საქართველოს ტექნიკური უნივერსიტეტი

ო.ქართველიშვილი, ს.ხოშტარია

მიკროპროცესორული სისტემების პროექტირება

ნაწილი 2

2018

შესავალი

ნებისმიერი მექანიკური ან ელექტრონული სისტემა, რომელიც შეიცავს გამომთვლელ მართვის მოწყობილობას, წარმოადგენს ჩაშენებულ სისტემას (Embedded Systems). ყველა გამომთვლელი მოწყობილობა შესდგება ცენტრალური პროცესორის, დამამახსოვრებელი მოწყობილობის, შეტანა/გამოტანის და მოდულებს შორის მაგისტრალების ბლოკებისაგან.

ინტეგრალური სქემები, რომლებიც აერთიანებენ ერთ კრისტალში გამომთვლელის ყველა ზემოთ ნახსენებ ფუნქციონალურ ბლოკებს, წარმოადგენენ მიკროკონტროლერებს (მკ).

N განსხვავება კომპიუტერსა და ჩაშენებულ სისტემებში გამოყენებულ მიკროკონტროლერებს შორის მდგომარეობს შემდეგში: საერთო სარგებლობის კომპიუტერებს, მაგალითად, პერსონალურ კომპიუტერებს, შეუძლია სხვადასხვა პროგრამების შესრულება ტექსტური რედაქტორიდან დაწყებული, რთული გამოთვლითი ამოცანებით დამთავრებული. ჩაშენებულ სისტემაში გამოყენებული მიკროკონტროლერი კი, რომლის გამოთვლის წარმადობა ჩამოუვარდება კომპიუტერს, ასრულებს მხოლოთ მართვის სპეციალურ პროგრამას.

ამჟამად ნახევარგამტარული კომპონენტების ბევრი მწარმოებელი, როგორიც არიან Intel, Microchip, Hitachi, NEC, Atmel, Texas Instruments და სხვა, უშვებენ სხვდასხვა სირთულის მიკროკონტროლერებს. შედარებით მარტივი მკ გამოიყენება საყოფაცხოვრებო ტექნიკასა და სათამაშოებში. უფრო რთული - კომუნიკაციურ მოწყობილობებში, თვითმფრინავების მართვისათვის და სამხედრო ტექნიკაში.

ჩაშენებული სისტემებით ვსარგებლობთ ჩვენს ყოველდღიურ ცხოვრებაში. მაგალითად, მაღვიძარა საათი, ტელეფონი და ჯიბის კომპიუტერი წარმოადგენს მიკროკონტროლერზე აგებულ ჩაშენებულ სისტემებს. ჩვენი სახლი სავსეა მიკროკონტროლერზე აგებული ჩაშენებული სისტემებით: ყავის სახარში, ტელევიზორი დისტანციური მართვის პულტით, სარტეცხი მანქანა, სამზარეულოს კომბაინი, მაღალი სიხშირიანი ღუმელი, დაცვის სიგნალიზაციის სისტემა, მუსიკალური ცენტრი, DVD მუსიკის დამკვრელი, ავტომობილი შეიცავს 10-დან 60-მდე ჩაშენებულ მიკროკონტროლერს, რომლებიც ასრულებენ სხვადასხვა ფუნქციებს. ზემოთ მოყვანილი ჩამონათვალი არის არსებული მოწყობილობების მხოლოდ ნაწილი.

ახლა მივმართოთ ჩვენი ყოფა-ცხოვრების იმ სფეროებს, რომელშიც ჩაშენებული სისტემები ასრულებს მნიშვნელოვან როლს: სახელმწიფოს უსაფრთხოება და სახელმწიფოს მართვის კომუნიკაციის სისტემა ეფუძნება მრავალ მაღალი წარმადობის მქონე ჩაშენებულ სისტემას. კოსმიური სადგურების და თანამგზავრების ბორტზე სხვადასხვა გამოთვლებისა და მართვისათვის გამოიყენებიან ჩაშენებული სისტემები. ნებისმიერი თანამედროვე ჩარხი და გამზომი ხელსაწყო აგრეთვე შეიცავს ჩაშენებულ სისტემას. სამედიცინო-დიაგნოსტიკული კომპლექსების უმრავლესობა გამოკვლევების შედეგების დასამუშავებლად იყენებენ ჩაშენებულ სისტემებს. და ბოლოს, საყოფაცხოვრებო ტექნიკა და საოჯახო გასართობი მულტიმედიური ტექნოლოგიით იყენებენ ჩაშრენებულ სისტემებს.

მიკროკონტროლერის ბაზაზე აგებული გამოთვლითი სისტემის დამუშავება მოითხოვს მათ ცოდნასა და მუშაობის პრინციპების გააზრებას. ერთ-ერთი ცენტრალური საკითხი სისტემის დაგეგმარების დროს არის მიკროკონტროლერის მართვის პროგრამის შექმნა.

თავი 1

Mმიკროკონტროლერების დაპრომირების საფუძვლები

როგორც ზევით იყო ნათქვამი, C ენა გახდა ტექნიკურ მოწყობილობებში ერთ-ერთი ხშირად გამოყენებული დაპროგრამირების მაღალი დონის ენად. ეს აიხსნება იმით, რომ ალგორითმებისა და მონაცემთა წარმოდგენის აბსტრაქტული ხასიათის შენარჩუნებასთან ერთად, რაც დამახასიათებელია მაღალი დონის ენისათვის, C ენა შესაძლებლობას აძლევს დამპროგრამირებელს უშუალოდ მართოს კომპიუტერის აპარატურული საშუალებების და მიკროკონტროლერების კომპონენტები. მაგალითად, C-ში დამპროგრამებელს შესაძლებლობა აქვს უშუალოდ მიმართოს მეხსიერების უჯრედებისა და პერიფერიულ მოდულების შემცველობებს, რაც დამახასიათებელია ასემბლერისთვის.

მიკროკონტროლერის C ენაზე პროგრამირებისათვის უპირველეს ყოვლისა უნდა შესწავლილი იყოს ამ ენის ძირითადი კონსტრუქციები. მინიმალური საჭირო ცნობები C ენის შესახებ მოცემულია სახელმძღვანელოს პირველ თავში. ამ ენის უფრო სრული შესწავლისათვის მკითხველი უნდა გაეცნოს სპეციალურ ლიტერატურას. რეკომენდირებული ლიტერატურის სია ავტორების მიერ მოცემულია წინამდებარე სახელმძღვანელოს ბოლოს.

1.1. C ენის ელემენტები

1.1.1. C-ენაში გამოყენებული მონაცემთა ტიპები

C პროგრამაში გამოყენებული მონაცემები კომპილატორის მიერ გადაისახება ორობით ფორმატში და ინახება მიკროკონტროლერის მეხსიერების გამოყოფილ ბაიტურ უჯრედებში, რომელთაც ენიჭებათ სახელები. ვინაიდან ხშირ შემთხვევაში, მათში შეგვიძლია რიცხვების მნიშვნელობების ცვლილებები, პროგრამისათვის მეხსიერების უჯრედები წარმოადგენენ ცვლადებს.

ამგვარად, **ცვლადი** მეხსიერების უჯრედების ნაკრებია, სადაც შეიძლება რიცხვის შენახვა და ასევე მათი მნიშვნელობების შეცვლა. ცვლადებს აქვს მისამართი და სახელი.

კონსტანტა - ესეც ცვლადია იმ განსხვავებით, რომ მისი შემცველობის შეცვლა არ შეიძლება.

კონსტანტებს წარმოადგენენ პროგრამაში გამოყენებული მონაცემები. C ენაში ასხვავებენ ოთხი ტიპის კონსტანტას: მთელი კონსტანტები, კონსტანტები მცურავი მძიმით, სიმბოლური კონსტანტები და სტრიქონული ლიტერალები. კონსტანტის ტიპი განსაზღვრავს მიკროკონტროლერის მეხსიერებაში ბაიტების რაოდენობას, რომლებიც გამოიყოფიან ტრანსლატორის მიერ მის შესანახად.

მთელი კონსტანტა არის რიცხვი, რომელიც წარმოადგენს მთელ სიდიდეს ერთ-ერთ ფორმატში: ათობით, რვაობით, თექვსმეტობით ან ორობითში.

ათობითი კონსტანტა შესდგება ერთი ან რამოდენიმე ათობითი ციფრისაგან, ამასთან, პირველი ციფრი არ უნდა იყოს ნული (წინააღმდეგ შემთხვევაში რიცხვი აღიქმება როგორც რვაობითი).

რვაობითი კონსტანტა შესდგება რიცხვის წინ დაწერილი აუცილებელი ნულისგან და ერთი ან რამოდენიმე რვაობითი ციფრისაგან.

თექვსმეტობითი კონსტანტა იწყება აუცილებელი თანმიმდევრობით 0X ან 0x და შედგება ერთი ან რამოდენიმე თექვსმეტობითი ციფრისაგან.

ორობითი კონსტანტა ჩაიწერება ორობითი სიმბოლოების საშუალებით, რომელთაც წინ უნდა ახდეს მიმდევრობა 0b.

მაგალითად, მთელი რიცხვები შეიძლება ჩაიწეროს შემდეგი სახით:

- ათობით ფორმაში: 12; 234; -5493;
- ორობით ფორმაში 0b პრეფიქსით: 0b101001:
- თექვსმეტობით ფორმაში 0X პრეფიქსით: 0X5A;
- რვაობით ფორმაში 0 პრეფიქსით: 07775.
- უარყოფითი მთელი კონსტანტის ჩაწერისათვის გამოიყენება „–„ ნიშანი კონსტანტის წინ (რომელსაც უწოდებენ უნარულ მინუსს). მაგალითად, -0x2A; -077; -16.

მთელ კონსტანტას ენიჭება ტიპი, რომლითაც განისაზღვრება ტრანსლატორის მიერ ორობით კოდში გარდასახული კონსტანტის სიგრძე (თანრიგების რაოდენობა). ცვლადების ტიპი უნდა მიეთითოს მის გამოყენებამდე პროგრამაში. ამის საფუძველზე ტრანსლატორი მიკროკონტროლერის მეხსიერებაში დაარეზერვირებს შესაბამისი რაოდენობის უჯრედებს მის ჩასაწერად. აქვე აღვნიშნავთ, მეხსიერების უჯრედი რვა ორობით თანრიგს (ბაიტს) შეიცავს.

განვიხილოთ მეხსიერების არის განაწილება სხვადასხვა ტიპის კონსტანტებისათვის:

- char ტიპის კონსტანტის შემთხვევაში ტრანსლატორი გამოყოფს ერთ ბაიტიან უჯრედს, რომელშიც შეიძლება მოვათავსოთ რიცხვი 0-255 დიაპაზონიდან.
- ათობითი კონსტანტები განიხილება როგორც სიდიდეები ნიშანთან ერთად და მიენიჭებათ int ან long ტიპი, გამომდინარე კონსტანტის მნიშვნელობიდან - თუ კონსტანტის მნიშვნელობა არ აღემატება 32768-ს, მაშინ მას უნდა მივანიჭოთ int ტიპი, წინააღმდეგ შემთხვევაში - Long. ვინაიდან, პირველ შემთხვევაში მონაცემების მანქანურ კოდში გამოსახვისათვის საჭიროა 16 ორობითი თანრიგი, int ტიპის კონსტანტისათვის ტრანსლატორმა მეხსიერებაში უნდა დაარეზერვიროს ორი უჯრა, ხოლო Long ტიპის კონსტანტისათვის - მეხსიერების ოთხი უჯრა (32 ბიტი).
- რვაობით ან თექვსმეტობით კონსტანტებს მიენიჭება int, unsigned int, long, unsigned long ტიპები. სამომსახურო სიტყვა unsigned (უნიშნო) მთელი რიცხვების სპეციპიკაციის წინ განსაზღვრავს მის უფროსი თანრიგის გამოყენებას. ნიშნიანი მთელი რიცხვებისათვის უფროსი თანრიგი გამოიყენება ნიშნის ჩასაწერად, რაც იწვევს უჯრედში რიცხვის დასაშვები მნიშვნელობების შემცირებას. უნიშნო მთელ რიცხვებში უფროსი თანრიგი განიხილება როგორც რიცხვის დამატებითი თანრიგი, შედეგად მეხსიერების უჯრედში ჩასაწერი რიცხვების დიაპაზონი დიდდება. უთქმელობით ე.ი. unsigned სიტყვის გარეშე char, int და long ტიპის კონსტანტები ყოველთვის ითვლებიან ნიშნიანად.

ცხრილ 1.1-ში მოცემულია სხვადასხვა ტიპის კონსტანტების მნიშვნელობები.

ცხრილი 1.1

თექვსმეტობითი კონსტანტების დიაპაზონი	რვაობითი კონსტანტების დიაპაზონი	ტიპი
0x0-0x7F	0-077	char
0x0-0xFF	0-0377	unsigned char
0x0-0x7FFF	0_077777	int
0X8000-0XFFFF	0100000-0177777	unsigned int
0X10000_0XFFFFFFF	0200000_017777777777	long
0X80000000_0XFFFFFFFF	0200000000000_037777777777	unsigned long

მცურავი მძიმით გამოსახული რიცხვები შედგება მთელი და წილადი ნაწილისაგან და (ან) ექსპონენტასაგან. ისინი წარმოდგენილი არიან როგორც ნიშნიანი სიდიდეები ერთმაგი (float) ან ორმაგი (double) სიზუსტით. უარყოფითი სიდიდეებისათვის კონსტანტას წინ მიეწერება „- „ ნიშანი.

მაგალითად, 115,75; 1,5E-2; -0,025; 0,75; 0,85E2.

მცოცი მძიმით წარმოდგენილ რიცხვებისათვის ტრანსლატორმა უნდა გამოყოს მეხსიერებაში - 32 ბიტი (ოთხი უჯრა) float ტიპისათვის და 64 ბიტი (რვა უჯრა) double ტიპისათვის.

სიმბოლური კონსტანტა არის სიმბოლო, რომელიც ჩასმულია აპოსტროფებში. სიმბოლური კონსტანტა ტრანსლატორის მიერ გარდაისახება ორობით კოდში და ენიჭება char ტიპი, ვინაიდან კომპუტერში სიმბოლოები წარმოდგენილი უნდა იყვნენ რვ ბიტიანი კოდით (ASCII კოდში)

მაგალითად, 'Q' არის Q ასო.

სტრიქონული კონსტანტა (ლიტერალი) - სიმბოლოების თანმიმდევრობაა (ლათინური პატარა და დიდი ასოების და ასევე ციფრების ჩათვლით), ჩასმული ბრჭყალებში.

მაგალითად: „Heloo".

სტრიქონული კონსტანტის თვითეული სიმბოლო გარდასახვის შემდეგ ინახება მიკროკონტროლერის მეხსიერების მეზობელ უჯრებში. ტრანსლატორი სტრიქონის ბოლოს უმატებს სტრიქონის დამთავრების ნიშნაკს \0. სტრიქონის თვითეულ სიმბოლოს ენიჭება char ტიპი.

1.1.2 იდენტიფიკატორი

იდენტიფიკატორი ეწოდება ციფრების, ასოების და აგრეთვე სპეციალური სიმბოლოების თანმიმდევრობას, იმ პირობით, რომ პირველ პოზიციაზე უნდა იდგეს ასო ან სპეციალური სიმბოლო. იდენტიფიკატორის შექმნისათვის შეიძლება გამოყენებული იქნას ლათინური ალფავიტის დიდი და პატარა ასოები, სპეციალური სიმბოლოს როლში შეიძლება გამოყენებული იყოს ხაზგასმის სიმბოლო (_). ორი სხვადასხვა იდენტიფიკატორი

განსხვავებულად ითვლება, თუ გამოყენებულია დიდი და პატარა ერთნაირი სიმბოლო. მაგალითად, ABC და abc, A128B და a128b.

იდენტიფიკატორი გამოიყენება ცვლადის, ფუნქციის, სტრუქტურის და სხვათა აღსანიშნავად. მისი გამოყენება შეიძლება შემდგომში პროგრამის ოპერატორებში. იდენტიფიკატორად არ უნდა იყოს ნახმარი საკვანძო სიტყვები, დარეზერვირებულ სიტყვები ან C ენის ტრანსლატორის ბიბლიოთეკის ფუნქციის სახელები.

1.1.3 საკვანძო სიტყვები

საკვანძო სიტყვები არიან დარეზერვირებული იდენტიფიკატორები, რომლებსაც იყენებს C ენის ტრანსლატორი თავის მუშაობის პროცესში.

ქვევით მოყვანილია საკვანძო სიტყვების სია.

auto	double	int	struct	break	else	long	switch
register	tupedef	char	extern	return	void	case	float
unsigned	default	for	signed	union	do	if	sizeof
volatile	continue	enum	short	while			

გააჩნია უთქმელობის პრინციპი, რის გამოც პროგრამაში გამოყენებული ყოველი ცვლადის გამოცხადების დროს უნდა მგარდა ჩამოთვლილი საკვანძო სიტყვებისა C ენის გამოყენებულ ვერსიაში დარეზერვირებულ სიტყვებს წარმოადგენენ აგრეთვე: asm, fortran, near, far, cdecl, huge, paskal, interapt. asm, fortran, cdecl, paskal საკვანძო სიტყვები ემსახურებიან სხვა ენებზე დაწერილი პროგრამების კავშირს C პროგრამასთან. საკვანძო სიტყვები არ შეიძლება გამოყენებული იყოს ჩვენს მიერ შედგენილ იდენტიფიკატორის როლში.

1.1.4 პროგრამის ტექსტში კომენტარების გამოყენება

C პროგრამის ტექსტში შეიძლება გამოყენებული იყოს კომენტარები. კომენტარები უნდა განსთავსდეს /* და */ სიმბოლოებს შორის (კომენტარის დასაწყისი და დასასრული), ერთ ან რამოდენიმე სტრიქონში ან „//“ სიმბოლოს შემდეგ ერთ სტრიქონში. მაგალითად,

/ * კომენტარები პროგრამაში */ ან

// ალგორითმის დასაწყისი

1.1.5 მონაცემთა გამოცხადება

C ენაში ცვლადის პროგრამაში გამოყენებისათვის უპირველეს ყოვლისა იგი უნდა შევქმნათ. ცვლადის შექმნა ნიშნავს მისი შენახვისათვის მეხსიერების ერთი ან რამოდენიმე უჯრედის გამოყოფას (ცვლადის ტიპისაგან დამოკიდებით) და მისთვის სახელის მინიჭებას, რომლითაც შესაძლებელია მასთან მიმართვა. C ენაში აუცილებელია აგრეთვე ცვლადის ტიპის მითითება ტრანსლატორისთვის მეხსიერების უჯრედში შენახული მნიშვნელობის ინტერპრეტირება. მაგალითად, არის თუ არა რიცხვი ნიშნიანი, მეხსიერების რამდენი უჯრედი უნდა იყოს გამოყენებული რიცხვის ჩასაწერად და სხვა. C ენას სხვა ენებთან

შედარებით არ გააჩნია უთქმელობის პრინციპი, რის გამოც პროგრამაში გამოყენებული ყოველი ცვლადის გამოცხადებასთან ერთად უნდა მიეთითოს მათი ტიპი.

მთაელი ტიპი	მცურავი ტიპი.
char	float
int	double
short	long double
long	
signed	
unsigned	

მაგალითად:

Unsigned char - ინახავს რიცხვის მნიშვნელობას 0-დან 255-მდე (ანუ ოროობით baits);
Unsigned int - ინახავს რიცხვის მნიშვნელობას 0_დან 65535-მდე (ორ ბაიტს);
Unsigned long - ინახავს რიცხვის მნიშვნელობას 0-დან 4294967295-მდე (ოთხ ბაიტს).
Unsigned char მაგივრად შეიძლება დაიწეროს char, რამდენადაც ტრანსლატორი თვლის , რომ უთქმელობით char ტიპის რიცხვი უნიშნოა.
Signed char - ნიშნავს ნიშნიან ცვლადს, წარმოდგენილს _128-დან 127-მდე დიაპაზონში.
Signed და Unsigned საკვანძო სიტყვები მიუთითებენ იმაზე, თუ როგორ ხდება გამოცხადებული ცვლადის ორობოით კოდში გარდასახვის შემდეგ უფროსი ბიტის ინტერპრეტირება, როგორც ზევით იყო განხილული. იმ შემთხვევაში, თუ ცვლადის განსაზღვრაში მოცემულია მხოლოდ **Signed** ან **Unsigned** საკვანძო სიტყვა, მაშინ იგი განიხილება როგორც int ტიპის ცვლადი.
მაგალითად

```
usigned int n ;
unsigned int b;
int c; (იგულისხმება signed in c );
unsigned d; (იგულისხმება unsigned int d);
signed f;    (იგულისხმება signed int f)
```

ნებისმიერი ცვლადი შეიძლება გამოცხადდეს არამოდიფიცირებადად. ეს სრულდება ტიპის სპეციპიკატორისისათვის საკვანძო სიტყვის, **const** დამატებით. **Const** ტიპის ობიექტები წარმოადგენენ ცვლადებს რომელთა მნიშვნელობის შეცვლა არ შეიძლება, ე.ი. ამ ცვლადს არ შეიძლება მიენიჭოს ახალი მნიშვნელობა. იმ შემთხვევაში, თუ **const** სიტყვის შემდეგ ტიპის სპეციფიკატორი არ არის იგულისხმება, რომ შესაბამისი ცვლადი არის **int** ტიპის. ხოლო თუ საკვანძო სიტყვა **const** დგას შედგენილი ტიპის ცვლადების (მასივი, სტრუქტურა, ჩამონათვალი) გამოცხადების წინ, თვითოეული ელემენტი აგრეთვე უნდა იყოს არამოდიფიცირებადი.

magaliTad:

```
const  double  A=2.128E-2;
const B=286; (იგულისხმება  const  int B=286)
const  char  string_ constant [ ] =″this  is  a  string  constant ″
```

იმ ცვლადებისათვის, რომლებიც წარმოდგენილი არიან მცურავი მძიმით, იყენებენ მოდიფიკატორების შემდეგ ტიპებს: float, double, long double.

მანქანურ კოდში float ტიპის ცვლადები იკავებენ 4 ბაიტს. **float** ტიპის ცვლადების მნიშვნელობების დიაპაზონი მიახლოებით ტოლია 3.14E-38-დან 3.14 E+38-მდე. **Double** ტიპის ცვლადი მეხსიერებაში იკავებს 8 ბაიტს. მნიშვნელობების დიაპაზონი ტოლია 11.7EM-308-დან 1.7E+308-მდე.

მაგალითი:

```
oat f, a, b;
ouble x,y;
```

ცვლადების გამოცხადების ფორმატს შემდეგი სახე აქვს::

[<storage modifier>]<type definition><identifier>

<storige modifier> -წარმოადგენს ცვლადის კლასის სპეციფიკატორს. განისაზღვრება C ენის ოთხი საკვანძო სიტყვიდან ერთ-ერთით: auto, extern, register, static და მიოთითებს თუ როგორ მოხდება მეხსიერების გამოყოფა განცხადებული ცვლადისათვის, ერთის მხრივ, და მეორე მხრივ, როგორია ამ ცვლადის გამოყენების არე ანუ პროგრამის რომელი ნაწილიდან შეიძლება მასთან მიმართვა. ეს არ არის აუცილებელი ელემენტი და მისი გამოყენების საჭიროება მხოლოდ ზოგიერთ შემთხვევაში დგება.

<extern>- თუ ცვლადი შეიძლება გამოყენებული იყოს პროგრამის სხვა ფაილებში. იგი წარმოადგენს **გლობალურ ცვლადს;**

<static>_ თუ ცვლადი ლოკალურია, მაშინ ის გამოცხადებულია ფუნქციაში და გამოიყენება მხოლოდ ამ ფუნქციაში. ფუნქციის შესრულების შემდეგ სრულდება მისი მნიშვნელობის შენახვა ამ ფუნქციის შემდგომ გამოძახებამდე.

<register> - ცვლადი განთავსდეს მკ-ს რეგისტრში.

გლობალური ცვლადების გამოცხადება ხდება პროგრამის დასაწყისში მის გამოჩენამდის რომელიმე ფუნქციის ტექსტში. გლობალური ცვლადები მისაწვდომია პროგრამის ნებისმიერი ადგილიდან.

ლოკალური ცვლადების გამოცხადება ხდება ფუნქციის დასაწყისში, ფიგურული ფრჩხილის შემდეგ. .

<type defination>- არის მონაცემთა ტიპის სპეციფიკატორი, რომელიც შეიძლება ჩაიწეროს ცვლადში. მთელი ტიპის მონაცემებისათვის გამოიყენება სხვადასხვა საკვანძო ციტყვა, რომლებიც განსაზღვრავენ ცვლადისათვის გამოყოფილ მეხსიერების არეს ზომას, როგორც ეს ზევით იყო განხილული.

<identifier>- ცვლადის სახელია. მაგალითად: chemi cvladi .
ცვლადებისათვის მიღებულია პატარა ასოების გამოყენება. ცვლადების სახელების განსხვავებისათვის ფუნქციის სახელებისაგან, ცვლადების სახელები უნდა დაიწყოს ასოებით, ხოლო ფუნქციის სახელი (გარდა main ფუნქციისა) ხაზგასმული სიმბოლოთი.
მაგალითად: **chemi cvladi, _tqveni funqcia.**

პროგრამის სტარტის და რესტარტის დროს გლობალური ცვლადები, ასევე ლოკალური ცვლადები **static** სხვა მნიშვნელობები.
ქვევით ნაჩვენებია ცვლადების გამოცხადების რამოდენიმე მაგალითი:
unsigned char chemi_cvladi=34; /* chemi_cvladi განსაზღვრულია როგორც unsigned char, რომელსაც მინიჭებული აქვს 34 (შეიძლება მინიჭების გარეშეც */
Unsigned Int big_ peremen=32634; /* big_ peremen გამოცხადებულია როგორც Unsigned Int */.

M1.1.6 მასივი

მასივი განსაზღვრავს ერთნაირი ტიპის მონაცემთა ობიექტების უწყვეტ ანაწყობს. მასივის მაჩვენებლად გამოიყენება კვადრატული ფრჩხილები, მოთავსებული ცვლადის იდენტიფიკატორის შემდეგ. მასივის გამოცხადების დროს კვადრატულ ფრჩხილებში მიეთითება მასივის ელემენტების რაოდენობა (მისი ზომა), ხოლო გამოსახულებებში გამოყენების დროს - საჭირო ელემენტის ნომერი. მასივები შეიძლება შეიცავდენ ელემენტებს მონაცემთა ზემოთ განხილულ ნებისმიერ ფორმატში წარმოდგენით (char, int, float double და სხვა). M

მასივის გამოყენების შემთხვევაში იგი წინასწარ უნდა გამოცხადდეს პროგრამის დასაწყისში.
მასივის გამოცხადებას გააჩნია ორი ფორმატი:

- <ტიპის სპეციფიკატორი><იდენტიფიკატორი>[კონსტანტური გამოსახულება]
- <ტიპის სპეციფიკატორი><იდენტიფიკატორი>[ ].

მასივის ელემენტები არ შეიძლება იყოს ფუნქციები და void ტიპის ცვლადები..

კონსტანტური გამოსახულება კვადრატულ ფრჩხილებში განსაზღვრავს მასივის ელემენტების რაოდენობას. კონსტანტური გამოსახულება მასივის გამოცხადების დროს შეიძლება გამოტოვებული იყოს შემდეგ შემთხვევაში:

- გამოცხადების დროს ხდება მასივის ინციალიზება (მის ელემენტებში მონაცემთა შეტანა);
- მასივი გამოცხადებულია, როგორც მითითებული მასივი, რომელიც ცხადად არის განსაზღვრული სხვა ფაილში;
- მასივი გამოცხადებულია როგორც ფუნქციის ფორმალური პარამეტრი.

ქვევით მოყვანილია მასივის გსნსაზღვრა, რომელიც შეიცავს ორბაიტიანი მთელი რიცხვების 10 ელემენტს.

Int list[10];

მასივის ზემოთ მოყვანილი განსაზღვრის მიხედვით მკ-ს მეხსიერებაში დარეზერვირდება ბაიტიანი 20 უჯრედი.

პროგრამაში მასივის ყოველ ელემენტთან მიმართვა შესაძლებელია ინდექსის საშუალებით. C ენაში ინდექსი ავტომატურად აითვლება 0-დან, ამიტომ ჩვენს მაგალითში ინდექსს შეიძლება მიენიჭოს მნიშვნელობა 0-დან 9-მდე ჩათვლით.

როგორც ნებისმიერი ცვლადი, მასივი ასევე შეიძლება გამოცხადდეს მისი მნიშვნელობების ერთდროული ინიციალიზებით. ამისათვის მინიჭების ოპერაცია „=“ მარჯვენა ნაწილში ფიგურულ ფრჩხილებში, მძიმით გამოყოფილად ჩამოთვლილი უნდა იყოს მასივის ყველა ელემენტის მნიშვნელობა:

int list [10]= {1,2,3,4,5,6,7,8,9}.

თუ მასივის განსაზღვრის დროს მისი ზომა არ არის მითითებული, ასეთი მასივი აუცილებლად უნდა იყოს ინიცირებული. ამ შემთხვევაში კომპილატორი ავტომატურად განსაზღვრავს მასივის ზომას ინიცირებული ელემენტების რაოდენობის შესაბამისად. მაგალითად, მასივის განსაზღვრებაში int list[]={1,2,3,4,5,6,7,8,9,10} მასივის ზომა ტოლი იქნება ინიციალიზებული ელემენტების რაოდენობის ანუ 10.

ასევე შესაძლებელია მითითებული ზომის მასივის ნებისმიერი რაოდენობის პირველი ელემენტების ინიციალიზება.

მაგალითად, განსაზღვრაში int list[10]={1,2,3,4,5}, სრულდება 10 ელემენტისაგან შემდგარი მასივის პირველი ხუთი ელემენტის ინიციალიზირება.

სიმბოლოებიანი მასივებისათვის შესაძლებელია გამოვიყენოთ შემოკლებული ჩანაწერი, რომლის დროს მინიჭების ოპერაციის მარჯვენა ნაწილში ბრჭყალებში იწერება სიმბოლოების სტრიქონი. ამასთან თუ არ არის მითითებული მასივის ზომა, კომპილატორი მას თვლის სტრიქონში სიმბოლოების ტოლად ერთის დამატებით, ვინაიდან C ენაში სიმბოლოების სტრიქონი თავდება ASCII კოდში წარმოდგენილი ნულით, რომელსაც კომპილატორი ავტომატურად უმატებს სტრიქონს.

მაგალითად,

Char message[6]=”Smile”;

Char message[]=”Smile”;

მასივის პირველი ელემენტი masage[0] შეიცავს „S” სიმბოლოს კოდს, bolo elementi message[5] – სტრიქონის დაბოლოების კოდს \0.

C ენაში განსაზღვრულია მხოლოდ ერთგანზომილებიანი მასივები, მაგრამ რადგანაც მასივის ელემენტი შეიძლება იყოს მასივი, მისი განსაზღვრა შესაძლებელია როგორც მრავალგანზომილებიანი მასივის. ისინი ჩაიწერებიან კონსტანტური გამოსახულების სიაში, მასივის იდენტიფიკატორის შემდეგ. ამასთან ყოველი კონსტანტური გამოსახულება ჩასმულია თავის კვადრატულ ფრჩხილებში.

მაგალითად:

int a[2][3]; წარმოდგენილია როგორც მატრიცა

a[0][0] a[0][1] a[0][2]

a[1][0] a[1][1] a[1][2]

ერთგანზომილებიანი მასივების ანალოგიურად, მრავალგანზომილებიანი მასივებისთვისაც შესაძლებელია ინიციალიზაცია მის გამოცხადების ეტაპზე.

მაგალითად:

int matrix[2][3]={1,2,3}{4,5,6}

მრავალგანზომილებიან მასივის ერლემენტებთან მიმართვა რამოდენიმე ინდექსის მითითებით. მაგალითად, ქვევით ნაჩვენები გამოსახულების შესრულების შედეგად m ცვლადი მიიღებს 6-ის მნიშვნელობას.

m=matrix [1][2]; .

1.1.7 მიმთითებელი

მიმთითებელი ეწოდება ცვლადს, რომელიც განკუთვნილია სხვა ცვლადის მისამართის შენახვისათვის. მაგალითად, ერთ ბაიტიანი ცვლადის მიმთითებელი შეიცავს char ტიპის ცვლადის მისამართს. მიმთითებელი შეიძლება იყოს გამოყენებული მასივებთან, სტრუქტურებთან და მონაცემთა სხვა ბლოკებთან მანიპულირების გამარტივების მიზნით. აგრეთვე მიკროკოპროცესორული სისტემების მეხსიერების კონკრეტულ ფიზიკურ უჯრედებთან მიმართვისათვის.

მიმთითებელი-ცვლადის განსაზღვრისათვის გამოიყენება აღნიშვნა " * ".

მაგალითად:

int *ptr.

მოყვანილი ჩანაწერი ნიშნავს, რომ ცვლადი ptr შეიცავს int ტიპის ორბაიტიანი ცვლადის უფროსი ბაიტის მისამართს.

ცვლადი-მიმთითებელი შესაძლებელია განსაზღვრული იყოს სხვადასხვა ტიპის ცვლადის დასამისამართებლად: მთელი ერთბაიტიანი char და ორბაიტიანი int ტიპის, ერთგანზომილებიანი და მრავალგანზომილებიანი მასივები char და int, ცალკეული ცვლადები ან მასივები მცოცი მძიმით წარმოდგენილი ელემენტებით.

მიმთითებლის ტიპის სპეციპიკაცია ტრანსლატორ ატყობინებ ობიექტის ზომისა და ტიპის შესახებ, რომელზეც უჩვენებს მიმთითებელი.

მოვიყვანოთ მიმთითებლის გამოყენების მაგალითი:

```
1.void main(void)
2.{
3.char *ptr;
4. static char message[] = «what a wonderful day!»;
5.ptr = message;
6.while (*ptr != '\0')
7.{
8.putchar(*ptr);
9.ptr++;
10.}
11.}
```

Aმოყვანილ მაგალითში ცვლადი ptr განსაზღვრულია როგორც char ტიპის ერთბაიტიანი ცვლადის მიმთითებელი (სტრიქონი 3). ამიტომ ცვლადი ptr უნდა შეიცავდეს ერთბაიტიან ცვლადის მისამართს. მე-4 სტრიქონში განსაზღვრული და ინიცირებულია ერთბაიტიანი ცვლადების მასივი massage. ეს მასივი შეიცავს 19 ელემენტს:18 სიმბოლო ASCII კოდში და ერთი ბაიტი სტრიქონის დასასრული. 5 სტრიქონში არსებული გამოსახურების მიზანია ptr ცვლადში ჩაიწეროს massage მასივის საწყისი მისამართი ე.ი. „w" სიმბოლოს მისამართი. ამ მაგალითში მიმთითებელი ptr გამოიყენება putchar ფუნქციაში ერთი სიმბოლოს კოდის გადასაცემად, რომელიც გამოყვანილი იქნება ეკრანზე. შემდეგ ptr იზრდება ერთით, რითაც ფორმირდება მასივის შემდეგი ელემენტის მისამართი. while ციკლის პირობის თანახმად ეკრანზე სიმბოლოების გამოყვანა დასრულდება, თუ სიმბოლოების თანმიმდევრობაში აღმოჩნდა სტრიქონის დამთავრების კოდი.

1.2 C ენის ოპერატორები

1.2.1 ოპერანდები და ოპერატორები

ოპერანდებზე რაიმე ოპერაციების შესრულებას, რომლის შედეგად ვიღებთ რაიმე მნიშვნელობას, გამოსახულება ეწოდება.

ოპერაციების ნიშნები განსაზღვრავენ მოქმედებას, რომლებიც უნდა შესრულდეს ოპერანდებზე.

გამოსახულების ყოველი ოპერანდა შეიძლება წარმოადგენდეს აგრეთვე გამოსახულებას. გამოსახულების მნიშვნელობა დამოკიდებულია გამოსახულებაში ოპერაციის ნიშნებისა და მრგვალი ფრჩხილების განლაგებაზე, აგრეთვე ოპერაციის შესრულების პრიორიტეტებზე. ოპერანდა შეიძლება იყოს კონსტანტა, ლიტერალი, იდენტიფიკატორი, ფუნქციის გამომძახებელი, ინდექსური გამოსახულება ან უფრო რთული გამოსახულება, შედგენილი ოპერანდების კომბინაციით, ოპერაციის ნიშნებით და მრგვალი ფრჩხილებით. ნებისმიერ ოპერანდას, რომელსაც აქვს კონსტანტის მნიშვნელობა კონსტანტური გამოსახულება ეწოდება. ყოველ ოპერანდს შეესაბამება ტიპი.

მრგვალი ფრჩხილები გამოიყენება ოპერანდებზე მოქმედებების შესრულებით თანმიმდეობის განსაზღვრისათვის. უთქმელობით ოპერაციების შესრულება ხდება გარკვეული თანმიმდევრობით. მაგალითად, გამოსახულება 2*23+15, ჯერ სრულდება გამრავლება, შემდეგ შეკრება. მივიღებთ 61-ს. თუ გვინდა ჯერ 23-ს მიუმატოთ 15 და შემდეგ მიღებული ჯამი გავამრავლოთ 2-ზე, უნდა ვისარგებლოთ მრგვალი ფრჩხილებით, რომლის საშუალებით იცვლება ოპერაციის შესრულების თანმიმდევრობა 2*(23+15). გამოთვლის შედეგი გვექნება 76.

1.2.2. მინიჭების ოპერატორი

მინიჭება C ენაში წარმოადგენს გამოსახულებას და ამ გამოსახულების მნიშვნელობა არის სიდიდე , რომელიც მას მიენიჭება. იმისათვის, რომ მოხდეს რიცხვის განთავსება ცვლადში (რეგისტრში) C -ენას აქვს „=“ მინიჭების ოპერატორი. ეს სიმბოლო ნიშნავს გამოითვალოს მინიჭების ოპერატორის მარჯვნივ მყოფი გამოსახულება და შედეგი ჩაიწეროს მინიჭების ოპერატორის მარცხნივ მყოფ ცვლადში.

მაგალითებჰი:

1) **PORTB = PINB+34;** /* ეს სტრიქონი C ენაზე ნიშნავს წავიკითხოთ PINB ცვლადის (რეგისტრის) მნიშვნელობა, დაემატოს მას 34 და შედეგი შევიტანოთ PORTB ცვლადში */
2) **PEREMEN=PINC** /* C ენაზე ეს სტრიქონი ნიშნავს: წავიკითხოთ PINC ცვლადის მნიშვნელობა და შედეგი ჩავწეროთ **PEREMEN** ცვლადში */
3) **PORTB** = **0x23;** /* C ენაზე ნიშნავს **PORTB** ცვლადს მიენიჭოს **0x23** მნიშვნელობა. წინათ ჩაწერილი მნიშვნელობა იშლება*/
4) სტრიქონი, რომელშიც სიმბოლო“ **= “** მოთავსებულია ცვლადის მარჯვნივ, & სიმბოლოს შემდეგ

PORTB & = 0x23;

/*C ენაზე ნიშნავს: **PORTB** ცვლადის მნიშვნელობაზე სრულდება ლოგიკური ოპერაცია „და“ (ლოგიკური გამრავლება) **0x23** კონსტანტასთან. მიღებული შედეგი იწერება ისევ **PORTB** ცვლადში.

ნაცვლად & ოპერაციისა შეგვიძლია გამოვიყენოთ სხვა ლოგიკური ოპერაციები: I - “an” (ლოგიკური შეკრება);

~ – ბიტების ინვერტირება (ცვლის რეგისტრის ბიტებს საწინააღმდეგო მნიშვნელობის ორობითი სიმბოლოთ)

მინიჭების ოპერატორთან გამოიყენება შემოკლება:

გრძელი ჩანაწერი		შინაარსი		შემოკლებული ჩანაწერი
`x = x + 1;`		x-ს დაემატოს 1		`x++;` ან `++x;`
`x = x - 1;`		x-ს გამოაკლდეს 1		`x--;` ან `--x;`
`x = x + y;`		x-ს დაემატოს y		`x += y;`
`x = x - y;`		x-ს გამოაკლდეს y		`x -= y;`
`x = x * y;`		x გამრავლდეს y-ზე		`x *= y;`
`x = x / y;`		x გაიყოს y-ზე		`x /= y;`
`x = x % y;`		X გაიყოს y-ზე		`x %= y;`

C ენაში არის ოპერაციები, რომლებიც ცვლიან ცვლადის მნიშვნელობას მინიჭების ოპერატორის გარეშე:

PORTA++; /* ეს სტრიქონი C ენაზე ნიშნავს: PORTA ცვლადის მნიშვნელობას დაემატოს 1 და შედეგი კვლავ PORTA-ში ჩაიწეროს. ამბობენ - მოხდეს PORTA ცვლადის ინკრიმენტი */

PORTA--; /* ეს სტრიქონი C ენაზე ნიშნავს საპირისპირო მოქმედებას: შესრულდეს დეკრემენტი = PORTC ცვლადის მნიშვნელობას გამოაკლდეს 1*/

როდესაც ინკრემენტი ან დეკრემენტი გამოიყენება გამოსახულებაში, მთავარია ვიცოდეთ რა მდგომარეობა უკავია ტოლობის მარჯვენა ნაწილში + ან - ნიშანს - ცვლადის წინ, თუ ცვლადის შემდეგ. მაგალითად:

1. a=4;

b=7;

a= b++; /* C ენაზე ეს სტრიქონი ნიშნავს: **b** ცვლადის მნიშვნელობა მივანიჭოთ a-s, შემდეგ b ცვლადის მნიშვნელობას დაემატოს 1 და შედეგი შეინახოს b-ში. ამრიგად, a ცვლადი მიიღებს 7-ის მნიშვნელობას, b-ში - რიცხვ 8-ს */

2. **a=4;**

b=7;

a=++b; /* C ენაში ეს სტრიქონი ნიშნავს: b ცვლადს დაემატოს 1 და შედეგი შევინახოთ b=ში, იგივე მნიშვნელობა ჩაიწეროს a ცვლადში. ამგვარად, a და b ცვლადებში იქნება რიცხვი 8.

1.2.3 არითმეტიკული ოპერაციების ოპერატორები

x + y // შეკრება;
x - y // გამოკლება;
x * y // გამრავლება;
x/y /* გაყოფა, თუ ოპერანდები მთელი რიცხვებია შედეგიც მთელი რიცხვი იქნება, გადაგდებული წილადი ნაწილით, დამრგვალების გარეშე. ე.ი. თუ გაყოფის შედეგად მიღებულია 6,2341 ან 6,94, შედეგი იქნება მთელი რიცხვი 6. იმ შემთხვევაში, თუ რიცხვი წარმოდგენილია მცურავი მძიმით ანუ **float** ან **double** ტიპის, მაშინ შედეგიც წარმოდგენილი იქნება მცურავი მძიმით, წილადი ნაწილის გადაგდების გარეშე. მაგალითად, 131,9739/6,18 ოპერაციის შესრულების შედეგად მივიღებთ 21,355*/.
x % y // გაყოფის შედეგად აიღება ნაშთი მთელი რიცხვების გარეშე

5 / 2 // მოგვცემს 2-ს;
5 % 2 // მოგვცემს 1-ს;

75 / 29 // მოგვცემს 2-ს;

75 % 29 // მოგვცემს 17.

1.2.4 შედარების (დამოკიდებულების) ოპერატორები

შედარების (ან დამოკიდებულების) ოპერატორები გამოიყენება ცვლადების, რიცხვების (კონსტანტების) ან გამოსახულებების შედარებისათვის.
x<y // x ნაკლებია y-ზე
x>y // x მეტია y-ზე
x<=y // x ნაკლებია ან ტოლია y-ის
x>=y //x მეტია ან ტოლია y-ის
x==y // x ტოლია y-ის
x!=y // x არ არის ტოლი y-ის

ამ ოპერატორების შესრულების შედეგი შეიძლება იყოს „ჭეშმარიტი“(1-ის ტოლი), თუ პირობა შესრულდა ან „ყალბი“ (0-ის ტოლი) , თუ პირობა არ შესრულდა. x და y ცვლადების (რეგისტრების) მნიშვნელობები არ იცვლებიან.

1.2.5. ლოგიკური ოპერაციების ოპერატორები

|| || // „ან“ - ლოგიკური შეკრება (მხოლოდ „ყალბი“ და „ყალბი“ ცვლადები გვაძლევენ „ყალბ“ შედეგს, სხვა მნიშვნელობების დროს შედეგი იქნება „ჭეშმარიტი“).
მაგალითად, ორობით რიცხვზე „ან“ ოპერაციის შესრულების შედეგი იქნება:

```
 01011100
|| 11000111
 --------
 11011111
```

&& // „ და“ - ლოგიკური გამრავლება (ცვლადების მხოლოდ „ჭეშმარიტი და „ჭეშმარიტი მნიშვნელობა გვაძლევს „ჭეშმარიტ“ შედეგს. სხვა შემთხვევაში ოპერაციის შედეგი „ყალბია“).

მაგალითად, ზევით განხილულ ორობით რიცხვებზე „და“ ოპერაციის შესრულების შედეგი იქნება:

```
  01011100
&& 11000111
  01000100
```

~ //“არა“- ლოგიკური უარყოფა. შედეგად მივიღებთ ორობითი რიცხვი, რომლის თანრი- გები ინვერტირებულია.

მაგალითად, ~1001010 = 0110101

ლოგიკური ოპერაციის შედეგად მიიღება „ჭეშმარიტი“ ან „ყალბი“ ლოგიკური მნიშვნელობა.

ლოგიკური ოპერაციები შესაძლებელია გავაერთიანოთ რამოდენიმე შესამოწმებელი პირობით.

მაგალითად,

If ((გამოსახულება 1) && ((გამოსახულება 2 || || (გამოსახულება 3)))
{ /* პრგრამის კოდი აქ შესრულდება, თუ გამოსახულება 1 „ჭეშმარიტია“ (ე.ი. ნული არ არის და ასევა გამოსახულება 2 და 3 გამოსახულებიდან ერთერთი მაინც „ჭეშმარიტია“.

}

ძვრის ოპერაციების საშუალებით შეგვიძლია რაიმე ორობით კოდში წარმოდგენილი ცვლადი დავძრათ მარჯვნივ ან მარცხნივ ერთი ან რამოდენიმე თანრიგით. Mმარჯვნივ ძვრის ოპერტორი - rub>>(გამოსახულება), მარცხნივ ძვრის ოპერატორი - rub<<(გამოსახულება). ორივე ოპერატორში ძვრის ნიშნის მარცხნივ იწერება ცვლადი (ზევით ნაჩვენებ მაგალითებში rub), რომელიც იძვრის. ძვრის ნიშნაკის მარჯვნივ იწერება გამოსახულება, რომელიც განსაზღვრავს რამდენი თანრიგით უნდა დაიძრას ცვლადი. იგი შეიძლება იყოს ციფრი ან რთული გამოსახულება, რომლის შედეგი არის მთელი რიცხვი.

time=24 (00011000) // time განსაზღვრულია როგორც ბაიტიანი, char ტიპის ცვლადი.

time>>1 /* time ცვლადი იძვრის მარჯვნივ 1 თანრიგით*/

შედეგი გვექნება - 00001100.

clock<< (a+b) /* clock ცვლადი იძვრის მარცხნივ იმდენი თანრიგით, რა მნიშვნელობასაც მიიღებს ფრჩხილებში მოთავსებული გამოსახურება */

1.3 C ენაში გამოყენებული კონსტრუქციები

ნებისმიერი პროგრამა შესაძლებელია აგებული იყოს სამი კონსტრუქციის გამოყენებით: ოპერატორების მიმდევრობა, ამორჩევა და იტერაცია. ამ პარაგრაფში განხილული იქნება C ენაში გამოყენებული ოპერატორების ნაკრები და მათი გამოყენების ტიპიური მაგალითები ზემოთ ხსენებული კონსტრუქციების შექმნისათვის.

1.3.1 გადაწყვეტილების მიღების ოპერატორები

if(){}-else; იდეალური კონსტრუქცია იმ შემთხვევაში, როდესაც საჭიროა შესრულდეს პრო- გრამის რაღაც ნაწილი, რაიმე პირობის არსებობის ან არ შესრულდეს ამ პირობის არსე- ბობის დროს:

if (გამოსახულება) **{** კოდი C ენაზე /* ეს კოდი შესრულდება იმ შემთხვევაში, , თუ გამოსახულების მნიშვნელობა „ჭეშმარიტია“ ე.ი. მისი გამოთვლის შედეგი ლოგიკური ერთია*/

}

else { კოდი C ენაზე / * ეს კოდი შესრულდება იმ შემთხვევაში, თუ გამოსახუ- ლების გამოთვლის შედეგი „ყალბია“ (ლოგიკური ნულის ტოლია) */

};

else კონსტრუქციის აუცილებელი ელემენტი არ არის, მის გარეშე კონსტრუქცია შემდეგი სახით გამოიყურება:

if (გამოსახულება) { კოდი C ენაზე /* ეს კოდი შესრულდება იმ შემთხვევაში, , თუ გამოსახულების მნიშვნელობა „ჭეშმარიტია“ ე.ი. მისი გამოთვლის შედეგი ლოგიკური ერთია*/

};

მაგალითი,

If(PINB,5){ კოდი C ენაზე };

/* Tu მიკროკონტროლერის PB5 გამოსასვლელზე არის ლოგიკური 1 (ე.ი.“ჭეშმარიტია“), პროგრამაში შესრულდება ფიგურულ ფრჩხილებში მოთავსებული კოდი, ხოლო თუ გამოსასვლელზე არის 0, იგი გადავა პროგრამის მომდევნო ოპერატორის შესრულებაზე*/

2. **switch(){}**; - მრავლობითი ამორჩევის ოპერატორი, რომელიც იძლევა საშუალებას პროგრამაში განხორციელდეს გადასვლის ამორჩევა რამოდენიმე ვარიანტიდან.

გამოსახულება, რომელიც switch საკვანძო სიტყვის შემდეგ არის მრგვალ ფრჩხიკებში მოთავსებული შეიძლება იყოს ნებისმიერი გამოსახულება, დასაშვები C ენისათვის და რომლის მნიშვნელობა უნდა იყოს მთელი. ამ გამოსახულების მნიშვნელობა წარმოადგენს გასაღებს პროგრამაში გადასვლის რამოდენიმე ვარიანტიდან ერთ-ერთის ამორჩევისათვის. switch ოპერატორის ტანი შესდგება რამოდენიმე ოპერატორისაგან, აღნიშნული case საკვანძო სიტყვით და მომდევნო კონსტანტური გამოსახულებით. ყველა კონსტანტური გამოსახულება switch ოპერატორში უნდა იყოს უნიკალური გარდა იმ ოპერატორებისა, რომლებიც აღნიშნულია case და default საკვანძო სიტყვით. ოპერატორების სია შეიძლება იყოს ცარიელი, ან შეიცავდეს ერთ ან მეტ ოპერატორს.

switch ოპერატორის შესრულების მიმდევრობა შემდეგია:

-გამოითვლება მრგვალ ფრჩხილებში მოთავსებული გამოსახულების მნიშვნელობა;

- სრულდება გამოთვლილი მნიშვნელობის თანმიმდევრული შედარება კონსტანტურ გამოსახულებებთან, რომლებიც case სიტყვების შემდეგაა მოთავსებული.

- თუ ერთ-ერთი კონსტანტური გამოსახულება ემთხვევა ფრჩხილებში მყოფ გამოსახულების გამოთვლილ მნიშვნელობას, მაშინ მართვა გადაეცემა იმ ოპერატორს, რომელიც აღნიშნულია შესაბამისი case საკვანძო სიტყვით.

- თუ კონსტანტურ გამოსახულებიდან არც ერთი არ დაემთხვა ფრჩხილებში მოთავსებული გამოსახულების მნიშვნელობას, მაშინ მართვა გადაეცემა ოპერატორს, რომელიც აღნიშნულია default საკვანძო სიტყვით. ხოლო მისი არ ყოფნის შემთხვევაში მართვა გადაეცემა მომდევნო ოპერატორს.

case -დან გამოსვლისათვის , ხშირ შემთხვევაში, გამოიყენება break ოპერატორი.

მაგალითად:

switch (გამოსახულება) {

case 5: <კოდი C ენაზე>

/* ეს კოდი შესრულდება, თუ გამოსახულების გამოთვლის შედეგი ტოლია 5-ის. ამით switch ოპერატორის მუშაობა დასრულდება. */

break;

case-32; <კოდი C-ზე>

/* ეს კოდი შესრულდება იმ შემთხვევაში, თუ გამოსახულების გამოთვლის შედეგი ტოლია -32-ის. ამით switch ოპერატორის მუშობა დასრულდება */

break;

case 'G': <კოდი C-ზე >

/* ეს კოდი შესრულდება იმ შემთხვევაში, თუ გამოსახულების გამოთვლის შედეგის მნიშვნელობა G სიმბოლოს რიცხვით მნიშვნელობას ASCII კოდში */

break;

default: <კოდი C-ზე>

/* ეს კოდი შესრულდება , თუ გამოსახულების შედეგის მნიშვნელობა არ არის 5,-32 ან 'G, ასევე ისეთი კოდის შესრულების შემდეგ, რომელსაც დაბოლოვებისას არ აქვს break. ამით switch ოპერატორის მუშაობა დასრულდება */

};

case შეიძლება იყოს იმდენი, რამდენიც მოითხოვება . პროგრამაში გადასვლის ყველაზე სავარაუდო ვარიანტები უნდ განთავსდეს კონსტრუქციის თავში.

Default აუცილებელი არ არის. იგი შეიძლება მოვათავსოთ სხვა ადგილზეც(არა მხოლოდ კონსტრუქციის დასასრულს).

break საშუალებით ხდება კონსტრუსქციიდან გამოსვლა ამორჩეული ოპერატორის პროგრამული კოდის შესრულების შემდეგ. თუ მას არ ვიყენებთ, ამორჩეული კოდის შესრულების შემდეგ პროგრამა გადადის შემდეგ case პირობის შესრულებაზე.

3.goto - უპირობო გადასვლის ოპერატორი.

goto ოპერატორი გადასცემს მართვას იმ ოპერატორს, რომელსაც მინიჭებული აქვს ჭდე. მონიშნული ოპერატორი უნდა იყოს იმავე ფუნქციაში, რომელშიც goto ოპერატორია. ჭდე უნდა იყოს უნიკალური ე.ი. ერთი ჭდის სახელი არ შეიძლება იყოს პროგრამის სხვადასხვა ოპერატორებში. ჭდის სახელი პროგრამის რომელიღაც კოდის იდენტიფიკატორია.

მაგალითად:

mesto_5: /* ამ ნაჭდევზე მოვხდებით პროგრამის **goto mesto_5** ოპერატორის შესრულების შედეგად */

<C პროგრამის კოდი>

mesto_1; /* აქ მოვხვდებით პროგრამის **goto mesto_1** ოპერატორის შესრულების შემდეგ */

<C პროგრამის კოდი>

goto mesto_1; /* პროგრამა გადადის ოპერატორზე ჭდით mesto_1: */;

1.3.2 პროგრამის ციკლების ორგანიზაციის ოპერატორები

1. while(){}; პირობითი ციკლია (ციკლი პირობით)- გამოიყენება იმ შემთხვევაში, თუ საჭიროა შესრულდეს პროგრამის რომელიღაც კოდი, ეს კოდი სრულდება მანამ, სანამ სრულდება ფრჩხილებში მოცემული გამოსახულების გამოთვლის შედეგის რაღაც პირობა (ჭეშმარიტობა).

while(გამოსახულება){ კოდი C-ზე /* ეს კოდი უნდა შესრულდეს იმ შემთხვევაში , თუ ფრჩხილებში მოცემული გამოსახულების გამოთვლის შედეგი „ჭეშმარიტია“. კოდის შესრულების შემდეგ სრულდება გადასვლა ისევ **while** ოპერატორზე და გამოსახულება კვლავ მოწმდება ჭეშმარიტობაზე. ეს მეორდება, სანამ გამოსახულების მნიშვნელობა არ გახდება „ყალბი“. */

};

whaile ოპერატორის გამოყენების საილუსტრციოდ მოვიყვანოთ შემდეგი პროგრამული ფრაგმენტი.

```
1. k= _10
2. while ( k < 40 )

3. {
4. Temperature = k*9/5+32;
5.. k + +
6. printf(«Current temperature is \%f\n», Temperature);

7. }
```

KKK ცვლადი გამოიყენება როგორც ციკლის მთვლელი, რომელიც უნდა იყოს ინიცალიზებული **while** ოპერატორის შესრულებამდე **(** სტრიქონი 1). სტრიქონ 2-ში **while** ოპერატორის ფრჩხილებში ჩაწერილია პირობა ,რომლის ჭეშმარიტობის შემთხვევაში ციკლი მეორდება. ციკლის შესრულება იწყება აღნიშნული პირობის შემოწმებით. თუ პირობა სრულდება ანუ გამოსახულების მნიშვნელობა "ჭეშმარიტია" (ლოგიკური ერთია) , სრულდება ოპერატორის ტანში მოთავსებული კოდი (4-6 სტრიქონი).

ციკლის ყოველ გასვლაზე k (სტრიქონი 5) ცვლადის მნიშვნელობა იზრდება ერთით. ციკლის ყოველი ბიჯის შესრულების შემდეგ პირობის მნიშვნელობა კვლავ მოწმდება და ა.შ. ციკლის შესრულება ხდება იქამდი, სანამ პირობა ჭეშმარიტია. პირობის არშესრულების შემთხვევაში (შედეგი „ყალბია“) ციკლი მთავრდება და პროგრამა გადადის შემდეგი ოპერატორის შესრულებაზე.

ციკლის ტანი შესძლებელია არც არსებობდეს. while ოპერატორის ამგვარი ინტერპრეტაცია შეიძლება გამოვიყენოთ მაგალითად, მიკროკონტროლერის აპარატურული ალმის პროგრამული მოლოდინის შემთხვევაში, რომელიც იცვლება ჩაშენებული პერიფერიული მოწყობილობის მიერ.

While ოპერატორს გააჩნია რამოდენიმე ვარიანტი:

do-while კონსტრუქციას აქვს შემდეგი სახე:

do { კოდი C-ზე}

while (გამოსახულება);

/* ეს კოდი შესრულდება ერთხელ, შემდეგ მოწმდება while ოპერატორში გამოსახულების ჭეშმარიტობა, თუ გამოსახულება „ჭეშმარიტია“ კოდის შესრულება მეორდება მანამ, სანამ გამოსახულების მნიშვნელობა არ გახდება ყალბი (ნულის ტოლი). ამ შემთხვევაში სრულდება გადასვლა პროგრამის მომდევნო ოპერატორზე. .

while ოპერატორი ხშირად იხმარება უსასრულო ციკლის შესაქმნელად.

while (1);
/* პროგრამა ამ ციკლს შეასრულებს დაუსრულებლად, სანამ არის კვება, არ არის განულება და არ არის წყვეტა. როდესაც წარმოიშვება წყვეტის მოთხოვნა პროგრამა გადადის წყვეტის დამუშავებაზე და მისი დასრულების შემდეგ კვლავ უბრუნდება ციკლს. (თუ წყვეტის დამუშავებისას პროგრამის სხვა ადგილზე გადასვლას ადგილი არა აქვს)*/

while(1) {

<პროგრამის კოდი C-ზე>
};

2. **for(;;){};** ოპერატორი გათვალისწინებულია მთვლელიანი ციკლის რეალიზაციისათვის. ოპერატორში შესაძლებელია ერთდროულად სამი ოპერაციის შესრულება: ციკლის მთვლელის ინიციალიზაცია, მისი მნიშვნელობის შემოწმება და მოდიფიკაცია.
განვიხილოთ ციკლის შესრულების მაგალითი for ოპერატორის გამოყენებით.

for (i=5; i<20; i+=4) {
< კოდი C-ზე >
}

/ * ციკლის დასაწყისში მოწმდება i<20 საკონტროლო გამოსახულების ჭეშმარიტობა (სრულდება, თუ არა პირობა). რამდენადაც მოყვანილ მაგალითში i ცვლადს მინიჭებული აქვს საწყისი მნიშვნელობა 5, ეს ნიშნავს იმას, რომ საკონტროლო გამოსახულება „ჭეშმარიტია“ და **for** ციკლის კოდი პირველად i=5-თვის შესრულდება. შემდეგ i+=4 გამოსახულების შედეგად i გახდება 9. კვლავ მოხდება საკონტროლო გამოსახულების ჭეშმარიტობის შემოწმება და რამდენადაც 9<20 **for** ციკლის კოდი კვლავ შესრულდება i=9-თვის. ასე გაგრძელდება მანამ, სანამ საკონტროლო გამოსახულებას ექნება „ჭეშმარიტი“ მნიშვნელობა. როდესაც სკონტროლო გამოსახულება გახდება „ყალბი“, პროგრამა გამოვა **for** ციკლიდან. ცხადია, i ცვლადი წინასწარ უნდა იყოს გამოცხადებული. */

5 - i ცვლადის საწყისი მნიშვნელობაა (რიცხვი 5 მაგალითისთვის არის მოყვანილი), მისი მნიშვნელობა უნდა შეესაბამებოდეს i ცვლადის ტიპს. ჩვენს შემთხვევაში ეს არის char, რომელიც უმრავლეს ტრანსლატორებში უთქმელობით განისაზღვრება როგორც უნიშნო ორობითი რიცხვი.
i<20 გამოსახულებაა. იგი შეიძლება წარმოდგენილი იყოს სხვადასხვა სახით. მთავარია ციკლის შესრულების დროს რაღაც მომენტში გახდეს „ყალბი“, წინააღმდეგ შემთხვევაში ციკლი არასდროს არ დსრულდება.

i+= 4 მთვლელია ანუ ციკლის ცვლადის ცვლილების განმსაზღვრელი. ჩვეულებრივად იხმარება i^{++} ანუ ცვლადზე ერთის დამატება ციკლის ყოველ ბიჯზე. ასევე შესაძლებელია სხვა კონსტანტის დამატება, როგორც ეს ზევით განხილულ მაგალითში არის ნაჩვენები.

