RĪGAS TEHNISKĀ UNIVERSITĀTE

DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE

DATORVADĪBAS, AUTOMĀTIKAS UN DATORTEHNIKAS INSTITŪTS

Datoru tīklu un sistēmas tehnoloģijas katedra

Mikroprocesoru tehnika

3. laboratorijas darbs

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

2012./2013. māc. gads

Saturs

Uzdevums

Papildināt 2. laboratorijas darbu ar rādītāju uz datu masīvu. Masīvā glabā veselas skaitļu vērtības, piemēram, no 1 līdz 5, masīvu aizpildīt izmantojot rādītāju. Masīva vērtības nolasīt un izmantot skrejošās gaismas ilguma veidošanā. Piemēram, masīva 1. elements ir 1, tad skrejošās gaismas ilgums ir 1 sekunde, masīva 2. elements ir 2, tad skrejošās gaismas ilgums ir 2 sekundes, utt.

Programmas pirmteksts

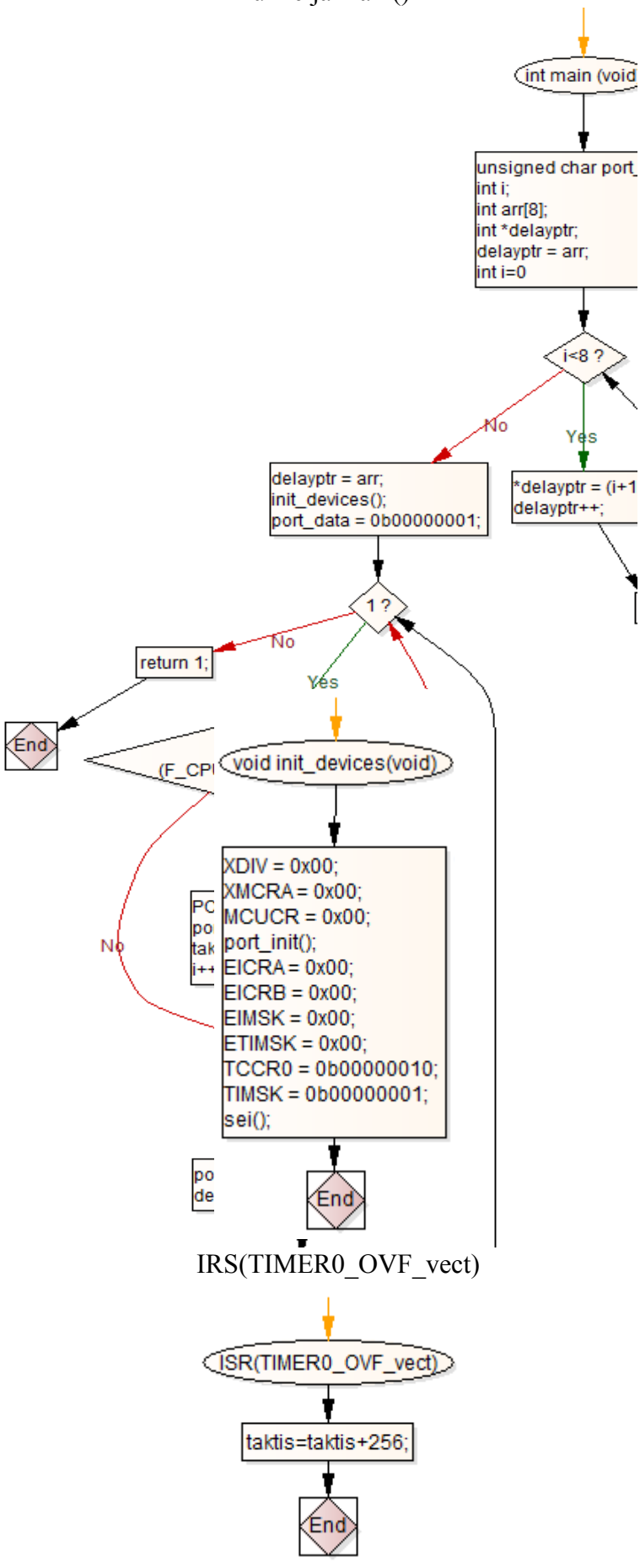
```
#define F_CPU 14745600UL          //Mikrokontrollera takts frekvences definēšana

/******* 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>
/*************************************************************************/

#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

volatile unsigned long long taktis; //Globālais mainīgais kurš glabā taktu skaitu

/********************** Pārtraukuma no taimera/skaitītāja0 pārpildes funkcija **********/
ISR(TIMER0_OVF_vect)
{
 taktis = taktis + 255;           //palielina taktis vērtību (inicializē taimeri)
}
/*************************************************************************/

/************************ 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 līmeņi 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
 XDIV  = 0x00;          //takts impulsu dalītājs NEtiek izmantots
 XMCRA = 0x00;          //ārējo atmiņu NEizmanto
 MCUCR = 0x00;          //NEtiek izmantoti nekādi enerģiju taupoši stāvokli

 port_init();           //Inicializē portus

 EICRA = 0x00;
 EICRB = 0x00;
 EIMSK = 0x00;          //Aizliedz visus ārējos pārtraukumus no portiem
 ETIMSK = 0x00;         //Aizliedz jebkāda veida pārtraukumus no 16 bitu taimeriem

/*** Taimeris/skaitītājs0 skaitītāju pieslēdz pie takts ģeneratora caur /8 dalītāju ***/
 TCCR0 = 0b00000010;

/*** Atļauj pārtraukumu no taimera/skaitītāja0 pārpildes ***/
 TIMSK = 0b00000001; //OCIE2|TOIE2|TICIE1|OCIE1A|OCIE1B|TOIE1|OCIE0|TOIE0

 sei();                 //Atļauj globālos pārtraukumus
}
/****************************************************************************/

/******************************** Main funkcija ******************************/
int main (void)
{
 unsigned char port_data;                   //Unsigned char mainīgais
 int i;                                     //Integer mainīgais
 int arr[8];                                //Izveidotais masīvs
 int *delayptr;                             //Rādītājs

 delayptr = arr;                //Norādām masīva pirmo elementu

 for (i=0; i<8; i++)            //Cikls, kurš aizpilda masīvu
 {
 *delayptr = (i+1);             //Izmantojot rādītāju piešķiram masīva elementam vērtību
  delayptr++;                   //Pārējam uz nākamo masīva elementu
 }

delayptr = arr;         //Norādām masīva pirmo elementu

init_devices();         //Inicializējam kontrolleri

port_data  = 0b00000001;      //Piešķiram bināro vērtību mainīgajam

while(1) //Mūžīgais cikls, lai programma nekad nebeigtos
{
  if (taktis > (F_CPU/8*(*delayptr)))         //Ja nosacījums izpildās, tad...
  {
```

