Rīgas Tehniska Universitāte

Datorvadības, automātikas un datortehnikas institūts

Datoru tīklu un sistēmas tehnoloģijas katedra

Mikroprocesoru tehnika

Laboratorijas darbs Nr. 5

Asist. R. Taranovs

Profesors V.Zagurskis

Students Vitālijs Hodiko

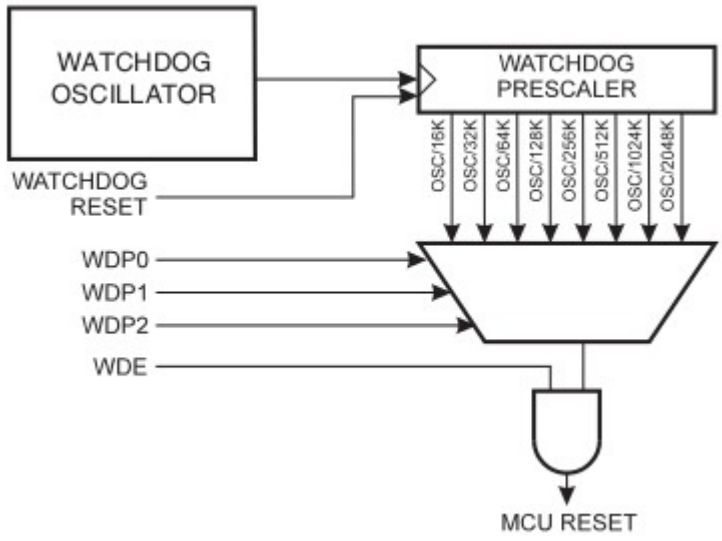
3.kurss 1.grupa

Uzdevums

Papildināt 2. laboratorijas darbu ar sarg-taimeri. Uzstādīt sarg-taimera taimauta/atiestatīšanas laiku vienādu ar aptuveni 2 sekundēm. Pierādīt (ar gaismas diodēm), ka sargtaimeris strādā ar izvēlēto taimauta laiku.

Teorētiskais apraksts

Sargtaimeris (watchdog timer), dažreiz saukts arī par COP (skaitļotājs darbojas pareizi), kontrolē vai programmas darbības tiek veiktas noteiktā laikā, t.i. kontrolē vai programma nav "uzkārusies". Bieži tiek lietots, lai pārbaudītu programmas izpildes rezultātus. Vienreiz to ieslēdzot taimeris sāk skaitīt atpakaļ, kad saskaita līdz 0 tiek izsaukta RESET instrukcija kura atiestata mikrokontrolleri kā arī sāk no jauna programmas izpildi. Lai šī atiestatīšana nenotiktu ir jāatslēdz sargtaimeris vai to ir jāatjauno. Ja programma novirzās no tās normālās izpildes, tad ar sargtaimeri tā tiek palaista no jauna.



Attēls 1: WD bloka diagramma

Sargtaimerim ir savs iekšējais oscilators (1 MHz), kas palielina viņa drošumu. Tā tipiskais barošanas spriegums ir Vcc = 5V. Kontrolējot sargtaimera priekš dalītāju var regulēt WDR (Watchdog Timer Reset) intervālu – laiku pēc kura sargtaimeris veiks mikrokontrollera atiestatīšanu.

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	14.8 ms	14.0 ms
0	0	1	32K (32,768)	29.6 ms	28.1 ms
0	1	0	64K (65,536)	59.1 ms	56.2 ms
0	1	1	128K (131,072)	0.12 s	0.11 s
1	0	0	256K (262,144)	0.24 s	0.22 s
1	0	1	512K (524,288)	0.47 s	0.45 s
1	1	0	1,024K (1,048,576)	0.95 s	0.9 s
1	1	1	2,048K (2,097,152)	1.9 s	1.8 s

Tabula 1: WD Priekšdalitājs

WDR – tā ir instrukcija, kas atiestata pašu sargtaimeri. Sargtaimeris arī tiek atiestatīts, kad to

atslēdz un kad notiek čipa atiestatīšana (Chip Reset). Astoņi dažādi taktēšanas ciklu periodi var būt izvēlēti, lai noteiktu atiestatīšanas periodu. Ja atiestatīšanas periods beigsies un cits WDR netiek padots, ATmega128 pārlādēsies. Lai novērstu nejaušu sargtaimera atslēgšanos kā arī sargtaimera nejaušu skaitīšanas perioda maiņu, tiek lietoti 3 dažādi drošības līmeņi, kas tiek iestādīti ar Fuse M103C un WDTON bitiem:

M103C	WDTON	Safety Level	WDT Initial State	How to Disable the WDT	How to Change Time-out
Unprogrammed	Unprogrammed	1	Disabled	Timed sequence	Timed sequence
Unprogrammed	Programmed	2	Enabled	Always enabled	Timed sequence
Programmed	Unprogrammed	0	Disabled	Timed sequence	No restriction
Programmed	Programmed	2	Enabled	Always enabled	Timed sequence

Tabula 2: Drošības līmeņi

0- drošības līmenis. Šis stāvoklis ir savietojams ar sargtaimera darbību Atmega103. Taimeris sākumā ir izslēgts, bet var tikt ieslēgts, ierakstot WDE bitu „1" bez jebkādiem aizliegumiem. Skaitīšanas periods var tikt mainīts jebkurā laikā bez jebkādiem aizliegumiem.

1- drošības līmenis. Šajā stāvoklī sākotnēji sargtaimeris ir izslēgts, bet var tikt ieslēgts ierakstot WDE bitu „1" bez jebkādiem aizliegumiem. Ir jāizpilda noteiktu darbu secība, lai izmainītu skaitīšanas periodu, lai izslēgtu sargtaimeri:

1. Vienā operācijā jāieraksta vieninieks WDCE un WDE bitos. WDE bitā jāieraksta vieninieks, neatkarīgi no iepriekšējās WDE bita vērtības.

2. Nākošo četru takšu laikā vienā operācijā jāieraksta WDE un WDP bitus, izvēloties attiecīgo taimauta laiku, bet notīrot WDCE bitu.

2- Drošības līmenis. Šajā stāvoklī sargtaimeris ir vienmēr ieslēgts, un WDE bits tiks vienmēr nolasīts kā „1". Arī šeit ir noteikta darbību secība, lai mainītu sargtaimera skaitīšanas periodu. Lai izmainītu skaitīšanas periodu, ir jāievēro sekojoša procedūra.

1. Vienas operācijas laikā jāieraksta vieninieks WDCE un WDE bitos. Lai arī WDE ir vienmēr iestatīts, WDE vienalga ir jāiestata vieninieka, lai iesāktu laika secību.

2. Nākošo četru takšu laikā vienā operācijā jāieraksta WDP biti, kā ir iecerēts, bet notīrot WDCE bitu. WDE bitā ierakstītā informācija ir nesvarīga.

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Attēls 2: Sargtaimera vadības reģistrs(Watchdog Timer Control Register)

7.-5. biti ir rezervētie biti ATmega128 un vienmēr tiek nolasīti kā nulles.

4. bits- WDCE: Watchdog Change Enable

Šis bits tiek izmantots, lai norādītu par sargtaimera uzstādījumu (taimauta laiku, ieslēgt/izslēgt sargtaimeri) mainīšanu. Ierakstot loģisko „1”, pēc četrām taktīm aparatūra to notīra.

3. bits - WDE: Watchdog Enable

Kad WDE bits ir ierakstīts kā loģiskais „1”, sargtaimeris ir ieslēgts un ja WDE ir ierakstīts kā loģiskā „0”, sargtaimeris ir izslēgts. WDE bitu var notīrīt tikai tad, ja WDCE bits ir loģiskais „1”. Lai ieslēgtu vai izslēgtu ir jāveic sekojošas procedūras:

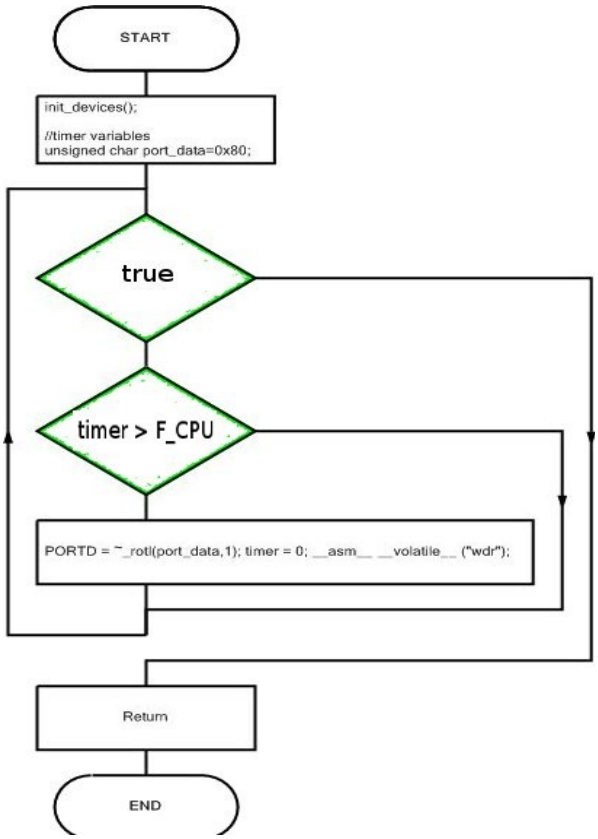
- Vienā operācijā ieraksta loģisko vieninieku WDCE un WDE. Loģiskajam vieniniekam jābūt ierakstītam WDE, lai arī tas ir ierakstīts jau iepriekš, pirms vel atslēgšanas operācija bija sākusies.
- Nākamajās četrās taktīs jāieraksta loģiskā „0” WDE. Tas atslēgs sargtaimeri. Drošības līmenī 2, nav iespējams atslēgt sargtaimeri.

Programmas kods

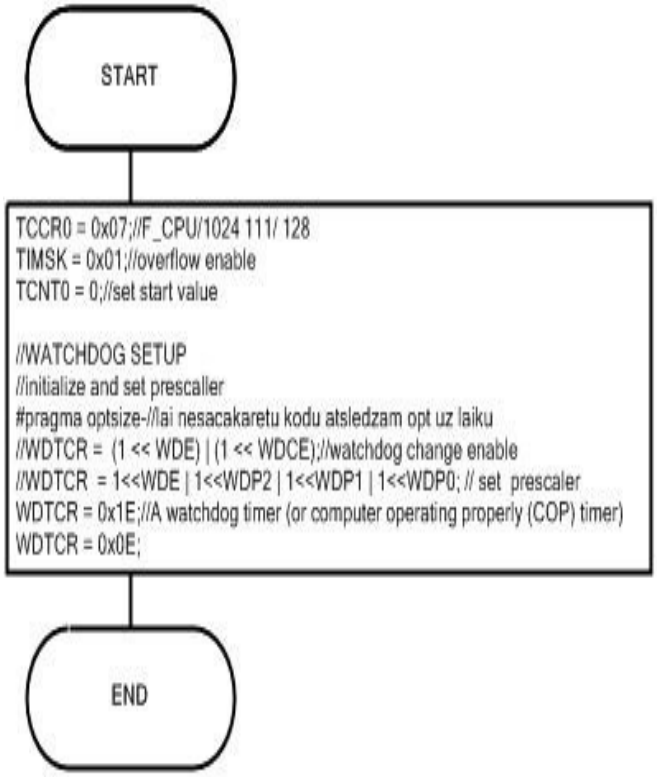
```
/***** GLOBAL CONST DEFINE ******************************************/
#define F_CPU 14745600UL     //cycles per second; repeats F_CPU times per second

/***** BIBLIOTEKU IEKLAUSANA ****************************************/
#include <avr/io.h>
#include <avr/interrupt.h>

/***** FUNCTION PROTOTYPES ******************************************/
void port_init(void);
void init_devices(void);

unsigned char _rotl(const unsigned char value, unsigned char shift);

/***** GLOBAL VARS **************************************************/
volatile unsigned long timer=0;

/***** INTERRUPTS ***************************************************/
ISR(TIMER0_OVF_vect){
      timer+=255*128;
}

//this MCU dont have 'WDT_vect' func

/***** MAIN *********************************************************/
int main (void){
      init_devices();

      //timer variables
      unsigned char port_data=0x80;
      while(1){
            if(timer > F_CPU){PORTD = ~_rotl(port_data,1); timer = 0; __asm__
__volatile__ ("wdr");}
      }
      return 1;
}

/*******************************************************************/
/*****  CIRCULAR SHIFT *********************/
unsigned char _rotl(const unsigned char value, unsigned char shift) {
    if ((shift &= sizeof(value)*8 - 1) == 0) return value;
    return (value << shift) | (value >> (sizeof(value)*8 - shift));
    //return (value >> shift) | (value << (sizeof(value)*8 - shift)); //right
}

/***** PORTU INICIALIZACIJA ************************/
void port_init(void){
      DDRD  = 0xFF; //visas porta D linijas uz IZvadi
}

/***** REZIMU[sakuma vertiibu] UZSTADISANA *********/
void init_devices(void){
      //TIMER0 SETUP
      TCCR0 = 0x07;     //F_CPU/1024 111/ 128
      TIMSK = 0x01;     //overflow enable
```

```
      TCNT0 = 0;        //set start value

      //WATCHDOG SETUP
      //initialize and set prescaller
      #pragma optsize-  //lai nesacakaretu kodu atsledzam opt uz laiku
            WDTCR = 0x1E;                        //A watchdog timer (or computer
operating properly (COP) timer)

            WDTCR = 0x0E;
            #ifdef _OPTIMIZE_SIZE_
                  #pragma optsize+
            #endif

      /**** ALLOW GLOBAL INTERRUPTS ********/
      sei();

      port_init();
}
```