საწყისი გამოსახულება შეიძლება იყოს ნებისმიერი C ენაში დასაშვები გამოსახულება, რომლის შესრულების შედეგია მთელი რიცხვი.

საკონტროლო გამოსახულება განსაზღვრავს პირობას, რომლის განმავლობაში სრულდება ციკილი..

მთვლელი ასრულებს ცვლადისათვის რაიმე კონსტანტის დამატებას ციკლის ყოველი ახალი ბიჯის დაწყების წინ.

გამოსახულება შეიძლება იყოს არა მარტო ცვლადი, არამედ სხვა ცვლადების ფუნქცია.

მაგალითად:

i=(7+i*4).

1.4 ფუნქცია

ფუნქცია არის საწყისი პროგრამის დამოუკიდებელი ფრაგმენტი, გათვალისწინებული რაიმე ამოცანის გადაწყვეტისათვის.

მიკროპროცესორული სისტემის პროგრამის დაგეგმარების დროს მიზანშეწონილია მთლიანი დიდი ამოცანა დაიყოს რამოდენიმე მცირე, ფუნქციონალურად დასრულებულ ფრაგმენტად, რომელიც შესაძლებელია დამუშავებული იყოს სხვადასხვა შემსრულებლის მიერ. ეს ფრაგმენტები გაფორმდებიან როგორც ფუნქციები, რომლებისგან შემდგომში შესდგება მთლიანი პროგრამა. ფუნქცია შესაძლებელი იყოს პროგრამის სხვადასხვა ადგილას ან სხვა პროექტში..

Nნებისმიერი ფუნქცია პროგრამის დასაწყისში უნდა იყოს გამოცხადებული. ფუნქციის გამოცხადების შემთხვევაში მას უნდა ქონდეს სათაური, რომელსაც აქვს შემდეგი ფორმატი:

< დასაბრუნებელი ცვლადის ტიპი><ფუნქციის სახელი>
(< ცვლადის ტიპი 1><ცვლადის სახელი 1>,
< ცვლადის ტიპი 2><ცვლადის სახელი 2>,
. . . . . .
< ცვლადის ტიპი N><ცვლადის სახელი >);
მრგვალ ფრჩხილებში ცვლადის სახელის ჩაწერა აუცილებელი არ არის.

ქვევით მოყვანილია ფუნქციის სათაურის მაგალითები:

მაგალითი 1: int compute(int,int);
მაგალითი 2: float cheinge(char name, float namber, int a);

მაგალით 1-ში ფუნქცია compute იყენებს ორ არგუმენტს. ფუნქციის არგუმენტების სპეციფიკაცია მოცემულია მრგვალ ფრჩხილებში. მაგალითში compute იყენებს ორ მთელ 16 ბიტიან არგუმენტს ე.ი. ფუნქციის გამოძახების შემთხვევაში მას უნდა მიეთითოს ორი int ტიპის ცვლადი. compute ფუნქციის შესრულების შედეგი იქნება ცვლადის

მნიშვნელობა, რომლის ტიპი მოცემულია ფუნქციის სახელის წინ (მაგალითში იგი ასევე არის 16 თანრიგა). მაგალით 2-ში ნაჩვენებია change ფუნქციის სათაური. ეს ფუნქცია იყენებს სამ არგუმენტს: ერთბაიტიანი მთელი ცვლადი -name, მცოც მძიმით წარმოდგენილი ცვლადი - number და ორბაიტიანი მთელი ცვლადი - a. change ფუნქცია აბრუნებს მცოცი მძიმით წარმოდგენილ ცვლადის მნიშვნელობას რომლის ტიპი მიეწერება წინ ფუნქციის სათაურს.

. ყოველი ფუნქცია უნდა იყოს გამოცხადებული შესაბამისი პროგრამული მოდულის წინ, რომელიც შესასრულებელი პროგრამის გენერაციის პროცესში დაემატება მოდულს.

მაგალითად, დაუშვათ, რომ ზევით ნაჩვენები ფუნქცია compute ითვლის a და b ორი ორტოგონალური შემადგენლის ვექტორის მოდულს და აბრუნებს rezult ცვლადით.

```
1. int compute(int a, int b)
2. {
3. int sum, result;
4. sum = a*a + b*b;
5. result=(int) (sqrt(sum));
6. return(result);
7. }
```

მოყვანილ მაგალითში სტრიქონი 1 შეიცავს ფუნქციის სათაურს, რომლის სტრუქტურა ზე- ვით იყო აღწერილი. სტრიქონ 2-ში ფიგურული ფრჩხილი აღნიშნავს ფუნქციის ტანის ასაწყისს. სტრიქონ 3-ში ცხადდებიან ლოკალური ცვლადები sum და result, რომლებიც გამოიყენებიან ფუნქციაში. 4,5 სტრიქონებში მოთავსებული ოპერატორები ასრულებენ ფუნქციის აღწერაში გაცხადებულ გამოთვლებს. სტრიქონ 5-ში მივაქციოთ ყურადღება int საკვანძო სიტყვას, რომელიც იმყოფება sqrt ფუნქციის წინ. იგი ასრულებს შედეგის გარდასახვას. Compute ფუნქციის სათაურში გამოცხადებულ მონაცემთა ტიპში, ვინაიდან კვადრატული ფესვის გამოთვლას ფუნქცია აბრუნებს მონაცემებს სხვა ფორმატში. სტრიქონ 6-ში გამოყენებულია დაბრუნების ოპერატორი return, რომელიც შესაძლებლობას იძლევა გამოვიყენოთ result ცვლადის მნიშვნელობა ძირითადი პროგრამის სხვა ოპერატორებისთვისაც. ფიგურული ფრჩხი- ლი სტრიქონ 7-ში ამთავრებს ფუნქციის აღწერას. მოყვანილი მაგალითის საფუძველ- ზე შეგვიძლია დავასკვნათ, რომ ყოველი ფუნქცია უნდა განსაზღვრული იყოს გარკვეული წესით დასაწყისში იწერება ფუნქციის სათაური, რომელშიც მიეთითება ფუნქციის სახელი, გამოყენებული და დაბრუნებული ცვლადების ტიპი (ფუნქციის განსაზღვრის დროს სათაურის ბოლოს წერტილი და მძიმე არ იწერებ). შემდეგ სტრიქონებში ჩამოწერილია ფუნქციის ოპერატორები, მოთავსებული ფიგურილ ფრჩხილებში, რომლებიც ქმნიან ფუნქციის ტანს. თუ დასაბრუნებელ ცვლადის ველში მითითებულია მისი ტიპი, მაშინ ფუნქციის ბოლო ოპერატორი უნდა იყოს return, ხოლო თუ დასაბრუნებელი ცვლადის ველში ჩაწერილია სამომხმარებლო სიტყვა void, მაშინ ფუნქცია მონაცემებს არ აბრუნებს. ასეთი ფუნქციის დანიშნულებაა გარკვეული მოქმედებების შესრულება მკ-ს პერიფერიული მოწყობილობის ან გარე მოწყობილობის მართვისათვის. ამ შემთხვევაში ოპერატორი return არ იწერება.

თუ ფუნქცია აღწერილია, იგი შეიძლება გამოძახებული იყოს პროგრამის ნებისმიერი ადგილიდან. ფუნქციის გამოძახებისათვის უნდა ჩავწეროთ მისი სახელი და მრგვალ ფრჩხილებში მიუთითოთ პარამეტრების მნიშვნელობა, თუ ასეთები განსაზღვრეული იყვნენ სათაურში.

ქვევით ნაჩვენებია განხილული ორი ფუნქციის გამოძახების მაგალითები:

compute (23,12);

change (‘b’,7.82,2);

მოყვანილ მაგალითებში დასაბრუნებელი ცვლადის ტიპის სპეციფიკაცია ფუნქციის გამოძახების ჩანაწერში არ არის მითითებული, ხოლო არგუმენტების ველში (მრგვალ ფრჩხილებში) მოცემულია მათი რიცხვითი მნიშვნელობები.

1.5 ძირითადი ფუნქცია

ძირითადი ფუნქცია main არის ფუნქციის განსაკუთრებული ტიპი. მისი განსხვავება სხვა ფუნქციებისაგან მდგომარეობს მასში , რომ იგი სრულდება ერთდროულად, რაიმე პროგრამის გაშვებასთან ერთად. main ძირითადი ფუნქციის ტექსტი ასახავს მთელი სამომსახურებო პროგრამის სტრუქტურას. ძირითადი ფუნქცია შეგვიძლია წარმოვიდგინოთ როგორც მმართველი, რომელიც აკონტროლებს მართვის ცალკეული „ბრძანების“ შესრულებას შესაბამისი ფუნქცია-პროგრამის გაშვებით. მაგალითად, შევასრულოთ n ამოცანა, რომლებიც გაფორმებული არიან ფუნქციის სხით. ამისათვის გამოვიყენებთ ძირითად პროგრამას, რომელშიც მიმდევრობით გამოიძხემა ეს ფუნქციები.

```
1 void main(void)
2 {
3 function_one();
4 function_two();
5 function_three();
6 :
7 :
8 :
9 function_n();
10 }
```

სტრიქონი1 შეიცავს main ფუნქციის იდენტიფიკატორს. სამომსახურებო სიტყვა void მრგვალ ფრჩხილებში გვატყობინებს, რომ ეს ფუნქცია არ მოითხოვს შემავალ არგუმენტებს. 3-9 სტრიქონები შეიცავენ ფუნქციის გამოძახების ოპერატორებს, ამასთან თვითეული ფუნქცია გამოიძახება ერთხელ. function_one() შესრულების შემდეგ თანმიმდევრულად გამოიძახება function_two(), function_three() და ასე ბოლომდის.

1.6 სათაურის ფაილები

სათაურის ფაილი (header file) არის გარე ფაილი, რომელიც იწერება ტრანსლირებული პროგრამის დასაწყისში #Include დირექტივის საშუალებით. იგი ჩვეულებრივად შეიცავს პროგრამაში გამოყენებული ტიპების და ცვლადების განსაზღვრას. C ენა სთავაზობს დამპროგრამირებელს სტანდარტული ფუნქციების გარკვეულ ანაკრეფს, რომლებიც მოთავსებული არიან რამოდენიმე ფაილში. მაგალითად, ჩვენს მიერ ზევით განხილულ ფუნქციაში compute გამოყენებულია კვადრატული ფესვის გამოთვლის ფუნქცია sqrt, რომელიც განსაზღვრულია მათემატიკური ფუნქციების ფაილში math.h.

სათაურის ფაილები შეიცავენ ფაილების განსაზღვრებებს, მაკროსებს, ფუნქციების გამო- ცხადებას, რაც შესაძლებლობას აძლევს დამპროგრამებელს გამოიძახოს ეს ფუნქციები, გამოი- ყენოს ცვლადები და მაკროსები შესაქმნელი პროგრამის ტექსტში მათი დამატებითი გამოცხა- დების გარეშე. ტრანსლაციის პროცესში ცვლადების მნიშვნელობები აღნიშნული ფაილებიდან ჩაენაცვლებიან ძირითად პროგრამაში მითითებულ მათ სიმბოლურ მნიშვნელობას.

პროგრამული უზრუნველყოფის დამმუშავებელი ფირმების მიერ მოწოდებული კომპილატორები უკვე შეიცავენ ბიბლიოთეკებს და მათ შესაბამის სათაურის ფაილებს. სათაურის ფაი;ლის ჩასმისათვის დასამუშავებელ პროგრამულ მოდულში უნდა ვისარგებლოთ #include დირექტივით.

მოვიყვანოთ რამოდენიმე მაგალითი:

#include <studio> ;

#include <mathj.h> ;

#include «myhader.h» ;

Pპირველ ორ ჩანაწერში ჩასასმელი ფაილის სახელი მოთავსებულია „<“ „>“ სიმბოლოებს შორის, რაც მიუთითებს კომპილატორს, რომ დასახელებული ფაილები იმყოფებიან გარკვეულ დირექტორიაში (საქაღალდეში). ჩვეულებრივად ეს არის საქაღალდე **include** სახელით, რომელიც განთავსებულია C კომპილიატორის ძირითად კატალოგში. Mმესამე ჩანაწერში ჩასასმელი ფაილის სახელი მოთავსებულია ბრჭყალებში. კომპილატორისთვის ეს ნიშნავს, რომ აღნიშნული ფაილი მოთავსებულია იმავე საქაღალდეში, რომელშიც არის შექმნილი პროგრამის მოდული.

1.7 C ენის პროგრამის სტრუქტურა

პროგრამა C ენაზე ტექსტური ფაილია, ჩვეულებრივად c გაფართოებით. მთელი შესასრულებელი C პროგრამის საწყისი კოდი წარმოადგენს ფუნქციას და მოთავსებულია ფიგურულ ფრჩხილებში „{ }„.

C პროგრამას აქვს განსაზღვრული სტრუქტურა:

1) სათაური;
2) საჭირო აუცილებელი გარე ფაილების ჩართვა - **#include**;
3) თქვენი განსაზღვრებები მოხერხებული მუშაობისთვის - **define;**
4) გლობალური ცვლადების და კონსტანტების გამოცხადება;
5) წყვეტის დამამუშავებელი ფუნქციების აღწერა;
6) im sxva funqciebis aRwera romelic gamoyenebulia programaSi;
7) **main** მთავარი ფუნქცია **(ეს ერთადერთი აუცილებელი ფუნქციაა).**

ეს თანმიმდევრობა საორიენტაციოა, იგი არ არის აუცილებელი.

ფუნქციას აქვს „ტანი“, ფიგურულ ფრჩხილებში მოთავსებული. C-ში ფუნქციის ტანი შეიცავს ოპერატორწბის მიმდევრობას, რომელიც განისაზღვრება ფუნქციის დანიშნულებით.

C-ში პროგრამა მუშაუბას იწტებს **main()** ფუნქციიდან. საჭიროების შემთხვევაში **main()** ფუნქციიდან შეიძლება განხორციელდეს პროგრამის სხვა ფუნქციების გამოძახება, საიდანაც შესძლებელია თანმიმდევრობით ასევე გამოძახებული იყოს სხვა ფუნქციები. ფუნქციის შესრულების შემდეგ პროგრამა იმავე გზით ბრუნდება საწყის პროგრამაში, რა გზითაც ხდებოდა ფუნქციების გამოძახება:

main()

... ძირითადი პროგრამის რაიმე კოდი . . . .

ფუნქცია 1-ის გამოძახება; //პროგრამა გადადის ფუნქცია1-ის შესრულებაზე

ძირითადი programis striqoni; // შესრულდება მოგვიანებით

// ფუნქცია 1-დან დაბრუნების შემდეგ

.... ძირითადი პროგრამის რომელიღაც კოდი ....

}

C ენაზე დაწერილი პროგრამის მაგალითი

მაგალითისათვის განვიხილოთ პროგრამა, რომლის საფუძველზე მიკროკონტროლერი ასრულებს გარედან მიწოდებული ძაბვის გადასახვას ორობით კოდში და მის გამოტანას ინდიკაციაზე. ძაბვა მიეწოდება PC პორტის 0 შესასვლელზე, რომელიც დაკავშირებულია მიკროკონტროლერში ჩაშენებული ანალოგურ ციფრული გარდამსახის (აცგ) შესასვლელთან. Aაცგ-ს მიერ გადასახული ძაბვის მნიშვნელობა ჩაიწერება PB პორტში, რომლის თვითოეული გამოსასვლელი (თანრიგი) დაკავშირებულია ერთ-ერთ შუქდიოდთან. შუქდიოდი ანთებულია, თუ პორტის შესაბამის თანრიგში არის 1 და ჩამქრალია თანრიგში 0-ის არსებობის შემთხვევაში.

ქვემოთ განხილული ამ პროგრამის სტრუქტურა.

პუნქტი 1 პროგრამის სათაური

/* მისი გაფორმება ხდება, როგორც კომენტარის და შეიძლება შეიცავდეს შემდეგ ინფორმაციას:

_ დასახელება, დანიშნულება, ვერსია და ინფორმაცია პროგრამის ავტორის შესახებ;

- პროგრამის ალგორითმის მოკლე აღწერა;

- მკ-ს გამომყვანების დანიშნულების და მუშაობის რეჟიმის განმარტება;

- გამოყენებული კომპილიატორი, მათი ვერსიები;

- სხვა ცნობები, რომელთა მითითებასაც ავტორი თვლის სასარგებლოთ.

*/

პუნქტი 2 გარე ფაილების ჩართვა

#include <mega 16.h>

/* კომპილაციის დაწყებამდე, კომპილქატორის წინაპროცესორი ამ სტრიქონის ნაცვლად სვამს „ხიდერ“ სათაურის ფაილის შემცველობას (ტექსტს) - ეს ფაილი შეიცავს მკ-ში

არსებული რეგისტრების ჩამონათვალს და მათი დასახელების შესაბამისობას მკ-ში მათ ფიზიკურ მისამართებთან. აუცილებელია მიეთითოს რომელ მკ-ს იყენებთ პროექტში (მოცემულ მაგალითში მითითებულია Atmega 16). მათი მოძიება შესაძლებელია Internet-ში. ისინი უნდა ჩაიწეროს სპეციალურ პაკეტში */

#include <delay.h>

/* კომპილაციის წინ, კომპილატორის წინაპროცესორი ამ სტრიქონის ნაცვლად პროგრამაში ჩასვამს delay.h „ხიდერ“-ის ტექსტს - ამ ფაილში არიან ფუნქციები პროგრამაში პაუზის (დაყოვნების) შესაქმნელად */

Delay_us(N);

/* შესრულდეს დაყოვნება N მიკროწამი. N უნდა იყოს კონსტანტა ან გამოსახულება რომლის შედეგის კონსტანტა. .

მაგალითად:

delay_us(12+7*3); /* შესრულდეს დაყოვნება ფრჩხილებში მოთავსებული გამოსახულების შედეგის ხანგრძლიობით მიკროწამებში*/

delay_us(117); // შესრულდეს დაყოვნება 117 მილი წამის ხანგრძლიობით

delay_ms(x); /* შესრულდეს პაუზა x მილიწამის ხანგრძლიობით*/

უნქტი 3 მომხმარებლის განსაზღვრებები

efine ADC_BUSY PINB.0
efine NCONVST PORTB.1

/*ამ ორი სტრიქონის მიხედვით კომპილაციის დაწყებამდე კომპილატორი შეცვლის ოგრამის სახელს ტექსტში PINB.0-ს ADC_BUSY - ით და PORTB.1-ს NCONVST-ით.

ჰრიგად, ზევით მოყვანილი დირექტივების საშუალებით შეგვიძლია ცვლადის დასახელება ცვალოთ ჩვენთვის უფრო გასაგებ დასახელებით. ზემოთ მოყვანილ მამაგალითებში PINB.0 ვირათ, რომელიც წარმოადგენს აცგ-ს დაკავებულობის მაჩვენებელ maგამომყვანს, ოგრამაში ნახმარი იქნება უფრო გასაგები დასახელება ADC_BUSY (ADC დდაკავებულია), ლო PORTB.1-ის მაგივრათ, რომელიც წარმოადგენს აცგ-ს გარდასახვის ნენების დართვის ასვლელს, ვიხმაროთ სახელი NCONVST (დაიწყოს აცგ-ს ახალი გაგადასახვა).

efine დირექტივის ხმარება არ არის აუცილებელი */

#define INIT_TIMER0 TCNT0=0x100L-F_XTAL/^$L/500L /* ეს მაგალითი გვიჩვენებს, რომ განსაზღვრება შეიძლება იყოს რთულიც*/

პუნქტი 4 ცვლადების გამოცხადება

C პროგრამაში ცვლადის გამოყენების წინ აუცილებელია მისი გამოცხადება ე.ი. უნდა მიეთითოს კომპილატორს როგორი ტიპის მონაცემების შენახვა შეუძლია მას და რა სახელწოდებით არის ის. ეს საკითხი წინა პარაგრაფებში დაწვრილებით არის განხილული.

ცემულ მაგალითში გლობალური ცვლადები არ გამოიყენებიან.

პუნქტი 5 წყვეტის დამუშავების ფუნქციის აღწერა

/* ამ პროგრამაში არის მხოლოდ ერთი წყვეტა და მაშასადამე წყვეტის დამუშავების ერთი ფუნქცია. პროგრამა განახორციელებს მასზე გადასვლას წყვეტის მოთხოვნის წარმოქმნის დროს, როდესაც აცგ დაასრულებს ანალოგური სიგნალის (ძაბვის) მნიშვნელობის გადასახვასოროზით კოდში. მიღებული კოდი იწერება აცგ-ს გამოსასვლელ რეგისტრში. წყვეტის დამმუშავებელი ფუნქციის დანიშნულებაა შეასრულოს მიღებული კოდის გადაწერა აცგ-ს გამოსასვლელ რეგისტრიდან PB პორტში შუქდიოდებისათვის მისი მიწოდების მიზნით */

ფუნქციის გამოცხადება ჩაიწერება შემდეგი სახით:

```
interrupt [ADC_INT]  void adc_isr(void)
{

PORTB = (unsigned char)( ~ADCW>>2);

/* შუქდიოდებზე აისახოს ანალოგურ - ციფრულ გადამსახის ,მუშაობის შედეგის
უფროსი 8  თანრიგი 127 მწმ დაყოვნებით*/

delay_ms (127);     // დაყოვნება  127  მილიწამი

 /*აცგ-ს ახალი გადასახვის დაწყება ხორციელდება ADCSRA რეგისტრის მე-6 ბიტში 1-ს
დაყენებით.*/

ADCSRA|= 0x40;     // ADCSRA  რეგისტრში იწერება კოდი 0100 0000
}                  // წყვეტის დამუშავების ფუნქციის დახურვის ფრჩხილი
```

განვიხილოთ წყვეტის ფუნქცია:
წყვეტის დამუშავების ფუნქციას შეგვიძლია დავარქვათ ნებისმიერი სახელი ისევე, როგორც სხვა ფუნქციებს, გარდა main-ისა.

ამ პროგრამაში იგი დასახელებულია როგორც adc_isr. P ფუნქციის წინ დაწერილი void (ცარიელი) ნიშნავს, რომ ფუნქცია არ გვიბრუნებს შედეგს, ანუ არ აფორმირებს რაიმე რიცხვით მნიშვნ3ელობას. ფრჩხილებში მოთავსებული void კი ნიშნავს, რომ ფუნქციას არ

აქვს პარამეტრი. რომელი წყვეტის დროს სრულდება მისი გამოძახება, კომპილატორი გაიგებს სტრიქონის მეორე ნაწილიდან.

Interrupt [ADC_INT};

პირველი დარეზერვირებული სიტყვიდან - interrupt იგი გაარკვევს, რომ ეს არის წყვეტის დამმუშავებელი ფუნქცია, ხოლო წყვეტის ვექტორის ნომერს კომპილატორი ჩაანაცვლებს ADC_INT იდენტიფიკატორის მაგიერ. წყვეტის ყოველ მოთხოვნას მინიჭებული აქვს ნომერი, რომლითაც ხდება მიმართვა მკ-ს პროგრამული მეხსიერების ერთ-ერთი უჯრედთან, სადაც იმყოფება წყვეტის შესაბამისი ვექტორი ანუ უპირობო გადასვლის ბრძანება ერთ-ერთი წყვეტის დამამუშავებელი ფუნქციის მისამართით, რომელიც უნდა შესრულდეს მოცემული წყვეტის დროს. აღნიშნული ნომრები მითითებულია სხვადასხვა წყვეტისათვის ჩვენს მიერ ადრე ჩასმული მკ Atmega 16-ის აღწერის სათაურის ფაილში („ხიდერში“).

ინფორმატიულია პროგრამის მომდევნო სტრიქონი:

PORTB=(unsigned char)(~ADCW>>2);

ეს ოპერატორი ასრულებს PORTB ცვლადისადმი ADCW ცვლადის მნიშვნელობის მინიჭებას. PORTB არის B პორტის გამოსასვლელი რეგისტრი, რომელთანაც მიერთებულია შუქდიოდები. ADCW აცგ-ს ორბაიტიანი გამოსასვლელი რეგისტრია, რომელშიც იწერება გარდასახვის 10-ბიტიანი შედეგი (0-9თანრიგები).

ვინაიდან PORTB 8 თანრიგაა, რომლის თვითოეულ გამოსასვლელზე მიერთებულია შუქდიოდი, ამიტომ PORTB-ში უნდა გადავწეროთ ADCW რეგისტრიდან შედეგის 8 უფროსი თანრიგი (ე.ი. 2-9 თანრიგი). ამისათვის ADCW რეგისტრის შემცველობას ვძრავთ ორი პოზიციით მარჯვნივ.

ADCW>>2 /* 0 და 1 ბიტები ძვრის შედეგად გამოდიან რეგისტრიდან მარჯვნივ და გადავარდებიან. ბიტი 9 გადაადგილდება თანრიგ 7-ში, ბიტი 8 თანრიგ 6-ში , ასევე სხვა თანრიგებიც*/

ახლა გადასახვის შედეგის 8 უფროსი თანრიგი მოთავსდება ADCW რეგისტრის უმცროსი ბაიტის 0-7 თანრიგებში.

>>n ნიშნავს ცვლადის დაძვრას მარჯვნივ n პოზიციით (თანრიგით).

<< n ნიშნავსცვლადის დაძვრას მარცხნივ n პოზიციით.

შუქდიოდის ანთებული მდგომარეობა უნდა ასახავდეს 1-ის არსებობას გარდასახვის შედეგად მიღებული კოდის შესაბამის თანრიგში. მეორეს მხრივ, ჩართვის სქემიდან გამომდინარე, შუქდიოდი ინთება პორტის შესაბამის გამოსასვლელზე 0-ის არსებობის შემთხვევაში. ამისათვის PORTB-ში უნდა ჩაიწეროს კოდი, რომელშიც 1 შეცვლილი იქნება 0-ით და პირიქით, 0 შეცვლილი იქნება 1-ით. ეს სრულდება თანრიგობრივი ინვერტირების ოპერაციით (~).

Aამრიგად, ~ (ADCW >>2 გამოსახულების შედეგი იქნება გარდასახული ორობითი კოდის ინვერტირებული 8 უფროსი ბიტი, რომელიც მოთავსებულია ორბაიტიან ADCW რეგისტრის უმცროს ბაიტში.

ვინაიდან PORTB ბაიტიანია, ხოლო ADCW- ორი ბაიტიანი, მინიჭების ოპერაციის შესრულებამდე საჭიროა ADCW ცვლადი (ე.ი. ორი ბაიტი) გარდაიქმნას უნიშნო ბაიტად.

ეს მოქმედება სრულება შემდეგი სახით:E

(unsigned char) (~ADCW>>2);

ამ სტრიქონის შედეგი ერთი ბაიტია და მას ვათავსებთ PORTB-ში.

პუნქტი 6. პროგრამაში გამოყენებული ფუნქციების აღწერა

// ისინი შეიძლება იყოს იმდენი, რამდენიც თქვენ გჭირდებათ

// განხილულ მაგალითში გვაქვს გარდ main ფუნქციისა, ორი - მკ-ს საწყისი კონფიგურირების //და წყვეტის დამმუშავებელი პროგრამა

void init_mk (void) {

/* ნებისმიერ ფუნქციის დასაწყისში ცხადდებიან ლოკალური ცვლადები საჭიროების მიხედვით. ჩვენ პროგრამაში ლოკალური ცვლადები არ გავაქვს */

// B პორტის ინიციალიზაცია

DDRB=0xFF; // B პორტის ყველა გამომყვანი გავაწყოთ როგორც გამოსასვლელი

PORTB =0xFF ;// B პორტის ყველა გამოსასვლელზე გამოვიყვანოთ 1, ვინაიდან //შუქდიოდები დასაწყისში ჩამქრალი უნდა იყოს

/* აცგ-ს დაყენება ხდება ADCSRA რეგისტრის თანრიგებში გარკვეული კოდის ჩაწერით. ADCSRA არის აცგ-ს მართვის რეგისტრი, რომლის თვითეულ თანრიგი შეესაბამება რაიმე დანიშნულების ალამს , რომელთა დაყენება 1-ში ან ნოლში განსაზღვრავს აცგ-ს პარამეტრებს:

- აცგ-ს მუშაობის ნებისდართვა (სრულდება ADCSR რეგისტრის მე-7ე თანრიგში ერთის ჩაწერით);

- საყრდენი სიხშირე რომელზედაც მუშაობს გამოყენებული მიკროკონტროლერი არის 3,69 მგჰც. აცგ-ს ტაქტირებისათვის დასაშვები 57,656 კგც სიხშირის მისაღებად იგი უნდა დავყოთ 64-ზე. ამისათვის ADCSR რეგისტრის უმცროს სამ ბიტში 0,1,2 უნდა ჩავწეროთ კოდი 110.

- ნება დავრთოთ წყვეტის სიგნალის დაფორმირებას გარდასახვის დამთავრების შემდეგ (ეს სრულდება ADCSRA რეგისტრის მე-3ე ბიტში ერთის ჩაწერით)

ამგვარათ, ამ მნიშვნელობების ერთობლიობა შეადგენს ორობით კოდს 1000 1110 ანუ თექსმეტობით სისტემაში 0x8E, რომელიც უნდა ჩაიწეროს ADCSRA რეგისტრში.

ADCSRA=0x8E;

/* ახლა ავირჩიოთ აცგ-ს შესასვ;ლელად C პორტის 0 შესასვლელი - PC0. აცგ-ს აღწერიდან ცნობილია, რომ მას შეიძლება მივაწოდოთ ანალოგური სიგნალები C პორტის 8 შესასვლელიდან. თვითოეული შესასვლელის მიერთება აცგ-სთან ხდება მულტიპლექსორზე შესაბამისი კოდის მიწოდებით, რომელიც წინასწარ უნდა ჩაიწეროს ADMUX რეგისტრის უმცროს ოთხ თანრიგში. ჩვენ შემთხვევაში PC0 შესასვლელი ამოირჩევა აღნიშნულ თანრიგებში 0000 კოდის ჩაწერით */

ADMUX=0;

/* ნება დავრთოთ გლობალურად ყველა წყვეტას.

#asm("sei");

} // ფუნქციის დამთავრების ფრჩხილი

Pპუნქტი 7 მთავარი ფუნქცია **main(). /*** C პროგრამის შესრულება იწყება ამ ფუნქციიდან */

```
void main (void) {
/* დასაწყისში უნდა გამოცხადდეს ლოკალური ცვლადები, თუ ამის საჭიროება არის */
init_mk;        // აცგ-ს ინიციალიზაცია;

// გაუშვებთ აცგ-ს

ADCSRA|=0x40;

// დაუსრულებელი ციკლი წყვეტის მოლოდინში

while (1);

}     // main ფუნქციის დამთავრება
```

პროგრამა სრულდება შემდეგი თანმიმდევრობით:

აცგ-ში გარდასახვის დამთავრების შემდეგ დაფორმირდება წყვეტის სიგნალი და პროგრამა გადადის წყვეტის დამმუშავებელ **adc_isr()** ფუნქციის შესრულებაზე. ამასთან აიკრძალება გლობალური წყვეტა. **adc_isr()** ფუნქციის ბოლოს გაიშვება ახალი გადასახვის პროცესი. წყვეტის დამმუშავებელი პროგრამიდან გამოსვლის შემდეგ კვლავ ნება დაერთვება წყვეტას და პროგრამა უბრუნდება დაუსრულებელ **whil(1)** ციკლს. შუქდიოდები აჩვენებენ სქემის შესასვლელზე მიწოდებული ძაბვის მნიშვნელობას ორობით კოდში.

თავი 2

პროგრამების გაწყობა და ტრანსლირება

იმისათვის, რომ მომხმარებლის მიერ დაწერილი პროგრამა გადაიქცეს მანქანურ კოდად და შესრულდეს კონკრეტულ მიკროკონტროლერში, უნდა მოხდეს მისი ტრანსლირება და მიკროკონტროლერის პროგრამულ მეხსიერებაში მისი ,,გაკერვა'' (ჩაწერა).

ამჟამად არსებობს ტრანსლატორების (კომპილატორების) დიდი რაოდენობა, რომლებიც შექმნილია სხვადასხვა მიკროპროცესორებისათვის პროგრამულ ენიდან მანქანურ კოდში ტრანსლირებისათვის (მაგალითად, AVR ოჯახის მიკროკონტროლერებისათვის-AVRStudio, WinAVR, CodeVisionAVR და სხვა კომპილატორები).

ჩვენს მიერ განხილული პროგრამებისთვის გამოყენებულია Atmel ფირმის მიერ AVR მიკროპროცესორებისათვის დამუშავებული კომპილატორი MikroC PRO for AVR, რომელიც ასრულებს C ენაზე დაწერილ პროგრამის ტრანსლირებას მანქანურ კოდში.

2.1. MikroC PRO for AVR პროგრამული არე

2.1. სურათზე ნაჩვენებია MikroC PRO for AVR კომპილატორის მთავარი პანელი, რომელიც დაყოფილია რამდენიმე სარკმელად. სურათზე ისინი აღნიშნულია ციფრებით 1,2,3,4,5.

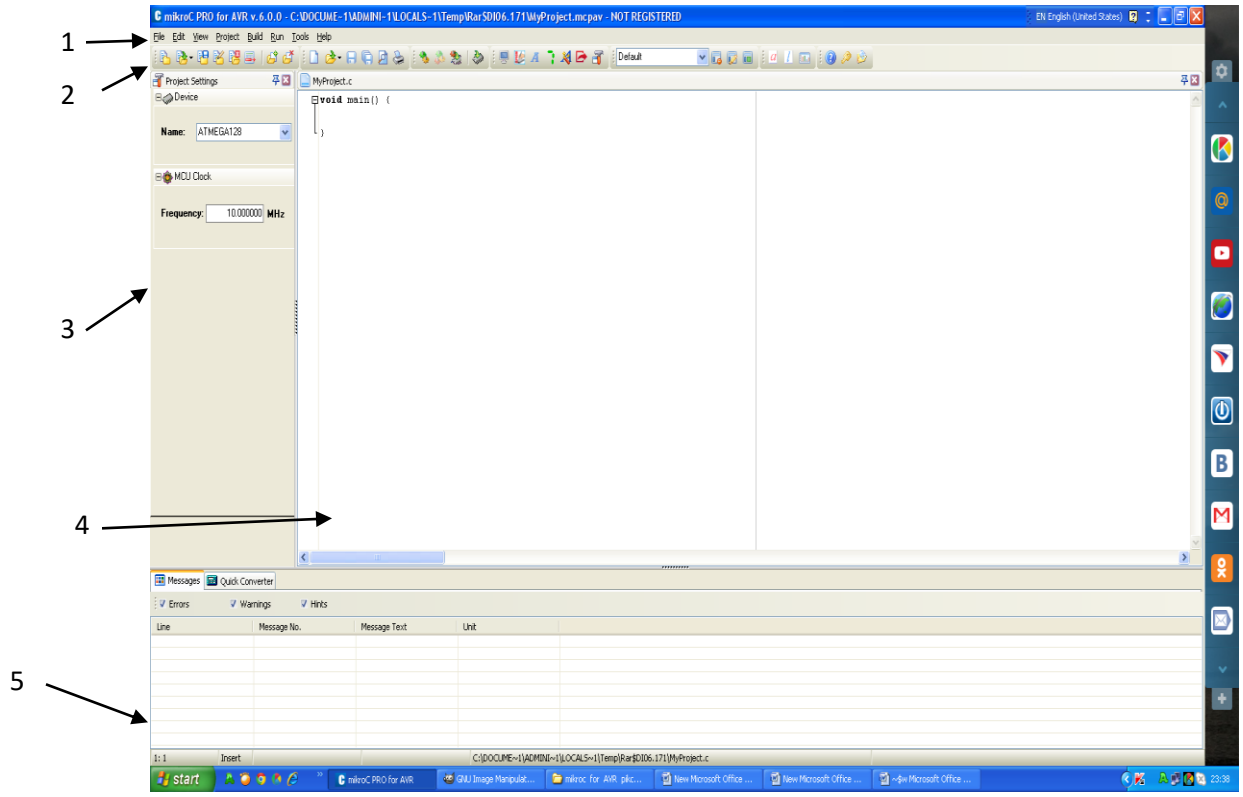
1 სტრიქონი წარმოადგენს **მენიუს სტრიქონს,** ამ სტრიქონის **File** და **Edit** მენიუს შემცველობა იგივეა, რაც **windows** -ის ფანჯარაში. **View** მენიუ შეიცავს ბრძანებებს მიმდინარე პროექტის ეკრანზე კონტროლისათვის.

მენიუების სტრიქონი 2 წარმოადგენს **ინსტრუმენტალურ პანელს,** სადაც გამოყვანილია ზოგიერთი ბრძანების ღილაკი. ისევე, როგორც **windows** -ის ნებისმიერ სხვა პროგრამაში, მენიუს საშუალებით ხდება MikroC PRO for AVR პროგრამის ყველა ფუნქციის გამოძახება და რეჟიმებზე გადართვა. ფანჯარა 3-ში ხორციელდება პროექტის პარამეტრების დაყენება: მიკროპროცესორის და მისი მუშობის სიხშირის არჩევა. პანელი 4 წარმოადგენს ძირითად ველს, სადაც იწერება მიკროკონტროლერის დასაკომპილირებელი სამოქმედო პროგრამა. ძირითადი პანელის ქვევით მოთავსებულია შეტყობინების პანელი 5, რომელსაც აქვს ორი ჩანართი. **Messages** ჩანართში, რომელიც უთქმელობით არის გაშლილი, აისახება ტრანსლირების პროცესი, შეტყობინებები სინტაქსური შეცდომების შესახებ, და სხვადასხვა გაფრთხილებები სიის სახით. შეცდომის მიზეზი და მისი ადგილი პროგრამაში აღინიშნება წითელი ფერით. თუ პროგრამაში ადგილი ჰქონდა შეცდომას მისი ტრანსლირება არ სრულდება შეცდომის აღმოფხვრამდე.

MikroC PRO for AVR მუშაობს ე.წ. **პროექტებთან**. თითოეული პროექტისათვის ხისტ დისკზე უნდა გამოყოფილი იყოს ცალკე კატალოგი.

MikroC PRO for AVR-ში ერთდროულად შეიძლება ჩატვირთული იყოს მხოლოდ ერთი პროექტი. პროექტი შეიცავს მთელ ინფორმაციას დასამუშავებელი პროგრამის და გამოყენებული მიკროკონტროლერის შესახებ და შედგება ფაილების დიდი ანაკრეფისგან. მათგან მთავარია პროექტის ფაილი. მას აქვს გაფართოება mcpay. პროექტის ფაილი შეიცავს

სხვადასხვა ცნობას პროცესორის ტიპის, გენერატორის სიხშირის შესახებ და სხვა. იგი ასევე შეიცავს მასში შემავალი ფაილების აღწერას, რომელიც გამოიყენება კომპილაციის დროს.



სურ.2.1.

თუ პროგრამამ ტრანსლაციის პროცესი გაიარა წარმატებით, შედეგად ფორმირდება რამდენიმე ფაილი, რომელთა შორის ჩვენთვის მნიშვნელოვანია ე.წ. hex ფაილი (ორობით ფორმატში წარმოდგენილიო ფაილი), რომელიც უნდა ჩაიწეროს მიკროკონტროლერის პროგრამის მეხსიერებაში.

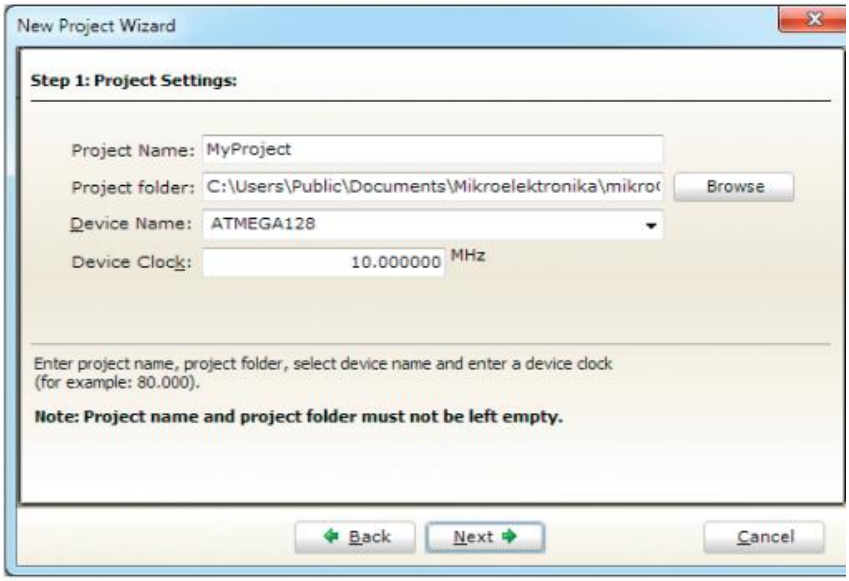
P2.2. პროექტის შექმნა

თუ MikroC PRO for AVR უკვე დაყენებულია თქვენს პერსონალურ კომპიუტერზე, კომპილატორის გამოძახების შემდეგ ეკრანზე გაიხსნება ფანჯარა, რომლის 4 და 5 პანელი ცარიელია. ახალი პროექტის შესაქმნელად ავირჩიოთ 1 პანელის მენიუ **Project**-დან პუნქტი **New Project** .გაიხსნება პირველი ფანჯარა (სურ. 2.2). იგივე მოქმედება შეგვიძლია შევასრულოთ ინსტრუმენტალური პანელის **New Project** ღილაკზე დაჭერით.

აღნიშნული ფანჯრით იწყება ახალი პროექტის შექმნა. მასში ჩამოთვლილია ის მოქმედებები, რომელიც ამ დროს შესრულდება. ამ ფანჯარაში არაფრის დაყენება არ ხდება, მხოლოდ უნდა დავაჭიროთ Next ღილაკს.

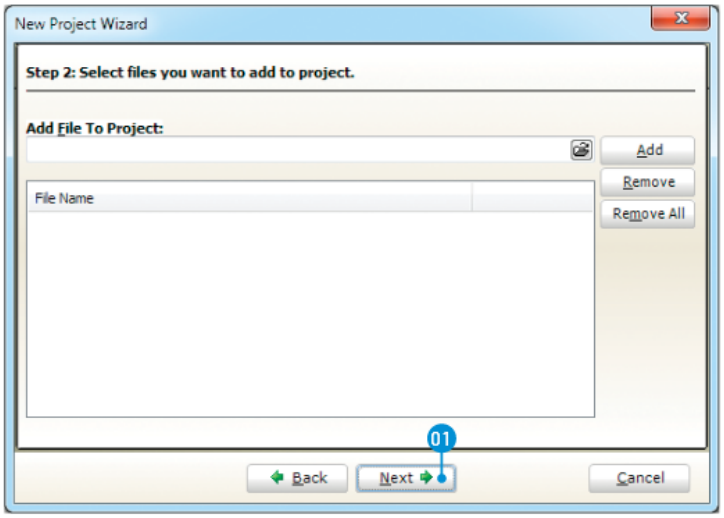
სურ. 2.2.

E ეკრანზე გაიხსნება მომდევნო ფანჯარა (სურ.2.3).



სურ. 2.3.

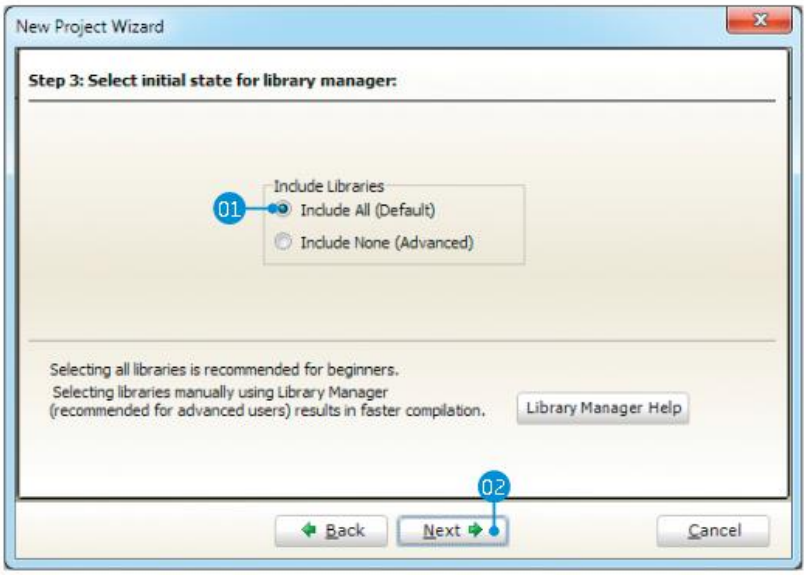
ამ ფანჯარაში ხდება სხვადასხვა პარამეტრების დაყენება: **Project name** ველში უნდა ჩაიწეროს ჩვენს მიერ დაყენებული პროექტის სახელი, **Project folder** ველში- გზა იმ საქაღალდისკენ, რომელშიც გვსურს პროექტის შენახვა, **Device name** ველში- ჩაიწერება პროექტში გამოყენებული მიკროპროცესორის ტიპი, **Device clock** ველში - პროცესორის სატაქტო სიხშირე. შემდეგ დავაჭიროთ **Next** ღილაკს. ეკრანზე იშლება მესამე ფანჯარა (სურ.2.4)



სურ.2.4.

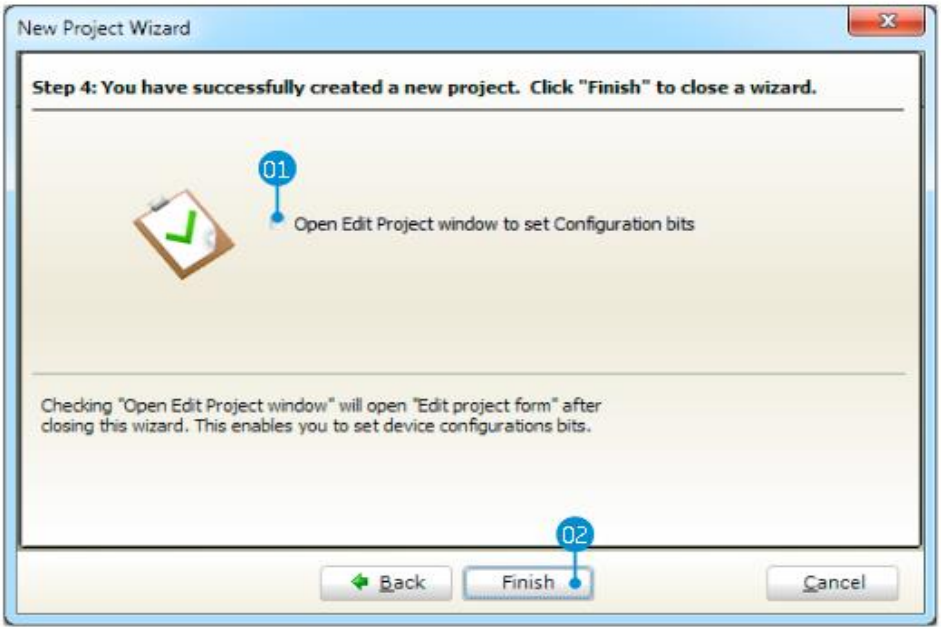
ეს ფანჯარა შესაძლებელია გამოვიყენოთ შედგენილ პროექტში ფაილების დამატებისთვის. რადგან ჩვენს მიერ შესასრულებელ ლაბორატორიულ სამუშაოებში ფაილების დამატება არ არის გათვალისწინებული, მას ყურადღებას არ ვაქცევთ, დავაჭიროთ **Next** ღილაკს. გაიხსნება ახალი ფანჯარა (სურ. 2.5.).

A ამ ფანჯრის საშუალებით შეგვიძლია პროექტში ჩავსვათ ფუნქციების ბიბლიოთეკები. ამ ბიბლიოთეკების საშუალებით კომპილატორი იყენებს ფუნქციებს, რომლებიც წარმოადგენენ პროგრამირების ენის სტანდარტული ფუნქციების ერთობლიობას პროცესორის სხვადასხვა ბლოკისათვის. მაგალითად, აცგ ბლოკისთვის, ინტერფეისების პროტოკოლების შესრულებისთვის, ტაიმერთან მუშაობისთვის და სხვა. მოვნიშნოთ სტრიქონი **Include All** (მონიშნულია უთქმელობით) და დავაჭიროთ **Next** ღილაკს.



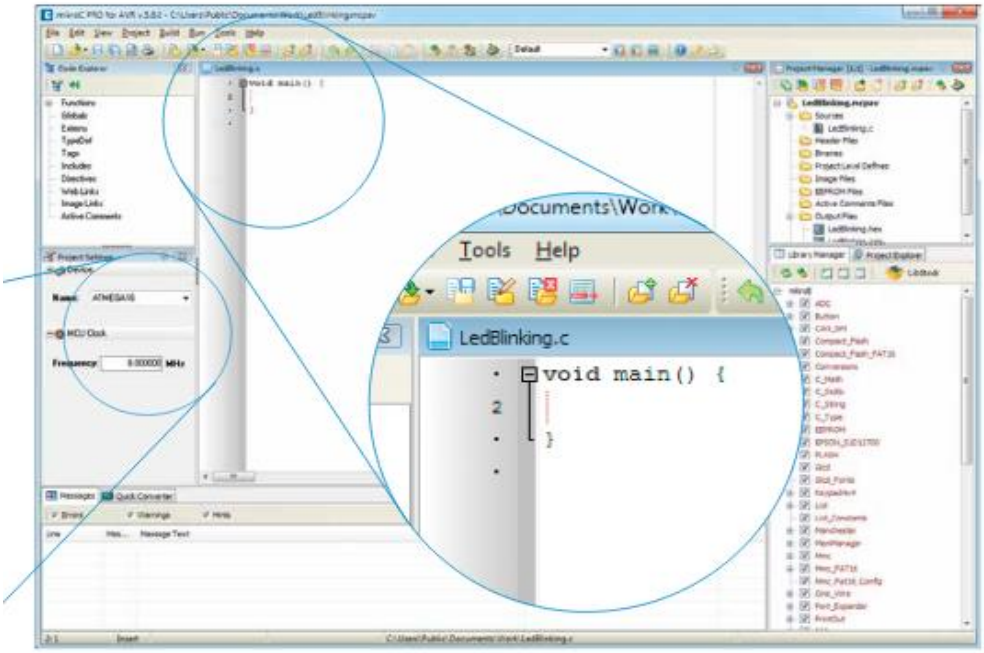
სურ.2.5.

გაიხსნება ბოლო ფანჯარა (სურ.2.6.), რომელიც გვატყობინებს პროცესის დამთავრებას. ყოველგვარი დაყენების გარეშე დავაჭიროთ **Finish** ღილაკს.



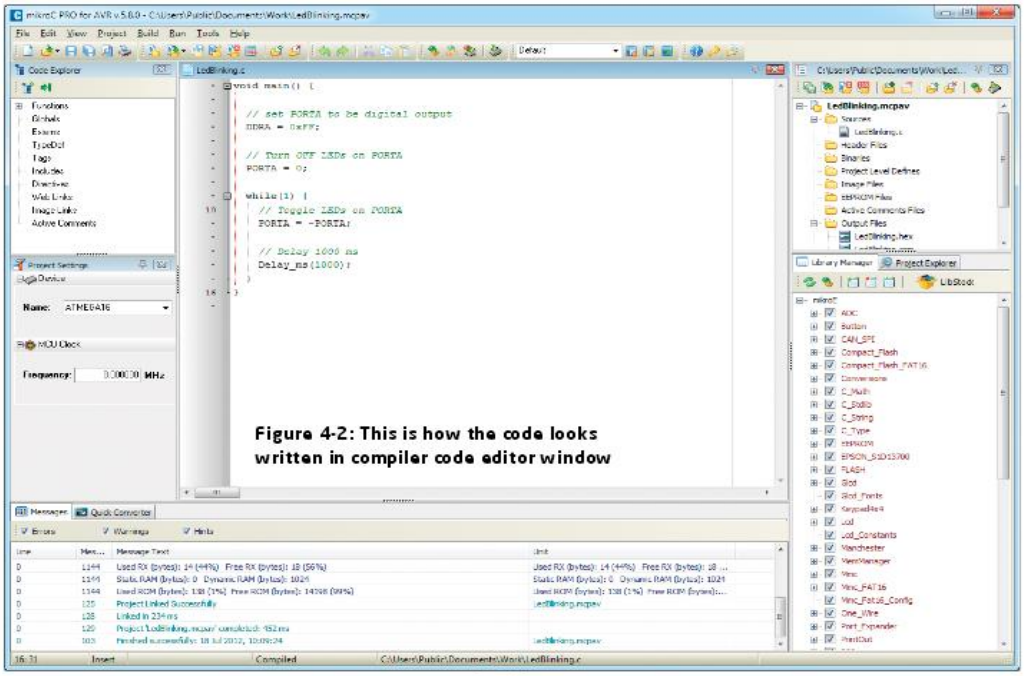
სურ. 2.6.

ეკრანზე გახსნილი იქნება ახალი პროექტის ბლანკი (სურ. 2.7.). იგი შეიცავს მხოლოდ მთავარ **main ()** ფუნქციის ცარიელ ბლანკს. მასში უნდა ჩაიწეროს პროგრამა C ენაზე, რომელიც ჩვენს მიერ შეიქმნება.



სურ. 2.7.

პროექტის ფანჯარას პროგრამის შედგენის შემდეგ ექნება შემდეგი სახე (სურ.2.8.):



Page 15

სურ.2.8.

2.3. პროგრამის ტრანსლირება

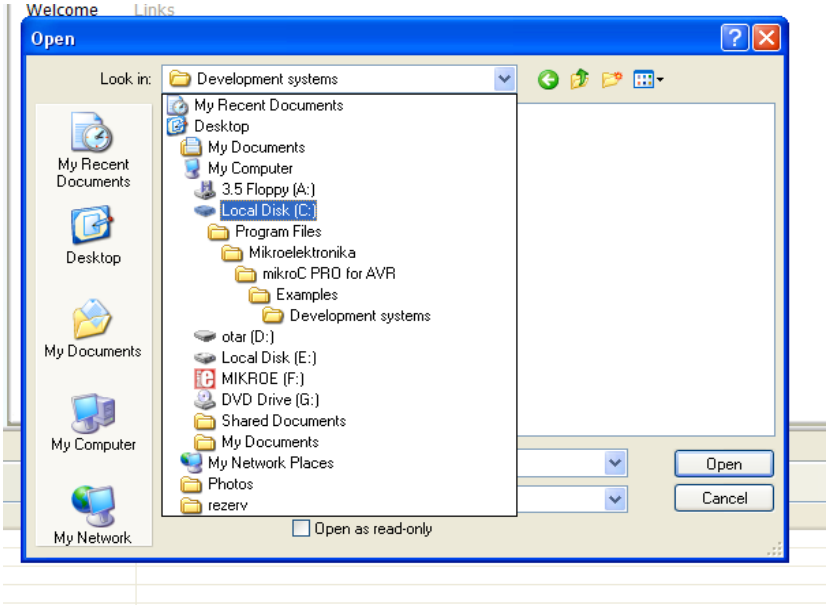
მთავარ ფანჯარაში პროგრამის ჩაწერის შემდეგ, უნდა შესრულდეს მისი ტრანსლირება, რის შედეგადაც მივიღებთ სხვადასხვა ფორმატში წარმოდგენილ რამდენიმე ფაილს. მათგან ჩვენთვის მნიშვნელოვანია **hex**-ფორმატში (ორობით კოდში) წარმოდგენილი ფაილი, რომელიც განკუთვნილია მიკროკონტროლერის პროგრამის მეხსიერებაში ჩასაწერად.

Pპროგრამის ტრანსლირებაზე გაშვება ხდება **Build** მენიუდან **Build** ბრძანების ამორჩევით ან ინსტრუმენტულ პანელში **Build** ღილაკზე დაჭერით. ტრანსლირების პროცესში კომპილატორი მთავარ პროგრამაში სვამს დამატებით ფაილებს და ამოწმებს პროგრამას სინტაქსურ შეცდომებზე (სურ.2.8.).კომპილირების თითოეული ეტაპის შესახებ შეტყობინება მოცემულია **Messages** ჩანართის სტრიქონებში. შეცდომის შემთხვევაში შესაბამისი შეტყობინება წითელი ფერით იქნება მოინიშნება. აქვე მოცემული იქნება შეცდომის ადგილი პროგრამაში და შეცდომის მიზეზი. მის საფუძველზე შესაძლებელია შეცდომის გასწორება, წინააღმდეგ შემთხვევაში ტრანსლირება არ შესრულდება.

P 2.4. პროექტის გამოძახება

Aპროექტები ინახება ერთ-ერთ საქაღალდეში, რომელიც თქვენს მიერ მითითებული იყო ახალი პროექტის შექმნის დროს (სურ.2.3.). პროექტის გამოძახებისთვის **Project** მენიუში ამოვირჩიოთ ბრძანება **Open Project** ან ინსტრუმენტულ პანელზე დავაჭიროთ **Open Project** ღილაკს. ეკრანზე გაიხსნება ფანჯარა (სურ.2.9.), რომლის ზედა ნაწილში არსებული გასაშლელ სიაში უნდა

ვიპოვოთ ჩვენთვის საჭირო პროექტის დასახელება. **Open** ღილაკზე დაჭერით ეკრანზე გამოდის ამორჩეული პროექტის ფანჯარა.



სურ. 2.9.

2.5. პროექტის რედაქტირება

რედაქტირების მიზანია პროექტის ზოგიერთი პარამეტრის ცვლილება. ამისათვის **Project** მენიუდან უნდა ავირჩიოთ ბრძანება **Edit Project** ან ინსტრუმენტალურ პანელზე დავაჭიროთ ღილაკს **Edit Project**. იხსნება რედაქტირების ფანჯარა. ლაბორატორიული სამუშაოს მოთხოვნის ფარგლებში შესაძლებელია მიკროპროცესორის ტიპის და მისი სატაქტო სიხშირის შეცვლა.

2.6. პროექტის დამახსოვრება

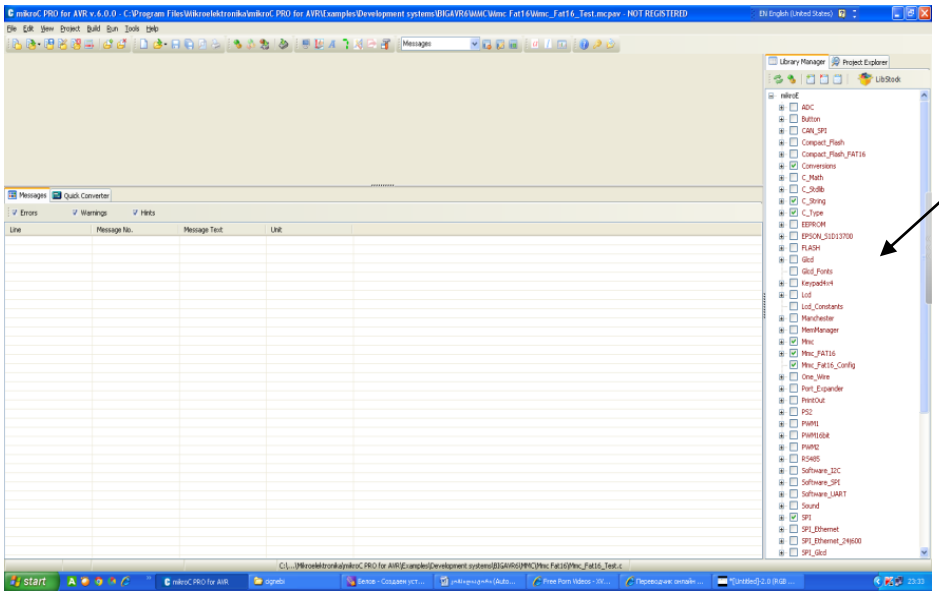
პროექტის შენახვისათვის **Project** მენიუდან უნდა ამოვირჩიოთ ბრძანება **Save Project** ან **Save Project As**...პირველ შემთხვევაში პროექტი ჩაიწერება მიმდინარე საქაღალდეში, რომლიდანაც იგი იყო ამოკითხული. მეორე შემთხვევაში ეკრანზე დგება ფანჯარა, რომელშიც შეგვიძლია ავირჩიოთ პროექტის ჩასაწერად ჩვენთვის სასურველი საქაღალდე.

2.7. პროექტის დახურვა

პროექტის დახურვა სრულდება **Project** მენიუდან **Close Project** ბრძანების არჩევით. ეკრანზე გამოდის ფანჯარა, რომელშიც გეძლევათ წინადადება პროექტის შენახვაზე. თქვენ შეგიძლიათ შეინახოთ პროექტი (Yes) ან დახუროთ შენახვის გარეშე (No).

2.8. ფუნქციების ბიბლიოთეკა

პროგრამის დაწერის დროს მოხერხებულია ვისარგებლოთ კომპილატორის ფუნქციებით, რომლებიც ბევრად აადვილებენ პროგრამირების პროცესს. ეს ფუნქციები თავმოყრილია ფუნქციების ბიბლიოთეკაში, რომლის გამოძახება შეგიძლიათ **Wiev** მენიუდან **Library Manager** ბრძანების ამორჩევით. ეკრანზე გამოდის ფუნქციების სია (სურ.2.10.), რომლითაც თქვენ შეგიძლიათ ისარგებლოთ. აქვეა ამ ფუნქციების აღწერა.



სურ.2.10.

ფუნქციები დაჯგუფებულია მიკროკონტროლერის ბლოკების მიხედვით. ბიბლიოთეკის ყოველ ელემენტზე მაუსის დაწკაპუნებით იხსნება შესაბამისი ბლოკის ფუნქციების სია. თითოეულ დაწკაპუნებით კი იხსნება ფუნქციის აღწერის ფანჯარა. კომპილატორის ფუნქციების გამოყენება ბევრად აადვილებს პროგრამის შექმნის პროცესს.

