ს. დალუაშვილი

მიკროკონტროლერების გამოყენების პრაქტიკული და პროგრამული უზრუნველყოფა

„ტექნიკური უნივერსიტეტი"

საქართველოს ტექნიკური უნივერსიტეტი

ს. დადუნაშვილი

მიკროკონტროლერების გამოყენების აპარატული და პროგრამული უზრუნველყოფა

დამტკიცებულია სახელმძღვანელოდ
სტუ-ის სარედაქციო-საგამომცემლო
საბჭოს მიერ. 11.10.2010, ოქმი №6

თბილისი
2011

განხილულია მიკროკონტროლერების არქიტექტურის საფუძვლები, **AVR** ოჯახის მიკროკონტროლერების აგების და ფუნქციონირების თავისებურებები, მიკროკონტროლერების პროგრამატორები და მათი გამოყენების შესაძლებლობლები

სხვადასხვა დანიშნულების ციფრული ელექტრონული მოწყობილობების სინთეზისათვის.

განსაზღვრულია **AVR** მიკროკონტროლერების დაპროგრამების პროცესის პროგრამული და ინსტრუმენტული უზრუნველყოფის პრინციპები და აპარატული საშუალებები.

განკუთვნილია ელექტრონიკის, ენერგეტიკის, ტელეკომუნიკაციის და ინფორმატიკის სპეციალობების სტუდენტებისათვის. მისი დახმარებით შესაძლებელია სასწავლო პროცესის უზრუნველყოფა ციფრული ელექტრონიკის, მიკროპროცესორული ტექნიკის და ჩაშენებული სისტემების კურსებში.

რეცენზენტები: სრული პროფესორი კ. კამკამიძე,
სრული პროფესორი გ. დგებუაძე

ISBN 978-9941-14-915-3
http://www.gtu.ge/publishinghouse/

შინაარსი

წინასიტყვაობა

მიკროკონტროლერი (**microcontroller, MCU, µC**) ნახევრად გამტარ კრისტალზე აგებული მოწყობილობაა, რომელიც დამუშავებულია ობიექტებსა და ტექნოლოგიურ პროცესებში მართვის სისტემების ჩაშენებისათვის.

მიკროკონტროლერები გამოიყენება ავტომატურად მართვად სისტემებში, მაგალითად, ავტომანქანების ძრავას მართვის სისტემებში, დისტანციურად მართვად სისტემებში, საოფისე და საყოფაცხოვრებო ტექნიკაში, გამზომ ხელსაწყოებში, სამედიცინო აპარატურაში, აგრეთვე საკომუნიკაციო და სხვა მოწყობილობებში. მიკროკონტროლერები უზრუნველყოფს მაღალ ეკონომიურობას იმ მოწყობილობებსა და პროცესებში, რომელთა მართვა ხორციელდება ციფრულ ფორმაში, ვინაიდან ამცირებს მათ ზომებს და ღირებულებას იმ მოწყობილობებთან შედარებით, რომლებშიც ცალ-ცალკე გამოყენებულია მიკროპროცესორი, მეხსიერება და ინფორმაციის შეტანა/გამოტანის საშუალებები.

ფაქტიურად, მიკროკონტროლერი ეს მინიკომპიუტერია, რომელიც შექმნილია ერთ ინტეგრალურ მიკროსქემაში. აქედან გამომდინარეობს მისი უნიკალური თვისებები: მცირე გაბარიტული ზომა, მაღალი მწარმოებლურობა, საიმედოობა და ადაპტირების შესაძლებლობა სხვადასხვა დავალების შესასრულებლად.

მიკროკონტროლერი შედგება ცენტრალური პროცესორისგან, პროგრამების და მონაცემების ენერგოდამოუკიდებელ დაპროგრამებულ მუდმივი მეხსიერებისაგან, ოპერატიული დამხსომებელი მოწყობილობისა და მრავალრიცხოვანი შეტანა/გამოტანის მოწყობილობებისაგან. შეტანა/გამოტანის მოწყობილობებს მიეკუთვნება ანალოგურ-ციფრული გარდამქმნელები, ინფორმაციის გადაცემის პარალელური და მიმდევრობითი არხები, რეალური დროის ტაიმერები, განედურ-იმპულსური მოდულატორები, იმპულსების დაპროგრამებული გენერატორები და ა.შ. ჰიბრიდული სასიგნალო მიკროკონტროლერები, რომლებიც გამოიყენება იმ სისტემების მართვისათვის, საიდანაც მიიღება არაციფრული მონაცემები, შეიცავს ანალოგურ (არაციფრულ) კომპონენტებს.

მიკროკონტროლერების დამუშავება ხდება მათი სპეციალიზებული გამოყენების მიზნით. ამგვარად, მიკროპროცესორებისაგან განსხვავებით, რომლებიც პერსონალურ კომპიუტერებში და სხვა უნივერსალურ აპარატებში გამოიყენება, მიკროკონტროლერები ასრულებს სპეციალურ ფუნქციებს და დავალებებს დაბალი ენერგომოხმარებით. ისინი ინარჩუნებენ ფუნქციურ შესაძლებლობებს ძაბვის ღილაკის დაჭერის მოლოდინის რეჟიმში ან კვების ძაბვის ხანგრძლივი შეწყვეტის შემდეგ. როდესაც მიკროკონტროლერს "ეძინა", მისი პერიფერიული ელემენტები გათიშული იყო და ელექტროენერგიის ხარჯი ნანოვატებს შეადგენდა.

ამგვარად, მიკროკონტროლერი არის მიკროსქემა, რომელიც განკუთვნილია სხვადასხვა ელექტრონული მოწყობილობების სამართავად. მიკროკონტროლერების ძირითადი დანიშნულებაა იმ ავტომატიზებული მართვის სისტემების გამოყენება,

რომლებიც ჩაშენდება სხვადასხვა მოწყობილობაში: ბანკომატებში, ფოტოაპარატებში, ფიჭურ ტელეფონებში, მუსიკალურ ცენტრებში, ტელევიზორებში, ვიდეოკამერებში, ვიდეომაგნიტოფონებში, სარეცხ მანქანებში, მიკროტალღურ ღუმლებში, დამცავი სიგნალიზაციის სისტემებში, ძრავების აალების სისტემებში, ლოკომოტივის ელექტროამძრავებში და ბევრ სხვა დანადგარებში. ჩაშენებული მართვის სისტემები იმდენად მასიური მოვლენა გახდა, რომ ფაქტიურად მოხდა მეცნიერების, ტექნიკის და ეკონომიკის ახალი დარგის ფორმირება, რომელსაც ჩაშენებადი სისტემები (**Embedded Systems**) ეწოდება.

არსებითად, მიკროკონტროლერი ერთკრისტალიანი კომპიუტერია, რომელსაც სხვადასხვა ამოცანის გადაჭრა შეუძლია. ერთი მიკროკონტროლერის გამოყენება, მიკროსქემების მთელი ნაკრების მაგივრად, მნიშვნელოვნად ამცირებს მიკროკონტროლერების გამოყენებით აწყობილ საბოლოო მოწყობილობის გაბარიტებს, ენერგომოხმარებას და ფასს. მიკროკონტროლერების ასეთი გამოყენება შესაძლებელია მათი წინასწარი დაპროგრამების გზით გადასაწყვეტი ამოცანის შესაბამისად.

მიკროკონტროლერები არის ჩაშენებული სისტემების გამოთვლითი ბირთვი და გამოიყენება მრავალ თანამედროვე მოწყობილობაში. მსოფლიოში წარმოებული პროცესორების უდიდესი ნაწილი სწორედ მიკროკონტროლერებია. თანამედროვე სამრეწველო მიკროკონტროლერების უმნიშვნელოვანესი თვისებაა ის, რომ მათი მუშაობის ლოგიკა შედარებით მარტივ დაპროგრამების საშუალებებს მოითხოვს.

ამას გარდა, მიკროკონტროლერების ეფექტური ჩაშენება ხორციელდება თანამედროვე ინტერფეისების გამოყენებით. **USB** ინტერფეისი არის სწრაფად განვითარებადი სტანდარტი, რომელიც უკვე გამოიყენება არა მარტო მასობრივი წარმოების მოწყობილობებში, არამედ შედარებით ნაკლები რაოდენობით არსებულ და სპეციფიკურ პერიფერიულ აპარატურაში. მაგალითად, კომპიუტერთან **USB** ინტერფეისის საშუალებით დაკავშირებულ პროგრამატორთან მუშაობა გაცილებით უფრო მოსახერხებელია, ვიდრე ნებისმიერი აქამდე არსებული ინტერფეისების გამოყენებისას. მიუხედავად იმისა, რომ **USB** პროგრამატორის აწყობა უფრო რთულია, ვიდრე ჩვეულებრივის. ახალი ინტერფეისი სირთულეებს მნიშვნელოვანწილად აბათილებს.

სახელმძღვანელოში ძირითადი ყურადღება ეთმობა მიკროკონტროლერების დაპროგრამების სწავლების საკითხებს და ამისათვის საჭირო ელექტრონული მოწყობილობების მუშაობის პრინციპების განხილვას, **AMTEL**-ის ფირმის მიკროკონტროლერების მაგალითებზე.

თავი 1. მიკროკონტროლერების არქიტექტურის საფუძვლები

1.1. მიკროკონტროლერების ძირითადი ტიპები და მათი არქიტექტურა

მიკროკონტროლერების წარმოების ისტორია იწყება 1980 წლიდან, როდესაც კომპანია **Intel**-მა გამოუშვა i8048 და i8051 სერიის მიკროკონტროლერები.

ამჟამად არსებობს i8051 სერიასთან შეთავსებელი მიკროკონტროლერების 200-ზე მეტი მოდიფიკაცია და სხვა მრავალი ტიპის მიკროკონტროლერი. დიდი პოპულარობით სარგებლობს 8-ბიტიანი Microchip Technology ფირმის PIC და **Atmel** ფირმის **AVR** მიკროკონტროლერები.

თანამედროვე მიკროკონტროლერების ძირითადი ტიპების განხილვისას ვხვდებით ამ კლასის სხვადასხვა მოწყობილობას, რომლებიც ხელმისაწვდომია მომხმარებლისათვის. ყველა ისინი შეიძლება დავყოთ შემდეგ ძირითად ტიპებად:

- ჩაშენებადი 8-თანრიგიანი მიკროკონტროლერები;
- 16- და 32- თანრიგიანი მიკროკონტროლერები;
- ციფრული სასიგნალო პროცესორები.

მწარმოებელი ფირმები უშვებენ ძალიან ფართო ნომენკლატურის ჩაშენებად მიკროკონტროლერებს. მათში ყველა საჭირო რესურსი (მეხსიერება, შეტანა/გამოტანის მოწყობილობები და სხვ.) განთავსდება ერთ კრისტალზე პროცესორულ ბირთვთან ერთად. თუ კვებასა და ტაქტურ იმპულსებს მივაწვდით მიკროკონტროლერის შესაბამის შესავლელებზე, მაშინ შეგვიძლია ვთქვათ, რომ ის "ცოცხლდება" და მასთან შეიძლება მუშაობა. ჩვეულებრივ, მიკროკონტროლერი შედგება დიდი რაოდენობის დამხმარე მოწყობილობებისგან, რომელთა მეშვეობით ხდება რეალურ სისტემაში მათი ჩართვის უზრუნველყოფა და დამატებითი კომპონენტების მინიმალური რაოდენობის გამოყენება.

ასეთი მიკროკონტროლერების შედგენილობაში შედის:

- პროცესორის საწყისი გაშვების სქემა;
- ტაქტური იმპულსების გენერატორი;
- ცენტრალური პროცესორი;
- პროგრამების მეხსიერება და დაპროგრამების ინტერფეისი;
- მონაცემთა შეტანა/გამოტანის საშუალებები;
- ტაიმერები, რომლებიც აფიქსირებენ ბრძანებათა ციკლების რაოდენობას.

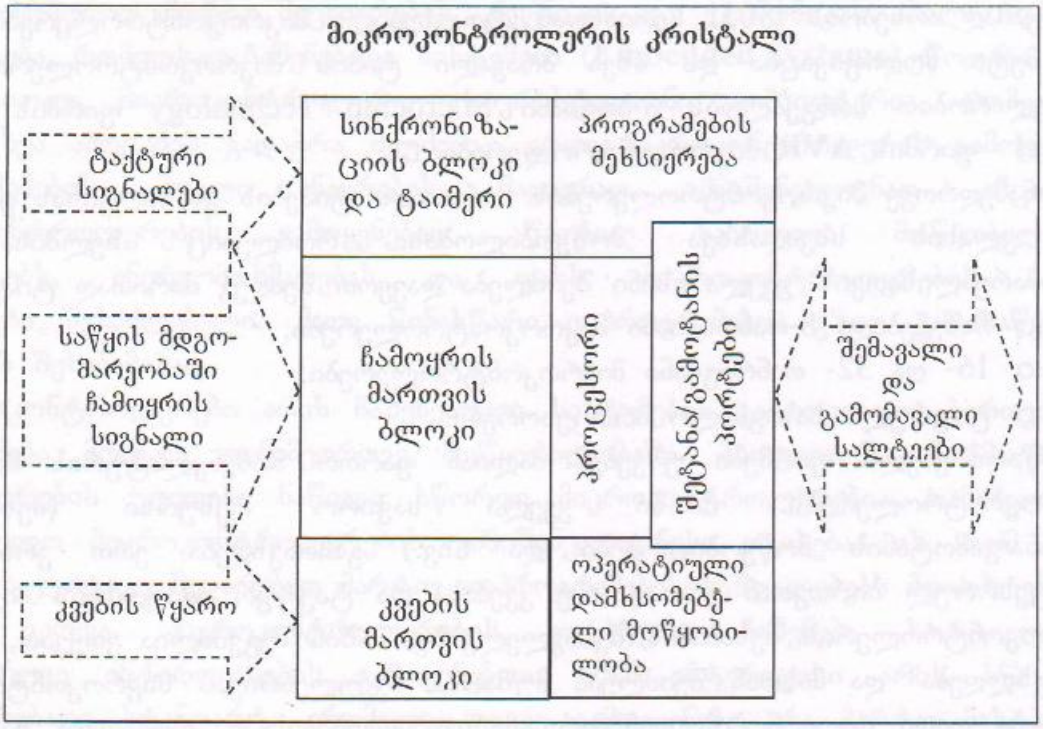
მიკროკონტროლერის ზოგადი სტრუქტურა მოცემულია 1.1 ნახაზზე. ეს სტრუქტურა იძლევა წარმოდგენას იმის შესახებ, თუ როგორ უკავშირდება მიკროკონტროლერი გარე სამყაროს.

უფრო რთულ ჩაშენებად მიკროკონტროლერებს შეუძლია დამატებით მოახდინოს შემდეგი შესაძლებლობების რეალიზება:

- პროგრამების ჩაშენებული მონიტორინგი და გამართვა;
- პროგრამების მეხსიერების დაპროგრამების შიგა საშუალებები;
- სხვადასხვა წყაროდან წყვეტების დამუშავება;
- ანალოგური მონაცემების შეტანა/გამოტანა;
- მიმდევრობითი შეტანა/გამოტანა (სინქრონული და ასინქრონული);

- პარალელური შეტანა/გამოტანა (კომპიუტერთან ინტერფეისის ჩათვლით);
- გარე მეხსიერების ჩართვა (მიკროპროცესორული რეჟიმი).

ყველა ეს შესაძლებლობა მნიშვნელოვნად ამაღლებს მიკროკონტროლერის გამოყენების მოქნილობას და აადვილებს მის ბაზაზე ახალი სისტემის დამუშავების პროცესს.



ნახ. 1.1. მიკროკონტროლერის სტრუქტურა

ზოგიერთი მიკროკონტროლერი (ჩვეულებრივ 16- და 32-თანრიგიანი) იყენებს მხოლოდ გარე მეხსიერებას, რომელიც შეიცავს როგორც პროგრამების, ისე გარკვეული მოცულობის მონაცემთა მეხსიერებას, რომელიც საჭიროა კონკრეტული გამოყენებისათვის. ასეთი მიკროკონტროლერები გამოიყენება ისეთ სისტემებში, სადაც მოითხოვება მეხსიერების დიდი მოცულობა და შედარებით მცირეა შეტანა/გამოტანის პორტების რაოდენობა. ასეთ გარემეხსიერებიანი მიკროკონტროლერის ტიპური გამოყენების მაგალითია ხისტი დისკის (ვინჩესტერის) კონტროლერი ბუფერული ქეშ (დაფარული) მეხსიერებით, რომელიც უზრუნველყოფს მონაცემების დიდი მოცულობის (რამდენიმე მეგაბაიტი) შუალედურ შენახვას და გავრცელებას. ამასთან, გარე მეხსიერება ასეთ მიკროკონტროლერს საშუალებას აძლევს იმუშაოს მაღალი სიჩქარით.

ციფრული სასიგნალო პროცესორები (**DSP**) – შედარებით ახალი მიკროკონტროლერების კატეგორია, რომელთა დანიშნულებაა ანალოგიური სისტემებიდან მიმდინარე მონაცემების მიღება, მონაცემების გადამუშავება და დროის რეალურ მასშტაბში შესაბამისი საპასუხო სიგნალის ფორმირება. ისინი შედიან ისეთი სისტემების შედგენილობაში, რომლებიც გარე მოწყობილობების მართვისათვის გამოიყენება და არ არის განკუთვნილი ავტონომიური გამოყენებისათვის.

მიკროკონტროლერის ბრძანებების სისტემები

გამოსაყენებელი ოპერაციების კოდების რიცხვზე დამოკიდებულებით, მიკროკონტროლერების ბრძანებების სისტემები ორ ჯგუფად იყოფა: **CISC** და **RISC**. ინგლისური ტერმინი **CISC** აღნიშნავს ბრძანებების რთულ სისტემას და არის ინგლისური განმარტების აბრევიატურა, რომელიც შემდეგნაირად ჩაიწერება: **Complex Instruction Set Computer**. ტერმინი **RISC** აღნიშნავს ბრძანებების შემცირებულ სისტემას და წარმოქმნილია განმარტებიდან: **Reduce Instruction Set Computer**. ამ ცნებების ფართოდ გავრცელების მიუხედავად, აუცილებელია აღინიშნოს, რომ თავად დასახელება არ ასახავს მთავარ განსხვავებას **CISC** და **RISC** ბრძანებების სისტემებს შორის. **RISC** არქიტექტურის ძირითადი იდეა ოპერაციების კოდების ისეთი კომბინაციების საფუძვლიანი შერჩევაა, რომლებიც შეიძლება შესრულდეს ტაქტური გენერატორის ერთ ტაქტში. ამით ხდება ცენტრალური პროცესორის აპარატული რეალიზაციის მკვეთრი გამარტივება და მისი მწარმოებლურობის მნიშვნელოვანი გაზრდის შესაძლებლობა.

აღსანიშნავია, რომ ზოგად შემთხვევაში **CISC**-ის ერთ ბრძანებას შეესაბამება **RISC**-ის რამდენიმე ბრძანება. მაგრამ მომგებიანია ის, რომ **RISC** სისტემის სწრაფქმედება ფარავს დანაკარგებს მიღებულს ნაკლებად ეფექტური ბრძანებების სისტემისაგან, რაც იწვევს **RISC** სისტემების უფრო მაღალ ეფექტურობას **CISC** სისტემებთან შედარებით.

დღეისათვის **CISC**-ის და **RISC**-ის არქიტექტურებს შორის ზღვარი თანდათან იშლება. მაგალითად, **ATMEL**-ის ფირმის **AVR** მიკროკონტროლერებს გააჩნია ბრძანებების სისტემა, რომელიც შედგება 120 ინსტრუქციისაგან, რაც **CISC**-ის ტიპს შეესაბამება. ბევრი მათგანი სრულდება ერთ ტაქტში, რაც **RISC**-ის არქიტექტურის დამახასიათებელი თვისებაა. დღეს შეიძლება ჩაითვალოს, რომ **RISC**-ის არქიტექტურის თვისებას წარმოადგენს ბრძანებების შესრულება ტაქტური გენერატორის ერთ ტაქტში და ბრძანებების რიცხვს მნიშვნელობა აღარ აქვს.

მიკროკონტროლერის მეხსიერებების ტიპები

გამოვყოთ მიკროკონტროლერში არსებული მეხსიერების სამი ძირითადი სახეობა:

1. *პროგრამების მეხსიერება;*
2. *მონაცემთა მეხსიერება;*
3. *მიკროკონტროლერის რეგისტრები.*

პროგრამების მეხსიერება არის მუდმივი მეხსიერება, რომელიც განკუთვნილია პროგრამული კოდებისა და კონსტანტების შენახვისათვის. ეს მეხსიერება არ ცვლის შემცველობას პროგრამის შესრულების პროცესში. ***მონაცემთა მეხსიერება*** განკუთვნილია ცვლადების დამახსოვრებისათვის პროგრამის შესრულებისას. ***მიკროკონტროლერის რეგისტრები*** – მეხსიერების სახეობა, რომელიც მოიცავს პროცესორის შიგა რეგისტრებს და რეგისტრებს, რომლებიც ემსახურება პერიფერიული მოწყობილობების მართვას.

ჩვეულებრივ, პროგრამების შენახვას ემსახურება ერთ-ერთი სახეობის მუდმივი მეხსიერება: **ROM** (Read-Only Memory) - ნიღაბური მუდმივი მეხიერება, **PROM**

(Programmable Read-Only Memory) - ერთჯერადად დაპროგრამებული მუდმივი მეხსიერება), **EPROM** (Electrically Programmable Read-Only Memory) – ელექტრულად პროგრამირებადი მუდმივი მეხსიერება ულტრაიისფერი წაშლით ან **EEPROM** (Electrically Erasable Programmable Read-Only Memory) - მუდმივი მეხსიერება ელექტრულად ჩაწერით და წაშლით. ამავე სახეობას მიეკუთვნება თანამედროვე **Flash**-მეხსიერების (მეხსიერება ჯგუფური გადაწერით) მიკროსქემები. მეხსიერების ყველა ეს სახეობა ენერგოდამოუკიდებელია, რაც ნიშნავს, რომ შემცველი მონაცემების შენახვა ხდება მიკროკონტროლერის კვების გამორთვის შემდეგაც.

მრავალჯერადად პროგრამირებადი მუდმივი **EPROM** და **EEPROM** მეხსიერებები იყოფა მუდმივ მეხსიერებად, რომელიც მიიღება ჩაწერით ულტრაიისფერი დასხივებით (მათი გამოშვება ხდება კორპუსში არსებული პატარა სარკმლით) და ელექტრულად კვლავპროგრამირებად მუდმივ მეხსიერებად.

თანამედროვე **EEPROM** მეხსიერების პროგრამირების პროტოკოლები საშუალებას გვაძლევს შევასრულოთ მიკროკონტროლერის დაპროგრამება უშუალოდ იმ მოწყობილობაში, სადაც მუშაობს. პროგრამირების ასეთმა ხერხმა მიიღო სახელწოდება - **ISP** (In System Programming). ამ შემთხვევაში შეიძლება პერიოდულად განახლდეს მიკროკონტროლერის პროგრამული უზრუნველყოფა პლატადან ამოუღებლად. ეს უდიდეს მოგებას იძლევა საწყის ეტაპზე მიკროკონტროლერების ბაზაზე სისტემების შემუშავებისას ან მათი შესწავლის პროცესში, როცა დროის უდიდესი ნაწილი იხარჯება მისი მუშაობის უუნარობის მიზეზის მრავალჯერად ძიებაში და პროგრამების მეხსიერების წაშლაში, დაპროგრამების შემდეგი ციკლების შესასრულებლად.

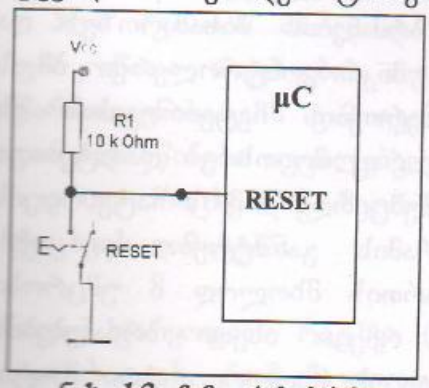
Flash-მეხსიერების ფუნქციონირება მცირედ განსხვავდება **EEPROM**-სგან. ძირითადი განსხვავება ჩაწერილი მონაცემების წაშლის შესაძლებლობაა. **EEPROM** მეხსიერებაში წაშლა ხდება ცალკე ყოველი უჯრედისათვის, ხოლო **Flash**-მეხსიერებაში წაშლა - მთლიანი ბლოკებით.

ოპერატიული დამხსომებელი მოწყობილობა **RAM** (Random Access Memory) გამოიყენება მონაცემების დახსომებისათვის. ამ მეხსიერებას უწოდებენ აგრეთვე მონაცემთა მეხსიერებას. მონაცემების წაკითხვისა და ჩაწერის ციკლების რიცხვი, ოპერატიული დამხსომებელი მოწყობილობისათვის შეუზღუდავია, მაგრამ კვების გამორთვის შემდეგ ხდება ყველა მონაცემის წაშლა.

1.2. მიკროკონტროლერების აპარატული საშუალებები მიკროკონტროლერების საწყის მდგომარეობაში ჩამოყრა

1.2 ნახაზზე გამოსახულია ჩამოყრის სქემა. ამ სქემის გამოყენება მიზანშეწონილია იმ შემთხვევაში, როცა გარანტირებულია კვების წყაროს ძაბვის შენარჩუნება სამუშაო დიაპაზონში. ღილაკი **RESET** (*ჩამოყრა*) გამოიყენება მოწყობილობის დამუშავების პროცესში, მიკროკონტროლერის საწყის მდგომარეობაში "ხელით" ჩამოყრისათვის.

მიკროკონტროლერების პრაქტიკული გამოყენების დროს, უმეტეს შემთხვევაში, გამოიყენება ბატარეიანი კვება. ზოგ შემთხვევაში გამოიყენება აგრეთვე დიდი ტევადობის კონდენსატორები, რომლებიც უზრუნველ-ყოფენ მუშაობის უნარის შენარჩუნებას, კვების მოკლე დროით გამორთვისას. ამიტომ ენერგოდაზოგვის პრობლემა ყოველთვის აქტუალურია მიკროკონტროლერებისათვის. იმის გამო, რომ პრაქტიკულად ყველა თანამედროვე მიკროკონტროლერი მზადდება **CMOS** (complementary metal-oxide semiconductor) ტექნოლოგიით, მოითხოვს მნიშვნელოვნად მცირე სიმძლავრეს, ადრე გამოშვებულ ბიპოლარულ ან **NMOS** (negative channel metal-oxide semiconductor) მიკროკონტროლერებთან შედარებით.

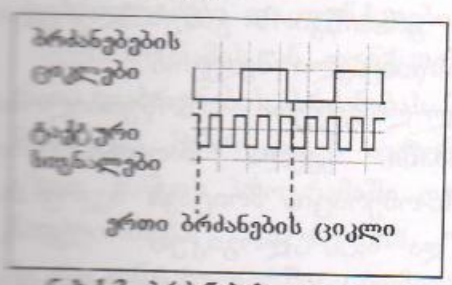


ნახ. 1.2. ჩამყრის სქემა

ტაქტირება და ბრძანებების ციკლები

არსებობს მიკროკონტროლერების ტაქტური სიხშირის დაკვეთის და, შესაბამისად, მისი სინქრონიზაციის სამი ხერხი:

- პირველი ხერხი – კვარცული რეზონატორის გამოყენება;
- მეორე ხერხი – **RC** გენერატორის გამოყენება;
- მესამე ხერხი – ტაქტური იმპულსების მიწოდება გარე გენერატორიდან. ამ შემთხვევაში შესაძლებელია სინქრონიზაციის ნებისმიერი სიხშირის დაკვეთა.



ნახ.1.3. ბრძანების ციკლი და მანქანური ტაქტი

აღსანიშნავია, რომ ბრძანებების ციკლები და სინქრონიზაციის ტაქტები არ არის ერთი და იგივე. ბრძანების ციკლი რამდენიმე ტაქტისაგან შედგება, რომლებიც სჭირდება პროცესორს დავალების შესასრულებლად. 1.3 ნახაზზე ნაჩვენებია ბრძანების ციკლი, რომელიც ოთხი ტაქტისაგან შედგება.

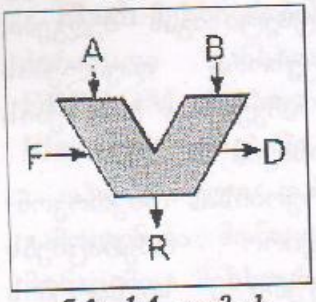
ბრძანების ციკლის მიმდინარეობისას მიკროკონტროლერი ასრულებს საჭირო ოპერაციებს სინქრონიზაციისათვის ტაქტური სიგნალების გამოყენებით. ზოგი ბრძანების შესასრულებლად საჭიროა მიკროკონტროლერის ერთზე მეტი ბრძანების ციკლი.

1.2.1. პროგრამული მთვლელი და არითმეტიკულ-ლოგიკური მოწყობილობა

პროგრამული მთვლელი (**Program Counter**) ანუ ბრძანებების მთვლელი (**ბმ**) გამოიყენება მოქმედ პროგრამაში თითოეული მომდევნო ბრძანების ჩვენებისათვის. ამ ფუნქციის რეალიზაცია შესამჩნევად რთულდება, როცა აუცილებელია მოქმედი პროგრამის შეთავსის შენახვა ქვეპროგრამის გამოძახებისას და წყვეტის შეკითხვების დამუშავებისას ან როცა საჭიროა უზრუნველვყოთ პროგრამის განშტოება. **ბმ** ტრადიციულად მთვლელია, მონაცემების პარალელური შეტანა/გამოტანით. ფონ-ნეიმანისარქიტექტურიან პროცესორების მეხსიერების მართვის სქემაში **ბმ**-ის შეთავსი მიეწოდება მონაცემთა სალტით, სადაც მიეთითება წასაკითხი ბრძანების მისამართი. ხშირად **ბმ** შედის მეხსიერების მართვის სქემის შედგენილობაში, რაც

საშუალებას გვაძლევს ავიცილოთ მისამართის გადაცემა შიგა მონაცემთა სალტით. მნიშვნელოვან თავისებურებას **ბმ**-ების ფუნქციონირებაში წარმოადგენს ახალი შიგთავსის პარალელური ჩატვირთვა მონაცემთა სალტიდან, ჩამოყრის შესაძლებლობა (დაბრუნება პროგრამის პირველი ბრძანების მისამართზე) და ინკრემენტის (ზრდის) რეალიზაცია. 8-თანრიგიან მიკროკონტროლერში **ბმ**-ის თანრიგობა 8 ბიტზე მეტია. **ბმ**-ში ახალი მისამართების ჩატვირთვისას ის მიეწოდება მონაცემთა სალტით 8-ბიტიან ნაწილებად, რაც მოითხოვს დამატებითი მანქანური ციკლების შესრულებას. იმისათვის, რომ შემცირდეს **ბმ**-ის ჩატვირთვის დრო, ზოგ მიკროკონტროლერს შეუძლია პროგრამის განშტოება და აქვს შესაბამისი ბრძანება, რომლითაც იტვირთება მისამართის მხოლოდ 8 უმცროსი თანრიგი, ხოლო უფროსი თანრიგები უცვლელი რჩება. ასეთი ბრძანებების შესრულებისას საკმარისია მონაცემთა სალტით გადაეცეს მხოლოდ ერთი ბაიტი, მაშინ, როდესაც სრული 16-რიგიანი მისამართის ჩატვირთვისათვის საჭიროა ორი ბაიტის გადაგზავნა.

მორიგი ბრძანების წაკითხვის შემდეგ მოქმედი **ბმ**-ის შიგთავსი იზრდება (ინკრემენტირებს), რათა უზრუნველყოფილ იქნეს შემდეგი ბრძანების მისამართზე გადასვლა. თუ სრულდება ქვეპროგრამის გამოძახება ან ხდება წყვეტა, მაშინ დაბრუნების მისამართი (**ბმ**-ის მიმდინარე მდგომარეობა) შეიძლება შენახულ იქნეს სტეკში (მეხსიერების არე), დამატებითი ტაქტების შესრულების გარეშე **ბმ**-ის შიგთავსის ინკრემენტირებისათვის. მიკროკონტროლერთან მუშაობისას აუცილებელია შემოწმდეს, რომ **ბმ**-ის მნიშვნელობა არ გამოვიდეს პროგრამების მეხსიერების ფარგლებიდან, რადგან ამან შეიძლება გამოიწვიოს განუსაზღვრელი ბრძანებების შესრულება და, შესაბამისად, გაუთვალისწინებელი შედეგი.



ნახ. 1.4. ალმ-ის სტრუქტურა

პროცესორის არითმეტიკულ-ლოგიკური მოწყობილობა (**ალმ**) გამოიყენება პროგრამაში ყველა მათემატიკური ოპერაციის შესასრულებლად. ეს ოპერაცია მოიცავს შეკრებას, გამოკლებას, ლოგიკურ "**და**"-ს, ლოგიკურ "**ან**"-ს, რეგისტრების შიგთავსის წანაცვლებას და მდგომარეობის რეგისტრის შიგთავსის დაყენებას მიღებული შედეგების შესაბამისად. **ალმ** არ გამოიყენება მონაცემების და ბრძანებების წაკითხვისას ან ჩაწერისას. ის ემსახურება მხოლოდ მონაცემთა დამუშავებას.

ალმ შეიძლება წარმოვიდგინოთ, როგორც აპარატული ბლოკი, რომელიც ამუშავებს მონაცემთა ორ სიტყვას (ოპერანდებს) და ინახავს მიღებულ შედეგს. 1.4 ნახაზზე ნაჩვენებია **ალმ**-ის აღნიშვნა. ოპერანდი 1 მიეწოდება **A** შესასვლელზე, ხოლო ოპერანდი 2 - **B** შესასვლელზე. შედეგი მიიღება **R** გამოსასვლელზე. **F** შესასვლელზე მიეწოდება სიგნალები მართვის მოწყობილობიდან. **D** გამოსასვლელზე აისახება მდგომარეობის ბიტები (ალარმები).

როგორ ხდება ოპერანდების შეტანა **ალმ**-ში და სად მიეწოდება შედეგი, დამოკიდებულია მიკროკონტროლერის კონკრეტულ ტიპზე. ეს ერთ-ერთი ძირითადი განსხვავებაა სხვადასხვა ტიპის პროცესორებსა და ბრძანებათა სისტემებს შორის. ზოგი მიკროკონტროლერი ირჩევს ერთ ოპერანდს რეგისტრ-

აკუმულატორიდან და შედეგს ინახავს აკუმულატორშივე. სხვა მიკროკონტროლერები იყენებს ოპერანდების სხვადასხვა წყაროს და შედეგების განთავსების ადგილებს.

ალმ მუშაობს მხოლოდ მთელი დადებითი რიცხვებით. გამოკლებისას მიიღება უარყოფითი რიცხვები, თუ მაკლები მეტია საკლებზე. უარყოფითი რიცხვების წარმოდგენისათვის გამოიყენება დამატებითი კოდი – "დამატება ორამდე". ამის გათვალისწინება აუცილებელია **ალმ**-ის მუშაობის გაცნობისას.

ერთი რიცხვიდან მეორის სხვაობის გამოსათვლელად საჭიროა უარყოფითი რიცხვის დამატება:

A-B=A+(-B),

სადაც უარყოფითი რიცხვი (**-B**) წარმოდგენილია დამატებით კოდში.

იმისათვის, რომ მივიღოთ უარყოფითი ორობითი რიცხვის დამატებითი კოდი, საჭიროა ყოველი ბიტი მნიშვნელობის ინვერტირება, შემდეგ შედეგს ემატება ერთი:

-B=(B^0xff)+1,

სადაც 0xff არის ბიტის მისამართი მეხსიერებაში;

^ არის "ურთიერთგამორიცხვის **ან**"-ის ოპერაცია.

ალმ-ის სირთულე ხშირად განსაზღვრავს მთლიანად მიკროკონტროლერის სირთულეს. **ალმ**-ის მუშაობაზე დამოკიდებულია მიკროკონტროლერის შედგენილობაში არსებული პროცესორის ფუნქციონირება, რაც მთლიანობაში ასახავს მიკროკონტროლერის ფუნქციონირებას.

ქვეპროგრამები და ფუნქციები

ქვეპროგრამის ან ფუნქციის გამოძახება მოითხოვს **ბმ**-ის შიგთავსის შენახვას, რათა დაბრუნების ბრძანებას შეეძლოს მართვა დააბრუნოს საწყის პროგრამაზე. ეს შეიძლება შესრულდეს ავტომატურად, დაბრუნების მისამართის სტეკში შენახვის გზით. საწყის პროგრამაში დაბრუნებისას მისამართი სტეკიდან ამოიღება და **ბმ**-ში ჩაიტვირთება. ფუნქციის გამოძახება შეიძლება რეალიზებულ იქნეს მიკროკონტროლერში, რომელსაც არ გააჩნია სტეკი, სტეკის ემულაციისათვის (ირიბი აღწარმოებისათვის) ინდექსური რეგისტრის გამოყენებით. თუ სტეკში **ბმ**-ის შიგთავსის ჩატვირთვა შეუძლებელია, მაშინ საწყის პროგრამაში დაბრუნების მისამართი შეიძლება შენახულ იქნეს ემულირებულ სტეკში.

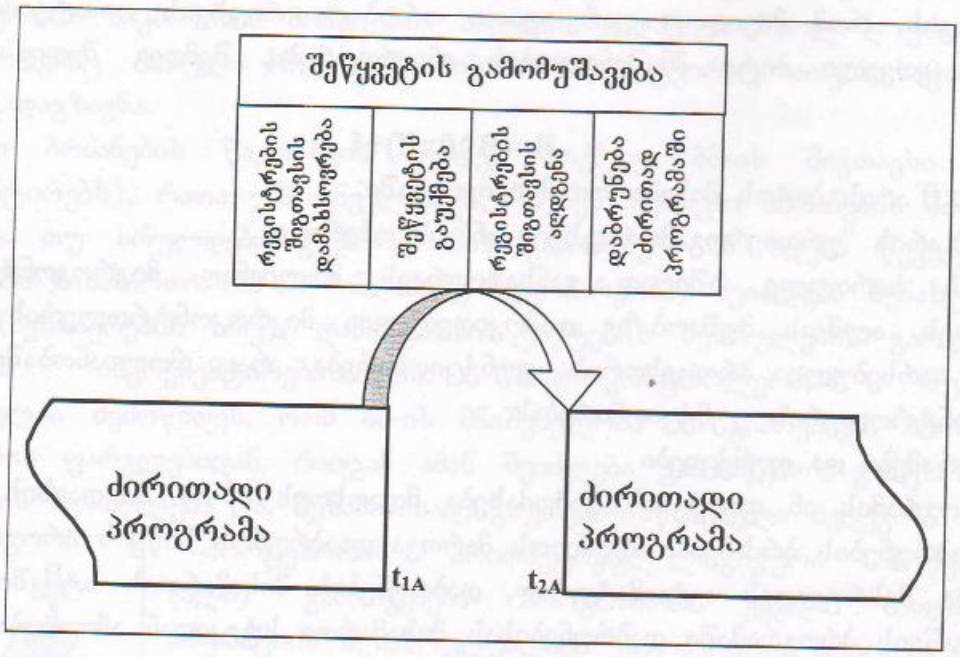
ზოგჯერ საჭიროა ფუნქციისათვის ზოგიერთი პარამეტრის მიწოდება. ერთერთი ყველაზე ეფექტური საშუალება პარამეტრების გადაცემის ფუნქციაში მათი მოთავსებაა სტეკში ფუნქციის გამოძახების წინ. ამ შემთხვევაში ქვეპროგრამაში შეიძლება ჩაიტვირთოს სტეკის მაჩვენებლის მნიშვნელობის ინდექსური რეგისტრი და შედეგად შევაღწიოთ პარამეტრებში.

პარამეტრების გადაცემის სხვა საშუალება მათი შენახვაა პროცესორის რეგისტრებში ან მონაცემთა მეხსიერებაში, როგორც სპეციალური ცვლადები. პარამეტრების გადაცემა რეგისტრების გავლით ამცირებს რეგისტრების რაოდენობას, რომლებიც მისაწვდომია ფუნქციის შესრულებისას. პარამეტრების შენახვა სპეციალური ცვლადების სახით ამცირებს მეხსიერების მოცულობას, რომელიც შეღწევადია პროგრამის გამოყენებისათვის. აღნიშნული შეზღუდვა

ძალიან მნიშვნელოვანია მიკროკონტროლერისათვის. ჩვეულებრივ, დასაბრუნებელი პარამეტრების მნიშვნელობები ჩაიტვირთება პროცესორის რეგისტრებში, ვინაიდან ეს ყველაზე სწრაფი და ეფექტური საშუალებაა მონაცემთა გადასაცემად.

შეწყვეტები

შეწყვეტა – სპეციალური ქვეპროგრამის გაშვება, რომელსაც ეწოდება "შეწყვეტის დამმუშავებელი" ან "შეწყვეტის მომსახურების პროგრამა" და რომელიც გამოიძახება მიკროკონტროლერის აპარატურის სიგნალით. ამ ქვეპროგრამის შესრულებისას, მიმდინარე პროგრამა მიკროკონტროლერში ჩერდება (ნახ. 1.5). ტერმინი "შეწყვეტის მოთხოვნა" (**interrupt request**) გამოიყენება, როდესაც პროგრამა უარს ამბობს შეწყვეტის დადასტურებაზე და შეწყვეტის დამუშავების მყის შესრულებაზე.



ნახ. 1.5. შეწყვეტის შესრულება

მაგალითად, მიკროკონტროლერმა შეიძლება არ მოახდინოს რეაგირება შეწყვეტაზე, სანამ არ დამთავრდება მიმდინარე დავალების შესრულება, რაც რეალიზდება შეწყვეტის მოთხოვნის მომსახურების აკრძალვით (შენიღბვა). დავალების შესრულების შემდეგ შეიძლება მივიღოთ ორი ვარიანტიდან ერთი-ერთი: ნიღაბის გაუქმება და შეწყვეტის მომსახურების ნებართვა, რაც იწვევს შეწყვეტის დამმუშავებლის გამოძახებას ან ბიტების მნიშვნელობების ანალიზი, რომელიც მიუთითებს შეწყვეტის მოთხოვნებსა და მომსახურების პროგრამის შეუჩერებლივ შესრულებაზე, შეწყვეტის დამმუშავებლის გამოძახების გარეშე. შეწყვეტის დამუშავების ასეთი მეთოდი გამოიყენება, როდესაც საჭიროა ძირითადი პროგრამის შესრულების მოცემული დროის უზრუნველყოფა, რადგან ნებისმიერი შეწყვეტა შეიძლება აუცილებელი ინტერფეისის რეალიზაციამ დაარღვიოს.

პროგრამის დამმუშავებელი ყოველთვის უზრუნველყოფს მოქმედებათა შემდეგ თანამიმდევრობას:

1. კონტექსტის რეგისტრების შიგთავსის **შენახვა;**
2. შეწყვეტის კონტროლერის და მოწყობილობის **ჩამოყრა,** რომელმაც გამოიწვია მოთხოვნა;
3. მონაცემთა **დამუშავება;**
4. კონტექსტის რეგისტრების შიგთავსის **აღდგენა;**
5. შეწყვეტილ პროგრამაზე **დაბრუნება.**

კონტექსტის რეგისტრები – განსაზღვრავს ძირითადი პროგრამის შესრულების მიმდინარე მდგომარეობას. ჩვეულებრივ, მათ მიაკუთვნებენ **ბმ**-ებს, მდგომარეობის რეგისტრებს და აკუმულატორებს. მიკროკონტროლერის სხვა რეგისტრები, მაგალითად, ინდექსური რეგისტრები, შეიძლება გამოყენებულ იქნეს შეწყვეტის დამუშავების პროცესში, ამიტომ მათი შიგთავსის შენახვა აუცილებელია. სხვა დანარჩენი რეგისტრები სპეციფიკურია ყოველი კონკრეტული მიკროკონტროლერის ტიპისა და მათი გამოყენების სფეროებისათვის.

საწყის მდგომარეობაში ჩამოყრის შემდეგ შეწყვეტის კონტროლერი მზადაა აღიქვას მომდევნო მოთხოვნა, ხოლო მოწყობილობა, რომელიც შეწყვეტის გამომწვევია, მზადაა გააგზავნოს მოთხოვნები, როცა წარმოიქმნება შესაბამისი მიზეზი. თუ მიიღება შეწყვეტის ახალი მოთხოვნა, მაშინ პროცესორის შეწყვეტის შენილბვის რეგისტრი შეაჩერებს მორიგი შეწყვეტის დამუშავებას, მაგრამ შეწყვეტის მდგომარეობის რეგისტრი აფიქსირებს იმ მოთხოვნას, რომელიც დაელოდება თავის მომსახურებას. მიმდინარე შეწყვეტის მომსახურების დასრულების შემდეგ შეწყვეტის შენიღბვა გაუქმდება და ახლად მიღებული მოთხოვნა დამუშავებაზე მიეწოდება.

ჩასმული შეწყვეტების შესრულება რთულია იმ ტიპის მიკროკონტროლერებისათვის, რომლებსაც არ გააჩნია სტეკი. ასეთმა შეწყვეტებმა დანარჩენ მიკროკონტროლერებში შეიძლება გამოიწვიოს პრობლემები, რომლებიც დაკავშირებულია სტეკის გადავსებასთან.

ზოგჯერ მიკროკონტროლერებს შეუძლია სწრაფად მოახდინოს რეაგირება შეწყვეტის მოთხოვნაზე საჭირო მონაცემების მიღებით, რომლებიც გამოიყენება მიმდინარე დავალების შესრულების შემდეგ. ეს რეალიზდება მიწოდებული მონაცემების შენახვის გზით მეხსიერების მასივში და მათი შემდგომი დამუშავებით საწყისი პროგრამის შესრულების დამთავრების შემდეგ. მომსახურების ასეთი ხერხი კარგი კომპრომისია შეწყვეტების სრული დამუშავებისას, რომელმაც შეიძლება მოითხოვოს დიდი დრო და შეწყვეტების იგნორირება, რაც მიგვიყვანს შეწყვეტის გამომწვევი მოვლენის შესაბამისი მონაცემების დაკარგვამდე.

შეწყვეტის დამუშავებისას მდგომარეობის რეგისტრის შიგთავსი, ჩვეულებრივ (მაგრამ არა ყოველთვის), ავტომატურად ინახება შეწყვეტის დამუშავებამდე **ბმ**-ის შიგთავსთან ერთად. ეს აგვაცილებს იმის აუცილებლობას, რომ შევინახოთ იგი მიკროკონტროლერის მეხსიერებაში პროგრამული საშუალებებით გადაგზავნის ბრძანების დახმარებით და შემდგომ აღვადგინოთ იგი საწყის პროგრამაში დაბრუნებისას. ასეთი ავტომატური შენახვის რეალიზაცია ყველა ტიპის მიკროკონტროლერში არ ხდება.

თუ მდგომარეობის რეგისტრის შიგთავსი შეინახება შეწყვეტის დამუშავების დაწყებამდე, მაშინ დაბრუნების ბრძანებით ხდება მისი ავტომატური განახლება. თუ სხვა რეგისტრების შიგთავსი იცვლება შეწყვეტის მომსახურების შესრულებისას, მაშინ ის ასევე უნდა იქნეს შენახული მეხსიერებაში ცვლილებამდე და აღდგენილი ძირითად პროგრამაში დაბრუნების წინ.

"შეწყვეტის ვექტორი" – მისამართი, რომელიც **ბმ**-ში ჩაიტვირთება შეწყვეტის შემუშავებაზე გადასვლისას. არსებობს ვექტორების რამდენიმე ტიპი. მისამართს, რომელიც **ბმ**-ში ჩაიტვირთება მიკროკონტროლერის გაშვებისას (**RESET**) "ჩამოყრის ვექტორი" ეწოდება. განსხვავებული შეწყვეტებისათვის განსხვავებული ვექტორები მოიცემა. თუმცა, ზოგჯერ განსხვავებულ შეწყვეტებს უნიშნავენ ერთ ვექტორს. ეს არ გამოიწვევს პრობლემებს მიკროკონტროლერის მუშაობისას, რადგან ყველაზე ხშირად ის იყენებს ერთადერთ პროგრამას. მიკროკონტროლერებში, სადაც აპარატული ნაწილი კარგადაა ცნობილი, ადგილი არ აქვს პრობლემას, შეწყვეტების ვექტორების ერთობლივი გამოყენების დროს.

დასასრულ, შეიძლება დავამატოთ, რომ სისტემური ქვეპროგრამები ერთგვარი პროგრამული შეწყვეტებია, რომლებიც სპეციალური პროცესორული ბრძანებებით ახდენს აპარატული შეწყვეტის იმიტაციას. ისინი განლაგებულია მეხსიერების ნებისმიერ ადგილას და შეუძლია მოითხოვოს სეგმენტთშორისი გადასვლები, პროგრამაში შეღწევისათვის.

ტაიმერები

მიკროკონტროლერებში ტაიმერები გამოიყენება არა მარტო მოცემული დაყოვნების უზრუნველყოფისათვის, არამედ უფრო ფართო წრის დავალებების გადასაწყვეტად. ჩვეულებრივ, ტაიმერების გადასართავად იყენებენ პროცესორის ტაქტურ იმპულსებს. ტაიმერში ჩაიტვირთება საწყისი მნიშვნელობა და შემდეგ შეიძლება აითვალოს დროის განსაზღვრული ინტერვალები, თუ ინტერვალის დასასრულს დაფიქსირდება ტაიმერის გადავსება. ხშირად, ტაიმერის წინ რთავენ ტაქტური სიხშირეების წინასწარ დამყოფს იმისათვის, რომ შესაძლებელი იყოს დროის უფრო ხანგრძლივი ინტერვალების ათვლა. დამყოფი უზრუნველყოფს ტაიმერის შიგთავსის ინკრემენტს, ტაქტური იმპულსების განსაზღვრული რიცხვის მიღების შემდეგ.

ზოგადად, ტაიმერები შეიძლება გამოყენებულ იქნეს დროის ინტერვალების ზუსტი ფორმირებისათვის, მიკროკონტროლერის გამოსასვლელებზე იმპულსების დათვლისათვის, იმპულსების თანამიმდევრობის ფორმირებისათვის, მიმდევრობითი კავშირის არხების მიმღებ-გადამცემის ტაქტირებისათვის. ტაიმერ-მრიცხველებს შეუძლია გამოიმუშაოს შეწყვეტების მოთხოვნები და გადართოს ცენტრალური პროცესორი (**ცპ**) მათ მომსახურებაზე მოვლენების მიხედვით და ამით გამოათავისუფლოს **ცპ** ტაიმერების მდგომარეობის პერიოდული გამოკითხვის აუცილებლობისაგან. იმის გამო, რომ მიკროკონტროლერები ძირითადად გამოიყენება რეალური დროის სისტემებში, ტაიმერ-მრიცხველები წარმოადგენს მათ სავალდებულო ელემენტებს. მიკროკონტროლერების ზოგიერთ მოდიფიკაციაში ტაიმერების რიცხვი 32 აღწევს.

1.2.2. მონაცემების შეტანა/გამოტანა

ძირითადი ინტერფეისი მიკროკონტროლერებსა და გარე მოწყობილობებს შორის რეალიზდება მონაცემთა შეტანა/გამოტანის პარალელური პორტების გავლით. ბევრ მიკროკონტროლერში ამ პორტების გამომყვანები ასევე ემსახურება სხვა ფუნქციების შესრულებას, მაგალითად, მიმდევრობით და ანალოგურ შეტანა/გამოტანას.

ბევრ მიკროკონტროლერში პორტების განცალკევებული გამომყვანები შეიძლება დაპროგრამებული იყოს მონაცემების შეტანაზე ან გამოტანაზე. აუცილებელია განსაკუთრებული ყურადღება გამახვილდეს იმაზე, რომ მონაცემთა შეტანის დროს იკითხებოდეს იმ სიგნალის მნიშვნელობა, რომელიც მიეწოდება გარე გამომყვანზე და არა მონაცემთა ტრიგერის შიგთავსზე. თუ გარე გამომყვანებზე ჩართულია სხვა მოწყობილობის გამომყვანები, მაშინ მათ შეუძლია დააყენონ გამოსავალი სიგნალის საკუთარი დონე, რომელიც წაიკითხება ტრიგერში ჩაწერილი მონაცემების მოსალოდნელი მნიშვნელობების ნაცვლად. ზოგიერთ მიკროკონტროლერში მონაცემების წაკითხვისას არის არჩევანის შესაძლებლობა, რომლებიც დამყარებულია ტრიგერის გამოსასვლელზე ან გარე გამომყვანზე.

როცა მონაცემების სალტეზე საჭიროა გამოტანილ იქნეს "0" ან "1", მაშინ დასაწყისში შესაბამისი მნიშვნელობები ჩაიწერება მონაცემთა ტრიგერში, შემდეგ მართვის ტრიგერის დახმარებით გამოსასვლელზე (გამომყვანზე) დაყენდება პოტენციალის საჭირო დონე. მართვის ტრიგერი იძლევა ნებართვას სალტეზე მონაცემების გამოყვანისათვის. თანამედროვე მიკროკონტროლერებში მონაცემებისა და მართვის ტრიგერებთან ინდივიდუალური შეღწევა უზრუნველყოფილია, მისამართების სალტის გამოყენებით.

გარე გამომყვანი აგრეთვე შეიძლება გამოყენებულ იქნეს შეწყვეტის მოთხოვნის მიწოდებისათვის. ეს ჩვეულებრივ რეალიზდება მაშინ, როცა გამომყვანი მუშაობს შეტანის რეჟიმში.

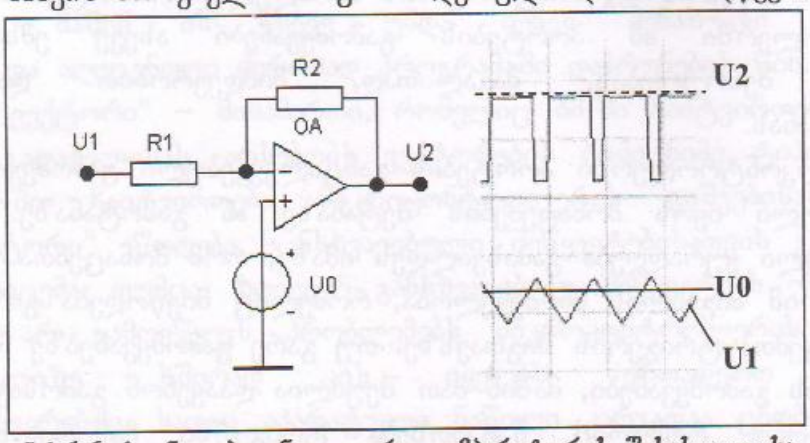
განცალკევებული ციფრული სისტემების კავშირის უფრო მეტად გავრცელებული სახეა მონაცემების მიმდევრობითი გაცვლა. ამ შემთხვევაში მიმღებსა და მონაცემთა წყაროს შორის სინქრონიზებულად ხდება მონაცემების ბაიტების გადაცემა ერთი სადენის საშუალებით, თითოეული ბიტით. მონაცემების მიმდევრობითი გადაცემის უპირატესობაა ის, რომ იგი არ მოითხოვს კავშირის ხაზების დიდ რაოდენობას.

არსებობს მრავალი სტანდარტული, მიმდევრობით მონაცემთა გადაცემის პროტოკოლი. ზოგ მიკროკონტროლერში ეს პროტოკოლები რეალიზდება შიგა სქემებით, რომლებიც განთავსებულია კრისტალზე, რაც იძლევა სხვადასხვა მოწყობილობის დამუშავების გამარტივების საშუალებას.

ანალოგური კომპარატორი

ხშირად მიკროკონტროლერებში ჩაშენდება ძაბვის ანალოგური კომპარატორი (**OA**). კომპარატორი წარმოადგენს სქემას, რომელიც ორ ძაბვას ადარებს. ჩვეულებრივ, ერთ სიგნალს ეწოდება შესავალი (**U1**), მეორეს - საყრდენი (**U0**). გამოსასვლელზე (**U2**) ყენდება "1", თუ შესავალი ძაბვა მეტია, ვიდრე საყრდენი (ნახაზი 1.6). ასეთი მოწყობილობის გამოყენება მოსახერხებელია

მიკროკონტროლერების თერმოსტატში, სადაც აუცილებელია შემოწმდეს ტემპერატურის შესაბამისი გამზომი სიდიდე, რომელიც მიეწოდება კომპარატორში ანუ შესავალი ძაბვის მნიშვნელობის განსაზღვრული დონის მიღწევა.



ნახ.1.6. სიგნალები ანალოგური კომპარატორის შესასვლელსა და გამოსასვლელზე

1.3. მიკროკონტროლერების გამართვისა და შემუშავების საშუალებები

მიკროკონტროლერების პროგრამების გამართვისათვის ყველაზე ეფექტურია სპეციალიზებული, პროფესიული ინსტრუმენტული გამართვის საშუალებების გამოყენება, რომლებსაც შეიძლება მივაკუთვნოთ:

- შიგასქემური ემულატორები (**შსე**) – პროგრამულ-აპარატული საშუალება, რომელსაც შეუძლია ჩაანაცვლოს ემულირებადი პროცესორი რეალურ მოწყობილობაში;
- პროგრამული სიმულატორები – პროგრამული საშუალება, რომელსაც შეუძლია მიკროკონტროლერის და მისი მეხსიერების მუშაობის იმიტაცია;
- გამართვის მონიტორები – სპეციალური პროგრამა, რომელიც იტვირთება გასამართი სისტემის მეხსიერებაში;
- შემფასებელი პლატები (**Evaluation Boards**) – თავისებური კონსტრუქტორები, გამოყენებითი სისტემების მაკეტირებისათვის;
- მუდმივი დამხსომებელი მოწყობილობის (**მდმ**) ემულატორი – პროგრამულ-აპარატული საშუალება, რომლითაც შესაძლებელია **მდმ** გასამართი მოწყობილობის **ოდმ**-ით ჩანაცვლება, რომელშიც შეიძლება ჩაიტვირთოს პროგრამა კომპიუტერიდან სტანდარტული კავშირის ერთ-ერთი არხით.

ამას გარდა, არსებობს კომბინირებული ინსტრუმენტული გამართვის საშუალებები.

სიმულატორები

როგორც წესი, სიმულატორი შედგება გამართვის მოწყობილობისგან, ცენტრალური პროცესორის (**ცპ**) მოდელისა და მეხსიერებისგან. უფრო სრულყოფილ სიმულატორი შეიცავს აგრეთვე ჩაშენებულ პერიფერიულ მოწყობილობებს, მოდელებს (ტაიმერებს, პორტებს, ანალოგურ-ციფრულ გარდამქმნელებს (**აცგ**) და შეწყვეტის სისტემებს).

სიმულატორს უნდა შეეძლოს პროგრამების ფაილების ჩატვირთვა ყველა პოპულარულ ფორმატში, მაქსიმალურად სრულად ასახოს ინფორმაცია მასიმულირებელი მიკროკონტროლერის რესურსების მდგომარეობის შესახებ, აგრეთვე წარმოადგინოს ჩატვირთული პროგრამის შესრულების სიმულაციის შესაძლებლობა სხვადასხვა რეჟიმში. გამართვის პროცესში მოდელი ასრულებს პროგრამას და კომპიუტერის მონიტორის ეკრანზე გამოისახება მოდელის მიმდინარე მდგომარეობა.

სიმულატორში პროგრამის ჩატვირთვისას მომხმარებელს შეუძლია გაუშვას იგი ბივჯურად ან უწყვეტ რეჟიმში, მონიშნოს პირობითი და უპირობო შეჩერების წერტილები, აკონტროლოს და მოდიფიცირება გაუკეთოს მასიმულირებელი მიკროკონტროლერის მეხსიერების უჯრედებისა და რეგისტრების შიგთავსს. სიმულატორი ახდენს პროგრამის შესრულების ლოგიკის სწრაფ შემოწმებას და არითმეტიკული ოპერაციების შესრულების გამართვას.

გამოყენებული გამართვის მოწყობილობის კლასის მიხედვით სიმულატორების ზოგი მოდელი ხელს უწყობს პროგრამების გამართვას უფრო მაღალ დონეზე.

სიმულატორი შეიძლება შეიცავდეს დამატებით პროგრამულ საშუალებებს, მაგალითად, გარე ინტერფეისს. ასეთი ინტერფეისის არსებობა საშუალებას იძლევა შეიქმნას და მოქნილად იქნეს გამოყენებული მიკროკონტროლერის გარე მოდელი, რომელიც ფუნქციონირებს და ზემოქმედებს გასამართ პროგრამაზე მოცემული ალგორითმით.

რეალურ სისტემებში მიკროკონტროლერი ჩვეულებრივ „დაკავებულია“ მასზე მიერთებული მოწყობილობის (გადამწოდის) მონაცემების წაკითხვით, მისი დამუშავებით და მართვის სიგნალების გადაცემით შემსრულებელ მოწყობილობაზე (აქტუატორებზე). იმისათვის, რომ მარტივ სიმულატორზე მოხდეს გადამწოდის მუშაობის მოდელირება, საჭიროა ხელით ვცვალოთ იმ პერიფერიული მოწყობილობის მოდელის მიმდინარე მდგომარეობა, რომელთანაც რეალურ სისტემაში მიერთებულია გადამწოდი. მაგრამ არსებობს ბევრი თანამედროვე პროგრამული სიმულატორი, რომლებშიც გარე პირობებისა და სიტუაციის იმიტაციისათვის გამოიყენება შესავალი ზემოქმედების სპეციალური ფაილი. ეს ფაილი იძლევა შესავალი სიგნალების თანამიმდევრობას, რომელიც მოდელირებად მოწყობილობას მიეწოდება.

ზოგი მოდელის სიმულატორებში გარე სიგნალების იმიტაციის ეს პრობლემა გადაწყვეტილია შემდეგნაირად: სიმულატორს აქვს ისეთი ჩაშენებული საშუალება, რომელიც მიკროკონტროლერთან მიერთებული გარე მოწყობილობების მოდელების შესაქმნელად უზრუნველყოფს ინფორმაციის გრაფიკულად გამოსახვას. პროგრამული სიმულატორების მნიშვნელოვანი თავისებურებაა ის, რომ მათში ჩატვირთული პროგრამების შესრულება ხდება დროის შუალედში, რომელიც რეალური დროისგან განსხვავდება. მაგრამ დაბალი ფასი და გაწყობის შესაძლებლობა, შესაქმნელი მოწყობილობის მაკეტის არარსებობის შემთხვევაშიც, პროგრამულ სიმულატორებს გადააქცევს მიმზიდველ გასაწყობ საშუალებად. აუცილებელია აგრეთვე აღინიშნოს, რომ არსებობს შეცდომების მთელი კლასი, რომლებიც შეიძლება აღმოჩენილ იქნეს მხოლოდ სიმულატორების დახმარებით.

1.3.1. პროგრამების შემუშავების ინტეგრირებული გარემო

მიკროკონტროლერის ბაზაზე პროგრამული და აპარატული უზრუნველყოფის ერთიანობის იდეა ძალიან მნიშვნელოვანია. პროგრამული უზრუნველყოფის დამუშავების ინსტრუმენტული საშუალებების გაერთიანება, აპარატული უზრუნველყოფის დამუშავების ინსტრუმენტულ საშუალებებთან, მნიშვნელოვან უპირატესობას ანიჭებს ასეთი სახის მოწყობილობებს.

სხვადასხვა სახის ე.წ. *შემუშავების ინტეგრირებული გარემო* არსებითად აადვილებს და აჩქარებს დამუშავების და მიკროკონტროლერული სისტემების გამართვის პროცესს. ისინი შეიცავენ ტექსტურ რედაქტორს საწყისი ტექსტების დაწერისათვის, ტრანსლატორებს - ასემბლერიდან და **C** ენიდან, ლინკერს, გამართვის მოწყობილობას, საცნობარო ინფორმაციას მიკროკონტროლერების შესახებ და სხვა აუცილებელ საშუალებებს. ტრანსლატორების, ლინკერისა და სხვა კომპონენტების დაყენება ხდება არა კომანდების სტრიქონში გასაღების მითითების მეთოდით, არამედ დიალოგური ფანჯრების სახით, სადაც საჭიროა მხოლოდ აღნიშვნების დასმა საჭირო ადგილებზე. პროგრამების შემომავალი ტექსტების გარდაქმნა მანქანური კოდების ფაილში ხდება ერთი კლავიშის დაჭერით.

პროგრამების შემუშავების ინტეგრირებული გარემოს შექმნამ უფრო მეტად აამაღლა მიკროკონტროლერებისათვის პროგრამების შექმნის ეფექტურობა, რამაც დამმუშავებელს საშუალება მისცა კონცენტრაცია მოეხდინა გადასაწყვეტი დავალების არსზე და ყურადღება არ გადაეტანა მისი რეალიზაციის კონკრეტულ დეტალებზე. პროგრამების შემუშავების ინტეგრირებული პაკეტები მუშავდება რამდენიმე ფირმის მიერ, სხვადასხვა მწარმოებლის პაკეტები ფუნქციურად მსგავსია, მაგრამ განსხვავდება წარმოდგენილი სერვისული შესაძლებლობებით, მუშაობის მოხერხებულობით და გენერირებული მანქანური კოდის ხარისხით.

პროგრამის დაწერა საწყის ეტაპზე, ტრადიციული მიდგომის მიხედვით, ხდება შემდეგი სახით: შესავალი ტექსტი აიკრიფება რომელიმე ტექსტური რედაქტორის დახმარებით. აკრეფის დასრულებისას ტექსტური რედაქტორით მუშაობა შეწყდება და გაიშვება კროსს-კომპილატორი. როგორც წესი, ახალი პროგრამა შეიცავს სინტაქსურ შეცდომებს და კომპილატორი აცნობებს მათ შესახებ ოპერატორის კონსოლზე. შემდეგ ხელახლა გაიშვება ტექსტური რედაქტორი და ოპერატორი იპოვის და შეასწორებს გამოვლენილ შეცდომებს. შეტყობინება მათი ხასიათის შესახებ, რომელიც გამოვლენილ იქნა კომპილატორით უკვე აღარ ჩანს, ვინაიდან ეკრანი დაკავებულია ტექსტური რედაქტორით.

ეს ციკლი შეიძლება ბევრჯერ განმეორდეს. თუ პროგრამა შედარებით რთულია და იკრიბება სხვადასხვა ნაწილისგან და ექვემდებარება რედაქტირებას ან მოდერნიზაციას, მაშინ ამ საწყისმა ეტაპმაც პროგრამისტისგან შეიძლება მოითხოვოს დიდი დრო და ენერგია.

როგორც აღინიშნა, რუტინული სამუშაოს თავიდან აცილებას და პროგრამისტის მწარმოებლურობის ამაღლებას ხელს უწყობს *პროგრამების შემუშავების ინტეგრირებული გარემო* (გარსი) - **IED** (integrated development environment).

მაღალი დონის **IED** აერთიანებს არსებული გამართვის საშუალებებს (შიგასქემურ ემულატორს, პროგრამულ სიმულატორს და პროგრამატორს) და

უზრუნველყოფს პროგრამისტის მუშაობას პროგრამების ტექსტებთან დიალოგური ფანჯრის სტილში.

ინტეგრირებული გარემო საშუალებას იძლევა გამოყენებულ იქნეს:

- ჩაშენებული მრავალფაილიანი ტექსტური რედაქტორი, რომელიც ორიენტირებულია პროგრამის საწყის ტექსტებთან სამუშაოდ;
- ერთდროული დაკვირვება მრავალფანჯრიან რეჟიმში კომპილაციის დროს გამოვლენილი შეცდომების დიაგნოსტიკაზე და წვდომა პროგრამის რედაქტირებისათვის საწყის ტექსტზე;
- პარალელური სამუშაოს წარმართვა რამდენიმე პროექტზე. პროექტების მენეჯერი ახდენს ნებისმიერი პროექტის გამოყენებას შაბლონის სახით ახალი პროექტის შექმნისათვის. გამოყენებული კომპილატორების ოპციები და პროექტის საწყისი ფაილების ჩამონათვალი დაყენდება დიალოგურ მენიუში და ინახება პროექტის ჩარჩოებში, რაც გამორიცხავს მუშაობის აუცილებლობას მოუხერხებელ ***.bat**-ფაილებთან;
- განახორციელოს რედაქტირებული მოდულების გადაკომპილირება;
- ჩატვირთოს გასაწყობი პროგრამა არსებულ გაწყობის საშუალებებში და მათთან იმუშაოს გარსიდან გამოსვლის გარეშე;
- მიუერთოს გარსს ნებისმიერი პროგრამული საშუალება.

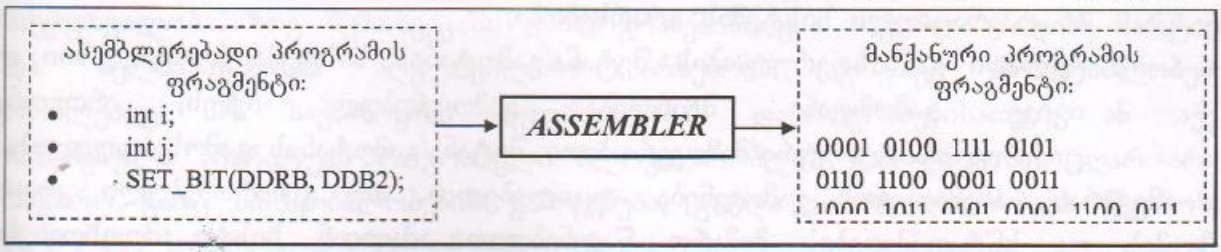
ბოლო დროს **IDE** ფუნქცია გადადის მაღალი დონის ემულატორებისა და სიმულატორების გამართვის მოწყობილობების პროგრამულ ინტერფეისებში. ასეთი ახალი ფუნქციური შესაძლებლობები, "მეგობრულ" ინტერფეისთან ერთად, არსებითად აჩქარებს პროგრამისტის მუშაობას.

ამგვარად, გამართვის ინსტრუმენტული საშუალებების არჩევისას მიზანშეწონილია შემდეგი მაჩვენებლების კომპლექსი: ადეკვატური მიკროკონტროლერების ჩამონათვალი, შეზღუდვები მაემულირებელი/მასიმულირებელი მიკროკონტროლერების რესურსებზე, სიმბოლური გამართვის შესაძლებლობა, ადეკვატური კომპილატორების ჩამონათვალი და სერვისული შესაძლებლობები.

1.3.2. ასემბლერი და მაღალი დონის პროგრამირების ენები

მიკროკონტროლერების ბაზაზე რომელიმე სისტემის დამუშავების დაწყებამდე ძალიან მნიშვნელოვანია გავეცნოთ პროგრამირების საფუძვლებს ასემბლერის ენაზე. მიკროკონტროლერების გამოყენებებისას საჭიროა პროგრამირების ამ მეთოდის კარგად შესწავლა და გაგება, თუ როგორ ნაბიჯ-ნაბიჯ სრულდება პროგრამა და, ამის გამო, რა ხდება მოწყობილობაში.

1.7 ნახაზზე ნაჩვენებია მანქანური პროგრამის მიღება ასემბლერების შედეგად.



ნახ. 1.7. ასემბლერების ფუნქციის წარმოდგენა

ასემბლერის ენის შესწავლის, პროგრამის დაწერისა და გასწორების რამდენიმე ხერხი არსებობს:

პირველი – პროცესორით ბრძანებების შესრულების ვიზუალიზაციის პროცედურის გამოყენება.

მეორე – სტრუქტურული პროგრამირების მეთოდების გამოყენება, რათა პროგრამა გახდეს უფრო მარტივი წასაკითხად და გასაგებად.

ბრძანებების შესრულების ვიზუალიზაცია ყველაზე კარგად ხორციელდება, თუ გამოვიყენებთ მიკროკონტროლერის სტრუქტურულ სქემას, რომელზეც აღინიშნება მონაცემების გავლა ყოველი ბრძანების შესრულებისას. შედეგად შესაძლებელი ხდება ბრძანებების შესრულების პროცესის კარგი ვიზუალური წარმოდგენა.

მაღალი დონის პროგრამირების ენები

მიკროკონტროლერის პროგრამირებისათვის შეიძლება გამოყენებულ იქნეს მაღალი დონის განსხვავებული ენები. ტერმინი „მაღალი დონის ენა“ გამოიყენება იმ ენების აღნიშვნისათვის, რომლებიც გამოიყენება ადვილად წასაკითხი პროგრამების დასაწერად და კონვერტირდება (კომპილირდება) ასემბლერის ენაში, შემდეგ გარდაიქმნება ობიექტურ კოდებად (ბიტებად და ბაიტებად) მიკროკონტროლერის მიერ მათი შესრულებისას.

ჩამოვთვალოთ მაღალი დონის ენების ძირითადი მახასიათებლები:

- ჩაშენებული ფუნქციების არსებობა (მაგალითად, კონსოლით შეტანა/გამოტანა) ბიბლიოთეკებთან ჩართვის შესაძლებლობით;
- მონაცემების განსხვავებულფორმიანი ტიპები (8-, 16-, 32-ბიტიანი და მცოცავი წერტილით);
- არითმეტიკული ოპერაციების შესრულება სტეკის გამოყენებით;
- ლოკალური და გლობალური ცვლადების, მაჩვენებლებისა და მონაცემთა სტრუქტურების გამოყენება;
- მეხსიერების განაწილება;
- აპარატულ რესურსებში შეღწევა;
- სიმბოლური ინფორმაციის არსებობა სიმულატორისა და ემულატორისათვის.

ამ მახასიათებლების რეალიზაცია შეიძლება იყოს პრობლემატური ჩაშენებული მიკროკონტროლერებისათვის, რომლებსაც შემდეგი თავისებურებები ახასიათებს:

- **ROM** პროგრამების მეხსიერების და **RAM** მონაცემთა მეხსიერების შეზღუდული მოცულობა;
- **BIOS**-ის (basic input/output system) ბაზური შეტანა/გამოტანის სისტემის ან ოპერაციული სისტემის არარსებობა;
- ცვლადი განსაზღვრულობის შეტანა/გამოტანის სისტემა (როცა ერთი და იგივე გამომყვანი შეიძლება გამოყენებულ იქნეს, როგორც ციფრული/ანალოგური/მიმდევრობითი შეტანა/გამოტანის გამოსასვლელი).

