ო.ქართველიშვილი, ც.ხოშტარია, ს.ხოშტარია

მეთოდური მითითება
ლაბორატორიული სამუშაოს შესრულებისათვის

საგანში "მიკროპროცესორული სისტემები"

თბილისი
2016

შესავალი

თანამედროვე მიკროელექტრონიკა ძნელად წარმოსადგენია ისეთი მნიშვნელოვანი შემადგენლის გარეშე, როგორიც არის მიკროკონტროლერი. ბოლო დროს ჩვენ შევეჩვიეთ ხვადასხვა ელექტრონულ დამხმარე მოწყობილობებს და ხშირად არც კი ვიცით, რომ ბევრ ასეთ მოწყობილობაში მუშაობს მიკროკონტროლერი.

მიკროკონტროლერული ტექნოლოგიები მეტად ეფექტიურია. მოწყობილობა, რომელიც ადრე აწყობილი იყო ტრადიციულ ელემენტებზე, შემდგომში აგებული მიკროკონტროლერების გამოყენებით, არის გაცილებით მარტივი. იგი არ მოითხოვს რეგულირებას და ზომითაც მცირეა.

ამის გარდა, მიკროკონტროლერების გამოყენებით იქმნება პრაქტიკულად ფართო შესაძლებლობები ახალი სამომხმარებლო ფუნქციების და შესაძლებლობების დამატებისათვის უკვე არსებულ მოწყობილობისათვის. საკმარისია მხოლოდ პროგრამის შეცვლა.

მიკროპროცესორები გამოიყენებიან ბევრ ელექტრონულ მოწყობილობებში, რომლითაც ჩვენ ვსარგებლობთ ყოველდღიურ საქმიანობაში: თანამედროვე ტელევიზორი წარმოუდგენელია მიკროკონტროლერის გარეშე, თანამედროვე საათი ფაქტიურად წარმოადგენს სპეციალიზებულ მიკროკონტროლერს,

მობილური ტელეფონი ხომ მინიატიურული კომპიუტერია. თანამედროვე მიკროტალღური ღუმელი, სარეცხი მანქანა, ლაზერული დისკის დამკვრელი, კალკულიატორი. ყველა ამ მოწყობილობებში მუშაობენ მიკროკონტროლერები. მიკროკონტროლერები გამოიყენებიან საყოფაცხოვრებო ხელსაწყოებში და რთულ სამრეწველო დანადგარებში.

ელექტრონული მოწყობილობების დამუშავება მიკროკონტროლერების გამოყენებით მოითხოვს მათი მუშაობის პრინციპის ცოდნას და რაც მთავარია მმართველი პროგრამის შედგენას. პროგრამის გარეშე მიკროპროცესორი წარმოადგენს უბრალოდ პლასტმასის კორპუსს. თუ არ გაგაჩნიათ საჭირო ცოდნა შეგიძლიათ მიმართოთ წიგნის ბოლოში მითითებულ ლიტერატურას.

წინამდებარე მეთოდურ მითითება ეხება ლაბორატორიული სამუშაოების შესრულების აღწერას, რომელიც ითვალისწინებს სტუდენტის მიერ დავალების შესაბამის სქემის აწყობას, მის დაპროგრამირებას სახელმძღვანელოში მოყვანილი ნიმუშების გამოყენებით და შექმნილი სისტემის მოშაობაზე დაკვირვებას. ლაბორატორიული სტენდი წარმოადგენს პლატას, რომლის ბირთვი არის Atmel ფირმის, ამჟამად პოპულარული, მიკროკონტროლერების AVR ოჯახის წარმომადგენელი Atmega 128. სტენდზე აგრეთვე მოთავსებულია სხვადასხვა პერიფერიული მოწყობილობა: ანალოგურ-ციფრული გარდამსახი, რეალური დროის საათი, ტემპერატურის სენსორი, ტექსტური და გრაფიკული დისპლეი, სენსორული ეკრანი, ფლეშ-დისკი, შუქდიოდები და კლავიატურა. მიკროპროცესორთან სტენდზე არსებული მოწყობილობების მიერთება ხდება გადამრთველების საშუალებით, ხოლო სტენდთან რაიმე გარე სქემების დაკავშირება სტანდარტული ინტერფეისებით.

წარმოდგენილი მეთოდური მითითება შესდგება სამი თავისაგან. პირველი თავი შეიცავს სტენდის აღწერას. განხილულია მისი შემადგენელი ელემენტები და

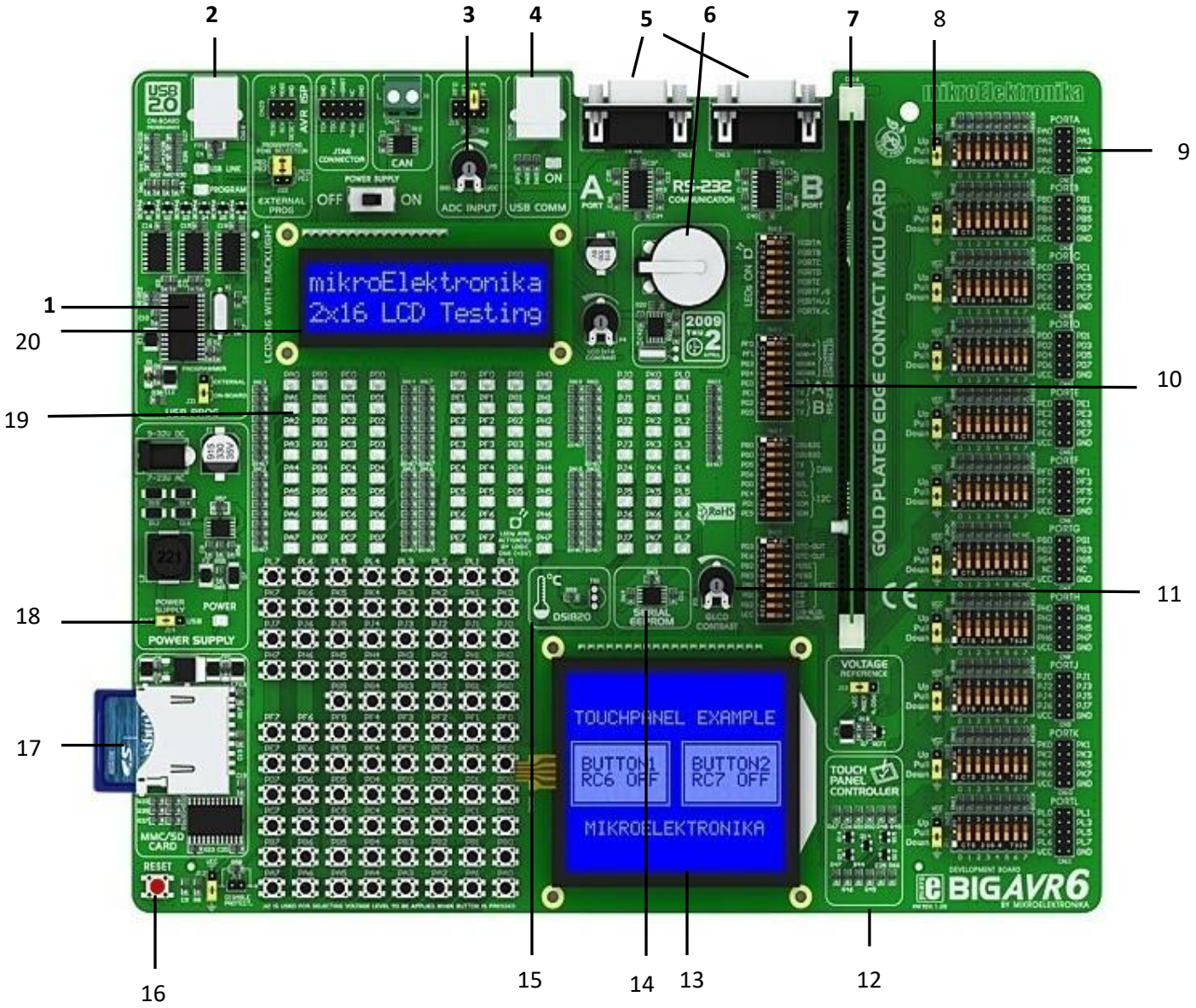
მიკროკონტროლერთან მათი მიერთების საშუალებები. მეორე თავი ეძღვნება პროგრამის გაწყობას და კომპილიატორთან მუშაობას. ლაბორატორიულ სამუშაოს შესრულების დროს გამოიყენება პროგრამული გარემო MicroC PRO for AVR, რომელიც შექმნილია AVR ოჯახის მიკროკონტროლერებისათვის C ენაზე დაწერილი პროგრამების კომპილირებისათვის. მესამე თავი შესდგება ლაბორატორიული დავალებებისაგან. თვითეული დავალება ითვალისწინებს მოსწავლის მიერ კონკრეტული სისტემის აწყობას მიკროკონტროლერთან შესაბამისი მოწყობილობების მიერთებით და მიკროკონტროლერისთვის პროგრამის შედგენას. ყოველ დავალებაში ნაჩვენებია გადამრთველების მდგომარეობა აღნიშნული კავშირების შესაქმნელად. თითოეულ დავალებას თან ახლავს საილუსტრაციო პროგრამა, რომელიც გვიჩვენებს შესაბამისი მოწყობილობის მიკროკონტროლერთან ურთიერთობის შესაძლებლობებს. ამ პროგრამების საფუძველზე სტუდენტს გაუადვილდება შექმნას ახალი პროგრამები დავალების მიხედვით.

ჩვენი რწმენით წარმოდგენილი ლაბორატორიული სამუშაოები დაეხმარება შესაბამისი სპეციალობის სტუდენტებს პრაქტიკული გამოცდილების მიღებაში მიკროპროცესორული სისტემების დაგეგმარებაში და შექმნაში.

თავი1

1.1. ლაბორატორიული სტენდის აღწერა

ლაბორატორიული სტენდი წარმოადგენს პლატას, რომელზეც მოთავსებულია მიკროკონტროლერი და მასთან დაკავშირებული სხვადასხვა დანიშნულების კომპონენტი, რომლებიც ნაჩვენებია სურ.1.1-ზე.



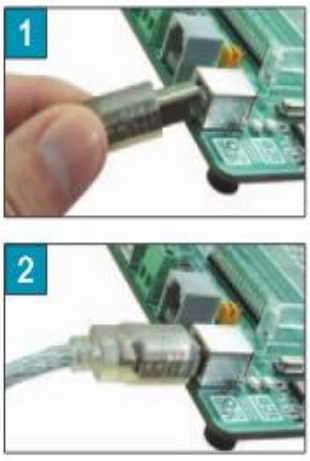
სურ.1.1

1. ჩაშენებული პროგრამატორი, რომლის საშუალებით სრულდება მიკროკონტროლერში პროგრამის ჩაწერა.
2. გასართი, რომლითაც პლატას შეიძლება მიუერთდეს გარე პროგრა- მატორი.
3. რეგულიატორი აცგ-ს შესასვლელზე ძაბვის ცვლილებისათვის .

4. USB გასართი.
5. RS 232 მიმდევრობითი ინტერფეისის გასართები.
6. რეალური დროის ტაიმერი.
7. ცენტრალური პროცესორის პლატის გასართი.
8. გადამრთველები მომჭიმავი წინააღმდეგობის მიერთებისათვის პორტების გამოსასვლელებზე.
9. პორტების გამოსასვლელები გარე მოწყობილობების მიერთებისათვის.
10. გადამრთველები ცენტრალურ პროცესორთან პლატაზე არსებული მოდულების მიერთებისათვის.
11. შაყრდენი ძაბვის წყარო.
12. სენსორული ეკრანის კონრტოლერი.
13. გრაფიკული დისპლეი.
14. მეხსიერება EEPROM.
15. კლავიატურა ციფრული ინფორმაციის შეყვანისათვის მიკროკონტრო- ლერში.
16. განულების ღილაკი.
17. ფლეშ დისკის (MMC) ჩასადები ბუდე.
18. კვების წყაროს ჩამრთველი.
19. ინდიკატორი შუქდიოდებზე.
20. ტექსტური დისპლეი.

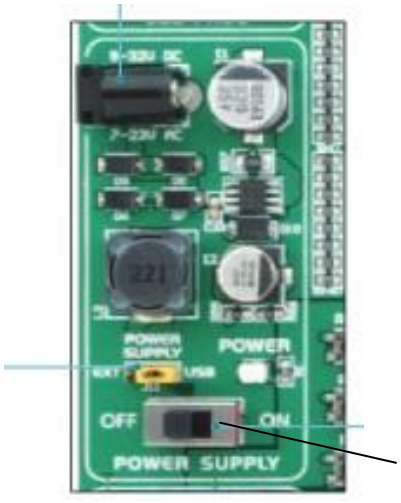
1.2. ლაბორატორიული სტენდის ჩართვა და მასთან მუშაობის დაწყება

სტენდთან მუშაობისათვის იგი უნდა მიუერთოთ პერსონალურ კომპიუტერს USB გასართის საშუალებით (სურ.1.2) და ჩავრთოთ კვების წყარო ღილაკის გადართვით (სურ.1.3)



USB კაბელის მიერთება

სურ.1.2



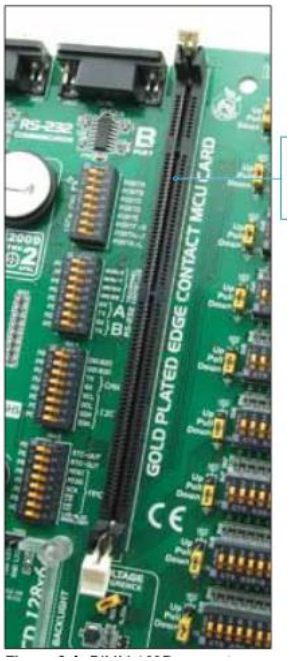
კვების წყაროს ჩამრთველი

სურ.1.3

ცხადია, წინასწარ პერსონალურ კომპიუტერში უნდა ჩატვირთული იყოს კომპილიატორი, რომლის საშუალებით მოხდება მიკროპროცესორში ჩასაწერი პროგრამის ტრანსლირება და USB სალტით შემდეგ მიკროპროცესორში მისი გაკერვა

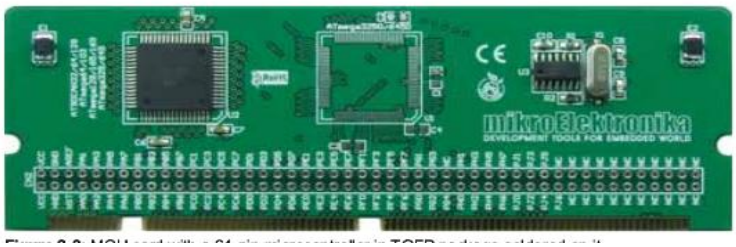
1.3. მიკროკონტროლერის დაკავშირება სტენდთან

მიკროკონტროლერი (სტენდზე გამოყენებულია AVR ოჯახის მიკროკონტროლერი Atmega 128) მოთავსებულია კარტაზე, რომელიც იდგმება სტენდის პლატაზე მირჩილულ გასართში (იხ. სურ. 1.4-1.6). გასართის კონტაქტები უკავშირდებიან პლატაზე მოთავსებულ სხვადასხვა სქემებს.



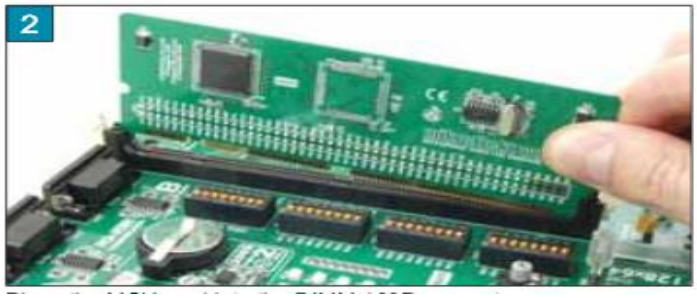
მიკროკონტროლერის კარტის გასართი

სურ.1.4



ცენტრალური პროცესორის კარტა

სურ.1.5

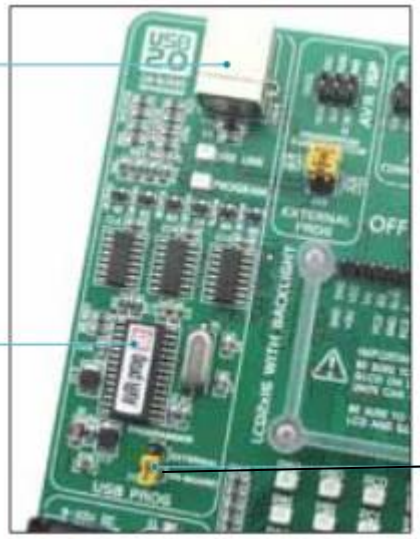


გასართში პროცესორის კარტის ჩაყენება

სურ.1.6

1.4. პროგრამატორის გამოყენება

პროგრამატორი წარმოადგენს აუცილებელ ინსტრუმენტს მიკროკონტრო- ლერებთან მუშაობის დროს. BIGAVR6 ლაბორატორიულ სტენდს გააჩნია ჩაშენებული პროგრამატორი AVRprog, რომელიც ანხორციელებს ინტერფეისს მიკროკონტროლერსა და პერსონალურ კომპიუტერს შორის. პერსონალურ კომპიუტერში C ენაზე დაწერილი პროგრამა გარდაიქმნება კომპილიატორის მიერ Hex-ფაილად და მიეწოდება USB სალტით პროგრამატორს, რომელიც თავის მხრივ ჩაწერს მას მიკროპროცესორის პროგრამის მეხსიერებაში. სურ.1.7-1.8-ზე ნაჩვენებია პროგრამატორის მდებარეობა პლტაზე და მისი გასართი.

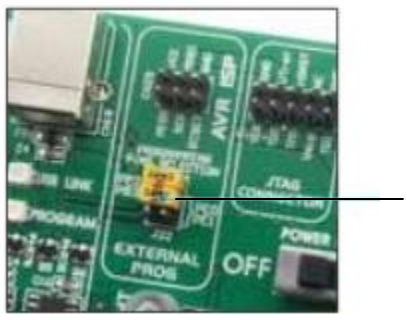


სურ.1.7

USB გასართის წინა ხედი

სურ.1.8.

შესაძლებელია გამოყენებული იყოს გარე პროგრამატორი, რომელიც უერთდება პლატას სპეციალური გასართის საშუალებით. ასეთ შემთხვევაში ზემოთ ნაჩვენები ჯამპერი J21 უნდა დავაყენოთ *EXTERNAL* პოზიციაში (წინაადმდეგ შემთხვევაში პოზიციაში *ON-BOARD*. სურ.1.9-1.10-ზე ნაჩვენებია გასართი გარე პროგრამატორის მიერთებისათვის და ჯამპერის მდგომარეობა სხვადასხვა პროგრამატორი გამოყენების შემთხვევაში.



სურ. 1.9

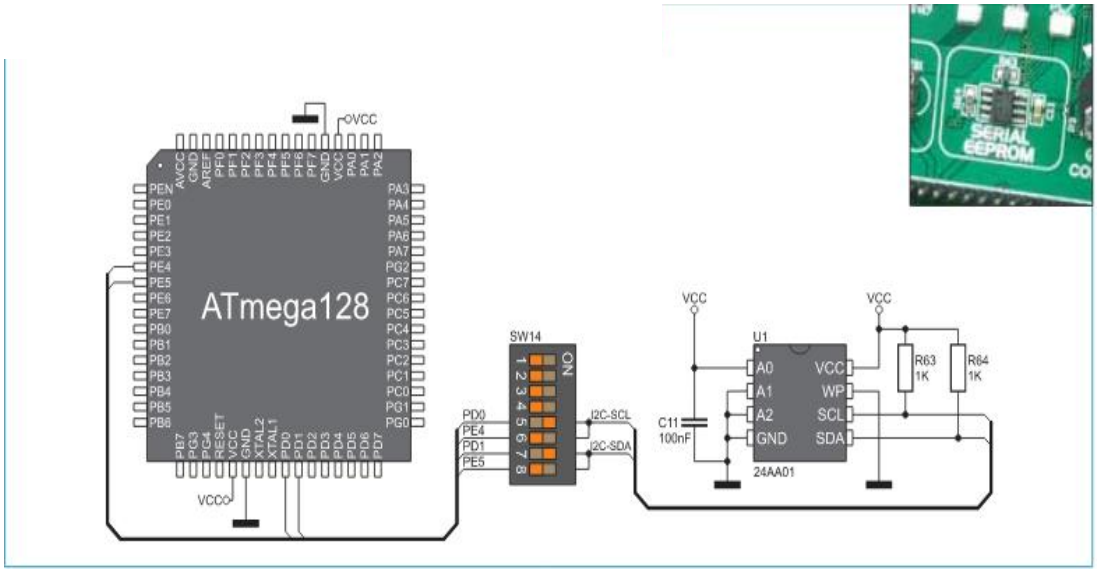
მდგომარეობა გარე პროგრამატორის მიერთების შემთხვევაში

მდგომარეობა ჩაშენებული პროგრამატორის მიერთების შემთხვევაში

სურ.1.10

1.5. EEPROM მეხსიერებასთან მუშაობა

პლატაზე არის დამონტაჟებული გარე მუდმივი მეხსიერება EEPROM 1კბიტი ტევადობით (მიკროკონტროლერს აქვს აგრეთვე ჩაშენებული EEPROM), რომელიც მიკროკონტროლერთან კომუნიკაციისთვის იყენებს I^2C ინტერფეისს PD0 დაPD1 ან PE4 და PE5 პორტების გამომყვანების საშუალებით. აღნიშნული კონტაქტების დასაკავშირებლად EEPROM ინტეგრალური სქემის შესასვლელებთან გდამრთველების SW14 პაკეტში 5 და 7 ან 6 და 8 გადამრთველები უნდა გადავრთოთ ON პოზიციაში. სურ. 1.11 მოცემულია EEPROM ინტეგრალური სქემის მდებარეობა პლატაზე და მისი მიერთების სქემა მიკროკონტრო- ლერთან.



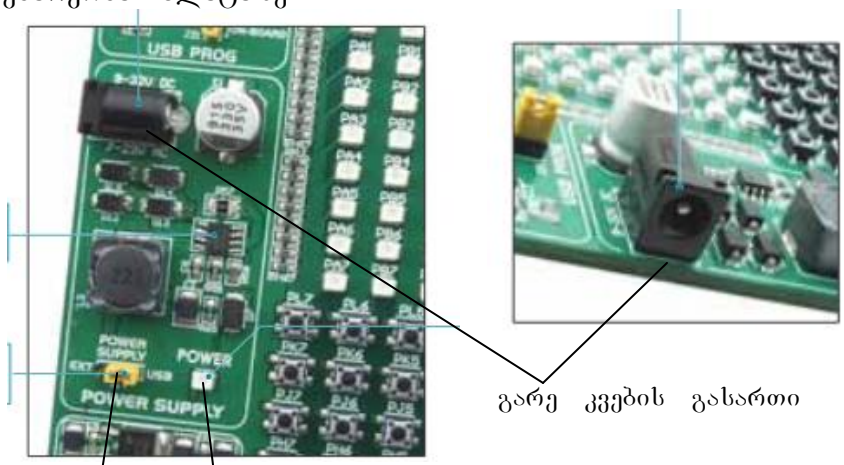
სურ.1.11

1.6. კვების წყარო

BIGAVR6 პლატაზე შეიძლება გამოყენებული იყოს ორიდან ერთ-ერთი კვების წყარო:

1. +5ვ USB პროგრამირების კაბელიდან;
2. არე კვების წყაროდან ცალკე გასართის საშუალებით, რომელიც დამონტაჟებულია პლატაზე.

გადართვა ერთი კვების წყაროდან მეორეზე სრულდება J14 გადამრთველის საშუალებით: თუ პლატაზე გამოიყენება კვება USB ინტერფეისიდან, მაშინ გადამრთველი ყენდება USB პოზიციაში, როდესაც გამოყენებულია გარე კვების წყარო გადამრთველი უნდა იყოს EXT პოზიციაში. სურ.1.12 ნაჩვენებია კვების ბლოკის მდებარეობა პლატაზე.

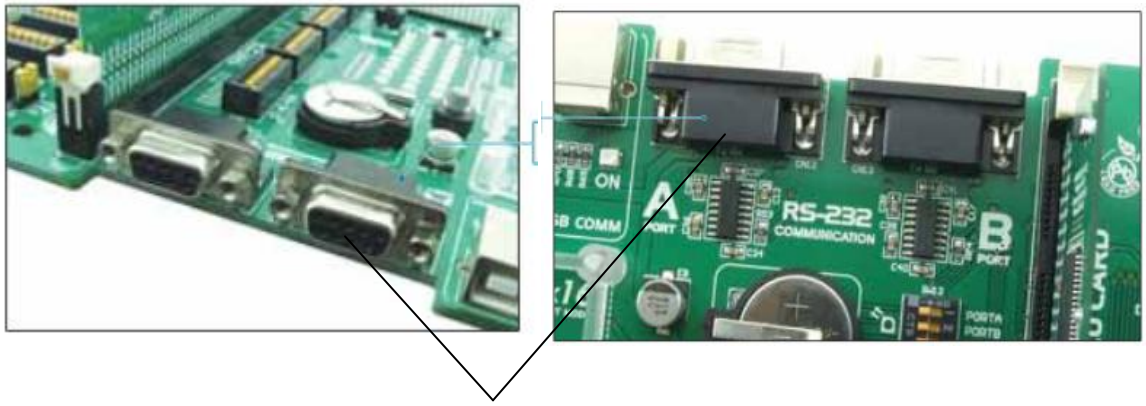


სურ.1.12

1.7. RS-232 მიმდევრობითი ინტერფეისი

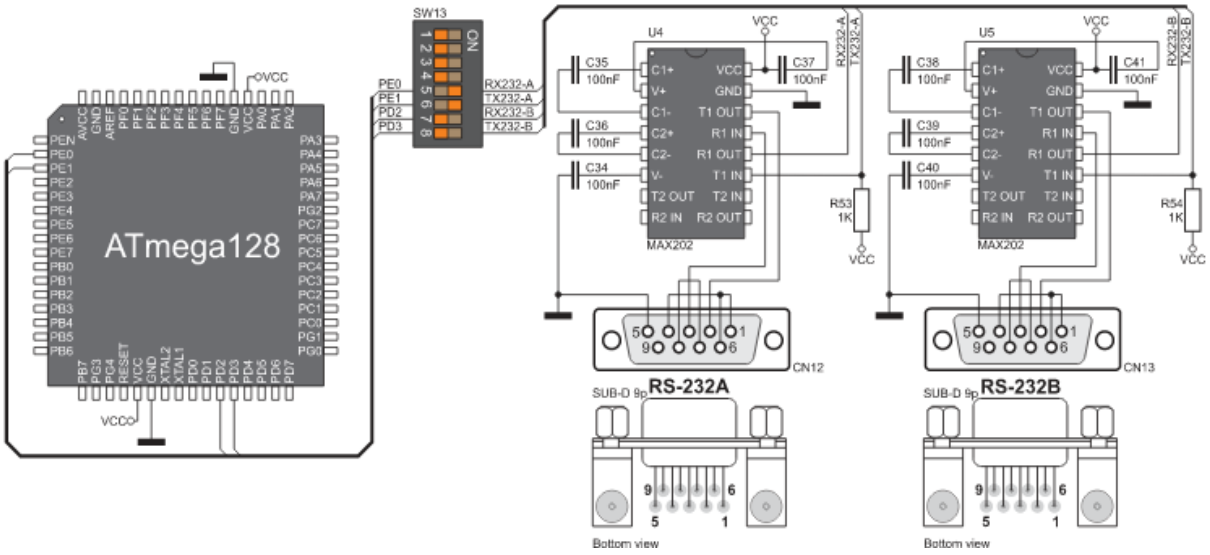
USART (უნივერსალური სინქრონულ/ასინქრონული მიმღებ/გადამცემი) არის ერთ-ერთი ყველაზე გავრცელებული საშუალება მონაცემთა გაცვლისათვის პერსონალურ კომპიუტერსა და გარე მოწყობილობას შორის. კავშირი კომპიუტერსა და სტენდზე არსებულ მიკროკონტროლერში ჩაშენებულ ორ USART მოდულს შორის ხორციელდება 9-კონტაქტიანი RS-232 გასართის საშუალებით. სტენდის პლატაზე დაყენებულია ორი RS-232 გასართი - RS-232A და RS-232B , რომლებიც დაკავშირებული არიან მიკროკონტროლერის ორივე USART მოდულის გამოსასვლელებთან RX232A (გამოსასვლელი PE0) და TX232A (გამოსასვლელი PE1) –USART0-თან, RX232B (გამოსასვლელი PD2) და TX232B (გამოსასვლელი PD3) შესაბამისად. აღნიშნული გამოსასვლელების მიერთება გასართის კონტაქტებთან სრულდება SW13 გადამრთველების საშუალებით. მიკროკონტროლერის გამომყვანები წარმოდგენილ შეერთებაში აღინიშნებიან, როგორც RX- მონაცემების მიღების ხაზი და TX- მონაცემების გაცემის ხაზი. ინტერფეისით გადაცემის სიჩქარე არის 115 კბ/წ. სურ.1.13-

ზე ნაჩვენებია პლატაზე RS-232 გასართების განლაგება, სურ.1.14-ზე გასართების მიერტების სქემა მიკროკონტროლერთან.