2.9. პროგრამული სიმულატორი (Debugger)

პროგრამული სიმულატორი არის MicroC PRO for AVR გარემოს განუყოფელი კომპონენტი. იგი შექმნილია AVR მიკროპროცესორის ოპერაციების სიმულაციისaTvis და მომხმარებლის დასახმარებლად ამ მოწყობილობებისთვის დაწერილი პროგრამების გაწყობაში.

პროექტის კომპილაციის წარმატებით შესრულების შემდეგ, შესაძლებელია პროგრამული სიმულატორის გაშვება **Run >Start Debugger** - ის ჩამოსაშლელი მენიუდან ან **Debugger Toolbar**- ის **Debugger Icon**- ის დაჭერით.

Watch Window არის პროგრამული სიმულატორის მთავარი ფანჯარა, რომელიც საშუალებას გაძლევთ შევასრულოთ თქვენი პროგრამის ელემენტების მონიტორინგი

სიმულაციის დროს. **Watch Window** ფანჯრის გამოყვანისათვის ეკრანზე უნდა ავირჩიოთ **View>Debug Windows** ჩამოსაშლელი მენიუდან **Watch Windows . Watch Window** ასახავს მიკროპროკონტროლერის ცვლადებსა და რეგისტრებს მათ მისამართებთან და შემცველობასთან ერთად. არსებობს ცვლადი / რეგისტრის სიაში დამატების ორი გზა:

- მისი ნამდვილი სახელით (ცვლადის სახელი კოდში). აირჩიეთ სასურველი ცვლადი / რეგისტრი ჩამოსატვირთი მენიუს სიიდან და დააჭირეთ **+Add** ღილაკს.
- მისი სახელით (ID სახელი ასამბლერში). შეიყვანეთ ცვლადის სახელი ID. ცვლადის შეყვანა უნდა შესრულდეს ასამბლერის სახელების ყუთიდან და **+Add** ღილაკზე დაჭერით.

ცვლადი შესაძლებელია წაიშალოს **Watch Window** - დან, აირჩიეთ ცვლადი, რომელიც გსურთ ამოიღოთ და დააჭირეთ ღილაკს **Remove.**

Add All უმატებს ყველა ცვლადს.

Remove All ყველა ცვლადის წაშლა.

ცვლადზე ორმაგი დაჭერით ან **Properties** ღილაკზე დაწკაპუნებით იხსნება **Edit Value** ფანჯარა, რომელშიც შეგიძლიათ დაამატოთ შერჩეული ცვლადის/რეგისტრის ახალი მნიშვნელობა.

გარდა ამისა, შეგიძლია ავირჩიოთ ცვლადი / რეგისტრი სხვადასხვა ფორმატში: ათობით, ორობით-ათობით, ორობით, მცოცი მძიმით ან სიმბოლოთი.

პროგრამული სიმულატორის **Stopwatch** ფანჯარა ხელმისაწვდომია მენიუდან, **View >Debug Windows > Stopwatch. Stopwatch** ფანჯარა გვიჩვენებს მიმდინარე ციკლებiს დროს პროგრამული სიმულატორის შესრულების შემდეგ. **Stopwatch** ზომავს შესრულების დროს (ციკლების რაოდენობას) პროგრამული სიმულატორის მუშაობის დაწყებიდან. მისი საშუალებით შეsაძლებელია აღვადგინოთ ნებისმიერ დროის პერიოდი.

სიმულატორის **View RAM** ფანჯრის საშუალებით შესაძლებელია დავათვალიეროთ მიკროკონტროლერის ოპერატიული მეხსიერების კარტა. მისი გახსნა ხდება მენიუდან **View >Debug Windows > View RAM**. რომელიმე უჯრის შემცველობის შეცვლა აღინიშნება წითელი ფერით.

P2.10. პროგრამის ჩაწერა მიკროკონტროლერში

ტრანსლირების ნორმალურად დამთავრების შემდეგ უნდა შესრულდეს hex-ფორმატში წარმოდგენილი პროგრამის ჩაწერა მიკროკონტროლერში. პროგრამის ჩასაწერად მიკროკონტროლერის მეხსიერებაში, გამოიყენება სპეციალური მოწყობილობა - **პროგრამატორი.**

პროგრამატორი წარმოადგენს პროგრამულ-აპარატურულ კომპლექს, რომელიც შესდგება მიკროკონტროლერის კომპიუტერთან დამაკავშირებელ მოწყობილებისა და ამ მოწყობილობის მართვის პროგრამისაგან. პროგრამატორს შეყავს მიკროპროცესორისათვის მომზადებული პროგრამა მის მეხსიერებაში.

AVR მიკროპროცესორების დაპროგრამების გავრცელებული ხერხი არის შიგა სქემური დაპროგრამირება საკომუნიკაციო ინტერფეისი SPA-ის გამოყენებით. აღნიშნული რეჯიმი მოხერხებულია იმით, რომ იგი შესაძლებლობას იძლევა დავაპროგრამოთ მიკროპროცესორი მზა სქემაში, მისი ამორჩილვის გარეშე.

პროგრამატორი ერთის მხრივ უკავშირდება კომპიუტერს და იმართება სპეციალური პროგრამის საშუალებით. მეორეს მხრივ პროგრამატორს უკავშირდება მიკროკონტროლერის მიკროსქემა. ნებისმიერ მიკროკონტროლერს გააჩნია სპეციალური რეჟიმი **-პროგრამირების რეჟიმი.** ამ რეჟიმში მიკროსქემის ყველა ან ზოგიერთი გამომყვანი იცვლის თავის ფუნქციებს. ახალ რეჟიმში ისინი იღებენ პროგრამატორიდან მონაცემებს და მმართველ სიგნსლებს. პროგრამირების რეჟიმზე გადართვა სრულდება **Reset** შესასვლელის საშუალებით. როგორც ცნობილია, მიკროკონტროლერის მუშა მდგომარეობის დროს ამ შესასვლელზე უნდა არსებობდეს ლოგიკური ერთი. შესასვლელზე ნულოვანი პოტენციალის მიწოდება იწვევს მიკროპროცესორის განულებას. უფრო ზუსტად, განულებისათვის საჭიროა **Reset** შესასვლელზე დაბალი პოტენციალი მივაწოდოთ მოკლე დროის განმავლობაში, შემდეგ ისევ გადავიყვანოთ ეს შესასვლელი ერთის მდგომარეობაში. თუ დაბალი პოტენციალი შესასვლელზე მივაწოდეთ ხანგრძლივი დროის განმავლობაში, მიკროკონტროლერი გადავა პროგრამირების რეჟიმში.

ამჟამად დამუშავებულია და ფართოდ გამოიყენება სხვადასხვა პროგრამატორების დიდი რაოდენობა. საკმარისია ამოვირჩიოთ მათგან ჩვენთვის შესაფერისი.

პროგრამატორების კლასიფიცირება შესაძლებელია სხვადასხვა მახასიათებლების მიხედვით.

PP დაპროგრამების მეთოდების მიხედვით. ძირითადად გამოიყენება ორი ხერხი:

- პარალელური დაპროგრამება;P
- მიმდევრობითი დაპროგრამება.

პარალელური დაპროგრამების შემთხვევაში მიკროკონტროლერის სქემას ჩვეულებრივად იღებენ მოწყობილობიდან, სადაც იგი უნდა მუშაობდეს და ათავსებენ პროგრამატორის გასართში. დაპროგრამების შემდეგ იგი ისევ უნდა დავაბრუნოთ მუშა სქემაში.

მიმდევრობითი დაპროგრამება არ მოითხოვს მიკროსქემის ამოღებას მოწყობილობიდან. შიგა სქემური დაპროგრამების ISP არხი, რომელიც ამ შემთხვევაში გამოიყენება, შესაძლებლობას იძლევა შესრულდეს შიგა სქემური დაპროგრამება, ე.ი. შიგ მოწყობილობაში, კვების გამორთვის გარეშე.

პარალელური დაპროგრამება უფრო რთულია პროგრამატორის რეალიზაციისა და დაპროგრამების თვალსაზრისით. მიმდევრობითი დაპროგრამება კი მეტად ადვილად რეალიზდება, არ მოითხოვს მაღალ ძაბვას, მუშაობს მაშინაც კი, როცა მიკრიკონტროლერი უკვე მირჩილულია მუშა სქემაში, რასაც სწორედ ეწოდება შიგა სქემური დაპროგრამება (ISP-In System Programmer).

სირთულის მიხედვით.

- უნივერსალური პროგრამატორი;
- სპეციალიზებული პროგრამატორი.

U უნივერსალური პროგრამატორები შესაძლებლობას იძლევიან დავაპროგრამოთ სხვადასხვა სახის მიკროსქემები. ამასთან, არა მხოლოდ ერთი სერიის ფარგლებში, არამედ სხვადასხვა სერიისა და მწარმოებლისაც კი.

სპეციალიზირებული პროგრამატორები გათვალისწინებულია ერთი სერიის მიკროსქემების ან მხოლოდ ერთი კონკრეტული ტიპის მიკროსქემის დასაპროგრამებლად.

შემდეგი პარამეტრი, რომლის მიხედვით შეიძლება ყველა პროგრამატორის კლასიფიცირება არის **კომპიუტერთან მიერთების საშუალება.**

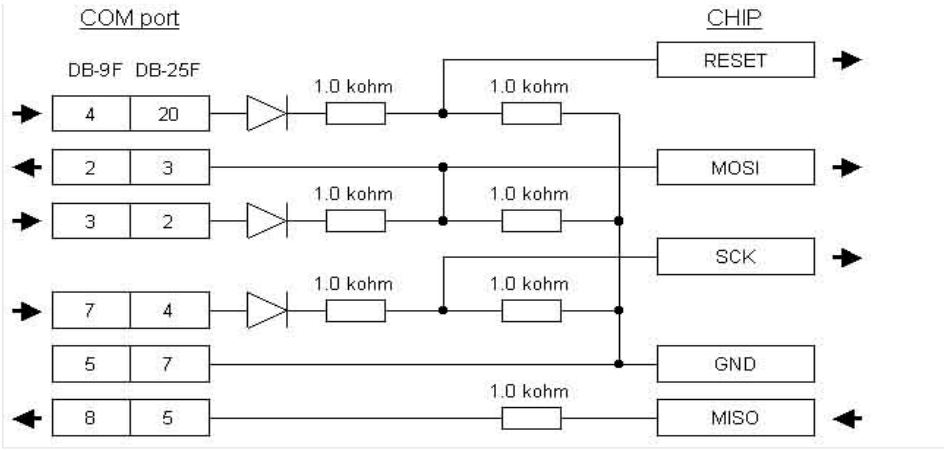
არსებობს კომპიუტერთან მიერთების სხვა და სხვა საშუალება, კერძოდ: AK

P • პარალელური პორტით (LPT);

• მიმდევრობითი პორტით (COM);

• USB სალტით.

როგორც მოყვანილი კლასიფიკაციიდან ჩანს, არსებობს პროგრამატორების ფართო სახესხვაობა. ყველაზე მარტივი ვარიანტი წარმოადგენს რამდენიმე გამტარს, რომლებიც ერთის მხრივ უერთდებიან კომპიუტერის რომელიღაც პორტს, მეორეს მხრივ მიკროკონტროლერის გამომყვანებს. საილუსტრაციო მაგალითად მოგვყავს გრომოვის პროგმატორად წოდებული სქემა (სურ.2.11.)

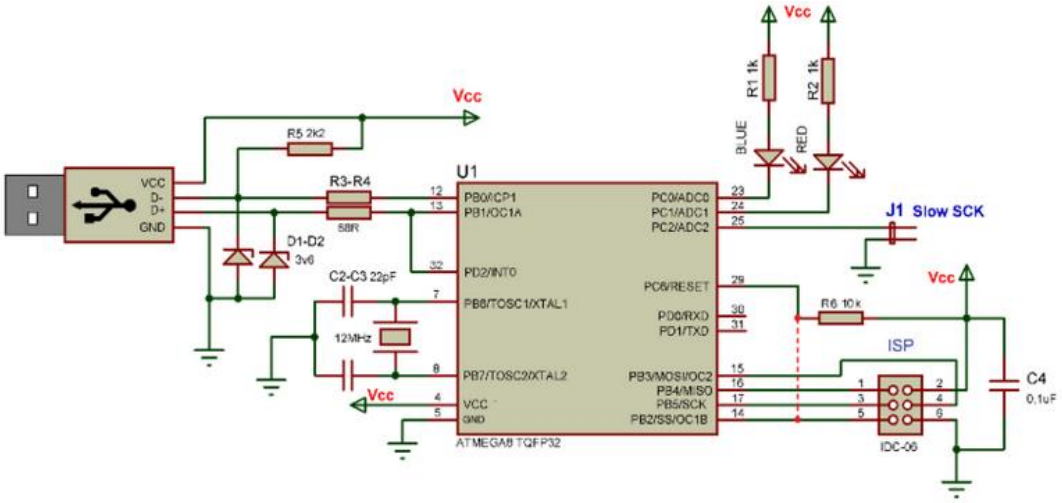


სურ.2.11.

სქემა ერთის მხრივ მიმდევრობითი (COM) პორტით- RS-232ინტერფეისის საშუალებით უკავშირდება კომპიუტერს, ხოლო SPI ინტერფეისის გამომყვანებით (MISO, MOSI, SCK, RESET)- დასაპროგრამირებელ მიკროკონტროლერს. ნაჩვენები პროგრამატორი გამოიყენება AVR ოჯახის მიკროკონტროლერის დასაპროგრამებლად.

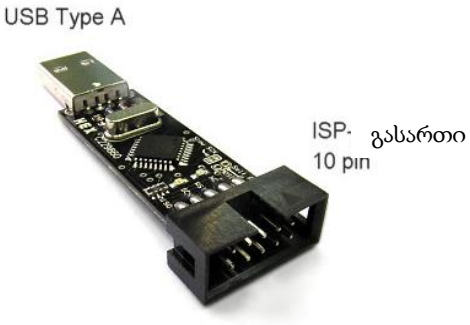
უფრო რთული პროგრამატორები შეიცავენ აგრეთვე მიკროკონტროლერებს. საილუსტრაციოდ სურ.2.12-ზე მოგვყავს USBasp პროგრამატორი , რომელიც აწყობილია Atmega8 ან Atmega48 მიკროკონტროლერზე. პროგრამატორი უერთდება კომპიუტერს USB

ინტერფეისით.



sur. 2.12

Eეს გადაწყვეტა ამჟამად მეტად პოპულარულია, ვინაიდან თანამედროვე კომპიუტერებში პარალელური და მიმდევრობითი პორტები აღარ იხმარებიან. ISP გასართის საშუალებით სქემა უკავშირდება დასაპროგრამირებელი მიკროკონტროლერის SPI გამომყვანებს. პროგრამატორის წარმოდგენილი სქემა უზრუნველყოფს სქემურ დაპროგრამირებას, რომლის დროსაც მიკროპროკონტროლერის მოწყობილობიდან ამორჩილვის აუცილებლობა არ არის. უნდა აღვნიშნოდ, რომ პროგრამატორის გამოყენების წინ მასში შემავალ მიკროპროკონტროლერში მწარმოებლის მიერ უნდა ჩაწერილი იყოს მმართველი პროგრამა. სურ.2.13-ზე წარმოდგენილია განხილული პროგრამატორის კონსტრუქციული სახე.



სურ.2.13

როგორც ზევით ითქვა, მიკროსქემის დაპროგრამებისათვის აგრეთვე ფართოდ გამოიყენება უნივერსალური პროგრამატორები, რომელთა საშუალებით შესაძლებელია სხვადასხვა ტიპის მიკროსქემების დაპროგრამება. საილუსტრაციოდ მოგვყავს ერთ-ერთი - ChipStar-Lynx პროგრამატორის კონსტრუქციული ხედი (სურ.2.14.) და ძირითადი მახასიათებლები.

სურ.2.14

იგი გამოიყენება 33340 დასახელების მიკროსქემების, მათ შორის, AVR და PIC მიკროკონტროლერების დასაპროგრამებლად და შეიცავს 40 კონტაქტიან DIP პანელს დასაპროგრამებელი სქემის მოსათავსებლად. სხვა კორპუსებში წარმოდგენილი მიკროსქემებისთვის (PLCC, TSOP, SO/SOIC) გამოყენებულია გადამყვანი ადაპტერები (სურ.2.15).

ა) ბ)

სურ.2.15

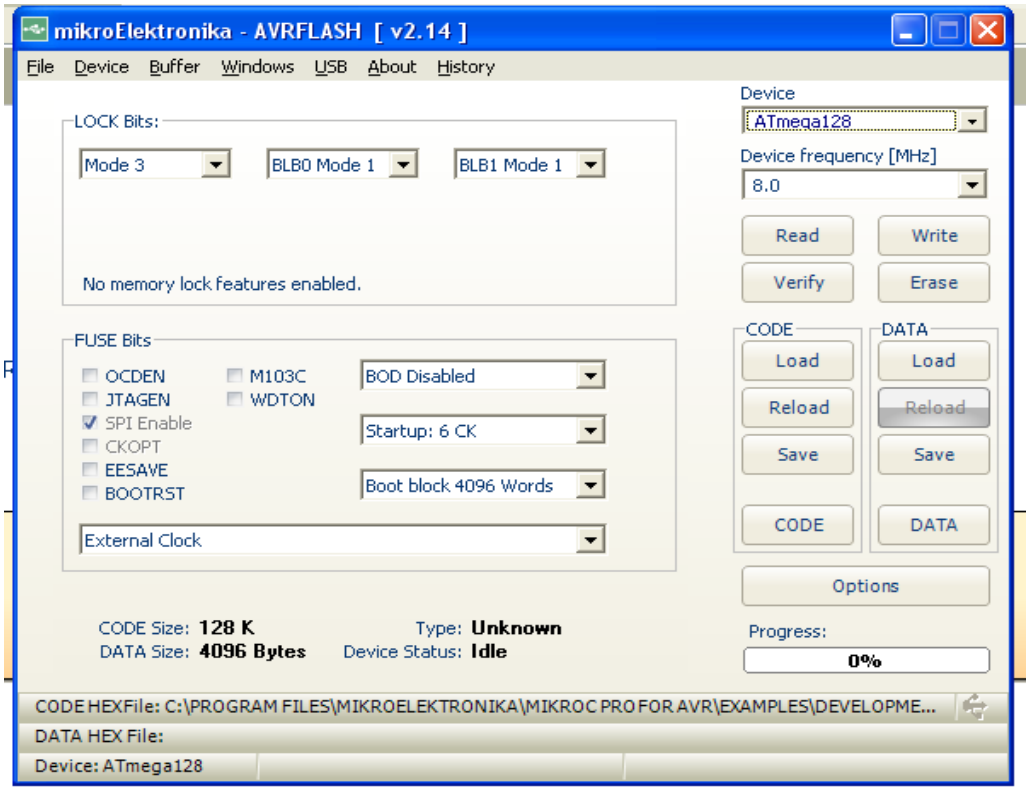
პროგრამატორი კომპიუტერს უკავშირდება USB ინტერფეისის გასართით, საიდანაც ხდება მისი მართვა. პროგრამატორს აგრეთვე აქვს შიგა სქემური დაპროგრამების გამომყვანი, რომლის საშუალებითაც შეუძლია ინტეგრალური სქემის დაპროგრამება მოწყობილობიდან მისი ამორჩილვის გარეშე.

პროგრამატორების ყველა მოდელი იმართება მმართველი პროგრამების (დრაივერების) საშუალებით, რომელთაც ასევე უწოდებენ პროგრამატორს. ამჟამად სხვადასხვა ფირმის მიერ დამუშავებულია პროგრამების დიდი რაოდენობა, რომლებიც გამოიყენებიან სხვადასხვა მიკროსქემების დასაპროგრამებლად. მაგალითად, პოპულარული პროგრამები: **Pony Prog** - გამოიყენება AVR მიკროკონტროლერების დასაპროგრამებლად. სურ.2.11-ზე ნაჩვენები პროგრამატორის გამოყენებით, **Khazma AVR vl.50 Programmer**, შეიძლება გამოყენებული იყოს სურ. 2.13-ზე მოყვანილ პროგრამატორის მართვისთვის, **AVRDUDE**- სურ.2.14-ზე მოყვანილი პროგრამატორისათვის და სხვა.

ჩამწერი პროგრამები, როგორც წესი, მუშაობენ რომელიმე კომპილატორის პროგრამულ არეში. მაგალითად, ზევით ხსენებული პროგრამები გამოიყენებიან **WinVR, Code Vision AVR, AVR Studio, UniProf** კომპილატორებთან ერთად, რომელთაც მხარს უჭერენ ოპერაციული სისტემები **Windows-XP, Windows-Vista, Windows-7, Windows-8, Windows-10, Linux.**

ქვევით საილუსტრაციოდ განიხილება დაპროგრამების მაგალითი **MikroC PRO for AVR** კომპილატორის შემადგენლობაში შემავალი **AVRFLASH** პროგრამატორის საშუალებით, რომელიც ჩვენს მიერ იყო გამოყენებული სახელმძღვანელოში წარმოდგენილი ამოცანების კომპილაციისა და პროცესორში ჩაწერისათვის.

ტრანსლირებული პროგრამის ჩაწერისათვის კომპილატორის **Tools** მენიუდან უნდა ამოვირჩიოთ ბრძანება **mE Programmer** ან ინსტრუმენტულ პანელზე დავაჭიროთ **Program** ღილაკს. ეკრანზე გაიხსნება პროგრამის ჩაწერის ფანჯარა (სურ.2.16). ჩაწერა სრულდება ავტომატურად. ფანჯრის მარჯვენა ნაწილის ქვევით მოთავსებულ ინდიკატორზე აისახება ჩაწერის პროცესის მსვლელობა. .



სურ.2.16

ზევით მოყვანილ ფანჯარაში შეგვიძლია შევასრულოთ მთელი რიგი დამატებითი მოქმედებები. ფანჯრის ზედა ნაწილში განლაგებულია მენიუების ზოლი, რომელიც შეიცავს რამდენიმე მენიუს დასახელებას:

- **File** მენიუ შეიცავს ჩვენს მიერ მითითებულ საქაღალდეში ფაილების ჩაწერის ან ამოკითხვის ბრძანებებს.

- **Device** მენიუში შედის ბრძანებები, რომელთა საშუალებით სრულდება მიკროპროცესორის მეხსიერებიდან წაკითხვა (**Read**), ჩაწერა (**Write**), მონაცემთა ვერიფიკაცია (**Verify**), მეხსიერების წაშლა (**Erase**).

- **Buffer** მენიუში შედიან ბუფერული უჯრედების წაშლის ან მათში შემთხვევითი რიცხვების შეტანის ბრძანებები (ბუფერების დასახელება და ადგილმდებარეობა მითითებულია ფანჯრის ქვედა ზოლში).

- **Windows** მენიუ შეიცავს ორ ბრძანებას **CODE View** - და **DATA View** , რომელთა საშუალებით შეგვიძლია მიკროკონტროლერის პროგრამის და მონაცემთა მეხსიერების შემცველობის ეკრანზე გამოყვანა და მონაცემთა მეხსიერების უჯრედებში ახალი ინფორმაციის შეტანა.

- **USB** მენიუში შემავალი **Show Devices** ბრძანების საშუალებით ვირჩევთ პროგრამატორის ჩვენს მიერ გამოყენებულ მოწყობილობას.

ზევით მოყვანილი ფანჯრის მარცხენა ნაწილში გამოყვანილია პარამეტრები, რომელთა დაყენება ხდება სხვადასხვა მოთხოვნის შესაბამისად:

- **Lock Bits** (ბლოკირების ბიტების) ზოლში არსებული ჩამოსაშლელი სიებიდან შეგვიძლია ამოვირჩიოთ სამომხმარებლო პროგრამისა და ჩამტვირთველის ურთიერთ მიმართვის დაბლოკვის რეჟიმები. ეს რეჟიმები გამოიყენება მხოლოდ მიკროპროკონტროლერში ჩამტვირთველის გამოყენების დროს. წინააღმდეგ შემთხვევაში ყველა სიიდან ამორჩეული უნდა იყოს **Mode 1** რეჟიმი.ამ ველში შეიძლება აგრეთვე ზოგიერთი საკონფიგურაციო ბიტების (Fuse Bit ველში) დაყენება:

- **SPI Enable** უნდა იყოს მონიშნული იმ შემთხვევაში, როცა პროგრამატორი **SPI** ინტერფეისის გამომყვანებთან არის მიერთებული (უთქმელობით მონიშნულია);

- **EESAVE** უნდა იყოს მონიშნული, თუ ჩაწერა საჭიროა EEPROM -ში (უთქმელობით არ არის მონიშნული);

- **BOOTRST** უნდა იყოს მონიშნული, თუ გამოიყენება ავტოდაპროგრამება (ჩამტვირთავის საშუალებით) (უთქმელობით არ არის მონიშნული);

- **WDTON** უნდა მოინიშნოს მოდარაჯე მთვლელის გამოყენების შემთხვევაში(უთქმელობით არ არის მონიშნული);

Aამავე ველში არის რამდენიმე ჩამოსაშლელი სია, რომლებიც განლაგებული არიან სვეტში. ზევიდან პირველ სიიდან შეიძლება ამოვირჩიოთ კვების ძაბვის ზღვრული მნიშვნელობა, რომლის ქვევით ძაბვის შემცირებისას კონტროლის სქემა (BOD) აიძულებს კონტროლერს გადავიდეს განულების რეჟიმში (უთქმელობით ამორჩეულია **BOD Disabled**). შემდეგ სტრიქონზე არსებული სიიდან შეგვიძლია დავაყენოთ მიკროკონტროლერის მუშობის დაყოვნების მნიშვნელობა მისი ჩართვის მომენტიდან (უთქმელობით არჩეულია 6 წმ).

მომდევნო სტრიქონზე არის სია, რომლიდანაც შეგვიძლია ამოვირჩიოთ ჩამტვირთველისათვის გამოყოფილი არის ზონა მიკროპროკონტროლერის პროგრამის მეხსიერებაში

ველის ქვედა ნაწილში მოთავსებულ ჩამოსაშლელ სიიდან შეგვიძლია ავირჩიოთ მიკროკონტროლერის სატაქტო იმპულსების წყარო. უთქმელობით მიკროკონტროლერი გაწყობილია შიდა გენერატორთან მუშაობაზე.

განხილული ფანჯრის მარჯვენა ველის ორ ჩამოსაშლელ სიიდან უნდა ავირჩიოთ დასაპროგრამებელი მიკროკონტროლერის ტიპი (ნაჩვენებ მაგალითში არჩეულია **Atmega 128**) და მუშაობის სიხშირე (მაგალითში არჩეულია 8.0 მჰც). ამავე ველში არის ღილაკები, რომელთა საშუალებით შესაძლებელია მიკროკონტროლერთან სხვადასხვა მოქმედებების შესრულება: **Read** - მიკროკონტროლერის კოდის ამოკითხვა ბუფერში, **Write** - მიკროკონტროლერში კოდის ჩაწერა ბუფერიდან, **Verify** - მიკროკონტროლერში არსებული კოდის ვერიფიკაცია ბუფერის შემცველობასთან, **Erase** - მიკროკონტროლერში კოდის წაშლა, **Load**- Hex-ფაილის შეტანა პროგრამის (**CODE**) ან მონაცემთა (**DATA**) ბუფერში, **Reload** - პროგრამული ბუფერის განახლება, **Save** - პროგრამული(**CODE**) ან მონაცემთა (**DATA**) ბუფერის შემცველობის ჩაწერა Hex-ფაილში.

როგორც ზევით იყო აღნიშნული, მიკროკონტროლერში პროგრამის ჩაწერა იწყება ავტომატურად, ფანჯრის გახსნასთან ერთად. ჩაწერის პროცესის ინდიკაცია ხდება **Progress** ზოლში. თუ არ არის მიკროპროკონტროლერზე სპეციალური მოთხოვნები, საკონფიგურაციო ბიტების მნიშვნელობების განსაზღვრა სრულდება უთქმელობით.

თავი 3

მიკროკონტროლერებზე აგებული მოწყობილობების პროექტირება

განვიხილოთ მიკროკონტროლერებზე მოწყობილობების პროექტირების ზოგიერთი ამოცანა

3.1. გადართვადი შუქდიოდები

ამოცანა ჩამოვაყალიბოთ შემდეგი სახით:

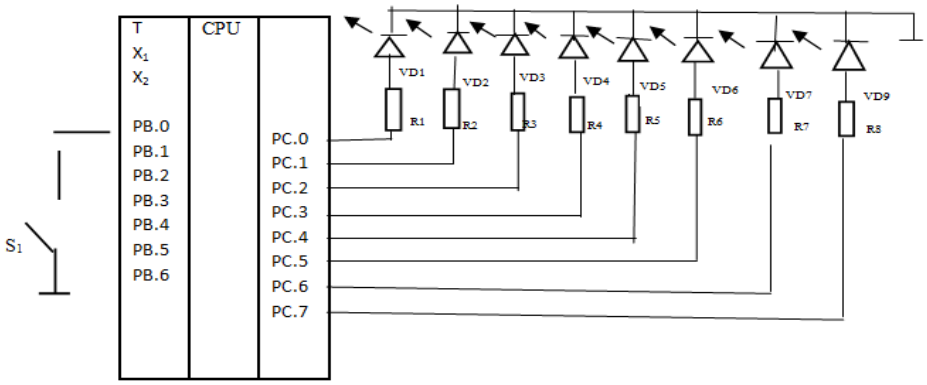
დავამუშაოთ მოწყობილობა 8 შუქდიოდის მართვისათვის ღილაკის გამოყენებით. შუქდიოდები უნდა ანათებდეს რაიმე ორობით კოდს. ღილაკის ყოველი დაჭერის შემდეგ შუქდიოდებზე გამონათებული კოდი ინვერტირდება.

პრინციპული ელექტრული სქემა

დავალების თანახმად უპირველეს ყოვლისა უნდა შევქმნათ ელექტრონული პრინციპული სქემა მიკროპროცესორის ბაზაზე, რომელიც შესძლებს ზემოთ ჩამოყალიბებული ამოცანის რეალიზაციას. მიკროკონტროლერს უნდა მიეერთოთ შუქდიოდები და მართვის ღილაკი. მიკროკონტროლერთან ნებისმიერი გარე მოწყობილობების მიერთებისთვის გამოიყენება შეყვანა - გამოყვანის პორტები. ამასთან ყოველ პორტს აქვს საშუალება იმუშაოს მონაცემთა როგორც შეყვანაზე, ასევე გამოყვანაზე.

მოსახერხებელია შუქდიოდები მიეერთოთ ერთ რომელიმე პორტს, ხოლო ღილაკი მეორე პორტს. ამ შემთხვევაში მმართველმა პროგრამამ ის პორტი, რომელთანაც მიერთებულია შუქდიოდები უნდა გააწყოს გამოყვანაზე, ხოლო პორტი, რომელიც მიერთებულია ღილაკთან - შეტანაზე. სხვა სპეციალური მოთხოვნები მიკროკონტროლერს არ წაეყენება. მაგალითვისათვის ავირჩიეთ Atmega 128 მიკროკონტროლერი, რომელსაც გააჩნია პერიფერიული მოწყობილობების ფართო სპექტრი და ჩვენს მიერ ის გამოყენებული იქნება

მომდევნო ამოცანებშიც. შევთანხმდეთ, რომ შუქდიოდების მართვისათვის გამოვიყენებთ C პორტის თანრიგებს (PC0-PC7 გამომყვანები), ხოლო მართვის ღილაკიდან ინფორმაციის ამოკითხვისათვის გამოიყენება B პორტის უმცროსი თანრიგი (PB.0 გამომყვანი). მოწყობილობის პრინციპიალური სქემა, რომელიც მოგვცემს საშუალებას განვახორციელოთ ჩვენს მიერ ჩამოყალიბებული ამოცანის რეალიზაცია მოცემულია სურ. 2.1-ზე.



სურ.2.1

S1 ღილაკის მიერთებისთვის გამოიყენება კლასიკური სქემა. საწყის მდგომარეობაში ღილაკის კონტაქტი გათიშულია. მოჭიმვის რეზისტორის გავლით მიკროკონტროლერის PB.0 შესასვლელს მიეწოდება "პლიუს" კვების ძაბვა, რაც შეესაბამება ლოგიკურ ერთიანს. AVR სერიის მიკროკონტროლერების პორტის ყოველ თანრიგს აქვს ჩაშენებული დატვირთვის რეზისტორი, რომლის ჩართვა შესაძლებელია პროგრამული გაწყობის გზით ყოველი პორტის გამომყვანისათვის, რითაც სრულდება ხსენებულ ძაბვის არსებობა შესაბამის შესასვლელზე.

ღილაკის დაჭერის დროს ხდება ძაბვის ნულამდე ვარდნა PB.0 პორტის შესასვლელზე, რაც შეესაბამება ლოგიკურ ნულიანის მიწოდებას. ამრიგად, პორტის შესაბამისი შესასვლელიდან სიგნალის მნიშვნელობის ამოკითხვით, პროგრამას შეუძლია განსაზღვროს ღილაკის დაჭერის მომენტი.

შუქდიოდის მიერთება ასევე ხორციელდება კლასიკური სქემით. მათი უშუალო მიერთებით პორტის გამოსასვლელებთან. პორტის ყოველი გამოსასვლელი გათვლილია შუქდიოდის მართვისათვის 20 მა დენის მოხმარებით. დიოდების წრედებში დენის შეზღუდვისთვის ჩართულია R1-R8 რეზისტორები.

შუქდიოდის ანთებისათვის, მიკროკონტროლერმა PC გამომყვანებით უნდა მიაწოდოს ლოგიკური ერთი. ასეთ შემთხვევაში ძაბვა რომელიც მოედება R-VD წრედს გამოიწვევს დენის გავლას შუქდიოდში და იგი აინთება. თუ PC გამომყვანს მიეწოდება ლოგიკური ნული, ძაბვის ვარდნა შუქდიოდზე და რეზისტორზე ნულის ტოლი იქნება, რაც შუქდიოდის ჩაქრობას გამოიწვევს.

მიზანშეწონილია გავაკეთოთ ზოგიერთი შენიშვნა: AVR ოჯახის უმეტეს მიკროკონტროლერებს, გარდა გარე კვარცული რეზონატორიან სატაქტო გენერატორისა, ასევე აქვთ შიგა RC გენერატორი, რომელიც არ მოითხოვს გარე წრედებს. იმ შემთხვევაში თუ მოცემულ გენერატორს არ წაეყენება რაიმე მაღალი მოთხოვნა სიხშირის სიზუსტესა და სტაბილურობაზე, შიძლება ავირჩიოთ შიგა გენერატორი (როგორც ეს გაკეთებულია შემოთავაზებულ სქემაში).

AVR ოჯახის ნებისმიერ მიკროკონტროლერს აქვს შიდა განულების სისტემა, რომელიც ხშირ შემთხვევაში უზრუნველყოფს სტაბილურ განულებას კვების ჩართვის დროს, ამის გამო მოცემულ სქემაში განულების ცალკე სქემა არ არის გათვალისწინებული. განულების გარე წრედები გამოიყენება იმ შემთხვევაში, როდესაც განსაკუთრებული მოთხოვნაა წაყენებული განულების იმპულსის ხანგრძლიობაზე.
ყველა ზემოთ აღწერილი გაწყობები სრულდება პროგრამულად ე.წ. fuse-გადამრთველის გამოყენებით.

ალგორითმი

ნებისმიერი პროგრამის შექმნა იწყება ალგორითმის დამუშავებით. ჩვენს შემთხვევაში ალგორითმი შემდეგია: საწყისი გაწყობის ოპერაციის შემდეგ მიკროკონტროლერი უნდა შევიდეს უწყვეტ ციკლში, ამ პროცესში მან უნდა მოახდინოს ღილაკთან მიერთებული შესასვლელის გამოკითხვა, და მისი მდგომარეობის შესაბამისად მართოს შუქდიოდი. აღვწეროთ დაწვრილებით ეს პროცესი.

საწყისი გაწყობის ოპერაციები:

- PC პორტი გავაწყოთ ინფორმაციის გამოტანაზე;
- ჩავწეროთ PC პორტის გამოსასვლელებზე რაიმე კოდი;
- მოვახდინოთ PB პორტის კონფიგურირება შესვლაზე;

ოპერაციები რომლებიც ქმნიან ციკლის ტანს

- უნდა მოხდეს PB (PB0) პორტის უმცროსი თანრიგის მდგომარეობის წაკითხვა;
- თუ ამ თანრიგის მნიშვნელობა ერთიანია (ღილაკი დაჭერილი არ არის), მაშინ PC გამოსასვლელების შემცველობა არ იცვლება;
- თუ PB.0 თანრიგის შემცველობა ნულია (ღილაკი დაჭერილია) მაშინ PC გამოსასვლელის შემცველობა შეიცვლება კოდის ინვერსულ მნიშვნელობით;
- მოხდება გადასვლა ციკლის დასაწყისზე.

პროგრამა C ენაზე

C ენაზე დაწერილი პროგრამისათვის ჩვენ გამოვიყენებთ MicroC Pro for AVR კომპილატორს. ეს პროგრამული გარემო სპეციალურად დამუშავებულია C ენაზე პროგრამების შესაქმნელად AVR მიკროკონტროლერებისთვის.

პროგრამა C ენაზე გამოიყურება შემდეგ სახით:

```
1. #Include < Atmega  128.h >
```

```
2. Bit  oldstate ;          //ძველი მდგომარეობის ალამი
3. Void  main () {          // მთავარი  ფუნქციის  დასაწყისი
4. DDB0  bit=0 ;            //PB0  გამომყვანის დაყენება ხდება, როგორც შესასვლელის
5. DDRC =  0xFF;            //PC-ის კონფიგურირება ხდება, როგორც გამოსასვლელის
6. PORTC=0xAA;              //C პორტში ჩაიწერება საწყისი კოდი
7. Oldstate =0;
8       . Do  {
 9. If (Button (& PINB,0,1,1))  {  // ლოგიგური ერთიანის აღმოჩენა  PB0-ის  შესასვლელზე
10. Oldstate =1;                        //ალმის განახლება
     }
11. if  (oldstate  &&   Button  (& PINB, 0,1,0)) { // 1-დან 0-ში გადასვლის აღმოჩენა
12 .PORTC= ~  PORTC;        //  PORTC ინვერტირება
13. oldstate=0;                       // ალმის განახლება
14.}  While (1);                    // უწყვეტი ციკლი
   }
```

წარმოდგენილ პროგრამაში სათაური და სხვადასხვა დანიშვნები, როლებსაც ასრულებს კომპილატორი, გამოტოვებულია, რამდენადაც ისინი უთქმელობით მიიღებიან. Include ბრძანება რომელიც 1-ელ სტრიქონშია განთავსებული აღმწერი ფაილის მიმაერთებელია. აღმწერი ფაილები (hader) შეიცავს მიკროკონტროლერის შემადგენლობაში შემავალი სხვადასხვა ელემენტების აღწერას: რეგისტრების ტიპებს და მისამართებს, წყვეტის ვექტორების დასახელებებს და მათ მისამართებს პროგრამულ მეხსიერებაში და სხვა. სტანდარტულ Mikro C PRO პაკეტს აქვს მთელი რიგი აღმწერი ფაილების ნაკრები AVR ოჯახის ყოველ მიკროკონტროლერისათვის. დამპროგრამებელს ევალება მხოლოდ შეარჩიოს საჭირო ფაილი და ჩართოს ის პროგრამის შესაბამის სტრიქონში. ფაილის მიერთების გარეშე პროგრამა არ იმუშავებს. Atmega 128 - თვის აქვს დასახელება Atmega128.h.

მე-2 სტრიქონში oldstate განსაზღრულია, როგორც ცვლადი ბიტი, რომელიც ღილაკის მდგომარეობაზე მიგვითითებს.

მე-3 სტრიქონიდან იწყება მთავარი main პროგრამა.

პროგრამის დასაწყისში ხორციელდება პორტების გამომყვანების დაყენება: PB.0, როგორც შესასვლელი (სტრიქონი 4); PC გამომყვანები განისაზღვრება, როგორც გამოსასვლელები (სტრიქონი 5); C პორტში იწერება 0xAA კოდი (სტრიქონი 6). oldstate ალამს ენიჭება საწყისი მნიშვნელობა- ნოლი (სტრიქონი 7).

მე-8 პუნქტიდან იწყება უსასრულო ციკლი do-while. if ოპერატორის გამოყენებით ხორციელდება Button (& PINB,0,1,1) ფუნქციის მნიშვნელობის შემოწმება. ეს ფუნქცია წარმოადგენს კომპილატორის ბიბლოთეკის ფუნქციას, რომლის პარამეტრებია: &PINB- შესამოწმებელი პორტი; პორტის შესამოწმებელი გამომყვანის ნომერი (განხილულ პროგრამაში ეს არის 0 ბიტი); დროითი დაყოვნება, ითვალისწინებს ღილაკის დაჭერის დროს კონტაქტის ვიბრაციას (პროგრამაში 1მწმ); ბიტის შესამოწმებელი მნიშვნელობა (პროგრამაში 1). ფუნქცია აბრუნებს ლოგიკური ერთიანს დამაკმაყოფილებელი პასუხის შემთხვევაში ან ლოგიკურ ნულიანს არადამაკმაყოფილებელი პასუხის შემთხვევაში.

ამგვარად, მე-9 პუნქტში სრულდება ერთიანის შემოწმება PB პორტის 0 ბიტში. თუ აღნიშნულ ბიტში აღმოჩნდება ერთიანი (ღილაკი აშვებულია), oldstate ალამს ენიჭება ერთიანის მნიშვნელობა (სტრიქონი 10), რის შემდეგ სრულდება B პორტის იმავე ბიტის ნულის მნიშვნელობაზე შემოწმება (სტრიქონი 11). თუ შესამოწმებელ ბიტში აღმოჩნდა ნულიანი და წინა მდგომარეობა ერთიანი იყო (oldstate ალამის მნიშვნელობა 1-იანია) ე.ი ღილაკმა შეიცვალა თავისი მდგომარეობა დაუჭერელ მდგომარეობიდან დაჭერილ მდგომარეობაში გადასვლით. ასეთ შემთხვევაში ხორციელდება PC გამომყვანების ინვერტირება (სტრიქონი 12) (შესაბამისად იცვლება შუქდიოდებზე გამოყვანილი კოდის მნიშვნელობა). თუ PB.0 მდგომარეობა არ იცვლება, შემოწმების დროს, მაშინ თითეულ if ოპერატორის გამოსახულების მნიშვნელობა, რომელიც ჩასმულია ფრჩხილებში არ აკმაყოფილებს პირობას, აქედან გამომდინარე PC გამომყვანების მნიშვნელობა არ იცვლება. პროგრამა გადადის While (პუნქტი 14) უწყვეტი ციკლის დასაწყისზე.

3.2. მორბენალი სხივი

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

დამუშავდეს მოწყობილობა, რომელიც შეასრულებს შუქდიოდების ოთხი ჯგუფის ციმციმის მართვას. თითეულ ჯგუფში შედის 8 შუქდიოდი. შუქდიოდები დასაწყისში უნდა ინთებოდეს ცალკეულ ჯგუფში თანმიმდევრობით, პირველიდან ბოლომდე, ხოლო შემდგომში უნდა მოხდეს იგივე თანმიმდევრობით შუქდიოდების ჩაქრობა.

პრინციპიალური ელექტრული სქემა

ამ ამოცანის გადასაწყვეტად შუქდიოდების თვითეული ჯგუფისთვის გამოიყენება იგივე პრინციპიალური სქემა, რომელიც ნაჩვენები იყო წინა შემთხვევაში. ღილაკის მიერთება ჩვენ არ დაგვჭირდება, ხოლო შუქდიოდების ჯგუფები უნდა მივუერთოთ PA, PB, PC, PD პორტების გამომყვანებს.

ვიზუალურად სქემის მუშაობა შესაძლებელია გამოიყურებოდეს შემდეგი სახით: პანელზე გამოვიყვანოთ შუქდიოდებისაგან შედგენილი მატრიცა, რომელიც შედგება ოთხი სვეტისა და რვა სტრისტრიქონისაგან. სვეტები შეესაბამებიან A,B,C,D პორტებთან მიერთებულ შუქდიოდებს, ხოლო სტრიქონები - პორტის თანრიგებს დაწყებული ზემოდან ქვემოთ 0-დან 7-მდე. სქემის მუშაობის პროცესში შუქდიოდები უნდა ინთებოდენ მიმდევრობით 0 სტრიქონიდან 7 სტრიქონამდე. შემდეგ ქრებოდენ იმავე მიმდევრობით.

ალგორითმი

აღვწეროთ პროგრამის შესრულების ალგორითმი
საწყისი გაწყობის ალგორითმი

- PA, PB, PC, PD პორტები გავაწყოთ ინფორმაციის გამოტანაზე;

- PA,PB,PC,PD პორტების გამოსასვლელებზე ჩავწეროთ ნოლები (ვაქრობთ ყველა შუქდიოდებს).

ოპერაციები რომლებიც ჰქმნის პროგრამის ტანს

- პორტების თანრიგებში თანმიმდევრობით შევიტანოთ ნოლები (ჩავაქროთ შუქდიოდები);
- პორტების თანრიგებში თანმიმდევრობით შევიტანოთ ერთიანები (ავანთოთ შუქდიოდები);
- მოხდეს გადასვლა ციკლის დასაწყისში.

პროგრამა C ენაზე

პოგრამას აქვს შემდეგი სახე :

```
1.  Char counter;  // ცვლადი  განისაზღვრა, როგორც თანრიგების  mTvleli
2.  Void  Wait ( ) {      //  დაყოვნების ფუნქციის აღწერა
3.  Delay  ms (100);
}
4.  Void  main ( )  {     // მთავარი პროგრამა
5.  DDRA= 0xFF;     //A  პორტის გამომყვანები დაყენება როგორც გამოსასვლელები
6.  DDRB= 0xFF; // //B პორტის გამომყვანების დაყენება როგორც გამოსასვლელები
7.  DDRC= 0xFF;    //C პორტის გამომყვანების დაყენება როგორც გამოსასვლელები
8.  DDRD= 0xFF;   //D პორტის გამომყვანების დაყენება როგორც გამოსასვლელები
9.  PORTA= 0x00; //  A  პორტის გამომყვანების დაყენება ნულში
10. PORTB= 0x00;  // B  პორტის გამომყვანების დაყენება ნულში
11. PORTC= 0x00;  // C პორტის გამომყვანების დაყენება ნულში
12. PORTD = 0x00; // D პორტის გამომყვანების დაყენება ნულში
13.  While  (1) {
14. For  (counter=0;  counter<8;  counter++) {
15. PORTA |= 1<< counter;       // ერთიანის დაძვრა A პორტში
16. PORTB |=  1<< counter;      //  ერთიანის დაძვრა B  პორტში
17. PORTC |=  1<< counter;      //  ერთიანის დაძვრა C  პორტში
18. PORTD |=  1<< counter;      //  ერთიანის დაძვრა D  პორტში
19. Wait ( );                   //  დაყოვნება
 }

20. Counter=0;
21.  While (counter<8 ) {
22.  PORTA &= ~ (1<< counter);  //  ნულიანის დაძვრა A პორტში
23. PORTB &= ~ (1<< counter);   //  ნულიანის დაძვრა B პორტში
24. PORTC &= ~ (1<< counter);   //  ნულიანის დაძვრა C პორტში
25. PORTD &= ~ (1<< counter);   //  ნულიანის დაძვრა D პორტში


26. Wait  ( );                  //  დაყოვნება

27. Counter++;                  //  მთვლელის ინკრიმენტი
 }
```

}

დასაწყისში ხდება დაყოვნების ფუნქციის აღწერა, რომელიც გამოიყენება შუქდიოდების მდგომარეობის გადართვის დროს. შუქდიოდების მდგომარეობის დაყოვნება აუცილებელია მათი ვიზუალური აღქმისთვის. დაყოვნების გარეშე შუქდიოდის ციმციმი განურჩეველი იქნება ადამიანის თვალისთვის.

ძირითადი პროგრამა იწყება პორტების ინიციალიზაციით: რამდენადაც A,B,C,D პორტების გამომყვანებზე მიერთებულია შუქდიოდები, ხდება მათი დანიშვნა გამომყვანად ერთიანების შეტანით აღნიშნული პორტების DDR რეგისტრებში (5-8 სტრიქონები). საწყის მდგომარეობაში შუქდიოდები ჩამქრალი უნდა იყოს, ამიტომ შესაბამის პორტებში ვწერთ ნულებს (9-12 სტრიქონები)

შემდეგ იწყება უწყვეტი While ციკლი (13 სტრიქონი). დასაწყისში სრულდება For ოპერატორი (14-18 სტრიქონი), რომლებიც ასრულებენ თანმიმდევრულად ერთიანების შეტანას პორტებში. ყოველი ციკლის ბიჯზე პორტის შემცველობაზე სრულდება “ან” ოპერაცია უმცროსი თანრიგის მიმართ მარცხნივ დაძრულ ერთიანის იმდენი თანრიგით, რომელსაც განსაზღვრავს Counter ცვლადი, შედეგი იწერება პორტში. შედეგად პორტის თანრიგში ჩაწერილი აღმოჩნდება ერთიანი, რომელიც ანთებს შუქდიოდს. რამდენადაც ციკლის ყოველი გასვლის შემდეგ Counter მნიშვნელობა იზრდება ერთით, ხდება ერთიანის შეტანა პორტის მომდევნო თანრიგებში, რის შედეგადაც ხორციელდება ყველა ჯგუფებში შუქდიოდების თანმიმდევრული ანთება. ციკლი დასრულდება იმ შემთხვევაში ,როდესაც პორტების ყველა თანრიგები ერთიანებით შეივსება (როდესაც counter=7) – აინთება ყველა შუქდიოდი.

რაიმე დაყოვნების შემდეგ (wait ფუნქცია) (19 სტრიქონი), იწყება შუქდიოდის ჩაქრობის ციკლი. ამისთვის გამოიყენება while ოპერატორი (21-15 სტრიქონი). ყოველ ციკლის ბიჯზე პორტის შემცველობაზე (სადაც ჩაწერილია ერთიანები) სრულდება “და ” ოპერაცია ნულით, რომელიც მიიღება, მას შემდეგ, როდესაც მოხდება ინვერტირება მარცხნივ დაძრული ერთიანის იმდენი თანრიგით, რომელსაც განსაზღვრავს counter ცვლადი. ციკლის დასაწყისში ცვლადი მიუთითებს პორტის უმცირეს თანრიგზე, რის გამოც ნული ამ თანრიგში ჩაიწერება და შესაბამისი თანრიგი ჩაქვრება. ციკლის შესრულების პროცესში counter მნიშვნელობა იზრდება ერთით, ამით ის მიუთითებს პორტების მომდევნო თანრიგებზე, სადაც შეიტანება ნულიანები.

შედეგად შუქდიოდები ჩაქვრება იმავე თანმიმდევრობით ,როგორც ეს შუქდიოდების ანთების დროს იყო. ციკლი გრძელდება counter=7 მნიშვნელობის მიღწევამდე (ნულების შეტანა პორტების ყველა თანრიგებში) ე.ი. მოხდება შუქდიოდის ყველა თანრიგების ჩაქრობა. დაყოვნების შემდეგ (სტრიქონი 26), პროგრამა გადადის While ციკლის დასაწყისზე (სტრიქონი 13).

3.3 მოციმციმე შუქდიოდები

ჩამოვაყალიბოთ ამოცანა შემდეგ სახით:

დამუშავდეს მოწყობილობა, რომელიც შეასრულებს შუქდიოდების ოთხი ჯგუფის ციმციმის მართვას. თვითეულ ჯგუფში შედის 8 შუქდიოდი. შუქდიოდები დასაწყისში უნდა ინთებოდეს ყველა ჯგუფში ერთდროულად, ხოლო შემდგომში მოხდეს შუქდიოდების ერთდროულად ჩაქრობა.

პრინციპიალური ელექტრული სქემა

ამ ამოცანის გადასაწყვეტად შუქდიოდების თვითეული ჯგუფისთვის გამოიყენება იგივე პრინციპიალური სქემა, რომელიც განხილული იყო წინა შემთხვევაში.

ალგორითმი

აღვწეროთ პროგრამის შესრულების ალგორითმი

პორტების საწყისი გაწყობა

- PA, PB, PC, PD პორტები გავაწყოთ ინფორმაციის გამოტანაზე;

ოპერაციები რომლებიც ჰქმნიან პროგრამის ტანს

- PA, PB, PC, PD გამოსასვლელებზე ჩავწეროთ ნულები (ვაქრობთ ყველა შუქდიოდს).
- პორტების თანრიგებში შევიტანოთ ერთები (ავანთოთ შუქდიოდები);
- პორტების თანრიგებში შევიტანოთ ნულები (ჩავაქროთ შუქდიოდები);
- მოხდეს გადასვლა ციკლის დასაწყისში.

პროგრამა C ენაზე

```
1.Void  main ( ) {      // მთავარი პროგრამა
2. DDRA= 0xFF;    // A  პორტის გამომყვანები დაყენება როგორც გამოსასვლელები
3.DDRB= 0xFF;        // B პორტის გამომყვანების დაყენება როგორც გსმოსასვლელები
4.DDRC= 0xFF;     // C  პორტის  გამომყვანების დაყენება როგორც გსმოსასვლელები
5.DDRD= 0xFF;     //  D პორტის  გამომყვანების დაყენება როგორც გსმოსასვლელები
6.Do  {
7.PORTA= 0X00;       //  A  პორტის გამომყვანებზე დიოდის  ჩაქრობა
8.PORTB= 0x00;        // B  პორტის გამომყვანებზე დიოდის  ჩაქრობა
9.PORTC= 0x00;        // C პორტის გამომყვანებზე დიოდის  ჩაქრობა
10.PORTD = 0x00;       //  D პორტის გამომყვანებზე დიოდის  ჩაქრობა
```

```
11.Delay_ms (1000);     //  დაყოვნება  1 წამი

12.PORTA= 0xFF;       //  A  პორტის გამომყვანებზე დიოდის  ანთება

13.PORTB= 0xFF;       //  B  პორტის გამომყვანებზე დიოდის  ანთება

14.PORTC= 0xFF;       // C პორტის გამომყვანებზე დიოდის  ანთება

15.PORTD = 0xFF;      // D პორტის გამომყვანებზე დიოდის

16.Delay_ms (1000);     // დაყოვნება  1 წამი

17. }  while (1);

  }
```

პროგრამა შედგება მხოლოდ ერთი main ფუნქციისაგან. ფუნქციის დასაწყისში სრულდება პორტების გაწყობა. ვინაიდან ოთხივე პორტი მუშაობს გამოყვანაზე, მათ DDR რეგისტრებში უნდა ჩავწეროთ ერთები (სტრიქონები 2-5). 6-17 სტრიქონებში სრულდება უსასრულო ციკლი Do-While. ციკლის ყოველ გასვლაზე დასაწყისში ოთხივე პორტის PORT რეგისტრებში იწერება ნულები, რაც იწვევს შუქდიოდების ჩაქრობას (სტრიქონები 7-10). 1წმ დაყოვნების შემდეგ (სტრიქონი 11), PORT რეგისტრებში იწერება ერთები (სტრიქონი 12-16) – შუქდიოდები ინთება. ვინაიდან აღნიშნული პროცესი მეორდება უსასრულოდ, შუქდიოდები იწყებენ ციმციმს.

3.4. ანალოგური სიგნალის გარდასახვა ციფრულ ფორმაში (აცგ)

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

დავაგეგმაროთ სქემა ანალოგური სიგნალის ციფრულ ფორმატში გარდასახვისათვის, შედეგის ორობით კოდში გამოტანით შუქდიოდებზე .

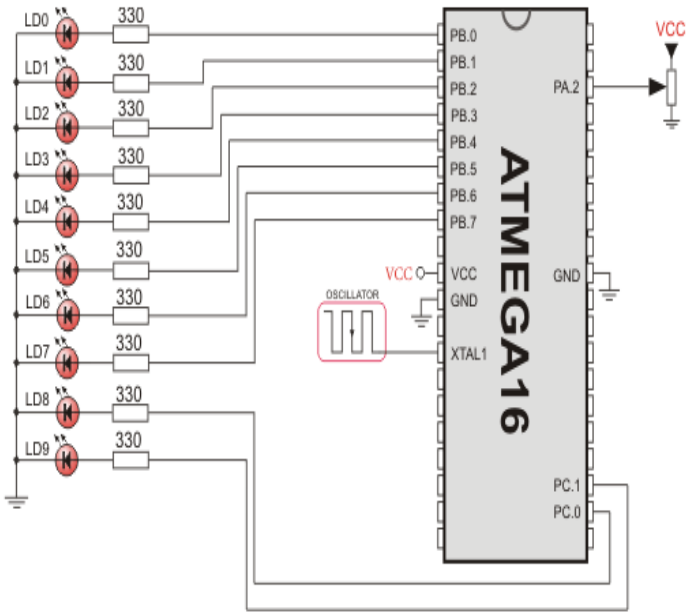
პრინციპიალური ელექტრული სქემა

ამ მიზნით გამოიყენება Atmega 128 მიკროკონტროლერში ჩაშენებული აცგ. აცგ ბლოკის შესასვლელებისთვის გათვალისწინებულია PC პორტის გამომყვანები, თვითეული მათგანი შეიძლება მიუერთდეს მას, თუ ADMUX რეგისტრში ჩაწერილია PC პორტის შესაბამისი გამომყვანის კოდი. ჩვენი პროექტისთვის ვირჩევთ PC2 პორტის შესასვლელს, რომელსაც მივაწოდებთ გარდასაქმნელ ანალოგურ სიგნალს. სიგნალები შეიძლება მივიღოთ სხვადასხვა გადამწოდებიდან (სენსორებიდან) ძაბვის სახით. მიკროკონტროლერის შესასვლელებს სიგნალები გარკვეული შეზღუდული სიდიდით მიეწოდება- ისინი არ უნდა აღემატებოდენ კონტროლერის კვების ძაბვას. ამიტომ, იმ შემთხვევაში, როდესაც გარდასასახი სიგნალის მნიშვნელობა აღემატება მითითებულ შეზღუდვას, მაშინ ეს სიგნალი მიეწოდება პორტის გამომყვანს პოტენციომეტრის მეშვეობით. გარდაქმნის შედეგი წარმოადგენს ათ თანრიგა ორობით კოდს. შედეგის ორობითი კოდის თანრიგების რაოდენობა მნიშვნელოვნად

განსაზღვრავს გარდაქმნის სიზუსტეს – რაც უფრო მეტია თანრიგების რაოდენობა, მით უფრო მაღალია სიზუსტე (გარჩევადობა).

გარდაქმნის შედეგის გამოტანისათვის გამოვიყენებთ შუქდიოდებს, რომელიც შეერთებული იქნებიან მიკროკონტროლერის პორტების გამოსასვლელებთან. იმის გამო, რომ შედეგი წარმოდგენილია 10 თანრიგით, ხოლო თვითეულ პორტი შეიცავს რვა თანრიგიან გამოსასვლელ რეგისტრს, გამოვიყენებთ PB და PD ორ პორტს. PB პორტის მეშვეობით გამოვიყვანთ 8 უმცროს თანრიგს, ხოლო PD პორტის მეშვეობით - 2 უფროს თანრიგს. პორტების გამომყვანები მიერთებული არიან შუქდიოდებთან, რომლებზედაც აისახება გარდასახვის შედეგი ორობითი კოდის სახით.

მოწყობილობის სქემა მოცემულია სურ.3.2-ზე.



სურ.3.2.

ალგორითმი

საწყისი გაწყობის ოპერაციების შესრულება

• გავაწყოთ PB და PD პორტები კოდების გაცემაზე, ხოლო PC პორტის გამომყვანები შესვლაზე. დანარჩენი დაყენებები: აცგ-სთან PC პორტის კონკრეტული შესასვლელის მიერთება, აცგ-ს სამუშაო სიხშირე, წყვეტის ნებადართვის ალამი დავტოვოთ ხელუხლებელი (უთქმელობით).

ოპერაციები რომლებიც ქმნიან პროგრამის ტანს

• შესრულდეს შემავალი სიგნალის გარდასახვა;

- გადაქმნის შედეგი გამოვიტანოთ შუქდიოდზე.

პროგრამა C ენაზე

```
1. #include   <built_in.h>;  // გამომყვანების დასახელების ფაილის მიერთება
2. Unsigned  int  adc_rd;     //  ორ ბაიტიანი ცვლადის განსაზღვრა
3. Void  main  ( ) {            //  მთავარი პროგრამის დასაწყისი
4. DDRB=0xFF;                  //   გავაწყოთ PB პორტი როგორც გამომყვანი
5.   DDRD=0xFF;                // გავაწყოთ   PD     პორტი როგორც გამომყვანი
6.  While (1)  {                  // უწყვეტი ციკლის დასაწყისი
7.  adc_rd=  ADC  read (2);  // გარდაქმნის შედეგის ამოკითხვა
8.  PORTB= adc_rd;              // შედეგის უმცროსი თანრიგის გამოტანა
9.  PORTC= Hi (adc_rd);     //   შედეგის უფროსი თანრიგის გამოტანა
  }
}
```

პროგრამა იწყება Include დირექტივით (სტრიქონი 1), რომლის მეშვეობით კომპილიატორი ახდენს იმ ფაილის მიერთებას ჩვენს მიერ C ენაზე დაწერილ პროგრამასთან, რომელიც განსაზღვრავს მიკროკონტროლერის ბლოკების გამომყვანებს. მომდევნო სტრიქონში (სტრიქონი 2) ხდება adc_rd ცვლადის გამოცხადება, როგორც უნიშნო ორბაიტიანის, რომელსაც ვიყენებთ გარდაქმნის შედეგის შესანახად. შემდეგ იწყება main მთავარი პროგრამა. 4 და 5 სტრიქონებში ხდება B და D პორტების გაწყობა გამოყვანაზე. 6-ე პუნქტიდან იწყება While უწყვეტი ციკლი. ციკლში სრულდება მიღებული ორობითი კოდის შედეგის ჩაწერა adc_rd ცვლადში (სტრიქონი 7). ამ ოპერაციაში გამოიყენება კომპილატორის ADC Read (2) ფუნქცია, რომელიც იწვევს გარდაქმნის პროცესის გაშვებას და ინახავს შედეგს აცგ შიდა ბუფერულ ბლოკში. ფუნქციის პარამეტრი არის შესასვლელის ნომერი , საიდანაც მოიხსნება გარდასაქმნელი სიგნალი (ჩვენს შემთხვევაში გვაქვს PC პორტის არხი 2). ციკლის დასრულების დროს შედეგის გამოყვანა ხდება PB და PD პორტებზე. რამდენადაც შედეგის კოდი წარმოდგენილია 10 ორობითი თანრიგით, ხოლო თითეული პორტი 8 გამომყვანიანია, ვიყენებთ ორ პორტს : უმცროს ბაიტს ვწერთ B პორტში (სტრიქონი 8), ხოლო უფროს 2 თანრიგს D პორტში. შემდგომ საჭიროა გადავიდეთ ციკლის დასაწყისზე. მიღებული შედეგის კოდის ინდიკაცია ხორციელდება შუქდიოდებზე, რომლებიც მიერთებულია PB და PD პორტებთან ზემოთ ნაჩვენები სქემის შესაბამისად.