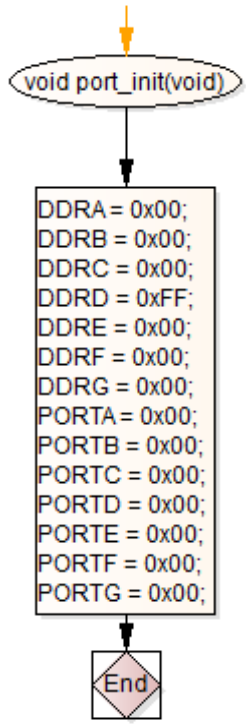
```
   PORTD=~port_data;         //Izvada uz Porta D līnijām apgrieztu port_data mainīga vērtību
                             //ieslēdz nākamo gaismas diodi
   port_data = port_data<<1; //Pārbīda bināro vērtību uz vienu bitu pa kreisi
   taktis = 0;               //Maina taktis vērtību uz 0 (iestāda taimeri sākuma stāvoklī)
   i++;                      //Palielinām i vērtību - skaitam gaismas diodes
  }
   if (i>=8)                         //Ja nosacījums izpildās, tad...
   {
   port_data  = 0b00000001;          //Piešķiram jaunu bināro vērtību mainīgajam port_data
   delayptr++;                       //Pārējam uz nākamo elementu – jauna pārtraukuma vērtība
   if (*delayptr>=8)                 //Ja nosacījums izpildās, tad...
   {
    delayptr = arr;                  //Norādām masīva pirmo elementu
   }
   i = 0;                            //Piešķiram i vērtību 0, lai skaitītu diodes no sākuma
   }
}
 return 1;        //Funkcija main atgriež vērtību 1
}
/*****************************************************************************/
```

Programmas pirmteksta algoritma blokshēma

Funkcija main()



Funkcija port_init()

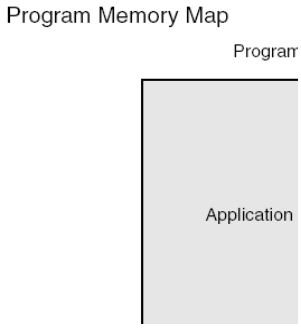


Funkcija init_devices()