RS-232 ინტერფეისის გასართი

სურ.1.13

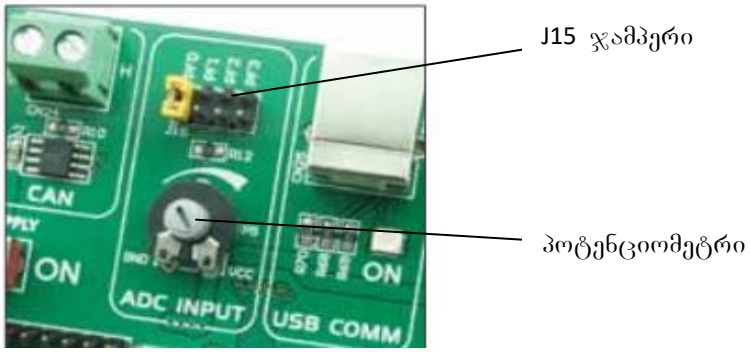


სურ.1.14

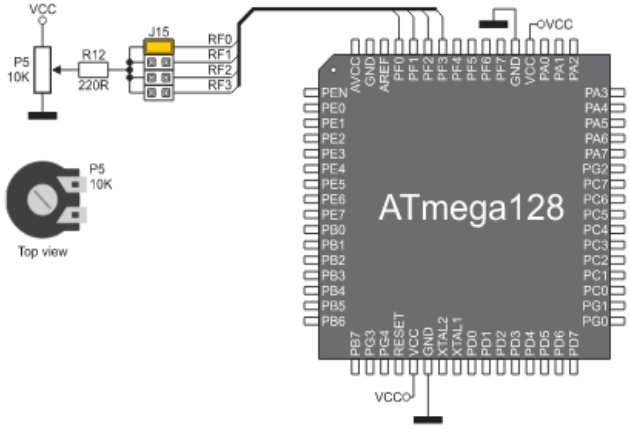
1.8. ანალოგურ-ციფრული გარდამსახი (აცგ)

აცგ გამოიყენება ანალოგური სიგნალის (ძაბვის) გარდასახვისათვის ციფრულ ფორმატში, იგი შედის მიკროკონტროლერის შემადგენლობაში. გარდასასახი ძაბვა,რომელიც მას მიეწოდება მიკროკონტროლერის ერთ-ერთ შესასვლელზე გარდაისახება 10 თანრიგა ორობით კოდად. ლაბორატორიულ სტენდზე აცგ-ს მუშაობის დემონსტრირების მიზნით მის შესასვლელზე მიერთებულია პოტენციომეტრი P5, რომლითაც შესაძლებელია შემავალი ძაბვის მნიშნელობის ცვლა 0-დან 5ვ-ის ფარგლებში. პოტენციომეტრი შეიძლება მიუერთდეს სხვადასხვა შესასვლელებს.

სტენდზე გათვალისწინებულია PF0,PF1,PF2,PF3 თან მიერთება. თვითოეულ შესასვლელთან მიერთება ხორციელდება J15 გადამრთველების (ჯამპერების) საშუალებით. სურ.1.15-ზე ნაჩვენებია პოტენციომეტრის მდებარეობა პლატაზე, ხოლო სურ.1.16-ზე პოტენციომეტრის მიერთების სქემა.



სურ. 1.15.



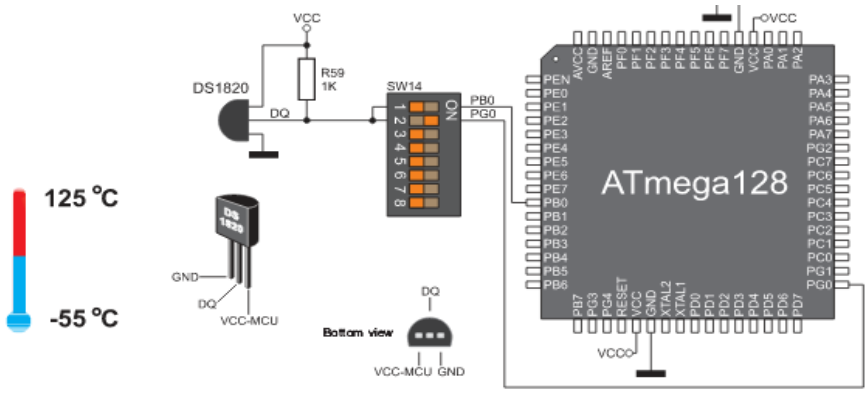
სურ.1.16

1.9. ტემპერატურის გადამწოდი (სენსორი) DS1820

DS1820 ტემპერატურის სენსორი ზომავს გარემოს ტემპერატურას -55დან 125^0მდე შუალედში. იგი წარმოადგენ ინტეგრალურ სქემას, რომელშიც სრულდება ანალოგური სიგნალის გადასახვა 9-თანრიგა ორობით კოდად. კავშირი მიკროკონტროლერთან სრულდება 1-Wire პროტოკოლით ერთი გამტარის საშუალებით, რომლითაც იგი უერთდება მიკროკონტროლერის ერთ-ერთ გამომყვანს. სტენდზე ამ მიზნით გამოიყენება მიკროკონტროლერის PB0 ან PG0 გამომყვანი. მათი მიერთება სენსორთან ხდება SW14 გადამრთველებიდან 1 ან 2 გადამრთველის დაყენებით ON პოზიციაში.

სურ.1.17-ზე ნაჩვენებია სენსორის მდებარეობა სტენდზე, სურ.1.18-ზე - მიკროკონტროლერთან მისი მიერთების სქემა .

სურ.1.17



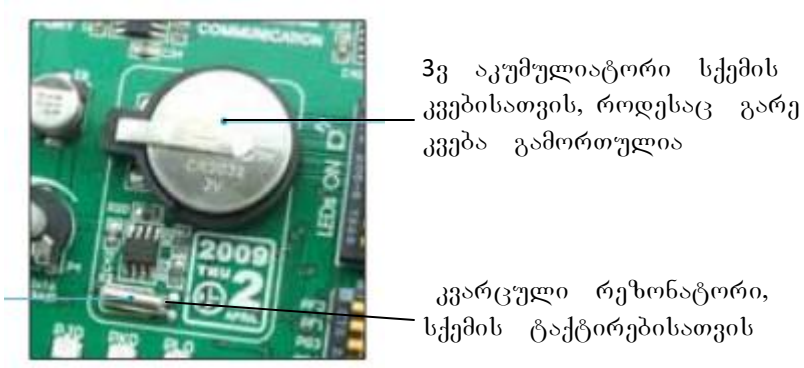
სურ.1.18

1.10 რეალური დროის საათი (RTC)

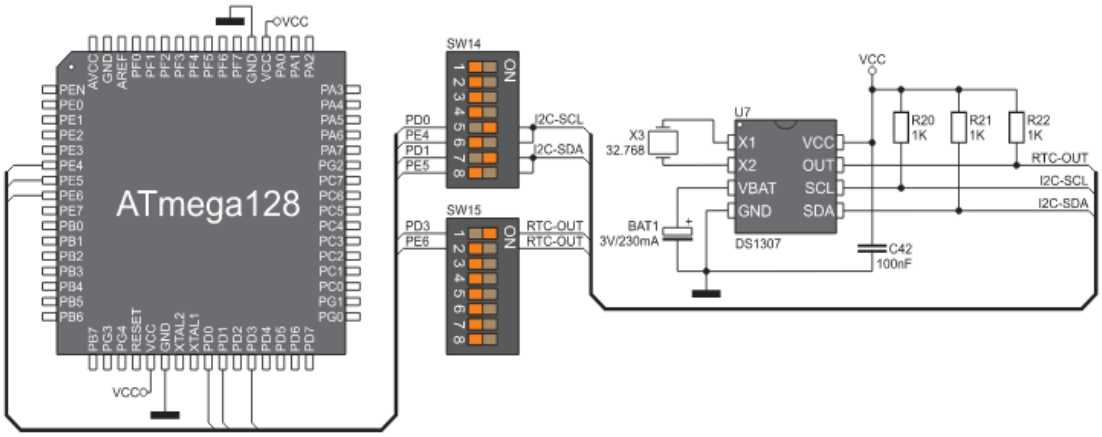
როდესაც მიკროკონტროლერის მიერ შესრულებული ამოცანიდან გამომდინარე, მას უხდება რეალურ დროის რეჟიმში მუშაობა, შესაძლებელია გამოყენებული იყოს გარე რეალური დროის საათი, რომელიც წარმოადგენს ინტეგრალურ სქემას DS1307. მის ძირითად თვისებებია:

- ინფორმაციის წარმოდგენა მიმდინარე წამების, წუთების, საათის, დღეების, კვირის დღეების, წლის შესახებ. თარიღში შესწორების შეტანა ნაკიანი წლის შემთხვევაში;
- მიკროკონტროლერთან კავშირს ანხორციელებს I^2C ინტერფეისის საშუალებით;
- კვების წყაროს ავტომატური კონტროლი. მისი წყაროდან გამორთვის შემთხვევაში აკუმულიატორთან მიერთება;

სქემის გამოსასვლელები უკავშირდებიან მიკროკონტროლერის PD0,PD1 დაPD3 გამსასვლელებს ან PE4,PE5,PE6 გამოსასვლელებს SW14 და SW15 გადამრთველების საშუალებით. სურ.1.19 ნაჩვენებია რეალური დროის საათის მდებარეობა პლატაზე, ხოლო სურ.1.20–ზე - მისი მიერთების სქემა მიკროკონტროლერთან.



სურ.1.19

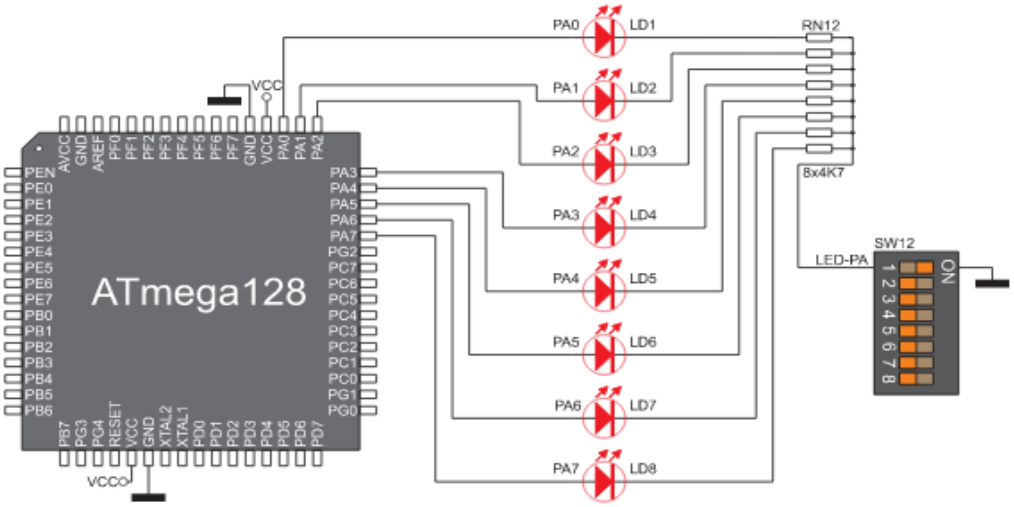


სურ.1.20

1.11 შუქდიოდები (LED)

მიკროკონტროლერის გამოსასვლელების მდგომარეობის ინდიკაციისათვის სტენდის პლატაზე გამოყვანილია 86 შუქდიოდი. თვითოეული შუქდიოდი მიერთებულია პორტის ერთ გამომყვანზე და ასახავს ამ გამომყვანის მდგომარეობას: თუ რაიმე გამომყვანზე არის ლოგიკური 1 (მაღალი პოტენციალი), შუქდიოდი ანთია, გამომყვანზე 0-ის არსებობის დროს (დაბალი პოტენციალი) – შუქდიოდი ჩამქრალია. პლატაზე გამოყვანილ შუქდიოდების მატრიცაში თვითეული სვეტი შეესაბამება რაიმე პორტის გამომყვანებს თანრიგების ნომრების ზრდით ზევიდან ქვევით (თვთოეულ შუქდიოდს აქვს წარწერა). რაიმე პორტი შუქდიოდების გააქტიურებისათვის SW12 გადამრთველებიდან შესაბამისი გადამრთველი უნდა გადავრთოთ ON მდგომარეობაში. სურ.1.21-ზე ნაჩვენებია შუქდიოდების მდებარეობა სტენდზე. სურ.1.22-ზე შუქდიოდების მიერთება მიკროკონტროლერის გამომყვანებთან.

სურ.1.21



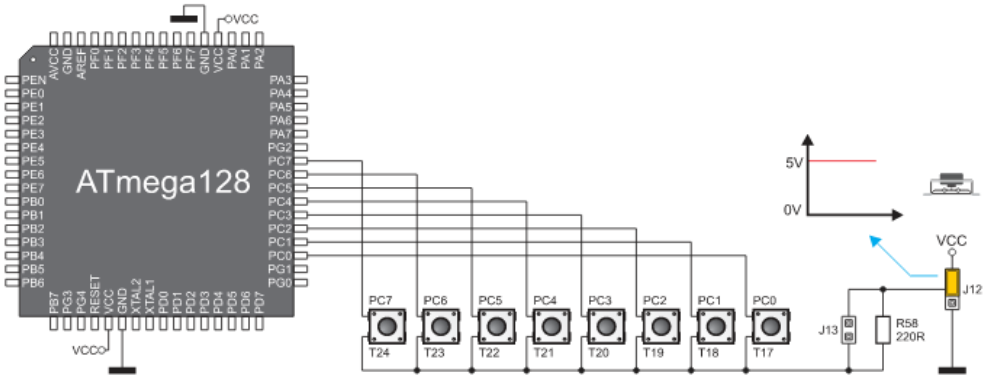
სურ.1.22

1.12. დასაჭერი ღილაკები

სტენდზე გამოყვანილია ღილაკები, რომლის საშუალებით შეგვიძლია შევცვალოთ მიკროკონტროლერის გამომყვანის ლოგიკური მდგომარეობა. ყოველ გამომყვანზე მიერთებულია ერთი ღილაკი. ღილაკის აშვებული მდგომარეობის დროს შესაბამის გამომყვანზე დგება 1, დაჭერილ მდგომარეობაში – 0. სტენდზე მოთავსებულ

ღილაკების მატრიცაზე ყოველი სტრიქონი შეესაბამება რომელიმე პორტის გამომყვანებს მცირე ნომრის თანრიგიდან დიდი ნომრის თანრიგამდე მარჯვნიდან მარცხნივ. რაიმე პორტის ღილაკების გააქტიურება სრულდება J12 ჯამპერის გადართვით Vcc (კვების წყაროს ძაბვა) მდგომარეობაში. სურ.1.23-ზე ნაჩვენებია ღილაკების განლაგება სტენდზე. სურ.1.24-ზე - მიკროკონტროლერთან მიერთების სქემა.

სურ.1.23

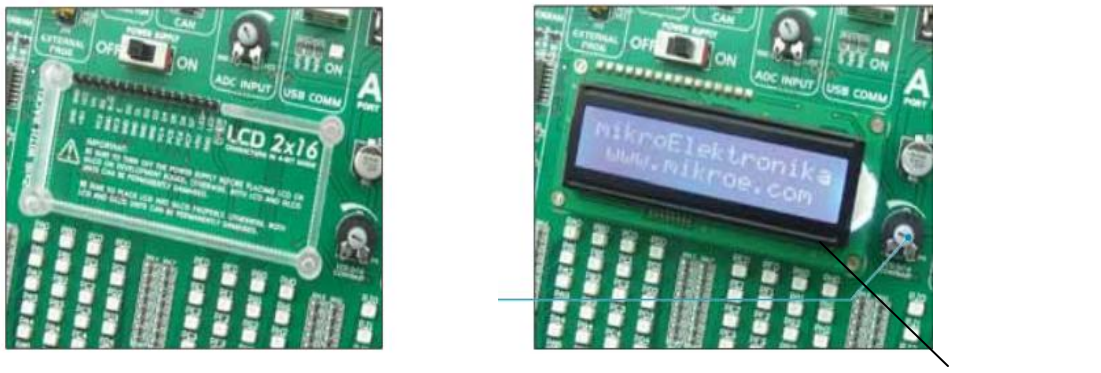


სურ.1.24

1.13 ტექსტური დისპლეი

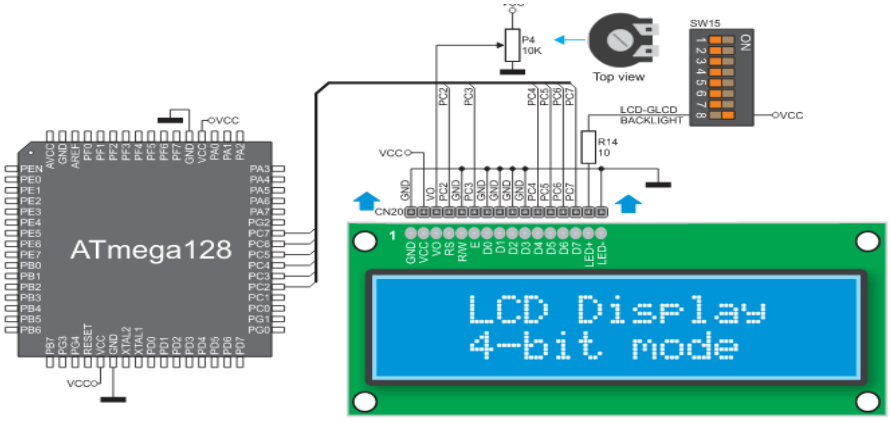
სტენდზე მოთავსებულია გასართი 2x16 LCD ტექსტური დისპლეისათვის. აღნიშნული გასართი დაკავშრებულია მიკროკონტროლერთან PC პორტის გამომყვანებით. P4 პოტენციომეტრი გამოიყენება დისპლეის კონტრასტის რეგულირებისთვის. SW15 გადამრთველებიდან მე-8 გადამრთველი ემსახურება დისპლეის ფერის ჩართვა / გამორთვას. სიმბოლოების გადაცემა ხდება 4-ბიტიან

რეჟიმში. ტექსტის გამოყვანა სრულდება ორ სტრიქონად. თვითეული სტრიქონი შეიცავს 16 სიმბოლოს 7x5 პიქსელის ზომით. სურ.1.25-ზე ნაჩვენებია დისპლეის მდებარეობა სტენდზე, სურ.1.26-ზე დისპლეის მიკროკონტროლერთან მიერთების სქემა.



ტექსტური დისპლეის გასართი ტექსტური დისპლეი კონტრასტის პოტენციომეტრი

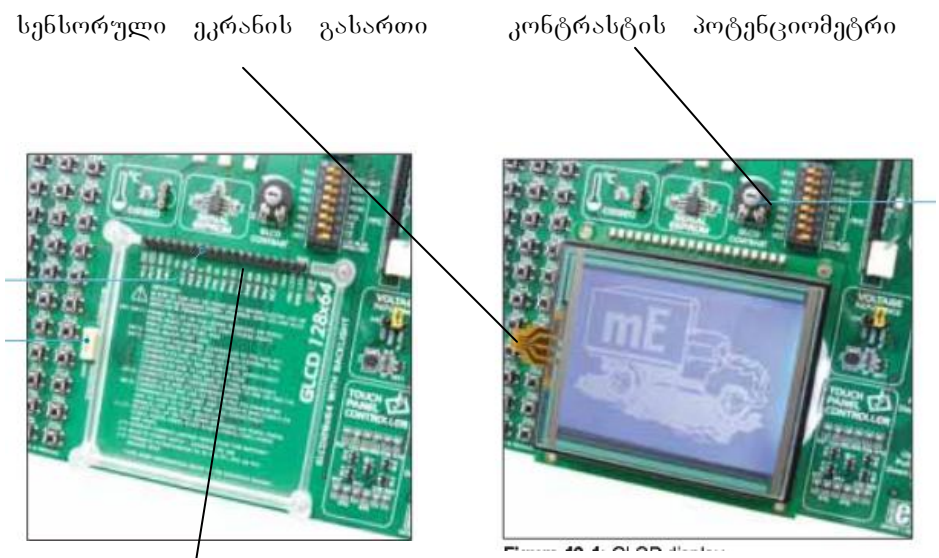
სურ.1.25



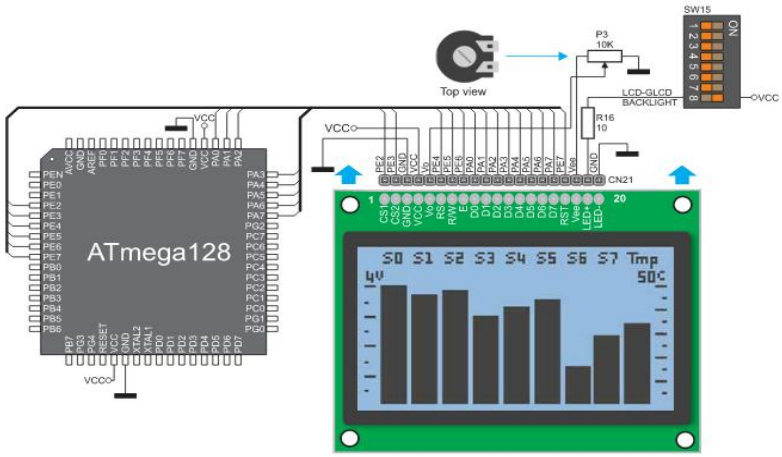
სურ.1.26

1.14. გრაფიკული დისპლეი (GLCD)

128x64 გრაფიკული LCD დისპლეი განკუთვნილია მიკროკონტროლერიდან გრაფიკული გამოსახულების გამოსატანად. იგი უკავშირდება მიკროკონტროლერს PA და PE პორტების საშუალებით. მისი გარჩევითობა შეადგენს 128x64 პიკსელს. P3 პოტენციომეტრით ხდება დისპლეიზე გამოსახულების კონტრასტის რეგულირება. სურ.1.27-ზე ნაჩვენებია გრაფიკული დისპლეის მდებარეობა სტენდზე, სურ.1..28-ზე – მისი მიკროკონტროლერთან მიერთების სქემა.



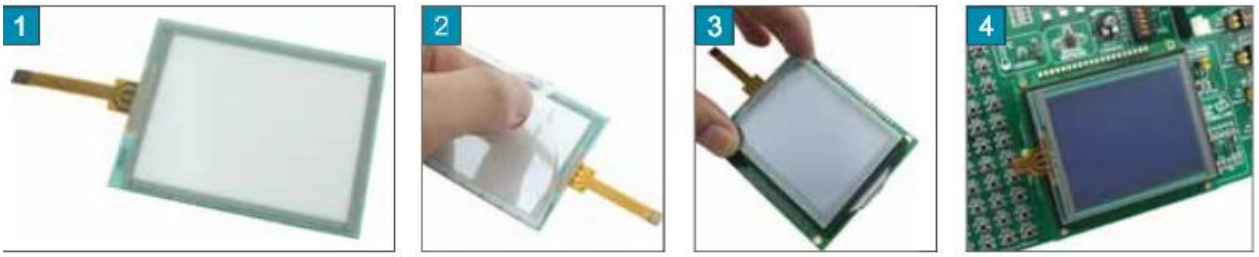
სურ.1.27



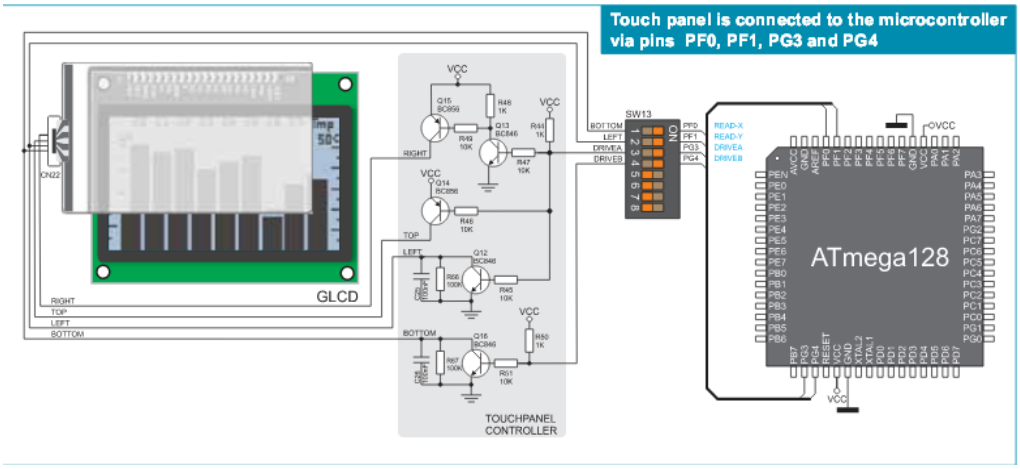
სურ.1.28

1.15 სენსორული ეკრანი

სენსორული ეკრანი წარმოადგენს დაჭერაზე მგრძნობიარე თხელ ფირფიტას, რომელიც განთავსებულია გრაფიკული დისპლეიზე. მასზე დაჭერის შემთხვევაში დაჭერილი ადგილის კოორდინატებს გარდასახავს ანალოგურ სიგნალებად და აწვდის მიკროკონტროლერს, რომელიც პროგრამის შესაბამისად გრაფიკულ ეკრანზე გამოიყვანს საჭირო გამოსახულებას. SW13 გადამრთველებიდან 1,2,3 და 4 გადამრთველების საშუალებით ხდება სენსორული ეკრანის მიერთება მიკროპროცესორთან PF0,PF1 და PG3,PG4 გამომყვანებით. სურ.1.29-ზე ნაჩვენებია სენსორული ეკრანის მდებარეობა სტენდზე, სურ.1.30-ზე მისი მიკროკონტროლერთან მიერთების სქემა.



სურ.1.29

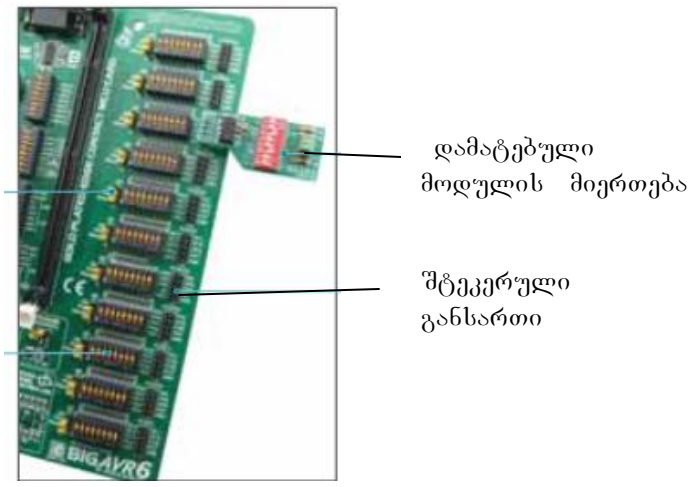


სენსორული პანელის მიერთება

სურ.1.30

1.16 შეტანა გამოტანის პორტები

სტენდზე განლაგებულია პორტები, რომლის საშუალებითაც შესაძლებელია გარე მოწყობილობის მიერთება. სურ.1.31-ზე ნაჩვენებია პორტების განლაგება სტენდზე.



