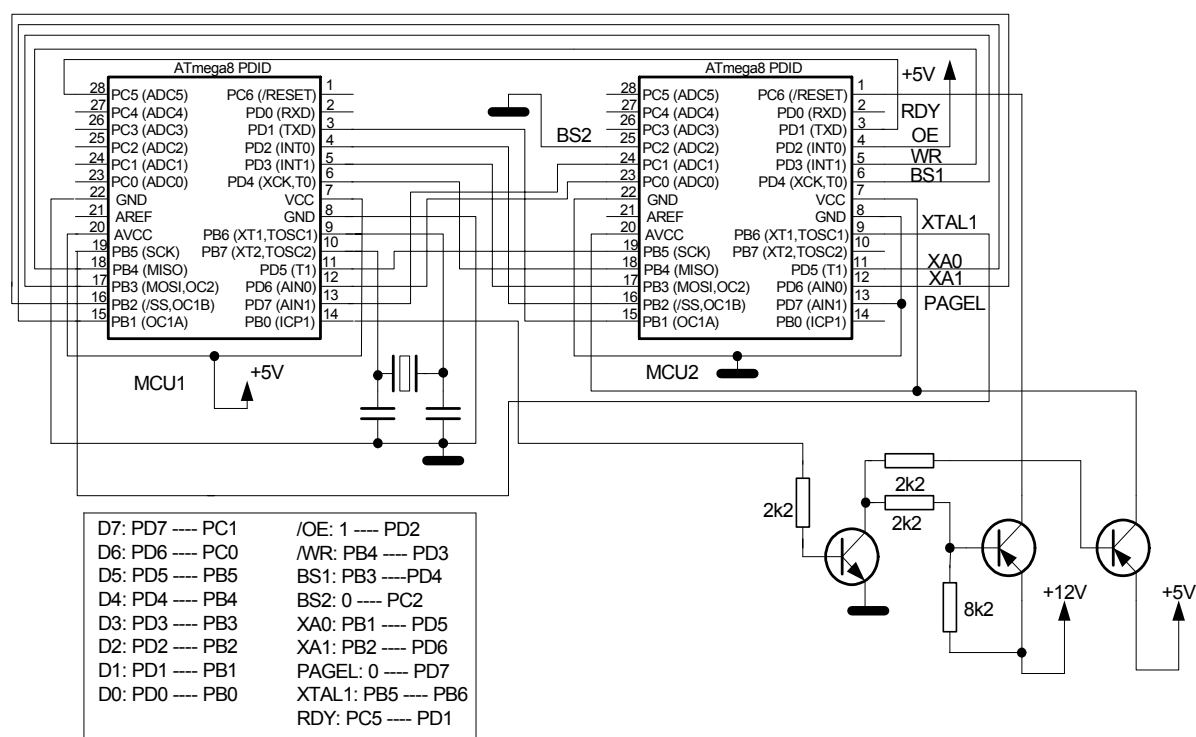
Význam jednotlivých signálov pri paralelnom programovaní je uvedený v tab.1.

Označenie	Vývod	Vstup/Výstup	Funkcia
RDY//BSY	PD1	Výstup	0: MCU zaneprázdnený programovaním 1: MCU pripravené na prijatie nového príkazu
/OE	PD2	Vstup	Povolenie výstupu (aktívna „0“)
/WR	PD3	Vstup	Impulz zápisu (aktívna „0“)
BS1	PD4	Vstup	Výber Bytu 1 („0“ určuje spodný Byte, „1“ určuje horný Byte)
XA0	PD5	Vstup	XTAL bit 0
XA1	PD6	Vstup	XTAL bit1

PAGEL	PD7	Vstup	Naplnenie stránky pamäte
BS2	PC2	Vstup	Výber Bytu 2 („0“ určuje spodný Byte, „1“ určuje horný Byte)
DATA	PC1:0 PB5:0	Vstup/výstup	Obojsmerná údajová zbernica výstupná ak /OE je „0“)

Tab.1 Význam signálov MCU v režime paralelného programovania

Podrobný popis postupu programovania MCU v paralelnom režime je uvedený v technickej dokumentácii výrobcu.