IRS(TIMER0_OVF_vect)

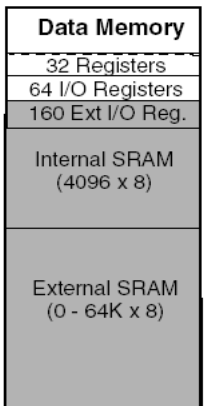
ATmega128 atmiņas veidi

1) **128 KB iebūvētās pārprogrammējamās Flash programmas atmiņas**
   - ✓ Organizēta kā 64k x 16 (jo visas instrukcijas ir 16 vai 32 bitu lielas).
   - ✓ Sadalīta divās daļās:
     - Ielādes programmas (Boot Loader Program) daļa;
     - Pārējo programmu (Application Program) daļa;
   - ✓ Izturība: 10,000 rakstīšanas / dzēšanas cikli.
   - ✓ Ielādes programmas daļā ir aizsargāts ar Boot Lock bitiem.



   - ✓ Atbalsta SPI, JTAG un paralēlās programmēšanas režīmus
   - ✓ Konstantu tabulas var tikt izvietotas visā programmas atmiņas adrešu laukā izmantojot LPM (Load Program Memory) un ELPM (Extended Load Program Memory) instrukcijas.

2) **4 KB iebūvētās SRAM datu atmiņas (iespējams iegūt līdz 64 KB, pieslēdzot ārējo SRAM atmiņu)**
   - ✓ Iekšējā atmiņa: 4352 vietas
     - Pirmās 32 vietas priekš reģistru failā
     - Nākamās 64 vietas priekš standarta I/O atmiņas (var piekļūt ar konkrētu I/O norādījumu vai izmantojot atmiņas piekļuves instrukciju)
     - Nākamie 160 vietas priekš paplašinātās I/O atmiņas (var piekļūt tikai atmiņas piekļuves instrukcijas - LD / LDS / ST / STS / STS instrukcijas)
     - Nākamās 4096 vietas priekš iekšējās SRAM atmiņas



LDD,

   - ✓ Papildu ārējs SRAM
     - Izmērs: līdz 61.184 baitiem (0x1100 ~ 0xffff: aizņem atlikušās adreses vietās 64k adrešu laukā) ≈ 60KBaiti
     - Piekļuves laiks aizņem vienu papildu ciklā par baitu salīdzinot ar pieeju iekšējo SRAM
   - ✓ Datu atmiņas adresācijas režīmi:
     - Tieša (aptver visu datu lauku)
     - Netieša ar pārvietošanu
     - Netieša
     - Netieša ar priekš samazināšanu (pre-decrement) vērā samazinājumu
     - Netieša ar pēc palielināšanu (post-increment) (X, Y, Z reģistrus izmanto automātiskās pirms vērā samazinājumu un pēc porciju)**;**

3) **4 KB iebūvētās EEPROM datu atmiņas**
   - ✓ Adrešu lauks: atsevišķs no SRAM

- ✓ Izturība: 100000 rakstīšanas / dzēšanas cikli
- ✓ EEPROM programmēšanas (rakstīšanas) laiks: 8.5 ms
- ✓ Piekļuves (lasīšanas/rakstīšanas) operāciju CPU var izdarīt izmantojot sekojošus reģistrus:
  - EEPROM adrešu reģistrs
  - EEPROM datu reģistrs
  - EEPROM kontroles reģistrs
- ✓ EEPROM Adrešu reģistra (EEAR)
  - Bits 15 .. 12: rezervētie biti
  - Bits 11 .. 0: EEPROM adrese (4096 baiti)

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	EEAR11	EEAR10	EEAR9	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	X	X	X	
	X	X	X	X	X	X	X	X	

- ✓ EEPROM datu reģistrs (EEDR)
  - Bits 7 .. 0: EEPROM dati

Bit	7	6	5	4	3	2	1
: $1D($3D)							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- ✓ EEPROM kontroles registrs (EECR)
  - Bits 7..4 : rezervēti
  - Bits 3 : EERIE (EEPROM Ready Interrupt Enable)
  - Bits 2 : EEMWE (EEPROM Master Write Enable)
  - Bits 1 : EEWE (EEPROM Write Enable)
  - Bits 0 : EERE (EEPROM Read Enable)