სურ.1.31

გარე მოწყობილობა უკავშირდება სტენდს შტეკერულ გასართში პლატის დაყენებით, რომელზედაც გარე მოწყობილობა არის დამონტაჟებული (როგორც ეს სურათზეა ნაჩვენები), ან კაბელის საშუალებით. გასართების მიერთებისათვის სტენდის გამომყვანებთან საჭიროა გადამრთველი Pull Up/ Pull Out გადავრთოთ Pull Up მდგომარეობაში.

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 (კომპილიატორების) დიდი რაოდენობა, რომლებიც შექმნილია სხვადასხვა მიკროპროცესორებისათვის პროგრამულ ენიდან მანქანურ კოდში ტრანსლირებისათვის (მაგალითად, AVR ოჯახის მიკროპროცესორებისათვის - AVRStudio, WinAVR, CodeVisionAVR და სხვა კომპილიატორები).

ჩვენს მიერ წარმოდგენილი ლაბორატორიული სამუშაოებისათვის გამოყენებულია Atmel ფირმის მიერ AVR მიკროპროცესორებისათვის დამუშავებული კომპილიატორი *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 fot AVR კომპილიატორის მთავარი პანელი, რომელიც დაყოფილია რამოდენიმე სარკმლად. სურათზე ისინი აღნიშნულია ციფრებით: 1,2,3,4,5.

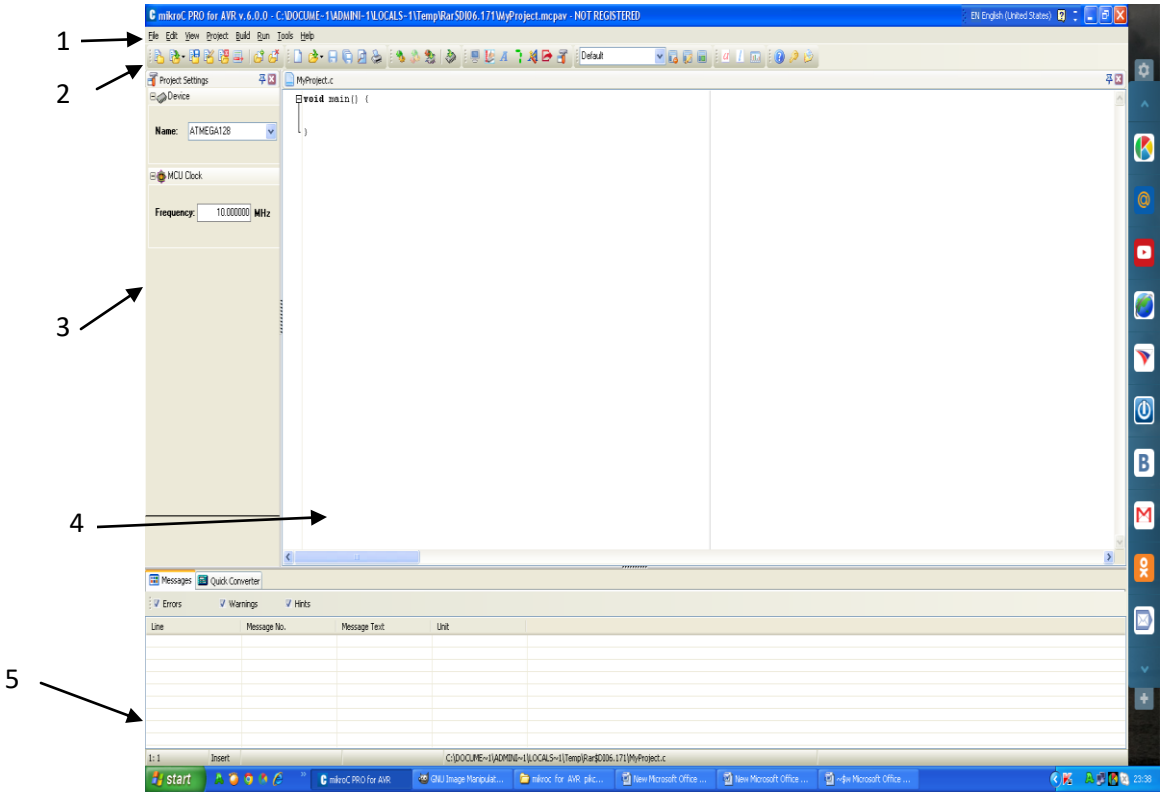
1 სტრიქონი წარმოადგენს **მენიუს სტრქონს** , 2 სტრიქონი - **ინსტრუმენტალურ პანელს,** სადაც გამოყვანილია ზოგიერთი ბრძანების ღილაკი. ისევე, როგორც Windows-ის ნებისმიერ სხვა პროგრამაში, მენიუს საშუალებით ხდება *MikroC PRO for AVR* პროგრამის ყველა ფუნქციის გამოძახება და რეჟიმებზე გადართვა. 3 ფანჯარაში ხორციელდება პროექტის პარამეტრების დაყენება: მიკროპროცესორის და მისი მუშაობის სიხშირის არჩევა. 4 პანელი 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 შეცდომას ქონდა ადგილი მისი ტრანსლირება არ სრულდება შეცდომის აღმოფხვრამდე.

MikroC PRO for AVR მუშაობს ე.წ. **პროექტებთან.** თვითოეულ 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. იგი შესდგება ფაილების დიდი ანაკრეფისაგან.

მათგან მთავარია - პროექტის ფაილი. მას აქვს გაფართოება .mcpav. პროექტის ფაილი შეიცავს ცნობას პროცესორის ტიპის, სიხშირის გენერატორის და სხვ.

შესახებ. იგი ასევე შეიცავს მასში შემავალი ფაილების აღწერას, რომელიც გამოიყენება კომპილიაციის დროს.



სურ.2.1

თუ პროგრამამ ტრანსლიაციის პროცესი გაიარა წარმატებით, შედეგად ფორმირდება რამოდენიმე ფაილი, რომელთა შორის ჩვენთვის მნიშვნელოვანია ე.წ. hex - ფაილი (ორობით ფორმატში წარმოდგენილი ფაილი), რომელიც უნდა ჩაიწეროს მიკროკონტროლერის პროგრამის მეხსიერებაში.

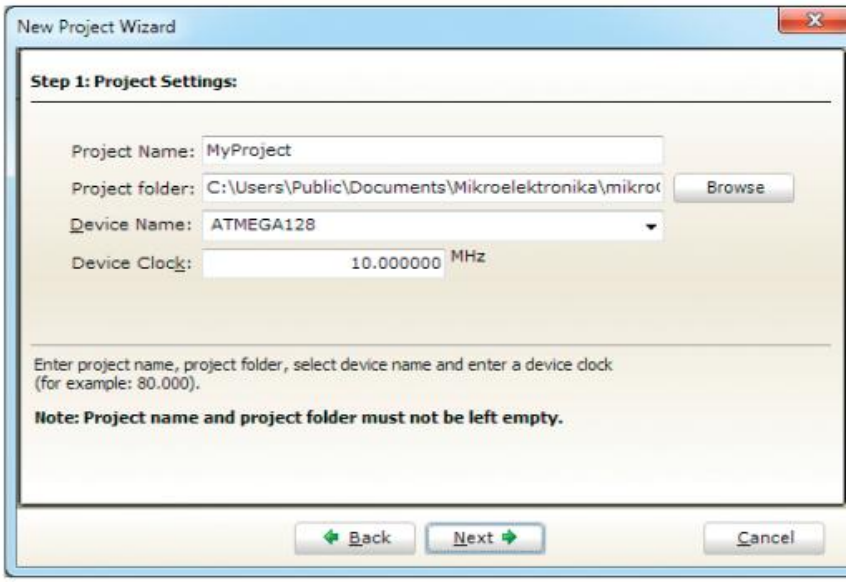
2.2პროექტის შექმნა

დაუშვათ, MikroC PRO for AVR უკვე დაყენებულია თქვენს პერსონალურ კომპიუტერზე . კომპილიატორის 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 ინსტრუმენტალურ 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 ღიაკს.

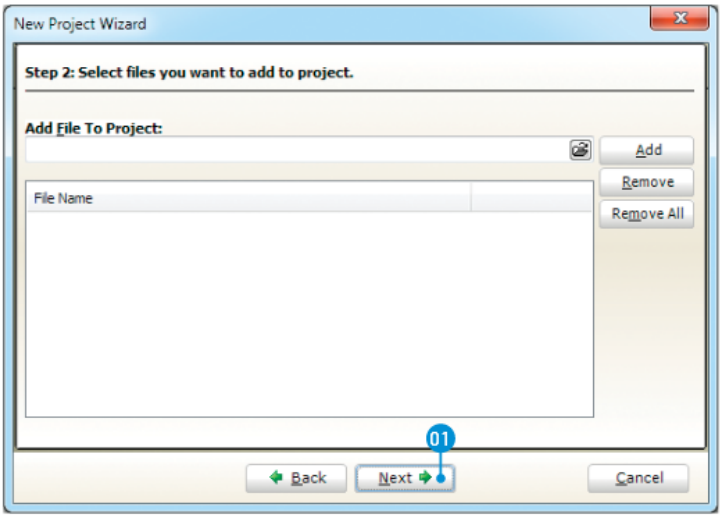
სურ.2.2

ეკრანზე გაიხსნება მომდევნო ფანჯარა (სურ.2.3).



სურ.2.3.

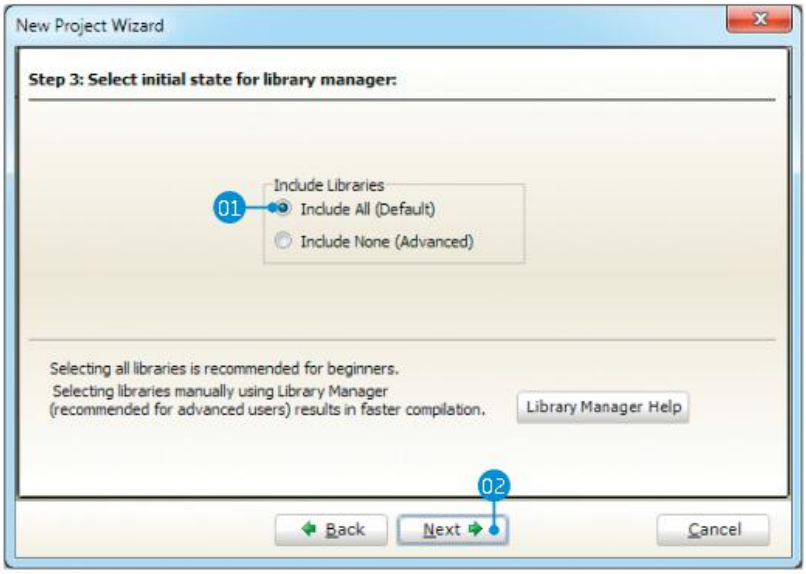
ამ ფანჯარაში ხდება სხვადასხვა პარამეტრების დაყენება: **Project name** ველში უნდა ჩაიწეროს ჩვენს მიერ მინიჭებული პროექტის სახელი, **Project folder** ველში - გზა იმ საქაღალდისათვის, რომელშიც გვსურს პროექტის შენახვა, **Device Name** ველში - ჩაიწერება პროექტში გამოყენებული მიკროპროცესორის ტიპი, **Device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



სურ.2.4

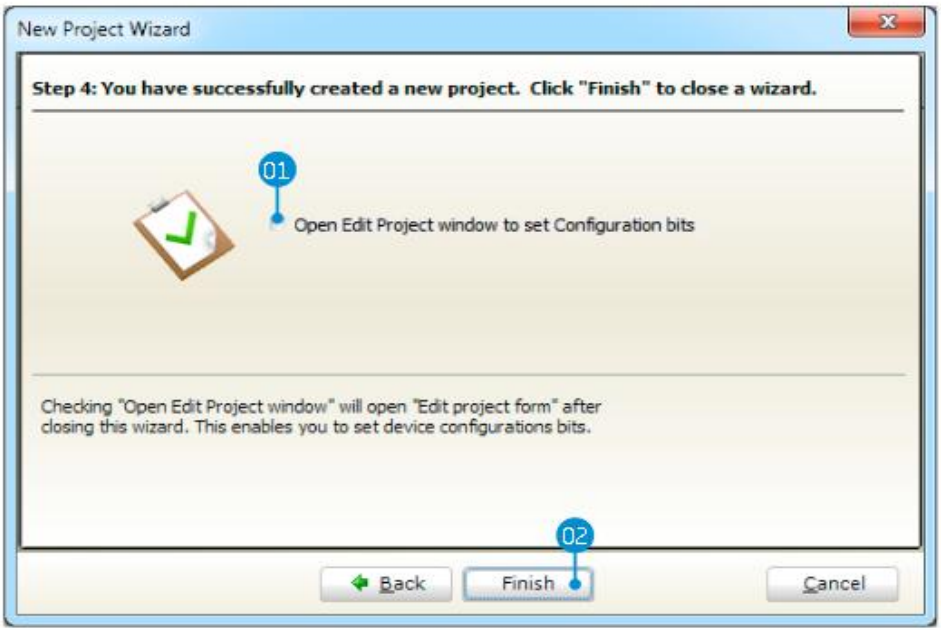
ეს ფანჯარა შესაძლებელია გამოვიყენოთ შედგენილ პროექტში ფაილების დამატებისათვის. ვინაიდან, ჩვენს მიერ შესასრულებელ ლაბორატორიულ სამუშაოებში ფაილების დამატება არ არის გათვალისწინებული, მას ყურადღებას არ ვაქცევთ, დავაჭიროთ **Next** ღილაკს. გაიხსნება ახალი ფანჯარა (სურ. 2.5).

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



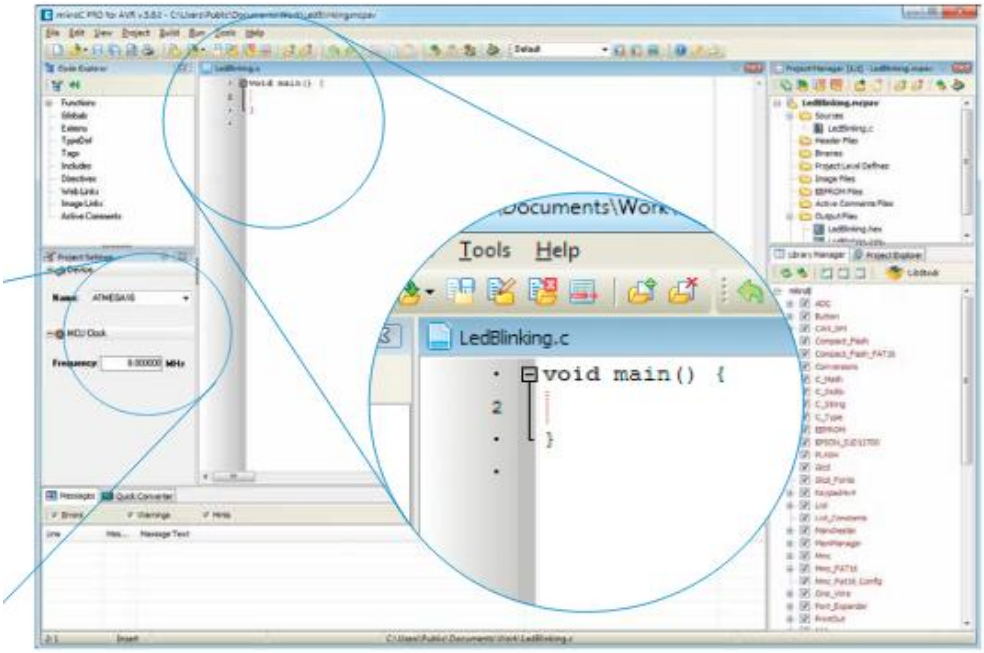
სურ. 2.5.

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



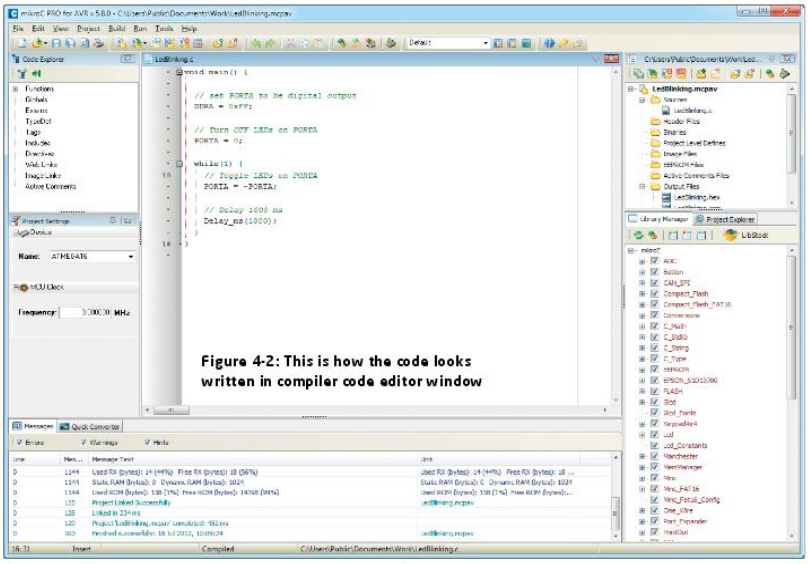
სურ.2.6.

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



სურ.2.7.

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



სურ.2.8

2.3 პროგრამის ტრანსლირება

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

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

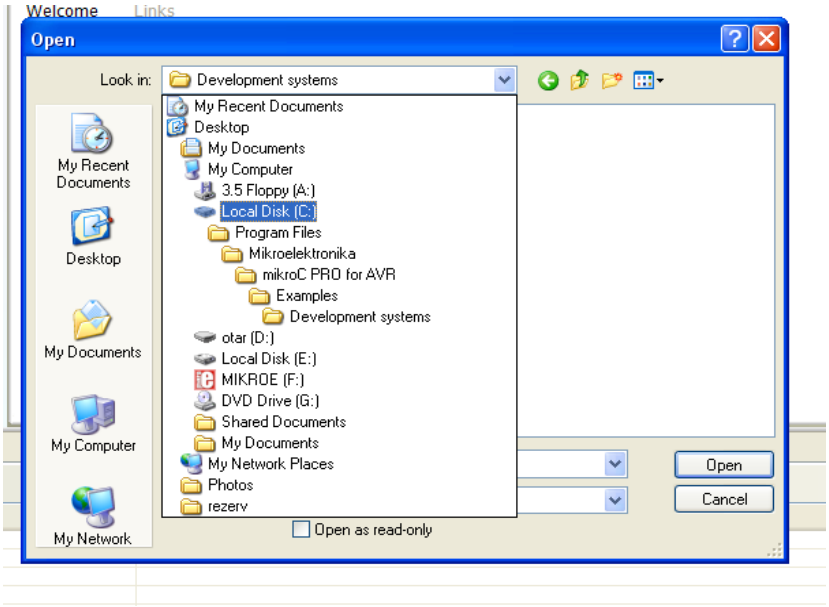
2.4 პროგრამის ჩაწერა მიკროკონტროლერში

ტრანსლირების ნორმალურად დამთავრების შემდეგ შესაძლებელია hex-ფორმატში წარმოდგენილი პროგრამის ჩაწერა მიკროკონტროლერში. ჩაწერა სრულდება პროგრამატორის საშუალებით, რომელიც ლაბორატორიულ სტენდზეა მოთავსებული და ჩასაწერ პროგრამას იღებს კომპიუტერიდან USB პორტის საშუალებით. პროგრამის ჩაწერისათვის კომპილიატორის **Tools** მენიუდან უნდა ამოვირჩიოთ ბრძანება **mE Programmer** ან ინსტრუმენტულ პანელზე დავაჭიროთ **Program** ღილაკს. ეკრანზე

გაიხსნება პროგრამის ჩაწერის ფანჯარა, რომლის ქვევით მოთავსებულ ინდიკატორზე აისახება ჩაწერის პროცესის მსვლელობა. სტენდზე იგივე პროცესის ინდიკაცია ხდება ნათურის ციმციმით. ჩაწერა სრულდება ავტომატურად. ამის შემდეგ თქვენს მიერ სტენდზე წინასწარ აწყობილი სისტემა იწყებს ფუნქციონირებას.

2.5 პროექტის გამოძახება

პროექტები ინახებიან ერთ-ერთ პაპკაში, რომელიც თქვენს მიერ მითითებული იყო ახალი პროექტის შექმნის დროს (იხ.სურ.2.3). პროექტის გამოძახებისათვის **Project** მენუში ამოვირჩიოთ ბრძანება **Open Project** ან ინსტრუმენტულ პანელზე დავაჭიროთ **Open Project** ღილაკს. ეკრანზე გაიხსნება ფანჯარა (სურ.2.9.), რომლის ზედა ნაწილში არსებულ გასაშლელ სიაში უნდა ვიპოვოთ ჩვენთვის საჭირო პროექტის დასახელება. **Open** ღილაკზე დაჭერით ეკრანზე გამოდის ამორჩეული პროექტის ფანჯარა.



სურ.2.9

2.6 პროექტის რედაქტირება

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

2.7 პროექტის დამახსოვრება

პროექტის შენახვისათვის **Project** მენიუდან უნდა ამოვირჩიოთ ბრძანება Save Project ან Save Project As…პირველ შემთხვევაში პროექტი ჩაიწერება მიმდინარე

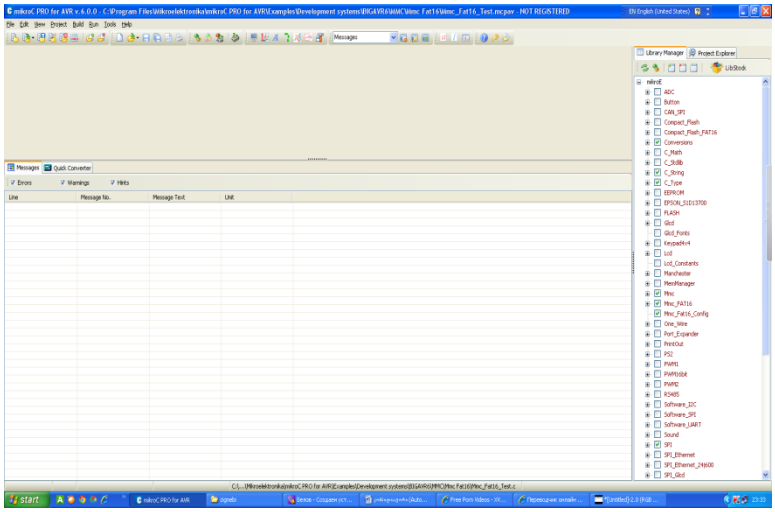
საქაღალდეში, რომლიდანაც იგი იყო ამოკითხული. მეორე შემთხვევაში ეკრანზე დგება ფანჯარა, რომელშიც შეგვიძლია ავირჩიოთ ჩვენთვის პროექტის ჩასაწერად სასურველი საქაღალდე.

2.8 პროექტის დახურვა

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

2.9 ფუნქციების ბიბლიოთეკა

პროგრამის დაწერის დროს მოხერხებულია ისარგებლოთ კომპილიატორის 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. თვითეულ ფუნქციაზე დაწკაპუნებით კი იხსნება ფუნქციის აღწერის ფანჯარა. კომპილიატორის ფუნქციების გამოყენება ბევრად აადვილებს პროგრამის შექმნის პროცესს.

თავი 3
ლაბორატორიული სამუშაოების აღწერა
ლაბორატორიული სამუშაო №1

გადართვადი შუქდიოდები

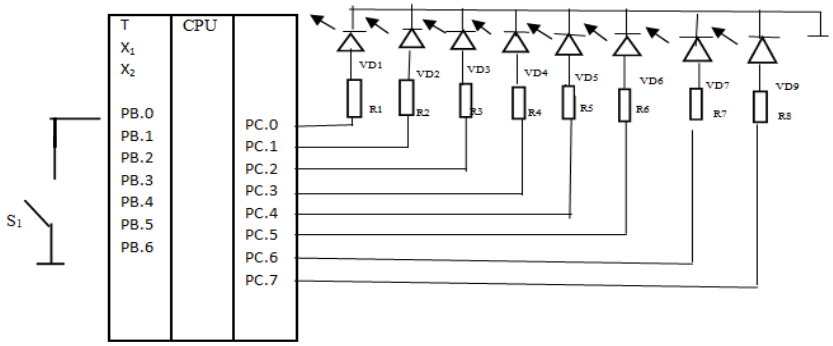
ამოცანა ჩამოვაყალიბოთ შემდეგ სახით:

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

პრინციპიალური ელექტრული სქემა

დავამუშაოთ ელექტრული პრინციპიალური სქემა, რომელიც შესძლებს ზემოთ ჩამოყალიბებული ამოცანის რეალიზაციას. მიკროკონტროლერს ჩვენ უნდა მიუერთოთ შუქდიოდები და მართვის ღილაკი. მიკროკონტროლერთან ნებისმიერი გარე მოწყობილობების მიერთებისთვის გამოიყენება შეტანა - გამოტანის პორტები. ამასთან ყოველ პორტს აქვს საშუალება იმუშაოს როგორც შეტანაზე, ასევე გამოტანაზე.

შუქდიოდები მიუერთოთ 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. ლაბორატორიულ სტენდზე გამოყენებულია Atmega 128 მიკროკონტროლერი., შუქდიოდების მიუერთოთ C პორტის თანრიგებს (PC0-PC7 გამომყვანები), ხოლო მართვის ღილაკიდან ინფორმაციის ამოკითხვისათვის - B პორტის უმცროსი თანრიგი (PB.0 გამომყვანი). მოწყობილობის სრული სქემა , რომელიც მოგვცემს საშუალებას განვახორციელოთ ჩვენს მიერ ჩამოყალიბებული ამოცანის რეალიზაცია მოცემულია სურ. 3.1 (აგრეთვე იხ. სურ.1.22)/



სურ.3.1

S1 ღილაკის მიერთებისთვის გამოიყენება კლასიკური სქემა. საწყის მდგომარეობაში ღილაკის კონტაქტი გათიშულია. დატვირთვის 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 ჩართვის დროს ხდება ძაბვის ნულამდე ვარდნა, რაც შეესაბამება ლოგიკურ ნულიანს. ამრიგად, პორტის შესაბამისი გამოსასვლელიდან სიგნალის მნიშვნელობის ამოკითხვით, პროგრამას შეუძლია განსაზღვროს ღილაკის დაჭერის მომენტი. შუქდიოდის მიერთება ასევე ხორციელდება კლასიკური სქემით. მათი უშუალო მიერთებით პორტის გამოსასვლელებთან. 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 ნოლი , ძაბვის ვარდნა შუქდიოდზე და რეზისტორზე ნულის ტოლი იქნება, რაც შუქდიოდის ჩაქრობას გამოიწვევს. მიზანშეწონილია გავაკეთოთ ზოგიერთი შენიშვნა: AVR ოჯახის უმრავლეს მიკროკონტროლერებს, გარდა გარე სატაქტო გენერატორებისა , ასევე აქვთ შიგა RC გენერატორი, რომელიც არ მოითხივს არავითარ გარე წრედებს. იმ შემთხვევაში თუ მოცემულ გენერატორს არ წაეყენება რაიმე მაღალი მოთხოვნა სიხშირის სიზუსტესა და სტაბილურობაზე, შიძლება ავირჩიოთ შიგა გენერატორი (როგორც ეს გაკეთებულია სტენდზე). 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.

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 გამოსასვლელებზე რომელიღაც კოდი;
- მოვახდინოთ PB პორტის კონფიგურირება შესვლაზე;

ოპერაციები რომლებიც ჰქმნიან 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.