3.5 ტაიმერის გამოყენება დაყოვნებისათვის

ამოცანის დასმა

წინათ განხილულ მაგალითებში დაყოვნების ფორმირებისათვის ჩვენს მიერ გამოყენებული იყო სტანდარტული ბრძანებები. ასეთი ხერხის გამოყენება ყოველთვის არ არის მიზანშეწონილი. ამ მეთოდის მთავარი უარყოფითი მხარე მდგომარეობს იმაში, რომ იგი მთლიანად ტვირთავს ცენტრალურ პროცესორს - სანამ პროცესორი დაკავებულია დაყოვნების ფორმირებით, მას არ შეუძლია სხვა ამოცანის შესრულება.

ამ მეთოდის კიდევ ერთი უარყოფითი მხარე - დაყოვნების დროის ზუსტი მნიშვნელობის არჩევის შეუძლებლობა. გაცილებით კარგ შედეგს იძლევა სხვა ხერხი - დროის ინტერვალების ფორმირება მიკროკონტროლერის შემადგენლობაში მყოფი ტაიმერ / მთვლელის საშუალებით. ყოველი მათგანი შეიძლება მუშაობდეს როგორც წყვეტის გამოყენებით, ისე წყვეტის გარეშე. შემდეგში ჩვენ განვიხილავთ ორივე ვარიანტს. დავიწყოთ უფრო მარტივი შემთხვევიდან.

გამოვიყენოთ ადრე განხილული ამოცანა " მოციმციმე შუქდიოდები ", რომელშიც შევცვალოთ დაყოვნების ფორმირების პროცედურა. ახალ პროცედურაში გამოვიყენოთ ერთ-ერთი შიგა ტაიმერი/მთვლელი წყვეტის გარეშე.

მოწყობილობის სქემა

ვინაიდან ჩვენ არ ვქმნით ახალ მოწყობილობას, არამედ მხოლოდ ვაუმჯობესებთ მმართველ პროგრამას, სქემა დარჩება იგივე (იხ.პარაგრაფი3.3).

ალგორითმი

Atmega 128 მიკროპროცესორის შემადგენლობაში შედის ოთხი ტაიმერ/მთვლელი. ავირჩიოთ, რომელიმე მათგანი.

ამოცანის ამომავალ წერტილს წარმოადგენს დაყოვნების დრო 1000 მწ. როგორც ცნობილია, დროითი ინტერვალების ფორმირებისათვის ტაიმერ/მთვლელმა უნდა დაითვალოს სისტემური გენერატორიდან მიწოდებული სატაქტო იმპულსები.

გენერატორის იმპულსების სიხშირე ჩვენ შემთხვევაში, დაუშვათ ტოლია 4 მჰც-ის. ხოლო იმპულსების პერიოდი 1/4=0,25 მკწ. ტაიმერ/მთვლელის გამოსასვლელზე 1000 მწ-ის მისაღებად საჭიროა გვქონდეს გაყოფის კოეფიციენტი ტოლი $1000 \cdot 10^{-3}/0,25 \cdot 10^{-6}=4000 \cdot 10^{3}$.

მიკროკონტროლერის ოთხი ტაიმერიდან ორი რვა თანრიგაა (T0,T2), ორი თექვსმეტ თანრიგაა (T1,T3). რვა თანრიგა ტაიმერს გადათვლის მაქსიმალური კოეფიციენტი (ტაიმერში ჩაწერილი მაქსიმალური მნიშვნელობა) აქვს $2^8=256$, ხოლო თექვსმეტ თანრიგა ტაიმერს - $2^{16}=65536$. როგორც ჩანს, თექვსმეტ თანრიგა ტაიმერიც კი არ არის საკმარისი საჭირო დაყოვნების მისაღებად. გამოვიყენოთ სიხშირის წინგამყოფი. აღნიშნული წინაგამყოფი ასრულებს სატაქტო სიგნალების წინასწარ გაყოფას ტაიმერ / მთვლელის შესასვლელზე მიწოდებამდე.

ავირჩიოთ დაყოფის კოეფიციენტის ყველაზე დიდი მნიშვნელობა - (1024). მაშინ წინგამყოფის გამოსასვლელზე მივიღებთ $4 \cdot 10^6/1024=3906$ ჰც სიხშირის სიგნალს . ამ სიგნალის პერიოდი არის 1/3906= 0,256• 10-3 წ ანუ 0,256 მწ. სწორედ ეს სიგნალი მიეწოდება ჩვენს ტაიმერს. ახლა გამოვითვალოთ დაყოფის კოეფიციენტი, რომელიც უნდა უზრუნველყოს ჩვენმა ტაიმერმა:

1000/0,256=3906,25. გადათვლის ასეთი კოეფიციენტი ჩვენთვის შეუძლია უზრუნველყოს T1 ან T3 ტაიმერ / მთვლელმა.

მას შემდეგ, რაც ამოვირჩიეთ ჩვენთვის გამოსადეგი ტაიმერი და გავერკვიეთ მის დაყენებაში, შეგვიძლია გადავიდეთ დაყოვნების პროგრამის შედგენაზე. მანამდის წარმოვადგინოთ მისი მუშაობის ალგორითმი. ეს ალგორითმი გულისხმობს, რომ ტაიმერის სიხშირის წინასწარი დამყოფის ყველა სჭირო დაყენებები უკვე შესრულებულია პროგრამის პირველ გამოძახებამდე და იმყოფება უსასრულო თვლის რეჟიმში.

ქვევით მოყვანილია დაყოვნების ქვეპროგრამის ალგორითმი.

1. ჩაიწეროს T1 ტაიმერის თვლის რეგისტრში ნული.
2. დაიწყოს თვლის რეგისტრის შემცველობის შემოწმების ციკლი. ციკლის ტანში მრავალჯერ უნდა იყოს ამოკითხული სათვლელი რეგისტრის შემცველობა და მოხდეს მისი ბოლო მნიშვნელობის შემოწმება (ე.ი. 3906,25).
3. თვლის რეგისტრის მიერ ბოლო მნიშვნელობის მიღწევის შემთხვევაში, დასრულდეს შემოწმების ციკლი.
4. დაყოვნების ქვეპროგრამიდან გამოსვლა.

მოყვანილი ალგორითმის მიხედვით შევადგინოთ დაყოვნების ფუნქცია ტაიმერ/მთვლელის საშუალებით. ამ ფუნქციის გამოყენების საილუსტრაციოთ განვიხილოთ ადრე წარმოდგენილი ამოცანა “მოციმციმე შუქდიოდები” –ის ახალი ვერსია ზოგიერთი ცვლილებების შეტანით.

პროგრამა C ენაზე

```
1. void  wait  (void)              // დაყოვნების  ფუნქცია

  {

2. TCNT1= 0;

3.  while ( TCNT1A< 1000)  {};

     }

4.  Void  main ( )  {      // მთავარი პროგრამა

5.DDRA= 0xFF;     // A  პორტის გამომყვანების   დაყენება როგორც გამოსასვლელები

6.  DDRB= 0xFF;   // B პორტის გამომყვანების დაყენება როგორც გამოსასვლელები

7. DDRC= 0xFF;  // C  პორტის  გამომყვანების დაყენება როგორც გამოსასვლელები

8.DDRD= 0xFF; //  D პორტის  გამომყვანების დაყენება როგორც გამოსასვლელები

9.TCCR1A=0x00;    //  მთვლელის  ინიციალიზაცია

10.TCCR1B=0x05;

11.ACSR=0x80;  // ანალოგური კომპარატორის ინიციალიზაცია-გამოთიშვა
```

```
12.Do  {
13.PORTA= 0X00;       //  A  პორტის გამომყვანებზე დიოდის  ჩაქრობა
14.PORTB= 0x00;       //  B  პორტის გამომყვანებზე დიოდის  ჩაქრობა
15.PORTC= 0x00;       //  C პორტის გამომყვანებზე დიოდის  ჩაქრობა
16.PORTD = 0x00;      // D პორტის გამომყვანებზე დიოდის  ჩაქრობა
17.Wait ();                 //  დაყოვნება  1 წამი
18.PORTA= 0xFF;     //  A  პორტის გამომყვანებზე დიოდის  ანთება
19.PORTB= 0xFF;       // B  პორტის გამომყვანებზე დიოდის  ანთება
20.PORTC= 0xFF;       // C პორტის გამომყვანებზე დიოდის  ანთება
21.PORTD = 0xFF;      // D პორტის გამომყვანებზე დიოდის  ანთება
22.wait ();    //  დაყოვნება  1 წამი
23.          }  while ();
    }
```

დაყოვნების ფუნქციის ტექსტი მოყვანილია 1-3 სტრიქონებში, რომელიც აღწერილია მთავარი ფუნქციის წინ C ენის წესის მიხედვით. ფუნქციას მიენიჭა სახელი wait. ფუნქცია wait-ს არა აქვს პარამეტრები, ვინაიდან გათვალისწინებულია დაყოვნების ფიქსირებული მნიშვნელობისათვის.

განვიხილოთ wait ფუნქცია უფრო დაწვრილებით. პირველი სტრიქონი (სტრიქონი 1) არის ფუნქციის აღწერის სათაური. ფუნქციის ტანს შეადგენს 2 და 3 სტრიქონი. სტრიქონ 2-ზე T1 ტაიმერ/მთვლელის TCNT1 თვლის რეგისტრს, ენიჭება ნული. 3 სტრიქონზე მოთავსებულია შემოწმების ციკლი. ეს არის ცარიელი ციკლი, რომლის პირობას წარმოადგენს გამოსახულება TCNT1<1000. შემოწმების ციკლი შესრულდება მანამდის, სანამ თვლის რეგისტრის შემცველობა გახდება 1000-ის ტოლი. ასეთ შემთხვევაში ფუნქცია დაითვლის დაყოვნებისათვის საჭირო დროის ინტერვალს

აღნიშნული ფუნქციის გამოყენებასთან დაკავშირებით ადრე განხილულ პროგრამაში საჭიროა ცვლილების შეტანა: Delay ბრძანების მაგივრად 19 სტრიქონში შევიტანოთ დაყოვნების ფუნქციის სახელი wait. მთვლელის გამოყენებასთან დაკავშირებით აუცილებელი გახდა პროგრამაში კიდევ რამოდენიმე ცვლილების შემოტანა, რაც ეხება მთვლელის ინიციალიზაციას. ინიციალიზაცია სრულდება TCCR1A და TCCR1B რეგისტრებში შესაბამისი კოდების ჩაწერით - TCCR1A რეგისტრის ყველა თანრიგში იწერება ნულები (სტრიქონი 9), მათ შორის 3 და 4 თანრიგებში, რაც უზრუნველყოფს მთვლელის მუშაობას Normal რეჟიმში, ხოლო TCCR1B რეგისტრში იწერება კოდი 0x05 (სტრიქონი 10), ანუ უმცროს სამ თანრიგში - კოდი 101, რომელიც შეესაბამება სიხშირის დაყოფის კოეფიციენტის ამორჩევას (1024) მთვლელის წინგამყოფში. პროგრამაში აგრეთვე დამატებულია ანალოგური

კომპარატორის ინიციალიზაციის პროცედურა, რომელიც ხორციელდება სტრიქონ 11-ზე. როგორც ცნობილია მიკროკონტროლერის მიერთების დროს კვებასთან ანალოგური კომპარატორი გადადის მუშა მდგომარეობაში. იმ შემთხვევაში, როდესაც ამოცანაში კომპარატორი არ არის გამოყენებული, სასურველია მისი გათიშვა, რაც სრულდება ACSR რეგისტრის მე-3 თანრიგში ერთის ჩაწერით (ACSR=0x08).

3.6 ტაიმერით წყვეტის მოთხოვნის გამოყენება დაყოვნებისათვის

წინა მაგალითში დაყოვნების ფორმირებისთვის ჩვენ გამოვიყენეთ ტაიმერი, მაგრამ არ გვისარგებლია მისი მთავარი უპირატესობით: წყვეტის გამოწვევის უნარით . პრაქტიკაში უფრო ხშირად მსგავს შემთხვევებში იყენებენ წყვეტას ტაიმერით. ეს შესაძლებელს ხდის უფრო ზუსტად დავაფორმიროთ დროის ინტერვალები და რაც უფრო მნიშვნელოვანია, ნებას იძლევა განიტვირთოს ცენტრალური პროცესორი. სანამ ტაიმერი აფორმირებს დაყოვნებას, პროგრამას შეუძლია შეასრულოს ნებისმიერი სხვა მოქმედებები.

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

დავამუშაოთ პოროგრამა “მოციმციმე შუქდიოდების” ახალი ვარიანტი დაყოვნების ფუნქციაში ტაიმერით წყვეტის გამოყენებით.

მოწყობილობის სქემა
სქემა დავტოვოთ ცვლილების გარეშე (იხ. პარაგრაფი 3.3).

ალგორითმი

ზევით დასმული ამოცანა მოითხოვს წინათ განხილული პროგრამის სრულ გადაკეთებას, ვინაიდან მოგვიწევს მთვლელის მუშაობის რეჟიმის ცვლილება. მოცემულ კონკრეტულ შემთხვევაში მიზანშეწონილია განივ იმპულსური მოდულიაციის CTC (“განულება თანხვედრის შემთხვევაში”) რეჟიმის გამოყენება, ამ რეჟიმში ტაიმერი პერიოდულად თითონ გამოიმუშავებს წყვეტის მოთხოვნებს წინასწარ განსაზღვრული პერიოდით.

შუქდიოდების ციმციმის მართვის ყველა ფუნქციას ასრულებს წყვეტის დამუშავების პროგრამა. შუქდიოდების ანთების და ჩაქრობის იგივე სიჩქარის უზრუნველყოფისათვის, რაც წინა პროგრამაში გვქონდა, უნდა გამოვიყენოთ სიხშირის დაყოფის იგივე კოეფიციენტები: ძირითადი სიხშირის გაყოფისათვის - 1024, თანხვედრის რეგისტრში ვწერთ 1000მწ-ის შესაბამის დაყოფის კოეფიციენტს.

ტაიმერის მუშაობის პროცესში სრულდება თვლის რეგისტრის ინკრიმენტი. როდესაც მისი მნიშვნელობა მიაღწევს შედარების რეგისტრის მნიშვნელობას ტაიმერი ავტომატურად ნულდება და აგრძელებს მუშაობას ნულიდან. ტაიმერის ნულოვან მდგომარეობაში გადასვლის მომენტში ფორმირდება წყვეტის მოთხოვნა და მიკროკონტროლერი გადადის წყვეტის მომსახურების პროგრამის შესრულებაზე.

ზევით თქმულიდან გამომდინარე, პროგრამის მუშაობის ალგორითმი შედგება ორი დამოუკიდებელი ნაწილისაგან.

უპირველეს ყოვლისა ეს არის ძირითადი პროგრამის ალგორითმი, მეორე, წყვეტის მომსახურების ალგორითმი.

ძირითადი პროგრამის ალგორითმი:

1.მიკროკონტროლერის შეყვანა-გამოყვანის პორტების გაწყობა;
2.ტაიმერის და წყვეტის სისტემის დაყენება;

3. შედარების რეგისტრში საწყისი მნიშვნელობის ჩაწერა;
4. ტაიმერის მუშაობის ნების დართვა;
5. წყვეტის ნების დართვა;
6. ძირითად ციკლის შესრულებაზე გადასვლა.

ვინაიდან შუქდიოდების ციმციმთან დაკავშირებულ ოპერაციებს ასრულებს წყვეტის მომსახურების პროცედურა, ძირითად ციკლში ხდება მხოლოდ ზოგიერთი დაყენების შესრულება.

პროგრამა C ენაზე

```
1.# include  <Atmega 128.h>
2.Bit oldstate=0;
3.interrapt [ TIME1_COMP]  void  timer1_com_isr(void)
        {
4.if  (oldstate==0)
5.      {
6.      PORTA= 0x00;
7.      PORTB= 0x00;
8.      PORTC= 0x00;
9.      PORTD = 0x00;
10.     oldstate=1
11.       }
12.     else
13.     {
14.     PORTA= 0xFF;
15.     PORTB= 0xFF;
16.     PORTC= 0xFF;
17.     PORTD = 0xFF;
18.     oldstate=0;
19.     }
20.     }
21.     void  main  (void)
22.     {
```

```
23.  DDRA= 0xFF;
24. DDRB= 0xFF;
25. DDRC= 0xFF;
26.DDRD= 0xFF;
27.TCNT1H=0x00;
28.TCNT1L=0x00;
29.TCCR1A=0x42;
30.TCCR1B=0x05;
31.OCR1AH=0x0F;
32.OCR1AL=0xA0;
33.TIMASK=0x10;        //
34.ACSR=0x80;          //
35.#asm("sei") ;       //
36.while(1)  {};     //
37.{
```

მოყვანილი პროგრამა წარმოადგენს ზემოთ განხილული "მოციმციმე შუქდიოდების" ამოცანის ახალ რედაქციას, როდესაც დაყოვნებისათვის გამოიყენება წყვეტა ტაიმერიდან თანხვედრის დროს. შუქდიოდების გადართვას ანხორციელებს წყვეტის დამმუშავებელი ფუნქცია, რომელიც აღწერილია პროგრამის დასაწყისში 3-20 სტრიქონებში. ფუნქციის შესრულების დროს მოწმდება ცვლადი oldstate, თუ მასში ჩაწერილია ნული, PORTA, PORTB, PORTC, PORTD რეგისტრებში იწერება ნულები, რაც იწვევს შუქდიოდების ჩაქრობას. წინააღმდეგ შემთხვევაში რეგისტრებში იწერებიან ერთები და შუქდიოდები ინთება. შუქდიოდების გადართვის დროს olstate ცვლადის მნიშვნელობა იცვლება.

წყვეტის ფუნქციას დავარქვით timer1_com_isr. აღწერის სათაურში არის ჩაწერილი სამართავი სიტყვა Interrapt, რომელიც მიუთითებს კომპილატორს, რომ მოცემული ფუნქცია არის წყვეტის დამუშავების პროცედურა. ამ სიტყვას კვადრატულ ფრჩხილებში მოსდევს წყვეტის ტიპის მითითება **[TIMER1_COM]**, რომლის საფუძველზე კომპილიატორი ათავსებს წყვეტის ვექტორს პროგრამის მეხსიერებაში და აფიქსირებს მის მისამართს.

მთავარი ფუნქციის დასაწყისში სრულდება მიკროკონტროლერის ინიციალიზაცია. ვინაიდან შუქდიოდების ჩაქრობა/ანთების მართვა ხორციელდება მიკროკონტროლერის A,B,C,D პორტების გამომყვანებიდან შესაბამისი კოდების გაცემით, ხდება აღნიშნული პორტების გაწყობა გამოყვანაზე DDR რეგისტრებში 0xFF კოდების ჩაწერით (23-26 სტრიქონი). მთვლელი რეგისტრი TCNT1 ყენდება საწყის მდგომარეობაში (რადგან აღნიშნული მთვლელი 16 თანრიგიანია, იგი შესდგება ორი ბაიტიანი შეყვანა - გამოყვანის რეგისტრისაგან – TCNT1H, TCNT1L, რომლებშიც იწერებიან კოდები 0x00) (სტრიქონი 27,28).

როგორც იყო აღნიშნული, განხილულ ამოცანაში შუქდიოდების გადართვას შორის დაყოვნებისათვის გამოყენებულია წყვეტა ტაიმერიდან განივ – იმპულსურ მოდულიაციის CTC (მთვლელის განულება თანხვედრის შემთხვევაში) რეჟიმში. ამ რეჟიმში ტაიმერი ფუნქციონირებს როგორც ჩვეულებრივი ამჯამავი, რომლის ინკრიმენტი ხორციელდება ყოველ სატაქტო იმპულსზე. ტაიმერის მაქსიმალური მნიშვნელობა განისაზღვრება

შედარების OCR1A რეგისტრში წინასწარ ჩაწერილი დაყოფის კოეფიციენტის მნიშვნელობით, რომლის მიღწევის შემდეგ ტაიმერი აგრძელებს თვლას მისი საწყისი მნიშვნელობიდან. მთვლელის ყოველი გადასვლისას საწყის მდგომარეობაში ფორმირდება წყვეტის მოთხოვნა. მიკროკონტროლერი გადაირთვება წყვეტის სამომსახურო პროგრამაზე, რომელიც ასრულებს შუქდიოდების მდგომარეობის ცვლილებას. გადართვებს შორის შუალედის (დაყოვნების) დრო განისაზღვრება შედარების რეგისტრში ჩაწერილი დაყოფის კოეფიციენტისა და მთვლელის სიხშირის წინგამყოფის დაყოფის კოეფიციენტის მნიშვნელობით და ჩვენს მიერ მიღებულია 1000მწ-ის ტოლად. წინა პარაგრაფში მოყვანილი მსჯელობის საფუძველზე დაყოვნების ასეთი მნიშვნელობა მიიღება მთვლელის საყრდენი სიხშირის (4მჰც) გაყოფით 1024 კოეფიციენტზე და მთვლელის თვლის მოდულად 4000-ის არჩევით. აღნიშნული პარამეტრების დაყენება ხდება მთვლელის მართვის რეგისტრებში TCCR1A, TCCR1B. პირველი რეგისტრის უმცროს ორ თანრიგში ჩაწერილია CTC განივ-იმპულსური მოდულიაციის რეჟიმის კოდი (10), ხოლო მეორე რეგისტრის სამ უმცროს თანრიგში საყრდენი სიხშირის დაყოფის კოეფიციენტის კოდი 101 (რაც შეესაბამება 1024-ს) (29, 30 სტრიქონი). მომდევნო 31,32 სტრიქონზე სრულდება შედარების OCR1A რეგისტრში მთვლელის თვლის მოდულის ჩაწერა (ვინაიდან აღნიშნული რეგისტრი 16 თანრიგიანია იგი შედგენილია ორი ბაიტიანი რეგისტრისაგან OCR1AH, OCR1AL). ჩაწერილი კოდი შეესაბამება 4000-ს. TIMASK ნიღბის რეგისტრის თანრიგში, რომელიც შეესაბამება წყვეტის მოთხოვნას, თანხვედრის შემთხვევაში ვწერთ ერთს, რითაც ნებას ვრთავთ აღნიშნული ტიპის წყვეტას (სტრიქონი 33). 34 სტრიქონზე სრულდება ფუნქცია #asm() - რომლის საშუალებით ხდება წყვეტის გლობალური ნების დართვა. #asm () არის სპეციალური სისტემური ფუნქცია, რომელიც შესაძლებლობას იძლევა C პროგრამაში შევასრულოთ ასემბლერის ბრძანებები. ფუნქციის პარამეტრი არის სტრიქონული ცვლადი ან სტრიქონული კონსტანტა, რომლის მნიშვნელობას წარმოადგენს ბრძანების ტექსტი ასემბლერზე. მოცემულ პროგრამაში #asm () ფუნქციის საშუალებით სრულდება წყვეტის შესრულების ასემბლერის ბრძანება sei.

main ფუნქციის ცენტრალურ პროცედურას წარმოადგენს უსასრულო ციკლი, რომელიც სრულდება While ოპერატორის მიერ. ციკლის ხანგრძლიობა განისაზღვრება მთვლელის თვლის ხანგრძლიობით განივ-იმპულსურ მოდულიაციის რეჟიმში და შეადგენს 1000მწ-ს. მთვლელის თვლის ყოველი ციკლის ბოლოს ფორმირდება წყვეტის მოთხოვნის სიგნალი, რომელიც აიძულებს მიკროკონტროლერს გადაერთოს წყვეტის სამომსახურებო პროგრამაზე. აღნიშნული პროგრამა იწვევს შუქდიოდების გადართვას საპირისპირო მდგომარეობაში: ჩამქრალიდან - ანთებულში ან პირიქით. წყვეტის პროგრამის შესრულების შემდეგ (შუქდიოდების გადართვის შემდეგ) პროგრამა იმეორებს უსასრულო ციკლს. ამგვარად ხორციელდება დაყოვნება შუქდიოდების გადართვებს შორის.

3.7. ტექსტური დისპლეის მართვის მოწყობილობა

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

დავაგეგმაროთ მოწყობილობა, რომელსაც გამოყავს პროგრამაში ჩაწერილი ტექსტი ტექსტურ დისპლეიზე. ეკრანზე ტექსტი გადაადგილდეს ჯერ მარჯვნივ, შემდეგ - მარცხნივ.

პრინციპიალური ელექტრული სქემა

ვირჩევთ LCD 16x2 ტექსტურ დისპლეის (DS18B20). მოცემულ დისპლეის ტექსტი ორ სტრიქონში გამოყავს, თვითეულ სტრიქონში 16 სიმბოლო (სურ.3.3.). სქემას აქვს 16 გამომყვანი და მოთავსებულია ერთ კორპუსში. აქედან 8 (D0-D7) გამომყვანი განკუთვნილია მონაცემების და ბრძანებების გადაცემისათვის, 3 (EN, R/W, RS) - მმართველი სიგნალებისთვის, დანარჩენი შესასვლელები გამოიყენებიან კვების (Vcc), მიწის (GND) და სიკაშკაშის მარეგულირებელი ძაბვის მიერთებისთვის.

სურ.3.3

8 მონაცემების გამომყვანი შეიძლება გამოყენებული იქნას მონაცემების და ბრძანებების 4- ან 8- ბიტიან გადაცემისათვის. ჩვენს შემთხვევაში გადაცემა სრულდება 4-ბიტად, რაც იძლევა იმის საშუალებას, რომ მკ-ის ერთი 8 თანრიგიანი პორტი გამოვიყენოთ მონაცემების ან ბრძანებების გადაცემისათვის მმართველ სიგნალებთან ერთად. ამასთან დაკავშირებით, ვინაიდან დისპლეის D0-D3 გამომყვანები არ გამოიყენებიან, ამ გამომყვანებს მიწას ვაწოდებთ.

(EN)- "ნების დართვის" გამომყვანი საშუალებას იძლევა, რომ მოხდეს მიმართვა დისპლეის შიდა მონაცემის რეგისტრთან, თუ გამომყვანზე ერთიანს მივაწოდებთ. ნულიანის მიწოდების შემთხვევაში - დისპლეი გაითიშება.

(R/W)- "ჩაწერა/ამოკითხვის" გამომყვანის საშუალებით განისაზღვრება გადაცმის მიმართულება: როდესაც მასზე ნულია, სრულდება ჩაწერა დისპლეიში, ერთიანის დროს - ამოკითხვა. ვინაიდან ჩვენს შემთხვევაში შესრულდება მხოლოდ დისპლეიში ჩაწერა, შესასვლელს მიეწოდება ნულიანი (ანუ "მიწა").

(RS)- "რეგისტრის ამორჩევის" გამომყვანი განსაზღვრავს მონაცემთა სალტით გადაცემული ინფორმაციის ხასიათს: ბრძანებაა თუ მონაცემი.

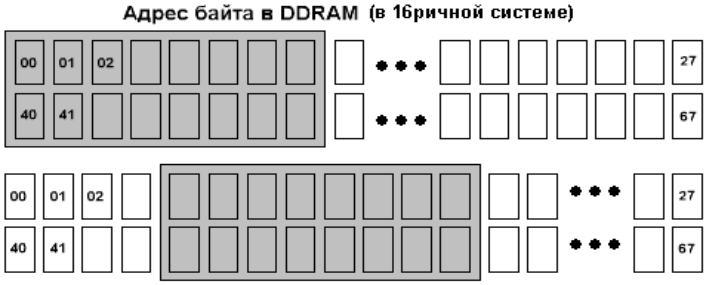
დისპლეის მართვისათვის საჭიროა დისპლეის შესასვლელები მიუერთოთ მკ-ის PC პორტის გამოსასვლელებს. PC2 გამოსასვლელი RS შესასვლელს, PC3 _ EN-ს, PC4 - D4-ს, PC5 - D5-ს, PC6 - D6-ს , PC7 – D7.

LCD HD44780 კონტროლერის ლოგიკური სტრუქტურა

დისპლეის მართვას ანხორციელებს კონტროლერი, რომელიც ახდენს ბრძანებების დამუშავებას და შეიცავს სამი სახის მეხსიერებას:

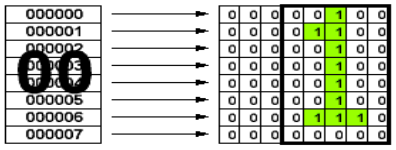
DDRAM – დისპლეის მეხსიერება. ის რაც ჩაწერილია DDRAM მეხსიერებაში, მისი გამოტანა ხდება ეკრანზე. ეკრანზე სიმბოლოების გამოყვანა ხდება "ნიშნის ადგილებში", რომლებიც წარმოადგენენ სტრიქონისა და სვეტებისგან შექმნილ მატრიცებს . მათ გადაკვეთაზე მყოფი წერტილების მდგომარეობა (ჩამქრალი, თუ განათებული) გამოსახავენ ეკრანზე სიმბოლოს. თვითოეული მატრიცის ზომა არის 5X7 (5 სვეტი, 7 სტრიქონი), რაც

განსაზღვრავს ეკრანზე გამოყვანილი სიმბოლოს ფორმატს. ეკრანზე ნიშნის ადგილები ორ სტრიქონად არიან განლაგებული, თითო სტრიქონში - 16 ადგილით. DDRAM-ში ეკრანზე გამოსაყვანი სიმბოლოები წარმოდგენილი არიან ასევე მატრცების სახით, მაგრამ DDRAM მეხსიერების ტევადობა გაცილებით მეტია ვიდრე ეკრანის ხილული არე. როგორც წესი DDRAM – ს აქვს 80 უჯრედი – 40 პირველი სტრიქონისთვის და 40 მეორე სტრიქონისთვის, დისპლეიმ შეიძლება იმოძრაოს ამ სახაზავზე, როგორც ფანჯარამ და გამოანათოს მხოლოდ ხილული არე (სურ.3.4). დისპლეის გადაადგილებისთვის არის სპეციალური ბრძანება . ასევე არსებობს კურსორის ცნება – ეს ის ადგილია, სადაც უნდა ჩაიწეროს მომდევნო სიმბოლო, ე.ი. მისამართის მთვლელის მიმდინარე მნიშვნელობა. კურსორი სავალდებულო არ არის იყოს ეკრანზე , ის შეიძლება განთავსებული იყოს ეკრანის მიღმა ან საერთოდ გამორთული.



ნახ. 3.4

CGROM – სიმბოლოების ცხრილია, რომელშიც მოთავსებულია სიმბოლოები მატრიცების სახით. როდესაც DDRAM უჯრედში ვწერთ სიმბოლოს, იგი გადაიწერება CGROM ცხრილიდან, რომელიც იხატება ეკრანზე. CGROM –ის ცვლილება არ შეიძლება .
CGRAM - ასევე სიმბოლოების ცხრილია, მაგრამ იმ განსხვავებით რომ მასში შეგვიძლია შევიტანოთ ცვლილებები, საკუთარი სიმბოლოების შექმნის შემთხვევაში. დამისამართება ხორციელდება წრფივად, ე.ი. სტრიქონებში წარმოდგენილია ერთი სიმბოლოს 8 ბაიტი ქვევიდან ზევით დამისამართებით - ერთ ბიტს ეკრანზე შეესაბამება ერთი წერტილი. მეორე სიმბოლოსთვისაც ანალოგიურად. სურ.3.5-ზე ნაჩვენებია სიმბოლოს წარმოდგენა მატრიცაში.



სურ. 3.5

დისპლეის კონტროლერებში სხვადასხვა ოპერაციების შესრულებისათვის გამოიყენება სპეციალური ბრძანებები. 3.1 ცხრილში მოცემულია ბრძანებების შესაბამისი კოდები:

ცხრილი 3.1

0	0	0	0	0	0	0		ეკრანის გასუფთავება. DDRAM მისამართების მთვლელი ნულოვან პოზიციაზეა.
0	0	0	0	0	0	1	–	DDRAM- თან მიმართვა. ძვრების განულება. სამისამართო მთვლელში ნულის ჩაწერა
0	0	0	0	0	1	I/D	S	ეკრანის ძვრის და კურსორის გაწყობა
	0	0	0	1	D	C	B	ასახვის რეჟიმის გაწყობა.
0	0	0	1	S/C	R/L	–	1	კურსორის ან ეკრანის ძვრა,
0	0	1	DL	N	FF	–	–	ხაზების რაოდენობის, (სალტის სიგანის) და სიმბოლოების ზომის ამორჩევა.
0	1	AG	AG	AG	AG	AG	A-G	მისამართის გადართვა SGRAM–ზე და SGRAM-ის მისამართის დაფორმირება
	AD	AD	AD	AD	AD	AD	AD	გადაირთოს მისამართი DDRAM-ზე და DDRAM-ს მისამართის დაფორმირება

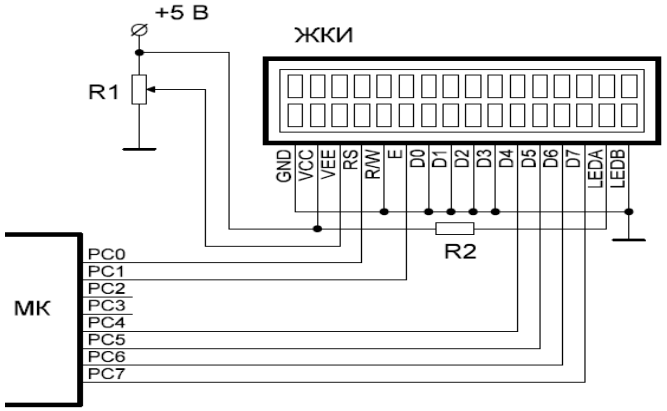
ბრძანების პარამეტრებს აქვს შემდეგი მნიშვნელობები:

- I/D – განსაზღვრავს სრულდება მისამართის მთვლელის ზრდა (ინკრიმენტი), თუ კლება (დეკრემენტი). თუ მნიშვნელობა ნულია – დეკრემენტი. ე.ი. ყოველი მომდევნო ბაიტი ჩაიწერება n-1 უჯრედში. თუ 1 – შესრულდება ინკრიმენტი;
- S – ეკრანის ძვრა, თუ ჩავწერთ 1-ს, მაშინ ყოველი ახალი სიმბოლოს შეტანის შემთხვევაში მოხდება ეკრანის ძვრა, მანამ არ მიაღწევს DDRAM-ის დასასრულს;
- D – დისპლეის ჩართვა. თუ მასში ჩავწერთ 0-ს, მაშინ გამოსახულება გაქრება. იმისათვის, რომ ნახატი გამოჩდეს ამ პოზიციაში საჭიროა ჩაიწეროს 1;
- C – ჩაირთოს კურსორი წახაზული სიმბოლოს სახით. როდესაც 1–ია კურსორი ჩაირთვება;

- B – გამოისახოს კურსორი ეკრანზე როგორც მოციმციმე შავი კვადრატი;
- S/C - კურსორის ან ეკრანის ძვრა. თუ 0 – ია, მაშინ იძვრის კურსორი. თუ 1, მაშინ ეკრანი. ყოველი ბრძანების შესრულების დროს ერთი პოზიციით;
- R/L – განსაზღვრავს კურსორის და ეკრანის ძვრის მიმართულებას. 0- მარცხნივ, 1- მარჯვნივ;
- D/L – ბიტი რომელიც განსაზღვრავს მონაცემის სალტის გამოყენებული თანრიგების რაოდენობას. 1 - 8 ბიტია, 0 - 4 ბიტია;

- N – ეკრანზე სტრიქონების რაოდენობა. 0 ერთ სტრიქონი, 1 - ორი სტრიქონი;
- F - სიმბოლოს ზომა: 0 – 5X8 წერტილი. 1 - 5X10 წერტილი (იშვიათად გვხდება);
- AG – მისამართი CGRAM მეხსიერებაში;
- AD- მისამართი DDRAM მეხსიერებაში.

ქვემოთ ნაჩვენებია LCD-ს (თხევად კრისტალური ინდიკატორი) მიერთება მიკროკონტროლერთან (სურ.3.6).



სურ 3.6

დისპლეის მართვისათვის მიუერთოთ დისპლეის შესასვლელები PC პორტის შესასვლელებს: PC2 - RC, PC3 –EN, PC4 – D4, PC5 –D5, PC6 –D6, PC7 –D7.

ალგორითმი

საწყისი გაწყობის ოპერაციები

1. ვასრულებთ მოწყობილობის საწყის გაწყობას: ვანხორციელებთ მიკროკონტროლერის პორტის გამოსასვლელებისა და დისპლეის შესასვლელების მიერთებას.
2. შეგვაქვს პროგრამაში გამოსაყვანი ტექსტი.

ოპერაციები რომელიც ჰქმნის პროგრამის ტანს

1. გამოგვყავს ტექსტი დისპლეიზე;
2. ვძრავთ ტექსტს დისპლეიზე მარჯვნივ;
3. ვძრავთ ტექსტს დისპლეიზე მარცხნივ;
4. გადავდივართ ციკლის დასაწყისში.

პროგრამა C ენაზე

// დისპლეის გამოსასვლელების მიერთება C პორტის გამოსასვლელებთან

```
1.  sbit LCD_RS  at  PORTC2_bit;
2.  sbit  LCD_EN  at  PORTC3_bit;
3.  sbit  LCD_D4  at  PORTC4_bit;
4.  sbit  LCD_D5  at  PORTC5_bit;
5.  sbit  LCD_D6  at  PORTC6_bit;
6.  sbit  LCD_D7  at  PORTC7_bit;
```

// გადაცემის მიმართულების დაყენება

```
7.  sbit  LCD  RS Direction at  DDRC2  bit;
8.   sbit  LCD  EN Direction at  DDRC3 bit;
9.   sbit  LCD  D4 Direction at  DDRC4  bit;
10.  sbit  LCD  D5 Direction at  DDRC5  bit;
11.  sbit  LCD  D6 Direction at  DDRC6  bit;
12.  sbit  LCD  D7 Direction at  DDRC7  bit;
```

// ტექსტის შეტანა

```
13. char  txt1[]=”MikroElektronika”;
14. char txt2[]=”BigAVR6”;
15. char  txt3[]=”LCD4bit”;
16. char  txt4[]=” Example”;
17. char  i;
18. void  Move_Delay ( ) {
19. Delay_ ms (500);
20. }
21. void  main  ( )  {
22. Lcd_ Init  ();  // DS1820 მოდულის ინიციალიზაცია
23. Lcd_ Cmd( LCD_CLEAR);  // ეკრანის გასუფთავება
24. Lcd_ Cmd (LCD_CURSOR_OFF); // კურსორის გამორთვა
```

// ტექსტის ეკრანზე გამოტანა

```
25.  Lcd_Out  (1,6,txt3);
26.  Lcd_Out  (2,6,txt4);
27.  Delay_ ms(2000);
28.  Lcd_ Cmd( LCD_CLEAR);
29.  Lcd_Out  (1,6,txt1);
30.  Lcd_Out  (2,6,txt2);
31.  Delay_ ms(2000);
```

//ტექსტის გადაადგილება

```
32.  For  (i=0, I<4, i++) {
33.  Lcd_Cmd (LCD_SHIFT_RIGHT);  // ტექსტის მარჯვნივ დაძვრა
34.  Move_Delay ( );                  // ტექსტის დაყოვნება დისპლეიზე
```

```
35.  }
36.  while  (1) {
37.  for  (i=0, I<7, i++) {
38.  Lcd_Cmd (LCD_SHIFT_LEFT);    // ტექსტის დაძვრა მარცხნივ
39.  Move_Delay ( );                    // დაყოვნება
40.  }
41.  for  (i=0, I<7, i++) {
42.  Lcd_Cmd (LCD_SHIFT_RIGHT);  // ტექსტის მარჯვნივ დაძვრა
43.  Move_Delay ( );                    // ტექსტის დაყოვნება დისპლეიზე
44.  }
45.  }
46.  }
```

პროგრამების შექმნის დროს მოსახერხებელია გამოყენებული იყოს მზა ფუნქციები მკ-ის სხვადასხვა მოწყობილობებს შორის ურთიერთქმედებისთვის, რომლებსაც გვთავაზობენ კომპილატორის დამმუშავებლები. ეს ფუნქციები თავიდან გვაცილებენ მათი აღწერის აუცილებლობას. ამ უკანასკნელიდან გამომდინარე მარტივდება პროგრამის შედგენის პროცესი. ამ ფუნქციებიდან ზოგიერთს ვიყენებთ ჩვენს პროგრამაში.

პროგრამის დასაწყისში ხორციელდება კავშირის დამყარება მკ-ის გამომყვანებსა და დისპლეის შესასვლელებს შორის. (1-6 სტრიქონები). მაგალითად, sbit LCD_RS at PORTC2-bit ფუნქცია ამყარებს კავშირს RS LCD გამომყვანებსა და C პორტის გამომყვან 2-ს შორის.

ფუნქციების შემდეგი ჯგუფი (7-12 სტრიქონები) განსაზღვრავენ გადაცემის მიმართულებას. მაგალითად, sbit LCD_EN Direction at DDRC3-bit ფუნქცია განსაზღვრავს C პორტის 3 არხს, როგორც გამოსასვლელს, რომელიც მიერთებულია EN დისპლეის შესასვლელთან.

13-16 სტრიქონებზე გამოცხადებულია სიმბოლოების txt მასივები, როგორც ბაიტური მთელები, ტექსტური მნიშვნელობების შეტანით. მათ შემდგომში გამოვიყვანთ დისპლეის ეკრანზე ტექსტის გამოყვანის დროს.

შემდეგ განსაზღვრულია დაყოვნების ფუნქცია (სტრიქონი 19). ეს ფუნქცია იყენებს სტანდარტულ ფუნქციას Delay_ms(), რომელიც უზრუნველყოფს 500 მწ დაყოვნებას. იგი გამოყენებული იქნება პროგრამაში.

20 სტრიქონიდან იწყება ტექსტის გამოყვანის მთავარი ფუნქცია main. პირველ რიგში სრულდება დისპლეის ინიციალიზაცია. ამისთვის ვიყენებთ კომპილატორის ბიბლიოთეკის ფუნქციას Lcd_ Init (). ეს ფუნქცია აყენებს დისპლეის პარამეტრების საწყის მნიშვნელობებს (სტრიქონი 21).

სტრიქონ 22-ზე ხდება ეკრანის გასუფთავება Lcd_Cmd(LCD_CLEAR) ფუნქციის საშუალებით. 23-ე სტრიქონზე გამოირთვება კურსორი Lcd_Cmd(LCD_CURSOR_OFF) ფუნქციის გამოყენებით.

24-ე და 25-ე სტრიქონებზე სრულდება ტექსტის გამოყვანა ეკრანზე. გამოყენებულია ფუნქცია Lcd_Out (სტრიქონი, სვეტი, გამოსაყვანი ტექსტი). ამ ფუნქციას გამოყავს ტექსტი ეკრანზე ფრჩხილებში პარამეტრებით მითითებული პოზიციიდან დაწყებული - პირველად ნაჩვენებია სტრიქონი, მეორედ - სვეტი, მესამედ - ტექსტის შემცველი მასივი). პირველ ფუნქციას გამოჰყავს txt3 მასივში ჩაწერილი ტექსტი ეკრანის პირველი სტრიქონის მე-6

პოზიციიდან დაწყებული, მეორე ფუნქციას – txt4 მასივში ჩაწერილი ტექსტი ეკრანის მეორე სტრიქონის მე-6 პოზიციიდან.

შემდეგ სრულდება დაყოვნება 2მწ, რომ შესაძლებელი იყოს ტექსტის წაკითხვა ეკრანზე (სტრიქონი 26-ე. თუ 27-ე სტრიქონზე ეკრანს კვლავ ვასუფთავებთ და შემდეგ 28, 29 სტრიქონებში გამოიყვანება txt1 და txt2-ის მასივების შემცველობები ეკრანის პირველ და მეორე სტრიქონებში, შესაბამისად. ისევ სრულდება დაყოვნება (სტრ. 30).

შემდეგ სრულდება ტექსტის ძვრა მარჯვნივ. ციკლში ვასრულებთ ფუნქციას Lcd_Cmd (LCD_SHIFT_RIGHT), რომელიც ციკლის ერთ გასვლაზე ძრავს ტექსტს ერთი პოზიციით მარჯვნივ (სტრ. 32). ციკლის დამთავრების შემდეგ ტექსტი დაძრული იქნება ოთხი პოზიციით მარჯვნივ.

34-ე სტრიქონიდან დაწყებული, სრულდება უსასრულო ციკლი while, რომლის შესრულების დროს ტექსტი მიმდევრობით იძვრის ეკრანზე ჯერ მარცხნივ ოთხი პოზიციით Lcd_Cmd (LCD_SHIFT_LEFT) ფუნქციის გამოყენებით (სტრ. 35-37), შემდეგ მარჯვნივ ოთხი პოზიციით, ზემოთ ნაჩვენები ფუნქციის საშუალებით (სტრ. 38-40). ყოველი მიმართულებით ძვრის დამთავრების შემდეგ ვასრულებთ დაყოვნებას და კვლავ გადავდივართ while ციკლის დასაწყისზე.

3.8 მიკროკონტროლერის მუშაობა ტემპერატურის გადამწოდთან (სენსორთან)

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

ავაგოთ მოწყობილობა, რომელიც გარდასახავს ტემპერატურის გადამწოდიდან მიღებულ ანალოგურ სიგნალებს ოროობით კოდში და გამოჰყავს მიღებული მნიშვნელობები ტექსტურ დისპლეიზე.

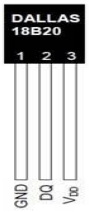
პრინციპური ელექტრული სქემა:

მოწყობილობის აგების დროს ვიყენებთ Atmega 128 მიკროკონტროლერს, DS18B20 ტემპერატურის გადამწოდს და 2x16 LCD ტექსტურ დისპლეის, რომელიც ჩვენ მიერ წინა პარაგრაფში იყო განხილული.

ტემპერატურის გადამწოდი (სენსორი)

მაკომპლექტებელი ელემენტების თანამედროვე ბაზარი გვთავაზობს ტემპერატურის გადამწოდების ფართო ასორტიმენტს. მათ შორის ძირითადი განსხვავება მდგომარეობს გასაზომი ტემპერატურის დიაპაზონში, კვების ძაბვის მნიშვნელობაში, გამოყენების სფეროში, გაბარიტულ ზომებში, ტემპერატურის გარდასახვის წესში, მართვის სისტემასთან დაკავშირების ინტერფეისში. ამჟამად ერთ-ერთი ყველაზე პოპულარული ტემპერატურის გადამწოდი არის Dallas Semicondactor ფირმის გადამწოდი DS18B20.
DS18B20 წარმოადგენს ციფრულ გადამწოდს გარდასახვის პროგრამირებადი გარჩევითობით.

გადამწოდს აქვს შემდეგი სახე (სურ.3.7):



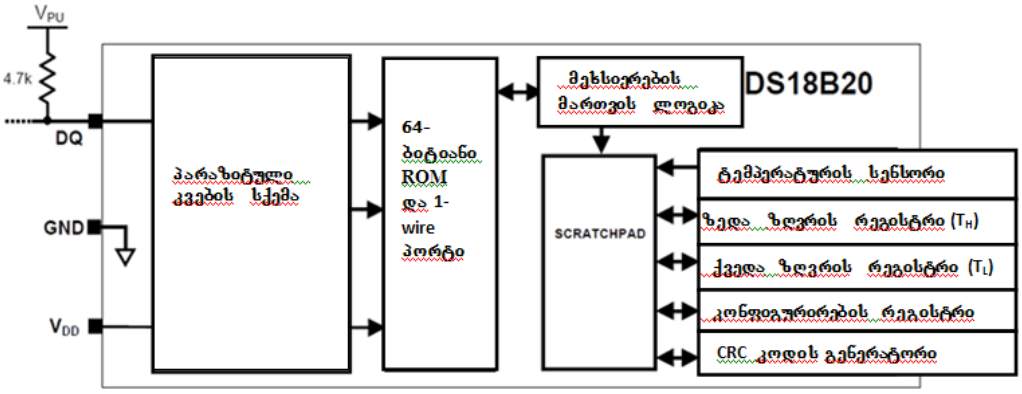
სურ. 3.7

Vcc - კვების ძაბვის შესასვლელი;
GND - მიწა;
DQ - მონაცემთა შესასვლელი.

გადამწოდს აქვს შემდეგი თავისებურებები:

1) მმართველ სისტემასთან ურთიერთობისათვის გამოიყენებს 1-wire ინტერფეისის მონაცემთა სალტეს;
2) უნიკალური 64-ბაიტიანი საიდენტიფიკაციო კოდის გამოყენება, რომელიც განთავსებულია შიგა ROM-მეხსიერებაში და გათვალისწინებულია მრავალწერტილოვანი სისტემებისათვის, სადაც საჭირო არის კონკრეტული გადამწოდების დამისამართება;
3) გასაზომი ტემპერატურის დიაპაზონი შეადგენს -55⁰….+125⁰ C;
4) გაზომვის სიზუსტე ±0,5ვ, თუმცა ეს მართებულია მხოლოდ დიაპაზონისათვის -10⁰• • • +85⁰C;
5) გარდასახვის გარჩევითობა განისაზღვრება მომხმარებლის მიერ და შეადგენს 9-დან 12-მდე ბიტს;
6) აქვს ქვედა და ზედა ზღვარის შიგა რეგისტრები, რომლთა საშუალებით გამომუშავდება განგაშის სიგნალები გაზომილი ტემპერატურის ზღვრის გადასვლის შემთხვევაში.

განვიხილოთ გადამწოდის სტრუქტურა (სურ. 3.8):



სურ. 3.8

მონაცემთა გადაცემა ხორციელდება DQ გამომყვანის საშუალებით, რომელიც ქმნის 1-wire ინტერფეისს. ,,პარაზიტული კვების" სქემა გამოიყენება ინტერფეისიდან კვების მიწოდების შემთხვევაში. ვინაიდან მოცემულ მაგალითში გამოიყენება გარე კვება, ჩვენ მას აქ არ განვიხილავთ.

"64-ბიტიანი ROM და 1-wire პორტი" შეიცავს უნიკალურ 64 ბიტიან საიდენტიფიკაციო კოდს, რომელიც განთავსებულია ROM მეხსიერებაში. ამ კვანძში ასევე იმყოფება 1-wire მართვის სისტემასთან ურთიერთობის ინტერფეისი.

,,მეხსიერების მართვის ლოგიკ"-ის ქვესისტემა ახორციელებს მონაცემთა გადაცემას…1-wire ინტერფეისსა და Scratchpad მეხსიერებას შორის. რომელსაც თავის მხრივ აქვს წვდომა ტემპერატურის გადამწოდის რეგისტრებთან, განგაშის სიგნალის ზემო და ქვემო ზღვრების დაყენების რეგისტრებთან, საკონფიგურაციო რეგისტრთან და 8-ბიტიანი საკონტროლო ჯამის გენერატორის რეგისტრთან.

კვებასთან მიერთების შემთხვევაში, უთქმელობით, გადამწოდს აქვს გარდასახვის გარჩევითობა 9 ბიტი და გადადის შემცირებული ენერგომოხმარების რეჟიმში (,,ძილის" მდგომარეობაში). გარდასახვის დაწყებისათვის წამყვანმა მოწყობილობამ უნდა გაუგზავნოს ბრძანება Convert_T. ტემპერატურის ოროობით კოდში გარდასახვის შემდეგ ეს კოდი ჩაიწერება Scratchpad მეხსიერებაში ორბაიტიანი სიტყვის სახით (ორ უჯრედში) და გარდამსახი კვლავ გადადის ,,ძილის" რეჟიმში.

ტემპერატურის გადამწოდი შეიცავს აცგ-ს (ანალოგურ ციფრულ გადამსახს). მის გამოსასვლელი რეგისტრიდან გარდასახვის შედეგად მიღებული კოდი ჩაიწერება Scratchpad მეხსიერებაში, რომელსაც აქვს შემდეგი სახე (სურ.3.9):

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LS Byte	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit 15	**bit 14**	**bit 13**	**bit 12**	**bit 11**	**bit 10**	**bit 9**	**bit 8**
MS Byte	S	S	S	S	S	2^6	2^5	2^4

S წარმოადგენს ნიშნის ალამს, გამოიყენება ნიშნის მითითებისათვის (S=0, 0-10 თანრიგების შემცველი რიცხვი დადებითია, და S=1, თუ იმავე თანრიგებში ჩაწერილი რიცხვი უარყოფითია. (უარყოფითი რიცხვი წარმოდგენილია დამატებით კოდში).

როგორც მოყვანილი ნახატიდან ჩანს, უმცროს ოთხ ბიტში იწერება კოდის წილადი ნაწილი (ბიტები 0-3), ხოლო უფროს რვა ბიტში - მთელი ნაწილი (ბიტები 4-101).

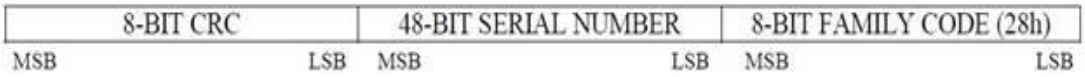
განგაშის სიგნალი - თერმოსტატის ფუნქცია

ამ მიზნით გათვალისწინებულია 2 8-ბიტიანი რეგისტრი TH და TL. TH შეიცავს ტემპერატურის ზედა ზღვარის მნიშვნელობას, TL-ქვედა ზღვარის მნიშვნელობას. თუ ტემპერატურა TH მნიშვნელობაზე მეტია ან TL მნიშვნელობაზე ნაკლებია ყენდება განგაშის ალამი. ამ ალმის აღმოჩენა ხდება წამყვანი მოწყობილობის მიერ Alarm Search ბრძანების გაცემით DQ ხაზზე. განგაშის ალამი ახლდება ტემპერატურის ყოველი გარდასახვის ოპერაციის შემდეგ. ამასთან, TH და TL რეგისტრებთან შედარებისათვის გამოიყენება მხოლოდ

ბიტები 4-10, აქედან გამომდინარე, თერმოსტატის ფუნქცია მუშაობს მხოლოდ მთელი ნაწილისათვის. რეგისტრები ფიზიკურად წარმოადგენენ EEPROM მეხსიერებას, ამიტომ ისინი ინარჩუნებენ თავის მნიშვნელობას კვების გამორთვის შემთხვევაშიც. თვით რეგისტრები ტემპერატურის რეგისტრების ანალოგიურია, მხოლოდ 8-ბიტიანი, S ალამს აქვს იგივე მნიშვნელობა რაც წინა შემთხვევაში.

64-ბიტიანი საიდენტიფიკაციო კოდი

ეს კოდი, როგორც ადრე იყო აღნიშნული, საჭიროა თითოეული მოწყობილობის იდენტიფიკაციისათვის მრავალწერტილიან ტემპერატურის გაზომვის სისტემაში.
ამ მეხსიერებას აქვს ქვევით ნაჩვენები ფორმატი (სურ.3.10):



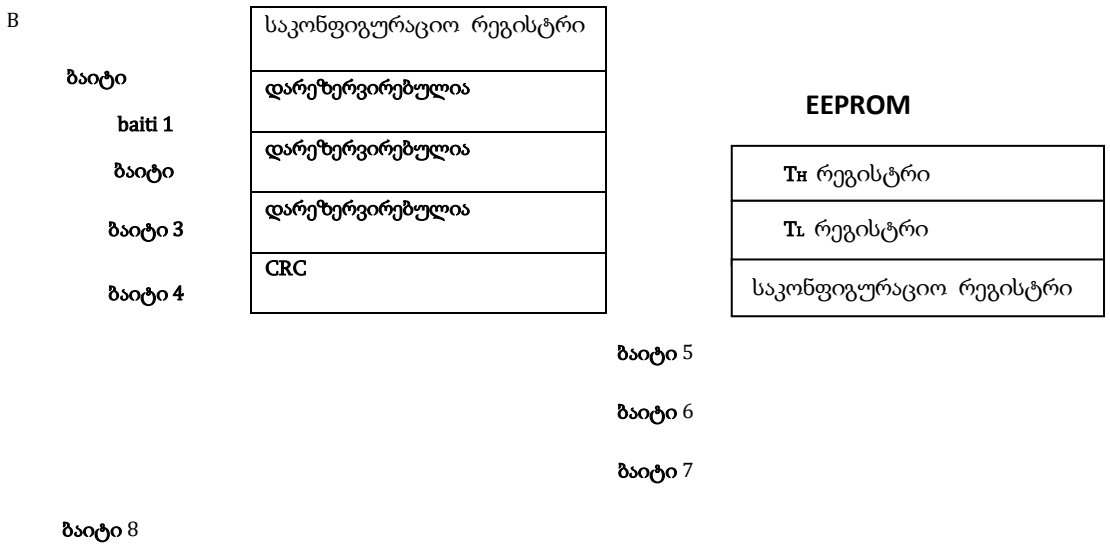
სურ. 3.10

მეხსიერების ორგანიზაცია

უმცროსი 8-ბიტი განკუთვნილია ოჯახის აღნიშვნისათვის (FAMILI CODE) და შეიცავს მნიშვნელობას 0x28 (მწარმოებლის მიერ დანიშნული). 48 ბიტი შეიცავს მოწყობილობის უნიკალურ სერიულ ნომერს (SERIAL NAMBER). უფროსი ბაიტი შეიცავს CRC საკონტროლო კოდს, რომლის საშუალებით ხდება მონაცემთა სისწორის შემოწმება ROM მეხსიერებაში.

გადამწოდის მეხსიერება შედგება ბლოკნოტური ტიპის მეხსიერებისა (SCRATCHPAD) და EEPROM მეხსიერებისაგან (სურ. 3.11), რომელიც განკუთვნილია საკონფიგურაციო რეგისტრის მონაცემთა და განგაშის სიგნალის ზედა და ქვედა ზღვრების რეგისტრების შემცველობების შესანახად. კვების ძაბვის გამორთვის შემთხვევაში 2,3,4 ბაიტების მონაცემები ინახება EEPROM-ში. 0,1 ბაიტებში იწერება ტემპერატურის გარდასახვის შედეგად მიღებული ორობითი კოდის მნიშვნელობა. 5,6,7, ბაიტები დარეზერვირებულია შიგა გამოყენებისათვის და მომხმარებლისათვის ისინი მიუწვდომელია.

SCRATCHPAD



სურ 3.11

უნდა აღინიშნოს, რომ თუ თერმოსტატის ფუნქცია არ გამოიყენება, TH და TL შეიძლება გამოყენებული იყოს, როგორც საერთო დანიშნულების მეხსიერება.

მონაცემები იწერება 2,3,4 ბაიტებში მიმდევრობით, დაწყებული მე-2 ბაიტის უმცროსი ბიტიდან Write Scratchpad ბრძანების საშუალებით. მონაცემთა წაკითხვა კი შეიძლება შესრულდეს Read Scratchpad ბრძანებით, რომლის შესრულების დროს წამყვანი მოწყობილობა იღებს მონაცემებს 0 ბაიტის უმცროსი ბიტიდან დაწყებული.

მონაცემთა შესანახად 2,3,4 ბაიტებიდან EEPROM-ში, წამყვან მოწყობილობამ გადამწოდს უნდა გაუგზავნოს ბრძანება Copy Scratchpad.

როგორც ზევით იყო ნათქვამი, EEPROM მეხსიერებაში ჩაწერილი მონაცემები კვების გამორთვის შემთხვევაში შენარჩუნდება. კვების ჩართვის შემთხვევაში კი შესაბამის EEPROM უჯრედებიდან გადაიტვირთება Scratchpad მეხსიერების რეგისტრებში. გარდა ამისა, EEPROM-ში ჩაწერილი მონაცემები ნებისმიერ დროს შეიძლება გადაწერილი იყოს Scratchpad მეხსიერებაში.

საკონფიგურაციო რეგისტრი

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	R1	R0	1	1	1	1	1

სურ.3.12

საკონფიგურაციო რეგისტრში მომხმარებლის მიერ გამოიყენება მხოლოდ ორი ბიტი: R0 და R1 (სურ. 3.12). ეს ბიტები განსაზღვრავენ ტემპერატურის გარდამსახის გარჩევითობას, (ანუ მიღებული კოდის თანრიგთა რაოდენობას). უთქმელობით ამ თანრიგებში დაყენებულია 0, რაც შეესაბამება გარდასახვის 9-ბიტიან გარჩევითობას.

ამ ბიტების შესაძლებელი კომბინაციები და მათი შესატყვისობა გარჩევითობებთან ნაჩვენებია ცხრილ. 3.2-ში. უნდა აღინიშნოს, რომ რაც მეტია გარდასახვის გარჩევითობა, მით უფრო მეტია გარდასახვისათვის საჭირო დრო.

ცხრილი 3.2

R1	R0	Resolution	Max Conversion Time	
0	0	9-bit	93.75 ms	(t_{CONV}/8)
0	1	10-bit	187.5 ms	(t_{CONV}/4)
1	0	11-bit	375 ms	(t_{CONV}/2)
1	1	12-bit	750 ms	(t_{CONV})

მონაცემთა გადაცემის ოპერაციების თანამიმდევრობა

ყოველთვის მმართველი სისტემის გადამწოდთან მიმართვის შემთხვევაში დაცული უნდა იყოს მოქმედებების შემდეგი თანმიმდევრობა:

1) ინიციალიზაცია;
2) ROM ბრძანების გადაცემა;
3) გადამწოდის ფუნქციონალური ბრძანების გადაცემა;

თუ რომელიმე ბიჯი გადამწოდთან მიმართვის დროს გამოტოვებულია, მას არ ექნება რეაქცია.

