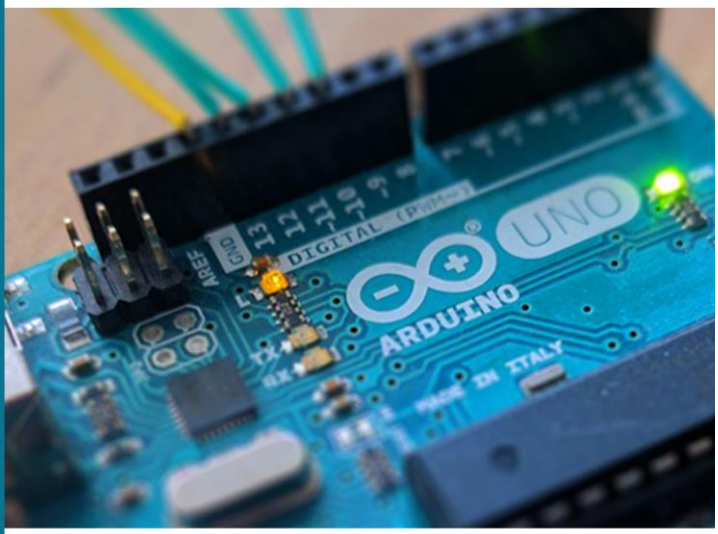
ზაზა ცაბაძე, თეა თოდუა

არდუინო



პრაქტიკული სახელმძღვანელო

დამწყებთათვის

ზაზა ტაბატაძე, თეა თოდუა

არდუინო

პრაქტიკული სახელმძღვანელო დამწყებთათვის

თბილისი
2019

უაკ 681. 3

ნაშრომი წარმოადგენს პრაქტიკულ სახელმძღვანელოს დამწყებთათვის, რომელშიც დეტალურადაა განხილული ის საკითხები, რომელთა ცოდნაც აუცილებელია არდუინოს შესწავლის საწყის ეტაპზე. წიგნში მაქსიმალურად კომპაქტური სახითაა მოცემული ელექტრონიკის საკითხები, მარტივად და გასაგებად არის ახსნილი არდუინოს პლატფორმაზე დაპროგრამების გზით სხვადასხვა ამოცანის გადაჭრის ხერხები, განხილულია მარტივი და საშუალო სირთულის ამოცანების საინჟინრო და პროგრამული რეალიზების მეთოდები და საშუალებები.

განკუთვნილია არდუინოს შესწავლით დაინტერესებული ნებისმიერი პირისთვის, ელექტრონიკასა და დაპროგრამებაში საბაზისო ცოდნის დონით.

რეცენზენტები:

საქართველოს ტექნიკური უნივერსიტეტის
პროფესორი: ჯ. გრიგალაშვილი
საქართველოს ტექნიკური უნივერსიტეტის
პროფესორი ი. მოსაშვილი

ISBN 978-9941-8-1738-0

წინასიტყვაობა

„მთვარე სამთვარიოში ჩასულა უკვე და მესმის თუ როგორ ღუღუნებს ჩემი ბინის ელექტრომრიცხველი - ეს ინდუსტრიის ცისფერი მტრედი.“

კ. გამსახურდია. „მთვარის მოტაცება“

ჯერ საუკუნეც კი არ გასულა ამ სტრიქონების დაწერიდან და მისი წაკითხვისას, დღეს, მეოთხე ინდუსტრიული რევოლუციის ეპოქაში, თვალწინ გაგვიელვებს ყველა ის ტექნოლოგიური მიღწევა თუ სიახლე, რომელიც თავბრუდამხვევი სისწრაფით შემოიჭრა ჩვენს ცხოვრებაში. თუ იმ პერიოდისათვის, როდესაც ეს დიდებული ნაწარმოები იწერებოდა, ინდუსტრიის ცისფერი მტრედი ელექტრომრიცხველი იყო, დღეს უამრავი მსგავსი მოწყობილობის დასახელება შეგვიძლია. რაც არ უნდა გასაკვირი იყოს, მათ გარკვეულ ნაწილს ინტელექტიც კი გააჩნია, ადამიანის ინტელექტს მიმსგავსებული და მისდარად შექმნილი.

ჩვენი წიგნი სწორედ ერთ ასეთ საინტერესო მოწყობილობაზე მოგითხრობთ, ამ მოწყობილობას არდუინო ჰქვია და ის იტალიაში შეიქმნა. არდუინო არის იაფი დაფა მიკროკონტროლერით, მისი საშუალებით დამწყებებსაც კი უამრავი საინტერესო პროექტის შექმნა შეუძლიათ. ის სწორედ მისი სიმარტივისა და ხელმისაწვდომობის გამო გახდა პოპულარული მთელ მსოფლიოში. არდუინოს შეიძლება მივუერთოთ სხვადასხვა სახის სენსორი, ნათურა,

ძრავები და სხვა მოწყობილობები, შეიძლება მარტივად დავაპროექტოთ ინტერაქტიული დისპლეი ან მობილური რობოტი. არდუინო ღია კოდის სისტემას წარმოადგენს. საინტერესოა, რომ არდუინო სტუდენტებისთვის შეიქმნა, რათა მათთვის ელექტრონული პროექტების სწრაფად შექმნის შესაძლებლობა მიეცათ. ისევე როგორც დღეს კომპიუტერის მომხმარებლებს არ სჭირდებათ ფიქრი მისი ცალკეული ნაწილების მუშაობის პრინციპებზე, მსგავსად, არდუინოს დაფაც მომხმარებლებს საშუალებას აძლევს მთელი ყურადღება პროექტების შექმნაზე გადაიტანონ.

არდუინოს შექმნამდე მიკროკონტროლერის დაპროგრამების შესწავლას მუდამ თან ახლდა გარკვეული სირთულეები. მას შემდეგ, რაც არდუინო შეიქმნა, ელექტრონიკის სფეროში მინიმალური ცოდნითაც, საინტერესო პროექტების განხორციელება გახდა შესაძლებელი. ინტერნეტში უამრავი მასალა შეგიძლიათ ნახოთ, ბავშვებიც კი როგორ მარტივად და სიამოვნებით სწავლობენ არდუინოსთან მუშაობას.

ვისთვის არის ეს წიგნი განკუთვნილი

წიგნი დაიწერა იმ იდეით, რომ ქართველი ახალგაზრდებისთვის მშობლიურ ენაზე მიგვეწოდებინა ის ცოდნა და გამოცდილება, რაც წიგნის ავტორებს ამ სფეროში გაგვაჩნია. ნაშრომი გამოქვეყნების დღიდანვე ღიად იქნება ხელმისაწვდომი ინტერნეტ სივრცეში. საყოველთაოდ ცნობილი ჭეშმარიტებაა, რომ ქვეყნისთვის სასიცოცხლოდ მნიშვნე-

ლოვანია ჭკვიანი და განათლებული ადამიანების არსებობა. ვფიქრობთ, ამით ჩვენს მოკრძალებულ წვლილს შევიტანთ ჩვენი სამშობლოს ცოტათი მაინც გაძლიერებასა და წინსვლაში. ვიმედოვნებთ, რომ ამ წიგნის გამოშვებით გარკვეულწილად ხელს შევუწყობთ ელექტროინჟინერიის დარგის პოპულარიზაციას საქართველოში.

წიგნი წარმოადგენს პრაქტიკულ სახელმძღვანელოს დამწყებთათვის. მასში დეტალურადაა განხილული ის საკითხები, რომელთა ცოდნაც აუცილებელია არდუინოს შესწავლის საწყის ეტაპზე. წიგნში მაქსიმალურად კომპაქტური სახითაა მოცემული ელექტრონიკის საკითხები, მარტივად და გასაგებად არის ახსნილი არდუინოს პლატფორმაზე დაპროგრამების გზით სხვადასხვა ამოცანის გადაჭრის ხერხები, განხილულია მარტივი და საშუალო სირთულის პროექტების საინჟინრო და პროგრამული რეალიზების მეთოდები და საშუალებები.

მადლიერება

მადლობას ვუხდით ყველა იმ ადამიანს, რომლებიც წიგნზე მუშაობის პროცესში დაგვეხმარნენ.

პირველ რიგში, დიდი მადლობა და მოკრძალება მშობლებს, რომლებიც ჩვენთვის ადამიანობის, წესიერებისა და პატიოსნების უდიდეს მაგალითს წარმოადგენენ.

მადლობა პედაგოგებს, რომელთაც უდიდესი წვლილი შეიტანეს ჩვენს პროფესიულ ჩამოყალიბებაში.

მადლობა გვინდა გადავუხადოთ ჩვენს ამერიკელ კოლეგას, კომპანია MEMX-ის დამფუძნებელსა და დირექტორს პოლ მაკვორტერს (Paul McWorther), რომლის ვიდეო ლექციების არაჩვეულებრივმა კურსმა გვიბიძგა იმისკენ, რომ ჩვენც წამოგვეწყო ეს საქმე და მსგავსი ტიპის, მაქსიმალურად მარტივად და გასაგებად დაწერილი წიგნი გამოგვეცა ქართველი საზოგადოებისთვის.

მადლობა მეგობრებს და კოლეგებს, რომლებიც მუდმივად სტიმულს გვაძლევდნენ, რომ ეს ნაშრომი ბოლომდე მიგვეყვანა. მადლობა გვინდა გადავუხადოთ საქართველოს ტექნიკური უნივერსიტეტის ინფორმატიკისა და მართვის სისტემების ფაკულტეტის კომპიუტერული ინჟინერიის დეპარტამენტის კურსდამთავრებულებს: ანდრია ხანჯალიაშვილს, გიორგი კალანდაძეს, გიორგი გაბოშვილს, შალვა ლაბაძეს და როლანდ ართმელაძეს, ელექტრონიკის ინჟინერს გიორგი კურტანიძეს, რომლებიც განსაკუთრებულ ინტერესს იჩენდნენ ჩვენი ნაშრომის მიმართ და წიგნის წერის პროცესში ხშირად გვიზიარებდნენ თავიანთ შენიშვნებსა თუ საინტერესო მოსაზრებებს.

ზაზა ტაბატაძე

თეა თოდუა

შესავალი

არდუინო წარმოადგენს ელექტრონულ კონსტრუქტორს და მოხერხებულ პლატფორმას სხვადასხვა სახის ელექტრონული მოწყობილობის შესაქმნელად. ის პოპულარულია გამოყენების ფართო არეალის, დაპროგრამების ენის სიმარტივის, ღია კოდისა და არქიტექტურის გამო. არდუინო საშუალებას აძლევს კომპიუტერს გასცდეს ვირტუალური სამყაროს საზღვრებს და გადავიდეს ფიზიკურში, მასთან ურთიერთქმედების მიზნით. არდუინოს ბაზაზე შექმნილ მოწყობილობებს შეუძლიათ მიიღონ ინფორმაცია გარემომცველ გარემოზე სხვადასხვა სენსორის მეშვეობით, ასევე შეუძლიათ სხვადასხვა სახის მმართველი ზემოქმედების განხორციელება.

არდუინოს საშუალებით შესაძლებელია საინტერესო, ინტერაქტიული პროექტების შექმნა. ეს შეიძლება იყოს დისტანციურად მართვადი რობოტი, ჭკვიანი სახლი და ა.შ.

არდუინოს პოპულარობა მისი გამოყენების სიმარტივემ და დაბალმა ფასმაც განაპირობა. მოწყობილობა მუშაობს “plug-and-play” პრინციპით, რაც ნიშნავს იმას, რომ მომხმარებელს მისი კომპიუტერთან შეერთების მარტივი პროცესის შემდეგ დაუყოვნებლივ შეუძლია შეუდგეს მუშაობას.

არდუინოს პლატფორმის რამდენიმე ვერსია არსებობს. დღეისათვის ყველაზე გავრცელებულია Arduino Uno. მას უმეტესწილად ირჩევენ ადამიანები, რომლებიც მიკროკონ-

ტროლერების დაპროგრამების შესწავლას იწყებენ. არდუინო წარმოადგენს დაფას, მასზე განთავსებული კომპონენტებით, რომელთა შორის ყველაზე მნიშვნელოვანია მიკროკონტროლერი. Arduino Uno აგებულია ATmega328P-ს ბაზაზე. მიკროკონტროლერი ამ პლატფორმის ძირითადი გამოთვლითი სისტემაა, სწორედ მისთვის იქმნება პროგრამული უზრუნველყოფა, რომლის საშუალებითაც ის ურთიერთქმედებს გარე სამყაროსთან მონაცემთა შეტანა/გამოტანის სპეციალური პორტებით. Arduino Uno-ს აქვს 20 ციფრული და ანალოგური (6 ანალოგური შესასვლელი, 14 - ციფრული შესასვლელ-გამოსასვლელი) საკონტაქტო გამომყვანი.

Arduino Nano გამოყენებული კომპონენტების შედარებით მცირე ზომებით განსხვავდება Arduino Uno-სგან. შესაბამისად, დაფაც უფრო მცირე ზომისაა. Arduino Uno-ს დიდი ზომის მქონე ანალოგია Arduino Mega, მასში მოთავსებულია ATmega1280 ან ATmega2560 მიკროკონტროლერი. მისი საშუალებით უფრო სერიოზული პროექტების განხორციელება შეიძლება. Arduino Mega-ს შეიძლება მეტი რაოდენობით მოწყობილობები მიუერთდეს, მას ჯამში აქვს 70 ციფრული და ანალოგური (16 ანალოგური შესასვლელი, 54 ციფრული შესასვლელ-გამოსასვლელი) საკონტაქტო გამომყვანი. ასევე მეტია მეხსიერების მოცულობაც Uno-სა და Nano-სთან შედარებით.

Arduino Leonardo თავისი ზომებით ემთხვევა Arduino Uno-ს, განსხვავება გამოყენებულ მიკროკონტროლერშია.

მასში გამოიყენება ATmega32u4. ამ დაფას კომპიუტერი ამოიცნობს ისევე, როგორც მასთან მიერთებულ კლავიატურას ან მაუსს, ამიტომ ის ხშირად გამოიყენება სხვადასხვა სახის ჯოისტიკისა და შეტანის მოწყობილობების შესაქმნელად.

Arduino Due წარმოადგენს დაფას ARM მიკროპროცესორის ბაზაზე - 32bit Cortex-M3 ARM SAM3U4E.

Arduino Yun არის დაფა ჩაშენებული WiFi მხარდაჭერით - Atmega32u4 და Atheros AR9331–ის ბაზაზე.

Arduino Micro წარმოადგენს კომპაქტურ გადაწყვეტას ATmega32u4–ის ბაზაზე.

Arduino Mini შექმნილია სპეციალურად მცირე ზომის მოწყობილობების დასაპროექტებლად. მისი ზომები კომპაქტურია და აგებულია ATmega168 მიკროკონტროლერის ბაზაზე. არსებობს ასევე Arduino Pro Mini ვერსია, მას არა აქვს საკუთარი USB, მისი დაპროგრამება ხდება სპეციალური USB გარდამქმნელებისა და ადაპტერების მეშვეობით.

Arduino Ethernet წარმოადგენს კონტროლერს ქსელთან მუშაობის ჩაშენებული მხარდაჭერით. დაფას შეიძლება დაემატოს კვების დამატებითი (Power over Ethernet) მოდული.

თავი 1. არდუინოს დაფა და IDE

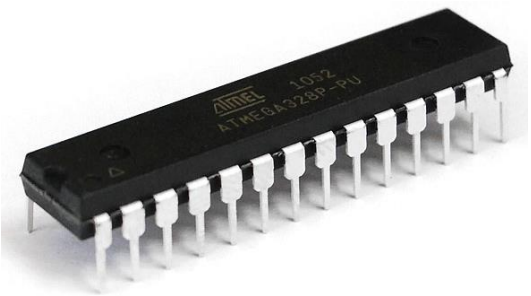
არდუინოს დაფა

არდუინო წარმოადგენს პატარა ზომის კომპიუტერს, რომლის დაპროგრამებაც შეიძლება ფიზიკურ ობიექტებთან ურთიერთქმედებისთვის, სხვადასხვა სახის შესასვლელი და გამოსასვლელი სიგნალების მეშვეობით. დღეისათვის ყველაზე გავრცელებული მოდელი Arduino Uno (სურ. 1.1.) ასე გამოიყურება:

სურ. 1.1. Arduino Uno

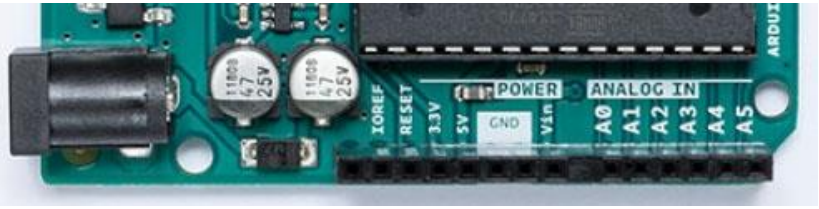
არდუინოს კომპიუტერთან მიერთება ხდება USB კაბელის მეშვეობით. ეს არდუინოზე კვების ძაბვის მიწოდების, პროგრამის ატვირთვის და მონაცემების მიღება/გადაგზავნის საშუალებას იძლევა. არდუინოს ასევე შესაძლებელია მივუერთოთ სტანდარტული კვების ბლოკი.

Arduino Uno აგებულია ATmega328P-ს ბაზაზე.



სურ. 1.2. მიკროკონტროლერი

მიკროკონტროლერი არდუინოს „ტვინია“, პატარა კომპიუტერი, რომელიც შეიცავს მიკროპროცესორს, ასრულებს ინსტრუქციებს, გააჩნია სხვადასხვა სახის მეხსიერება მონაცემებისა და ბრძანებების შესანახად, ასევე შესასვლელები და გამოსასვლელები მონაცემების შეტანა/გამოტანისთვის. მიკროკონტროლერის ქვედა მხარეს, არდუინოს დაფაზე, განლაგებულია კონტაქტების ორი მწკრივი, როგორც ეს სურ. 1.3-ზეა ნაჩვენები.