Bit	7	6	5	4	3	2	1
: $1C($3C)		–	–	–	EERIE	EEMWE	EEWE
Read/Write	R	R	R	R	R/W	R/W	R/W

Īss radītāju apraksts

Rādītājs ir datu tips, kura vērtība tieši „norāda" uz atmiņas apgabalu, karā glabājas kāda mainīgā vērtība. Tātad rādītājs ir mainīgais, kas glabā adresi. Rādītāju deklarē sekojoši:

<datu_tips> *ptr1, *ptr2,...,*ptrn;

Kur ptr1, *ptr2,...,*ptrn – rādītājs uz datu_tips tipa mainīgajiem. Šie mainīgie kalpo par atsaucēm uz objektiem ar datu_tips tipu. Tādu tipu sauc par mainīgo-rādītāju bāzes tipu. Piemēram:

int *ptr, *qi; /* rādītāji uz veseliem objektiem */
char *c; /* rādītājs uz teksta objektu */

Deklarēts (neinicializēts) rādītās rāda uz nenoteiktu adresi, tāpēc nepieciešams izmantot atsauces. Rādītāji var nodrošināt atsauci uz iepriekš definētiem objektiem. Tāda objekta adrese var būt noteikta lietojot adresācijas operatoru & (address of operator). Piemēram, apskatīsim mainīgus i un ptr, definētus kā:

int i, *ptr;

Lai rādītājs ptr norādītu uz objekta i adresi veic sekojošu operāciju:

ptr = &i;

Tagad rādītājs ptr norāda uz objekta i adresi. Tāpēc uz objektu i tagad arī ir iespējams atsaukties ar rādītāju ptr un to var veikt izmantojot operatoru *, sekojošā pierakstā: *ptr. Šajā gadījumā vārdus i un *ptr dēvē par pseido-vārdiem.

Masīvu gadījumā sintakse ir ļoti līdzīga:
int *ptr;
int array[3] = {1,2,3};
Lai norādītu uz masīvu:
ptr=&array[];
//vai
ptr=array;

Bet pēc šīs operācijas rādītājs rādīs tikai uz masīva pirmo elementu. Bet kā tad nolasīt nākamo masīva elementu? Šim nolūkam var izmantot rādītāju aritmētiku, proti, rādītāju saskaitīšanu un atņemšanu:

Piemēram:
ptr++; //Pāriet uz masīva nākamo elementu
ptr--; //Pāriet uz masīva iepriekšējo elementu
ptr=ptr+2; //Pāriet par diviem elementiem masīvā uz priekšu
ptr=ptr-2; //Pāriet par diviem elementiem masīvā atpakaļ

Lai mainītu vai nolasītu masīva elementu, kā iepriekš, izmanto operatoru * un rādītāju.

Secinājumi

Trešā laboratorijas darba gaitā es iepazinos ar mikrokontrollera ATmega128 atmiņas veidiem un rādītāju pielietojumu, modificējot otrā laboratorijas darba programmas pirmkodu.

Otrajā laboratorijas darbā, lai nodrošinātu pārtraukumu starp gaismas diožu pārslēgšanām es izmantoju ATmega128 taimeri/skaitītāju0 laika mērīšanai. Šajā darba bija nepieciešamas modificēt iepriekšējā darba kodu, papildinot to ar integer skaitļu masīvu un radītāju, lai varētu uzstādīt noteiktu skrejošas gaismas ilgumu.

Laboratorijas darbs nebija grūts, jo radītāju pielietojums ir labi aprakstīts laboratorijas darba teorētiskajā aprakstā. Es izmantoju ar rādītāja palīdzību aizpildīta masīva elementu vērtības, lai ar katru diožu pārslēgšanas ciklu mainītu pārtraukuma ilgumu starp pārslēgšanām. To realizēt arī nebija grūti, jo vajadzēja tikai atbilstoši papildināt pārtraukuma nosacījumu ar rādītāju un pievienot dažus jaunus nosacījumus, lai kontrolētu uzdevuma izpildi.

Uzskatu, ka laboratorijas darbs ir veiksmīgi izpildīts, jo modificēta programma strādā bez kļūdām un, izmantojot ATmega128 taimeri/skaitītāju 0, skaitļu masīvu un rādītāju, nodrošina pārtraukumu starp gaismas diožu pārslēgšanām atbilstoši masīva elementa vērtībai, kā arī tika iegūtas jaunas zināšanas par ATmega128 atmiņas veidiem.