ყველა გადაცემა იწყება ინიციალიზაციით. ეს ოპერაცია სრულდება წამყვანი მოწყობილობის მიერ მიმყოლისთვის ნულზე დაყენების სიგნალის გაგზავნით, რომლის საპასუხოდ მიმყოლი მოწყობილობები (ჩვენ შემთხვევაში ტემპერატურის გადამწოდი) უგზავნის წამყვანს სიგნალს ყოფნის შესახებ, რომლითაც ატყობინებენ მას, რომ გადამწოდები მიერთებულია და არიან მუშაობისთვის მზადყოფნაში. საერთოდ 1-wire ინტერფეისის სალტე, რომელიც რეალიზებულია გადამწოდში, მონაცემთა სალტეზე იყენებს სიგნალების რამდენიმე ტიპს: განულების იმპულსს, ყოფნის იმპულსს, 0-ის ჩაწერას, 1-ის ჩაწერას, 0-ის ამოკითხვას, 1-ის ამოკითხვას. ყველა აღნიშნულ ოპერაციას ახორციელებს წამყვანი მოწყობილობა.

გადამწოდის ბრძანებები

გადამწოდის მუშაობა განისაზღვრება ბრძანებების საშუალებით. გამოიყენება ორი ტიპის ბრძანება: ROM- ბრძანებები და ფუნქციური ბრძანებები.

ROM- ბრძანებები

ეს ბრძანებები იხმარება გადამწოდის ინიციალიზაციის შემდეგ და შეიცავს ინსტრუქციებს გადამწოდთან მუშაობის შესახებ. თითოეული ბრძანება 8 თანრიგაა. შესაბამისი ბრძანების შესრულების შემდეგ შესაძლებელია ფუნქციონალური ბრძანების შესრულება. ROM ბრძანებების აღწერა მოცემულია ცხრილ 3.3-ში, ხოლო ფუნქციონქლური ბრძანებების აღწერა - ცხრილ 3.4-ში.

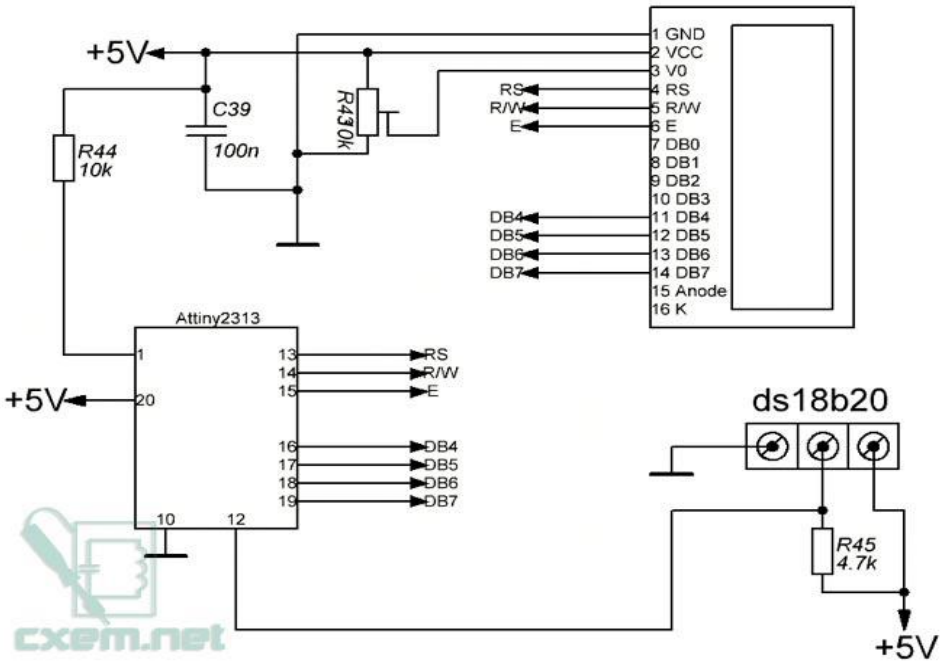
ცხრილი 3.3

SEARCH ROM [F0h]	ამ ბრძანების საშუალებით მმართველი მოწყობილობა არკვევს სალტესთან მიერთებულ მოწყობილობებს. ვინაიდან ჩვენ შემთხვევაში მხოლოდ ერთი გადამწოდია მიერთებული , ამ ბრძანებას არ გამოვიყენებთ.
READ ROM [33h]	ეს ბრძანება გამოიყენება, როდესაც სალტეზე მხოლოდ ერთი გადამწოდია. Eეს შესაძლებლობას აძლევს წამყვან მოწყობილობას წაიკითხოს 64- ბიტიანი ROM მეხსიერება მისი მოძებნის ბრძანების გარეშე.
MATCH ROM [55h]	ეს ბრძანება შესაძლებლობას აძლევს წამყვანს იპოვოს რამოდენიმე მოწყობილობიდან მისთვის საჭირო მოწყობილობა 64 ბიტიანი მისამართის საშუალებით, რომელთანაც შემდეგ შეასრულებს საჭირო ოპერაციას.
SKIP ROM [CCh]	ამ ბრძანების საშუალებით წამყვანი მიმართავს ყველა მასთან დაკავშირებულ მოწყობილობას განურჩევლად მისამართისა, მათთან რაიმე ოპერაციის ერთდროული შესრულებისათვის.
ALARM SEARCH [ECh]	ამ ბრძანების საშუალებით წამყვანი ეძებს განგაშის ალამს ყველა მოწყობილობაში

ცხრილი 3.4

CONVERT T [44h]	ტემპერატურის გარდასახვის დაწყება. ამ ბრძანების შესრულების შედეგად 2-ბაიტიანი კოდი ჩაიწერება აცგ-ს გამომავალ რეგისტრში
amWRITE SCRATCHPAD [4Eh]	წერს მონაცემებს 2-4 რეგისტრებში მეორე რეგისტრიდან დაწყებული უმცროსი ბაიტით.
READ SCRATCHPAD [BEh]	ასრულებს Scratchpad -ის ყველა რეგისტრებში შემცველი მონაცემის გადაცემას მიკროკონტრეოლერში, დაწყებული 0 ბაიტის უმცროსი ბიტიდან, დსრულებული 8 ბაიტის უფროსი ბიტით.
COPY SCRATCHPAD [48h]	ეს ბრძანება გადაწერს 2,3,4 რეგისტრების შემცველობის ასლებს Scratchpad -მეხსიერებიდან შესაბამის EEPROM უჯრედებში.
RECALL E2 [B8h]	ეს ბრძანება მონაცემთა ასლებს EEPROM-იდან Scratchpad -მეხსიერების შესაბამის უჯრედებში. კვების ჩართვის დროს ეს ოპერაცია სრულდება ავტომატურად.
READ POWER SUPPLY [B4h]	ეს ბრძანება აუცილებელია წამყვან მოწყობილობისთვის ინფორმაციის მიწოდებისათვის კვების წყაროს შესახებ, რომელიც გამოიყენება გადამწოდის კვებისათვის. წაკითხვის დროს გადამწოდი პასუხობს 0-ით, . თუ გამოიყენება პარაზიტული კვება და 1-ით , თუ გამოიყენება გარე კვება.

მოწყობილობების პრინციპული სქემა



სურ. 3.13

შემოთავაზებულ სქემაში (სურ. 3.13) ტექსტური დისპლეი მიერთებულია C პორტთან: PC2- დისპლეის RS შესასვლელთან, PC3-EN შესასვლელთან, PC4-D4-თან, PC5-D5-თან, PC6-D6-თან, PC7-D7-თან. ტემპერატურის გადამწოდის შესასვლელი მიერთებულია B პორტის PB0 გამოსასვლელთან. დისპლეისთვის მონაცემთა/ბრძანებების გადაცემა სრულდება ტეტრადებად. ვინაიდან D0-D3 შესასვლელები მოცემულ სქემაში არ გამოიყენება, მათთან მიყვანილია მიწა (ლოგიკური 0). იმის გამო, რომ დისპლეი მუშაობს მხოლოდ მიკროკონტროლერიდან მონაცემთა მიღებაზე, R/W შესასვლელზე მიეწოდება ლოგიკური 0 (მიწა). R43 პოტენციომეტრი გამოიყენება ეკრანზე გამოსახულების სიკაშკაშის ცვლილებისათვის.

ალგორითმი

საწყისი დაყენების ოპერაციები:

- ავაწყოთ PC პორტის გამომყვანები როგორც გამოსასვლელები, დანარჩენი დაყენებები დავტოვოთ უთქმელობით მნიშვნელობებით.

ოპერაციები, რომლებიც შეადგენს პროგრამის ტანს:

- დისპლეის ეკრანის გასუფთავება;
- კურსორის გამორთვა;
- ტემპერატურის გადამწოდში გარდასახვის დაწყება;
- გარდასახვის შედეგის გადაწერა მიკროკონტროლერში;
- ტემპერატურის მნიშვნელობის გამოყვანა დისპლეის ეკრანზე.

პროგრამა C-ენაზე:

/// LCD მოდულის მიკროკონტროლერთან კავშირის აწყობა:

```
1.  sbit LCD_RS at POTC2 bit;
2.  sbit LCD_EN at POTC3 bit;
3.  sbit LCD_D4 at POTC4 bit;
4.  sbit LCD_D5 at POTC5 bit;
5.  sbit LCD_D6 at POTC6 bit;
6.  sbit LCD_D7 at POTC7 bit;
```

///მიკროკონტროლერის გამოსასვლელების გადაცემის მიმართულების აწყობა:

```
7.  sbit LCD_RS_Direction at POTC2 bit;
8.  sbit LCD_EN_Direction at POTC3 bit;
9.  sbit LCD_D4_Direction at POTC4 bit;
10. sbit LCD_D5_Direction at POTC5 bit;
11. sbit LCD_D6_Direction at POTC6 bit;
12. sbit LCD_D7_Direction at POTC7
13. const unsigned short TEMP_RESOLUTION=9; // მოდულის გარდასახვის გარჩევითობის დაყენება

14. char *text=”000.0000”;
15. unsigned temp;
16. void Display_Temperature (unsigned int temp2write) {
17. const unsigned short RES_SHIFT=TEMP_RESOLUTION-8;
18. char temp_whole; // ცვლადის გამოცხადება მთელი ნაწილისათვის;
19. unsigned init temp_fraction; // ცვლადის გამოცხადება წილადი ნაწილისათვის;
```

// ტემპერატურის ნიშნის შემოწმება

```
20. if (temp2write & 0x8000) {
21. text [0]= ‘ – ‘ ;
22. temp2write = ~ temp2write +1; // უარყოფითი რიცხვის გარდასახვა პირდაპირ კოდში;
        }
23. temp_whole = temp2write >> RES_SHIFT; // მთელი ნაწილის გამოყოფა;
   // მთელი ნაწილის წარმოდგენა სიმბოლოებით
24. if (temp_whole/100);
25. text [1] = temp_whole/100+48;   // ასეულების განსაზღვრა;
      else
26. text [1]=’0’;
27. text [2] =( temp_whole/10)%10+48; // ათეულების განსაზღვრა;
28. text [3] = temp_whole%10+48; // ერთეულების განსაზღვრა;
        // წილადი ნაწილის გამოყოფა
29. temp_fraction = temp2write <<( 4- RES_SHIFT);
30. temp_fraction &= 0x0F;
31. temp_fraction *= 625;
     // წილადი ნაწილის გარდასახვა სიმბოლოებად
32. text [4]= temp_fraction/1000        +48;
33. text [5]=( temp_fraction/100)%10 +48;
34. text [6]=( temp_fraction/10)%10  +48;
```

```
35.  text [7]= temp_fraction%10          +48;
36. Lcd_Out (2,5,txt);  // ტემპერატურის მნიშვნელობის გამოყვანა დისპლეიზე;
        }
37. void  main  () {
38.Lcd_Init ( );  //  LCD ინიციალიზაცია;
39.Lcd_Cmd (_LCD_CLEAR); // ეკრანის გასუფთავება;
40. Lcd_Cmd (_LCD_CURSOR_OFF); //  კურსორის გამოთიშვა;
41.Lcd_Out  (1,1," Temperature :   "); //  ტექსტის გამოყვანა ეკრანზე;
42. Lcd_Chr (2,13,223);  // გრადუსის სიმბოლოს გამოყვანა ეკრანზე;
43. Lcd_Chr (2,14, 'C'); //  C სიმბოლოს გამოყვანა;
44. do  {
45. Ow_Reset (& PORB,0);  // 0-ის გაცემა გადამცემის სალტეზე PORTB-ს 0 გამოსასვლელიდან;
46.  Ow_Write ( & PORTB,0, 0xCC); // SKIP_ROM ბრძანების გადაცემა;
47. Ow_ Write  (&  PORTB,0, 0x44);  // CONVERT_T დისპლეის ბრძანების გადაცემა
48. Delay_us (120);
49. Ow_Reset (& PORB,0);
50. Ow_Write ( & PORTB,0, 0xCC); // SKIP_ROM  ბრძანების გადაცემა;
51 Ow_Write ( & PORTB,0, 0xBE); // READ_SCRATCHPAD  ბრძანების გადაცემა;
52.temp = Ow_Read (&PORTB,0);  .// შედეგის უმცროსი ბაიტის წაკითხვა;
53.temp=   Ow_Read (&PORTB,0)+ temp;  // გარდასახვის სრული შედეგის ფორმირება;
54. Display_Temperature  (temp);
55. Delay_ms (500);
     }
56. while (1);  //  უსასრულო ციკლის დასაწყისზე გადასვლა;
}
```

პროგრამა იწყება მიკროკონტროლერის C პორტის გამოსასვლელებსა და დისპლეის შესასვლელებს შორის კავშირის დამყარებით (სტრ. 1-6), აგრეთვე განისაზღვრება გადაცემის მიმართულება C პორტის გამოსასვლელებისთვის (სტრ. 7-12). ამ მიზნით გამოიყენება ზევით განხილული კომპილიატორის ფუნქციები.

შემდეგ სრულდება ზოგიერთი ცვლადების გამოცხადება (სტრ. 13-15): TEMP_RESOLUTION იდენტიფიკატორი ცხადდება როგორც კონსტანტა, რომელსაც ენიჭება მნიშვნელობა 9, რაც შეესაბამება გარდასახვის გარჩევითობას, რომელიც მოცემულ პროგრამაში მიღებულია უთქმელობით; text - მიუთითებს სიმბოლოების მასივის დასაწყისზე; temp-int ტიპის ცვლადი, რომელშიც დისპლეიდან ჩაიწერება გარდასახვის შედეგი.

მე-16 სტრიქონდან აღწერილია ფუნქცია Display_Temperature, რომელსაც გადამწოდიდან წაკითხული მნიშვნელობები გამოჰყავს დისპლეიზე. გარდასახვის მიღებული შედეგი შეიცავს მთელ და წილად ნაწილებს. წილადი ნაწილის თანრიგების რაოდენობა დამოკიდებულია გარდასახვის გარჩევითობაზე (მიღებული კოდის თანრიგების რაოდენობაზე) და გაზომვის დიაპაზონზე. 9 თანრიგის გარჩევითობის შემთხვევაში, რომელიც ჩვენ ავირჩიეთ მოცემულ ამოცანაში, წილად ნაწილს უკავია ერთი უმცროსი თანრიგი. წილადი ნაწილის თანრიგების რაოდენობა ფიქსირდება ცვლადში RES_SHIFT (სტრ. 17). 18 და 19 სტრიქონებში წარმოდგენილია

ცვლადები temp_whole და temp_fraction, რომლებშიც შესაბამისად ჩავწერთ ტემპერატურის მნიშვნელობის მთელ და წილად ნაწილს.

როგორც ზემოთ იყო ნათქვამი, ტემპერატურის მნიშვნელობა შეიძლება იყოს როგორც დადებითი, ისე უარყოფითი. უარყოფითი მნიშვნელობა წარმოდგენილია დამატებით კოდში. ცხადია, დისპლეიზე გამოყვანისათვის ტემპერატურის მნიშვნელობა უნდა წარმოდგენილი იყოს აბსოლუტური მნიშვნელობით (ანუ პირდაპირ კოდში).

უარყოფითი მნიშვნელობის პირდაპირ კოდში გადაყვანის მიზნით სტრიქონ 20-ში მოწმდება გარდასახვის შედეგად მიღებული მნიშვნელობის ნიშანი უფროსი თანრიგის მდგომარეობის მიხედვით if ოპერატორის მეშვეობით, რომლის ფრჩხილებში ჩაწერილი პირობა ჭეშმარიტია აღნიშნულ თანრიგში 1-ის არსებობის შემთხვევაში (ეს თანრიგი გამოიყოფა 0x8000 კოდით temp2write ცვლადიდან, სადაც ჩაწერილია ტემპერატურის მნიშვნელობა), თუ აღნიშნულ თანრიგში იმყოფება 1 (ე.ი. უარყოფითი რიცხვია), მაშინ text მასივის პირველ ელემენტში ჩაიწერება სიმბოლო ”-“ (სტრ. 21) და რიცხვი გარდაიქმნება პირდაპირ კოდში (სტრ. 22).

23 სტრიქონზე ხდება მთელი ნაწილის გამოყოფა მიღებული კოდის ერთი თანრიგით მარჯვნივ ძვრით (RES_SHIFT მნიშვნელობით, რის შედეგად უმცროსი თანრიგი იკარგება) და შედეგი იწერება temp_whole ცვლადში.

24-28 სტრიქონებზე სრულდება მთელი ნაწილის ასეულების, ათეულების და ერთეულების ათობით სიმბოლოებში გამოსახვა: თითოეულ მიღებულ სიმბოლოს ემატება 48 ASCII კოდში წარმოდგენისათვის, რაც აუცილებელია მათი დისპლეიზე გამოყვანისათვის. მიღებული სიმბოლოები იწერებიან text მასივის მეზობელ ელემენტებში.

29-35 სტრიქონებზე სრულდება წილადი ნაწილის გარდასახვა ათობით სიმბოლოებით. თვითოეულ სიმბოლოს ასევე ემატება 48 (ASCII კოდში მისი წარმოდგენისათვის). მიღებული შედეგი ინახება text მასივში.

36 სტრიქონზე Lcd_Out (2,5,txt) ფუნქციის საშუალებით სრულდება text მასივში ჩაწერილი მონაცემების გამოყვანა ეკრანის მეორე სტიქონზე მე-5 პოზიციიდან დაწყებული (რაც მითითებულია ფუნქციის პარამეტრებში).

37 სტრიქონიდან იწყება მთავარი ფუნქციის შესრულება. ფუნქციის დასაწყისში 38,39,40 სტრიქონებზე ხდება დისპლეის ინიციალიზაცია, ეკრანის განულება და კურსორის გამოთიშვა.

41 სტრიქონზე სრულდება ტექსტის ,,Temperature:“ გამოყვანა დაწყებული ეკრანის პირველი სტრიქონის პირველი პოზიციიდან.

42 სტრიქონზე სრულდება გრადუსის აღმნიშვნელი სიმბოლო (223 მნიშვნელობა არის ამ სიმბოლოს კოდი) ეკრანზე გამოყვანის ბრძანება.

43 სტრიქონზე ხდება „C’ სიმბოლოს გამოყვანა ეკრანზე.

44 სტრიქონიდან სრულდება do-while ციკლი, რომლის დროსაც გამოიყენება 1-wire ინტერფეისის ფუნქციები ამ ფუნქციებით სრულდება გადამწოდთან ურთიერთობა, როგორც ეს ზევით იყო ნაჩვენები: ფუნქცია Ow_Reset (&PORB,0) საშუალებით PORB 0 გამოსასვლელით გადამწოდს მიეწოდება 0 სიგნალი, რომელითაც ტემპერატურის გადამწოდი დგება საწყის მდგომარეობაში (სტრ. 45). ამის შემდეგ პორტის იგივე გამოსასვლელიდან გაიცემა ბრძანება SKIP_ROM, რომელიც ამზადებს გადამწოდს შემდეგი ფუნქციონალური ბრძანების შესასრულებლად (სტრ. 46). ფუნქცია Ow_Write (& PORTB,0, 0x44) ასრულებს გადამწოდისთვის CONVERT_T ბრძანების გადაცემას, რომლის საფუძველზე იწყება ტემპერატურის გადამწოდში

გადასახვის პროცესი (სტრ. 47). გარკვეული დაყოვნების შემდეგ, რომელიც საჭიროა გარდასახვის პროცესის შესრულებისათვის გადამწოდი კვლავ გადაყავთ საწყის მდგომარეობაში (სტრ. 49) შესასვლელ სალტეზე 0-ის მიწოდებით. 50 სტრიქონზე კვლავ გადაიცემა SKIP_ROM ბრძანება. 51 სტრიქონზე გაიცემა READ_SCRATCHPAD გადამწოდის მეხსიერებიდან წაკითხვის ბრძანება. 52 და 53 სტრიქონებზე ხორციელდება გარდასახვის შედეგის ჩაწერა temp ცვლადში - ჯერ უმცროსი ბაიტის, შემდეგ უფროსი ბაიტის და სრული შედეგის დაფორმირება.

შემდეგ სრულდება ფუნქცია Display_Temperature, რომელსაც გამოყავს მიღებული შედეგი ეკრანზე.

გარკვეული დაყოვნების შემდეგ პროგრამა გადადის ციკლის დასაწყისში 37 სტრიქონზე და განხილული მოქმედებები მეორდება.

3.9 რეალური დროის საათი

ჩამოვაყალიბოთ შემდეგი სახით ამოცანა:

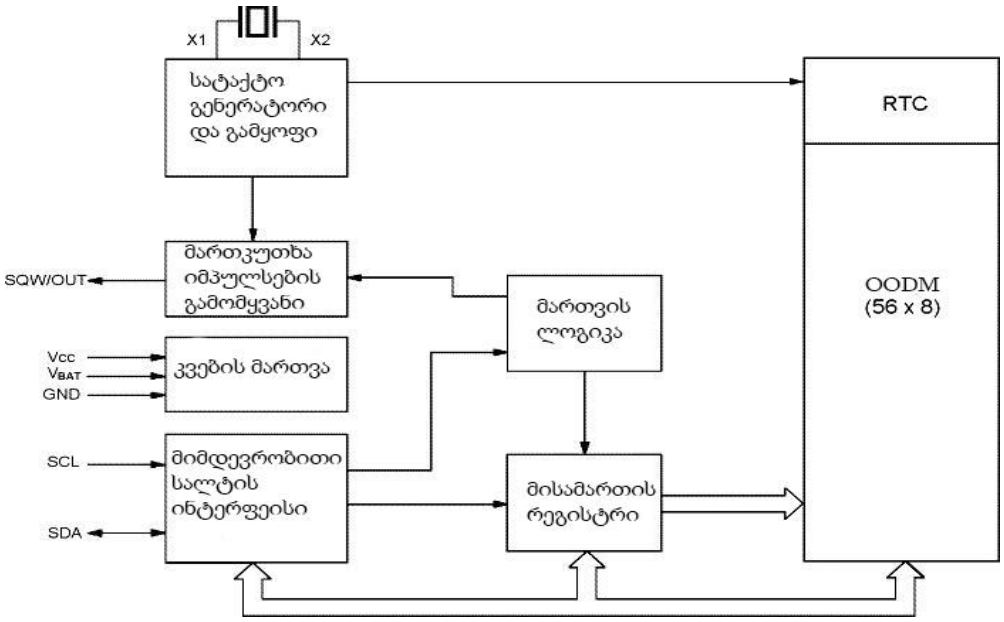
ავაგოთ მოწყობილობა, რომელსაც დროის გარკვეულ ინტერვალებში ტექსტურ დისპლეიზე გამოყავს რეალური დროისა და კალენდარის მნიშვნელობები.

პრინციპიალური ელექტრული სქემა

მოწყობილობის ასაგებად გამოყენებულია მიკროკონტროლერი Atmega 128, რეალური დროის საათის ინტეგრალური სქემა DS1307 და ტექსტური მონიტორი 2x16 LCD, რომელიც ჩვენ მიერ განხილული იყო წინა პარაგრაფებში.

ინტეგრალური სქემა DS1307

მიმდევრობითი ინტერფეისიანი რეალური დროს საათი DS1307 (RTC) წარმოადგენს ორობით-ათობით საათ-კალენდარს, რომელიც შეიცავს 56 ბაიტიან ენერგოდამოუკიდებელ სტატიკურ ოპერატიულ მეხსიერებას. მისამართებისა და მონაცემების გადაცემა ხორციელდება ორგამტარიანი, ორმიმართულებიანი სალტით TWI. საათი-კალენდარი ითვლის წამებს, წუთებს, საათს, კვირის დღეებს, თარიღს, თვეებსა და წელს. თვის ბოლო დღე ავტომატურად კორექტირდება 31-ზე ნაკლებ დღის მქონე თვისთვის ნაკიანი წლის ჩათვლით. საათი მუშაობს როგორც 24, ისე 12 საათიან რეჟიმში AM/PM ინდიკატორით. DS1307-ს აქვს კვებაზე თვალყურის დევნების ჩაშენებული სქემა, რომლიც კვების უწესივრობის შემთხვევაში გადაირთვება ბატარეის კვებაზე. სურ. 3.14-ზე მოცემულია RTC ინტეგრალური სქემის სტრუქტურა.



სურ. 3.14

გამომყვანების აღწერა

V_{cc}, GND - ამ გამომყვანებზე მიეწოდება კვება;

Vcc შესასვლელზე მიეწოდება +5v. როდესაც კვების ძაბვა მეტია 1.25* VBAT, მოწყობილობა მიღწევადია და შესაძლებელია მასში მონაცემთა ჩაწერა ან ამოკითხვა. 3v-იან ბატარეის მიერთებისას, როდესაც Vcc-ს მნიშვნელობა გახდება 1.25*VBAT–ზე ნაკლები, წაკითხვა ან ჩაწერა ვერ სრულდება, მაგრამ დროს თვლის ფუნქცია აგრძელებს მუშაობას. როგორც კი Vcc მნიშვნელობა შემცირდება VBAT-ზე ქვევით, ოპერატიული მეხსიერება და RTC გადაერთვება ბატარეის კვებაზე.

VBAT შესასვლელი ნებისმიერი სამვოლტიანი ბატარეისთვის ან ენერგიის სხვა წყაროსათვის. DS1307 სქემის ნორმალური მუშაობისთვის აუცილებელია, რომ ბატარეის ძაბვა იყოს 2.0.....2.5v დიაპაზონში.

SCL (Serial Clock Input – მიმდევრობითი სინქროიმპულსების შესასვლელი) გამოიყენება მიმდევრობით ინტერფეისში მონაცემთა სინქრონიზაციისთვის.

SDA (Serial Data Input/Output-მიმდევრობითი მონაცემების შესავალ-გამოსავალი) მონაცემების შესასვლელ/გამოსასვლელი ორწვერიანი მიმდევრობითი ინტერფეისისათვის. SDA გამოსასვლელი მოითხოვს გარე დატვირთვის რეზისტორის მიერთებას.

SQW/OUT (Square Wave/Output Driver – მართკუთხა იმპულსების გამოსასვლელი), ამ გამოსასვლელით მოდული აფორმირებს ოთხიდან ერთ-ერთი სიხშირის მართკუთხა იმპულსებს (1hc, 4khc, 8khc, 32khc).

X1,X2-გამომყვანები 32,768კჰც სიხშირის კვარცული რეზონატორის მიერთებისათვის. შიგა ტაქტური გენერატორის სქემა გათვალისწინებულია კვარცულ გენერატორთან მუშაობისათვის.

RTC მეხსიერების დამისამართების ბარათი

RTC მეხსიერების რეგისტრების დამისამართების ბარათი მოცემულია სურ. 3.15-ზე. RTC რეგისტრები განთავსებული მეხსიერების უჯრედებში მისამართით 00h-07h, ხოლო ოპერატიული მეხსიერების რეგისტრები 08h-3Fh მისამართის მქონე უჯრედებში. უჯრედების დამისამართება ხდება მაჩვენებლის (მისამართის რეგისტრის) საშუალებით, რომლის მნიშვნელობა ავტომატურად იზრდება მეხსიერებასთან ყოველი მიმართვის დროს და მიუთითებს შემდეგ უჯრედზე. როდესაც უჯრედებთან მრავალჯერ მიმართვის შედეგად მისამათის მაჩვენებელი მიაღწევს 3Fh მნიშვნელობას, მასში იწერება 00h - არის RTC-ს პირველი უჯრედის მისამართი.

სურ. 3.15

საათი და კალენდარი

საათისა და კალენდრის შესახებ ინფორმაციის წაკითხვა ხდება მეხსიერების შესაბამისი რეგისტრებიდან. RTC მოდულის რეგისტრები ნაჩვენებია სურ. 3.16-ზე. დროისა და კალენდარის რეგისტრების შემცველობა წარმოდგენილია ორობით-ათობით ფორმატში. რეგისტრ 0-ის მე-7-ე ბიტი არის საათის გაჩერების ბიტი (Clock halt-CH). როდესაც ამ ბიტში ჩაწერილია 1-სატაქტო გენერატორი გამორთულია და მოდული არ მუშაობს. 0-ის ჩაწერის შემთხვევაში გენერატორი ჩართულია და მოდული მუშაობს.

DS1307 მოდულს შეუძლია იმუშაოს 12 ან 24 საათიან რეჟიმში. საათის რეგისტრის (რეგისტრი 2) მე-6 ბიტი გათვალისწინებულია რეჟიმის დასაყენებლად: როდესაც ჩავწერთ 1-ს, ამორჩეული იქნება 12 საათიანი რეჟიმი. ამ რეჟიმში ბიტ 5-ში ფორმირდება აღნიშვნა AM/PM (დღის პირველი ნახევარი/ დღის მეორე ნახევარი). 24 საათიან რეჟიმში ეს ბიტი გამოიყენება საათის მეორე ათეულის ჩასაწერად (20-23 საათი).

<table>
<tr><td>ბიტი7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>biti</td></tr>
<tr><td>CH</td><td colspan="3">ათეული წამები</td><td colspan="4">წამები</td></tr>
<tr><td>0</td><td colspan="3">ათეული წუთები</td><td colspan="4">წუთები</td></tr>
<tr><td>0</td><td>12
24</td><td>საათის მეორე ათეული
A/P</td><td>საათის პირველი ათეული</td><td colspan="4">საათი</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="3">dRe</td></tr>
<tr><td>0</td><td>0</td><td colspan="2">თარიღის ათეული</td><td colspan="4">თარიღი</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>თვის ათეულები</td><td colspan="4">თვე</td></tr>
<tr><td colspan="4">წლის ათეულები</td><td colspan="4">წელი</td></tr>
</table>

სურ. 3.16

ორგამტარიანი მონაცემთა მიმდევრობითი სალტე

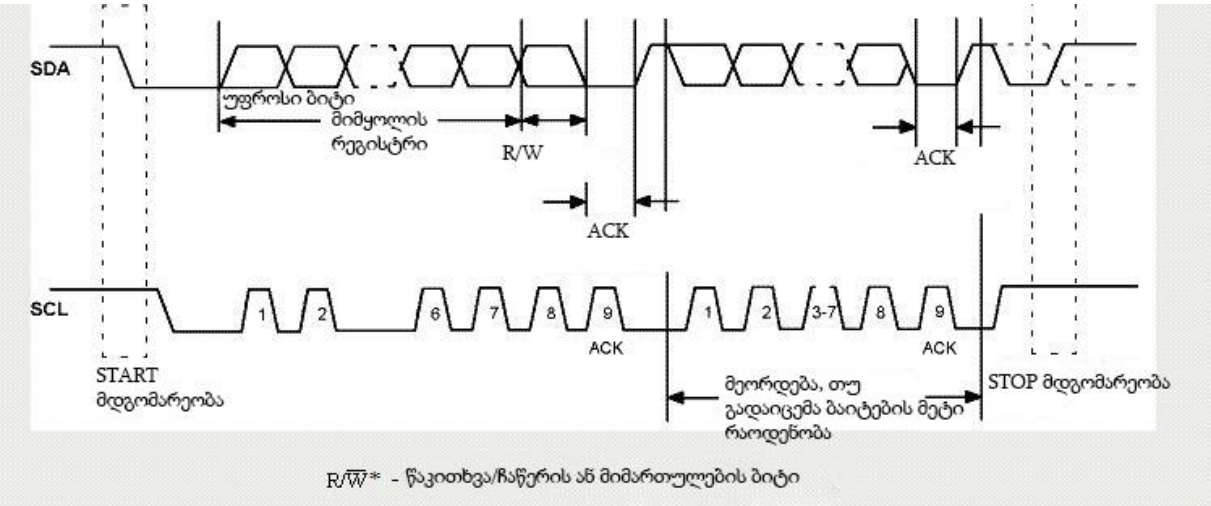
DS1307 ასრულებს მონაცემთა გადაცემას TWI ორგამტარიანი ორმიმართულებიანი სალტის პროტოკოლით. მოწყობილობა, რომელიც გადასცემს მონაცემებს სალტეზე, წარმოადგენს გადამცემს (ანუ წამყვანს), ხოლო მოწყობილობა, რომელიც იღებს მონაცემებს – მიმღებს (ანუ მიმყოლს). წამყვანი მოწყობილობა აფორმორებს მასინქრონებელ იმპულსებს (serial clock- SCL) და ქმნის START და STOP მდგომარეობას. აღნიშნულ სისტემაში DS1307 მოდული მუშაობს როგორც მიმყოლი, მიკროკონტროლერი, როგორც წამყვანი.

TWI სალტით მონაცემთა გადაცემის პრინციპი

სალტე შეიძლება იყოს შემდეგ მდგომარეობაში:

- სალტე არ არის დაკავებული - SDA და SCL გამომყვანებზე შენარჩუნებულია მაღალი პოტენციალი;
- მონაცემთა გადაგზავნის დასაწყისი (START მდგომარეობა) – SDA გამომყვანის მდგომარეობის ცვლილება მაღალიდან დაბალ დონით, როდესაც SCL გამომყვანზე არის მაღალი დონე;
- მონაცემთა გადაგზავნის დამთავრება (STOP მდგომარეობა) - SDA გამომყვანზე მდგომარეობის ცვლილება დაბალიდან მაღალი დონით, როდესაც SCL გამომყვანზე არის მაღალი დონე.

მონაცემთა ყოველი გადაცემა იწყება START მდგომარეობით. და მთავრდება STOP მდგომარეობით. ბაიტიან მონაცემთა რაოდენობა, რომელიც გადაიცემა START და STOP მდგომარეობებს შორის არ არის შეზღუდული და განისაზღვრება წამყვანი მოწყობილობის მიერ. ინფორმაციის გადაცემა სრულდება ბაიტებით და თითოეული ბაიტის მიღებას მიმღები ადასტურებს მე-9 ბიტით (დასტურის ბიტი-ACK). DS1307 მოდული მუშაობს 100კჰც სიხშირით.
სურ. 3.17-ზე ნაჩვენებია მონაცემთა გადაცემის დროითი დიაგრამა ორგამტარიანი სალტით.



1. START მდგომარეობა;
2. R/W ბიტი, რომელიც მიუთითებს გადაცემის მიმართულებას - გაცემა ან მიღება;
3. „დასტური“ სიგნალი;
4. STOP მდგომარეობა.

სურ. 3.17

R/W ბიტისგან დამოკიდებული, შესაძლებელია გადაცემის ორი ტიპი:

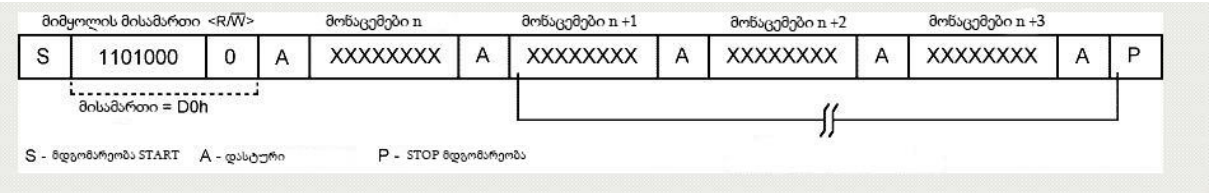
1) მონაცემთა გაგზავნა წამყვან - გადამცემიდან მიმყოლ - მიმღებისკენ.
   წამყვანის მიერ გადაცემული პირველი ბაიტი წარმოადგენს მიმყოლის მისამართს. შემდეგ გადაიცემა მონაცემთა ბაიტების გარკვეული რაოდენობა. მიმყოლი უპასუხებს ყოველი ბაიტის მიღების შემდეგ დასტურის ბიტით. მონაცემები იგზავნება უფროსი ბაიტიდან დაწყებული.
2) მონაცემთა გაგზავნა მიმყოლი გადამცემიდან წამყვან მიმღებისკენ.
   პირველი ბაიტი (მიმყოლის მისამართი) გადაეცემა წამყვანს. წამყვანი უპასუხებს დასტურის ბიტით. ამის შემდეგ მიმყოლი იწყებს ბაიტების გარკვეული რაოდენობის გადაცემას. წამყვანი ყოველ გადაცემულ ბაიტზე პასუხობს დასტურით, გარდა ბოლო ბაიტისა. უკანასკნელი ბაიტის მიღების შემდეგ წამყვანი პასუხობს “არდადასტურებით”. წამყვანი აფორმირებს სინქროიმპულსების მიმდევრობას და START – STOP მდგომარეობას. გადაცემა მთავრდება STOP მდგომარეობით ან START მდგომარეობის გამეორებით. ვინაიდან START მდგომარეობის გამეორება ნიშნავს შემდეგი გადაცემის დაწყებას იმავე წამყვანის მიერ, სალტე არ თავისუფლდება.

ზემოთ თქმულიდან გამომდინარე, DS1307 მოდულს შეუძლია იმუშაოს შემდეგ ორ რეჟიმში:

1.წამყვანი გადამცემის რეჟიმი (DS1307 -ში ჩაწერის რეჟიმი)

მონაცემები მიმდევრობით მიიღებიან SDA გამომყვანით და სინქრონიზირდება SCL სიგნალებით. START და STOP მდგომარეობები აღიქმება როგორც მიმდევრობით გადაცემის დასაწყისი და დამთავრება. მისამართის გამოცნობა სრულდება აპარატულად მიმყოლის მისამართის და მიმართულების ბიტის მიღების შემდეგ (სურ. 3.18).

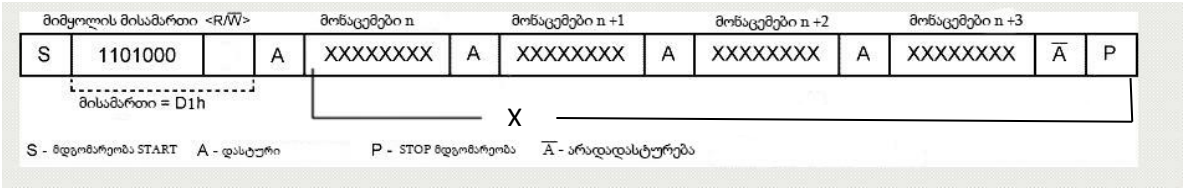
მისამართის ბაიტი არის პირველი ბაიტი, რომელსაც გადასცემს წამყვანი დაწყების მდგომარეობის გენერაციის შემდეგ. იგი შეიცავს DS1307 მოდულის 7 ბიტიან მისამართს, რომელსაც აქვს მნიშვნელობა 1101000 და მიმართულების ბიტს (R/W), რომელიც ჩაწერის შემთხვევაში ნულის ტოლია. სამისამართო ბაიტის მიღებისა და დეკოდირეების შემდეგ DS1307 გასცემს SDA გამომყვანზე დასტურის შეტყობინებას, რის შემდეგ წამყვანი აგზავნის მოწყობილობაში რეგისტრის მისამართს, რომელიც აყენებს მისამართის მაჩვენებელს. შემდეგ წამყვანი იწყებს მონაცემთა ბაიტების გადაცემას. თვითოეული მათგანის მიღება DS1307-ს მიერ დასტურდება შეტყობინებით. მონაცემთა გადაცემის დამთავრებისათვის მიმყოლი აფორმირებს დამთავრების მდგომარეობას.



სურ. 3.18

2) წამყვანი მიმღების რეჟიმი (DS1307 - დან წაკითხვის რეჟიმი)

პირველი ბაიტის მიღება და დამუშავება ხდება ისევე, როგორც მიმყოლი მიმღების რეჟიმის დროს. იმ განსხვავებით, რომ ამ რეჟიმში მიმართულების ბიტი (R/W=1) უჩვენებს, რომ მონაცემთა გადაცემა უნდა შესრულდეს DS1307-დან. START და STOP მდგომარეობის გამოცნობა ხორციელდება მიმდევრობითი გადაცემის დასაწყისში და დასრულების დროს. (სურ. 3.19). მისამართის მიღების და დეკოდირების შემდეგ DS1307 იწყებს მონაცემთა გაცემას მაჩვენებელში მყოფი მისამართიდან დაწყებული. თუ წაკითხვის წინ მისამართის მაჩვენებელში არ ჩავწერთ მისამართს, მაშინ პირველი წაკითხული მისამართი იქნება ის მისამართი, რომელიც ბოლოს შენახული იყო მაჩვენებელში.



სურ. 3.19

დასაგეგმარებელი მოწყობილობის სტრუქტურა

როგორც ზემოთ იყო ნათქვამი, რეალური საათისა და კალენდარის მაჩვენებელი მოწყობილობის ასაგებად ვიყენებთ მიკროკონტროლერ Atmega 128-ს, DS1307 მოდულს და ტექსტურ მონიტორს 2x16 LCD..

DS1307 რეალური დროს საათის გამომყვანები უკავშირდებიან მიკროკონტროლერის D პორტის გამომყვანებს: PD1 - მონაცემთა DS1307 სალტეს. PD0 - სინქროსიგნალების SCL სალტეს. ხოლო ტექსტური დისპლეის გამომყვანები მიკროკონტროლერის C პორტის გამომყვანებს, ზემოთ განხილული ამოცანების ანალოგიურად.

მიკროკონტროლერისა და RTC მოდულის ურთიერთმოქმედება შეიძლება განხორციელდეს ორიდან ერთ-ერთ რეჟიმში: ჩაწერა ან ამოკითხვა.

განვიხილოთ პროგრამა C-ზე, რომელიც ახორციელებს ჩაწერას RTC-ში

```
Void   RT_ Write ( ) {
```

```
1.Delay  ms(1000);
2. TWI_Init (100000) ;  //  წამყვანის რეჟიმის ინიციალიზაცია;
3. TWI_Start  ( );          //  START  მდგომარეობის ფორმირება;
4.TWI_Write (0xD0);   // DS1307  მისამართი +W;
5.TWI_Write (0);         //  მაჩვენებლის დაყენება REG0 მისამართზე;
6.TWI_Write (0x80) ;  //    REG0  -ში $80 -ის ჩაწერა (მოდულის გამორთვა) და 0წმ;
7.TWI_Write (0) ;        //    0 წუთის ჩაწერა REG1 -ში;
8.TWI_Write (0x40);   //  0 საათის ჩაწერა REG2-ში;  12 საათიანი რეჟიმის დაყენება;
9.TWI_Write (0);         //    0 კვირის დღის ჩაწერა REG3-ში;
10.TWI_Write (0);       //    0 თარიღის ჩაწერა  REG4-ში;
11.TWI_Write (0);       //    0 თვის ჩაწერა REG5-ში;
12.TWI_Write (0);       //  0 წლის ჩაწერა REG6-ში;
13.TWI_Stop ( );          //  STOP  მდგომარეობის დაფორმირება;


14.TWI_Start ( );          //  START  მდგომარეობის დაფორმირება;
15.TWI_Write (0xD0); //    DS1307  მისამართი +W;
16. TWI_Write  (0);      //  მისამართის მაჩვენებლის 0-ზე დაყენება ( REG0-ის მისამართი);
17.TWI_Write  (0);      //  მოდულის ჩართვა;
18.TWI_Stop ( );         //    STOP მდგომარეობის დაფორმირება;

 }
```

დასაწყისში სრულდება მიკროკონტროლერში TWI ბლოკის ინიციალიზაცია, რაც მდგომარეობს მის დაყენებაში წამყვან გადამცემის რეჟიმში (სტრიქონი 1-ფუნქცია TWI_Init (100000). 3 სტრიქონიდან იწყება მონაცემთა გადაცემა მიკროკონტროლერიდან RTC მოდულში START მდგომარეობის დაფორმირებით TWI სალტეზე TWI_START () ფუნქციის გამოყენებით. პირველი გადაცემული პაკეტი (სტრიქონი 4) უნდა შეიცავდეს 7 თანრიგა მისამართს, რომელსაც RTC მოდულისათვის აქვს თექვსმეტობითი მნიშვნელობა 68h (ორობით კოდში 1101000) რომელსაც მიეწერება 0 - ჩაწერის ნიშანთვისება. მთლიანად გადასაცემი ბაიტი იქნება D0h (TWI_Write (0xD0)). შემდეგ, სტრიქონ 5-ზე ბაიტების მაჩვენებელში იწერება 0 (TWI_Write (0)), რომელიც მიუთითებს პირველ REG0 რეგისტრზე, საიდანაც იწყება მონაცემთა ჩაწერა მოდულის უჯრედებში.

6 სტრიქონზე REG0-ში იწერება წამების მნიშვნელობა, ხოლო ამ რეგისტრის უფროს თანრიგში, რომელიც მართავს მოდულის ჩართვა/გამორთვას, შეგვაქვს 1 (მოდულს გამოვრთავთ). ამისათვის გამოიყენება ფუნქცია TWI_WRITE (0x80).

7-12 სტრიქონებზე მომდევნო რეგისტრებში იწერება წუთების, საათის, კვირის დღეების, თარიღის, თვის და წლის საწყისი მნიშვნელობები. საათის ჩაწერის დროს ერთდროულად ვაყენებთ 12-საათიან რეჟიმს REG4-ის ბიტ 6-ში 1-ის შეტანით (სტრიქონი 8). მონაცემთა გადაცემა სრულდება წამყვანის მიერ (მიკროკონტროლერი) დასრულების მდგომარეობის ფორმირებით (სტრ. 13, ფუნქცია TWI_STOP ()).

შემდეგ სრულდება მოდულთან განმეორებით მიმართვა მისამართის მაჩვენებლის ნულში (საწყის მდგომარეობაში) დაყენების მიზნით (REG0 რეგისტრის მისამართი) მოდულის მომზადებისათვის წასაკითხად და მისი ჩართვისათვის. ახალი მიმართვის დასაწყისში

მიკროკონტროლერი აფორმირებს გაშვების მდგომარეობას (14 სტრ., ფუნქცია TWI_START ()). 15 სტრიქონზე სრულდება მოდულის მისამართის გაგზავნა, როგორც წინა მიმართვის დროს. 16 სტრიქონზე მისამართის მაჩვენებელში იწერება 0. 17 სტიქონზე REG0-ში შეგვაქვს 0 (ფუნქცია TWI_WRITE (0)). ვინაიდან ნოლის ჩაწერის დროს REG0-ის უფროსი თანრიგი განულდება, მოდული ჩაირთვება და იწყებს დროს ათვლას. 18 სტრიქონზე ფორმირდება გაჩერების მდგომარეობა და მიმართვის ციკლი მთავრდება.

ახლა განვიხილოთ პროგრამა, რომლის მეშვეობით მიკროპროცესორი კითხულობს RTC მოდულიდან მიმდინარე დროს და გამოჰყავს ტექსტურ დისპლეიზე.

```
1.  Unsigned char sec, min, hr, week day,  day, mn,year;
2.  Char  *txt, tnum[4];

// LCD  მოდულთან კავშირის დამყარება:

3.  sbit  LCD_RS  at  PORTC2_bit;
4.  sbit  LCD_EN  at  PORTC3_bit;
5.  sbit  LCD_D4  at  PORTC4_bit;
6.  sbit  LCD_D5  at  PORTC5_bit;
7.  sbit  LCD_D6  at  PORTC6_bit;
8.  sbit  LCD_D7  at  PORTC7_bit;


9.  sbit  LCD_RS_Direction  at  DDC2_bit;
10.  sbit  LCD_EN_Direction  at  DDC3_bit;
11. sbit  LCD_D4_Direction  at  DDC4_bit;
12. sbit  LCD_D5_Direction  at  DDC5_bit;
13. sbit  LCD_D6_Direction  at  DDC6_bit;
14. sbit  LCD_D7_Direction  at  DDC7_bit;


 // DS1307  მოდულიდან წაკითხვის ფუნქციის აღწერა :

15.void  read_Time  (char  *sec, char  *min,  char  *hr,  char  *week_ day, char  *day, char  *mn, char
   *year)  ;  {
16.  TWI  Start  ( );  // გადაცემის დასაწყისის მდგომარეობის ფორმირება;
17.  TWI  Write (0xD0);  // მოდულის მისამართის გადაცემა ჩაწერის ალამთან ერთად;
18.  TWI  Write (0);        // მისამარის მაჩვენებელში 0-ის ჩაწერა ;
19.  TWI  Stop  ( );        // გადაცემის დამთავრების მდგომარეობის ფორმირება ;


20. TWI  Start  ( );        // გადაცემის დასაწყისის მდგომარეობის ფორმირება;
21. TWI  Write (0xD1);  // მოდულის მისამართის გაგ ზავნა წაკითხვის ალამთან ერთად;
22.  *sec=  TWI_Read  (1);  // REG0-დან წამების მნიშვნელობის წაკითხვა;
23.*min= TWI_Read  (1);  // REG1-დან წუთების წაკითხვა;
24.  *hr=  TWI_Read  (1);  // REG2-დან საათის წაკითხვა;
25.  *week_day= TWI_Read  (1); // REG3-დან კვირის დღეების წაკითხვა;
26.  *day= TWI_Read  (1);  // REG 4-დან დღეების წაკითხვა;
27.  *mn= TWI_Read  (1);  // REG5-დან თვის წაკითხვა;
28.  *year= TWI_Read  (0);// REG6-დან წლის წაკითხვა;
```

```
29. TWI Stop ( );     // გადაცემის დამთავრების მდგომარეობის ფორმირება;
    }
// მიღებული მნიშვნელობების სიმბოლოების გარდასახვის ფუნქციის აღწერა:
30. void  Transform_time (char *sec, char *min, char *hr, char *week_ day, char *day, char
    *mn, char *year) ;  {
31. *sec= (( *sec & 0x70)>>4)*10 +  (*sec &  0x0F);
32. *min= (( *min & 0x70)>>4)*10 + (*min & 0x0F);
33. *hr = (( *hr & 0x30)>>4)*10 + (*hr & 0x0F);
34. *week  day = (* week_day & 0x07);
35. *day = (( *day & 0x30)>>4)*10 + (*day & 0x0F);
36. *mn = (( *mn & 0x10)>>4)* 10 +  (* mn  & 0x0F);
37.  *year= ((* year & 0xF0)>>4)*10 + (*year & 0x0F);
}
// LCD-ზე მნიშვნელობის ფუნქციის გამოყვანის აღწერა:
38. void Display_Time (char sec, char min, char hr, char week_ day, char day, char mn, char
    year) ;
{
switch (week_day) ;{
case 1: txt = "Sun";   break;
case 2: txt = "Mon";  break;
case 3: txt = "Tue";   break;
case 4: txt = "Wed";  break;
case 5: txt = "Thu";   break;
case 6: txt = "Fri" ;   break;
case 7: txt = "Sat";   break;
}
39. Lcd_Out (1,1,txt);
40. Lcd_Chr (1,6, (day/10)  +  48);
41. Lcd_Chr (1,7, (day%10)  +  48);
42. Lcd_Chr (1,9, (mn/10)  +  48);
43. Lcd_Chr (1,10, (mn%10)  +  48);
44. Lcd_Chr (1,14, (year/10)  +  48);
45. Lcd_Chr (1,15, (year%10) +48);
46. Lcd_Chr (2,6, (hr/10)  +  48);
47. Lcd_Chr (2,7, (hr%10)  +  48);
48. Lcd_Chr (2,9, (min/10)  +  48);
49. Lcd_Chr (2,10, (min%10)  +  48);
50. Lcd_Chr (2,12, (sec/10)  +  48);
51. Lcd_Chr (2,13, (sec%10)  +  48);
}
```

// *ინიციალიზაციის ფუნქციის აღწერა* :

```
52.void  Init_Main ( ){
53. Lcd_Init ( );
54. Lcd_Cmd (_LCD_CLEAR);
55. Lcd_Cmd (_LCD_CURSOR_OFF);
56. TWI_Init  (100000);
57. Lcd_Chr (1,8,'.');
58. Lcd_Chr (1,11,'.');
59.  txt= " time: " ;
60.  Lcd_Out (2,1,txt);
61.  Lcd_Chr (2,8,' :');
62.  Lcd_Chr (2,11,' :');
63.  txt= "2000";
64.  Lcd_Out (2,12, txt);
65.  Lcd_Cmd ( _LCD_CURSOR_OFF);
}
66. void  main  ( ) {
67.  Init_main  ( );
68.  while  (1)  {
69.  read_Time  (&sec, &min,  &hr,  &week_ day, &day, &mn, &year);
70.  Transform_time  (&sec, &min,  &hr,  &week_ day, &day, &mn, &year);
71.    Display_Time (sec,  min,    hr,    week_ day,  day,  mn, year);
72.    Delay_ms (1000);

       }

  }
```

1,2 სტრიქონებზე წარმოდგენილია ცვლადები და სიმბოლოების მასივი, რომლებიც გამოიყენება პროგრამაში. 3-8 და 9-14 სტრიქონებზე ადრე განხილული ფუნქციების საშუალებით მყარდება კავშირი და გადაცემის მიმართულება ტექსტური დისპლეისა და მიკროკონტროლერის გამომყვანებს შორის.

15 სტრიქონიდან დაწყებული აღწერილია DS1307 მოდულიდან მონაცემთა წაკითხვის ფუნქცია. დასაწყისში (16-19 სტრიქონები) RTC მოდულის მისამართის მაჩვენებელში ჩაიწერება მეხსიერების პირველი რეგისტრის მისამართი, როგორც ეს შესრულებული იყო ზემოთ განხილულ ჩაწერის პროგრამაში. შემდეგ სრულდება მონაცემთა მიმდევრობით ამოკითხვა მოდულის რეგისტრებიდან (20-29 სტრიქონი). TWI ინტერფეისის მუშაობის პროტოკოლის შესაბამისად, პირველ რიგში, მიკრიკონტროლერი აფორმირებს გადაცემის დაწყების მდგომარეობას (სტრ. 20), ამის შემდეგ მიკროკონტროლერი გასცემს SDA სალტით მოდულის მისამართს წაკითხვის ნიშანთვისებასთან ერთად (21 სტრიქონი, ფუნქცია TWI_WRITE (0xD1)) (ბაიტური კონსტანტა 0xD1 ფუნქციის ფრჩხილები შეიცავს 7 თანრიგა მოდულის მისამართს+ ერთთანრიგა წაკითხვის ალამით -1).

22-28 სტრიქონებზე სრულდება მოდულის მეხსიერების რეგისტრებიდან მნიშვნელობების თანმიმდევრული ამოკითხვა და ჩაწერა უჯრედებში, რომლებიც აღნიშნულია, როგორც *sec,

*min, *hr, *week_day, *day, *mn, *year. მონაცემთა გადაცემა მთავრდება დასრულების მდგომარეობის ფორმირებით (29 სტრიქონი).

30-37 სტრიქონებში აღწერილია ფუნქცია, რომელსაც მიღებული მნიშვნელობები გადაყავს ორობით კოდში. როგორც იყო ნათქვამი, დრო და კალენდარი RTC მოდულში წარმოდგენილი არიან ორობით-ათობით ფორმატში. ეკრანზე მათი გამოყვანისათვის მოხერხებულია მათი გარდასახვა ჯერ ორობით შემდეგ კი ათობით სიმბოლოებად. 38-51 სტრიქონებში აღწერილია ფუნქცია, რომელსაც გამოჰყავს მოდულიდან მიღებული მონაცემები ტექსტურ დისპლეიზე. ამ ფუნქციის პარამეტრებს წარმოადგენენ RTC მოდულიდან წაკითხული დროის და კალენდარის მნიშვნელობები, წარმოდგენილი ორობით ფორმატში. ფუნქციის დასაწყისში SWITCH ოპერატორის მიერ txt ცვლადში იწერება კვირის დღის მიმდინარე მნიშვნელობა, რომელიც გამოისახება დისპლეიზე LCD_OUT (1,1,txt) ფუნქციის საშუალებით (სტრ. 39). 40-51 სტრიქონებში სრულდება დროს და კალენრის ათობით ფორმატში გამოსახვა და ეკრანზე გამოყვანა. მაგალითად, ფუნქცია Lcd_Chr (1,6, (day/10) + 48) ანგარიშობს დღეების უფროს თანრიგის (ათეულების) მნიშვნელობას და გამოყავს იგი ეკრანის პირველი სტრიქონის მე-6 პოზიციაზე, ხოლო ფუნქცია Lcd_Chr (1,7,(day%10)+48) ანგარიშობს დღეების უმცროს თანრიგის (ერთეულების) მნიშვნელობას და გამოყავს იგი ეკრანის პირველი სტრიქონის მე-7 პოზიციაზე. 48 ემატება მიღებულ სიმბოლოებს მათი ASCII კოდში წარმოდგენისათვის, რომელიც გამოიყენება სიმბოლოების გამოყვანისათვის ეკრანზე.

52-65 სტრიქონები შეიცავენ ინიციალიზაციის ფუნქციის აღწერას. პირველ რიგში სრულდება TWI ბლოკის ინიციალიზაცია ფუნქცია Lcd_Init () საშუალებით, რომელსაც გადაყავს ბლოკი საწყის მდგომარეობაში (სტრ.53). მომდევნო ფუნქციებით სრულდება ეკრანის გასუფთავება, კურსორის გამოთიშვა და TWI ბლოკის დაყენება წამყვანად. შემდეგ სრულდება დისპლეიზე გამოყვანის ბრძანებები.

66-72 სტრიქონებზე წარმოდგენილია მთავარი ფუნქცია, რომელიც თანამიმდევრობით იძახებს და ასრულებს ზევით აღწერილ ფუნქციებს while უსასრულო ციკლში.

3.10 ბგერითი სიგნალების მაფორმირებელი მოწყობილობა

ავაგოთ მარტივი მოწყობილობა, რომელიც სხვადასხვა ღილაკზე დაჭერით გამოსცემს სხვადასხვა სიხშირის ბგერებს.

დავამუშაოთ ელექტრონული მოწყობილობა, რომელსაც აქვს შვიდი შესასვლელი და ერთი ბგერის გამოსასვლელი. თითოეულ შესასვლელთან დაკავშირებულია გადამწოდი, რომელიც წარმოადგენს ნორმალურად გათიშულ ღილაკს. ნებისმიერი ღილაკის დაჭერის შემთხვევაში მოწყობილობამ უნდა გამოიმუშაოს განსაზღვრული სიხშირის ბგერის სიგნალი. ყოველ ღილაკს უნდა შეესაბამებოდეს თავისი სიხშირის ბგერის სიგნალი. თუ ყველა ღილაკი გათიშულია ბგერითი სიგნალი გამოსასვლელზე არ ფორმირდება.

მოცემულ ამოცანაში გამოვიყენოთ მიკროკონტროლერი Atmega 128.

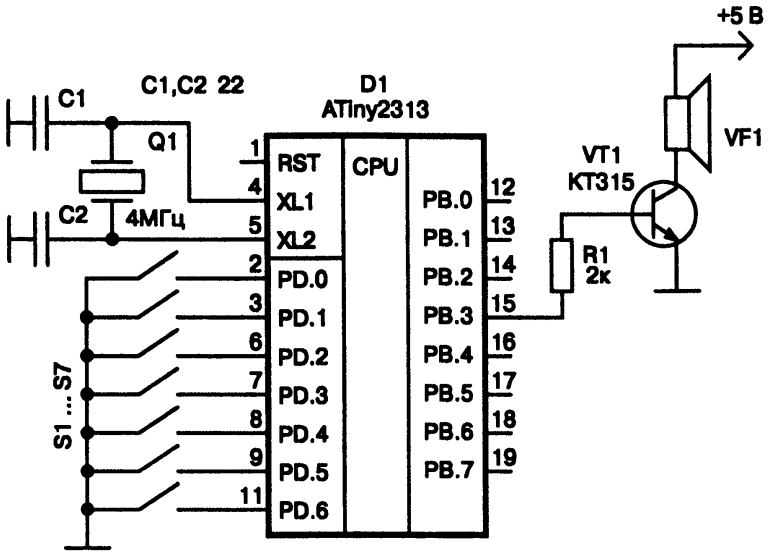
მიკროკონტროლერის ძირითად ამოცანას წარმოადგენს ბგერითი სიხშირის სიგნალების ფორმირება. ბგერითი სიგნალების ფორმირებისათვის მოხერხებულია მიკროკონტროლერის ტაიმერ/მთვლელის გამოყენება.

