RĪGAS TEHNISKĀ UNIVERSITĀTE

DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE

DATORVADĪBAS, AUTOMĀTIKAS UN DATORTEHNIKAS INSTITŪTS

Datoru tīklu un sistēmas tehnoloģijas katedra

Mikroprocesoru tehnika

2. laboratorijas darbs

Izpildīja:
Grupa: III RDB F02
Apl. numurs: 101RDB121

2012./2013. māc. gads

Saturs

Uzdevums

Modificēt pirmā laboratorijas darba programmas pirmkodu tā lai pauze starp gaismas diožu pārslēgšanām būtu vienāda vienai sekundei, laika mērīšanai izmantojot ATmega128 Taimeri/skaitītāju 0.

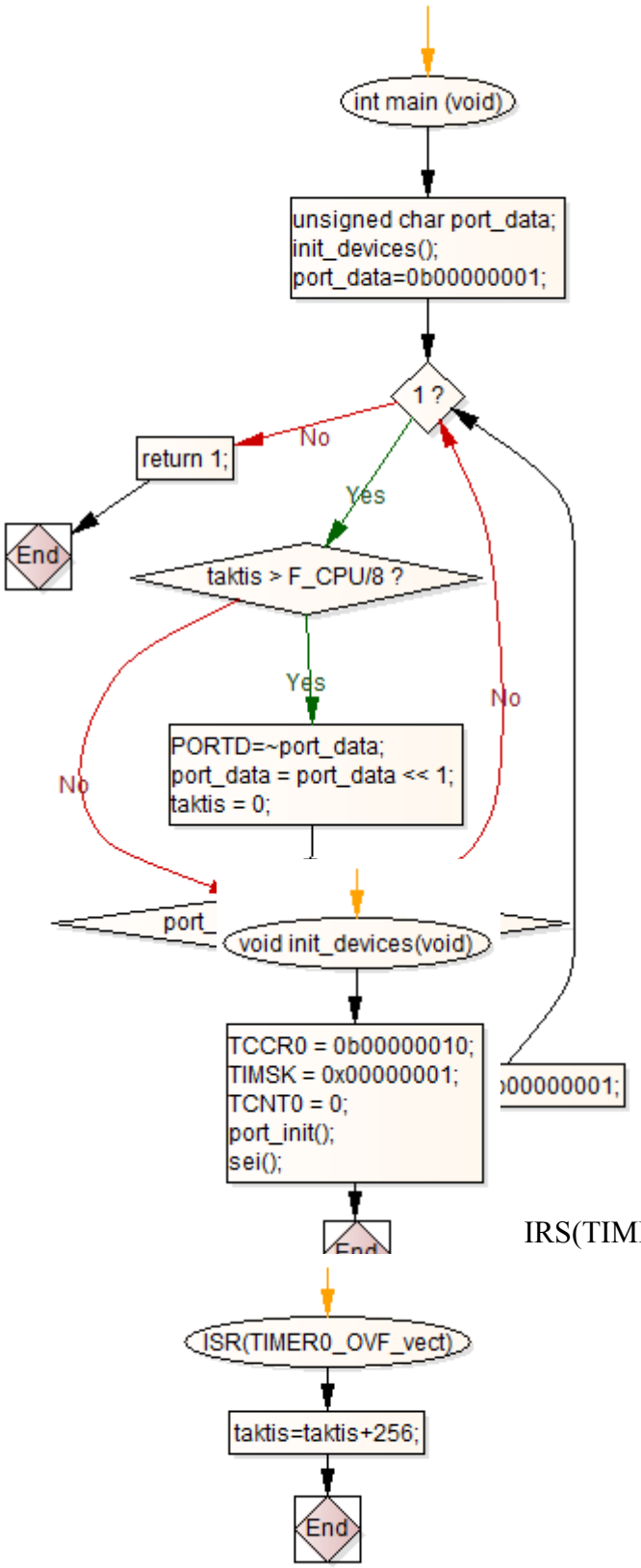
Programmas pirmteksts

```
/*****************************************************************/
#define F_CPU 14745600UL          //Mikrokontrollera takts frekvences definēšana

/********* Standarta C un specialo AVR bibliotēku iekļaušana*********/
#include <avr/io.h>
#include <avr/iom128.h>
#include <avr/interrupt.h>
#include <math.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <util/delay.h>
/*****************************************************************/

#define BIT0 0x01  //Nodefinēta 0-taa bita maska
#define BIT1 0x02  //Nodefinēta 1-taa bita maska
#define BIT2 0x04  //Nodefinēta 2-taa bita maska
#define BIT3 0x08  //Nodefinēta 3-taa bita maska
#define BIT4 0x10  //Nodefinēta 4-taa bita maska
#define BIT5 0x20  //Nodefinēta 5-taa bita maska
#define BIT6 0x40  //Nodefinēta 6-taa bita maska
#define BIT7 0x80  //Nodefinēta 7-taa bita maska

/******************** Portu inicializācijas funkcija ******************/
void port_init(void)
{
 DDRA  = 0x00; //visas porta A līnijas uz IEvadi
 DDRB  = 0x00; //visas porta B līnijas uz IEvadi
 DDRC  = 0x00; //visas porta C līnijas uz IEvadi
 DDRD  = 0xFF; //visas porta D līnijas uz IZvadi
 DDRE  = 0x00; //visas porta E līnijas uz IEvadi
 DDRF  = 0x00; //visas porta F līnijas uz IEvadi
 DDRG  = 0x00; //visas porta G līnijas uz IEvadi

 PORTA  = 0x00; //porta A atsienošie rezistori pret +Vcc NEtiek izmantoti
 PORTB  = 0x00; //porta B atsienošie rezistori pret +Vcc NEtiek izmantoti
 PORTC  = 0x00; //porta C atsienošie rezistori pret +Vcc NEtiek izmantoti
 PORTD  = 0x00; //porta D izejas līniju līmeni uz 0
 PORTE  = 0x00; //porta E atsienošie rezistori pret +Vcc NEtiek izmantoti
 PORTF  = 0x00; //porta F atsienošie rezistori pret +Vcc NEtiek izmantoti
 PORTG  = 0x00; //porta G atsienošie rezistori pret +Vcc NEtiek izmantoti
}
/*****************************************************************/

/******************** Kontrollera inicializācija**********************/
void init_devices(void)
{
//cli();                          //aizliedz visus pārtraukumus
 TCCR0 = 0b00000010;        //F_CPU/1024 111/ 128
 TIMSK = 0x00000001;        //atļaut pārpildīšanos (overflow)
```