კომპილატორის გაშვებისთანავე ყველა პარამეტები უთქმელობით დებულობენ მნიშვნელობებს (ყველა შიგა მოწყობილობები გამორთულია, შიგა შეყვანა-გამოყვანის პორტები გაწყობილი არიან შეყვანაზე, გამოიყენება შიგა გენერატორი 8 მჰც სატაქტო სიხშირით). ეს შეესაბამება მიკროკონტროლერის საწყის მდგომარეობას სისტემური განულების შემდეგ. ის პარამეტრები, რომლებიც ჩვენ არ გვჭირდება ამჟამად ხელს არ ვახლებთ (დავტოვებთ უთქმელობით).

პროგრამა C ენაზე:

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

2    Bit  oldstate ;          // ძველი მდგომარეობის ალამი

3   Void  main ()  {

4    DDB0  bit=0 ;        // PB0 გამომყვანის დაყენება  ხდება, როგორც შესასვლელის
5    DDRC =  0xFF;        //  PC კონფიგურირება ხდება, როგორც გამოსასვლელის
6    PORTC=0xAA;          // C პორტში ჩაიწერება საწყისი კოდი

7    Oldstate =0;

8    Do  {

9    If (Button (& PINB,0,1,1))   {  // ლოგიკური ერთიანის აღმოჩენა

10   Oldstate =1;                    // ალამის განახლება

       }
11   if  (oldstate  &&   Button  (& PINB, 0,1,0)) { // 1-დან 0-ში გადასვლის აღმოჩენა
12    PORTC= ~  PORTC;        //  PORTC ინვერტირება
13    oldstate=0;                    // ალამის გაახლება

14        }  While (1);               // უწყვეტი ციკლი
   }
```

წარმოდგენილ პროგრამაში სათაური და სხვადასხვა დანიშვნები, რომლებსაც ასრულებს კომპილიატორი, გამოტოვებულია, რამდენადაც ისინი უთქმელობით მიიღებიან. **Include** ბრძანება რომელიც 1-ელ სტრიქონშია განთავსებული აღმწერი ფაილის მიმაერთებელია. სტანდარტულ MikroC PRO პაკეტს აქვს მთელი რიგი აღმწერი ფაილების ნაკრები. AVR ოჯახის ყოველ მიკროკონტროლერს აქვს თავისი პაკეტი. პროგრამისტს ევალება მხოლოდ შეარჩიოს საჭირო ფაილი და ჩართოს ის პროგრამის შესაბამის სტრიქონში. ფაილის მიერთების გარეშე პროგრამა არ იმუშავებს. Atmega 128 მიკროკონტროლერისთვის ფაილს აქვს დასახელება Atmega 128.h.

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

3-ე 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 ხდება 0xAA კოდის შეტანა (სტრიქონი 6). oldstate ალამს ენიჭება ნული (სტრიქონი7).

8-ე სტრიქონიდან სრულდება უსასრულო ციკლი do-while. if ოპერატორის გამოყენებით ხორციელდება Button (& PINB,0,1,1) ფუნქციის მნიშვნელობის შემოწმება. ეს ფუნქცია წარმოადგენს სპეციალურ 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-ია). ფუნქცია აბრუნებს ლოგიკური ერთიანს დამაკმაყოფილებელი პასუხის შემთხვევაში ან ლოგიკურ ნულიანს - არადამაკმაყოფი- ლებელი პასუხის შემთხვევაში.

ამგვარად, 9-ე პუნქტში სრულდება ერთიანის შემოწმება PB პორტის 0 ბიტში. თუ აღნიშნულ ბიტში აღმოჩნდება ერთიანი (ღილაკი დაჭერილია), oldstar 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 (oldstar ალამის მნიშვნელობა 1-იანია) ე.ი ღილაკმა შეიცვალა თავისი მდგომარეობა - დაუჭერელ მდგომარეობიდან დაჭერილ მდგომარეობაში გადასვლით. ხორციელდება PC გამომყვანების ინვერტირება (სტრიქონიი 12) (შესაბამისად იცვლება შუქდიოდებზე გამოყვანილი კოდის მნიშვნელობა). თუ PB.0 მდგომარეობა არ იცვლება, შემოწმების დროს, მაშინ თვითეულ if ოპერატორის გამოსახულების მნიშვნელობა, რომელიც ჩასმულია ფრჩხილებში არ აკმაყოფილებს პირობას, მაშასადამე PC გამომყვანების მნიშვნელობა არ იცვლება. პროგრამა გადადის While (სტრიქონი 14) უწყვეტი ციკლის დასაწყისში.

ლაბორატორიული სამუშაოს დავალება:

1. გააკომპილირეთ მოცემული პროგრამა;
2. ჩაწერეთ მიკროკონტროლერში;
3. შეამოწმეთ სქემის მუშაობა;
4. შეცვალეთ საწყისი კოდი.

ლაბორატორიული სამუშაო №2

მოციმციმე შუქდიოდები 1

ჩამოვაყალიბოთ ამოცანა შემდეგ სახით:

დამუშავდეს მოწყობილობა, რომელიც შეასრულებს შუქდიოდების ოთხი ჯგუფის ციმციმის მართვას. თვითეულ 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. შუქდიოდები მივუერთოთ PA, PB, PC, PD პორტებს.

ვიზუალურად სქემის მუშაობა შესაძლებელია გამოიყურებოდეს შემდეგი სახით: პანელზე გამოვიყვანოთ შუქდიოდებისაგან შედგენილი მატრიცა, რომელიც შესდგება ოთხი სვეტისა და რვა სტრისტრიქონისაგან. სვეტები შეესაბამებიან A,B,C,D პორტებთან მიერთებულ შუქდიოდებს, ხოლო სტრიქონები - პორტის თანრიგებს დაწყებული ზევიდან ქვევით 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 გაწყობა

- PA, PB, PC, PD პორტები გავაწყოთ ინფორმაციის გამოტანაზე;
- PA, PB, PC, PD გამოსასვლელებზე ჩავწეროთ ნულები (ჩავაქროთ ყველა შუქდიოდები).

ოპერაციები რომლებიც ჰქმნიან პროგრამის ტანს

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

პროგრამა C ენაზე

პროგრამას აქვს შემდეგი სახე :

```
1.  Char counter;        // ცვლადის განისაზღვრა როგორც თანრიგების მთვლელის
2.  Void Wait ( ) {      // დაყოვნების ფუნქციის განსაზღვრა
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

26Wait  ( );                    // დაყოვნება

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

დასაწყისში ხდება დაყოვნების ფუნქციის გამოცხადება, რომელიც გამოიყენება შუქდიოდების მდგომარეობის გადართვის დროს (სტრიქონი 2) . შუქდიოდების მდგომარეობის დაყოვნება აუცილებელია მათი ვიზუალური აღქმისთვის. დაყოვნების გარეშე შუქდიოდის ციმციმი გაურჩეველი იქნება ადამიანის თვალისთვის.

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

შემდეგ იწყება უწყვეტი While ციკლი (13 სტრიქონი). დასაწყისში სრულდება For ოპერატორი (14-18 სყრიქონები), რომლებიც ასრულებენ თანმიმდევრულად ერთიანების შეტანას პორტებში. ყოველი ციკლის ბიჯზე პორტის შემცველობაზე სრულდება “ან” ოპერაცია უმცროსი თანრიგის მიმართ მარცხნივ დაძრულ წინა შემცველობასთან იმდენი თანრიგით, რომელსაც განსაზღვრავს Counter ცვლადი. შედეგის შენახვა ხორციელდება პორტში. პორტის თანრიგში ჩაწერილი აღმოჩნდება ერთიანები, რომლებიც ანთებანთებენ შუქდიოდებს. რამდენადაც ყოველი ციკლის 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.

განსაზრვრული 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 ნულთან, რომელიც მიიღება მარცხნივ დაძრული ერთიანის ინვერტირებით. ყოველ ბიჯზე ნული იწერება მომდევნო თანრიგში, რომელიც განსაზღვრულია counter ცვლადის მნიშვნელობით. ციკლის დასაწყისში counter ცვლადი მიუთითებს პორტის უმცირეს თანრიგზე, რის გამოც ნული ამ თანრიგში ჩაიწერება და შესაბამისი შუქდიოდი ჩაქრება.

ციკლის შესრულების პროცესში counter მნიშვნელობა იზრდება ერთით, ამით ის მიუთითებს პორტების მომდევნო თანრიგებზე, სადაც ჩაიწერება ნულიანები. შუქდიოდები ჩაქრება 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 (სყრიქონი 26), პროგრამა გადადის While ციკლის დასაწყისზე (სტრიქონი 13).

ლაბორატორიული სამუშაოს დავალება:

1. მიუერთეთ შუქდიოდები მიკროკონყროლერს გადამრთველების საშუალებით
2. გააკომპილირეთ მოცემული პროგრამა;
3. ჩაწერეთ მიკროპროკონტროლერში;
4. შეამოწმეთ სქემის მუშაობა;

ლაბორატორიული სამუშაო №3
მოციმციმე შუქდიოდები 2

ჩამოვაყალიბოთ ამოცანა შემდეგ სახით:

დამუშვდეს მოწყობილობა, რომელიც შეასრულებს შუქდიოდების ოთხი ჯგუფის ციმციმის მართვას. თვითეულ ჯგუფში შედის 8 შუქდიოდი. შუქდიოდები დასაწყისში უნდა ინთებოდეს ყველა ჯგუფში ერთდროულად, ხოლო შემდგომში უნდა მოხდეს 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 შუქდიოდები უნდა მივუერთოთ PA, PB, PC, PD პორტებს.

ვიზუალურად სქემის მუშაობა შესაძლებელია გამოიყურებოდეს შემდეგი სახით: პანელზე გამოვიყვანოთ შუქდიოდებისაგან შედგენილი მატრიცა, რომელიც შესდგება ოთხი სვეტისა და რვა სტრისტრიქონისაგან. სვეტები შეესაბამებიან A,B,C,D პორტებთან მიერთებულ შუქდიოდებს, ხოლო სტრიქონები - პორტის თანრიგებს დაწყებული ზევიდან ქვევით 0 დან 7-მდე.

ალგორითმი

აღვწეროთ პროგრამის შესრულების ალგორითმი

საწყისი გაწყობის ალგორითმი

- PA, PB, PC, PD პორტები გავაწყოთ ინფორმაციის გამოტანაზე;
- PA, PB, PC, PD გამოსასვლელებზე ჩავწეროთ ერთები (ჩავაქრობთ ყველა შუქდიოდებს).

ოპერაციები რომლებიც ჰქმნიან პროგრამის ტანს

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

პოგრამა 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 გამომყვანებზეს დიოდის  ჩაქრობა
8.   PORTB= 0x00;       // B პორტის გამომყვანებზეს დიოდის  ჩაქრობა
9.   PORTC= 0x00;       // C პორტის გამომყვანებზეს დიოდის  ჩაქრობა
10.  PORTD = 0x00;       // D პორტის გამომყვანებზეს დიოდის  ჩაქრობა
11.  Delay_ms (1000);    // დაყოვნება  1 წამი
12.  PORTA= 0xFF;       // A პორტის გამომყვანებზეს დიოდის  ანთება
13.  PORTB= 0xFF;       // B პორტის გამომყვანებზეს დიოდის  ანთება
14.  PORTC= 0xFF;       // C პორტის გამომყვანებზეს დიოდის  ანთება
15.  PORTD = 0xFF;       // D პორტის გამომყვანებზეს დიოდის  ანთება
16.  Delay_ms (1000);    // დაყოვნება  1 წამი
17.   } while ();
   }
```

პროგრამა შესდგება მხოლოდ ერთი main ფუნქციისაგან. უნქციის 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. Vინაიდან აღნიშნული პროცესი მეორდება უსასრულოდ, შუქდიოდები იწყებენ ციცმციმს.

ლაბორატორიული სამუშაოს დავალება

1. გააკომპილირეთ პროგრამა და ჩაწერეთ მიკროკონტროლერში;
2. გაუშვით პროგრამა და დააკვირდით სქემის მუშაობას სტენდზე;
3. მოცემული პროგრამის საფუძველზე შექმენით პროგრამა, რომლის საშუალებით იციმციმებს შუქდიოდების იმავე მატრიცაში რაიმე სიმბოლო (მაგალითად 9).

ლაბორატორიული სამუშაო №4

ანალოგური სიგნალის გარდასახვა ციფრულ ფორმაში (აცგ)

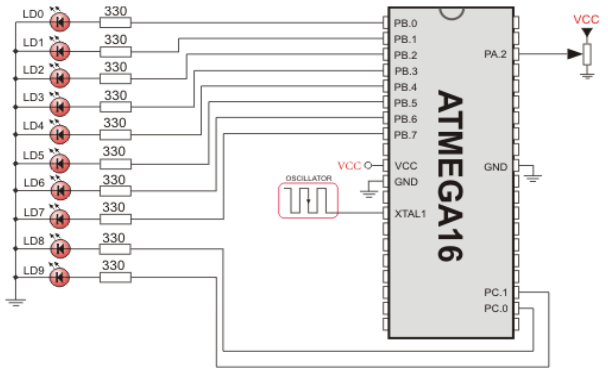
ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

დავაპროექტოთ 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 გამოსასვლელები, თითეული მათგანი შეიძლება მიუერთდეს მას, თუ ADMUX რეგისტრში ჩაწერილია შესაბამისი კოდი. ჩვენი პროექტისთვის ვირჩევთ PC.2 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 აღემატებოდეს კონტროლერის კვების ძაბვას. ამიტომ, იმ შემთხვევაში, როდესაც გარდასასახი სიგნალის მნიშვნელობა აღემატება მითითებულ შეზღუდვას, მაშინ ეს სიგნალი მიეწოდება 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).

გარდამქმნის შედეგის გამოტანისათვის გამოვიყენებთ შუქდიოდებს, რომელიც შეერთებული იქნებიან მიკროკონტროლერის პორტებთან. იმის გამო, რომ შედეგი წარმოდგენილია 10 თანრიგით, ხოლო თითეულ პორტს აქვს რვა თანრიგი, გამოვიყენებთ PB და PD ორ პორტს. PB პორტის მეშვეობით გამოვიყვანთ 8 უმცროს თანრიგს, ხოლო PD პორტის მეშვეობით - 2 უფროს თანრიგს. მოწყობილობის სქემა მოცემულია სურ.3.2.-ზე (აგრეთვე იხ. სურ.1.15,1.16)



სურ.3.2

ალგორითმი

საწყისი გაწყობის ოპერაციები

- გავაწყოთ PB და PD პორტების გამომყვანები გამოყვანაზე, ხოლო PC პორტის გამომყვანები შესვლაზე. დანარჩენი დაყენებები, როგორიც არიან აცგ-სთან PC პორტის კონკრეტული შესასვლელის მიერთება, აცგ-ს სამუშაო სიხშირე, წყვეტის ნებადართვის ალამი დავტოვოთ უთქმელობით.

ოპერაციები რომლებიც ქმნიან პროგრამის ტანს

- შესრულდეს შემომავალი სიგნალის გარდასახვა;
- გადაქმნის შედეგი გამოვიტანოთ შუქდიოდებზე.

პროგრამა C-ი ენაზე

```
1. #include  <built_in.h>; //გამომყვანების დასახელების ფაილის მიერთება
2. Unsigned int adc_rd;   // ორ ბაიტიანი ცვლადების 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** ოპერატორით (სტრიქონი 1), რომლის მეშვეობით კომპილატორი ახდენს იმ ფაილის მიერთებას, რომელიც განსაზღვრავს მიკროკონტროლერის ბლოკების გამომყვანებს. მომდევნო სტრიქონში (სტრიქონი 2) ხდება adc_rd ცვლადის გამოცხადება, როგორც უნიშნო ორბაიტიანის, რომელსაც ვიყენებთ გარდაქმნილი შედეგის შესანახად. შემდეგ იწყება main მთავარი პროგრამა. 4 და 5 სტრიქონებში ხდება B და D პორტების გაწყობა გამოყვანაზე. 6-ე სტრიქონიდან 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 გარდაქმნილი სიგნალი (ჩვენს შემთხვევაში გვაქს PC პორტის არხი 2). ციკლის დასრულების შემდეგ 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.

ლაბორატორიული სამუშაოს დავალება:

1. მიუერთეთ პოტენციომეტრი მიკროკონტროლერის გამოსასვლელებს J15 ჯამპერის საშუალებით (იხ. სურ.1.15, 1.16);
2. გააკომპილირეთ მიკროპროცესორის მუშაობის პროგრამა და ჩაწერეთ მიკროკონტროლერში;
3. პოტენციომეტრის საშუალებით ცვალეთ აცგ-ს შესასვლელზე მიწოდებული ძაბვა;
4. დააკვირდით შუქდიოდებზე გამოსულ კოდების მნიშვნელობებს.

ლაბორატორიული სამუშაო №5

ტექსტური დისპლეის მართვის მოწყობილობა

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

დავამუშაოთ მოწყობილობა, რომელსაც გამოჰყავს პროგრამაში ჩაწერილი ტექსტი ტექსტურ დისპლეიზე. ეკრანზე ტექსტი გადაადგილდება 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 (V_{CC}), მიწის (GND) და სიკაშკაშის მარეგულირებელი ძაბვის მიერთებისთვის.

სურ.3.3.

8 მონაცემების გამომყვანით შეიძლება მონაცემების და ბრძანებების 4- ან 8- ბიტიან გადაცემა. ჩვენს შემთხვევაში სრულდება მონაცემების და ბრძანებების 4 ბიტიანი გადაცემა, რაც იძლევა იმის საშუალებას, რომ მკ-ის ერთი 8 თანრიგიანი პორტი გამოვიყენოთ მონაცემების ან ბრძანებების გადაცემისათვის მმართველ სიგნალებთან ერთად. რამდენადაც ჩვენს შემთხვევაში D0-D3 გამომყვანები არ გამოიყენებიან, მათ მიწას (ანუ ლოგიკური 0) ვაწოდებთ.

(EN) -"ნების დართვის" გამომყვანი საშუალებას იძლევა, რომ მოხდეს მიმართვა დისპლეის შიგა მონაცემის რეგისტრთან, თუ გამომყვანზე ერთიანს მივაწოდებთ. ნოლიანის 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").

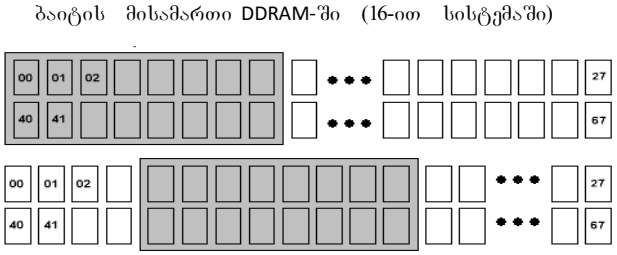
(RS) "რეგისტრის ამორჩევის" გამომყვანი განსაზღვრავს მონაცემთა სალტით გადაცემული ინფორმაციის ხასიათს: ბრძანებაა თუ მონაცემი.

დისპლეის მართვისათვის საჭიროა დისპლეის შესასვლელები მივუერთოთ მკ-ის PC პორტის გამოსასვლელებს. PC.2 გამოსასვლელი RS შესასვლელს, PC.3 – EN-ს, PC.4 - D4-ს, PC.5 - D5-ს, PC.6 - D6-ს , PC.7 – D7-ს.

LCD HD44780 კონტროლერის ლოგიკური სტრუქტურა

დისპლეის მართვას ანხორციელებს LCD HD44780 კონტროლერი, რომელიც ახდენს ბრძანებების დამუშავებას და შეიცავს სამი სახის მეხსიერებას:

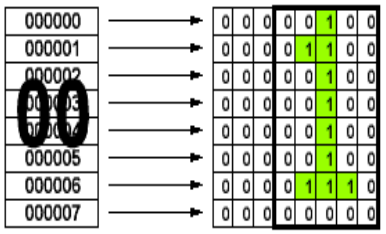
DDRAM – დისპლეის მეხსიერება. ის რაც ჩაწერილია **DDRAM** მეხსიერებაში მისი გამოტანა ხდება ეკრანზე. ეკრანზე სიმბოლოების გამოყვანა ხდება "ნიშნის ადგილებში", რომლებიც წარმოადგენენ სტრიქონისა და სვეტებისგან შექმნილ მატრიცებს . მათ გადაკვეთაზე მყოფი წერტილების მდგომარეობა (ჩამქრალი, თუ განათებული) გამოსახავენ 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.



სურ.3.4

CGROM – სიმბოლოების ცხრილია, რომელშიც მოთავსებულია სიმბოლოები მატრიცების სახით. როდესაც **DDRAM** უჯრედში ვწერთ სიმბოლოს, იგი გადაიწერება **CGROM** ცხრილიდან, რომელიც იხატება ეკრანზე. **CGROM** – ში 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 ნაჩვენებია სიმბოლოს წარმოდგენა მატრიცაში.



.

სურ.3.5

ბრძანებათა სისტემა

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

ცხრილი 3.18

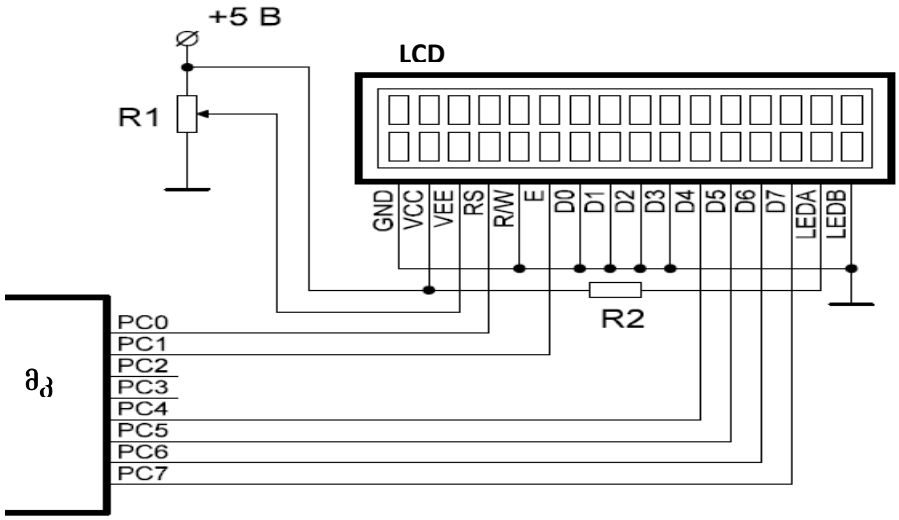
DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	მნიშვნელობები
0	0	0	0	0	0	0		ეკრანის გასუფთავება. მისამართების მთვლელი DDRAM ნულოვან პოზიციაზეა.
0	0	0	0	0	0	1	–	DDRAM-თან მიმართვა. ძვრების განულება. სამისამართო მთვლელში ნულის ჩაწერა
0	0	0	0	0	1	I/D	S	ეკრანის ძვრის და კურსორის გაწყობა
	0	0	0	1	D	C	B	ასახვის რეჟიმის გაწყობა.
0	0	0	1	S/C	R/L	–	1	კურსორის ან ეკრანის ძვრა,
0	0	1	DL	N	F	–	–	ხაზების რაოდენობის, (სალტის სიგანის) და სიმბოლოების ზომის ამორჩევა.
0	1	AG	AG	AG	AG	AG	AG	მისამართის გადართვა SGRAM–ზე და SGRAM-ის მისამართის ფორმირება.
	AD	AD	AD	AD	AD	AD	AD	გადაირთოს მისამართი DDRAM-ზე და დაფორმირდეს DDRAM-ს მისამართი

ბრძანების პარამეტრებს აქვს შემდეგი მნიშვნელობები:

- I/D – განსაზღვრავს სრულდება მისამართის მთვლელის ზრდა (ინკრიმენტი), თუ კლება (დეკრემენტი). თუ პარამეტრის მნიშვნელობა ნულია ყოველი მომდევნო ბაიტი ჩაიწერება n-1 უჯრედში. თუ 1 – მისამართი იზრდება მომდევნო ბაიტის ჩაწერის შემდეგ;

- **S –** ეკრანის ძვრა, თუ მივანიჭებთ 1-ს, ყოველი ახალი სიმბოლოს შეტანის შემთხვევაში მოხდება ეკრანის ძვრა, მანამ არ მიაღწევს DDRAM დასასრულს;
- D – დისპლეის ჩართვა. თუ მასში ჩავსვამთ 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 პარამეტრის მნიშვნელობა 0 – ია, მაშინ იძვრის კურსორი, თუ 1, მაშინ ეკრანი ერთი პოზიციით ყოველი ბრძანების შესრულების დროს;
- **R/L** – განსაზღვრავს კურსორის და ეკრანის ძვრის მიმართულებას. 0- მარცხნივ ძვრა, 1- მარჯვნივ ძვრა;
- **D/L –** ბიტი რომელიც განსაზღვრავს მონაცემის სალტის თანრიგების რაოდენობას. 1 - 8 ბიტის გადაცემა, 0 - 4 ბიტის გადაცემა;
- **N** – სტრიქონების რაოდენობა ეკრანზე. 0 - ერთ სტრიქონი, 1 - ორი სტრიქონი;
- **F** - სიმბოლოს ზომა: 0 – 5X8 წერტილი. 1 - 5X10 წერტილი (იშვიადათ გვხდება);
- **AG** – მისამართი **CGRAM** მეხსიერებაში;
- **AD**- მისამართი **DDRAM** მეხსიერებაში/

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



სურ.3.6

დისპლეის მართვისათვის მიუერთოთ დისპლეის შესასვლელები PC პორტის შესასვლელებს: PC.2 - RS, PC.3 –EN, PC.4 – D4, PC.5 –D5, PC.6 –D6, PC.7 –D7.

ალგორითმი

საწყისი გაწყობის ოპერაციები

1. მიკროკონტროლერის პორტის გამოსასვლელებისა და დისპლეის შესასვლელების მიერთება.
2. პროგრამაში გამოსაყვანი ტექსტის შეტანა.

ოპერაციები რომელიც ჰქმნის პროგრამის ტანს

1. ტექსტის გამოყვანა დისპლეიზე;
2. ტექსტს დაძვრა დისპლეიზე მარჯვნივ;
3. ტექსტს დაძვრა დისპლეიზე მარცხნივ;
4. ციკლის დასაწყისში გადასვლა.