Obr.32 Paralelné programovanie MCU

Výpis programu pre ATmega8

;program "paralelpr" umožňuje preprogramovať horný FUSE Byt
;v prípade, že v režime seriového programovania sme zakázali
;funkciu RST. V tomto prípade sme totiž stratili možnosť použiť
;seriový programátor. Jediná možnosť je preprogramovať horný
;FUSE Byt v paralelnom režime.
;Potrebujeme: 1ks MCU s možnosťou programovania ATmega8,
;1 krystal v uvedenom prípade 12MHz (frekvencia nie je kritická), 3 tranzistory a pár ;odporov. Zapojenie je na obr.32

```
.include "m8def.inc" ; definíčný súbor ATmega8
.def temp=r16 ; definovanie symb. mena
.def temp1=r17
.def temp2=r18

.macro CAKAJ ; definujeme MAKRO bez parametrov
;*****čakacia slučka cca 82*255*255*3*62.5 ns 0.999759 s

ldi temp,82 ; reg.temp naplníme hodnotou 82
```

```

C3:          ldi          temp1,0xff      ;reg.temp1 naplníme hodnotou 0xff=255
C2:          ldi          temp2,0xff      ;reg.temp2 naplníme hodnotou 0xff
C1:          dec          temp2
           brne          C1
           dec          temp1
           brne          C2
           dec          temp2
           brne          C3

.endmacro                                         ;koniec MAKRA

.cseg                                             ;segment programu
.org        0x0                                ;hex kód uložiť od adresy 0
rjmp        RESET                             ; skok na štart programu
;nasleduje tabuľka vektorov prerušení, zatiaľ ich
;využívať nebudeme, pretože budú globálne zakázané
;prerušená.
           rjmp        PRER          ;      rjmp        EXT_INT0          ;IRQ handler
           rjmp        PRER          ;      rjmp        EXT_INT1
           rjmp        PRER          ;      rjmp        TIM2_COMP
           rjmp        PRER          ;      rjmp        TIM2_OVF
           rjmp        PRER          ;      rjmp        TIM1_CAPT
           rjmp        PRER          ;      rjmp        TIM1_COMPA
           rjmp        PRER          ;      rjmp        TIM1_COMPB
           rjmp        PRER          ;      rjmp        TIM1_OVF
           rjmp        PRER          ;      rjmp        TIM0_OVF
           rjmp        PRER          ;      rjmp        SPI_STC
           rjmp        PRER          ;      rjmp        USART_RXC
           rjmp        PRER          ;      rjmp        USART_UDRE
           rjmp        PRER          ;      rjmp        USART_TXC
           rjmp        PRER          ;      rjmp        ADCC
           rjmp        PRER          ;      rjmp        EE_RDY
           rjmp        PRER          ;      rjmp        ANA_COMP
           rjmp        PRER          ;      rjmp        TWST
           rjmp        PRER          ;      rjmp        SPM_RDY

RESET:
           ldi          temp,high(RAMEND);Nastavenie ukazovateľa zásobníka-SP
           out          SPH,temp          ;na koniec pamäte RAM
           ldi          temp,low(RAMEND);poznamenajme,že zásobník zatiaľ
           out          SPL,temp          ;nepotrebujeme
;***** Nastavenie smeru portu B*****
           ldi          temp,0
           out          PORTB,temp
           sbi          PORTB,4
           out          PORTD,temp
           out          DDRC,temp
           ldi          temp,0xff          ;0x01
           out          DDRB,temp          ;do DDRB,0 log.1,PB,0 výstup          out
           out          DDRD,temp
;*****
           CAKAJ
           sbi          PORTB,0          ;pripojenie 5V a 12V
           nop
           nop
           nop
           nop
           rcall        PRIK
           rcall        DATA
           sbi          PORTB,3
           cbi          PORTB,4
           nop
           nop
           nop
           nop
           sbi          PORTB,4
           nop
           nop
           nop
           sbis         PINC,5          ;cakaj na nastavenie RDY
           rjmp        X
           cbi          PORTB,0          ;odpojenie 5V a 12V
KON:          rjmp        KON

```

```

PRIK:      sbi      PORTB,2
            ldi      temp,0x40      ;Prikaz
            out      PORTD,temp
            nop
            nop
            nop
            nop
            sbi      PORTB,5      ;Xtal1
            nop
            nop
            nop
            nop
            cbi      PORTB,5
            cbi      PORTB,2
            ret

DATA:      sbi      PORTB,1
            ldi      temp,0xd9      ;horny FUSE Byte
            out      PORTD,temp
            nop
            nop
            nop
            nop
            sbi      PORTB,5      ;Xtal1
            nop
            nop
            nop
            nop
            cbi      PORTB,5
            cbi      PORTB,1
            ret

PRER:      reti

```

Toto je len druhá predbežná časť textov o AVR. Prvá obsahuje technický popis, pričom táto druhá mala obsahovať príklady.