ამგვარად, ასემბლერის გამოყენება აუცილებელია, თუ გენერირებული კოდის ზომის და სწრაფქმედების მიმართ წაყენებულია ძალიან ხისტი მოთხოვნები. დღეისათვის ასეთი შემთხვევები ნაკლებად გვხვდება, რადგან პრაქტიკულად ყოველთვის შეიძლება ავიღოთ უფრო „სწრაფი“ მიკროკონტროლერი მეხსიერების

დიდი მოცულობით. გარდა ამისა, თანამედროვე "კროს" საშუალებების პაკეტები საშუალებას იძლევა ადვილად დაიწეროს შერეული პროგრამები, სადაც მოდულების ნაწილები დაწერილია C ენაზე, ხოლო უფრო კრიტიკული სწრაფქმედების მიმართ პროგრამის ნაწილები – ასემბლერზე. C კომპილატორები ახდენს აგრეთვე პროგრამის საწყის ტექსტში ასემბლერზე ინსტრუქციების შეტანას.

მიკროკონტროლერისათვის პროგრამული უზრუნველყოფის დამუშავების რამდენიმე წესი არსებობს, რომელთა შესრულება საჭიროა იმისათვის, რომ გამოსაყენებელი რესურსების მოცულობამ არ გადააჭარბოს ხელმისაწვდომ ზღვარს:

1. მხოლოდ ერთი სახის ინტერფეისის გამოყენება აპარატულ საშუალებებთან (გარე მოწყობილობებთან). განსხვავებული ინტერფეისების გამოყენება ქმნის პრობლემებს იმ შემთხვევაში, თუ მოთხოვნა იქნება სხვა ტიპის გარე მოწყობილობების ჩართვაზე;
2. გლობალური ცვლადების იდენტიფიცირება, რომლებიც სპეციფიკურია ქვეპროგრამებისათვის და მათი არგამოყენება სხვა რომელიმე კოდის შიგნით.
3. შესაძლებელია ყველგან ლოკალური ცვლადების გამოყენება (ეს შესაძლებებელია მხოლოდ მაღალი დონის ენების გამოყენების დროს).
4. თუ სავარაუდოა დროებითი გამოყენების ცვლადების არსებობა, მაშინ პროგრამამ უნდა უზრუნველყოს მათი უნიკალური გამოყენება.

ამ წესებით პროგრამების დამუშავებისას თავს დავაღწევთ ისეთ პრობლემებს, რომლებიც დაკავშირებულია პროგრამაში ძნელად გამოსავლენ არამდგრად შეცდომებთან.

შექმნილი პროგრამის შესაბამისად მოწყობილობებისა და მიკროკონტროლერების დაპროგრამება, ჩვეულებრივ, მიმდინარეობს პროგრამატორის საშუალებით. პროგრამატორის რეგისტრში ჩაიტვირთება მნიშვნელობა, რომელიც უნდა განლაგდეს მიკროკონტროლერის განსაზღვრულ მისამართზე, შემდეგ ჩაირთვება სქემა, რომელიც გადააგზავნის ამ რეგისტრის შიგთავს მითითებულ მისამართზე. შერჩეული მეხსიერების უჯრედის დაპროგრამების პროცესის დასრულების შემდეგ ხორციელდება ვერიფიკაცია ანუ მოწმდება ჩაწერილი მნიშვნელობის მართებულობა. მთელი მოწყობილობის დაპროგრამება შეიძლება გაგრძელდეს რამდენიმე წამიდან რამდენიმე წუთამდე, მეხსიერების მოცულობისა და პროგრამის ალგორითმზე დამოკიდებულებით.

არსებობს სხვადასხვა დონის დაპროგრამების მოწყობილობები და აღჭურვილობა. ზოგ შემთხვევაში დაპროგრამებისათვის საჭიროა ძალიან მარტივი და ხელმისაწვდომი აღჭურვილობა. არსებობს აგრეთვე მოწყობილობები, რომლებიც არ საჭიროებს აღჭურვილობას დაპროგრამებისათვის ან აქვს ჩაშენებული აპარატულ-პროგრამული ბლოკი, რომლის გამოყენებისას აღარ არის საჭირო გარე მოწყობილობების გამოყენება, გარდა გაზრდილი ძაბვის წყაროსი, რომელიც საჭიროა დაპროგრამებისათვის.

ძალიან მნიშვნელოვანია ასპექტი, რომელიც ეხება მოწყობილობების დაპროგრამებას და მდგომარეობს მოწყობილობის სისტემაში დაპროგრამების

შესაძლებლობის დადგენაში. ამ პროცედურას ეწოდება შიგასისტემური დაპროგრამება **ISP** (In-Sistem Programming). თუ მიკროკონტროლერი დაუშვებს ასეთი დაპროგრამების შესაძლებლობას, მაშინ ეს ნიშნავს, რომ ის უნდა დამონტაჟდეს პლატაზე ცარიელი პროგრამების მეხსიერებით, რომელიც შემდეგ შეიძლება დაპროგრამებულ იქნეს რაიმე ზეგავლენის გარეშე სქემის დანარჩენ კომპონენტებზე. ეს მნიშვნელოვანი გარემოებაა მიკროკონტროლერის არჩევისათვის. **ISP**-ს გამოყენება გამორიცხავს სპეციალური პროგრამატორის შეძენის აუცილებლობას, შესაძლებლობას იძლევა განახლდეს მოწყობილობის პროგრამული უზრუნველყოფა პლატაზე განლაგებული აპარატული საშუალებების შეცვლის გარეშე და ხელს შეუწყობს მწარმოებლებს შექმნან მზა ნაკეთობების მარაგი, რომლებიც ადვილად მოდიფიცირდება მიმდინარე შეკვეთების მიხედვით.

პროგრამების მეხსიერების უსაფრთხოება

მრავალი პრაქტიკული დანართების გამოყენებებისათვის სასურველია დაცულ იქნეს პროგრამული კოდი, რომელიც ჩაწერილია მიკროკონტროლერში. იმისათვის, რომ უზრუნველყოფილ იქნეს ასეთი შესაძლებლობა ბევრი მიკროკონტროლერი შეიცავს სპეციალურ საშუალებებს, რათა თავიდან აიცილონ შენახული პროგრამების წაკითხვა. ხშირად ასეთი შესაძლებლობა რეალიზდება პროგრამირების პროცესში კონფიგურაციული ბიტის განსაზღვრული მნიშვნელობის დაყენების გზით. ჩვეულებრივ, ამ ბიტის მნიშვნელობა შეიძლება შეიცვალოს მხოლოდ მიკროკონტროლერის მეხსიერების შიგთავსის კვლავპროგრამირების პროცესში, მაგალითად, **EPROM**-ის შიგთავსის ულტრაიისფერული წაშლისას.

პროგრამული კოდის წაკითხვის შესაძლებლობას ჩაშენებული დაცვა ვერ აგვარიდებს. ასეთი წაკითხვა შეიძლება მოხდეს ლაბორატორიებში, სადაც ხდება მიკროსქემების მტყუნების მიზეზების ანალიზი მოკლე დროში. იმისათვის, რომ გართულდეს და, შესაბამისად, წაკითხვის ოპერაცია იყოს ნაკლებ ეფექტური, ზოგიერთი კომპანია ახდენს პროგრამის ჩაწერას დაშიფვრით, ბრძანებების შერევის გზით და რთავს სპეციალურ აპარატულ ბლოკებს, რომლებიც შერეულ მონაცემებს გარდაქმნის პროცესორის ბრძანებების ნაკადში. დაცვის ბიტის დაყენება ვერ მოგვცემს მიკროკონტროლერში ჩატვირთული პროგრამული კოდის აბსოლუტური დაცვის გარანტიას.

1.4. მიკროკონტროლერის დაპროგრამების საშუალებები

ტერმინში "პერსონალური კომპიუტერი" იგულისხმება **IBM PC** არქიტექტურის ტიპის კომპიუტერი. ყველა ამ კომპიუტერს ერთი დიდი ნაკლი აქვს. ისინი არათავსებადია და ვერ პოვა მასობრივი გავრცელება. **IBM PC** არ იყო არც ყველაზე მწარმოებლური პერსონალური კომპიუტერი, არც ყველაზე იაფი, მაგრამ ჰქონდა დიდი უპირატესობა - მისი არქიტექტურა და მუშაობის პრინციპი იყო ყველასათვის ღია. ნებისმიერ მსურველ კომპანიას შეეძლო მიეღო არქიტექტურის დოკუმენტაცია და მის საფუძველზე ახალი აპარატული თუ პროგრამული პროდუქტი შეექმნა. ამის გამო, **IBM PC** არქიტექტურამ დაიწყო სწრაფი

განვითარება და დღესაც რჩება ლიდერად სამომხმარებლო პერსონალური კომპიუტერების ასპარესზე.

წლების განმავლობაში დამუშავებულ იქნა საკმაოდ ბევრი სხვადასხვანაირი ინტერფეისი, რომლის საშუალებით დაკავშირება ხდებოდა პერსონალური კომპიუტერის სისტემებსა და გარე, პერიფერიულ მოწყობილობებს შორის. ინტერფეისები ვითარდებოდა თვით პერსონალური კომპიუტერის პლატფორმის განვითარების პარალელურად. შეიქმნა ინტერფეისები, რომელთა იმპლემენტაცია რთულია და შედარებით ძვირი, მაგრამ უზრუნველყოფს გადასაცემი ინფორმაციის სწრაფ და საიმედო გადაცემას. აგრეთვე შეიქმნა ინტერფეისები, რომლებიც შედარებით ნელი, მაგრამ სამაგიეროდ მარტივი და იაფია. სხვადასხვა მოწყობილობას თავისი მოთხოვნილება ჰქონდა კომპიუტერთან კავშირზე და შედეგად დღეს არსებობს (და აქტიურად გამოიყენება) არაერთი ინტერფეისი, რომელთა შორის ერთ-ერთი ყველზე პერსპექტიულია **USB** (universal serial bus).

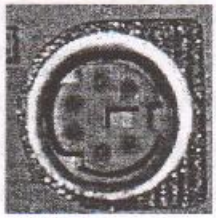
USB გამოყენებისას მის ბაზაზე მოწყობილობების აგება საკმაოდ რთული ტექნიკური ამოცანაა. დღეს არსებობს მრავალი მზა ტექნიკური გადაწყვეტილება, რომლის საშუალებით შესაძლებელია ამ ინტერფეისის გამოყენება ან სწრაფი ინტეგრაცია ახალ, ან უკვე არსებულ ელექტრონულ მოწყობილობებში, რომლებსაც სჭირდება პერსონალურ კომპიუტერთან იაფი, სწრაფი და საიმედო კავშირი.

დიდი ხნის განმავლობაში ისეთი პერიფერიული მოწყობილობების მისაერთებლად, როგორიცაა პროგრამატორი, ძირითადად **COM (RS-232)** ან **LPT** პორტები გამოიყენებოდა. იმის გამო, რომ ეს ორი ინტერფეისი ნელ-ნელა გადავიდა "მოძველებულის" სტატუსში, დღეს საჭირო ხდება **USB** ინტერფეისის იმპლემენტაცია მიკროკონტროლერების პროგრამატორების კომპიუტერებთან დასაკავშირებლად.

1.4.1. კომპიუტერის პორტების აღწერა და მათი გამოყენების სპეციფიკა

კომპიუტერულ ტექნიკაში პორტი ფიზიკურად დამაკავშირებელი ინტერფეისია ორ კომპიუტერს ან კომპიუტერსა და გარე მოწყობილობას შორის. პორტი, როგორც ფიზიკური ინტერფეისი, არის სხვადასხვა ფორმის და დანიშნულების. მაგალითად, "დედალი", "მამალი", მრგვალი, ოთხკუთხედი და ა.შ. ყველაზე გავრცელებული პორტების (როგორიცაა, მაგალითად, კომპიუტერის კლავიატურის პორტი) ტექნიკური პარამეტრები მკაცრად სტანდარტიზებულია.

ნახ. 1.8

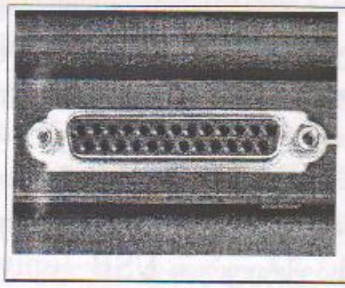


ნახ. 1.9

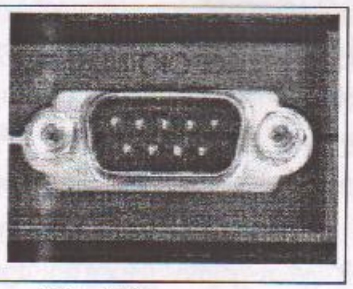
1.8 ნახაზზე გამოსახულია კომპიუტერის კლავიატურის **PS/2** ტიპის პორტის "მამალი", 1.9 ნახაზზე კი - ამავე პორტის "დედალი".

მონაცემთა გადაცემის მექანიზმის მიხედვით, აპარატული პორტები პრაქტიკულად ყველა შემთხვევაში შეიძლება ორ ჯგუფად დაიყოს:

- მიმდევრობითი პორტები, სადაც დროის ერთ მონაკვეთში გადაეცემა თითო ბიტი, ერთი წყვილი სადენის მეშვეობით;
- პარალელური პორტები, სადაც დროის ერთ მონაკვეთში გადაეცემა რამდენიმე ბიტი, მრავალი სადენის მეშვეობით.



ნახ. 1.10



ნახ. 1.11

1.10 ნახაზზე ნაჩვენებია **IEEE 1284 (LPT)** პარალელური პორტი, 1.11 ნახაზზე კი - **COM (RS-232)** პორტის კონექტორი.

ერთი ერთეული ფიზიკური პორტის საშუალებით შესაძლებელია რამდენიმე მოწყობილობის ერთ კომპიუტერთან დაკავშირება.

დღეს კომპიუტერსა და გარე მოწყობილობებს შორის კავშირის დასამყარებლად ყველაზე ხშირად გამოიყენება პორტების ორი სახეობა: მიმდევრობითი **COM (RS-232)** და შედარებით ახალი **USB** ინტერფეისები.

RS-232 არის **DTE** (Data terminal equipment) მონაცემთა ტერმინალი მოწყობილობა და **DCE** (Data Circuit-terminating Equipment) მონაცემთა წრედის დამაბოლოებელი მოწყობილობა, რომლებიც შეესაბამება მონაცემთა მიმდევრობით ორობითი გადაცემის ტელეკომუნიკაციურ სტანდარტებს. მოწყობილობების **DTE/DCE** ტიპებად კლასიფიკაცია კომპანია **IBM**-მა შემოიღო და, როგორც წესი, **DTE**-თი კომპიუტერი აღინიშნებოდა, ხოლო **DCE**-თი - მოდემი.

RS-232 სტანდარტი განსაზღვრავს ძაბვის დონეებს, რომლებიც შეესაბამება ლოგიკურ "ერთიანს" და "ნულს". კორექტული სიგნალების ძაბვა უნდა მერყეობდეს (დადებით ან უარყოფით) 3-დან 15 ვოლტამდე ინტერვალში. სიგნალები, რომელთა ძაბვა ნულთან ახლოსაა არ წარმოადგენს კორექტულ **RS-232** სიგნალებს. ლოგიკური ერთიანი განისაზღვრება უარყოფითი ძაბვით. სიგნალის ამ მდგომარეობას markingi ეწოდება და მისი ფუნქციური მნიშვნელობაა **OFF** (გამორთულია). ლოგიკური ნული დადებითი ძაბვით განისაზღვრება. სიგნალის ამ მდგომარეობას sfeisingi ეწოდება და მისი ფუნქციური მნიშვნელობაა **ON** (ჩართულია). სტანდარტი განსაზღვრავს ღია წრედის მაქსიმალურ ძაბვას 25V ფარგლებში. უმეტესად გამოიყენება ±5V,±10V,±12V და ±15V ძაბვის სიგნალები, რაც დამოკიდებულია კონკრეტული მოწყობილობის კვების წყაროზე. წრედს, რომელიც იმპლემენტაციას უკეთებს **RS-232** თავსებად ინტერფეისს უნდა შეეძლოს განუსაზღვრელი პერიოდის განმავლობაში გაუძლოს მოკლე ჩართვას მიწასა და 25 V ნებისმიერ ძაბვას შორის.

იმის გამო, რომ მონაცემთა გადაცემის ძაბვის დონეები უფრო მაღალია, ვიდრე ლოგიკური სიგნალების ძაბვის დონეები, რომლებიც გამოიყენება ინტეგრალურ სქემებში, საჭირო ხდება სპეციალური მაინვერტირებელი დამატებითი სქემები, რომლებიც ძაბვას გარდაქმნიან ლოგიკის ძაბვის დონეებიდან - გადაცემის ძაბვის დონეებში და პირიქით. ეს მაინვერტირებელი სქემები აგრეთვე იცავს ძირითად სქემას **RS-232** ინტერფეისზე შესაძლო მოკლე ჩართვებისგან ან მაღალი ძაბვის იმპულსებისგან.

ქვემოთ ცხრილში მოცემულია **RS-232** სიგნალებისა და პორტის შესაბამისი კონტაქტების ჩამონათვალი:

ცხრილი 1

სიგნალი		DB-25	DE-9 (TIA-574)
Common Ground		7	5
Transmitted Data	TD	2	3
Received Data	RD	3	2
Data Terminal Ready	DTR	20	4
Data Set Ready	DSR	6	6
Request To Send	RTS	4	7
Clear To Send	CTS	5	8
Carrier Detect	CD	8	1
Ring Indicator	RI	22	9

სიგნალების აღწერა:

- Transmitted Data (TD).

მონაცემები გაიგზავნება DTE-დან DCE-სკენ.

- Received Data (RD)

მონაცემები გაიგზავნება DCE-დან DTE-სკენ.

- Request To Send (RTS)

დაყენდება (სიგნალი იღებს მნიშვნელობას 0) DTE-ს მიერ, რათა მოამზადოს DCE მონაცემთა მიღებისათვის.

- Clear To Send (CTS)

დაყენდება DCE-ს მიერ, რათა დაამტკიცოს RTS და ნება მისცეს DTE-ს დაიწყოს მონაცემების გადაცემა.

- Data Terminal Ready (DTR)

დაყენდება DTE-ს მიერ იმის საჩვენებლად, რომ ის მზადაა კავშირის დასამყარებლად. იმ შემთხვევაში, თუ სიგნალი შეწყდა, DCE-მ უნდა გაწყვიტოს დამყარებული კავშირი.

- Data Set Ready (DSR)

დაყენდება DCE-ს მიერ აქტიური კავშირის მისათითებლად. იმ შემთხვევაში, თუ DCE არის ნულ-მოდემური კაბელი ან სხვა მსგავსი მოწყობილობა, სიგნალი მუდმივად 0-ის ტოლი უნდა იყოს (ამის მიღწევა შეიძლება კონტაქტის დაკავშირებით სხვა კონტაქტთან ჯამპერის საშუალებით).

- Carrier Detect (CD)

გამოიყენება DCE-ს მიერ, როცა მოშორებულ მოწყობილობასთან კავშირი დამყარებულია.

- Ring Indicator (RI)

გამოიყენება DCE-ს მიერ, როდესაც ის სატელეფონო ხაზიდან (მოდემის შემთხვევაში) ზარის სიგნალს ალმოაჩენს.

მიმდევრობითი პორტები იყენებს ორდონიან (ბინარულ) სიგნალების ნაკადს, რის შედეგად მონაცემთა გადაცემის სიჩქარე ბიტ/წამებში გაუტოლდება სიგნალის ცვლილების სიჩქარეს ბოდ-ებში. ასინქრონული სტარტ/სტოპ ტიპის კომუნიკაციისას ყველაზე გავრცელებული სიჩქარეებია 300, 1200, 2400, 9600 ბოდი (baud - არის სიგნალის მკაფიოდ განსაზღრული მდგომარეობის ცვლილების სიჩქარე). ხშირ შემთხვევაში, მონაცემების კოდირება ხდება ისე, რომ სიგნალის ერთი მდგომარეობა შეესაბამება ერთ ბიტს: 1 - არის სიგნალი, 0 - არაა სიგნალი და ბოდ-სიჩქარე ემთხვევა ბიტ/წამ სიჩქარეს. ამის მიუხედავად, ბოდი არ უნდა ავურიოთ მონაცემთა გადაცემის სიჩქარესთან, რომელიც ბიტ/წამებში იზომება, რადგან გადაცემულმა სიგნალმა შეიძლება გადაიტანოს ერთზე მეტი ბიტი.

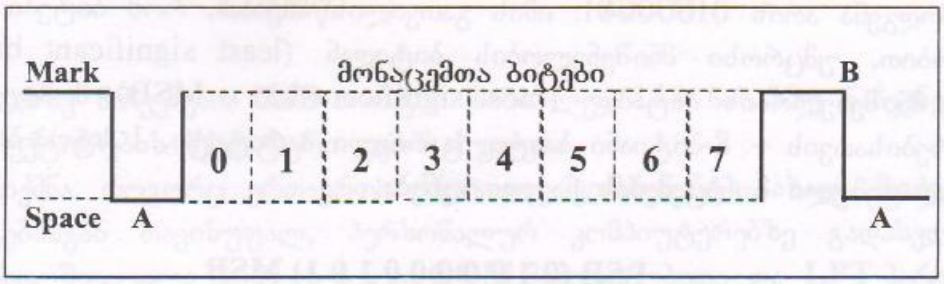
პორტისა და მოწყობილობის სიჩქარეები ერთმანეთს უნდა დაემთხვეს, თუმცა ზოგი მოწყობილობით შესაძლებელია პორტის სიჩქარის დეტექტირება. მიუხედავად იმისა, რომ ფორმალურად **RS-232** სტანდარტი ზღუდავს პორტის სიჩქარეს 20000 ბიტ/წმ სიჩქარით, თანამედროვე მიმდევრობითი პორტით შეიძლება მონაცემთა გადაცემა 115200 ბიტ/წმ სიჩქარით. ყველა სერიულ პორტს არ შეუძლია ნებისმიერი სიჩქარით მუშაობა, თანაც ზოგიერთ სპეციალურ პროტოკოლს (როგორიცაა მაგალითად, **MIDI** მუსიკალური ინსტრუმენტებთან სამუშაო პროტოკოლი) შეიძლება სჭირდებოდეს მონაცემთა გადაცემის არასტანდარტული სიჩქარე.

გადასაცემი მონაცემები შეიცავს სერვისულ ბიტებს, რომლებიც საჭიროა ფრეიმინგისათვის (სტარტ-სტოპ ბიტები, ლუწობის ბიტები და ა.შ.). ამგვარად, მონაცემთა გადაცემის ეფექტური სიჩქარე უფრო ნაკლებია, ვიდრე სიგნალების სიხშირე/სიჩქარე.

მიმდევრობითი კომუნიკაცია არის მონაცემთა გადაცემა ერთ ჯერზე თითო ბიტად. კომპიუტერი აკეთებს მონაცემთა ადრესაციას ბაიტებში ანუ რამდენიმე ბიტით ერთდროულად. ბიტი არის ლოგიკური 1 ან 0. ზოგადად, ბაიტი შედგება 8 ბიტისაგან. მიმდევრობითი პორტი გამოიყენება მონაცემთა ბაიტის კონვერტაციისთვის ბიტების ნაკადში, მათი შემდგომი გადაცემისათვის რომელიმე არხით და ბიტების ნაკადიდან ისევ ბაიტის მისაღებად. მიმდევრობითი პორტი შეიცავს ელექტრონულ მიკროსქემას **UART** (Universal Asynchronous Receiver/Transmitter) - უნივერსალური ასინქრონული მიმღებ/გადამცემი, რომელიც ასრულებს ზემოაღნიშნულ კონვერტაციას.

განვიხილოთ მონაცემთა გადაცემის და მიღების მექანიზმი. ელექტრონული ტერმინებით რომ ვთქვათ, როდესაც მიმდევრობითი პორტი გზავნის ლოგიკურ 1-ს, მაშინ უარყოფითი ძაბვა გადაეცემა გადამცემ კონტაქტზე, როდესაც გადაეცემა ლოგიკური **0** - გადამცემ კონტაქტზე დგება დადებითი ძაბვა. როდესაც

მონაცემები არ გადაეცემა, მიმდევრობითი პორტის გადამცემ კონტაქტზე აღმოჩნდება უარყოფითი ძაბვა (1) და ამბობენ, რომ პორტი არის **Mark** მდგომარეობაში. უნდა აღინიშნოს, რომ მიმდევრობითი პორტის გადამცემი კონტაქტი შეიძლება დარჩეს დადებითი ძაბვის მდგომარეობაში, რომელსაც **Space** მდგომარეობას უწოდებენ.



ნახ. 1.12. ბაიტის გადაცემის სტრუქტურა

ბაიტის გადაცემისას **UART** (მიმდევრობითი პორტი) თავდაპირველად გზავნის "სასტარტო ბიტს" (**A**) ანუ დადებით ძაბვას (0) და შემდგომ გზავნის მონაცემთა ბაიტს (ზოგადად 8 ბიტს) და გადაცემას ამთავრებს უარყოფითძაბვიანი (1) "სტოპ ბიტით" (**B**). ეს მიმდევრობა მეორდება ყოველი ბაიტისათვის. 1.12 ნახაზზე მოყვანილ დიაგრამაზე სქემატურადაა ნაჩვენები მონაცემთა ბაიტის გადაცემის პროცესი.

პროცესის ანალიზისას შეიძლება გაჩნდეს შეკითხვა, რა სიდიდისაა "ბიტის ხანგრძლივობა" ანუ რამდენი ხნის განმავლობაში ინარჩუნებს ელექტრონული სიგნალი კონკრეტულ მდგომარეობას (დადებით ან უარყოფით დონეს), რათა მოხდეს სიგნალის ბიტად აღქმა. პასუხი მარტივია - დამოკიდებულია ბოდ-სიჩქარეზე. როგორც უკვე აღვნიშნეთ, ბოდ-სიჩქარე არის ერთ წამში სიგნალის მდგომარეობის ცვლილებათა რაოდენობა. შესაბამისად, თუ ხაზი მუშაობს 9600 ბოდ სიჩქრეზე, ეს ნიშნავს, რომ სიგნალი მდგომარეობას იცვლის წამში 9600-ჯერ ანუ თითო ბიტის ხანგრძლივობა შეადგენს წამის 1/9600, რაც დაახლოებით 100 მიკროწამია.

მონაცემთა ასო-ნიშნის გადაცემისას ბოდ-სიჩქარის გარდა არსებობს სხვა მახასიათებლებიც, რომლებიც განსაზღვრავენ მონაცემთა ნაკადის სწორ ინტერპრეტაციას.

პირველი მახასიათებელი - არის გადასაცემი ბაიტის სიგრძე ბიტებში. ეს სიგრძე ზოგადად შეიძლება იყოს 5-დან 8 ბიტამდე.

მეორე მახასიათებელი - არის პარიტეტი (parity). პარიტეტი შეიძლება იყოს ლუწი, კენტი, mark, space ან ცარიელი (none). ლუწი პარიტეტის შემთხვევაში ბოლო გადაცემული ბიტი იქნება ლოგიკური 1, თუ გადაცემული მონაცემების ბაიტი შეიცავს 0-იანი ბიტების ლუწ რაოდენობას. კენტი პარიტეტის შემთხვევაში ბოლო გადაცემული ბიტი იქნება ლოგიკური 1, თუ გადაცემული მონაცემების ბაიტი შეიცავს 0-იანი ბიტების კენტ რაოდენობას. **Mark** პარიტეტისას ბოლო გადაცემული ბიტი ყოველთვის არის ლოგიკური 1, ხოლო **Space** პარიტეტისას

ბოლო გადაცემული ბიტი ყოველთვის **0**-ია. ცარიელი პარიტეტისას პარიტეტის ბიტი საერთოდ არ გადაეცემა.

მესამე მახასიათებელი - არის სტოპ-ბიტების რაოდენობა. ეს მნიშვნელობა 1-ის ან 2-ის ტოლია.

ვთქვათ, საჭიროა მიმდევრობითი პორტით გადავცეთ ასო "A". ამ ასოს ორობითი წარმოდგენა არის 01000001. იმის გათვალისწინებით, რომ ბიტები გადაეცემა მიმდევრობით, უმცროსი მნიშვნელობის ბიტიდან (least significant bit – LSB) უფროსი მნიშვნელობის ბიტამდე (most significant bit – MSB) მომდევნო ხაზის პარამეტრებისათვის – 8-ბიტიანი ბაიტი, ცარიელი პარიტეტი, 1 სტოპ-ბიტი, 9600 ბოდი, გადაცემული მონაცემების ნაკადი იქნება:

LSB (0 1 0 0 0 0 0 1 0 1) MSB

ეს არის 1 სტარ-ტბიტი, 8 მონაცემთა ბიტი და ერთი სტოპ-ბიტი.

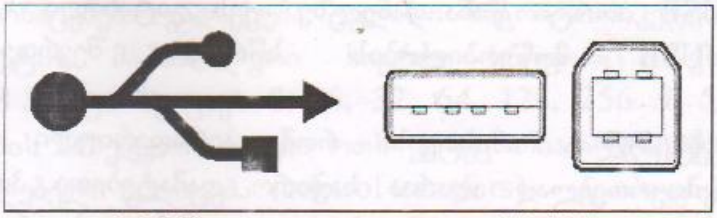
ხაზის რეალური სიჩქარის გასაანგარიშებლად, ბოდ-სიჩქარე საჭიროა გავყოთ ერთი ბაიტის გადასაცემად საჭირო ბიტების რაოდენობაზე. ჩვენ მაგალითში, ერთი ბაიტის გადასაცემად გამოიყენება 10 ბიტი, რაც 9600 ბოდ-სიჩქარის შემთხვევაში გვაძლევს 960 ბაიტ/წმ სიჩქარეს.

ზემომოყვანილი ინფორმაცია აღწერს მონაცემთა ნაკადის ელექტრონულ-ლოგიკურ მახასიათებლებს. რაც შეეხება ხაზის პროტოკოლს, მიმდევრობითი კომუნიკაცია შეიძლება იყოს ნახევარ ან სრულდუპლექსიანი. სრული დუპლექსი ნიშნავს, რომ მოწყობილობას შეუძლია ერთდროულად მონაცემთა გადაცემა და მიღება. ნახევარი დუპლექსი ნიშნავს, რომ მოწყობილობას არ შეუძლია ერთდროულად მონაცემთა გადაცემა და მიღება ანუ გადაცემა და მიღება უნდა ხდებოდეს მორიგეობით.

ნახევარდუპლექსიანი მიმდევრობითი კავშირი საჭიროებს მინიმუმ ორ სადენს - სიგნალის მიწას და მონაცემთა ხაზს. სრულდუპლექსიან მიმდევრობით კავშირს სჭირდება მინიმუმ სამი სადენი - სიგნალის მიწა, გადასაცემი მონაცემების ხაზი და მისაღები მონაცემების ხაზი. **RS-232** სტანდარტი განსაზღვრავს მიმდევრობითი კავშირის ფიზიკურ და ელექტრულ მახასიათებლებს. ამ სტანდარტში ასევე მოცემულია რამდენიმე დამატებითი სიგნალი, რომელიც აკონტროლებს მონაცემთა გადაცემის პროცესს. ეს სიგნალები ზემოთ უკვე იყო აღწერილი.

1.4.2. პორტი USB

USB (Universal Serial Bus) უნივერსალური მიმდევრობითი სალტე გამოიყენება მონაცემთა გადაცემისათვის. კომპიუტერებისთვის შექმნილმა ამ სტანდარტმა დიდი პოპულარობა მოიპოვა **PDA**-ებში, პორტატიულ **DVD** და მედია-პლეერებში, მობილურ ტელეფონებსა და ისეთ აპარატურაშიც კი, როგორიცაა აუდიო და ვიდეოტექნიკა და მონაცემთა შენახვის მოწყობილობები. **USB**-ს იმპლემენტაცია უმავთულო კავშირისათვის ცნობილია, როგორც **Wireless USB**.

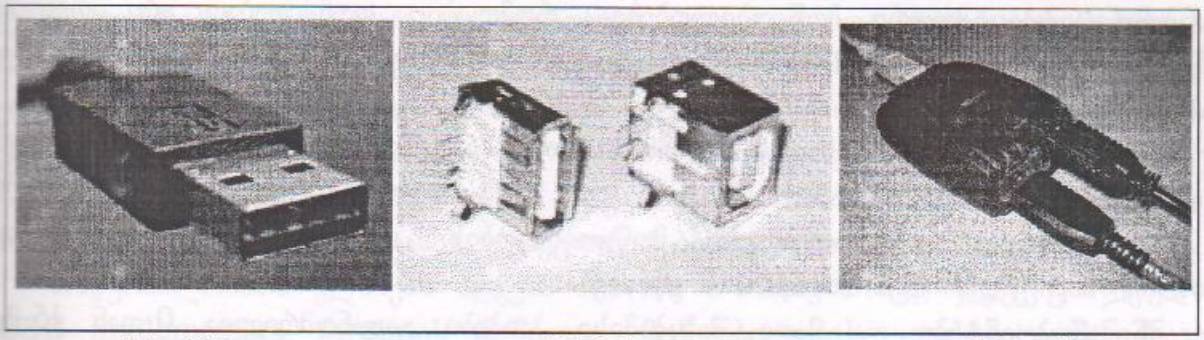


ნახ.1.13 ნახ.1.14

1.13 ნახაზზე ნაჩვენებია **USB** პორტის სიმბოლო, 1.14 ნახაზზე ნაჩვენებია **USB** ტიპის კონექტორების სქემატური გამოსახულება.

USB შეიქმნა, როგორც ერთ-ერთი ძირითადი კომპონენტი, ახალ, მოძველებული სტანდარტებისგან თავისუფალ, პერსონალურ კომპიუტერებზე გადასვლისათვის. **USB**-ს უნდა შეეცვალა მოძველებული **COM** მიმდევრობითი და **LPT** პარალელური პორტები პერსონალურ კომპიუტერებზე, რადგან ისინი არ იყო სწორად სტანდარტიზებული და საჭიროებდა მოწყობილობათა უამრავი დრაივერის შექმნას და მხარდაჭერას.

USB სისტემას აქვს ასიმეტრიული დიზაინი, რომელიც შედგება **host** (ჰოსტ, პატრონი) კონტროლერისა და მრავალი გადაჯაჭვული მოწყობილობისგან. დამატებითი **USB** ჰაბები (**USB** ჰაბი (**hab**) – მოწყობილობა, რომელიც საშუალებას იძლევა ერთ host **USB** პორტზე რამდენიმე მოწყობილობის ან სხვა ჰაბის მიერთებისა) შეიძლება იყოს ჩართული ამ ჯაჭვში, რაც საშუალებას იძლევა მოწყობილობების ხისებრ სტრუქტურაში გასაერთიანებლად, თუმცა მოწყობილობათა ჯამური რაოდენობა ერთ კონტროლერზე 127-ს არ უნდა აღემატებოდეს და "მოწყობილობათა ჩალაგების" მაქსიმალური დონე შეზღუდულია 5 განშტოებით. თანამედროვე კომპიუტერებს, როგორც წესი, გააჩნია რამდენიმე ჰოსტ კონტროლერი, რაც საერთო ჯამში ბევრი **USB** მოწყობილობის მიერთების საშუალებას იძლევა. **USB** კაბელები არ საჭიროებს ტერმინირებას.



ნახ. 1.15 ნახ 1.16 ნახ. 1.17

1.15 ნახაზზე მოცემულია პორტის კონექტორი (მამალი), 1.16 ნახაზზე – პორტის კონექტორები (დედალი), 1.17 ნახაზზე კი – ოთხპორტიანი **USB** ჰაბი.

მიუხედავად იმისა, რომ ერთ **USB** პორტზე მრავალი მოწყობილობის მიერთებაა შესაძლებელი ჰაბების საშუალებით, სხვადასხვა ტექნიკური თუ ეკონომიკური მიზეზების გამო ამ პრაქტიკამ ვერ პოვა მასობრივი გავრცელება. **USB** ჰაბების გამოყენების აუცილებლობის შესამცირებლად, თანამედროვე კომპიუტერებს გააჩნია ბევრი **USB** პორტი (როგორც წესი, 6 ცალი). დესკტოპ სისტემათა უმრავლესო-

ბას რამდენიმე **USB** პორტი წინა პანელზე აქვს გატანილი სხვადასხვა ტიპის პორტატიული **USB** მოწყობილობის სწრაფი შეერთების პროცესის გასაადვილებლად.

USB დაპროექტებულია იმისთვის, რომ პერიფერიული მოწყობილობების კომპიუტერთან მისაერთებლად აღარაა საჭირო დამატებითი პლატების ჩაყენება **ISA, EISA, PCI** ან სხვა სლოტებში. აგრეთვე, იმისათვის, რომ გაუმჯობესდეს მოწყობილობათა ***plug-and-play*** შესაძლებლობა. **USB** მოწყობილობები არის "ცხლად შეცვლადი" (hot-swapped), რაც ნიშნავს, რომ მათი კომპიუტერთან მიერთება ან გამორთვა შესაძლებელია კომპიუტერის გათიშვის ან რესტარტის გარეშეც. როდესაც **USB** მოწყობილობა პირველად ჩაირთვება კომპიუტერში, ჰოსტი ახდენს მის აღრიცხვასა და ამოცნობას და შემდეგ ტვირთავს შესაბამის საჭირო დრაივერს.

USB-ზე შესაძლებელია ისეთი პერიფერიული მოწყობილობების შეერთება, როგორიცაა მაუსი, კლავიატურა, **PDA**, ჯოისტიკი, სკანერი, ციფრული ფოტოკამერა, პრინტერი, მონაცემთა გარე შემნახველი აპარატურა და სხვ. მოწყობილობათა მრავალი სახეობისთვის, მაგალითად, სკანერების და ფოტოკამერებისათვის, **USB** გახდა კომპიუტერთან მათი დაკავშირების სტანდარტული ხერხი. **USB** სულ უფრო ხშირად გამოიყენება არაქსელური პრინტერების შესაერთებლად და ცვლის **LPT** პარალელურ პორტს, რომელიც ადრე ფართოდ გამოიყენებოდა. **USB** ამარტივებს ერთ კომპიუტერზე რამდენიმე პრინტერის შეერთებასაც. **USB**-ს საშუალებით რამდენიმე მოწყობილობა ერთდება ერთ ჰოსტ კონტროლერზე ჰაბების ჯაჭვის მეშვეობით.

USB ტერმინოლოგიაში მოწყობილობებს უწოდებენ ფუნქციებს, რადგან ერთმა ფიზიკურმა მოწყობილობამ სინამდვილეში შეიძლება შეასრულოს რამდენიმე ფუნქცია, მაგალითად, როუტერმა (მარშრუტიზატორმა) შეიძლება თავის ძირითად ფუნქციასთან ერთად წაიკითხოს Secure Digital Card-ები (მეხსიერების ბარათები). ჰაბები სპეციალური მოწყობილობებია, რომლებიც ფუნქციებად არ ითვლება. **USB**-ს ყოველთვის აქვს ერთი სათავე ჰაბი, რომელიც უშუალოდ მიერთებულია ჰოსტ კონტროლერზე.

მოწყობილობებს (ფუნქციებს) გააჩნია მათთან ასოცირებული პაიპები (pipe) ანუ ლოგიკური არხები. პაიპი ბაიტ-ნაკადია. ეს არის კავშირის არხი ჰოსტ კონტროლერიდან შეერთებული მოწყობილობის ლოგიკურ სტრუქტურამდე, რომელსაც ენდპოინტი (endpoint) ეწოდება.

ეს ენდპოინტები და მათი შესაბამისი პაიპები გადანომრილია 0-დან 15-მდე ორივე მიმართულებით ანუ მოწყობილობას (ფუნქციას) აქვს 32 აქტიური პაიპი, 16 პაიპი ჰოსტ კონტროლერისკენ და 16 პაიპი ჰოსტ კონტროლერიდან. თითო ენდპოინტს შეუძლია მონაცემთა გადაცემა-მიღება მხოლოდ ერთი მიმართულებით ანუ ყველა პაიპი არის ერთმიმართულებიანი (unidirectional). მაგრამ **0** ენდპოინტი არის დარეზერვებული სალტის მართვისათვის ორივე მიმართულებით, რაც ნიშნავს, რომ ის იყენებს 32 პაიპიდან ორს. ყველა **USB** მოწყობილობამ აუცილებელია იმპლემენტირება გაუკეთოს **0** ენდპოინტს ანუ ნებისმიერ მოწყობილობას აქვს შიგა და გარე მიმართული პაიპი **0** ნომრით.

ამ პაიპებში მონაცემები გადაეცემა ცვლადი სიგრძის პაკეტებით. ყოველ პაიპს ახასიათებს პაკეტის მაქსიმალური სიგრძე (როგორც წესი, **2n** ბაიტი), შესაბამისად **USB** პაკეტი შეიცავს 8, 16, 32, 64, 128, 256 ან 512 ბაიტს.

გადაცემის ტიპის მიხედვით პაიპები ოთხ კატეგორიად (transfer type) იყოფა:

- მაკონტროლებელი გადაცემა (control transfers) - გამოიყენება მოწყობილობისათვის მოკლე და მარტივი ბრძანებების ან მდგომარეობის რეპორტების გადასაცემად, რომელიც იყენებს სალტის მაკონტროლებელ **0**-ვან პაიპს.
- იზოქრონული გადაცემა (isochronous transfers) - როდესაც მონაცემები გადაიცემა გარკვეული (არაა აუცილებელი, რომ მაქსიმალური) გარანტირებული სიჩქარით, მაგრამ შესაძლო მონაცემთა დანაკარგით. ასე ხდება აუდიო- ან ვიდეოსტრიმინგი (რეალურ დროში გამოსახულების ან ხმის გადაცემა).
- წყვეტის გადაცემა (interrupt transfers) - იყენებენ მოწყობილობები, რომელსაც სჭირდება სწრაფი პასუხი და რეაგირება, მაგალითად, კლავიატურა ან მიმთითებელი მოწყობილობები (pointing devices).
- მასიური გადაცემა (bulk transfers) - იყენებს არხის მთელ დარჩენილ თავისუფალ გამტარობას (მაგრამ შეყოვნების არარსებობის და მუდმივი სიჩქარის გარანტიის გარეშე). ასეთი გადაცემა გამოიყენება ფაილების კოპირებისას.

როდესაც მოწყობილობა ან ფუნქცია მიერთებულია ჰოსტ კონტროლერზე ჰაბის ან სალტის საშუალებით, მას ჰოსტ კონტროლერის მიერ ეძლევა უნიკალური 7-ბიტიანი მისამართი.

მოწყობლობა, რომელიც უერთდება **USB**-ს შეიძლება იყოს სპეციფიკური და არასტანდარტული და მოითხოვდეს არასტანდარტულ დრაივერებს კორექტული ფუნქციონირებისათვის ან ეს მოწყობილობა შეიძლება მიეკუთვნებოდეს ე.წ. მოწყობილობათა კლასს. მოწყობილობათა კლასები განსაზღვრავს მოწყობილობათა ფუნქციონირების წინასწარ მოსალოდნელ ლოგიკას და ფუნქციონირებისთვის საჭირო ინტერფეისებსა და დესკრიპტორებს. შესაბამისად, შესაძლებელი ხდება ერთი და იგივე დრაივერის გამოყენება ყველა მოწყობილობისათვის, რომელიც ერთ კლასს მიეკუთვნება. იმისათვის, რომ ოპერაციულმა სისტემამ უზრუნველყოს **USB** ინტერფეისის ფუნქციონირება, საჭიროა რომ მასში წარმოდგენილი იყოს მოწყობილობათა ყველა კლასის ფუნქციონირებისთვის საჭირო დრაივერები.

მოწყობილობათა კლასებს ადგენს Device Working Group ჯგუფი, რომელიც **USB** implementers forum-ის ნაწილია. მოწყობილობის კლასის აღსანიშნავად დრაივერის ინტერფეისის დესკრიპტორში გამოყოფილია ერთი ბაიტი სიგრძის ველი, შესაბამისად, შესაძლოა მაქსიმუმ 254 კლასის არსებობა (0x00 და 0xFF მნიშვნელობები დარეზერვებულია).

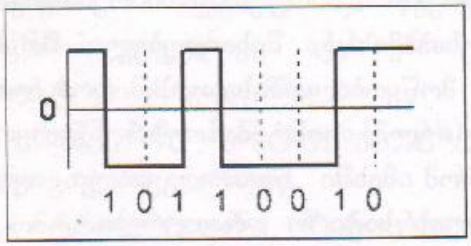
ყველაზე ხშირად გამოყენებული მოწყობილობათა კლასები მოცემულია შემდეგ ცხრილში:

ცხრილი 2

0x00	დარეზერვებული მნიშვნელობა – გამოიყენება მოწყობილობის დესკრიპტორში იმის დასადასტურებლად, რომ ინტერფეისის დესკრიპტორი ინარჩუნებს იმ მოწყობილობების კლასს, რომელიც იდენტიფიცირდება ყოველ ინტერფეისთან
0x01	აუდიომოწყობილობების კლასის ხმის ბარათების მსგავსი მოწყობილობების **USB**
0x02	საკომუნიკაციო მოწყობილობების კლასის **USB.** გამოიყენება მოდემებისათვის, ქსელური ბარათებისათვის, **ISDN** შეერთებებისათვის, ფაქსებისათვის და ა.შ.
0x03	ადამიანებთან ინტერფეისის მოწყობილობების კლასის (**HID**) **USB**, მაგალითად, კლავიატურა, მაუსი და ა.შ.
0x06	ილუსტრაციების შეტანის მოწყობილობების კლასის **USB.** ანალოგიურია სურათების გადაცემის პროტოკოლის, გამოიყენება **USB**-ს გავლით გადაცემისას
0x07	საბეჭდი და საბეჭდის მსგავსი მოწყობილობების კლასის **USB**
0x08	დიდი მოცულობის დამხსომებელი მოწყობილობების კლასის **USB.** მაგალითად, ფლეშ-მოწყობილობები, პორტატიული winჩესტერები, მეხსიერებითი ბარათების წამკითხავი მოწყობილობები, ციფრული კამერები, ციფრული აუდიოპლაიერები
0x09	ჰაბების **USB**
0x0E	ვიდეომოწყობილობების, ვებ-კამერების მსგავსი მოწყობილობების, მოძრაობების, სურათების შეყვანის მოწყობილობების კლასის **USB**
0xE0	უკაბელო კონტროლერების, მაგალითად, ბლუტუსის მოწყობილობების კლასის **USB**
0xFF	**USB** სპეციალური მოწყობილობების კლასისათვის, რომლებიც არ არის დამოკიდებული სტანდარტული კლასის მოწყობილობებზე და მოითხოვს სპეციალურ აპარატულ გადაწყვეტას

მონაცემთა გადაცემა

მონაცემთა კოდირებისათვის **USB** სტანდარტში გამოიყენება **NRZI** სისტემის კოდირება. Non-Return-to-Zero Inverted (**NRZI**) კოდირება არის მეთოდი, რომელიც საშუალებას იძლევა ლოგიკური ბინარული სიგნალის გარდაქმნისა ფიზიკურ სიგნალად და მის გადასაცემად მონაცემთა რაიმე მატარებლის მეშვეობით. ორდონიან **NRZI** სიგნალში ერთი დონიდან მეორეზე გადასვლა ხდება იმ შემთხვევაში, თუ გადაცემული ბიტი ლოგიკური 1-ია, 0-ის გადაცემისას დონის შეცვლა არ ხდება (ნახ. 1.18).



ნახ. 1.18. NRZI კოდირება

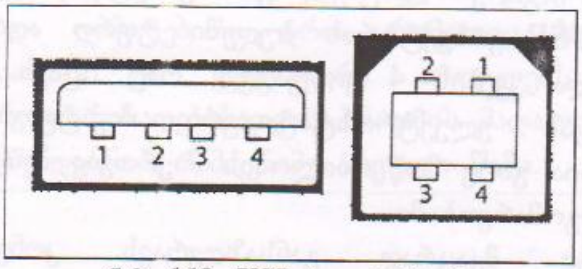
ნახაზზე ნაჩვენებია **NRZI** კოდირების მაგალითი, სადაც "ერთიანი" არის სიგნალის ფიზიკური დონის ცვლილება, ხოლო "ნული"–დონის შენარჩუნება.

USB სიგნალი გადაიცემა დახვეული წყვილი კაბელის საშუალებით, რომელიც აღინიშნება, როგორც D+ და D-. ამ კაბელებში გამოიყენება სიგნალების ნახევრად დუპლექსური დიფერენცირებული გადაცემა, ელექტრომაგნიტური ხმაურის ეფექტთან საბრძოლველად. D+ და D- მუშაობს ერთად. ეს არ არის ორი განცალკევებული სიმპლექს-ხაზი. გადაცემული სიგნალის დონე შეესაბამება 0.0 - 0.3 ვოლტს დაბალი ლოგიკური მნიშვნელობისათვის და 2.8 - 3.6 ვოლტს მაღალი ლოგიკური მნიშვნელობისათვის.

USB საშუალებით მონაცემთა გადაცემა სამი განსხვავებული სიჩქარითაა შესაძლებელი:

- Low Speed rate 1.5 მბიტ/წმ-მდე. ეს სიჩქარე გამოიყენება უმთავრესად Human Interface Devices (**HID**) მოწყობილობებში, მაგალითად, კლავიატურებში ან მაუსებში.
- Full Spedd rate 12 მბიტ/წმ-მდე. ეს სიჩქარე იყო ყველაზე მაღალი **USB2.0** სპეციფიკაციის გამოჩენამდე.
- Hi-Speed rate 480 მბიტ/წმ-მდე.

USB-ს კონექტორებში (ნახ.1.19) კონტაქტებს აქვს შემდეგი დანიშნულება (ცხრილი 3):



ნახ. 1.19. USB-ს კონექტორი

ცხრილი 3

კონტაქტი	დანიშნულება
1	V_{BUS} (4.75–5.25 ვ.)
2	D–
3	D+
4	GND
კორპუსი	ეკრანი

USB კონექტორები ისეა შემუშავებული, რომ დაკმაყოფილდეს ყველა წაყენებული მოთხოვნა. გარდა ამისა, გათვალისწინებულია სხვა მოდელების კონექტორების ექსპლუატაციის გამოცდილება:

- კონექტორები უნდა იყოს გამძლე და ძნელად მტვრევადი. მრავალი, მანამდე შექმნილი კონექტორი იყო მყიფე, გააჩნდა კონტაქტები, რომლებიც ადვილად იღუნებოდა და ტყდებოდა სუსტი ზემოქმედების დროსაც კი. **USB** კონექტორის ელექტრული კონტაქტები დაცულია პლასტმასის საფარველით, ჩასმულია ლითონის კორპუსში. შედეგად შესაძლებელია კონექტორის უსაფრთხო ექსპლუატაცია. დაცული კორპუსის გამო, კონექტორი შეიძლება დააგდოთ, დააბიჯოთ, მაგრამ ეს ქმედება ზიანს ვერ მიაყენებს.
- პრაქტიკულად შეუძლებელია **USB** კონექტორის არასწორად მიერთება. ამოტრიალებულ მდგომარეობაში კონექტორი არ შევა ბუდეში. თუმცა

გამოუცდელი მომხმარებლისათვის არაა თვალსაჩინო შეერთების სწორი ვარიანტი, ამიტომ საჭირო ხდება ორივენაირი შეერთების მოსინჯვა.

- კონექტორების წარმოება ძალიან იაფია.
- კონექტორის დიზაინი მომხმარებელს აიძულებს სწორი ტოპოლოგიის დაცვას.
- კონექტორის შესაერთებლად და გამოსართავად საჭიროა მცირე ძალა. **USB** კონექტორი ჩერდება თავის ბუდეში მოჭიდების ძალის საშუალებით, შესაბამისად, არაა საჭირო ხრახნების ან სხვა რაიმე მომჭერების გამოყენება.
- კონექტორის კონსტრუქცია უზრუნველყოფს იმას, რომ კონექტორის მიერთებისას გარე დამცავი კორპუსი-მიწა უკავშირდება შესაბამის კონტაქტს **USB** კონექტორის ბუდეში უფრო ადრე, ვიდრე მონაცემთა გადაცემის და კვების 4 კონტაქტი, რაც იცავს ორივე მხარის ფაქიზ ელექტრონიკას ელექტროსტატიკური მუხტისგან. მისი მეშვეობით შესაძლებელია გარე მოწყობილობის მიერთება მთავარ მოწყობილობასთან ჩართულ მდგომარეობაში.

USB სტანდარტი მკაცრად განსაზღვრავს კონექტორების ფიზიკურ მახასიათებლებს, მაგალითად, ზომებს, რაც მიზნად ისახავს მიღწეულ იქნეს შეუთავსებლობის მინიმიზაცია სხვადასხვა მწარმოებლის მიერ დამზადებულ კონექტორებს შორის. სტანდარტი განსაზღვრავს როგორც კონექტორის ზომებს, ისე კონექტორის კორპუსის ზომებზე შეზღუდვას. ეს გაკეთდა იმისათვის, რომ ზედმეტად დიდი ზომის კორპუსს არ დაებლოკა თავისი კონექტორის ბუდის გვერდით მდებარე ბუდეები.

USB კაბელებს გააჩნია მხოლოდ მამალი კონექტორები, ხოლო მოწყობილობებს - მხოლოდ დედალი. **USB** კაბელის მაქსიმალური სიგრძე შეიძლება იყოს 5მ. უფრო გრძელ მანძილზე შესაერთებლად **USB** ჰაბები გამოიყენება.

კვების მიწოდება

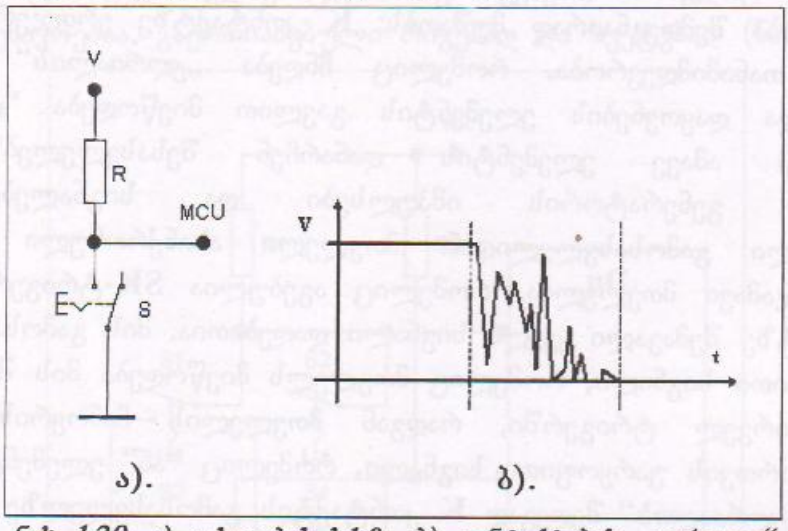
USB-ს სპეციფიკაციის თანახმად, **USB** პორტს აქვს ერთი ხაზი, რომლის საშუალებით მოწყობილობას შეუძლია **5ვ** კვების ძაბვა ჰოსტისგან მიიღოს. სტანდარტის თანახმად, **+V** და **-V** ძაბვა კვების ხაზებს შორის არ უნდა იყოს **5.25ვ** მეტი და **4.35ვ** ნაკლები. შეერთებისას მოწყობილობას შეუძლია მხოლოდ **100მა** დენის მოხმარება. იმ შემთხვევაში, თუ მოწყობილობა მეტ დენს საჭიროებს, შეუძლია მოითხოვოს ის ჰოსტისგან. მაქსიმალური დენი, რომლის მიწოდება შეუძლია **USB** პორტს არის **500მა**. ჰაბის გამოყენების შემთხვევაში, მასზე მიერთებულმა მოწყობილობებმა შეიძლება გამოიყენონ არა უმეტეს **400მა** დენი, რაც ასევე ზღუდავს მასზე პორტების რაოდენობას 4 ერთეულამდე.

1.5. მიკროკონტროლერების გარე მოწყობილობები

გადამრთველების ჩართვისა და გამორთვისას წრედში ჩნდება იმპულსური დამახინჯებები, რომლებიც გამოწვეულია კონტაქტების დაწოლითი დრეკადობით (ხტომით). ამ დამახინჯებებს *„ჟრიალს“* უწოდებენ. ასეთი მოვლენა ხშირად გვხვდება მიკროკონტროლერების ბაზაზე შექმნილ სისტემებში, სადაც მონაცემთა

შეყვანისათვის კლავიატურა გამოიყენება და „ჟრიალი“ შეიძლება აღიქვას, როგორც კლავიშზე მრავალჯერადი დაწოლა (ნახ.1.20). „ჟრიალი“ წარმოიქმნება კონტაქტის დაყენებისა და გაწყვეტისას კლავიშზე დაჭერის გზით.

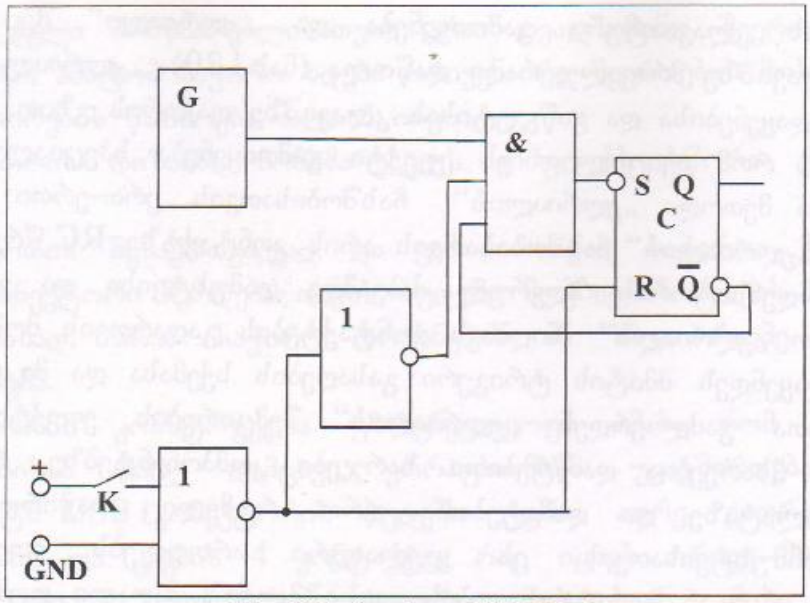
იმისათვის, რომ ავიცილოთ ეს ეფექტი გამოიყენება სპეციალური სქემები ან პროგრამული მეთოდი „ჟრიალის“ ჩახშობისათვის. ერთ-ერთი სქემატექნიკური საშუალება „ჟრიალის“ ჩახშობისათვის არის კონტაქტზე **RC** წრედის ჩართვა. ამ სქემაში კონდენსატორის ზღვრულ ძაბვამდე დამუხტვისა და განმუხტვისათვის საჭირო დრო „ჟრიალს“ ნიღბავს კონტაქტების გადართვის მომენტში. აგრეთვე შეიძლება დაყენდეს შმიტის ტრიგერი გასაღების სქემასა და მიკროკონტროლერს შორის, რათა გაძლიერდეს „ჟრიალის“ შემცირების ეფექტი. ამ მეთოდის უარყოფითი მხარეებია დამატებითი ხარჯები კომპონენტებზე, რომლებიც უნდა დაყენდეს პლატაზე და დამატებითი დრო, რომელიც საჭიროა **RC** წრედის დამუხტვა/განმუხტვისათვის. ეს ყველაფერი ართულებს მოცემული სქემის გამოყენებას, რადგან ზოგი გასაღებისათვის ხმაურის მაღალი დონით დამატებითმა დაყოვნებამ შეიძლება შეადგინოს რამდენიმე ათეული წამი.



ნახ. 1.20. ა) გასაღების სქემა, ბ) კონტაქტების „ჟრიალი“

მოწყობილობა, რომელიც მთლიანად გამორიცხავს „ჟრიალის“ მოვლენას ნაჩვენებია 1.21 ნახაზზე. ამას გარდა, ეს მოწყობილობა შეიძლება მოემსახუროს კონტაქტების შეუზღუდავ რაოდენობას და რთულ სქემებში გამოყენებისას მნიშვნელოვნად ზრდის მათ სწრაფქმედებას და საიმედოობას.

მოწყობილობა შედგება **K** კონტაქტისაგან, რომელიც მიერთებულია “ან” ელემენტის პირველ შესასვლელთან. ამავე ელემენტის მეორე შესასვლელი მიერთებულია “მიწის” სალტესთან (**GND**). “ან” ელემენტის გამოსასვლელი მიერთებულია **C** მთვლელის ჩამოყრის **R** შესასვლელზე. იგივე გამოსასვლელი ინვერტორის (დაყოვნების ელემენტის) გავლით მიერთებულია სამშესასვლელიან “და” ელემენტის ერთ-ერთ შესასვლელთან. ამ ელემენტის დანარჩენ შესასვლელებთან მიერთებულია **G** გენერატორის გამოსასვლელი და **C** მთვლელის ინვერსირებული გამოსასვლელი. “და” ელემენტის გამოსასვლელი მიერთებულია მთვლელის **S** შესასვლელზე.



ნახ. 1.21. "ჟრიალის" ჩახშობა

მოწყობილობა შემდეგნაირად მუშაობს: **K** კონტაქტზე ერთჯერადი დაჭერისას სიგნალების თანამიმდევრობა, რომელიც ჩნდება *„ჟრიალის"* შედეგად **"ან"** ელემენტის და დაყოვნების ელემენტის გავლით მიეწოდება **"და"** ელემენტის შესასვლელზე. ამავე ელემენტის დანარჩენ შესასვლელებზე მიეწოდება ავტორხევითი გენერატორის იმპულსები და სიგნალები მთვლელის ინვერტირებული გამოსასვლელიდან. მთვლელი ასინქრონული ოთხთანრიგიანი ორობითი ამჯამავი მთვლელია, რომელიც აგებულია **SK**-ტრიგერებზე. როდესაც **"და"** ელემენტზე შემავალი ყველა სიგნალი დადებითია, მის გამოსასვლელზე ასევე ჩნდება დადებითი სიგნალი, რომელიც მთვლელს მიეწოდება მის შესასვლელზე და ჩაიწერება პირველ ტრიგერში, რადგან მთვლელის ჩამოყრის შესასვლელზე იმყოფება ნებართვის უარყოფითი სიგნალი, რომელიც **"ან"** ელემენტიდან მიიღება.

ვინაიდან *„ჟრიალის"* შედეგად **K** კონტაქტის გამოსასვლელზე ყოველ გაჩენილ იმპულს შორის ხდება "ან" ელემენტის შესასვლელის გათიშვა, ამ შუალედში "ან" ელემენტი მუშაობს, როგორც ინვერტორი და აინვერტირებს მეორე შესასვლელის უარყოფით სიგნალს, რის შედეგადაც მთვლელის ჩამოყრის შესასვლელზე მიეწოდება დადებითი სიგნალი, რომელიც ყოველი ჩაწერილი იმპულსის შემდეგ ჩამოყრის მთვლელს საწყის მდგომარეობაში. ამავე შუალედში "და" ელემენტის შესასვლელზე დაყოვნების ელემენტის მუშაობის შედეგად მიეწოდება აკრძალვის უარყოფითი სიგნალი.

ზემოაღწერილი პროცესი მეორდება კონტაქტის *„ჟრიალის"* მთელი პერიოდის განმავლობაში. **K** კონტაქტის საბოლოო შერთვის შემდეგ "და" ელემენტის შესასვლელზე მიიღება დადებითი სიგნალები, რის შედეგადაც ის ამუშავდება და გაიმეორებს გენერატორის იმპულსების თანამიმდევრობას, რომელიც მიეწოდება მთვლელის შესასვლელზე და აითვლება მის მიერ, რადგან ჩამოყრის შესასვლელზე იმყოფება ნებართვის უარყოფითი სიგნალი.

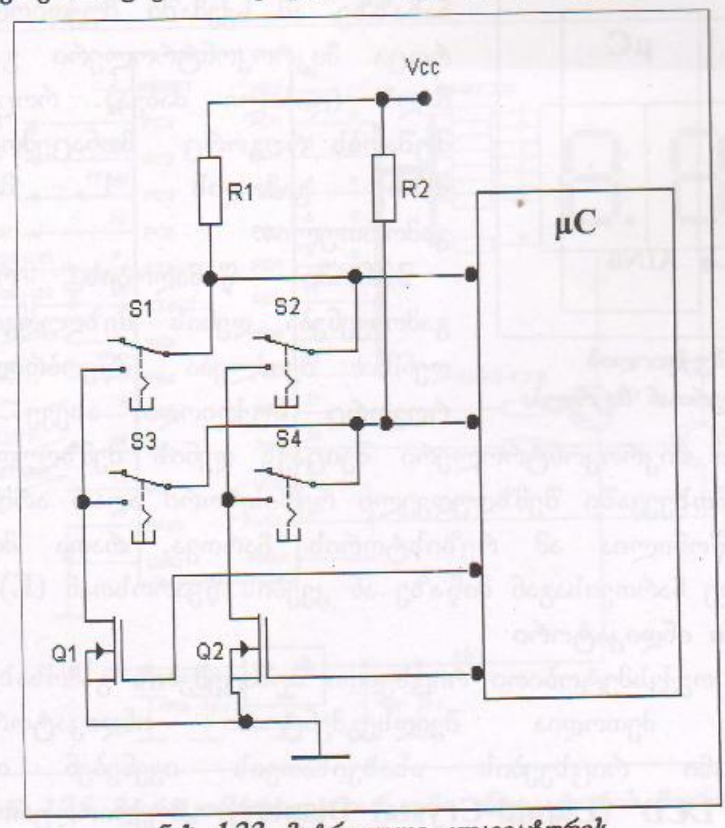
მთვლელის შევსების შემდეგ, მის პირდაპირ გამოსასვლელზე (**Q**) ჩნდება დადებითი სიგნალი, რომელიც მიეწოდება სხვა მოწყობილობებს, ხოლო უარყოფითი

სიგნალი, რომელიც ამავე დროს ჩნდება ინვერტირებულ გამოსასვლელზე ($\overline{Q}$) ბლოკავს "და" ელემენტს. **K** კონტაქტის საბოლოო გამორთვის შემდეგ მთვლელი ჩამოიყრება, რადგან ჩამოყრის შესასვლელზე მიეწოდება დადებითი სიგნალი "ან" ელემენტიდან. მოწყობილობა ბრუნდება საწყის მდგომარეობაში და მზადაა შემდგომი მუშაობისათვის.

კიდევ ერთი კარგი საშუალება *„ჟლრიალის“* შემცირებისათვის: თუ ძაბვის დონე გასაღების გამოსავლელზე არ იცვლება 20 მწმ განმავლობაში, მაშინ შეიძლება ჩაითვალოს, რომ *„ჟლრიალი“* დამთავრდა და კონტაქტის მდგომარეობის მეტი ცვლილება არაა მოსალოდნელი.

1.5.1. მატრიცული კლავიატურის გამოყენება

ბევრი პრაქტიკული ამოცანა მოითხოვს მონაცემების კლავიატურით შეყვანას. ეს შეიძლება რეალიზებულ იქნეს განცალკევებული ღილაკების დახმარებით, მაგრამ ასეთი მიდგომა სქემატექნიკურად გაუმართლებელია, რადგან მოითხოვს მიკროკონტროლერის შეტანა/გამოტანის ხაზების გამოყენებას. საუკეთესო გადაწყვეტილებაა მატრიცული კლავიატურის გამოყენება, რომელიც ელექტრონული გასაღებების შენაკრებია, გაერთიანებული რიგებად და სვეტებად (ნახ.1.22).



ნახ. 1.22. მატრიცული კლავიატურის მიკროკონტროლერთან მიერთება

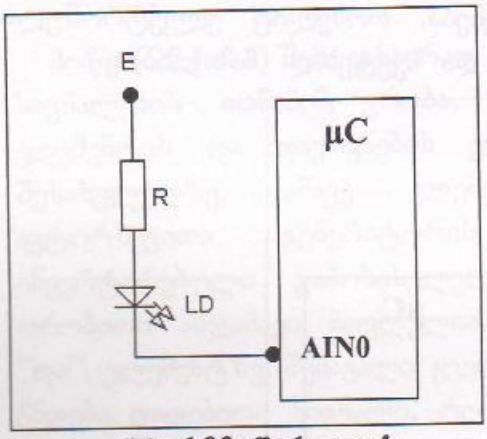
განსაზღვრული გასაღების მდგომარეობის წაკითხვისათვის სვეტს მიეწოდება სიგნალი, შემდეგ იკითხება რიგების მდგომარეობა. ჩვეულებრივ, რიგები ჩაირთვება მაღალ პოტენციალზე, ხოლო მოთხოვნილი სვეტი შეერთდება დამიწებაზე. თუ სკანირებისას ხდება დაბალი დონის სიგნალის წაკითხვა, ეს ნიშნავს, რომ გასაღები პოზიციაში რიგი-სვეტი ჩართულია. 1.22 ნახაზზე ნაჩვენებია ორი **MOS-**

ტრანზისტორი, რომლებიც გამოიყენება სვეტების დამიწებაზე ჩართვისათვის. მაგრამ ზოგ მიკროკონტროლერში გამომყვანებმა შეიძლება იმუშაოს ღია კოლექტორების რეჟიმში და, შესაბამისად, მოახდინოს ამ ტრანზისტორების მუშაობის იმიტაცია, მაშინ მათი ჩართვა აღარ იქნება საჭირო.

მატრიცული კლავიატურა პრაქტიკულად ნებისმიერ ზომამდე შეიძლება გაფართოვდეს, ამისათვის გამოიყენება მიკროკონტროლერის გამომყვანების არცთუ დიდი რაოდენობა. მაგალითად, პერსონალური კომპიუტერის 104-კლავიშიანი კლავიატურა ესაა მატრიცა, რომელიც 13X8 გასალებისაგან შედგება.

1.5.2. შუქური ინდიკაცია

ხშირად მონაცემთა გამოყვანა მიკროკონტროლერიდან ხდება **LED (Light Emiting Diode)** შუქდიოდების საშუალებით, რომლებიც იაფია და ადვილად ირთვება მიკროკონტროლერთან. ჩვეულებრივ შუქდიოდების ნათებისათვის საჭიროა 16მა-ზე მეტი დენი, რაც მიკროკონტროლერების უმეტესი ნაწილისათვის მოთავსებულია გამოსასვლელი დენების დასაშვებ დიაპაზონში.



ნახ. 1.23. შუქდიოდის მიკროკონტროლერთან მიერთება

მიკროკონტროლერის გამოსასვლელთან შუქდიოდის ჩართვის ტიპური სქემა ნაჩვენებია 1.23 ნახაზზე. ამ სქემაში შუქდიოდი ანათებს მაშინ, როცა მიკროკონტროლერი გასცემს "0" სიგნალს (დაბალი ძაბვა). როცა გამოსასვლელი მუშაობს როგორც მონაცემთა შესასვლელი ან მასზე გამოდის "1", მაშინ შუქდიოდი გამორთულია.

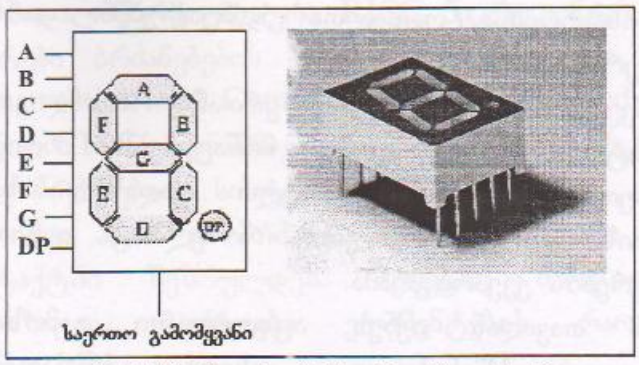
220ომი წინაღობის რეზისტორი **(R)** გამოიყენება დენის შეზღუდვისათვის: მაღალმა დენმა შეიძლება მწყობრიდან გამოიყვანოს როგორც შუქდიოდი, ასევე მიკროკონტროლერიც. ზოგიერთი მიკროკონტროლერი შეიცავს დენის შემზღუდველებს გამომყვან ხაზებზე. ამ შემთხვევაში შემზღუდველი რეზისტორი აღარ არის საჭირო. მაგრამ მაინც მიზანშეწონილია ამ რეზისტორის ჩართვა, რათა მიკროკონტროლერი დავიცვათ მოკლე ჩართვისაგან მიწაზე ან კვების წყაროსთან **(E)**.

შვიდსეგმენტიანი ინდიკატორი

ათობითი და თექვსმეტობითი რიცხვითი მონაცემების გამოსახვისათვის ყველაზე მოსახერხებელი მეთოდია შვიდსეგმენტიანი ინდიკატორის გამოყენება. მრავალთანრიგიანი რიცხვების ასახვისათვის იყენებენ თხევადკრისტალიან ინდიკატორებს **LCD (Liquid-Crystal Display)**. შუქდიოდური შვიდსეგმენტიანი ინდიკატორები სქემებში ჩაირთვება დიდი სირთულის გარეშე და არ შექმნის პრობლემას მიკროკონტროლერის პროგრამული უზრუნველყოფისას.