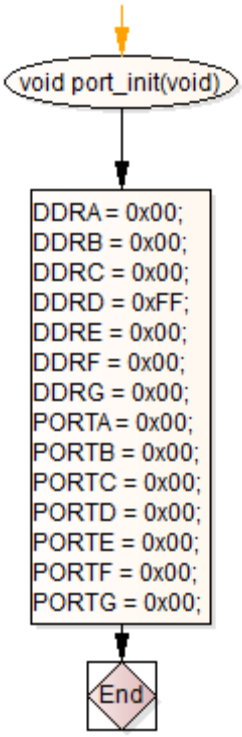
```
 TCNT0 = 0;            //piešķir sakuma vērtību
 port_init();          //inicializē portus
 sei();                //atļauj globālos pārtraukumus
 return;
}
/*********************************************************************/

/******************** Globalie mainigie*********************************/
volatile unsigned long taktis = 0; //tips volatile norāda kompilatoram, ka šim mainīgam nevar
                                   //taisīt kopijas reģistrā un kodā to nedrīkst optimizēt
/*********************************************************************/

/******************** Partraukumi ************************************/
ISR(TIMER0_OVF_vect)
{
taktis=taktis+256;                 //palielina taktis vērtību (inicializē taimeri)
}
/*********************************************************************/

/***************************** Main funkcija *************************/
int main (void)
{
 unsigned char port_data;   //Definējam nepieciešamos mainīgos
 init_devices();            //Inicializējam kontrolleri
 port_data=0b00000001;      //Piešķir bināro vērtību mainīgajam

 while(1)                   //Mūžīgais cikls, lai programma nekad nebeigtos
 {
 if(taktis > F_CPU/8)       //Ja nosacījums izpildās, tad...
 {
  PORTD=~port_data;         //Izvada uz Porta D līnijām apgrieztu port_data mainīga vērtību –
                            //ieslēdz nākamo gaismas diodi
  port_data = port_data << 1; //Pārbīda bināro vērtību uz vienu bitu pa kreisi
  taktis = 0;               //Maina taktis vērtību uz 0 (taimeris sākuma stāvoklī)
 }
 if(port_data == 0b00000000)  //Ja nosacījums izpildās, tad...
 {
 port_data = 0b00000001;      //...piešķir port_data jaunu (sākuma) vērtību
 }
}
 return 1;                  //funkcija main atgriež vērtību 1
}
/*********************************************************************/
```

Programmas pirmteksta algoritma blokshēma

Funkcija main()



Funkcija port_init()



Funkcija init_devices()