სურ. 1.3. ელექტროკვებისა და ანალოგური შესასვლელების კონტაქტები

პირველ მწკრივში მოთავსებულია ელექტროკვების და Reset-ის კონტაქტები, მეორე მწკრივში - ანალოგური შესასვლელების ექვსი კონტაქტი (სურ.1.3). არდუინოს დაფის ზედა მხარეს კონტაქტების კიდევ ორი მწკრივია მოთავსებული (სურ. 1.4).

სურ. 1.4. ციფრული შესასვლელების/გამოსასვლელების კონტაქტები

საკონტაქტო გამომყვანები ნომრებით 0-დან 13-ის ჩათვლით ციფრულ შესასვლელ/გამოსასვლელებს წარმოადგენს. მათი საშუალებით შეიძლება განისაზღვროს შესასვლელი ელექტრული სიგნალების არსებობა ან არარსებობა ან შეიძლება მოხდეს გამოსასვლელი სიგნალების გენერირება. საკონტაქტო გამომყვანები ნომრებით 0 და 1, რომლებიც ასევე ცნობილია როგორც მიმდევრობითი პორტები, შეიძლება გამოყენებული იყოს სხვა მოწყობილობებთან (მაგ., კომპიუტერთან) მონაცემების გაცვლისთვის, RS232 პროტოკოლის საშუალებით. საკონტაქტო გამომყვანებს, რომელთა შესაბამისი ნომრების წინ ტილდა წერია, განივ იმპულსური მოდულაციის (PWM) გენერირება შეუ-

ძლია. ეს საჭიროა მაგალითად, განათების ეფექტების შესაქმნელად ან ელექტროძრავას მართვისთვის.

არდუინოს დაფაზე, ზემოთ აღნიშნული გამომყვანების ქვემოთ, მოთავსებულია შუქდიოდები. ისინი ანათებს, როცა მათში დენი გადის. არდუინოს დაფაზე ოთხი შუქდიოდია: ერთი, რომელსაც აწერია ON, დაფაზე მარჯვენა მხარესაა მოთავსებული და მასზე მიერთებული ელექტროკვების ინდიკატორს წარმოადგენს, დანარჩენი სამი მარცხენა მხარეს, ერთმანეთის მიყოლებითაა განთავსებული (სურ. 1.5):

სურ. 1.5. შუქდიოდები არდუინოს დაფაზე

- Serial communication LEDs - მიმდევრობითი კომუნიკაციის შუქდიოდები

- Flash LED - L შუქდიოდი
- Power ON LED - შუქდიოდი ელექტროკვების ინდიკაციისთვის

RX და TX შუქდიოდები იმ შემთხვევაში ინთება, როდესაც ხდება მონაცემების გაცვლა არდუინოს დაფასა და მიმდევრობითი პორტისა და USB-ს მეშვეობით ჩართულ მოწყობილობებს შორის. L შუქდიოდი მიერთებულია მე-13 საკონტაქტო გამომყვანთან. ამ შუქდიოდებისგან მარცხნივ მოთავსებული მცირე ზომის შავი კვადრატი მიკროკონტროლერია, რომელიც USB ინტერფეისს მართავს. სწორედ ეს მიკროკონტროლერი აძლევს საშუალებას არდუინოს დაფას გააგზავნოს მონაცემები კომპიუტერზე ან მიიღოს ისინი კომპიუტერიდან.

სურ. 1.6-ზე ნაჩვენებია არდუინოს დაფაზე არსებული Reset ღილაკი.

სურ. 1.6. Reset ღილაკი

როგორც ეს ზოგჯერ ხდება კომპიუტერების შემთხვევაში, არდუინოს დაფასაც შეიძლება დასჭირდეს გადატვირთვა. Reset ღილაკი სწორედ ამისთვისაა საჭირო.

არდუინოს სისტემის ერთ-ერთი ღირსშესანიშნავი თავისებურებაა მისი გაფართოების სიმარტივე, მასში მარტივად შეიძლება ჩაემატოს ახალი აპარატურული ფუნქციები. არდუინოს დაფის ზედა და ქვედა მხარეზე მოთავსებული კონტაქტების ორივე მწკრივი გაფართოების დაფის მიერთების საშუალებას იძლევა. მაგალითისთვის, სურ. 1.7-ზე ნაჩვენებია გაფართოების დაფა Ethernet ინტერფეისით, რომლის საშუალებითაც არდუინოს შეუძლია გაცვალოს მონაცემები ქსელსა და ინტერნეტში.

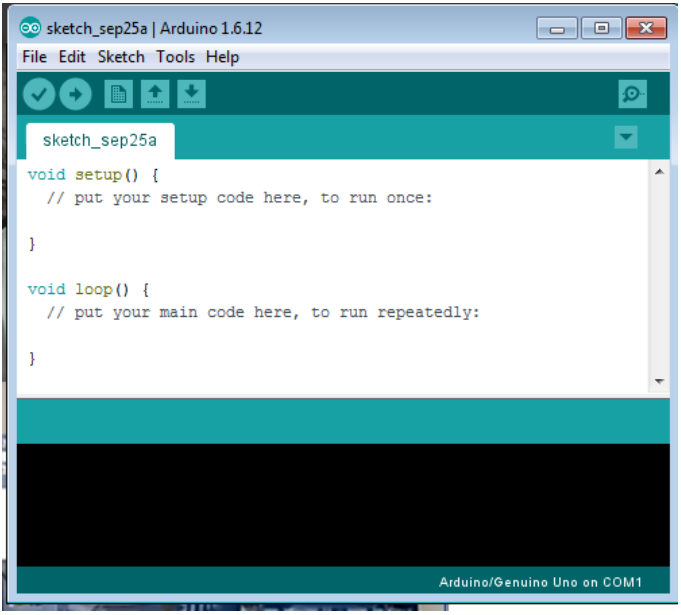
სურ. 1.7. Ethernet გაფართოების დაფა არდუინოსთვის

აღსანიშნავია, რომ Ethernet გაფართოების დაფას ასევე აქვს კონტაქტები (სურ. 1.7), რომლებზეც დამატებითი გაფართოების დაფების მიერთებაა შესაძლებელი.

არდუინო IDE

არდუინოს დასაპროგრამებლად გამოიყენება C/C++ დაპროგრამების ენების სპეციალიზებული ვარიანტი AVR მიკროკონტროლერებისთვის.

არდუინოს ვებსაიტიდან (arduino.cc/en/guide/windows) ჩამოვტვირთოთ შესაბამისი პროგრამული უზრუნველყოფა (Arduino Integrated Development Environment (IDE)), გარემო, რომელშიც არდუინოს დაპროგრამებაა შესაძლებელი და დავაინსტალიროთ ჩვენს კომპიუტერში. პროგრამის გაშვების შედეგად, ეკრანზე ასეთ სურათს მივიღებთ:



სურ. 1.8. Arduino IDE

Arduino IDE მარტივ ტექსტურ რედაქტორს წააგავს. პროგრამებს, რომლებიც ამ გარემოში იწერება, სკეტჩებს უწოდებენ. ნაგულისხმევი წესის თანახმად, სკეტჩს ენიჭება სახელი, რომელიც მიმდინარე თარიღს შეიცავს (სურ.1.8).

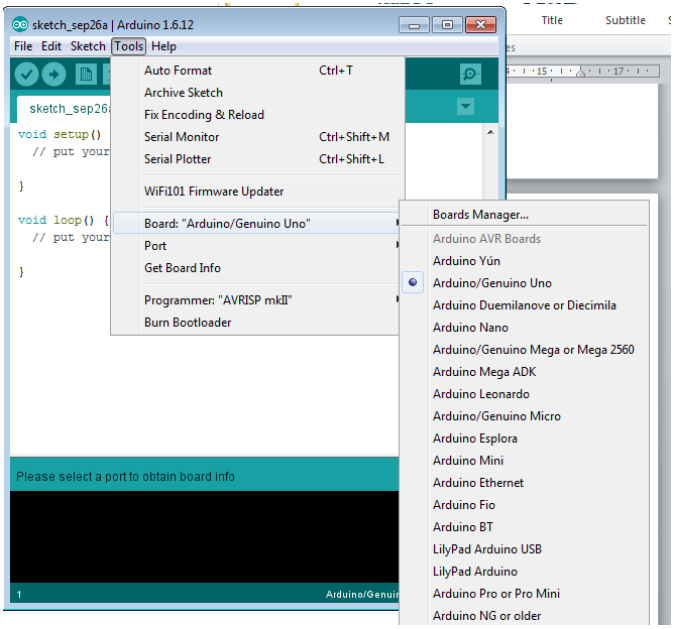
პირველი სკეტჩის შექმნა IDE-ში

ყველა პროგრამა, რომელიც არდუინოს გარემოში იწერება, ორი ძირითადი ნაწილისგან შედგება: void setup() და void loop().

setup() ფუნქციის გამოძახება ერთხელ, მიკროკონტროლერის ჩართვისას, ხდება. ამ ფუნქციის შიგნით იწერება ცვლადების ინიციალიზაციის კოდი, ხორციელდება საკონტაქტო გამომყვანების მუშაობის რეჟიმის დაყენება და ა.შ. ის მონაცემები, რასაც პროგრამა რამდენჯერმე იყენებს, მოთავსებულია loop() ნაწილში. loop() არის მთავარი პროგრამული ფუნქცია, რომელიც სრულდება მანამ, სანამ არდუინოს კვება მიეწოდება. loop() ფუნქცია უსასრულო ციკლს წარმოადგენს, რომელშიც მიმდევრობით სრულდება ამ ფუნქციის ტანში აღწერილი ბრძანებები. ფუნქციის მუშაობის დასრულების შემდეგ ისევ და ისევ ხდება მისი გამოძახება.

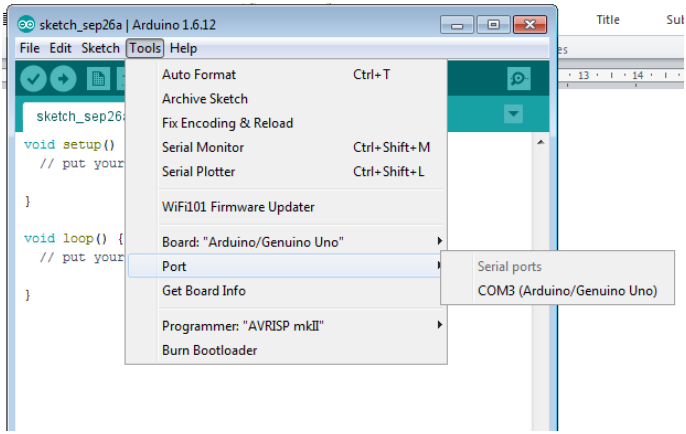
სანამ მუშაობას შევუდგებოდეთ, პროგრამის გარემოში ორი პარამეტრი უნდა დავაყენოთ. პირველი, რაც ჩვენ უნდა გავაგებინოთ არდუინოს გარემოს (IDE – Integrated Development Environment) არის ის, თუ რომელ არდუინოს ვი-

ყენებთ. ამისათვის შევასრულოთ ქვემოთ მოცემულ სურათზე ნაჩვენები მოქმედება:



სურ. 1.9. არდუინოს შესაბამისი ვერსიის დაყენება

ჩვენს შემთხვევაში, უკვე ჩანს, რომელ არდუინოს ვიყენებთ, ამიტომაც არაფრის დაყენება არ არის საჭირო. ზოგჯერ, ეს პარამეტრი მითითებული არ არის, ამიტომაც ის უნდა დაყენდეს მომავალში პრობლემების თავიდან აცილების მიზნით. მეორე პარამეტრი, რომელიც უნდა განისაზღვროს, არის ის მიმდევრობითი პორტი, რომელთანაც არდუინოა მიერთებული. ჩვენს შემთხვევაში, ეს არის COM3.



სურ. 1.10. მიმდევრობითი პორტი, რომელზეც მიერთებულია არდუინო

თუ დავათვალიერებთ არდუინოს, ვნახავთ რომ მას რამდენიმე საკონტაქტო გამომყვანი აქვს. მის ერთ-ერთ მხარეზე მოთავსებულია გამომყვანები 0-დან 13-ის ჩათვლით. მე-13 გამომყვანის შემდეგ მოდის „მიწა“ (GND). მე-13 გამომყვანთან L შუქდიოდია დაკავშირებული. ეს ნიშნავს რომ თუ ვაკეთებთ რაიმეს მე-13 გამომყვანზე, ეს მასთან დაკავშირებულ შუქდიოდსაც შეეხება.

დავწეროთ ჩვენი პირველი პროგრამა. ამოცანის მიზანია ავანთოთ/ჩავაქროთ შუქდიოდი, რომელიც მე-13 საკონტაქტო გამომყვანთანაა დაკავშირებული.

როგორც ზემოთ უკვე ავღნიშნეთ, პროგრამის ჩართვისას ეკრანზე გამოჩნდება შემდეგი კოდი:

```
void setup() {
    // აქ მოათავსეთ პარამეტრების დასაყენებელი კოდი
}
```

```
void loop() {
    // აქ მოათავსეთ ძირითადი კოდი
}
```

პროგრამის კოდში ორი დახრილი ხაზი ერთსტრიქონიან კომენტარს აღნიშნავს. მრავალსტრიქონიანი კომენტარის აღსანიშნავად გამოიყენება /* */ სიმბოლოები. კომენტარის ტექსტი იგნორირდება კომპილატორის მიერ. ხშირ შემთხვევაში, ის კარგ დახმარებას უწევს პროგრამისტს, როდესაც საკუთარ პროგრამაში გარკვეული ცვლილებების შეტანა უწევს. კომენტარი მკითხველს პროგრამის კოდის აღქმას უადვილებს.

პროგრამის დასაწერად დაგვჭირდება ფუნქცია pinMode()-ის გამოყენება იმისათვის, რომ განვსაზღვროთ საკონტაქტო გამომყვანი, რომელთანაც ვმუშაობთ და მისი მდგომარეობა (INPUT ან OUTPUT). pinMode ფუნქციის არგუმენტებია 13 და OUTPUT. თუ ვკითხულობთ ინფორმაციას საკონტაქტო გამომყვანიდან, ეს იქნება INPUT, თუ ვწერთ ინფორმაციას საკონტაქტო გამომყვანში, მაშინ გვექნება OUTPUT. რადგანაც ჩვენ ვწერთ (ვაგზავნით) მე-13 გამომყვანში ინფორმაციას, ამიტომაც pinMode ფუნქციის არგუმენტი იქნება OUTPUT. pinMode(13, OUTPUT) ბრძანებით საკონტაქტო გამომყვანის კონფიგურირება ხდება, როგორც გამოსასვლელის. მიაქციეთ ყურადღება, რომ OUTPUT რეგისტრისადმი მგრძნობიარეა, ყველა სიმბოლო მთავრული უნდა იყოს.

პროგრამის კოდი მიიღებს შემდეგ სახეს:

```
void setup() {
  pinMode(13,OUTPUT);
}
void loop() {
}
```

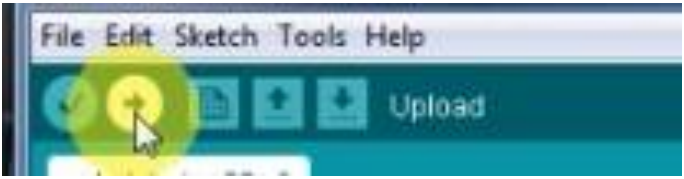
loop() ფუნქციის შიგნით ვწერთ digitalWrite ფუნქციას. მისი საშუალებით შუქდიოდის ჩართვა ან გამორთვა შეგვიძლია. ამ ფუნქციის ერთი არგუმენტია იმ საკონტაქტო გამომყვანის ნომერი, რომელშიც ხდება ინფორმაციის ჩაწერა, ხოლო მეორე არგუმენტი შეიძლება იყოს HIGH ან LOW. თუ ჩვენ ვაგზავნით, ვთქვათ 5V სიგნალს მე-13 საკონტაქტო გამომყვანზე, მაშინ არგუმენტად ვწერთ HIGH-ს. ეს ნიშნავს რომ ამ დროს შუქდიოდი ინთება.

პროგრამის კოდი:

```
void setup() {
//მე-13 საკონტაქტო გამომყვანის კონფიგურირება, როგორც გამოსასვლელის
 pinMode(13,OUTPUT);
}
void loop() {
//მე-13 საკონტაქტო გამომყვანში ვწერთ მაღალ ლოგიკურ მნიშვნელობას ანუ ვაწვდით +5 ვოლტს
 digitalWrite(13,HIGH);
}
```

ჩვენ განვსაზღვრეთ საკონტაქტო გამომყვანის მუშაობის რეჟიმი და გავაგზავნეთ სიგნალი მასზე. პროგრამის შესრულების შედეგად, შუქდიოდი უნდა აინთოს. პროგრამა, რომელიც გვაქვს კომპიუტერში, ავტვირთოთ არდუინოზე.

ცხადია, რომ არდუინო USB კაბელის საშუალებით კომპიუტერთანაა მიერთებული. პროგრამის არდუინოში ჩასაწერად upload ღილაკს უნდა დავაჭიროთ.



სურ. 1.11. პროგრამის ატვირთვა არდუინოზე

ატვირთვის პროცესის დასრულების შემდეგ დავინახავთ, რომ შუქდიოდი ჩაირთვება.

ახლა გავაუმჯობესოთ პროგრამა. გამოვაცხადოთ მთელი ტიპის გლობალური ცვლადი LEDPin და მივანიჭოთ მას მნიშვნელობა 13. პროგრამის კოდში 13-ის ნაცვლად ცხადია, რომ ცვლადის სახელი (LEDPin) უნდა დავწეროთ.