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  DDC2_bit;

8. sbit  LCD  EN Direction at  DDC3_bit;

9. sbit  LCD  D4 Direction at  DDC4_bit;

10.  sbit  LCD  D5 Direction at  DDC5_bit;

11.  sbit  LCD  D6 Direction at  DDC6_bit;

12.  sbit  LCD  D7 Direction at  DDC7_  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. Void  main  ( )  {
21. Lcd_ Init  ();                    // DS1820 მოდულის ინიციალიზაცია
22. Lcd_ Cmd( _LCD_CLEAR);    // ეკრანის გასუფთავება
23. Lcd_ Cmd (_LCD_CURSOR_OFF); // კურსორის გამორთვა
// ტექსტის ეკრანზე გამოტანა
24. Lcd_Out  (1,6,txt3);
25. Lcd_Out  (2,6,txt4);
26. Delay_ ms(2000);
27.Lcd_ Cmd( _LCD_CLEAR);
28.Lcd_Out  (1,6,txt1);
29.Lcd_Out  (1,6,txt2);
30.Delay_ ms(2000);
//ტექსტის გადაადგილება
31.For  (i=0; I<4; i++) {
32.Lcd_Cmd (_LCD_SHIFT_RIGHT); // ტექსტის მარჯვნივ დაძვრა
33.Move_Delay ( );                    // ტექსტის დაყოვნება დისპლეიზე
}
34.While  (1) {
35.For  (i=0; I<7; i++) {
```

```
36.Lcd_Cmd (_LCD_SHIFT_LEFT);   // ტექსტის დაძვრა მარცხნივ

37.Move_Delay ( );              // დაყოვნება
        }

    }
```

პროგრამების პროექტირების დროს მოსახერხებელია გამოყენებული იყოს მზა ფუნქციები მკ-ის სხვადასხვა მოწყობილობებს შორის ურთიერთქმედებისთვის, რომლებსაც გვთავაზობენ კომპილატორის დამმუშავებლები. ეს ფუნქციები ჩვენ თავიდან გვაცილებს მათი აღწერის აუცილებლობას. ამ უკანასკნელიდან გამომდინარე მარტივდება პროგრამის შედგენის პროცესი. ზოგიერთს ამ ფუნქციებდან ჩვენ ვიყენებთ 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 დისპლეის RS გამომყვანსა და C პორტის გამომყვან 2-ს შორის.

შემდეგი ჯგუფი ფუნქციების (7-12 პუნქტები) განსაზღვრავენ გადაცემის მიმართულებას. მაგალითად, **sbit LCD_EN_Direction at DDC3_bit** ფუნქცია განსაზღვრავს C პორტის 3 არხს, როგორც გამოსასვლელს, რომელიც მიერთებულია EN დისპლეის შესასვლელთან.

13-16 სტრიქონებში გამოცხადებულია ტექსტის txt მასივები, რომელთა ელემენტები წარმოადგენენ ბაიტური მთელებს. მათში ვწერთ ტექსტს. რომელიც შემდგომში გამოდის დისპლეის ეკრანზე.

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

სტრიქონი 20-დან იწყება ტექსტის გამოყვანის მთავარი ფუნქცია main. პირველ რიგში სრულდება დისპლეის ინიციალიზაცია. ამისთვის ვიყენებთ მზა ფუნქციას **Lcd_ Init ( ).** ეს ფუნქცია აყენებს დისპლეის პარამეტრების საწყის მნიშვნელობებს (სტრიქონი 21).

სტრიქონ 22-ში ხდება ეკრანის გასუფთავება **Lcd_ Cmd(_LCD_CLEAR)** ფუნქციის საშუალებით. 23 სტრიქონში გამოირთვება კურსორი **Lcd_ Cmd (_LCD_CURSOR_OFF)** ფუნქციის გამოყენებით.

24 და 25 სტრიქონებში სრულდება ტექსტის გამოყვანა ეკრანზე. გამოყენებულია ფუნქცია **Lcd_Out (სტრიქონი, სვეტი, გამოსაყვანი ტექსტი).** ამ ფუნქციას გამოყავს ტექსტი ეკრანზე ფრჩხილებში პარამეტრების სახით მითითებული პოზიციიდან დაწყებული - პირველად ნაჩვენებია ეკრანზე სტრიქონის ნომერი, მეორედ - სვეტის ნომერი, მესამედ - ტექსტის შემცველი მასივი). პირველ ფუნქციას გამოჰყავს txt3 მასივში ჩაწერილი ტექსტი პირველი სტრიქონის მე-6 პოზიციიდან დაწყებული, მეორე ფუნქციას – txt4 მასივში ჩაწერილი ტექსტი მეორე სტრიქონის მე-6 პოზიციიდან.

შემდეგ სრულდება დაყოვნება 2მწ, რომ შესაძლებელი იყოს ტექსტის წაკითხვა ეკრანზე (სტრიქონი 26). 27 სტრიქონში ეკრანს კვლავ ვასუფთავებთ და შემდეგ 28,

29 სტრიქონებში გამოიყვანება txt1 და txt2-ის შემცველობები ეკრანის პირველ და მეორე სტრიქონებში, შესაბამისად. ისევ სრულდება დაყოვნება (სტრიქონი 30).

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

34 სტრიქონიდან დაწყებული სრულდება უსასრულო ციკლი **while,** რომლის შესრულების დროს ტექსტი მიმდევრობით იძვრის ეკრანზე ჯერ მარცხნივ ოთხი პოზიციით **Lcd_Cmd (_LCD_SHIFT_LEFT)** ფუნქციის გამოყენებით (სტრიქონი 35-37), შემდეგ მარჯვნივ ოთხი პოზიციით, ზემოთ ნაჩვენები ფუნქციის საშუალებით (სტრიქონი 38-40). ყოველი მიმართულებით ძვრის შემდეგ ვასრულებთ დაყოვნებას.

ლაბორატორიული სამუშაოს დავალება:

1. მიუერთეთ დისპლეი მიკროკონტროლერს SW15 გადამრთველების საშუალებით (იხ. სურ.1.25,1.26);
2. გააკომპილირეთ მიკროკონტროლერის მუშაობის პროგრამა;
3. შეცვალეთ გამოსაყვანი საწყისი ტექსტი;
4. შეცვალეთ ტექსტის გადაადგილების მიმართულება და პოზიციების რაოდენობა დისპლეიზე;
5. ჩაწერეთ პროგრამა მიკროკონტროლერში;
6. დააკვირდით სქემის მუშაობას

ლაბორატორიული სამუშაო №6

მიკროკონტროლერის მუშაობა ტემპერატურის გადამწოდთან

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:
ავაგოთ მოწყობილობა, რომელიც გარდასახავს ტემპერატურის გადამწოდიდან მიღებულ ანალოგურ სიგნალებს ორობით კოდში და გამოჰყავს მიღებული მნიშვნელობები ტექსტურ დისპლეიზე.

პრინციპიალური ელექტრული სქემა

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

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

პოპულიარული ტემპერატურის გადამწოდი არის Dallas Semicondactor ფირმის გადამწოდი DS18B20.

DS18B20 წარმოადგენს ციფრულ გადამწოდს გარდასახვის პროგრამირებადი გარჩევითობით.

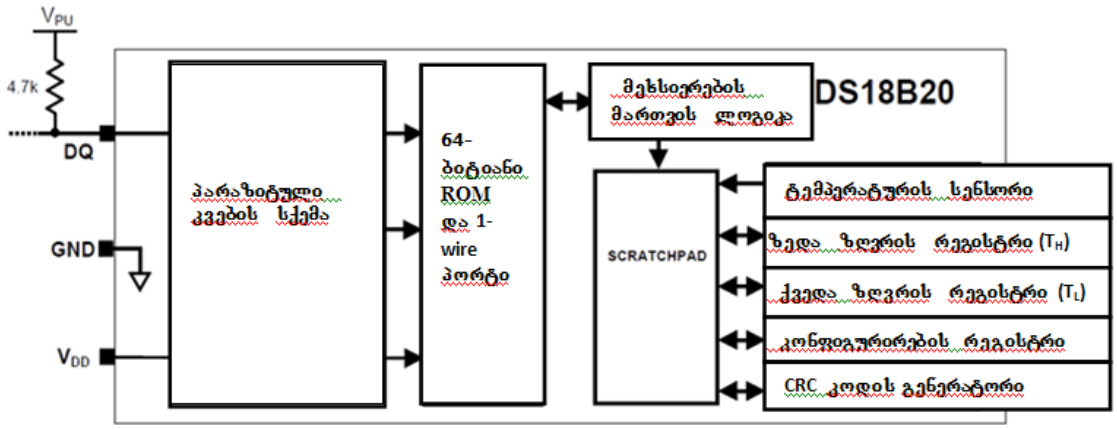
გადამწოდს აქვს შემდეგი სახე:

V_{cc} - კვების ძაბვის შესასვლელი;
GND - მიწა;
DQ - მონაცემთა შესასვლელი.

გადამწოდს გააჩნია შემდეგი თავისებურებები:

1) მართველ სისტემასთან ურთიერთობისათვის გამოიყენებს 1-wire ინტერფეისის მონაცემთა სალტეს;
2) უნიკალური 64-ბაიტიანი მიმდევრობითი საიდენტიფიკაციო კოდის გამოყენება, რომელიც განთავსებულია შიგა ROM- მეხსიერებაში და გათვალისწინებულია მრავალწერტილოვანი სისტემებისათვის, სადაც საჭირო არის კონკრეტული გადამწოდების დამისამართება;
3) გასაზომი ტემპერატურის დიაპაზონი შეადგენს -55 .+125^0 C;
4) გაზომვის სიზუსტე ±0,5ვ, თუმცა ეს მართებულია მხოლოდ დიაპაზონისათვის -10• • • +85^{0}C;
5) გარდასახვის გარჩევითობა განისაზღვრება მომხმარებლის მიერ და შეადგენს 9 …..12 ბიტს;
6) გააჩნია შიგა ქვედა და ზედა ზღვარის რეგისტრები, რომელთა საშუალებით გამომუშავდებიან განგაშის სიგნალები გაზომილი ტემპერატურის ზღვრის გადასვლის შემთხვევაში.

განვიხილოთ გადამწოდის სტრუქტურა (სურ.3.7)



სურ.3.7

მონაცემთა გადაცემა ხორციელდება DQ გამომყვანის საშუალებით, რომელიც ქმნის 1-wire ინტერფეისს. "პარაზიტული კვების" სქემა გამოიყენება ინტერფეისიდან კვების მიწოდების შემთხვევაში. ვინაიდან მოცემულ მოწყობილობაში 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 ლოგიკის ქვესისტემა ანხორციელებს 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 ორობით კოდში გარდასახვის შემდეგ, ეს კოდი ჩაიწერება Scratchpad მეხსიერებაში ორბაიტიანი სიტყვის სახით (ორ უჯრედში) და გარდამსახი კვლავ გადადის "ძილის" რეჟიმში.

ახლა განვიხილოთ, როგორ ხდება ტემპერატურის გადასახვა გადამწოდში. ტემპერატურის გადამწოდი შეიცავს აცგ-ს (ანალოგურ ციფრულ გადამსახს), და მის გამოსასვლელი რეგისტრიდან გარდასახვის შედეგად მიღებული კოდი გადაიტანება Scratchpad მეხსიერებაში რომელსაც აქვს შემდეგი სახე:

	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 (თუ 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 4-11).

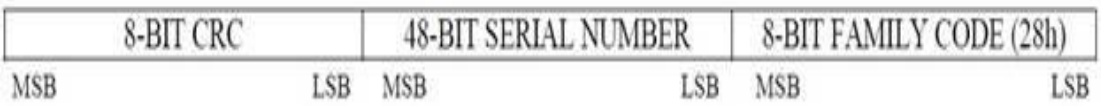
განგაშის სიგნალი - თერმოსტატის ფუნქცია

ამ მიზნით გათვალისწინებულია 2 8-ბიტიანი რეგისტრი T_H და T_L. T_H შეიცავს ტემპერატურის ზედა ზღვარის მნიშვნელობას, T_L- ქვედა ზღვარის მნიშვნელობას. თუ ტემპერატურა T_H მნიშვნელობაზე მეტია ან T_L მნიშვნელობაზე ნაკლებია ყენდება განგაშის ალამი. ამ ალამის აღმოჩენა ხდება წამყვანი მოწყობილობის მიერ Alarm Search ბრძანების გაცემით DQ ხაზზე. განგაშის ალამი ახლდება ტემპერატურის ყოველი გარდასახვის ოპერაციის შემდეგ. ამასთან, T_H და T_L რეგისტრებთან შედარებისათვის გამოიყენება მხოლოდ ბიტები 4-11, 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

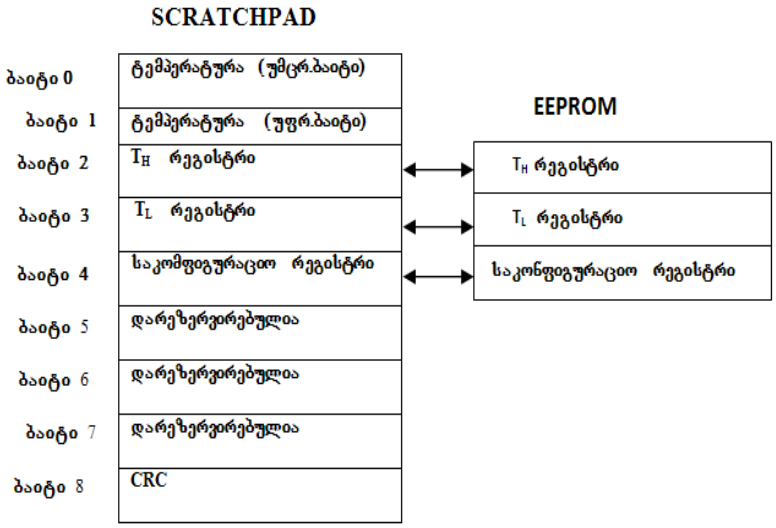
ეს კოდი, როგორც ადრე იყო აღნიშნული, საჭიროა თითეული მოწყობილობის იდენტიფიკაციისათვის მრავალწერტილიან ტემპერატურის გაზომვის სისტემაში.

ამ მეხსიერებას აქვს ქვევით ნაჩვენები ფორმატი:



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.

მეხსიერების ორგანიზაცია (სურ.3.8)



სურ.3.8

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

უნდა აღინიშნოს, რომ თუ თერმოსტატის ფუნქცია არ გამოიყენება, T_H და T_L შეიძლება გამოყენებული იყვნენ როგორც საერთო დანიშნულების მეხსიერება.

მონაცემები იწერებიან 2,3,4 ბაიტებში მიმდევრობით, დაწყებული მე-2 ბაიტის უმცროსი ბიტიდან Write Scrathcpad 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 შენარჩუნდებიან. კვების ჩართვის შემთხვევაში კი შესაბამის EEPROM უჯრედებიდან გადაიტვირთებიან 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

საკონფიგურაციო რეგისტრში მომხმარებლის მიერ გამოიყენება მხოლოდ ორი ბიტი: R0 და R1. ეს ბიტები განსაზღვრავენ ტემპერატურის გარდასახვის 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

გარჩევითობა, მით უფრო მეტია გარდასახვისათვის საჭირო დრო.

ცხრილი 3.2

R0	**R1**	**გარჩევითობა**	**მაქს. არდასახვის დრო**	
0	0	9-ბიტი	93,75 მწ	($t_{გარდ}$/8)
0	1	10-ბიტი	187,5 მწ	($t_{გარდ}$/4)
1	0	11-ბიტი	375 მწ	($t_{გარდ}$/2)
1	1	12-ბიტი	750 მწ	($t_{გარდ}$)

მონაცემთა გადაცემის ოპერაციების თანმიმდევრობა

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

1) ინიციალიზაცია;
2) ROM ბრძანება (რომელსაც მოყვება მონაცემთა საჭირო გაცვლა);
3) გადამწოდის ფუნქციონალური ბრძანება (რომელსაც მოყვება მონაცემთა საჭირო გაცვლა).

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

ამგვარად, ყველა გადაცემა იწყება ინიციალიზაციით. ეს ოპერაცია სრულდება წამყვანი მოწყობილობის მიერ მიმყოლისთვის ნულზე დაყენების სიგნალის გაგზავნით, რომელის საპასუხოდ მიმყოლი მოწყობილობები (ჩვენ შემთხვევაში ტემპერატურის გადამწოდი) უგზავნიან წამყვანს სიგნალს ყოფნის შესახებ, რომლითაც ატყობინებენ მას, გადამწოდები მიერთებულია და არიან მუშაობისთვის მზადყოფნაში.

საერთოდ 1-wire ინტერფეისის სალტე, რომელიც რეალიზებულია გადამწოდში, მონაცემთა სალტეზე იყენებს სიგნალების რამოდენიმე ტიპს: განულების იმპულსს, ყოფნის იმპულსს, 0-ის ჩაწერას, 1-ის ჩაწერას, 0-ის ამოკითხვას, 1-ის ამოკითხვას. ყველა აღნიშნულ ოპერაციებს ანხორციელებს წამყვანი მოწყობილობა. გამონაკლისს წარმოადგენს ყოფნის იმპულსი, რომელსაც აფორმირებს მხოლოდ გადამწოდი.

გადამწოდის ბრძანებები

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

ROM- ბრძანებები

ეს ბრძანებები იხმარებიან გადამწოდის ინიციალიზაციის შემდეგ და შეიცავენ ინსტრუქციებს გადამწოდთან მუშაობის შესახებ. თვითოეული ბრძანება 8-თანრიგაა. შესაბამისი ROM ბრძანების შესრულების შემდეგ შესაძლებელია ფუნქციონალური ბრძანების შესრულება ((ცხრილი 3.3.). .

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. ეს შესაძკლებლობას 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 ოპერაციებს.
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

ფუნქციონალური ბრძანებები

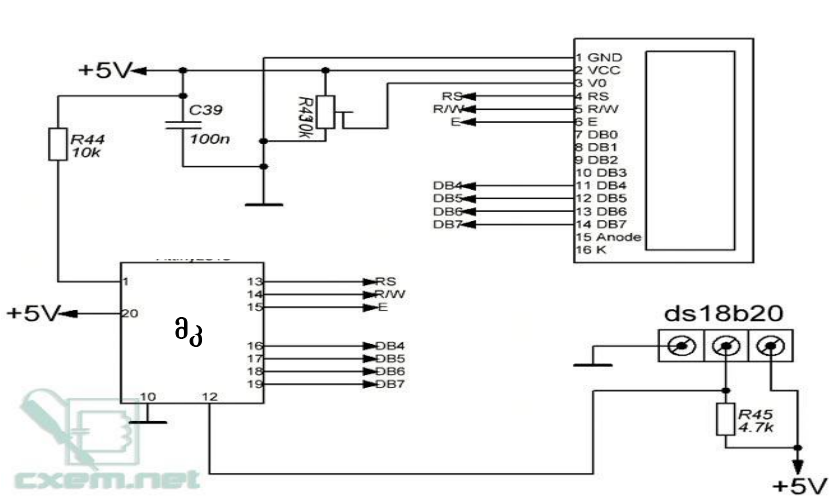
ამ ბრძანებების საშუალებით სრულდება სხვადასხვა მოქმედებები ტემპერატორის სენსორის ინტეგრალურ სქემაში, დაკავშირებული მის ფუნქციონირებასთან (ცხრილი 3.4.)

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, 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 გადაწერს EPROM-იდან scratchpad-მეხსიერების შესაბამის ადგილებში. 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.

მოწყობილობის პრინციპიალური სქემა

სურ.3.9-ზე მოცემულია მოწყობილობის პრინციპიალური სქემა.



სურ.3.9

შემოთავაზებულ სქემაში ტექსტური დისპლეი მიერთებულია C პორტთან: PC.2-დისპლეის RS შესასვლელთან, PC.3- EN შესასვლელთან, PC.4- D4-თან, PC.5-D5-თან, PC.6-D6-თან, PC.7-D7-თან. ტემპერატურის გადამწოდის შესასვლელი მიერთებულია B პორტის PB.0 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 შეადგენენ პროგრამის ტანს

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

პროგრამა C-ზე

```
// 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 DDC2_bit;
8.  sbit LCD_EN_Direction at DDC3_bit;
9.  sbit LCD_D4_Direction at DDC4_bit;
10. sbit LCD_D5_Direction at DDC5_bit;
11. sbit LCD_D6_Direction at DDC6_bit;
12. sbit LCD_D7_Direction at DDC7_bit;


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-ს  გაცემა  გადამცემის  სალტეზე  PORB-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); // შედეგის  გაცემა  დისპლეიზე