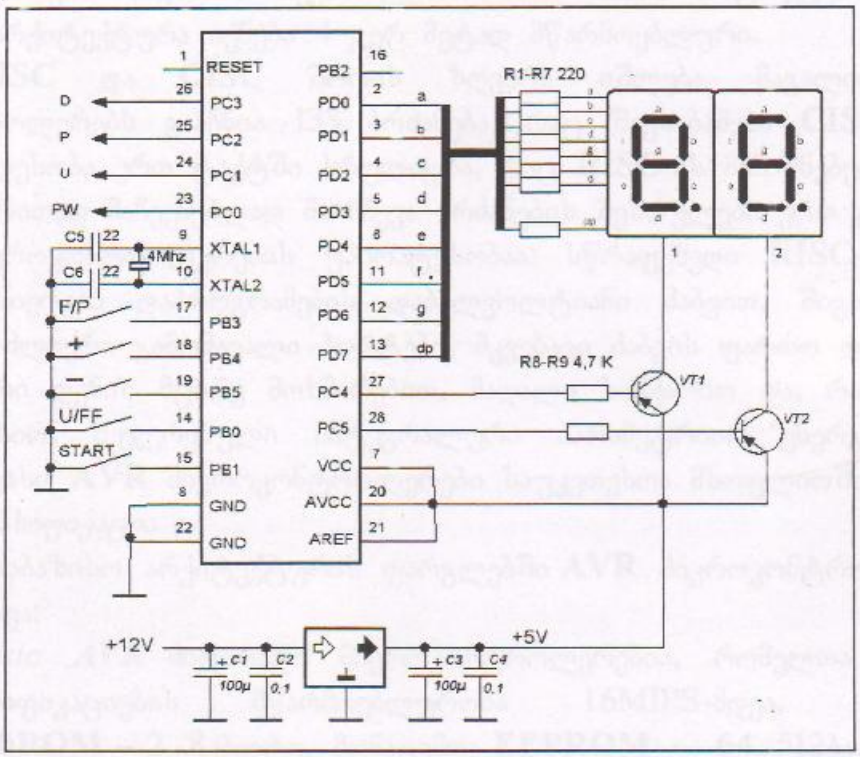
ინდიკატორში განსაზღვრული შუქდიოდების ჩართვისას (ანუ სეგმენტების ანთება) მიიღება ათობითი რიცხვის გამოსახულება (ნახ.1.24). ყოველ შუქდიოდს აქვს თავისი შესაბამისი ასო-იდენტიფიკატორი **(A, B, C, D, E, F, G,)** და **DP (dew point)** წერტილის შუქდიოდი. ჩამოთვლილი შუქდიოდების ერთი ფეხი მიერთებულია ინდიკატორის შესაბამის გარე გამომყვანზე. ყველა შუქდიოდის

მეორე ფეხი შეერთებულია ერთად და ჩართულია ინდიკატორის საერთო გამომყვანზე. საერთო გამომყვანი განსაზღვრავს ინდიკატორის ტიპს: საერთო კათოდით და საერთო ანოდით.



ნახ.1.24. შვიდსეგმენტიანი ინდიკატორი

ინდიკატორის ჩართვა მიკროკონტროლერთან საკმაოდ მარტივად ხორციელდება. ჩვეულებრივ, ინდიკატორს რთავენ, როგორც შვიდ ან რვა (თუ გამოიყენება ათობითი წერტილი) დამოუკიდებელ შუქდიოდს.



ნახ. 1.25. მიკროკონტროლერთან ორი ინდიკატორის მიერთება

ყველაზე მნიშვნელოვანი საკითხი მიკროკონტროლერთან რამდენიმე შუქინდიკაციის სისტემის ჩართვისას არის შეტანა/გამოტანის ხაზის დანიშნულება ყოველი შუქდიოდისათვის. ამ საკითხის გადაწყვეტა პროექტის შესრულების საწყის ეტაპზე ამარტივებს შემდგომში მოწყობილობების მონტაჟს და გამართვას. რამდენიმე ინდიკატორის ჩართვის ტიპური საშუალება იმაში მდგომარეობს, რომ ისინი ჩავრთოთ პარალელურად და შემდეგ ვმართოთ გამავალი დენით ყოველი

განცალკევებული ინდიკატორის საერ-თო გამომყვანების გავლით. ვინაიდან ამ დენის სიდიდე აღემატება მიკროკონტრო-ლერის გამომყვანებზე დასაშვები გამოსავალი დენის სიდიდეს, ამიტომ ინდიკატორის მართვისათვის ჩართავენ დამატებით ტრანზისტორებს, რომელთა საშუალებით აირჩევა, თუ რომელი ინდიკატორია აქტიურ მდგომარეობაში.

1.25 ნახაზზე ნაჩვენებია მიკროკონტროლერთან ორი ინდიკატორის მიერთება. ამ სქემაში მიკროკონტროლერი გასცემს მონაცემებს ინდიკაციისათვის, ერთი ინდიკატორიდან მეორეზე თანამიმდევრული გადასვლისას. ყოველი ციფრი გამოშუქდება დროის იმ ინტერვალში, რომელსაც განსაზღვრავს ტაიმერის შეწყვეტის მომსახურების ქვეპროგრამა.

იმისათვის, რომ თავიდან იქნეს აცილებული გამოსახულების ციმციმი, ქვეპროგრამაში გათვალისწინებულია ისეთი სწრაფქმედება, რომელიც უზრუნველყოფს ინდიკატორის ჩართვას ერთი წამის განმავლობაში სულ მცირე 50-ჯერ. რაც უფრო მეტი ინდიკატორია მიერთებული, მით უფრო სწრაფად მიეწოდება შეწყვეტა ტაიმერისგან. მაგალითად, რვა ინდიკატორის გამოყენებისას ციფრები ერთ წამში 400-ჯერ გამოდის.

თავი 2. AVR ოჯახის მიკროკონტროლერების აგების და ფუნქციონირების თავისებურებები

2.1. AVR ოჯახის მაღალმწარმოებლური RISC მიკროკონტროლერები

მიკროკონტროლერები ბრძანებების სისტემების მიხედვით ორ ტიპად იყოფა: **CISC** (Complex Instruction Set Computer) და **RISC** (Reduced Instruction Set Computer). ტერმინი **CISC** აღნიშნავს მიკროკონტროლერების რთულ, ხოლო **RISC** – შეზღუდულ ბრძანებათა სისტემას.

RISC სისტემის იდეა არის იმ ბრძანებების საფუძვლიანი შერჩევა, რომლებიც შეიძლება ერთ ტაქტში შესრულდეს. შედეგად მარტივდება პროცესორის აპარატული რეალიზაცია, მცირდება ელემენტების რაოდენობა, დაბლდება მოხმარებული სიმძლავრე და ფასი.

საერთო ჯამში ერთ **CISC**-ბრძანებას უნდა შეესაბამებოდეს რამდენიმე **RISC**-ბრძანება. მაგრამ **RISC**-ის სწრაფქმედების უპირატესობა ფარავს აღნიშნულ დანაკარგს. მაგალითად, თუ ყველაზე სწრაფი ბრძანება მიკროკონტროლერში **CISC**-ის არქიტექტურით 12 ტაქტში სრულდება და ყველა **CISC**-ინსტრუქციისათვის საჭიროა შევასრულოთ სამი **RISC**-ინსტრუქცია, მივიღებთ, რომ **RISC**-არქიტექტურა იქნება 4-ჯერ მეტად მწარმოებლური.

დღეს **RISC** და **CISC** შორის ზღვარი იშლება. მაგალითად, **AVR** მიკროკონტროლერებს გააჩნია 133 ბრძანება, რაც შეესაბამება **CISC**-ს, მაგრამ მათი უმრავლესობა ერთ ტაქტში სრულდება, რაც **RISC**-ის მანიშნებელია. ამიტომ **RISC**-ის ძირითად მაჩვენებლად მიიჩნევა ბრძანების შესრულება ერთ ტაქტში.

AVR მიკროკონტროლერების უპირატესობაა: სწრაფქმედი **RISC**-პროცესორი, **FLASH**-მეხსიერება დაპროგრამების დაბალვოლტიანი ძაბვით, შიგა გადამწერი **flashROM**, ძლიერი გამომავალი პორტები, მკვებავი ძაბვის ფართო დიაპაზონი და ეს ყველაფერი დენის მცირე მოხმარებით, მაღალი სიჩქარით და, რაც მთავარია, დაბალი ფასით. შეჯერებული ინტეგრალური პარამეტრით "ენერგომოხმარება-წარმადობა-ფასი" **AVR** მიკროკონტროლერები საუკეთესოა მსოფლიოში.

AVR-ის კლასიფიკაცია

ერთიანი საბაზისო არქიტექტურის ფარგლებში **AVR** მიკროკონტროლერები სამ ოჯახად იყოფა:

- *classic* ***AVR*** ძირითადი მიკროკონტროლერებია, რომელთა ცალკეული მოდიფიკაციების მწარმოებლურობა 16MIPS-მდეა, პროგრამების **flashROM** – 2...8კბაიტი, მონაცემთა **EEPROM** – 64...512ბაიტი, **SRAM** – 128...512ბაიტი;
- *mega* ***AVR*** წარმადობით 1–16MIPS რთული დანართებისთვის მოითხოვს დიდი მოცულობის მეხსიერებას, პროგრამების **flashROM** – 4...128კბაიტი, მონაცემთა **EEPROM** – 64...512ბაიტი, **SRAM** – 2...4კბაიტი. ჩაშენებული 10-თანრიგიანი 8-არხიანი **აცგ**, აპარატული მამრავლებელი 8x8. *მეგა* ***AVR*** ძლიერი მიკროკონტროლერია, რომელსაც გააჩნია პერიფერიების კარგი ნაკრები. ამ ოჯახს აქვს ყველაზე დიდი, მრავალფეროვანი მოდელების არჩევანი.

- *tiny AVR* - დაბალფასიანი მიკროკონტროლერებია 8 გამომყვანით, აქვს კვების ძაბვის კონტროლის ჩაშენებული სქემა, რაც საშუალებას აძლევს იმუშაოს გარე სუპერვიზორული მიკროსქემების გარეშე.

განხილული ოჯახები განსხვავდება პერიფერიის განვითარების ხარისხით და მონაცემთა და პროგრამების მახსოვრობის მოცულობით.

MEGA ოჯახის კონტროლერებს აქვს საუკეთესო მაჩვენებლები და, ამასთან ერთად, რომელი მიკროკონტროლერიც არ უნდა ავირჩიოთ, ყველა ოჯახის ბრძანების სისტემა ერთმანეთის თავსებადია სუსტიდან მძლავრ მიკროკონტროლერზე პროგრამის გადატანის დროს.

თითოეული ოჯახის მიკროკონტროლერს აქვს ენერგომოხმარების რამდენიმე რეჟიმი, წყვეტის ბლოკი, მოდარაჯე ტაიმერი და საშუალებას გვაძლევს ჩავატაროთ დაპროგრამება უშუალოდ მზა მოწყობილობაში.

შემდეგ განხილული იქნება ფუნქციურად ყველაზე სრულყოფილი **Mega** ოჯახის მიკროკონტროლერები.

AVR მიკროკონტროლერები ერთკრისტალიანი, 8-თანრიგიანი **RISC**-კონტროლერებია, აქვს ორი ტიპის ენერგოდამოუკიდებელი პროგრამების **flash**-მეხსიერება და მონაცემთა **EEPROM**-მეხსიერება (Erasable Programmable Read Only Memory – წაშლადი დასაპროგრამებელი მუდმივი დამხსომებელი მოწყობილობა). აგრეთვე აქვს ოპერატიული მეხსიერება **RAM**, შესავალ/გამოსავალი პორტები და სხვადასხვა პერიფერიული ინტერფეისული სქემები. ამ კონტროლერებს უშვებს ფირმა **ATMEL** (Advanced Technology Memory and Logic).

კომპანია **ATMEL** - მსოფლიოს ერთ-ერთი ლიდერია, რომელიც აწარმოებს ენერგოდამოუკიდებელი მეხსიერების ფართო სპექტრს, **FLASH**- მიკროკონტროლერებს და მიკროსქემებს დასაპროგრამებელი ლოგიკით – აიღო სტარტი **RISC**-მიკროკონტროლერების დამუშავებაზე, სადაც გამოიყენა ყველა ტექნიკური გამოცდილება, რითაც დღემდე მოვიდა.

ამერიკელმა მათემატიკოსმა ჯონ ფონ ნეიმანმა ჩამოაყალიბა თანამედროვე კომპიუტერების მუშაობის ძირითადი პრინციპები. მან შემოგვთავაზა არქიტექტურა, რომელმაც მისი სახელი დაერქვა (von Neumann architecture) და რომელიც ითვალისწინებდა პროგრამებისა და მონაცემების შენახვას საერთო მეხსიერებაში. დღეს ასეთი არქიტექტურა დამახასიათებელია კომპიუტერებში გამოყენებისათვის.

არქიტექტურა, რომელიც გვთავაზობდა პროგრამებისა და მონაცემების მეხსიერების ცალ-ცალკე გამოყენებას ეწოდება ჰარვარდული (Harvard architecture). ჰარვარდული არქიტექტურა ცენტრალურ პროცესორს საშუალებას აძლევს ერთდროულად იმუშაოს როგორც პროგრამების, ასევე მონაცემთა მეხსიერებებთან, რაც მნიშვნელოვნად ზრდის მარგი ქმედების კოეფიციენტს.

კომპანიაში დამუშავებული **AVR**-არქიტექტურა აერთიანებს მძლავრ ჰარვარდულ **RISC**-პროცესორს, პროგრამებისა და მონაცემების მეხსიერებაში ცალ-ცალკე შეღწევადობით და საერთო დანიშნულების 32 რეგისტრს, რომელთაგან თითოეული მუშაობს, როგორც რეგისტრ-აკუმულატორი. **AVR**-არქიტექტურა უზრუნველყოფს განვითარებული ბრძანების სისტემას ფიქსირებული 16 ბიტი სიგრძის ბრძანებებში. ბრძანებების უმრავლესობა სრულდება ერთი მანქანური

ტაქტით, ასევე ერთდროულად სრულდება შემდგომი ბრძანებების ამორჩევა, რაც უზრუნველყოფს ტაქტური სიხშირის ყოველ მეგაჰერცზე 1MIPS-მდე წარმადობას.

2.1.1. AVR მიკროკონტროლერის ბირთვის არქიტექტურა

პროცესორი

როგორც აღინიშნა, **AVR** მიკროკონტროლერები შექმნილია ჰარვარდის არქიტექტურის მიხედვით, რაც გულისხმობს მახსოვრობის გაყოფას ორ ძირითად ნაწილად: მონაცემებისა და პროგრამებისათვის. თითოეულში შეღწევა საკუთარი სალტით ხორციელდება. მიკროკონტროლერები იყენებს ერთსაფეხურიან კონვეიერულ დამუშავებას, რაც ნიშნავს, რომ მიმდინარე ბრძანების შესრულებისას სრულდება მომდევნო ბრძანების ჩატვირთვა პროგრამების მეხსიერებიდან. აქედან გამომდინარე, მიიღწევა ბრძანების შესრულება ერთ ტაქტურ ციკლში. ამიტომ სისტემური ტაქტირებისათვის კვარცული ოსცილატორით, უშუალოდ გენერირებული ტაქტური სიხშირის გამოყენებისას (უმრავლეს პროცესორსა და კონტროლერში სისტემური ტაქტის მიღებისას კვარცული ოსცილატორით გენერირებული სიხშირის გაყოფა მოცემულ კოეფიციენტზე), მიიღება სწრაფქმედება, რომელიც უტოლდება გამოყენებული კვარცის სიხშირეს.

მაგალითად, 7-მეგაჰერციანი კვარცის სიხშირეზე მუშაობისას ვღებულობთ დაახლოებით 7 მილიონ შესრულებულ ბრძანებას ერთ წამში (0,14 მიკროწამი ერთ ბრძანებაზე). მცირედი გამონაკლისის გარდა, როდესაც კონვეიერი ირღვევა, **AVR** ოჯახის მიკროკონტროლერები ამუშავებს ყველა ბრძანებას ერთი სისტემური ტაქტის განმავლობაში.

AVR მიკროკონტროლერების "გული" 8-ბიტიანი მიკროპროცესორული ბირთვია ანუ ცენტრალური პროცესორული მოწყობილობა (**ცპუ**). **AVR**-ის ყველა მოდელში ის ფაქტობრივად ერთნაირია, რაც დიდი პლუსია. სწორედ არქიტექტურის ერთნაირობა განაპირობებს კოდის ადვილ გადატანას ერთი ოჯახის მოდელიდან მეორეზე. პირველ რიგში, ბირთვი წარმოდგენილია არითმეტიკულ-ლოგიკური მოწყობილობით (**ALU-ალმ**), პროგრამების მეხსიერებით (Flash Programm Memory), მართვის ბლოკით და პროგრამული მთვლელით (Program Counter- **PC**). შეიცავს ტაქტურ გენერატორს, რომლის იმპულსებით მუშაობს მიკროკონტროლერების ბლოკები.

პროგრამების მეხსიერებიდან, ბრძანების მთვლელის შიგთავსის შესაბამისად, სისტემური ტაქტური სიგნალის მიხედვით, შეირჩევა მორიგი ბრძანება და **ალმ**-ში სრულდება. ბრძანების შერჩევისას პროგრამების მეხსიერებიდან ხდება წინა შერჩეული ბრძანების შესრულება. რაც საშუალებას იძლევა მივაღწიოთ სწრაფქმედებას 1MIPS - 1 მეგაჰერცზე.

მიკროკონტროლერების სტარტის დროს პროგრამული მთვლელის მნიშვნელი უდრის "0000". ეს არის პირველი ბრძანების მისამართი **flashROM**-ში. იკროკონტროლერი იქიდან იღებს ორ ბაიტს (ბრძანების კოდი და მისი არგუმენტები) და შესასრულებლად აძლევს ბრძანების დეკოდერში (Instruction Decoder).

მონაცემთა მომავალი ბედი დამოკიდებულია ბრძანებაზე. თუ ეს უბრალო ბრძანებაა რაიმე მოქმედების შესასრულებლად, ის შესრულდება. ხოლო შემდგომ ტაქტზე პროგრამული მრიცხველის ნიშნული გაიზრდება და შემდეგი

მახსოვრობის უჯრედების წყვილიდან ამოიღება ბრძანების შემდეგი 2 ბაიტი და ასევე გაიგზავნება შესრულებაზე.

იმ შემთხვევაში, როდესაც გვხვდება გადასვლის ბრძანება, პროგრამულ მრიცხველში ჩაიტვირთება ბრძანებაში მითითებული მისამართი (აბსოლუტური გადასვლა) ან მისი მნიშვნელობა გაიზრდება არა ერთით, არამედ იმდენით, რამდენიც საჭიროა. შემდგომ ტაქტზე მიკროკონტროლერი ბრძანებას უკვე ახალი მისამართიდან აიღებს.

ბრძანების დეკოდერით ხდება ბრძანების "ჩატაცება" და მისი გადაცემა მართვის ბლოკის ლოგიკისათვის, რომელიც უკვე ზემოქმედებს დანარჩენ ბლოკებზე, აიძულებს შეასრულოს საჭირო ქმედება, საჭირო რიგით.

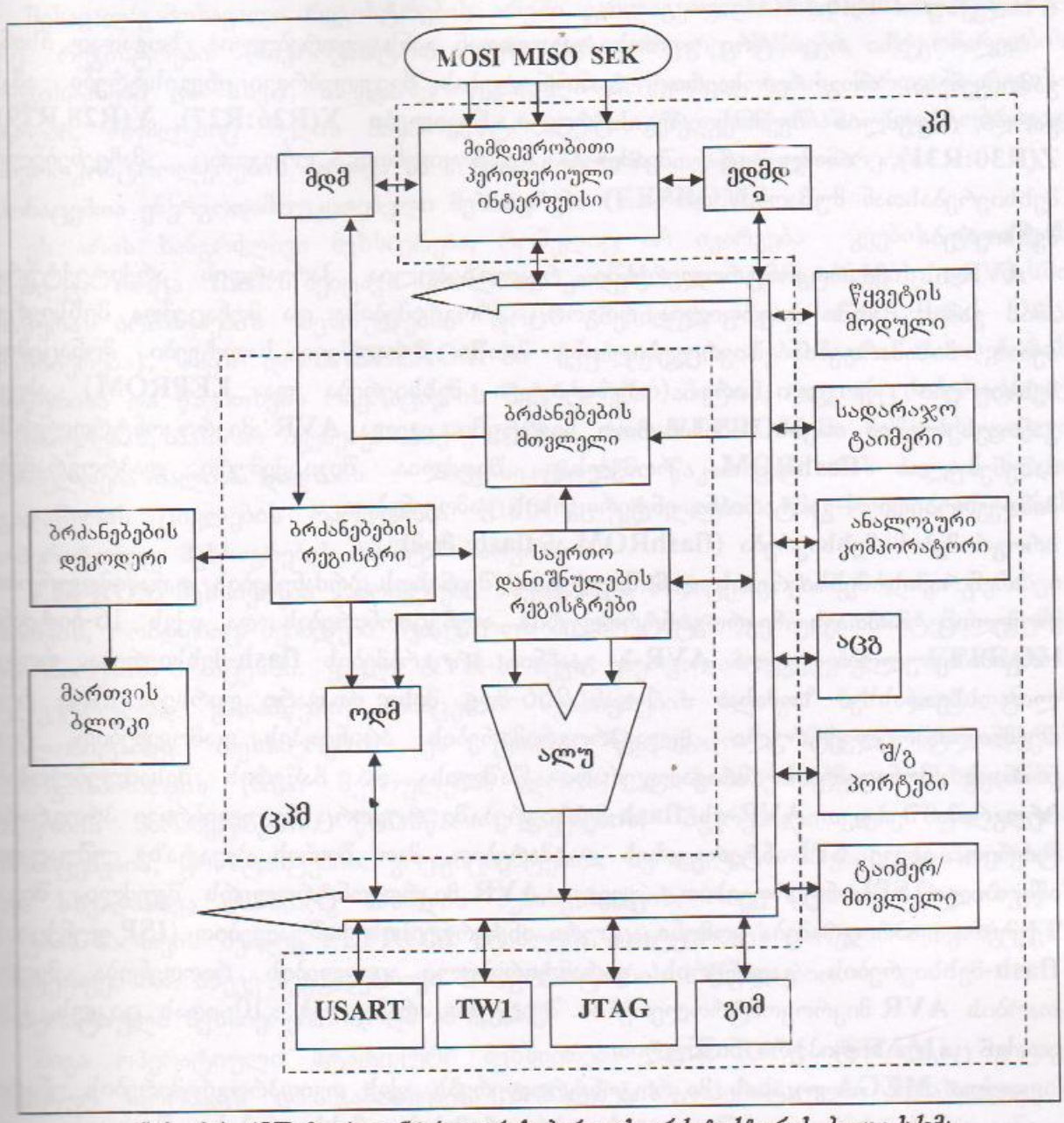
ყველა მათემატიკური ოპერაცია და რიცხვების დამუშავება სრულდება **ALU**-ს მეშვეობით. ეს არის თავისებური კალკულატორი, შეუძლია მიმატება, გამოკლება, შედარება, რიცხვების გადაადგილება სხვადასხვა მეთოდით; ხანდახან გაყოფა და გამრავლება.

განსაკუთრებული არქიტექტურის მეშვეობით (ნახ. 2.1), გამოთვლებში შემგროვებელი ამჯამავის გარდა, მოქმედებაშია 32 თანასწორი მუშა რეგისტრი, რომლებიც დაკავშირებულია არითმეტიკულ-ლოგიკურ მოწყობილობასთან, რაც გამორიცხავს იმის აუცილებლობას, რომ გამოთვლითი ოპერაციების დასრულების შემდეგ მიკროკონტროლერმა მიმართოს დამხმარე რეგისტრებს ან საშუალედო დამხსომებელ მოწყობილობებს.

ALU შეერთებულია საერთო დანიშნულების რეგისტრებთან (General Purpose Registers - **GPR**). სულ საერთო დანიშნულების 32 რეგისტრია, რომლებსაც აქვთ ბაიტური ფორმატი ანუ თითოეული მათგანი შედგება რვა ბიტისაგან. **GPR** იმყოფება ოპერატიული მეხსიერების მისამართის სივრცის დასაწყისში, მაგრამ ფიზიკურად არ ითვლება მის ნაწილად. ამიტომ მათთან ორი წესით შეიძლება შეღწევა: როგორც რეგისტრთან და როგორც მეხსიერებასთან. ასეთი გადაწყვეტილება არის **AVR**-ის განსაკუთრებულობა და ამაღლებს მუშაობის ეფექტურობას და მიკროკონტროლერის წარმადობას.

რეგისტრებსა და ოპერატიულ მეხსიერებას შორის განსხვავება ისაა, რომ რეგისტრებთან შეიძლება აწარმოო ნებისმიერი ოპერაცია (არითმეტიკული, ლოგიკური, ბიტებიანი), ოპერატიულ მეხსიერებაში კი შეიძლება მხოლოდ ჩაიწეროს რეგისტრის მონაცემები.

საერთო დანიშნულების 32 რეგისტრი ქმნის სწრაფი მიწვდომის სარეგისტრო ფაილს, სადაც ყოველი რეგისტრი პირდაპირკავშირშია **ALU**-სთან. სარეგისტრო ფაილიდან ერთი ტაქტით ამოირჩევა ორი ოპერანდი, სრულდება ოპერაცია და შედეგი ბრუნდება სარეგისტრო ფაილში. **ALU** უზრუნველყოფს არითმეტიკული და ლოგიკური ოპერაციების რეგისტრებთან, რეგისტრებსა და კონსტანტებს შორის ან საერთო დანიშნულების რეგისტრებთან. რეგისტრული ფაილი ასევე მისაწვდომია, როგორც მეხსიერების მონაცემების ნაწილი. 32 რეგისტრიდან 6 შეიძლება გამოყენებულ იქნეს, როგორც სამი 16-თანრიგიანი რეგისტრ-მაჩვენებელი ირიბი ადრესაციისათვის. **AVR**-ის ოჯახის უფროსი დონის მიკროკონტროლერების **ALU** შედგენილობაში შედის აპარატული სამრავლებელი.



ნახ. 2.1. AVR მიკროკონტროლერის ბირთვის არქიტექტურის ბლოკ-სქემა

შუალედურ ოპერანდებად გამოიყენება 32 უჯრედი - საერთო დანიშნულების ოპერატიული რეგისტრები **GPR**. შეღწევა სწრაფად ხდება, ხოლო ოპერაციების რიცხვი თავისი შედგენილობით უფრო მდიდარია. ასემბლერში ეს რეგისტრები აღინიშნება მარტივად: **R0,R1,R2 ... R31.** თანაც იყოფა სამ ჯგუფად:

- **უმცროსი R0..R15**

საერთო მოხმარების ჩვეულებრივი რეგისტრები. მათთან არ მუშაობს ბევრი ბრძანება, მაგალითად, უშუალოდ რიცხვის ჩატვირთვა. ე.ი. არ შეიძლება ავიღოთ და მივანიჭოთ რეგისტრს რიცხვი. სამაგიეროდ შეგვიძლია გადმოვიტანოთ ნებისმიერი სხვა რეგისტრიდან "კოპირების" გზით.

- **უფროსი R16..R31**

სრულფასოვანი რეგისტრები, რომლებიც ყველა ბრძანებასთან მუშაობს გამონაკლისის გარეშე.

- **ინდექსური R26...R31**

ბოლო ექვსი რეგისტრი უფროსი ჯგუფიდან განსაკუთრებულია. ზოგადად ისინი გამოიყენება, როგორც საერთო დანიშნულების ჩვეულებრივი რეგისტრები. ამას გარდა, შეუძლია შექმნას რეგისტრული წყვილები **X(R26:R27), Y(R28,R29), Z(R30:R31),** რომლებიც შეიძლება გამოვიყენოთ, როგორც მაჩვენებელი, მეხსიერებასთან მუშაობის დროს.

მეხსიერება

AVR-ის მიკროკონტროლერებში რეალიზებულია ჰარვარდის არქიტექტურა. ამის გამო, მათში დაყოფილია როგორც პროგრამებისა და მონაცემთა მეხსიერებების, მისამართების სივრცეები, ისე მათში შესაღწევი სალტეები. მონაცემთა მეხსიერების ყოველი არე (ოპერატიული მეხსიერება და **EEPROM**) ასევე განთავსებულია თავის სამისამართო სივრცეში. ყველა **AVR** მიკროკონტროლერში ჩაშენებულია **flashROM,** რომელსაც შეუძლია შიგასქემური დაპროგრამება მიმდევრობითი 4-გამტარიანი ინტერფეისის გამოყენებით.

პროგრამების მეხსიერება (flashROM ან flash მდმ)

პროგრამის მეხსიერების დანიშნულებაა, შეინახოს ბრძანებების თანამიმდევრობა, რომელიც მართავს მიკროკონტროლერის ფუნქციონირებას და აქვს 16-ბიტიანი ორგანიზებულობა. ყველა **AVR**-ს გააჩნია პროგრამების **flash**-მეხსიერება, რომელიც სხვადასხვა ზომისაა – 1-დან 256-მდე. მისი მთავარი ღირსება ისაა, რომ შექმნილია ელექტრული კვლავპროგრამირების პრინციპის გამოყენებით, რაც ნიშნავს მონაცემების მრავალჯერადი წაშლისა და ჩაწერის შესაძლებლობას. პროგრამა შეჰყავთ **AVR**-ის **flash**-მეხსიერებაში როგორც ჩვეულებრივი პროგრამატორით, ასევე **SPI**-ინტერფეისის დახმარებით, მათ შორის პლატაზე უშუალოდ აწყობილი **SPI**-ინტერფეისით. ყველა **AVR** მიკროკონტროლერს შეუძლია შიგასქემური დაპროგრამება კომუნიკაციური ინტერფეისის საშუალებით (**ISP** ფუნქცია). **flash**-მეხსიერების გადაწერის გარანტირებული ციკლების რაოდენობა მეორე თაობის **AVR** მიკროკონტროლერებში შეადგენს არანაკლებ 10 ათას ციკლს, 100 ათასი ციკლის ტიპური ნიშნულით.

ყველა **MEGA** ოჯახის მიკროკონტროლერებს აქვს თვითპროგრამირების უნარი, ე.ი. დამოუკიდებლად შეცვალოს პროგრამების მეხსიერების შიგთავსი. ეს განსაკუთრებულობა საშუალებას იძლევა შეიქმნას მოქნილი სისტემები, რომელთა მუშაობის ალგორითმი იცვლება და რომელიც დამოკიდებულია სხვადასხვა შიგა პირობებსა და გარე მოვლენებზე.

მონაცემთა მეხსიერება

მონაცემთა მეხსიერება დაყოფილია 3 ნაწილად: რეგისტრული მეხსიერება, ოპერატიული მეხსიერება (**ოდმ** ოპერატიული დამხსომებელი მოწყობილობა ან **RAM**) და ენერგოდამოუკიდებელი მეხსიერება (**EEPROM**).

რეგისტრული მეხსიერება

რეგისტრულ მეხსიერებაში ჩართულია ფაილში გაერთიანებული, საერთო დანიშნულების 32 რეგისტრი (**სდრ** ან **GPR**), გაერთიანებული ფაილში - შესავალ/გამოსავალი სამომხმარებლო რეგისტრი (**შგრ**). ისინი განლაგებულია **ოდმ**-ის სამისამართო სივრცეში, მაგრამ არ წარმოადგენს მის ნაწილს.

შესავალ/გამოსავალი რეგისტრების არეში განლაგებულია სხვადასხვა მომსახურე რეგისტრები (მიკროკონტროლერის მართვის რეგისტრები, მდგომარეობის რეგისტრები და სხვ.), ასევე პერიფერიული მოწყობილობების მართვის რეგისტრები, რომლებიც შედის მიკროკონტროლერის შედგენილობაში. ფაქტობრივად მიკროკონტროლერების მართვა ამ რეგისტრების მართვაში მდგომარეობს.

მონაცემთა ენერგოდამოუკიდებელი მეხსიერება (EEPROM)

ეს არის ხანგრძლივი მეხსიერება, რომელიც არ იკარგება კვების გათიშვის შემდეგ. როცა **flash** შეიცავს მხოლოდ კოდებსა და კონსტანტებს და მასში ჩაწერა ბრძანებების შესრულების დროს შეიძლება (ეს არის Read Only მეხსიერება), ასეთ დროს **EEPROM**-ში შეუძლებელია ნებისმიერი რაოდენობის ჩაწერისა და წაკითხვის ოპერაციების შესრულება, ამიტომ, როგორც ოპერატიულ მეხსიერებას, მას არ იყენებენ. საქმე იმაშია, რომ **EEPROM**-ში ჩაწერის ციკლი გრძელდება ძალიან დიდხანს — მილისეკუნდებში. წაკითხვაც არის ძალიან ნელი. გადაწერის ციკლების რაოდენობა 100000 უტოლდება, რაც დიდი არ არის ოპერატიული მეხსიერების მასშტაბისათვის.

EEPROM-მეხსიერება გამოიყენება სხვადასხვა მონაცემთა ხანგრძლივი შენახვისათვის, რომლებიც შეიძლება შეიცვალოს მიკროკონტროლერული სისტემის ფუნქციონირების პროცესში. ყველა **AVR**-ს აქვს ენერგოდამოუკიდებელი **EEPROM,** ელექტრულად გადამწერი მონაცემთა მეხსიერების ბლოკი 64 ბაიტიდან 4 კილობიტამდე. მეხსიერების ეს ტიპი "წვდომადია" მიკროკონტროლერის პროგრამისათვის (მისი შესრულების დროს) და მოსახერხებელია სხვადასხვა აწყობის პარამეტრების, წინასწარი დაყენების, შეკრებილი და შუალედური მონაცემების, კონსტანტების, კოეფიციენტების, სერიული ნომრების, გასაღებებისა და სხვადასხვა დამხმარე მასალის შენახვისათვის, რაც საჭიროა წაკითხვისას კვების ჩართვის შემდეგ. **EEPROM** შეიძლება დაიტვირთოს გარედან როგორც **SPI** ინტერფეისით, ასევე ჩვეულებრივი პროგრამატორის საშუალებით.

ოპერატიული მეხსიერება (ოდმ ან RAM)

შიგა ოპერატიული სტატიკური მეხსიერება Static RAM (**SRAM**) შეიცავს ბაიტურ ფორმატს და გამოიყენება მონაცემების ოპერატიული შენახვისათვის. ჩიპებში ოპერატიული მეხსიერების მოცულობა იცვლება 64 ბაიტიდან 4 კილობაიტამდე. ამ უჯრედებში ინახება ნებისმიერი მონაცემი, მასთან შეღწევა ხორციელდება Load და Store ბრძანებებით, ე.ი. არ შეიძლება მეხსიერების უჯრედს დავამატოთ ერთეული. თავდაპირველად უნდა შესრულდეს ოპერაცია Load **ოდმ**-იდან **სდრ**-ში, შემდეგ რეგისტრში დავამატოთ ერთეული და Store ოპერაციით დავაბრუნოთ მეხსიერებაში. **RAM**-ში წაკითხვისა და ჩაწერის ციკლების რაოდენობა შეუზღუდავია, მაგრამ მკვებავი ძაბვის გათიშვის შემთხვევაში მთელი მონაცემი იკარგება. ზოგიერთი მიკროკონტროლერისათვის შესაძლებელია 64 კილობაიტამდე გარე სტატიკური **ოდმ**-ის მიერთების ორგანიზება.

2.1.2. AVR მიკროკონტროლერის პერიფერიის არქიტექტურა

პერიფერიული მოწყობილობები (პმ) არის მიკროკონტროლერის შიგა "ასორტიმენტი", რომელიც მის უნივერსალობას განაპირობებს. თუ **ALU-ს, RAM-ის, FLASH-ის** და მართვის ბლოკის გარეშე მიკროკონტროლერი არ ჩაირთვება,

პერიფერია უკვე ოპციაა, რომელიც განსხვავებული სირთულის ხარისხისაა, ასევე რიგითობის მიხედვით კომბინირებული.

სწორედ კრისტალზე ამა თუ იმ პერიფერიის არსებობის მიხედვით ხდება მიკროკონტროლერის შერჩევა დავალების შესასრულებლად.

AVR მიკროკონტროლერის პერიფერია შედგება ტაიმერ-მრიცხველისაგან, განედურ-იმპულსური მოდულატორისაგან, გარე წყვეტის მხარდაჭერისაგან, ანალოგური კომპარატორებისაგან, 10-თანრიგიანი 8-არხიანი **აცგ**-საგან, პარალელური პორტებისაგან (3-დან 48-მდე შესავალ/გამოსავალი ხაზით), **UART, JTAG** და **SPI** ინტერფეისებისაგან, დარაჯი ტაიმერისა და ჩამოყრის კვანძისაგან, შემცირებული კვების ფიქსირებისათვის. ყველა ეს მოწყობილობა **AVR** მიკროკონტროლერს გადააქცევს ძლიერ ინსტრუმენტად თანამედროვე, ეკონომიკური და მაღალმწარმოებლური სხვადასხვა დანიშნულების კონტროლისა და მართვის მოწყობილობების შესაქმნელად.

ქვემოთ მოყვანილია პერიფერიების ნაკრები, რომელიც თითქმის ყველა **AVR**-ში და თანამედროვე კონტროლერებშია.

შესავალ/გამოსავალი პორტები (I/O)

შესავალ/გამოსავალი პორტების გარეშე შეუძლებელია კონტროლერის ურთიერთქმედება გარე სამყაროსთან. უშუალოდ პორტები უზრუნველყოფს იმ "შემავალ ვარიაციებს", რომლებიც სქემის სხვა ელემენტებს მართავს. მიკროკონტროლერის თითქმის ყველა გამომყვანს შეუძლია იმუშაოს შესავალ/გამოსავალი პორტების რეჟიმში.

AVR-ის შესავალ/გამოსავალი პორტების დამოუკიდებელი შესავალ/გამოსავალი ხაზების რაოდენობა 3-დან 53-მდეა. პორტის ყველა ხაზი შეიძლება იყოს დაპროგრამებული შესვლაზე ან გამოსვლაზე. ძალური გამოსავალი დრაივერები უზრუნველყოფს **20მა** დენით დატვირთვის შესაძლებლობას პორტის ხაზზე (შედინების დენი) მაქსიმალური ნიშნულით **40მა**, რაც, მაგალითად, საშუალებას იძლევა მიკროკონტროლერზე უშუალოდ ჩაერთოს შუქდიოდები და ბიპოლარული ტრანზისტორები. დენის საერთო დატვირთვა ერთი პორტის ყველა ხაზზე **80მა** არ უნდა აღემატებოდეს (ყველა მნიშვნელობა მოცემულია **5ვ** კვების ძაბვისათვის).

AVR-ის შესავალ/გამოსავალი პორტების არქიტექტურული განსაკუთრებულობა იმაში მდგომარეობს, რომ ყველა ფიზიკური გამოსასვლელისათვის (პინისათვის) არსებობს კონტროლ/მართვის 3 ბიტი. ამის გამო, აღარ არის აუცილებელი მეხსიერებაში არსებული პორტის შიგთავსის ასლის არსებობა, რითაც იზრდება მიკროკონტროლერის მუშაობის სიჩქარე გარე ხელსაწყოებთან, განსაკუთრებით გარე ელექტრული დაბრკოლებების შემთხვევაში.

წყვეტა (INTERRUPTS)

წყვეტის სისტემა – მიკროკონტროლერების ერთ-ერთი უმნიშვნელოვანესი ნაწილია. ყველა **AVR** მიკროკონტროლერი შეიცავს მრავალდონიანი წყვეტის სისტემას. წყვეტა აჩერებს პროგრამის ნორმალურ მსვლელობას პრიორიტეტული დავალებების შესრულებისათვის, რომელიც განისაზღვრება შიგა და გარე მოვლენებით.

ყველა ასეთი მოვლენისათვის მუშავდება ცალკეული პროგრამა, რომელსაც წყვეტის შეკითხვის დამუშავების ქვეპროგრამას უწოდებენ (შემოკლებულად წყვეტის ქვეპროგრამა) და ათავსებენ პროგრამების მეხსიერებაში.

წყვეტის გამომწვევი მოვლენების წარმოშობისას მიკროკონტროლერი ინახავს ბრძანების მრიცხველის შიგთავსს, წყვეტს ცენტრალური პროცესორის შესასრულებელ მიმდინარე პროგრამას და გადადის წყვეტის პროგრამის შესრულებაზე.

წყვეტის ქვეპროგრამის შესრულების შემდეგ ხორციელდება მრიცხველის მიერ წინასწარშენახული ბრძანების აღდგენა და პროცესორი უბრუნდება შეწყვეტილი პროგრამის შესრულებას.

თითოეული მოვლენისათვის შეიძლება დაწესდეს პრიორიტეტი. პრიორიტეტის არსი იმაში მდგომარეობს, რომ შესასრულებელი წყვეტის პროგრამა შეიძლება შეწყდეს სხვა მოვლენებით იმ შემთხვევაში, თუ მას აქვს უფრო მაღალი პრიორიტეტი, ვიდრე მიმდინარეს. წინააღმდეგ შემთხვევაში, ცენტრალური პროცესორი გადადის ახალი მოვლენების დამუშავებაზე, წინა დამუშავების შემდეგ.

ტაიმერ/მრიცხველები (TIMER/COUNTERS)

ტაიმერ/მრიცხველების ამოცანა – "წიკწიკის" ათვლაა. მაგალითად, თუ დავავალებთ დათვალოს პროცესორის 100 ტაქტი, იწყებს თვლას და როგორც კი დაითვლის სიგნალს იძლევა. მისი საშუალებით შეიძლება განისაზღვროს შემოსული სიგნალების ხანგრძლივობა და შემოსული იმპულსების რაოდენობა.

AVR მიკროკონტროლერების შედგენილობაში შედის 1-დან 4-მდე ტაიმერ/მრიცხველი 8 ან 16-ბიტიანი თანრიგით, რომლებიც მუშაობს როგორც ტაიმერად ტაქტური სიხშირის შიგა წყაროდან, ასევე გარე პროცესების აღმრიცხველად.

ტაიმერ/მრიცხველი შეიძლება გამოვიყენოთ დროის ინტერვალების ზუსტი ფორმირებისათვის, მიკროკონტროლერის გამოსასვლელზე იმპულსების დასათვლელად, იმპულსების თანამიმდევრობითი ფორმირებისათვის, არხის მიმღებ/გადამცემის მიმდევრობითი კავშირის არხის ტაქტირებისათვის. განედურ-იმპულსური მოდულაციის (**გიმ** ან **PWM**) რეჟიმში ტაიმერ/მრიცხველი არის განედურ-იმპულსური მოდულატორი და გამოიყენება სიგნალის გენერირებისათვის დაპროგრამებული სიხშირით და სიმეჩხრით. ტაიმერ-მრიცხველს შეუძლია აწარმოოს მოთხოვნა წყვეტაზე, პროცესორის გადართვით მათ მომსახურებაზე, მოვლენების მიხედვით და ათავისუფლებს აუცილებლობისაგან, რომ პერიოდულად მოხდეს შეკითხვა ტაიმერის მდგომარეობაზე. ვინაიდან მიკროკონტროლერი გამოიყენება რეალური დროის სისტემებში, ტაიმერ-მრიცხველი წარმოადგენს მათ ერთ-ერთ უმნიშვნელოვანეს ელემენტს.

გუშაგი ტაიმერი (WDT)

გუშაგი ტაიმერი (WatchDog Timer) გამოიყენება პროგრამის შეფერხებისას კატასტროფული შედეგის თავიდან ასაცილებლად. აქვს საკუთარი **RC**-გენერატორი, რომელიც მუშაობს 1მეგაჰერც სიხშირეზე. **RC**-გენერატორისთვის 1მეგაჰერცის ნიშნული მიახლოებითია და, პირველ რიგში, დამოკიდებულია ტემპერატურასა და კვების ძაბვის სიდიდეზე. გუშაგი ტაიმერის გამოყენების იდეა შედგება მის რეგულარულ ჩამოყრაში პროგრამის მართვის ან გარე ზემოქმედების

ქვეშ, სანამ არ დამთავრდება მისი დროის შეყოვნება და არ მოხდება პროცესორის ჩამოყრა. თუ პროგრამა ნორმალურად მუშაობს, მაშინ გუშაგი ტაიმერის ბრძანება ჩამოყრაზე სრულდება რეგულარულად, რითაც იცავს პროცესორს ჩამოყრისაგან. თუ მიკროპროცესორი შემთხვევით გამოვიდა პროგრამის ფარგლებიდან (მაგალითად, ძლიერი დამახინჯების შედეგად კვების წრედში) ან ჩაიციკლა პროგრამის რომელიღაც მონაკვეთზე, გუშაგი ტაიმერის ბრძანება ჩამოყრაზე არ შესრულდება გარკვეული დროის განმავლობაში, მოხდება პროცესორის მთლიანი ჩამოყრა და ყველა რეგისტრის ინიციალიზაცია, რასაც სისტემა მოჰყავს მუშა მდგომარეობაში.

ანალოგური კომპარატორი (AC)

ანალოგური კომპარატორი არის კიდევ ერთი ანალოგური ინტერფეისი, მაგრამ **აცგ**-სგან განსხვავებით ის არ ზომავს, არამედ ადარებს ორ ანალოგურ სიგნალს და იძლევა შედეგს **A>B** ან **A<B** ორობითი სახით.

ანალოგური კომპარატორი (Analog Comparator) ახდენს ძაბვების შედარებას მიკროკონტროლერის ორ გამოსასვლელზე (პინზე). შედარების შედეგი მიიღება ლოგიკური მნიშვნელობით, რომელიც შეიძლება წაკითხულ იქნეს პროგრამიდან.

ანალოგური კომპარატორის გამოსასვლელი შეიძლება ჩაირთოს წყვეტაზე ანალოგური კომპარატორიდან. მომხმარებელს შეუძლია დააყენოს წყვეტის ამუშავება სიგნალის მზარდი ან კლებადი ფრონტის დროს ან მისი გადართვით.

ანალოგურ/ციფრული გარდამქმნელი (A/D converter)

აცგ-ს ანალოგური შესასვლელი ყველა მიკროკონტროლერს არ აქვს, ის აუცილებელია იმ შემთხვევაში, როდესაც საჭიროა ანალოგური სიგნალის მიღება და გაზომვა. **აცგ** გამოიყენება იმისათვის, რომ მიღებულ იქნეს მის შესასვლელზე მიწოდებული ძაბვის ციფრული მონაცემები. ეს შედეგი ინახება **აცგ**-ს მონაცემთა რეგისტრში. მიკროკონტროლერის რომელი გამოსასვლელი (პინები) იქნება **აცგ**-სთვის შესასვლელი განისაზღვრება შესაბამის რეგისტრში შეტანილი რიცხვით.

უნივერსალური თანამიმდევრობითი მიმღებ/გადამცემი (UART an USART)

UART/USART მიმღებ/გადამცემი მიმდევრობითი პორტია და მუშაობს ასინქრონული პროტოკოლით. ესაა მარტივი და საიმედო პროტოკოლი, რომელიც გამოიყენება კომპიუტერებსა და მიკროკონტროლერებთან კავშირისათვის.

უნივერსალური ასინქრონული ან უნივერსალური სინქრონულ-ასინქრონული მიმღებ/გადამცემი (Universal Synchronous/Asynchronous Receiver and Transmitter - **UART** ან **USART**) მოხერხებული და მარტივი მიმდევრობითი ინტერფეისია, რომელიც ახდენს მიკროკონტროლერის გარე სამყაროსთან ურთიერთობის ორგანიზებას. შეუძლია იმუშაოს დუპლექსის რეჟიმში (მონაცემების ერთდროული მიღება და გადაცემა) და **RS-232** სტანდარტულ პროტოკოლზე, რაც უზრუნველყოფს კავშირს პერსონალურ კომპიუტერთან. მიკროკონტროლერის და კომპიუტერის დასაკავშირებლად აუცილებელია სიგნალების დონეების შეთავსება. ამისათვის გამოიყენება სპეციალური მიკროსქემები. მაგალითად, **MAX232**.

მიმდევრობითი პერიფერიული ინტერფეისი SPI

SPI კიდევ ერთი მიმდევრობითი პროტოკოლია, რომელიც **IIC** წააგავს, მაგრამ არ შეუძლია ქსელური ორგანიზება. მუშაობს მხოლოდ წამყვან-ამყოლის **(Master-Slave)** რეჟიმში და ძალიან სწრაფია.

მიმდევრობითი პერიფერიული 3-გამტარიანი **SPI** ინტერფეისის (Serial Peripheral Interface) დანიშნულებაა ორ მოწყობილობას შორის მონაცემების გაცვლის ორგანიზება. მისი დახმარებით შეიძლება ძოხდეს მონაცემების გაცვლა მიკროკონტროლერსა და სხვადასხვა მოწყობილობას შორის. ასეთი მოწყობილობებია: ციფრული პოტენციომეტრი, **ცაგ/აცგ**, **flash-მდმ** და სხვ. ამ ინტერფეისის დახმარებით მოსახერხებელია მონაცემების გაცვლა რამდენიმე **AVR** მიკროკონტროლერს შორის. ამას გარდა, **SPI** ინტერფეისის გავლით შეიძლება განხორციელდეს მიკროკონტროლერების დაპროგრამება.

ორგამტარიანი მიმდევრობითი ინტერფეისი TWI

I2C(TWI) ინტერფეისი არის მიმდევრობითი სალტე **IIC**. მისი გავლით ხორციელდება კავშირი სხვა მოწყობილობებთან. ერთი მოწყობილობის ფარგლებში მიკროკონტროლერებით **IIC**-ზე შეიძლება თავისებური ლოკალური ქსელის ორგანიზება.

ორგამტარიანი მიმდევრობითი ინტერფეისი **TWI** (Two-wire Serial Interface) საშუალებას გვაძლევს ერთდროულად გავაერთიანოთ 128-მდე სხვადასხვა მოწყობილობა ორმიმართულებიანი სალტის დახმარებით, რომელიც შედგება ტაქტური სიგნალის ხაზისა (**SCL**) და მონაცემების ხაზისაგან (**SDA**).

ინტერფეისი JTAG

JTAG/DebugWire გაწყობის საშუალებაა, რომელიც ნებას იძლევა ჩავიხედოთ მიკროკონტროლერის "ტვინში" სპეციალური ადაპტერის საშუალებით. ინტერფეისი **JTAG** დამუშავდა სპეციალისტების ჯგუფის მიერ, რომელიც იკვლევდა ელექტრონული კომპონენტების ტესტირების პრობლემებს (Joint Test Action Group) და დარეგისტრირდა, როგორც საწარმოო სტანდარტი. ოთხგამტარიანი ინტერფეისი **JTAG** გამოიყენება ნაბეჭდი პლატების ტესტირებისათვის, შიგასქემური გაწყობისა და მიკროკონტროლერების დაპროგრამებისათვის.

Mega ოჯახის ბევრ მიკროკონტროლერს მიეკუთვნება IEEE Std 1149.1-თან თავსებადი ინტერფეისი **JTAG** ან **debugWIRE** ჩაშენებული გაწყობისათვის. ამას გარდა, ყველა **Mega** მიკროკონტროლერი 16კბაიტი და მეტი **FLASH** მეხსიერების მოცულობით შეიძლება დაპროგრამდეს **JTAG** ინტერფეისის გავლით.

ტაქტური გენერატორი

ტაქტური გენერატორი გამოიმუშავებს იმპულსებს მიკროკონტროლერის ყველა კვანძის სინქრონული მუშაობისათვის. **AVR**-ის შიგა ტაქტური გენერატორი შეიძლება გაიშვას საყრდენი სიხშირის რამდენიმე წყაროდან: გარე გენერატორი, გარე კვარცული რეზონატორი, შიგა ან გარე **RC**-ჯაჭვი. მინიმალური დასაშვები სიხშირე არ იზღუდება (ბიჯურ რეჟიმამდეც). მაქსიმალური სამუშაო სიხშირე განისაზღვრება მიკროკონტროლერის კონკრეტული ტიპის მიხედვით და მითითებულია მის მახასიათებლებში. პრაქტიკულად ყველა **AVR**-მიკროკონტროლერის მოცემული სამუშაო სიხშირე, მაგალითად, 10მეგაჰერცი ოთახის ტემპერატურაზე, შეიძლება აიწიოს 12 მეგაჰერცამდე და მეტიც.

რეალური დროის სისტემა (RTC)

RTC რეალიზებულია ყველა **Mega** მიკროკონტროლერში. **RTC**-ის ტაიმერ/მრიცხველს აქვს ცალკე წინაგამყოფი, რომელიც პროგრამული

საშუალებით შეიძლება მიუერთდეს ძირითადი ტაქტური სიხშირის არხს ან საყრდენი სიხშირის დამატებით ასინქრონულ წყაროს (კვარცული რეზონატორი ან გარე სინქროსიგნალი). ამ მიზნისათვის დარეზერვებულია მიკროსქემის ორი გამოსასვლელი. შიგა ოსცილატორი გათვლილია 32,768 კილოჰერცი გარე "საათობრივ" კვარცულ რეზონატორთან სამუშაოდ.

PWM (გიმ-გენერატორი)

ეს არ არის ცალკე ბლოკი, იგი ასრულებს ტაიმერის დამატებით სასარგებლო ფუნქციას. **გიმ**-გენერატორის დახმარებით ადვილია ანალოგური სიგნალის ფორმირება, რომლის მეშვეობით შესაძლებელია, მაგალითად, შეცვალოთ შუქდიოდების გამოსხივების სიკაშკაშე ან ძრავას ბრუნვის სიჩქარე. **გიმ**-ის არხების რიცხვი სხვადასხვაა კონტროლერში სხვადასხვაა.

პერიფერიაში დამატებით ჩაშენებულია **USB**, **Ethernet** ინტერფეისი, რეალური დროის საათი, **LCD** დისპლეების კონტროლერები და სხვ.

ბირთვის ურთიერთქმედება პერიფერიასთან

ბირთვი საერთოა ოჯახის ყველა მიკროკონტროლერისათვის, პერიფერია კი — სხვადასხვა. მათ შორის ურთიერთობა მეხსიერების საშუალებით ხდება, ე.ი. პერიფერიას აქვს თავისი მეხსიერების უჯრედები — პერიფერიების რეგისტრები. ყოველ პერიფერიულ მოწყობილობას რამდენიმე რეგისტრი აქვს. ამ რეგისტრებში იმყოფება კონფიგურაციის ბიტები. იმის მიხედვით, თუ როგორ არის ეს ბიტები დაყენებული, იმავე რეჟიმში მუშაობს პერიფერიული მოწყობილობა. ამავე რეგისტრებში საჭიროა ჩაიწეროს ის მონაცემები, რომლებიც მიკროკონტროლერმა უნდა გასცეს, მაგალითად, თანამიმდევრულ პორტზე ან ამოიკითხოს მონაცემები, რომლებიც დაამუშავა **აცგ**-მა. პერიფერიებთან სამუშაოდ არის სპეცბრძანებები **IN** და **OUT** პერიფერიებიდან წასაკითხად **სდრ** რეგისტრში და ჩაწერა **სდრ**-დან შესაბამის პერიფერიაში.

იმის გამო, რომ ბირთვი ერთი და იგივეა, ხოლო პერიფერია სხვადასხვა, სხვა მოდელის მიკროკონტროლერზე კოდის გადატანისას საჭიროა მხოლოდ შესწორდეს ეს მიმართვები, ვინაიდან პერიფერიული რეგისტრების დასახელება მოდელიდან მოდელამდე შეიძლება განსხვავდებოდეს. მაგალითად, თუ კონტროლერში არის ერთი **UART** მიმღებ/გადამცემი, მაშინ მონაცემთა მიღების რეგისტრს ეწოდება **UDR**, ხოლო თუ ორია, მაშინ გვაქვს **UDR0** და **UDR1**. მაგრამ ჯამში ყველაფერი გამჭირვალე და ლოგიკურია და, როგორც წესი, კოდის პორტირება ერთი მიკროკონტროლერიდან მეორეზე, თუნდაც დაწერილი იყოს ასემბლერზე, დიდ სირთულეს არ წარმოადგენს, განსაკუთრებით მაშინ, თუ სწორადაა.

AVR-მიკროკონტროლერები უზრუნველყოფს "მძინარე" და მიკრომოხმარების რეჟიმების მხარდაჭერას. "მძინარე" რეჟიმში ჩერდება ცენტრალური პროცესორული ბირთვი, მაგრამ რეგისტრები, ტაიმერ/მრიცხველები, გუშაგი ტაიმერი და წყვეტის სისტემა აგრძელებს ფუნქციონირებას. მიკრომოხმარების რეჟიმში ინახება ყველა რეგისტრის შემადგენელი შიგთავსი, ჩერდება ტაქტური გენერატორი, იკრძალება მიკროკონტროლერის ყველა ფუნქცია, სანამ გარე წყვეტის ან აპარატული ჩამოყრის სიგნალი არ მოვა. მოდელების მიხედვით **AVR**-მიკროკონტროლერები მუშაობს 2.7–6ვ ან 4–6ვ ძაბვის დიაპაზონში.

2.2. ATMega48/ATMega88/ATMega168 მიკროკონტროლერები

როგორც ყველა **AVR** მიკროკონტროლერი, **Mega** ოჯახის წარმომადგენელი მიკროკონტროლერები 8-თანრიგიანია და მიმართულია ჩაშენებული გამოყენებისათვის. მათ აქვთ ელექტრულად წაშლადი პროგრამების მეხსიერება (**FLASH**) და მონაცემების მეხსიერება (**EEPROM**), აგრეთვე სხვადასხვა პერიფერიული მოწყობილობები. მზადდება მცირე ელექტრომოხმარების ლითონი-ჟანგეული-ნახევარგამტარი კომპლემენტარული სტრუქტურის ტექნოლოგიით, რომელიც გაუმჯობესებულ **RISC** არქიტექტურასთან ურთიერთობისას საშუალებას იძლევა მიღწეულ იქნეს საუკეთესო შეფარდება – სწრაფქმედება/ენერგომოხმარება.

Mega ოჯახის მიკროკონტროლერები **AVR**-ის შედგენილობაში ყველაზე სრულყოფილად ითვლება. ქვემოთ მოყვანილია **AVR** მიკროკონტროლერის **ATmega88** მოდელის პარამეტრების ცხრილი.

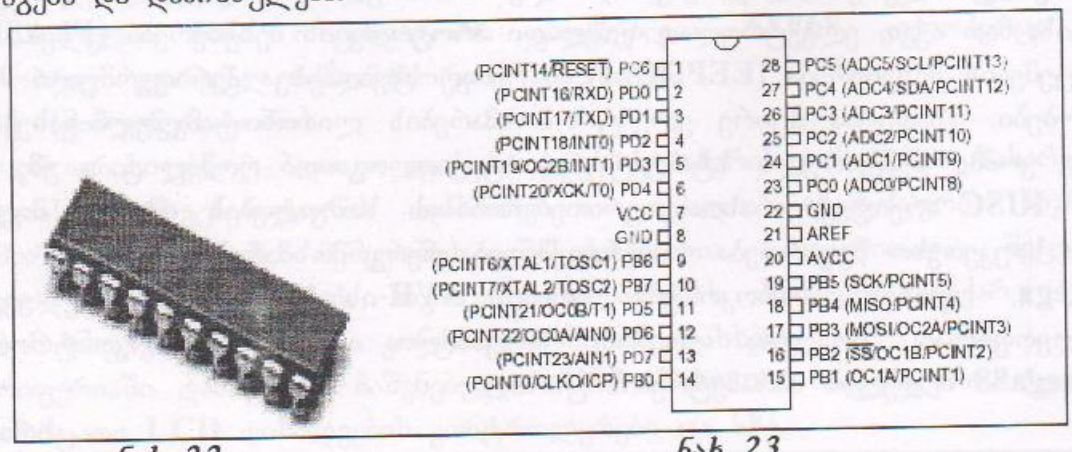
ცხრილი 4

Device ATmega88															
Flash ROM	*EEPROM*	*RAM*	*Max I/O*	*F.max*	*Vcc*	*16-bit timer*	*8-bit timer*	*PWM*	*RTC*	*SPI*	*UART*	*TWI*	*AD*	*Int.*	*Ext Int.*	*Package*
8	0,5	1024	23	20	1.8-5.5	1	2	3	Yes	1+USART	1	Yes	8	26	26	MLF 32 PDIP 28 TQFP 32

ცხრილის განმარტება

- **Flash ROM** –პროგრამების ენერგოდამოუკიდებელი მეხსიერების მოცულობა (კ/ბაიტებში);
- **EEPROM** – მონაცემთა ენერგოდამოუკიდებელი მეხსიერების მოცულობა (ბაიტებში);
- **RAM** – მონაცემთა სტატიკური მეხსიერების მოცულობა (ბაიტებში);
- **Max I/O** –შესავალ/გამოსავალი მისაწვდომი ხაზების მაქსიმალური რაოდენობა;
- **F. max** –მაქსიმალური სიხშირე;
- **Vcc** – კვების სამუშაო ძაბვების დიაპაზონი (ვოლტებში);
- **Timer(s) 8/16 bit** – ტაიმერ/მრიცხველების რაოდენობა და თანრიგი;
- **PWM** – განედურ-იმპულსური მოდულაციის დამოუკიდებელი არხების რაოდენობა;
- **RTC** –რეალური დროის სისტემა;
- **SPI** –სინქრონული სამგამტარიანი მიმდევრობითი ინტერფეისი;
- **UART** –ასინქრონული მიმდევრობითი მიმღებ/გადამცემი;
- **TWI** –ორგამტარიანი მიმდევრობითი ინტერფეისი;
- **AD** – ანალოგურ-ციფრული გარდამქმნელის არხების რაოდენობა;
- **Int.** – წყვეტები (interrupts);
- **Ext. Int.** –გარე წყვეტები;
- **Package** – კორპუსების ტიპები, რომელშიც იწნეხება მიკროკონტროლერი და მისი გამომყვანების საერთო რაოდენობა.

2.2 ნახაზზე მოცემულია **TMega88**ის კორპუსის ტიპი **PDIP** (Plastic Dual. In-Line Package). 2.3 ნახაზზე მოცემულია **ATMega88**ის გამოსასვლელების განლაგება და დანიშნულება.



ნახ. 2.2 *ნახ. 2.3*

კვება

AVR ფუნქციონირებს 1.8-დან 6.0ვ კვების ძაბვის ფარგლებში. აქტიურ რეჟიმში მოხმარებული დენი დამოკიდებულია კვების ძაბვის სიდიდესა და იმ სიხშირეზე, რაზეც მუშაობს მიკროკონტროლერი და შეადგენს: 1მა-ს 500 კილოჰერცისათვის, 5...6მა-ს 5 მეგაჰერცისათვის და 8...9მა-ს 12 მეგაჰერცი სიხშირისათვის.

AVR შეიძლება პროგრამული გზით გადავიყვანოთ სამიდან ერთ-ერთ დაბალ ენერგომოხმარების რეჟიმზე.

უქმი სვლის მოხმარების რეჟიმი (IDLE) – შეწყდება მხოლოდ პროცესორის მუშაობა და ფიქსირდება მონაცემთა მეხსიერების შიგთავსი, ხოლო სინქროსიგნალების შიგა გენერატორი, ტაიმერები, შეწყვეტის სისტემა და გუშაგი ტაიმერი აგრძელებს ფუნქციონირებას. მოხმარებული დენი არ აღემატება 2,5მა-ს და 12 მეგაჰერც სიხშირეს.

გაჩერების რეჟიმი (POWER DOWN) - ინახება სარეგისტრაციო ფაილის შიგთავსი, მაგრამ ჩერდება სინქროსიგნალების შიგა გენერატორი და, აქედან გამომდინარე, ჩერდება ყველა ფუნქცია, სანამ არ მოვა გარე წყვეტის ან აპარატული ჩამოყრის სიგნალი. თუ ჩართულია გუშაგი ტაიმერი, მოხმარებული დენი ამ რეჟიმში შეადგენს დაახლოებით 80 მიკროამპერს, ხოლო გამორთულზე - 1 მიკროამპერზე ნაკლებს (ყველა მოცემული მონაცემი სწორია 5ვ კვების ძაბვისათვის).

ეკონომიური რეჟიმი (POWER SAVE) - მუშაობს მხოლოდ ტაიმერის გენერატორი, რაც უზრუნველყოფს დროის ბაზის შენახვას. ყველა დანარჩენი ფუნქცია გათიშულია.

ჩამოყრა კვების ძაბვის დაწევის შემთხვევაში (BOD)

სქემა **BOD** (Brown-Out Detection) თვალყურს ადევნებს კვების წყაროს ძაბვას. თუ სქემა ჩართულია, მაშინ გარკვეულ ნიშნულამდე კვების ძაბვის დაწევისას ამ სქემას მიკროკონტროლერი გადაჰყავს ჩამოყრის მდგომარეობაში. როდესაც კვების ძაბვა ისევ გაიზრდება საჭირო ნიშნულამდე, ჩაირთვება ჩამოყრის დაყოვნების ტაიმერი. დროის გასვლის შემდეგ ჩამოყრის შიგა სიგნალი წყდება და ხდება მიკროკონტროლერის გაშვება.

2.2.1. Mega ოჯახის პარამეტრები

ATMega48/ATMega88/ATMega168 –მცირე ენერგიის მომხმარებელი 8-ბიტიანი ლითონი-ჟანგეული-ნახევარგამტარი კომპლემენტარული სტრუქტურის მიკროკონტროლერები, **AVR RISC** არქიტექტურით და 4/8/16კილობაიტიანი შიგასისტემური დაპროგრამებელი **flash** მეხსიერებით. ერთ ციკლში ბრძანებების შესრულებისას **ATMega48/ATMega88/ATMega168** აღწევს 1MIPS მწარმოებლურობას. თუ ამგზნები გენერატორის სიხშირეა 1მეგაჰერცი, ეს სისტემის დამმუშავებლებს საშუალებას აძლევს მოახდინონ მწარმოებლურობის და ენერგიის მოხმარების ოპტიმიზაცია.

AVR ბირთვი აერთიანებს ბრძანებების მდიდარ კრებულს და საერთო დანიშნულების 32 მუშა რეგისტრს. ყველა (32) რეგისტრი უშუალოდაა დაკავშირებული არითმეტიკულ-ლოგიკურ მოწყობილობასთან, რაც საშუალებას იძლევა ერთი ბრძანების შესრულების დროს მიღებულ იქნეს ორ დამოუკიდებელ რეგისტრთან შეღწევის საშუალება. შედეგად, ამ არქიტექტურის მწარმოებლურობა 10-ჯერ მეტია, ვიდრე სტანდარტული **CISC** არქიტექტურის.

ATMega48/ATMega88/ATMega168ის მახასიათებლებია: 4/8/16კილობაიტიანი შიგასისტემური დაპროგრამებელი **flash** პროგრამების მეხსიერება, 256/512/512-ბაიტიანი მონაცემთა მეხსიერება **EEPROM**, 512/1კ/1-კბაიტიანი **SRAM** (სტატიკური **ოდმ**), საერთო დანიშნულების 23 შესავალ/გამოსავალი ხაზი, საერთო დანიშნულების 32 მუშა რეგისტრი, სამი მოქნილი ტაიმერ/მრიცხველი შედარების სქემით, შიგა და გარე წყვეტების წყაროები, მიმდევრობითი დამპროგრამებელი **USART**, ბაიტ-ორიენტირებული მიმდევრობითი ორგამტარიანი ინტერფეისი, 6-არხიანი **აცგ** (8-არხიანი **TQFP** და **MFL**-კორპუსიან მოწყობილობებში). აქედან 4 არხს აქვს 10-ბიტიანი გარჩევადობა, 2 არხს - 8-ბიტიანი, პროგრამული გუშაგი ტაიმერი ჩაშენებული გენერატორით, **SPI** პორტი და დაბალი ენერგომოხმარების 5 პროგრამულად ინიციალიზირებული რეჟიმი.

Idle რეჟიმში ბირთვი წყვეტს მუშაობას, ხოლო **SRAM**, ტაიმერ/მრიცხველი, **SPI** პორტი და წყვეტის სისტემა განაგრძობს ფუნქციონირებას. **Power-down** რეჟიმში რეგისტრების შიგთავსი ინახება, ჩერდება მომწოდებელი გენერატორი და ითიშება მიკროპროცესორის ყველა ფუნქცია, ვიდრე არ მოხდება წყვეტა ან აპარატული ჩამოყრა. **Power-save** რეჟიმში ასინქრონული ტაიმერები განაგრძობს ფუნქციონირებას, რაც იძლევა დროის ინტერვალების ათვლის საშუალებას, მაშინ, როცა მიკროკონტროლერი იმყოფება "ძილის" რეჟიმში. **ADC Noise Reduction** რეჟიმში გამომთვლელი ბირთვი და ყველა შესავალ/გამოსავალი მოდული, ასინქრონული ტაიმერისა და **აცგ**-ს გარდა, წყვეტს მუშაობას, რაც ხმაურს ამცირებს ანალოგურ-ციფრული გარდაქმის შესრულების დროს. **Standby** რეჟიმში მომწოდებელი გენერატორი აგრძელებს მუშაობას, ხოლო დანარჩენი ხელსაწყოები – წყვეტს. ეს იძლევა ხელსაწყოების სწრაფი ჩართვისა და მოხმარებული ენერგიის ერთდროული შემცირების საშუალებას.

ATMega ხელსაწყო დამზადებულია კომპანია **Atmel**-ის მიერ და დამუშავებულია მაღალი სიმჭიდროვის ენერგოდამოუკიდებელი მეხსიერების დამზადების ტექნოლოგიით. ჩაშენებული **ISP Flash**-ის მეშვეობით შეიძლება პროგრამების მეხსიე-

რების კვლავპროგრამირება მიმდევრული **SPI** ინტერფეისის გამოყენებით. იგი ხორციელდება პროგრამა-ჩამტვირთველის მიერ, რომელიც სრულდება **AVR**-ის ბირთვში ან ენერგოდამოუკიდებელი მეხსიერების ჩვეულებრივი პროგრამატორით. პროგრამა-ჩამტვირთველს შეუძლია ჩატვირთოს მონაცემები, მიკროკონტროლერის ნებისმიერი ინტერფეისით. ჩატვირთვის სექტორში პროგრამა აგრძელებს მუშაობას მაშინაც, როცა მიმდინარეობს მეხსიერების ჩატვირთვა გამოყენებული პროგრამით, რაც "წაკითხვა-ჩაწერისას" რეალურ რეჟიმს უზრუნველყოფს. 8-ბიტიანი **RISK** ბირთვის და თვითპროგრამირებადი შიგა სისტემის **FLASH** მეხსიერების გაერთიანებით კორპორაცია **Atmel**-მა დაამზადა **ATMega48/ATMega88/ATMega168** ხელსაწყოები, როგორც მძლავრი მიკროკონტროლერები, რომლებიც მმართველი ხელსაწყოების დიდ მოქნილობასა და ეკონომიკურ ეფექტურობას უზრუნველყოფს.

ATMega48/ATMega88/ATMega168 მიკროკონტროლერების მხარდაჭერა ხორციელდება სხვადასხვა პროგრამული და დამუშავების ინტეგრირებული საშუალებებით, როგორიცაა: **C** კომპილატორები, მიკროასემბლერები, პროგრამული ამწყობ-სიმულატორები და შიგასქემური ემულატორები.

მოცემული ოჯახის მიკროკონტროლერები **AVR**-ის ყველაზე სრულყოფილი წარმომადგენლებია. მათი განმასხვავებელი ნიშან-თვისებებია:

- **FLAS** –მეხსიერება 8128კილობაიტი მოცულობით (128კილობაიტი – **ATmega128**). წაშლა/ჩაწერის ციკლების რაოდენობა არანაკლებ 10000;
- ოპერატიული მეხსიერება (სტატიკური **ოდმ**) 1...4კილობაიტი მოცულობით (4კილობაიტი –**ATmega128**);
- **EEPROM**-ის ბაზაზე მონაცემთა მეხსიერების მოცულობა 512ბაიტი...4კილობაიტი (4კილობაიტი – **ATmega128**). წაშლა/ჩაწერის ციკლების რაოდენობა არანაკლებ 100000;
- პროგრამებისა და მონაცემების მეხსიერებების წაკითხვისა და მოდიფიკაციის დაცვის საშუალება;
- მიმდევრობითი **SPI** და **JTAG** ინტერფეისების გამოყენებით სისტემაში უშუალო დაპროგრამების შესაძლებლობა;
- თვითპროგრამირების შესაძლებლობა;
- შიგასქემური გაწყობის შესაძლებლობა IEEE 1149.1 (**JTAG**) სტანდარტის შესაბამისად;
- სინქრონიზაციის სხვადასხვა საშუალება: ჩაშენებული **RC**გენერატორი, შიგა ან გარე დროის განმსაზღვრელი **RC**რგოლით ან გარე რეზონატორით (პიეზოკერამიკული ან კვარცული), სინქრონიზაციის გარე სიგნალი;
- დაბალი ენერგომოხმარების რამდენიმე რეჟიმის არსებობა;
- კვების ძაბვის დაწევის დეტექტორი;
- ტაქტური გენერატორის სიხშირის პროგრამული დაწევის საშუალება.

ცენტრალური პროცესორი და მეხსიერება:

- მთლიანად სტატიკური არქიტექტურა (მინიმალური ტაქტური სიხშირე ნულს უტოლდება);
- **ალშ** მიერთებულია საერთო დანიშნულების რეგისტრებზე;
- ბრძანებების უმრავლესობა სრულდება ერთი სამანქანო ციკლით;

- წყვეტის მრავალდონიანი სისტემა;
- წყვეტის რიგის დაცვა;
- წყვეტის 27 წყარო, აქედან 8 გარე;
- პროგრამული სტეკის არსებობა;
- აპარატული სამრავლებლის არსებობა.

*მაღალხარისხიანი დაბალმომხმარებელი 8-ბიტიანი **AVR** მიკროკონტროლერი:*

- მოწინავე **RISC** არქიტექტურა;
- 130 ბრძანება, რომელთა უმრავლესობა ერთ ტაქტურ ციკლში სრულდება;
- 32 8-ბიტიანი საერთო მოხმარების მუშა რეგისტრი;
- მთლიანად სტატიკური არქიტექტურა;
- წარმადობა 16MIPS-მდე 16 მეგაჰერც ტაქტურ სიხშირეზე;
- ჩაშენებული ორციკლიანი გასამრავლებელი;
- პროგრამებისა და მონაცემების ენერგოდამოუკიდებელი მეხსიერებები;
- 4/8/16 კილობაიტი შიგასისტემურად დაპროგრამებადი **Flash** პროგრამების მეხსიერება, რომელსაც შეუძლია გაუძლოს ჩაწერა/წაშლის 10 000 ციკლს;
- ჩამტვირთველი პროგრამის დამხმარე სექცია დამოუკიდებელი დაცვის ბიტით;
- შიგასისტემური დაპროგრამება ჩაშენებული პროგრამა-ჩამტვირთველით;
- წაკითხვის რეალური ფუნქცია დაპროგრამირებისას;
- 256/521/512-ბაიტიანი **EEPROM**, რომელსაც შეუძლია გაუძლოს 100 000 ჩაწერა/წაშლის ციკლს;
- 512/1კ/1კბაიტის ჩაშენებული **SRAM** მეხსიერება (სტატიკური ოდმ).
- პროგრამული დაცვა წაკითხვებისაგან.