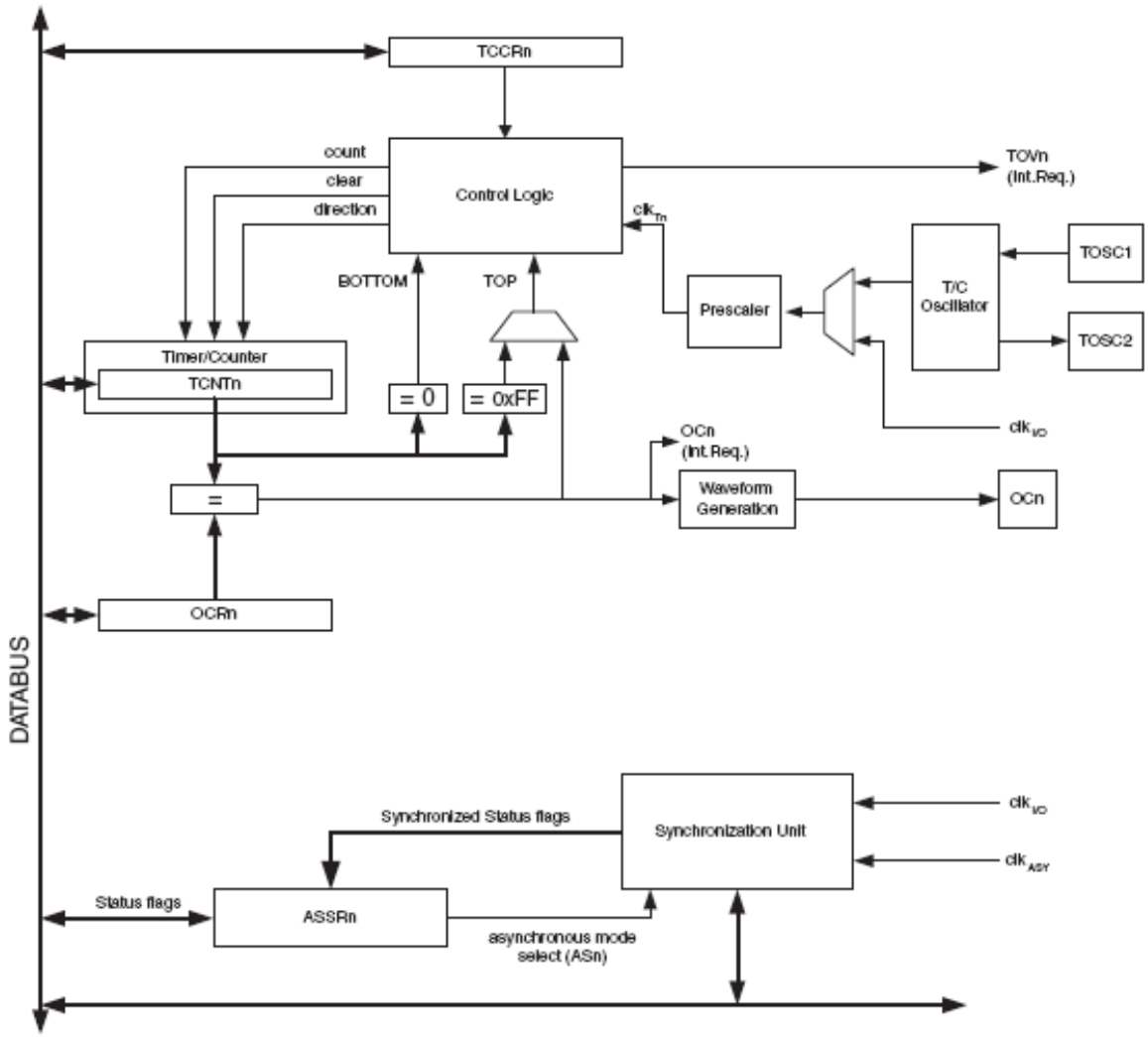
IRS(TIMER0_OVF_vect)

ATmega128 mikrokontrollera 8 bitu taimeris/skaitītājs 0

Taimeris/Skaitītājs 0 ir vienkanāla, 8-bitu Taimera/Skaitītāja modulis.

Taimera/Skaitītāja 0 galvenās īpašības ir:

- vienkanāla skaitītājs;
- taimera notīrīšana pēc salīdzināšanas sakrišanas;
- fāzē korekta impulsa platuma modulators (PWM);
- frekvences ģenerators;
- 10-bitu takts priekš dalītājs;
- pārpildīšanās un salīdzināšanas sakritības pārtraukumu avoti(TOV0 un OCF0);
- Atļauj taktēšanu no ārējā 32kHz kristāla neatkarīgi no I/O takts



1. attēls 8 bitu taimera/skaitītāja struktūrshēma

Taimera/Skaitītāja (TCNT0) un izejas salīdzināšanas reģistrs (OCR0) ir 8-bitu reģistri. Pārtraukuma pieprasījuma signāls (saīsinājums Int.Req) ir redzams taimera pārtraukumu karodziņu reģistrā (TIFR). Visi pārtraukumi ir individuāli maskēti TIMSK (Timer Interrupt Mask Registr) reģistrā.