54. Delay_ms (500);
        }
55. while (1);  //  უსასრულო  ციკლის  დასაწყისზე  გადასვლა
}
```

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

შემდეგ სრულდება ზოგიერთი ცვლადების გამოცხადება (სტრიქონები 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 რეზულტატი.

სტრიქონ 16-დან განისაზღვრება ფუნქცია **Display_Temperature**, რომელსაც გადამწოდიდან წაკითხული მნიშვნელობები გამოჰყავს დისპლეიზე. გარდასახვის მიღებული შედეგი შეიცავს მთელ და წილად ნაწილს. 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 (**სტრიქონი 17**).** სტრიქონში 18 და 19 წარმოდგენილია ცვლადები

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

როგორც ზევით იყო ნათქვამი, ტემპერატურის მნიშვნელობა შეიძლება იყოს როგორც დადებითი, ისე უარყოფითი. უარუოფითი 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 ამოკითხული მნიშვნელობა), თუ აღნიშნულ თანრიგში იმყოფება 1 (ე.ი. უარყოფითი რიცხვია), მაშინ text მასივის პირველ ელემენტში ჩაიწერება სიმბოლო ” - “ (სტრიქონი 21) და რიცხვი გარდაიქმნება პირდაპირ კოდში (სტრიქონი 22).

41 სტრიქონში სრულდება ტექსტის ” Temperature : “ გამოყვანა ეკრანის პირველი სტრიქონის პირველი პოზიციიდან დაწყებული.

42 სტრიქონში ფუნქციას ეკრანზე გამოყავს გრადუსის აღმნიშვნელი სიმბოლო (223 მნიშვნელობა არის ამ სიმბოლოს კოდი).

43 სტრიქონში ხდება ‘C’ სიმბოლოს გამოყვანა ეკრანზე.

44 სტრიქონში 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 (სტრიქონი 45). ამის შემდეგ პორტის იგივე გამოსასვლელიდან გაიცემა ბრძანება **SKIP_ROM,** რომელიც ამზადებს გადამწოდს შემდეგი ფუნქციონალური ბრძანების შესასრულებლად (სტრიქონი 46). ფუნქცია **Ow_ Write (& PORTB,0, 0x44)** ასრულებს გადამწოდისთვის ***CONVERT_T*** ბრძანების გადაცემას, რომლის საფუძველზე იწყება ტემპერატურის გარდასახვის პროცესი (სტრიქონი 47). გარკვეული დაყოვნების შემდეგ, რომელიც საჭიროა გარდასახვის პროცესის დამთავრებისთვის, გადამწოდი კვლავ გადაყავთ საწყის მდგომარეობაში (სტრიქონი 49) შესასვლელ სალტეზე 0-ის მიწოდებით. სტრიქონ 50-ში კვლავ გადაიცემა **SKIP_ROM** ბრძანება. სტრიქონ 51-ში გაიცემა **READ_SCRATCHPAD** გადამწოდის მეხსიერებიდან წაკითხვის ბრძანება . 52 და 53 სტრიქონებში 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 დისპლეიზე.

გარკვეული დაყოვნების შემდეგ პროგრამა გადადის ციკლის დასაწყისში სტრიქონ 37-ზე და განხილული მოქმედებები მეორდებიან.

ლაბორატორიული სამუშაოს დავალება:

1. გადართველების საშუალებით დაუკავშირეთ ერთმანეთს მოწყობილობის კომპონენტები;
2. სილიუსტრაციო პროგრამაში გადაადგილეთ ეკრანზე გამოსაყვანი ტექსტის სტრიქონები;

3. გააკომპილირეთ პროგრამა და ჩაწერეთ მიკროკონტროლერში;
4. შეამოწმეთ მოწყობილობის მუშაობა სხვადასხვა ტემპერატურის მნიშვნელობისათვის,

ლაბორატორიული სამუშაო №7
რეალური დროის საათი

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

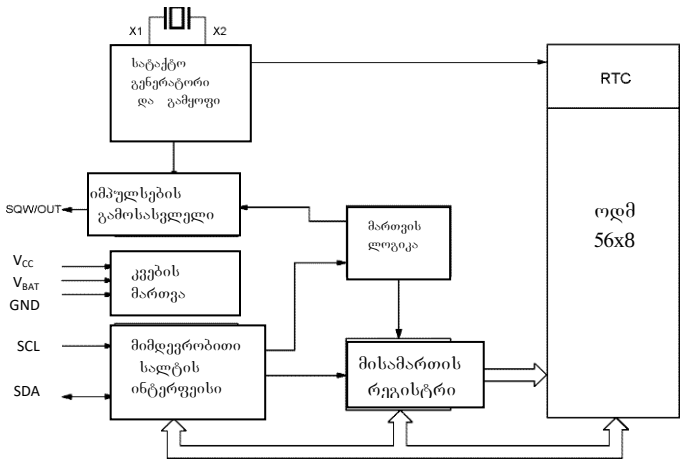
დავამუშაოთ მოწყობილობა, რომელსაც დროის გარკვეულ ინტერვალებში ტექსტურ დისპლეიზე გამოჰყავს რეალური დროის და კალენდარის მნიშვნელობები.

პრინციპიალური ელექტრული სქემა:

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

ინტეგრალური სქემა DS1307

მიმდევრობითი ინტერფეისიანი რეალური დროის საათი DS1307 (RTC) წარმოადგენს ორობით-ათობით საათ-კალენდარს, რომელიც შეიცავს 56 ბაიტიან ენერგოდამოუკიდებელ სტატიკურ ოპერატიულ მეხსიერებას. მისამართების და მონაცემების გადაცემა ხორციელდება ორგამტარიანი, ორმიმართულებიანი სალტით TWI. საათი-კალენდარი ითვლის წამებს, წუთებს, საათს, კვირის დღეებს, თარიღს, თვეებს 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.3.10-ზე მოცემულია RTC ინტეგრალური სქემის სტრუქტურა.



სურ.3.10

გამომყვანების აღწერა

V_{cc}, GND – ამ გამომყვანებზე მიეწოდება კვება. **V_{cc}** შესასვლელზე მიეწოდება +5ვ. როდესაც კვების ძაბვა მეტია **1.25*V_{BAT},** მოწყობილობა მიღწევადია და

შესაძლებელია მასში მონაცემთა ჩაწერა ან ამოკითხვა. 3ვ-იან ბატარეის მიერთების შემთხვევაში, თუ V_{cc}-ს მნიშვნელობა გახდება **1.25*V_{BAT}** –ზე ნაკლები, წაკითხვა ან ჩაწერა ვერ სრულდება, მაგრამ დროის თვლის ფუნქცია აგრძელებს მუშაობას. როგორც კი V_{cc} მნიშვნელობა შემცირდება V_{BAT}-ზე ქვევით, ოპერატიული მეხსიერება და RTC გადაერთვება ბატარეის კვებაზე.

V_{BAT} – შესასვლელი ნებისმიერი სამვოლტიანი ბატარეისთვის ან ენერგიის სხვა წყაროსათვის. DS1307 სქემის ნორმალური მუშაობისთვის აუცილებელია, რომ ბატარეის ძაბვა იყოს 2.0.....2.5ვ დიაპაზონში.

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

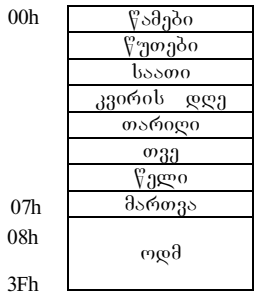
SDA (Serial Data Input/Output- მიმდევრობითი მონაცემების შესავალ-გამოსავალი) მონაცემების შესასვლელ/გამოსასვლელი ორწვერიანი მიმდევრობითი ინტერფეისისათვის. SDA გამოსასვლელი მოითხოვს გარე დატვირთვის რეზისტორის მიერთებას.

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

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

RTC მეხსიერების დამისამართების ქარტა

RTC მეხსიერების რეგისტრების დამისამართების ქარტა მოცემულია სურ.3.11-ზე. RTC-ს რეგისტრები განთავსებული არიან 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 შემდეგ და მიუთითებს შემდეგ უჯრედზე. როდესაც უჯრედებთან მრავალჯერ მიმართვის შედეგად მისამათის მაჩვენებელი მიაღწევს 3Fh მნიშვნელობას, მასში იწერება 00h - RTC-ს პირველი უჯრედის მისამართი.



სურ.3.11

საათი და კალენდარი

საათისა და კალენდარის შესახებ ინფორმაციის წაკითხვა ხდება მეხსიერების შესაბამისი რეგისტრებიდან. RTC მოდულის რეგისტრები ნაჩვენებია ცხრილ 3.5-ში. დროისა და კალენდარის რეგისტრების შემცველობა წარმოდგენილია ორობით-ათობით ფორმატში. რეგისტრ 0-ის მე-7-ე ბაიტი 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 მე-6-ე ბიტი გათვალისწინებულია რეჟიმის დასანიშნად: როდესაც ჩავწერთ 1-ს, ამორჩეული იქნება 12-საათიანი რეჟიმი. ამ რეჟიმში ბიტ 5-ში ფორმირდება აღნიშვნა AM/PM (შესაბამისად, დღის პირველი ნახევარი/ დღის მეორე ნახევარი). 24-საათიან რეჟიმში ეს ბიტი გამოიყენება საათის მეორე ათეულის ჩასაწერად (20-23 საათი).

ცხრილი 3.5.

	ბიტი7	6	5	4	3	2	1	ბიტი0
00h	CH	ათეული წამები			წამები			
	0	ათეული წუთები			წუთები			
	0	12 / 24	საათის მეორე ათეული	საათის პირველი ათეული	საათი			
			AM/PM					
	0	0	0	0	0	დღე		
	0	0	თარიღის ათეულები		თარიღი			
	0	0	0	თვის ათეულები	თვე			
07h	წლის ათეულები				წელი			

ორგამტარიანი მიმდევრობითი მონაცემთა 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 სინქროიმპულსებს (serial clock- SCL) და ქმნის START და STOP მდგომარეობას. DS1307 მოდული მუშაობს როგორც მიმყოლი, მიკროკონტროლერი - როგორც წამყვანი..

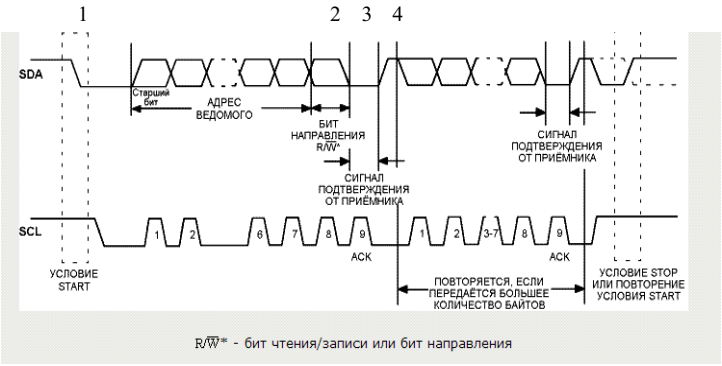
TWI სალტით მონაცემთა გადაცემის პრინციპი

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 გადაიცემიან START და STOP მდგომარეობებს შორის არ არის შეზღუდული და განისაზღვრება წამყვანი მოწყობილობის მიერ. ინფორმაციის გადაცემა სრულდება ბაიტებით და თვითოეული ბაიტის მიღებას მიმღები ადასტურებს მე9-ე ბიტით (დასტურის ბიტი- ACK). DS1307 მოდული მუშაობს 100კჰც სიხშირით.

ს‘ურ.3.12-ზე ნაჩვენებია მონაცემთა გადაცემის დროითი დიაგრამა ორგამტარიანი სალტით.



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

ს‘ურ.3.12

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 პასუხობს ყოველი ბაიტის მიღების შემდეგ დასტურის ბიტით. მონაცემები იგზავნებიან უფროსი ბაიტიდან დაწყებული.
2) მონაცემთა გაგზავნა მიმყოლი გადამცემიდან წამყვან მიმღებისკენ.
   პირველი ბაიტი (მიმყოლის მისამართი) გადაეცემა წამყვანს. წამყვანი პასუხობს დასტურის ბიტით. მის 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

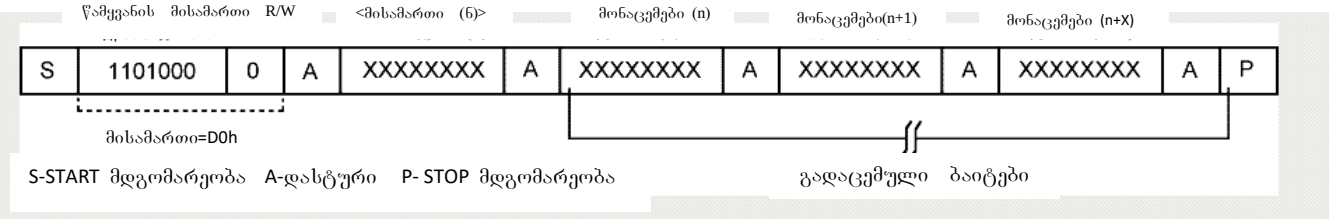
START - STOP მდგომარეობას. გადაცემა მთავრდება STOP მდგომარეობით ან START მდგომარეობის გამეორებით. ვინაიდან START მდგომარეობის გამეორება ნიშნავს შემდეგი გადაცემის დაწყებას, სალტე არ თავისუფლდება.

DS1307 მოდულს შეუძლია იმუშაოს შემდეგ ორ რეჟიმში:

1) წამყვანი მიმღების რეჟიმი (DS1307-ში ჩაწერის რეჟიმი)

მონაცემები მიმდევრობით მიიღებიან SDA გამომყვანით და სინქრონიზირდებიან SCL სიგნალებით. START და STOP მდგომარეობები აღიქმებიან როგორც მიმდევრობით გადაცემის დასაწყისი და დამთავრება. მისამართის გამოცნობა სრულდება აპარატურულად მიმყოლის მისამართის და მიმართულების ბიტის მიღების შემდეგ (სურ. 3.13).

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. თვითეული მათგანის მიღება DS1307-ს მიერ დასტურდება შეტყობინებით. მონაცემთა გადაცემის დამთავრებისათვის მიმყოლი აფორმირებს დამთავრების მდგომარეობას.

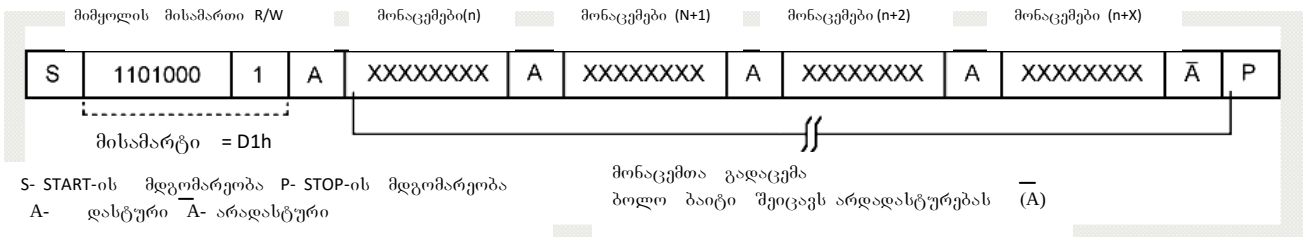


სურ.1.13

2) წამყვანი გადამცემის 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 მდგომარეობების გამოცნობა ხორციელდება მიმდევრობითი გადაცემის დასაწყისში და დასრულების დროს. (სურ. 3.14.).

მისამართის მიღების და დეკოდირების შემდეგ DS1307 იწყებს მონაცემთა გაცემას მისამართიდან დაწყებული, რომელიც იმყოფება მისამართის მაჩვენებელში. თუ წაკითხვის წინ მისამართის მაჩვენებელში არ ჩავწერთ მისამართს, მაშინ პირველი წაკითხული მისამართი იქნება ის მისამართი რომელიც შენახული იყო მაჩვენებელში ბოლოს.



სურ.3.14.

დასაგეგმარებელი მოწყობილობის სტრუქტურა

როგორც ზევით 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: PD.1 - მონაცემთა SDA სალტეს. PD.0 - 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 ანხორციელებს 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  0 წმ
  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 (4 სტრიქონი) უნდა შეიცავდეს 7 თანრიგა მისამართს, რომელსაც RTC მოდულისათვის აქვს თექვსმეტობითი მნიშვნელობა $68 (ორობით კოდში 1101000) რომელსაც მიეწერება 0-ჩაწერის ნიშანთვისება. მთლიანად გადასაცემი ბაიტი იქნება $D0 (.**TWI_Write (0xD0**). შემდეგ. 5 სტრიქონზე ბაიტების მაჩვენებელში იწერება 0 (**TWI_Write (0)),** რომელიც მიუთითებს პირველ REG0 რეგისტრზე **,** საიდანაც იწყება მონაცემთა ჩაწერა მოდულის უჯრედებში.

6 სრიქონზე 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 ყენდება 12-საათიანი რეჟიმი REG4-ის ბიტ 6-ში 1-ის შეტანით (სტრიქონი 8). მონაცემთა გადაცემა სრულდება წამყვანის მიერ (მიკროკონტროლერი) დასრულების მდგომარეობის ფორმირებით (სტრიქონი 13, ფუნქცია **TWI_Stop ( )**).

შემდეგ სრულდება მოდულთან განმეორებით მიმართვა მისამართის მაჩვენებლის ნულში დაყენების მიზნით (REG0 რეგისტრის მისამართი) მოდულის მომზადებისათვის წასაკითხად და მისი ჩართვისათვის. ახალი მიმართვის დასაწყისში მიკროკონტროლერი აფორმირებს გაშვების მდგომარეობას (14 სტრიქონი, 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 ნულის 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 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);      // მისამართის მაჩვენებელში ნულის ჩაწერა
19. TWI Stop ( );       // გადაცემის დამთავრების მდგომარეობის ფორმირება

20. TWI Start ( );      // გადაცემის დასაწყისის მდგომარეობის ფორმირება
21. TWI Write (0xD1); // მოდულის მისამართის გაგზავნა 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 სიმბოლოებად 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 მნიშვნელობების გამოყვანის ფუნქციის 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 განსაზღვრული არიან ცვლადები და სიმბოლოების მასივი, რომლებიც გამოიყენებიან 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 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 (სტრიქონი 20), ამის შემდეგ მიკროკონტროლერი გასცემს SDA სალტით მოდულის მისამართს წაკითხვის ნიშანთვისებასთან ერთად (21 სტრიქონი, ფუნქცია **TWI Write (0xD1)**) (ბაიტური კონსტანტა 0xD1 ფუნქციის ფრჩხილებში შეიცავს 7 თანრიგა მოდულის მისამართს + ერთთანრიგა წაკითხვის ალამი -1).

22- 28 სტრიქონებზე სრულდება მოდულის მეხსიერების რეგისტრებიდან მნიშვნელობების თანმიმდევრული ამოკითხვა და ჩაწერა უჯრედებში, რომლებიც აღნიშნული არიან 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. ათობით სიმბოლოებით დისპლეიზე გამოსაყვანად საჭიროა მათი წარმოდგენა ორობით ფორმატში. ამისათვის მონაცემთა უფროს და უმცროს ტეტრადებზე ნიღბების ზედდებით სრულდება ათეულების და ერთეულების გამოყოფა. შემდეგ უფროსი ტეტრადა მრავლდება 10-ზე (ენიჭება 10 წონა) და იკრიბება უმცროს ტეტრადასთან. მიღებული შედეგი არის ორობით-ათობითი რიცხვის ორობით ფორმატში წარმოდგენა. 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 (სტრიქონი 39).

40-51 სტრიქონებში სრულდება დროს და კალენდარის 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-ე პოზიციაზე, ხოლო ფუნქცია **Lcd_Chr (1,7, (day%10) + 48)** ანგარიშობს დღეების უმცროს თანრიგის (ერთეულების) მნიშვნელობას და გამოყავს იგი ეკრანის პირველი სტრიქონის მე-7-ე 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 (სტრიქონი 53). მომდევნო ფუნქციებით სრულდება ეკრანის

გასუფთავება, კურსორის გამოთიშვა და TWI ბლოკის დაყენება წამყვანად. შემდეგ სრულდებიან დისპლეიზე გამოყვანის ბრძანებები.

66-72 სტრიქონებზე წარმოდგენილია მთავარი ფუნქცია, რომელშიც თანმიმდევრობით სრულდებიან ზევით აღწერილი ფუნქციები **while** უსასრულო ციკლში.

ლაბორატორიული სამუშაოს დავალება:

1. დაუკავშირეთ მიკროკონტროლერის გამოსასვლელები დისპლეის შესასვლელებს (იხ.სურ. 3.25,3.26);
2. სადემონსტრაციო პროგრამაში შეცვალეთ ეკრანზე გამოსაყვანი ტექსტი;
3. შეცვალეთ ეკრანზე ტექსტის ძვრის მიმართულება;
4. გააკომპილირეთ პროგრამა და ჩაწერეთ მიკროპროცესორში;
5. გაუშვით პროგრამა და დააკვირდით სტენდზე მის შესრულებას.

ლაბორატორიული სამუშაო №8
მიკროკონტროლერის მუშაობა გრაფიკულ დისპლეისთან

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

ლაბორატორიულ სტენდზე შევქმნათ სისტემა, რომლის საშუალებით მიკროკონტროლერიდან სრულდება სხვადასხვა გრაფიკული გამოსახულების ან ტექსტის გამოყვანა.

ელექტრული სქემა

ამ მოწყობილობისათვის გამოვიყენოთ გრაფიკული დისპლეი WG 12854b კონტროლერით KS0107 და მიკროკონტროლერი Atmega 128.

სურ.3.15-ზე ნაჩვენებია გრაფიკული დისპლეის საერთო ხედი.

სურ.3.15

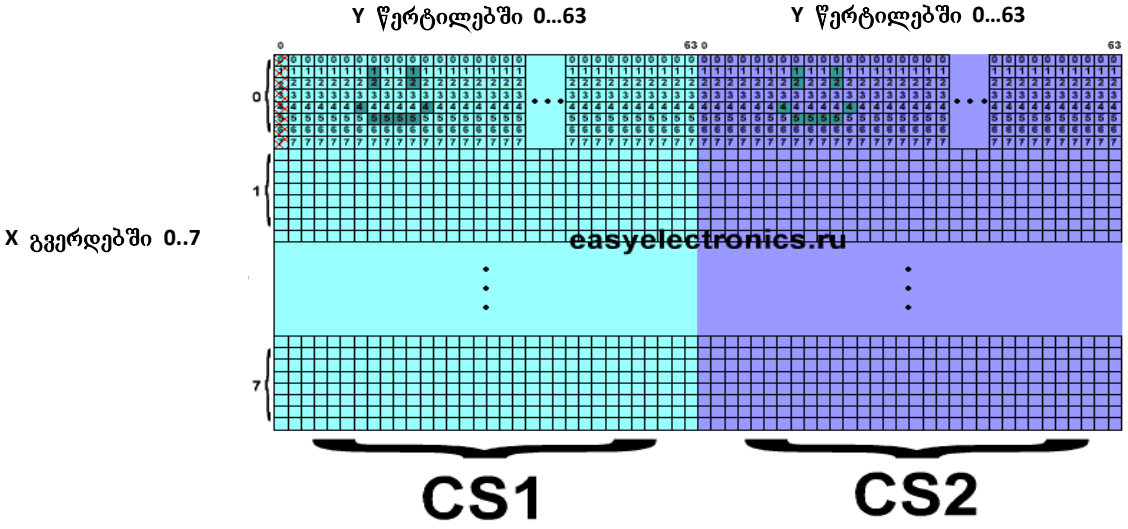
კონტროლერს აქვს გამომყვანები, რომელთა საშუალებით ხორციელდება მონაცემთა, ბრძანებების და სამართავი სიგნალების გადაცემა.

- **Vdd и Vss** -კვების წყაროს შესასვლელი, რომელზედაც მიეწოდება +5 ვ;.

- **Vee** — უარყოფითი 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 დანიშნულებას: 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 სლტე, რომელითაც გადაიცემა საჭირო კოდი.
- **CS1 и CS2** — კონტროლერის ამორჩევა.
- **RST** — განულების სიგნალის შესასვლელი. ამ შესასვლელზე ნულის მიწოდებით კონტროლერები დგებიან ნულში. ვიდეომეხსიერება არ იშლება.
- **A и K** — განათების შუქდიოდების კვება.

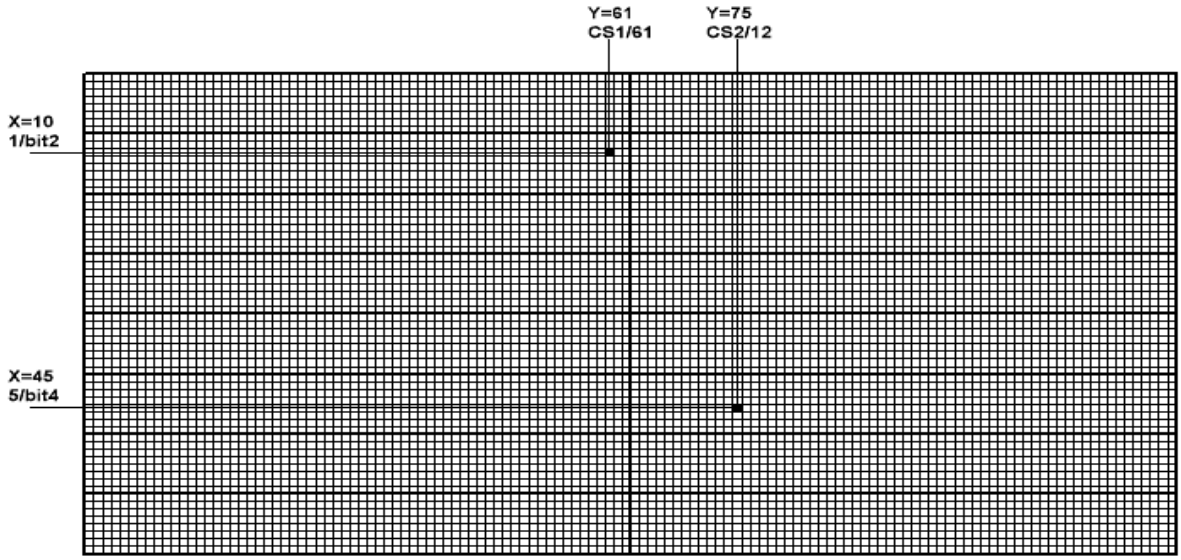
WG12864A დისპლეი არის მონოქრომული თხევადკრისტალური მატრიცა 128x64 გარჩევითობით (ვერტიკალურად 128, ჰორიზონტალურად 64 პიკსელი), შიგა მმართველი კონტროლერებით KS0108 და შიგა 1 კბაიტის ტევადობის ვიდეო მეხსიერებით. ვინაიდან თვითოეულ კონტროლერს შეუძლია მხოლოდ 64x64 ზომის მ\ატრიცის ორგანიზაცია, ამიტომ გამოიყენება ორი კონტროლერი: ერთი პასუხს აგებს ეკრანის მარცხენა ნაწილზე, მეორე - მარჯვენა ნაწილზე. ისინი შეიცავენ მახსიერებას, სადაც იწერება ეკრანზე გამოსაყვანი მონაცემები. მეხსიერების თითო ბიტი ეკრანზე აისახება წერტილით. დისპლეის მეხსიერების კარტას აქვს შემდეგი სახე (სურ.3.16)



სურ.3.16

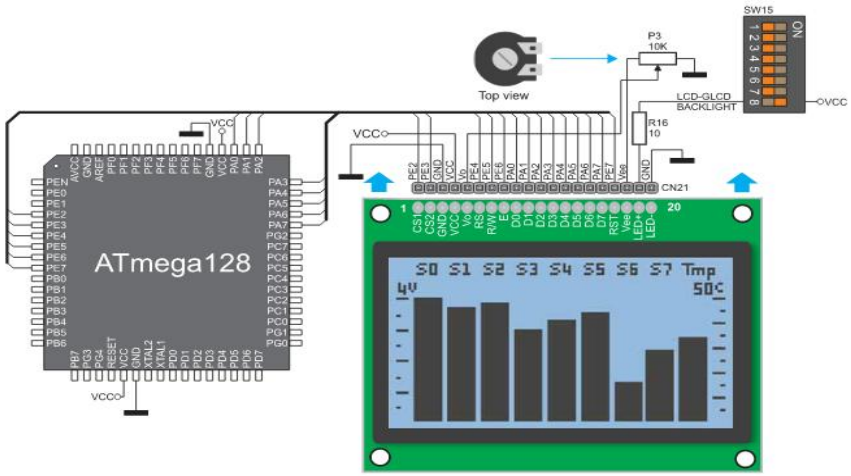
ბაიტები ჩაწყობილია ორ კონტროლერში გვერდებად, თითოში 64 ბაიტი. სულ თითო კონტროლერი შეიცავს 8 გვერდს. იმისათვის, რომ გამოვიყვანოთ ეკრანზე წერტილი კოორდინატებით, მაგალითად, X=10, Y= 61 საჭიროა გამოვითვალოთ რომელ

კონტროლერში იმყოფება იგი. თუ ნაკლებია 63-ზე, მაშინ წერტილი იმყოფება პირველ 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.3.17-ზე ნაჩვენებია პიქსელების მდებარეობა ვიდეო მეხსიერებაში.



სურ.3.17

დისპლეის მიკროკონტროლერთან მიერთების სქემა მოყვანილია სურ. 3.18-ზე:



სურ.3.18

მონაცემთა გამომყვანები მიერთებულია პორტ A- სთან. სამართავი შესასვლელები- პორტ E-სთან: CS1 - PE.2-თან; CS2 - PE.3; RS - PE.4; RW -PE.5; EN - PE.6; RST - PE.7.

ბრძანებების სისტემა

დისპლეიში სხვადასხვა ოპერაციების შესრულებისათვის მას აქვს რიგი ბრძანებებისა (ცხრილი 3.6):

ცხრილი 3.6

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 მდგომარეობაზე და გამოსახულების 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 შემდეგ 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.

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( ) {
      Delay_ms (2000);    // დაყოვნების  ფუნქცია  2  წამი
}
17.  void  main ( ) {
18.  char  counter;        // couter როგორც  მთელის  განსაზღვრა

19. char  *someText;  // უჯრედის  გამოყოფა  ტექსტისთვის

20.Glcd_Init ( );       // დისპლეის  ინიციალიზაცია
21. Glcd_Fill(0x00);  // დისპლეის  გასუფთავება


22. while  (1)  {
22.Glcd_Image (truck_bmp); //გამოსახულების  გამოყვანა  ეკრანზე
23. Delay2S( ); Delay2S( );    //დაყოვნება  4  წამი
24. Glcd_fill (0x00);     // ეკრანის  გასუფთავება
25.Glcd_PartialImage  (0,0,68,30,,128,64, truck_bmp); // გამოსახულების  ნაწილის  გამოყვანა
//ეკრანზე

26.  Delay_ms (500); // დაყოვნება     500მწმ
27. Glcd_PartialImage  (24,16,68,30,,128,64, truck_bmp);// გამოსახულების  ნაწილის  გამოყვანა
//ეკრანზე


28. Delay_ms (500);  // დაყოვნება     500მწმ
29. Glcd_PartialImage  (56,34,68,30,,128,64, truck_bmp);// გამოსახულების  ნაწილის  გამოყვანა
//ეკრანზე
30. Delay_ms (500);  // დაყოვნება     500მწმ
.31. Glcd_Fill(0x00);  // ეკრანის  გასუფთავება

32. Glcd_Box (62, 40, 124,56,1); // მართკუთხედის  გამოყვანა  ეკრანზე
```

```
33. Glcd_Rectangle (5,5,84,35,1);  // მართკუთხედის  გამოყვანა  ეკრანზე
34.  Delay_ms (1000);
35.  Glcd_Rectangle_Round_Edgee  (2,2,87,38,7,1);// მომრგვალებული  კუთხეებიანი  მართკუთხედის
//გამოყვანა  ეკრანზე
36. Delay_ms (1000);
37. Glcd_Rectangle_Round_Edgee  (8,8,81,32,12,1);  // მომრგვალებული  კუთხეებიანი  მართკუთხედის
//გამოყვანა  ეკრანზე

38. Delay_ms (1000);
39. Glcd_Line (0,0,127,63,1); //ხაზის  გამოყვანა
40. Delay2S ( );

41.for (counter=5;counter<60; counter+=5) {  //  ხაზის  გამოყვანის  ციკლის  დასაწყისი
42. Delay_ms (250);
43. . Glcd_V_Line (2,54,counter , 1); //  ვერტიკალური  ხაზების  გამოყვანა
44. . Glcd_H_Line (2,120,counter,1); //  ჰორიზონტალური  ხაზების  გამოყვანა
}
45. Delay2S ( );
46.Fill(0x00);  //  ეკრანის  გასუფთავება

47. Glcd_Set_Font (Character8x7,8,7,32); //  შრიფტის  ფორმატის  გაწყობა
48. Glcd_Write_Text ("mikroE",5,7,2);     //  ტექსტის  გამოყვანა  ეკრანზე

49.for (counter=1;  counter<=10; counter++);//  წრეხაზის  გამოყვანის  ციკლი
50. Glcd_Circle (63,32,3*counter,1);  //  წრეხაზის  გამოყვანა
51. Delay2S ( );

52. . Glcd_Circle_Fill (63,32,30,1);  //წრეხაზის  შევსება
53. Delay2S ( );
54. Glcd_Box (12,20,70,57,2); //მართკუთხედის  გამოყვანა
55. Delay2S ( );

56.Glcd_Fill (0xFF);  //  ეკრანის  გამუქება

57. Glcd_Set_Font (Character8x7,8,7,32); //  შრიფტის  ფორმატი  განსაზღვრა
58.someText= "8x7  font";  //ტექსტის  ჩაწერა  რეგისტრში
59. Glcd_Write_Text ( someText ,5,0,2);//რეგისტრში  ჩაწერილი ტექსტის  გამოყვანა  ეკრანზე
60. Delay2S ( );

61. Glcd_Set_Font (Sistem3x5,3,5,32);  //  შრიფტის  ფორმატი  განსაზღვრა
62. someText= "3x5  CAPITALS  ONLY";  //  ტექსტის  ჩაწერა  რეგისტრში
63. Glcd_Write_Text (someText,60,2,2);  //  შრიფტის  ფორმატი  განსაზღვრა
64. Delay2S ( );
```

```
65. Glcd_Set_Font (Font5x7,5,7,32);// შრიფტის ფორმატი განსაზღვრა
66. someText= “5x7 Font”; // ტექსტის ჩაწერა რეგისტრში
67. Glcd_Write_Text (someText,5,4,2); // რეგისტრში ჩაწერილი ტექსტის გამოყვანა ეკრანზე
68. Delay2S ( );

69. Glcd_Set_Font (FontSistem5x7,v2,5,7,32); // შრიფტის ფორმატი განსაზღვრა
70. someText= “5x7 Font (v2)”; // ტექსტის ჩაწერა რეგისტრში
71. Glcd_Write_Text (someText,50,6,2);// რეგისტრში ჩაწერილი ტექსტის გამოყვანა ეკრანზე

72. 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 სტრიქონებში განსაზღვრულია მკ-ის გამომყვანების კავშირი დისპლეის შესასვლელებთან და გადაცემის მიმართულება.

სტრიქონ 16-ში აღწერილია დაყოვნების ფუნქცია, რომელიც გამოიყენება პროგრამაში.

სტრიქონ 17-დან იწყება მთავარი პროგრამა.

სტრიქონ 18-ში განსაზღვრულია ცვლადი **counter,** რომელიც გამოიყენება პროგრამაში ციკლის გასვლების რაოდენობის დასათვლელად.

სტრიქონ 19-ში 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 ( )** (სტრიქონი 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)** (სტრიქონი 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) (სტრიქონი 22). სრულდება 4 წამიანი დაყოვნება გამოსახულების აღქმისათვის (სტრიქონი 23) და შემდეგ ეკრანი კვლავ სუფთავდება (სტრიქონი 24).

მომდევნო სტრიქონებში 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 სიგანეს 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 **truck_bmp).** თვითოეული განხილული ფუნქციის შესრულების შემდეგ სრულდება 500მწ დაყოვნება გამოსახულების აღქმისათვის.

სტრიქონ 31-ში კვლავ სრულდება ეკრანის გასუფთავება.

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

35,37 სტრიქონებში ფუნქციებს ეკრანზე გამოყავთ მართკუთხედები მომრგვალებული კუთხეებით. მაგალითად, 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), მომდევნო 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 კოორდინატებს, ბოლო პარამეტრი- გამოსახულების ფერს.

41 სტრიქონიდან იწყება პარალელური ვერტიკალური და ჰორიზონტალური ხაზების ეკრანზე გამოყვანის ციკლი **Glcd_V_Line (2,54,counter , 1)** (სტრიქონი 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 კოორდინატა (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, თვითოეულ გასვლაზე ციკლში ეკრანზე გამოყვანილი იქნება პარალელური ჰორიზონტალური და ვერტიკალური ხაზები (conter=5 მნიშვნელობის მიღწევამდე). დაყოვნების შემდეგ ეკრანი კვლავ სუფთავდება (სტრიქონი 46).

შემდეგ პროგრამაში სრულდება ტექსტის გამოყვანა სხვადასხვა შრიფტით. დასაწყისში საჭიროა კონტროლერი აეწყოს გამოსაყვანი შრიფტის ფორმატზე. ამისათვის გამოიყენება ფუნქცია **Glcd_Set_Font,** რომლის პირველი პარამეტრი აწყობს სისტემას შრიფტის ფორმატზე, შემდეგი ორი განსაზღვრავს შრიფტის ზომას, ხოლო

ბოლო - შრიფტების ცხრილის დასაწყისს (microc PRO for AVR კომპილიატორის 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 შრიფტს ფორმატზე 8x7 (შრიფტის სიმაღლე 8, სიგანე 7 წერტილი– პიკსელი).
ტექსტის გამოყვანა ეკრანზე ხორციელდება ფუნქციით **Glcd_Write_Text,** რომლის პირველი პარამეტრი არის გამოსაყვანი ტექსტი, მეორე პარამეტრი X კორდინატა, მესამე გვერდის ნომერი, ბოლო - შრიფტის მაგალითად, **Glcd_Write_Text** (“mikroE”,5,7,2) (სტრიქონი 48).