მიკროკონტროლერის შემადგენლობაში შედის ოთხი ტაიმერ/მთვლელი: რვა თანრიგა T0,T2 და თექვსმეტთანრიგა T1,T3. ბგერის ფორმირებისათვის უმჯობესია თექვსმეტთანრიგა ტაიმერის გამოყენება. რაც მეტია თანრიგების რაოდენობა, მით უფრო ზუსტად შეგვიძლია შევარჩიოთ სიხშირის დაყოფის კოეფიციენტი. ეს მეტად მნიშვნელოვანია სანოტო რიგის ფორმირებისათვის. ამიტომ ავირჩიოთ T1 ტაიმერი. ახლა განვსაზღვროთ არჩეული ტაიმერის მუშაობის რეჟიმი. ბგერის გენერაციისათვის ყველაზე მიზანშეწონილია CTC რეჟიმის გამოყენება (განულება თანხვედრის შემთხვევაში). ამ რეჟიმში მთვლელი რეგისტრი ფუნქციონირებს როგორც ჩვეულებრივი ამჯამავი მთვლელი, რომლის ინკრემენტი ხორციელდება ყოველ სატაქტო იმპულსზე. მაგრამ მთვლელი რეგისტრის მაქსიმალური შესაძლებელი მნიშვნელობა და მაშასადამე მთვლელის გარჩევადობა განისაზღვრება შედარების OCR1 რეგისტრის მნიშვნელობით. შედარების რეგისტრში ჩაწერილი მნიშვნელობის მიღწევის შემდეგ, თვლა გრძელდება 00h მნიშვნელობიდან. ტაიმერის შედარების ბლოკს აქვს რამოდენიმე OC გამოსასვლელი, რომლებზეც აისახება ტაიმერის გადასვლა მაქსიმალური მდგომარეობიდან 00h მდგომარეობაში გამომყვანების მდგომარეობის ცვლილებით. მოცემულ შემთხვევაში გამოიყენება OC1A გამოსასვლელი. შედეგად გამოსასვლელზე ფორმირდება განივიმპულსური მოდულიაციის სიგნალები, რომლებიც მიეწოდება ბგერის ელემენტს PB პორტის მესამე კონტაქტიდან, რომელთანაც ხსენებული გამომყვანი არის დაკავშირებული. გენერირებული სიგნალების სიხშირე განისაზღვრება გამოსახულებით fOCR=fCLKIO/2N(1+OCRn), სადაც N-წინაგამყოფის გაყოფის კოეფიციენტია, OCRn – შესადარებელი მნიშვნელობა.

სიხშირის დიაპაზონი, რომელიც შეიძლება აღქმული იყოს ადამიანის მიერ, იმყოფება 50 ჰც-დან 15 კჰც-მდე ფარგლებში.

უნდა ავირჩიოთ გაყოფის ისეთი კოეფიციენტი, რომელიც OCR1A გამოსასვლელზე მოგვცემს სიხშირეს ადამიანის სმენისათვის დასაშვებ ბგერით დიაპაზონში. თუ მიკროკონტროლერის სიხშირეს ავირჩევთ 4მგც-ს, N=1, OCR-ათასს, მაშინ OC1A გამოსასვლელზე გენერირებული სიხშირე იქნება სასურველ დიაპაზონში.

ბგერის გამოსაყვანად გამოვიყენოთ PB პორტის ერთ-ერთ თანრიგს, ხოლო გადამწოდების მიერთებისათვის - PD პორტი. ზემოთ აღწერილი მოწყობილობის პრინციპული სქემის ვარიანტი ნაჩვენებია სურ. 3.20-ზე.



სურ. 3.20

როგორც მოყვანილი სურათიდან ჩანს, მიკროკონტროლერის ტაქტირებისათვის გამოყენებულია კვარცული რეზონატორი (Q1). გადამწოდების მიერთებისათვის იხმარება იგივე სქემა, რაც გამოიყენებოდა ადრე განხილულ მოწყობილობებში გადამრთველი კონტაქტების მიერთებისათვის. გადამწოდები მიერთებულია PD პორტის გამომყვანებთან. ამასთან გადამწოდების გამართული მუშაობისათვის PD პორტის თითოეულ გამოსასვლელს უნდა მივუერთოთ ჩაშენებული დატვირთვის რეზისტორები. დინამიკის მიერთებისათვის გამოყენებულია გასაღები VT1 ტრანზისტორზე. ეს არის ბგერის ფორმირების ყველაზე მარტივი საშუალება.

აღნიშნულ სქემას აქვს უარყოფითი მხარე. ბგერითი სიგნალის არარსებობის შემთხვევაში მიკროკონტროლერის შესაბამის გამომყვანზე უნდა დავაყენოთ დაბალი ლოგიკური დონე. მაღალ ლოგიკურ დონის არსებობის შემთხვევაში VT1 ტრანზისტორი მუდმივად იქნება გაღებული. ეს გამოიწვევს დინამიკის ხვიაში დაუშვებლად დიდი დენის გავლას, რამაც შესაძლებელია გამოიწვიოს როგორც ტრანზისტორის, ისე დინამიკის მწყობრიდან გამოსვლა. პროგრამის შედგენის დროს ეს მომენტი უნდა იყოს გათვალისწინებული.

ალგორითმი

ერთი შეხედვით ასეთი მოწყობილობის ალგორითმი მარტივია. ნებისმიერი გადამწოდის კონტაქტზე (ღილაკზე) დაჭერის შემთხვევაში მიკროკონტროლერმა უნდა ჩატვირთოს შედარების რეგისტრში საჭირო დაყოფის კოეფიციენტი და მიუერთოს ტაიმერის შედარების სქემის გამოსასვლელი OCR1A გამომყვანს. კონტაქტის აშვებისას მიკროკონტროლერმა გამორთოს ტაიმერის OCR1A გამოსასვლელი პორტის გამომყვანებიდან და გასცეს დაბალი ლოგიკური დონე. თუ ყველა გადამწოდის კონტაქტები აშვებულია, მაშინ პორტის გარე გამომყვანი უნდა დარჩეს გათიშულ მდგომარეობაში. სქემა შედგენილია იმგვარად, რომ შესაძლებელია ერთდროულად რამდენიმე კონტაქტის დაჭერა, რაც ეწინააღმდეგება მოწყობილობის მუშაობის პრინციპს. ამ შემთხვევაში გამოიყენება პრიორიტეტების სისტემა. რამდენიმე კონტაქტის ჩართვის შემთხვევაში პროგრამა უნდა რეაგირებდეს მხოლოდ ერთზე,

რომლის პრიორიტეტი უფრო მაღალია. გამოყენებულია შემდეგი ხერხი. პროგრამა თანამიმდევრობით ამოწმებს ყველა გადამწოდების მდგომარეობას. პირველივე ჩართული კონტაქტის აღმოჩენის შემდეგ პროგრამა წყვეტს სკანირებას და გასცემს ამ გადამწოდის შესაბამის ბგერას.

შევთანხმდეთ, PD.0 შესასვლეზე მიერთებულ გადამწოდს შევუსაბამოთ ნოტა ,,დო", მომდევნო გადამწოდს - ნოტა ,,რე", და ა.შ ნოტა ,,სი"-მდე. სიხშირის დაყოფის კოეფიციენტები თითოეული ნოტისათვის ამოირჩევა მუსიკალური რიგის არსებული ცხრილიდან.

მოწყობილობის მუშაობის პროგრამა C-ზე

```
1. # Include <Atmega 128>  // დაყოფის კოეფიციენტების მასივის გამოცხადება და ინიციალიზაცია;
2. flash unsigned int tabkd [7]={4748,4480,4228,3992,3768,3556,3356};
3. void main (void) ;
{
4. unsigned char count;  // count ცვლადის გამოცხადება;
5. unsigned char temp;  // temp ცვლადის გამოცხადება;
6. PORTB=0x00; // PB პორტის ინიციალიზაცია;
7. DDRB=0x08;
8. PORTD=0x7F; // PD პორტის ინიციალიზაცია;
9. DDRD= 0x00;
10. ACSR=0x80; // კომპარატორის ინიციალიზაცია (გამორთვა);
11. TCCR1A=0x00;  //  T1 ტაიმერ/მთვლელის ინიციალიზაცია;
12. TCCR1B=0x09;
13. while (1);
{
14. m1:  temp=PIND;
15. For  (counter=0;  counter<7;  counter++ ) // სკანირების ციკლი;
            {
16. If (( temp&1)== 0) goto m2; //  temp ცვლადის უმცროსი ბიტის შემოწმება;
17. Temp>>1; // temp ცვლადის მარჯვნივ ძვრა;
         }
18. TCCR1A=0x00; // ბგერის გამორთვა;
19. Goto  m1;  // დასაწყისზე გადასვლა;
20. m2:  OCR1A= tabkd [ count];  // სიხშირის გაყოფის კოეფიციენტის ჩაწერა;
21. TCCR1A=0x04);  // ხმის ჩართვა;
              }
          }
```

პროგრამა შედგება მხოლოდ ერთი main ფუნქციისაგან. პროგრამის დასაწყისში განსაზღვრულია tabkd მასივი (სტრ. 2), რომელის ელემენტებში ჩაწერილია სიხშირის დაყოფის კოეფიციენტები სხვადასხვა ხმის ტონალობისათვის.

4-5 სტრიქონებში შექმნილია ორი ცვლადი:

- ცვლადი count, რომელიც გამოიყენება დაჭერილი ღილაკის განსაზღვრისათვის;
- ცვლადი temp, რომელიც გამოიყენება დამხმარე მიზნებისათვის.

6-12 სტრიქონები უკავია ინიციალიზაციის ოპერატორებს. 6-9 სტრიქონებში სრულდება PORTB და PORTD პორტების გაწყობა, რაც მდგომარეობს, უპირველესყოვლისა, პორტებში გამომყვანების გადაცემის მიმართულების განსაზღვრაში. ამისათვის თითოეული პორტის DDR რეგისტრების შესაბამის თანრიგებში ჩაიწერება 0 ან 1. 0 აყენებს პორტის გამოსასვლელს მონაცემთა შეტანაზე (ჩვენ შემთხვევაში პორტი PD-სტრიქონი 9), 1 - მონაცემთა გამოტანაზე (ჩვენ შემთხვევაში PB პორტის მესამე გამოსასვლელი – სტრიქონი 7). მეორე მხრივ საჭიროა პორტის გამოსასვლელებზე (რომლებიც დაყენებულია სიგნალების შეტანაზე) ჩაირთოს მომჭიმავი რეზისტორები PORT რეგისტრების შესაბამის თანრიგებში 1-ის ჩაწერით (სტრ. 8). 10 სტრიქონში სრულდება კომპარატორის გათიშვა მის მართვის ACSR რეგისტრის უფროს თანრიგში 1-ის ჩაწერით (ანუ 0x80 მნიშვნელობის შეტანით). 11 და 12 სტრიქონებში სრულდება T1 ტაიმერის გაწყობა. ტაიმერის მართვის TCCR1A და TCCR1B რეგისტრებში იწერება კოდები, რომლითაც განისაზღვრება მუშაობის რეჟიმი (მოცემულ შემთხვევაში CTC), ტაიმერის ტაქტირების წყარო და შედარების ბლოკის გამოსასვლელის მდგომარეობის ცვლილების ხასიათი თანხვედრის დროს.

პროგრამის ძირითად ციკლს უკავია 13-21-ე პუნქტები. ეს არის უსასრულო ციკლის ოპერატორი while.

მე-14 სტრიქონში პროგრამა კითხულობს PD პორტის შემცველობას და ათავსებს წაკითხულ ბაიტს temp ცვლადში. ეს ბაიტი შეიცავს ინფორმაციას გადამწოდების მდგომარეობის შესახებ. ერთიანის არსებობა მიღებული ბაიტის რომელიმე ბიტში ნიშნავს შესაბამისი გადამწოდის აშვებულ მდგომარეობას, ხოლო ნოლი - ღილაკის დაჭერილ მდგომარეობას.

15-17-ე სტრიქონებზე სრულდება გადამწოდების სკანირების ციკლი. ციკლის პარამეტრს წარმოადგენს counter ცვლადი. ციკლში მოწმდება temp ცვლადის ყოველი ბიტი მისი უმცროსი ბიტის მდგომარეობის მიხედვით (სტრ. 16).

ბიტის მნიშვნელობის შემოწმებისათვის გამოიყენება გამოსახულება temp &1. ოპერატორი “&” ასრულებს ბიტიან “და” ოპერაციას temp ცვლადსა და 1 (0x01h) შორის. შედეგად, უფროსი თანრიგები ნულდება და გამოსახულების მნიშვნელობა იქნება temp ცვლადის უმცროსი თანრიგი.

ოპერატორი if მე-16 სტრიქონში ამოწმებს გამოსახულების შედეგს. თუ იგი ნულია (მორიგი გადამწოდის კონტაქტი ჩართულია), მართვა გადაეცემა m2 ნიშნაკზე. საწინააღმდეგო შემთხვევაში, (კონტაქტი გათიშულია) გამოკითხვის ციკლი გრძელდება.

მე-17 სტრიქონზე სრულდება temp ცვლადის ძვრა მარჯვნივ, შედეგად ტემპ ცვლადის უმცროს თანრიგში აღმოჩნდება ჩაწერილი კოდის მომდევნო უფროსი ბიტი და სკანირების ციკლი მეორდება. ციკლის შვიდი გავლის შედეგად მოწმდება ყველა შვიდი გადამწოდის მდგომარეობა. თუ ყველა გადამწოდის კონტაქტები აღმოჩნდება გათიშული, მაშინ ციკლი for (სტრიქონები 15-17) მთავრდება ბუნებრივი წესით და პროგრამა გადადის მე-18 სტრიქონზე. აქ სრულდება ხმის გამორთვა TCCR1A რეგისტრის ნულზე დაყენებით.

შემდეგ მე-19 სტრიქონზე მართვა გადაეცემა პროცედურის დასაწყისს. ამისათვის გამოიყენება უპირობო გადასვლის ოპერატორი goto m1.

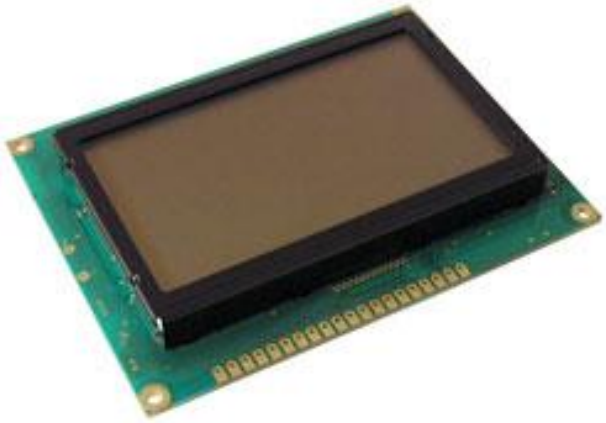
თუ ციკლი მთავრდება ვადაზე ადრე მე-20 სტრიქონზე გადასლით (ნიშნაკი m2), იწყება ხმის ფორმირების პროცესი. ვინაიდან გადამწოდების სკანირების პროცესი დამთავრდა ვადაზე ადრე, count ცვლადში იმყოფება იმ გადამწოდის ნომერი, რომლის კონტაქტი იყო ჩართული.

ახლა საჭიროა tabkd მასივიდან ამ კონტაქტის ნომრის შესაბამისი სიხშირის დაყოფის კოეფიციენტი ჩავწეროთ ტაიმერის შედარების რეგისტრში.

21-ე სრიქონზე სრულდება ხმის ჩართვა. ხმა ჩაირთვება TCCR1A რეგისტრში 0x40 კოდის ჩაწერით. ამით მთავრდება პროგრამის ძირითადი ციკლი. ვინაიდან ძირითადი ციკლი წარმოადგენს უსასრულო ციკლს, მისი ბოლო ბრძანების დამთავრების შემდეგ ციკლი მეორდება.

3.11 მიკროკონტროლერის მუშაობა გრაფიკულ დისპლეისთან

განვიხილოთ სისტემა, რომლის საშუალებით მიკროკონტროლერიდან სრულდება სხვადასხვა გრაფიკული გამოსახულების ან ტექსტის გამოყვანა. ამ სქემისთვის გამოვიყენოთ გრაფიკული დისპლეი WG 12854b კონტროლერით KS0107 და მიკროკონტროლერი Atmega 128. სურ. 3.21-ზე ნაჩვენებია გრაფიკული დისპლეის საერთო ხედი.



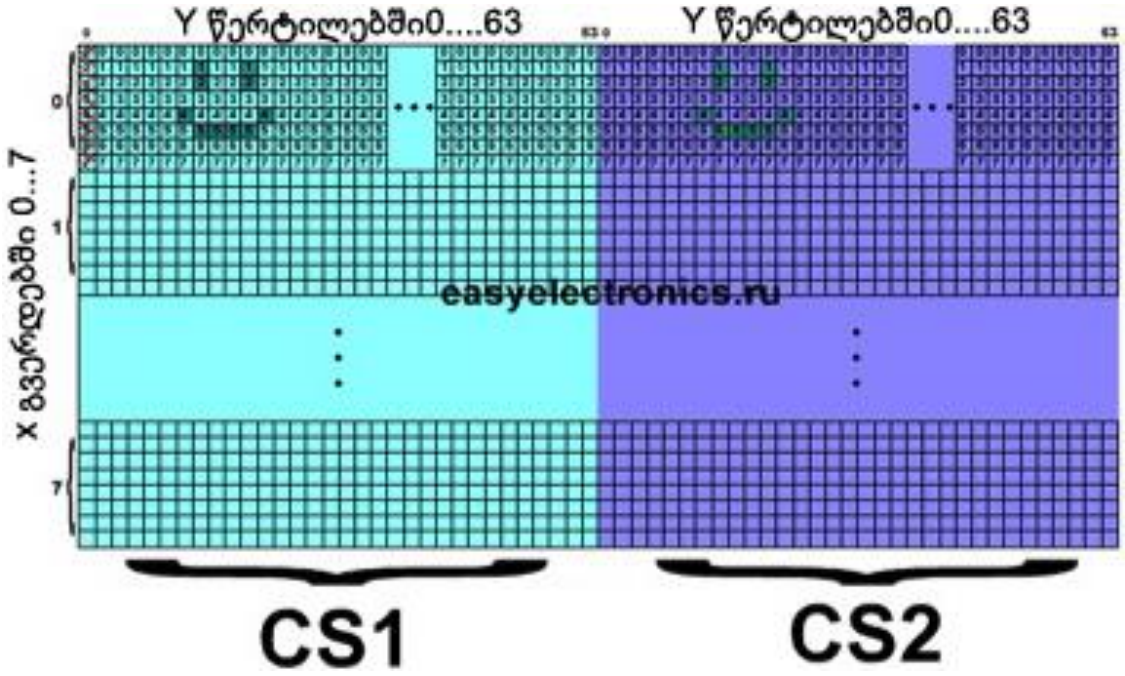
სურ. 3.21

კონტროლერს აქვს გამომყვანები, რომელთა საშუალებით ხორციელდება მონაცემთა, ბრძანებების და სამართავი სიგნალების გადაცემა.

- Vdd და Vssკვების წყაროს შესასვლელი, რომელზეც მიეწოდება +5 ვ.
- Vee — უარყოფიტი ძაბვის წყაროს შესასვლელი, რომელზეც მიეწოდება დაახლოებით -5ვ.
- Vo — კონტრასტის რეგულირების შესასვლელი. ამ შესასვლელზე მიეწოდება 0-5 ვოლტი.
- RS — მონაცემები/ბრძანება. ლოგიკური დონის მნიშვნელობა ამ გამოსასვლელზე განსაზღვრავს მონაცემთა სალტეზე მოწოდებული კოდის ხასიათს: 1-მონაცემები, 0-ბრძანება.
- R/W— წაკითხვა/ჩაწერა. ლოგიკური დონე ამ გამომყვანზე მიუთითებს მოქმედების ხასიათს: 1-წაკითხვა, 0- ჩაწერა.
- EN — მუშაობის ნების დართვის სიგნალის შესასვლელი.
- DB0..7 — მონაცემთა სალტე, რომლითაც გადაიცემა საჭირო კოდი.
- CS1 и CS2 — კონტროლერის ამორჩევა.

- RST — განულების სიგნალის შესასვლელი. ამ შესასვლელზე ნულის მიწოდებით კონტროლერები დგება ნულში. ვიდეომეხსიერება არ იშლება.
- A და K — განათების შუქდიოდების კვება.

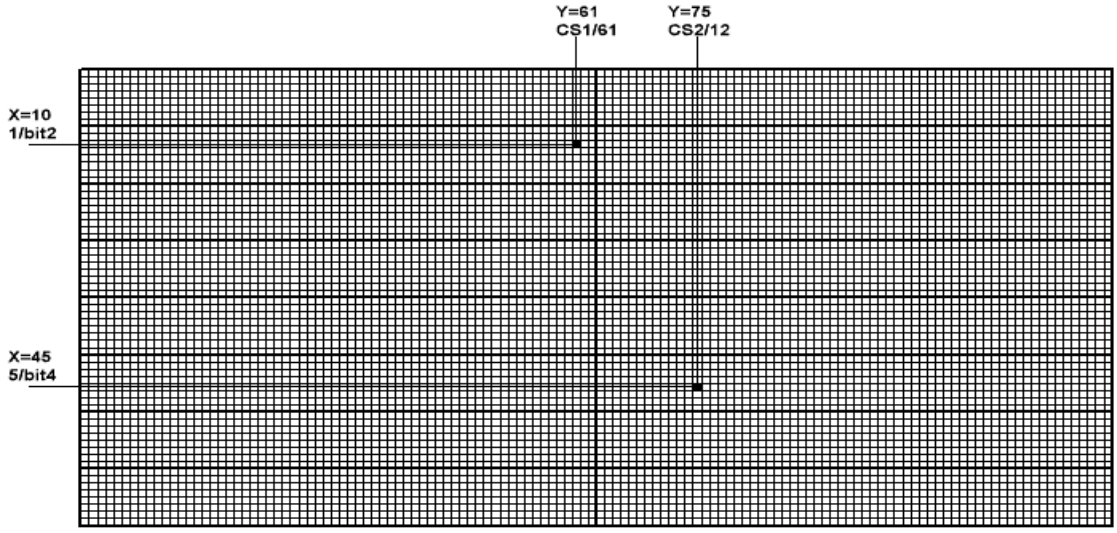
WG12864A დისპლეი არის მონოქრომული თხევადკრისტალური მატრიცა 128x64 გარჩევითობით (ვერტიკალში 128, ჰორიზონტალში 64 პიქსელი), შიგა მმართველი კონტროლერებით KS0108 და შიგა ვიდეო მეხსიერებით 1კბაიტის ტევადობით. ვინაიდან თითოეულ კონტროლერს შეუძლია მხოლოდ 64x64 ზომის მატრიცის ორგანიზაცია, ამიტომ გამოიყენება ორი კონტროლერი: ერთი პასუხს აგებს ეკრანის მარცხენა ნაწილზე, მეორე - მარჯვენა ნაწილზე. ისინი შეიცავს მეხსიერებას, სადაც იწერება ეკრანზე გამოსაყვანი მონაცემები. მეხსიერების თითო ბიტი ეკრანზე აისახება წერტილით. დისპლეის მეხსიერების ბარათს აქვს შემდეგი სახე (სურ. 2.23)



სურ. 3.22

ბაიტები ჩაწყობილია ორ კონტროლერის მეხსიერებაში გვერდებად, თითოში 64 ბაიტი. სულ თითო კონტროლერი შეიცავს 8 გვერდს. იმისათვის, რომ გამოვიყვანოთ ეკრანზე წერტილი კოორდინატებით, მაგალითად, X=10, Y= 61 საჭიროა გამოვითვალოთ რომელ კონტროლერში იმყოფება იგი. თუ Y ნაკლებია 63-ზე, მაშინ წერტილი იმყოფება პირველი კონტროლერის არეში, თუ მეტი, მაშინ მეორე კონტროლერის არეში და წერტილის კოორდინატას უნდა გამოაკლდეს 64. მის შემდეგ გამოითვალოს გვერდის და ბიტის ნომერი. გვერდის ნომერი არის X/8, ხოლო ბიტის ნომერი - გაყოფის ნაშთი (X/8). შემდეგ ამოვიკითხოთ საჭირო ბაიტი ამ

გვერდიდან ჩავწეროთ საჭირო ბიტი და დავაბრუნოთ თავის ადგილას. სურ. 3.23-ზე ნაჩვენებია პიქსელების მდებარეობა ვიდეო მეხსიერებაში.



სურ. 3.23

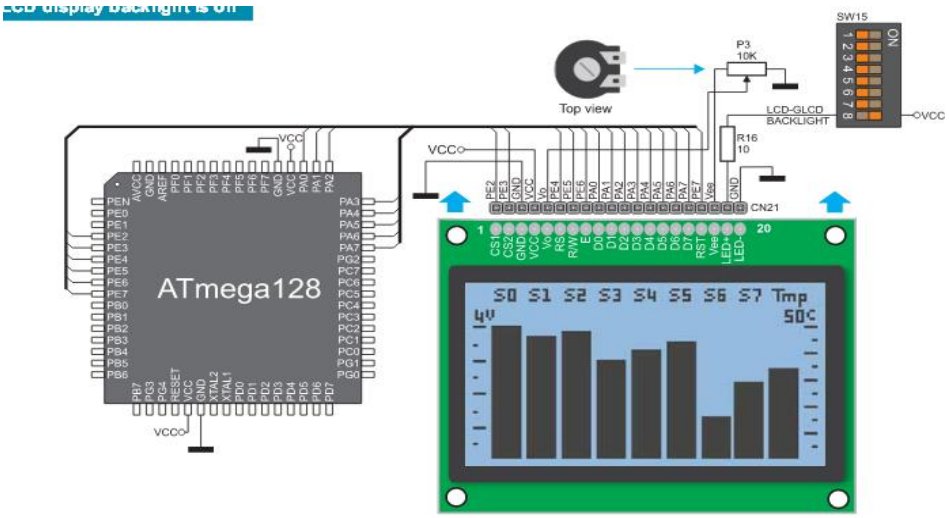
ბრძანებების სისტემა

დისპლეიში სხვადასხვა ოპერაციების შესრულებისათვის მას აქვს რიგი ბრძანებებისა (ცხრილი 3.5):

ცხრილი 3.5 ბრძანებების სისტემა

ბრძანებები	D/I	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	დანიშნულება
გამოსახულების ჩართვა/გამორთვა	0	0	0	0	1	1	1	1	1	L/H	მართავს გამოსახულების ჩართვა/გამორთვას. არ მოქმედებს შიგა მდგომა-რეობაზე და გამოსახუ-ლე ბის მეხსიერების მონაცემებზე. L: გამორთვა H: ჩართვა
Y მისამართის დაყენება	0	0	0	1	Y მისამართი (0 ~ 63)						შეაქვს Y მისამართი Y მისამართის მთვლელში
გვერდის დაყენება (X მისამართი)	0	0	1	0	1	1	1	გვერდი (0 ~ 7)			შეაქვს X მისამართი X მისამართის მთვლელში
გამოსახულების პირველი სტრიქონი	0	0	1	1	გამოსახულების საწყისი სტრიქონი (0 ~ 63)						დაძვრა ზევით. ღამდენი პიქსელით დაიძრას სამისამართო არე.
მდგომარეობის წაკითხვა	0	1	BUSY	0	ON/OFF	RESET	0	0	0	0	მდგომარეობის წაკითხვა : BUSY 0: მზადყოფნა 1: დაკავებულია ბრძანება ON/OFF 0: აჩვენებს ჩართვას 1: აჩვენებს გამორთვას RESET 0: ნორმალური რეჟიმი 1: განულება
გამოსახულების მონაცემთა ჩაწერა	1	0	ჩასაწერი მონაცემები								წერს მონაცემებს (DB0:7) გამოსახულების მონაცემთა მეხსიერებაში. ჩაწერის შრმდეგ Y მისამართი იზრდება 1-ით. ავტომატურად.
გამოსახულების მონაცემთა წაკითხვა	1	1	ამოსაკითხი მონაცემები								კითხულობს მონაცემებს (DB0:7) გამოსახულების მონაცემთა მეხსიერებიდან.ამოიკითხვის შემდეგ Y მისამარტი ავტომატურად იზრდება 1-ით.

მონაცემთა გამომყვანები მიერთებულია პორტ A-სთან. სამართავი შესასვლელები - პორტ E-სთან: CS1-PE2-თან, CS2-PE3-თან, RS-PE4-თან, RW-PE5-თან, EN-PE6, RST-PE7-თან.
ქვემოთ ნაჩვენებია დისპლეის მიკროკონტროლერთან მიერთების სქემა (სურ. 3.24):



სურ. 3.24

მიკროკონტროლერის მუშაობის პროგრამა C ენაზე:

//მასივის გამოცხადება:

```
1. Const  code chare  truck_bmp [1024];
 // Glcd  მოდულის მიერთება მიკროკონტროლერთან;
2. char  GLCD  DataPort  at  PORTA;
3. char  GLCD  DataPort  Direction  at  PORTA;
4. sbit   GLCD_ CS1  at  PORTE2_bit;
5. sbit   GLCD_ CS2  at  PORTE3_bit;
6. sbit   GLCD_ RS   at  PORTE4_bit;
7. sbit   GLCD_ RW  at  PORTE5_bit;
8. sbit   GLCD_ EN  at  PORTE6_bit;
9. sbit   GLCD_ RST at  PORTE7_bit;
10. sbit   GLCD_ CS1_Direction   at  DDE2_bit;
11. sbit   GLCD_ CS2_Direction   at  DDE3_bit;
12. sbit   GLCD_ RS_Direction   at  DDE4_bit;
13. sbit   GLCD_ RW_Direction   at  DDE5_bit;
14. sbit   GLCD_ EN_Direction   at  DDE6_bit;
15. sbit   GLCD_ RST_Direction   at  DDE7_bit;
16. void  delay2S( ) {
17. delay_ms (2000);    // დაყოვნების ფუნქცია 2 წმ;
}
18. void  main ( ) {
19. char  counter;         //  counter  როგორც მთელის განსაზღვრა;
20. char  *someText;   //  უჯრედის გამოყოფა ტექსტისათვის;
21.Glcd_Init ( );       //  დისპლეის ინიციალიზაცია;
22.Glcd_Fill(0x00);  //  დისპლეის გასუფთავება;
23. while  (1)  {
24.Glcd_Image (truck_bmp); // გამოსახულების გამოყვანა ეკრანზე;
25. Delay2S( ); Delay2S( );    //დაყოვნება 4 წმ;
26. Glcd_fill (0x00);     //  ეკრანის გასუფთავება;
27.Glcd_PartialImage    (0,0,68,30,,128,64,  truck_bmp);  //  გამოსახულების ნაწილის გამოყვანა
ეკრანზე;
28. Delay_ms (500);// დაყოფნება 500 წმ ;
29.  Glcd_PartialImage   (24,16,68,30,,128,64,  truck_bmp);//  გამოსახულების ნაწილის გამოყვანა
ეკრანზე;
30. Delay_ms (500);  // დაყოვნება 500 წმ;
31.  Glcd_PartialImage   (56,34,68,30,,128,64,  truck_bmp);  //  გამოსახულების ნაწილის გამოყვანა
ეკრანზე;
32. Delay_ms (500);  // დაყოვნება 500 წმ;
33. Glcd_Fill(0x00);  // ეკრანის გასუფთავება;
```

```
34. Glcd_Box (62, 40, 124,56,1);    // მართკუთხედის გამოყვანა ეკრანზე;
35. Glcd_Rectangle (5,5,84,35,1);  //  მართკუთხედის გამოყვანა ეკრანზე;
36. Delay_ms (1000);

37.     Glcd_Rectangle_Round¬_Edgee          (2,2,87,38,7,1);     //მომრგვალეულ  კუთხეებიანი
    //მართკუთხედის გამოყვანა ეკრანზე
38. Delay_ms (1000);
39.       lcd_Rectangle_Round¬_Edgee       (8,8,81,32,12,1);       //მომრგვალეულ  კუთხეებიანი
    //მართკუთხედის გამოყვანა ეკრანზე
40. Delay_ms (1000);
41. Glcd_Line (0,0,127,63,1); //ხაზის გამოყვანა;
42. Delay2S ( );
43.for (counter=5;counter<60; counter+=5) {  //  ხაზის გამოყვანის ციკლის დასაწყისი;
44. Delay_ms (250);
45. . Glcd_V_Line (2,54,counter , 1); //  ვერტიკალური ხაზების გამოყვანა;
46. . Glcd_H_Line (2,120,counter,1); //  ჰორიზონტალური ხაზებუს გამოყვანა;
}
47. Delay2S ( );
48.Fill(0x00);  // ეკრანის გასუფთავება;
49. Glcd_Set_Font (Character8x7,8,7,32); // შრიფტის ფორმატის გაწყობა;
50. Glcd_Write_Text (“mikroE”,5,7,2);     //  ტექსტის გამოყვანა ეკრანზე
51.for (counter=1;  counter<=10; counter++);//
52. Glcd_Circle (63,32,3*counter,1);  //  წრეხაზის გამოყვანის ციკლი;
53. Delay2S ( );
54. . Glcd_Circle_Fill (63,32,30,1);  //წრეხაზის გამოყვანა;
55. Delay2S ( );
56. Glcd_Box (12,20,70,57,2); // მართკუთხედის გამოყვანა;
57. Delay2S ( );
58.Glcd_Fill (0xFF);  // ეკრანის გამუქება;
59. Glcd_Set_Font (Character8x7,8,7,32); //შრიფტის ფორმატის განსაზღვრა;
60.someText= “8x7  font”;  //ტექსტის ჩაწერა რეგისტრში;
61. Glcd_Write_Text ( someText ,5,0,2);// რეგისტრში ჩაწერილი ტექსტის გამოყვანა ეკრანზე;
62. Delay2S ( );
63. Glcd_Set_Font (Sistem3x5,3,5,32);  // შრიფტის ფორმატის განსაზღვრა;
64. someText= “3x5  CAPITALS  ONLY”;  //  ტექსტის ჩაწერა;
65. Glcd_Write_Text (someText,60,2,2);  //   შრიფტის ფორმატის განსაზღვრა;
66. Delay2S ( );
67. Glcd_Set_Font (Font5x7,5,7,32); //  შრიფტის ფორმატის განსაზღვრა;
68. someText= “5x7  Font”;  // ტექსტის ჩაწერა რეგისტრში;

69. Glcd_Write_Text (someText,5,4,2); // რეგისტრში ჩაწერილი ტექსტის გამოყვანა ეკრანზე;
70. Delay2S ( );
```

```
71. Glcd_Set_Font (FontSistem5x7,v2,5,7,32);  // შრიფტის ფორმატის განსაზღვრა;
72. someText= “5x7  Font (v2)”; // ტექსტის ჩაწერა რეგისტრში;
73. Glcd_Write_Text (someText,50,6,2);// რეგისტრში ჩაწერილი ტექსტის გამოყვანა ეკრანზე;
74. Delay2S ( );
   }

}
```

პროგრამის დასაწყისში მოცემულია ზოგიერთი გამოცხადება: სტრიქონ 1-ში ცხადდება მონაცემთა მასივი truck_bmp, რომლის ზომა ემთხვევა კონტროლერის მეხსიერების ზომას, ეკრანზე გამოსატანი გამოსახულების (სურათის) ჩასაწერად; 2-15 სტრიქონებზე განსაზღვრულია მკ-ის გამომყვანების კავშირი დისპლეის შესასვლელებთან და გადაცემის მიმართულება.

მე-16 სტრიქონზე აღწერილია დაყოვნების ფუნქცია, რომელიც გამოიყენება პროგრამაში.

მე-17 სტრიქონიდან იწყება მთავარი პროგრამა.

მე-18 სტრიქონზე განსაზღვრულია ცვლადი counter, რომელიც გამოიყენება პროგრამაში ციკლის გასვლების რაოდენობის დასათვლელად.

მე-19 სტრიქონზე განსაზღვრულია ტექსტის ჩაწერის არის საწყისი უჯრედი.

პროგრამაში გამოიყენება Glcd დისპლეის მზა ფუნქციები MicroC Pro for AVR ფუნქციების ბიბლიოთეკიდან.

მთავარი პროგრამის დასაწყისში სრულდება ინიციალიზაციის ფუნქცია Glcd_Int( ) (სტრ. 20), რომლის შედეგად ეკრანი დგება საწყის მდგომარეობაში: საწყისი კოორდინატები (წერტილის მისამართი და გვერდი), დაძვრის მნიშვნელობა (ჩვეულებრივად 0) და სრულდება გამოსახულების ჩართვა. ჩართვის შემდეგ დისპლეიზე შესაძლებელია იყოს გაურკვეველი გამოსახულება(,,ნაგავი”). ამის გამო გამოსახულება ჩვეულებრივად გამოყავთ ეკრანზე მისი გასუფთავების შემდეგ ფუნქციით Glcd_Fill(0x00) (სტრ. 21).

შემდგომ სრულდება უსასრულო ციკლი While. დასაწყისში ეკრანზე გამოდის მთლიანი გრაფიკული გამოსახულება Glcd_Image (truck_bmp) ფუნქციის საშუალებით, რომლის პარამეტრი არის ადრე განსაზღვრული მასივი, რომელიც შეიცავს გამოსახულებას (მოცემულ მაგალითში truck_bmp) (სტრ. 22). სრულდება 4 წამიანი დაყოვნება გამოსახულების აღქმისათვის (სტრ. 23) და შემდეგ ეკრანი კვლავ სუფთავდება (სტრ. 24).

მომდევნო სტრიქონებში (სტრიქონები 25,27,29) მიმდევრობით ეკრანის სხვადასხვა ადგილზე გამოდის გამოსახულების ნაწილი Glcd_PartialImage (24,16,68,30,128,64, truck_bmp) ფუნქციის საშუალებით. ფუნქციის პირველი ორი პარამეტრი განსაზღვრავს გამოსახულების მარჯვენა ზევითა წვეროს კოორდინატებს ეკრანზე (მოცემულ მაგალითში X=24; Y=16), შემდეგი ორი პარამეტრი აჩვენებს ეკრანზე გამოყვანილი გამოსახულების ნაწილის სიგანესა და სიმაღლეს (მოცემულ მაგალითში 30 და 68), შემდეგი ორი პარამეტრი აჩვენებს საწყისი გამოსახულების სიგანეს და სიმაღლეს (მოცემულ მაგალითში 128 და 64, შესაბამისად), ბოლო პარამეტრი უჩენებს გამოსაყვანი გამოსახულების ადგილმდებარეობას (მაგალითში, ჩვენ შემთხვევისათვის, truck_bmp). თითოეული განხილული ფუნქციის შესრულების შემდეგ სრულდება 500 მწ დაყოვნება გამოსახულების აღქმისათვის.

31-ე სტრიქონზე კვლავ სრულდება ეკრანის გასუფთავება.

შემდეგ სრულდება ფუნქცია Glcd_Box (62, 40, 124,56,1) (სტრ. 32), რომელსაც გამოყავს ეკრანზე მართკუთხედი. ფუნქციის პირველი ორი პარამეტრი განსაზღვრავს მართკუთხედის მარჯვენა ზევითა წვეროს კოორდინატებს (X=62, Y=40), შემდეგი ორი პარამეტრი ქვევით მარჯვენა წვეროს კოორდინატებს (X=124, Y=56), შემდეგი პარამეტრი - გამოსახულების ფერს: 0-თეთრი, 1-შავი. 33-ე სტრიქონზე ფუნქცია Glcd_Rectangle (5,5,84,35,1) ეკრანზე გამოსახავს აგრეთვე მართკუთხედს. პირველი ორი პარამეტრი განსაზღვრავს მარცხენა ზევითა წვეროს კორდინატებს (x=5,y=5), შემდეგი ორი პარამეტრი - მარჯვენა ქვევითა წვეროს კოორდინატებს (x=84,y=35), ბოლო პარამეტრი-გამოსახულების ფერს (0-თეთრი, 1-შავი. 2- ინვერტირებული).

35-37-ე სტრიქონებზე ფუნქციებს ეკრანზე გამოყავთ მართკუთხედები მომრგვალებული კუთხეებით (ფუნქცია Glcd_Rectangle_Round¬_Edgee (2,2,87,38,7,1)). პირველი ორი პარამეტრი - მარცხენა ზევითა წვეროს კოორდინატები (x=2,y=2), შემდეგი ორი პარამეტრი - მარჯვენა ქვედა წერტილის მდებარეობა(x=83,y=38), შემდეგი პარამეტრი კუთხის მომრგვალების რადიუსი (R=7), ბოლო პარამეტრი - გამოსახულების ფერი.

Glcd_Line (0,0,127,63,1) ფუნქციას ეკრანზე გამოყავს ხაზი. პირველი ორი პარამეტრი განსაზღვრავს ხაზის დასაწყისის კოორდინატებს, მომდევნო ორი პარამეტრი ხაზის დასასრულის კორდინატებს, ბოლო პარამეტრი- გამოსახულების ფერს.

41-ე სტრიქონიდან იწყება პარალელური ვერტიკალური და ჰორიზონტალური ხაზების ეკრანზე გამოყვანის ციკლი Glcd_V_Line (2,54,counter, 1) (სტრ. 43) და Glcd_H_Line (2,120,counter,1) (სტრიქონი 44) ფუნქციების საშუალებით. პირველ ფუნქციას გამოყავს ვერტიკალური ხაზი, რომლის პირველი ორი პარამეტრი არის ხაზის დასაწყისის და დამთავრების Y კოორდინატები, შემდეგი პარამეტრი – ხაზის X კორდინატი (მაგალითში counter ცვლადის მნიშვნელობა), ბოლო პარამეტრი განსაზღვრავს ფერს. მეორე ფუნქციას გამოყავს ჰორიზონტალური ხაზი. პირველი ორი პარამეტრი განსაზღვრავს ხაზის დასაწყისის და ბოლოს X კოორდინატებს. შემდეგი პარამეტრი – ხაზის Y კოორდინატას (counter), ბოლო პარამეტრი - გამოსახულების ფერი. ამგვარად, counter ცვლადის ცვლილებით 5 ბიჯით, თითოეულ გასვლაზე ციკლში ეკრანზე გამოყვანილი იქნება პარალელური ჰორიზონტალური და ვერტიკალური ხაზები (counter=5 მნიშვნელობის მიღწევამდე). დაყოვნების შემდეგ ეკრანი კვლავ სუფთავდება (სტრ. 46).

შემდეგ პროგრამაში სრულდება ტექსტის გამოყვანა სხვადასხვა შრიფტით. დასაწყისში საჭიროა კონტროლერი აეწყოს გამოსაყვანი შრიფტის ფორმატზე. ამისთვის გამოიყენება ფუნქცია Glcd_Set_Font, რომლის პირველი პარამეტრი აწყობს სისტემას შრიფტის ფორმატზე, შემდეგი ორი ფუნქცია განსაზღვრავს შრიფტის ზომას, ხოლო ბოლო - შრიფტების ცხრილის დასაწყისს (MikroC PRO for AVR კომპილატორის შემთხვევაში იწერება 32). ზევით განხილულ პროგრამაში გამოიყენება რამოდენიმე შრიფტი: 8x7; 3x5; 5x7; 5x7 v2. (მათი დაყენება ხდება 47,61, 65,69, სტრიქონებში შესაბამისად). მაგალითად, ფუნქცია Glcd_Set_Font(Character8x7,8,7,32) აყენებს დისპლეის შრიფტის ფორმატზე 8x7 (შრიფტის სიმაღლე 8, სიგანე 7 წერტილი– პიქსელი).

ტექსტის გამოყვანა ეკრანზე ხორციელდება ფუნქციით Glcd_Write_Text, რომლის პირველი პარამეტრი არის გამოსაყვანი ტექსტი, მეორე პარამეტრი X კოორდინატა, მესამე გვერდის ნომერი, ბოლო - შრიფტის ზომა. მაგალითად, lcd_Write_Text (“mikroE”,5,7,2) (სტრიქონი48).

49-ე სტრიქონიდან სრულდება წრეხაზების გამოყვანა Glcd_Circle (63,32,3*counter,1) ფუნქციის საშუალებით (სტრ. 50). პირველი ორი პარამეტრი განსაზღვრავენ წრეხაზის ცენტრის კოორდინატებს, მესამე პარამენტი – წრეხაზის რადიუსს, ბოლო პარამეტრი - გამოსახულების ფერს. ციკლის ყოველ გავლაზე რადიუსის მნიშვნელობა 3*counter იზრდება ounter ცვლადის 10-ის მნიშვნელობის მიღწევამდე. შედეგად ეკრანზე გამოდის სხვადასხვა რადიუსის მქონე კონცენტრირებული წრეხაზები.

52 სტრიქონზე Glcd_Circle_Fill (63,32,30,1) ფუნქციის საშუალებით სრულდება გამოყვანილი წრეხაზის შიგნით მუქი ფერის ,,ჩასხმა". 2 წმ დაყოვნებით ეკრანზე ხდება მართკუთხედის გამოყვანა (სტრიქონი 54).

შემდეგ ეკრანზე გამოიყვანება ტექსტი სხვადასხვა ფორმატით. ყოველი გამოყვანის წინ ხდება კონტროლერის აწყობა შრიფტის შესაბამის ფორმატზე (57,61,65,69). 59,67,71 სტრიქონებზე სრულდება someText ცვლადში წინასწარ ჩაწერილი ტექსტის გამოყვანა ეკრანზე.

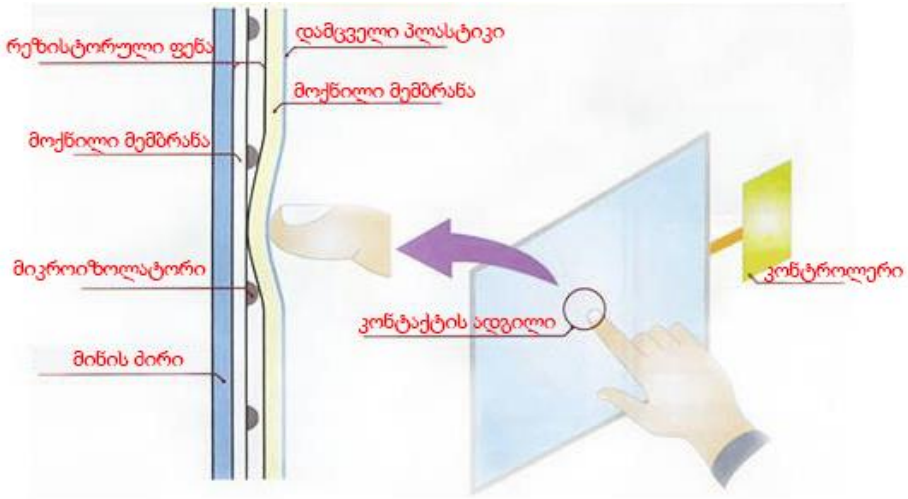
ხდება გადასვლა პროგრამის დასაწყისზე.

3.12 სენსორული პანელის მართვა (Touch Panel)

განვიხილოთ სისტემა, რომელიც შეიცავს სენსორულ პანელს TS12864CRNA. პანელზე ფანქრის დაჭერით GLCD დისპლეიზე გამოდის სხვადასხვა გამოსახულება.

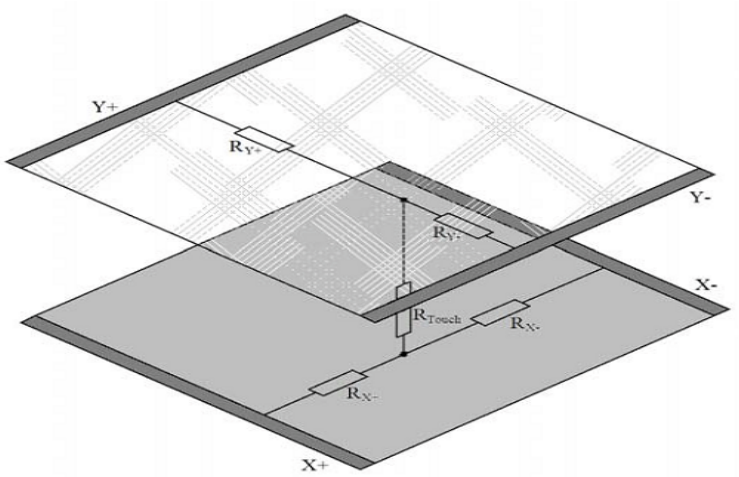
სენსორული პანელის სტრუქტურა და მუშაობის პრინციპი

ამჟამად გამოიყენება სხვადასხვა სახის სენსორული პანელები. ჩვენ განვიხილავთ რეზისტორული ტიპის ოთხგამოსასვლელიან პანელს TS12864CRNA. პანელი, რომელიც იყენებს რეზიტორულ ტექნოლოგიებს შედგება ორი ნაწილისაგან – მოქნილი ზედა და ხისტი ქვედა ფენებისაგან. მოქნილი ფენისთვის გამოიყენება სხვადასხვა პლასტიკური ან პოლიეფირული აფსკები, ხოლო მეორე მზადდება მინისაგან. ორივე ფენის შიგა ნაწილი დაფარულია დენის გამტარი თხელი ფენით, რომელთაც აქვთ გარკვეული წინააღმდეგობა. მათ შორის სივრცე შევსებულია მიკროსკოპული იზოლატორით, რომელიც თანაბრად არის განაწილებული მთელ ზედაპირზე, ჰყოფს ორ ფენას და არ აძლევს მათ შეხების საშუალებას. სურ. 3.25-ზე ნაჩვენებია სენსორული პანელის სტრუქტურა. მოქნილ ფენაზე დაჭერის შემთხვევაში, გამტარი ფენები ეხებიან ერთმანეთს და ქმნის კონტაქტს შეხების წერტილში, ამასთან შეხების წერტილი ქმნის მარტივი ძაბვის გამყოფს. ამის გამო დაწოლის წერტილის კოორდინატები მარტივად შეიძლება გამოითვალოს აცგ-ს საშუალებით 2 გაზომვის შედეგად (ჯერ ერთი კოორდინატას, შემდეგ მეორე კოორდინატას). X კორდინატას გამოთვლისათვის უკანა აფსკის განაპირა სალტეებზე უნდა მივაწოდოთა მუდმივი ძაბვა, მაგალითად +5ვ (მარცხენა X+ სალტეზე მიეწოდება 5ვ, მარჯვენა სალტეზე X--მიწა). ამის შედეგად, უკანა ფენის ყოველ ჰორიზონტალურ უბანზე დენი იწვევს ძაბვის ვარდნას, რომელის მნიშვნელობა უბნის სიგრძის პროპორციულია. ეს მნიშვნელობა ჩვენს მიერ უნდა იყოს წაკითხული. ამისათვის ვიყენებთ წინა ფენას, რომელის სალტეებზე ამ შემთხვევაში ძაბვა არ არის მიწოდებული. მის ერთ-ერთ გამოსასვლელზე, რომელიც აცგ-სთან არის მიერთებული, გამოდის უკანა ფენასთან შეხების წერთილში ძაბვის მნიშვნელობა. ეს იქნება X კოორდინატა.



სურ. 3.25

ანალოგიურად წაიკითხება Y კოორდინატა. ამ შემთხვევაში კვება მიეწოდება წინა ფენის განაპირა სალტეებს (ზევითა Y+ სალტეს მიეწოდება 5ვ, ქვევითა Y- სალტეს - მიწა. წინა ფენის დაჭერილ წერტილში ძაბვის მნიშვნელობა მიეწოდება აცგ-ს უკანა პანელის ერთ-ერთ გამოსასვლელიდან (ამ შემთხვევაში უკანა ფენის გამოსასვლელებზე ძაბვა არ მიეწოდება. სურ. 3.26-ზე ნაჩვენებია სენსორული პანელის ორგანიზაცია.

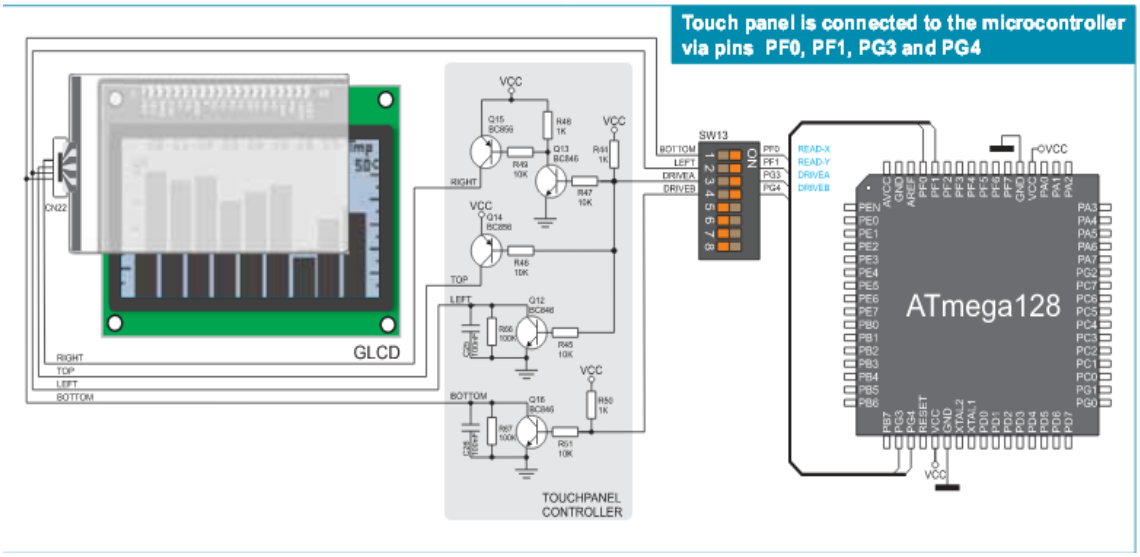


სურ. 3.26

სისტემის პრინციპული სქემა

სისტემა, რომლის დანიშნულებაა სენსორულ პანელზე დაჭერის ადგილის კოორდინატების გამოთვლა და შემდგომ გრაფიკულ დისპლეიზე დაჭერილი ადგილის

შესაბამისი სხვადასხვა გრაფიკული გამოსახულების გამოყვანა, შეიცავს მიკროკონტროლერ Atmega 128-ს, სენსორულ პანელს TS12864CRNA და ჩვენ მიერ ზემოთ განხილულ გრაფიკულ დისპლეის WG12864A 128x64. სურ. 3.27-ზე ნაჩვენებია სენსორული პანელის დაკავშირება მიკროკონტროლერის გამომყვანებთან.



სურ. 3.27

ალგორითმი

1. **საწყისი გაწყობის ოპერაციები**

- სენსორულ პანელის და მიკროკონტროლერის გამომყვანების კავშირის განსაზღვრა;
- პანელზე დაჭერის აღმოჩენის ფუნქციის აღწერა;
- X კოორდინატას განსაზღვრის ფუნქციის აღწერა;
- Y კოორდინატას განსაზღვრის ფუნქციის აღწერა;
- ეკრანის დაკალიბრების ფუნქციის აღწერა.

2. **ოპერაციები რომლებიც ქმნის პროგრამის ტანს**

- სისტემის ინიციალიზაცია;
- გრაფიკულ დისპლეიზე რაიმე ტექსტის და გრაფიკული გამოსახულების გამოყვანა;
- დაჭერის ადგილის კოორდინატების განსაზღვრა;

დაჭერის ადგილის შესაბამისი ინფორმაციის გამოყვანა გრაფიკულ დისპლეიზე (პროგრამის პირველ ვარიანტში) ან ეკრანზე გამოსახულების ხატვა (პროგრამის მეორე ვარიანტში).

პროგრამა C-ზე (პირველი ვარიანტი)

```
1.  char GLCD_DataPort at PORTA;
2.  char GLCD_DataPort_Direction at DDRA;
3.  sbit GLCD_CS1 at PORTE2_bit;

4.  sbit GLCD_CS2 at PORTE3_bit;
5.  sbit GLCD_RS  at PORTE4_bit;
6.  sbit GLCD_RW  at PORTE5_bit;
7.  sbit GLCD_EN  at PORTE6_bit;
8.  sbit GLCD_RST at PORTE7_bit;
9.  sbit GLCD_CS1_Direction at DDE2_bit;
10. sbit GLCD_CS2_Direction at DDE3_bit;
11. sbit GLCD_RS_Direction  at DDE4_bit;
12. sbit GLCD_RW_Direction  at DDE5_bit;
13. sbit GLCD_EN_Direction  at DDE6_bit;
14. sbit GLCD_RST_Direction at DDE7_bit;

    // Glcd მოდულის დაკავშირების დასასრული:

// Touch Panel მოდულის დაკავშირება მიკროკონტროლერთან:

15. sbit DRIVE_A at PORTG3_bit;
16. sbit DRIVE_B at PORTG4_bit;
17. sbit DRIVE_A_Direction at DDG3_bit;
18. sbit DRIVE_B_Direction at DDG4_bit;
19. const char READ_X_CHANNEL = 0;        // READ X ხაზი დაკავშირებულია აცგ-ს 0;
//ანალოგურ არხთან
20.  const char READ_Y_CHANNEL = 1;            // READ Y ხაზი დაკავშირებულია აცგ-ს 1
//ანალოგურ არხთან
21.  unsigned int x_coord, y_coord;
22.int cal_x_min, cal_y_min, cal_x_max, cal_y_max; //მაკალიბრებელი კონსტანტები;
23. char write_msg[] = "WRITE";          // GLCD შეტყობინების მენიუ
24. char clear_msg[] = "CLEAR";
25. char erase_msg[] = "ERASE";
26. const unsigned int ADC_THRESHOLD = 900;     //     ძვრის მნიშვნელობა დაჭერის;
//აღმოჩენისათვის
27. char PressDetect() {    // დაჭერის აღმოჩენის ფუნქცია;
28. unsigned adc_rd;
29. char result;

//დაჭერის აღმოჩენა:

30. DRIVE_A = 0;      // DRIVEA = 0 (LEFT drive off, RIGHT drive off, TOP drive on);
31. DRIVE_B = 0;      // DRIVEB = 0 (BOTTOM drive off);
32. Delay_ms(5);
```

```
33. adc_rd = ADC_Read(READ_Y_CHANNEL);     // Y-წაკითხვა
34. result = (adc_rd > ADC_THRESHOLD);


//დაჭერის აღმოჩენის გამეორება 2 მწ შემდეგ


35.  Delay_ms(2);
 36.  adc_rd = ADC_Read(READ_Y_CHANNEL);     // Y-წაკითხვა;
 37.  result = result & (adc_rd > ADC_THRESHOLD);
 38.  return result;
}
{
 40.  unsigned int result;
 // X-ის წაკითხვა
 41.  DRIVE_A = 1;       // DRIVEA = 1 (LEFT drive on, RIGHT drive on, TOP drive off)
 42.  DRIVE_B = 0;            // DRIVEB = 0 (BOTTOM drive off)
 43. Delay_ms(5);
 44.   result = ADC_Read(READ_X_CHANNEL);     // X-ის წაკითხვა (BOTTOM)
 45.   return result;
 }
 46.   unsigned int GetY() ;
     {
 47.   unsigned int result;
   // Y-ის წაკითხვა
 48.   DRIVE_A = 0;      // DRIVEA = 0 (LEFT drive off, RIGHT drive off, TOP drive on);
 49.   DRIVE_B = 1;     // DRIVEB = 1 (BOTTOM drive on)
 50.   Delay_ms(100);
 51.   result = ADC_Read(READ_Y_CHANNEL); // Y-ის წაკითხვა (LEFT);
 52.   return result;
}
 53.  void Calibrate() {
 54.  Glcd_Dot(0,63,1);
 55.  Glcd_Write_Text("TOUCH BOTTOM LEFT",12,3,1);
 56.   while (!PressDetect());
 57.   cal_x_min = GetX() - 10;
 58.   cal_y_min = GetY() - 10;
 59.   Delay_ms(1000);
 60.  Glcd_Fill(0);
 61.  Glcd_Dot(127,0,1);
 62.  Glcd_Write_Text("TOUCH UPPER RIGHT",12,4,1);
 63.  while (!PressDetect()) ;
 // მაკალიბრებელი კონსტანტის მიღება:
 64.  cal_x_max = GetX() + 5;
```

```
 65.  cal_y_max = GetY() + 5;
 66.Delay_ms(1000);
}
67.  void Initialize() {
68.  DRIVE_A_Direction = 1;    // DRIVE_A კონტაქტის დაყენება, როგორც გამოსასვლელი;
69.  DRIVE_B_Direction = 1;   // DRIVE_B კონტაქტის დაყენება, როგორც გამოსასვლელი;
70.  Glcd_Init();                        // GLCD-ის ინიციალიზაცია;
}
71.  void main() {
72.   Initialize();
73.   Glcd_Fill(0x00);                    // GLCD-ის გასუფთავება;
74.   Glcd_Set_Font(font5x7, 5, 7, 32);      // შრიფტის არჩევა;
75. Glcd_Write_Text("CALIBRATION", 30, 2, 1);
76.  Delay_ms(1500);
 77.  Glcd_Fill(0);
 78.  Calibrate();
 79.  Glcd_Fill(0);
 80.  Glcd_Write_Text("WRITE ON SCREEN", 20, 5, 1) ;
 81.  Delay_ms(1000);
 82.  Glcd_Fill(0);
 83.  Glcd_Fill(0);
 84.  Glcd_V_Line(0,7,0,1);
 85.  Glcd_Write_Text(clear_msg,1,0,1);
 86.  Glcd_V_Line(0,7,97,1);
 87.  Glcd_Write_Text(erase_msg,98,0,1);
  88.  Glcd_Rectangle(41,0,52,9,1);
 89.  Glcd_Box(45,3,48,6,1);
 90.  Glcd_Rectangle(63,0,70,7,1);
 91.  Glcd_Box(66,3,67,4,1);
 92.  Glcd_Rectangle(80,0,86,6,1);
 93.   Gld_Dot(83,3,1);
 94.  Glcd_write_text("Y", 50, 6, 1);
 95.  Glcd_write_text("X", 50, 4, 1);
 96 .   while (1) {
 97.   if(PressDetect());
   {
98.  x_coord = GetX() - cal_x_min;
99.  y_coord = GetY()- cal_y_min;
100.  x_coord = x_coord / 7;
101.  y_coord = (y_coord / 4) - 130;
102. if (x_coord > 88 && y_coord > 60){
103. Glcd_Fill(0);
104. Glcd_wrie_text("otar", 2, 0, 1);
```

```
105. Glcd_Rectangle(1,0,28,9,1);
    }
104. else {
105.   Glcd_Fill(0);
106.   Glcd_Circle(60,30,10,1);
         }
107.  Delay_ms(1500);
         }
      }

  }
```

პროგრამის ზევით მოყვანილი ვარიანტი ითვალისწინებს სხვადასხვა გამოსახულების გამოყვანას გრაფიკულ დისპლეიზე სენსორულ პანელზე დაჭერილი ადგილის შესაბამისად. პროგრამის დასაწყისში სრულდება მიკროკონტროლერის და გრაფიკული დისპლეის გამომყვანებს შორის კავშირის (სტრ.1-8) და გადაცემის მიმართულების (სტრ.9-14) გამოცხადება. სტრიქონებში 15,16 ცხადდება კავშირი მიკროკონტროლერის PG3, PG4 გამომყვანებსა და სენსორული პანელის მართვის შესასვლელებს DRIVE_A, DRIVE_B შორის შესაბამისად, 17,18 სტრიქონებში განისაზღვრება გადაცემის მიმართულება აღნიშნულ გამომყვანებისათვის. 19,20 სტრიქონებში ცხადდება კონსტანტები READ_X_CHANNEL=0; READ_Y_CHANNEL=1, რომლებიც პროგრამაში გამოიყენება შესასვლელი არხის მითითებისათვის აცგ-სთან მიმართვიდან – პირველი აკავშირებს წინა ფენის BOTTOM სალტეს მიკროკონტროლერის PF0 გამომყვანთან (X კოორდინატის წაკითხვა), მეორე - უკანა ფენის LEFT სალტეს მიკროკონტროლერის PF1 გამომყვანთან (Y კოორდინატის წაკითხვა). 21-ე სტრიქონში ცხადდება ცვლადები x_coord, y_coord, რომლებშიც პროგრამის შესრულების დროს ჩაიწერება სენსორულ პანელზე დაჭერის ადგილის კოორდინატები. 22-ე სტრიქონში გამოცხადებულია ე.წ. მაკალიბრებელი კონსტანტები რომლითაც ხდება მიღებული კოორდინატების მნიშვნელობების კორექცია. 23-25-ე სტრიქონებში ცხადდება ტექსტური მასივი, რომლის ელემენტებში იწერება სხვადასხვა შეტყობინების ტექსტი. 26-ე სტრიქონზე განსაზღვრულია კონსტანტა ADC_THRESHOLD, რომელშიც ჩაიწერება აცგ-ს გარდასახვის ზღვარი. ეს არის გარდასახული კოდის მნიშვნელობა, როდესაც აცგ-ს შესასვლელზე არ არის მიწოდებული გარდასასახი ძაბვა. აღნიშნული კოდი განისაზღვრება ე.წ. საყრდენი ძაბვის მნიშვნელობით, რომელიც ყოველთვის მიეწოდება აცგ-ს და წარმოადგენს გარდასახვის ათვლის საწყის მნიშვნელობას (პროგრამაში მიღებულია 900). ამრიგად, აცგ-ს შესასვლელზე ძაბვის მიწოდების შემთხვევაში მისი მნიშვნელობა აითვლება გარდასახვის ზღვრიდან. 27-34-ე სტრიქონებზე განსაზღვრულია პანელზე დაჭერის აღმოჩენის ფუნქცია PressDetect(). 28-29-ე სტრიქონზე ცხადდებიან ცვლადები, რომლებიც გამოიყენება ფუნქციაში. 30-31-ე სტრიქონებში პანელის კონტროლერის შესასვლელებს ენიჭებათ მნიშვნელობები DRIVE_A=0; DRIVE_B=0, ამ მნიშვნელობების საფუძველზე სენსორული პანელის კონტროლერი პანელის გამომყვანებს აყენებს შემდეგ მდგომარეობაში: უკანა ფენის სალტეები (LEFT და RIGHT) გათიშულია კვების წყაროდან, წინა ფენის TOP სალტეს მიეწოდება მაღალი პოტენციალი, BOTTOM სალტე აგრეთვე გათიშულია. ვინაიდან სენსორის წინა ფენის BOTTOM სალტე გათიშულია, დენი ამ ფენაში არ გაივლის და მის ყველა წერთილში ძაბვის მნიშვნელობა იქნება TOP სალტეზე მიწოდებული ძაბვის

ტოლი. ამგვარად, პანელზე დაჭერით მის LEFT გამოსასვლელზე დაფიქსირდება აღნიშნული ძაბვის მნიშვნელობა. ეს გამოსასვლელი თავის მხრივ, დაკავშირებულია აცგ-ს 1 ანალოგურ შესასვლთან, რომლითაც იზომება Y კოორდინატას მნიშვნელობა. გარდასახვის შედეგის წაკითხვა ხდება კომპილატორის ფუნქციის ADC_Read (READ_Y_CHANNEL) საშუალებით (სტრ. 33), რომლის პარამეტრსაც წარმოადგენს არხი 1-ის ნომერი (ფრჩხილებში ჩასმულ კონსტანტას ადრე მივანიჭეთ 1). წაკითხული მნიშვნელობა იწერება ცვლადში adc_rd. როგორც ზევით იყო ნათქვამი, დაჭერის ფაქტი ფიქსირდება, როდესაც გარდასახული კოდის მნიშვნელობა მეტია აცგ-ს გარდასახვის ზღვარზე (მოცემულ პროგრამაში მივიღეთ 900) result ცვლადში ლოგიკური ერთის ჩაწერით (სტრიქონი 34). იმის გასარკვევად, ნამდვილად იყო დაჭერა თუ არა, 2 მწმ-ის შემდეგ კვლავ მოწმდება პანელზე დაჭერა. თუ დაჭერა გრძელდება, ეს ნიშნავს იმას, რომ დაჭერა არ ყოფილა შემთხვევითი (35-38 სტრიქონები) და ფიქსირდება დაჭერის ფაქტი result ცვლადში.