Príklady uvedené v tomto texte sú veľmi jednoduché školské príklady, ktoré sa opierajú o demonštračný modul ATmega8, vyvinutý na KTK. V ďalšom by bolo vhodné doplniť text o vybrané príklady, najmä z oblasti sériovej komunikácie - SPI, TWI, a selfprogramingu. Možnosti a obmedzenia MCU by bolo vhodné demonštrovať na viacerých konkrétnych aplikáciách, ktoré boli v priebehu predchádzajúcich rokov úspešne vyvinuté na našom pracovisku. Pokiaľ by mal niekto chuť spraviť z tohto textu čitateľnú publikáciu rád prispejem radou a pomocou; Možno v rámci projektovania v inžinierskom štúdiu št. programu „Počítačové inžinierstvo“.

micek@frtk.fri.utc.sk

PRÍLOHY

Popis modulu Atmega8

Demonštračný modul ATmega8 slúži na experimentálne overenie metód a algoritmov implementovaných do MCU ATmega8 a programovanie vybraných obvodov ATMEL AVR

Demonštračný modul obsahuje dva mikrokontroléry: ATMEL- ATmega8 a AT90S2313.

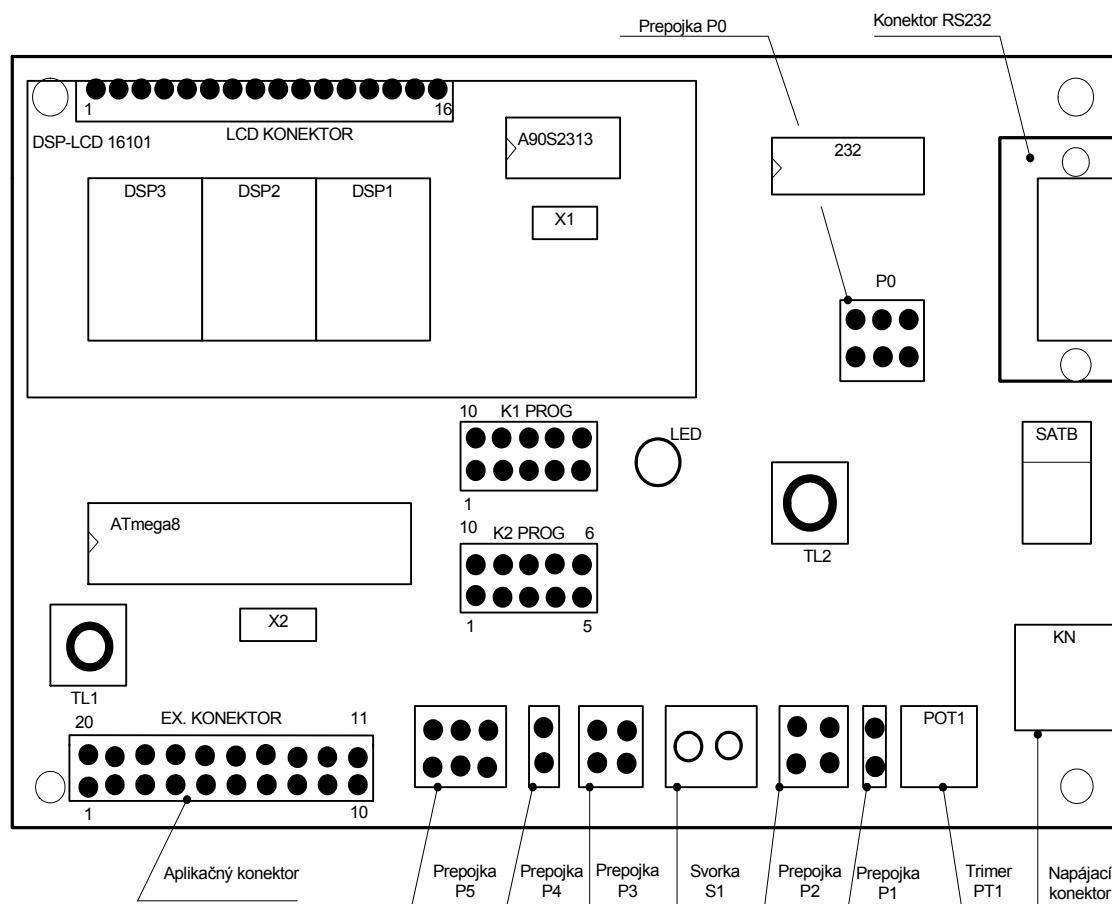
Mikrokontrolér AT90S2313 slúži výhradne na programovanie mikrokontrolérov prostredníctvom rozhrania ISP. Firmware mikrokontroléra zabezpečuje kompatibilitu s integrovaným vývojovým prostredím AVR Studio a dovoľuje programovať širokú škálu obvodov ATMEL AVR.