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

სტრიქონ 52-ში **Glcd_Circle_Fill (63,32,30,1)** ფუნქციის საშუალებით სრულდება გამოყვანილი წრეხაზის შიგნით მუქი ფერის “ჩასხმა”. 2წმ დაყოვნებით ეკრანზე სრულდება მართკუთხედის გამოყვანა (სტრიქონი 54).

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

პროგრამის დასასრულს სრულდება გადასვლა პროგრამის დასაწყისზე.

ლაბორატორიული სამუშაოს დავალება:

1. სტენდზე მიუერთეთ მიკროკონტროლერის გამოსასვლელები გრაფიკული დისპლეის შესასვლელებს გადამრთველების საშუალებით (იხ.სურ.3.18);
2. სადემონსტრაციო პროგრამის მიხედვით შექმენით პროგრამა სხვადასხვა გრაფიკული გამოსახულების და ტექსტის გამოყვანისათვის ეკრანზე[
3. გააკომპილირეთ პროგრამა და ჩაწერეთ მიკროკონტროლერში;
4. გაუშვით პროგრამა და დააკვირდით მოწყობილობის მუშაობას.

ლაბორატორიული სამუშაო №9
სენსორულ პანელთან მუშაობა (Touch Panel)

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

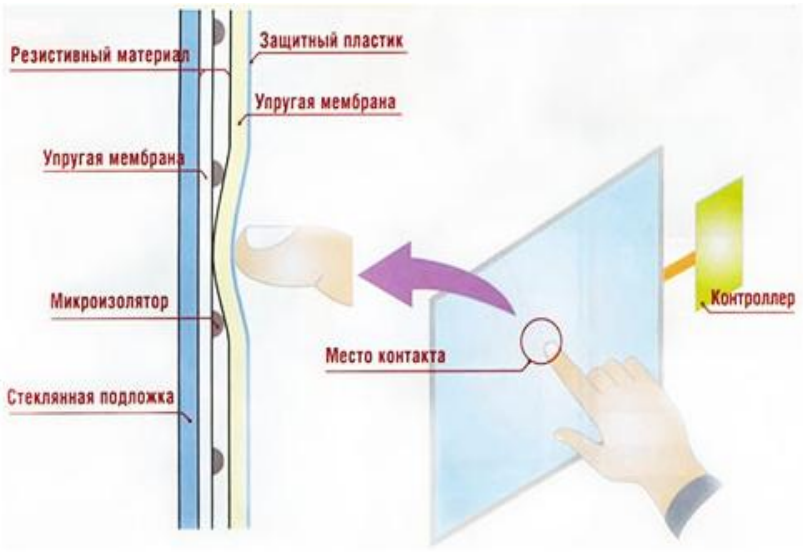
შევისწავლოთ სენსორულ ეკრანთან მიკროკონტროლერის მუშაობა. ეკრანზე ფანქრის დაჭერით GLCD დისპლეიზე გამოვიყვანოთ სხვადასხვა გამოსახულება.

სენსორული პანელის სტრუქტურა და მუშაობის პრინციპი

ამჟამად გამოიყენება სხვადასხვა სახის სენსორული პანელები. ჩვენ განვიხილავთ რეზისტორული ტიპის ოთხგამოსასვლელიან პანელს TS12864CRNA. პანელი, რომელიც

იყენებს რეზიტორულ ტექნოლოგიებს შესდგება ორი ნაწილისაგან – მოქნილი ზედა და ხისტი ქვედა ფენებისაგან. მოქნილი ფენისთვის გამოიყენებიან სხვადასხვა პლასტიკური ან პოლიეფირული აფსკები, ხოლო მეორე მზადდება მინისაგან. ორივე ფენის შიგა ნაწილი დაფარულია დენის გამტარი თხელი ფენით, რომელთაც გააჩნიათ გარკვეული წინააღმდეგობა. მათ შორის სივრცე შევსებულია მიკროსკოპული იზოლიატორით, რომელიც თანაბრად არის განაწილებული მთელ ზედაპირზე, ჰყოფს ორ ფენას და არ აძლევს მათ შეხების საშუალებას. სურ. 3.19-ზე ნაჩვენებია სენსორული პანელის სტრუქტურა.

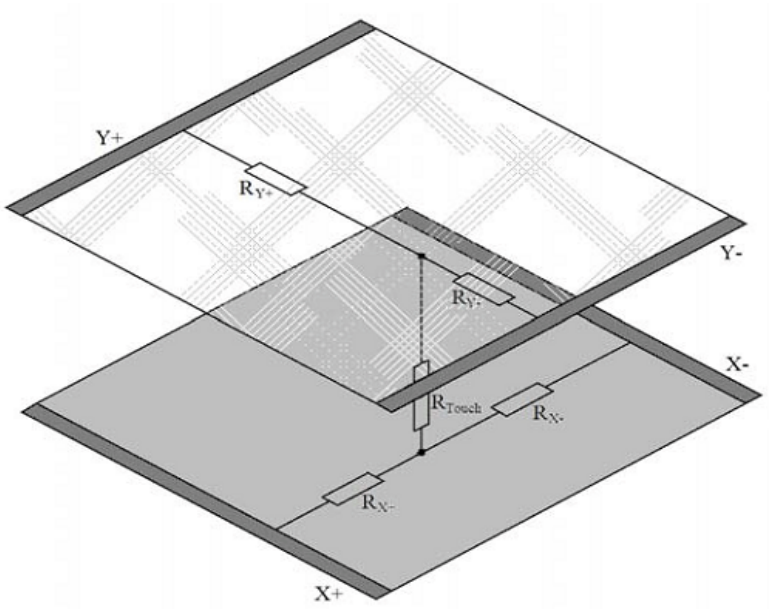
მოქნილ ფენაზე დაჭერის შემთხვევაში, გამტარი ფენები ეხებიან ერთმანეთს და ქმნიან კონტაქტს შეხების წერტილში, ამასთან შეხების წერტილი ქმნის მარტივ ძაბვის გამყოფს. ამის გამო დაწოლის წერტილის კოორდინატები მარტივად შეიძლება გამოითვალოს აცგ-ს საშუალებით 2 გაზომვის შედეგად (ჯერ ერთი კოორდინატის, შემდეგ მეორე კოორდინატის). X კოორდინატას გამოთვლისათვის უკანა აფსკის განაპირა სალტეებზე უნდა მივაწოდოთა მუდმივი ძაბვა, მაგალითად +5ვ (მარცხენა X+ სალტეზე მიეწოდება 5ვ,



სურ.3.19

მარჯვენა სლტეზე 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 კორდინატა. ანალოგიურად წაიკითხება Y კორდინატა. ამ შემთხვევაში კვება მიეწოდება წინა ფენის განაპირა სალტეებს (ზევითა Y+ სალტეს მიეწოდება 5ვ, ქვევითა Y- სალტეს- მიწა). წინა ფენის დაჭერილ წერთილში ძაბვის მნიშვნელობა მიეწოდება აცგ-ს უკანა პანელის ერთ-ერთ

გამოსასვლელიდან (ამ შემთხვევაში უკანა ფენის გამოსასვლელებზე ძაბვა არ მიეწოდება. სურ.3.20-ზე ნაჩვენებია სენსორული პანელის ორგანიზაცია.

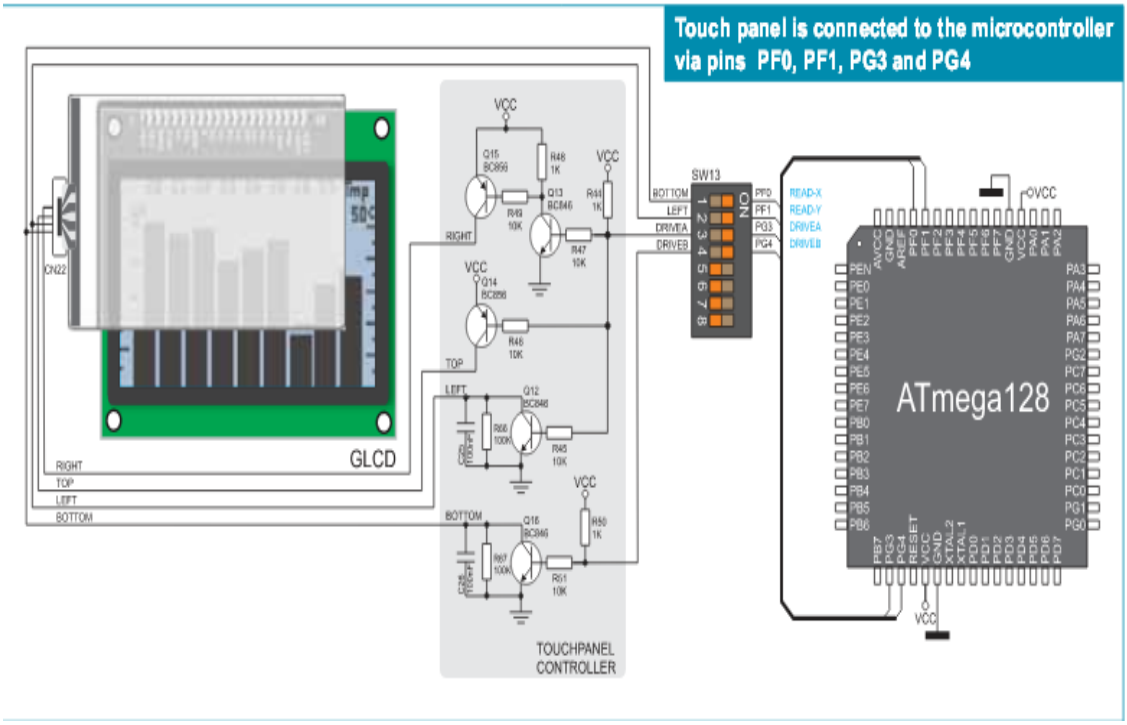


სურ.3.20

სიტემის პრინციპიალური 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 ჩვენს მიერ ზემოთ განხილულ გრაფიკულ დისპლეის WG12864A 128x54. სურ.3.21.-ზე ნაჩვენებია სენსორული პანელის დაკავშირება მიკროკონტროლერის გამომყვანებთან.

მოყვანილი სქემის მიხედვით მიკროკონტროლერის PG3, PG4 გამოსასვლელები მიერთებულია სენსორული პანელის სამართავ შესასვლელებთან DRIVEA, DRIVEB, ამ გამოსასვლელებით მიკროკონტროლერი გასცემს, ორთანრიგა კოდს, რის საფუძველზე პანელის ცალკეულ ფენის სალტეებზე ფორმირდებიან X დაY კოორდინატის წაკითხვის ძაბვის მნიშვნელობები. მიკროკონტროლერის PF0, PF1 გამომყვანები უკავშირდებიან სენსორული პანელის გამოსასვლელებს, საიდანაც ამოიკითხებიან დაჭერის ადგილის X და Y კოორდინატას მნიშვნელობა. მიკროკონტროლერის გრაფიკულ დისპლეისთან დაკავშირება ისევე ხდება, როგორც წინა ამოცანაში იყო ნაჩვენები: მონაცემთა გამომყვანები მიერთებულია პორტ A- სთან. სამართავი შესასვლელები- პორტ E-სთან: CS1 - PE.2-თან; CS2 - PE.3; RS - PE.4; RW -PE.5; EN - PE.6; RST - PE.7.



სურ.3.21

ალგორითმი

1. საწყისი გაწყობის ოპერაციები

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

2. ოპერაციები რომლებიც ჰქმნიან პროგრამის ტანს

- შევასრულოთ სისტემის ინიციალიზაცია;
- გამოვიყვანოთ გრაფიკულ დისპლეიზე რაიმე ტექსტი და გრაფიკული გამოსახულება;
- დაჭერის ადგილის კორდინატების 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-ს0 ანალოგურ
//არხთან
20. const char READ_Y_CHANNEL = 1;       // READ Y ხაზი  დაკავშირებულია  აცგ-ს1 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;  // ზღვრის  მნიშვნელობა  დაჭერის  ალმოჩენისათვის

27. char PressDetect() {   // დაჭერის  ალმოჩენის  ფუნქცია
28. unsigned adc_rd;
29. char result;

 // დაჭერის  ალმოჩენა
 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  ალმოჩენის  გამეორება  2მწ  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-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()) ;
 // მაკალიბრირებელი  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-ს  ინიციალიზაცია
}

71.  void main() {

72.   Initialize();
73.   Glcd_Fill(0x00);                    // GLCD-ს  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 > 88 && y_coord > 60){
101.     Glcd_Fill(0);
102.     Glcd_write_text("otar", 2, 0, 1);
103.     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 (სტრიქონი 1-8) და გადაცემის მიმართულების (სრიქონი 9-14) გამოცხადება. სტრიქონებში 15,16 ცხადდება კავშირი მიკროკონტროლერის **PG3,PG4** გამომყვანებსა და სენსორული პანელის მართვის შესასვლელებს **DDRIVEA, DRIVEB** შორის. შესაბამისად, 17,18 სტრიქონებში განისაზღვრება გადაცემის მიმართულება აღნიშნულ გამომყვანებისათვის. 19,20 სტრიქონებში ცხადდებიან კონსტანტები **READ_X_CHANNEL = 0; READ_Y_CHANNEL =1,** რომლებიც პროგრამაში გამოიყენებიან შესასვლელი არხის მითითებისათვის აცგ-სთან მიმართვის შემთხვევაში – პირველი აკავშირებს წინა ფენის BOTTOM სალტეს მიკროკონტროლერის PF0 გამომყვანთან (X კორდინატის წაკითხვა), მეორე - უკანა ფენის LEFT სალტეს მიკროკონტროლერის PF1 გამომყვანთან (Y კორდინატის წაკითხვა). 21 სტრიქონში ცხადდებიან ცვლადები **x_coord, y_coord,** რომლებშიც პროგრამის შესრულების დროს ჩაიწერებიან სენსორულ პანელზე დაჭერის ადგილის კოორდინატები. 22 სტრიქონში გამოცხადებულია ე.წ. მაკალიბრებელი კონსტანტები რომლითაც ხდება მიღებული კოორდინატების მნიშვნელობების კორექცია. 23-25 სტრიქონებში ცხადდება ტექსტური მასივი, რომლის ელემენტებში იწერება სხვადასხვა შეტყობინების ტექსტი. 26 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 სტრიქონებზე განსაზღვრულია პანელზე დაჭერის აღმოჩენის ფუნქცია **PressDetect().** 28-29 ცხადდებიან ცვლადები, რომლებიც გამოიყენებიან ფუნქციაში. 30-31 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 ეწოდება მღალი 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 კორდინატას მნიშვნელობა. გარდასახვის შედეგის წაკითხვა ხდება კომპილიატორის ფუნქციის **ADC_Read (READ_Y_CHANNEL)** საშუალებით (სტრიქონი 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, 2მწ 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 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 სტრიქონზე ხდება გარდასახულ კოდის აცგ-დან ამოკითხვა და **result** ცვლადში ჩაწერა.

46-52 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** გამორთულია, წინა ფენის **TOP** სალტეს მიეწოდება მაღალი ძაბვა**, BOTTOM** სალტეს – მიწა. ამის შედეგად პანელზე დაჭერის შემთხვევაში იზომება ძაბვა წინა პანელის დაჭერის წერტილში (Y კოორდინატა) და მისი მნიშვნელობა უკანა ფენის **TOP** სალტით მიეწოდება აცგ-ს 1 შემავალ ანალოგურ არხს. 51 სტრიქონში ხდება გარდასახულ კოდის აცგ-დან ამოკითხვა და **result** ცვლადში ჩაწერა.

53-66 სტრიქონებში განსაზღვრულია დაკალიბრების ფუნქცია **Calibrate()** . ამ ფუნქციის დანიშნულებაა სენსორული ეკრანიდან წაკითხული კორდინატი შეუსაბამოს გრაფიკული დისპლეის კორდინატთა არეს. ფუნქციის შესრულების დასაწყისში (სტრიქონი 54) დისპლეის ეკრანზე გამოდის წერტილი, რომელიც გვიჩვენებს კორდინატთა არის მარცხენა ქვედა ზღვარს და აგრეთვე ტექსტი **TOUCH BOTTOM LEFT** (სტრიქონი 55)**.** იმ შემთხვევაში**,** როდესაც სენსორული პანელი მოთავსებულია გრაფიკულ დისპლეიზე, უნდა დავაჭიროთ აღნიშნულ წერტილს, რითაც ვაფიქსირებთ პანელის ქვედა ზღვრულ კოორდინატას. 56 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 (სტრიქონი 57,58).

ანალოგიურად სრულდება მარჯვენა ზედა ზღვარის კორექტირება (სტრიქონი 61-66). დისპლეის ეკრანის გასუფთავების შემდეგ (სტრიქონი 60), გამოგვყავს ეკრანის მარჯვენა ზედა წერტილი (სტრიქონი 61) და ტექსტი **TOUCH UPPER RIGHT**. კვლავ სრულდება დაჭერის მოლოდინი (სტრიქონი 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 სტრიქონებში განსაზღვრულია ინიციალიზაციის ფუნქცია **Initialize().** ამ ფუნქციის შესრულების დროს ცხადდება **DRIVE_A, DRIVE_B** გამომყვანების გადაცემის მიმართულება (სტრიქონი 68,69) და გრაფიკული დისპლეის ინიციალიზაცია (სტრიქონი 70).

71 სტრიქონიდან იწყება მთავარი main() პროგრამა. 72 სტრიქონში სრულდება სისტემის ინიციალიზაცია ზევით განხილული **Initialize()** ფუნქციის გამოყენებით. 78 სტრიქონში ხდება მაკორექტირებელი კონსტანტების განსაზღვრა **Calibrate()** ფუნქციის გამოყენებით. 79-93 სტრიქონზე სრულდება სხვადასხვა ინფორმაციის გამოყვანა ეკრანზე ჩვენს მიერ განხილული გრაფიკული დისპლეის ფუნქციების გამოყენებით, რომელიც ატარებს სადემონსტრაციო ხასიათს. 94 სტრიქონიდან სრულდება დაჭერის აღმოჩენის უსასრულო ციკლი. პანელზე დაჭერის შემთხვევაში (სტრიქონი 95) სრულდება მიღებული კოორდინატების კორექცია მაკორექტირებელი კონსტანტების საშუალებით (სტრიქონები 96-99). 100 სტრიქონში მოწმდება პანელის არე. რომელშიც იმყოფება დაჭერის კორდინატები. 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 რამოდენიმე არე უფრო რთული პირობით, რომელზე დაჭერის შემთხვევაში ეკრანზე გამოვა ჩვენს 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-ს0 ანალოგურ  არხთან
20. const char READ_Y_CHANNEL = 1;      // READ Y ხაზი  დაკავშირებულია  აცგ-ს1 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;  // ზღვრის  მნიშვნელობა  დაჭერის  ადმოჩენისათვის

27.  char PressDetect() {    // დაჭერის  ადმოჩენის  ფუნქცია
28.  unsigned adc_rd;
29.  char result;

 // დაჭერის  ადმოჩენა
 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  ადმოჩენის  გამეორება  2მწ  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-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()) ;
 // მაკალიბრირებელი   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-ს ინიციალიზაცია
}

71.  void main() {

72.   Initialize();
73.   Glcd_Fill(0x00);                    // GLCD-ს 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 სტრიქონებში სრულდება იგივე მოქმედებები, რაც პირველ ვარიანტში იყო განხილული. 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 (სტრიქონი 101), რომელებიც შეადგენენ გამოსახულებას (სწორ ხაზს ან მრუდს).

ლაბორატორიული სამუშაოს დავალება:

1. მიუერთეთ სენსორული პანელი და გრაფიკული დისპლეი მიკროკონტროლერის შესასვლელებს;
2. სადემონსტრაციო პროგრამებში შეცვალეთ ტექსტის გამოყვანის არეები. პირველ ვარიანტში –თვით ტექსტი;
3. გააკომპილირეთ პროგრამები და ჩაწერეთ მიკროკონტროლერში;
4. გაუშვით პროგრამები და დააკვირდით მის შესრულებას.დააკვირდით

ლაბორატორიული სამუშაო №10

ბგერითი სიგნალების მაფორმირებელი მოწყობილობა

ლაბორატორიული სამუშაოს მიზანი:

შევისწავლოთ მოწყობილობა, რომელიც სხვადასხვა ღილაკზე დაჭერით გამოსცემს სხვადასხვა სიხშირის ბგერებს.

მოწყობილობის სტრუქტურული სქემა

გავიხილოთ ელექტრონული მოწყობილობა, რომელსაც აქვს შვიდი შესასვლელი და ერთი ბგერის გამოსასვლელი. თვითოეულ 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. ბგერის ფორმირებისათვის მოხერხებულია მიკროკონტროლერის ტაიმერ/მთვლელის გამოყენება.

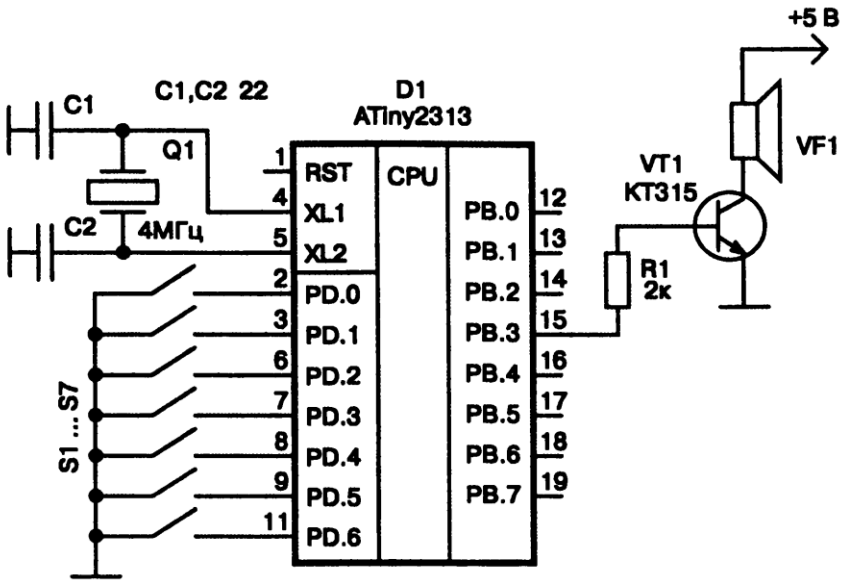
მიკროკონტროლერის შემადგენლობაში შედის ოთხი ტაიმერ/მთვლელი: რვა თანრიგა T0,T2 და თექვსმეტთანრიგა T1,T3. ბგერის ფორმირებისათვის უმჯობესია თექვსმეტთანრიგა ტაიმერის გამოყენება. რაც მეტია თანრიგების რაოდენობა, მით უფრო ზუსტად შეგვიძლია შევარჩიოთ სიხშირის დაყოფის კოეფიციენტი. ეს მეტად მნიშვნელოვანია სანოტო 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 ინკრიმენტი 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, რომლებზედაც აისახება ტაიმერის გადასვლა მაქსიმალური მდგომარეობიდან 00h მდგომარეობაში გამომყვანების მდგომარეობის ცვლილებით. მოცემულ შემთხვევაში გამოიყენება OC1A გამოსასვლელი. შედეგად გამოსასვლელზე ფორმირდება განივიმპულსური მოდულიაციის საგნალები, რომლებიც მიეწოდებიან ბგერის ელემენტს PB პორტის მესამე კონტაქტიდან, რომელთანაც ხსენებული გამომყვანი არის შეთავსებული. გენერირებული სიგნალების სიხშირე განისაზღვრება გამოსახულებით $f_{OCR}= f_{CLKIO}/2N(1+OCRn)$, სადაც N- წინაგამყოფის გაყოფის კოეფიციენტია, OCRn – შესადარებელი მნიშვნელობაა.

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

ჩვენ უნდა ავირჩიოთ გაყოფის ისეთი კოეფიციენტი, რომელიც OCR1A გამოსასვლელზე მოგვცემს სიხშირეს ბგერით დიაპაზონში. თუ მიკროკონტროლერის სიხშირეს ავირჩევთ 4მჰც-ს, N=1, OCR- ათასეულს, მაშინ OC1A გამოსასვლელზე გენერირებული სიხშირე იქნება სასურველ დიაპაზონში.

ვინაიდან ბგერის გამოსაყვანად გამოვიყენებთ PB პორტის ერთერთ თანრიგს, გადამწოდების მიერთებისათვის გამოვიყენოთ სხვა პორტი – PD. ზემოთ აღწერილი მოწყობილობის პრინციპიალური სქემის ვარიანტი ნაჩვენებია სურ.3.22-ზე.

როგორც მოყვანილი სურათიდან ჩანს, მიკროკონტროლერის ტაქტირებისათვის გამოყენებულია კვარცული რეზონატორი (Q1). გადამწოდების მიერთება შესრულებულია იგივე სქემით, რაც გამოიყენებოდ ზევით განხილულ მოწყობილობებში გადამრთველი კონტაქტების მიერთებისათვის. გადამწოდები მიერთებული არიან PD პორტის გამომყვანებთან. ამასთან გადამწოდების გამართული მუშაობისათვის PD პორტის თვითოეულ გამოსასვლელს უნდა მიუერთოთ ჩაშენებული დატვირთვის რეზისტორები. დინამიკის მიერთებისათვის გამოყენებულია გასაღები VT1 ტრანზისტორზე. ეს არის ბგერის ფორმირების ყველაზე მარტივი საშუალება.



სურ.3..22

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

ალგორითმი

პირველი 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 აშვების შემთხვევაში მიკროკონტროლერმა გამორთოს ტაიმერი OCR1A გამოსასვლელიდან და გასცეს დაბალი ლოგიკური დონე. თუ ყველა გადამწოდის კონტაქტები აშვებულია, მაშინ გარე გამომყვანი უნდა დარჩეს გათიშულ მდგომარეობაში. სქემა შედგენილია იმგვარად, რომ შესაძლებელია ერთდროულად რამოდენიმე კონტაქტის დაჭერა, რაც ეწინააღმდეგება მოწყობილობის მუშაობის პრინციპს. ამ შემთხვევაში გამოიყენება პრიორიტეტების სისტემა. რამოდენიმე კონტაქტის ჩართვის შემთხვევაში პროგრამა უნდა რეაგირებდეს მხოლოდ ერთზე, რომლის პრიორიტეტი უფრო მაღალია. გამოყენებულია შემდეგი ხერხი. პროგრამა თანმიმდევრობით ამოწმებს ყველა გადამწოდების მდგომარეობას. პირველივე ჩართული კონტაქტის აღმოჩენის შემდეგ პროგრამა წყვეტს სკანირებას და გასცემს ამ გადამწოდის შესაბამის ბგერის სიხშირეს.

შევთანხმდეთ, PD.0 შესასვლეზე მიერთებულ გადამწოდს შეუსაბამოთ ნოტა “დო”, მომდევნო გადამწოდს - ნოტა “რე”, და ა.შ ნოტა “სი”-მდე. სიხშირის დაყოფის კოეფიციენტები თვითოეული 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.usigned char count; // count ცვლადის გამოცხადება
5. usigned 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 შესდგება მხოლოდ ერთი main ფუნქციისაგან. ამ ფუნქციის დასაწყისში განსაზრვრულია tabkd მასივი (სტრიქონი 2), რომელშიც ჩაწერილია სიხშირის დაყოფის კოეფიციენტები სხვადასხვა ხმის ტემბრისათვის. ინიციალიზაციის შედეგად მასივის ელემენტებს ენიჭებათ ცნობილი მნიშვნელობები.

4-5 სტრიქონებში შექმნილია ორი ცვლადი:

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

6-12 სტრიქონები უკავიათ ინიციალიზაციის ოპერატორებს. 6-9 სტრიქონებში სრულდება PORTB და PORTD პორტების გაწყობა. პორტების გაწყობა მდგომარეობს, უპირველეს ყოვლისა, პორტებში გამომყვანების გადაცემის მიმართულების განსაზღვრაში. ამისათვის თვითეული პორტის DDR რეგისტრების შესაბამის თანრიგებში ჩაიწერება 0 ან 1. 0 გააწყობს პორტის გამოსასვლელს მონაცემთა შეტანაზე (ჩვენ შემთხვევაში პორტი PD- სტრიქონი 9), 1 - მონაცემთა გამოტანაზე (ჩვენ შემთხვევაში PB პორტის მესამე გამოსასვლელი – სტრიქონი 7). მეორეს მხრივ საჭიროა პორტის (გამოსასვლელებზე რომლებიც გაწყობილი არიან სიგნალების შეტანაზე) ჩაირთოს დამტვირთველი რეზისტორები PORT რეგისტრების შესაბამის თანრიგებში 1-ის ჩაწერით (სტრიქონი 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 აწყობა. 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 პუნქტები. ეს არის უსასრულო ციკლის ოპერატორი while.

14 ტრიქონში 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 - კონტაქტის დაჭერილ მდგომარეობას.

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

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

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

17 სტრიქონზე სრულდება temp ცვლადის ძვრა მარჯვნივ და სკანირების ციკლი მეორდება. ციკლის შვიდი გავლის შედეგად მოწმდება ყველა შვიდი ღილაკის მდგომარეობა. თუ ყველა ღილაკის კონტაქტები აღმოჩნდებიან გათიშული, მაშინ ციკლი for (სტრიქონები 15-17) მთავრდება ბუნებრივი წესით და პროგრამა გადადის 18 სტრიქონზე. აქ სრულდება ხმის გამორთვა TCCR1A რეგისტრის ნულზე დაყენებით.

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

თუ ციკლი მთავრდება ვადაზე ადრე 20 სტრიქონზე გადასლით (ნიშნაკი m2), იწყება ხმის ფორმირების პროცესი. ვინაიდან ღილაკების 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 სტრიქონზე სრულდება ხმის ჩართვა. ხმა ჩაირთვება TCCR1A რეგისტრში 0x40 კოდის ჩაწერით. ამით მთავრდება პროგრამის ძირიტადი ციკლი. ვინაიდან ძირითადი ციკლი წარმოადგენს უსასრულო ციკლს, მისი ბოლო ბრძანების დამთავრების შემდეგ ციკლი მეორდება.

ლაბორატორიული სამუშაოს დავალება:

1. მიუერთეთ სქემა ლაბორატორიულ სტენდს გარე გამომყვანებით;
2. გააკომპილირეთ პროგრამა და ჩაწერეთ მიკროკონტროლერში;
3. გაუშვით პროგრამა და გამოიკვლიეთ მისი მუშაობა სტენდზე.

ლაბორატორიული სამუშაო №11
მიკროკონტროლერის მუშაობა MMC/CD დისკებთან

ლაბორატორიული სამუშაოს მიზანი:
მიკროკონტროლერის საშუალებით ფლეშ დისკთან მუშაობის გაცნობა, რომელიც მდგომარეობს დისკთან შემდეგი ოპერაციების შესრულებაში:

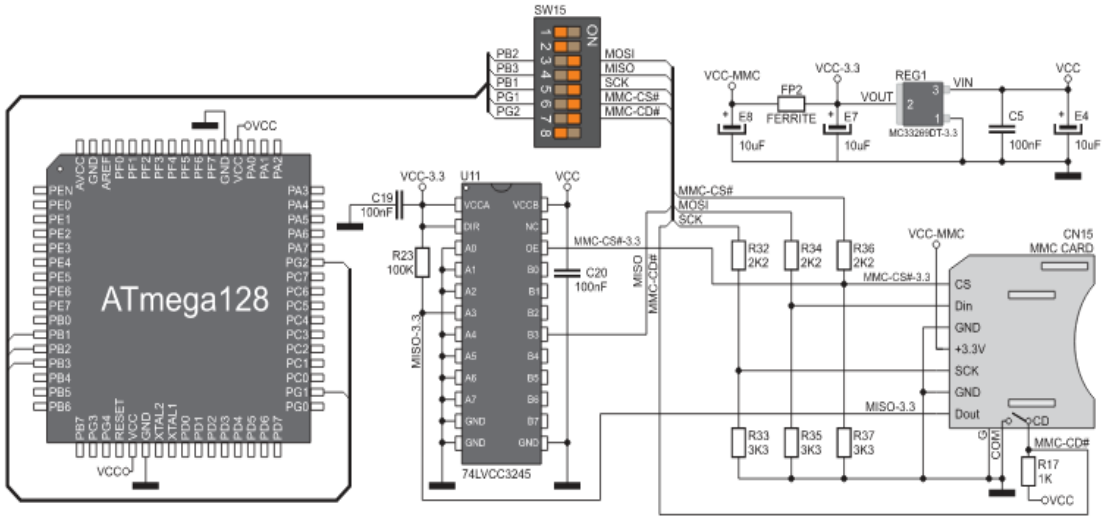
- *ახალი ფაილის შექმნა და მასში ჩაწერა;*
- *არსებული ფაილის გახსნა და მასში ახალი მონაცემების ჩაწერა;*
- *არსებული ფაილის გახსნა და მასში მონაცემთა დამატება;*
- *ფაილის გახსნა და მონაცემების წაკითხვა (მათი გაგზავნა USART პორტის საშუალებით);*
- *ფაილის წაშლა.*

ლაბორატორიულ სამუშაოში გამოყენებულია SD დისკი (სურ.3.23 1,8 მბაიტის ტევადობით, იგი თავსდება სტენდზე არსებულ გასართში და იყენებს FAT16 ფაილურ სისტემას.

სურ.3..23

დისკგამწრევის მიერთება მიკროკონტროლერთან

სურ.2.24-ზე მოცემულია დისკგამწევის მიერთების სქემა მიკროკონტროლერთან.



სურ.2.24

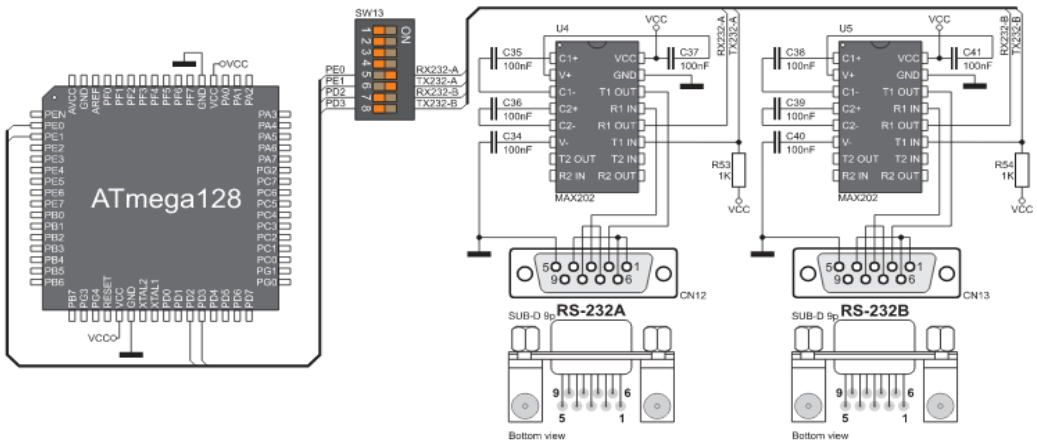
მიკროკონტროლერი მიერთებულია დისკგამწევთან SPI ინტერფეისით და მონაცემთა გაცვლა ხორციელდება ამ ინტერფეისის პროტოკოლის თანახმად. სქემაში მიკროკონტროლერი არის წამყვანი (Master), დისკგამწევი მიმყოლი (Slave). SPI ინტერფეისის MOSI გამოსასვლელი წარმოადგენს მიკროპროცესორის PB.2 გამომყვანს, რომლითაც იგზავნება 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 დაყენებული იყოს ON მდგომარეობაში.

ფაილის გამოყვანა სტენდიდან ხდება USART ინტერფეისით RS232A და RS232B გასართების საშუალებით (სურ.3.25).

სურ.3..25

სურ. 2.26-ზე მოცემულია მიკროკონტროლერთან RS232 გასართების მიერთების სქემა. მიკროკონტროლერის PE.0 პორტი წარმოადგენს USART1 ინტერფეისის RX232A შესასვლელს, PE.1 პორტი TX232A გამოსასვლელს, PD.2 პორტი - USART2 ინტერფეისის RX232B შესასვლელს, PD.3 პორტი - TX232B გამოსასვლელს. გასართების მიერთებისათვის მიკროპროცესორთან SW13 ბლოკიდან 5,6 ჯამპერი (UART1) ან 7,8 ჯამპერი (USART2) უნდა იყოს ON მდგომარეობაში დაყენებული.



სურ.2.26

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 ახლ 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

void M_Create_New_File() {
filename[7] = 'A';                    // ფაილის სახელის დაყენება
Mmc_Fat_Set_File_Date(2005,6,21,10,35,0);    // ფაილის თარიღის და დროს დაყენება
Mmc_Fat_Assign(&filename, 0xA0);      // არსებული ფაილის მოძებნა ან ახლის შექმნა
Mmc_Fat_Rewrite();                   // ფაილის გასუფთავება და ახალი მონაცემების დაწყება
for(loop = 1; loop <= 99; loop++) {
UART1_Write('.');
file_contents[0] = loop / 10 + 48;
file_contents[1] = loop % 10 + 48;
Mmc_Fat_Write(file_contents, LINE_LEN-1);   // მონაცემთა ჩაწერა გახსნილ ფაილში

  }

 }

// რამოდენიმე ახალი ფაილის შექმნა  და მათში მონაცემების ჩაწერა

void M_Create_Multiple_Files() {
for(loop2 = 'B'; loop2 <= 'Z'; loop2++) {
UART1_Write(loop2);
filename[7] = loop2;                  // ფაილის სახელის დაყენება
Mmc_Fat_Set_File_Date(2005,6,21,10,35,0);  // ფაილის თარიღის და დროს დაყენება
Mmc_Fat_Assign(&filename, 0xA0);      // არსებული ფაილის მოძებნა ან ახალი
  //ფაილის შექმნა
Mmc_Fat_Rewrite();                   // ფაილის გასუფთავება და მონაცემთა ჩაწერის დასაწყისი
for(loop = 1; loop <= 44; loop++) {
file_contents[0] = loop / 10 + 48;
file_contents[1] = loop % 10 + 48;
Mmc_Fat_Write(file_contents, LINE_LEN-1); // შექმნილ ფაილში მონაცემთა ჩაწერა

   }

  }

 }

// არსებული ფაილის გახსნა და მისი გასუფთავება

void M_Open_File_Rewrite() {
filename[7] = 'C';                    // ფაილის სახელის დაყენება
 Mmc_Fat_Assign(&filename, 0);
 Mmc_Fat_Rewrite();
for(loop = 1; loop <= 55; loop++) {
file_contents[0] = loop / 10 + 65;
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 შესაძლებელია შეექმნად 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 და იგ ზავნება მონაცემები და ფაილის ზომა
 // UARTI-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';                    //uncomment this line to search for file that DOES exists
//  filename[7] = 'F';                    //uncomment this line to search for file that DOES NOT exist
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 მოცულობა არის 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- ს 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);

  }

}
```