39-45-ე სტრიქონებში განსაზღვრულია X კოორდინატას წაკითხვის ფუნქცია GetX(). ფუნქციის შესრულების დასაწყისში სენსორული პანელის კონტროლერის შესასვლელებს მიკროკონტროლერიდან მიეწოდება კოდი DRIVE_A = 1; DRIVE_B =0 (41,42 სტრიქონები), რის სფუძველზე პანელის სალტეებზე იქმნება შემდეგი მდგომარეობა: უკანა ფენის LEFT სალტეს მიეწოდება მაღალი პოტენციალი, RIGHT სალტეს - მიწა, წინა ფენის TOP და BOTTOM სალტეები გათიშულია კვებისაგან. ამის შედეგად პანელზე დაჭერის შემთხვევაში იზომება ძაბვა უკანა პანელის დაჭერის წერტილში (X კოორდინატა) და მისი მნიშვნელობა წინა ფენის BOTTOM სალტით მიეწოდება აცგ-ს 0 შემავალ ანალოგურ არხს. 44-ე სტრიქონზე ხდება გარდასახულ კოდის აცგ-დან ამოკითხვა და result ცვლადში ჩაწერა.

46-52-ე სტრიქონებში განსაზღვრულია Y კოორდინატას წაკითხვის ფუნქცია GetY(). ფუნქციის შესრულების დასაწყისში სენსორული პანელის კონტროლერის შესასვლელებს მიკროკონტროლერიდან მიეწოდება კოდი DRIVE_A = 0; DRIVE_B = 1 (48,49 სტრიქონები), რის სფუძველზე პანელის სალტეებზე იქმნება შემდეგი მდგომარეობა: უკანა ფენის LEFT და RIGHT გამოსასვლელები გამორთულია, წინა ფენის TOP სალტეს მიეწოდება მაღალი ძაბვა, BOTTOM სალტეს – მიწა. ამის შედეგად პანელზე დაჭერის შემთხვევაში იზომება ძაბვა წინა პანელის დაჭერის წერტილში (Y კორდინატა) და მისი მნიშვნელობა უკანა ფენის TOP სალტით მიეწოდება აცგ-ს 1 შემავალ ანალოგურ არხს. 51-ე სტრიქონში ხდება გარდასახულ კოდის აცგ-დან ამოკითხვა და result ცვლადში ჩაწერა.

53-66 სტრიქონებში განსაზღვრულია დაკალიბრების ფუნქცია Calibrate() . ამ ფუნქციის დანიშნულებაა სენსორული ეკრანიდან წაკითხული კოორდინატა შეუსაბამოს გრაფიკული დისპლეის კოორდინატთა არეს. ფუნქციის შესრულების დასაწყისში (სტრ. 54) დისპლეის ეკრანზე გამოდის წერტილი, რომელიც გვიჩვენებს კოორდინატთა არეს მარცხენა ქვედა ზღვარს და აგრეთვე ტექსტი TOUCH BOTTOM LEFT (სტრ. 55). იმ შემთხვევაში, როდესაც სენსორული პანელი მოთავსებულია გრაფიკულ დისპლეიზე, უნდა დავაჭიროთ აღნიშნულ წერტილს, რითაც ვაფიქსირებთ პანელის ქვედა ზღვრულ კოორდინატას. 56-ე სტრიქონში სრულდება დაჭერის შემოწმება. ვინაიდან ბევრ შემთხვევაში პანელიდან მოწოდებული კოორდინატები არ ემთხვევა დისპლეის ზღვრული წერტილის კორდაინატას, სრულდება მაკორექტირებელი კონსტანტის cal_x_min, cal_y_min გამოთვლა (სტრ. 57,58).

ანალოგიურად სრულდება მარჯვენა ზედა ზღვარის კორექტირება (სტრიქონი 61-66). დისპლეის ეკრანის გასუფთავების შემდეგ (სტრ. 60), გამოგვყავს ეკრანის მარჯვენა ზედა წერტილი (სტრ. 61) და ტექსტი TOUCH UPPER RIGHT. კვლავ სრულდება დაჭერის მოლოდინი (სტრ. 63). პანელზე აღნიშნულ წერტილზე დაჭერით ფორმირდება მისი კოორდინატები, მათი კორექტირების მიზნით გამოითვლება მაკორექტირებელი კონსტანტების მნიშვნელობები cal_x_max , cal_y_max (სტრიქონები 64,65).

67-70-ე სტრიქონებში განსაზღვრულია ინიციალიზაციის ფუნქცია Initialize(). ამ ფუნქციის შესრულების დროს ცხადდება DRIVE_A, DRIVE_B გამომყვანების გადაცემის მიმართულება (სტრიქონი 68,69) და გრაფიკული დისპლეის ინიციალიზაცია (სტრ. 70).

71-ე სტრიქონიდან იწყება მთავარი main() პროგრამა. 72-ე სტრიქონში სრულდება სისტემის ინიციალიზაცია ზევით განხილული Initialize() ფუნქციის გამოყენებით. 78-ე სტრიქონში ხდება მაკორექტირებელი კონსტანტების განსაზღვრა Calibrate() ფუნქციის გამოყენებით. 79-93-ეში სრულდება სხვადასხვა ინფორმაციის გამოყვანა ეკრანზე ჩვენ მიერ განხილული გრაფიკული დისპლეის ფუნქციების გამოყენებით რომელიც ატარებს სადემონსტრაციო ხასიათს. 94-ე სტრიქონიდან სრულდება დაჭერის აღმოჩენის ციკლი. პანელზე დაჭერის დადასტურების შემთხვევაში (სტრ. 95) სრულდება მიღებული კოორდინატების კორექცია მაკორექტირებელი კონსტანტების საშუალებით (სტრიქონები 96-99). მე-100 სტრიქონში მოწმდება პანელის არე. რომელშიც იმყოფება დაჭერის კოორდინატები. თუ დაჭერის ადგილი იმყოფება წინასწარ განსაზღვრულ არის ფარგლებში (განხილულ პროგრამაში x_coord > 88 && y_coord > 60) ეკრანზე გამოვა გარკვეული გამოსახულება (განხილულ პროგრამაში ეს არის ტექსტი ,,Otar”), თუ დაჭერის ადგილი არ იმყოფება მითითებულ არეში, მაშინ ეკრანზე გამოდის სხვა გამოსახულება (განხილულ პროგრამაში წრეხაზი). ზოგადად, შეგვიძლია განვსაზღვროთ პანელის რამდენიმე არე უფრო რთული პირობით, რომელზე დაჭერის შემთხვევაში ეკრანზე გამოვა ჩვენ მიერ წინასწარ შექმნილი სხვადასხვა გამოსახულება (გრაფიკული ან ტექსტური).

პროგრამა C-ზე (მეორე ვარიანტი)

```
1.  char GLCD_DataPort at PORTA;
2.  char GLCD_DataPort_Direction at DDRA;
3.  sbit GLCD_CS1 at PORTE2_bit;
4.  sbit GLCD_CS2 at PORTE3_bit;
5.  sbit GLCD_RS  at PORTE4_bit;
6.  sbit GLCD_RW  at PORTE5_bit;
7.  sbit GLCD_EN  at PORTE6_bit;
8.  sbit GLCD_RST at PORTE7_bit;
9.  sbit GLCD_CS1_Direction at DDE2_bit;
10. sbit GLCD_CS2_Direction at DDE3_bit;
11. sbit GLCD_RS_Direction  at DDE4_bit;
12. sbit GLCD_RW_Direction  at DDE5_bit;
13. sbit GLCD_EN_Direction  at DDE6_bit;
14. sbit GLCD_RST_Direction at DDE7_bit;
```

```
    // Glcd მოდულის დაკავშირების დასასრული;

// Touch Panel მოდულის დაკავშირება მიკროკონტროლერთან,

15.  sbit DRIVE_A at PORTG3_bit;
16.  sbit DRIVE_B at PORTG4_bit;
17.  sbit DRIVE_A_Direction at DDG3_bit;
18.  sbit DRIVE_B_Direction at DDG4_bit;
19.  const char READ_X_CHANNEL = 0;        // READ X ხაზი დაკავშირებულია აცგ-ს 0 ანალოგურ
არხთან;
20.  const char READ_Y_CHANNEL = 1;        // READ Y ხაზი დაკავშირებულია აცგ-ს 1 ანალოგურ
ხაზთან;
21.  unsigned int x_coord, y_coord;
 22.   int   cal_x_min, cal_y_min, cal_x_max, cal_y_max;  // მაკალიბრებელი კონსტანტები;
23.   char write_msg[] = "WRITE";              // GLCD შეტყობინების მენიუ;
24.    char clear_msg[] = "CLEAR";
25.     char erase_msg[] = "ERASE";
26.     const unsigned int ADC_THRESHOLD = 900;  // ძვრის მნიშვნელობა დაჭერის აღმოჩენისათვის ;
 27.  char PressDetect() {    // დაჭერის აღმოჩენის ფუნქცია;
 28.  unsigned adc_rd;
 29.  char result;
 // daWeris  aRmoCena
 30. DRIVE_A = 0;        // DRIVEA = 0 (LEFT drive off, RIGHT drive off, TOP drive on)
 31. DRIVE_B = 0;        // DRIVEB = 0 (BOTTOM drive off)
 32. Delay_ms(5);
 33. adc_rd = ADC_Read(READ_Y_CHANNEL);     // Y-წაკითხვა
 34. result = (adc_rd > ADC_THRESHOLD);
 //დაჭერის აღმოჩენის გამეორება 2 მწმ შემდეგ
 35.  Delay_ms(2);
 36.  adc_rd = ADC_Read(READ_Y_CHANNEL);     // Y-წაკითხვა
 37.  result = result & (adc_rd > ADC_THRESHOLD);
 38.  return result;
}
 39.  unsigned   int   GetX() // X კორდინატას წაკითხვის ფუნქცია;
{
 40.  unsigned int result;
 // X-ის წაკითხვა
 41.  DRIVE_A = 1;       // DRIVEA = 1 (LEFT drive on, RIGHT drive on, TOP drive off);
 42.  DRIVE_B = 0;            // DRIVEB = 0 (BOTTOM drive off);
 43. Delay_ms(5);
 44.   result = ADC_Read(READ_X_CHANNEL);     // X-ის წაკითხვა (BOTTOM);
 45.   return result;
 }
```

```
 46.  unsigned int GetY() ;
      {
 47.  unsigned int result;
   // Y-ის; წაკითხვა 48.  DRIVE_A = 0;      // DRIVEA = 0 (LEFT drive off, RIGHT drive off, TOP drive on)
 49.  DRIVE_B = 1;     // DRIVEB = 1 (BOTTOM drive on)
 50.  Delay_ms(100);
 51.  result = ADC_Read(READ_Y_CHANNEL); // Y-is  wakiTxva (LEFT)
 52.  return result;
}
 53. void Calibrate() {
 54. Glcd_Dot(0,63,1);
 55. Glcd_Write_Text("TOUCH BOTTOM LEFT",12,3,1);
 56.  while (!PressDetect());
 57.  cal_x_min = GetX() - 10;
 58.  cal_y_min = GetY() - 10;
 59.  Delay_ms(1000);
 60. Glcd_Fill(0);
 61. Glcd_Dot(127,0,1);
 62. Glcd_Write_Text("TOUCH UPPER RIGHT",12,4,1);
 63. while (!PressDetect()) ;
 // მაკალიბრებელი კონსტანტის მიღება;
 64. cal_x_max = GetX() + 5;
 65. cal_y_max = GetY() + 5;
 66.Delay_ms(1000);
}
67. void Initialize() {
68. DRIVE_A_Direction = 1;   // DRIVE_A კონტაქტის დაყენება, როგორც გამოსასვლელი ;
69. DRIVE_B_Direction = 1;  // DRIVE_B კონტაქტის დაყენება, როგორც გამოსასვლელი;
70. Glcd_Init();                    // GLCD-ის ინიციალიზაცია;
}
71. void main() {
72.  Initialize();
73.  Glcd_Fill(0x00);                 // GLCD-ის გასუფთავება;
74.  Glcd_Set_Font(font5x7, 5, 7, 32);     // შრიფტის არჩევა;
75. Glcd_Write_Text("CALIBRATION", 30, 2, 1);
76. Delay_ms(1500);
 77. Glcd_Fill(0);
 78. Calibrate();
 79. Glcd_Fill(0);
 80. Glcd_Write_Text("WRITE ON SCREEN", 20, 5, 1) ;
 81. Delay_ms(1000);
 82. Glcd_Fill(0);
 83. Glcd_Fill(0);
 84. Glcd_V_Line(0,7,0,1);
```

```
 85.  Glcd_Write_Text(clear_msg,1,0,1);
 86.  Glcd_V_Line(0,7,97,1);
 87.  Glcd_Write_Text(erase_msg,98,0,1);
 88.  Glcd_Rectangle(41,0,52,9,1);
 89.  Glcd_Box(45,3,48,6,1);
 90.  Glcd_Rectangle(63,0,70,7,1);
 91.  Glcd_Box(66,3,67,4,1);
 92.  Glcd_Rectangle(80,0,86,6,1);
 93.   Glcd_Dot(83,3,1);
 //Glcd_write_text("Y", 50, 6, 1);
 //Glcd_write_text("X", 50, 4, 1);
94 .   while (1) {
95.   if(PressDetect())
    {
96.   x_coord = GetX() - cal_x_min;
97.  y_coord = GetY()- cal_y_min;
98.   x_coord = x_coord / 7;
99.    y_coord = (y_coord / 4) - 130;
100.    if (x_coord < 120)
   {
 101.    Glcd_Dot(x_coord,y_coord,1);
       }

   }
}
```

პროგრამის მეორე ვარიანტი შესაძლებლობას იძლევა ეკრანზე შევასრულოთ რაიმე გამოსახულების ხატვა. პროგრამის 1-99-ე სტრიქონებში სრულდება იგივე მოქმედებები, რაც პირველ ვარიანტში იყო განხილული. მე-100 სტრიქონიდან კი სრულდება ხატვის პროცედურა. ამ სტრიქონში განისაზრვრება სენსორული პანელის არე, რომელშიც უნდა მოხდეს გამოსახულების ხატვა. პანელზე ფანქრის გავლების შედეგად გრაფიკული დისპლეის ეკრანზე უსასრულო ციკლში აისახება წერტილების ერთობლიობა (სტრ. 101), რომელებიც შეადგენს გამოსახულებას (სწორ ხაზს ან მრუდს).

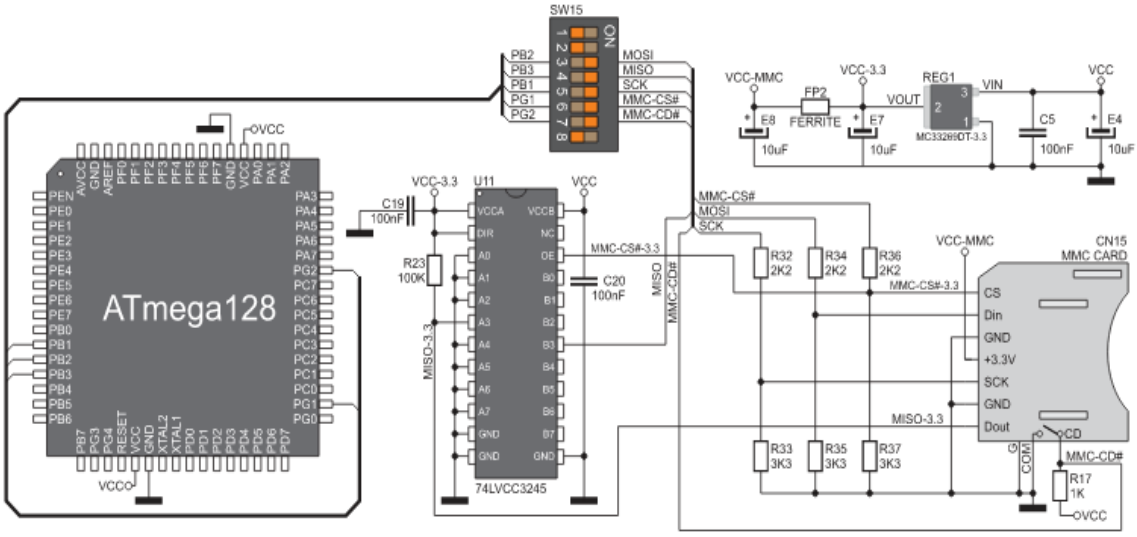
3.13 MMC/CD დისკების მართვა

მიკროკონტროლერის ურთიერთობა დისკთან მდგომარეობს შემდეგი ოპერაციების შესრულებაში:

- ახალი ფაილის შექმნა და მასში ჩაწერა;
- არსებული ფაილის გახსნა და მასში ახალი მონაცემების ჩაწერა;
- არსებული ფაილის გახსნა და მასში მონაცემთა დამატება;
- ფაილის გახსნა და მონაცემების წაკითხვა (მათი გაგზავნა USART პორტის საშუალებით);
- ფაილის წაშლა.

L გამოყენებულია SD დისკი (სურ.3.28 1,8 მბაიტის ტევადობით, იგი თავსდება გასართში და იყენებს FAT16 ფაილურ სისტემას.

სურ. 3.28



სურ. 3.29

ზევით მოყვანილ სურათზე ნაჩვენებია დისკგამწევის მიერთების სქემა მიკროკონტროლერთან SPI ინტერფეისის საშუალებით (სურ.3.29) მონაცემთა გაცვლა ხორციელდება ამ ინტერფეისის პროტოკოლის თანახმად. სქემაში მიკროკონტროლერი არის წამყვანი (Master), დისკგამწევი მიმყოლი (slave). SPI ინტერფეისის MOSI გამოსასვლელი დაკავშირებულია მიკროპროცესორის PB.2 გამომყვანთან, რომლითაც იგზავნება მონაცემები დისკზე, ხოლო MISO შესასვლელი დაკავშირებულია PB.3 პორტის გამომყვანთან, რომლითაც დისკიდან გადაეცემა მონაცემები მიკროკონტროლერს. მიკროკონტროლერის PB.1 პორტიდან ფორმირდება მასინქრონებელი სიგნალები SCK.

მიკროკონტროლერის მიერთებისათვის დისკგამწევთან SW15 გადამრთველის 3,4,5,6,7 კონტაქტები უნდა დაყენდეს ON მდგომარეობაში.

მიკროკონტროლერის მუშაობის პროგრამა C-ზე

//MMC მოდულის მიერთება:

1. sbit Mmc_Chip_Select at PORTG1_bit;

```
2.  sbit Mmc_Chip_Select_Direction at DDG1_bit;
3.  const LINE_LEN = 43;
4.  char err_txt[20]    = "FAT16 not found";
5.  char file_contents[LINE_LEN] = "XX MMC/SD FAT16 library by Anton Rieckert\n";
6.  char  filename[14] = "MIKRO00x.TXT";    // ფაილის სახელი
7.  unsigned short loop, loop2;
8.  unsigned long  i, size;
9.  char          Buffer[512];
```

// UART1 *ტექსტის ჩაწერა და ახალ სტრიქონზე გადასვლა:*

```
10. void UART1_Write_Line(char *uart_text) {
11. UART1_Write_Text(uart_text);
12. UART1_Write(13);
13. UART1_Write(10);

 }
```

// ახალი ფაილის გახსნა და მასში რაიმე მონაცემის ჩაწერა:

```
14. void M_Create_New_File() {
15. filename[7] = 'A';                          // ფაილის სახელის დაყენება;
16. Mmc_Fat_Set_File_Date(2005,6,21,10,35,0);      // ფაილის თარიღისა და დროის დაყენება;
17. Mmc_Fat_Assign(&filename, 0xA0);         // არსებული ფაილის მოძებნა და ახლის შექმნა ;
18. Mmc_Fat_Rewrite();                  // ფაილის გასუფთავება და ახალი მონაცემის დაყენება
19. for(loop = 1; loop <= 99; loop++) {
20. UART1_Write('.');
21. file_contents[0] = loop / 10 + 48;
22. file_contents[1] = loop % 10 + 48;
23. Mmc_Fat_Write(file_contents, LINE_LEN-1);    // მონაცემთა ჩაწერა გახსნილ ფაილში
   }
 }
```

// რამდენიმე ახალი ფაილის შექმნა და მათში მონაცემების ჩაწერა :

```
24. void M_Create_Multiple_Files() {
25. for(loop2 = 'B'; loop2 <= 'Z'; loop2++) {
26. UART1_Write(loop2);
27. filename[7] = loop2;                        // ფაილის სახელის დაყენება
28. Mmc_Fat_Set_File_Date(2005,6,21,10,35,0);    // ფაილის თარიღისა და დროის დაყენება;
29. Mmc_Fat_Assign(&filename, 0xA0);         // არსებული ფაილის მოძებნა და ახლის შექმნა;
30. Mmc_Fat_Rewrite();                  // ფაილის გასუფთავება და მონაცემთა ჩაწერის დასაწყისი;
31. for(loop = 1; loop <= 44; loop++) {
32. file_contents[0] = loop / 10 + 48;
33. file_contents[1] = loop % 10 + 48;
34. Mmc_Fat_Write(file_contents, LINE_LEN-1);  // შექმნილ ფაილში მონაცემთა ჩაწერა;
    }
  }
 }
```

// არსებული ფაილის გახსნა და მისი გასუფთავება:

```
35. void M_Open_File_Rewrite() {
36. filename[7] = 'C';                    // ფაილის სახელის დაყენება;
37. Mmc_Fat_Assign(&filename,0);
38. Mmc_Fat_Rewrite();
39. for(loop = 1; loop <= 55; loop++) {
40. file_contents[0] = loop / 10 + 65;
41. file_contents[1] = loop % 10 + 65;
42. Mmc_Fat_Write(file_contents, LINE_LEN-1);    // შექმნილ ფაილში მონაცემთა ჩაწერა;

 }

 }
```

// არსებული ფაილის გახსნა და მასში მონაცემთა დამატება;

```
43. void M_Open_File_Append() {
44. filename[7] = 'B';
45. Mmc_Fat_Assign(&filename, 0);
46. Mmc_Fat_Set_File_Date(2009, 1, 23, 17, 22, 0);
47. Mmc_Fat_Append();                    // ფაილის მომზადება მონაცემთა დასამატებლად;
48. Mmc_Fat_Write(" for mikroElektronika 2005\n", 27);   // ფაილში თარიღის ჩაწერა;
}
```

// არსებული ფაილის გახსნა, მისგან მონაცემთა წაკითხვა და UART-ზე გადაცემა

```
49.  void M_Open_File_Read() {
 50.   char character;
 51. filename[7] = 'B';
 52. Mmc_Fat_Assign(&filename, 0);
 53. Mmc_Fat_Reset(&size);       // ფაილის წაკითხვა, პროცედურა აბრუნებს ფაილის ზომას;
 54. for (i = 1; i <= size; i++) {
 55. Mmc_Fat_Read(&character);
 56. UART1_Write(character);                  // მონაცემთა ჩაწერა UART-ში;
   }
}
```

// ფაილის წაშლა, თუ ფაილი არ არსებობს იგი შეიძლება შევქმნათ და მერე წავშალოთ;

```
 57. void M_Delete_File() {
 58. filename[7] = 'F';
 59. Mmc_Fat_Assign(filename, 0);
 60. Mmc_Fat_Delete();
}
```

// როდესაც ფაილი არსებობს, იგ ზავნება მონაცემები და ფაილის ზომა UART-ის საშუალებით;

```
61. void M_Test_File_Exist() {
62. unsigned long  fsize;
63. unsigned int   year;
64. unsigned short month, day, hour, minute;
65. unsigned char  outstr[12];
66. filename[7] = 'B'
67. if (Mmc_Fat_Assign(filename, 0)) {

  //--- ფაილი მოძებნილია - მიღებულია მისი მონაცემები:

68. Mmc_Fat_Get_File_Date(&year, &month, &day, &hour, &minute);
69. UART1_Write_Text(" created: ");
70. WordToStr(year, outstr);
71. UART1_Write_Text(outstr);
72. ByteToStr(month, outstr);
73. UART1_Write_Text(outstr);
74. WordToStr(day, outstr);
75. UART1_Write_Text(outstr);
76. WordToStr(hour, outstr);
77. UART1_Write_Text(outstr);
78. WordToStr(minute, outstr);
79. UART1_Write_Text(outstr);

    //--- ფაილი მოძებნილია - მიღებულია შეცვლილი მონაცემები:

80. Mmc_Fat_Get_File_Date_Modified(&year, &month, &day, &hour, &minute);
81. UART1_Write_Text(" modified: ");
82. WordToStr(year, outstr);
83. UART1_Write_Text(outstr);
84. ByteToStr(month, outstr);
85. UART1_Write_Text(outstr);
86. WordToStr(day, outstr);
87. UART1_Write_Text(outstr);
88. WordToStr(hour, outstr);
89. UART1_Write_Text(outstr);
90. WordToStr(minute, outstr);
91. UART1_Write_Text(outstr);

    //--- ფაილის ზომის მიღება:

92. fsize = Mmc_Fat_Get_File_Size();
93. LongToStr((signed long)fsize, outstr);
94. UART1_Write_Line(outstr);
 }
 else {

  //--- ფაილი არ მოიძებნა:

95. UART1_Write(0x55);
```

```
 96. Delay_ms(1000);
 97. UART1_Write(0x55);
  }
}
// გაცვლის ფაილის შექმნა, რომლის მოცულობაა 100 სექტორზე მეტი:

 98. void M_Create_Swap_File() {
 99. unsigned int i;
 100.     for(i=0; i<512; i++)
  101. Buffer[i] = i;
   102. size = Mmc_Fat_Get_Swap_File(5000, "mikroE.txt", 0x20);
   103. if (size) {
   104. LongToStr((signed long)size, err_txt);
   105. UART1_Write_Line(err_txt);
   106. for(i=0; i<5000; i++) {
   107. Mmc_Write_Sector(size++, Buffer);
   108. UART1_Write('.');
    }

  }

}

// მთავარი პროგრამა:

   109. void main() {

      // UART1 მოდულის ინიციალიზაცია:

   110. UART1_Init(19200);
   111. Delay_ms(10);
   112. UART1_Write_Line("MCU-Started");              // MCU -ს დაწყების შეტყობინება:

  // SPI1 მოდულის ინიციალიზაცია:

   113. SPI1_Init_Advanced(_SPI_MASTER, _SPI_FCY_DIV128, _SPI_CLK_LO_LEADING);
   114.  if (Mmc_Fat_Init() == 0) {

      // SPI- ის რეინიციალიზაცია უფრო მაღალ სიჩქარეზე:

   115.    SPI1_Init_Advanced(_SPI_MASTER, _SPI_FCY_DIV2, _SPI_CLK_LO_LEADING);

    //--- ტესტის სტარტი:

   116. UART1_Write_Line("Test Start.");

    //--- Test routines.:

   117. M_Create_New_File();
   118. M_Create_Multiple_Files();
   119. M_Open_File_Rewrite();
```

```
  120. M_Open_File_Append();
  121. M_Open_File_Read();
  122. M_Delete_File();
  123. M_Test_File_Exist();
  124. M_Create_Swap_File();
  125. UART1_Write_Line("Test End.");
  }
  else {
  126. UART1_Write_Line(err_txt);

    }

}
```

პროგრამის დასაწყისში სრულდება მიკროკონტროლერისა და დისკგამწევის გამომყვანებს შორის კავშირის (სტრიქონი 1) და მიმართულების (სტრ. 2) გამოცხადება. სტრიქონებში 3-8 ცხადდებიან ცვლადები, რომლებიც შემდგომში გამოიყენებიან პროგრამის შესრულების დროს. სტრიქონ 9-ში გამოცხადებულია 512 ბაიტიანი ელემენტის შემცველი მასივი **Buffer**, რომლის ელემენტების რაოდენობა შეესაბამება დისკის სექტორის ზომას.

10-სტრიქონიდან აღწერილია ფუნქცია, რომელსაც გამოჰყავს ტექსტი USART ინტერფეისის საშუალებით (შესაძლებელია კომპიუტერში). ფუნქციის ტანი შეიცავს კომპილატორის ფუნქციებს **UART1_Write_Text()**, რომლის პარამეტრს წარმოადგენს გამოსაყვანი ტექსტი. სტრიქონ 11-ში სრულდება ტექსტის გამოყვანა, სტრიქონ 12 და 13-ში გაიცემა მიმდინარე სტრიქონის დამთავრებისა და ახალ სტრიქონზე გადასვლის კოდები. სტრიქონ 14-იდან აღწერილია ახალი ფაილის შექმნის და მასში მონაცემების ჩაწერის ფუნქცია. ფუნქციის დასაწყისში სრულდება ფაილის სახელის დაყენება (სტრიქონი 15). პუნქტ 16-ში იწერება ფაილის შექმნის თარიღი და დრო, რისთვისაც გამოყენებულია კომპილატორის ფუნქცია **Mmc_Fat_Set_File_Date(2005,6,21,10,35,0)**. ფუნქციის პარამეტრებია: წელი, დღე, თვე, საათი, წუთი, წამი. სტრიქონ 17-ში გამოყენებულია ფაილის დანიშვნის კომპილატორის ფუნქცია **Mmc_Fat_Assign (&filename, 0xA0)**. მისი პარამეტრებია ფაილის სახელი და ატრიბუტი. სტრიქონ 18-ში სრულდება ბიბლიოთეკის ფუნქცია **Mmc_Fat_Rewrite(),** რომლიც ასრულებს არსებული ფაილის მოძებნას და მის მომზადებას ჩასაწერად. თუ ფაილი არ არის ცარიელი, მისი შემცველობა იშლება. 19-23-ე სტრიქონებზე ციკლში სრულდება მონაცემთა ჩაწერა ფაილში 99-ჯერ. ციკლის თითოეულ გასვლაზე ფორმირდება მეხსიერების უჯრედებში ჩასაწერი მონაცემები, რომელიც განისაზღვრება ციკლის ყოველ ბიჯზე loop ცვლადის მნიშვნელობით. იგი წარმოდგენილი უნდა იყოს ASCII კოდში, ამიტომ მიღებულ მნიშვნელობას ემატება 48 და იწერება file_contents მასივის ორ ბაიტიან ელემენტში (სტრიქონები 21,22). დისკზე ჩაწერა ხორციელდება **Mmc_Fat_Write** ბიბლიოთეკის ფუნქციის საშუალებით (სტრ. 23), რომლის პარამეტრებია ჩასაწერი მონაცემები და ჩასაწერი მონაცემთა რაოდენობა.

24-34-ე სტრიქონებში განსაზღვრულია ფუნქცია, რომელიც ასრულებს რამდენიმე ფაილის შექმნას და მასში მონაცემების ჩაწერას. ფუნქცია სრულდება ციკლში. ციკლის ყოველ გასვლაზე განისაზღვრება ფაილის სახელი loop2 ცვლადით, რომლის შემცველობა

იცვლება A-დან Z-მდე და იწერება **FileName** მასივის მე-7 ელემენტში (სტრ. 27). თითოეული ფაილის შექმნა და მასში მონაცემების ჩაწერა სრულდება ზევით განხილული ფუნქციის ანალოგიურად.

35-44-ე სტრიქონებში აღწერილია ფუნქცია, რომელიც ხსნის ფაილს და წერს მასში მონაცემებს. გამოყენებულია ბიბლიოთეკის ფუნქციები, რომლებიც ადრე იყო აღწერილი.

43-48-ე სტრიქონებში აღწერილია ფუნქცია, რომლის საშუალებით არსებულ ფაილში სრულდება მონაცემთა დამატება. ზევით განხილულ ბიბლიოთეკის ფუნქციების გარდა გამოყენებულია ფუნქცია **Mmc_Fat_Append(),** რომელიც ფაილს ამზადებს მონაცემთა ჩასაწერად (უჯრედის მაჩვენებელს აყენებს ფაილის ბოლო ბაიტის შემდეგ, საიდანაც ჩაიწერება დამატებითი მონაცემები (სტრიქონ 47). სტრიქონ 48-ში სრულდება მონაცემთა ჩაწერის განხილული ბიბლიოთეკის ფუნქცია.

49-56-ე სტრიქონებში აღწერილია ფუნქცია, რომელიც ხსნის არსებულ ფაილს, კითხულობს მისგან მონაცემებს და გამოჰყავს ისინი USART ინტერფეისით. ფაილის გახსნის შემდეგ, რაც სრულდება 51,52-ე სტრიქონებში, 53-ე სტრიქონში **Mmc_Fat_Reset(&size)** ბიბლიოთეკის ფუნქციის საშუალებით ხდება მისამართის მაჩვენებლის დაყენება ფაილის დასაწყისში, რითაც მზადდება ფაილი წაკითხვისათვის. ფუნქციის პარამეტრს წარმოადგენს ფაილის ზომა. 54-ე სტრიქონიდან სრულდება ციკლი, რომლის ყოველ გასვლაზე **Mmc_Fat_Read(&character);** ბიბლიოთეკის ფუნქციის საშუალებით ფაილის უჯრედიდან **character** ცვლადში იკითხება მონაცემი (სტრ. 55). სტრიქონ 56-ში სრულდება ამოკითხული მონაცემის გაცემა USART ინტერფეისით (RS-232 გასართით). ამისთვის გამოყენებულია ბიბლიოთეკის ფუნქცია **UART1_Write(character).**

57-60-ე სტრიქონებში აღწერილია ფუნქცია, რომლის საშუალებით ხდება ფაილის წაშლა. დასაწყისში ვხსნით ფაილს ზევით განხილული ფუნქციების ანალოგიურად, ხოლო შემდეგ ბიბლიოთეკის ფუნქცია **Mmc_Fat_Delete()** შლის ფაილის შემცველობას.

61-97-ე სტრიქონებში აღწერილია ფუნქცია, რომელიც არსებულ ფაილიდან კითხულობს მონაცემებს და ფაილის ზომას UART ინტერფეისის საშუალებით.

67-ე სტრიქონში IF ოპერატორის მიერ მოწმდება მითითებული ფაილის არსებობა დისკზე. აღნიშნული ოპერატორის პარამეტრს წარმოადგენს ფუნქცია **Mmc_Fat_Assign(filename, 0),** რომელიც აბრუნებს ერთს ფაილის არსებობის შემთხვევაში და ნულს ფაილის არ არსებობის შემთხვევაში. ფაილის არსებობის შემთხვევაში მისგან ამოიკითხება შექმნის თარიღი და დრო **Mmc_Fat_Get_File_Date(&year, &month, &day, &hour, &minute)** ფუნქციის გამოყენებით. ამოკითხული მონაცემები, შესაბამისად, იწერება year, month, day, hour, minute ცვლადებში (სტრიქონი 68). შემდეგ სტრიქონებში თანამიმდევრობით თითოეული მონაცემი იწერება outstr ბუფერში (ფუნქციაWordToStr) და ამ ბუფერიდან მონაცემთა გაცემა USART ინტერფეისით (ფუნქცია **UART1_Write_Text(outstr)**).

80-91-ე სტრიქონებში სრულდება გახსნილი ფაილის მონაცემთა მოდიფიცირება (აღდგენა) და USART ინტერფეისით გამოყვანა. ამისათვის გამოიყენება ბიბლიოთეკის ფუნქცია **Mmc_Fat_Get_File_Date_Modified(&year, &month, &day, &hour, &minute)** და შემდეგ იმავე ფუნქციების თანამიმდევრობა, რაც ზემოთ იყო აღწერილი.

92-94-ე სტრიქონებში სრულდება ფაილის ზომის გამოყვანა USART ინტერფეისის საშუალებით. ამისათვის გამოიყენება ბიბლიოთეკის ფუნქცია **Mmc_Fat_Get_File_Size()**, რომლის შესრულების შედეგად **Size** ბუფერში ჩაიტვირთება ფაილის ზომა. შემდგომში ზევით განხილული ფუნქციების საშუალებით იგი გაიცემა USART ინტერფეისით.

თუ ფაილი არ მოიძებნა სრულდება 95-97-ე სტრიქონებზე მოთავსებული ბრძანებები, რომლის შედეგად USART ინტერფეისით გაიცემა კოდი 0x55.

98-108-ე პუნქტებში აღწერილია ფუნქცია, რომელიც ქმნის გაცვლის (**swap**) ფაილს. ფუნქციის პირველ სტრიქონებში ციკლში იწერება **BUFFER** მასივის ელემენტებში მონაცემები 0 დან 512 მდე. 102-ე სტრიქონში სრულდება ბიბლიოთეკის ფუნქცია **Mmc_Fat_Get_Swap_File(5000, "mikroE.txt", 0x20)**, რომლის დანიშნულებაა **swap** ფაილის შექმნა. ფუნქციის პირველი პარამეტრია სექტორების რაოდენობა, რომელიც უნდა დაეთმოს ფაილს, მეორე პარამეტრი არის ფაილის სახელი, ბოლო პარამეტრი- ფაილის ატრიბუტი. ფუნქცია აბრუნებს საწყისი ფაილის ნომერს, თუ დისკზე აღმოჩნდა საკმარისი ადგილი და 0 - თუ შესაქმნელი ფაილისათვის ცარიელი ადგილი დისკზე არ არის. 106-ე სტრიქონიდან სრულდება ციკლი, რომლის თითოეულ გასვლაზე სეგმენტში იწერება 512 ბაიტი **Buffer** მასივიდან. ციკლი სრულდება 500-ჯერ წინასწარ დაყენებული სექტორების რაოდენობის შესაბამისად, სექტორში მონაცემთა ჩასაწერად გამოყენებულია ბიბლიოთეკის ფუნქცია **Mmc_Write_Sector(size++, Buffer)**, რომლის პირველი პარამეტრია სექტორის ნომერი რომელშიც იწერება ცვლადები, მეორე პარამეტრი - ფაილის წყაროს დასახელება , რომლიდანაც იწერება ცვლადები.

109-ე სტრიქონიდან იწყება **main** ფუნქცია. უპირველეს ყოვლისა სრულდება USART1 პორტის ინიციალიზაცია, რისთვისაც გამოიყენება ბიბლიოთეკის ფუნქცია **UART1_Init(19200)** (სტრ. 110). ინიციალიზაცია მდგომარეობს გადაცემის სიჩქარის დაყენებაში, რომელიც მითითებულია ფრჩხილებში პარამეტრის სახით. გარდა ამისა ფუნქციის შესრულების დროს ავტომატურად ხდება USART ინტერფეისის სხვა პარამეტრების დაყენება: გადაცემის ნების დართვა, მიღების ნების დართვა, წყვილობაზე კონტროლის ნების დართვა, გადაცემა 8 ბიტიანი მონაცემებით, 1 STOP ბიტი, ასინქრონული რეჟიმი. 10 მწ-იანი დაყოვნების შემდეგ USART ინტერფეისიდან გაიცემა შეტყობინება პროგრამის შესრულების დაწყების შესახებ (სტრიქონი112). 113-ე სტრიქონში სრულდება SPI ბლოკის ინიციალიზაცია, რომლითაც მიკროკონტროლერი უკავშირდება დისკს. ამ მიზნით გამოიყენება ბიბლიოთეკის ფუნქცია **SPI1_Init_Advanced(_SPI_MASTER,_SPI_FCY_DIV128,_SPI_CLK_LO_LEADING**). ფუნქციის პარამეტრებს წარმოადგენენ მიკროკონტროლერის ბლოკის სტატუსი (წამყვანი- SPI_MASTER), ტაქტირების სიხშირე (128-ზე გაყოფილი -_SPI_FCY_DIV128), პოლარობის და ფაზის დაყენება (სატაქტო იმპულსების შუალედი დაბალი დონე, მონაცემთა ფიქსირება წინა ფრონტზე **(_SPI_CLK_LO_LEADING)**.

114-ე სტრიქონიდან იწყება სადემონსატრაციო პროგრამა. IF ოპერატორში მოწმდება დისკის არსებობა დისკგამწევში (ფუნქცია **Mmc_Fat_Init** აბრუნებს 0-ს დისკის აღმოჩენის შემთხვევაში და 1-ს თუ დისკი არ აღმოჩნდა). თუ დისკი იმყოფება დისკგამწევში, მიყოლებით სრულდება ადრე აღწერილი ფუნქციები. საწინააღმდეგო შემთხვევაში USART ინტერფეისიდან გაიცემა შეტყობინება შეცდომის შესახებ (სტრ. 126).

3.14 კოდური კლიტე

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

აიგოს ელექტრონული კოდური კლიტე, რომელზეც მიერთებულიაკოდის შესაყვანად ათი ღილაკი , აღნიშნული 0-დან 9-მდე. კლიტეს უნდა ჰქონდეს რეჟიმების გადამრთველი "ჩაწერა/მუშაობა". კოდის სწორად აკრეფის შემთხვევაში უნდა ჩაირთოს კლიტის შემსრულებელი მექანიზმი (სოლენოიდი ან ელექტრომაგნიტური გასაღები).

კლიტის მუშაობის პრინციპი შემდეგია: თავდაპირველათ რაიმე საკონტროლო კოდის ჩაწერა. კოდის ჩაწერის რეჟიმში მომხმარებელი აჭერს ღილაკებს ნებისმიერი თანამიმდევრობით და ნებისმიერი კომბინაციით (დასაშვებია ერთდროულად რამდენიმე ღილაკის დაჭერაც). მიკროკონტროლერი თვალყურს ადევნებს კლავიატურაზე ყველა ცვლილებას და შეაქვს ოდმ-ში. კოდური მიმდევრობის სიგრძე შეზღუდულია მხოლოდ ოდმ-ის ზომით. კოდის შეყვანის დასასრულის მაჩვენებლად მიღებულია კლავიატურასთან მანიპულაციის დამთავრება. ჩაითვლება, რომ მანიპულაცია დამთავრდა, თუ კლავიატურის მდგომარეობა არ შეიცვალა საკონტროლო დროის განმავლობაში. განხილულ ამოცანაში იგი მიღებულია ერთი წამის ტოლად. კოდის შეყვანის დამთავრების შემდეგ (საკონტროლო დროს გასვლის შემდეგ) მიკროპროცესორი ჩაწერს მიღებულ კოდს EEPROM-ში. კოდი წარმოადგენს ბაიტების თანამიმდევრობას, რომელიც ასახავს კლავიატურის ყველა მდგომარეობას დაჭერის მომენტში. მას შემდეგ, რაც კოდი იყო ჩაწერილი, კლიტე შესაძლებელია გადაყვანილი იყოს მუშა რეჟიმში. ამისათვის გათვალისწინებულია რეჟიმის ამორჩევის სპეციალური ტუმბლერი.

მუშა რეჟიმში კლიტე ელოდება კოდის შეყვანას. კარების გაღებისათვის საჭიროა იგივე მანიპულაციები ღილაკებზე, რაც შესრულებული იყო ჩაწერის რეჟიმში. მიკროკონტროლერი, ისევე, როგორც ზევით განხილულ შემთხვევაში თვალყურს ადევნებს აღნიშნულ მანიპულაციებს და წერს მიღებულ კოდს ოდმ-ში. კოდის შეყვანის დამთავრების შემდეგ (საკონტროლო დროს გასვლის შემდეგ) პროგრამა გადადის ოდმ-ში ჩაწერილი კოდის შედარებაზე ადრე EEPROM-ში ჩაწერილ კოდთან. პირველად ედარება კოდების სიგრძეები. შემდეგ კოდები ედარებიან ერთმანეთს ბაიტ-ბაიტ. თუ შედარება დამთავრდა წარმატებით, მიკროკონტროლერი გასცემს სიგნალს კლიტის გახსნის მექანიზმს.

ალგორითმი

ალგორითმის შედგენისათვის საჭიროა განვსაზღვროთ ,,კლავიატურის მდგომარეობის" ცნება. კლავიატურის ღილაკები მიერთებულია მიკროკონტროლერთან პორტების საშუალებით. ათი ღილაკის მიერთებისათვის (0-9) ერთი პორტი არ არის

საკმარისი (პორტს აქვს რვა გამომყვანი). რამოდენიმე ღილაკის მისაერთებლად უნდა გამოვიყენოთ დამატებით მეორე პორტი.

კონტროლერი კითხულობს ამ პორტების შემცველობას და იღებს კოდს, რომელიც შეესაბამება მათ მდგომარეობას. ყოველ ღილაკს ამ კოდში შეესაბამება ერთი თანრიგი. როდესაც ღილაკი დაჭერილია შესაბამის თანრიგში იწერება ნული. როცა აშვებულია - ერთიანი. ამის გამო დაჭერილი და აშვებული ღილაკების სხვადასხვა კომბინაციის დროს კლავიატურის მდგომარეობის კოდს ექნება სხვადასხვა მნიშვნელობა. კვების წყაროსთან მიერთების მომენტში ყველა ღილაკი უნდა იყოს აშვებული. საწინააღმდეგო შემთხვევაში კლიტის მუშაობაში წარმოიშობა გაურკვევლობა. ამის გამო, ალგორითმი უნდა იწყებოდეს ყველა ღილაკის აშვებულ მდგომარეობაში გადასვლის მოლოდინის პროცედურით. როგორც კი ყველა ღილაკი აღმოჩნდება აშვებული ან საერთოდ არ იყვნენ დაჭერილი, იწყება მოლოდინის სხვა პროცედურა. ამჯერად პროგრამა ელოდება ღილაკების დაჭერის მომენტს. ნებისმიერი ღილაკის დაჭერის მომენტიდან იწყება საწყისი კომბინაციის შეყვანის ციკლი.

საწყისი კომბინაციის შეყვანის პროცედურა წარმოადგენს მრავალჯერ განმეორებად პროცესს, რომლის დროსაც პერიოდულად იკითხება კლავიატურის მდგომარეობის კოდი. ყოველთვის კოდის მორიგი ამოკითხვის შემდეგ პროგრამა ამოწმებს, შეიცვალა თუ არა კოდი. როგორც კი კოდი შეიცვლება, მისი შემდეგი მნიშვნელობა იწერება ოდმ-ის მეზობელ უჯრედში. სანამ კლავიატურის მდგომარეობა არ იცვლება, პროგრამა იმყოფება მოლოდინის რეჟიმში.

ოდმ-ში ჩაწერილი საწყისი კომბინაცია წარმოადგენს კლავიატურის მდგომარეობის კოდის ყველა მნიშვნელობების ჩამონათვალს, რომელსაც იგი იღებდა საწყისი კომბინაციის შეყვანის დროს. კლავიატურის მდგომარეობის შეცვლის აღმოჩენის შემთხვევაში პროგრამა მყისიერად არ ინახავს შესაბამის კოდურ კომბინაციას ოდმ-ში. კონტაქტების ვიბრაციასთან ბრძოლის მიზნით და აგრეთვე რამდენიმე ღილაკზე დაჭერის უზუსტობის კომპენსაციის მიზნით პროგრამა ჯერ ასრულებს დამცველ პაუზას, შემდეგ განმეორებით კითხულობს კლავიატურის მდგომარეობის კოდს და მხოლოდ ამის შემდეგ ჩაწერს ოდმ-ში.

დამცავი პაუზის ხანგრძლიობა არჩეულია 48 მწმ. ასეთი ხანგრძლიობა განსაკუთრებით მიზანშეწონილია იმ შემთხვევაში, თუ კოდური კომბინაციის აკრეფის შემთხვევაში გამოიყენება რამდენიმე ღილაკის ერთდროული დაჭერა. როგორც არ უნდა ვეცადოთ ღილაკების ერთდროულად დაჭერას, ამის შესრულება პრაქტიკულად შეუძლებელია. კონტაქტები ჩართვის მომენტებს შორის იქნება განსხვავება. კონტაქტების ჩართვის მიმდევრობა დამოკიდებულია სხვადასხვა ფაქტორზე და პრაქტიკულად არის შემთხვევითი. თუ არ იქნება მიღებული სპეციალური ზომები, მაშინ ერთი დაჭერის განმავლობაში პროგრამა დააფიქსირებს არა ერთ, არამედ კლავიატურის მდგომარეობის ცვლილების რამდენიმე კოდს. ასეთი გზით მიღებული კოდის კომბინაციის EEPROM-ში ჩაწერით კლიტის გახსნა პრაქტიკულად შეუძლებელია. იმავე დაჭერის გამეორების შემთხვევაში კონტაქტების ჩართვის თანმიმდევრობა იქნება სხვა. პროგრამა აღიქვამს მას როგორც სხვა კოდს. რა მიმდევრობითაც არ უნდა ჩაირთონ კონტაქტები რამდენიმე ღილაკის ერთდროულად დაჭერის შემთხვევაში, პაუზის შემდეგ ყველა პროცესი

მთავრდება. განმეორებითი წაკითხვა იძლევა უკვე დამყარებული კლავიატურის მდგომარეობას.

გარდა დამცავი პაუზისა, ვიბრაციის წინააღმდეგ გამოიყენება მდგომარეობის კოდის მრავალჯერადი წაკითხვა. ანუ ერთი და იგივე კოდის წაკითხვა ხორციელდება მანამდის, სანამ ერთმანეთის მიყოლებით რამდენჯერმე არ მიიღება ერთი და იგივე კოდი.

როგორც ზემოთ იყო ნათქვამი, საწყისი კოდის შეყვანის დამთავრების დასაფიქსირებლად გამოიყენება დროის დამცავი შუალედი. ამ შუალედის დასაფორმირებლად ვირჩევთ ტაიმერს. ტაიმერი მუშაობს ნორმალ რეჟიმში. ამ რეჟიმში იგი მხოლოდ ითვლის სატაქტო სიგნალებს. კოდური კომბინაციის შეყვანის პროცედურა ითვალისწინებს ნებისმიერი ღილაკის ყოველი დაჭერის ან აშვების შემთხვევაში ტაიმერის განულებას. დაჭერის შორის შუალედში მისი შემცველობა იზრდება ყოველ სატაქტო სიგნალზე. თუ დამცავი დროის განმავლობაში არ იქნება დაჭერილი არც ერთი ღილაკი, ტაიმერის მნიშვნელობა გაიზრდება საკონტროლო ზღვრამდე. პროგრამა მუდმივად ამოწმებს აღნიშნულ პირობას. როგორც კი მთვლელის შემცველობა გადააჭარბებს საკონტროლო ზღვარს, კოდური კომბინაციის შეყვანა მთავრდება. დროის საკონტროლო შუალედის მნიშვნელობა ტოლია 1წმ-ის.

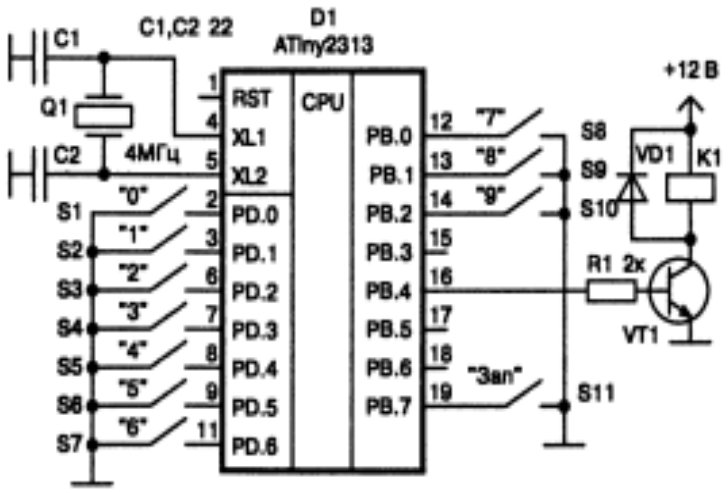
საწყისი კოდური კომბინაციის შეყვანის დამთავრების შემდეგ მოწმდება მუშაობის რეჟიმის განმსაზღვრელი გადამრთველის მდგომარეობა. თუ გადამრთველის კონტაქტები შეერთებულია, მაშინ პროგრამა გადადის EEPROM-ში კოდის კომბინაციის ჩაწერის პროცედურაზე. ჯერ EEPROM-ში იწერება კოდური კომბინაციის სიგრძე, შემდეგ კოდური კომბინაციის ბაიტები ერთმანეთის მიმდევრობით. თუ რეჟიმების გადამრთველის კონტაქტები გათიშულია, პროგრამა გადადის კოდების შემოწმების პროცედურაზე. ეს პროცედურა კითხულობს EEPROM-ში ადრე ჩაწერილ კოდურ კომბინაციის სიგრძეს და ადარებს მას ახლად შეყვანილ კოდის სიგრძეს. თუ ეს სიდიდეები არ არიან ტოლი, კოდების შემოწმება მთავრდება უარყოფითი შედეგით. თუ ორივე კომბინაციის სიგრძე ერთნაირია, პროგრამა იწყებს მათი ბაიტ-ბაიტ შედარებას. ამისათვის იგი მიმდევრობით კითხულობს EEPROM-დან ბაიტებს და ადარებს ოდმ-ში ჩაწერილ შესაბამის ბაიტებს. პირველივე არათანხვედრის შემთხვევაში შედარების პროცესი სრულდება უარყოფითი შედეგით. თუ ყველა ბაიტები ოდმ-ში და EEPROM-ში აღმოჩნდენ ერთნაირი, შედარება ჩაითვლება წარმატებული. პროგრამა გადადის კლიტის გახსნის პროცედურაზე. გახსნის პროცედურა იწყება გახსნის სიგნალის გაცემით შემსრულებელ მექანიზმზე. პროგრამა ასრულებს დაყოვნებას 2წმ ხანგრძლიობით და შემდეგ სიგნალი იხსნება. ეს დრო საკმარისია კარის გაღებისათვის. შემდეგ პროგრამა გადადის საწყის მდგომარეობაში.

სისტემის სტრუქტურა

კლიტის შესაძლებელი ვარიანტი მოცემულია სურ.3.30-ზე. S1-S10 ღილაკები განკუთვნილია კოდის ასაკრეფად. S11 გადამრთველის საშუალებით შესაძლებელია მუშაობის რეჟიმების არჩევა. თუ S11 გადამრთველის კონტაქტები შეერთებულია კლიტე გადადის "ჩაწერის" რეჟიმში. გათიშული კონტაქტები შეესაბამება "მუშაობის" რეჟიმს.

კლიტის მექანიზმის მართვის სქემა შესდგება TV1 ტრანზისტორული გასაღებისა და K1 ელექტრომაგნიტური რელესაგან. R1 რეზისტორი ზღუდავს გასაღების ბაზის დენს.

VD1 დიოდი ემსახურება თვითინდუქციისაგან დაცვას, რომელიც წარმოიქმნება რელეს გრაგნილებზე. რელეს კვება ხორციელდება 12- ვოლტიანი კვების წყაროდან.



სურ. 3.30

მოყვანილ სქემაში PD.0-PD.6, PB.0-PB.2 და PB.7 პორტების გამომყვანები მუშაობენ როგორც შესასვლელები, PB.4 - როგორც გამოსასვლელი.