პერიფერიის დახასიათება:

- ორი 8-თანრიგიანი ტაიმერ/მრიცხველი ცალკე დამყოფით და შედარების რეჟიმით;
- ერთი 16-თანრიგიანი ტაიმერ-მრიცხველი ცალკე დამყოფით, შედარებისა და ჩატაცების რეჟიმებით;
- რეალური დროის მრიცხველი განცალკევებული გენერატორით.
- ხუთი **გიმ** არხი;
- 8-არხიანი **აცგ** ხელსაწყოები **TQF** და **MFL** კორპუსებში;
- 6 10-ბიტიანი არხი;
- 2 8-ბიტიანი არხი;
- 6-არხიანი **აცგ** ხელსაწყოები **PDIP** კორპუსში;
- 4 10-ბიტიანი არხი;
- 2 8-ბიტიანი არხი;
- მიმდევრობითი დაპროგრამირებადი **USART**;
- წამყვან/მიმყოლი **SPI** ინტერფეისი;
- ბაიტ-ორიენტირებული მიმდევრობითი ორგამტარიანი ინტერფეისი;
- დაპროგრამირებადი გუშაგი ტაიმერი ჩაშენებული გენერატორით;
- ჩაშენებული ანალოგური კომპარატორი;
- წყვეტა და "გამოფხიზლება" გამომყვანების მდგომარეობის შეცვლისას.

სპეციალური დახასიათება:

- ჩამოყრა კვების ჩართვისას და ხანმოკლე კვების დაკარგვის დეტექტორი;
- ჩაშენებული დაკალიბრებული გენერატორი. წყვეტის გარე და შიგა წყარო;
- შემცირებული მოხმარების 5 რეჟიმი: **Idle, ADC Noise Reduction, Power-Save, Power-down** da **Standby**;
- შესავალ/გამოსავალი პორტები და კორპუსში შესრულება;
- შესავალ/გამოსავალი პორტების 23 დაპროგრამირებადი ხაზი;
- 32-გამომყვანიანი **TQFP** და **MFL** კორპუსები;
- კვების ძაბვის დიაპაზონი:
  - 1.8-დან 5.5ვ-მდე **TMega48/ATMega88/ATMega168V;**
  - 2.7-დან 5.5ვ-მდე **TMega48/ATMega88/ATMega168L;**
  - 4.5-დან 5.5ვ-მდე **ATMega48/ATMega88/ATMega168;**
- კომერციული მუშა ტემპერატურული დიაპაზონი;
- მუშა ტაქტური სიხშირის სხვადასხვა დიაპაზონი:
  - 0-დან 1 მეგაჰერცამდე **ATMega48/ATMega88/ATMega168V;**
  - 0-დან 8 მეგაჰერცამდე **ATMega48/ATMega88/ATMega168L;**
  - 0-დან 16 მეგაჰერცამდე **ATMega48/ATMega88/ATMega168 ;**
- უმცირესი ენერგომოხმარება;
- აქტიური რეჟიმი:
  - 300**მკა** 1 მეგაჰერც სიხშირეზე და 1.8ვ კვების ძაბვაზე;
  - 20**მკა** 32 კილოჰერც სიხშირეზე და 1.8ვ კვების ძაბვაზე;
- დაბალი მოხმარების რეჟიმი: 0.5**მკა** 1.8ვ კვების ძაბვით.

2.2.2. Mega ოჯახის არქიტექტურა

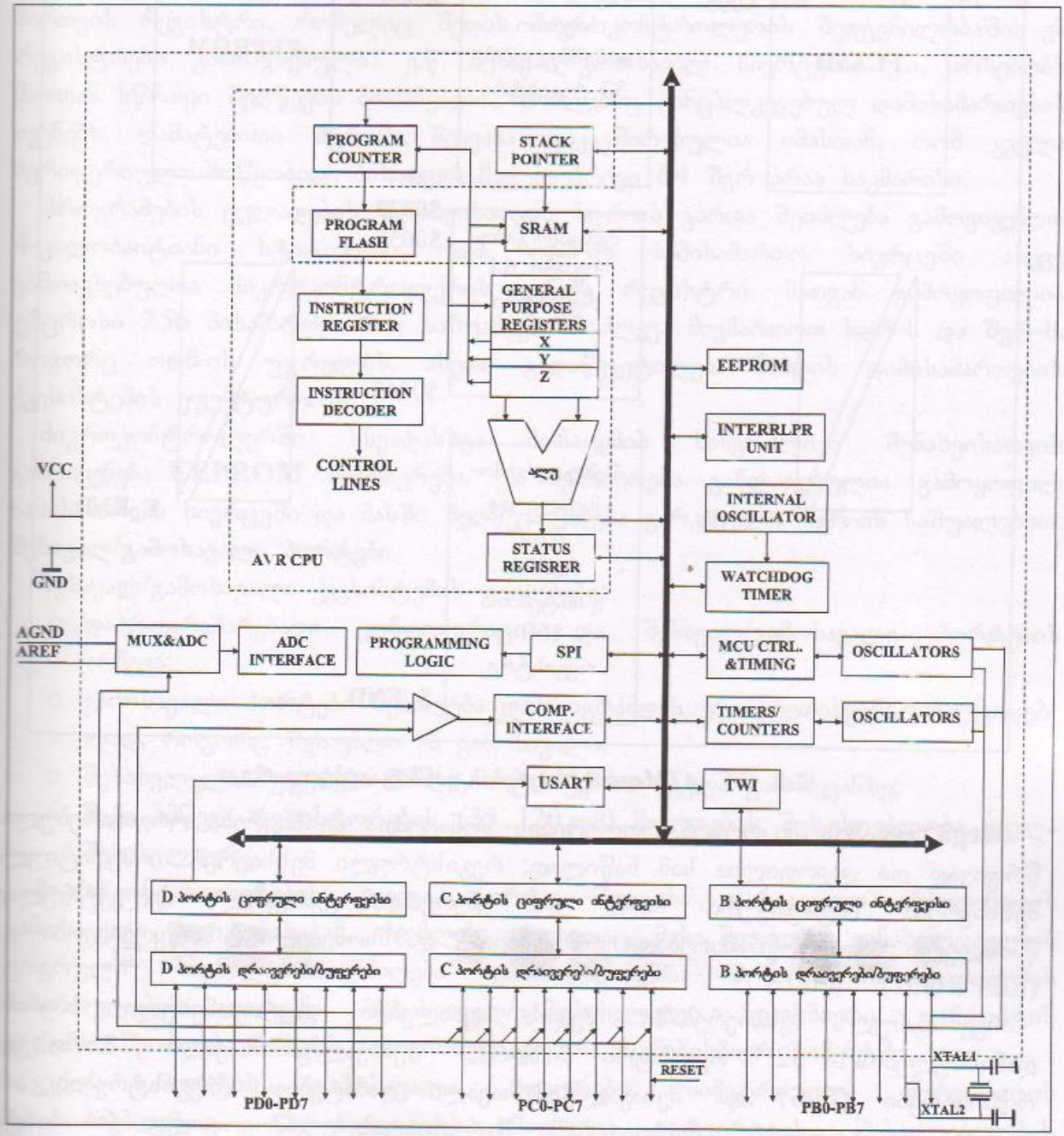
Mega ოჯახის **AVR** მიკროკონტროლერებში რეალიზებულია ჰარვარდის არქიტექტურა, რომლის მიხედვითაც გაყოფილია არა მარტო პროგრამების მეხსიერების და მონაცემთა მეხსიერების სამისამართო არე, არამედ მათთან შეღწევის სალტეებიც (ნახაზი 2.4), ე.ი. დამისამართების და შეღწევის საშუალებები ამ მეხსიერებათა არეებში სხვადასხვაა. ასეთი სტრუქტურა ცენტრალურ პროცესორს საშუალებას აძლევს იმუშაოს ერთდროულად პროგრამების მეხსიერებასა და მონაცემთა მეხსიერებასთან.

პროგრამების მეხსიერების დანიშნულებაა ბრძანებების შენახვა, რომლებიც მიკროკონტროლერებისა და კონსტანტების ცხრილების აგებას მართავს (არ იცვლება მუშაობის დროს). როგორც ზემოთ აღინიშნა, იგი წარმოადგენს ელექტრულად წამშლელ ნახევრად **მდმ**-ს (**flash – მდმ**). იმის გამო, რომ ყველა ბრძანების სიგრძე ერთი 16-ბიტიანი სიტყვის ჯერადია, პროგრამების მეხსიერებას აქვს 16-თანრიგიანი ორგანიზაცია.

მეხსიერების რუკა მოყვანილია 2.5 ნახაზზე.

პროგრამების მეხსიერების დამისამართებისათვის გამოიყენება ბრძანების მთვლელი (**PC** – Program Counter). პროგრამების ნორმალური შესრულებისას ყოველ მანქანურ ციკლში ბრძანების მთვლელის შედგენილობა ავტომატურად იზრდება ერთით ან ორით (შესასრულებელი ბრძანების ზომის შესაბამისად). ეს წყობა ირღვევა, როცა ხდება გადასვლის ბრძანების შესრულება, ქვეპროგრამიდან

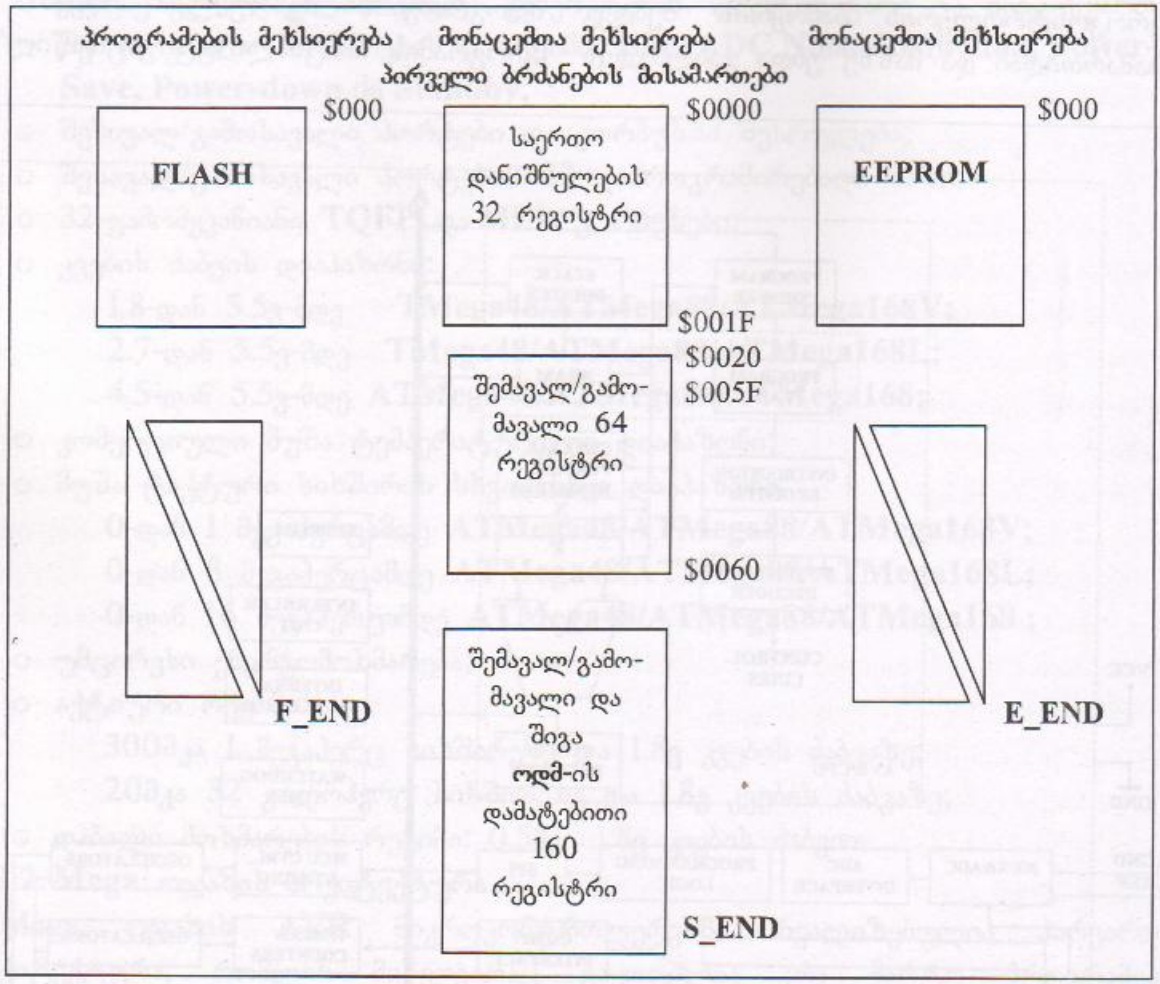
გამოძახება და დაბრუნება, აგრეთვე წყვეტის წარმოქმნა. პროგრამების მახსოვრობის $0000 მისამართზე მდებარეობს ჩამოყრის ვექტორი, ე.ი. მიკროკონტროლერის ჩამოყრის შემდეგ პროგრამების შესრულება იწყება ამ მისამართიდან და მასზე უნდა განლაგდეს პროგრამის ინიციალიზაციურ ნაწილზე



ნახ. 2.4. AVR Mega მიკროკონტროლერის არქიტექტურის ბლოკ-სქემა

გადასვლის ბრძანება. $0001 მისამართიდან დაწყებული პროგრამის მეხსიერებაში განლაგებულია წყვეტის ვექტორების ცხრილი. წყვეტის წარმოშობისას, ბრძანების მრიცხველის მიმდინარე მნიშვნელობის სტეკში შენახვის შემდეგ, ხდება ბრძანების შესრულება, რომელიც განლაგებულია შესაბამისი ვექტორის მისამართზე. ამ

მისამართზე უნდა განლაგდეს ქვეპროგრამებზე გადასვლის ბრძანებები, რომლებიც შესაბამის წყვეტებს ამუშავებს.



ნახ. 2.5. ATMega მეხსიერების განზოგადებული რუკა

Mega ოჯახის მიკროკონტროლერების მონაცემთა მეხსიერება ორგანიზებულია წრფივად და დაყოფილია სამ ნაწილად: რეგისტრული მეხსიერება, ოპერატიული მეხსიერება (სტატიკური **ოდმ**) და ენერგოდამოუკიდებელი ელექტრულად წამშლელი დაპროგრამირებადი მუდმივი დამხსომებელი მოწყობილობა - (**EEPROM**).

რეგისტრულ მეხსიერებაში შედის ფაილებში გაერთიანებული საერთო დანიშნულების 32 რეგისტრი (**სდრ**), შესავალ/გამოსავალი მომსახურე რეგისტრები (**შგრ**) და შესავალ/გამოსავალი დამატებითი რეგისტრების არე (**დშგრ**). კონტროლერის მეხსიერებაში **შგრ**-ისათვის გამოყოფილია 64 ბაიტი, **დშგრ**-ისათვის 160 ბაიტი.

საერთო დანიშნულების ყველა რეგისტრი გაერთიანებულია სწრაფი შეღწევის რეგისტრულ ფაილში. როგორც უკვე ითქვა, **AVR** მიკროკონტროლერის ყველა 32 **სდრ** უშუალოდ მიღწევადია **ალმ**-ისათვის, რაც საშუალებას იძლევა გამოვიყენოთ ნებისმიერი **სდრ** პრაქტიკულად ყველა ბრძანებაში, როგორც ოპერანდ-წყარო და ოპერანდ-მიმღები. *ასეთი გადაწყვეტილება (მონაცემების კონვეიერულ დამუშავებასთან ერთად)* **ალმ**-*ს საშუალებას აძლევს შეასრულოს ერთი ოპერაცია*

(ოპერანდების ამოღება რეგისტრული ფაილიდან, ბრძანების შესრულება და შედეგების ჩაწერა უკან რეგისტრულ ფაილში) ერთ სამანქანო ციკლში.

შესავალ/გამოსავალი რეგისტრების ორივე არეში განლაგებულია სხვადასხვა მომსახურების რეგისტრები (მიკროკონტროლერის მართვის რეგისტრი, მდგომარეობის რეგისტრი და ა.შ). აგრეთვე პერიფერიული მოწყობილობების მართვის რეგისტრი, რომელიც შედის მიკროკონტროლერის შედგენილობაში. ეს რეგისტრები განთავსებულია ე.წ შესავალ/გამოსავალ სივრცეში, ე.ი. არსებობს მათთან სწრაფი შეღწევის ბრძანებები, რომლებიც განცალკევებულ დამისამართებას იყენებს. დამატებითი **შგრ**-ის შეყვანა დაკავშირებულია იმასთან, რომ ყველა პერიფერიული მოწყობილობისათვის ჩვეულებრივი 64 **შგრ** არაა საკმარისი.

პროგრამების ცვლადების შენახვისათვის **სდრ**-ის გარდა შეიძლება გამოვიყენოთ 4-კილობაიტიანი სტატიკური **ოდმ**. **ოდმ**-ის სამისამართო სივრცეში ასევე განთავსებულია მიკროკონტროლერის ყველა რეგისტრი. მათგან გამოყოფილია უმცროსი 256 მისამართი, რაც საშუალებას იძლევა მივმართოთ **სდრ**-ს და **შგრ**-ს, როგორც **ოდმ**-ის უჯრედებს. ასეთი გადაწყვეტილება ზრდის დამისამართების ქვესისტემის ეფექტურობას.

მიკროკონტროლერში სხვადასხვა მონაცემის ხანგრძლივი შენახვისათვის გამოიყენება **EEPROM** მეხსიერება. ეს მეხსიერება განლაგებულია გამოყოფილ სამისამართო სივრცეში და მასში შეღწევა ხდება გარკვეული **შგრ**-ის საშუალებით.

შესავალ/გამოსავალი პორტები

შესავალ/გამოსავალი ქვესისტემის თვისებები:

- დაპროგრამირებადი კონფიგურაცია და შესავალ/გამოსავალი პორტების არჩევა;
- გამოსავალი პორტები შეიძლება დაპროგრამდეს ერთმანეთისგან დამოუკიდებლად, როგორც შესავალი ან გამოსავალი;
- შესასვლელი ბუფერები შმიდტის ტრიგერით ყველა გამომყვანზე;
- შიგა ამწევი რეზისტორების (35...120კომ) მიერთების შესაძლებლობა ყველა შესასვლელზე.

მიკროკონტროლერების ყველა პორტი შედგება გარკვეული რაოდენობის გამოსავალი პორტებისაგან, რომლის გავლითაც მას შეუძლია განახორციელოს ციფრული სიგნალების მიღება და გადაცემა. მონაცემების გადაცემის მიმართულების დადგენა შესასვლელ/გამოსასვლელი ნებისმიერი კონტაქტის გავლით, შეიძლება მოხდეს პროგრამულად დროის ნებისმიერ მომენტში.

ყველა პორტის გამოსავალი ბუფერები სიმეტრიული დატვირთვის მახასიათებლებით უზრუნველყოფს მაღალი დატვირთვის შესაძლებლობას სიგნალის ნებისმიერი დონის დროს.

ყველა სახის მიმღები ბუფერი აწყობილია შმიდტის ტრიგერის სქემით. ყველა მიმღებს აქვს საშუალება მიიერთოს შიგა ამწევი რეზისტორი მიმღებსა და კვების სალტეს შორის.

*AVR მიკროკონტროლერების პორტების განსაკუთრებული თავისებურებაა ის, რომ შეუძლია შეასრულოს ოპერაცია ნებისმიერ გამომსვლელზე (**CBI** და **SBI** ბრძანებების დახმარებით), რაც გავლენას არ ახდენს პორტის სხვა გამომყვანებზე.*

ეს შეეხება შესასვლელ/გამოსასვლელი კონტაქტის მუშაობის რეჟიმის შეცვლას, გამოსავალი ბუფერის მდგომარეობის შეცვლას (გამოსასვლელისათვის) და შიგა ამწევი რეზისტორის მდგომარეობის შეცვლას (შესასვლელისათვის).

ATmega მიკროკონტროლერს აქვს ექვსი 8- და ერთი 5-თანრიგიანი შესავალ/გამოსავალი პორტი, ე.ი. სულ 53 შესავალ/გამოსავალი ბიტი. შესავალ/გამოსავალი პორტების მართვა, კონფიგურაცია და მიმართვა ხორციელდება შესაბამისი შესავალ/გამოსავალი რეგისტრების **შგრ**-ის დახმარებით.

ATmega-ში გათვალისწინებულია შემომსვლელ/გამომსვლელ კონტაქტებთან დაკავშირებული 8 გარე წყვეტა და მათი გენერაციის რამდენიმე პირობა: გამომყვანზე დაბალი დონის მიხედვით, სიგნალის ვარდნის ფრონტის მიხედვით, გამომყვანზე სიგნალის მატების ფრონტის მიხედვით. წყვეტის დამუშავების მართვა შესასვლელ/გამოსასვლელი კონტაქტებიდან ხდება შესაბამისი **შგრ**-ის დახმარებით.

აღსანიშნავია, რომ შესავალ/გამოსავალი პორტის ხაზები ითავსებს სხვადასხვა ფუნქციას, ეს დამოკიდებულია კონტროლის მუშაობის რეჟიმსა და პერიფერიულ მოწყობილობებზე, რომლებიც ჩართულია ამ ხაზებში.

ტაიმერ/მრიცხველები

ATmega მიკროკონტროლერში არის 4 ტაიმერ/მრიცხველი, მათი აღნიშვნებია: **T/C0, T/C1, T/C2, T/C3**. ამას გარდა, მიკროკონტროლერის შედგენილობაშია ე.წ. გუშაგი ტაიმერი, რომელიც საშუალებას იძლევა გამოვრიცხოთ პროგრამების „დაკიდება“. 8-თანრიგიანი **T/C0,T/C2** და 16-თანრიგიანი **T/C1,T/C2** ტაიმერ/მრიცხველები იდენტურია.

სქემატურად ტაიმერ/მრიცხველები შეიცავს:

1. საანგარიშო რეგისტრს, რომელიც ინკრემენტირებს ან დეკრემენტირებს, ტაიმერის მუშაობის რეჟიმის მიხედვით, მრიცხველის ყველა ტაქტში;
2. რამდენიმე დამხმარე რეგისტრს, რომლებიც მრიცხველს ავალებს მუშაობის რეჟიმებს (მაგალითად, შედარების ბლოკის რეგისტრი ან “ამტაცი” ბლოკის რეგისტრი);
3. შემავალ და გამომავალ ხაზებს;
4. წყვეტის გენერაციის მექანიზმს მოვლენების მიხედვით, რომლებიც დამოკიდებულია მრიცხველი რეგისტრის მდგომარეობაზე.

T/C0 და **T/C2** ტაიმერ/მრიცხველებში შედის 8-თანრიგიანი მრიცხველ/რეგისტრები. აქედან გამომდინარე, ამ მრიცხველების მაქსიმალური დაშვება 256 ტაქტია. მრიცხველი რეგისტრები **T/C1** და **T/C3** 16-თანრიგიანებია და დროის გაზომვის მაქსიმალური ინტერვალი არის ტაიმერის 65536 ტაქტი. ასეთი სიგნალის მიღებისათვის ტაიმერ/მრიცხველებში გამოიყენება წინადამყოფები, რომლებიც სისტემურ ტაქტურ სიგნალს ყოფს პროგრამულად მოცემული კოეფიციენტის მიხედვით. **ATmega**-ში არის 2 წინაგამყოფი: ერთი **T/C0**-თვის, ხოლო მეორე **T/C1,T/C2,T/C3**-თვის ტაიმერებთან ერთად. ტაიმერების ტაქტური სიგნალისათვის შეიძლება გამოყენებულ იქნეს უშუალოდ ***clc0*** *მიკროპროცესორის სისტემური ტაქტური სიგნალი ან გარე სიგნალი არა უმეტეს* ***clc0/2*** სიხშირისა.

ATmega-ში გათვალისწინებულია ტაიმერ/მრიცხველის და მასთან დაკავშირებული წყვეტის მუშაობის რამდენიმე რეჟიმი:

1. **Normal** (ჩვეულებრივი);
2. **CTC** (ჩამოყრა დამთხვევისას);
3. **Fast PWM** (სწრაფმოქმედი **გიმ**);
4. **Phase Correct PWM** (გიმ ზუსტი ფაზით).

Normal რეჟიმში ტაქტური სიგნალის ყველა იმპულსზე ხორციელდება მთვლელი რეგისტრის ინკრემენტი. მაქსიმალურ შესაძლებელ ნიშნულზე გადასვლისას (**$FF- T/C0**-თვის და **T/C2**-თვის, **$FFFF- T/C1** და **T/C2**-თვის) წარმოიქმნება გადავსება და ათვლა $0000-დან გრძელდება. სიგნალის იმავე ტაქტში, რომელშიც მრიცხველი/რეგისტრი განულდება, გენერირდება შეკითხვა წყვეტაზე გადავსებისას. ამ რეჟიმში შესაძლებელია წყვეტის გენერაცია დამთხვევის მიხედვით, როდესაც მთვლელი რეგისტრის ნიშნული ემთხვევა შედარების რეგისტრის ნიშნულს (16-თანრიგიანი ტაიმერების შემთხვევაში არსებობს რამდენიმე წყვეტა დამთხვევის მიხედვით, რადგანაც მათში არსებობს სამი შედარების ბლოკი). ტაიმერის შედარების ბლოკები ასევე შეიძლება გამოვიყენოთ შესაბამისი ტაიმერ/მრიცხველების გამოსავალი ხაზების მდგომარეობის შეცვლისათვის.

CTC რეჟიმში (როგორც წინა შემთხვევაში) მთვლელი/რეგისტრის ინკრემენტი ხორციელდება ტაქტური სიგნალის ყველა იმპულსის მიხედვით, მაგრამ მთვლელი რეგისტრის მაქსიმალურ ნიშნულს და, შესაბამისად, მრიცხველის გარჩევითობის უნარს განსაზღვრავს შედარების რეგისტრი (16-თანრიგიანი ტაიმერის შემთხვევაში, განისაზღვრება პირველი (**A**) ბლოკის შედარების რეგისტრით). შედარების რეგისტრში ჩაწერილი ნიშნულის მიღწევის შემდეგ ათვლა გრძელდება $0000 ნიშნულიდან. სიგნალის ამავე ტაქტში ხდება წყვეტაზე მოთხოვნის გენერაცია შესაბამის ტაიმერ/მრიცხველის გადავსებისა და დამთხვევისას. ამავე დროს, წყვეტის გენერაციისას, შეიძლება შეიცვალოს შესაბამისი ტაიმერის გამოსასვლელის მდგომარეობა.

Fast PWM და **Phase Correct PWM** რეჟიმები განკუთვნილია განედურ-იმპულსური მოდულაციის სიგნალების გენერაციისათვის. მრიცხველ/ტაიმერებში შეღწევა და მათთან დაკავშირებული წყვეტებით და წინგამყოფებით მართვა ხდება შესაბამისი **შგრ**-ის დახმარებით.

უნივერსალური სინქრონულ/ასინქრონული მიმღებ-გადამცემი

ATmega მიკროკონტროლერს გააჩნია მიმდევრობითი კავშირის ორი მოდული - უნივერსალური სინქრონულ/ასინქრონული მიმღებ-გადამცემი (**USART**). **USART**-ის ორივე მოდული უზრუნველყოფს სრულ დუპლექსურ გაცვლას მიმდევრობით არხზე. ამასთან, მონაცემთა გადაცემის სიჩქარე იცვლება ფართო ფარგლებში. გზავნილის სიგრძე შეადგენს 5-დან 9 თანრიგს ლუწობის კონტროლით ან მის გარეშე. ამას გარდა, **USART**-ის მოდულებს შეუძლია აღმოაჩინოს სხვადასხვა არაშტატური სიტუაციები, როგორიცაა მიმღები ბუფერის გადავსება, შეცდომა კადრირებაში, არასწორი სტარტ-ბიტი. შეფერხების არსებობის შესამცირებლად მოდულებში რეალიზებულია დამახინჯებების ფილტრაციის ფუნქცია. პროგრამასთან ურთიერთობისათვის მოდულებში გათვალისწინებულია 3 წყვეტა, რომელთა გენერაციაზე მოთხოვნა გენერირდება

მოვლენების დადგომისას: გადაცემა დამთავრებულია, მონაცემთა გადაცემის რეგისტრი ცარიელია, მიღება დამთავრებულია.

USART მოდულის მიერ გამოყენებული მიკროკონტროლერის გამოსასვლები საერთო დანიშნულების შესასვლელ/გამოსასვლელი პორტების ხაზებია, რომელთა ფუნქციონირება განისაზღვრება მიმღების ან გადამცემის ჩართვისას.

USART-ის ყველა მოდული შედგება სამი ძირითადი ნაწილისაგან: ტაქტირების, გადამცემი და მიმღები ბლოკებისგან. ტაქტირების ბლოკი შეიცავს სინქრონიზაციის სქემას, რომელიც გამოიყენება სინქრონულ რეჟიმში მუშაობისას და გადაცემის სიჩქარის კონტროლერს. გადაცემის ბლოკში გაერთიანებულია: ერთდონიანი ბუფერი, მძვრელი რეგისტრი, ლუწობის ბიტის ფორმირების სქემა და მართვის სქემა. მიმღები ბლოკი აერთიანებს ტაქტურ სიგნალს, მონაცემთა აღდგენის სქემას, ლუწობის კონტროლის სქემას, ორდონიან **FIFO**-ბუფერს, მძვრელ რეგისტრს და მართვის სქემას. **USART** მოდულებზე შეღწევა და მართვა ჩვეულებრივ ხდება შესაბამისი **შგრ**-ის დახმარებით.

2.3. Mega მიკროკონტროლერების ჩართვა ელექტრონულ სქემებში

მეორე თაობის **AVR** მიკროკონტროლერებს აქვს ხანმოკლე (brown-out) და სრული (black-out) კვების დაწევისგან დაცვა, ასევე low-pass ფილტრი, რათა აღმოფხვრას ხმაური და ძაბვის პიკები, რომლებმაც შეიძლება გამოიწვიოს მიკროკონტროლერის ჩამოყრა. მეორე თაობის ყველა **AVR** მიკროკონტროლერს აქვს ჩაშენებული რეზისტორი, რომელიც "წევს" **RESET** კვების მნიშვნელობასთან. ჩაშენებული pull-up რეზისტორის ნომინალი შეირჩევა **Atmel**-ის ქარხანაში ისე, რომ უზრუნველყოს ჩიპის მაქსიმალურად მდგრადი მუშაობა. რაც არ უნდა თავისთავად მდგრადი იყოს **AVR**, საჭიროა ჩავატაროთ დამატებითი ღონისძიებები მისი სტაბილური მუშაობისათვის.

კვება და ტაქტური სიხშირე

არსებობს **AVR** მიკროკონტროლერების ორი ძირითადი ტიპი. პირველი განკუთვნილია მაქსიმალური სწრაფქმედების მისაღებად მაღალ სიხშირეზე, ხოლო მეორე - დაბალ ტაქტურ სიხშირეზე ეკონომიკური მუშაობისათვის. მეორე ტიპის მიკროსქემის მარკირება იმით განსხვავდება პირველისაგან, რომ ბოლოში ემატება "**L**". მაგალითად, **ATmega8** და **ATmega8L**.

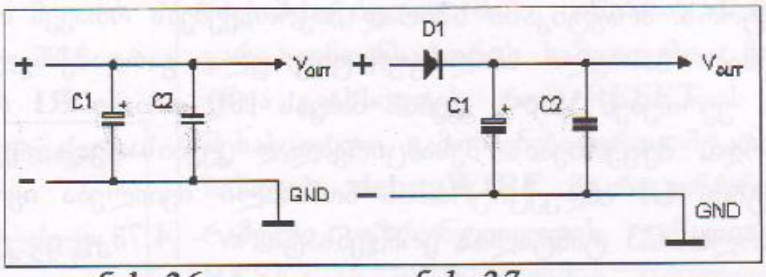
პირველი ჯგუფის მიკროკონტროლერებისათვის დაშვებული კვების დიაპაზონია 4,5-დან 5,5 ვოლტამდე, ტაქტური სიხშირით 0...16 მეგაჰერცი (ზოგიერთი მოდელისათვის - 20 მეგაჰერცამდე), მეორე ჯგუფისთვის - 2,7...5,5 ვოლტი, 0...8 მეგაჰერცი სიხშირით. შესაბამისად, მიკროკონტროლერი ინდექსით "**L**" მოიხმარს ნაკლებ ელექტროენერგიას.

არსებობს მიკროკონტროლერები, რომლებიც მოიხმარენ 1.8ვ-მდე დაწეულ კვებას, ისინი მარკირდება ასო "**V**"-თი, მაგალითად, **ATtiny2313V**. კვების დაწევის შესაბამისად, საჭიროა შემცირდეს ტაქტური სიხშირეც, **Ttiny2313V**-თვის 1,8...5,5ვ ძაბვის დროს სიხშირე უნდა იმყოფებოდეს 0...4მეგაჰერც ინტერვალში, 2,7...5,5ვ ძაბვის დროს კი - 0...10მეგაჰერც ინტერვალში.

ექსპერიმენტული რობოტების შესაქმნელად გამოდგება ყველა ჯგუფის მიკროკონტროლერი. ამასთან, თუ საჭიროა **ATmega8** სქემაში დაყენდეს 3 ვოლტის ტოლი ძაბვა და ჩაირთოს დაბალ ტაქტურ სიხშირეზე, ამ რეჟიმში ის თავისუფლად იმუშავებს. ერთადერთი რისი გარანტიაც არ არსებობს ისაა, თუ როგორი იქნება მიკროკონტროლერის მდგრადი ჩართვა ტემპერატურის უკიდურესი მნიშვნელობისას. მოსახმარი დენიც უფრო მაღალი იქნება, ვიდრე **ATmega8L**-ს.

აქედან გამომდინარე, თუ გვჭირდება მაქსიმალური სწრაფქმედება, ამისათვის საჭიროა დავაყენოთ **Ttiny26** ან **ATmega8** და ავწიოთ ტაქტური სიხშირე 8...16 მეგაჰერცამდე 5ვ ძაბვიდან. თუ უფრო მნიშვნელოვანია რობოტის ეკონომიკური მუშაობა, მაშინ უკეთესია გამოვიყენოთ **Ttiny26L** ან **ATmega8L** და დავწიოთ სიხშირე და კვების ძაბვა. უკეთესი ვარიანტი მეორე შემთხვევისათვის შეიძლება იყოს კვების განხორციელება სამი 1,2-ვოლტიანი აკუმულატორისგან, რაც ჯამში იძლევა 3,6ვ ან სამი ალკინური 1,5ვ ბატარიიდან, რაც ჯამში გვაძლევს 4,5ვ.

Datasheets-ში მითითებულია, რომ რეკომენდებული მაქსიმალური ძაბვა 5,5ვ-ია. მიუხედავად ამისა, პრაქტიკაში ხშირად იყენებენ წყაროს 6ვ ძაბვით. **AVR** მიკროკონტროლერმა შეიძლება იმუშაოს ოთხი ბატარიით, თითოეული 1,5ვ ძაბვით. ეს მოსახერხებელია იმ შემთხვევაშიც, როცა არ გამოიყენება რობოტების ძრავებიდან განცალკევებული წყარო. გასათვალისწინებელია, რომ რაც მაღალია სიხშირე, უფრო მეტია ენერგიის მოხმარება და, შესაბამისად, მიკროსქემის გაცხელებაც. ე.ი. ამ შემთხვევაში უკეთესია არ ჩავრთოთ მიკროკონტროლერი ტაქტური სიხშირის ზღვრულ ნიშნულზე. ასევე უნდა გავითვალისწინოთ, რომ **GND, Vcc** გამოსასვლელებზე მაქსიმალური დენი 200მა-ს არ უნდა აღემატებოდეს.



ნახ. 2.6

ნახ. 2.7

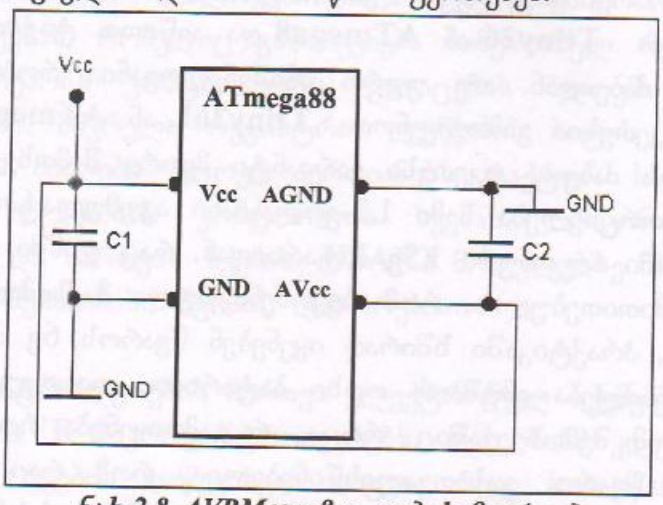
იმისათვის, რომ ავიცილოთ ძაბვის შესაძლო ნახტომები, განსაკუთრებით მიკროსქემების და ძრავების საერთო კვების სქემებში, კვების ხაზებზე პარალელურად რთავენ 100...1000მკფ ელექტროლიტურ კონდენსატორს (C1) (ნახ. 2.6) და ამატებენ დაახლოებით 0,1მკფ კერამიკულ კონდენსატორს (C2), რომელიც ფილტრავს მაღალი და საშუალო სიხშირის შეფერხებებს (ამ კონდენსატორის ზუსტი ნომინალი შეიძლება გამოითვალოს მხოლოდ მაშინ, როცა ზუსტადაა ცნობილი პარაზიტული სიხშირე).

გარდა ამისა, ძრავებისა და მიკროსქემების კვების ძაბვის გამართისათვის, მიკროკონტროლერის კვების დადებით ხაზზე რთავენ დიოდს (ნახ. 2.7).

ექსპერიმენტულ რობოტოტექნიკაში ხშირად იყენებენ 9ვ ძაბვის ბატარეების ნაკრებს და 12-ვოლტიან აკუმულატორებს. ამ შემთხვევაში სქემაში რთავენ დადებითი ძაბვის 5-ვოლტიანი სტაბილიზატორის მიკროსქემას.

Mega მიკროკონტროლერებზე კვების მიერთება

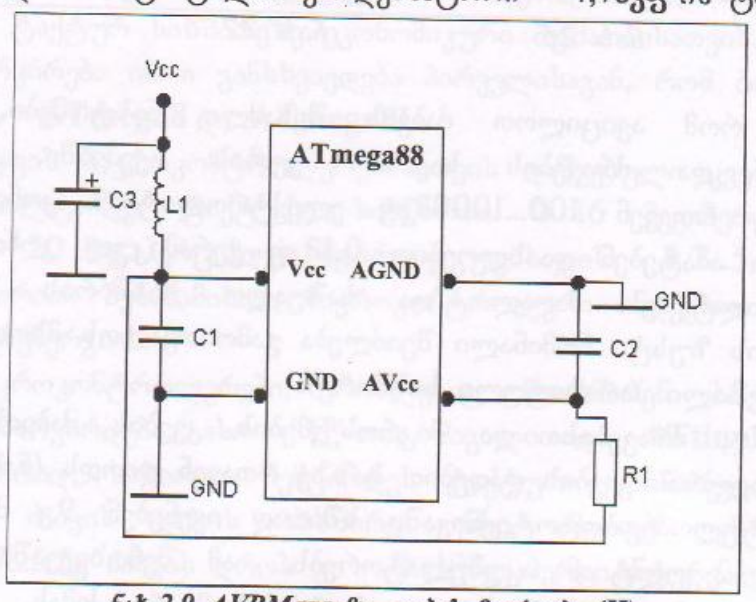
AVR მიკროკონტროლერების ოჯახში არის ორმაგი კვების მიკროსქემები: "ციფრული" (გამოსავალი **Vcc** და **GND**) და "ანალოგური" (**AVcc** და **AGND**, რომელიც აღინიშნება **GND**-თი). ასეთ მიკროსქემებს მიეკუთვნება, მაგალითად, **ATmega8**. სტანდარტული ჩართვისას **Vcc** და **AVcc** გამოსასვლელები უერთდება ერთმანეთს. გამოსასვლელები **GND** შეკრულია მიკროკონტროლერის შიგნით 0,7**ომი** წინაღობის გავლით და მათ "მიწას" უერთებენ.



ნახ. 2.8. AVRMega-ზე კვების მიერთება

C1 და **C2** 0,1**მკფ** ტევადობის კერამიკულ კონდენსატორებს (ნახ. 2.8) სქემაზე განალაგებენ მაქსიმალურად ახლოს შესაბამის გამოსასვლელებთან. თუ **AVR** მიკროკონტროლერს არ აქვს **AVcc** გამოსასვლელი, ორის მაგივრად აყენებენ ერთ კონდენსატორს. პრაქტიკაში ხშირად მიკროსქემებს რთავენ ორმაგი კვებით.

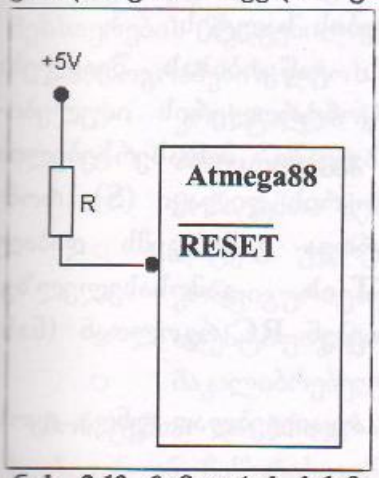
თუ გამოიყენება ჩასმული ანალოგურ-ციფრული გარდამქმნელი, მაშინ **AVcc** გამოსასვლელს უერთებენ **Vcc**-ს კვების ძაბვას 100 **ომიანი R1** რეზისტორის გავლით. ამას გარდა, შეფერხების შესამცირებლად კვებისათვის იყენებენ მიმდევრობით **LC**-ფილტრს. **L1** ინდუქციურობის ნომინალი შეიძლება იყოს 30...47**მკჰნ**-ის დიაპაზონში, ხოლო **C3** ტანტალის კონდენსატორი - 4,7**მკფ**-ის ტოლი (ნახ. 2.9).



ნახ. 2.9. AVRMega-ზე კვების მიერთება (II)

გაუთვალისწინებელი ჩამოყრის რისკის შემცირება

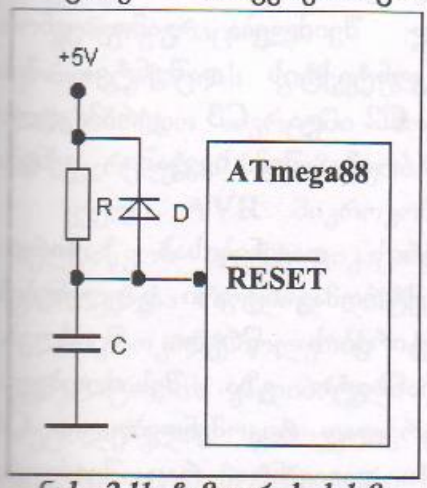
როგორც უკვე აღინიშნა, **RESET** ხაზს აქვს მიბმის შიგა რეზისტორი კვების სალტეზე, რათა გაიზარდოს დაცვა. რეზისტორის დაკალიბრება ხდება მიკროკონტროლერის დამამზადებელ ფაბრიკაში. მეორე თაობის **AVR** მიკროკონტროლერს აქვს უკეთესი დაცვა ხანმოკლე (brown-out) და მთლიანი (black-out) ძაბვის ვარდნისგან. აქედან გამომდინარე, **AVR**-ის მარტივ სქემებში ხანდახან უარს ამბობენ დამატებით საშუალებებზე **RESET**-გამოსასვლელთან მიმართებაში და ზოგჯერ ტოვებენ "ჰაერში ჩამოკიდებულს".



ნახ. 2.10. ჩამოყრის სქემა

ასეთი მიდგომა შეიძლება გამოყენებულ იქნეს ექსპერიმენტული მაკეტირებისას ან სამოყვარულო ნაკეთობებისათვის. საწარმოო ავტომატიკაში კი ასეთმა გადაწყვეტამ შეიძლება შეფერხებები გამოიწვიოს, კვების ხარვეზის გამო. გარე ძლიერი შეფერხებისას ამ რეზისტორის წინაღობა (100-500**კომი**) აღმოჩნდება ძალიან დიდი და **RESET** ხაზზე მაღალი დონის სიგნალის არარსებობამ შეიძლება მიკროკონტროლერის შემთხვევითი ჩამოყრა გამოიწვიოს. არსებობს რამდენიმე საშუალება, რათა შევამციროთ გაუთვალისწინებელი ჩამოყრის რისკ-ფაქტორი. ერთერთი ყველაზე მარტივი მეთოდი **RESET**-ის ხაზზე გარე ამწევი რეზისტორის მიერთებაა, რეკომენდებული წინაღობით 4,7დან 10**კომ**-მდე (ნახ. 2.10).

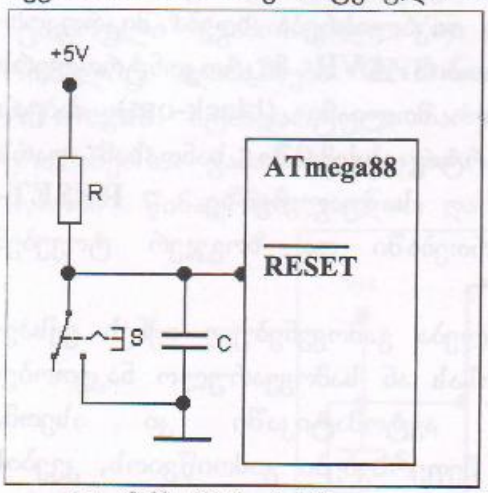
გარე ხარვეზებისაგან **RESET** ხაზის დამატებითი დაცვისათვის რეკომენდებულია მისი მიწასთან 0,1**მკფ** ტევადობის გარე კონდენსატორის საშუალებით მიერთება. მაგრამ უნდა გვახსოვდეს, რომ **RESET**-ის გარე ჩამოყრის შესასვლელი გამოიყენება ერთგამტარიანი ინტერფეისისთვის **debugWIRE** მიკროკონტროლერის პროგრამული უზრუნველყოფის გაწყობის შემთხვევაში. **RESET**-ის შესასვლელთან პარალელურად მიერთებული კონდენსატორი გამოიწვევს ხარვეზებს ამ ინტერფეისის მუშაობაში, ამიტომ თუ დაგეგმილია მიკროკონტროლერის გაწყობა სპეციალურ პლატაზე **debugWIRE**-ის მეშვეობით, აუცილებელია გავითვალისწინოთ ჩამრთველი, რათა გამოვრთოთ კონდენსატორი მიკროკონტროლერის გაწყობისას, პროგრამული უზრუნველყოფის გამოყენებით.



ნახ. 2.11. ჩამოყრის სქემა (II)

მაღალვოლტიანი დაპროგრამებისას **AVR** მიკროკონტროლერს არ გააჩნია სტანდარტული შიგა დიოდი, რომელიც დაიცავს გადამეტებული ძაბვისაგან **RESET**-ის შესასვლელზე. ამიტომ, თუ მაღალვოლტიანი დაპროგრამება არ გამოიყენება, ხარვეზებისაგან დასაცავად რეკომენდებულია (**D**) გარე დიოდის ჩართვა **RESET** ხაზსა და მიკროკონტროლერის კვების სალტეს შორის. აქედან

გამომდინარე, **RESET** ხაზისათვის გარე "კიდული" ელემენტების ტიპურ სქემას აქვს 2.11 ნახაზზე მოცემული სახე.



ნახ. 2.12. "ჩამოყრის" ღილაკი

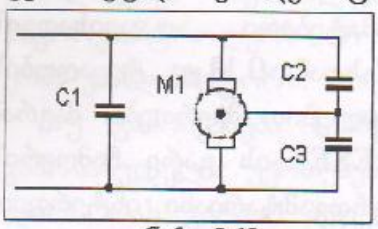
თუ ჩამოყრის ხაზი არ გამოიყენება და შიგასქემური დაპროგრამება არ გვჭირდება, მაშინ დამთავრებულ მოწყობილობაში **RESET**-ის გამოსასვლელი შეიძლება უშუალოდ მივუერთოთ მიკროკონტროლერის კვების სალტეს.

რთული პროგრამების გაწყობისას შეიძლება საჭირო გახდეს მიკროკონტროლერის იძულებითი ჩამოყრა. ამ შემთხვევაში მოსახერხებელია სქემაში დავამატოთ ჩამოყრის ღილაკი (**S**), რომლის კონტაქტების ჩართვა გამოსცემს დაბალ სიგნალს **RESET**-ის გამოსასვლელზე. ჩამოყრის ღილაკს აერთებენ **RC** რგოლთან (ნახ. 2.12).

შიგასქემური დაპროგრამებისას ღილაკის კონტაქტები აუცილებლად უნდა იყოს გათიშულ მდგომარეობაში.

Mega მიკროკონტროლერის გამოყენება სქემებში ელექტროძრავებით

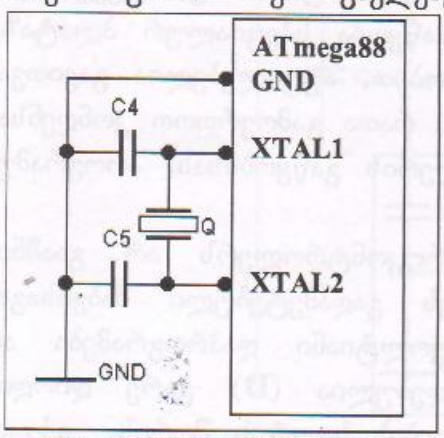
ელექტროძრავას "ნაპერწკლიანობით" გამოწვეული დამახინჯებების შესამცირებლად ძრავების პარალელურად რთავენ 0,01...0,1მკფ ტევადობის კერამიკულ კონდენსატორებს (ნახ. 2.13). კონდენსატორებს განათავსებენ უშუალოდ ძრავას კონტაქტებზე.



ნახ. 2.13

დამატებით ღონისძიებად შეიძლება გამოვიყენოთ ელექტროძრავას ყველა კონტაქტის დაშუნტვა მის კორპუსზე ან "მიწაზე". **C2** და **C3** კერამიკული კონდენსატორის ტევადობა ამ შემთხვევაში იქნება 0,01...0,1მკფ დიაპაზონში.

ძრავას უშუალო სიახლოვეს მიკროკონტროლერის დაყენებისას საჭიროა ვიზრუნოთ **AVR**-ის გარე წრედებზე შესაძლო ელექტრომაგნიტური გავლენების რისკების შემცირებაზე. მაგალითად, გარე ტაქტირების წრედი შეიძლება მოემსახუროს ისეთ გავლენებს, როგორიცაა სატრანზიტო გზა. შესაძლებელი ხარვეზების აღმოსაფხვრელად რეკომენდებულია **C4** და **C5** კონდენსატორი დავაყენოთ რაც შეიძლება ახლოს **XT1** და **XT2** გამოსასვლელებთან და მათი "მიწის" გამოსასვლელები მივუერთოთ მიკროკონტროლერის **GND** გამოსასვლელს მოკლე მავთულით.



ნახ. 2.14. გარე წრედის დაცვა

რეკომენდებულია აგრეთვე **Q** კვარცული რეზონატორის ეკრანი მიუერთდეს **GND** წრედს მოკლე მავთულით. უფრო მეტ უსაფრთხოებას უზრუნველყოფს მაეკრანებელი კონტური ნაბეჭდ პლატაზე, კვარცული რეზონატორისა და კონდენსატორების გარშემო.

2.4. Mega-ს დაპროგრამების პრინციპები

ATMega მიკროკონტროლერი ნებას რთავს დაპროგრამების 3 რეჟიმს:

- თანამიმდევრობითი დაპროგრამება დაბალი ძაბვის დროს (**SPI** ინტერფეისით);
- პარალერული დაპროგრამება მაღალი ძაბვის დროს;
- დაპროგრამება **JTAG** ინტერფეისით.

ასევე შეუძლია თვითდაპროგრამებაც. ეს ტერმინი გულისხმობს პროგრამების მეხსიერების შედგენილობის შეცვლას, რომელსაც მიკროკონტროლერი მართავს.

დაპროგრამების დროს სრულდება შემდეგი ოპერაციები:

- კრისტალის წაშლა;
- პროგრამების მეხსიერების წაკითხვა/ჩაწერა;
- მონაცემთა მეხსიერების წაკითხვა/ჩაწერა;
- დაცვის უჯრედების წაკითხვა/ჩაწერა;
- კონფიგურაციული უჯრედების წაკითხვა/ჩაწერა;
- იდენტიფიკატორის უჯრედების წაკითხვა;
- მაკალიბრებელი ბაიტის წაკითხვა.

მონაცემთა და პროგრამების მეხსიერების შიგთავსი დაცული უნდა იყოს წაკითხვისაგან ან ჩაწერისაგან, შესაბამისი დაცვის უჯრედების დაპროგრამების საშუალებით. ამ უჯრედების დაპროგრამების შემდეგ კონტროლერის მეხსიერება იბლოკება შესაბამისი ოპერაციისათვის. დაცვის უჯრედები შეიძლება გადაიწეროს მხოლოდ კრისტალის წაშლის ბრძანების შესრულების შემდეგ, რაც ასევე გაანადგურებს მიკროკონტროლერის პროგრამებისა და მონაცემების მეხსიერების შიგთავსს.

კონფიგურაციული უჯრედები განსაზღვრავს მიკროკონტროლერის კონფიგურაციის გარკვეულ პარამეტრებს. ეს უჯრედები განლაგდება ცალკე სამისამართო სივრცეში მხოლოდ დაპროგრამებისას. კრისტალის წაშლის ბრძანება ამ უჯრედების მდგომარეობაზე არ მოქმედებს.

ყველა **AVR** მიკროკონტროლერს აქვს სამი 8-თანრიგიანი უჯრედი, რაც ხელსაწყოს იდენტიფიკაციის საშუალებას იძლევა (იდენფიკატორის უჯრედები). გამოიყენება მხოლოდ დაპროგრამების რეჟიმში და მხოლოდ წაკითხვისათვის.

მიკროკონტროლერის დამზადებისას კალიბრების უჯრედში შეაქვთ **RC**-გენერატორის კალიბრული კონსტანტა. მუშაობის დროს შიგა **RC**-გენერატორიდან აუცილებელია გამოვყოთ ამ უჯრედის შიგთავსი, რათა შიგა გენერატორს მივუსადაგოთ ნომინალური სიხშირე.

მიკროკონტროლერის დაპროგრამებისათვის სახელმძღვანელოში აღწერილია რამდენიმე პროგრამატორი, რომლებიც **SPI** ინტერფეისზე მუშაობს, ხოლო პროგრამის დაწერის და გაწყობისათვის ასემბლერზე გამოიყენება ინტეგრირებული გარსი «**AVR-Studio**», რომელიც დამუშავებულია **ATMEL** კომპანიის მიერ.

«**AVR-Studio**» პროგრამა განკუთვნილია **Windows**-ის ოპერატიული სისტემებისათვის, რომელიც საშუალებას იძლევა შეიქმნას რედაქტირებისა და გაწყობის ისეთი პროგრამები ასემბლერისა და **C** ენაზე, რომლებიც იმუშავებენ **AVR** მიკროკონტროლერებზე. იგი ადვილად მიღწევადია **ATMEL** კომპანიის საიტიდან. ყველა მოდელის მიკროკონტროლერს აქვს თავისი მოდელის შესაბამის-

სათაურიანი ფაილი, სადაც განსაზღვრულია მახასიათებელი კონსტანტები, როგორიცაა პროგრამებისა და მონაცემების მაქსიმალური მისამართი, შესავალ/გამოსავალი რეგისტრების მისამართები და ა.შ. პროგრამირების პროცესი არსებითად მარტივდება. ყველა მოდელის პროგრამული სიმულატორი საშუალებას იძლევა მოდელირება ჩავუტაროთ მიკროკონტროლერის მუშაობას დამუშავებული პროგრამის შესრულებისას და ვიპოვოთ შეცდომები მიკროკონტროლერში ჩაწერამდე.

«**AVR-Studio**»-ს მუშაობის შედეგია **HEX**-ფაილი, ბრძანების კოდებით, მიკროკონტროლერისთვის. მიკროკონტროლერის მეხსიერებაში ამ ფაილის ჩასაწერად გამოიყენება სხვადასხვა პროგრამატორი საკუთარი მართვის პროგრამებით.

მიკროკონტროლერების პროგრამის დაწერა, როგორც წესი, ხდება ასემბლერის ენაზე ან **C**-ზე, თუმცა არსებობს სხვა ენებისთვის განკუთვნილი კომპილატორები. პროგრამების გამართვისათვის გამოიყენება სპეციალური პროგრამული სიმულატორები და შიგასქემური ემულატორები, რომლებიც მიკროკონტროლერის მუშაობის იმიტაციას ახდენს.

პროგრამის კოდის, მისი ობიექტური მოდულის და მუშა **.HEX** ფაილის შედგენა არაა საკმარისი მიკროკონტროლერის ასამუშავებლად. საჭიროა მექანიზმი, რომლის მეშვეობითაც შესაძლებელია პროგრამის "გაჭოლვა" (**burn, flash**) მიკროკონტროლერში. ასეთ მექანიზმს წარმოადგენს პროგრამატორი და მისი პროგრამული უზრუნველყოფა, რომლის საშუალებით ხდება პროგრამატორის მართვა და მიკროკონტროლერის პროგრამის (**.HEX** ფაილის) ჩაწერა თვით მიკროკონტროლერში.

2.5. AVR მიკროკონტროლერების პროგრამატორები

მიკროკონტროლერის დაპროგრამებისათვის ("გაჭოლვისათვის") აუცილებელია პროგრამატორი. პროგრამატორი აპარატულ-პროგრამულ კომპლექსია, რომელიც აიგება კომპიუტერისა და მიკროკონტროლერის შემაერთებელი მოწყობილობისა და პროგრამისაგან, რომელიც მართავს ამ მოწყობილობას. პროგრამატორს მიკროკონტროლერის მეხსიერებაში შეაქვს წინასწარ მომზადებული პროგრამა.

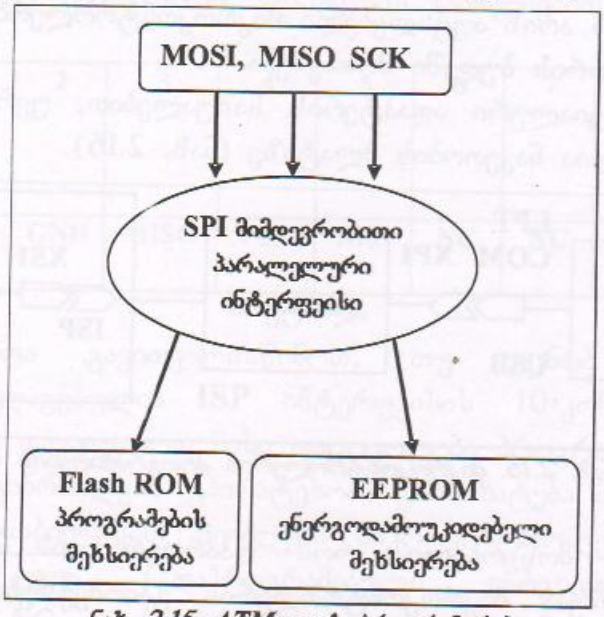
ATMega სერიის მიკროკონტროლერები შეიცავს ფლეშ-მეხსიერების პროგრამირების პარალელურ ინტერფეის. მონაცემების ჩასაწერად საჭიროა მიკროკონტროლერზე 12ვ დაპროგრამების ძაბვის მიწოდება. პროცესის კონტროლისათვის გამოიყენება მიკროკონტროლერის თითქმის ყველა პორტი. დაპროგრამების ასეთი მეთოდი შედარებით სწრაფია, მაგრამ საჭიროებს სპეციალურ პროგრამატორებს.

ზოგიერთი სერიის მიკროკონტროლერი იყენებს **ISP** (In System Programming) მეთოდს. მიმდევრობითი დაპროგრამების ეს მეთოდი არ საჭიროებს 12ვ ძაბვას, რადგან **ISP** ინტერფეისს გააჩნია ძაბვის კონვერტერი და, შესაბამისად, 5ვ ძაბვა, რომელიც საკმარისია ამ პროცესისათვის. **ISP**-თი დაპროგრამებისას მონაცემები გადაეცემა **SPI** (Serial Peripheral Interface) ინტერფეისის საშუალებით, რომლისთვისაც მხოლოდ ოთხი ხაზია საჭირო - **RESET, SCK, MOSI, MISO.**

ISP-თი დაპროგრამების მეთოდი არ საჭიროებს მიკროკონტროლერის სამუშაო სქემიდან ამოღებას.

AVR მიკროკონტროლერებისათვის დაპროგრამების ყველაზე გავრცელებული ხერხია შიგასქემური დაპროგრამება (ფუნქცია **ISP**) **SPI** კომუნიკაციური ინტერფეისის გამოყენებით. ეს შეუძლია ყველა **AVR** მიკროკონტროლერს და მოსახერხებელია იმით, რომ იძლევა იმ **AVR** მიკროკონტროლერის დაპროგრამებისა და კვლავპროგრამირების საშუალებას, რომელიც განლაგებულია მზა მოწყობილობაში.

2.15 ნახაზზე ნაჩვენებია მიმდევრობითი ინტერფეისით დაპროგრამების ზოგადი პრინციპული სქემა. მიკროკონტროლერის დაპროგრამებისას პროგრამატორი ყოველთვის ფუნქციონირებს, როგორც წამყვანი **(Master)**, ხოლო მიკროკონტროლერი - როგორც ამყოლი **(Slave)** მოწყობილობა.



ნახ. 2.15. ATMega-ს პროგრამირება

SPI ინტერფეისი შედგება სამი ხაზისაგან: **SCK, MISO** და **MOSI**, სადაც **SCK (SPI ClocK)** ტაქტირების სიგნალია, რომელსაც პროგრამატორი აფორმირებს **SCK** ხაზზე.

MOSI (Master Out, Slave In – წამყვანის გამოსასვლელი, ამყოლის შესასვლელი) არის მონაცემთა გადაცემის ხაზი პროგრამატორიდან (წამყვანი) დასაპროგრამებელ მიკროკონტროლერამდე (ამყოლი). ყოველი იმპულსის განმავლობაში **SCK** ხაზზე, **MOSI**-ის ხაზის გამოყენებით, გადაიცემა ერთი ბიტი პროგრამატორიდან დასაპროგრამებელ მიკროკონტროლერში.

MISO (Master In, Slave Out - წამყვანის შესასვლელი, ამყოლის გამოსასვლელი) არის მონაცემთა გადაცემის ხაზი დასაპროგრამებელი მიკროკონტროლერიდან (ამყოლი) - პროგრამატორამდე (წამყვანი). ყოველი იმპულსის განმავლობაში **SCK** ხაზზე გადაიცემა ერთი ბიტი დასაპროგრამებელი მიკროკონტროლერიდან პროგრამატორში, **MISO** ხაზის გამოყენებით.

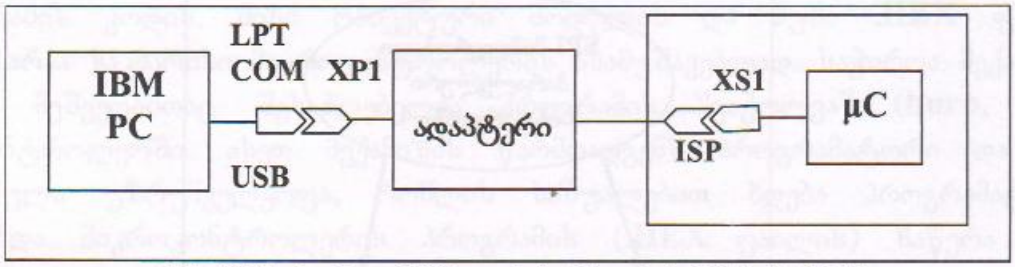
იმისათვის, რომ უზრუნველვყოთ ნორმალური კავშირი **SPI**-ს სამი ხაზის გამოყენებით, აუცილებელია გავაერთიანოთ მიწის სალტე (**GND**) პროგრამატორსა და დასაპროგრამებელ მოწყობილობაზე.

მიმდევრობითი დაპროგრამების რეჟიმში შესასვლელად და მასზე ყოფნისათვის გამოიყენება ჩამოყრის ხაზი (**RESET**). ის უნდა იყოს აქტიურ მდგომარეობაში (სიგნალის დაბალი დონე) **AVR**-ის დაპროგრამებისა და ჩიპის წაშლის დროს. **RESET** ხაზზე წაშლის ბოლოს უნდა იყოს ფორმირებული შესაბამისი იმპულსი.

პროგრამატორით კონტროლერის ტაქტირებისათვის შესაძლებელია გამოვიყენოთ მიკროკონტროლერის **XTAL1** გამოსასვლელი, კვარცული რეზონატორის არარსებობის დროს.

ამგვარად, პრაქტიკულად ყველა **AVR**-კონტროლერს აქვს შიგასქემური დაპროგრამების **ISP** ფუნქცია. ეს ნიშნავს, რომ პროგრამის კოდების "გაჭოლვისათვის" არ არის აუცილებელი მიკროკონტროლერის პლატადან ამოღება და მისი პროგრამატორის ბუდეში მოთავსება.

კომპიუტერი, სპეციალური ადაპტერის საშუალებით, უერთდება **ISP** გასართს, რომელიც დაყენებულია ნაკეთობის პლატაზე (ნახ. 2.16).



ნახ. 2.16. მიკროკონტროლერის პროგრამირების სისტემა

ადაპტერი ისეა მოწყობილი, რომ დაპროგრამების დამთავრების შემდეგ ავტომატურად ითიშება მიკროკონტროლერის გამომყვანებიდან და არ უშლის პლატაზე დანარჩენი კვანძების მუშაობას. შედეგად მრავალჯერადი ექსპერიმენტებისა და "გაჭოლვების" პროცესი მიმდინარეობს გაცილებით უფრო სწრაფად.

კავშირის სალტე მიკროკონტროლერსა და ადაპტერს შორის გადასცემს ექვს სიგნალს (ცხრილი 5): სამი შესავალი, ერთი გამოსავალი და ორი კვების. ინფორმაცია გადაიცემა მიმდევრობით **SPI** პროტოკოლის შესაბამისად.

ცხრილი 5

სიგნალი	გაშიფვრა	ფუნქცია	დანიშნულება
SCK	Serial ClocK	მკ-ს შესასვლელი	მკ-ს ტაქტირების სიგნალი
MOSI	Master Out - Slave In	მკ-ს შესასვლელი	საინფორმაციო სიგნალი მკ-ში
MISO	Master In - Slave Out	მკ-ს გამოსასვლელი	საინფორმაციო სიგნალი მკ-დან
	GrouND	დამიწება	საერთო ხაზი
RES	RESet	მკ-ს შესასვლელი	ლოგიკური "0" - დაპროგრამება
VCC	Voltage Common Collector	კვება	კვების ძაბვა 2,7...5,5ვ

მონაცემების სიგნალების ყოველი ხაზი უერთდება მიკროკონტროლერის გარკვეულ გამომყვანს პორტის ხაზთან, რომელსაც აქვს ალტერნატიული დასახელება: **MISO, MOSI** ან **SCK** (ცხრილი 6). გამომყვანების ორმაგი დანიშნულება აღებულია **MCS-51** პლატფორმის შესაბამისად. **AVR**

მიკროკონტროლერების მრავალსახეობის მიუხედავად **SPI** გამომყვანები მათში ხისტადაა გამოყოფილი.

ცხრილი 6

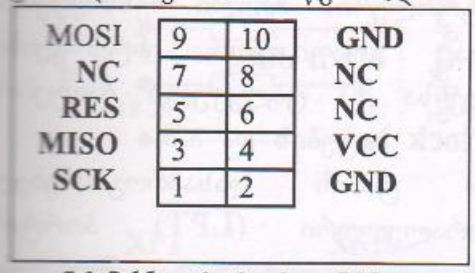
მკ-ს კორპუსი	SPI ინტერფეისის სიგნალები და მიკროკონტროლერის გამომყვანები					
	VCC	GND	RES	SCK	MOSI	MISO
DIP-28	7	8	1	19	17	18

XP1, XS1 გასართი (ნახ. 2.16) 10-კონტაქტიანია. გასართის ჩანგალი პლატაზე **BH-10** ტიპისაა, ხოლო როზეტი განკუთვნილია **IDC-10F** ბრტყელი კაბელისათვის. გამომყვანების **განრჩილვის** უნიფიკაცია არ არსებობს. მე-7 ცხრილში ნაჩვენებია ყველაზე ხშირად გამოყენებული ვარიანტი, რომელიც დამუშავებულია **Altera** ფირმის მიერ დაპროგრამებადი ლოგიკური ინტეგრალური სქემებისათვის (**EPLD**) **ByteBlaster** ადაპტერის გამოყენებით.

ცრილი 7

IDC-10 გასართის როზეტის კონტაქტები	1	2	3	4	5	6	7	8	9	10
ინტერფეისი ByteBlaster	SCK	GND	MISO	VCC	RES	NC	NC	NC	MOSI	GND

ამგვარად, საჭიროა გავითვალისწინოთ, თუ უცნობ პლატაზე **AVR**-კონტროლერით განლაგებულია **ISP** ინტერფეისის 10-კონტაქტიანი გასართი, პირველ რიგში, უნდა დავადგინოთ მისი **დაცოკოლება** პროგრამატორის ჩართვამდე. **"ByteBlaster"**-ის განრჩილვის უპირატესობა გამოიხატება მის უნივერსალობაში, ვინაიდან ერთი მოწყობილობით შეიძლება მიკროკონტროლერის "გაჭოლვა" და დაპროგრამებული ლოგიკური ინტეგრალური სქემების კვლავპროგრამირება. ეს მოითხოვს სპეციალურ პროგრამულ უზრუნველყოფას.



ნახ.2.16. გასართის განრჩილვა

განხილული განრჩილვის ვარიანტი არის კონტაქტების განლაგება გასართში (ნახ. 2.16), რომელიც არავითარ გავლენას არ ახდენს შესაბამისი სიგნალების ელექტრულ პარამეტრებზე.

2.5.1. პროგრამატორის სქემის შერჩევა

პროგრამატორის ელექტრული სქემის შერჩევა დამოკიდებულია სამ ფაქტორზე:

- მმართველ კომპიუტერულ პროგრამაზე, რომლის დანიშნულება მიკროკონტროლერის "გაჭოლვაა";
- პორტის ტიპზე, რომელიც კომპიუტერში (**COM, LPT** ან **USB**) გამოიყენება;
- სერვისული და დამცავი ფუნქციების დონეზე.

ცნობილია რამდენიმე მმართველი კომპიუტერული პროგრამა, რომლებიც გამოდგება **AVR**-თან მუშაობისათვის. მაგალითად, **AVReAL, IC-Prog, PonyProg, Willem Eprom**. ყველა უფასოა და უზრუნველყოფს **AVR** კონტროლერების

ფართო ნომენკლატურას. შესაბამისი ადაპტერის დამზადებისას შესაძლებელია ნებისმიერის გამოყენება.

AVReAl დანიშნულია **AVR** კონტროლერების დაპროგრამებისათვის **SPI** ინტერფეისის გამოყენებით. ეს პროგრამა ადვილად გამოსაყენებელია და უზრუნველყოფს მარტივი პროგრამატორების ძირითადი ტიპების წარმატებულ მუშაობას, რომლებიც კომპიუტერის პარალელურ (**LPT**) პორტს უერთდება.

არსებობს პროგრამის **Win** ვერსია, სადაც მისი გაშვება ხდება კომანდური სტრიქონიდან აუცილებელი გასაღების გამოყენებით. გრაფიკული ინტერფეისი პროგრამას არ აქვს.

IC-Prog მუშაობს **LPT**-პროგრამატორებთან ერთად. კონტროლერის დაპროგრამება ხორციელდება **SPI** ინტერფეისის გამოყენებით. აქვს გრაფიკული ინტერფეისი.

PonyProg – არის თავისუფალი სერვისული პროგრამა (უტილიტა) ღია საწყისი კოდით მიმდევრობითი შეღწევის **Flash**-მიკროსქემების დაპროგრამებისათვის. არსებობს **PonyProg**-ის **Windows** ვერსია. პროგრამას შეუძლია გამოიყენოს სტანდარტული მიმდევრობითი (**COM**) ან პარალელური (**LPT**) პორტი და ახორციელებს პროგრამატორების მხარდაჭერას.

AVRDUDE უტილიტა შედის **WinAVR**-ის პაკეტის შედგენილობაში. აღნიშნული უტილიტა ტვირთავს პროგრამებსა და მონაცემებს მიკროკონტროლერის მეხსიერებაში და შეუძლია იქ არსებული პროგრამებისა და მონაცემების წაკითხვა. **AVRDUD** იყენებს **SPI**-ინტერფეის.

იმისათვის, რომ ამოვიკითხოთ ან ჩავწეროთ მონაცემები კონტროლერის მეხსიერებების ყველა ტიპში **AVRDUD** უნდა მუშაობდეს კომანდური სტრიქონის გამოყენებით ან გრაფიკული ინტერფეისის საშუალებით. **AVRDUDE**ს გამოყენება კომანდური სტრიქონიდან მოსახერხებელია კონტროლერის მთელი მეხსიერების სრული დაპროგრამებისათვის, მაშინ, როდესაც გრაფიკული ინტერფეისი სასარგებლოა მეხსიერების შიგთავსის გამოკვლევისა და ცალკეული ბაიტების შეცვლისას **EEPROM**-ში, მაგალითად, **fuse** და **lock** ბიტების და ა.შ.

AVRDUDE დაკავშირებულია სხვადასხვა ტიპის დასაპროგრამებელ მოწყობილობებთან, რომლებიც მუშაობენ პარალელური (**LPT**) პორტის ინტერფეისისა და მიმდევრობითი (**COM**) პორტის გავლით. უტილიტა შეიძლება იყოს კონფიგურირებული ისე, რომ იმუშაოს ნებისმიერ **LPT**-მოწყობილობასთან.

PonyProg პროგრამის მუშაობა

ამ პროგრამის გამოყენება განპირობებულია იმით, რომ **PonyProg**-ის მხარდაჭერა ჩადებულია **C**-კომპილატორებში და პროგრამების გამმართველებში.