Druhý mikrokontrolér ATmega8 slúži na overenie a demonštráciu činnosti základných princípov práce mikrokontroléra a riešenie jednoduchých aplikačných úloh. Na rozširujúcom konektore sú vyvedené dôležité signály mikrokontroléra, čo umožňuje vývoj a overenie ďalších neštandardných aplikácií.

Konfiguráciu modulu je možné meniť pomocou prepajovacích vývodov (prepajok) P0 až P7.

Modul môže byť napájaný zo sieťového adaptéra s napätím v rozsahu 8 až 15V/120mA. V napájacích obvodoch je zapojená ochranná dióda proti prepólovaniu napájacieho napätia a 5-voltový stabilizátor.

Na obr.1 je zobrazené rozmiestnenie dôležitých komponentov modulu.



Obr. 1 Rozmiestnenie základných komponentov modulu

označenie	Popis
Konektor RS232	9-pinový konektor sériového rozhrania RS232
KN	Konektor na pripojenie sieťového adaptéra 8-20V/100mA
K1 PROG	Programovací konektor programátora AVR
K2 PROG	Programovací konektor ATmega8. Pripojením s K1 je možné sériovo programovať MCU ATmega8.
Aplikačný konektor	Aplikačný konektor na pripojenie modulu s externými obvodmi.
LCD konektor	16-pinový konektor na pripojenie LCD zobrazovacej jednotky DEM 16101.
TI1	Tlačidlo na reštartovanie MCU ATmega8.
TI2	Tlačidlo pripojené na vývod MCU ATmega8, PD2 (INT0).
Prepojka P0	Prepojka na pripojenie sériového rozhrania k programátoru (AT90S2313), alebo k aplikačnému MCU ATmega8.
Prepojka P1	Prepojka na pripojenie bežca trimra POT1 k vstupu A/D prevodníka, ADC0.
Prepojka P2	Prepojka na pripojenie vstupov interného analógového komparátora k bežcu POT1-AIN0 a katóde D2-AIN1.
Prepojka P3	Prepojka na pripojenie vstupov A/D prevodníka (ADC1 a ADC2) k výstupom OZ.
Prepojka P4	Prepojka na pripojenie PWM výstup OC1A k dolnopriepustnému filtru.
Prepojka P5	Prepojka na voľbu funkcie vývodov PD6 a PD7. DSP versus externé obvody.
Svorka S1	Svorky na pripojenie odporového snímača.
Trimer PT1	Trimer na nastavenie veľkosti napätia v rozsahu 0 – VCC.

Tab.1 Popis nastavovacích a modifikačných prvkov, tlačidiel a konektorov

Demonštračný modul sa skladá z časti programátora (AT90S2313) a z časti aplikačnej, na báze MCU ATmega8.