Blokshēma



Attēls 3: main()



Attēls 4: init_devices()

Secinājumi

Sarg-taimera galvenu funkciju lieliski apraksta tā otrais nosaukums Computer Operating Properly, jeb skaitļotais strādā pareizi. Ir acīmredzami, ka tas kontrole programmas darbības pareizību. Tas tiek nodrošināts atvēlējot programmai noteiktu izpildes laiku, kurā tai vajag paspēt atiestatīt WD taimeru, kurš tiek darbināts uz tā iekšēja oscilatora nodrošinot lielāku drošību.

Ka arī , lai izvairītos no neparedzētas WD taimera atslēgšanas un taimauta izmaiņas ir ieviesti dažādas drošības pakāpes, kur atkarība no izvēlētas var vai nu tikai mainīt taimauta laiku, vai arī atslēgt/ieslēgt WD taimeri.

Turpinot pirmo teikumu, lai sekotu, ka programma strādā pareizi vajag, lai kaut kas izpildītos, tāpēc sanāk ka to atiestatīšanas funkciju jāliek pēc noteikta koda apgabala. Atceroties, ka tas strādā neatkarīgi no sistēmas pulksteņa jāņem vēra to koda apgabala izpildes laiku, lai tas nepārsniegtu WD taimera taimautu. WD taimerim ir max taimauta vērtība un ja dotais kods pārsniedz to, neviens neaizliedz sadalīt to mazākas daļas un pēc katras ielikt WD taimera atiestatīšanu.

Sava programma es ieliku to iekšā nosacījuma pārbaudēs pozitīva rezultāta izpildes daļā, kura tiek izpildīta reizi ~8 sekundes:

$TGOVF = \frac{N}{FCPU} * 2^{bit}(s) -> TGOVF = \frac{1024}{FCPU} * 2^8 = 0{,}0177(s)$, tas nozīme ka 8 bitu taimeris ar priekšdalītāju 1024 pārplūdis katru 0,0177 sekundi. Apzīmēsim to ar 1 OVF = 0,0177 s. Tā ka es salīdzinu mainīgo *timer* ar FCPU kuri ir "bišči" par lielu par taimera lielāko vērtību, tad tā nosacījuma izpildes biežums būs reizi: $\frac{FCPU}{128 * 255} = 452 OVF$, jeb 452 * 0,0177 = 8,0004 sekundes. Sargtaimera taimauta laiks ir maksimālais un tas ir 1,9 sekundes, kas rezultēs nepārtraukta plates re-startēšana, kas arī bija novērots. Tīri intuitīvi es samazināju FCPU vērtību salīdzināšanas daļa dalot to ar 8, kas palielināja šī koda patiesas daļas izpildi līdz vienai reizei ~1 sekundē, kurā atrodas WD taimera notīrīšana.