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

10-სტრიქონიდან აღწერილია ფუნქცია, რომელსაც გამოჰყავს ტექსტი USART ინტერფეისის საშუალებით. ფუნქციის პარამეტრს წარმოადგენს გამოსაყვანი ტექსტი. ფუნქცის ტანი შეიცავს კომპილიატორის ფუნქციებს **UART1_Write_Text(),** რომლის პარამეტრს წარმოადგენს გაქმოსაყვანი ტექსტი. სტრიქონ 11-ში სრულდება ტექსტის გამოყვანა, სტრიქონ 12 და 13-ში გაიცემა მიმდინარე სტრიქონის დამთავრების და ახალ სტრიქონზე გადასვლის კოდები.

სტრიქონ 14- დან 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 კომპილიატორის მზა 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 კომპილიატორის 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 სტრიქონებში ციკლში სრულდება მონაცემთა ჩაწერა ფაილში 99 ჯერ. ციკლის თვითოეულ 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 (სყრიქონი 23) , რომლის პარამეტრებია ჩასაწერი მონაცემები და ჩასაწერი მონაცემთა რაოდენობა.

24-34 სტრიქონებში განსაზღვრულია ფუნქცია, რომელიც ასრულებს რამოდენიმე 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 (სტრიქონი 27). თვითეული ფაილის შექმნა და მასში მონაცემების ჩაწერა სრულდება ზევით განხილულ ფუნქციის ანალოგიურად.

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

43-48 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 მონაცემები (სტრიქონი 47). სტრიქონ 48-ში სრულდება მონაცემთა ჩაწერის განხილული ბიბლოთეკის ფუნქცია.

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

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

61-97 სტრიქონებში აღწერილია ფუნქცია, რომელიც არსებულ ფაილიდან კითხულობს მონაცემებს და ფაილის ზომას UART ინტერფეისის საშუალებით.
67 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, იწერებიან year, month, day, hour, minute ცვლადებში (სტრიქონი 68). შემდეგ სტრიქონებში

თანმიმდევრობით თვითეული მონაცემი იწერება outstr ბუფერში (ფუნქცია **WordToStr)** და ამ ბუფერიდან მონაცემთა გაცემა USART ინტერფეისით (ფუნქცია **UART1_Write_Text(outstr**)).

80-91 სტრიქონებში სრულდება გახსნილი ფაი;ლის მონაცემთა მოდიფიცირება (აღდგენა) და USART ინტერფეისით გამოყვანა. ამისათვის გამოიყენება ბიბლიოთეკის ფუნქცია **Mmc_Fat_Get_File_Date_Modified(&year, &month, &day, &hour, &minute)** და შემდეგ იგივე ფუნქციების თანმიმდევრობა, რაც ზევით იყო აღწერილი.

92-94 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 სტრიქონები, რომლის შედეგად USART ინტერფეისით გაიცემა კოდი 0x55.

98-108 პუნქტებში აღწერილია ფუნქცია, რომელიც ჰქმნის გაცვლის (swap) ფაილს. უნქციის პირველ სტრიქონებში ციკლში იწერება Buffer მასივის ელემენტებში მონაცემები 0 დან 512 მდე. სტრიქონ 102-ში 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 სტრიქონიდან სრულდება ციკლი. რომლის თვითოეულ გასვლაზე სეგმენტში იწერება 512 ბაიტი Buffer მასივიდან. ციკლი სრულდება 500 ჯერ წინასწარ დაყენებული სექტორების რაოდენობის შესაბამისად, სექყორში მონაცემთა ჩასაწერად გამოყენებულია ბიბლიოთეკის ფუნქცია **Mmc_Write_Sector(size++, Buffer),** რომლის პირველი პარამეტრია სექტორის ნომერი რომელშიც იწერება ცვლადები, მეორე პარამეტრი - წყაროს დასახელება , რომლიდანაც იწერება ცვლადები.

109 სტრიქონიდან იწყება main ფუნქცია. უპირველეს ყოვლისა სრულდება USART1 პორტის ინიციალიზაცია, რისთვისაც გამოიყენება ბიბლიოთეკის ფუნქცია **UART1_Init(19200) (**სტრიქონი 110**).** ინიციალიზაცია მდგომარეობს გადაცემის სიჩქარის დაყენებაში, რომელიც მითითებულია ფრჩხილებში პარამეტრის სახით. გარდა ამისა ფუნქციის შესრულების დროს ავტომატურად ხდება 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 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 სტრიქონიდან იწყება სადემონსატრაციო პროგრამა. IF ოპერატორში მოწმდება დისკის არსებობა დისკგამწევში (ფუნქცია Mmc_Fat_Init აბრუნებს 0 დისკის აღმოჩენის შემთხვევაში და 1-ს თუ დისკი არ აღმოჩნდა). თუ დისკი იმყოფება დისკგამწევში, მიყოლებით სრულდება ადრე აღწერილი ფუნქციები. წინააღმდეგ შემთხვევაში USART ინტერფეისიდან გაიცემა შეტყობინება შეცდომის შესახებ (სტრიქონი 126).

ლაბორატორიული სამუშაოს დავალება:

1. სტენდზე გადართეთ ჯამპერები ზევით ნაჩვენებ პოზიციებში;
2. სადემონსტრაციო პროგრამის გამოყენებით შექმენით პროგრამები დისკთან შემდეგი მოქმედებების შესრულებისათვის და შეამოწმეთ მათი შესრულება სტენდზე:
   - შექმენით ახალი ფაილი და ჩაწერეთ მონაცემები;
   - გახსენით არსებული ფაილი და მოამზადეთ იგი ახალი მონაცემების ჩასაწერად;
   - გახსენით არსებული ფაილი და დაუმატეთ მასში მონაცემები;
   - გახსენით არსებული ფაილი და ამოიკითხეთ მისი შემცველობა;
   - წაშალეთ არსებული ფაილი;
   - შექმენით სარეზერვო (swap) ფაილი/

ლაბორატორიული სამუშაო №12
კოდური კლიტე

ჩამოვაყალიბოთ ამოცანა შემდეგი სახით:

აიგოს ელექტრონული კოდური კლიტე, რომელზედაც მიერთებულია ათი ღილაკი კოდის შესაყვანად, 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: კოდის ჩაწერის რეჟიმში მომხმარებელი აჭერს ღილაკებს ნებისმიერი თანმიმდევრობით და ნებისმიერი კომბინაციით (დასაშვებია ერთდროულადაც 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 მანიპულიაციის დამთავრება.

ჩაითვლება, რომ მანიპულიაცია 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 თანმიმდევრობას, 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 მანიპულიაციები ღილაკებზე, რაც შესრულებული იყო ჩაწერის რეჟიმში. მიკროკონტროლერი, ისევე, როგორც განხილულ შემთხვევაში თვალყურს ადევნებს აღნიშნულ მანიპულიაციებს 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 ედარებიან კოდების სიგრძეები. შემდეგ კოდები ედარებიან 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. წინააღმდეგ შემთხვევაში კლიტის მუშაობაში წარმოიშვება 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. მჯერად 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 რამოდენიმე 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 48მწ. ასეთი ხანგრძლიობა განსაკუთრებით მიზანშეწონილია იმ შემთხვევაში, თუ კოდური კომბინაციის აკრეფის შემთხვევაში გამოიყენება რამოდენიმე 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 რამოდენიმე 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 რამოდენიმე ღილაკის ერთდროულად დაჭერის შემთხვევაში, პაუზის შემდეგ ყველა პროცესი მთავრდება. განმეორებითი წაკითხვა იძლევა უკვე დამყარებულ 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 ზევით იყო ნათქვამი, საწყისი კოდის შეყვანის დამთავრების დასაფიქსირებლად გამოიყენება დროის დამცავი შუალედი. ამ შუალედის დასაფორმირებლად ვირჩევთ ტაიმერს. ტაიმერი მუშაობს Normal 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

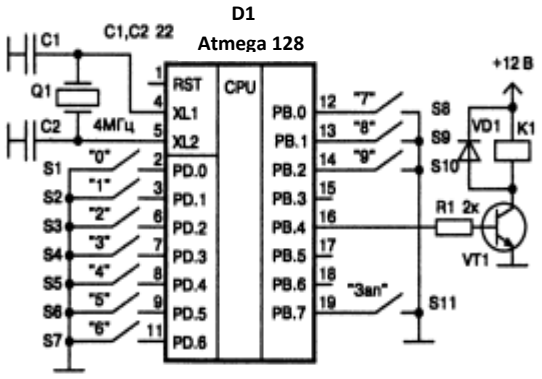
შედარებას. ამისათვის იგი მიმდევრობით კითხულობს EEPROM- დან ბაიტებს და ადარებს ოდმ-ში ჩაწერილ შესაბამის ბაიტებს. პირველივე არათანხვედრის შემთხვევაში შედარების პროცესი სრულდება უარყოფითი შედეგით.

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 კარების გაღებისათვის. შემდეგ პროგრამა გადადის საწყის მდგომარეობაში.

სისტემის სტრუქტურა

კლიტის შესაძლებელი ვარიანტი მოცემულია სურ.3.27-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 დასაცავად, რომელიც წარმოიქმნება რელეს გრაგნილებზე. რელეს კვება ხორციელდება 12 ვოლტიანი კვების წყაროდან.



სურ.3.27

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 ანტივიბრაციისათვის
4. #define bsize 30;        // კოდის ბუფერის ზომა
```

```
5.  unsighned char flz;     // დაყოვნების ალამი
6.  unsighned int bufr[bsize] ;      //ბუფერი ოდმ-ში კოდის შენახვისათვის
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 ფორმირების პროცედურა

24.  void wit (unsighned  char kodz)
    {
25. if  (codz== 1)  TIMSK= 0x40;  //ტაიმერიდან წყვეტის ნებების ამორჩევა
26. else  TIMSK= 0x80;
27. TCNT1=0;                        // ტაიმერის განულება
28. Flz=0;                          // დაყოვნების ალმის 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. TCNT1=0;              // მთვლელი რეგისტრის განულება
42. OCR1A=kzad ;          // თანხვედრის რეგისტრის ინიციალიზაცია
43. ACSR=0x80;            // ანალოგური კომპარატორის გამორთვა
44. while (1)
     {
45. m1: while(icod() !=klfree);  //დილაკის აშვების მოლოდინი
46.       while (icode() ==klfree); //დილაკის დაჭერის მოლოდინი
47.       ii=0;
48. m2: #asm("cli");          // წყვეტის აკრძალვა
49.       wait (1);             //პირველი ტიპის დაყოვნება
50.       codS=incod();         // კოდის შეყვანა როგორც ძველის
51.       bufr (ii++)=codS;     // მორიგი კოდის ჩაწერა ბუფერში
52.       If (ii>=bsize) goto m4 //ბუფერის დასასრულის შემოწმება
53.       wait(2);              // მეორე ტიპის დაყოვნება
54. m3:   if (incod!= codS) goto m2; // მდგომარეობის ცვლილების შემოწმება
55.       if(flz==0)  goto m3;      // დაყოვნების ალმის შემოწმება
56. m4:   if (PINB 7==1) goto comp;  // რეჟიმების გადამრთველის შემოწმება

      // კოდის ჩაწერა EEPROM-ში

57.       klen=ii;                  // კოდის სიგრძის ჩაწერა
58.       For (i=0;i<ii; i++ ) bufe (i)=bufr(i); // კოდის ყველა ბაიტების ჩაწერა
59.       goto zamok;

// კოდის შემოწმება

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

//კლიტის გახსნა
```

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

პროგრამა იწყება კონსტანტების აღწერის ბლოკით (სტრიქონი 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 ჩაიწერებიან EEPROM-ში. ცვლადი **klen** განკუთვნილია კოდური კომბინაციის სიგრძის შესანახად. მასივი **bufe -** თვით ამ კომბინაციის შესანახად. იმისათვის, რომ ტრანსლიატორს მიეთითოს ცვლადისათვის გამოყოფილი უჯრედების ადგილმდებარეობა, ამ ცვლადის აღწერაში გამოყენებული უნდა იყოს მმართველი სიტყვა eeprom.

შემდეგ პროგრამაში იწყება ყველა მის შემადგენელი ფუნქციების აღწერა. მოცემული პროგრამა შესდგება ხუთი ფუნქციისაგან:

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

პროგრამის განხილვა მოხერხებულია დავიწყოთ მთავარი main ფუნქციით. ფუნქცია main იწყება ლოკალური ცვლადების აღწერით (სტრიქონი 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 (სტრიქონი 32-43). ეს მოქმედება მდგომარეობს შეყვანა-გამოყვანის პორტების, ტაიმერის 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 (სყრიქონი 36,38). მთვლელის გაწყობის დროს გარდა ტაიმერის მუშაობის რეჟიმის არჩევისა (სტრიქონები 39,40), განულდება მთვლელი რეგისტრის TCNT1 მნიშვნელობა (სტრიქონი41) და შედარების რეგისტრში OCR1A იწერება დაყოვნების კოდი kzad (სტრიქონი 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 შესდგება მხოლოდ While. ოპერატორისა და ფრჩხილებში მოთავსებული გამოსახულებისაგან, რომელიც განსაზღვრავს ციკლის გაგრძელების პირობას. ციკლი სრულდება იქამდის, სანამ კლავიატურის მდგომარეობის კოდი და klfree კონსტანტა არ არიან 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 (სტრიქონი 45) დასრულდება, და პროგრამა გადადის 46 სტრიქონზე, სადაც სრულდება ნებისმიერი ღილაკის დაჭერის მოლოდინის ციკლი. ეს ციკლი ჰგავს 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 (სტრიქონი 48). შემდეგ ფორმირდება დამცველი პაუზა. (სტრიქონი 49). პაუზის ფორმირებისათვის გამოიყენება ფუნქცია wait, რომელიც აფორმირებს პირველი ტიპის დაყოვნებას (48 მწ). 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 მოლოდინის ციკლი და კოდური კომბინაციის შეყვანის ციკლი მთავრდება. პროგრამა გადადის 56 სყრიქონის 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 (რომელზედაც მიერთებულია გადამრთველი). თუ გადამრთველის კონტაქტები გათიშულია (PINB.7 ერთის ტოლია), პროგრამა გადადის comp ნაჭდევზე (კოდების შედარების პროცედურაზე). წინააღმდეგ შემთხვევაში სრულდება კოდის ჩაწერა EEPROM-ში.

EEPROM-ში კოდის ჩაწერა სრულდება 57-59 სტრიქონებში. 57იქონში 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 bufr 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 მადგანი 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 უბრუნდ3ბა მის საწყის მდგომარეობას (კლიტე იკეტება). კლიტის გაღების პროცედურის დამთავრებასთან ერთად მთავრდება პროგრამის ძირითადი ციკლი. ვინაიდან ძირითადი ციკლი უსასრულოა მართვა გადაეცემა მის დასაწყისს (სტრიქონი 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/მთვლელი1 გადავსების წყვეტის მომსახურების პროცედურას, მეორე – იმავე ტაიმერის A რეგისტრთან თანხვედრის წყვეტის მომსახურების პროცედურას. თვითოეული 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 გამოიყენებიან 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 ამოკითხვა რამოდენიმე ჯერ (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 სრულდემა 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 ერთიანდებიან 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 ნულის მნიშვნრელობა (სტრიქონი 21). 23 სტრიქონში cod1 ცვლადის მნიშვნელობა იწერება 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 kodz. 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 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 48მწ- 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 ტაიმერის და წყვეტის სისტემის გაწყობა მთავრდება,

პროგრამა გადადის მოლოდინის ციკლის შესრულებაზე. მოლოდინის რეჟიმი გათვალისწინებულია 1 და 3 რეჟიმებში სამუშაოდ. (სტრიქონი 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 30სტრიქონში 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.

ლაბორატორიული სამუშაოს დავალება:

1. მიუერთეთ კოდური კლიტის სქემის გამომყვანები სტენდის გარე გასართებს;
2. გააკომპილირეთ პროგრამა და ჩაწერეთ მიკროკონტროლერში;
3. გაუშვით პროგრამა და დააკვირდით მის მუშაობას.

ლიტერატურა.

1. ო.ქართველიშვილი, ც.ხოშტარია,ს.ხოშტარია. მიკროპროცესორული სისტემები ნაწ1. მიკროკონტროლერების არქიტექტურა. “ტექნიკური უნივერსიტეტი”. 2015
2. ო.ქართთველიშვილი, ს.ხოშტარია. მჯროპროცესორული სისტემები ნაწ.2. მიკროპროცესორული სისტემების დაგეგმარება. ელექტრონული ვერსია. 2016
3. Бепов А.С. Создаем устройства на микроконтроллерах. Наука и техника С.Петербург. 2007

სარჩევი