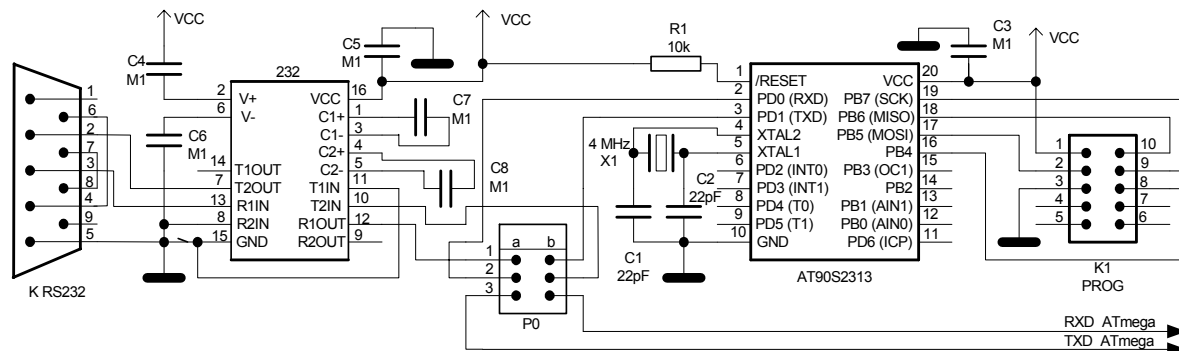
Programátor AVR

Programátor AVR spolupracuje s integrovaným vývojovým prostredím AVR Studio (v personálnom počítači) a umožňuje programovať nasledovné obvody ATMEL AVR:

AT89S53, AT89S8252
 AT90S1200
 AT90S2313, AT90S2323, AT90S2333, AT90S2343
 AT90S4414, AT90S4433, AT90S4434
 AT90S8515, AT90S8535
 ATmega103
 ATmega128
 ATmega16
 ATmega161
 ATmega163
 ATmega32
 ATmega103
 ATmega8
 ATmega83
 ATmega8515

ATtiny10, ATtiny11, ATtiny12, ATtiny15, ATtiny19, ATtiny26

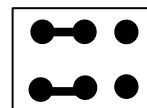
Programátor AVR komunikuje s personálnym počítačom prostredníctvom sériového rozhrania RS232. Komunikačná rýchlosť je nastavená na 19200 b/s. Schéma programátora AVR je uvedená na obr.2.



Obr.2 Programátor AVR

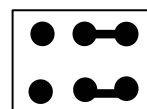
Modul programátora AVR komunikuje s personálnym počítačom prostredníctvom sériového rozhrania, konektor K RS232. Komunikačné signály (RXD, TXD) sú na úroveň RS232 upravené obvodom MAX 232. Prepojka P0 musí byť v prípade programovania konfigurovaná podľa obr.3. Sériové rozhranie je taktiež využívané na komunikáciu aplikačného MCU ATmega8 s okolím, preto je možné pomocou prepojky P0 zvoliť jeden z dvoch možných komunikačných režimov demonstračného modulu.

V režime **programovanie** je potrebné P0 konfigurovať podľa obr.3.



Obr.3

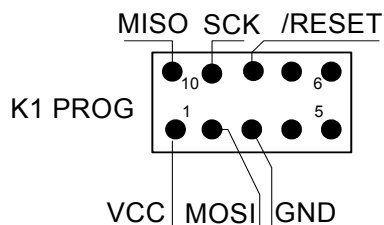
V režime **sériovej komunikácie** MCU ATmega8 je potrebné prepojku P0 konfigurovať podľa obr.4.



Obr.4

Pri komunikácii s programovaným zariadením programátor AVR využíva rozhranie ISP. Rozhranie ISP obsahuje nasledujúce komunikačné signály: SCK, MISO, MOSI. Okrem spomenutých komunikačných signálov je potrebné ovládať vývod /RESET programovaného obvodu. V programovacom konektore **K1 PROG** sú preto priradené štyri vývody komunikačným a riadiacim signálom SCK, MISO, MOSI a /RESET, ďalší vývod je priradený digitálnej zemi - GND.