დავაჭიროთ upload ღილაკს. პროგრამის შესრულების შედეგი იგივეა. განსხვავება ის არის რომ, თუ ცვლადის მნიშვნელობის შეცვლა დაგვჭირდება, პროგრამაში ამას მხოლოდ ერთხელ გავაკეთებთ.

პროგრამის კოდს ექნება ასეთი სახე:

```
// საკონტაქტო გამომყვანების აღნიშვნისას  int-ის წინ const-ს იმიტომ ვიყენებთ, რომ ამ მონაცემის პროგრამაში შემდგომი შეცვლა არ ხდება და ის მუდმივაა. const-ის ნაცვლად შეიძლება #define-ის გამოყენებაც: #define LEDPin 13
const int LEDPin=13;
void setup() {
  pinMode(LEDPin,OUTPUT);
```

```
}
void loop() {
  digitalWrite(LEDPin,HIGH);
}
```

იმისათვის, რომ შუქდიოდი გამოვრთოთ, digitalWrite ფუნქციას HIGH-ს ნაცვლად არგუმენტად LOW უნდა დავუწეროთ. თუ ისევ ავტვირთავთ პროგრამას არდუინოზე, დავინახავთ, როგორ გამოირთვება შუქდიოდი.

ახლა სხვა მიზანი დავისახოთ. შევქმნათ პროგრამა, რომლის შესრულების შედეგად შუქდიოდი მონაცვლეობით აინთება (ON) და ჩაქრება (OFF). იმისათვის, რომ შუქდიოდი გამოირთოს, პროგრამის კოდში უნდა დაემატოს ფუნქცია digitalWrite (LEDPin, LOW);

პროგრამის კოდი:

```
const int LEDPin=13;
void setup() {
  pinMode(LEDPin,OUTPUT);
}
void loop() {
 digitalWrite(LEDPin,HIGH);
 digitalWrite (LEDPin, LOW);
}
```

დავაჭიროთ upload ღილაკს. თითქოს უცნაურია, შუქდიოდი მუდმივად ანათებს. სინამდვილეში, შუქდიოდი ციმციმებს. პროგრამა მუშაობს, როგორც უსასრულო ციკლი და შუქდიოდის ანთება/ჩაქრობა იმდენად სწრაფად ხდება, რომ მას ჩვენი მხედველობა ვერ აღიქვამს. იმისათვის, რომ

თვალნათლივ დავინახოთ შუქდიოდის ციმციმი, პროგრამაში უნდა ჩავრთოთ delay ფუნქცია, რომელიც ON და OFF მდგომარეობას შორის დაყოვნებას უზრუნველყოფს. პროგრამის კოდში ჩავამატოთ სტრიქონი delay(1000). ამ ფუნქციის პარამეტრი მილიწამებშია მოცემული. ჩვენს შემთხვევაში, დაყოვნების დრო იქნება 1000 მილიწამი (1 წამი).

პროგრამის კოდი შემდეგ სახეს მიიღებს:

```
const int LEDPin=13;
void setup() {
  pinMode(LEDPin,OUTPUT);
}
void loop() {
  digitalWrite(LEDPin,HIGH);
  delay(1000);
  digitalWrite (LEDPin, LOW);
  delay (1000);
}
```

როგორც პროგრამის კოდიდან ჩანს, ჯერ უნდა მოხდეს შუქდიოდის ანთება, შემდეგ 1 წამის განმავლობაში ხდება დაყოვნება ანუ შუქდიოდი რჩება ანთებულ მდგომარეობაში, ამის შემდეგ შუქდიოდი გამოირთვება და მისი გამორთულ მდგომარეობაში ყოფნის დრო ისევ 1 წამია, შემდეგ ისევ აინთება შუქდიოდი და ა.შ. ეს პროცესი უსასრულოდ, ციკლურად მეორდება.

დავაჭიროთ upload ღილაკს. დავინახავთ, როგორ ციმციმებს შუქდიოდი.

პროგრამის კოდში შევიტანოთ კიდევ ერთი ცვლილება. გამოვაცხადოთ int ტიპის ცვლადი waitTime=1000.

კოდს ექნება სახე:

```
const int LEDPin=13;
int waitTime=1000;
void setup() {
  pinMode(LEDPin,OUTPUT);
}
void loop() {
 digitalWrite(LEDPin,HIGH);
 delay(waitTime);
 digitalWrite (LEDPin, LOW);
 delay (waitTime);
}
```

დავაჭიროთ ისევ upload ღილაკს. გასაგებია, რომ პროგრამის შესრულებით იგივე შედეგი მიიღება.

თუ ჩვენ შევცვლით waitTime ცვლადის მნიშვნელობას და მივანიჭებთ მას 250-ს, პროგრამის შესრულების შედეგად ვნახავთ, რომ შუქდიოდი უფრო სწრაფად ციმციმებს; მიზეზი დაყოვნების დროის შემცირებაა.

თუ გვინდა, რომ დაყოვნების დრო სხვადასხვა იყოს შუქდიოდის ჩართული და გამორთული მდგომარეობებისთვის, მაშინ ორი ცვლადის შემოტანა მოგვიწევს. დავარქვათ მათ, შესაბამისად waitTimeOn და waitTimeOff.

პროგრამის კოდი შემდეგ სახეს მიიღებს:

```
const int LEDPin=13;
int waitTimeOn=100;
```

```
int waitTimeOff=900;
void setup() {
  pinMode(LEDPin,OUTPUT);
}
void loop() {
  digitalWrite(LEDPin,HIGH);
  delay(waitTimeOn);
  digitalWrite (LEDPin, LOW);
  delay (waitTimeOff);
}
```

პროგრამის შესრულების შედეგად დავინახავთ, რომ შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო უფრო მეტია. თუ გვინდა პირიქით იყოს, მაშინ შეგვიძლია შევცვალოთ დაყოვნების დროის მნიშვნელობები (მაგ.: WaitTimeOn=900; waitTimeOff=100).

არდუინოს ბიბლიოთეკები

საზოგადოდ, ბიბლიოთეკების ორი სახე არსებობს: სტანდარტული და დამატებითი.

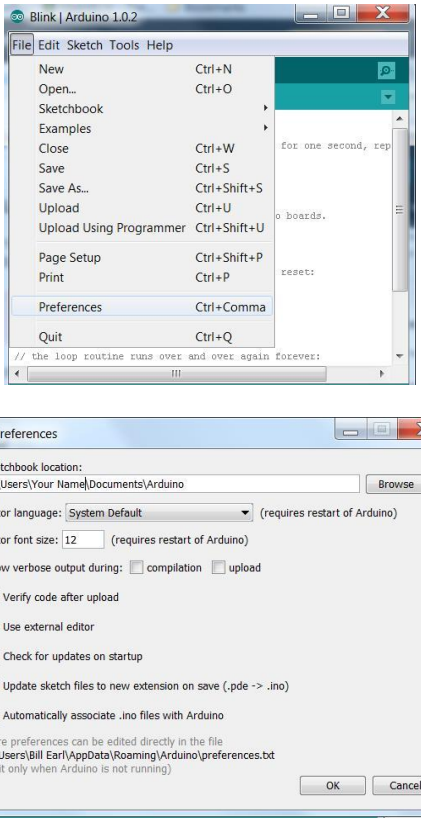
Arduino IDE-ში არის სტანდარტული ბიბლიოთეკები, რომლებიც ხშირად გამოიყენება და შედის Arduino IDE-ს პროგრამულ უზრუნველყოფაში. ეს ბიბლიოთეკები შეიცავს ფუნქციებს, ისინი სტანდარტულ ინტერფეისებთან სამუშაოდ გამოიყენება, მაგალითად, თხევადკრისტალური ეკრანი, სერვოძრავები და ა.შ. სტანდარტული ბიბლიოთეკები

Arduino IDE-ს ინსტალაციის დროს C:\Program Files \Arduino\libraries საქაღალდეში თავსდება.

ინტერნეტში დიდი რაოდენობითაა დამატებითი ბიბლიოთეკები სხვადასხვა მოწყობილობისთვის, საკმაოდ მოხერხებული ფუნქციებით. ძირითადად, ეს ბიბლიოთეკები მოთავსებულია Arduino Playground, Github და Google Code რესურსებზე. არდუინოს დამატებით ბიბლიოთეკებს ხშირად წერენ ენთუზიასტი პროგრამისტები და სხვადასხვა მოწყობილობების მწარმოებელი კომპანიები. მაგალითად, კომპანია Adafruit-ს აქვს არდუინოს ყველა მოდელთან თავსებადი 100-ზე მეტი ბიბლიოთეკა.

დამატებითი ბიბლიოთეკების დასაყენებლად მნიშვნელოვანია საჭირო ბიბლიოთეკა შესაბამის საქაღალდეში დავაყენოთ, წინააღმდეგ შემთხვევაში, ბიბლიოთეკას კომპილატორი ვერ იპოვის. Windows ოპერაციული სისტემისთვის ეს საქაღალდეა Arduino და ის მოთავსებულია Documents საქაღალდეში, საქაღალდის სახელი იგივეა MAC სისტემისთვის, Linux-ში საქაღალდეს ეწოდება Scetchbook და მოთავსებულია /home/<username>-ში.

დამატებითი ბიბლიოთეკის დასაყენებლად, Arduino IDE-ში გავხსნათ File->Preferences. მოვძებნოთ ჩვენი სკეტჩების ადგილმდებარეობა. ჩვეულებრივ, ეს არის საქაღალდე Arduino, რომელიც თავის მხრივ მოთავსებულია Documents საქაღალდეში (სურ.1.12).



სურ. 1.12. არდუინოს დამატებითი ბიბლიოთეკის დაყენება

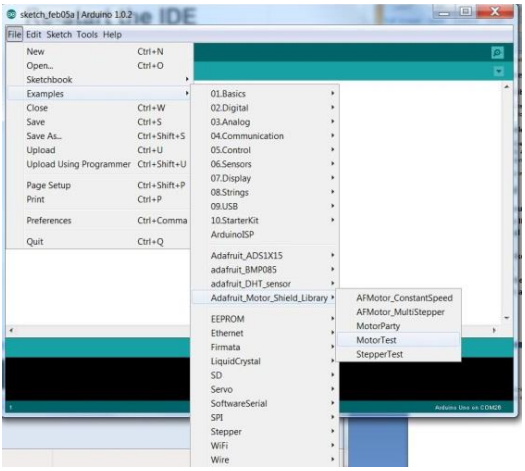
ამის შემდეგ დავხუროთ ფანჯარა და Arduino IDE-ც. Explorer-ის მეშვეობით გადავიდეთ Documents\Arduino\-ში და შევქმნათ (თუ არ არის შექმნილი) ახალი საქაღალდე, დავარქვათ Libraries.

დავრწმუნდეთ, რომ Arduino IDE დახურულია და გადმოვწეროთ რაიმე დამატებითი ბიბლიოთეკა, მაგალითად Github-დან. დავუშვათ, ავირჩიეთ Adafruit-Motor-

Shield-library. როგორც წესი, ეს არის Zip ფაილი. განარქივებული საქაღალდე ჩავსვათ Libraries საქაღალდეში.

ზოგიერთ შემთხვევაში, გადმოწერილი საქაღალდის სახელწოდება ტირეებს შეიცავს, Arduino IDE ვერ ცნობს საქაღალდეებს, რომელთა სახელწოდებაშიც ტირეებია. ასე რომ, ტირეს ნაცვლად შეგვიძლია ჩავსვათ ქვედა ტირე.

ჩავტვირთოთ თავიდან Arduino IDE და შევამოწმოთ, გაჩნდა თუ არა ახლად დამატებული ბიბლიოთეკა ბიბლიოთეკების სიაში. ეს ხდება მენიუს პუნქტიდან File->Examples. ჩავტვირთოთ მაგალითი და გავუშვათ შემოწმებაზე. ამ კონკრეტულ შემთხვევაში, File->Examples-ში უნდა ვნახოთ Adafruit_Motor_Shield_Library (სურ. 1.13).



სურ.1.13. არდუინოს დამატებითი ბიბლიოთეკის დაყენება

თავი 2. ელექტრონიკის საფუძვლები

დენის ძალა. ძაბვა. სიმძლავრე

ელექტრული დენი დამუხტული ნაწილაკების მოწესრიგებულ მოძრაობას ეწოდება. გამტარების უმეტესობაში ელექტრული დენის გატარებაზე პასუხს უარყოფითად დამუხტული ელექტრონები აგებენ. ბატარეის ან აკუმულატორის მიერ გამომუშავებულ დენს, რომელიც მხოლოდ ერთი მიმართულებით მოძრაობს, მუდმივი დენი ეწოდება. დენის ძალა იზომება ამპერებში (A).

ძაბვა წარმოადგენს პოტენციალთა სხვაობას სქემის დადებით და უარყოფით პოლუსებს შორის, იზომება ვოლტებში (V). რაც მეტია ძაბვა, მით მეტია დენის ძალა.

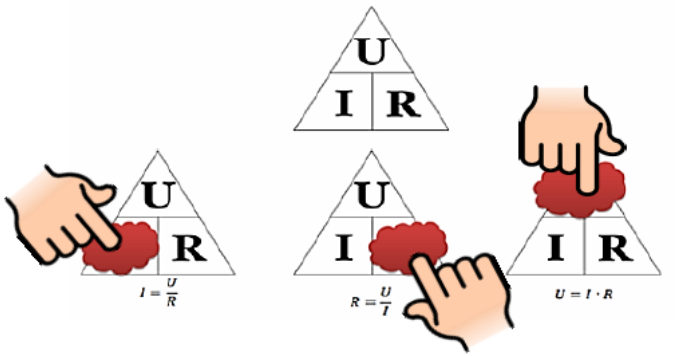
სიმძლავრე იმ სიჩქარის გამოხატულებას წარმოადგენს, რომლითაც ელექტრული მოწყობილობა ენერგიას ერთი ფორმიდან მეორეში გარდაქმნის. სიმძლავრე ვატებში (W) იზომება. 100W სიმძლავრის ნათურას უფრო კაშკაშა ნათება აქვს, ვიდრე 60W სიმძლავრისას, რადგანაც უფრო მძლავრი ნათურა დროის ერთეულში მეტ ელექტრულ ენერგიას გარდაქმნის სინათლედ და სითბოდ. ძაბვას, დენის ძალასა და სიმძლავრეს შორის მარტივი მათემატიკური დამოკიდებულება არსებობს:

$$P = U * I,$$

სადაც P არის სიმძლავრე, U - ძაბვაა, I-დენია.

ომის კანონი

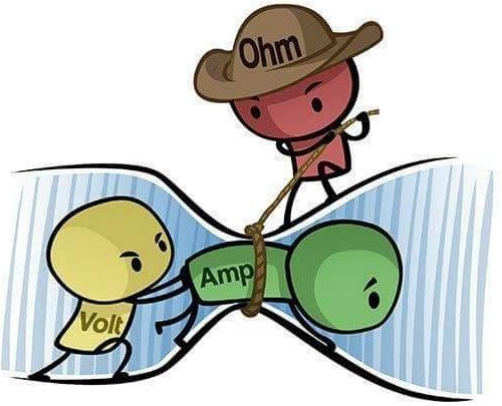
ომის კანონი დენის ძალას, წინაღობასა და ძაბვას შორის დამოკიდებულებას გამოხატავს: I=U/R. თუ ცნობილია რომელიმე ორი სიდიდე, მარტივად შეიძლება გამოვთვალოთ მესამე. ომის კანონის დამახსოვრება არანაირ სირთულეს არ წარმოადგენს, თუმცა ამისთვის ხშირად იყენებენ სურ. 2.1-ზე მოცემულ გამოსახულებას:



სურ. 2.1. სამკუთხედი ომის კანონის დასამახსოვრებლად

ამ სურათის მეშვეობით, ადვილად შეიძლება მივხვდეთ, როგორ გამოვთვალოთ ძაბვა, დენის ძალა ან წინაღობა რომელიმე ორი სიდიდის მნიშვნელობის ცოდნის შემთხვევაში. დავაფაროთ საძიებელ სიდიდეს თითი, დარჩენილი გამოსახულების მიხედვით მის მნიშვნელობას ადვილად ვიპოვით.

ომის კანონის არსს და ფორმულაში შემავალ სიდიდეებს შორის ურთიერთდამოკიდებულებას კარგად ასახავს ქვემოთ ნაჩვენები სურათი.



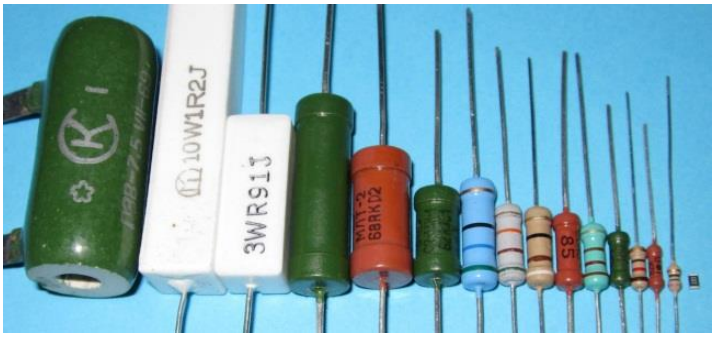
სურ.2.2. დენს, ძაბვასა და წინაღობას შორის ურთიერთდამოკიდებულების ილუსტრაცია

რეზისტორი

რეზისტორი კომპონენტია, რომელსაც განსაზღვრული ელექტრული წინაღობა გააჩნია. სამართლიანობისთვის უნდა აღინიშნოს, რომ წინაღობა მხოლოდ რეზისტორს არა აქვს. წინაღობა აქვს ნათურებს, ძრავებს, ტრანზისტორებს, უბრალო გამტარებს. თუმცა, მათთვის წინაღობა მთავარი მახასიათებელი არ არის. ნათურა ანათებს, ძრავა ბრუნავს, ტრანზისტორი აძლიერებს, გამტარი დენს ატარებს. რეზისტორის ძირითადი ფუნქცია მასში გამავალი დენისთვის წინააღმდეგობის გაწევაა.