კომპიუტერთან მიერთების პორტი შეირჩევა სისტემური კონფიგურაციის მიხედვით. თუ **LPT**-პორტი მუდმივად დაკავებულია პრინტერით, აირჩევა **COM**-პორტი და პირიქით. შემაერთებელი კაბელის სიგრძე ადაპტერსა და **COM**-პორტს შორის 5...8 მეტრს შეადგენს, **LPT**-პორტის შემთხვევაში – 1,5...2 მეტრს. მეორე მხრივ, **LPT**-ადაპტერების ელექტრონული სქემები უფრო მარტივია და ნაკლებ დეტალებს შეიცავს.

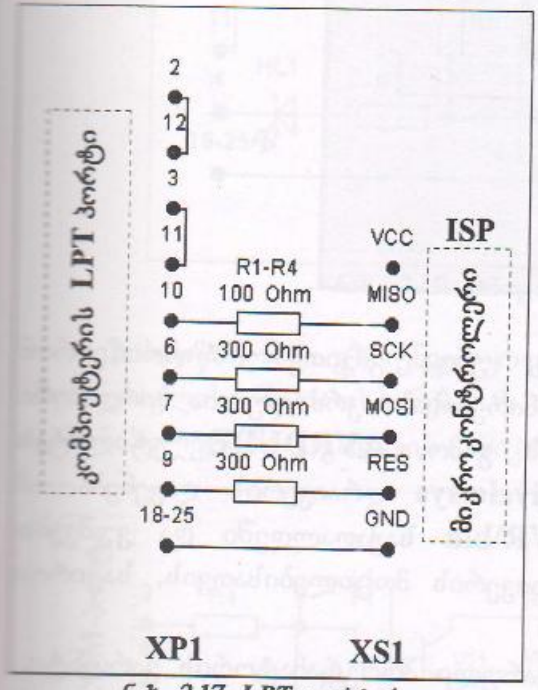
ამას გარდა, არსებობს ადაპტერები, რომლებიც **USB**-პორტს უერთდება. ადაპტერის შერჩევის საბოლოო გადაწყვეტილება მიიღება მათი სერვისული და დამცავი ფუნქციების გათვალისწინებით.

მარტივი AVR-პროგრამატორი

განვიხილოთ **LPT** და **COM** ადაპტერები იმ მოდიფიკაციებით, რომლებიც მათი დაცვის ხარისხისა და სერვისის შესაძლებლობებს ზრდის.

ერთ-ერთი მარტივი პროგრამატორი შედგება 25-კონტაქტიანი გასართისგან, მიმდევრობითი **(LPT)** პორტისაგან, ოთხი რეზისტორისა და შემაერთებელი კაბელისაგან, რომლის სიგრძე 1,5მ არ უნდა აღემატებოდეს. რეზისტორები აუცილებელია მიმდევრობითი პორტის დაცვისათვის.

LPT-ადაპტერის ელექტრული სქემა ნაჩვენებია 2.17 ნახაზზე. რეზისტორები **R1-R4** ზღუდავს ექსტრადენებს და ამცირებს "ჟღრიალს" სიგნალების იმპულსების ფრონტზე. კომპიუტერის პარალელური პორტის **XP1** მისაერთებელ გასართზე დამოკლებული გამომყვანები 2-12 და 3-11 იძლევა პროგრამისათვის აპარატული ნაწილის (ადაპტერის) არსებობის იდენტიფიკაციის საშუალებას.



ნახ. 2.17. LPT-ადაპტერი

შემაერთებელი კაბელის სიგრძე უნდა იყოს, რაც შეიძლება მოკლე. სასურველია ლენტისებრი კაბელის გამოყენება, რომელშიც აუცილებელია მონაცემების სიგნალების ხაზები ჩავანაცვლოთ "მიწის" ხაზებით.

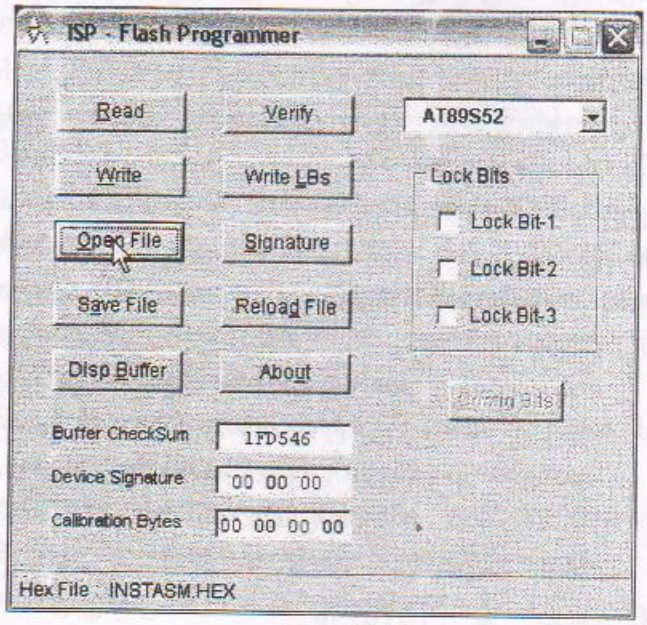
პროგრამატორი დაკავშირებულია მიკროკონტროლერთან **ISP** კონექტორის საშუალებით. სიმარტივის მიუხედავად, ასეთი ადაპტერი მდგრადად მუშაობს კომპიუტერების უმრავლესობასთან.

აღსანიშნავია, რომ სქემას არა აქვს შუალედური ბუფერი და გალვანური განრთვა კომპიუტერის პარალელური პორტის მიმართ. ამის გამო, პარალელური პორტის დაზიანების გამორიცხვისათვის, შემაერთებელი კაბელის ჩართვისა და გამორთვის დროს, საჭიროა მიკროკონტროლერის პლატას ძაბვის გამორთვა. როდესაც პროგრამატორი მიერთებულია პარალელურ პორტზე, უნდა ავარიდოთ მისი გამომყვანები მოკლე ჩართვას.

არსებობს საკმაოდ ბევრი პროგრამა, რომლებითაც ხდება მომზადებული პროგრამული კოდის ჩაწერა და მართვა მიკროკონტროლერში. ამ პროგრამების ძირითადი ამოცანაა კომპიუტერის პორტის სწორი მართვა და პროგრამის ბაიტების სწორი მიმდევრობითი გადაწოდება დაპროგრამების ინტერფეისში, მიკროკონტროლერის მოდელისა და დაპროგრამების არჩეული პრინციპის წინასწარ დადგენილი ოქმის შესაბამისად.

განვიხილოთ ერთ-ერთი ასეთი პროგრამა. 2.18 ნახაზზე ნაჩვენებია **ISP** ფლეშ-პროგრამატორის პროგრამის ფანჯრის ინტერფეისი. იმისათვის, რომ მისი საშუალებით მოხდეს მიკროკონტროლერის დაპროგრამება, საჭიროა

პროგრამატორის ინტერფეისის მიერთება კომპიუტერზე, დასაპროგრამებელი მიკროკონტროლერის მოდელის არჩევა ჩამოსაშლელი სიიდან, შემდეგ ჩასაწერი პროგრამის **.HEX** ფაილის გახსნა და **Write** ღილაკზე დაჭერით დაპროგრამების პროცესის დაწყება. დაპროგრამებას, როგორც წესი, სჭირდება გარკვეული დრო, რომელიც მერყეობს რამდენიმე წამიდან რამდენიმე წუთამდე. დაპროგრამების პროცესის დასრულების შემდეგ ხდება გადამოწმება, კერძოდ, მიკროკონტროლერში რეალურად ჩაწერილი პროგრამისა და მისი ორიგინალის შედარება.



ნახ. 2.18

თუ გამოვიყენებთ **AVRDUDE** პროგრამას, უნდა გავითვალისწინოთ, რომ **Windows XP** ოპერაციული სისტემის ფუნქციონირებისას აკრძალულია პირდაპირი მუშაობა პარალელურ (**LPT**) პორტთან. ამის გამო, **AVRDUDE** უტილიტას ნორმალური მუშაობისათვის აუცილებელია **giveio.sys** დრაივერის დაყენება. ამ დრაივერის დასაყენებლად შევდივართ **WinAVR\bin** საქაღალდეში და ვუშვებთ **install_giveio.bat** ფაილს (სისტემიდან ამ დრაივერის მოცილებისათვის, საჭიროა გავუშვათ **remove_giveio.bat** ფაილი).

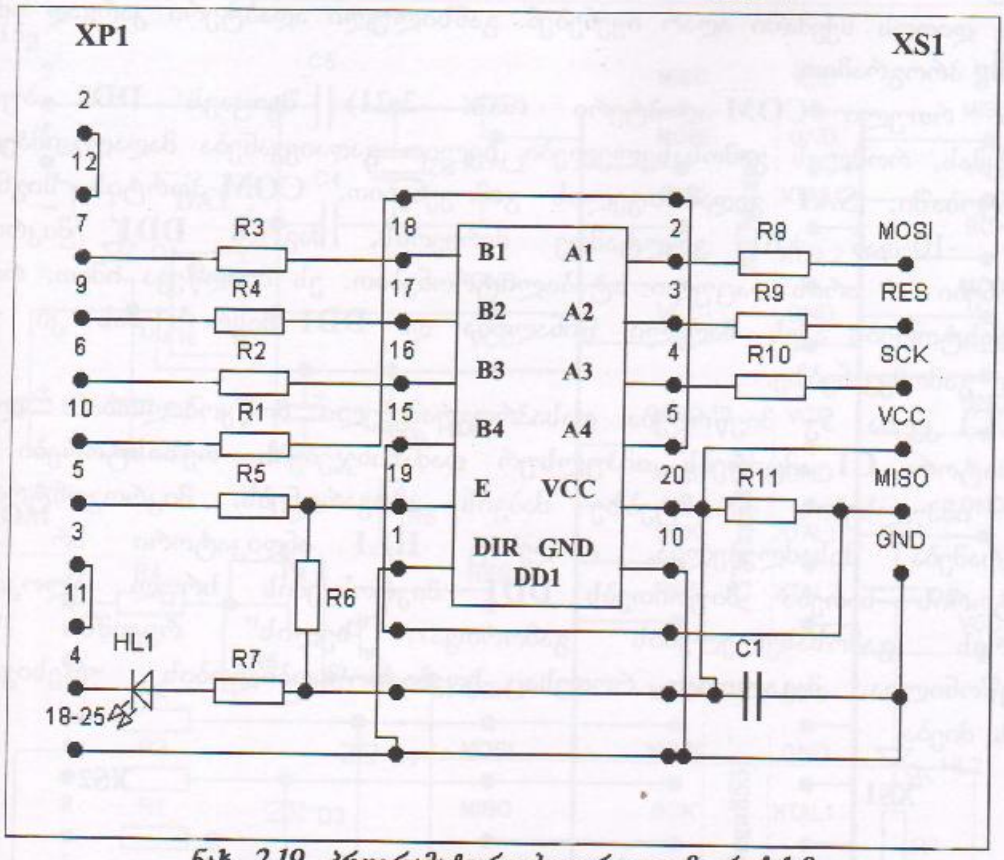
ნახაზებზე წარმოდგენილია რამდენიმე სტანდარული პროგრამატორის ვარიანტი.

2.19 ნახაზზე ნაჩვენებია ინტელექტუალური **LPT**-ადაპტერის სქემა, რომლის გამოსასვლელები ავტომატურად გადაიყვანება მაღალიმპედანსურ მდგომარეობაში, დაპროგრამების დამთავრების შემდეგ.

მმართველი სიგნალის გავლის გზა შემდეგია: **XP1** გასართის მე-5 კონტაქტი, რეზისტორი **R5**, **DD1** (**SN74LS245**) მიკროსქემის მე-19 გამომყვანი. "0" და "1" ლოგიკური დონეების შეცვლას ახორციელებს **PonyProg** პროგრამა. ამას გარდა, პროგრამა დაპროგრამების პროცესში აყენებს ლოგიკურ "0"-ს **XP1** გასართის მე-4 კონტაქტზე, რაც იწვევს **HL1** ინდიკატორის ნათებას.

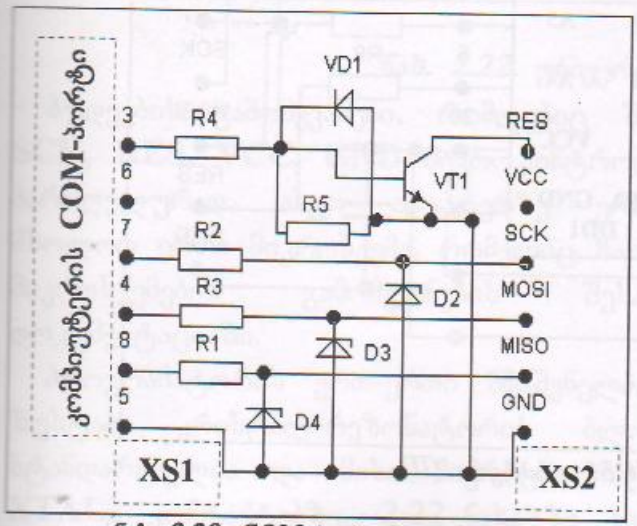
DD1 მიკროსქემაში მძლავრი ბუფერული გამმეორებლების არსებობა, ერთი მხრივ, აუმჯობესებს სიგნალების ფრონტების დახრილობას, მეორე მხრივ, არაპირდაპირი გზით იცავს კომპიუტერს ავარიული ძაბვის გადამეტებისგან. **R1**-

R5, R8-R10 რეზისტორები აუმჯობესებს იმპედანსურ შეთანხმებას და კრიტიკულ სიტუაციაში შეიძლება იმუშაოს, როგორც დნობადმა მცველმა.



ნახ. 2.19. პროგრამატორი ბუფერული მიკროსქემით

რეზისტორი **R6** უზრუნველყოფს ლოგიკურ "1"-ს მიკროსქემის 1 შესასვლელზე, კაბელის შეპირისპირებისას **LPT**-პორტიდან. ამით მიკროსქემის ბუფერების გამოსასვლელები გადადის მიკროკონტროლერის **SPI** ინტერფეისის ხაზიდან გამორთულ მდგომარეობაში. **DD1** მიკროსქემის კვება (**VCC=5v**) მიეწოდება დასაპროგრამებელი მოწყობილობის პლატადან. **C1** კონდენსატორი გამოიყენება ბლოკირებისათვის, რის გამოც იგი უნდა განლაგდეს **DD1** მიკროსქემის მე-10 და მე-20 გამომყვანების სიახლოვეს. განხილული ადაპტერი, ბუფერული ლოგიკური მიკროსქემით, ეფექტურია ძლიერი ინდუსტრიული ხელის შემშლელ პირობებში.



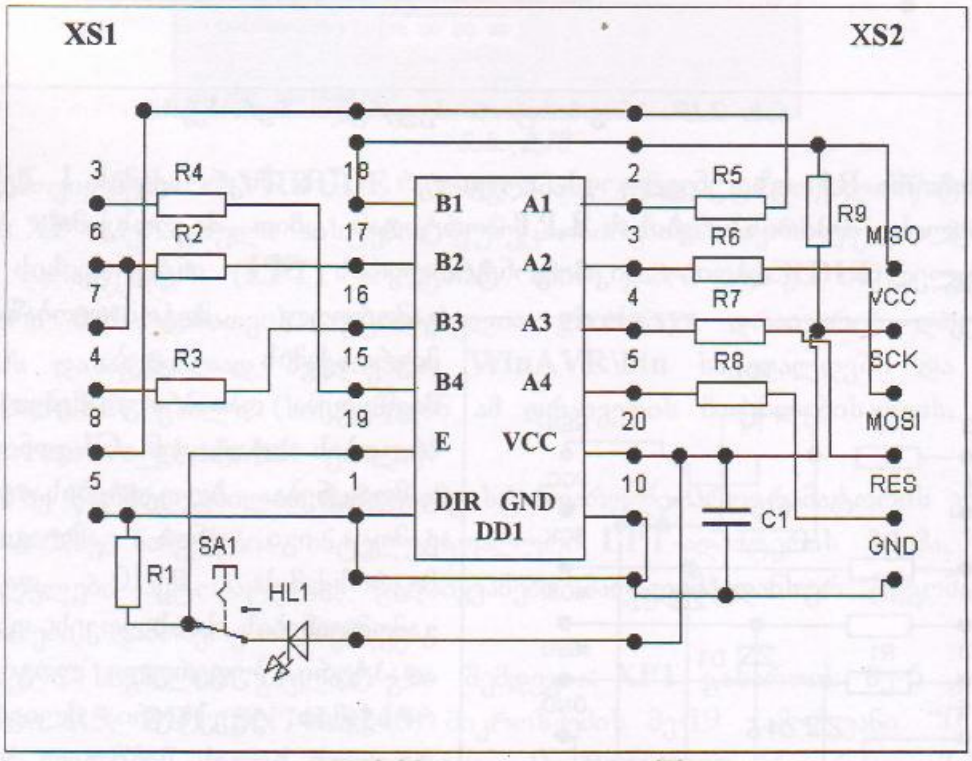
ნახ. 2.20. COM-პორტის ადაპტერი

2.20 ნახაზზე ნაჩვენებია **COM**-პორტისათვის დამუშავებული ადაპტერის ელექტრული პრინციპული სქემა. ელემენტების დანიშნულება: **R1-R4** რეზისტორები ზღუდავს დენებს, სტაბილიტრონები **D2-D4** კი - ძაბვებს (სტაბილური 4,7ვ), რეზისტორი **R5** კეტავს **VT1** ტრანზისტორს კომპიუტერიდან

კაბელის გათიშვის შემდეგ. **VD1** დიოდი ზღუდავს უარყოფითი პოლარობის ძაბვას, რომელიც **COM**-პორტიდან მიიღება. თუ **VT1** ტრანზისტორის დასაშვები ძაბვა მაღალია, დიოდს სქემაში აღარ იყენებენ. განხილული ადაპტერი კარგად იმართება **PonyProg** პროგრამით.

უფრო რთული **COM**-ადაპტერი (ნახ. 2.21) შეიცავს **DD1** ბუფერულ მიკროსქემას, რომლის გამოსასვლელები ხელით გადაიყვანება მაღალ იმპედანსურ მდგომარეობაში, **SA1** გადამრთველის გამოყენებით. **COM**-პორტის სიგნალების დონეები -10-დან +10 ვოლტამდე მერყეობს, მაგრამ **DD1** მიკროსქემის გამომყვანები არ არის დაცული სტაბილიტრონებით. ეს მიიღწევა იმით, რომ **R1-R4** რეზისტორებს აქვს მაღალი წინაღობა და **DD1** მიკროსქემას კი – შიგა დიოდები გამომყვანებზე.

(**VCC**) კვება 5ვ მიეწოდება დასაპროგრამებელი მოწყობილობის პლატადან. კონდენსატორი **C1** ამცირებს იმპულსურ დამახინჯებებს. რეზისტორები **R5-R9** ახშობს იმპულსების ფრონტებზე ძაბვის ამოვარდნებს. მიკროკონტროლერის დაპროგრამება შესაძლებელია, როდესაც **HL1** ინდიკატორი არ ანათებს. ინდიკატორის ნათება მიუთითებს **DD1** მიკროსქემის სრულ ბლოკირებაზე. ადაპტერის გამოსასვლელების გამორთვა "ხელის" რეჟიმში ზოგჯერ მიზანშეწონილია, მაგალითად, როდესაც ხდება მოწყობილობის უწესივრობების მიზეზის ძიება.

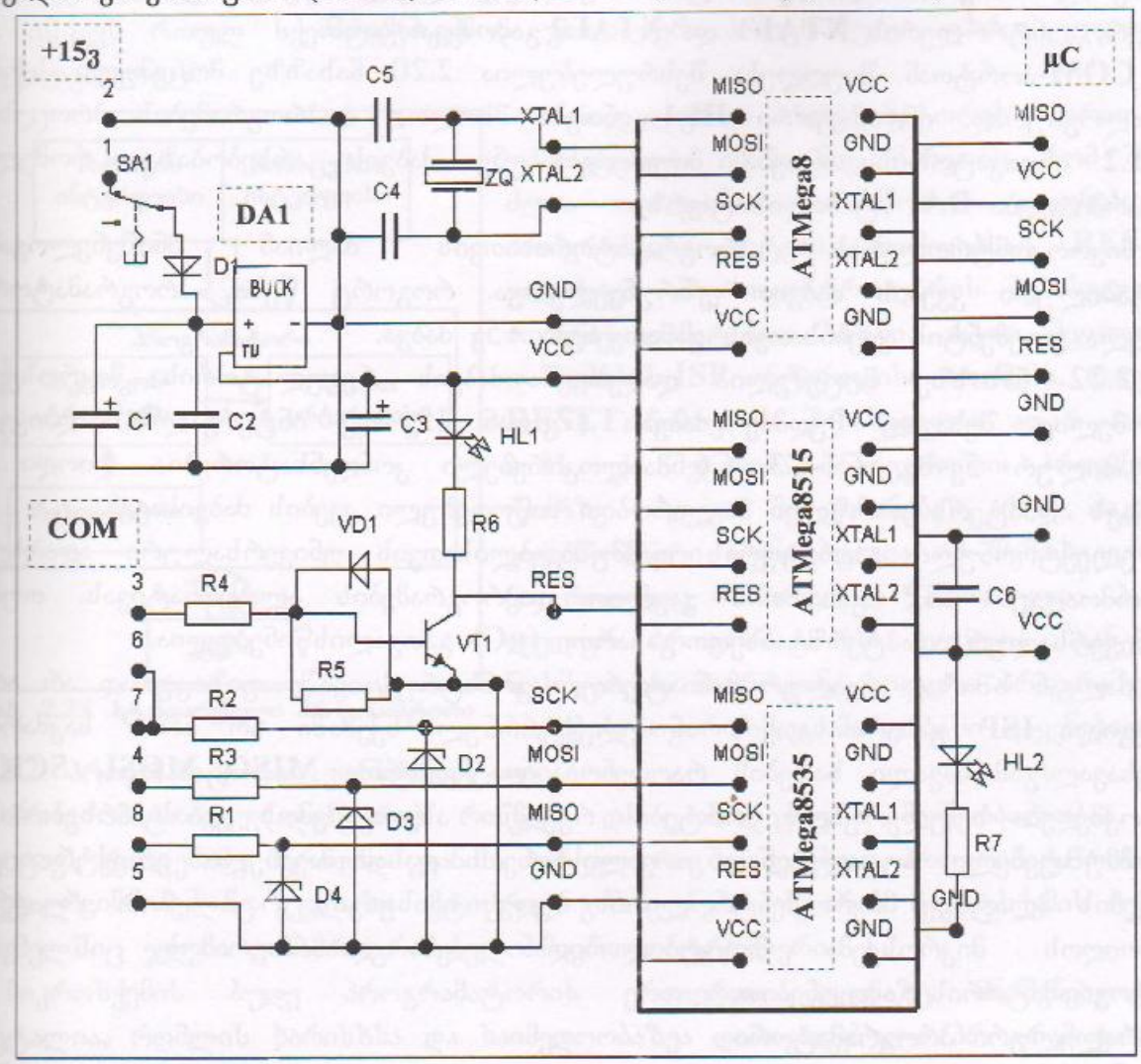


ნახ. 2.21. COM-ადაპტერი (II)

უნივერსალური პროგრამატორი

დიდი რაოდენობის სხვადასხვანაირი **AVR**-კონტროლერების დაპროგრამების დროს, ან მათი საწყისი კონტროლირებისას, მიზანშეწონილია უნივერსალური

ადაპტერის გამოყენება, რომელშიც მიკროსქემების ბუდეები განლაგებულია სხვადასხვა კორპუსში (**DIP-8, DIP-20, DIP-28, DIP-40**) (ნახ.2.22).



ნახ. 2.22. უნივერსალური ადაპტერი

ბუდეების გამომყვანები, რომლებიც შეესაბამება მარკირებით **MISO, MOSI, SCK, RES, VCC, GND** მიკროკონტროლერების გამომყვანებს, შეერთებულია პარალელურად. ასეთ ადაპტერზე ერთდროულად შეიძლება დაპროგრამდეს მხოლოდ ერთი მიკროსქემა, რომელიც ჩასმულია შესაბამის ბუდეში. ბუდეებისა და მიკროსქემების გამომყვანების შესაბამისობა მოცემულია ტექნიკურ დოკუმენტაციაში.

პროგრამატორის ერთ-ერთი მნიშვნელოვანი ელემენტი ოსცილატორია, უფრო ზუსტად კრისტალ-რეზონატორის ბლოკი. როგორც წესი, მისი სქემაც სტანდარტულია და მისი მიერთება ხდება მიკროკონტროლერის **XTAL1** და **XTAL2** კონტაქტებზე. 2.22 ნახაზზე ნაჩვენებია გარე რეზონატორის ტიპური სქემა. კვარცული რეზონატორი **ZQ, C4** და **C5** კონდენსატორებთან ერთად შეადგენს მიკროკონტროლერის გარე ამგზნები გენერატორის ტიპურ სქემას. ზოგიერთი ტიპის მიკროკონტროლერი შეიძლება დაპროგრამდეს რეზონატორის გარეშე, შიგა **RC**-გენერატორის გამოყენებით, მაგრამ უნივერსალურ

პროგრამატორში გარე გენერატორი აუცილებელია. ასეთი გენერატორის მდგრადი გენერაციისათვის **ZQ** რეზონატორი უნდა განლაგდეს, რაც შეიძლება ახლოს მიკროკონტროლერის **XTAL1** და **XTAL2** გამომყვანებთან.

COM-პორტთან შეუღლება შესრულებულია 2.20 ნახაზზე მოცემული სქემის ანალოგიურად. ინდიკატორი **HL1** ინთება მხოლოდ დაპროგრამების პროცესში. **HL2** ინდიკატორის ანთება მიუთითებს 5ვ ძაბვის არსებობაზე, რომელიც ფორმირდება **DA1** სტაბილიზატორზე.

მიკროკონტროლერის პროგრამატორისათვის ძალიან მნიშვნელოვანია სტაბილური კვების ძაბვით უზრუნველყოფა. როგორც წესი, პროგრამატორის ყველა ელემენტის კვებისათვის გამოიყენება +5ვ ძაბვა.

2.22 ნახაზზე მოცემულია დადებითი ძაბვის რეგულატორის მიკროსქემა, რომელიც შესავალ მუდმივ ძაბვას (7ვ-დან 15ვ-მდე ინტერვალში) დაიყვანს სტაბილურ 5ვ-ზე. **C1-C3, C6** მაბლოკირებელი კონდენსატორებია. დიოდი **D1** იცავს ადაპტერს არასწორი პოლარობით მიწოდებული კვების ძაბვისაგან.

ელექტრონული აპარატურის დამმუშავებლისათვის უნივერსალური ადაპტერი წარმოადგენს **ISP** გასართის გავლით დაპროგრამების ალტერნატივას. თუმცა უნივერსალურ ადაპტერში ასეთი გასართი (**µC**) გათვალისწინებულია.

ძალიან ხშირად რეალურ მინიატურულ კონსტრუქციებში ფიზიკურად არ არის ადგილი **ISP** გასართისათვის ან ელექტრონულ სქემაში არ არის საკმარისი შესავალ/გამოსავალი ხაზების რაოდენობა და საჭიროა **MISO, MOSI, SCK**-ის ამოქმედება სრულყოფილი პორტების რეჟიმში. ასეთი შემთხვევების არსებობისას დამმუშავებელი აუცილებლად ითვალისწინებს დასამუშავებელი ელექტრონული სქემის პლატაზე მიკროკონტროლერის ბუდეს იმისათვის, რომ ნებისმიერ დროს ამოიღოს მიკროსქემა მოწყობილობიდან და დააპროგრამოს ის გარე პროგრამატორის გამოყენებით.

პარალელური პროგრამატორი

2.23 ნახაზზე ნაჩვენებია სტანდარტული პროგრამატორის სქემა და მისი ძირითადი კვანძები.

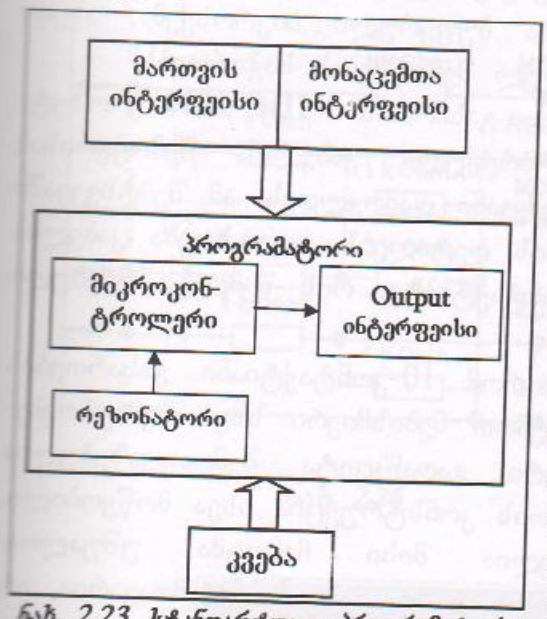
გარდა ამისა, პროგრამატორს შეიძლება გააჩნდეს მონაცემების გამოტანის (**output**) დამხმარე, ჩაწერილი პროგრამის დამატესტირებელი ინტერფეისი. ტექნიკურად ეს შეიძლება იყოს **LED** ინდიკატორები ან **LCD** დისპლეი, სხვადასხვა სახის ბიპერი და ა.შ.

თუ პროგრამატორი იყენებს მიმდევრობითი **SPI** ინტერფეისის სიგნალებს, მაშინ მას უწოდებენ მიმდევრობით პროგრამატორს.

პარალელური პროგრამატორები აღწერილია **ChaN** საიტზე (http: elm-chan.org/works/avrx/report.e.html, იაპონია), სადაც მოცემულია ელექტრული პრინციპული სქემა 20- და 8-გამომყვანიანი მიკროსქემებისათვის. იქვეა მოცემული მმართველი პროგრამა **WinXP** ოპერაციული სისტემისათვის (http: elm-chan.org/works/avrx/avrxtool32.zip).

DIP-28, DIP-40 კორპუსებში მიკროკონტროლერები დაპროგრამდება სპეციალური გადამყვანის გამოყენებით 20-კონტაქტიან ბუდესთან მისაერთებლად. პარალელური პროგრამატორი იძლევა მიკროკონტროლერის გაჭოლვის

საშუალებას, მაგრამ მიმდევრობით პროგრამატორთან შედარებით მასში გამოიყენება ბევრად უფრო მეტი კავშირის ხაზი და უფრო მაღალი ძაბვა (12ვ). ეს იწვევს რთულ სქემატექნიკას და უნიკალური პროგრამული უზრუნველყოფის აუცილებლობას.



ნახ. 2.23. სტანდარტული პროგრამატორი

პარალელური პროგრამატორის ძირითადი უპირატესობა იმაში მდგომარეობს, რომ შეუძლია აღადგინოს არასწორად დაყენებული პროგრამული ბიტები, რომლებსაც **RESET** გადაჰყავს მიკროკონტროლერის შესასვლელი პორტის დამატებით ხაზში. ასეთი შეცდომის დაშვებისას **ISP** ინტერფეისი ითიშება, რადგან **RESET** ჩამოყრის შესასვლელი აღარ არსებობს. ეს შეცდომა მიმდევრობით პროგრამატორზე იძლევა შეტყობინებას, რომ მიკროკონტროლერი წუნდებულია იმ შემთხვევაშიც კი, როდესაც ის სრულად გამართულია. პარალელურ პროგრამატორს შეუძლია "განკურნოს" ასეთი მიკროკონტროლერი პროგრამული ბიტების საჭირო მდგომარეობაში დაყენებით.

2.5.2. პროგრამატორის კონსტრუქციები

პროგრამატორი (ადაპტერი) რთული და კომპლექსური მოწყობილობაა. ის შეიცავს რამდენიმე ძირითად კვანძს. პირველ რიგში შევეხოთ პროგრამატორის იმ დეტალს, რომელიც ყველაზე აქტიურად გამოიყენება. ეს არის მიკროსქემის ბუდე, რომელშიც ჯდება დასაპროგრამებელი მიკროკონტროლერი.

მიკროსქემის ბუდე პროგრამატორის ერთ-ერთი ყველაზე მნიშვნელოვანი დეტალია, რომლის ხარისხსა და საიმედოობაზეა დამოკიდებული პროგრამატორის უნარი შეასრულოს თავისი ფუნქცია. როგორც გამოცდილება აჩვენებს, ნებისმიერი პროგრამატორი, მისი სირთულის, ფასისა და ფუნქციონალობის მიუხედავად, უნდა იყოს აღჭურვილი სპეციალური "სოკეტით", რომელიც უზრუნველყოფს მრავალჯერად და საიმედო კონტაქტს დასაპროგრამებელ მიკროსქემასთან.

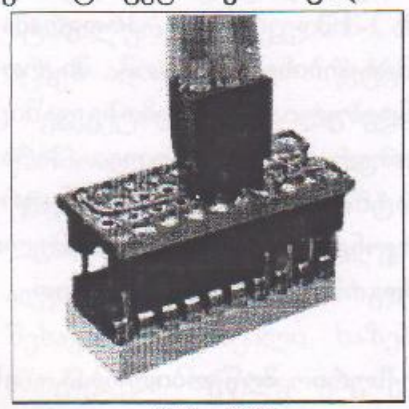

ნახ. 2.24. "სოკეტი"

გამოყენებისათვის ყველაზე მოსახერხებელია ნულოვანი ძალის "სოკეტები" (**ZIF socket**). ეს არის სპეციალური "სოკეტი" პატარა სახელურით, რომლის საშუალებით ხდება დასაპროგრამებელი მიკროსქემის კონტაქტების ფიქსაცია "სოკეტში". სახელურის გახსნისას მიკროსქემა თავისუფლად შედის და გამოდის "სოკეტიდან", ხოლო სახელურის დახურვისას "სოკეტის" კონტაქტები ეჭირება მიკროსქემის კონტაქტებს და უზრუნველყოფს საიმედო კონტაქტს. 2.24 ნახაზზე ნაჩვენებია 40-კონტაქტიანი უნივერსალური **ZIF**-სოკეტი.

პროგრამატორის აწყობა ხდება ცალკე ნაბეჭდ პლატაზე, რომელიც უერთდება 10-კონტაქტიან **ISP** გასართს 20...30სმ და კომპიუტერს 1,5მ სიგრძის კაბელებით. პლატას კორპუსში ათავსებენ.

ზოგიერთ შემთხვევაში პროგრამატორს ათავსებენ უშუალოდ **LPT-** ან **COM-**პორტის გასართში. როდესაც არ გამოიყენება ბუფერული მიკროსქემა, ასეთი პროგრამატორის მუშაობის უნარი განსაკუთრებულ შემოწმებას საჭიროებს.

მიმდევრობით პროგრამატორებში **XS2**-ის გასართის **IDC-10F** როზეტი ერთჯერადი გამოყენებისაა. ამ როზეტში ბრტყელი კაბელის შემოჭერისას შესაძლებელია, რომ კაბელის ზოგიერთი გამომყვანი დამოკლდეს. ამ შემთხვევაში როზეტის ზედა ნაწილს აცილებენ პლასტმასის დეტალებს და ხდება კაბელის მავთულების უშუალო მოკალვა როზეტის კონტაქტებზე, რის შემდეგ მიღებული კონსტრუქცია კომპაუნდით იფარება.



ნახ. 2.25

შესაძლებელია, რომ 10-კონტაქტიანი გასართების ნაცვლად გამოვიყენოთ ნებისმიერი სხვა შემაერთებელი. ორიგინალური გადაწყვეტა შემოთავაზებულია იაპონიაში: გასართის კონსტრუქცია ისეა მოწყობილი, რომ შესაძლებელია მისი ჩამოცმა უშუალოდ მიკროსქემის გამომყვანებზე (ნახ. 2.25). მთავარია, არ მოხდეს აღნიშნული კონსტრუქციის წანაცვლება მიკროსქემის გამომყვანების მიმართ.

როგორც უკვე აღინიშნა, მიკროკონტროლერებისათვის პროგრამების შესადგენად ყველაზე ხშირად სპეციალური ასემბლერები ან **C** ენა გამოიყენება. პროგრამის კოდის შესადგენად გამოიყენება ჩვეულებრივი ტექსტური რედაქტორი და სპეციალიზებული ე.წ. დამუშავების გარსები (Integrated Development Environment - **IDE**), რომლებშიც მუშა კოდის დაწერა და გამართვა გაცილებით ადვილია. ასეთი გარსია, მაგალითად, **AVR Studio**, რომელიც განკუთვნილია **Atmel**-is **AVR** ოჯახის მიკროკონტროლერებისათვის პროგრამების შესადგენად.

დამუშავების გარსები საკმაოდ რთული და კომპლექსური პროექტების შედგენის შესაძლებლობას იძლევა. მათი საშუალებით ადვილია მრავალმოდულიანი პროგრამების შექმნა. პრაქტიკულად ყველა **IDE**-ს, როგორც წესი, გააჩნია კოდის მძლავრი რედაქტორი, რომლის მეშვეობით ხდება კოდის გამოყოფა (**code highlight**), რაც მნიშვნელოვნად ამარტივებს დაწერილი კოდის სინტაქსურ კონტროლს. პროგრამის გამართვისას, დამუშავების გარსის საშუალებით, შესაძლებელია მეხსიერებაში ჩატვირთული პროგრამის კონტროლი რეალურ დროში, მიკროკონტროლერის რეგისტრების მონიტორინგი, შეცვლა და ა.შ.

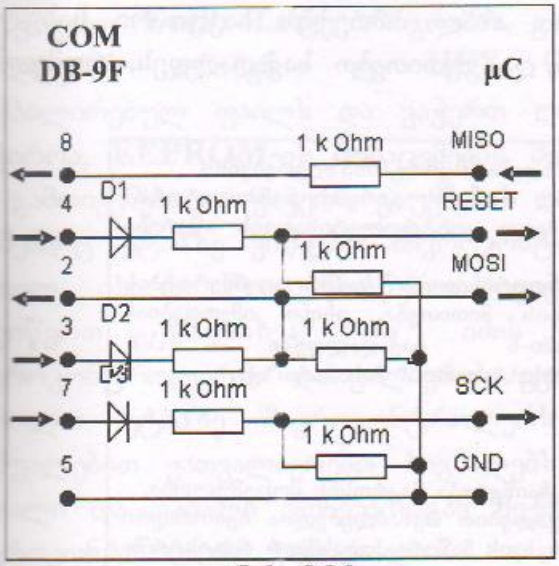
2.5.3. პროგრამატორის დამუშავებისა და გაშვების მაგალითები

განვიხილოთ პროგრამატორი (ნახ. 2.26), რომელიც მუშაობს **COM** პორტის (**RS232**) გავლით და მიკროკონტროლერის გაჭოლვის მოსახერხებელი პროგრამა **UniProf**.

პროგრამატორის აწყობისათვის საჭიროა:

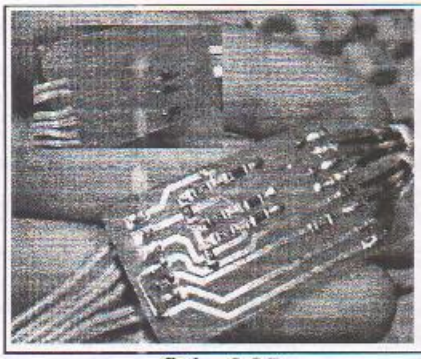
- სამი მცირე სიმძლავრის დიოდი;

- შვიდი რეზისტორი 1კომი ნომინალით;
- სამი 0-ომიანი რეზისტორი ანუ შესაკრავი.

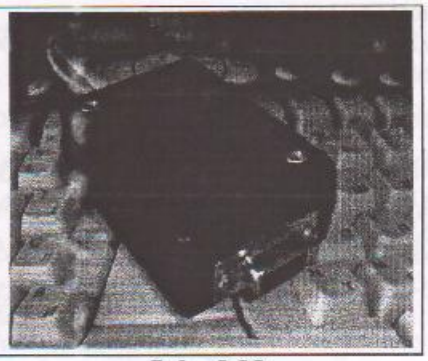


ნახ. 2.26

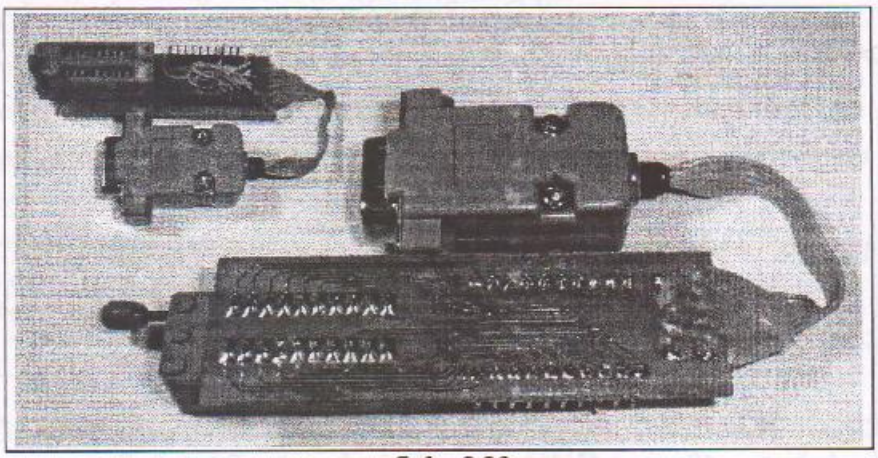
2.27 ნახაზზე ნაჩვენებია აწყობილი ნაბეჭდი პლატა. გამზადებული პლატა თავსდება კორპუსში და კაბელის საშუალებით უერთდება კომპიუტერს. კაბელის სიგრძე ერთ მეტრს არ აღემატება. **DB-9F** გასართს უერთდება კაბელის გამომყვანები ან სტანდარტული ადაპტერი (ნახ. 2.28). მეორე კაბელის სიგრძე, პროგრამატორიდან მიკროკონტროლერამდე, 10-20სმ არ უნდა აღემატებოდეს. აწყობილი პროგრამატორი შესაბამისი კაბელებით ნაჩვენებია 2.29 ნახაზზე.



ნახ. 2.27



ნახ. 2.28



ნახ. 2.29

პროგრამატორის აწყობისა და მასში მიკროკონტროლერის მოთავსების შემდეგ კომპიუტერზე გაიშვება **UniProf.exe** პროგრამა, სადაც აირჩევა კომპიუტერის COM-პორტის ნომერი, რომელთანაც მიერთებულია პროგრამატორი. პროგრამა განსაზღვრავს მიკროკონტროლერის ტიპს, რომელიც ეკრანზე გამონათდება. ტეს-

ტის სახით ვიყენებთ **Atmel AVR Studio**-ს, რომელიც პროგრამული გარსია **AVR** მიკროკონტროლერების დაპროგრამებისათვის.

AVR Studio-ს გაშვების შემდეგ ვქმნით ახალ პროექტს საკუთარი სახელით, ვირჩევთ **Assembler** პროგრამირების ენას და ვუთითებთ საქაღალდეს, რომელშიც ახალი პროექტი განთავსდება.

```
#include <test1> -      ;ამ ბრძანებით ხდება მაკროგანსაზღვრებების
                        სიის მიერთება, რომლის გარეშე კომპილატორმა
                        არ იცის რომელი კონტროლერისათვის იწერება
                        პროგრამა.
                        სხვა კონტროლერისათვის საჭიროა მიერთების
                        სხვა ბრძანების მითითება. ისინი იმყოფებიან
                        AVR Studio-ს საქაღალდეში "AVR
                        Tools\AvrAssembler\Appnotes\" მისამართზე.
MACRO outi
LDI R16,@1
OUT @0,R16
.ENDMACRO -             ;ამ ბრძანებით ჩაიწერება მაკროსის მოცემულობა,
                        რომლის საშუალებით შესაძლებელია ნებისმიერი
                        მოცემული რიცხვის ჩაწერა ნებისმიერ რეგისტრში
                        კოდის ერთი სტრიქონით.
CSEG
ORG 0x0000
RJMP RESET
ORG 0x0030 -            ;კოდის, 0x0030 მისამართით, დაწყების დირექტივა;
                        მისამართი აღებულია დიდი მარაგით, ვინაიდან
                        სხვადასხვა AVR-ის წყვეტების ცხრილი
                        სხვადასხვა ზომისაა.
RESET: -                ;სასტარტო ჭდეა.
OUTI DDRA,0xFF
OUTI DDRB,0xFF
OUTI DDRC,0xFF
OUTI DDRD,0xFF
                        ;ამ ბრძანებებით ხდება პორტების მიმართულე–ბის
                        კონფიგურაცია გამოსასვლელისაკენ.
                        ;თუ მოცემულ კონტროლერს არა აქვს
                        მაგალითად, პორტი C, მაშინ შესაბამის სტრიქონს
                        სჭირდება კომენტარი.
OUTI PORTA,0xAA
OUTI PORTB,0xAA
OUTI PORTC,0xAA
OUTI PORTD,0xAA
                        ;გამოვიტანთ გამოსასვლელებზე 10101010 კოდურ
                        კომბინაციას იმისათვის, რომ მივიღოთ მკაფიო
                        სურათი იმისა, რომ პორტებზე მოხდა
                        ცვლილებები.
                        პროგრამის შესრულების შემდეგ მიკროკონტრო-
                        ლერის გამოსასვლელებზე მონაცვლეობით იქნება
                        კვების ძაბვა ან მიწის პოტენციალი. მისი
                        შემოწმება შეიძლება ან ვოლტმეტრით, ან
                        სასინჯი შუქდიოდით.
RJMP RESET -            ;ამ ბრძანებით ხდება პროგრამის დაციკლვა.
```

ნახ. 2.30. პროგრამის მაგალითი

პროგრამის გამმართველად ვირჩევთ **AVR SIMULATOR**-ს და ვუთითებთ რომელ მიკროკონტროლერთან ვიმუშავებთ. ტექსტის ფანჯარაში ვწერთ მომდევნო პროგრამას. შემდეგ ვაჭერთ კომპილაციის ღილაკს (ან **F7**-ს), შევდივართ

პროექტის საქაღალდეში, სადაც უნდა იყოს ***.hex** ფაილი გამზადებული გაჭოლვის პროგრამით.

ვუშვებთ **UniProf.exe** პროგრამას, გახსნილ ფანჯარაში ვაჭერთ ღილაკს ღია ფოლდერის აღნიშვნით და **HEX** წარწერით. ავირჩევთ პროექტის ახლად კომპილირებულ ფაილს და ვაჭერთ ღილაკს **OK. UniProf** მეორე ფანჯარაში საჭიროა, **EEPROM**-ის მონაცემების შეტანა. განხილულ შემთხვევაში **EEPROM** არ გამოიყენება და ვაჭერთ გაუქმების ღილაკს.

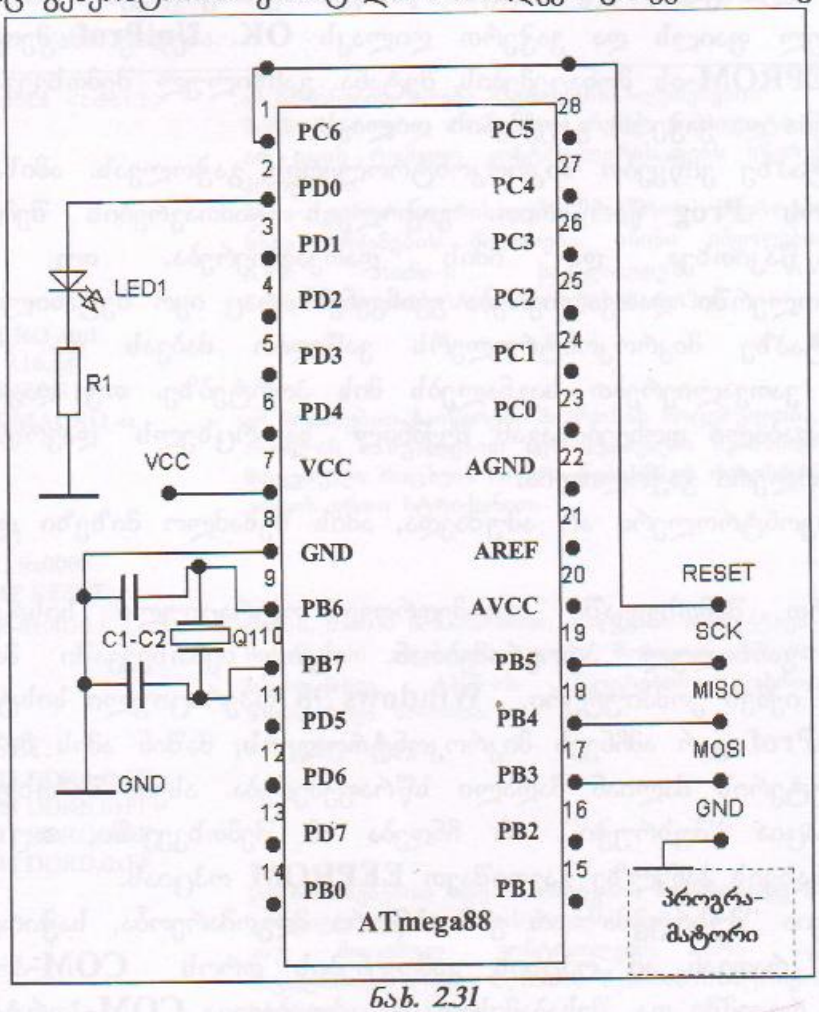
შემდეგ ეტაპზე ვიწყებთ მიკროკონტროლერის გაჭოლვას. ამისათვის, ვაჭერთ წითელ ისარს **Prog** წარწერით. გაჭოლვის დამთავრების შემდეგ შეიძლება დავიწყოთ წაკითხვა და იმის დათვალიერება, თუ რა ჩაიწერა მიკროკონტროლერში. დათვალიერება გვიჩვენებს რაც იყო შეტანილი ფანჯარაში.

ბოლო ეტაპზე მიკროკონტროლერს ვაწვდით ძაბვას და ოსცილოგრაფის საშუალებით ვათვალიერებთ სიგნალებს მის პორტებზე. თუ დავინახავთ ძაბვის მაღალი და დაბალი დონეებისაგან შექმნილ "სავარცხელს" დავრწმუნდებით, რომ მიკროკონტროლერი გაჭოლილია.

თუ მიკროკონტროლერი არ ამუშავდა, ამის შესაძლო მიზეზი და აღმოფხვრის ხერხებია:

- ზოგიერთ შემთხვევაში თანამედროვე ოპერაციული სისტემა არ არის თავსდებადი განხილულ პროგრამებთან. ასეთ შემთხვევაში მიზანშეწონილია გამოყენებულ იქნეს კომპიუტერი, **Windows'98** ოპერაციული სისტემით.
- თუ **UniProf** ვერ ამჩნევს მიკროკონტროლერს, მაშინ ამის მიზეზი შეიძლება იყოს კომპიუტერის ძალიან მაღალი სწრაფქმედება. ასეთ შემთხვევაში საჭიროა ჩავრთოთ ოპცია "მუხრუჭი". ის ჩნდება იმ შემთხვევაში, თუ **EEPROM**-ის მონაცემთა ასახვის პანელზე გავთიშავთ **EEPROM** ოპციას.
- თუ ოპცია "მუხრუჭმა" არ გამოასწორა მდგომარეობა, საჭიროა შეიცვალოს კომპიუტერი, რადგან ამ ოპციის გამოყენების დროს **COM**-პორტი მუშაობს არასაშტატო რეჟიმში და, შესაბამისად, შესაძლებელია **COM**-პორტის შეფერხება.
- თუ კომპიუტერს არ აქვს **COM**-პორტის გასართი კორპუსზე, მაშინ შესაძლებელია ის იყოს **motherboard**-ზე.
- ზოგიერთ შემთხვევაში პრობლემის გადაწყვეტა ხდება გაჭოლვის პროგრამის შეცვლით, მაგალითად, **avrdude**-ის გამოყენებით.
- განხილული სქემა არ მუშაობს **USB-COM** გადამყვანის გამოყენების დროს ან მუშაობს ძალიან ნელა.
- სქემის აწყობის დროს სასურველია კვების მიწოდება კომპიუტერის კვების ბლოკიდან. **+5ვ** და **GND** არის მიწა და კონტაქტი **COM**-პორტიდან და მიიღება გარე დამოუკიდებელი კვების წყაროდან. კომპიუტერიდან **+5ვ** კვება მიიღება წითელი მავთულიდან ნებისმიერ გასართზე და **GND** – კომპიუტერის კორპუსიდან.
- ექსპერიმენტულად დადგენილია, რომ ამ პროგრამატორის სწორი მუშაობისათვის მიკროკონტროლერის კვების ძაბვა უნდა იყოს არანაკლებ **5ვ** და არა უმეტეს **5.5ვ**.
- აწყობილი სქემა მოწმდება რამდენიმეჯერ, ვინაიდან ძირითადი პრობლემები გამოწვეულია არასწორი მონტაჟით.

- მიკროკონტროლერში პროგრამის გაჭოლვამდე საჭიროა პროგრამატორის გათიშვა და **RESET** შესასვლელზე +5ვ ძაბვის მიწოდება 1...10კომ წინაღობიანი რეზისტორის გავლით. ჩართული პროგრამატორის დროს ხდება **RESET** სიგნალის შემცირება, რაც გენერატორის კრისტალს არ აძლევს გაშვების საშუალებას.



ნახ. 2.31

მიკროკონტროლერზე პროექტის შესრულების შემდეგ, მაგალითად, განვიხილოთ შუქდიოდის ციმციმი, რომელიც მიერთებულია მიკროკონტროლერის ერთ-ერთ გამომყვანთან. შევარჩიოთ **ATmega8** მიკროკონტროლერი. ეს მიკროკონტროლერი შეიცავს შეტანა/გამოტანის პორტების საკმარის რაოდენობას და აუცილებელ პერიფერიას, რომელიც საჭიროა მიკროკონტროლერის შესწავლისას. რომელი კონტროლერიც არ უნდა შევარჩიოთ, ყველა **Atmel** მიკროკონტროლერი გამოდგება პროგრამის გადატანის დროს უფრო სუსტიდან შედარებით მძლავრ მიკროკონტროლერზე.

მოწყობილობის ელექტრული პრინციპული სქემა მოცემულია 2.31 ნახაზზე.

სქემის აწყობისათვის საჭიროა:

- კვარცული რეზონატორი 8 მეგაჰერცი სიხშირით (ეს სიხშირე შეიძლება მერყეობდეს დიდ დიაპაზონში, მაგრამ არ უნდა აღემატებოდეს შერჩეული ტიპის მიკროკონტროლერის მაქსიმალურ დასაშვებ სიხშირეს);
- ორი ერთნაირი კერამიკული კონდენსატორი 15-დან 30 პიკოფარადამდე ტევადობით;

- მცირე სიმძლავრის შუქდიოდი;
- რეზისტორი 150 – 500 ომამდე წინაღობით;
- მიკროკონტროლერი (**ATmega88**).

ჩამოთვლილი დეტალებიდან ავაწყოთ სქემა. **XTAL1** და **XTAL2 (PB6, PB7)** მიკროკონტროლერის გამომყვანებთან მიუერთოთ ტაქტური გენერატორი, რომელიც აწყობილია **Q1** კვარცული რეზონატორის და **C1** და **C2** კონდენსატორების საფუძველზე. **GND** გამომყვანს ვაერთებთ "მიწასთან" (კვების წყაროს უარყოფითი პოლუსი). **VCC** გამომყვანს ვაერთებთ კვების წყაროს დადებით პოლუსთან.

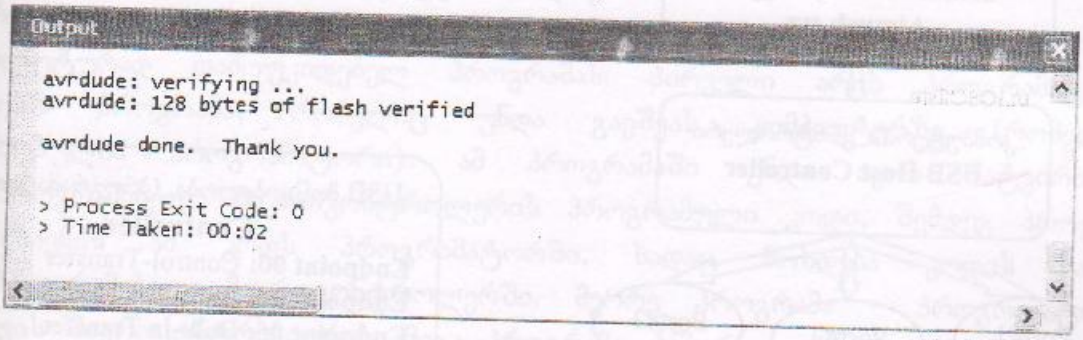
LED1 შუქდიოდის "მინუსს" მაკომპენსირებელი **R1** რეზისტორის გავლით ვაერთებთ "მიწაზე" (კვების წყაროს უარყოფითი პოლუსი), "პლუსს" ვაერთებთ მიკროსქემის **D**-პორტის ნებისმიერ გამოსასვლელზე (სქემაზე - **PD0**-თან).

ვაერთებთ **RESET (PC6), SCK (PB5), MISO (PB4), MOSI (PB3)**-ს და "მიწის" ხაზს (**GND**) აწყობილ პროგრამატორთან.

ძაბვის ჩართვამდე კიდევ ერთხელ ვამოწმებთ ყველა ელემენტის მიერთების სისწორეს. შემდეგ მივაერთებთ პროგრამატორს კომპიუტერის პარალელურ (**LPT**) პორტთან და მხოლოდ ამის შემდეგ მივაწოდებთ ძაბვას სქემაზე.

შემდეგ გავუშვებთ **Programmers Notepad** და ვხსნით შესაბამის პროგრამას. შეიძლება ის კიდევ ერთხელ დავაკომპილიროთ **Tools -> [WinAVR] Make All** ბრძანებით.

შემდეგ გავჭოლავთ დაკომპილირებულ ფაილს **my_test.hex** მიკროკონტროლერის მეხსიერებაში **Tools->[WinAVR]Make Program** ბრძანებით. პროცესის დამთავრების შემდეგ პროგრამის ფანჯარაში "**Output**" ჩნდება შეტყობინება, რომ გაჭოლვა წარმატებით შესრულდა (ნახ. 2.32).



ნახ. 2.32

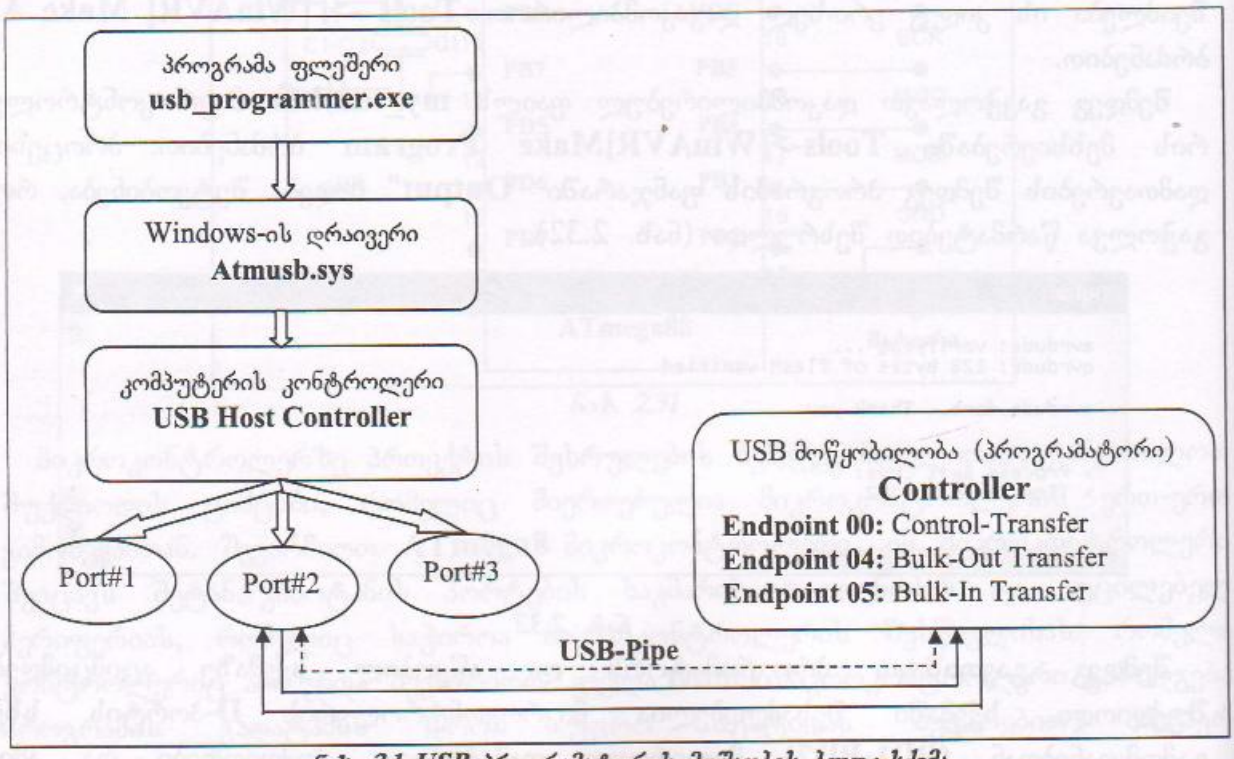
შემდეგ გავთიშავთ პროგრამატორს და აწყობილ სქემაზე აციმციმდება შუქდიოდი. სქემაში შესაძლებელია მიკროკონტროლერის **D**-პორტის სხვა გამომყვანებთან (**PD1-PD7**) მივაერთოთ დამატებითი შუქდიოდები და ამით მივიღოთ მოციმციმე გირლანდა.

3.1. USB პროგრამატორის მუშაობის ზოგადი პრინციპები

კომპიუტერთან **USB** ინტერფეისის მეშვეობით დაკავშირებული პროგრამატორის დამუშავების ამოცანა შეიძლება რამდენიმე ქვეამოცანად დაიყოს.

საჭიროა დაიწეროს პროგრამა ფლეშერი, რომელიც პროგრამატორს მიმართავს დაამყაროს მასთან კავშირი და, წინასწარ დაგეგმილი ოქმის თანახმად, გაგზავნოს მასში დასაპროგრამებელი მიკროსქემის პროგრამული კოდი. უფრო კონკრეტულად, ეს პროგრამა უნდა დაუკავშირდეს სისტემურ **USB** დრაივერს, გახსნას პორტი და **USB** კონტროლერის საშუალებით საჭირო მოწყობილობის (ამ შემთხვევაში პროგრამატორის) პაიპში დაიწყოს მონაცემების გაგზავნა და პასუხის მიღება.

პროგრამატორს უნდა შეეძლოს **USB** ინტერფეისიდან ბრძანებებისა და მონაცემების მიღება, მათი დამუშავება და დასაპროგრამებელი მიკროსქემის მართვა და დაპროგრამება. ეს ნიშნავს, რომ თვითონ მიკროკონტროლერი უნდა იყოს "ჭკვიანი" მოწყობილობა, უნდა ჰქონდეს საკუთარი მართვის მოდული (განხორციელებული მიკროკონტროლერის სახით), რომელიც საკუთარ თავზე აიღებს ამ ფუნქციებს.



ნახ. 3.1 USB პროგრამატორის მუშაობის ბლოკ-სქემა

საჭიროა მოწყობილობა, რომელიც უზრუნველყოფს კავშირს კომპიუტერის USB ინტერფეისა და პროგრამატორის მართვის მოდულს შორის ანუ პროგრამატორს უნდა ჰქონდეს საკუთარი **USB** კონტროლერი.

გარდა ამისა, საჭიროა დაიწეროს პროგრამატორის მართვის პროგრამა მიკროკონტროლერისათვის. მას უნდა შეეძლოს **USB** კონტროლერთან კავშირი და უზრუნველყოს დასაპროგრამებელ მიკროსქემასთან ინტერფეისი.

3.1 ნახაზზე ნაჩვენებია **USB** პროგრამატორის მუშაობის ზოგადი სქემა. პროგრამა ფლეშერი დრაივერის საშუალებით უკავშირდება კომპიუტერის **USB** კონტროლერს. იხსნება პორტი და იწყება მონაცემთა გაცვლა პროგრამატორის კონტროლერზე.

USB კონტროლერი და მართვის მიკროკონტროლერი **USB** პროგრამატორის ორი ძირითადი ბლოკია. **USB** კონტროლერის ამოცანაა, უზრუნველყოს კომპიუტერის პორტსა და მართვის მიკროკონტროლერს შორის კავშირი.

არსებობს მიკროკონტროლერები, რომელთაც **USB** კონტროლერი ინტეგრირებული აქვს. ასეთი ტიპის მიკროკონტროლერის გამოყენება **USB** პროგრამატორში კარგი გადაწყვეტილებაა, რადგან შესაძლებელი ხდება ერთ ინტეგრალურ მიკროსქემაში ორი ფუნქციური ბლოკის ინტეგრაცია.

USB პროგრამატორი, რომლის ძირითადი ბირთვია მართვის მიკროკონტროლერი არის ასაგებად მარტივი და ძალიან საიმედო მოწყობილობა. ეს საიმედოობა მიიღწევა პროგრამატორის კონსტრუქციაში მინიმალური შესაძლო რაოდენობის კომპონენტების გამოყენებით.

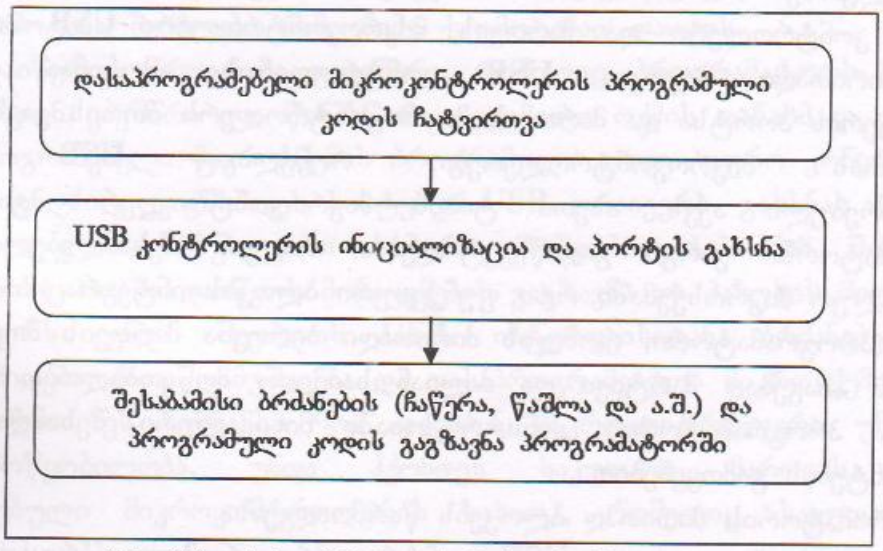
პროგრამატორის ძირითად ბლოკებს წარმოადგენს:

- მართვის მოდული და USB კონტროლერი, რომელიც რეალიზებულია ერთ მიკროსქემაში - მართვის მიკროკონტროლერში;
- USB კონექტორის ბლოკი;
- ოსცილატორის ბლოკი;
- პროგრამატორის სტატუსის ინდიკაციის ბლოკი;
- დასაპროგრამებელი მიკროკონტროლერის სოკეტის ბლოკი;
- +5v / +12v ძაბვის კონვერტერის ბლოკი.

პროგრამატორი მუშაობისათვის საჭიროებს პროგრამულ უზრუნველყოფას - ორ აბსოლუტურად დამოუკიდებელ პროგრამას. პირველი არის პროგრამატორის მართვის პროგრამა, რომელიც უნდა გაეშვას კომპიუტერზე (რომელზეც მიერთებულია პროგრამატორი). ამ პროგრამაში უნდა იყოს ჩატვირთული დასაპროგრამებელი მიკროკონტროლერის პროგრამული კოდი, შემდეგ პროგრამა გადაგზავნის ამ კოდს პროგრამატორში, სადაც მოხდება კოდის ჩაწერა დასაპროგრამებელ მიკროკონტროლერში. მეორე პროგრამა - პროგრამატორის მართვის მიკროკონტროლერის შიგა პროგრამა. სწორედ ამ პროგრამის ამოცანაა კომპიუტერთან ინტერფეისის უზრუნველყოფა პროგრამატორის მხრიდან. ამ პროგრამამ აგრეთვე უნდა უზრუნველყოს კომპიუტერიდან მიღებული პროგრამული კოდის ჩაწერა დასაპროგრამებელ მიკროკონტროლერში, ამ მიკროკონტროლერის დაპროგრამების ალგორითმის შესაბამისად.

პროგრამატორის მართვის პროგრამის ამოცანაა USB პორტის საშუალებით პროგრამატორთან დაკავშირება, მისთვის ბრძანებების მიცემა (დააპროგრამოს მიკროკონტროლერი, შეამოწმოს ჩაწერილი პროგრამის კორექტულობა, წაშალოს მიკროკონტროლერში ჩაწერილი პროგრამა), დასაპროგრამებელ მიკროკონტროლერში ჩასაწერი პროგრამული კოდის გაგზავნა პროგრამატორში.

3.2 ნახაზზე ნაჩვენებია პროგრამატორის მართვის პროგრამის ძირითადი ალგორითმის სქემა. პროგრამატორის მართვის პროგრამა დაწერილია C++ ენაზე, და შედგება სამი მოდულისგან.



ნახ. 3.2. პროგრამატორის მართვის პროგრამა

პირველი ფაილი პროგრამის ძირითადი მოდულია. მასში არის მთავარი (main) ფუნქცია, რომელიც მიმდევრობით იძახებს მიკროკონტროლერის წაშლას, წაშლის შემოწმების, ჩაწერის და ჩაწერის შემოწმების ფუნქციებს.

usb_programmer.cpp

მეორე მოდულში გაერთიანებულია ფუნქციები, რომელთა საშუალებით ხდება **USB** პორტთან დაკავშირება. ეს მოდული აერთიანებს პორტის გახსნის ინიციალიზაციის, დახურვის, მონაცემთა გაგზავნის და მიღების ფუნქციებს.

usb.cpp

მესამე მოდულში არის ფუნქციები, რომელიც პასუხისმგებელია პროგრამული კოდის წაკითხვაზე ფაილიდან და ამ კოდით მიკროკონტროლერის დაპროგრამებაზე. აგრეთვე ამ მოდულში გაერთიანებულია რამდენიმე სერვისული ფუნქცია.

commands.cpp

მართვის მიკროკონტროლერის შიგა პროგრამის ამოცანაა უზრუნველყოს მიკროკონტროლერის მიერ USB პორტის საშუალებით ჩასაწერი პროგრამული კოდის მიღება და ამ კოდის ჩაწერა მიკროკონტროლერში.

დაპროგრამების ალგორითმი შედგება შემდეგი ბიჯებისგან:

1. VCC და GND კონტაქტებზე კვების მიწოდება. RST და XTAL კონტაქტებზე GND მნიშვნელობის დაყენება;
2. RST და P3.2 კონტაქტებზე "1" მნიშვნელობის დაყენება;
3. P3.3, P3.4, P3.5, P3.7 კონტაქტებზე, დაპროგრამების სასურველი ოპერაციის ასარჩევად, შესაბამისი "1" და "0" მნიშვნელობების დაყენება;

4. დასაპროგრამებელი კოდის 000H პოზიციის ბაიტის მიწოდება 1.0-დან 1.7-მდე კონტაქტებზე;
5. RST კონტაქტზე +12ვ ძაბვის დაყენება დაპროგრამებისათვის;
6. P3.2 კონტაქტზე იმპულსების მიწოდება იმისათვის, რომ PEROM მეხსიერებაში მოხდეს პროგრამის ბაიტის ჩაწერა. ბაიტის ჩაწერას დაახლოებით 1.2მწმ სჭირდება;
7. ჩაწერილი მონაცემის შესამოწმებლად, RST კონტაქტის +12-დან ლოგიკურ "1" დონეზე დაწევა. P3.3, P3.4, P3.5, P3.7 კონტაქტებზე, შესაბამისად "1" და "0" მნიშვნელობების დაყენება. მონაცემები შეიძლება იყოს წაკითხული P1 პორტიდან;
8. XTAL1 კონტაქტზე მომდევნო ბაიტის დასაპროგრამებლად პულსის მიწოდება და ახალი მონაცემების მიწოდება P1 პორტის კონტაქტებზე;
9. 5-8 ბიჯების გამეორება პროგრამის კოდის ბოლომდე ჩასაწერად;
10. XTAL და RST კონტაქტებზე "0" მნიშვნელობის დაყენება.

3.2. USB გარდამქმნელის შექმნა FTDI მიკროსქემების გამოყენებით

კომპანია Future Technology Devices International (**FTDI**) ამუშავებს და უშვებს ინტერფეისების (**USB-UART, USB-FIFO** და სხვა) გარდამქმნელების მიკროსქემებს, პროგრამულ უზრუნველყოფას (დრაივერები, უტილიტები) და გაწყობის საშუალებებს ინტერფეისების გარდამქმნელების დასამუშავებლად.

FT232BL წარმოადგენს წამყვან თავისუფალ ვერსიას, **FTDI**-ს პოპულარული **USB UART** ინტეგრალური მიკროსქემების მეორე გენერაციას. ეს მიკროსქემა ქმნის არა მარტო დამატებით ფუნქციურ შესაძლებლობებს თავის წინამორბედთან (**FT8U232AM**) შედარებით, არამედ ამცირებს გარე კომპონენტების რაოდენობას, ინარჩუნებს გამომყვანების ორიგინალთან თავსებადობის მაღალ ხარისხს, რითაც ამოლებს და აიაფებს არსებული მოწყობილობების მოდერნიზაციას და ამაღლებს მათი გამოყენების პოტენციალს.