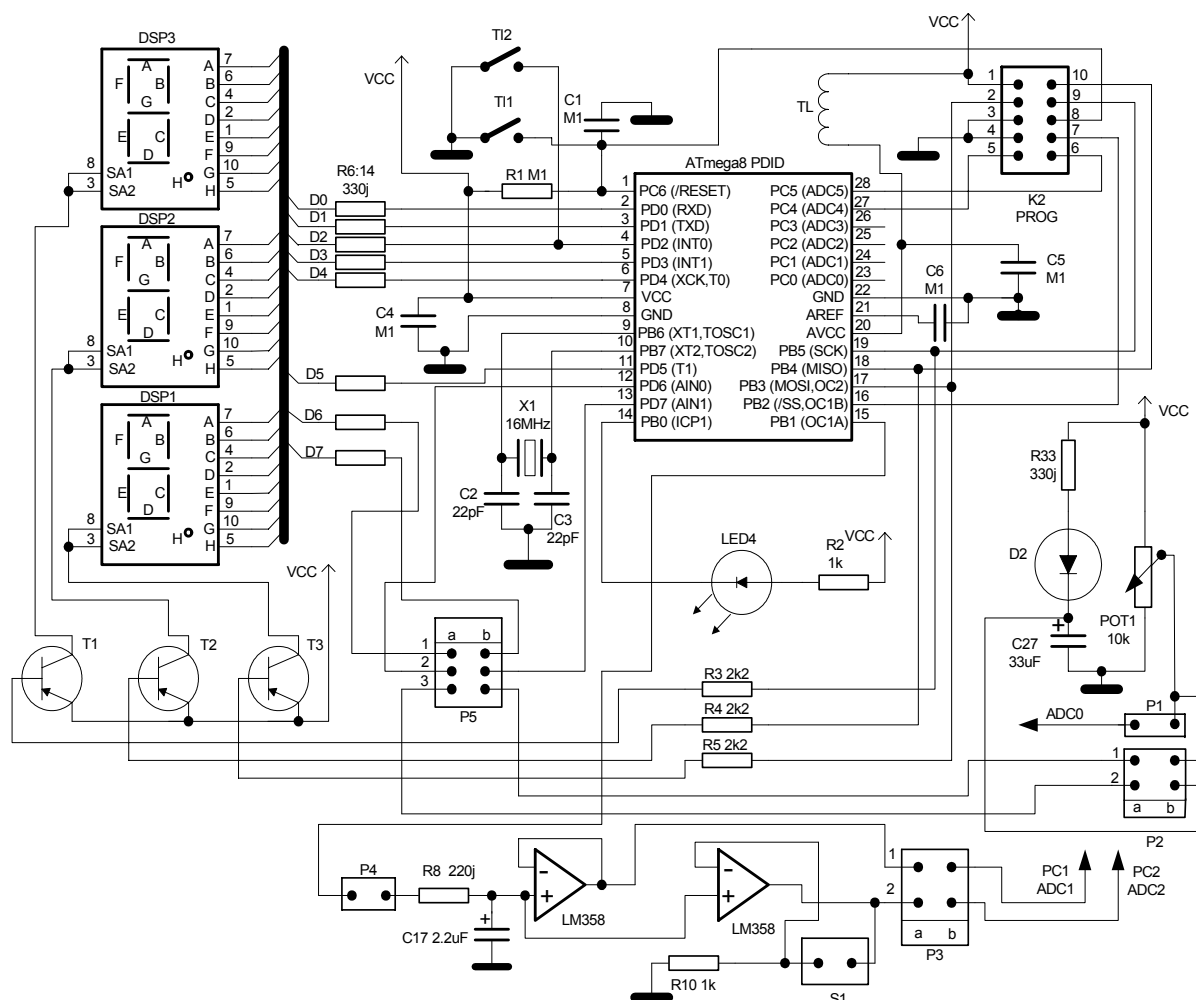
V prípade, že chceme programovanú aplikáciu napájať s programátora AVR je na konektor K1 PROG vyvedené aj napájacie napätie VCC. Rozmiestnenie signálov na programovacom konektore K1 PROG je uvedené na obr.5.