არსებობს ორი სახის რეზისტორი: მუდმივი და ცვლადი.

სურ. 2.3-ზე გამოსახულია სხვადასხვა სახის მუდმივი რეზისტორი. სურათის მარჯვენა ნაწილში, სულ ბოლოს მოთავსებული, მცირე ზომის მართკუთხა ელემენტიც რეზისტორია, მხოლოდ „საკონტაქტო გამომყვანების“ გარეშე. ასეთი დეტალები ზედაპირული მონტაჟისთვის გამოიყენება და მათ SMD რეზისტორებს უწოდებენ.



სურ. 2.3. რეზისტორები

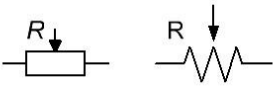
ელექტრულ სქემაზე მუდმივი რეზისტორი შემდეგნაირად გამოისახება:

სურ. 2.4. მუდმივი რეზისტორების ელექტრულ სქემაზე აღნიშვნის ვარიანტები

სურ. 2.5-ზე გამოსახულია ცვლადი რეზისტორები.

სურ. 2.5. ცვლადი რეზისტორები

სურ. 2.6-ზე მოცემულია ელექტრულ სქემაზე ცვლადი რეზისტორის აღნიშვნები.



სურ. 2.6. ცვლადი რეზისტორის ელექტრულ სქემაზე აღნიშვნის ვარიანტები

წინაღობის ერთეულია ომი. არსებობს წინაღობის უფრო დიდი ერთეულებიც.

1კილოომი = 1000 ომი

1 მეგაომი = 1000 კილოომი = 1000000 ომი

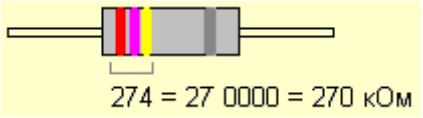
თითოეული რეზისტორი განსაზღვრულ წინაღობაზეა გათვლილი. მას ხშირად ნომინალურ წინაღობას უწოდებენ.

იმისათვის, რომ მივხვდეთ რამდენია კონკრეტული რეზისტორის წინაღობა, უნდა შევხედოთ მასზე მოცემულ ზოლებს. როგორც წესი, კორპუსზე არის 4 ან 5 ზოლი, მათგან სამი ან ოთხი ზოლი მოთავსებულია ერთად, ერთი - ცოტა მოშორებით. ვატრიალებთ რეზისტორს ისე, რომ ის ერთი მოშორებით არსებული ზოლი მარჯვენა მხარეს მოთავსდეს. შემდეგ ვიღებთ ცხრილს, რომელშიც მოცემულია რეზისტორის ფერადი ზოლების შესაბამისი ციფრები; მარცხნივ მოთავსებული ზოლის ფერები გადაგვყავს ციფრებში.

0	შავი
1	ყავისფერი
2	წითელი
3	ნარინჯისფერი
4	ყვითელი
5	მწვანე
6	ლურჯი
7	იასამნისფერი
8	რუხი
9	თეთრი

ცხრილი 2.1. რეზისტორზე ფერადი ზოლების შესაბამისი ციფრები

მივიღებთ სამნიშნა რიცხვს (სურ. 2.7)

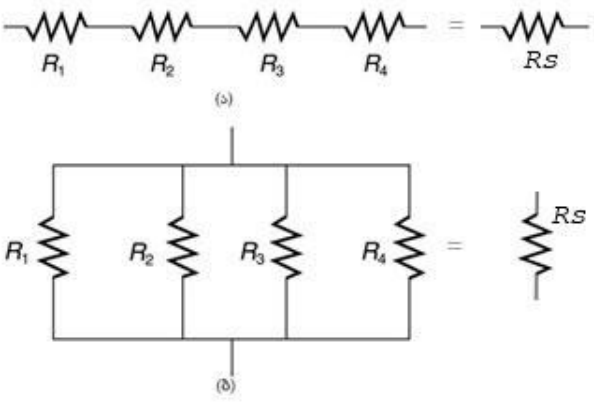


სურ. 2.7. რეზისტორის მარკირების მიხედვით წინაღობის განსაზღვრა

პირველი ზოლი წინაღობის პირველ ციფრს აღნიშნავს, მეორე ზოლი აღნიშნავს წინაღობის მეორე ციფრს. მესამე ზოლი წარმოადგენს მამრავლს (რეზისტორებისთვის, რომელთაც ოთხი ზოლი აქვთ) ან წინაღობის მესამე ციფრს (რეზისტორებისთვის ხუთი ზოლით). მეოთხე ზოლი წარმოადგენს მამრავლს რეზისტორებისთვის ხუთი ზოლით ან არის მნიშვნელობების გადახრის დიაპაზონი (სიზუსტე) ნომინალთან მიმართებაში, ოთხზოლიანი რეზისტორე-ბისთვის. მეხუთე ზოლი განსაზღვრავს მნიშვნელობების გადახრის დიაპაზონს (სიზუსტეს) ნომინალთან მიმარ-თებაში. ეს ნიშნავს, რომ ის ასახავს რეზისტორის წინაღობის სიზუსტის შესაბამისობას მის ნომინალთან. რადგანაც პრაქტიკაში საკმაოდ რთულია წინაღობის ზუსტი მნიშვნელობის მქონე რეზისტორების წარმოება, მათი ყიდვისას შეგვიძლია ავირჩიოთ გადახრის დიაპაზონის მნიშვნელობა პროცენტებში. ყავისფერი ზოლი შეესაბამება 1% გადახრის დიაპაზონს, ოქროსფერი - 5%-ს და ვერცხლისფერი - 10%-ს.

იმ შემთხვევაში, თუ გარკვეული მიზეზების გამო, რეზისტორის მარკირების წაკითხვა ვერ ხერხდება, წინაღობის განსაზღვრა შესაძლებელია სპეციალური მოწყობილობების საშუალებით. წინაღობის გასაზომად შეიძლება გამოვიყენოთ მულტიმეტრი. აქვე აღვნიშნავთ, რომ მულტიმეტრის საშუალებით შესაძლებელია არა მხოლოდ წინაღობის გაზომვა. ის გამოიყენება ასევე ძაბვის და დენის ძალის გასაზომად.

სქემებში ხშირად გამოიყენება რეზისტორების მიმდევრობითი და პარალელური შეერთება.

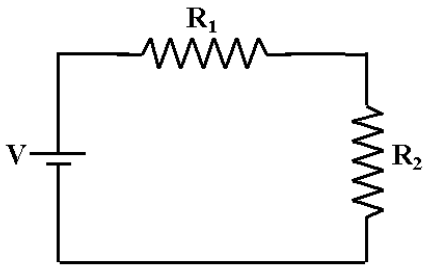


სურ 2.8. (ა) მიმდევრობითი და (ბ) პარალელური შეერთება

მიმდევრობითი შეერთებისას ეკვივალენტური წინაღობა Rs = R1 + R2 + R3 + R4, თუ მიმდევრობით ჩართული რეზისტორების რაოდენობა n-ის ტოლია, მაშინ ფორმულა მიიღებს სახეს Rs = R1 + R2 + R3 + … + Rn.

პარალელური შეერთებისას ეკვივალენტური წინაღობის გამოთვლა შემდეგი ფორმულით ხდება: 1/Rs = 1/R1 + 1/R2 + 1/R3 +1/R4, ხოლო n რაოდენობის რეზისტორებისთვის 1/Rs = 1/R1 + 1/R2 +… + 1/Rn.

განვიხილოთ სქემა ორი მიმდევრობით ჩართული რეზისტორით (სურ. 2.9):



სურ. 2.9. სქემა ორი მიმდევრობით ჩართული რეზისტორით

სქემაზე (სურ.2.9) გვაქვს ორი მიმდევრობით ჩართული რეზისტორი (R1=1000Ω, R2=9000Ω). დავუშვათ, კვების ძაბვის მნიშვნელობაა 5V. ეს ძაბვა ორივე რეზისტორზე ნაწილდება. მიმდევრობით ჩართული ორი რეზისტორი მუშაობს ისევე, როგორც ერთი, რომელსაც მათი ჯამური წინაღობა აქვს. მიმდევრობითი ჩართვის დროს წინაღობები იკრიბება R=R1+R2=10000Ω. დენის მნიშვნელობა ტოლი იქნება I=U/R=5/10000=0.0005A. რადგანაც დენის მნიშვნელობა უკვე ვიცით, შეგვიძლია გამოვთვალოთ ძაბვის მნიშვნელობები რეზისტორებზე. ძაბვა R1 რეზისტორზე იქნება

U=IR=0.0005*1000=0.5V. ძაბვა R2 რეზისტორზე იქნება U=IR=0.0005 *9000=4.5V.

ახლა განსხვავებული შემთხვევა განვიხილოთ: ვთქვათ, გვაქვს მიმდევრობით ჩართული ორი თანაბარი წინაღობის მქონე რეზისტორი. R1=R2=5000Ω. ცხადია, რომ მისი ეკვივალენტური სქემაა იგივე სქემა ერთი 10000Ω წინაღობის მქონე რეზისტორით. დენის მნიშვნელობა: I=U/R=5/10000=0.0005A. რადგანაც დენის მნიშვნელობა ვიცით, უკვე შეგვიძლია გამოვიანგარიშოთ U1 და U2 ძაბვები R1 და R2 რეზისტორებისთვის, შესაბამისად.

U1=I*R1=0.0005 *5000 = 2,5V

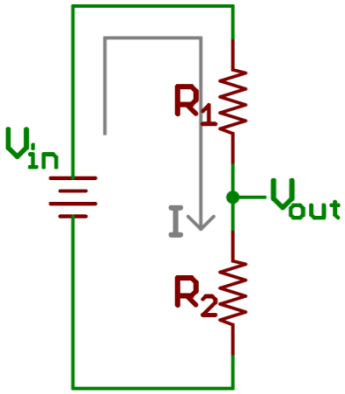
U2=I*R2=0.0005 *5000 = 2,5V

გასაგებია, რატომაც მიიღება ძაბვის თანაბარი მნიშვნელობები. იმ შემთხვევისთვის, როცა რეზისტორების წინაღობები განსხვავებულია (მაგ.: R1=1000, R2=9000), განსხვავებულია ძაბვები ამ რეზისტორებზე. R1-ზე ძაბვის ვარდნა იყო 0.5 ვოლტი, R2-ზე - 4.5 ვოლტი. მეტი წინაღობის მქონე რეზისტორზე ძაბვის ვარდნა მეტია. ასეთ შემთხვევაში ხდება ძაბვის გაყოფა. არდუინოსთან მუშაობის დროს ძალიან ხშირად გამოვიყენებთ ძაბვის გამყოფებს.

განვიხილოთ პოტენციომეტრი. პოტენციომეტრი არის ძაბვის რეგულირებადი გამყოფი, იგივე ცვლადი რეზისტორი. პოტენციომეტრს სამი გამომყვანი აქვს. მთლიანი წინაღობა (R_p) არ იცვლება, იცვლება R1 და R2-ის მნიშვნელობები. თუ ავიღებთ პოტენციომეტრს, რომლის

წინაღობაა 10000Ω, ეს ნიშნავს, რომ გარე გამომყვანებს შორის წინაღობა ყოველთვის იქნება 10000Ω.

თუ პოტენციომეტრის რეგულატორს მოვატრიალებთ, ვთქვათ, პირობითად მარცხნივ, მაშინ სქემის ზედა ნაწილში (სურ.2.10) წინაღობა გვექნება ნული, ხოლო ქვედა ნაწილში (იგულისხმება შუა ნაწილიდან ბოლომდე) წინაღობა იქნება 10000Ω. თუ რეგულატორს მოვატრიალებთ მარჯვნივ, მაშინ გვექნება პირიქით (ზედა მხარეს წინაღობა 10000 ომი, ქვედა მხარეს - ნული). თუ რეგულატორს მოვათავსებთ შუაში, მაშინ ერთ მხარეს გვექნება 5000Ω და მეორე მხარესაც 5000Ω წინაღობა. მთლიანი წინაღობა ტოლია $R_p=R_1+R_2$. გამოსასვლელი ძაბვა $U_{out}= (R_2/R_p)*U_{in}$



სურ. 2.10. ძაბვის გამყოფი

იმ შემთხვევაში, როდესაც $R_2=10000$, მივიღებთ $U_{out}= (10000/10000)*5 = 5$. ანუ გამოდის, რომ შესასვლელი და

გამოსასვლელი ძაბვები ტოლია. იმ შემთხვევაში, როდესაც R_2=0, მივიღებთ U_{out}= 0.

როდესაც შესასვლელზე 5V ძაბვა გვაქვს, პოტენციომეტრის გარე გამომყვანებს შორის ძაბვა არის 5V, პოტენციომეტრის რეგულატორის ტრიალით შესაძლებელია ძაბვის ნელ-ნელა გაზრდა/შემცირება 0-დან 5 ვოლტამდე.

კონდენსატორი

კონდენსატორი ორპოლუსა მოწყობილობაა, ტევადობის განსაზღვრული ან ცვლადი მნიშვნელობითა და მცირე გამტარობით. კონდენსატორს შეუძლია შეინახოს და გადასცეს ელექტრული დენის მუხტი ელექტრული წრედის სხვა ელემენტებს.

უმარტივესი კონდენსატორი შედგება ორი ელექტროდისგან, რომლებიც ერთმანეთისგან დიელექტრიკის თხელი ფენითაა გამოყოფილი.

სურ. 2.11. კონდენსატორის აღნიშვნა სქემაზე

კონდენსატორის ძირითად მახასიათებელს მისი ელექტრული ტევადობა წარმოადგენს. ეს მახასიათებელი გვიჩვენებს, თუ რა რაოდენობის მუხტს დაიგროვებს კონდენსატორი მისი 1 ვოლტ ძაბვამდე დამუხტვისას. რაც მეტია

კონდენსატორის ტევადობა, მით მეტ მუხტს იგროვებს იგი ერთი და იგივე ძაბვამდე დამუხტვის დროს:

$$Q = C * V$$

C-კონდენსატორის ტევადობაა, V-ძაბვა, Q - დაგროვილი მუხტი.

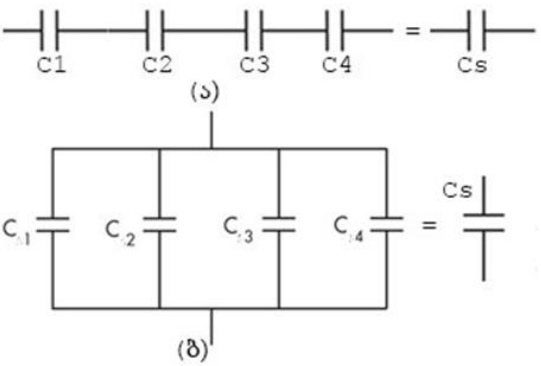
კონდენსატორის ელექტრული ტევადობა იზომება ფარადებში. 1 ფარადი ისეთი კონდენსატორის ტევადობაა, რომელიც 1 ვოლტამდე დაიმუხტება, თუ მას 1 კულონ მუხტს გადავცემთ. 1 ფარადი ძალიან დიდი სიდიდეა. პრაქტიკაში იყენებენ უფრო მცირე სიდიდეებს.

1 მიკროფარადი (1მკფ) = 0.000001ფ

1 ნანოფარადი = 0,001მკფ,

1 პიკოფარადი = 0,000000მკფ.

ზოგჯერ სქემებში გამოიყენება კონდენსატორების მიმდევრობითი და პარალელური შეერთება.



სურ 2.12. (ა) მიმდევრობითი და (ბ) პარალელური შეერთება

მიმდევრობითი შეერთებისას ეკვივალენტური ტევადობა $1/Cs = 1/C1 + 1/C2 + 1/C3 + 1/C4$, თუ მიმდევრობით ჩართული კონდენსატორების რაოდენობა n-ის ტოლია, მაშინ ფორმულა მიიღებს სახეს $1/Cs = 1/C1 + 1/C2 + \dots + 1/Cn$.

პარალელური შეერთებისას ეკვივალენტური ტევადობის გამოთვლა შემდეგი ფორმულით ხდება $Cs = C1 + C2 + C3 + C4$; n რაოდენობის კონდენსატორებისთვის $Cs = C1 + C2 + \dots + Cn$.

კვების წყაროსთან მიერთებისას კონდენსატორის ფირფიტებზე მუხტები გროვდება. ეს წრედში დენის წარმოქმნას იწვევს. კონდენსატორის დამუხტვის დრო დამოკიდებულია მისი ტევადობის სიდიდეზე. რაც მეტია ტევადობა, მით მეტია დამუხტვის დრო. განმუხტვის დრო დამოკიდებულია მიერთებული დატვირთვის სიდიდეზე. რაც მეტი იქნება R წინაღობა, მით უფრო დიდი იქნება განმუხტვის დრო.

დიდი ელექტრული ტევადობით გამოირჩევა ე.წ. ელექტროლიტური კონდენსატორები. განსხვავებით მცირე ტევადობის კერამიკული კონდენსატორებისგან, ისინი პოლარიზებულნი არიან და მათი ჩართვა სქემაში პოლარობის დაცვით უნდა მოხდეს. ასეთი ტიპის კონდენსატორის სქემაზე აღნიშვნისას ამატებენ პლუს ნიშანს.

სურ.2.13. ელექტროლიტური კონდენსატორის აღნიშვნა სქემაზე

კონდენსატორის მეორე მნიშვნელოვანი მახასიათებელია მუშა ძაბვა. კონდენსატორის შერჩევისას აუცილებლად უნდა იქნეს გათვალისწინებული ეს პარამეტრი, რადგან კონდენსატორის ჩართვა უფრო დიდი ძაბვის წრედში, მისი მწყობრიდან გამოსვლას გამოიწვევს.