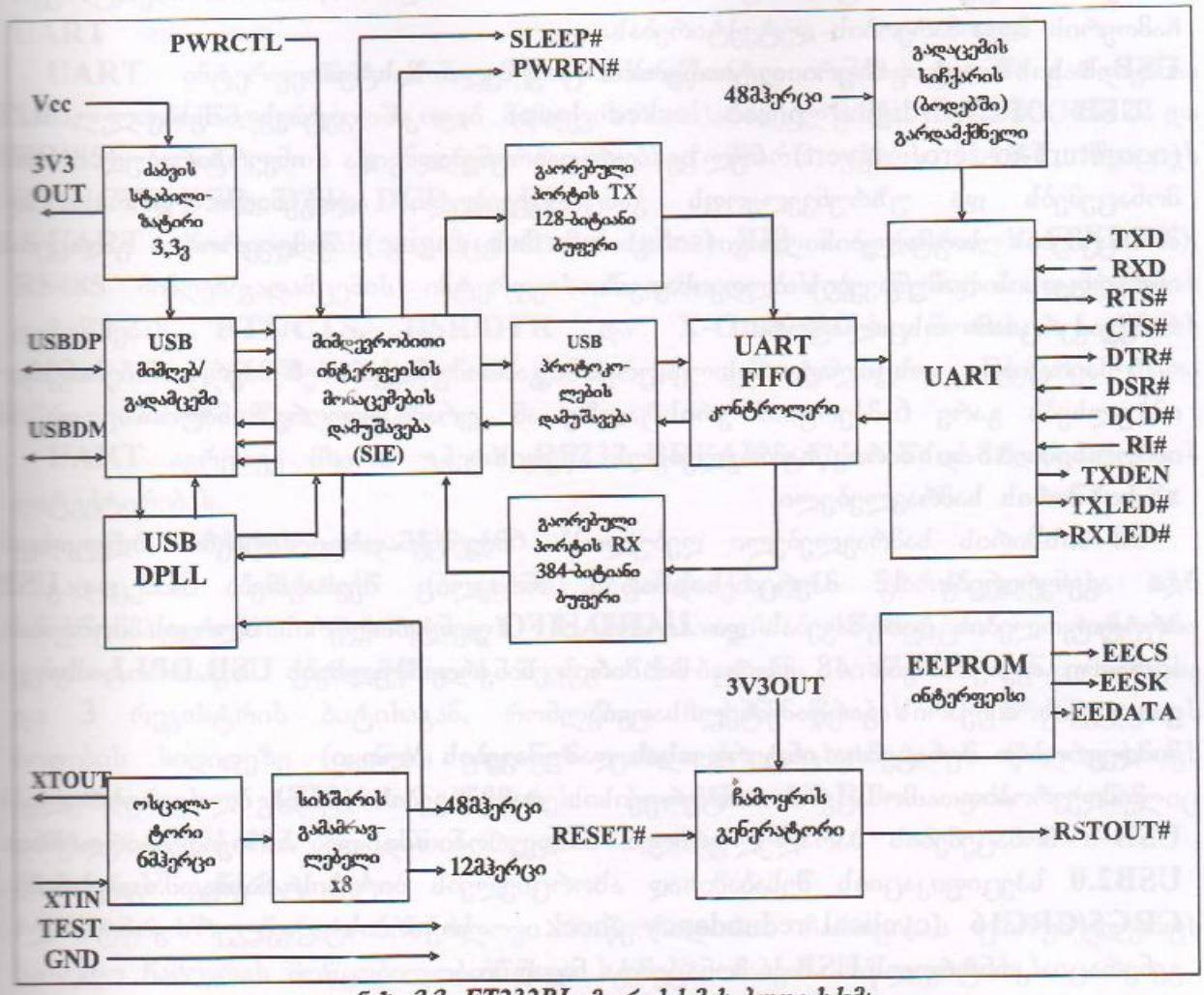
FT232BL ინტერფეისების გარდამქმნელის ძირითადი მახასიათებლებია:

- მარტივი და იაფი **USB** ინტერფეისის გარდამქმნელი მიმდევრობით **RS-** ინტერფეისში;
- გადაცემის ნაკადის მართვისა და მოდემის ინტერფეისის სიგნალების სრული მხარდაჭერა;
- **UART** ინტერფეისის 7/8-ბიტიანი მონაცემების, აგრეთვე 1/2 სტოპ-ბიტის, კენტობის, ლუწობის, მარკირების, ინტერვალების, უტოლობის მხარდაჭერა;
- მონაცემთა გადაცემის სიდიდე 300...3 მბოდი (TTL);
- მონაცემთა გადაცემის სიდიდე 300...1 მბოდი (RS232);
- მონაცემთა გადაცემის სიდიდე 300...3 მბოდი (RS422/RS485);
- 384ბიტიანი მიმღები ბუფერი, 128-ბიტიანი გადამცემი ბუფერი მონაცემების გამტარობის ამაღლებისათვის;
- **RX** ბუფერის მართვადი წყვეტა (time out);
- სრულად მართვადი **HARDWARE** ანუ ნაკადების **X-On/X-Off** სრული მხარდაჭერა;

- სპეციალური სიმბოლოებისა და სტრიქონის წყვეტის მდგომარეობის ჩაშენებული მხარდაჭერა;
- გადაცემის ბუფერის ავტომატური კონტროლი RS485-სათვის;
- **USB** ინტერფეისის „დაძინება“/“გაღვიძების“ რეჟიმების მხარდაჭერა **SLEEP#** და **RI#** გამომყვანების გამოყენებით;
- **USB**-ს მაღალი ძაბვის სალტისგან კვების მხარდაჭერა **PWREN#** გამომყვანის გამოყენებით;
- მართვის სიგნალების და **UART**-ის დონეების ჩაშენებული გარდამქმნელი 5 და 3,3ვ ლოგიკასთან შეთავსებისათვის;
- ინტეგრირებული 3,3ვ სტაბილიზატორი **USB**-ს შესასვლელ/გამოსასვლელებისათვის;
- ჩართვისას ჩამოყრის ინტეგრირებული წრედი;
- ინტეგრირებული 6მჰერცი – 48მჰერცი სიხშირის სამრავლებელი (**PLL**);
- **USB**-ს ფუძე ანუ იზოქრონული გადაცემის რეჟიმი;
- მუშაობის უნიფიცირებული მხარდაჭერა 4,35-დან - 5,25 ვოლტამდე;
- თავსებადობა ჰოსტ-კონტროლერებთან **UHCI/OHCI/EHCI**;
- თავსებადობა USB 1.1-სა და USB 2.0-თან;
- **USB**-ს იდენტიფიკატორების VID, PID სერიული ნომრის და პროდუქტის კოდის მოთავსება გარე **EEPROM**-ში;
- **EPROM**-ის დაპროგრამება **USB**-ს გავლით;
- ვირტუალური **COM** პორტის (**VCP**) დრაივერები Windows-ის ყველა ოპერაციული სისტემისათვის.

გამოყენების სფეროები:

- USB ↔ RS232 გარდამქმნელები;
- USB ↔ RS422/RS485 გარდამქმნელები;
- მოძველებული RS232 პერიფერიის გაუმჯობესება **USB**-ს სტანდარტებამდე;
- ფიჭური და უკაბელო ტელეფონების **USB** მონაცემების გადაცემა კაბელებსა და ინტერფეისებზე;
- მიკროკონტროლერული მოწყობილობების დაპროექტება **USB**-ს სტანდარტების გამოყენებით;
- აუდიო და მცირე გატარების ზოლის ვიდეომონაცემების გადაცემა **USB**-ს გამოყენებით;
- **PDA↔USB** მონაცემთა გადაცემა;
- ელექტრონული ბარათების **USB** წამკითხველი მოწყობილობა;
- შტრიხ-კოდების **USB** წამკითხველი მოწყობილობა;
- **USB** ინტერფეისით გამზომი და დიაგნოსტიკური მოწყობილობები;
- **USB** მოდემები და უკაბელო მოდემები;
- Set Top Box (S.T.B.) – სატელევიზიო აპარატურის მოწყობილობების USB-ინტერფეისები.



ნახ. 3.3. FT232BL მიკროსქემის ბლოკ-სქემა

3.3 ნახაზზე ნაჩვენებია **FT232BL** მიკროსქემის ბლოკ-სქემა. ქვემოთ მოცემულია სქემაში ნაჩვენები ფუნქციური ბლოკების აღწერა.

3,3-ვოლტიანი სტაბილიზატორი მცირე ძაბვის ვარდნით

3,3-ვოლტიანი (Low DropOut - მცირე ძაბვის ვარდნით) სტაბილიზატორი გამოიმუშავებს 3,3ვ შესაბამის ძაბვას USB-ის მიმღებ/გადამცემის გამოსასვლელი ბუფერის უჯრედის მართვისათვის, აგრეთვე მიაწოდებს 3,3ვ კვებას **RSTOUT#** გამოსასვლელზე.

ამ ბლოკის ძირითადი ფუნქციაა უპირატესად კვებოს **USB** მიმღებ/გადამცემი და "ჩამოყრის" გენერაციის უჯრედი, ვიდრე გარე ლოგიკა. მაგრამ გარე წრედები მოითხოვს 3,3 ვოლტის ნომინალს არა უმეტეს **5მა** დენით, რაც ასევე შეიძლება განხორციელდეს **3V3OUT** გამომყვანიდან აუცილებლობის შემთხვევაში.

USB მიმღებ/გადამცემი

USB მიმღებ/გადამცემი უზრუნველყოფს **USB1.1/USB2.0** სრული სიჩქარის ფიზიკურ ინტერფეის **USB** კაბელში, შესაბამისი დრაივერები კი - სიგნალების გადაცემის მაქსიმალურ სიჩქარეზე 3,3ვ დონის კონტროლს, სანამ სხვა მიმღები

და ორი განცალკევებული ბოლო მიმღები ახდენს **USB**-ს შესავალი მონაცემების, საძიებო სისტემის ოპტიმიზაციის (**SEO** - search engine optimization) და **USB**-ს ჩამოყრის მდგომარეობის დეტექტირებას.

USB-ს სიხშირის ფაზური ავტოაწყობის ციფრული სისტემა

USB DPLL (digital phase locked loop) ბლოკი კეტავს შესავალ **RZI** (nonreturn-to-zero, invert) ნულზე არადაბრუნებად და ინვერსირებულ **USB** მონაცემებს და უზრუნველყოფს განცალკევებული აღდგენილი დროისა და მონაცემების სიგნალების **SIE** (serial interface engine) მიმდევრობით გადაცემას ინტერფეისის დამუშავების ბლოკისათვის.

6-მეგაჰერციანი ოსცილატორი

6-მჰერციანი ოსცილატორის ბლოკი გამოიმუშავებს 6-მჰერციან სინქრო-იმპულსებს გარე 6-მჰერციან კრისტალზე ან კერამიკულ რეზონატორზე და ამ იმპულსებს **x8** სიხშირის სამრავლებელს აწვდის.

x8 სიხშირის სამრავლებელი

x8 სიხშირის სამრავლებელი ღებულობს 6მჰერცს ოსცილატორის ბლოკისგან და აგენერირებს 12 მჰერც სიხშირეს, რომელიც შეესაბამება **SIE** და **USB** პროტოკოლების დამუშავებას და **UART FIFO** კონტროლერის ბლოკის მუშაობას. აგრეთვე აგენერირებს 48 მჰერც სიხშირის სინქროიმპულსებს **USB DPLL**-ისა და გადაცემის სიჩქარის გარდამქმნელისათვის.

მიმდევრობით მონაცემთა ინტერფეისის დამუშავების ბლოკი

მიმდევრობით მონაცემთა ინტერფეისის დამუშავების (**SIE**) ბლოკი ასრულებს **USB**-ს მონაცემების პარალელურიდან მიმდევრობითში და პირიქით გარდაქმნას. **USB2.0** სპეციფიკაციის შესაბამისად ახორციელებს ბიტების ჩასმა/ამოგდებას და **CRC5/CRC16** (cyclical redundancy check – სიჭარბის ციკლურ კონტროლს) გენერაცია/კონტროლს **USB**-ს მონაცემთა ნაკადში.

USB-ს პროტოკოლების დამუშავება

USB-ს პროტოკოლების დამუშავებისას ხდება მონაცემთა ნაკადის მართვა ბოლო **USB** მოწყობილობიდან. ის ამუშავებს დაბალი დონის **USB** პროტოკოლის მოთხოვნებს, რომელიც გენერირებულია **USB** ჰოსტ კონტროლერით და ბრძანებებს **UART**-ის ფუნქციური პარამეტრების მართვისათვის.

გაორებული პორტ TX-ის ბუფერი

მონაცემები ბოლო წერტილის მოწყობილობის **USB**-ს გამოსასვლელიდან მიეწოდება გაორებული პორტ **TX**-ის ბუფერში და გადაიცემა კონტროლის ქვეშ **UART** ბუფერიდან **UART FIFO** გადაცემის რეგისტრში.

გაორებული პორტ RX-ის ბუფერი

მონაცემები **UART** მიმღები რეგისტრიდან მიეწოდება **RX** გაორებული პორტის ბუფერში, რომელიც **USB**-ს მოთხოვნით **SIE**-მა გაათავისუფლა მონაცემებისათვის ბოლო წერტილის მოწყობილობიდან.

UART FIFO კონტროლერი

UART FIFO (First In, First Out – "პირველი მოვიდა – პირველი დამუშავდა") კონტროლერი ხელმძღვანელობს მონაცემთა გადაცემის რიგითობას გაორებული

RX პორტ და **TX** ბუფერს შორის და **UART**-ის მიღებისა და გადაცემის რეგისტრებს შორის.

UART

UART ინტერფეისის მონაცემების ასინქრონული 7/8-ბიტიანი გარდაქმნაა პარალელურ-მიმდევრობითში და მიმდევრობით-პარალელურში, **RS232** (**RS422** და **RS485**). კონტროლის სიგნალები, რომლებსაც მხარს უჭერს **UART**, შეიცავს **RTS, CTS, DSR , DTR, DCD** და **RI**-ს.

UART ახორციელებს გადამცემის ჩართვის სიგნალის კონტროლს **(TXDEN), RS485** მიმღებ/გადამცემის ინტერფეისთან ერთად, ასევე კავშირის დამყარების დამოწმებას **RTS/CTS, DSR/DTR** და **X-On/X-Off**-ს შორის. კავშირის დამყარების დამოწმება საჭიროა განხორციელდეს უშუალოდ სქემაში, რათა სწრაფი პასუხი მივიღოთ.

UART აგრეთვე მხარს უჭერს **RS232 BREAK** გაწყობის და მდგომარეობის დიტექტირებას.

გადაცემის სიჩქარის გარდამქმნელი

გადაცემის სიჩქარის (ბოდებში) გარდამქმნელი უზრუნველყოფს **x16** სინქროიმპულსების შესასვლელს **UART**-ისათვის 48-მჰერციანი ტაქტური გენერატორისაგან. გარდამქმნელი შედგება 14-ბიტიანი სიხშირის წინაგამყოფისა და 3 რეგისტრის ბიტისაგან, რომლებიც ზუსტ აწყობას უზრუნველყოფს ბოდების სიდიდეზე (იყენებს გაყოფას უახლოეს რიცხვს პლუს წილადი). ეს განსაზღვრავს გადაცემის სიჩქარეს ბოდებში **UART**-ისათვის, რომელიც დაპროგრამდება 183 ბოდიდან 3 მილიონ ბოდამდე.

"ჩამოყრის" გენერატორი

"ჩამოყრის" გენერატორის ბლოკი მოწყობილობის ჩართვისას უზრუნველყოფს საიმედო ჩამოყრას მოწყობილობის შიგა სქემებზე ძაბვის მიწოდებისას.

დამატებითი **RESET#** შესასვლელისა და **RSTOUT#** გამოსასვლელის მეშვეობით გაიცემა ნებართვა სხვა მოწყობილობებისათვის ჩამოყაროს **FT232BL** ან, პირიქით, **FT232BL** – ჩამოყაროს სხვა მოწყობილობები. ჩამოყრის განმავლობაში **RSTOUT#** იმართება დაბალი დონით, რომელიც შემცირდება გამოსასვლელზე 3,3ვ ძაბვამდე და გამომუშავდება სქემაში არსებული სტაბილიზატორით. **RSTOUT#** გამოსასვლელზე შეიძლება იყოს გამოყენებული 1,5კომი ამწევი რეზისტორი კონტროლისათვის უშუალოდ **USBDP**-ში, სადაც **USB**-ს დაყოვნება აუცილებელია. ის ასევე გამოიყენება სხვა მოწყობილობების ჩამოყრისათვის. **RSTOUT#** გაჩერდება მაღალი იმპედანსის მდგომარეობაში დაახლოებით 5 მილი/წამი, სანამ VCC აიწევა 3,5 ვოლტამდე და მოწყობილობის ოსცილატორი გაიშვება, ამავე დროს **RESET#**-ზე მიიღება მაღალი დონე. **RESET#** დაფიქსირდება **VCC**-მდე, სანამ იქნება მოთხოვნა მოწყობილობის ჩამოყრაზე გარე ლოგიკიდან ან გარე ჩამოყრის გენერატორისგან.

EEPROM ინტერფეისი

FT232B შეიძლება იყოს ჩაშენებული **EEPROM**-ის გარეშე, მაგრამ გარე **93C46 (93C56** ან **93C66) EEPROM** გამოიყენება **USB –VID** და **-PID**-ში სერიული ნომრის, პროდუქტის კოდის, კვების პარამეტრების მონაცემებისა და მათი

მოდიფიკაციების შენახვისათვის, აგრეთვე გამოიყენება **OEM** (original equipment) ორიგინალურ მოწყობილობებში.

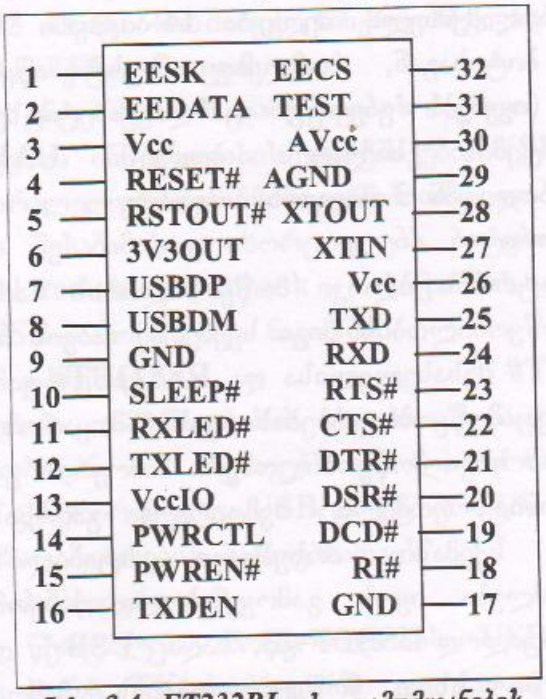
სხვა პარამეტრები კონტროლირებადი **EEPROM**-ის საშუალებით შეიცავს გარედან გაშვებას, იზოქრონული გადაცემის რეჟიმს, პროგრამის გაჩერებას, ძაბვის გამორთვას და **USB 2.0** იდენტიფიკატორის რეჟიმს.

EEPROM არის 16-ბიტიანი ფართო კონფიგურაციის, როგორც **MicroChip 93LC46B** ან ეკვივალენტური, რომელსაც აქვს 1მბიტ/წმ სიხშირე **Vcc** ძაბვის **4.35** ვოლტიდან **5.25** ვოლტამდე ცვლილების დროს. **EEPROM**-ის დაპროგრამება ხდება **FTDI** უტილიტების გამოყენებით **USB**-ს გავლით უშუალოდ სქემაში. ამას ხელს უწყობს ცარიელი არე ნაბეჭდ პლატაზე (**PCB**-printed circuit board) და დაპროგრამდება, როგორც წარმოების და გამართვის პროცესების ნაწილი.

თუ **EEPROM** არ არის მიერთებული (ან **EEPROM** ცარიელია), **FT232BL** იყენებს ჩაშენებულ (წინასწარ მოცემულ) **VID, PID,** პროდუქტის კოდის და კვების პარამეტრების მონაცემებს. ამ შემთხვევაში მოწყობილობას არა აქვს **USB**-ს იდენტიფიკატორის ნაწილის სერიული ნომერი.

3.2.1. ინტერფეისის გარდამქმნელის გამომყვანების დანიშნულება

3.4 ნახაზზე ნაჩვენებია **FT232BL**-ის გამომყვანების ნუმერაცია და დასახელება.



ნახ. 3.4. FT232BL-ის გამომყვანების აღნიშვნა

UART ინტერფეისის გამომყვანების ჯგუფი

გამომყვანი	სიგნალი	ტიპი	დანიშნულება
25	TXD	UT	ასინქრონული მონაცემების გადაცემა გამოსასვლელზე
24	RXD	IN	ასინქრონული მონაცემების მიღება შესასვლელზე
23	RTS#	OUT	მოთხოვნა (გადაცემის მართვის)/(კავშირის დადასტურების) სიგნალის გაგზავნაზე
22	CTS#	IN	(გადაცემის მართვის)/(კავშირის დადასტურების) სიგნალის

			მიღების დადასტურება
21	**DTR#**	**UT**	(გადაცემის მართვის)/(კავშირის დადასტურების) სიგნალის მზადყოფნა მონაცემთა ტერმინალზე
20	**DSR#**	**IN**	(გადაცემის მართვის)/(კავშირის დადასტურების) სიგნალის მზადყოფნა მოდემზე
19	**DCD#**	**IN**	მონაცემთა მატარებლის დააფიქსირება გადაცემის მართვით.
18	**RI#**	**IN**	გადაცემის მართვის გამოძახების მაჩვენებელი, როდესაც დისტანციური ტაიმერული ჩართვის ფუნქცია ჩართულია **EEPROM**-ში, გადასვლა **RI#** ქვედა დონეზე გამოიყენება **PC USB Host** კონტროლერის მუშაობის გაგრძელებისათვის "დაკიდული" მდგომარეობის შემდეგ
16	**TXDE**	**OUT**	მონაცემების გადაცემის ჩართვა **RS485**-ში

USB-ს ინტერფეისების ჯგუფი

გამომყვანი	სიგნალი	ტიპი	დანიშნულება
7	**USBD**	**I/O**	**USB**-ს დადებით მონაცემთა სიგნალი
8	**USBD**	**I/O**	**USB**-ს უარყოფით მონაცემთა სიგნალი

EEPROM-ის ინტერფეისების ჯგუფი

გამომყვანი	სიგნალი	ტიპი	დანიშნულება
32	**EECS**	**I/O**	**EPROM**-ჩიპის შერჩევა. 48-მჰერციანი ოპერაციებისათვის **ECS** დაიწევა **GND**-მდე 10**კომი** რეზისტორის გამოყენებით. 6-მჰერციანი ოპერაციებისათვის რეზისტორი არ არის საჭირო. ადგილი აქვს სამ მდგომარეობას მოწყობილობის ჩამოყრის განმავლობაში
1	**EESK**	**T**	სინქროსიგნალი **EEPROM**-ისათვის. სამი მდგომარეობა მოწყობილობის ჩამოყრის პერიოდში გამოსასვლელის მართვის გარდა
2	**EEDATA**	**I/O**	**EEPROM**-ის მონაცემების **I/O** უშუალოდ უერთდება **EEPROM**-ის მონაცემთა შეტანას და **EEPROM**-ის მონაცემთა გამოტანას 2,2**კომი** რეზისტორის გავლით. ამის გარდა, ხორციელდება **EEPROM**-ის მონაცემთა გამოტანის აწევა **Vcc**-მდე 10**კომი** რეზისტორის გავლით, ოპერაციის კორექტირების მიზნით. ადგილი აქვს სამ მდგომარეობას მოწყობილობის ჩამოყრის პერიოდში

კვების კონტროლის ჯგუფი

გამომყვანი	სიგნალი	ტიპი	დანიშნულება
10	**SLEEP#**	**T**	მიმდინარეობს დაბალ დონეზე **USB**-ს "დაკიდების" რეჟიმში. ჩვეულებრივ, გამოიყენება ძაბვის დაგდებისათვის გარე **TTL/RS232** დონის გარდამქმნელში ანუ **USB↔RS232** გარდამქმნელის შექმნისას
15	**PWREN#**	**UT**	მიმდინარეობს დაბალ დონეზე მას შემდეგ, რაც მოწყობილობა კონფიგურირებულია **USB**-ს გავლით, შემდეგ მაღალ დონეზე **USB**-ს "დაკიდებისას". გამოიყენება გარე ლოგიკის კვების კონტროლისათვის. რთავს ინტერფეისის დაწევის რეჟიმს **EEPROM**-ში, როდესაც გამოიყენება **PWREN#** გამომყვანი
14	**PWRCTL**	**I**	კაბელით კვება-მიერთება დაბალი დონით ან კვება-მიერთება მაღალი დონით (**VccIO**-სთან)

შერეული სიგნალების ჯგუფი

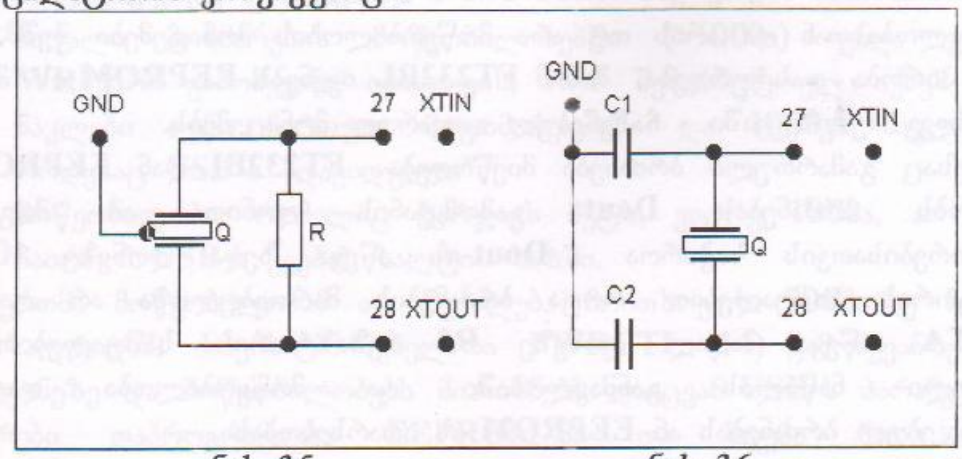
გამომყვანი	სიგნალი	ტიპი	დანიშნულება
4	RESET#	IN	გამოიყენება გარე მოწყობილობით FT232-ის ჩამოსაყრელად. თუ ამის აუცილებლობა არ არის Vcc უერთდება.
5	RSTOUT#	T	შიგა "ჩამოყრის" გენერატორის გამოსასვლელი. ინარჩუნებს (~5მწამს) მაღალ იმპედანს მას შემდეგ, რაც Vcc>3,5ვ და შიგა საათი იწყებს მუშაობას, შემდეგ აფიქსირებს თავის გამოსასვლელს შიგა სტაბილიზატორის 3,3ვ გამოსასვლელზე. RESET#-ს მიღება დაბალ დონეზე გამოიწვევს RSTOUT#-ის მოქმედებას დაბალ დონეზე. RSTOUT# არ მოქმედებს, როგორც USB-ს ჩამოყრის ხაზი
12	TXLED#	O.C.	შუქდიოდის მართვის სიგნალები, წარმოადგენს დონეების პულსირებას USB-ში მონაცემების გადაცემის დროს
11	RXLED#	O.C.	შუქდიოდის მართვის სიგნალები, წარმოადგენს დონეების პულსირებაში USB-ში მონაცემების მიღების დროს.
27	XTIN	IN	შესასვლელი 6-მჰერციანი კრისტალური ოსცილატორის კვანძისათვის. ეს შესავალი აგრეთვე შეიძლება გამოყენებული იყოს გარე 6-მჰერციანი საათის სახით
28	XTOUT	OUT	გამოსასვლელი 6-მჰერციანი კრისტალური ოსცილატორის კვანძისათვის. XTOUT აჩერებს რხევებს, სანამ USB "დაკიდულია", თუმცა მოქმედებს, თუ იყენებს სინქრო-სიგნალებს გარე ლოგიკიდან
31	TEST	IN	ამზადებს მოწყობილობას ინტეგრალური სქემის ტესტირების რეჟიმში. სხვა შემთხვევაში უერთდება მიწას, მოწყობილობის ნორმალური ოპერირებისათვის

კვების და დამიწების ჯგუფი

გამომყვანი	სიგნალი	ტიპი	დანიშნულება
6	3V3OUT	T	3,3ვ გამოსასვლელი დაბალი ძაბვის ვარდნით ინტეგრალური სტაბილიზატორისაგან. ეს გამომყვანი გამორთული უნდა იყოს მიწისგან კერამიკული კონდენსატორით 33-პიკოფარადის ტევადობით, რომელიც მაგრდება მოწყობილობის გამომყვანთან ახლოს. მისი ძირითადი დანიშნულებაა მიაწოდოს შიგა 3,3ვ კვება USB-ს მიმღებ/გადამცემ ბლოკს და RSTOUT# გამომყვანს. დენის მცირე სიდიდე (5მა) შეიძლება მიეწოდოს ამ გამოსასვლელიდან 3,3ვ გარე ლოგიკის კვებისათვის (საჭიროების შემთხვევაში)
3,26	Vcc	WR	+4,35-დან +5,25 ვოლტამდე Vcc ძირითადი მოწყობილობებისათვის მცირე ძაბვის ვარდნით და არა UART-ის ინტერფეისის გამომყვანებზე
13	VccIO	PWR	+3,0-დან +5,25 ვოლტამდე Vcc UART-ის ინტერფეისის გამომყვანებისათვის: 10...12, 14...16 და 18...25
9,17		WR	მოწყობილობის დამიწების გამომყვანები
30	Vcc	WR	მოწყობილობის ანალოგური კვება შიგა x8 ოსცილატორის მამრავლებლისათვის
29		WR	მოწყობილობის ანალოგური მიწის წყარო შიგა x8 ოსცილატორის მამრავლებლისათვის

3.2.2. ინტერფეისის გარდამქმნელის კონფიგურაციის მაგალითები

გარე ოსცილატორის კონფიგურაცია

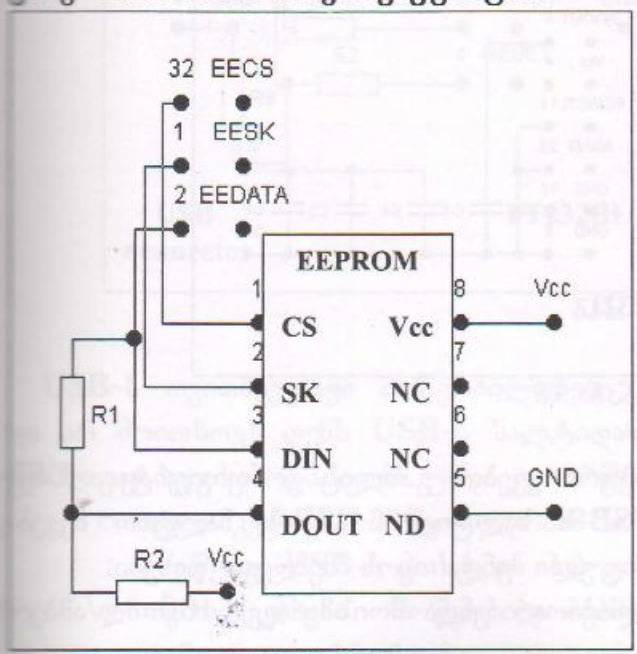


ნახ. 3.5 *ნახ. 3.6*

3.5 ნახაზზე მოცემულია სამგამომყვანიანი კერამიკული რეზონატორის შეერთება **FT232BL**-თან. სამგამომყვანიან კერამიკულ რეზონატორს ჩაშენებული კონდენსატორი აქვს და, ამის გამო, გარე კონდენსატორები აღარ არის საჭირო. **R** ერთ მეგაომიანი დატვირთვის რეზისტორი რეკომენებულია **XTIN** და **XTOUT** შორის სქემის მუშაობის სიზუსტის დონის შენარჩუნებისათვის.

3.6 ნახაზზე კი მოცემულია როგორ გამოვიყენოთ **FT232BL** 6-მჰერციან კრისტალთან ან ორგამომყვანიან კერამიკულ რეზონატორთან ერთად. ასეთ კრისტალს არა აქვს ჩაშენებული კონდენსატორი და, ამის გამო, ვიყენებთ **C1** და **C2** კონდენსატორებს, რომლებიც ჩართულია **XTIN, XTOUT**-სა და **GND**-ს შორის. მათი ტევადობა (27 პიკოფარადი) კარგად თავსდება ბევრ კრისტალსა და რეზონატორთან.

გარე **EEPROM**-ის კონფიგურაცია



ნახ. 3.7

3.7 ნახაზზე მოცემულია როგორ უერთდება **FT232BL - 93C46 (93C56, 93C66) EEPROM**-ს. **FT232BL**-ის **EECS** გამოსასვლელი უშუალოდ უერ-თდება **EEPROM**-ის ჩიპის შერჩევის გამომყვანს (**CS**-chip select). ეს არის პოტენციური მდგომარეობა, რომელსაც ერთდროულად შეუძლია მართოს მონაცემთა გამოტანა (**DOUT**) **EEPROM**-იდან და **EEDATA** გამომყვანი **FT232BL**-ზე. ამ სიტუაციაში მონაცემთა შეჯახების აცილებისათვის, **EEPROM**-ის **DOUT** უშუალოდ შეერთებულია **FT232BL**-ის **EEDATA**-სთან 2,2კომი **R1** რეზისტორით.

მოწყობილობის ჩართვასთან დაკავშირებული ჩამოყრის ან **USB**-ს ჩამოყრის დროს, **FT232BL** ასკანერებს **EEPROM**-ს და ადგენს შეერთებულია თუ არა ის მოწყობილობასთან და არის თუ არა მოწყობილობის მონაცემები ჭეშმარიტი. თუ ორივე პირობა დასტურდება, მაშინ **FT232BL** იყენებს **EEPROM**-ის მონაცემებს, წინააღმდეგ შემთხვევაში - ჩაშენებულ დაფარულ მონაცემებს.

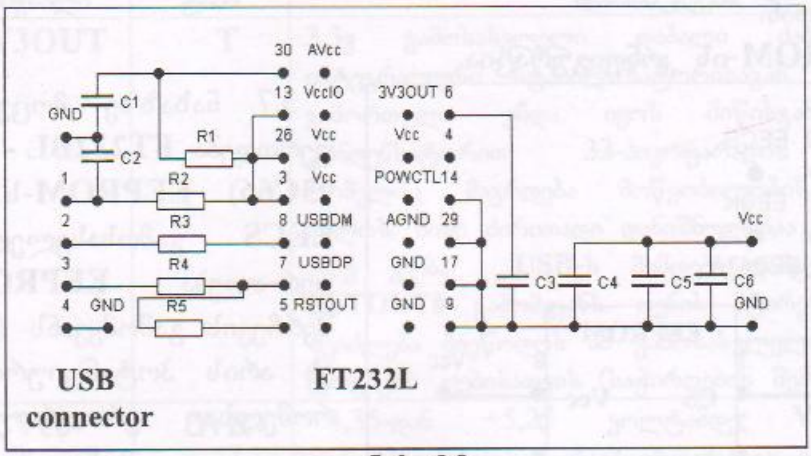
როდესაც გამართული ბრძანება მიეწოდება **FT232BL**-დან **EEPROM**-ში, ის ამოიცნობს ბრძანებას **Dout** გამომყვანის დონით. ამ მდგომარეობის დაფიქსირებისათვის საჭიროა **Dout**-ის აწევა ზედა დონეზე **10კომი R2** რეზისტორის საშუალებით. როცა ბრძანების მართებულობა არ დასტურდება, **EEDATA** აიწევა მაღალ დონეზე **R2** რეზისტორის საშუალებით ციკლის გარკვეული ნაწილის განმავლობაში და მოწყობილობა დააფიქსირებს დამახინჯებულ ბრძანებას ან **EEPROM**-ის არარსებობას.

FT232BL მუშაობს 16-ბიტიანი ფართო კონფიგურაციის **EEPROM**-თან. **EEPROM**-ს უნდა ჰქონდეს უნარი წაიკითხოს სინქროსიგნალის ერთი მეგაბიტი მონაცემი ერთ პერიოდში, როცა კვების წყაროს ძაბვის მნიშვნელობა 4,35-დან 5,25 ვოლტამდეა.

უნდა აღინიშნოს, რომ სხვადასხვა ფირმა-მწარმოებელი **EEPROM**-ში სხვადასხვანაირად იყენებს მე-6 და მე-7 გამომყვანებს. ზოგ შემთხვევაში, ისინი დატოვებულია "ჰაერში", ზოგ შემთხვევაში იყენებს 8- და 16-ბიტიანი რეჟიმების შერჩევისათვის ან ტესტირებისათვის.

შესაძლებელია **EEPROM** ნაწილობრივ გამოვიყენოთ **FT232BL**-ისათვის და დანარჩენი ნაწილი, სხვა გარე მოწყობილობისათვის, მაგალითად, **MCU**-სათვის. ამ შემთხვევაში **FT232BL** უნდა იყოს ჩამოყრის მდგომარეობაში.

USB-ს კვების გაშლის კონფიგურაცია



ნახ. 3.8

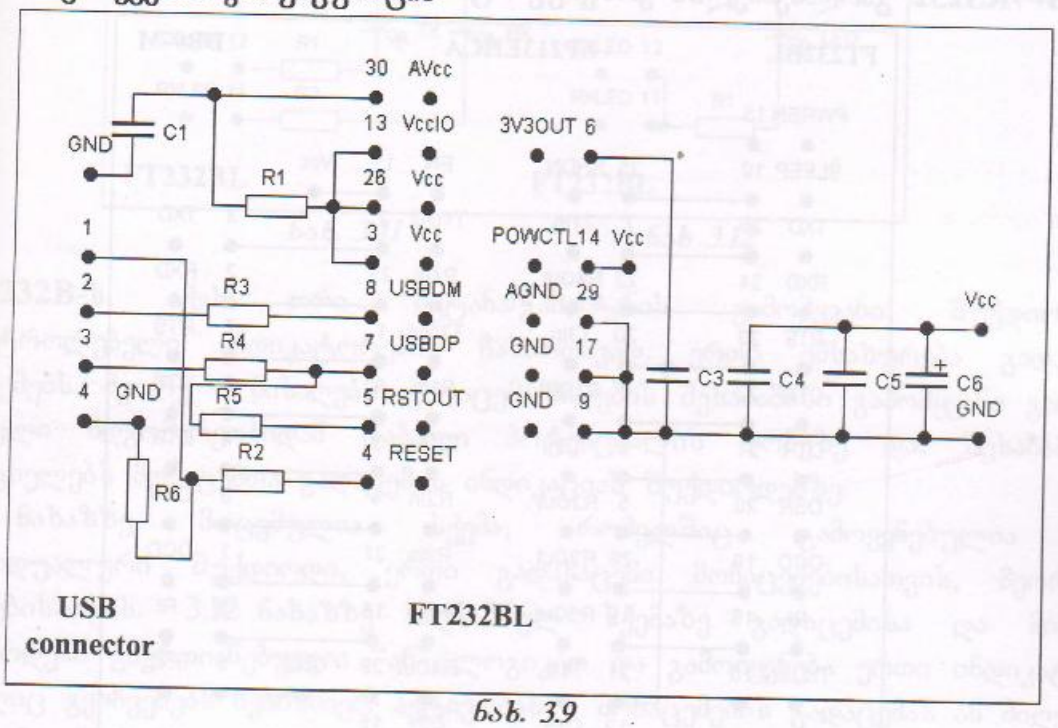
3.8 ნახაზზე მოცემულია **USB**-ს ტიპური კვების გაშლის კონფიგურაცია. **USB**-ს მოწყობილობები ღებულობს კვებას **USB**-ს სალტიდან. **USB**-ს სალტით მკვებავი მოწყობილობების ექსპლუატაციისას დაცული უნდა იყოს შემდეგი წესები:

- ქსელში ჩართვისას მოწყობილობა არ უნდა მოიხმარდეს 100 მლ/ამპერზე მეტს დენს;

- **USB**-ს "დაკიდებისას" მოწყობილობა არ უნდა მოიხმარდეს 100მკ/ამპერზე მეტ დენს;
- მაღალი მოხმარების მოწყობილობებში (>100მა) უნდა გამოიყენონ **WRPEN#** გამომყვანი იმისათვის, რომ შენარჩუნებულ იქნეს **100მა**-ზე ნაკლები დენი ჩართულ მდგომარეობაში და **100მკა** ნაკლები დენი, როდესაც **USB**-ს "დაკიდულია".
- მოწყობილობა, რომელიც ხარჯავს მეტს, ვიდრე **100მა**, არ შეიძლება ჩაირთოს **USB**-ს კვების სალტის ხაზში.

არც ერთმა მოწყობილობამ არ შეიძლება მოითხოვოს **500მა**-ზე მეტი დენი **USB**-ს სალტიდან. ძაბვის მნიშვნელობა **PWRCTL (14)** გამომყვანზე დაიწევს დაბალ დონეზე და მოწყობილობებს მიანიშნებს დაიცვან **USB**-ს პარამეტრები. ეს პარამეტრები დაპროგრამდება **EEPROM**-ში და ხდება მათი შედარება მოწყობილობების ფაქტიურ პარამეტრებთან. ფერიტის საყელური (**R2**) ჩაირთვება **USB**-ს კვებასთან მიმდევრობით, მოწყობილობიდან წინმსწრები ხმოვანი სიგნალის შესაქმნელად, რომელიც ასოცირდება ელექტრომაგნიტურ დამახინჯებებთან (**EMI** electromagnetic interference) და წარმოიშობა **USB**-ს კაბელიდან ჰოსტამდე.

USB-ს თვითკვების კონფიგურაცია



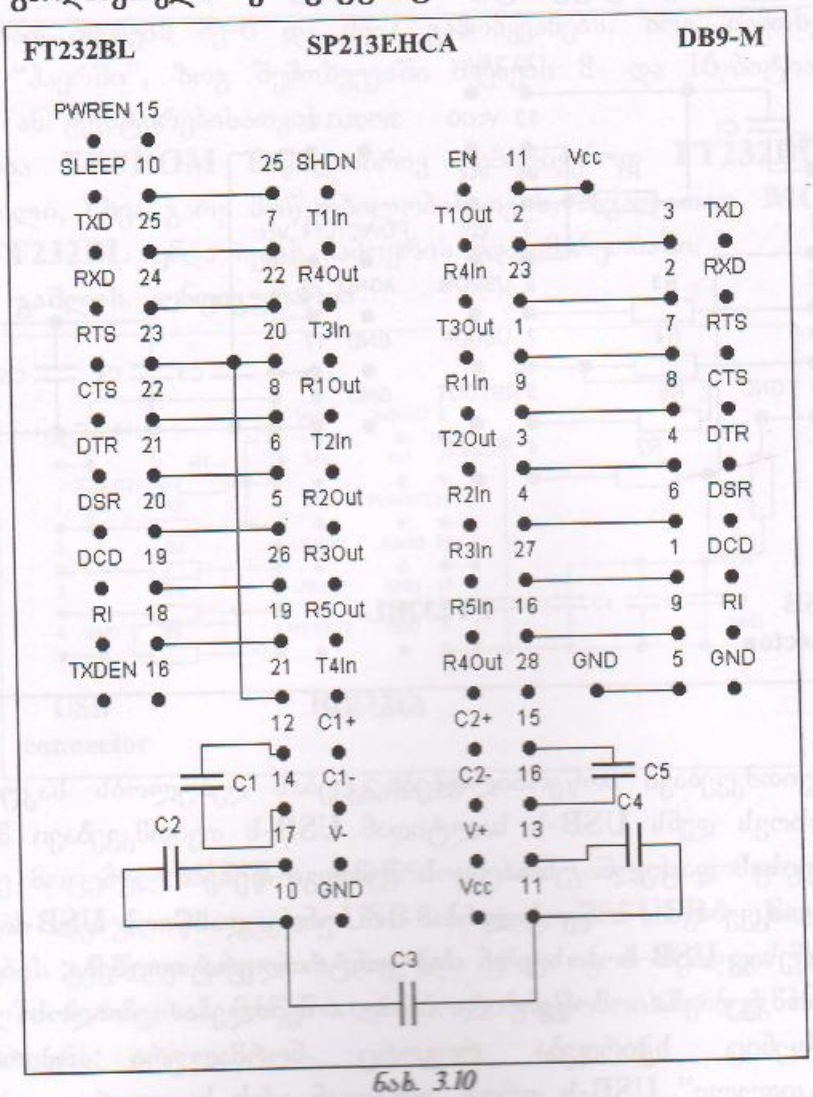
ნახ. 3.9

USB-ს თვითმკვებავი მოწყობილობები კვებას ღებულობს საკუთარი წყაროდან და არ მოითხოვს დენს **USB**-ს სალტიდან. **USB**-ს თვითმკვებავი მოწყობილობების ექსპლუატაციისას დაცული უნდა იყოს შემდეგი წესები:

- თვითმკვებავმა მოწყობილობამ არ უნდა დასწიოს **USB**-ს სალტის კვება, როდესაც **USB**-ს ჰოსტ ან ჰაბ-კონტროლერი ითიშება;
- თვითმკვებავმა მოწყობილობებმა შეიძლება მიიღოს იმდენი დენი, რამდენიც სჭირდება როგორც ნორმალური ოპერირებისას, ისე "დაკიდული" **USB**-ს დროს, ვინაიდან აქვს საკუთარი კვების წყარო;

- თვითმკვებავი მოწყობილობა გამოიყენება ნებისმიერ ჰოსტ **USB**-სთან, სალტესა და თვითმკვებავ **USB**-ს ჰაბთან. **PWRCTL (14)** იწევა ზემოთ, რითაც მოწყობილობას მიანიშნებს გამოიყენოს **USB**-ს სალტის პარამეტრები. კვების პარამეტრები **EEPROM**-ში დაპროგრამდება ნული მნიშვნელობით თვითკვების დროს. აუცილებელი მოთხოვნების შესრულებისათვის **1,5კომი** წინაღობის რეზისტორი **(R5)** აერთიანებს **USBDP**-სა და **RSTOUT#**-ს კვების სალტის წრედში. ამგვარად, **USB**-ს სალტე გამოიყენება **FT232B** მოწყობილობის **RESET#** გამომყვანის კონტროლისათვის. როდესაც **USB**-ს ჰოსტი ან ჰაბი ჩაირთვება, **RSTOUT#** აწევს **1,5კომ** რეზისტორს **USBDP**-მდე ანუ **3,3ვ**-მდე, რაც შეესაბამება მოწყობილობის იდენტიფიცირებას, როგორც სრული სიჩქარის **USB** მოწყობილობა. როდესაც **USB**-ს ჰოსტ ან ჰაბის კვება ითიშება, **RESET#** გადადის დაბალ დონეზე და მოწყობილობა ჩადგება ჩამოყრილ მდგომარეობაში. როდესაც **RESET#** და **RSTOUT#** დაბალია, **USBDP**-ს ძაბვის დონე დაიწევა **1,5კომი** რეზისტორით, რაც შეესაბამება მოწყობილობის გამორთულ მდგომარეობას.

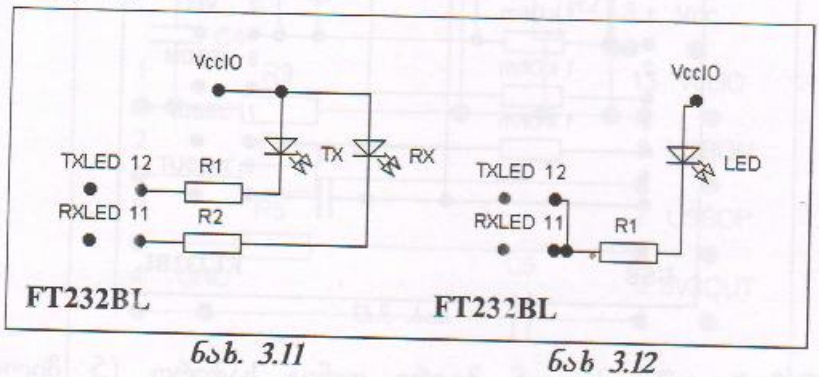
USB↔RS232 გარდამქმნელის კონფიგურაცია



ნახ. 3.10

3.10 ნახაზზე ნაჩვენებია **FT232BL**-ის **UART** ინტერფეისის **TTL – RS232** დონეების გარდამქმნელის ინტეგრალურ მიკროსქემასთან შეერთების კონფიგურაცია. **USB↔RS232** გარდამქმნელის სქემაში დონეების გარდაქმნისათვის გამოყენებულია ცნობილი "213" სერიის **TTL** მიკროსქემა. ამ მოწყობილობის ბრტყელ კორპუსში მოთავსებულია 4 გადამცემი და 5 მიმღები. ამავე კორპუსში ჩაშენებულია ძაბვის სტაბილიზატორი, რომელიც ნომინალური 5ვ **Vcc** ძაბვას გარდაქმნის **RS232**-ისათვის აუცილებელ ±9ვ ძაბვად. ამ მოწყობილობის განსაკუთრებული თავისებურებაა **SHDN#** გამომყვანის არსებობა, რომელსაც შეუძლია შეამციროს მოწყობილობის კვება დაბალ დონეზე, როცა **USB** იმყოფება "დაკიდების" რეჟიმში.

შუქდიოდის ინტერფეისი



ნახ. 3.11 *ნახ 3.12*

FT232B-ს აქვს ორი შეტანა/გამოტანის გამომყვანი შუქდიოდებზე მაკონტროლებელი ინდიკატორების ჩასართავად. ერთი ემსახურება გადასაცემ მონაცემებს, მეორე – მისაღებს. გადაცემა/მიღების შესაბამისი გამომყვანი გადადის ჩართული მდგომარეობიდან დაბალი პოტენციალის დონეზე და, შესაბამისად, ახორციელებს მონაცემთა გადაცემის ინდიკაციას შუქდიოდებზე.

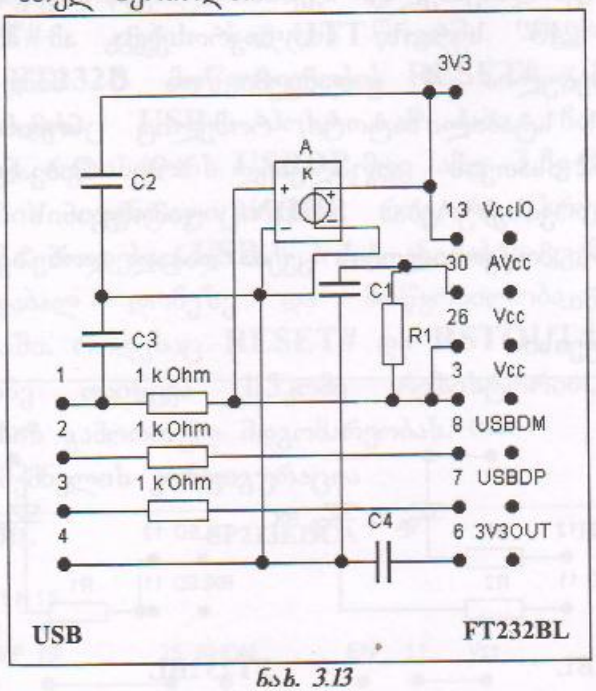
3.11 ნახაზზე მოცემულია სქემა, რომელშიც გამოყენებულია ორი ინდივიდუალური შუქდიოდი, ერთი გადასაცემი მონაცემებისათვის, მეორე – მისაღებისათვის. 3.12 ნახაზზე აღნიშნულ სქემაზე გადაცემისა და მიღების შუქდიოდები გაერთიანებულია "**ან**" ლოგიკით და გამოიყენება ერთი ინდიკატორი, რომელიც გვიჩვენებს ნებისმიერ აქტივობას – მონაცემების გადაცემას ან მიღებას.

კვების სალტის წრედი 3,3ვ გამშვები ლოგიკური სიგნალებისა და წყაროს ძაბვით

3.13 ნახაზზე აღნიშნულია, როგორი უნდა იყოს **FT232BL**-ის კონფიგურაცია იმისათვის, რომ შეიქმნას ინტერფეისი 3,3ვ ლოგიკურ მოწყობილობასთან. სქემაზე დისკრეტული 3,3ვ სტაბილიზატორი გამოიყენება 3,3ვ ლოგიკური მოწყობილობის კვებისათვის **USB** წყაროდან. **VccIO** შეერთებულია 3,3ვ სტაბილიზატორის გამოსასვლელთან, შედეგად **UART**-ის ინტერფეისის **IO** გამომყვანზე ჩნდება 3,3ვ ძაბვა.

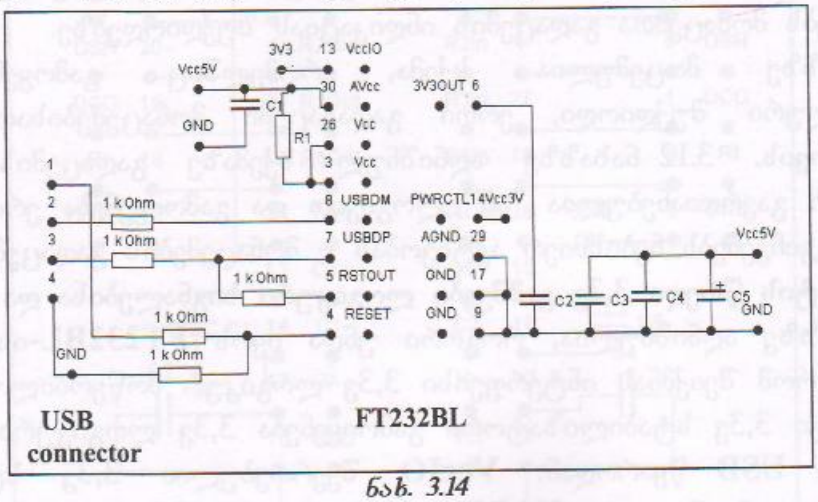
სქემებისათვის, რომლებიც **USB**-ს სალტიდან იკვებება, სტაბილიზატორის შერჩევისას საჭიროა გავითვალისწინოთ რამდენიმე გარემოება:

- სტაბილიზატორი უნდა ინარჩუნებდეს გამოსასვლელ ძაბვას, როდესაც შესავალი ძაბვა 4,35ვ ნაკლებია. სტაბილიზატორი უნდა შეირჩეს მცირე ძაბვის ვარდნით (**LDO**).
- სტაბილიზატორის სიმშვიდის დენი უნდა იყოს დაბალი, რათა მივიღოთ **USB**-ს "დაკიდებისას" 500 მიკროამპერზე ნაკლები ჯამური დენი, "დაკიდების" მთელ პერიოდში.



ნახ. 3.13

ზოგიერთ შემთხვევაში ძალიან მცირე დენია საჭირო (5 მილიამპერზე ნაკლები). ამ შემთხვევაში შესაძლებელია ჩაშენებულ **FT232BL**-ში სტაბილიზატორის გამოყენება 3,3ვ მისაღებად, ყოველგვარი სხვა კომპონენტების გარეშე. ამ შემთხვევაში მომხმარებელს უერთებენ **VccIO**-ის **3V3OUT** გამომყვანს **FT232BL**-ზე. **თვითკვების სქემა 3,3ვ ლოგიკის სიგნალებიდან და ძაბვის წყაროდან**

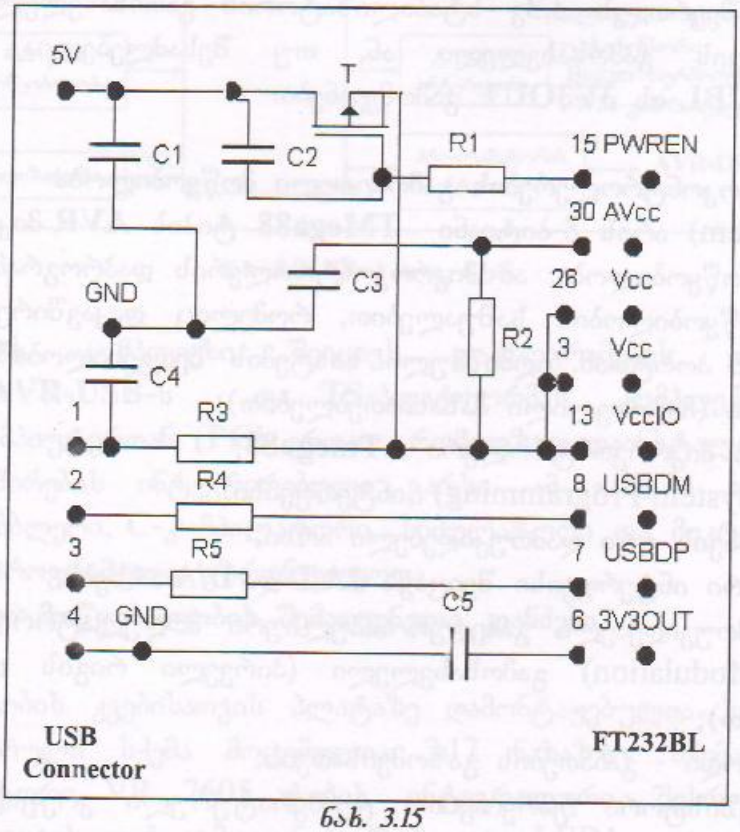


ნახ. 3.14

ამ შემთხვევაში **VccIO** იკვებება 3,3ვ გარე კვების წყაროდან იმ მიზნით, რომ მოწყობილობის **IO** გამომყვანს, მართვადი 3,3ვ ლოგიკის სიგნალებით, საშუალება მიეცეს მიუერთდეს 3,3ვ **MCU**-ს ან გარე მოწყობილობის ლოგიკას.

USB-ს თვითკვების დაპროექტებისას იყენებენ საკუთარი კვების წყაროს და არ ითვალისწინებენ არავითარი კვების მიღებას **USB**-ს სალტიდან. ასეთ შემთხვევაში სპეციალური ღონისძიებების გატარება არ არის საჭირო. **USB**-ს "დაკიდების" რეჟიმის დენის შესამცირებლად (0,5მა) მოწყობილობა არ ღებულობს კვებას **USB**-ს პორტიდან. შესაბამისი სქემა ნაჩვენებია 3.14 ნახაზზე.

სალტიდან მკვებავი წრედი კვების კონტროლით



ნახ. 3.15

თუ წრედი იკვებება **USB**-ს სალტიდან, აუცილებელია ძაბვის დაწევა **USB**-ს "დაკიდების" რეჟიმში, რომ მივიღოთ 500მკა-ზე ნაკლები დენი (გარე ლოგიკის ჩათვლით). ზოგიერთ გარე ლოგიკას თავად აქვს უნარი დასწიოს კვების მოხმარება დაბალ დენზე **PWREN#** გამომყვანის მონიტორინგით. იმ ლოგიკისათვის, რომელსაც ამის უნარი არ აქვს, **FT232BL** სთავაზობს მარტივ, მაგრამ ეფექტურ გზას გარე წრედების კვების გამორთვისათვის, სანამ **USB** იმყოფება "დაკიდების" რეჟიმში.

3.15 ნახაზზე მოცემულია, როგორ გამოვიყენოთ **P**-არხიანი **MOSFET** ტრანზისტორი გარე ლოგიკური წრედების კვების კონტროლისათვის.

კვების კონტროლის დაპროექტებისას გათვალისწინებულია შემდეგი ფაქტორები:

- კონტროლირებად კვების ლოგიკას უნდა ჰქონდეს საკუთარი ჩამოყრის წრედი, რომლითაც ავტომატურად ჩამოიყრება, როდესაც მოწყობილობა გამოდის "დაკიდული" მდგომარეობიდან;
- **FT232BL**-ს **EEPROM**-ში უნდა იყოს დაყენებული "დაკიდების" ოპცია;
- მოწყობილობებისათვის, რომლებიც იკვებებიან **USB**-ს მაღალი ძაბვით კვების სალტიდან (100-დან 500მა-დე დენი **USB**-ს სალტიდან), მოწყო-

ბილობების კვების მოხმარება უნდა იყოს დაყენებული **EEPROM**-ის მაქსიმალურ კვების ველში. მაღალი მოხმარების მოწყობილობებმა უნდა გამოიყენოს თავისი იდენტიფიკატორი **EEPROM**-ში სისტემის ინფორმირებისათვის საკუთარი კვების პარამეტრების შესახებ;

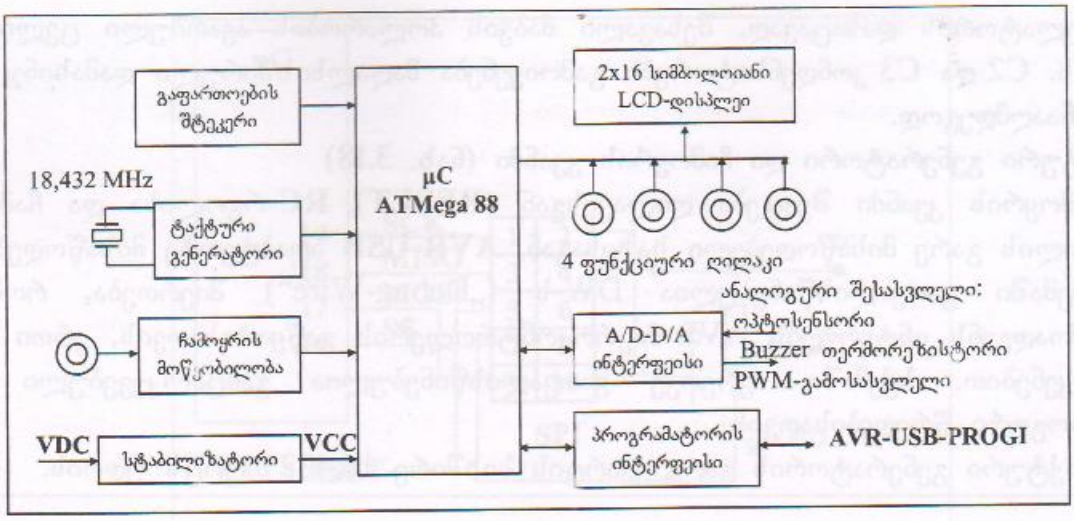
- 3,3ვ კონტროლირებადი წრედებისათვის **VccIO** არ უნდა დაიწიოს გარე ზეგავლენით (**PWREN#** ღებულობს **Vcc**-ს წყაროს **VccIO**-დან). უნდა შეერთდეს 3,3ვ სტაბილიზატორის გამოსასვლელი და გარე 3,3ვ ლოგიკის გამოსასვლელი ან, თუ შესაძლებელია, კვების **VccIO** **FT232BL**-ის **3V3OUT** გამომყვანები.

3.3. AVR მიკროკონტროლერების გამომცდელი მოწყობილობა

TS (test system) არის 8-ბიტიანი **TMega88** ტიპის **AVR** მიკროკონტროლერის გამომცდელი მოწყობილობა. ამ მიკროკონტროლერის დაპროგრამება ხდება **AVR-USB** გარე მოწყობილობის საშუალებით, რომელიც დაკავშირებულია **PC**-ის ან ლეპტოპის **USB** პორტთან. აღნიშნული სისტემის შედგენილობაში შედის შემდეგი მოწყობილობები (ჩამოთვლილი მახასიათებლებით):

- 8-ბიტიანი მიკროკონტროლერი **ATmega88**;
- **ISP** (In-System-Programming) ინტერფეისი;
- კვების ძაბვის ორი დამოუკიდებელი არხი;
- ანალოგური ინტერფეისი შეიცავს **2A/D** და **D/A** არხებს;
- ორი დამოუკიდებელი განედურ-იმპულსური მოდულატორის (**PWM** - Pulse Width Modulation) გამოსასვლელი (პირველი რიგის დაბალი სიხშირის ფილტრით);
- ფოტოდიოდი - განათების გაზომვისათვის;
- თერმორეზისტორი უარყოფითი ტემპერატურული კოეფიციენტით (utk-ით) (**NTC** - negative temperature coefficient) - ტემპერატურის გაზომვისათვის;
- ინტეგრირებული ხმოვანი სიგნალიზატორი (**buzzer**) - ხმის გამოცემისათვის;
- ორსტრიქონიანი 2x16 ტევადობის დისპლეი (**LCD**) - ინფორმაციის ვიზუალიზაციისათვის;
- ოთხი ფუნქციური ღილაკი მონაცემების შეყვანისათვის;
- **SPI** სპეციალური ინტერფეისი წარმოებული **USART** (Universal Synchronous-Asynchronous Receiver/Transmitter) უნივერსალური სინქრონულ-ასინქრონული მიმღებ/გადამცემის საფუძველზე;
- აღნიშნული მოწყობილობები განლაგებულია ორმხრივ ნაბეჭდ პლატაზე.

3.16 ნახაზზე მოცემულია **TS**-ის ბლოკ-სქემა და ბლოკების ფუნქციური დანიშნულება.



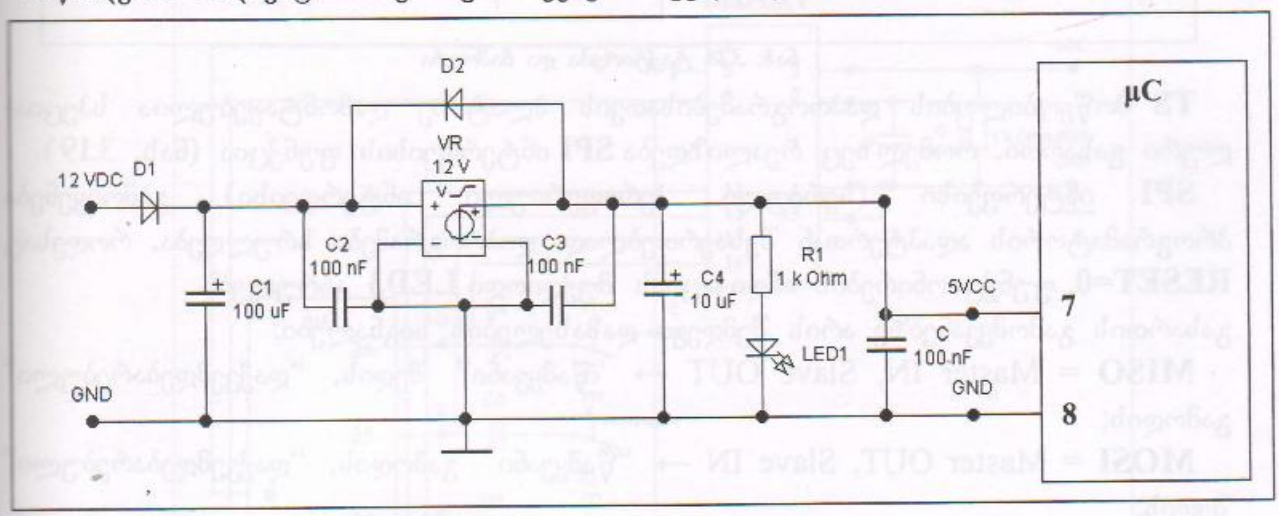
ნახ. 3.16. TS-ის ბლოკ-სქემა

პროგრამატორის კომპლექსი შეიცავს დაპროგრამების ადაპტერს **USB** ინტერფეისით, **AVR-USB**-ს და **TS**-პლატფორმას. კომპლექსი გამოიყენება პერსონალურ კომპიუტერთან (**PC**) ერთად, რომელზეც დაინსტალირებულია **AVR Studio** პროგრამირების ინტეგრირებული გარსი. ამ გარსში ინტეგრირებულია რედაქტორი, ასემბლერი, **C**-კომპილატორი, სიმულატორი და მიკროკონტროლერის დაპროგრამების პროგრამული უზრუნველყოფა.

3.3.1. გამომცდელი მოწყობილობის შემადგენელი კვანძები

სტაბილიზატორი

TS მოწყობილობის კვებისათვის პლატაზე დამონტაჟებულია სტაბილიზატორი, რომლის ელექტრული სქემა მოცემულია 3.17 ნახაზზე. სქემაში გამოიყენება ძაბვის რეგულატორი VR 7605 ტიპის ინტეგრალური შესრულებით. ძაბვის მიწოდების ინდიკაციისათვის გამოიყენება შუქდიოდი LED1.



ნახ. 3.17. სტაბილიზატორი

შესავალი მუდმივი ძაბვა VDC იცვლება 12-დან 9 ვოლტის დიაპაზონში. გამოსასვლელზე მიიღება 5ვ VCC სტაბილური ძაბვა. სქემის შესასვლელზე D1 დიოდი იცავს მოწყობილობას არასწორი პოლარობით. დიოდი D2 გამოიყენება

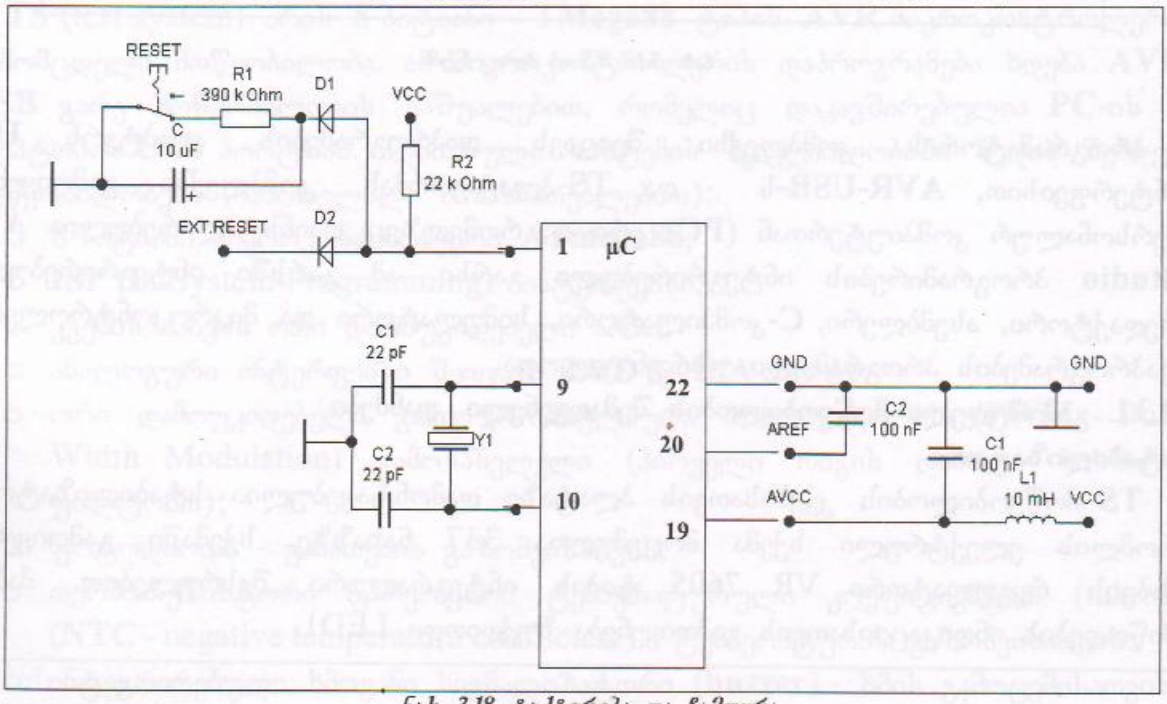
რეგულატორის დასაცავად, შესავალი ძაბვის პოლარობის ავარიული ცვლილების დროს. C2 და C3 კონდენსატორები გამოიყენება მაღალსიხშირული დამახინჯებების საწინააღმდეგოდ.

ტაქტური გენერატორი და ჩამოყრის კვანძი (ნახ. 3.18)

ჩამოყრის კვანძი შედგება ღილაკისგან (**RESET**), **RC**-რგოლისა და ჩამოყრის სიგნალის გარე მისაწოდებელი ხაზისაგან, **AVR-USB** ადაპტერზე მისაწოდებლად.

სქემაში გათვალისწინებულია DW-ს („Debug-Wire") მიერთება, რომელიც წარმოადგენს ინტერფეის AVR მიკროკონტროლერის გაწყობისათვის, ერთი ხაზის გამოყენებით. სქემაში აგრეთვე გათვალისწინებულია განცალკევებული კვება ანალოგური წრედებისათვის.

ტაქტური გენერატორის გარე კვარცის სიხშირე 16,432 მჰერცს უდრის.



ნახ. 3.18. ტაქტირება და ჩამოყრა

TS მოწყობილობის დაპროგრამებისათვის პლატაზე დამონტაჟებულია სპეციალური გასართი, რომელზეც რეალიზდება **SPI** ინტერფეისის ფუნქცია (ნახ. 3.19).

SPI ინტერფეისი (სერიული პერიფერიული ინტერფეისი) გამოიყენება პროგრამატორის ადაპტერთან შესაერთებლად. დაპროგრამება სრულდება, როდესაც **RESET=0**. ფუნქციონირების ინდიკაციას შუქდიოდი **LED3** ასრულებს.

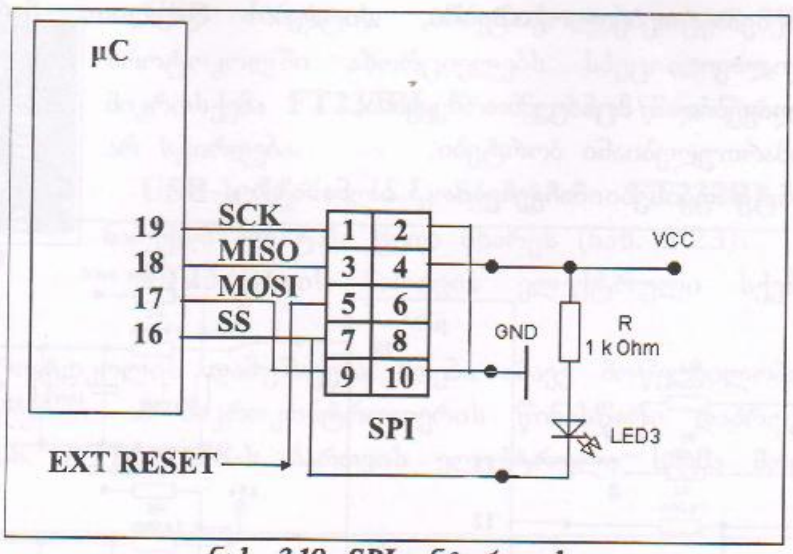
გასართის გამომყვანებზე არის შემდეგი დასახელების სიგნალები:

MISO = Master IN, Slave OUT → "წამყვანი" შედის, "დაქვემდებარებული" გამოდის;

MOSI = Master OUT, Slave IN → "წამყვანი" გამოდის, "დაქვემდებარებული" შედის;

SCLK = Serial Clock მიმდევრობითი საათი;

SS = Slave Select "დაქვემდებარებულის" შერჩევა.



ნახ. 3.19. SPI ინტერფეისი

მომსახურე ინტერფეისები 3.20 ნახაზზეა მოცემული.