Taimeris/Skaitītājs 0 var būt taktēts no iekšēja oscilatora, ar priekš dalītāju vai asinhroni no TOSC1/2 līnijām. Taktēšanas izvēles loģika kontrolē kurš taktēšanas avots izraisīs

taimera/skaitītāja vērtības palielināšanu (vai samazināšanu). Taimeris/Skaitītājs ir neaktīvs, kad taktēšanas avots nav izvēlēts.

Lai vadītu 8 bitu taimeru/skaitītāju 0 ir nepieciešams operēt ar to reģistriem, lai sakonfigurētu un nolasītu taimera/skaitītāja vērtības.

Bits	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Lasīt/Rakstīt	R	L/R	L/R	L/R	L/R	L/R	L/R	L/R	
Sākumvērtība	0	0	0	0	0	0	0	0	

7.

bits – FOC0 Force Output compare (piespiedu izejas salīdzināšana)

FOC0 bits ir aktīvs tikai tad, kad WGM biti ir norādīti kā ne-PWM režīms. Savietojamības dēļ šim bitam jātiek uzstādītam uz nulli, kad tiek ierakstīts TCCR0 bits, darbojoties PWM režīmā. Kad FOC0 bitā tiek ierakstīts 1, nekavējoties notiek salīdzināšanas sakritības uzspiešana impulsa platuma ģeneratoram. FOC0 bits vienmēr ir nolasāms kā nulle.

6. un 3. bits – WGM01:0: Waveform Generation Mode (viļņa formas ģenerēšanas režīms)

Biti kontrolē skaitītāja skaitīšanas secību, skaitītāja maksimālās (TOP) vērtības avotu un viļņa formas ģenerācijas tipu.

Taimera/skaitītāja režīmi:

- Normālais režīms
- Taimera notīrīšanas uz salīdzināšanas sakritības režīms (CTC)
- Divi impulsa platuma modulācijas veidi (PWM)

Režīms	WGM01[1] (CTC0)	WGM00[1] (PWM0)	Taimeris/Skaitītājs Darbības režīms	TOP	OCR0 tiek koriģēts	TOV0 karodziņš
0	0	0	Normal	0xFF	**Tūlītēji**	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	**Tūlītēji**	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

5. un 4. bits – COM01:0: salīdzināšanas sakritības izejas režīms

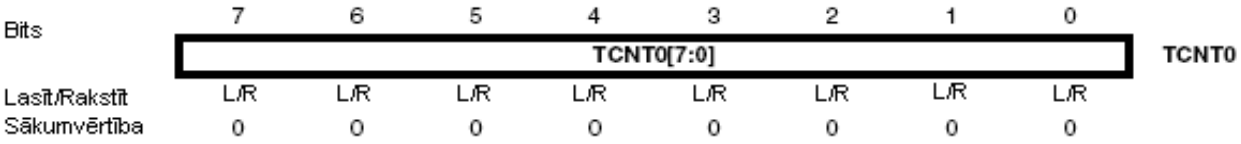
Šie biti nosaka izejas salīdzināšanas pina (OC0) uzvedību. Ja viens vai abi biti ir iestatīti, OC0 izeja tiek izmainīta normāla izejas/ieejas porta funkcionalitāte, pie kura tas ir pievienots. Jāpiezīmē, ka DDR reģistram jābūt iestatītam, lai atļautu pinam darboties kā izejai.

COM01	COM00	Apraksts
0	0	Normāla porta darbība, OC0 atvienots
0	1	Uzstāda OC0 uz salīdzināšanas sakritības
1	0	Pārslēdz OC0 uz salīdzināšanas sakritības
1	1	Nodzēš OC0 uz salīdzināšanas sakritības

2. , 1., un 0. bits – CS02:0: takts izvēle

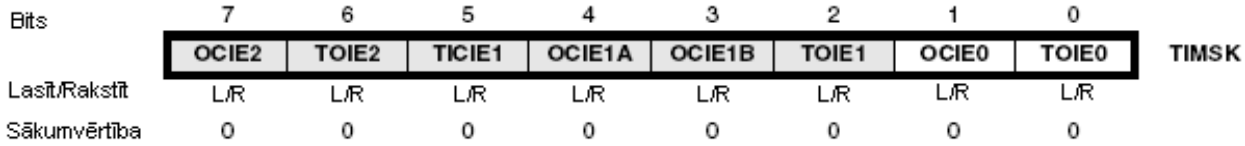
Trīs takts izvēles biti iestata izvēlēto takts avotu, kuru izmantos taimeris/skaitītājs0. Mainot priekš-dalītāja vērtību iespējams palielināt laika intervālu ko tas mēra, bet tiek zaudēta precizitāte.