Obr.5 Konektor K1

Aplikačná časť modulu ATmega8

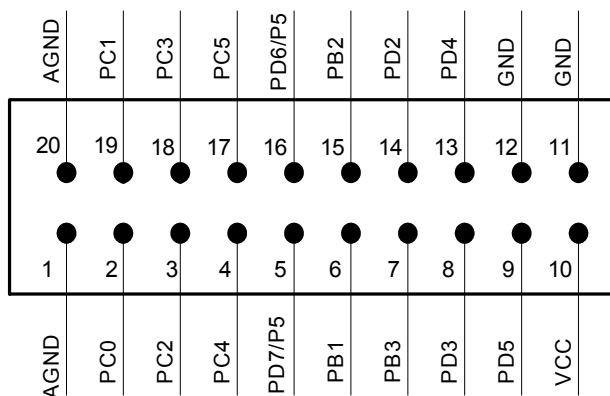
Aplikačná časť modulu ATmega8 slúži na overenie činnosti, demonštráciu základných princípov práce mikrokontroléra, a riešenie jednoduchých aplikačných úloh. Na rozširujúcom konektore sú vyvedené dôležité signály mikrokontroléra, čo umožňuje vývoj a overenie ďalších neštandardných aplikácií. Pomocou prepojok P1 až P5 je možné konfigurovať aplikačnú časť modulu tak, aby bolo možné overiť aj alternatívne funkcie vývodov MCU. Základným prvkom aplikačnej časti je mikrokontrolér ATmega8 synchronizovaný kryštálovým rezonátorom s frekvenciou 16 MHz. Reštartovať MCU je možné pomocou tlačidla T11. Aplikačná časť má možnosť zobraziť požadované údaje prostredníctvom troch sedemsegmentových zobrazovacích prvkov DSP1 až DSP3, prípadne prostredníctvom LCD zobrazovacieho prvku, umožňujúceho zobraziť 16 alfanumerických znakov. Schéma aplikačnej časti je uvedená na obr.6. Na obr.6 nie sú uvedené obvody napájania, zapojenie aplikačného konektora a konektora LCD zobrazovacieho prvku.



Obr.6 Aplikačná časť modulu ATmega8

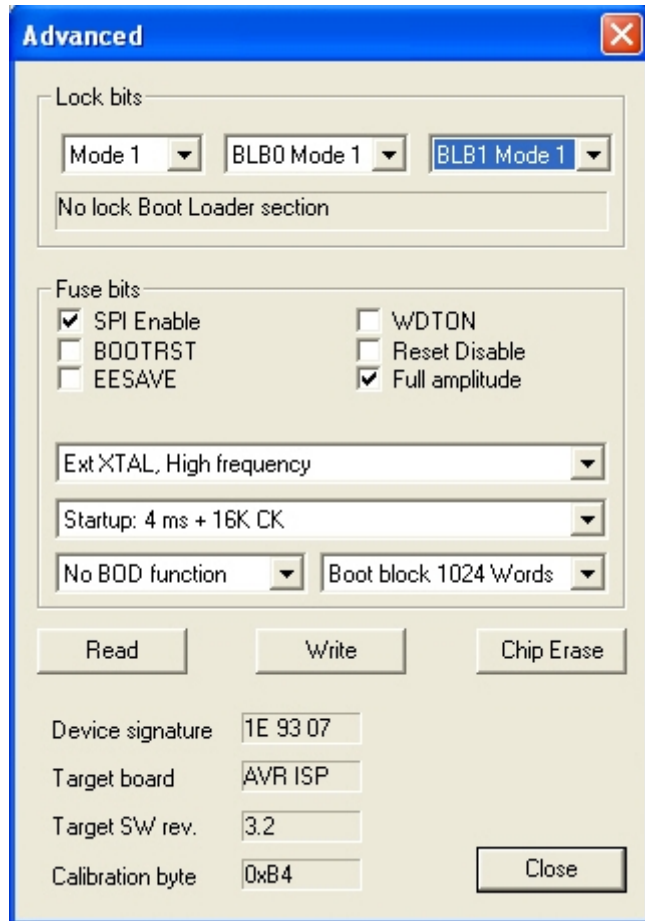
Podrobnejší popis a účel jednotlivých komponentov, prepojení-Px a svoriek-S1 aplikácej časti modulu bude uvedený pri popise jednoduchých aplikácií. Na tomto mieste popíšeme rozmiestnenie signálov aplikačného konektora, ktorý slúži na prípadné prepojenie modulu s okolím.

Aplikačný (rozširujúci) konektor „EX. KONEKTOR“ má 10 vývodov v dvoch radoch. Rozmiestnenie signálov aplikačného konektora je uvedené na obr.7.