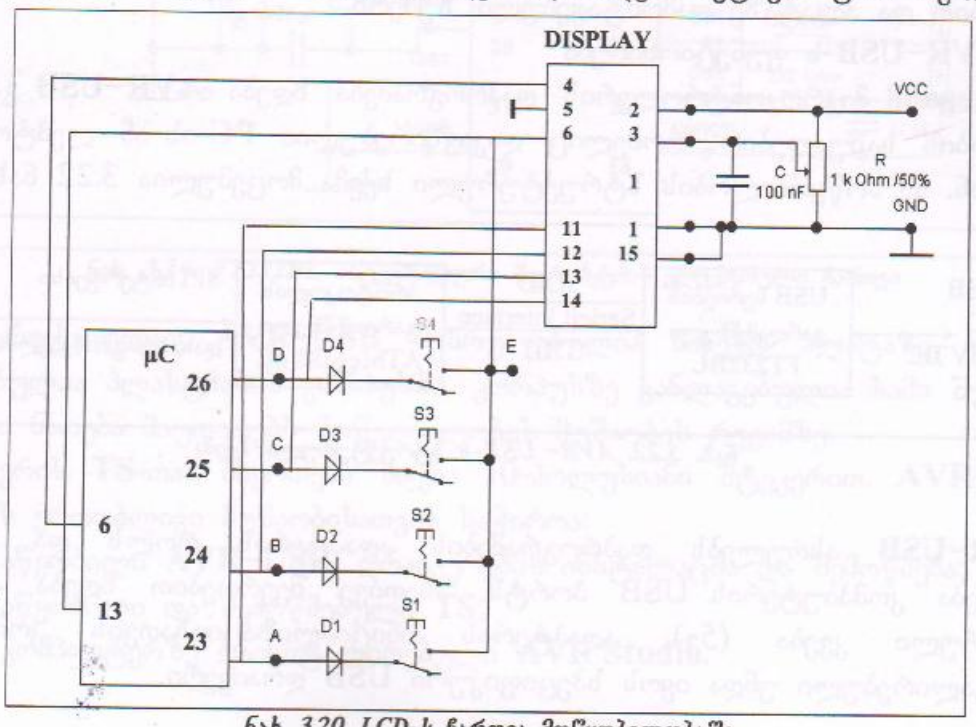
მომსახურე მოწყობილობას წარმოადგენს **LCD** ორსტრიქონიანი დისპლეი ნათებით. დისპლეის მართვა ხორციელდება ოთხი ბიტით;

RS: = Daten/Control) მონაცემები/მართვა;

=1: = Enable Display დისპლეი ჩართულია;

Poti: = კონტრასტულობა.

დისპლეის კონტრასტის შეცვლა შესაძლებელი **R** პოტენციომეტრით. განსაკუთრებული მუშაობის რეჟიმში, პოტენციომეტრის მბრუნავი ღერძი იმყოფება მარცხენა ბოლო პოზიციაში. ამ შემთხვევაში, დისპლეის აქვს განსაკუთრებულად სუსტი კონტრასტსი. ამიტომ პოტენციომეტრის ღერძი უნდა გადავაადგილოთ 45°-ზე. დისპლეის კონტრასტულობა რეგულირდება დისპლეის აქტივიზაციის შემდეგ.



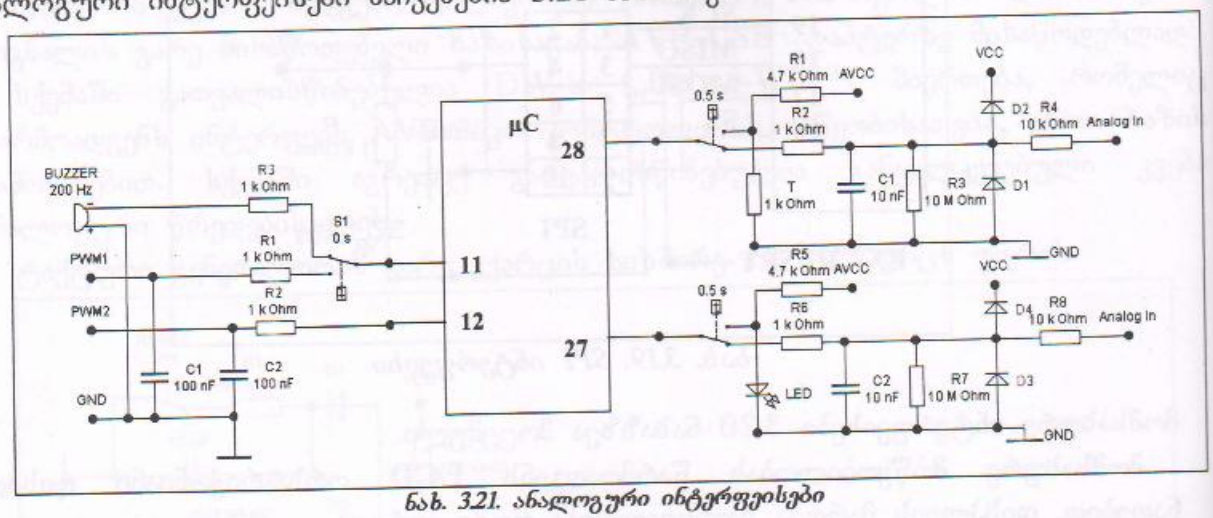
ნახ. 3.20. LCD-ს ჩართვა მოწყობილობაში

დისპლეის შემაერთებელ ხაზებში, დიოდების გავლით, ჩართულია ოთხი ფუნქციური ღილაკი:

E=0: = ღილაკებიდან მონაცემთა შეყვანა;

µC-ს ორმიმართულებიანი პორტები;

ანალოგური ინტერფეისები ნაჩვენებია 3.21 ნახაზზე.



ნახ. 3.21. ანალოგური ინტერფეისები

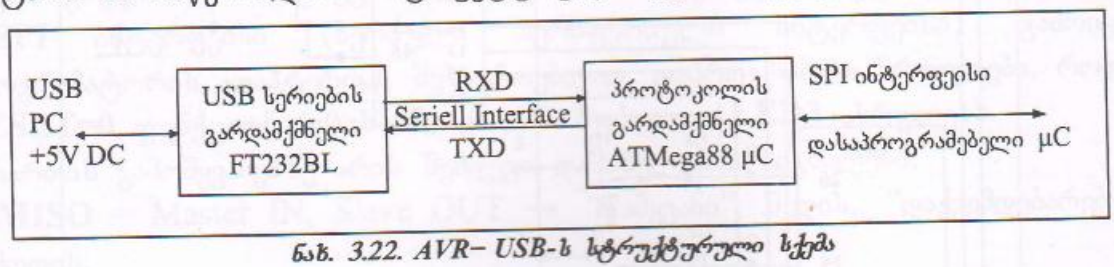
მოწყობილობაში გათვალისწინებულია ორი ანალოგური შესასვლელი 0...5ვ ძაბვაზე. ორივე შესასვლელზე გამოყენებულია დაცვა გადამეტებული ძაბვისგან (D1, D2, R4, ...), აგრეთვე DC/AC მაკავშირებელი წრედის (Analog In) შერჩევა გადამრთველიდან და დაბალი სიხშირის ფილტრი (C1, C2).

მოწყობილობით ხდება ტემპერატურისა (თერმორეზისტორი NTC – **T**) და განათებულობის (ფოტოელემენტი LED) გაზომვა.

მოწყობილობას აქვს ორი რვაბიტიანი PWM გამოსასვლელი დაბალსიხშირიანი შეერთებით და პლატაზე დამონტაჟებული ზუმერი.

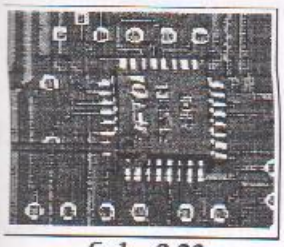
3.3.2. AVR–USB-ს ფუნქციონირება

ATmega88 მიკროკონტროლერის დაპროგრამება ხდება **AVR–USB** გარე მოწყობილობის საშუალებით, რომელიც დაკავშირებულია **PC**-ის ან ლეპტოპის **USB** პორტთან. ამ მოწყობილობის სტრუქტურული სქემა მოცემულია 3.22 ნახაზზე.



ნახ. 3.22. AVR– USB-ს სტრუქტურული სქემა

AVR–USB ასრულებს დაპროგრამების ადაპტერის როლს და უშუალოდ უერთდება კომპიუტერის USB პორტს. ასეთივე შეერთებით ხდება ადაპტერის ელექტრული კვება (5ვ). ადაპტერის ინიციალიზაციისათვის კომპიუტერზე დაინსტალირებული უნდა იყოს სპეციფიკური **USB** დრაივერი.



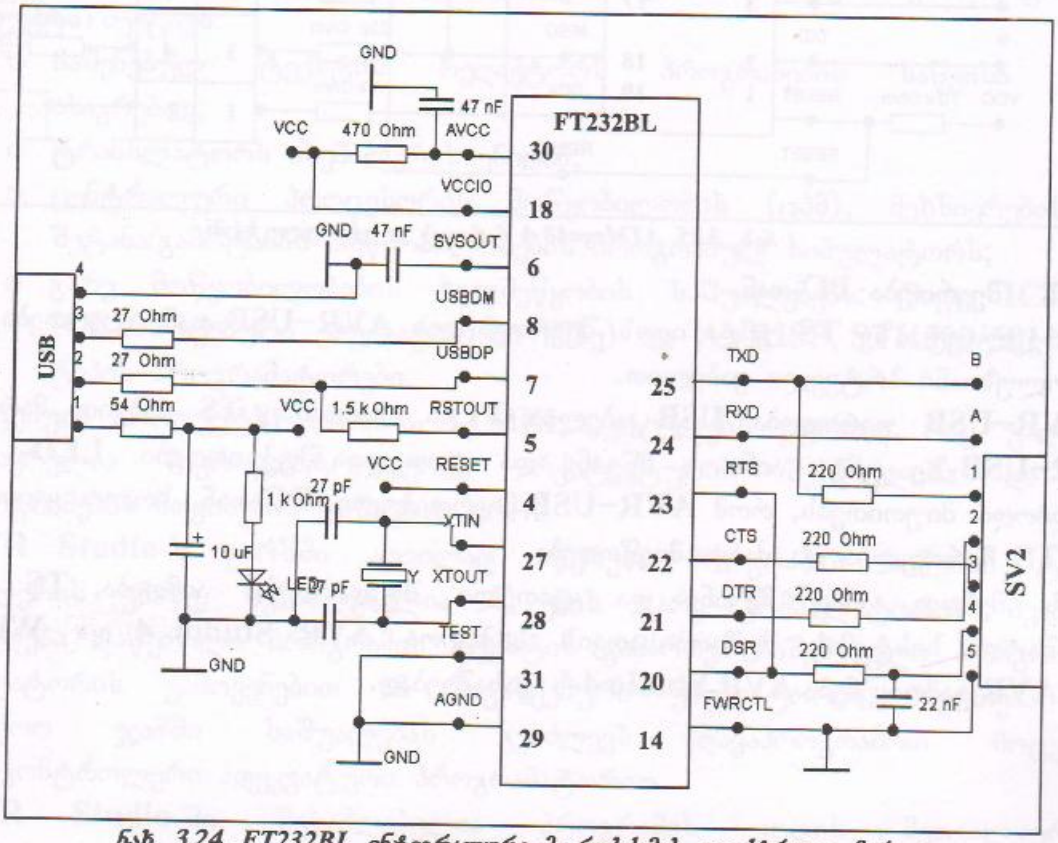
ნახ. 2.23

USB პორტის პროტოკოლის გარდაქმნას სერიულ პროტოკოლში ახორციელებს სპეციალური ინტეგრალური მიკროსქემა **FT232BL**, რომელსაც წინასწარი დაპროგრამება არ სჭირდება.

USB-ს სერიული კონვერტორი **FT232BL** განლაგებულია ნაბეჭდი პლატას ქვედა მხარეს (ნახ. 2.23).

FT232BL-ის ჩართვის ელექტრული სქემა მოცემულია 3.24 ნახაზზე.

სერიული პროტოკოლის დამუშავება ხდება ამავე მოწყობილობის **ATMega88** მიკროკონტროლერში. ამ მიკროკონტროლერის წინასწარი დაპროგრამება ხდება **USB** პორტიდან. **ATMega88**-ს ჩართვის ელექტრული სქემა მოცემულია 3.25 ნახაზზე.

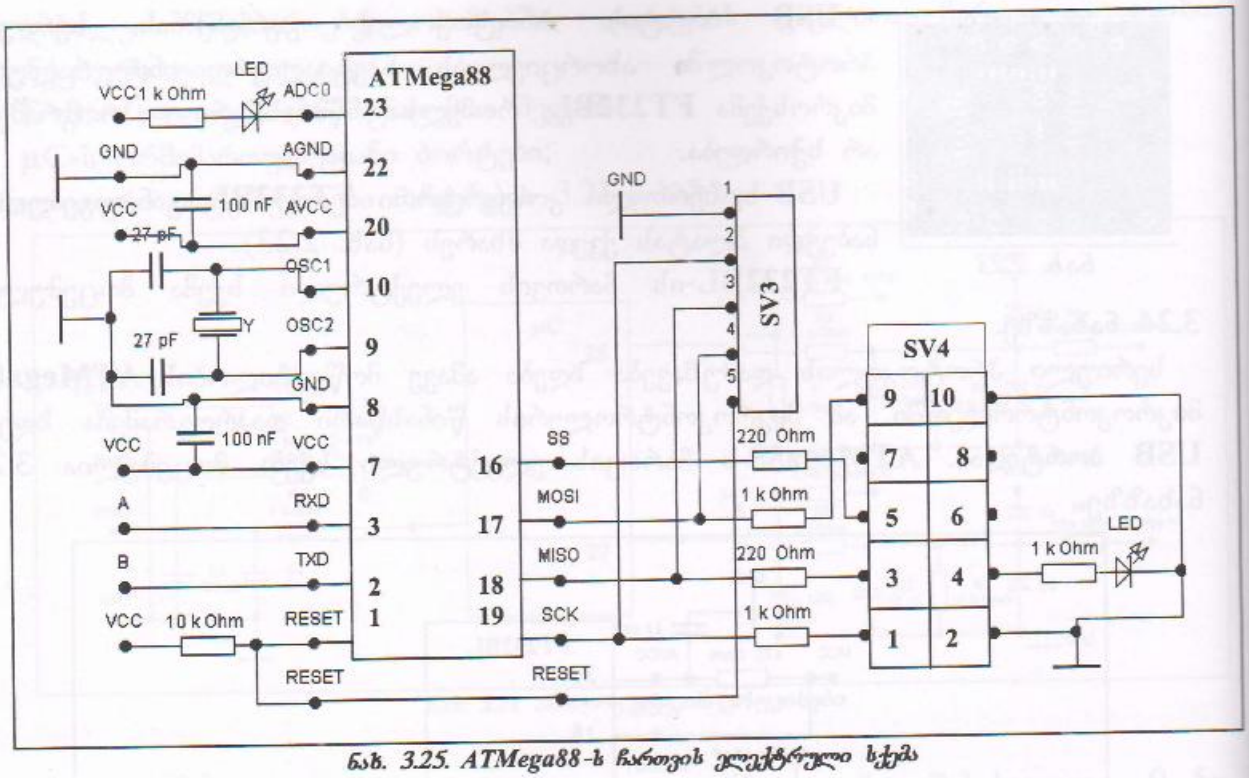


ნახ. 3.24. FT232BL ინტეგრალური მიკროსქემის ელექტრული ჩართვა

კონსტრუქციულად **AVR–USB** შესრულებულია ნაბეჭდ პლატაზე, რომელიც მოთავსებულია პლასტმასის კორპუსში. კორპუსზე განლაგებულია სამი შუქდიოდი, რომელთა ნათება მიუთითებს მოწყობილობის მუშაობის რეჟიმზე.

ადაპტერის **TS**-თან მიერთება ხდება 10-პოლუსიანი შტეკერით. **AVR–USB**-სა და **TS**-ის ერთობლივი მუშაობისათვის საჭიროა:

- აწყობილი **AVR–USB**, დრაივერების ინსტალაცია და შემოწმება;
- აწყობილი და შემოწმებული **TS**;
- კომპიუტერზე დაინსტალირებული **AVR Studio.**



ნახ. 3.25. ATMega88-ს ჩართვის ელექტრული სქემა

TS-ის შეერთება PC-თან

პირველ ეტაპზე **TS** უნდა იყოს შეერთებული **AVR–USB**-თან. შეერთება ხდება 10-პოლუსიანი ბრტყელი კაბელით.

AVR–USB უერთდება **USB**-კაბელით **PC**-ს, საიდანაც **TS** ძაბვით მარაგდება. **AVR–USB**-ზე უნდა აინთოს მწვანე და ყვითელი შუქდიოდები. **LED-I** მწვანე შუქდიოდი მიუთითებს, რომ **AVR–USB** მიერთებულია **PC**-თან, ხოლო ყვითელი – რომ **TS** ჩართულია და ძაბვა მიეწოდება.

მას შემდეგ, რაც მწვანე და ყვითელი შუქდიოდები აინთება, **TS** მზადაა სამუშაოდ. სისტემის მუშაობისათვის საჭიროა **AVR Studio 4** და **WinAVR**. **WinAVR** გამოიყენება **AVR Studio 4**-ს გასაშვებად.

თავი 4. AVR მიკროკონტროლერების დაპროგრამების პროცესის უზრუნველყოფა

4.1. პროექტის შექმნა და გამართვა AVR Studio-ს გამოყენებით

პროგრამების შემუშავების ცნობილი ინტეგრირებული გარსები ამაღლებს პროგრამისტის შრომის ნაყოფიერებას და საშუალებას იძლევა თავიდან ავიცილოთ რუტინული სამუშაო. სხვადასხვა მწარმოებლების მიერ გამოშვებული პროგრამების შემუშავებისთვის საჭირო ინტეგრირებული პაკეტები ფუნქციურად ერთმანეთის მსგავსია, მაგრამ განსხვავდება წარმოდგენილი სერვისული შესაძლებლობით, მუშაობის მოხერხებულობითა და გენერირებული მანქანური კოდის ხარისხით.

მაგალითად, პროგრამების პაკეტი **AVR Studio** გვევლინება ინტეგრირებულ გარსად **AVR** სერიის მიკროკონტროლერების გამართვისათვის. ეს პროგრამული პროდუქტი შეიცავს:

- ჩაშენებულ ტექსტის რედაქტორს პროგრამების საწყისი კოდის ასაკრებად;
- ტრანსლატორს ასემბლერის ენიდან;
- ცენტრალური პროცესორის მოწყობილობის (ცპმ), მეხსიერებისა და შეტანა/გამოტანის მოწყობილობების პროგრამულ სიმულატორს;
- გარე მოწყობილობების ხელშეწყობის საშუალებებს: შიგა **ICEPRO, ICE200** სქემური ემულატორები (**შსე**) და **AVRISP, STK500/501** და **TS** ტიპის პროგრამატორები.

მოცემულ პროდუქტს კომპანია **ATMEL** ავრცელებს უფასოდ, რაც უაღრესად ხელსაყრელია მიკროკონტროლერის არქიტექტურის შესწავლისა და მისი დაპროგრამების ათვისების დაწყების დროს.

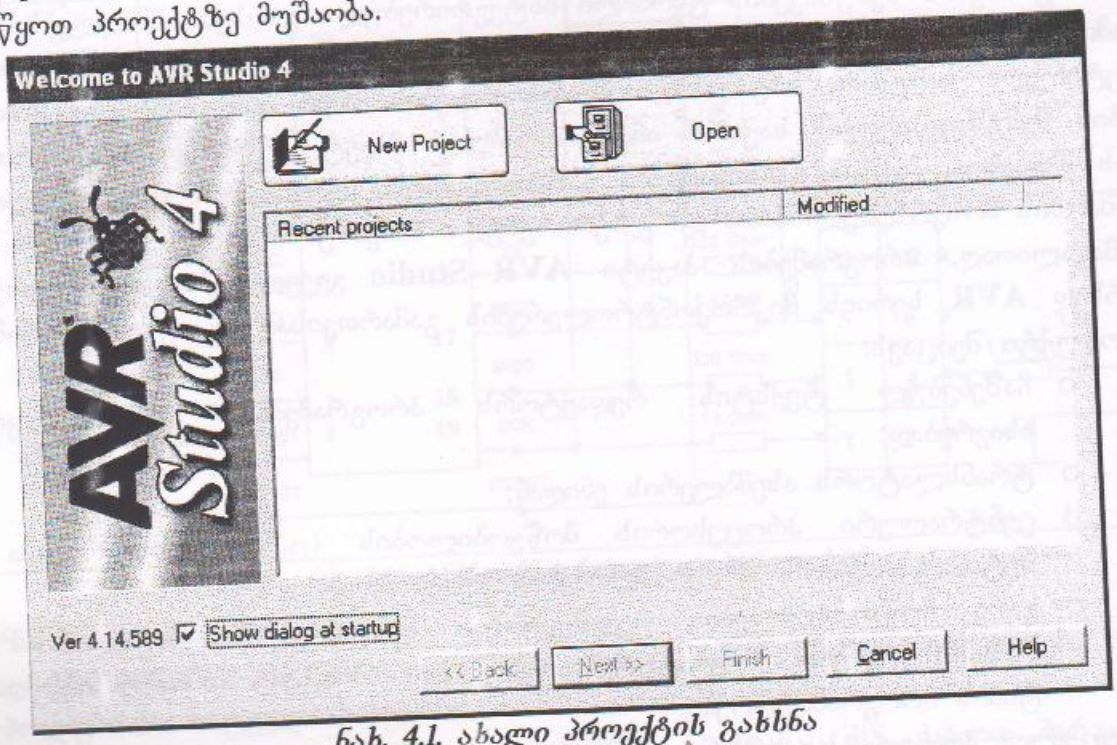
AVR Studio-ს გარსში ადვილად იწერება პროგრამის საწყისი ტექსტი ასემბლერის ენაზე. შესაძლებელია **C** ენის გარე კომპილატორის შეერთება. სრულდება დაწერილი პროგრამის ტექსტის გამართვა ჩაშენებული პროგრამული სიმულატორის გამოყენებით ან შიგასქემური ემულატორის მიერთება, რაც საბოლოო ჯამში საშუალებას გვაძლევს დავაპროგრამოთ მოცემული მიკროკონტროლერი ადეკვატური პროგრამატორით.

AVR Studio-ში შესაძლებელია პროგრამის კოდის მეთვალყურეობა, შიგასქემური ემულატორის ან პროგრამული სიმულატორის საშუალებით. პროგრამის კოდის მეთვალყურეობისათვის საჭიროა მისი კომპილირება ობიექტის ფაილის გენერირებისათვის, რომელიც ჩაიტვირთება **AVR Studio**-ში.

AVR Studio-ს გარსში პროგრამირებისას აუცილებელია შესრულდეს მოქმედებების სტანდარტული თანამიმდევრობა:

1. პროექტის შექმნა;
2. ფაილის ჩატვირთვა;
3. კომპილაცია;
4. სიმულაცია;
5. **hex**-კოდის ჩატვირთვა მიკროკონტროლერში.

კომპიუტერზე **AVR Studio**-ს დაინსტალირებისათვის საჭიროა ოპერაციული სისტემა **Microsoft Windows XP**. დაინსტალირებული **AVR Studio**-ს გახსნის შემდეგ იხსნება 4.1 ნახაზზე ნაჩვენები ფანჯარა. მიღებულ ფანჯარაში მაუსის მარცხენა ღილაკით ვაჭერთ **"New Projekt"**-ს, რაც საშუალებას გვაძლევს დავიწყოთ პროექტზე მუშაობა.



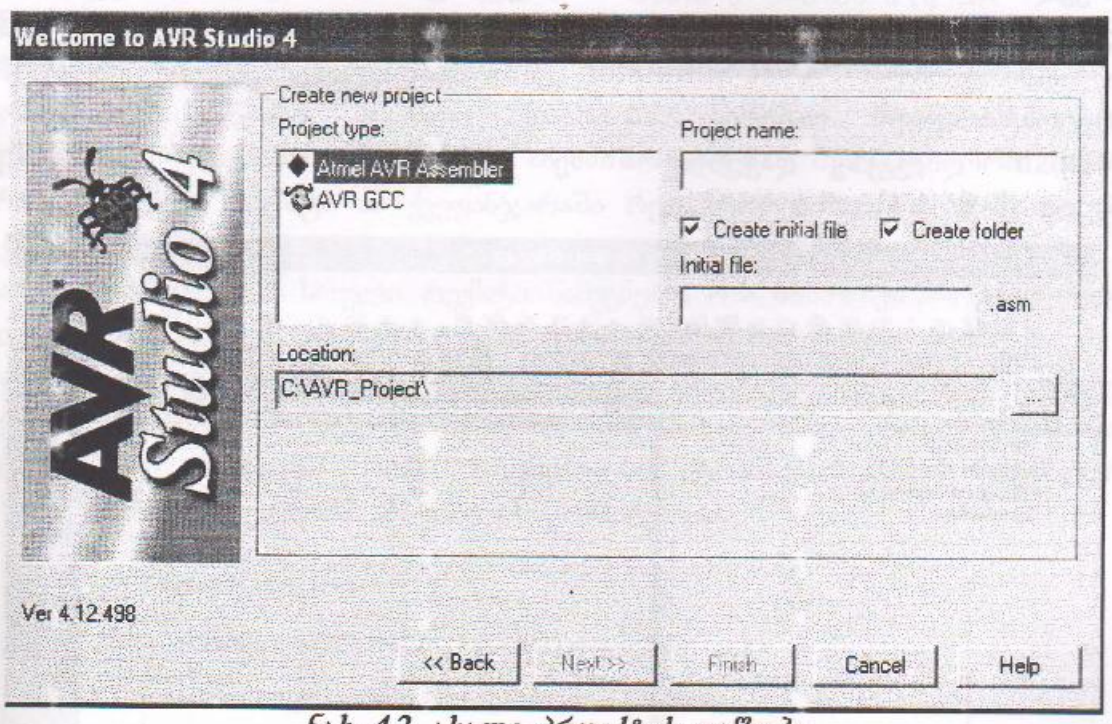
ნახ. 4.1. ახალი პროექტის გახსნა

გახსნილ **"Create new Project"**დიალოგურ ფანჯარაში (ნახ. 4.2) არსებულ **"Project name"**-ს გრაფაში შეგვაქვს შერჩეული პროექტის სახელი (მაგ., **lab1**), რაც **"Initial file"**-შიც აისახება და ინიციალიზაციის ფაილის სახელი (ჩვენ მაგალითში - **lab1.asm**).

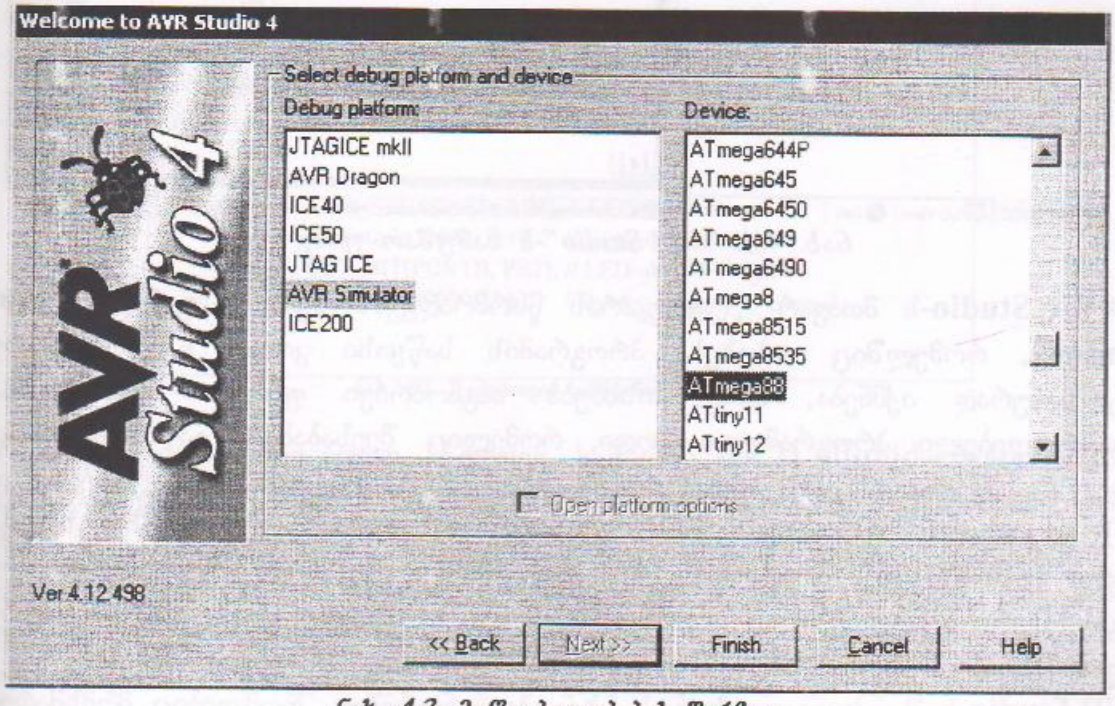
ამ ფაილების ადგილმდებარეობა ფიქსირდება **"Location"** სტრიქონში. კატალოგი შეიძლება დაიწეროს შესაბამის ტექსტურ ველში ან შეირჩეს არსებული საქაღალდეების ჩამონათვლიდან. ეს სრულდება სტრიქონის მარჯვნივ სამი წერტილით აღნიშნულ ღილაკზე დაჭერით. თუ კატალოგი არ დაფიქსირდება, მაშინ ის შეიქმნება ავტომატურად, ყოველგვარი შეხსენების გარეშე. შესასრულებელი პროექტის ყველა ფაილს **AVR Studio** შეინახავს შერჩეულ კატალოგში.

"Projekt type" სტრიქონში აირჩევა პროექტის ტიპი. პროექტის ტიპს ირჩევენ შემდეგი მონაცემების გათვალისწინებით:

- **AVR Assembler**-ის პროექტის ტიპს ვირჩევთ მაშინ, როდესაც პროექტის ტექსტის კოდის კომპილაციისათვის ვიყენებთ ჩაშენებულ ტრანსლატორს. ამ შემთხვევაში არ არის აუცილებელი ტრანსლატორის დამატებითი აწყობა.
- **AVR GCC (GNU Compiler)** ტიპი გამოიყენება მაშინ, როდესაც ხელით ხდება **AVR Studio**-ს აწყობა (გარე კომპილატორების გამოსაყენებლად). ჩვენ მაგალითში სწორედ ეს ტიპია გამოყენებული, რადგან პროექტის პროგრამირებისათვის **C** ენა გამოიყენება.



ნახ. 4.2. ახალი პროექტის დაწყება

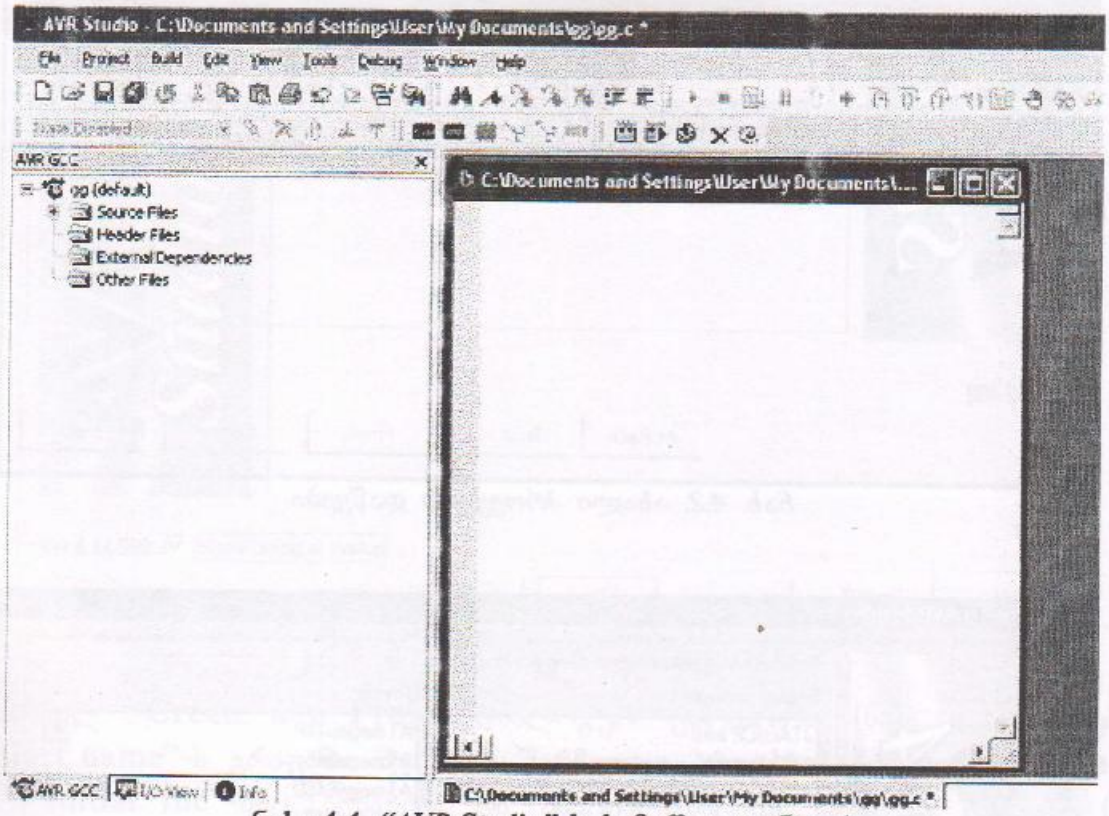


ნახ. 4.3. მოწყობილობების შერჩევა

ყველა ჩამოთვლილი მონაცემების შეტანის შემდგომ პროგრამის გასაგრძელებლად ვაჭერთ **"Next"** ღილაკს, გაიღება ახალი ფანჯარა **"Select debug platform and device"** (ნახ. 4.3), სადაც შეირჩევა გასამართი პლატფორმა (სიმულატორი ან ემულატორი) და მიკროკონტროლერის ტიპი.

ყოველ შემოთავაზებულ ემულატორს აქვს გასამართი მიკროკონტროლერების საკუთარი ნუსხა, რის გათვალისწინებითაც ხდება მისი შერჩევა. განსახილველ მაგალითში ვირჩევთ **"AVR Simulator"** გასამართ პლატფორმას და **"ATmega88"** მიკროკონტროლერს.

"Finish" ღილაკზე დაჭერით იხსნება **"AVR Studio"** ცარიელი სამუშაო ფანჯარა (ნახ. 4.4).



ნახ. 4.4. "AVR Studio"-ს სამუშაო ფანჯარა

AVR Studio-ს მთავარ ქვეფანჯარას წარმოადგენს წყაროს ფანჯარა (**Source window**), რომელშიც აისახება პროგრამის საწყისი კოდი. წყაროს ფანჯარა ავტომატურად იქმნება, როცა იხსნება საგნობრივი ფაილი. მასში აისახება შესასრულებელი პროგრამული კოდი, რომელიც შეესაბამება გახსნილი ფაილის სახელს. ამ ფანჯარაში პროგრამული მთვლელის მაჩვენებელი (ყვითელი ისარი =>) ყოველთვის მიუთითებს პროგრამის შესასრულებელ სტრიქონზე. სტატუსის ზოლი (**Status bar**) მიუთითებს, თუ რა სახისაა შემსრულებელი მოწყობილობა: შიგასქემური ემულატორი თუ პროგრამული სიმულატორი. წყაროს ფანჯარა ინფორმაციას გვაძლევს შესასრულებელი პროგრამის ლოგიკის შესახებ. ამასთან, **AVR Studio**-ს შეუძლია გვიჩვენოს სხვა ქვეფანჯრებიც, რომლებიც მომხმარებელს შესაძლებლობას მისცემს განახორციელოს საკმარისი კონტროლი შემსრულებელი მოწყობილობის (ემულატურის ან სიმულატორის) ყოველი ელემენტის მდგომარეობაზე.

მაგალითად, ქვეფანჯარაში (**AVR GCC - Project windows**) აისახება პროექტში შესავალი ფაილების სახელები. **"Assembler Files"** და **"Other Files"**

საქაღალდეებში ინახება დასამუშავებელ პროექტთან დაკავშირებული ფაილები (4.4 ნახაზზე ისინი ჯერ ცარიელია).

4.1.1. საწყისი ტექსტის ჩატვირთვა

პროგრამის საწყისი ტექსტი თავსდება სამუშაო ფანჯრის წყაროს ქვეფანჯარაში **(Source window**-ში). ეს სრულდება შემდეგი ორი ხერხით: აიკრიფოს მთელი ტექსტი ამ ქვეფანჯარაში რედაქტორებისათვის ან ჩაიტვირთოს უკვე არსებული ფაილი.

მარტივი პროგრამის სრული ტექსტი ნაჩვენებია 4.5 ნახაზზე. პროექტი შედგება რამდენიმე ფაილისაგან. ამ შემთხვევაში ერთი ფაილი ძირითადად მიიჩნევა.

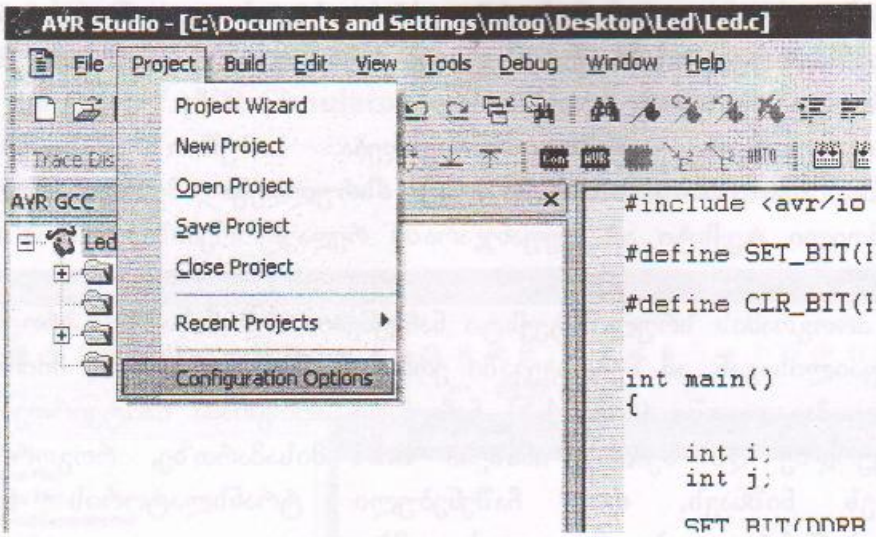
განხილულ მაგალითში (ნახ. 4.5) ნაჩვენები პროგრამა ჩაიტვირთება პროექტის პირველ გვერდზე და შემდეგ ინახება **lab1** მისამართზე, როგორც ძირითადი ფაილი. ეს ნიშნავს, რომ ჩაშენებული ტრანსლატორის (ასემბლერის) გამოძახებისას შესრულდება ამ ფაილის კომპილაცია.

```
#include <lab1.h>

#define SET_BIT(PORT, BIT)  ((PORT) |= (1<<BIT))

#define CLR_BIT(PORT, BIT)  ((PORT) &= ~(1<<BIT))

int main()
{
        int i;
        int j;

        SET_BIT(DDRB, DDB2); // გამოსვლის დაშვება Data Direction-ით
                             // რეგისტრაცია
        while(1)
        {
                CLR_BIT(PORTB, PB2); // LED –ის ჩართვა!
                for(i=0; i<30000; i++);    // ციკლის დალოდება
                SET_BIT(PORTB, PB2); // LED-ის გამორთვა!
                for(j=0; j<30000; j++);    // მეორე ციკლის დალოდება
        }
}
```

ნახ. 4.5. შუქდიოდის ციმციმის პროგრამა

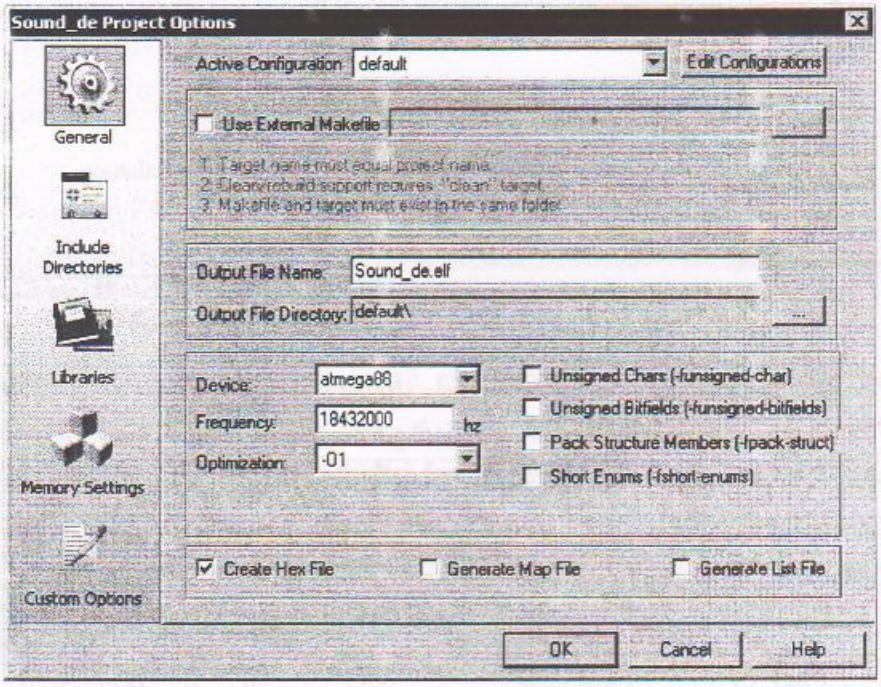
პროექტის შესასრულებლად აგრეთვე საჭიროა **LCD (liquid-crystal display)** თხევადკრისტალიანი დისპლეის ფუნქციების განსაზღვრა და შესაბამისი ფაილის შექმნა დასაპროექტებელი მოწყობილობის ინდიკაციის გამართვისათვის. ამ ფუნქციების ტექსტი ჩაიტვირთება პროექტის მეორე გვერდზე და ინახება საკუთარი სახელით იმავე მისამართზე.

კომპილაციის დაწყებამდე, შესაქმნელი ***.hex** ფაილის პარამეტრების განსაზღვრისათვის, სამუშაო ფანჯრის **"Project"** მენიუში ვირჩევთ **"Configuration Options"** (ნახ. 4.6).



ნახ. 4.6. ბრძანება "Configuration Options"

გაიხსნება ფანჯარა, რომელიც მოცემულია 4.7 ნახაზზე, რომელშიც განისაზღვრება სასურველი პარამეტრები.



ნახ. 4.7. პროექტის პარამეტრების დაყენება

4.1.2. კომპილაცია

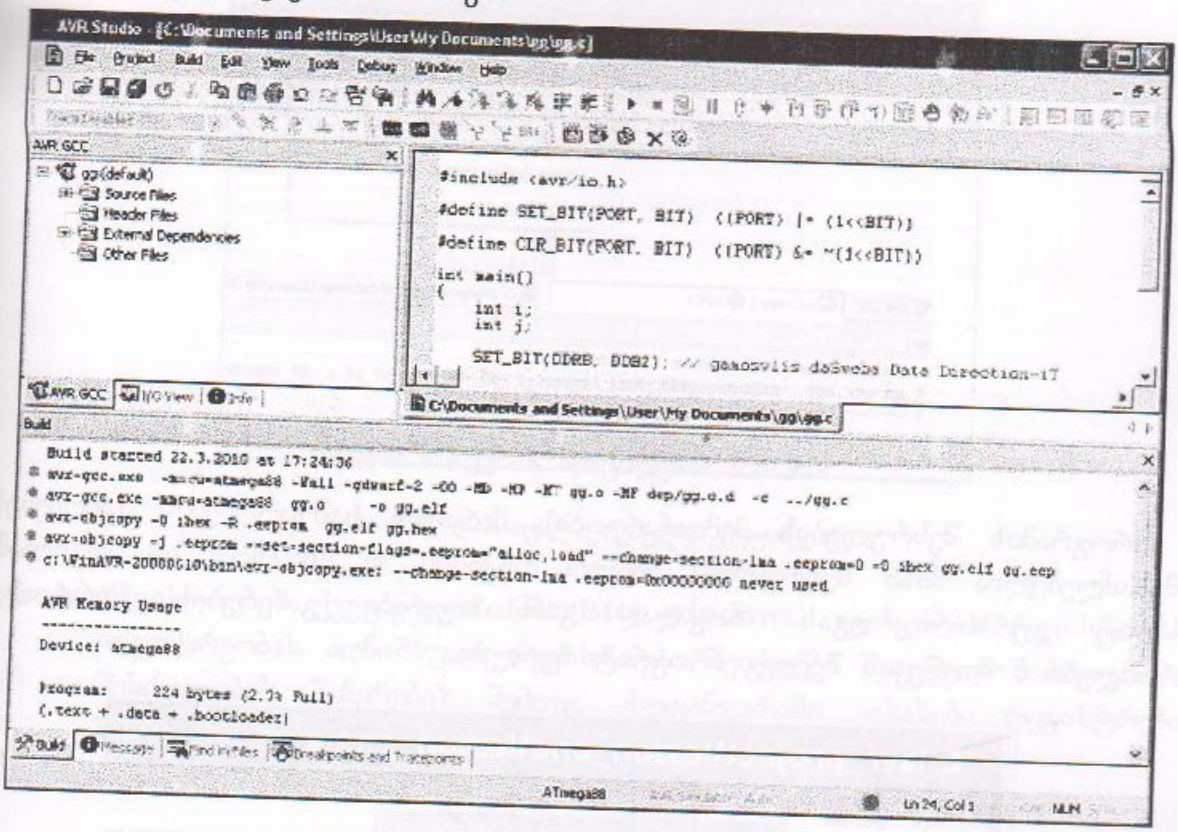
პროექტის კომპილაცია სრულდება "**\Build\Build**" ბრძანებით მენიუდან ან კომპიუტერის "**F7**" ღილაკზე დაჭერით. კომპილაციის პროცესი აისახება "**Build**" ქვეფანჯარაში (ნახ. 4.8).

თუ "**Build**" ქვეფანჯრის ზოლში გამოჩნდება წითლად აღნიშნული შეცდომა, მაშინ ის აუცილებლად უნდა შესწორდეს. ყვითელი აღნიშვნა მიუთითებს

სინტაქსურ შეცდომაზე, რომელიც ასევე სასურველია გასწორდეს. თუ **"Build"** პროცესი შეცდომის გარეშე ჩაივლის, მაშინ მიღებული ***.hex** ფაილი მზადაა პროგრამატორზე გასაშვებად.

სამუშაო ფანჯარაში, პირველ რიგში, ვხსნით **"File"** მენიუს **"Save As"** ბრძანების მეშვეობით, ვინახავთ შექმნილ პროგრამას მითითებულ მისამართზე. ***hex** ფაილი ავტომატურად შეიქმნება აღნიშნულ მისამართზე, საიდანაც პროგრამატორის გავლით ჩაიტვირთება მიკროკონტროლერში.

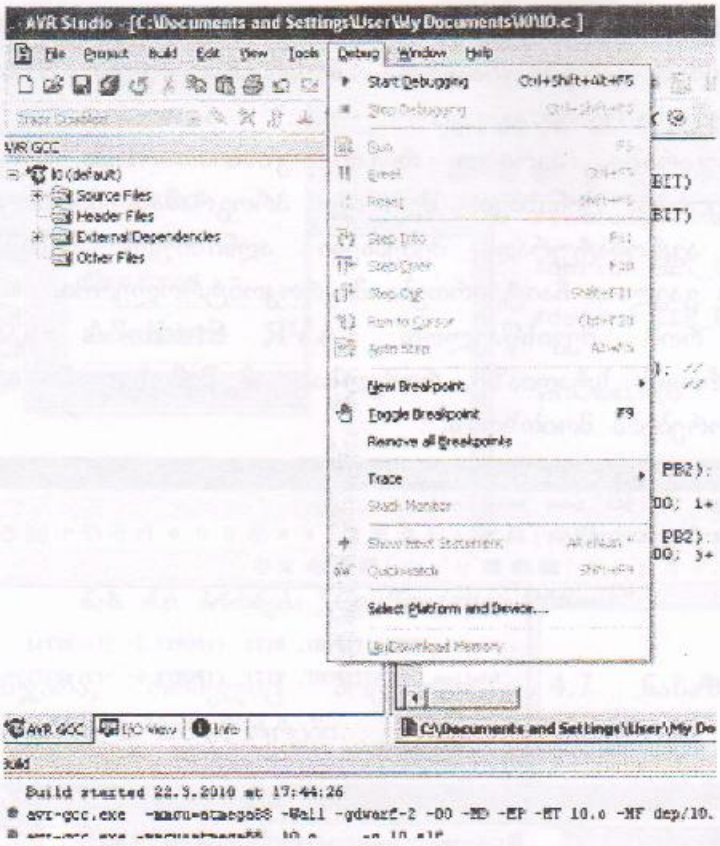
იმისათვის, რომ შევისწავლოთ **"AVR Studio"**-ს გარემოში მუშაობა, გადავიდეთ გამართვის სტადიაზე, რომელსაც იმ შემთხვევაში ატარებენ, როდესაც პროგრამა შესწორებებს მოითხოვს.



ნახ. 4.8. კომპილაციის შედეგები

4.1.3. სიმულაცია

პროგრამის გამართვა ანუ პროგრამული კოდის სიმულაციის რეჟიმში შესრულება იწყება **"Debug\Start Debugging"** ბრძანებით (ნახ. 4.9). პროცესის გაშვების შემდეგ პროგრამის საწყისი კოდის ფანჯარაში ჩნდება მარკერი, რომელიც იმყოფება მომდევნო ბიჯზე შესასრულებელ სტრიქონში (ნახ. 4.10). ყვითელი ისარი პროგრამული მთვლელის მაჩვენებელია. მიუთითებს შემდგომ შესასრულებელ ინსტრუქციაზე.



ნახ. 4.9. სიმულაციის პროცესის დაწყება

პროგრამის შესრულების მიმდინარეობის მართვის საშუალებების გამოყენებით შესაძლებელია მისი შესრულება ბიჯურ რეჟიმში ან მხოლოდ იმ ადგილამდე, სადაც კურსორი დგას. ზოგადად განისაზღვრება გაჩერების წერტილები, რომლებთან მიღწევის შემდეგ პროგრამა ჩერდება.

C:\avr_project\lesson_one\lesson_one.asm

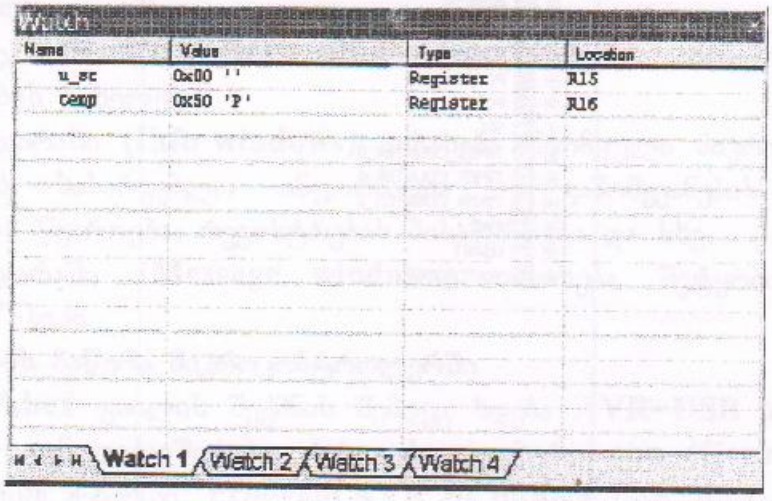
```
.equ  Fclc        = 8000000         ;
.equ  pause       = 256-103         ;
.equ  pause_1     = 256-150
;*****************************************************************************
.CSEG
  rjmp  RESET                       ;Reset Handle                 0
  rjmp  EXT_INT0                    ;IRQ0 Handle                  1
  reti                              ;IRQ1 Handle                  2
  reti                              ;timer2 comp Handle           3
  rjmp  TIM2_OVF                    ;timer2 overflow Handle       4
  reti                              ;timer1 capture Handle        5
  rjmp  TIM1_CMPA                   ;timer1 compA match  Handle   6
  reti                              ;timer1 compB match  Handle   7
  reti                              ;timer1 overflow Handle       8
  rjmp  TIM0_OVF                    ;Timer0 Overflow Handle       9
  rjmp  SPI                         ;spi_stc Handle               a
  rjmp  UART_RX                     ;UART RXc Handle              b
  rjmp  UART_UDRE                   ;UART udre Handle             c
  reti                              ;UART TXc Handle              d
  reti                              ;ADC  Handle                  e
  reti                              ;EEPROM  Handle               f
  reti                              ;An Comp  Handle              10
  reti                              ;twi  Handle                  11
  reti                              ;spm  rdi  Handle             12
```

ნახ. 3.10. სიმულაციის პროცესის მიმდინარეობა

"**AVR Studio**"-ს პაკეტში შედის მძლავრი საშუალებები, რომლებიც გამოიყენება მიკროკონტროლერის სიმულაციის დროს შიგა რეგისტრების და შეტანა/გამოტანის პორტების მდგომარეობის დათვალიერებისა და რედაქტირებისათვის.

გამართვას აადვილებს **AVR Studio**-ში არსებული შემდეგი ქვეფანჯრები:

1. დასათვალიერებელი ქვეფანჯარა (**Watch windows**) გვიჩვენებს მოცემული ცვლადების მნიშვნელობებს და მისამართებს (ნახ. 4.11). შესაძლებელია ცვლადების ხელით ან მენიუს საშუალებით ჩამატება. ცვლადების მნიშვნელობები შეიძლება შევცვალოთ ან დავამატოთ პროგრამის შესრულების შეჩერების პერიოდში.

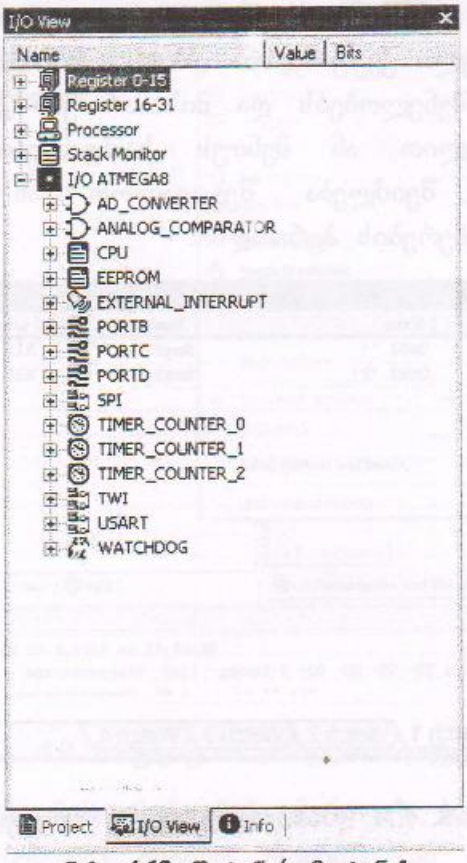


ნახ. 4.11. დასათვალიერებელი ქვეფანჯარა

2. რეგისტრების ქვეფანჯარა (**Register windows**) გვიჩვენებს რეგისტრების მდგომარეობას. ალნიშნული რეგისტრები შეიძლება შეიცვალოს პროგრამის შესრულების შეჩერების შემდეგ. ქვეფანჯარაში აისახება რეგისტრების შიგთავსი (ნახ. 4.12).

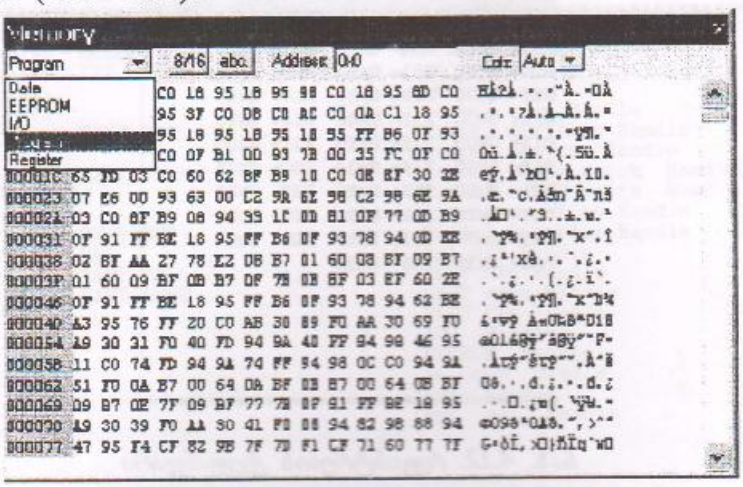
ნახ. 4.12. რეგისტრების ქვეფანჯარა

3. შეტანა/გამოტანის რეგისტრების ქვეფანჯარა (**I/O windows**) გვიჩვენებს მდგომარეობის რეგისტრის, ტაიმერების, **EEPROM**-ის რეგისტრების, შეტანა/გამოტანის პორტების შიგთავსს და ა.შ (ნახ. 4.13).



ნახ. 4.13. შეტანა/გამოტანის რეგისტრების ქვეფანჯარა

4. მეხსიერების ქვეფანჯარა (**Memory windows**) გვიჩვენებს პროგრამების მეხსიერების, მონაცემთა მეხსიერების, შეტანა/გამოტანის პორტების შიგთავსს და ენერგოდამოუკიდებელი მუდმივი მეხსიერების (**EEPROOM**) შიგთავსს (ნახ. 4.14).



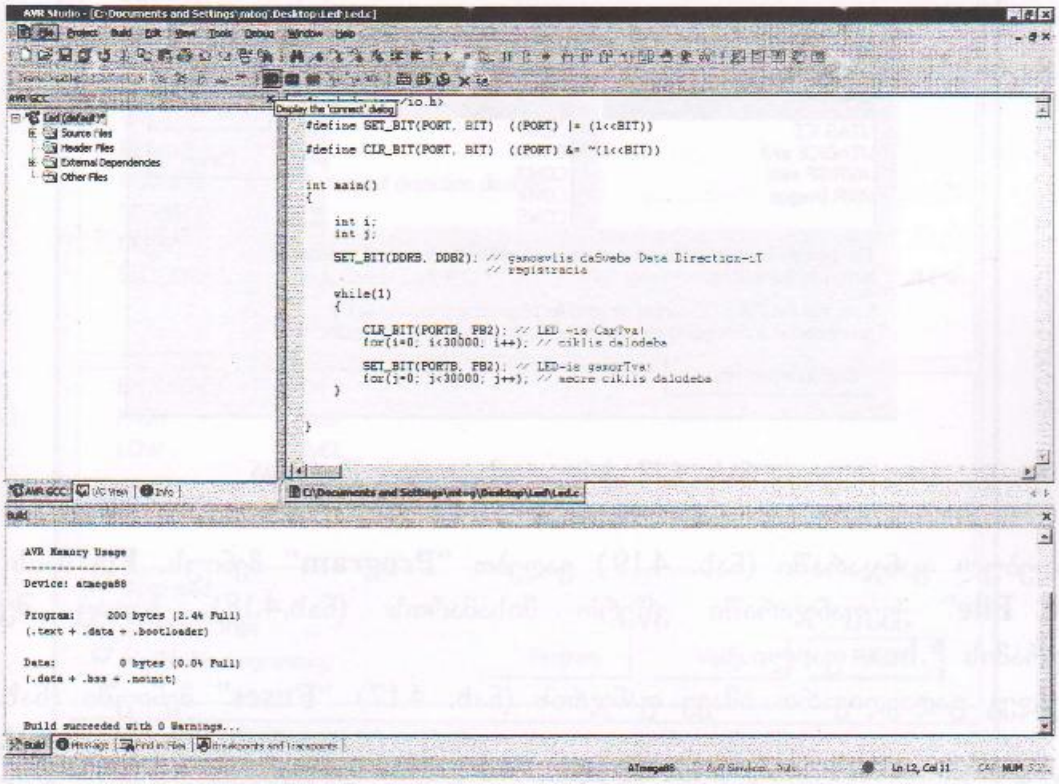
ნახ. 4.14. მეხსიერებების ქვეფანჯარა

მეხსიერების უჯრედები აისახება სხვადასხვა ფორმატში: 16-ით, 10-ით ან 2-ით და ***.hex** ფორმატში, ან **ASCII**-ის სიმბოლოებით. მეხსიერების შიგთავსი, როგორც რეგისტრების შიგთავსი, შეიძლება შეიცვალოს სურვილისამებრ პროგრამის შეჩერების შემთხვევაში.

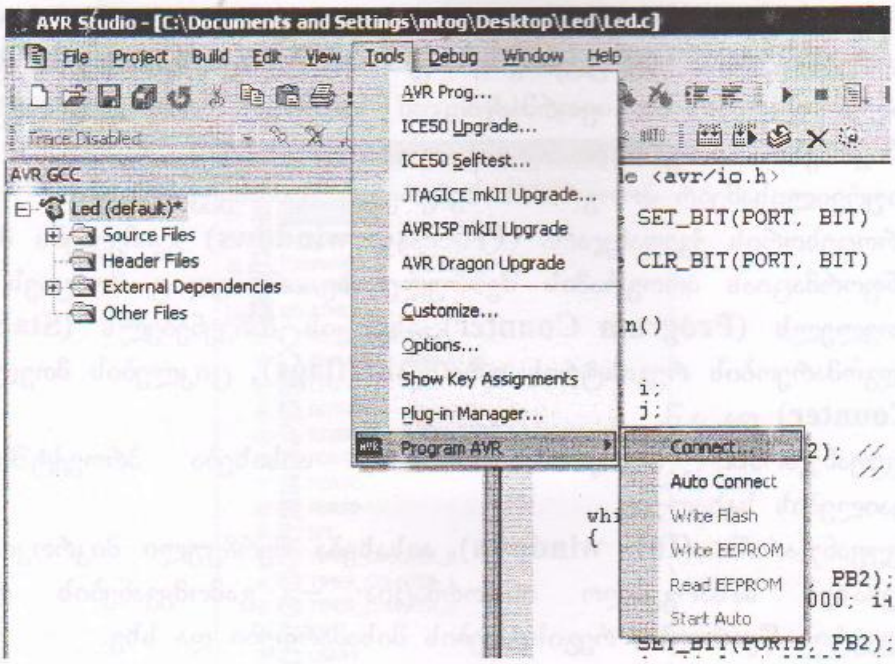
5. პროცესორის ქვეფანჯარა (**Processor windows**) გვიჩვენებს მნიშვნელოვან ინფორმაციას პროგრამის შესრულებაზე, რომელიც მოიცავს ბრძანებების მთვლელს (**Program Counter**), სტეკის მაჩვენებელს (**Stack Pointer**), მდგომარეობის რეგისტრის დროშებს (**flags**), ციკლების მთვლელს (**Cycle Counter**) და ა.შ.
6. ქვეფანჯარაში (**Project windows**) აისახება პროექტში შემავალი ფაილების სახელები.
7. ქვეფანჯარაში (**Info windows**) აისახება შერჩეული მიკროკონტროლერის შესახებ სასარგებლო ინფორმაცია – გამომყვანების დანიშნულება, წყვეტის წყაროები, რეგისტრების მისამართები და სხვ.
8. ქვეფანჯარაში (**Message windows**) აისახება შეტყობინებები **"AVR Studio"**-სგან.

4.1.4. hex-ფაილის ჩაწერა მიკროკონტროლერში

გამართული ***.hex** ფაილის შექმნის შემდეგ ხდება **AVR–USB** პროგრამატორის **"AVR Studio"**-სთან დაკავშირება, რისთვისაც ვაჭერთ **con (Connect...)** ღილაკს (ნახ.4.15) ან **Tools** მენიუში: **Program AVR > Connect** (ნახ. 4.16).

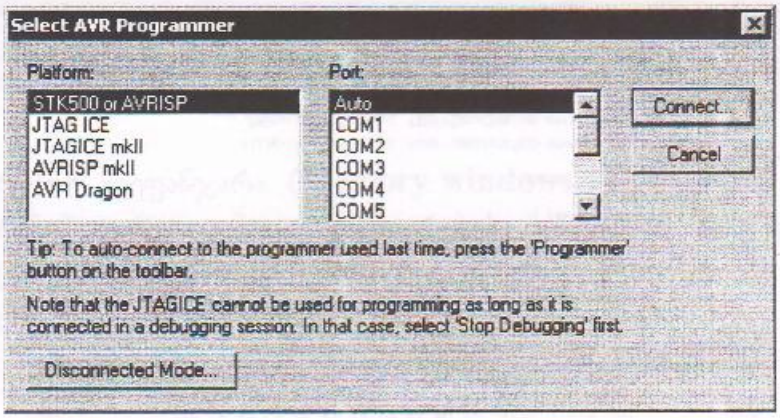


ნახ. 4.15. პროგრამატორთან დაკავშირების ბრძანება (I)



ნახ. 4.16. პროგრამატორთან დაკავშირების ბრძანება (II)

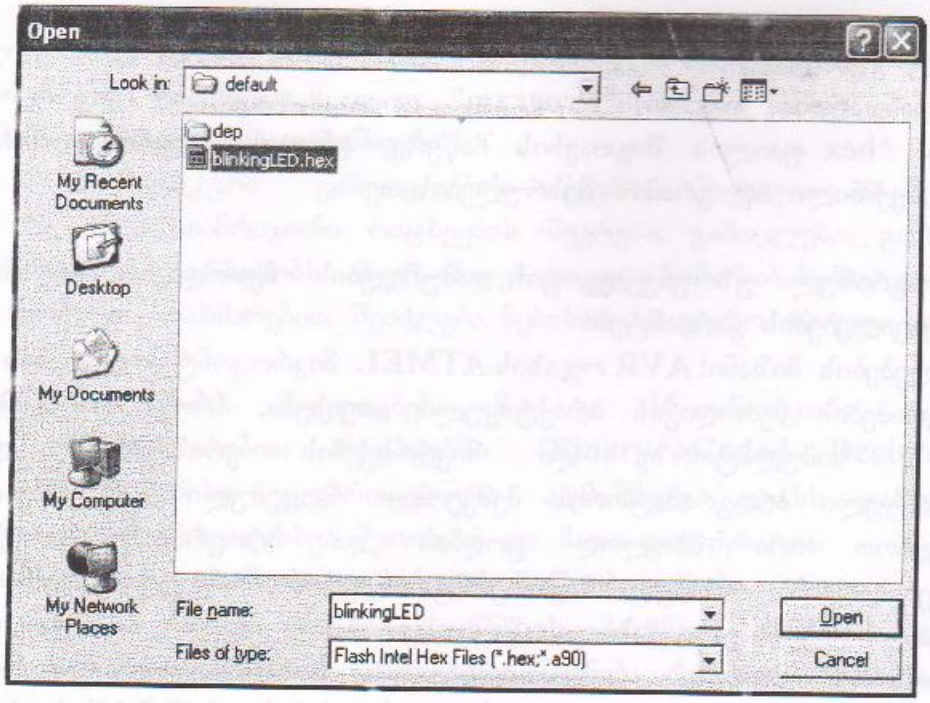
ამის შედეგად გახსნილი ფანჯრის (ნახ. 4.17) **Platform**-ის ქვეფანჯარაში მოვნიშნავთ "**STK500 or AVRISP**"-ს, ხოლო **PORT**-ის ქვეფანჯარაში – „**Auto**"-ს და ვაჭერთ „**Connect**" ღილაკს.



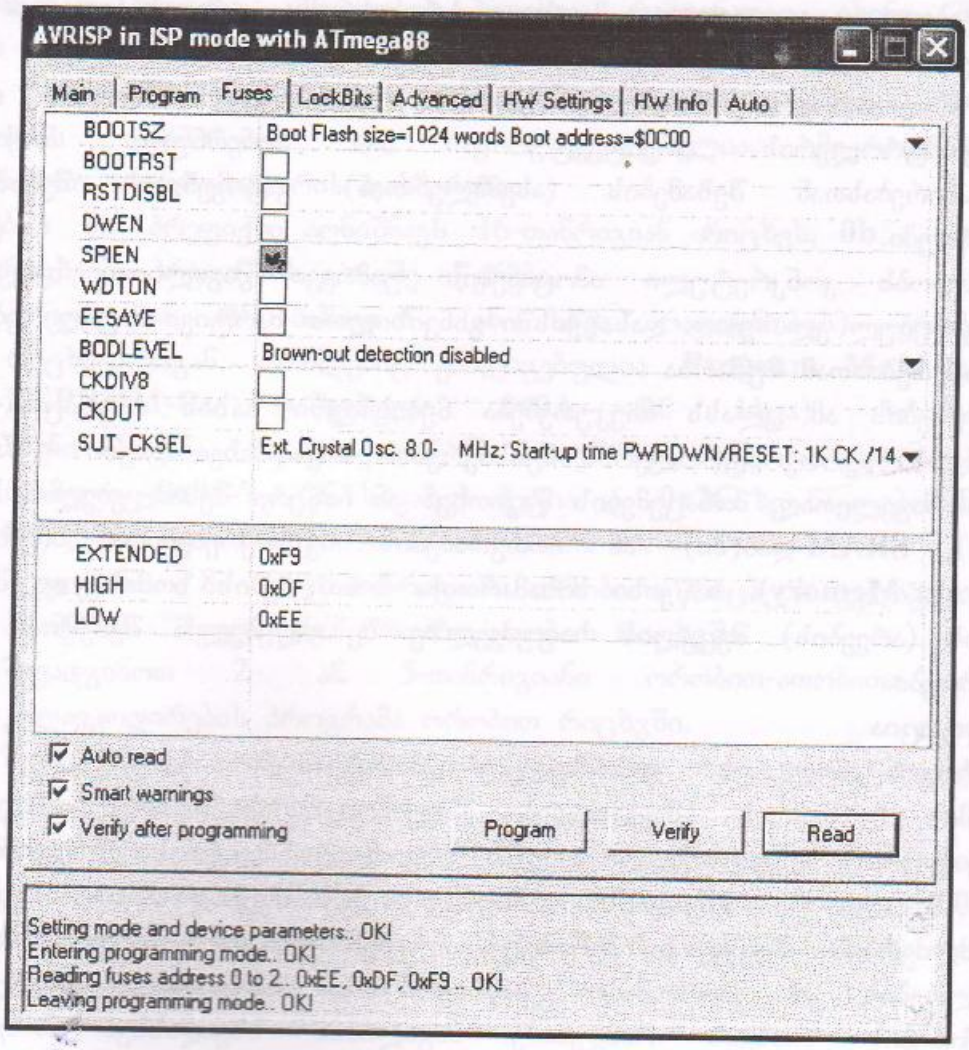
ნახ. 4.17. პროგრამატორის შერჩევა

მიღებულ ფანჯარაში (ნახ. 4.19) ვაღებთ "**Program**" მენიუს, **Flash**-ის "**Input HEX File**" ქვეფანჯარაში ვწერთ მისამართს (ნახ.4.18), სადაც შევინახეთ პროგრამის ***.hex** ფაილი.

შემდეგ გადავდივართ იმავე ფანჯრის (ნახ. 4.17) "**Fuses**" მენიუში (ნახ. 4.19). თუ პარამეტრები განსაზღვრული არ არის (როგორც ნახაზზეა ნაჩვენები), განვსაზღვრავთ და თანამიმდევრობით ვაჭერთ ღილაკებს: **Read, Verify** და **Program.**



ნახ. 4.18. პროექტის ფაილების მისამართი



ნახ.4.19. მიკროკონტროლერის მუშაობის რეჟიმის დაყენება

შემდეგ ვუბრუნდებით იმავე ფანჯრის (ნახ. 4.19) "**Program**" მენიუს და **Flash**-ის ქვეფანჯარაში ვაჭერთ "**Program**" ღილაკს, შედეგად მიკროკონტროლერში დაიწყება ***hex** ფაილის შიგთავსის ჩაწერა. პროცესის დამთავრების შემდეგ **TS** მზადაა შექმნილი პროგრამის შესრულებისათვის.

4.2. პროგრამული უზრუნველყოფის დამუშავების მეთოდიკის ანალიზი

4.2.1. დავალებების ვარიანტები

დავალებების მიზანი: **AVR** ოჯახის **ATMEL** მიკროკონტროლერების ასემბლერზე დაპროგრამების მეთოდიკის პრაქტიკული ათვისება, პროგრამის გამართვა **AVR Studio**-ს სიმულატორზე და მიკროსქემის კრისტალის დაპროგრამება სტანდარტული პროგრამატორის მეშვეობით. მოცემული მაგალითები შეგვიძლია გამოვიყენოთ არა მხოლოდ ცოდნის და პროგრამირების უნარ-ჩვევების შემოწმებისათვის, არამედ მიკროკონტროლერის არქიტექტურის თვალსაჩინო ახსნისათვის თეორიული მასალის წაკითხვისას.

მიკროკონტროლერის არქიტექტურის, აპარატული და პროგრამული საშუალებების, ინტერფეისების ძირითადი ტიპების, ბრძანებების სისტემისა და მიკროკონტროლერის სიმულატორის თეორიული მასალის შესწავლის შემდეგ შესაძლებელია გადავიდეთ მიკროკონტროლერის გამოყენებით პრაქტიკული ამოცანების რეალიზებაზე.

სწორედ პროგრამების შედგენა და მათი ახსნის უნარ-ჩვევების დაუფლებაა მიკროკონტროლერის არქიტექტურისა და აღნიშნული არქიტექტურის პროგრამირებასთან შეხამების (ასემბლერთან) გამოყენების შესაძლებლობის კრიტერიუმი.

არსებობს კონკრეტული ამოცანების უამრავი მაგალითი, დავყოთ ისინი განზოგადებულ კლასებად. დასაწყისში განვიხილოთ მარტივი მაგალითები.

1. მეხსიერებასთან მუშაობა

ამოცანების ამ კლასს მიეკუთვნება ნებისმიერი სახის ოპერაცია, რომელიც სრულდება მიკროკონტროლერის მეხსიერების სხვადასხვა არეში - წაკითხვა და ჩაწერა. მაგალითად, მონაცემების წაკითხვა ან ჩაწერა შესაძლებელია **EEPROM**-დან(ში), **SRAM**-დან(ში) ან პროგრამების მეხსიერებებიდან (მეხსიერებაში) **(Program Memory).** საწყისი მისამართის, მონაცემების ზომისა და მეხსიერების ტიპების (არეების) შეცვლის საფუძველზე შესაძლებელია შეიქმნას ამოცანების მაგალითები.

2. ინდიკაცია

მიკროკონტროლერის გამომსვლელ პორტებთან ინდიკაციის სხვადასხვა სისტემის ჩართვისას (შუქდიოდები, შუქინდიკაციის სისტემები და თხევად-კრისტალური ინდიკაციის დისპლეები) შეიძლება შეიქმნას ამოცანათა ვრცელი კლასი. მაგალითების სხვადასხვა ვარიანტი აიგება შუქინდიკაციის რიგითობისა და ხანგრძლივობის შეცვლის შემთხვევაში. შესაძლებელია მოვიყვანოთ მოძრავი სიმბოლოების ან მორბენალი სტრიქონების, სხვადასხვა სიმბოლოსა და სტრიქონების ასახვის რამდენიმე მაგალითი. სხვადასხვა ხანგრძლივობის

ინდიკაციის რეალიზაცია ხორციელდება ტაიმერით, დაყოვნების ციკლების საშუალებით ან სხვა ხერხებით.