CS02	CS01	CS00	Apraksts
0	0	0	Nav taktēšanas avota (Taimers/Skaitītājs apturēts)
0	0	1	clk_{TOS}/ (Bez priekšdalīšanas)
0	1	0	clk_{TOS}/8 (No priekšdalītāja)
0	1	1	clk_{TOS}/32 (No priekšdalītāja)
1	0	0	clk_{TOS}/64 (No priekšdalītāja)
1	0	1	clk_{TOS}/128 (No priekšdalītāja)
1	1	0	clk_{TOS}/256 (No priekšdalītāja)
1	1	1	clk_{TOS}/1024 (No priekšdalītāja)



3. attēls Taimera/skatītāja reģistrs - TCNT0

Taimera/skatītāja 0 TCNT reģistrs dod tiešu pieeju, gan pie rakstīšanas, gan pie lasīšanas operācijām, saistītām ar taimera/skatītāja pašreizējo vērtību.



4. attēls Taimera/skatītāja reģistrs - TIMSK

1. bits – OCIE0: Taimera/Skaitītāja0 Output Compare sakrišanas pārtraukuma ieslēgšana
Kad reģistrā OCIE0 ir ierakstīts viens, un I-bits Status Reģistrā arī ir uzstādīts (vieninieks), taimera/skaitītāja 0 sakrišanas pārtraukums ir ieslēgts.

0. bits – TOIE0: Taimera/Skaitītāja0 pārpildes pārtraukuma ieslēgšana
Kad reģistrā TOIE0 ir ierakstīts viens, un I-bits Status Reģistrā arī ir uzstādīts (vieninieks), taimera/skaitītāja0 pārpildes pārtraukums ir ieslēgts. Attiecīgs pārtraukums tiek izsaukts ja parādās taimera/skaitītāja0 pārpilde, t.i. kad TOV0 bits ir uzstādīts taimera/skaitītāja0 pārtraukumu karodziņu reģistrā – TIFR.

Secinājumi

Otrā laboratorijas darba gaitā es iepazinos ar mikrokontrollera ATmega128 taimeri/skaitītāju 0, modificējot pirmā laboratorijas darba programmas pirmkodu.

Pirmajā laboratorijas darbā, lai nodrošinātu pauzi starp gaismas diožu pārslēgšanām bija iespējams izmantot kādu no delay tipa funkcijām vai ciklu bez noteiktas darbības veikšanas. Es izmantoju _delay_ms() iebūvēto funkciju, jo tai ir īsāks pieraksts un lai nevajadzētu zīmēt vēl vienu ciklu blokshēmā. Bet funkcijai _delay_ms() ir savi ierobežojumi un bez optimizācijas tā nav pielietojama, jo pārtraukuma ilgums kļūst neprognozējams. Tomēr ka es uzzināju no otrā laboratorijas darba prezentācijas, _delay_ms() un cikls bez darbības ir bloķējoši. Citiem vārdiem sakot, tie ir slikti pārtraukuma realizācijas veidi, jo CPU neko citu nevar paralēli izpildīt un daudz labāk laika mērīšanai un pārtraukuma veidošanai ir izmantot taimeri, jo tad paralēli var veikt arī citas darbības.

Lai izveidotu pārtraukumu otrajā laboratorijas darbā tika izmantoti volatile unsigned long globālais mainīgais, interrupt.h bibliotēka un ATmega128 taimeris/skaitītāju 0 laika mērīšanai. Izpildīt laboratorijas darbu nebija grūti, jo prezentācija bija dots taimera realizācijas piemērs un tika aprakstīta laboratorijas darba izpildei nepieciešamā taimera pārpildes pārtraukuma realizācijas būtība, vajadzēja tikai atbilstoši izmainīt programmas pirmkodu, pievienojot nepieciešamos if nosacījumus un modificējot funkcijas.

Uzskatu, ka laboratorijas darbs ir veiksmīgi izpildīts, jo modificēta programma strādā bez kļūdām un, izmantojot ATmega128 taimeri/skaitītāju 0 laika mērīšanai, nodrošina 1 sekundes pauzi starp gaismas diožu pārslēgšanām.