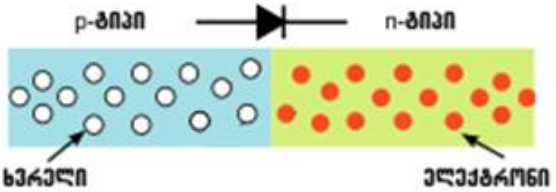
სურ. 2.14. კონდენსატორები

დიოდი და მისი ნაირსახეობები

დიოდი ეწოდება ორელექტროდიან ელექტრონულ ელემენტს, რომელიც დენს ერთი მიმართულებით, ანოდიდან კათოდისკენ, ატარებს. ანოდი მიერთებულია დენის წყაროს დადებით პოლუსთან, კათოდი - უარყოფით პოლუსთან. თანამედროვე ტექნიკაში, ძირითადად, ნახევარგამტარულ დიოდებს იყენებენ.

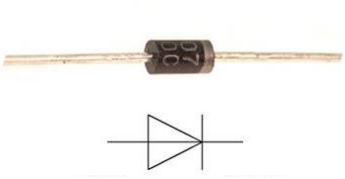
გამტარებლობის შეცვლის მიზნით, ნახევარგამტარებს სხვადასხვა ნივთიერების მინარევებს უმატებენ. ამ პროცესს ლეგირება ეწოდება. ლეგირების შედეგად, ნახევარგამტარში

დენის გატარებისას, მუხტის გადამტანები შეიძლება იყვნენ უარყოფითი ნაწილაკები – ელექტრონები, ან დადებითი ნაწილაკები - ხვრელები. პირველ შემთხვევაში, ნახევარგამტარს N ტიპის ნახევარგამტარი ეწოდება, მეორე შემთხვევაში – P ტიპის. თუ ნახევარგამტარის კრისტალის ერთ ნაწილს ისეთ ლეგირებას გავუკეთებთ, რომ იგი P-ტიპის გახდება, ხოლო მეორე ნაწილს კი ისეთს, რომ იგი N-ტიპის ნახევარგამტარად იქცეს, მივიღებთ ნახევარგამტარულ ელემენტს, რომელსაც ნახევარგამტარული დიოდი ეწოდება.



სურ. 2.15. ნახევარგამტარული დიოდის სტრუქტურა

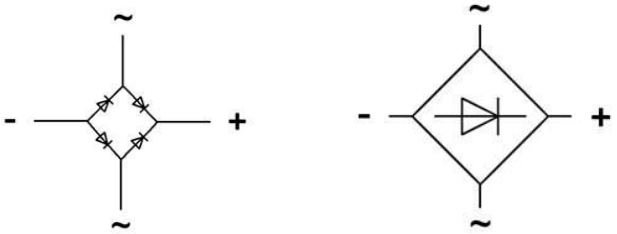
P და N ნაწილების გამყოფ არეს PN გადასასვლელი ეწოდება. დიოდის თვისებები დამოკიდებულია PN გადასასვლელის სისქეზე, მალეგირებელი ნივთიერებების და თვით საწყისი ნახევარგამტარის გვაროვნებაზე. ამ პარამეტრების შერჩევით ხდება სრულიად განსხვავებული თვისებებისა და დანიშნულების მქონე დიოდების დამზადება.



სურ. 2.16. დიოდი და მისი აღნიშვნა სქემაზე

სქემაზე სამკუთხედით აღინიშნება ანოდი (+), ვერტიკალური ხაზით - კათოდი (-).

გამმართველი დიოდები განკუთვნილია ცვლადი დენის მუდმივში გარდასაქმნელად. ამისათვის გამოიყენება დიოდური ბოგირი. დიოდური ბოგირი ერთმანეთთან მიმდევრობით ჩართულ 4 დიოდს წარმოადგენს. ორი დიოდი შემხვედრი მიმართულებითაა ჩართული. (სურ.2.17).

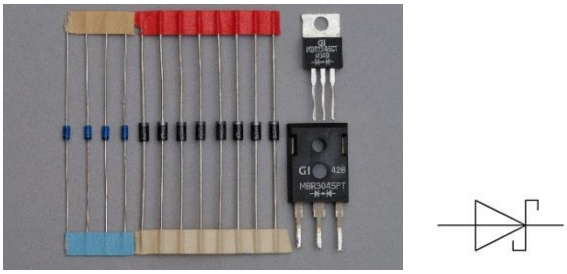


სურ. 2.17. დიოდური ბოგირის სქემაზე აღნიშვნის ვარიანტები

დიოდური ბოგირი გამოიყენება რადიოაპარატურის კვებისთვის, კვების ბლოკებსა და დამმუხტავ მოწყობილობებში. მისი დამზადება შეიძლება ოთხი ერთნაირი დიოდისგან, ასევე შესაძლებელია გამზადებული დიოდური ბოგირის შეძენაც (სურ. 2.18)

სურ. 2.18. დიოდური ბოგირი

დიოდის კიდევ ერთი ნაირსახეობაა შოტკის დიოდი. მას ძალიან დაბალი ძაბვის ვარდნა აქვს და ჩვეულებრივ დიოდებთან შედარებით დიდი სწრაფქმედებით გამოირჩევა.

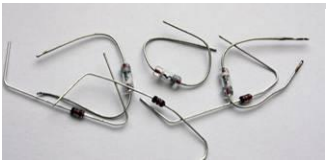


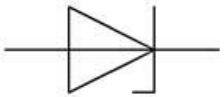
სურ. 2.19. შოტკის დიოდი და მისი აღნიშვნა სქემაზე

სქემაში შოტკის დიოდის ნაცვლად ჩვეულებრივი დიოდის ჩასმა რეკომენდებული არ არის. ჩვეულებრივი დიოდი შეიძლება ადვილად გამოვიდეს მწყობრიდან.

ზენერის დიოდი (სტაბილიტრონი), ჩვეულებრივი დიოდისგან განსხვავებით, წრედში ჩართვისას ახორციელებს მასზე მოდებული ძაბვის სტაბილიზაციას – ძაბვა თითქმის არ არის დამოკიდებული გამავალ დენზე; სტაბილიტრონი ხელს უშლის ძაბვის ზრდას განსაზღვრულ ზღურბლს ზემოთ, სქემის კონკრეტულ უბანზე; შეუძლია შეასრულოს

როგორც დამცავი, ასევე შემზღუდავი ფუნქციები, მუშაობს მხოლოდ მუდმივი დენის წრედში. სტაბილიტრონის მიერთებისას პოლარობის დაცვა აუცილებელია.





სურ. 2.20. სტაბილიტრონი და მისი აღნიშვნა სქემაზე

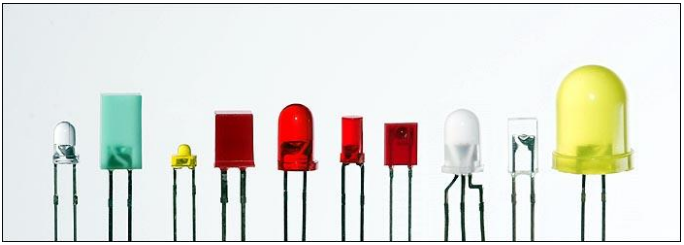
შესაძლებელია ასევე ისეთი დიოდის დამზადება, რომელიც გამტარობას სინათლის ზემოქმედებით იცვლის, ასეთ დიოდებს ფოტოდიოდები ეწოდება.

სურ. 2.21. ფოტოდიოდი და მისი აღნიშვნა სქემაზე

დიოდის კიდევ ერთი ნაირსახეობა - შუქდიოდი კი პირიქით, მასში დენის გატარებისას ხილულ სინათლეს ასხივებს; ქვემოთ სწორედ მათ განვიხილავთ დაწვრილებით.

შუქდიოდი

შუქდიოდები ძალიან გავრცელებული და სასარგებლო კომპონენტებია, რომლებიც ელექტრულ დენს სინათლედ გარდაქმნიან. მათ სხვადასხვა ფორმა, ზომა და ფერი აქვთ.



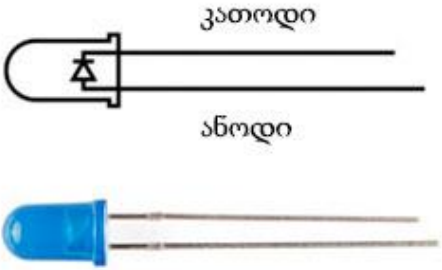
სურ. 2.22. შუქდიოდები

შუქდიოდი მუშაობისთვის დიდ დენის ძალას არ მოითხოვს, საკმარისია მისი მნიშვნელობა დაახლოებით 10 მილიამპერამდე იყოს. დიდი დენის შემთხვევაში, შუქ-დიოდი შეიძლება გადაიწვას. დენის სიდიდის შესამ-ცირებლად, დენის წყაროსა და შუქდიოდს შორის რეზისტორს ათავსებენ. დენი თავისუფლად მიედინება გამტარებში, მაგრამ როდესაც მას გზაზე ხვდება რეზისტორი, გამავალი დენის სიდიდე მცირდება. ელექტროენერგიის ნაწილი გარდაიქმნება სითბურ ენერგიაში და გაიფანტება რეზისტორის მიერ. შუქდიოდების უმეტესობის ჩართვისთვის საჭიროა სულ მცირე 1,5V ძაბვა (დამოკიდებულია შუქდიოდის ტიპზე), ამ დროს ის იწყებს დენის გატარებას და ნათებას.

შუქდიოდის სქემაში ჩართვა გარკვეულ ყურადღებას მოითხოვს, რადგანაც ამ დროს აუცილებელია პოლარობის დაცვა. შუქდიოდი დენს მხოლოდ განსაზღვრული მიმართულებით ატარებს. დენი შედის ანოდიდან (პლუსი) და გადის კათოდის (მინუსი) გავლით (სურ. 2.23).

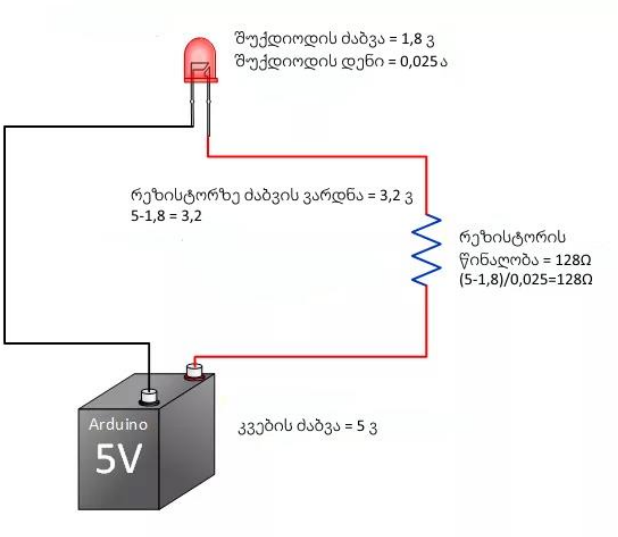
სურ. 2.23. დენის მიმართულება შუქდიოდში

შუქდიოდის კონსტრუქცია საშუალებას იძლევა განისაზღვროს, რომელი კონტაქტია ანოდი და რომელი - კათოდი. ანოდთან დაკავშირებული საკონტაქტო გამომყვანი უფრო გრძელია, ვიდრე კათოდთან დაკავშირებული (სურ. 2.24).



სურ. 2.24. შუქდიოდის კონსტრუქცია

შუქდიოდების გამოყენებისას, არ უნდა დაგვავიწყდეს მუშა ძაბვისა და დენის ძალის შესახებ. მაგალითად, წითელი ფერის ტიპიური შუქდიოდები მოითხოვენ 1.8 V ძაბვას და დენს 25mA-მდე. ეს გარკვეულ პრობლემას ქმნის, რადგანაც არდუინოს გამოსასვლელებზე ძაბვა არის 5 ვოლტი და დენიც უფრო დიდია. როგორც ზემოთ უკვე ავღნიშნეთ, ასეთ შემთხვევებში ვიყენებთ რეზისტორს, იმისათვის რომ შევამციროთ შუქდიოდში გამავალი დენის ძალა. ისმის კითხვა: როგორი ნომინალის რეზისტორი უნდა ავირჩიოთ? ამაში ომის კანონი დაგვეხმარება.



სურ.2.25. სქემა რეზისტორითა და შუქდიოდით

შემზღუდავი რეზისტორის წინაღობის გამოსათვლელად გამოიყენება ფორმულა:

$$R = (V_s - V_f)/I,$$

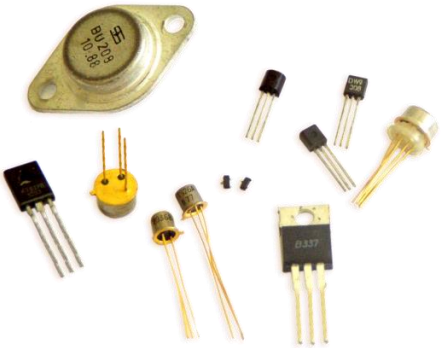
სადაც V_s არის კვების ძაბვა (არდუინოს შემთხვევაში, ეს არის 5 ვოლტი), V_f - ძაბვის ვარდნაა (მუშა ძაბვა, მაგ., 1.8V) შუქდიოდზე, I - შუქდიოდისთვის მოთხოვნილი დენის ძალაა (მაგ. 25 mA). გაანგარიშებისთვის დაგვჭირდება დენის ძალის მნიშვნელობის გამოსახვა ამპერებში: 25 მილიამპერი (mA)=0,025ამპერი (A).

ვისარგებლოთ რეზისტორის წინაღობის ზემოთ მოცემული ფორმულით. ჩავსვათ ძაბვების და დენის ძალის მნიშვნელობები: (5-1,8)/0,025=128Ω (დავამრგვალოთ მეტობით სტანდარტულ მნიშვნელობამდე 220Ω). წინაღობა გაანგარიშებულია იმ შემთხვევისთვის, როცა დენის ძალა 25 მილიამპერია.

ტრანზისტორი

ტრანზისტორი წარმოადგენს ნახევარგამტარული მასალისგან დამზადებულ რადიოელექტრონულ კომპონენტს. მას მცირე შესასვლელი სიგნალით გამოსასვლელ წრედში დიდი დენის მართვა შეუძლია. ტრანზისტორის სწორედ ეს თვისება საშუალებას იძლევა ის გამოყენებული იქნას გაძლიერების, კომუტაციისა და ელექტრული სიგნალების გარდასაქმნელად. დღეისათვის ტრანზისტორი წარმოადგენს

ელექტრონული მოწყობილობებისა და ინტეგრალური სქემების სქემოტექნიკურ საფუძველს.



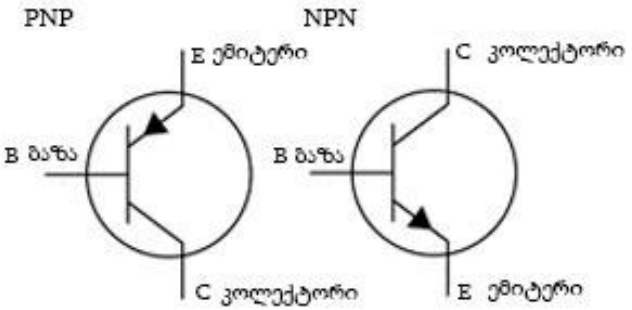
სურ. 2.26. ტრანზისტორები

შუქდიოდის მსგავსად, ტრანზისტორის გამოსასვლელებსაც თავისი უნიკალური ფუნქცია აქვს, სქემაში მათი ჩართვა სტრუქტურის გათვალისწინებით უნდა მოხდეს.

სტრუქტურის, მოქმედების პრინციპისა და პარამეტრების მიხედვით ტრანზისტორები ორ კლასად იყოფა: ბიპოლარული და უნიპოლარული.

ბიპოლარული ტრანზისტორი, ისევე როგორც ნახევარგამტარული დიოდი, ნახევარგამტარულ კრისტალს წარმოადგენს, რომელიც PN გადასასვლელებს შეიცავს. დიოდისგან განსხვავებით, ტრანზისტორში ასეთი გადასასვლელი ორია. მისი სტრუქტურა შეძლება იყოს NPN ან PNP. NPN ტიპის ტრანზისტორში ელექტრული დენი მიედინება კოლექტორიდან ემიტერისკენ. PNP ტიპის ტრანზისტორებში

ელექტრული დენის დინების მიმართულება არის პირიქით - ემიტერიდან კოლექტორისკენ (სურ. 2.27).



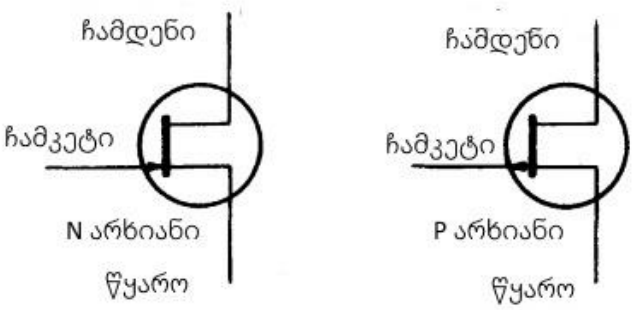
სურ. 2.27. PNP და NPN ტიპის ტრანზისტორები

გამტარობის თვალსაზრისით, NPN ტრანზისტორი შემდეგნაირად მუშაობს: ბაზა-ემიტერს შორის გაჩენილი მცირე დენი იწვევს კოლექტორ-ემიტერს შორის დიდი დენის გავლას. PNP ტრანზისტორი პირიქით მუშაობს: ემიტერ-ბაზას შორის გაჩენილი მცირე დენი იწვევს ემიტერ-კოლექტორს შორის დიდი დენის გავლას.