3. მონაცემების გამოთვლა და დამუშავება

არსებობს მათემატიკური გამოთვლების ბევრი მაგალითი. შეგვიძლია ვაწარმოოთ: 8- ან 16-თანრიგიანი რიცხვების შეკრება, გამოკლება, გამრავლება, გაყოფა და შედარება. არსებობს ასევე მათემატიკური ან საბაზისო ლოგიკური ფუნქციები, რომელთა დახმარებით შეიძლება ნებისმიერი უფრო რთული ფუნქციის აგება.

ამ ამოცანების კლასს მიეკუთვნება 8- და 16-თანრიგიანი რიცხვების "შეფუთულ" ორობით-ათობით რიცხვებში **(Binary Coded Decimal-BCD)** გარდაქმნა და შესაბამისი უკუპროცედურა. აღნიშნულ კლასს განეკუთვნება რიცხვების სორტირება სხვადასხვა მეთოდით და მათი ფილტრაცია.

4. გარე პარამეტრების გაზომვის და მართვის მოწყობილობები

აქ შედის: დენის, ძაბვის, სიხშირის, წნევის, ბრუნვის სიჩქარის და სხვა. პარამეტრების გაზომვის ამოცანებს ასევე მიეკუთვნება რეალური დროის საათის შექმნა, კალკულატორის, კოდური საკეტის, ძრავას მართვის, ტემპერატურის რეგულირების და სხვ.

ეს იმდენად რთული ამოცანების კლასია, რამდენადაც ისინი შეიძლება შეიცავდეს ზოგიერთ ზემოჩამოთვლილ კლასს (1-3).

ყველა ვარიანტში აუცილებელია შევქმნათ და გადავაწყოთ პროგრამა სიმულატორის დახმარებით და თუ აუცილებელია, მიკროკონტროლერი დავაპროგრამოთ პროგრამატორის დახმარებით.

ამოცანებში **0x** პრეფიქსი აღნიშნავს 16-თანრიგიან რიცხვს, **0b** - ორობითს. ათობითი რიცხვები პრეფიქსის გარეშე იწერება.

1. მოცემულია **A** მასივი 8 (ან სხვა რაოდენობის) ერთბაიტიანი რიცხვებისაგან, რომლებიც განლაგებულია **Program Memory**-ში ან **EEPROM**-ში დაწყებული **0x0A** (ან სხვა) მისამართიდან. ჩვენი ამოცანაა **A** მასივიდან **B** მასივში გადავწეროთ ყველა რიცხვი, რომელიც **0x05**-ზე (ან სხვაზე) მეტია და **0x2C**-ზე ნაკლები. **B** მასივი შეიძლება იყოს **EEPROM**-ში ან **SRAM**-ში.
2. შევადგინოთ 8- ან 16-თანრიგიანი ორობითი რიცხვის კვლავკოდირების პროგრამა "შეფუთულ" ორობით-ათობით რიცხვში.
3. შევადგინოთ 2- ან 5-თანრიგიანი ორობით-ათობითი რიცხვის კვლავკოდირების პროგრამა ორობით რიცხვში.
4. შევადგინოთ ორი 2-თანრიგიანი ათობითი რიცხვების შეკრების ან გამოკლების პროგრამა.
5. მოცემულია **A** მასივი 8 (ან სხვა რაოდენობის) ერთბაიტიანი ან ორბაიტიანი კოდებისაგან. დავადგინოთ მასივი შეიცავს თუ არა **0xAC** ან **0xAFBC** კოდებს. თუ შეიცავს, მაშინ **Rn** რეგისტრში შევიტანოთ ასეთი კოდების რაოდენობა.

6. მოცემულია **A** მასივი 10 (ან სხვა) ერთბაიტიანი რიცხვებისაგან. გადავწეროთ რიცხვები მასივში ისე, რომ დავალაგოთ ზრდადობის ან კლებადობის მიხედვით.
7. მოცემულია **A** მასივი 10 (ან სხვა) ერთბაიტიანი რიცხვებისაგან. გადავწეროთ **B** მასივში რიცხვები, რომლებიც შეიცავენ ერთიანების (ან ნულების) ლუწ ან კენტ რაოდენობას.
8. დავშიფროთ **A** მასივში შემავალი 10 ასოითი გამოსახულების ან ციფრების კოდები შემდეგი წესით: ა) ციკლური ძვრა მარცხნივ 5 (ან სხვა) თანრიგზე და ყოველი ბიტის ინვერსია; ბ) დავამატოთ კონსტანტა 2 (ან სხვა) და შევასრულოთ ციკლური ძვრა მარჯვნივ 3 თანრიგით.
9. მოცემულია **A** მასივი 5 ერთბაიტიანი რიცხვებისაგან. განვსაზღვროთ შეიცავს თუ არა მასივი **0x1F** (ან სხვა) რიცხვს. თუ შეიცავს, მაშინ შევასრულოთ ოპერაცია "ლოგიკური და" (ან სხვა) **A** მასივის ყველა რიცხვზე.
10. გადავწეროთ **A** მასივიდან 10 რვათანრიგიანი (ან სხვა) რიცხვი მეხსიერებაში **C** მისამართზე განლაგებულ სტეკში და შევასრულოთ მათი შეჯამების (ან სხვა) ოპერაცია.
11. მოცემულია **A** მასივი 10 ერთბაიტიანი რიცხვებისაგან, რომელიც შეიცავს **0xAA** რიცხვს. შევასრულოთ ოპერაცია "ლოგიკური და" (ან სხვა) ყველა იმ რიცხვზე, რომელიც იმყოფება **0xAA**-მდე და ოპერაცია "ლოგიკური ან" (ან სხვა) ყველა იმ რიცხვზე, რომელიც იმყოფება **0xAA**-ას შემდეგ.
12. დავწეროთ ლოგიკური ფუნქციისათვის **V=X*(NOT.Y*Z+NOT.W)** გამოთვლის პროგრამა, სადაც ნიშანი (+) "ლოგიკურ ან"-ს აღნიშნავს, ნიშანი (*) – "ლოგიკურ და"-ს, **X,Y,Z,W** – ლოგიკურ ცვლადებს, რომლებიც ინახება **EEPROM** მეხსიერების 3–0 თანრიგების უჯრედებში **0x00** მისამართზე.
13. დავწეროთ ორობით რიცხვების შეკრების ან გამოკლების პროგრამა ორი 8- თანრიგიანი (ან სხვა) ნიშნით ან მის გარეშე.
14. მოცემულია **A** მასივი 10 ერთბაიტიანი რიცხვებისაგან. გადავწეროთ **A** მასივიდან **B** მასივში ყველა ის რიცხვი, რომელიც მეტია (ან ნაკლები) **0x20**-ზე და ჩავწეროთ **Rn** რეგისტრში (ან გამოვიტანოთ ინდიკატორზე).
15. განვსაზღვროთ ლუწი ან კენტი ერთიანების (ან ნულების) რაოდენობა 8 ორობით კოდებში ჩაწერილ **A** მასივში და ინდიკატორზე გამოვიტანოთ შეტყობინება "ლ" ან "კ" (ან მათი რიცხვების რაოდენობა).
16. მოცემულია **A** მასივი 10 ერთბაიტიანი რიცხვებისაგან. გადავწეროთ **B** მასივში ყველა ის რიცხვი, რომელიც მდებარეობს **0x0F-0xF0** დიაპაზონში. ინდიკატორზე გამოვიტანოთ ასეთი რიცხვების რაოდენობა.

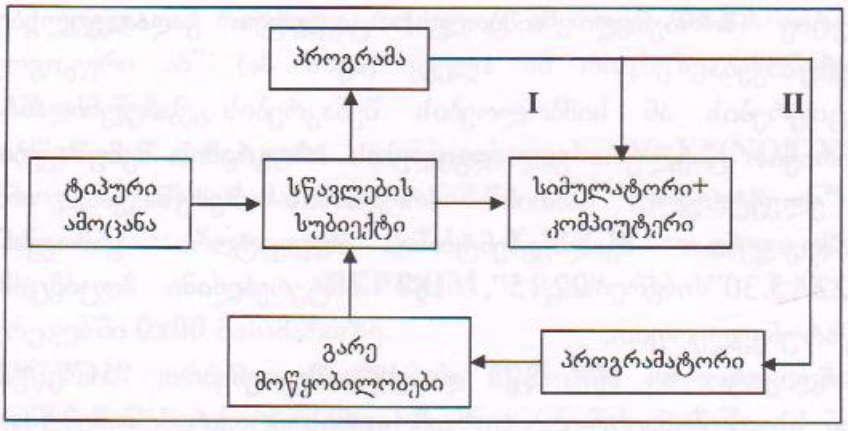
17. მოცემულია **A** მასივი 8 ერთბაიტიანი რიცხვებისაგან, რომელიც შეიცავს **0xA1** რიცხვს. გავიგოთ **A** მასივის ყველა რიცხვის ჯამი, რომლებიც აღნიშნულ რიცხვამდეა. შედეგი გამოვიტანოთ ინდიკატორზე.
18. **A** მასივში ინახება ავადმყოფების ტემპერატურის 8 მაჩვენებელი. შევამოწმოთ არის თუ არა მათ შორის 37°-ზე მაღალი და 36°-ზე დაბალი ტემპერატურა. ინდიკატორზე გამოვიტანოთ ორი შესაბამისი რიცხვი ერთი პოზიციის გამოტოვებით.
19. გადამწყობი მოწყობილობის ინდიკატორზე დავამუშაოთ 1-დან - 8-მდე მიმდევრობითი ციფრების გამოყვანის პროგრამა ისე, რომ "შეიქმნას" 12345678 რიგი.
20. შემუშავდეს პროგრამა **SREG** პროცესორის მდგომარეობის რეგისტრის ყველა ბიტის ინდიკატორზე გამოყვანის შემდეგ ფორმატში: ყველა დროშას (ბიტს) – ინდიკატორის ერთი თანრიგი.
21. შემუშავდეს ერთ შვიდსეგმენტიან ინდიკატორზე ყველა ციფრის ერთწამიანი ინტერვალით მიმდევრობითი გამოყვანის პროგრამა.
22. შემუშავდეს დისპლეის ერთი ან რამდენიმე თანრიგის 3-წამიანი (ან სხვა) ინტერვალით ციმციმის პროგრამა.
23. დისპლეის ჩართვის შემდეგ ინდიკატორების თანამიმდევრული გათიშვის პროგრამის შემუშავება.
24. მეხსიერების უჯრედში შენახული ერთი სიმბოლოს ინდიკატორის გასწვრივ (მრობენალი სიმბოლო) ციკლური გადაადგილების პროგრამის შემუშავება.
25. ციფრების ან სიმბოლოების შენაკრების მარჯვნიდან მარცხნივ ან პირიქით ციკლური გადაადგილების პროგრამის შემუშავება.
26. "ელექტრული საათის" პროგრამის შემუშავება. ინდიკაციისათვის გამოვიყენოთ შვიდსეგმენტიანი ინდიკატორი. გამოყვანის ფორმატი "22.15.30" ან "22.15", **1წმ** ინტერვალით მოციმციმე წერტილის უზრუნველყოფით.
27. ინდიკატორზე **1წმ**, **2წმ** და **4წმ** შეყოვნებით, **"1C"**, **"2C"** და **"4C"** (ან სხვა) მონაცვლეობითი გამოყვანის პროგრამის შემუშავება.
28. მატრიცული კლავიატურის (3X3-ზე) გამოყენებით და შუქდიოდით შემუშავდეს პროგრამა და მოწყობილობა, რომელიც შუქდიოდის ციმციმს იმ სიხშირით უზრუნველყოფს, რომელ ციფრსაც ვაჭერთ კლავიატურაზე (1-დან 9-ის ჩათვლით).
29. მატრიცული კლავიატურის 4X4-ზე და ინდიკაციის სისტემის გამოყენებით დამუშავდეს კოდური საკეტის პროგრამა და მოწყობილობა, რომელსაც აქვს 4-თანრიგიანი (ან სხვა) ციფრული კოდი.
30. მატრიცული კლავიატურის 4X4-ზე და შვიდი შვიდსეგმენტიანი ინდიკატორის გამოყენებით შემუშავდეს პროგრამა და კალკულატორის მოწყობილობა, რომელიც უზრუნველყოფს 16-თანრიგიანი რიცხვების შეკრებას, გამოკლებას, გამრავლებასა და გაყოფას.

31. ხმოვანი რხევების გენერატორისა და შუქინდიკაციის სისტემის რამდენიმე თანრიგის გამოყენებით შემუშავდეს პროგრამა და მოწყობილობა, რომელიც გაზომავს სიხშირეს 20ჰც - 20კჰც დიაპაზონში.

4.2.2. დავალების შესრულების ძირითადი პრინციპები

დავალების შესრულებისათვის სტუდენტს წარმოდგენა უნდა ჰქონდეს **AVR** მიკროკონტროლერების არქიტექტურასა და ბრძანებების სისტემაზე. დავალების შესრულების დროს უნდა დავიცვათ შემდეგი წესები:

1. გავარკვიოთ რაშია დავალების არსი და მიზანი;
2. დავალების მოთხოვნებიდან გამომდინარე, ავირჩიოთ შესაბამისი მიკროკონტროლერი;
3. შევადგინოთ დავალების აღწერისათვის აუცილებელი რეგისტრების დაწვრილებითი (ბიტების დონეზე) აღწერა არსებული შეტანა/გამოტანის რეგისტრების ცხრილიდან გამომდინარე;
4. განვსაზღვროთ წყვეტის აუცილებელი ვექტორები და შევადგინოთ მათი დამმუშავებლების შესაბამისი ალგორითმები;
5. შევადგინოთ ძირითადი პროგრამის ალგორითმი, მიკროკონტროლერის თავისებურებების გათვალისწინებით (სტეკი და სხვა);
6. ასემბლერის აუცილებელი ბრძანებების არჩევის შემდეგ შევუდგეთ პროგრამის დაწერას.



ნახ. 4.20. სწავლების პროცესის სტრუქტურული სქემა

4.20 ნახაზზე გამოსახულია სწავლების პროცესის სტრუქტურული სქემა, რომელიც შეიცავს:

1. მიკროკონტროლერის გაჭოლვის პროგრამას;
2. სიმულატორისა და კომპიუტერის მეშვეობით, პროგრამის მართებულობის კონტროლს (უკუკავშირის პირველი (I) მარყუჟი);
3. მიკროკონტროლერის გაჭოლვას და ინდიკაციის სისტემასთან დაკავშირების საშუალებას (გარე მოწყობილობებთან), ამოცანის გადაწყვეტის მართებულობის საბოლოო კონტროლისათვის (უკუკავშირის მეორე (II) მარყუჟი).

განვიხილოთ ინდიკაციის მუშაობის ამოცანა

ამოცანის შესრულებისათვის გამოვიყენოთ **TS** გამომცდელი მოწყობილობა და AVR Studio, C-Compiler, WinAVR-20080610 პროგრამული უზრუნველყოფა.

შესაქმნელი კვანძის ფუნქციებია: ყვითელი LED **TS**-ზე ციმციმებს 2წმ ინტერვალით (1წმ – ჩართვა, 1წმ – გამორთვა).

ამ ალგორითმის რეალიზაციის პროგრამის ტექსტის ლისტინგი კომენტარებით მოცემულია ქვემოთ.

```
// დეკლარაცია ==========================================

// კვარცის სიხშირის განსაზღვრა

#ifndef F_CPU                       // არჩევით განსაზღვრა
#define F_CPU 18432000UL            // TS-ის კვარცის სიხშირე – 18,432 მჰც
#endif

// თავსართი ფაილები

#include <avr/io.h>                 // I/O კონფიგურაცია
#include <util/delay.h>             // Delay-ს (მოცდის დრო) განსაზღვრა
#include "lcd_lib_ge.h"             // ფუნქციური ბიბლიოთეკა LCD-Display-სათვის

// მუდმივები

#define ON_TIME  10     // "ჩართვის დრო" ინკრემენტი 100 მწმ
#define OFF_TIME 10     // "გამორთვის დრო" ინკრემენტი 100 მწმ

// მაკროსები

#define SET_BIT(PORT, BIT)  ((PORT) |= (1<<BIT))     // ბიტის დასმა Port SET-ზე
#define CLR_BIT(PORT, BIT)  ((PORT) &= ~(1<<BIT))    // ბიტის დასმა Port RESET-ზე

// ფუნქციური პროტოტიპები

void initDisplay(void);             // Display-ს და საწყისი გამოსახულების ინიციალიზაცია

// მთავარი პროგრამა ====================================

int main()                          // მთავარი პროგრამის დასაწყისი
{
    int i;                          // ლოკალური 16-ბიტიანი რიცხვითი სიდიდის განსაზღვრა
    initDisplay();                  // Display-ს ინიციალიზაცია
    SET_BIT(DDRB, DDB2);            // Port B, Pin 2 (LED3) გამოსასვლელზე ჩართვა
    while(1)                        // უსასრულო ციკლის დასაწყისი
    {
        CLR_BIT(PORTB,PB2);         // Port B, Pin 2 LOW-ზე: LED ჩართვა
        for(i=0; i<ON_TIME; i++)    // მთვლელი ციკლი
```

```
        {
            _delay_ms(100);          // მოცდის დრო: ON_TIME * 100 ms
        }

        SET_BIT(PORTB, PB2);         // Port B, Pin 2 HIGH-ზე: LED გამორთვა
        for(i=0; i<OFF_TIME; i++)    // მთვლელი ციკლი
        {
            _delay_ms(100);          // მოცდის დრო: OFF_TIME * 100 ms
        }
    }                                // უსასრულო ციკლის დასასრული
}                                    // მთავარი პროგრამის დასასრული
// ფუნქციები ==========================================================

// Display-ს გამოსახულების ინიციალიზაცია
void initDisplay()                   // ფუნქციის დასაწყისი
{
    lcd_init();                      // ინიციალიზაციის თანამიმდევრობა
                                        lcd_lib_ge.h-დან
    lcd_gotoxy(0,0);                 // კურსორი: I სტრიქონის I სიმბოლოზე
    lcd_putstr("- Experiment 1 -");  // 16-სიმბოლოიანი ტექსტის შექმნა
    lcd_gotoxy(1,0);                 // კურსორი: II სტრიქონის I სიმბოლოზე
    lcd_putstr("Blinking the LED");  // 16-სიმბოლოიანი ტექსტის შექმნა
}                                    // ფუნქციის დასასრული
```

ამგვარად, დისპლეიზე გამოსახული იქნება: [**"- Experiment 1 -"** | **"Blinking the LED"**].

LCD-Display-ს სამართავი პროგრამის ფაილის (lcd_lib_ge.h) კომენტარებით მოცემულია ქვემოთ.

```
ბიბლიოთეკა TS-ის LCD-Display-სათვის
// დეკლარაცია ==========================================================
// კვარცის სიხშირის განსაზღვრა
#ifndef F_CPU                        // არჩევით განსაზღვრა
#define F_CPU 18432000UL             // TS-ის კვარცის სიხშირის სიდიდე – 18.432 MHz
#endif
// თავსართი ფაილები
#include <avr/io.h>                  // I/O კონფიგურაცია
#include <avr/interrupt.h>           // გლობალური წყვეტის განსაზღვრა (sei() და cli())
#include <stdint.h>                  // ტიპების განსაზღვრა (int, char, ...)
#include <stdbool.h>                 // 1-ბიტიანი (bool) სიდიდეების ბიბლიოთეკა
#include <util/delay.h>              // Delay-ს მოცდის დროის განსაზღვრა
// მუდმივები
#define CLEAR_DISPLAY1               // კოდი LCD-სთვის: Display-ზე გამოსახულების წაშლა
// Port-Bits
#define E 7                          // Enable-სიგნალი Display-სთვის: Port D, Bit 7
#define RS 4                         // Register შერჩევა Display-სთვის: Port D, Bit 4
// მაკროსები
#define P_DATA      PORTC            // Port C არის მონაცემების პორტი LCD-სთვის
```

```
#define P_STEUER   PORTD     // Port D არის მართვის პორტი LCD-სთვის
// ფუნქციების აღწერა
void lcd_enable (void);
void lcd_write (unsigned char byte);
void lcd_init (void);
void lcd_putc (unsigned char character );
void lcd_putstr (char *string);
void lcd_gotoxy (unsigned char line, unsigned char pos);
void lcd_clearDisplay(void);
void lcd_clearline (unsigned char line);
void lcd_displayMessage(char *string, unsigned char, unsigned char);
// LCD-ს მართვადი ფუნქციები ==========================================
// ------------------------------------------------------------------
// LCD_ENABLE:  უშვებს Enable-სიგნალის მაღალ (HIGH) იმპულს
// ------------------------------------------------------------------
void lcd_enable (void)
{
      P_STEUER |=(1<<E);          // E=1       1+-+
      P_STEUER &=~(1<<E);         // E=0       0--+ +---
}
// ------------------------------------------------------------------
// LCD_WRITE: წერს ბაიტს 4-ბიტიანი მოდუსის სახით LCD-ში
// ------------------------------------------------------------------
void lcd_write (unsigned char byte)     // 8-ბიტიანი გამოსასვლელი სიდიდე "byte"-ში
{
      unsigned char temp;           // "temp" ლოკალური სიდიდის განსაზღვრა
      temp = byte;                  // გამოსასვლელი სიდიდის დამახსოვრება
      P_DATA &= 0xF0;               // ქვედა 4 ბიტი მონაცემების პორტზე წაშლა
      P_DATA |= (temp>>4);          // 4 ბიტი გამოსასვლელის მარჯვნივ გადაწევა
                                    // მონაცემების ზედა ნახევარბაიტის (ნიბლის) LCD პორტზე
                                       გადაწევა
      lcd_enable();                 // Enable იმპულსის შექმნა მონაცემების შესანახად
                                    // მოცდის დრო ნიბლებს შორის საჭირო არ არის
      byte &=  0b0;                 // ქვედა ნიბლის გამოსასვლელი სიდიდეების გაწმენდა
      P_DATA &= 0xF0;               // ქვედა 4 ბიტი მონაცემების პორტზე წაშლა
      P_DATA |= byte;               // მონაცემების ქვედა ნიბლის LCD პორტზე დაწერა
      lcd_enable();                 // Enable იმპულსის შექმნა მონაცემების შესანახად
}
// ------------------------------------------------------------------
// LCD_INIT:LCD-ს ინიციალიზაცია
// ------------------------------------------------------------------
void lcd_init (void)
{
      cli();                              // გლობალური წყვეტის დეაქტივაცია
      DDRC = 0x0f;                        // Port C, Bit 0..3 (LCD-მონაცემები) გამოსასვლელზე
      DDRD |= ((1<<E) | (1<<RS));         // Port D, Bit 4 (RS: LCD რეგისტრის შერჩევა)
                                          // და Bit 7 (E: LCD Enable) გამოსასვლელზე
      PORTC = 0x0f;                       // Port C, Bit0..3 (LCD- მონაცემები) SET
                                          // მართვადი სიგნალი – LOW-ზე
      P_STEUER &=~((1<<E) | (1<<RS));     // E და RS 0-ზე დასმა
```

```
    _delay_ms (20);              // Display-ს 4-ბიტიან მოდუსში ინიციალიზაცია
     _DATA &=0xf0;
    P_DATA |=0x03;
    lcd_enable();
    _delay_ms (10);
    lcd_enable ();
    _delay_ms (10);
    lcd_enable ();
    P_DATA &=1;
    _delay_ms (2);
    lcd_enable();
    _delay_us (50);
                                 // 2 სტრიქონი, 5x7 Pixel
    lcd_write (0x28);
    _delay_us (40);
                                 // Display-ს ჩართვა
    lcd_write (0x0C);
    _delay_us (50);
                                 // "increment mode" კურსორის დაყენება
    lcd_write (0x02);
    _delay_us (50);
                                 // Display-ზე გამოსახულების წაშლა
                                    (ფუნქციონირებს 2-ჯერ გამოძახებისას)
    lcd_write (0x01);
    _delay_ms (3);
    lcd_write (0x01);
    _delay_ms (3);
    sei();                       // გლობალური წყვეტის აქტივიზაცია
}
// ---------------------------------------------------------------------
// LCD_PUTC:      წერს სიმბოლოს განსაზღვრულ ადგილას
// ---------------------------------------------------------------------
void lcd_putc (unsigned char character) // 8-ბიტიანი გამოსასვლელი სიდიდე "character"-ში
{
    P_STEUER |= (1<<RS);     // Register Select HIGH-ზე: "მონაცემების გაშვება"
    lcd_write (character);   // სიმბოლოების გაშვება
    _delay_us (55);          // შეყოვნება 55 წმ შიგა დამუშავებისათვის
}
// ---------------------------------------------------------------------
// LCD_PUTSTR:   წერს String-ს LCD-ზე
// ---------------------------------------------------------------------
void lcd_putstr (char *string)
{
    while (*string)              // დამამთავრებელი სიმბოლო 0x00-მდე
    {
        lcd_putc (*string);      // სიმბოლოს String-დან აღება და გაშვება
        string ++;               // პოინტერის შემდეგ სიმბოლოზე დასმა
    }
}
```

```
// ---------------------------------------------------------------
// LCD_GOTOXY:  კურსორის გადაწევა მოცემულ პოზიციაზე
// ---------------------------------------------------------------
void lcd_gotoxy (unsigned char line, unsigned char pos)
{
      P_STEUER &= ~(1<<RS);          // Register Select LOW-ზე: "კონტროლის გაშვება"
      lcd_write((1<<7)+0x40*line+pos); // კურსორი DDRAM-მისამართზე
      _delay_us (50);                // შეყოვნება 50 ნწმ შიგა დამუშავებისათვის

}
// ---------------------------------------------------------------
// LCD_CLEARDISPLAY: Display-ზე გამოსახულების მთლიანად წაშლა
// ---------------------------------------------------------------
void lcd_clearDisplay(void)
{
      P_STEUER &= ~(1<<RS);          // Register Select LOW-ზე: "კონტროლის გაშვება"
      lcd_write(CLEAR_DISPLAY);      // Display-ზე წაშლის ბრძანების გაშვება
      _delay_ms (2);                 // შეყოვნება 2 მწმ შიგა დამუშავებისათვის
}
// ---------------------------------------------------------------
// LCD_CLEARLINE:     Display-ზე ერთ-ერთი სტრიქონის წაშლა
// ---------------------------------------------------------------
void lcd_clearline (unsigned char line)
{
      switch(line)
      {
            case 0: lcd_gotoxy(0,0); // პირველი სტრიქონის დამისამართების დასაწყისი
            lcd_putstr ("    ");     // "sicarielis" გაშვება
            break;
            case 1: lcd_gotoxy(1,0); // მეორე სტრიქონის დამისამართების დასაწყისი
            lcd_putstr ("      ");   // "sicarielis" გაშვება
            break;
      }
}
// ---------------------------------------------------------------
// LCD_DISPLAYMESSAGE:    Display-ზე String სიდიდის გამოსახვა
// ---------------------------------------------------------------
void lcd_displayMessage(char* mess, unsigned char line, unsigned char pos)
{
      lcd_gotoxy(line,pos);          // კურსორის პოზიციონირება
      lcd_putstr(mess);              // String-ის გაშვება
}
```

მეორე მაგალითად განვიხილოთ ხმოვანი სიგნალის სინთეზის ამოცანა.

ამოცანის შესრულებისათვის გამოვიყენოთ **TS** გამომცდელი მოწყობილობა და AVR Studio, C-Compiler, WinAVR-20080610 პროგრამული უზრუნველყოფა.

შესაქმნელი კვანძის ფუნქციებია: **TS**-ზე დაბალი ხმის მასინთეზირებელ მოწყობილობაზე (buzzer) ხდება განგაშის ხმის გაშვება. ტონალობის მინიმალური და მაქსიმალური მნიშვნელობებისას სიხშირე მცირე ხნით სტაბილური ხდება. სიხშირე Timer 0-ის საშუალებით (CTC-Mode-ში) იქმნება და პირდაპირ Output-Compare-Pin-ზე Toggle-Mode-ში განისაზღვრება.

გამოსახულება დისპლეიზე აისახება: [**-Experiment 2 - | Creating Sound**].

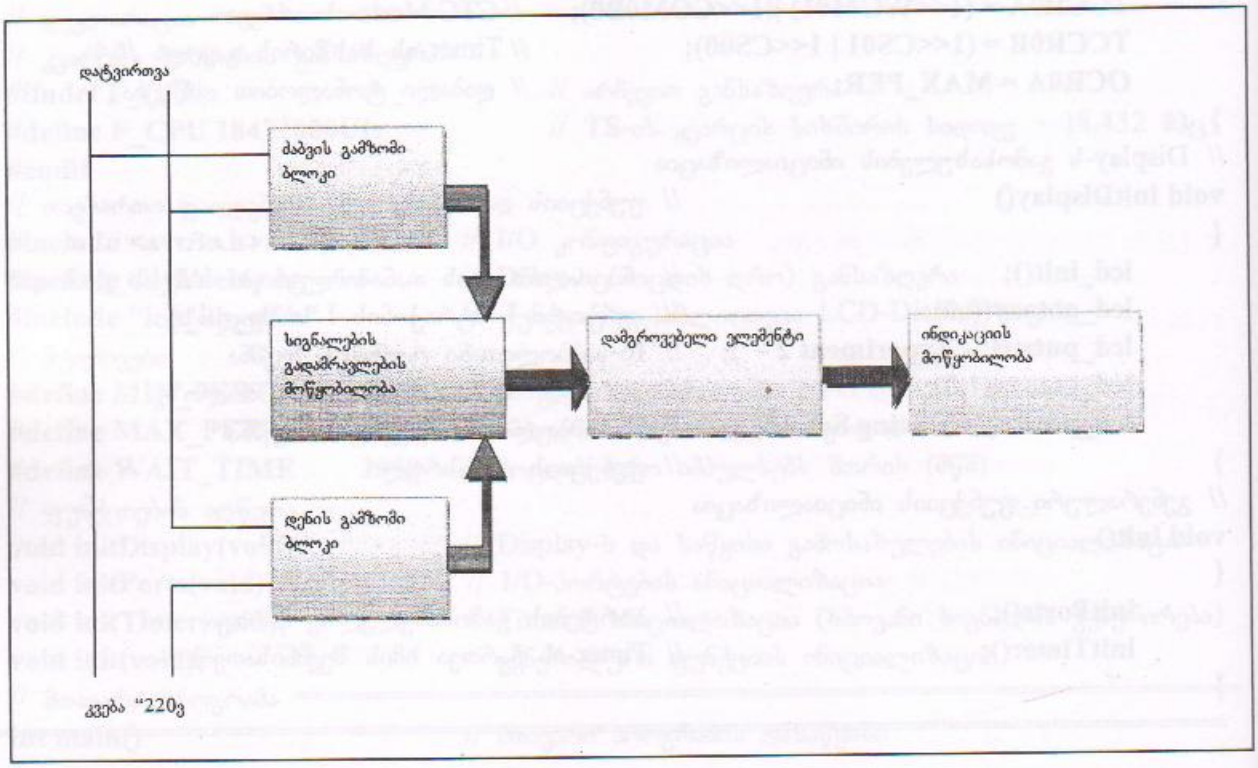
ამ ალგორითმის რეალიზაციის პროგრამის ტექსტის ლისტინგი კომენტარებით მოცემულია ქვემოთ.

```
// დეკლარაცია ==========================================================
// კვარცის სიხშირის განსაზღვრა
#ifndef F_CPU                       // არჩევით განსაზღვრა
#define F_CPU 18432000UL            // TS-ის კვარცის სიხშირის სიდიდე – 18,432 მჰც
#endif
// თავსართი ფაილები
#include <avr/io.h>                 // I/O კონფიგურაცია
#include <util/delay.h>             // Delay-ს (მოცდის დრო) განსაზღვრა
#include "lcd_lib_ge.h"             // ფუნქციური ბიბლიოთეკა LCD-Display-სთვის
// მუდმივები
#define MIN_PER      100            // იმპულსის მინიმალური პერიოდი
#define MAX_PER      255            // იმპულსის მაქსიმალური პერიოდი
#define WAIT_TIME    2000           // მოცდის დრო იმპულსებს შორის (მწმ)
// ფუნქციების აღწერა
void initDisplay(void);             // Display-ს და საწყისი გამოსახულების ინიციალიზაცია
void initPorts(void);               // I/O-პორტების ინიციალიზაცია
void initTimer(void);               // Timer 0 ინიციალიზაცია (ხმოვანი სიგნალის გენერირება)
void init(void);                    // გენერალური ფუნქციის ინიციალიზაცია
// მთავარი პროგრამა ====================================================
int main()                          // მთავარი პროგრამის დასაწყისი
{
    init();                         // პორტების და Timer 0 ინიციალიზაცია
    initDisplay();                  // Display-ს ინიციალიზაცია
    while(1)                        // უსასრულო ციკლის დასაწყისი
    {
      for (OCR0A=MAX_PER; OCR0A>MIN_PER; OCR0A--) // სიხშირის ამაღლება
      {
        _delay_ms(10);              // ბიჯი 10 მწმ
      }
        _delay_ms(WAIT_TIME);       // დაყოვნება მაღალი სიხშირისას

      for (OCR0A=MIN_PER; OCR0A<MAX_PER; OCR0A++) //სიხშირის დაკლება
      {
        _delay_ms(10);              // ბიჯი 10 მწმ
      }
```

```
        _delay_ms(WAIT_TIME);          // დაყოვნება დაბალი სიხშირისას
    }                                  // უსასრულო ციკლის დასასრული
}                                      // მთავარი პროგრამის დასასრული
// ფუნქციები ==========================================================
// I/O-პორტების ინიციალიზაცია
void initPorts()
{
    DDRB |= (1<<DDB2);                 // Port B, Pin 2 (LED3) გამოსასვლელზე ჩართვა
    DDRD |= (1<<DDD5);                 // Port D, Pin 5 (Buzzer) გამოსასვლელზე ჩართვა
}

// Timer 0-ის ინიციალიზაცია ხმოვანი სიგნალის შექმნისათვის
void initTimer()
{
    TCCR0A = (1<<WGM01) |(1<<COM0B0);  // CTC Mode-ის არჩევა
    TCCR0B = (1<<CS01 | 1<<CS00);      // Timer-ის სიხშირის გაყოფა /64
    OCR0A = MAX_PER;                   // დაბალი ტონალობით დაწყება
}
// Display-ს გამოსახულების ინიციალიზაცია
void initDisplay()                     // ფუნქციის დასაწყისი
{
    lcd_init();                        // ინიციალიზაციის თანამიმდევრობა lcd_lib_ge.h-დან
    lcd_gotoxy(0,0);                   // კურსორი I სტრიქონის I სიმბოლოზე
    lcd_putstr("- Experiment 2 -");    // 16-სიმბოლოიანი ტექსტის შექმნა
    lcd_gotoxy(1,0);                   // კურსორი II სტრიქონის I სიმბოლოზე
    lcd_putstr("Creating Sound");      // 16-სიმბოლოიანი ტექსტის შექმნა
}                                      // ფუნქციის დასასრული
// გენერალური ფუნქციის ინიციალიზაცია
void init()
{
    initPorts();                       // პორტების გამოსასვლელზე ჩართვა
    initTimer();                       // Timer-ის ჩართვა ხმის შექმნისათვის
```

ფუნქციონალობის გაფართოება. მიმწოდებლებს სჭირდებათ ოპერატიული წვდომა მოცემულ მომენტში გაყიდული ელექტროენერგიის რაოდენობის შესახებ და დისტანციური კონტროლი. მომხმარებელი კი დაინტერესებულია დახარჯული ელექტროენერგიის ეკონომიაში სხვადასხვა ტარიფის (დღის, ღამის) გამოყენების ხარჯზე და საფასურის გადახდის მოსახერხებელ საშუალებებში. ერთ-ერთი ასეთი ვარიანტია გადასახადის ელექტრონული სისტემების გამოყენება და მრიცხველებში კარტრიდერების ჩაშენება.

თანამედროვე მექანიკურ მრიცხველებს არ შეუძლია დასმული ამოცანის გადაჭრა, ფასი/ხარისხი ეფექტური თანაფარდობის შენარჩუნების პირობით. ამიტომ საჭირო ხდება ახალი მიდგომა ელექტროენერგიის ხარჯების აღრიცხვისა და გადახდის პრობლემისადმი.



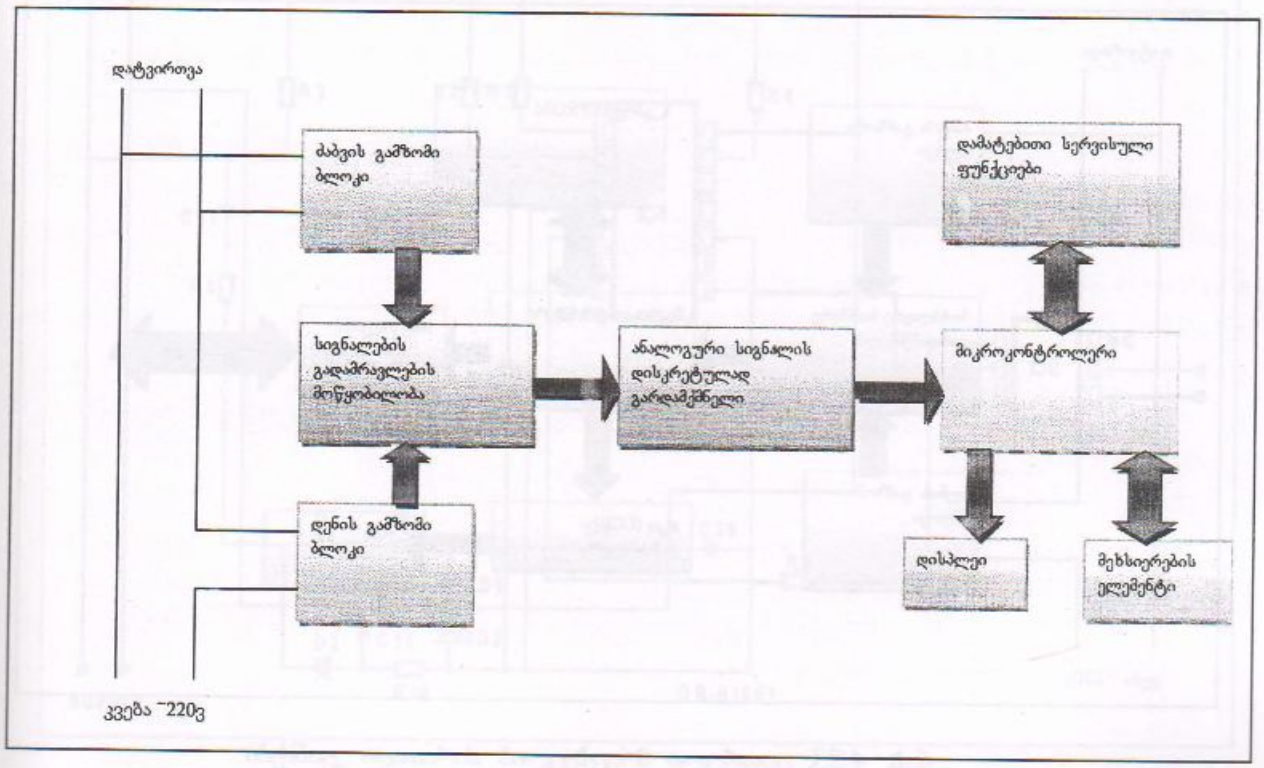
ნახ. 4.21. ელექტროენერგიის ელექტრომექანიკური მრიცხველის ბლოკ-სქემა

დღეს მიკროელექტრონიკის სწრაფი განვითარების და ელექტრონული კომპონენტების ფასების კლების პირობებში ციფრული მოწყობილობები ნელ-ნელა ცვლის ანალოგურ კონკურენტებს. ეს, პირველ რიგში, გამოწვეულია მიკროკონტროლერების ფართო არჩევანით და მათი დაბალი ღირებულებით. მიკროკონტროლერების ბაზაზე აგებული მართვისა და კონტროლის სისტემების ერთ-ერთი მთავარი უპირატესობაა მოქნილობა და მრავალფუნქციურობა, რომელიც მიიღწევა არა აპარატული, არამედ პროგრამული საშუალებების ხარჯზე, რაც არ საჭიროებს დამატებით მატერიალურ დანახარჯებს.

დენის მრიცხველების მიკროკონტროლელურ მართვაზე გადაყვანა გარკვეულ უპირატესობებს იძლევა. ესაა აღრიცხვის სიზუსტისა და საიმედოობის ზრდა და

მრავალფუნქციურობა, რაც მიიღწევა მცირე აპარატული დანახარჯების საშუალებით. მოთხოვნილებიდან გამომდინარე, თანამედროვე ციფრულ მრიცხველებს უნდა შეეძლოს ოპერატიულად გადასცეს საჭირო მონაცემები ენერგომომწოდებელი კომპანიების სადისპეტჩეროს, კავშირის სხვადასხვა არხის მეშვეობით, რათა შესაძლებელი გახდეს ელექტროენერგიის მოხმარების ოპერატიული კონტროლი და შემდგომი ეკონომიკური გათვლები. იმისათვის, რომ დროის გარკვეულ პერიოდში დათვლილ იქნეს მოხმარებული ელექტროენერგიის რაოდენობა, საჭიროა აქტიური სიმძლავრის დროის მომენტში ინტეგრირება.

სინუსოიდური სიგნალისათვის დროის მოცემულ მომენტში სიმძლავრე ქსელში არსებული ძაბვის ნამრავლის ტოლია. ამ პრინციპით მუშაობს დენის ნებისმიერი მრიცხველი.



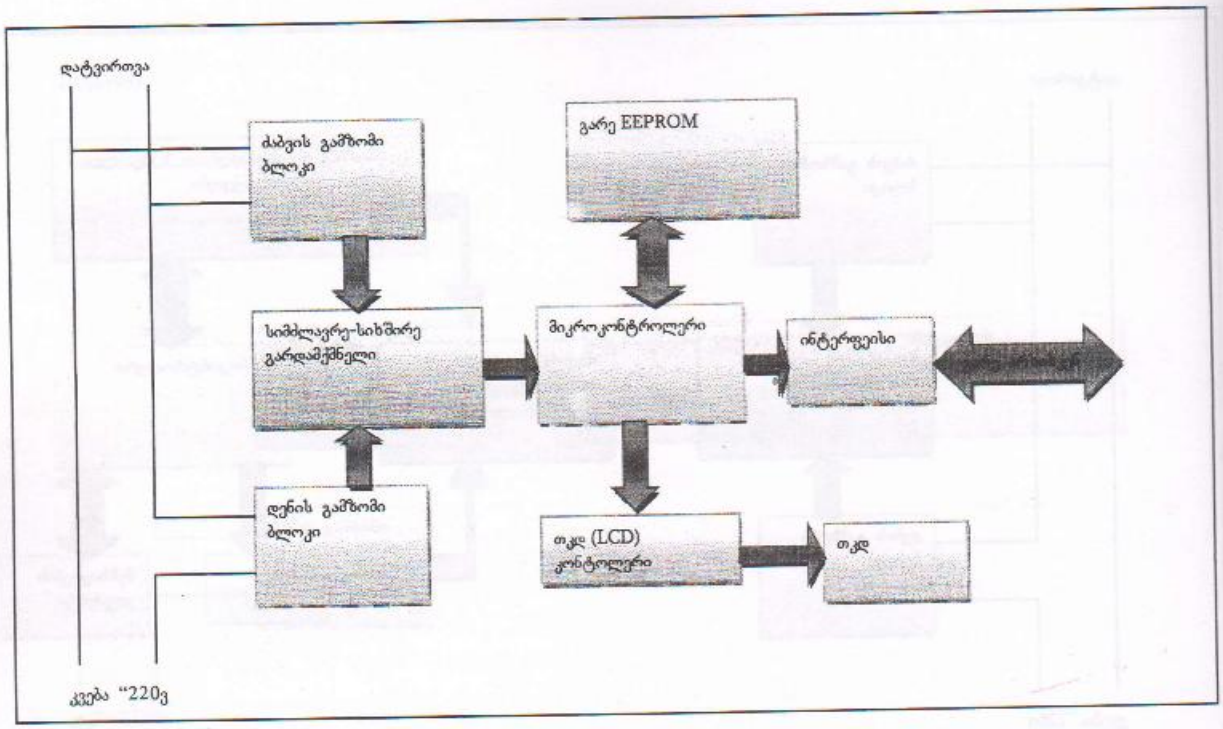
ნახ. 4.22. ელექტროენერგიის ციფრული მრიცხველის ბლოკ-სქემა

დენის ციფრული მრიცხველის (ნახ. 4.22) რეალიზაცია საჭიროებს სპეციალურ ინტეგრალურ მიკროსქემებს, რომლებსაც შეუძლია შესავალი სიგნალების გადამრავლება და შემდგომ მიღებული სიდიდის წარმოდგენა მიკროკონტროლერისათვის მოსახერხებელ ფორმაში. მაგალითად, მიკროსქემას შეუძლია გარდაქმნას აქტიური სიმძლავრე სხვადასხვა სიხშირის იმპულსთა მიმდევრობით. მიკროკონტროლერის მიერ დათვლილ იმპულსთა საერთო რაოდენობა მოხმარებული ელექტროენერგიის პირდაპირპროპორციული იქნება.

არანაკლებ მნიშვნელოვანია სხვადასხვა სერვისული ფუნქცია, როგორიცაა, მაგალითად, მრიცხველზე დისტანციური წვდომა, მოხმარებული ენერგიის რაოდენობაზე ინფორმაციის მიღება და მრავალი სხვა. ციფრული დისპლეის

არსებობა, რომელიც აგრეთვე მიკროკონტროლერის მიერ იქნება კონტროლირებადი, საშუალებას იძლევა პროგრამულად იყოს დაყენებული მონაცემთა ჩვენების სხვადასხვა რეჟიმი. ასე, მაგალითად, შესაძლებელია დისპლეიზე ინფორმაციის გამოტანა იმის შესახებ, თუ რა რაოდენობის ელექტროენერგია იყო დახარჯული ყოველი თვის განმავლობაში, სატარიფო გეგმა და ა.შ.

ციფრული მრიცხველის შემუშავების ანალიზი მოცემულია 4.23 ნახაზზე. ასეთი მრიცხველი შეიძლება აგებული იყოს **Atmel**-ის **AT89** სერიის მიკროკონტროლერისა და **Sames SA9602E** სიმძლავრის სიხშირედ გარდამქმნელი ინტეგრალური მიკროსქემის ბაზაზე. წარმოდგენილ გადაწყვეტაში რეალიზებულია ყველა ძირითადი საჭირო ფუნქცია.



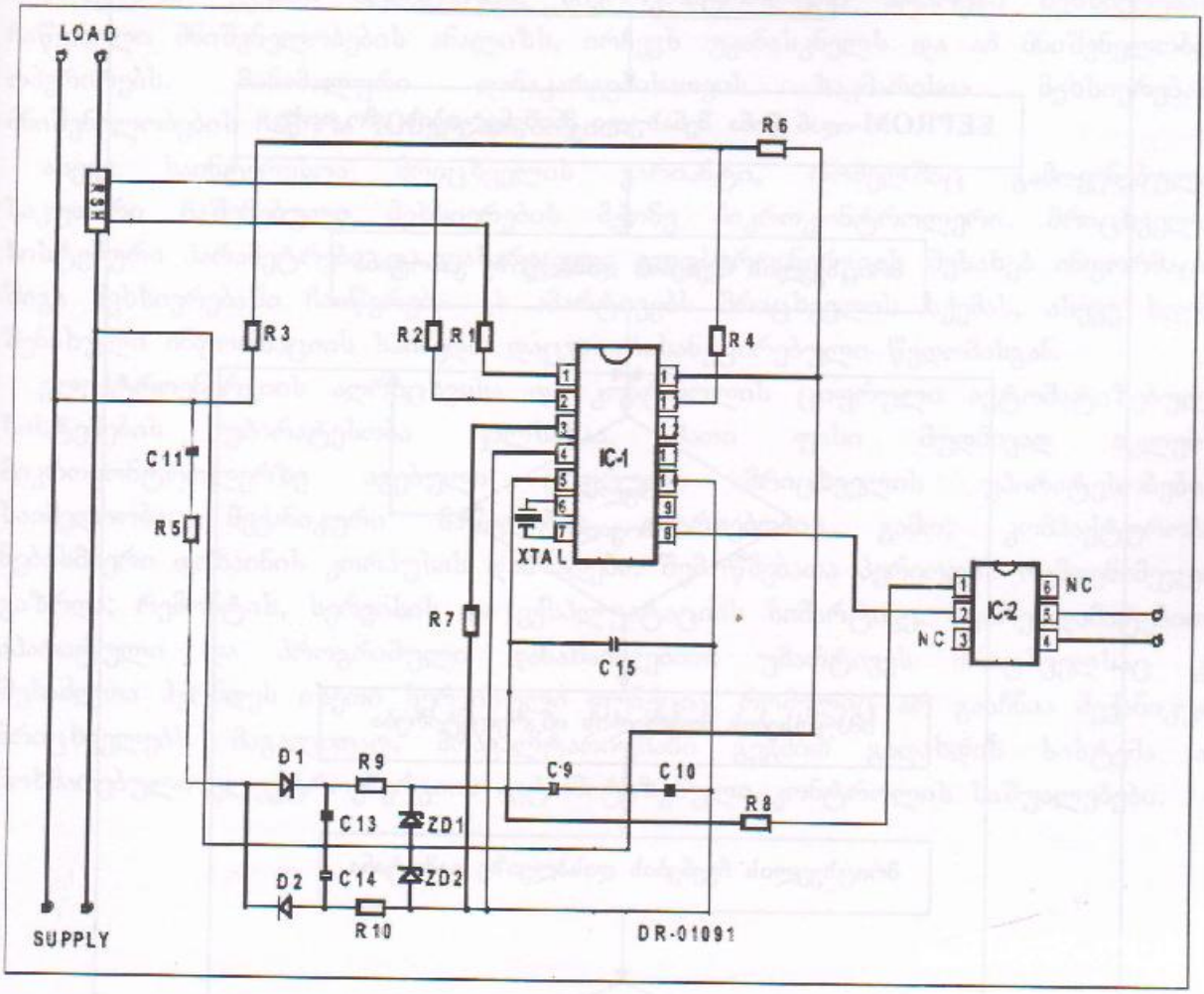
ნახ. 4.23. ციფრული მრიცხველის ძირითადი კვანძები

4.24 ნახაზზე მოცემულია გარდამქმნელი მიკროსქემის გამოყენების მაგალითი. მიკროკონტროლერი უნდა იყოს მიერთებული **IC-2** ოპტოდამაწყვილებელთან.

მრიცხველის ასეთი სტრუქტურის შემთხვევაში მოხმარებული ენერგიის აღრიცხვისათვის მიკროკონტროლერმა უნდა დაათვალოს გამზომი მიკროსქემიდან მოსული იმპულსების რაოდენობა, მოახდინოს ინფორმაციის გამოტანა დისპლეიზე და ინფორმაცია დაიცვას სხვადასხვა ავარიულ რეჟიმში. ასეთი მრიცხველი ფაქტიურად ჩვეულებრივი მექანიკური მრიცხველის ფუნქციური ანალოგია, რომლის შემდგომი განვითარება და ფუნქციური გაფართოება მარტივია.

დენის ძალისა და ძაბვის პროპორციული სიგნალები სენსორებიდან ამოიკითხება. გარდამქმნელის მიკროსქემა გადაამრავლებს შესავალ სიგნალებს და ღებულობს მყის მოხმარებულ სიმძლავრეს. შემდეგ სიგნალი გადაეწოდება

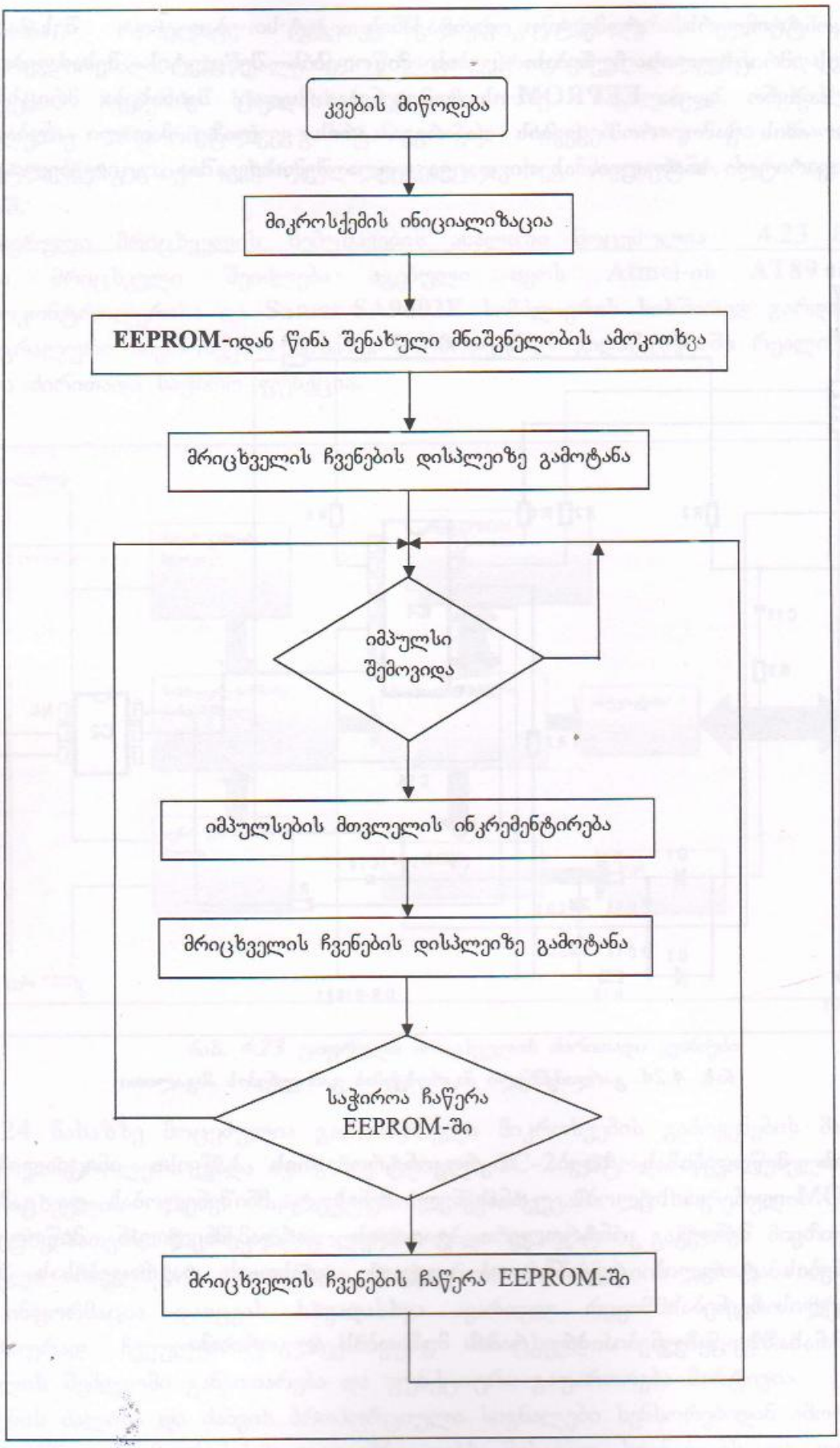
მიკროკონტროლერს, რომელიც გარდაქმნის კვტ·სთ-ებად და, შესაბამისად, შეცვლის მრიცხველის ჩვენებას. კვების მიწოდების შეწყვეტის შესაძლებლობის გამო, საჭირო ხდება **EEPROM**-ის გამოყენება, სადაც შეინახება მრიცხველის ჩვენება. იმის გამო, რომ კვების დაკარგვა არის ყველაზე მაღალი ალბათობის მქონე ავარიული სიტუაცია, ასეთი დაცვა ყველა შემთხვევაშია აუცილებელი.



ნახ. 4.24. გარდამქმნელი მიკროსქემის გამოყენების მაგალითი

კვების მიწოდებისას ხდება მიკროკონტროლერის საწყისი ინიციალიზაცია, **EEPROM**-იდან კითხულობს უკანასკნელ შენახულ მნიშვნელობას და გამოაქვს დისპლეიზე. შემდეგ კონტროლერი გადადის გარდამქმნელიდან მიწოდებული იმპულსების დათვლის რეჟიმში და ყოველი კვტ·სთ-ის დაგროვებისას ზრდის მრიცხველის ჩვენებას.

4.25 ნახაზზე ნაჩვენებია პროგრამის მუშაობის ალგორითმი.



ნახ. 4.25. პროგრამის მუშაობის ალგორითმი

EEPROM-ში დახარჯული ელექტროენერგიის მნიშვნელობის ჩაწერისას, კვების გათიშვის შემთხვევაში, ეს მნიშვნელობა შეიძლება დაიკარგოს. ამის გამო, დახარჯული ენერგიის მნიშვნელობები მეხსიერებაში ციკლურად იწერება, პროგრამულად მითითებული მრიცხველის მნიშვნელობის გარკვეული ცვლილების შემდეგ. ეს სისტემას იცავს მნიშვნელობების დაკარგვისაგან კვების გათიშვის შემთხვევაში. კვების აღდგენისას, მიკროკონტროლერი ატარებს მეხსიერებაში ჩაწერილი მნიშვნელობების ანალიზს, ირჩევს უკანასკნელს და ამ მნიშვნელობით ოპერირებს. მინიმალური დანაკარგებისათვის საკმარისია მეხსიერებაში მნიშვნელობების ჩაწერა **100ჯერ**-ით ბიჯით.

ასევე საინტერესოა მრიცხველის ვარიანტი, რომელშიც გამოყენებულია საკუთარი ჩაშენებული მეხსიერების მქონე მიკროკონტროლერი. მრიცხველის სისტემური პარამეტრები და დახარჯული ელექტროენერგიის შესახებ ინფორმაცია შიგა მეხსიერებაში ჩაიწერება. ეს ამარტივებს მრიცხველის სქემას, ასევე ხდება შენახული ინფორმაციის საიმედო დაცვა არასანქცირებული წვდომისგან.

ელექტროენერგიის აღრიცხვისა და კონტროლის ციფრული ავტომატიზებული სისტემების უპირატესობა ცალსახაა. მათი ფასი მუდმივად იკლებს. მიკროკონტროლერზე აგებული ციფრული მრიცხველის უპირატესობებია: საიმედოობა, მექანიკური ნაწილების არარსებობის გამო; კომპაქტურობა; ნებისმიერი დიზაინის კორპუსის დამზადება; შემოწმებათა პერიოდის რამდენიმეჯერ გაზრდა; რემონტის, სერვისის და ექსპლუატაციის სიმარტივე, მცირე დამატებითი აპარატული და პროგრამული დანახარჯებით. უმარტივეს მრიცხველსაც კი შესაძლოა ჰქონდეს ისეთი სერვისული ფუნქცია, რომელიც არ გააჩნია მექანიკურ მრიცხველებს. მაგალითად, მრავალტარიფიანი გეგმით გადახდის სისტემა ან მოხმარებული ელექტროენერგიის ავტომატიზებული კონტროლის საშუალებები.

დასკვნა

მიკროელექტრონიკის განვითარება და მისი ნაწარმის გამოყენება სამრეწველო სფეროში, მრავალფეროვან და სხვადასხვა სახის მოწყობილობებსა და მართვის სისტემებში დღეს ინოვაციური განვითარების ერთ-ერთი ძირითადი მიმართულებაა.

ელექტრონული ტექნიკის მოწყობილობათა ფართო ნომენკლატურაში განსაკუთრებული ადგილი უჭირავს დასაპროგრამებელი მიკროსქემების ოჯახს. მათი სწრაფი განვითრება მიკროელექტრონიკის, როგორც განვითარების კატალიზატორის, ერთგვარი ინდიკატორია თანამედროვე სამყაროში.

სულ უფრო იზრდება სამეცნიერო-ტექნიკურ მუშაკთა წრე, რომლებსაც პრაქტიკულ საქმიანობაში სჭირდება დამხსომებელი და ლოგიკური დასაპროგრამებელი მიკროსქემების გამოყენება. მათი საშუალებით მიიღწევა ახალი აპარატურის დამუშავების ვადების მკვეთრი შემცირება, შესაძლებელი ხდება უფრო მაღალი ტექნიკური მახასიათებლების მიღწევა.

არსებობს პრინციპული აუცილებლობა დაპროგრამებული მიკროსქემების გამოყენებისა ეკონომიკის თითქმის ყველა დარგში. წარმოების მოქნილი სისტემები, სხვადასხვა ტექნოლოგიური პროცესების მართვა, პერსონალური კომპიუტერები და საყოფაცხოვრებო აპარატურა, აი არასრული ჩამონათვალი იმ მიმართულებებისა, სადაც დაპროგრამებული მიკროსქემების გამოყენების გარეშე შეუძლებელია ეფექტური მუშაობა.

თანამედროვე აპარატურის განვითარების ერთ-ერთი მახასიათებელი ტენდენცია არის ინტეგრაციის დონის სწრაფი ზრდა. ასეთ პირობებში აქტუალური ხდება აპარატურის ისეთი კვანძების დამუშავება, რომლებშიც გამოყენებული იქნება ინტეგრაციის მაღალი და ზემაღალი დონის სქემები.

დაპროგრამებული ინტეგრალური მიკროსქემების ძირითადი უპირატესობა, მიკროელექტრონიკის სხვა მოწყობილობებთან შედარებით, შემდეგია: სტრუქტურის რეგულარობა, ფუნქციური დაშენებადობა, მათ საფუძველზე რეალიზებადი კომბინირებული ლოგიკის მქონე მოწყობილობების ფართო დიაპაზონი, სტრუქტურის დაპროგრამებულობა. ამასთან, მიღწეულია კრისტალზე ზემაღალი ინტეგრაციის დონე. ასეთი ინტეგრალური მიკროსქემების უპირატესობა მათ საფუძველზე ასაგები მოწყობილობების დაპროექტების ავტომატიზაციითაა განპირობებული. შექმნილი მოწყობილობების გამოყენების არეალი პრაქტიკულად შეუზღუდავია.

მიკროკონტროლერების დაპროექტებისას დაცული უნდა იყოს ბალანსი ზომასა და ფასს შორის, ერთი მხრივ, მოქნილობასა და წარმადობას შორის, მეორე მხრივ. სხვადასხვა გამოყენებისათვის ამ და სხვა პარამეტრების ოპტიმალური შეთავსება კარდინალურად განსხვავდება. ამიტომ არსებობს მიკროკონტროლერების უამრავი ტიპი, რომლებიც განსხვავდებიან საპროცესორო მოდულის არქიტექტურით, ჩაშენებული მეხსიერების ტიპით და მოცულობით, პერიფერიული მოწყობილობების ჩამონათვლით, კორპუსის ტიპით და ა.შ.

არის მრავალი ისეთი ამოცანა, სადაც მაღალი წარმადობა უმნიშვნელოა, სამაგიეროდ მნიშვნელოვანია მოწყობილობის ფასი. ამ შემთხვევაში ფართოდ გამოიყენება 8-ბიტიანი მიკროკონტროლერები. არის ასევე მაღალი მწარმოებლურობის მიკროკონტროლერებიც, მაგალითად, **DSP** ციფრული სიგნალის პროცესორები.

ფასით და ენერგომოხმარებით გამოწვეული შეზღუდვები აფერხებს მიკროკონტროლერების ტაქტური სიხშირის ზრდას. მწარმოებლები ცდილობენ მიკროკონტროლერები მაქსიმალურ სიხშირეზე ამუშაონ. თუმცა არჩევანის საშუალებასაც აძლევენ მომხმარებელს და უშვებენ ერთი და იგივე მოდელის მოდიფიკაციას, განკუთვნილს სხვადასხვა სიხშირეზე მუშაობისათვის. ბევრ მოდელში გამოიყენება სტატიკური მეხსიერება **SRAM**, რაც ამ მოდელებს შემცირებულ სიხშირეზე მუშაობის საშუალებას აძლევს კონტროლერისა და ტაქტური გენერატორის სრული გაჩერების შემთხვევაშიც კი, რადგან ეს არ იწვევს მეხსიერებაში მყოფი მონაცემების დაკარგვას. არსებობს ენერგოდაზოგვის რამდენიმე რეჟიმი, რომელთა გააქტიურებისას ითიშება პერიფერიული მოწყობილობების ნაწილი ან გამოთვლითი მოდული.

მიკროკონტროლერების მომზადება-გამოყენება შეიძლება ორ ეტაპად დაიყოს: პირველი – პროგრამირება, როცა სპეციალისტი ამუშავებს პროგრამას და "გაჭოლავს" ნახევრად გამტარ კრისტალში; მეორე – მიღებული დაპროგრამებული მიკროკონტროლერების შეთანხმება მართვის ობიექტსა და მის შემსრულებელ მოწყობილობებთან.

პირველ ეტაპზე პროგრამის გამართვას მნიშვნელოვნად აადვილებს სიმულატორი, რომელიც თვალსაჩინოდ ამოდელირებს მიკროკონტროლერის მუშაობას. მეორე ეტაპზე გამართვისათვის გამოიყენება შიგასქემური ემულატორი, რომელიც რთული მოწყობილობაა. მიღებული პროგრამის მიკროკონტროლერის კრისტალში "გაჭოლვისათვის" გამოიყენება პროგრამატორი, რომელიც აგრეთვე ემსახურება მიკროკონტროლერების პროგრამირების საფუძვლების თვალსაჩინო სწავლებას.

ლიტერატურა

1. Предко М. Руководство по микроконтроллерам. Том 1. / Пер. с англ. под ред. И. И. Шагурина и С. Б. Лужанского – М.: Постмаркет, 2001. – 416 с.
2. Предко М. Руководство по микроконтроллерам. Том 2. / Пер. с англ. под ред. И. И. Шагурина и С. Б. Лужанского – М.: Постмаркет, 2001. – 488 с.
3. Cady, Fredrick M. Microcontrollers and microcomputers: principles of software and hardware engineering. – New York – Oxford, Oxford University Press, 1997. – 252 p.
4. Вуд А. Микропроцессоры в вопросах и ответах. / Пер. с англ. под ред. Д. А. Поспелова. – М.: Энергоатомиздат, 1985. – 184 с.
5. Уильямс Г.Б. Отладка микропроцессорных систем. / Пер. с. англ. – М.: Энергоатомиздат, 1988. – 253с.
6. Угрюмов Е.П. Цифровая схемотехника. – Спб.: БВХ – Санкт-Петербург, 2000. – 528 с.
7. Алексенко А.Г., Шагурин И.И. Микросхемотехника. – М.: Радио и связь, 1990. – 496 с.
8. Бродин Б.В., Шагурин И.И. Микроконтроллеры: Справочник. – М.: ЭКОМ, 1999. – 395 с.
9. Мальцев П.П., Гарбузов Н.И., Шарапов А.П., Кнышев А.А. Программируемые логические ИМС на КМОП-структурах и их применение. – М.: Энергоатомиздат, 1998. – 158 с.
10. Соловьев В.В., Васильев А.Г. Программируемые логические интегральные схемы и их применение. – Мн.: Беларусская наука, 1998. – 270 с.
11. Bursky D. Embedded Logic and Memory Find a Home in FPGA. – Electronic Design, 1999, №14, pp. 43-56.
12. Axelson Jan USB Complete – Lakeview research, 2001.
13. John Hyde. USB Design by Example. A Practical Guide to Building I/O Devices. – Intel Press, 2001.
14. USB in a NutShell (http://www.beyondlogic.org/)
15. Art Baker, Jerry Lozano - Programming in Windows.
16. USB Specifications (http://www.usb.org).
17. Гук М. Аппаратные средства IBM PC// Энциклопедия. – СПб: Петербург, 2002.
18. Агуров П. В. Последовательные интерфейсы. Практика программирования. – СПб.: БХВ-Петербург, 2004.
19. Дадунашвили С.А. Устройство для устранения дребезга контакта. Ас. №2725093\18-21, 1979.
20. Дадунашвили С.А. Многофункциональное электронное устройство ввода информации//, Труды седьмой всесоюзной научно-техн. конференции по влагометрии. Кутаиси, 1984, с. 179-180.
21. Дадунашвили С. А., Дгебуадзе Г. В. К вопросу проектирования интеллектуальных систем на микроконтроллерах// GEN, №1, 2006, с.72-79.
22. ს. დადუნაშვილი, გ. დგებუაძე. ვირტუალური ლაბორატორიული სამუშაოები ანალოგურ ელექტრონიკაში. თბილისი, 2006. - 46 გვ.
23. Dadunashvili S. Informational Phenomenos in Embedded Systems – 21st international CODATA Conference, Conference Proceedings, Kyiv, 2008. p.218-225.
24. Дадунашвили С. А. Определение структуры многоуровневой «встроенной системы»// Proceedings of the international scientific conference "Information Technologies 2008", Tbilisi, 2008, p.252-256.

25. Дадунашвили С.А., Гвалия Т.В., Донашвили И.М. Некоторые вопросы организации встраиваемых систем// Proceedings of the second international scientific conference „Information Tecnology“, Kutaisi, 2009, p. 52-56.
26. Дадунашвили С.А., Дзагания Н.С. Принципы построения встраиваемых систем// GEN, №2, 2009, p.51-56.
27. სერგო დადუნაშვილი. ლაბორატორიული სამუშაოები მიკროპროცესორულ ტექნიკაში TS მოწყობილობის გამოყენებით. თბილისი, 2009,-59 გვ.
28. სერგო დადუნაშვილი. ჰიბრიდული ინტეგრალური მიკროსქემების გაანგარიშება და დაპროექტება. თბილისი: სტუ, 1995, -56 გვ.
29. სერგო დადუნაშვილი. საინჟინრო სისტემების ტექნოგენეტიკური ფორმირება // მეცნიერება და ტექნიკა, №4-5, თბილისი, 1997, გვ. 96-103.
30. Дадунашвили С.А. Микропроцессорная система для комплексной оценки параметров пищевых продуктов// Пятая всесоюзная научно-техн.конференция «Электрофизические методы обработки пищевых продуктов». Тезисы докладов. Москва, 1985,.с. 356.
31. Дадунашвили С.А. Система на микро-ЭВМ для теледоступа к удаленным информационным ресурсам// Материалы восьмой школы-семинара «Персональные компьютеры и локальные сети». Тбилиси, 1986, с. 333-335.
32. Дадунашвили С.А., Малкин Л.А. Некоторые вопросы применения локальных ИИС на базе микро-ЭВМ в аналитической технике// Всесоюз. научно-техн. конференция «Измерительные информационные системы». Тезисы докладов. Ташкент, 1987, с. 14.
33. Дадунашвили С.А., Малкин Л.А. Вопросы моделирования случайных сигналов в структуре микро-ЭВМ –многомерный ЦАП. //Всесоюз. научно-техн. конференция. «Измерит. информ.системы». Тез. докладов. Ташкент, 1987, с. 43.
34. Лаптев В. Цифровой измеритель температуры на базе AVR микроконтроллера и RC-цепочки// – Электронные компоненты, 2001, №2, с. 46 – 49.
35. Chang D., Mazek-Sadowska M. Dynamically Reconfigable FPGA. – JEEE Transition on Computers, 1999, №6, pp. 565 – 578.
36. Bursky D. Advanced CPLD Architectures Challenge FPGA, Gas. – Electronic Design, 1998, №22, pp. 78 – 86.
37. Takai Y. a.o. 250 Mbytes Synchronous DRAM Using a 3-Stage-Pipeline Architecture. – JEEE. Journal of Solid-Stage Circuits. – 1994, v.29, №4, pp. 426 – 429.
38. Кулаков В. Программирование на аппаратном уровне: специальный справочник. 2-е изд. – СПб.: Питер, 2003.
39. Солдатов В. П. Программирование драйверов Windows. – М.: Бином, 2004.
40. Рихтер Д. Windows для профессионалов. – СПб.: Питер, Microsoft Press, 1999.
41. USB232 board EB039-00-1Technical datasheet
42. E-blocksTM USB232 board Document code: EB039-30-1
43. FT232BL USB UART (USB - Serial) I.C. FTDIChip
44. DS232BL Version 1.8 © Future Technology Devices Intl. Ltd. 2005
45. Описание Си-компилятора фирмы IAR. (www.iar.com).
46. Справочная система из состава AVR32 Studio.
47. Материалы европейского дистрибьюторского семинара. Январь 2007 года. http://atmel.argussoft.ru/seminars/
48. Королев Н., Шабынин А. Архитектура AVR: развитие вширь и вглубь// Компоненты и технологии, Часть 1, 2007, № 2.
49. Горелков Р. AVR-микроконтроллеры picoPower компании ATMEL с ультранизким потреблением// Компоненты и технологии, 2006, № 12.

50. Производители микроконтроллеров и их поставщики на российском рынке// Электронные компоненты, 2006, № 7.

51. Материалы практического семинара "День ATMEL на выставке "ЭкспоЭлектроника 2007".
52. Микроконтроллеры и DSP. – Электронные компоненты, 2005, № 7.
53. Atmel Specifications (http://www.atmel.com).
54. www.atmel.com/products/avr32/.
55. http://www.atmel.com/dyn/resources/prod_documents/doc32002.pdf.
56. http://www.atmel.ru/ - AVR მიკროკონტროლერების აღწერა.
57. http://www.dontronics.com/atmel.html - AVR-ის პროგრამული პროექტები.
58. http://www.gaw.ru/, http://www.cec-mc.ru.
59. http://trush.da.ru/, http://avr.da.ru/.
60. http://www.ln.com.ua/~real/avreal, http://www.chat.ru/~avreal.
61. http://trush.pp.ru/avr/ - AVR-ის სასარგებლო ბმულები.

რედაქტორი ლ. მამალაძე

გადაეცა წარმოებას 17.01.2011. ხელმოწერილია დასაბეჭდად 24.03.2011. ქაღალდის ზომა 60X84 1/8. პირობითი ნაბეჭდი თაბახი 9,5. ტირაჟი 100 ეგზ.

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, თბილისი, კოსტავას 77


ISBN 978-9941-14-915-3
9789941149153