Obr.7 Aplikačný konektor

Príklady uvedené v tejto časti slúžia na demonštráciu základných vlastností obvodu a integrovaných periférií. V uvedených aplikáciách je využitý demonštračný modul ATmega8 s nastavením „fuse“ a „lock“ bitov podľa obr.8.



Obr.8 Nastavenie parametrov MCU

Uvádzané príklady sú zoradené takým spôsobom, aby čitateľ postupoval od tých najjednoduchších aplikácií až po zložitejšie. Pripomeňme, že pomocou „lock“ a „fuse“ bitov sú nastavené nasledovné parametre MCU:

„Mode 1“ - Ochrana prístupu k pamätiam nie je nastavená, nie je obmedzené čítanie obsahu a programovanie pamätí FLASH a EEPROM.

„BLB0 Mode 1“ – Nie sú nastavené žiadne obmedzenia pre používanie inštrukcií LPM a SPM pri prístupe do aplikačnej časti pamäte programu.

„BLB1 Mode 1“ – Nie sú nastavené žiadne obmedzenia pre používanie inštrukcií LPM a SPM pri prístupe do časti pamäte zavádzacieho (boot) programu.

„SPI“ - Je povolené sériové programovanie SPI.

„BOOTRST“ – Voľba reštartovacej adresy. Reštartovacia adresa je 0x000.

„EESAVE“ – Ak je naprogramovaný bit EESAVE, tak sa obsah EEPROM pamäte pri mazaní obsahu MCU (chipu) uchová.

„**WDTON**“ – Počiatočné povolenie „watch-dog“ čítača. Pri nastavení uvedenom na obr.8 je činnosť WDT zakázaná, v prípade potreby je možné ju povoliť užívateľským programom.

„**Reset Disable**“ – Nastavenie funkcie RST/PC6 vývodu. Pri nastavení uvedenom na obr.8 má vývod priradenú funkciu RESET.

„**Full Amplitude**“ – Voľba veľkosti amplitúdy oscilátora. V aplikáciách pracujúcich v zašumených prostrediach a pri buzení ďalších aplikácií sa doporučuje daný bit CKOPT naprogramovať označením uvedenej voľby. Taktiež pri použití rezonátora s frekvenciou vyššou než 8 MHz je potrebné zvoliť „Full Amplitude“. Cenou za túto voľbu je zvýšenie spotreby MCU.

„**Ext XTAL, High frequency**“ – Voľba použitého rezonátora. V uvedenom prípade je použitý externý kryštálový rezonátor s vysokou frekvenciou.

„**Startup: 4ms + 16K CK**“ – Voľba štartovacieho času z režimov zo zníženou spotrebou a doplnkového oneskorenia po reštarte MCU.

„**NO BOD function**“ – Povolenie funkcie BOD - detekcia poklesu napájacieho napätia pod definovanú hodnotu. Táto hodnota je určená pomocou „fuse“ bitu „BODLEVEL“.

„**Boot block 1024 Words**“ – Voľba veľkosti časti pamäte určenej pre zavádzací program.

Na základe nastavených parametrov MCU pracuje v základnom režime t.j. po reštarte sa čítač inštrukcií nastaví na adresu 0x000. Ochrana pamätí MCU nie je nastavená. Nie sú nastavené žiadne ďalšie obmedzenia na používanie inštrukcií SPM a LPM pri prístupe k pamäti.

Po nastavení parametrov MCU podľa obr.8 je hodnota „fuse“ a „lock“ bitov nasledovná:

BODLEVEL	BODEN	SUT1	SUT0	CKSEL3	CKSEL2	CKSEL1	CKSEL0
0	1	1	1	1	1	1	1

„Fuse“ bity, dolný byt

RSTDISBL	WDTON	SPIEN	CKOPT	EESAVE	BOOTSZ1	BOOTSZ0	BOOTRST
1	1	0	0	1	0	0	1

„Fuse“ bity, horný byt

		BLB12	BLB11	BLB02	BLB01	LB2	LB1
1	1	1	1	1	1	1	1

„Lock“ bity

Význam jednotlivých bitov je uvedený v tabuľkách 68 až 70 v [1].