ტრანზისტორების კიდევ ერთ კატეგორიას წარმოადგენს ველით მართული ტრანზისტორი (Field Effect Transistor, FET). ამ ტიპის ტრანზისტორებს ხშირად უნიპოლარულსაც უწოდებენ.

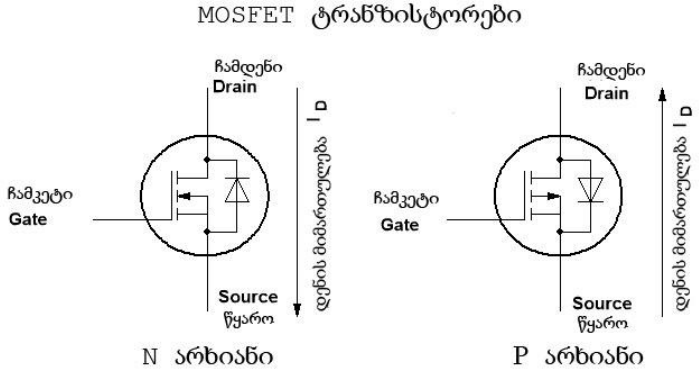
ველით მართული ტრანზისტორი აქტიურ ნახევარგამტარულ მოწყობილობას წარმოადგენს, რომელშიც გამოსასვლელი დენის მართვა ელექტრული ველის დახმარებით ხორციელდება, განსხვავებით ბიპოლარული ტრან-

ზისტორისგან, რომელშიც გამოსასვლელი დენი შესასვლელზე არსებული დენით იმართება.



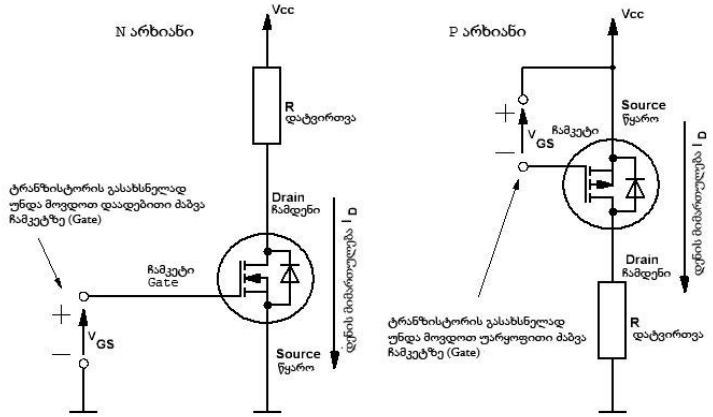
სურ. 2.28. ველით მართული ტრანზისტორები

უნიპოლარული ტრანზისტორების კიდევ ერთ კატეგორიას წარმოადგენს ლითონ-ოქსიდის-ნახევარგამტარული ველით მართული ტრანზისტორი (metal-oxide-semiconductor field-effect, MOSFET).



სურ. 2.29. N და P არხიანი ველის ტრანზისტორების სქემატური გამოსახულება

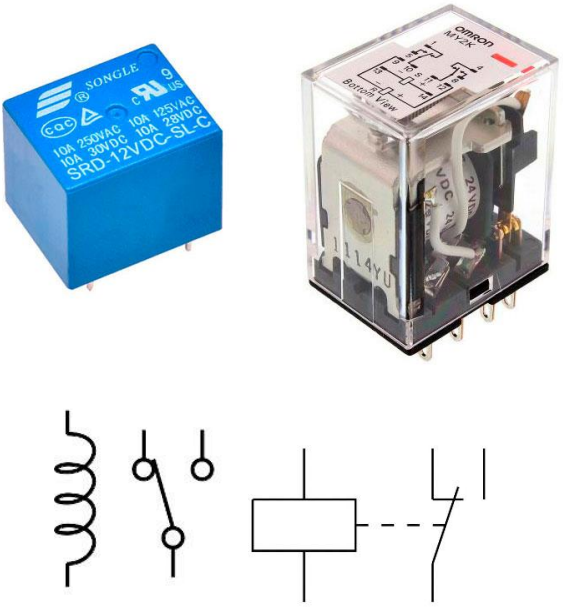
არსებობს ორი სახის MOSFET ტრანზისტორი, N და P არხიანი. ამ ტრანზისტორის გამომყვანების სახელწოდებებია ჩამკეტი (Gate), ჩამდენი (Drain) და წყარო (Source). ველის ტრანზისტორის მთავარი უპირატესობა ბიპოლარულ ტრანზისტორებთან შედარებით ისაა, რომ ბიპოლარულ ტრანზისტორებში აუცილებელია ბაზა-ემიტერს შორის დენი გავიდეს, ხოლო ველის ტრანზისტორის გასახსნელად მხოლოდ ძაბვაა საჭირო. სწორედ ამ მიზნით MOSFET ტრანზისტორში ჩამკეტი დანარჩენი გამომყვანებისგან გამოყოფილია მეტალის ჟანგით, რაც კარგი იზოლატორია და დენს არ ატარებს. ამ მიზეზით იზოგება სიმძლავრე წრედში. N არხიანი ველის ტრანზისტორის გასახსნელად საჭიროა ჩამკეტზე მოვდოთ დადებითი პოტენციალი ხოლო P არხიანზე კი - უარყოფითი პოტენციალი.



სურ.2.30. MOSFET ტრანზისტორების ჩართვის სქემა

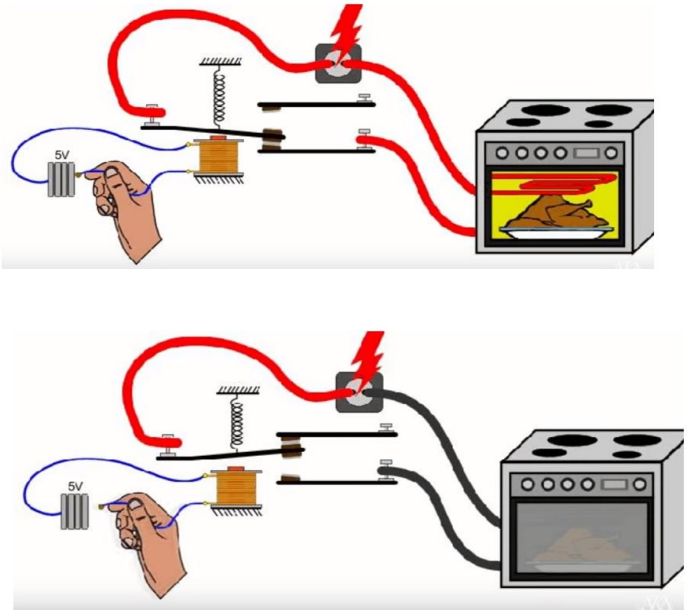
ელექტრომაგნიტური რელე

ელექტრომაგნიტური რელეს მმართველი სქემისგან სრული ელექტრონული იზოლაცია, არდუინოს საშუალებას აძლევს ჩართოს ან გამორთოს მაღალ დენზე და ძაბვაზე მომუშავე მოწყობილობები. იზოლაცია აუცილებელია იმისათვის, რომ სქემა დიდი ძაბვისგან დაიცვას. რელეს შიგნით მოთავსებულია კონტაქტები (მათი გადართვა შეიძლება მექანიკურად) და დაბალ ძაბვაზე მომუშავე კოჭა.



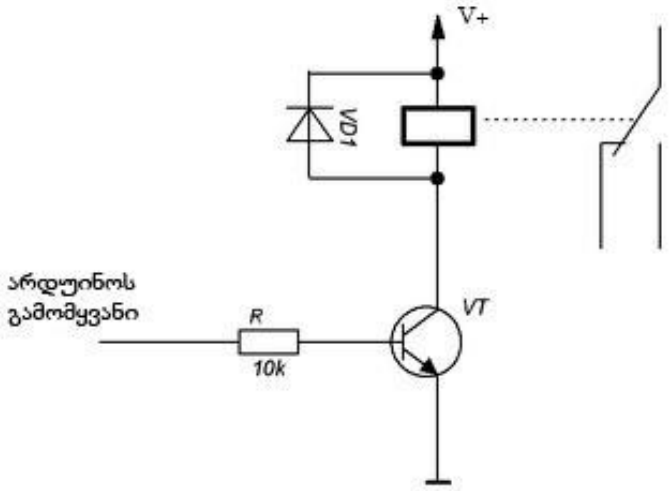
სურ. 2.31. ელექტრომაგნიტური რელე

როდესაც კოჭას მიეწოდება დენი, ის იწყებს მოქმედებას, როგორც ელექტრომაგნიტი და მიიზიდავს ლითონის ფირფიტას, რომელიც კრავს კონტაქტებს. როდესაც ელექტრომაგნიტი გამორთულია, მაშინ ლითონის ფირფიტა უბრუნდება საწყის მდგომარეობას და გათიშავს კონტაქტებს. ეს საშუალებას იძლევა კონტაქტების მართვა განხორციელდეს კოჭაზე ძაბვის მიწოდებისა და მოხსნის გზით. (სურ. 2.32)



სურ. 2.32. რელეს მუშაობის პრინციპი

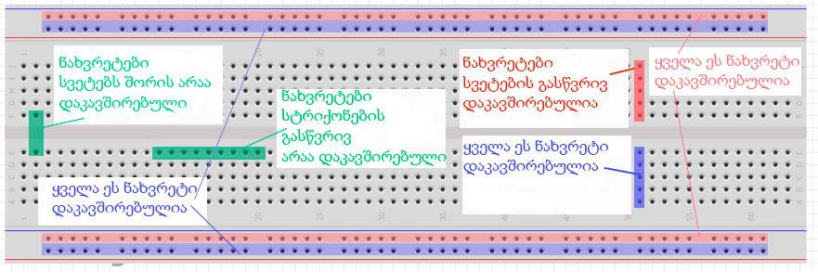
რელეს ხვია წარმოადგენს ინდუქტიურობას. რელეს ჩართვა-გამორთვისას მის ხვიაში თვითინდუქციის გამო შეიძლება წარმოიქმნას დიდი პოტენციალთა სხვაობა, რაც გამოიწვევს არდუინოს ან რელეს მართვის სხვა მოწყობილობის შესაბამისი გამომყვანის დაზიანებას. ამის თავიდან ასაცილებლად იყენებენ დიოდს, რომელიც რელეს გამომყვანების პარალელურად არის ჩართული.



სურ. 2. 33. დამცავი დიოდის მიერთება რელეს ხვიებთან

სამაკეტო დაფა

სქემის ასაწყობად ძალიან მოხერხებულია სამაკეტო დაფის გამოყენება. სამაკეტო დაფა შეიცავს ნახვრეტებს. შუქდიოდი, რეზისტორი და სხვა ელექტრონული კომპონენტები ამ ნახვრეტებში მაგრდება. სამაკეტო დაფაზე ნახვრეტები ერთმანეთთან გარკვეული წესების დაცვითაა დაკავშირებული. დაკავშირების წესები ნაჩვენებია ქვემოთ მოცემულ სურათზე:



სურ. 2.34. სამაკეტო დაფა და მასზე კომპონენტების დაკავშირების წესები

როგორც სურ. 2.34-ზეა ნაჩვენები, ნახვრეტები, რომლებიც მოთავსებულია ერთ სვეტში, ერთმანეთთან არის დაკავშირებული. დავუშვათ, გვინდა რეზისტორი დავაკავშიროთ შუქდიოდთან. ამისათვის, რეზისტორის საკონტაქტო გამომყვანი უნდა ჩავამაგროთ რომელიმე ნახვრეტში და იმავე სვეტში, სხვა ნახვრეტში ჩავამაგროთ შუქდიოდის შესაბამისი გამომყვანი. საერთო სვეტში კომპონენტების ჩამაგრებით ჩვენ ისინი ერთმანეთთან დავაკავშირეთ.

ერთმანეთს არ უკავშირდება ნახვრეტები, რომლებიც ერთ რიგშია მოთავსებული. ეს არ ეხება სამაკეტო დაფაზე არსებულ უკიდურეს ზედა და უკიდურეს ქვედა მხარეს არსებულ რიგებს. არდუინოს ზედა და ქვედა კიდეებზე ერთი რიგის გასწვრივ არსებული ყველა ნახვრეტი ერთმანეთს უკავშირდება. როგორც სურ. 2.34-დან ჩანს, სამაკეტო დაფის ზედა და ქვედა კიდეებზე სულ ოთხი რიგი გვაქვს. დაფის ზედა და ქვედა კიდეებზე არსებული რიგებიდან ზოგი მონიშნულია „+“ ნიშნით, ზოგი - „-“-ით. ეს აღნიშვნები პირობითია. რიგი, რომელსაც აწერია „+“, დადებითი არ არის იმის გამო, რომ „+“ აწერია, დადებითი ხდება იმ შემთხვევაში, თუ მიეწოდება ვთქვათ, 5V ძაბვა. რიგი, რომელსაც აწერია „-“ (მიწა), უარყოფითი ხდება, თუ ის მიწასთან იქნება მიერთებული.

არდუინოს ციფრული და ანალოგური საკონტაქტო გამომყვანები

არდუინოს ყველა ციფრული საკონტაქტო გამომყვანი შესაძლებელია გამოყენებული იყოს ციფრული შესასვლელების და გამოსასვლელების სახით. არდუინოს დაფების უმეტესობას აქვს მხოლოდ ანალოგური საკონტაქტო შესასვლელები განსხვავებით ციფრული გამომყვანებისგან, რომლებიც შეიძლება კონფიგურირებული იყოს როგორც შესასვლელის, ასევე გამოსასვლელის სახით. ანალოგური გამო-

სასვლელების ემულირება შესაძლებელია განივ-იმპულსური მოდულაციის საშუალებით.

არდუინოს ყველა ციფრული კონტაქტი, ნაგულისხმევი წესის თანახმად, კონფიგურირებულია როგორც შესასვლელი. თუ არდუინოს ციფრული საკონტაქტო გამომყვანის გამოყენება აუცილებელია გამოსასვლელის სახით, საჭიროა მისი შესაბამისად კონფიგურირება.

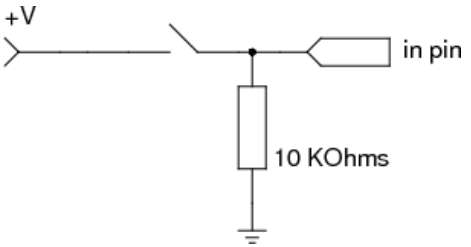
ციფრულ შესასვლელზე ინფორმაციის სწორად წაკითხვისთვის რეკომენდებულია მომჭიმავი (Pull-up და Pull-down) რეზისტორების გამოყენება. რეზისტორების მნიშვნელობის კარგად გასაგებად განვიხილოთ სურ.2.35.

სურ.2.35. ღია წრედი

როდესაც წრედი შეკრული არ არის, იმ საკონტაქტო გამომყვანზე, რომელზეც არაფერია მიერთებული, გვექნება ძაბვის არა ნულოვანი მნიშვნელობა, როგორც წესით არის მოსალოდნელი, არამედ შემთხვევითი სიდიდის მნიშვნელობა, გამოწვეული ელექტრომაგნიტური ხმაურით ან ახლომდებარე საკონტაქტო გამომყვანების ტევადური ურთიერთკავშირებით. თუ შესასვლელ საკონტაქტო გამომყვანზე სიგნალი არ არის, ასეთ შემთხვევაში უმჯობესია, ის გადავიყვანოთ რაიმე ცნობილ მდგომარეობაში. ამის

გაკეთება შესაძლებელია 10kΩ მომჭიმავი (Pull-up ან Pull-down) რეზისტორის საშუალებით, რომელსაც ვაერთებთ კვების წყაროსთან (+5V) ან მიწასთან.

დავუშვათ, გვინდა, რომ როდესაც წრედი არ არის შეკრული, შესასვლელზე ძაბვის არარსებობა დაფიქსირდეს. ამისათვის საჭიროა მომჭიმავი (Pull-down) რეზისტორის დაყენება (სურ. 2.36).

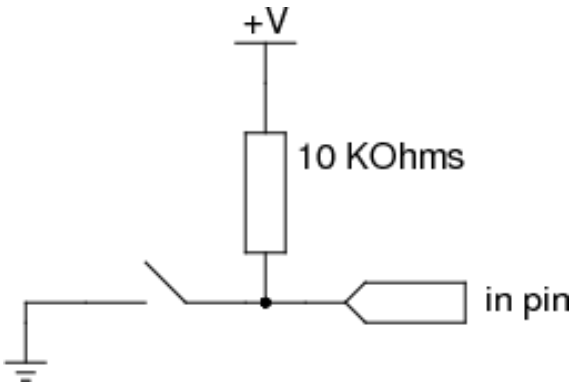


სურ. 2.36. მომჭიმავი (Pull down) რეზისტორის გამოყენება

როდესაც წრედი შეკრული არ არის (სურ. 2.36) და შესასვლელი საკონტაქტო გამომყვანი მიერთებულია მიწასთან 10kΩ მომჭიმავი (pull-down) რეზისტორის მეშვეობით, რეზისტორში გაივლის გაჟონვის დენი და შესასვლელ კონტაქტზე გვექნება ლოგიკური ნული (Low). მომჭიმავი (Pull-down) რეზისტორი, როგორც წესი, წარმოადგენს დიდი წინაღობის მქონე რეზისტორს (10kΩ და მეტი). წრედის შეკვრის შემთხვევაში, შესასვლელი კონტაქტი პირდაპირ აღმოჩნდება კვებასთან (+5V) შეერთებული. ახლა

უკვე დენის გატარების ორი შესაძლო გზა არსებობს: დიდი წინაღობის გავლით მიწისკენ და პრაქტიკულად ნულოვანი წინაღობის გავლით არდუინოს შესასვლელისკენ. ომის კანონის თანახმად, დენის დიდი ნაწილი გაივლის იქით, საითაც წინაღობა ნაკლებია. დიდი წინაღობის მქონე რეზისტორი დენის დიდ ნაწილს არ აძლევს საშუალებას გადავიდეს მიწაში: სიგნალი წავა არდუინოს შესასვლელი კონტაქტისკენ, სადაც გვექნება ლოგიკური ერთი (High). რეზისტორის მცირე წინაღობის შემთხვევაში კი, ჩაკეტილ წრედში მოკლე ჩართვა მოხდებოდა.