მიკროკონტროლერის მუშაობის პროგრამა C ენაზე

```
1. #define klfree 0x77F ;    // მდგომარეობის კოდი ყველა აშვებული ღილაკის შემთხვევაში;
2. #define kzad 3000 ;       // დაყოვნების კოდი სკანირების დროს;
3. #define kandr 20;         // კონსტანტა ანტივიბრაციისთვის;
4. #define bsize 30;         // კოდის ბუფერის ზომა;
5. unsighned char flz;     // დაყოვნების ალამი;
6. unsighned int bufr[bsize] ;      //ბუფერი ოდმ-ში კდის შენახვისთვის;
7. eeprom unsighned char klen;   // კოდის სიგრძის შესანახი უჯრედი;
8. eeprom unsighned int bufe[bsize]; //ბუფერი EEPROM-ში კოდის შესანახად;

//წყვეტა ტაიმერ1-დან

9. interapt [TIM1_OVF] void timer 1_ovf_isr (void);
    {

10. fiz=1;
    }

// წყვეტა ტაიმერ1-ის A-რეგისტრთან თანხვედრის დროს:

11. interapt [TIM1_COMPA] void timer1_compa_isr (void);
    {
12.  fiz=1;
```

```
    }
// კლავიატურის გამოკითხვის და ვიბრაციის ფუნქცია:
13. unsighned   int  incod (void);
     {
14. unsighned   int  cod0=0;   // დამხმარე ცვლადი
15. unsighned   int  cod1;      // დამხმარე ცვლადი
16. unsighned   char  k;        // ანტივიბრაციის ციკლის პარამეტრი
17. for (k=0; k<kandr; k++)       // ანტივიბრაციის ციკლი
     {
18. cod1=PINB&0x7;         // ვაფორმირებთ კოდის პირველ ბაიტს
19. .cod1=(cod1<<8)+(PIND&0x7F); //ვაფორმირებთ კლავიატურის მდგომარეობის სრულ
//კოდს
20. if (cod0!=cod1)              //ვადარებთ საწყის კოდს
   {
21. k=0;              // უტოლობის შემთხვევაში ვასუფთავებთ მთვლელს
22. cod0=cod1;                  //  საწყისი კოდის ახალი მნიშვნელობა
       }
     }
23.  return  cod1
}
// დაყოვნების ფორმულების პროცედურა
24.  void  wit  (unsighned   char  kodz)
     {
25. if  (codz== 1)  TIMSK= 0x40;   //ტაიმერიდან წყვეტის ნიღბის ამორჩევა
26. else  TIMSK= 0x80;
27. TCNT1=0;                        //  ტაიმერის განულება
28. Flz=0;                          //  დაყოვნების ალამის განულება
29. #asm("sei");                    //წყვეტის ნების დართვა
30. If (kodz!=2)  while(flz==0);       //  დაყოვნების ციკლი
     }
//ძირითადი ფუნქცია
31.  void  main  (void)
    {
32. unsighned   char  ii;  //მასივის მაჩვენებელი
33. unsighned   char  I;    // დამხმარე მაჩვენებელი
34. unsighned   int  codS; //ძველი კოდი
35. PORTB=0xE7;
36. DDRB=0x10;
37. PORTD-0x0x7F;
38. DDRD=0x00;
```

```
39. TCCR1A=0x00;
40. TCCR1B=0x03;;
41. OCR1A=kzad ;            //  თანხვედრის რეგისტრის ინიციალიზაცია;
42. ACSR=0x80;              //  ანალოგური კომპარატორის გამორთვა;
43. while (1)
     {

44. m1:  while(icod() !=klfree); //ღილაკის აშვების მოლოდინი;
45.    while (icode() ==klfree);  //ღილაკის დაჭერის მოლოდინი;
46.          ii=0;
47. m2:  #asm("cli");             // წყვეტის აკრძალვა;
48.          wait (1);             //პირველი ტიპის დაყოვნება;
49.          codS=incod();          // კოდის შეყვანა როგორც ძველის;
50.          bufr (ii++)=codS;      // მორიგი კოდის ჩაწერა ბუფერში;
51.          If (ii>=bsize)  goto  m4  //ბუფერის დასასრულის შემოწმება;
52.          wait(2);               //  მეორე ტიპის დაყოვნება;
53. m3:    if (incod!= codS)  goto  m2;  // მდგომარეობის ცვლილების შემოწმება;
54.          if(flz==0)   goto m3;          //  დაყოვნების ალამის შემოწმება;

        // კოდის ჩაწერა EEPROM-ში

55.      klen=ii;                          //  კოდის სიგრძის ჩაწერა;
56.          For (i=0;i<ii; i++ )   bufe  (i)=bufr(i); //  კოდის ყველა ბაიტის ჩაწერა;
57.          goto  zamok;

//კოდის შემოწმება

58. Comp:  if(klen0 != ii)  goto  m1;      //კოდის სიგრძის შემოწმება;
59.              For (:=0; i<ii; i++)  if (bafe[i]=bafr[i])  goto  m1;  //  კოდის შემოწმება;

//კლიტის გახსნა:

60. zamok:  PORTB 4=1;     //ვხსნით კლიტეს;
61.              wait  (3);  //   მესამე ტიპის დაყოვნება;
62.              PORTB 4= 0;   //  ვხურავთ კლიტეს;
                }

              }
```

პროგრამა იწყება კონსტანტების აღწერის ბლოკით (სტრ. 1-4). ამისათვის გამოიყენება დირექტივა **#define**, რომლის პირველი პარამეტრია კონსტანტის სახელი, მეორე პარამეტრი კონსტანტა, რომელსაც იგი ცვლის. პირველ კონსტანტას აქვს სახელი **klfree** და მნიშვნელობა 0x77F. ეს კონსტანტა ასახავს კლავიატურის მდგომარეობის კოდს აშვებულ ღილაკების შემთხვევაში. შემდეგი სამი კონსტანტა: **kzad** –დაყოვნების მნიშვნელობა, **kndr** - ანტივიბრაციის კონსტანტა და **bsize** – კოდური კომბინაციის სიგრძე.

5-8 სტრიქონებში აღწერილია ცვლადები და მასივები. 5 სტრიქონში აღწერილია ცვლადი **flz** , რომელიც გამოიყენება როგორც დაყოვნების ალამი. 6 სტრიქონში

აღწერილია მასივი საწყისი კოდური კომბინაციის დროებითი შენახვისათვის ოდმ-ში. მასივის სიგრძე განისაზღვრება **bsize** მნიშვნელობით.

7-8 პუნქტებში აღწერილია ცვლადი და მასივი, რომლებიც ჩაიწერება EEPROM-ში. ცვლადი **klen** განკუთვნილია კოდური კომბინაციის სიგრძის შესანახად. მასივი **bufe** - თვით ამ კომბინაციის შესანახად. იმისათვის, რომ ტრანსლატორს მიეთითოს ცვლადისათვის გამოყოფილი უჯრედების ადგილმდებარეობა, ამ ცვლადის აღწერაში გამოყენებული უნდა იყოს მმართველი სიტყვა **eeprom.**

შემდეგ, პროგრამაში იწყება ყველა მის შემადგენელი ფუნქციების აღწერა. მოცემული პროგრამა შეიცავს ხუთი ფუნქციას:

- წყვეტის მომსახურების ორი ფუნქცია (სტრ. 9,10 და 11,12) ;
- კლავიატურის მდგომარეობის კოდის შეყვანის ფუნქცია (სტრ. 13-23);
- დაყოვნების ფორმირების ფუნქცია (სტრ. 24-30);
- პროგრამის მთავარი ფუნქცია (სტრ. 31-64).

პროგრამის განხილვა მოხერხებულია დავიწყოთ მთავარი main ფუნქციით. ფუნქცია main იწყება ლოკალური ცვლადების აღწერით (სტრ. 32-35). გარდა codS ცვლადისა, რომელიც განკუთვნილია კლავიატურის მდგომარეობის კოდის დროებით შესანახად, აქვე განსაზღვრულია კიდევ ორი დამხმარე ცვლადი სახელებით i და ii. ცვლადების აღწერის შემდეგ იწყება ინიციალიზაციის ბლოკი (სტრ. 32-43). ეს მოქმედება მდგომარეობს შეყვანა-გამოყვანის პორტების, ტაიმერისა და კომპარატორის გაწყობაში. პორტების გაწყობა მდგომარეობს მათი გამომყვანების მიმართულების განსაზღვრაში. ამ მიზნით DDRB რეგისტრში იწერება 0x10 მნიშვნელობა (B პორტის 0-3 და 7 გამომყვანი შესასვლელია, 4 გამომყვანი-გამოსასვლელი). ვინაიდან PD პორტის ყველა გამომყვანი შესვლაზე უნდა იყოს გაწყობილი, DDRD რეგისტრში ვწერთ 0x00 მნიშვნელობას (სტრ. 36,38). მთვლელის გაწყობის დროს გარდა ტაიმერის მუშაობის რეჟიმის არჩევისა (სტრ. 39,40), განულდება მთვლელი რეგისტრის TCNT1 მნიშვნელობა (სტრიქონი 41) და შედარების რეგისტრში OCR1A იწერება დაყოვნების კოდი kzad (სტრ. 42).

პროგრამის ძირითადი ციკლი სრულდება სტრიქონებში 43-64. 45 სტრიქონში სრულდება ღილაკის აშვების მოლოდინის ციკლი. იგი წარმოადგენს ცარიელ while ციკლს. ციკლს არ აქვს ტანი. ამის გამო ფიგურული ფრჩხილების აუცილებლობაც არ არის. მთელი ციკლი შედგება მხოლოდ while. ოპერატორისა და ფრჩხილებში მოთავსებული გამოსახულებისაგან, რომელიც განსაზღვრავს ციკლის გაგრძელების პირობას. ციკლი სრულდება მანამ, სანამ კლავიატურის მდგომარეობის კოდი და klfree კონსტანტა არ არის ერთმანეთის ტოლი. კონსტანტის მნიშვნელობა წარმოადგენს კლავიატურის მდგომარეობის კოდს ყველა აშვებული ღილაკის შემთხვევისათვის. კლავიატურის მდგომარეობის კოდის განსაზღვრისათვის გამოიყენება ფუნქცია incod(). ფუნქცია incod() კითხულობს პორტების მდგომარეობას და იყენებს ანტივიბრაციის ალგორითმს. დაწვრილებით ფუნქციების მუშაობას განვიხილავთ ამ პარაგრაფის ბოლოს.

როგორც კი ყველა ღილაკი იქნება აშვებული, ციკლი (სტრ. 45) დასრულდება, და პროგრამა გადადის 46 სტრიქონზე, სადაც სრულდება ნებისმიერი ღილაკის დაჭერის მოლოდინის ციკლი. ეს ციკლი გავს წინას. შეიცვალა მხოლოდ პირობა. ნიშანი != (არა

ტოლი) შეიცვალა ნიშნით == (ტოლი). ღილაკის დაჭერის შემთხვევაში ციკლი მთავრდება, მართვა გადადის შემდეგ 47 სტრიქონზე.

აღნიშნული სტრიქონიდან იწყება საწყისი კოდური კომბინაციის შეყვანის ციკლი. თავდპირველად ვანულებთ ცვლადს ii, რომელიც შემდგომში გამოყენებული იქნება როგორც მიღებული კოდების მთვლელი და მიმდინარე ელემენტის ნომერი bufr ბუფერში. ციკლი სრულდება 48-55 სტრიქონებში. ციკლი იწყება ყველა წყვეტის გლობალური აკრძალვით Y (სტრიქონი 48). შემდეგ ფორმირდება დამცველი პაუზა. (სტრიქონი 49). პაუზის ფორმირებისათვის გამოიყენება ფუნქცია wait, რომელიც აფორმირებს პირველი ტიპის დაყოვნებას (48მწ). მას აქვს პარამეტრი ერთი. დაყოვნების დამთავრების შემდეგ პროგრამა განმეორებით კითხულობს კლავიატურის მდგომარეობის კოდს და იწერება ცვლადში codS.

51 სტრიქონში წაკითხული კოდი იწერება ოდმ-ის ბუფერში (bufr). ერთდროულად ბუფერის მაჩვენებელი იზრდება ერთით. სტრიქონ 52-ში მოწმდება ბუფერის გადავსება. აღნიშნული შემოწმება წყვეტს ციკლის შესრულებას გაცილებით დიდი კოდური კომბინაციის შეყვანის მცდელობის გამო. თუ ii მნიშვნელობამ გდააჭარბა bsize კონსტანტის სიდიდეს, ეს ნიშნავს, რომ ბუფერი მთლიანად შევსებულია. ამ შემთხვევაში მართვა გადადის m4 ნაჭდევზე, ანუ ციკლის ბოლოს.

თუ ii მნიშვნელობა არ აღწევს ბუფერის ბოლოს, მაშინ პროგრამა გადადის პუნქტ 53-ზე, სადაც სრულდება ფუნქცია wait(). მოცემულ შემთხვევაში იგი აფორმირებს მეორე ტიპის დაყოვნებას, ამიტომ აქვს პარამეტრი ორი. 54 სტრიქონში სრულდება კლავიატურის მდგომარეობის ახალი კოდის წაკითხვა და ძველ კოდთან შედარება, რომელიც ინახება codS ცვლადში. თუ კოდები არ არიან ერთმანეთის ტოლი (კლავიატურის მდგომარეობა შეიცვალა), დაყოვნების ციკლი წყდება და პროგრამა გადადის m2 ნაჭდევზე. ე. ი. კოდური კომბინაციის შეყვანის ციკლის დასაწყისზე. აქ ისევ ფორმირდება დამცველი პაუზა და მორიგი კოდი იწერება ბუფერში. თუ 54 სტრიქონში შედარების შედეგად ახალი და ძველი კოდი აღმოჩნდა ტოლი, გადასვლა არ ხდება და სრულდება სტრიქონი 55. ამ პუნქტში მოწმდება flz ალმის მნიშვნელობა. თუ საკონტროლო დრო არ არის დამთავრებული, ალმის მნიშვნელობა ნოლის ტოლია და პროგრამა გადადის m3 ნაჭდევზე. ციკლი გრძელდება.

თუ საკონტროლო დრო ამოიწურა, flz ალამი იღებს ერთის მნიშვნელობას, აქ ღილაკის დაჭერის მოლოდინის ციკლი და კოდური კომბინაციის შეყვანის ციკლი მთავრდება. პროგრამა გადადის 56 სტრიქონის შესრულებაზე.

ამ სტრიქონში მოწმდება S11 ტუმბლერის მდგომარეობა. მისი მდგომარეობის მიხედვით სრულდება ან მიღებული კოდური კომბინაციის ჩაწერა EEPROM-ში, ან წაკითხული კოდური კომბინაციის შედარება EEPROM-ში ჩაწერილ კოდურ კომბინაციასთან. გადამრთველის მდგომარეობის გასარკვევად მოწმდება PB პორტის მეშვიდე თანრიგი (რომელზეც მიერთებულია გადამრთველი). თუ გადამრთველის კონტაქტები გათიშულია (PINB.7 ერთის ტოლია), პროგრამა გადადის comp ნაჭდევზე (კოდების შედარების პროცედურაზე). საწინააღმდეგო შემთხვევაში სრულდება კოდის ჩაწერა EEPROM-ში. EEPROM-ში კოდის ჩაწერა სრულდება 57-59 სტრიქონებში. 57 სტრიქონში კოდური კომბინაციის სიგრძე იწერება klen ცვლადში. ვინაიდან klen ცვლადის აღწერის დროს

მის ადგილმდებარეობად მითითებული იყო EEPROM, ცვლადის შემცველობა ავტომატურად ჩაიწერება მასში. C ენა თითონ ასრულებს ამისათვის საჭირო ყველა პროცედურას. 58 სტრიქონში სრულდება ციკლი, რომლის დროსაც ხდება თვით კოდური კომბინაციის ჩაწერა EEPROM-ში. ციკლის ყოველ გასვლაზე bufr მასივის ელემენტი იწერება bufe მასივის შესაბამის ელემენტში. ციკლის პარამეტრს წარმოადგენს i ცვლადი. ციკლის შესრულების დროს ამ ცვლადის მნიშვნელობა იცვლება ნულიდან ii-მდე. ე. ი. გადაისინჯება კოდური კომბინაციის ყველა ელემენტის ნომერი (ii ტოლია მისი სიგრძის). ციკლის ტანი შეიცავს მხოლოდ ერთ გამოსახულებას, რომელიც სწერს Bufer ბუფერის მორიგი ელემენტის მნიშვნელობას bufe ბუფერში. ჩაწერის ციკლის დამთავრების შემდეგ პროგრამა გადადის 59 სტრიქონზე რომელშიც სრულდება zamok ნაჭდევზე უპირობო გადასვლის ოპერატორი. ანუ კლიტის გახსნის პროცედურა.

60,61 სტრიქონებში სრულდება შედარების პროცედურა. ამ პროცედურის დასაწყისისათვის ii ცვლადი შეიცავს ახლად შეყვანილ კოდურ კომბინაციის სიგრძეს, ხოლო ცვლადი bufr - თვით ამ კომბინაციას. თავდაპირველად 60 პუნქტში ედარება klen (EEPROM-ში ჩაწერილი სიგრძე) ii ცვლადის მნიშვნელობას.

თუ შემოწმების შედეგად ეს ორი სიდიდე აღმოჩნდა განსხვავებული, პროგრამა გადადის m1 ნაჭდევზე. ე. ი. პროგრამის დასაწყისზე. თუ ორივე მნიშვნელობა აღმოჩნდა ერთნაირი პროგრამა გადადის კოდური კომბინაციების შედარებაზე (სტრიქონი 61). იგი სრულდება ციკლში, რომლის პარამეტრია i. ციკლი შესდგება if ოპერატორისაგან. ეს ოპერატორი ადარებს ორი მასივის ელემენტებს, ერთ-ერთი მათგანი bufr იმყოფება EEPROM-ში, მეორე bufe - ოდმ-ში. პირველივე არატოლობის შემთხვევაში პროგრამა გადადის m1 ნაჭდევზე. თუ ყველა კოდი დაემთხვა, ციკლი მთავრდება ნორმალურად და პროგრამა გადადის 52 სტრიქონზე, ანუ კლიტის გაღების პროცედურაზე.

კლიტის გახსნის პროცედურა სრულდება 62-64 სტრიქონებში. 62 სტრიქონში PB პორტის მეოთხე თანრიგში იწერება ერთი. შედეგად მის გამოსასვლელზე ფორმირდება მაღალი პოტენციალი, რაც ხსნის მასთან დაკავშირებულ ელექტრონულ გასაღებს. გასაღების კოლექტორულ წრედში ჩართულ სოლენოიდში გაივლის დენი და კლიტის ჩამკეტი მიიზიდება. კლიტე იხსნება. მიზიდულ მდგომარეობაში ჩამკეტი რჩება გარკვეული დროის განმავლობაში, რასაც უზრუნველყოფს დაყოვნების wait ფუნქცია (სტრიქონი 63). 64 სტრიქონში PB პორტის მეოთხე თანრიგში იწერება ნული. შედეგად მის გამოსასვლელზე ფორმირდება დაბალი პოტენციალი, ელექტრონული გასაღები იკეტება, სოლენოიდში დენი აღარ გადის და ჩამკეტი უბრუნდება მის საწყის მდგომარეობას (კლიტე იკეტება). კლიტის გაღების პროცედურის დამთავრებასთან ერთად მთავრდება პროგრამის ძირითადი ციკლი. ვინაიდან ძირითადი ციკლი უსასრულოა მართვა გადაეცემა მის დასაწყისს (სტრ. 44). პროგრამის შესრულება იწყება თავიდან.

დაუბრუნდეთ დამხმარე ფუნქციებს, რომლებიც აღწერილი არის პროგრამის დასაწყისში. 9,10 და 11.12 სტრიქონებში განთავსებულია ორი სხვადასხვა დასახელების და ერთნაირი შინაარსის ფუნქცია. პირველი ფუნქცია წარმოადგენს ტაიმერ/მთვლელის გადავსების წყვეტის მომსახურების პროცედურას, მეორე – იმავე ტაიმერის A რეგისტრთან

თანხვედრის წყვეტის მომსახურების პროცედურას. თითოეული ფუნქცია შეიცავს ერთ ოპერატორს, რომელიც ანიჭებს flz ცვლადს ერთის მნიშვნელობას.

13-23 სტრიქონებში მოთავსებულია კლავიატურის მდგომარეობის შეყვანის ფუნქცია incod. 14-16 სტრიქონებში აღწერილია ლოკალური ცვლადები. cod0 და cod1 ცვლადები გამოიყენება კლავიატურის მდგომარეობის კოდების შუალედური მნიშვნელობების შესანახად. ამასთან cod1 ცვლადი გამოიყენება კოდის ახალი მნიშვნელობის შესანახად, cod0–ძველი მნიშვნელობის შესანახად. კიდევ ერთი ცვლადი k გამოიყენება, როგორც პარამეტრი ანტივიბრაციის ციკლში.

ფუნქციის ძირითად ნაწილს წარმოადგენს ანტივიბრაციის ციკლი (სტრიქონი 17-22). ციკლის დანიშნულებაა კლავიატურის მდგომარეობის კოდის რამდენიმეჯერ ამოკითხვა (განსაზრვრული kandr კონსტანტით). მოცემულ შემთხვევაში გამოიყენება for სტანდარტული ციკლი. ციკლი სრულდება მთლიანად იმ შემთხვევაში, თუ მისი შესრულების განმავლობაში კლავიატურის მდგომარეობის კოდი არ იცვლება. თუ ციკლის შესრულების პროცესში ღილაკების მდგომარეობა იცვლება, ციკლი თავიდან მეორდება. ანტივიბრაციის ციკლის დამთავრების შემდეგ სრულდება ოპერატორი return რომელიც განსაზღვრავს ფუნქციის მიერ დაბრუნებულ მნიშვნელობას.

როგორც ზემოთ იყო ნათქვამი, 18,19 სტრიქონებში სრულდება კლავიატურის მდგომარეობის კოდის ფორმირება. გამოთვლების წყაროს წარმოადგენს PINB და PIND რეგისტრების შემცველობა, რომლებიც უშუალოდ დაკავშირებული არიან PB და PD პორტების გამომყვანებთან. ორივე რეგისტრის შემცველობაზე გამოიყენება ნიღბები შესაბამისი თანრიგების გამოსაყოფად და შემდეგ ერთიანდება ერთ თექვსმეტთანრიგა cod1 ცვლადში. 20 სტრიქონში ეს მნიშვნელობა ედარება კოდის ამოკითხულ ,,ძველ" მნიშვნელობას, რომელიც ინახება cod0 ცვლადში. თუ კოდები ერთმანეთის ტოლი არ აღმოჩნდა, ციკლი იწყება თავიდან. k ცვლადს ენიჭება ნოლის მნიშვნრელობა (სტრიქონი 21). 23 სტრიქონში cod1 ცვლადის მნიშვნელობა გადაიწერება cod0 ცვლადში და ახლად მიღებული კოდი ხდება ,,ძველი".

24-30 სტრიქონებში წარმოდგენილია დაყოვნების ფორმირების ფუნქცია. ფუნქცია არ აბრუნებს რაიმე მნიშვნელობას, მაგრამ აქვს შესასვლელი პარამეტრი- დაყოვნების ტიპის კოდი. ფუნქცია wait აფორმორებს დაყოვნების სამ ტიპს. დაყოვნების ტიპის განმსაზღვრელი პარამეტრი არის codz. ფუნქციის შესრულება იწყება წყვეტის ნიღბის განსაზღვრით. ამ მიზნით 25,26 სტრიქონებში ირკვევა kodz ცვლადის მნიშვნელობა. თუ kodz მნიშვნელობა არის 1, წყვეტის ნიღბის რეგისტრში TIMASK იწერება კოდი 0x40. ეს კოდი ნებას რთავს ტაიმერიდან წყვეტას A რეგისტრთან თანხვედრის შემთხვევაში.

თუ kodz არ არის ერთის ტოლი, მაშინ TIMASK რეგისტრს ენიჭება მნიშვნელობა 0x80 (წყვეტა ტაიმერის გადავსების შემთხვევაში). ამგვარად, რეჟიმ 2-ში წყვეტა სრულდება ტაიმერის გადავსების შემთხვევაში. ამასთან ფორმირდება 48 მწ-ის ხანგრძლიობის დაყოვნება. სხვა რეჟიმში (3) ფორმირდება დაყოვნება ერთი წამის ხანგრძლიობით.

27 სტრიქონში ნულდება ტაიმერის სათვლელი რეგისტრი, ტაიმერის განულების მომენტიდან იწყება შესაბამისი დროითი ინტერვალის ფორმირება. 38 სტრიქონში ნულდება დაყოვნების ალამი. 39 სტრიქონში სრულდება წყვეტის გლობალური ნების

დართვის ოპერატორი. ამით ტაიმერისა და წყვეტის სისტემის გაწყობა მთავრდება, პროგრამა გადადის მოლოდინის ციკლის შესრულებაზე. მოლოდინის რეჟიმი გათვალისწინებულია 1 და 3 რეჟიმებში სამუშაოდ. (სტრ. 30). ეს არის ცარიელი ციკლი, რომელიც ორგანიზებულია while ოპერატორის საშუალებით. ციკლის გაგრძელების პირობაა flz ალმის ნულთან ტოლობა. ანუ სანამდის flz ნულის ტოლია, მანამდის მოლოდინი გრძელდება. დამთავრდება იგი, როცა წყვეტის მომსახურების პროცედურა არ შეცვლის ალმის მნიშვნელობას ერთიანით.

რეჟიმ 2-ის დროს გამოიყენება მოლოდინის სხვა ციკლი, რომელიც იმყოფება while ოპერატორის გარეთ. მოლოდინის ციკლის გარდა 30 სტრიქონში არის აგრეთვე შედარების ოპერატორი if. იგი ამოწმებს kodz ცვლადის მნიშვნელობას. თუ kodz ცვლადის მნიშვნელობა არის 2, მაშინ დაყოვნების ციკლი არ სრულდება და პროგრამა გადადის მთავარი ფუნქციის შესრულებაზე.

3.15 მრავალპარამეტრიანი სისტემის დაგეგმარება

ზემოთ მოყვანილი მაგალითებში განხილული იყო მიკროკონტროლერის გამოყენება ერთ პარამეტრის მქონე ობიექტების მართვისათვის. ახლა საილუსტრაციოთ განვიხილოთ მრავალ პარამეტრიანი პროცესის მართვა მიკროპროკონტროლერის საშუალებით.

ჩამოვაყალიბოთ ამოცანა:

დავაგეგმაროთ სისტემა, რომელიც სხვადასხვა პარამეტრებზე დამოკიდებით ასრულებს ნარგავების ავტომატური მორწყვის სხვადასხვა რეჟიმს.

პროექტის მომზადების ეტაპზე დადგენილი უნდა იყოს პირობები, რომელებსაც აკმაყოფილებს დასაგეგმარებელი სისტემა. ეს პირობები განსაზღვრავენ მორწყვის რეჟიმს, რაც მდგომარეობს შემდეგი პარამეტრების დადგენაში: მორწყვის დროითი ინტერვალი და გაფრქვეული წყლის რაოდენობა მოსარწყავი ფართის ერთეულზე. თავის მხრივ აღნიშნული პარამეტრები განისაზღვრება სხვა მნიშვნელოვანი მახასიათებლებით, როგორიც არის მორწყვის ხანგრძლიობა, ატმოსფეროს ტემპერატურა და ნიადაგის ტენიანობა. ცხადია, აღნიშნული მახასიათებლები სხვადასხვა ჯიშის მცენარეებისათვის სხვადასხვაა. მოყვანილ მაგალითში განხილულია გაზონის მორწყვის სისტემა. საცნობარო ლიტერატურის საფუძველზე დადგენილია მორწყვის პარამეტრები: გაზონის მორწყვის პერიოდულობა რეკომენდირებულია ერთხელ დღეში დილის ან საღამოს საათებში ატმოსფეროს ნორმალური ტემპერატურის შემთხვევაში და ორჯერ დღეში გვალვიან პერიოდში; მორწყვის ხანგრძლიობამ უნდა უზრუნველყოს 10-20 ლიტრი წყლის გაფრქვევა 1 მ^2-ზე; გათვალისწინებული უნდა იყოს ნიადაგის ტენიანობა - ნორმა განისაზღვრება 70%-ით, ვინაიდან მისი გადაჭარბება იწვევს ნიადაგის დაჭაობებას და არ არი სასურველი მცენარის ფესვებისათვის.

მორწყვის სისტემის დაგეგმარებისათვის მნიშვნელოვანია სხვადასხვა პარამეტრებზე ნორმების დაცვა, რომლებიც შეიძლება მოვძებნოთ შესაბამის საცნობარო ლიტერატურაში. კერძოთ, სხვადასხვა ტიპის გამფქვევის მახასიათებლები: წარმადობა და მათი დამოკიდებულება მილში წყლის წნევაზე, წყლის გაფრქვევის რადიუსი, წნევის კარგვა

მილებში, გამფრქვევების ოპტიმალური განაწილება მოსარწყავ ფართზე და სხვა. აღნიშნული მახასიათებლების საფუძველზე ამოირჩევა გამფრქვევის ტიპი.

მორწყვის მართვა ხორციელდება სარქველის საშუალებით, რომლის გახსნა და დაკეტვა განსაზღვრავს მორწყვის ხანგრძლიობას და დამოკიდებულია მილში წყლის ჭავლის წნევაზე - ნორმალური წნევის შემთხვევაში მორწყვის ხანგრძლიობამ უნდა უზრუნველყოს მცენარისთვის ნორმით გათვალისწინებული წყლის რაოდენობის მიღება. მილში დაბალი წნევის შემთხვევაში მორწყვის ხანგრძლიობა უნდა გაიზარდოს იმავე რაოდენობის წყლის გაფრქვევისათვის. ამგვარად, მორწყვის ხანგრძლიობის გათვლა უნდა მოხდეს მილში წყლის წნევისა და გამფრქვევის წარმადობის გათვალისწინებით. მორწყვის პერიოდი აგრეთვე დამოკიდებულია გარემოს ტემპერატურასა და ნიადაგის ტენიანობაზე.

სისტემის ცენტრალურ ნაწილს წარმოადგენს მიკროკონტროლერი Atmega 128, რომლის გამომყვანებთან მიერთებულია პერიფერიული მოწყობილობები: რეალური დროის მთვლელი, ატმოსფეროს ტემპერატურის, ნიადაგის ტენიანობის, წყლის წნევის სენსორები და წყლის სარქველი. მიკროკონტროლერის არჩევა აიხსნება მისი მახასიათებლებით: შედარებით დიდი ტევადობის მეხსიერება, პერიფერიული მოწყობილობისა და ინტერფეისების ფართე ნომენკლატურა.

რეალური დროის საათი

მორწყვის დროის განსაზღვრისათვის გამოყენებულია რეალური დროის საათის ინტეგრალური სქემა DS 1307. (აღნიშნული სქემა დაწვრილებით განხილული იყო 3.9 პარაგრაფში). როგორც იყო ნათქვამი, ინტეგრალური სქემა უკავშირდება მიკროკონტროლერს I^2C (TWI) ინტერფეისის საშუალებით, რაც აადვილებს სქემის მიკროკონტროლერთან დაკავშირების სქემურ გადაწყვეტას.

ტემპერატურის სენსორი

ჰაერის ტემპერატურის მიშვნელობის გასაზომად ვიყენებთ ტემპერატურის სენსორს DS18B20, რომელის პარამეტრები აკმაყოფილებს მოცემულ პროექტის მოთხოვნას (განხილული იყო 3.8 პარაგრაფში).

ტენიანობის სენსორი

წყალი წარმოადგენს მცენარის კვების ერთ-ერთ ძირითად კომპონენტს, ამის გარდა, მცენარის ფესვების სისტემა ეფექტურად მუშაობს ნიადაგში ტენიანობის გარკვეული შემცველობის შემთხვევაში. ნიადაგის განსაზღვრული ტენიანობა (მცენარის ყოველი კულტურის, ყოველი სახეობისთვის ეს მაჩვენებელი განსხვავდება) – ერთ-ერთი მნიშვნელოვანი მოთხოვნაა მცენარის არსებობისათვის. თუ ნიადაგში არის ტენიანობის ნაკლებობა ან სიჭარბე მცენარე შეიძლება დაიღუპოს, პირველ შემთხვევაში ფესვების გამოშრობისაგან, მეორე შემთხვევაში - მათი დალპობისაგან. მის გამო მცენარეთა მორწყვის წინ უნდა შემოწმებული იყოს ნიადაგის ტენიანობა.

ავტომატური მორწყვის სისტემებში ნიადაგის ტენიანობის გასაზომად გამოიყენება ტენიანობის სენსორი. თუ ნიადაგი საკმარისად ტენიანია, მაშინ მორწყვა უნდა შეწყდეს, ან და არ დაიწყოს თუ იყო დაგეგმილი.

სენსორის ამორჩევისათვის ყურადღება უნდა მიექცეს შემდეგ პარამეტრებს:

- კვების ძაბვა;
- გამოსასვლელი სიგნალის მნიშვნელობის დიაპაზონი;
- გაზომვის ცდომილება;
- ერთი გაზომვის დრო.

ჩვენ მიერ ამორჩეულია ნიადაგის ტენიანობის სენსორი DFROBOT, იგი იდეალურად გამოდგება ბაღის მონიტორინგისათვის.

სენსორს აქვს შემდეგი მახასიათებლები:

- გამოსასვლელი სიგნალის ძაბვა: 0-4,2ვ;
- მოხმარებული დენი: 35 მამპ;
- კვების ძაბვა: +3,3ვ ან +5ვ;
- ანალოგური ინტერფეისი;
- ერთი გაზომვის დრო: 50მწამ.

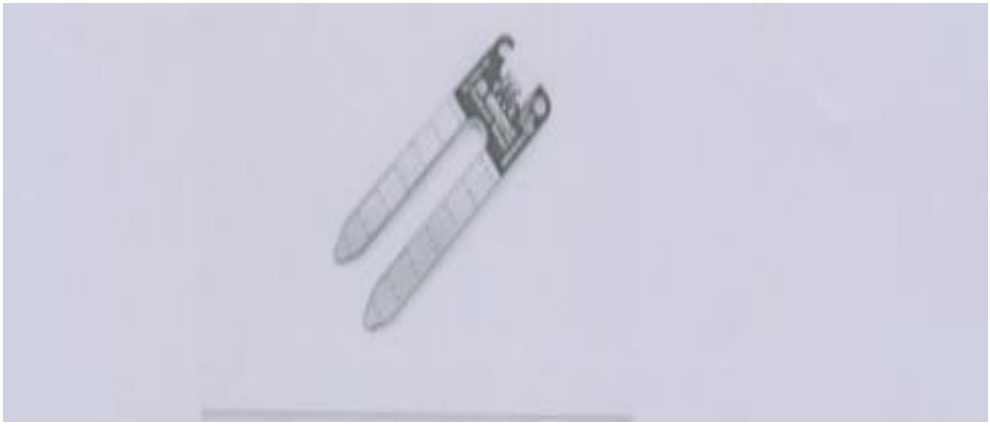
10-ბიტიან დიაპაზონზე გამოსასვლელი სიგნალის მნიშვნელობის ასახვა.

0-300: მშრალი ნიადაგი;

300-700: ტენიანი ნიადაგი;

700-950:სენსორი წყალშია.

სურ.3.31 ნაჩვენებია DFROBOT სენსორის კონსტრუქცია.



სურ. 3.31

ტენიანობის სენსორის გამოსასვლელიდან მიკროკონტროლერს მიეწოდება ანალოგური სიგნალი (ძაბვა), რომელიც ასახავს ტენიანობის მნიშვნელობას. ტენიანობის სენსორიდან მიწოდებული ანალოგური სიგნალის შემდგომი დამუშავებისათვის საჭიროა იგი გადაისახოს ციფრულ ფორმაში მიკროკონტროლერ Atmega128-ის შემადგენლობაში მყოფ ანალოგურ – ციფრულ გარდამსახში (აცგ). ამ მიზნით სენსორი უკავშირდება PC1 პორტს (ამ პორტის გამომყვანები გამოიყენება როგორც აცგ-ს შესასვლელები).

წყლის წნევის სენსორი

ბაღის მორწყვის დროს დიდი მნიშვნელობა აქვს გაფრქვეული წყლის მოცულობას და სიმძლავრეს. მცენარეთა ყოველ სახეობას აქვს ამ პარამეტრების თავისი ნორმა. თავის მხრივ ხსენებული მაჩვენებლები დამოკიდებულია მილში წყლის წნევაზე, რომელიც შეიძლება იცვლებოდეს სხვადასხვა მიზეზის გამო. კერძოთ, რამდენიმე მომხმარებლის მიერ ერთდროული მოხმარების გამო. წნევის შემცირების შემთხვევაში მორწყვა უნდა შესრულდეს უფრო ხანგრძლივად, ვიდრე ნორმალური წნევის შემთხვევაში. წყლის წნევა იზომება ბარებში (1 ბარი ტოლია 1 ატმოსფეროსი). წყლის წნევის ცვლილების დიაპაზონი ქალაქის ქსელში ან სააგარაკო ნაკვეთებზე, როდესაც სისტემა მუშაობს დამაკმაყოფილებლად შეადგენს 1,5-5 ბარს. ზემოთ თქმულიდან გამომდინარე, ავტომატური მორწყვის სისტემის დაგეგმარებისას უნდა გათვალისწინებული იქნას წყლის წნევა მისი გაზომვით წნევის სენსორის საშუალებით და კორექცია გაუკეთდეს მორწყვის ხანგრძლიობას.

წნევის სენსორების გამოყენების სფერო საკმაოდ ფართეა, მაგრამ, როგორც წესი ყოველ კონკრეტულ გამოყენებას აქვს თავისი სპეციფიკა, რომელიც უნდა გათვალისწინებული იყოს სენსორის კონსტრუქციაში.

წნევის სენსორის არჩევის დროს რეკომენდებულია შემდეგი პარამეტრები:

- გასაზომი გარემოს ტიპი;
- გაზომვის დიაპაზონი;
- პროცესის ტემპერატურა;
- სენსორის მილთან შეერთების ტიპი;
- სენსორის გამოსასვლელი სიგნალის ტიპი;
- გაზომვის საჭირო სიზუსტე.

ჩამოთვლილი პარამეტრებიდან ჩვენი მიზნისათვის არსებითია: გაზომვის დიაპაზონი, რომელიც უნდა იყოს 1,5-5 ბარი. სენსორის მილთან შეერთების ტიპი ამჟამად ყველაზე პოპულარული არის ხრახნული შეერთება G1/2", DIN16288, M20x1,5. გამოსასვლელი სიგნალის მიხედვით გამოიყენებიან წნევის სენსორები ანალოგური გამოსასვლელი სიგნალებით: 0-1ვ; 0-5ვ; 0-10ვ; ჩვენი სისტემისათვის ავირჩიოთ ფირმა WIKA-ს წნევის სენსორი S-10, რომლის პარამეტრები სრულად აკმაყოფილებენ ჩვენი ამოცანის მოთხოვნებს (იხ. სურ. 3.32)

სურ. 3.32

წყლის მილის საქველი

ავტომატური მორწყვის სისტემის ერთ-ერთ საკვანძო ელემენტს წარმოადგენს ელექტრომაგნიტური სარქველი.

სისტემის დაგეგმარების დროს ყურადღება ექცევა ისეთ ფაქტორებს, როგორიც არის ტერიტორიის მორწყვის ტიპი, ლანშაფტის თავისებურება და წყლის საწყისის ადგილმდებარეობა. ეს ყველაფერი აისახება სისტემის კომპონენტების, მათ შორის ელექტრომაგნიტურ სარქვლის არჩევაში. მორწყვის ავტომატური სისტემის ელექტრომაგნიტური სარქვლის ერთ-ერთ ძირითად ამოცანას წარმოადგენს წყლის ჩართვა /გამორთვა. მორწყვის სისტემის სარქველი იმართება მიკროკონტროლერიდან, რომელიც წინასწარ დაპროგრამებულია განსაზღვრულ რეჟიმში მუშაობაზე, ამასთან დაკავშირებით კონკრეტული მოდელის არჩევის დროს ყურადღება უნდა მიექცეს მათი მახასიათებლების შეთავსებას.

განვიხილოთ მორწყვის სისტემის ელექტრომაგნიტური სარქვლების ელექტრო მახასიათებლები:

- **დენის სახეობა** მუდმივი და ცვლადი; მორწყვის სისტემის მუდმივი დენის მაგნიტური სარქველის სოლენოიდი გამოიყენება, როგორც წესი, მიკროპროცესორთან, რომელიც იკვებება ავტონომიური კვების წყაროდან;
- **ძაბვის დონე.** პარამეტრი, რომელიც განსაზღვრავს სოლენოიდის მუშა ძაბვას. ძაბვის გაზრდას მივყავართ სოლენოიდის გამოყვანას მწყობრიდან, შემცირება, კი ვერ უზრუნველყოფს სარქვლის ნორმალურ ფუნქციონირებას. იმის გამო, რომ ავტომატური მორწყვის სისტემის ფუნქციონირება გამიზნულია ტენიან გარემოში, სარქვლის მართვის სიგნალები უმეტეს შემთხვევაში არიან დაბალვოლტიანი, რაც ადვილი ასახსნელია უსაფრთხოების თვალსაზრისით.
- გაშვების დენი (სასტარტო დენი) და დაჭერის დენი. გაშვების ეწოდება დენს, რომელიც გამოიყენება სოლენოიდის მიერ კომუტაციის შემთხვევაში მცირე დროის მონაკვეთში, რის შემდეგ დენის მოხმარება მცირდება დაჭერის დენის დონემდე.
- სარქვლის მექანიკური მახასიათებლები:

ავტომატური მორწყვის სიტემის ელქტრომაგნიტური სარქვლის ერთ-ერთი მნიშვნელოვანი პარამეტრი არის მისი გამტარუნარიანობა. იგი ასახავს ხარჯვის დონის დამოკიდებულებას სარქველში წნევის კარგვასთან ეს მომენტი უნდა იყოს გათვალისწინებული ავტომატური მორწყვის სისტემის დაგეგმარების დროს.

- შეერთების ტიპები. ელქტრომაგნიტური სარქვლის შეერთება შესაძლებელია დავყოთ მილტუჩა, ხრახნულ და ჯაგრისული. ავტომატური მორწყვის სისტემაში გამოიყენება ელექტრომაგნიტური სარქველი ხრახნული შეერთებით.
- მინიმალური და მაქსიმალური მუშა წნევა. ელექტრომაგნიტური სარქვლის მინიმალური მუშა წნევა არის მწარმოებლის მიერ განსაზღვრული მნიშვნელობა, რომელიც უზრუნველყოფს კვანძის ნორმალურ ფუნქციონირებას. მაქსიმალური მუშა წნევა კი არის დონე, როდესაც სარქველს შეეძლება ფუნქციონირება ექსპლოატაციის მთელი დროის

განმავლობაში გიდრავლიკური გადატვირთვის გარეშე. ეს მნიშვნელობა აგრეთვე რეგლამენტირდება მწარმოებლის მიერ.

- მდგომარეობა უმოქმედობის დროს. განასხვავებენ ნორმალურად გახსნილ და ნორმალურად ჩაკეტილ სარქველებს. ნორმალურად ჩაკეტილი არის სარქველი, რომელიც მართვის სიგნალის არ არსებობის დროს იმყოფება ჩაკეტილ მდგომარეობაში, ნორმალურად გახსნილი კი პირიქით ატარებს სითხეს. ავტომატური მორწყვის სისტემებში საჭიროა, რომ მართვის სიგნალების არ არსებობის შემთხვევაში სარქველი იყოს ნორმალურად ჩაკეტილი.

სპეციალისტების რეკომენდაციების გათვალისწინებით აღნიშნული პროექტისათვის ყველაზე უფრო შესაფერისი არის ფირმა Hunter-ის ელექტრო დინამიკური სარქვლი PGV-100 (სურ.3.32).

სურ. 3.32

სარქველს აქვს შემდეგი საექსპლოატაციო მახასითებლები:

- გამტარუნარიანობა: 0,7დან 454.2 ლ/წმ;
- წნევა:1,4 დან 10,3-მდე ბარ;
- სოლენოიდის სიმძლავრე: 24ვ ; ცვლადი დენი; გაშვების დენი 370მა, დაჭერის დენი 190მა.

სისტემის სქემაში სარქველი მიერთებულია მიკროკონტროლერის PD0 გამომყვანზე.

წყლის გამფრქვევი (სპრინკლერი)

მორწყვის სისტემაში წყლის გამფრქვევის არჩევას ენიჭება დიდი მნიშვნელობა. გამფრქვევის ამორჩევის დროს განსაკუთრებული ყურადღება უნდა მიექცეს შემდეგ მახასიათებლებს: მორწყვის რადიუსი და კუთხე; წყლის წნევის დიაპაზონი მილებში, რომლის დროსაც სრულდება ნორმალური მორწყვა; წყლის ხარჯი m^3/s..

გამფრქვევი არის მორწყვის თანამედროვე მოწყობილობა, რომლის უპირატესობა კლასიკურ მორწყვასთან (შლანგით) შედარებით, წარმოადგენს წყლის თანაბარი განაწილება და შესაძლებლობა ,,დაიმალოს" მიწაში.

არსებობს გამფრქვევის 3 ძირითადი ტიპი, რომელიც გამოიყენება სხვადასხვა ლანშაფტის შემთხვევაში:

როტორული სპინკლერი. გამიზნულია დიდი ღია ტერიტორიის მორწყვისათვის. მორწყვა ხორციელდება წყლის ჭავლის წრიული მოძრაობით. როტორული სპინკლერის ყველა მოდელს აქვს ადვილად შეცვლადი საქშენის ნაკრები, რომლის საშუალებით სრულდება ავტომატური მორწყვის რადიუსის რეგულირება.

ვეერული სპინკლერი განკუთვნილია გაზონის მცირე ზომის ნაკვეთების , ახალგაზრდა ხეების, და დეკორატიული ბუჩქების მოსარწყავად. წყლის ჭავლის მაქსიმალური რადიუსი - 5 მეტრი, მინიმალური-1,5. ვეერული გამრფქვევის მუშა წნევის დიაპაზონი 1,5-4,5 ატმოსფერო.

როტატორი - ეს არის მძლავრი გამრფქვევი, რომლის საშუალებით მორწყვა ხორციელდება წვრილი სხვადასვა მიმართულების მქონე ჭავლების საშუალებით. მისი გაფრქვევის შუალედია 3 მდან 9,1 მმდე.

ზემოთ განხილული გამრქვევებიდან ჩვენი სისტემისათვის ყველაზე შესაფერისია Hunter ფირმის გამრფქვევი PS-Ultra (სურ. 3.33).

სურ. 3.33

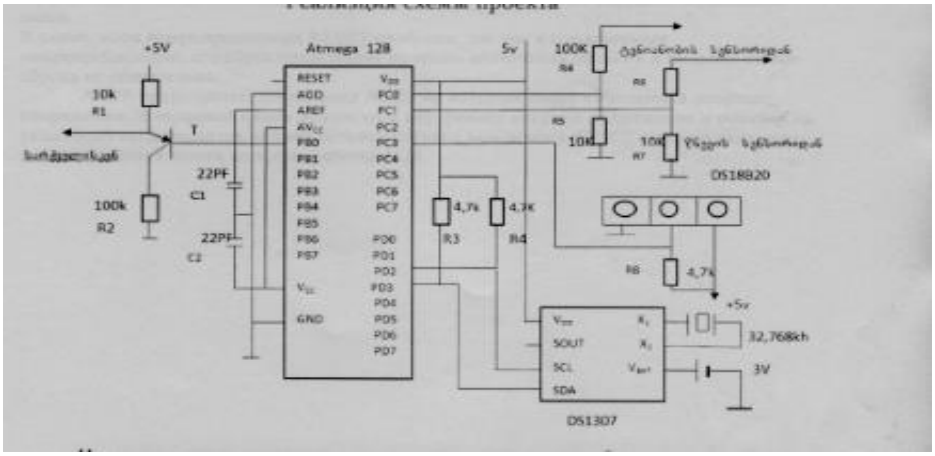
გამფქვევის თავაკს აქვს რადიუსისა და მორწყვის ნორმის რეგულირების შესაძლებლობა, ლანდშაფტის შესაბამისად. მორწყვის სექტორი 1 დან 360 გრადუსამდე. წყლის რეკომენდებული წნევა 1,40- 4,80 ბარი. მორწყვის რადიუსი განისაზღვრება დაყენებული თავაკით და შეიძლება იცვლებოდეს 3 დან 5 მეტრამდე (2 ატმოსფეროს წნევის შემთხვევაში); მორწყვის ინტენსივობა- 0,7 მ^3/ს-მდე.

რეალური დროის ფიქსირებისათვის გამოყენებულია ინტეგრალური სქემა DS1307 (რეალური დროის საათი-RTC), რომელიც მიერთებულია მიკროკონტროლერთან PD0,PD1 გამომყვანებით. მონაცემთა გადაცემა მიკროკონტროლერსა და RTC მოდულს შორის სრულდება TWI (I2C) ინტერფეისის პროტოკოლით.

სურ. 3.34-ზე მოყვანილია განხილული პროექტის სქემური რეალიზაცია. მორწყვის სისტემის ცენტრალურ ნაწილს წარმოადგენს მიკროკონტროლერი Atmega 128, რომელთანაც მიერთებულია პერიფერიული მოწყობილობები. ამ მიზნით გამოიყენება.

PB,PC და PD პორტები. კერძოდ, PB0 გამომყვანი კონფიგურირებულია როგორც გამოსასვლელი, რომელთანაც მიერთებულია წყლის სარქველი. თუ შესაბამის თანრიგში

პროექტის სქემური რეალიზაცია



სურ. 3.34

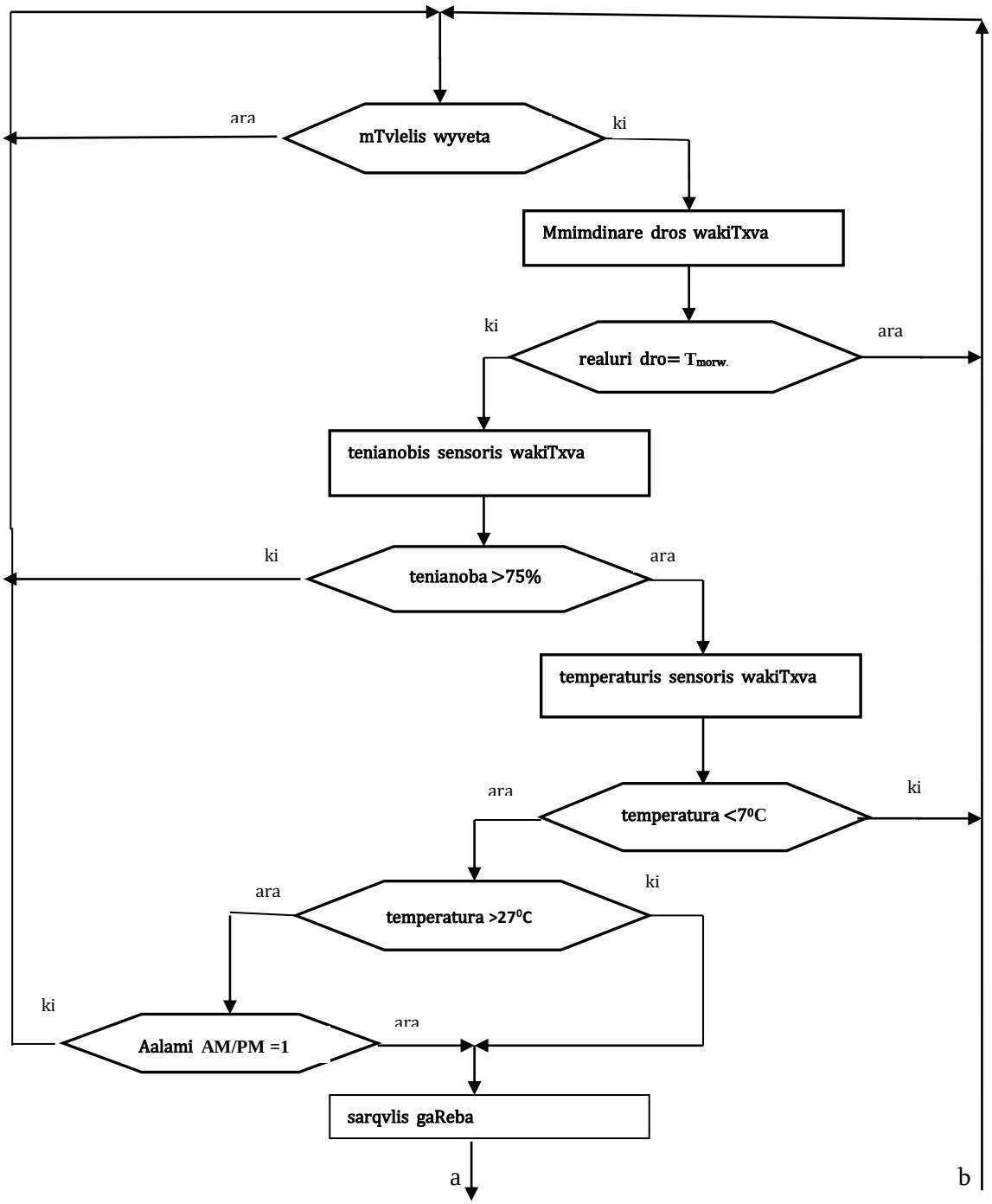
ჩაიწერება ერთიანი, მაშინ სარქველს მიეწოდება მაღალი პოტენციალი, რომლითაც იგი გაიღება, ხოლო ნულის ჩაწერის შემთხვევაში - სარქველს მიეწოდება დაბალი პოტენციალი და იგი იკეტება. ვინაიდან სარქვლის ჩართვისათვის საჭიროა დიდი დენი (570 მა), აღნიშნულ გამოსასვლელზე დაყენებულია გასაღები (სხვა ვარიანტში შესაძლებელია რელეც), რომელიც უზრუნველყოფს დენის საჭირო მნიშვნელობას.

PC0 და PC1 გამომყვანები დაკონფიგურირებულია როგორც შესასვლელები და დაკავშირებულია ტენიანობის და წნევის სენსორებთან, შესაბამისად. აღნიშნული შესასვლელები მულტიპლექსორის საშუალებით უკავშირდება აცგ-ს, რომლის საშუალებით სრულდება სენსორებიდან მოწოდებული ანალოგური სიგნალების გარდასახვა დისკრეტულ ფორმატში. სენსორებიდან მიწოდემული სიგნალების (0-10ვ) დასაყვანად მიკროკონტროლერის შესასვლელებისთვის დასაშვებ მნიშვნელობამდე, გამოყენებულია ძაბვის გამყოფები R4-R5 და R6-R7, რის შედეგად მიკროკონტროლერს მიეწოდება მნიშვნელობა 3,5ვ.

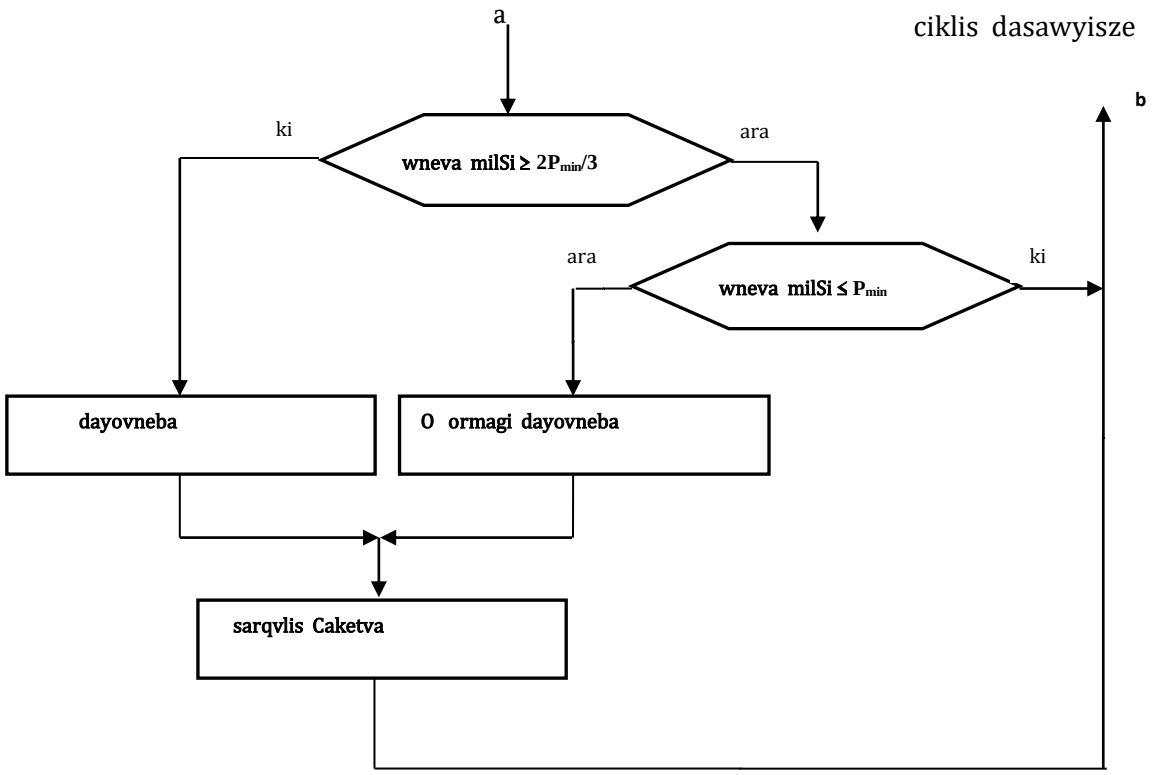
PC3 გამომყვანი დაკავშირებულია DS18B20 ტემპერატურის სენსორთან. როგორც ზემოთ იყო აღნიშნულია მისი კავშირი მიკროკონტროლერთან სრულდება ერთგამტარიანი 1-wire ინტერფეისის პროტოკოლით. მიკროკონტროლერის RESET გამოსასვლელი თავისუფალია, ვინაიდან მიკროპროცესორის განულება სრულდება მისი კვების წყაროსთან მიერთების დროს და ამიტომ განულების შესასვლელის გამოყენება არ არის აუცილებელი.

AREF წარმოადგენს აცგ-ს საყრდენი ძაბვის გარე შესასვლელს. მოყვანილ სქემაში გამოიყენება შიგა საყრდენი ძაბვა. ამის გამო აღნიშნულ შესასვლელზე მიეწოდება დაბალი პოტენციალი (შესასვლელი გამორთულია).

სისტემის მუშაობის ალგორითმი



სურ. 3.35



სურ. 3.35 (გაგრძელება)

სურ. 3.35-ზე მოცემულია ავტომატური მორწყვის სისტემის მუშაობის ალგორითმის გრაფ-სქემა. ალგორითმში რეალიზებულია ის პრინციპები, რომლებიც განხილული იყო ზევით. მუშაობის დასაწყისში სრულდება სისტემის ინიციალიზაცია, რაც მდგომარეობს მიკროკონტროლერის პორტებში მონაცემთა გადაცემის მიმართულების განსაზღვრაში, მისი წამყვან მოწყობილობად გამოცხადებაში TWI ინტერფეისისათვის და სხვ.

მორწყვა უნდა განხორციელდეს დღის გარკვეულ საათებში, რომლის მნიშვნელობა T_{morw} წინასწარ შეიტანება პროგრამაში. დროს ათვლას ახორციელებს რეალური დროს საათის მოდული, რომლიდანაც მიმდინარე დროის მნიშვნელობა პერიოდულად გამოიკითხება პროგრამის მიერ. გამოკითხვის პერიოდს განსაზღვრავს მიკროკონტროლერში შემავალი ერთ-ერთი 16 თანრიგა მთვლელი, რომლის თვლის პერიოდი (მაქსიმალური თვლის დრო) განსაზღვრავს რეალური დროის საათთან მიკითხვის მომენტს. მთვლელის ციკლის ბოლოს ფორმირდება წყვეტის სიგნალი და მიკროკონტროლერი გადადის სამომსახურებო პროგრამის შესრულებაზე, რომელიც

ასრულებს მოდულის გამოკითხვის ფუნქციას. ამოკითხული მნიშვნელობის თანხვედრის დროს მორწყვის დროსთან იწყება მორწყვის პროცედურა, საწინააღმდეგო შემთხვევაში პროგრამა გადადის ციკლის დასაწყისში და რეალური დროის საათის გამოკითხვა მეორდება. პროცესი იწყება ნიადაგის ტენიანობის შემოწმებით. თუ ნიადაგის ტენიანობა აღემატება 75%-ს მორწყვა არ არის რეკომენდებული, ვინაიდან ზედმეტი ტენიანობა იწვევს ნიადაგის დაჭაობებას, რაც სახიფათოა ნარგავის ფესვებისათვის. საწინააღმდეგო შემთხვევაში მოწმდება ჰაერის ტემპერატურა, ტემპერატურის სენსორის გამოკითხვით. თუ მისი მნიშვნელობა ნაკლებია 7^0C-ს, მორწყვა არ არის რეკომენდირებული. თუ ტემპერატურა არის 7^0C – დან 27^0C-მდე ფარგლებში, გაზონის მორწყვა სრულდება ერთხელ დღეში, დილის საათებში, ამისათვის მოწმდება რეალური დროის საათის მოდულიდან წაკითხულ მიმდნარე დროს ფორმატში არსებული ალმის მნიშვნელობა AM/PM=1, ამ პირობის ჭეშმარიტება ნიშნავს, რომ ამოკითხული დრო არის დილის საათი. საწინააღმდეგო შემთხვევაში მორწყვა სრულდება ორჯერ დღეში: დილით და საღამოთი. ზემოთ აღნიშნული პირობების შესრულების შემდეგ მიკროკონტროლერის PB0 გამოსასვლელზე ფორმირდება სარქველის ჩართვის სიგნალი. სარქვლის ჩართულ მდგომარეობაში ყოფნა დამოკიდებულია მილში წყლის წნევაზე, რომელმაც უნდა უზრუნველყოს ნარგავის მორწყვისათვის წყლის ნორმირებული რაოდენობის გაფრქვევა. თუ მილში წნევა მეტია ან ტოლია 2Pmax/3, მორწყვა სრულდება ნარგავის თვითოეული სახეობისთვის ნორმით გათვალისწინებული დროის ხანგრძლიობით. წინააღმდეგ შემთხვევაში მორწყვა სრულდება ორმაგი დროის ხანგრძლიობით. იმ შემთხვევაში კი, თუ მილში წნევა არის მინიმალურ მნიშვნელობაზე Pmin ნაკლები, მორწვა არ სრულდება. წარმოდგენილი ალგორითმი ბაზაზე დამუშავებულია პროგრამა C ენაზე, რომელის მოყვანა სახელმძღვანელოში დიდი მოცულობის გამო მიზანშეწონილად არ ჩავთვალეთ.

ლიტერატურა:

1. ო. ქართველიშვილი, ც. ხოშტარია, ს. ხოშტარია. მიკროპროცესორული სისტემები ნაწ. 1. მიკროკონტროლერების არქიტექტურა. ,,ტექნიკური უნივერსიტეტი". 2015.
2. ო. ქართველიშვილი, ს. ხოშტარია. მიკროპროცესორული სისტემები. მეთოდური მითითებები ლაბორატორიული სამუშაოების შესასრულებლად. ,,ტექნიკური უნივერსიტეტი". 2016.
3. ო. ქართველიშვილი. მიკროპროცესორი ავტომატური მორწყვის სისტემაში. მართვის ავტომატიზირებული სისტემები (შრომები) №1(19), თბილისი სტუ 2015.
4. ო. ქართველიშვილი, ს. ხოშტარია. სენსორული პანელის მართვა მიკროკონტროლერის საშუალებით. ,,მართვის ავტომატიზირებული სისტემები" (შრომები) #1(23), 2017.
5. Бепов А.С. Создаем устройства на микроконтроллерах. Наука и техника Санк Петербург. 2007.
6. Барретт С.Ф. Пак Д.Дж. Встраиваемые системы. Проектирование приложений на микроконтроллерах семейства 68HC12/HCS12 с применением языка С. ДМК. Москва.2007.

. სარჩევი