ანალოგიურად ზემოთ განხილული შემთხვევისა, მომჭიმავი (Pull up) რეზისტორი, შესასვლელზე უზრუნ-ველყოფს ლოგიკური ერთიანის არსებობას, სანამ წრედი ღიაა.



სურ. 2.37. მომჭიმავი (Pull up) რეზისტორის გამოყენება

ამ შემთხვევაშიც, გამოიყენება მაღალი ნომინალის მქონე რეზისტორები (10kΩ და მეტი), იმისათვის რათა მაქსიმალურად შემცირდეს ენერგიის დანაკარგი შეკრულ წრედში და თავიდან იქნეს აცილებული მოკლე ჩართვა ღია წრედში.

შესაძლებელია როგორც მიკროკონტროლერში ჩაშენებული, ასევე გარე მომჭიმავი რეზისტორების გამოყენება. ჩაშენებული მომჭიმავი რეზისტორების აქტივაცია ხდება ბრძანებით INPUT_PULLUP. მოჭიმვის წინაღობის სიდიდე გამოყენებულ მიკროკონტროლერზეა დამოკიდებული. ასევე უნდა გავითვალისწინოთ ისიც რომ, მიკროკონტროლერს აქვს მხოლოდ Pull up მომჭიმავი რეზისტორები.

მეცამეტე ციფრული საკონტაქტო გამომყვანის გამოყენება ციფრული შესასვლელის სახით, გაცილებით რთულია სხვა ციფრულ საკონტაქტო გამომყვანებთან შედარებით, რადგანაც ის დაკავშირებულია არდუინოს დაფაზე არსებულ შუქდიოდთან და რეზისტორთან. თუ ჩვენ აუცილებლად გვჭირდება გამოვიყენოთ მეცამეტე საკონტაქტო გამომყვანი, როგორც ციფრული შესასვლელი, pinMode()-ში უნდა მივუთითოთ INPUT და გამოვიყენოთ გარე, მიწისკენ მომჭიმავი რეზისტორი.

საკონტაქტო გამომყვანები, რომლებიც კონფიგურირებულია, როგორც გამოსასვლელები (OUTPUT), იმყოფებიან დაბალ იმპედანსურ მდგომარეობაში. Atmega-ს

საკონტაქტო გამომყვანების დატვირთვის დენი არ უნდა აღემატებოდეს 40 mA-ს.

არდუინოს საკონტაქტო გამომყვანებზე მოკლე ჩართვამ ან მძლავრი მოწყობილობების მიერთების მცდელობამ შეიძლება გამოიწვიოს საკონტაქტო გამომყვანის ან მთლიანად მიკროკონტროლერის დაზიანება. ამის გამო, რეკომენდებულია არდუინოს გამოსასვლელ (OUTPUT) საკონტაქტო გამომყვანებზე მოწყობილობები მიერთებული იყოს 470Ω რეზისტორების გავლით.

არდუინო კითხულობს მის ანალოგურ საკონტაქტო გამომყვანებზე მოდებული ძაბვის მნიშვნელობას. არდუინოს დაფა შეიცავს მრავალარხიან, 10-ბიტიან ანალოგურ ციფრულ გარდამქმნელს. ეს ნიშნავს, რომ შესასვლელი ძაბვების მნიშვნელობებს 0-დან 5V დიაპაზონში შეესაბამება ანალოგურ ციფრული გარდამქმნელის მთელი მნიშვნელობები 0-1023 დიაპაზონში (იმის გამო, რომ 2^{10}=1024-ს). არდუინო Uno-ს შემთხვევაში 5V ვოლტს შეესაბამება 1024 ერთეული, ანუ ყოველ ერთ ერთეულს შეესაბამება დაახლოებით 0.0049 ვოლტი.

ქვემოთ მოცემულია ცხრილი, სადაც ნაჩვენებია საკონტაქტო გამომყვანები, მოქმედი ძაბვა და მაქსიმალური გარჩევადობა არდუინოს ზოგიერთი დაფისთვის.

არდუინოს დაფებისთვის (Uno, Nano, Mini, Mega) შემავალი ანალოგური სიგნალის წასაკითხად დაახლოებით

100 მიკროწამია (0.0001 წმ) საჭირო; წაკითხვის მაქსიმალური სიჩქარეა დაახლოებით 10000 წაკითხვა წამში.

დაფა	მოქმედი ძაბვა (V)	საკონტაქტო გამომყვანი	მაქსიმალური გარჩევადობა (ბიტებში)
Uno	5	A0 – A5	10
Mini, Nano		A0 – A7	
Mega, Mega2560		A0-A14	

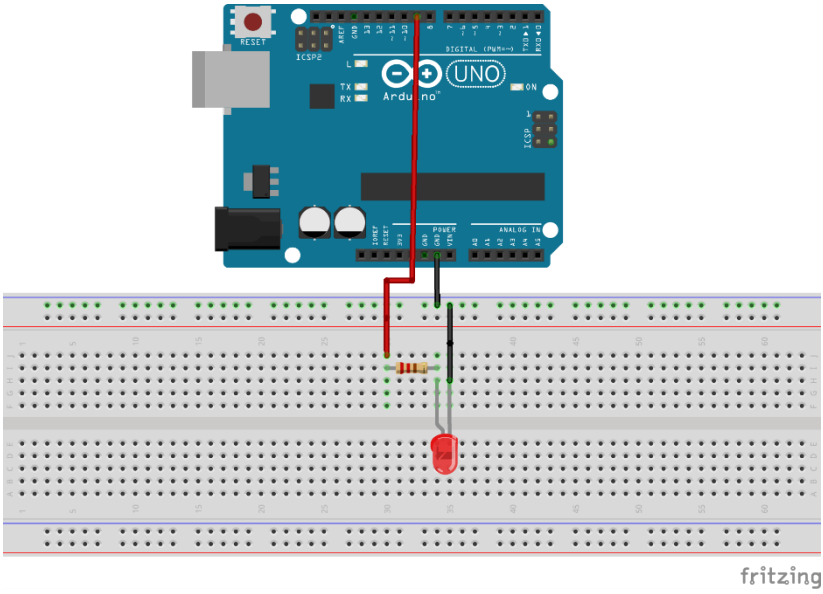
ცხრილი 2.2. არდუინოს დაფების ანალოგური შესასვლელების სპეციფიკაცია

ანალოგურ საკონტაქტო გამომყვანებსაც აქვთ მომჭიმავი რეზისტორები, ისინი მუშაობენ ციფრული საკონტაქტო გამომყვანების მომჭიმავი რეზისტორების მსგავსად. მათი ამოქმედება ხდება ბრძანებით pinMode(An, INPUT_PULLUP); მიაქციეთ ყურადღება იმას, რომ მომჭიმავი რეზისტორის ჩართვა გავლენას მოახდენს analogRead() მნიშვნელობაზე. ანალოგური საკონტაქტო გამომყვანების გამოყენება შეიძლება ციფრული შესასვლელი და გამო-სასვლელი საკონტაქტო გამომყვანების სახითაც.

თავი 3. დაპროგრამება არდუინოს პლატფორმაზე

სამაკეტო დაფაზე შუქდიოდის ჩართვა/გამორთვა

ავაწყოთ ჩვენი პირველი სქემა, რომლის მართვასაც არდუინოს საშუალებით განვახორციელებთ. სქემა მარტივია და შემდეგნაირად გამოიყურება:



სურ. 3.1. სქემის აწყობის შედეგი

ამოცანის მიზანი შუქდიოდის ჩართვა/გამორთვაა. შეგახსენებთ, რომ ჩვენი პირველი სკეტჩი, მე-13 გამომყვანთან დაკავშირებული შუქდიოდის ჩართვა/გამორთვის ამოცანას ასრულებდა. ამჯერად ჩავრთავთ და გამოვრთავთ

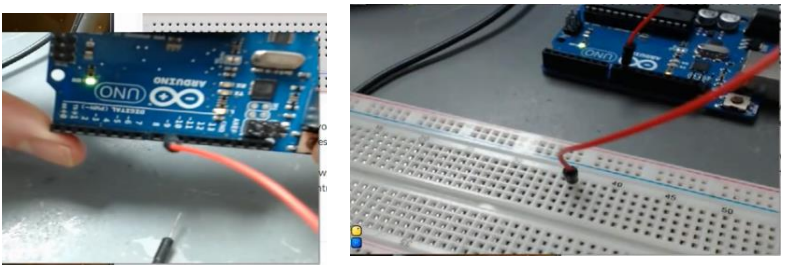
სამაკეტო დაფაზე მოთავსებულ შუქდიოდს. შევხედოთ კიდევ ერთხელ სქემას, რომლის აგებაც გვინდა. რეზისტორი და შუქდიოდი უნდა მოვათავსოთ სამაკეტო დაფაზე. კვების მიწოდება უნდა მოხდეს არდუინოს რომელიმე საკონტაქტო გამომყვანიდან, დაწყებული ნულიდან დამთავრებული მეცამეტეთი. დავუშვათ, ამისთვის ავირჩიეთ მე-9 საკონტაქტო გამომყვანი. კვება შუქდიოდს მიეწოდება რეზისტორის გავლით. ჩვენ ვიყენებთ 220 ომი წინაღობის მქონე რეზისტორს (იხილეთ გვ.52). როგორც ხედავთ, საჭიროა კვების წყარო მივაერთოთ რეზისტორის ერთ რომელიმე საკონტაქტო გამომყვანთან. რეზისტორის ერთ-ერთი საკონტაქტო გამომყვანი და გამტარი, რომელიც გამოდის მეცხრე საკონტაქტო გამომყვანიდან, დაკავშირებულია, რადგანაც ისინი ერთსა და იმავე სვეტშია მოთავსებული (სურ. 3.1).

რეზისტორის მეორე საკონტაქტო გამომყვანი და შუქდიოდის შესაბამისი გამომყვანი (ანოდი) ერთმანეთთანაა დაკავშირებული. შუქდიოდის მეორე გამომყვანი (კათოდი) უკავშირდება მიწას.

რეზისტორისთვის მნიშვნელობა არ აქვს შეერთების მიმართულებას ანუ სულერთია, რომელი გამომყვანი იქნება მიერთებული კვებასთან და რომელი - შუქდიოდთან. შეერთების მიმართულებას მნიშვნელობა აქვს შუქდიოდისთვის. არასწორად შეერთების შემთხვევაში, ის არ შეასრულებს თავის ფუნქციას. თუ დავაკვირდებით შუქდიოდს, ადვილად შევამჩნევთ, რომ მისი ერთი გამომყვანი უფრო

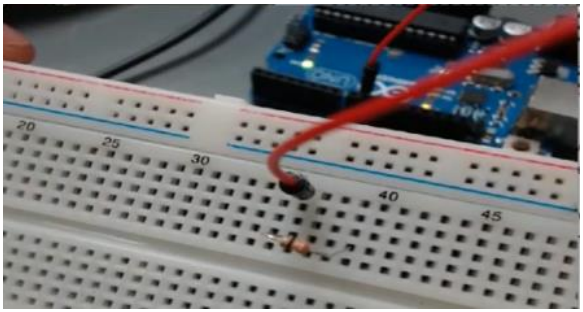
გრძელია მეორეზე. გრძელ გამომყვანს ეწოდება ანოდი, მოკლეს - კათოდი. ანოდი წრედის დადებით მხარეს უნდა იყოს მიერთებული. ჩვენს სქემაზე ანოდი მიერთებულია რეზისტორთან, კათოდი - მიწასთან.

დავიწყოთ სქემის აწყობა. არდუინოს მე-9 საკონტაქტო გამომყვანზე მივაერთოთ გამტარი. გამტარის მეორე ბოლო ჩავამაგროთ სამაკეტო დაფის რომელიმე ჩვენს მიერ შერჩეულ სვეტში (სურ.3.2).



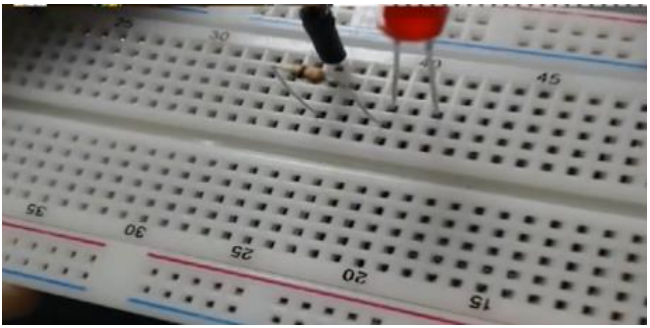
სურ. 3.2. სქემის აწყობის პროცესი

როგორც სურ. 3.3-დან ჩანს, რეზისტორის საკონტაქტო გამომყვანი იმავე სვეტშია მოთავსებული, რომელშიც გამტარი.



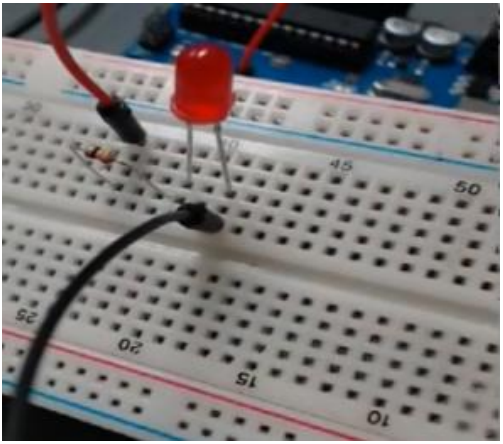
სურ. 3.3. სქემის აწყობის პროცესი

რეზისტორი უნდა მივაერთოთ შუქდიოდთან. ამისათვის რეზისტორის მეორე გამომყვანი უნდა მიუერთდეს შუქდიოდის ანოდს. უფრო მარტივად რომ ვთქვათ, რეზისტორის საკონტაქტო გამომყვანი უნდა იყოს იმავე სვეტში, რომელშიც ანოდი.



სურ. 3.4. სქემის აწყობის პროცესი

გამტარი, რომელიც მიერთებულია მიწასთან, უნდა იყოს იმავე სვეტში, რომელშიც კათოდი.



სურ. 3.5. სქემის აწყობის პროცესი

ამით ჩვენ დავასრულეთ სქემის აგება სამაკეტო დაფაზე. ახლა უკვე შეგვიძლია შევუდგეთ პროგრამის დაწერას.

პროგრამაში გამოვაცხადოთ გლობალური ცვლადი redLED. ცვლადი მთელი ტიპისაა და გვიჩვენებს იმ საკონტაქტო გამომყვანის ნომერს, რომელთანაც შუქდიოდია მიერთებული.

```
const int redLED=9;
```

გამოვაცხადოთ ასევე ცვლადები onTime და offTime, რომლებსაც მივანიჭებთ შუქდიოდის ჩართულ და გამორთულ მდგომარეობაში ყოფნის დროს (იზომება მილიწამებში). ვთქვათ, გვინდა რომ შუქდიოდი ჩართული იყოს ნახევარი წამის განმავლობაში და გამორთული იყოს ასევე ნახევარი წამის განმავლობაში. ასეთ შემთხვევაში, ვწერთ:

```
int onTime=500;
int offTime=500;
```

ამის შემდეგ გადავდივართ პროგრამის void setup() ნაწილში. განვსაზღვროთ ფუნქცია pinMode():

```
pinMode(redLED,OUTPUT);
```

void loop() ნაწილში ვწერთ digitalWrite() ფუნქციას. თუ გვინდა შუქდიოდი ჩავრთოთ, მაშინ კოდი შემდეგნაირად უნდა დაიწეროს:

```
digitalWrite(redLED,HIGH);
```

პროგრამა მიიღებს ასეთ სახეს:

```
int redLED=9;
```

```
int onTime=500;
int offTime=500;
void setup() {
pinMode(redLED, OUTPUT);
}
void loop() {
  digitalWrite(redLED, HIGH);
}
```

დავაჭიროთ upload ღილაკს. პროგრამის შესრულების შედეგად დავინახავთ, რომ წითელი შუქდიოდი აინთო. იმისათვის, რომ შუქდიოდი გამოვრთოთ, პროგრამის კოდში HIGH უნდა შევცვალოთ LOW-თი:

```
digitalWrite (redLED, LOW);
```

პროგრამაში ასევე შესაძლებელია მივუთითოთ, რამდენ ხანს იყოს ჩართული ან გამორთული შუქდიოდი:

```
digitalWrite(redLED, HIGH);
delay(onTime);
digitalWrite(redLED, LOW);
delay(offTime);
```

პროგრამის ზემოთ ნაჩვენები კოდის ფრაგმენტის მიხედვით, ჩაირთვება შუქდიოდი, ის ჩართულ მდგომარეობაში რჩება 500 მილიწამის განმავლობაში, შემდეგ გამოირთვება და ამ მდგომარეობაში რჩება ისევ 500 მილიწამის განმავლობაში. ეს კოდი მოთავსებულია loop() ფუნქციის შიგნით, რაც იმას ნიშნავს რომ, პროცესი უსასრულოდ, ციკლურად გრძელდება.

პროგრამის კოდი სრული სახით ასე გამოიყურება:

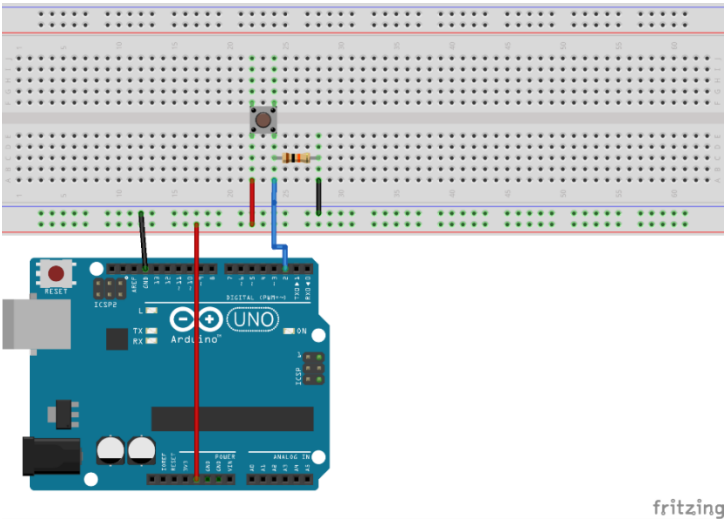
```
const int redLED=9;
int onTime=500;
int offTime=500;
void setup() {
  pinMode(redLED, OUTPUT);
}
void loop() {
  digitalWrite(redLED, HIGH);
  delay(onTime);
  digitalWrite(redLED, LOW);
  delay(offTime);
}
```

პროგრამის კოდში მარტივად შეგვიძლია შევცვალოთ შუქდიოდის გამორთულ და ჩართულ მდგომარეობაში ყოფნის დრო. თუ მივუთითებთ, რომ onTime=100 და offTime=900, შუქდიოდი გამორთული იქნება 900 მილიწამის, ხოლო ჩართული იქნება 100 მილიწამის განმავლობაში.

ახლა ჩავატაროთ ერთი საინტერესო და ლამაზი ექსპერიმენტი. ავიღოთ ყვითელი ფერის შუქდიოდი და ჩავამაგროთ ჩვენს სამაკეტო დაფაზე ისე რომ მისი გამომყვანები იყოს იმავე სვეტებში, რომელშიც წითელი შუქდიოდის ანოდი და კათოდია. ისევე როგორც წითელი შუქდიოდის შემთხვევაში, ყვითელი შუქდიოდის ანოდი უერთდება რეზისტორის გამომყვანს და კათოდი უერთდება მიწას. ამ შემთხვევაში, იციმციმებს ორივე შუქდიოდი, ისინი ერთდროულად იქნებიან ჩართულ ან გამორთულ მდგომარეობაში.

შუქდიოდის მდგომარეობის მართვა ღილაკის საშუალებით

ქვემოთ მოცემულ ამოცანაში ნაჩვენებია ღილაკის გადამრთველის სახით გამოყენება: ყოველთვის, როდესაც დავაჭერთ ღილაკს, შუქდიოდი ინთება (თუ ის გამორთულია) და გამოირთვება (თუ ის ჩართულია).



სურ. 3.6. შუქდიოდის მდგომარეობის მართვა ღილაკის საშუალებით

მნიშვნელოვანია გავიაზროთ სქემაში (სურ. 3.6) რეზისტორის დანიშნულება. სურ. 3.6-ზე მოცემული რეზისტორი წარმოადგენს მომჭიმავ (Pull-down) რეზისტორს (იხ. გვ.63). როდესაც ღილაკზე არ არის დაჭერილი, არდუინოს შესასვლელი საკონტაქტო გამომყვანი მიერთებულია მიწასთან რეზისტორის მეშვეობით, რის გამოც შესასვლელზე

გვექნება ლოგიკური ნული. ღილაკზე დაჭერის (წრედის შეკვრის) შემთხვევაში, შესასვლელი საკონტაქტო გამომყვანი შეერთებული იქნება კვებასთან, რეზისტორის დიდი წინაღობის გამო მასში დენის მხოლოდ მცირე ნაწილი გაივლის და არდუინოს შესასვლელზე გვექნება ლოგიკური ერთი.

პროგრამის კოდი:

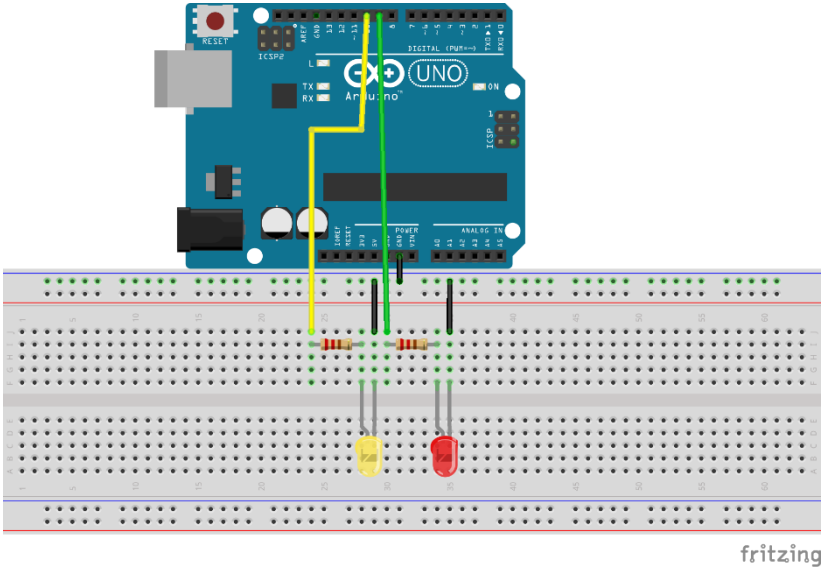
```
//ღილაკთან მიერთებული საკონტაქტო გამომყვანის ნომერი.
const int buttonPin=2;
//შუქდიოდის საკონტაქტო გამომყვანის ნომერი
const int ledPin=13;
//ცვლადი, რომელშიც ინახება ღილაკის მდგომარეობა
int buttonState = 0;
void setup() {
  //შუქდიოდის საკონტაქტო გამომყვანი, როგორც გამოსასვლელი
  pinMode(ledPin,OUTPUT);
  //ღილაკის საკონტაქტო გამომყვანი, როგორც შესასვლელი
  pinMode(buttonPin,INPUT);
}
void loop() {
    //ვკითხულობთ ღილაკის მდგომარეობას
    buttonState = digitalRead(buttonPin);
    //თუ ღილაკი დაჭერილია
    if (buttonState == HIGH) {
    //ვრთავთ შუქდიოდს
      digitalWrite(ledPin, HIGH);
    } else {
    //ვთიშავთ შუქდიოდს
      digitalWrite(ledPin, LOW);
    }
  }
```

ორი შუქდიოდისგან შემდგარი სქემის მართვა For ციკლის ოპერატორის გამოყენებით

ავაწყოთ კიდევ ერთი მარტივი სქემა და დავწეროთ პროგრამა, რომლის საშუალებითაც შესაძლებელი იქნება ორი შუქდიოდის მართვა.

სქემა სურ. 3.1-ზე მოცემული სქემის მსგავსია; დამატებულია მეორე შუქდიოდი, რომლის მართვაც პირველი შუქდიოდისგან დამოუკიდებლად უნდა მოხდეს. თითოეულ შუქდიოდს კვება სხვადასხვა საკონტაქტო გამომყვანიდან მიეწოდება.

ჩვენს მიერ ასაგები სქემა ასე გამოიყურება:



სურ. 3.7. ორი შუქდიოდისგან შემდგარი სქემა

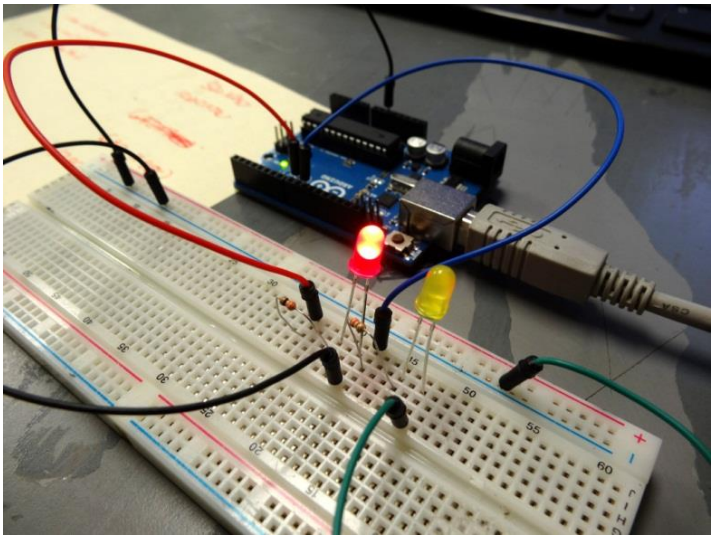
შუქდიოდების დამოუკიდებლად მართვა გულისხმობს იმას, რომ შესაძლებელია დროის მოცემულ მომენტში ერთი რომელიმე შუქდიოდი იყოს ჩართული და მეორე - არა. როდესაც ჩვენ ავაწყვეთ სურ.3.1-ზე მოცემული სქემა, და ავანთეთ წითელი შუქდიოდი, სქემაში ჩავრთეთ მეორე - ყვითელი შუქდიოდიც, ისე რომ მისი გამომყვანები იყო იმავე სვეტში, რომელშიც წითელი შუქდიოდის ანოდი და კათოდი. შედეგად მივიღეთ, რომ ორივე - წითელი და ყვითელი შუქდიოდი ერთდროულად აღმოჩდა ჩართულ ან გამორთულ მდგომარეობაში. ამის მიზეზი იყო ის, რომ ორივე შუქდიოდი მიერთებული იყო არდუინოს ერთ საკონტაქტო გამომყვანთან, შესაბამისად, ვერ ხორციელდებოდა მათი ცალ-ცალკე მართვა.

როგორც სქემიდან ჩანს, ორივე შუქდიოდის კათოდი მიწას უნდა მიუერთდეს. ამისათვის, არდუინოს სამაკეტო დაფის ზედა რიგის რომელიმე ნახვრეტში ჩავამაგროთ გამტარი, რომელიც არდუინოს GND (მიწა) გამომყვანიდან მოდის. ფაქტიურად, ეს ზედა რიგს აქცევს „მიწად“. ამის შემდეგ, ნებისმიერი მოწყობილობა, რომლის დამიწებაც დაგვჭირდება, შეიძლება უბრალოდ იყოს დაკავშირებული ზედა რიგთან გამტარის საშუალებით. იმის გამო, რომ ზედა რიგში ყველა ნახვრეტი ერთანეთთანაა დაკავშირებული, გასაგებია, რომ ამ რიგში მოთავსებული ყველა ელემენტი მიწას უკავშირდება. როგორც ზემოთ უკვე ავღნიშნეთ, ორივე შუქდიოდს უნდა ჰქონდეს საკუთარი 220 ომი წინაღობის

მქონე რეზისტორი, დენის შეზღუდვის მიზნით. სქემის აგების დროს ძალიან მნიშვნელოვანია გვახსოვდეს შეერთების მიმართულებები. როგორც უკვე ვიცით, დიოდის უფრო გრძელი საკონტაქტო გამომყვანი (ანოდი) უნდა მიუერთდეს რეზისტორს.

სურ.3.7-ზე მოცემული სქემა ავაწყოთ არდუინოს სამაკეტო დაფაზე. დავუშვათ, წითელი და ყვითელი შუქდიოდების სამართავად მეცხრე და მეათე საკონტაქტო გამომყვანები იყოს გამოყენებული. სამაკეტო დაფაზე სქემის აგება წესით უკვე სირთულეს არ უნდა წარმოადგენდეს, ვიცით შეერთების წესები და გარდა ამისა, სურ. 3.7-ზეც ნაჩვენებია დაფაზე ელემენტების განლაგება.

სურ.3.7-ის მიხედვით აგებული სქემა ასე გამოიყურება:



სურ. 3.8. ორი შუქდიოდის მართვის სქემა სამაკეტო დაფაზე

სქემის აგების შემდეგ შეგვიძლია პროგრამის წერა დავიწყოთ. ჩვენი ამოცანის მიზანია ორი შუქდიოდის დამოუკიდებლად მართვა. დავუშვათ, გვინდა წითელმა შუქდიოდმა იციმციმოს 10-ჯერ, ყვითელმა - ერთხელ. ეს ნიშნავს, რომ წითელი შუქდიოდი ჩაირთვება, ამ მდგომარეობაში დარჩება წამის მეოთხედის განმავლობაში, შემდეგ გამოირთვება და გამორთულ მდგომარეობაში რჩება ისევ წამის მეოთხედის განმავლობაში. ეს მეორდება 10-ჯერ. ამის შემდეგ ჩაირთვება ყვითელი შუქდიოდი, ის ჩართული იქნება წამის მეოთხედის განმავლობაში, გამოირთვება და შემდეგ ისევ თავიდან დაიწყება მთელი პროცესი. რადგანაც ორი შუქდიოდი გვაქვს, ჩვენ დაგვჭირდება გამოვაცხადოთ ორი ცვლადი იმ საკონტაქტო გამომყვანების აღსანიშნავად, რომლებსაც შუქდიოდები უკავშირდება. წითელი შუქდიოდი ჩვენს სქემაზე მიერთებულია მეცხრე საკონტაქტო გამომყვანთან, ყვითელი - მეათესთან. გარდა ამისა, რადგანაც უნდა იციმციმოს ორმა შუქდიოდმა, ჩვენ დაგვჭირდება onTime და offTime ცვლადები თითოეული შუქდიოდისთვის.

როგორც ზემოთ უკვე ავღნიშნეთ, ყველა პროგრამა, რომელიც არდუინოზე იწერება, ორი ძირითადი ნაწილისგან შედგება: void setup() და void loop(). setup() ფუნქციის გამოძახება ხდება ერთხელ, არდუინოს ჩართვისას. ამ ფუნქციის შიგნით იწერება ცვლადების ინიციალიზაციის კოდი, ხორციელდება საკონტაქტო გამომყვანების მუშაობის რეჟიმის

დაყენება და ა.შ. ის მონაცემები, რომელთაც პროგრამა რამდენჯერმე იყენებს, loop() ნაწილშია მოთავსებული. loop() ფუნქცია უსასრულო ციკლს წარმოადგენს, რომელშიც მიმდევრობით სრულდება ამ ფუნქციის ტანში აღწერილი ბრძანებები.

გამოვაცხადოთ გლობალური ცვლადები. საკონტაქტო გამომყვანების აღსანიშნავად ორი ცვლადის გამოცხადება დაგვჭირდება:

```
const int redLEDPin=9;
const int yellowLEDPin=10;
```

შუქდიოდების ანთება/ჩაქრობის აღსანიშნავად პროგრამაში გვექნება ოთხი ცვლადი: redOnTime, yellowOnTime, redOffTime, yellowOffTime. ამ ცვლადების მნიშვნელობები აღნიშნავს, რამდენ ხანს უნდა იყოს ჩამქრალი/ანთებული ყვითელი და წითელი შუქდიოდები. ზემოთ უკვე ავღნიშნეთ, რომ ეს დრო იყოს წამის მეოთხედი (250 მილიწამი).

ამის შემდეგ შეგვიძლია დავაყენოთ საკონტაქტო გამომყვანებისთვის მუშაობის რეჟიმი:

```
pinMode (redLEDPin, OUTPUT);
pinMode (yellowLEDPin, OUTPUT);
```

ჩვენი მიზანია ჯერ წითელმა შუქდიოდმა იციმციმოს 10-ჯერ, ამის შემდეგ აინთოს ყვითელი შუქდიოდი. ანთებულ და ჩამქრალ მდგომარეობებში ყოფნის დრო არის 250 მილიწამი. ამისათვის გამოვიყენოთ ციკლის for ოპერატორი.

პროგრამას ექნება შემდეგი სახე:

```
// წითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int redLEDPin=9;
// ყვითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int yellowLEDPin=10;
//წითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int redOnTime=250;
//წითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int redOffTime=250;
//ყვითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int yellowOnTime=250;
//ყვითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int yellowOffTime=250;

void setup()
{
  //შუქდიოდის საკონტაქტო გამომყვანი, როგორც გამოსასვლელი
  pinMode(redLEDPin,OUTPUT);
  //შუქდიოდის საკონტაქტო გამომყვანი, როგორც გამოსასვლელი
  pinMode(yellowLEDPin, OUTPUT);
}
void loop() {
  for(int j=1; j<=10; j=j+1)
  {
    //ჩავრთოთ წითელი შუქდიოდი
    digitalWrite(redLEDPin, HIGH);
    //დაყოვნების დრო redOnTime
    delay(redOnTime);
    //გამოვრთოთ წითელი შუქდიოდი
    digitalWrite(redLEDPin, LOW)
    //დაყოვნების დრო redOffTime
```

```
    delay(redOffTime);
   }
    //ჩავრთოთ ყვითელი შუქდიოდი
    digitalWrite(yellowLEDPin, HIGH);
    //დაყოვნების დრო yellowOnTime
    delay(yellowOnTime);
    //გამოვრთოთ ყვითელი შუქდიოდი
    digitalWrite(yellowLEDPin, LOW);
    //დაყოვნების დრო yellowOffTime
    delay(yellowOffTime);
}
```

თუ პროგრამას შევცვლით და 10-ის ნაცვლად ციკლის მმართველი ცვლადისთვის დავწერთ j<=5, ასეთ შემთხვევაში, წითელი შუქდიოდი აინთება/ჩაქრება 5-ჯერ, შემდეგ ერთხელ აინთება/ჩაქრება ყვითელი შუქდიოდი და ამგვარად, გაგრძელდება პროცესი.

სინათლის ტალღის მოძრაობის იმიტაცია შუქდიოდების გამოყენებით

ამოცანის მიზანია ავაწყოთ ხუთი შუქდიოდისგან შემდგარი სქემა და მოვახდინოთ სინათლის ტალღის მოძრაობის იმიტაცია შუქდიოდების ანთება/ჩაქრობის გზით.

ამოცანის ალგორითმი შეიძლება ასე ჩამოვაყალიბოთ:

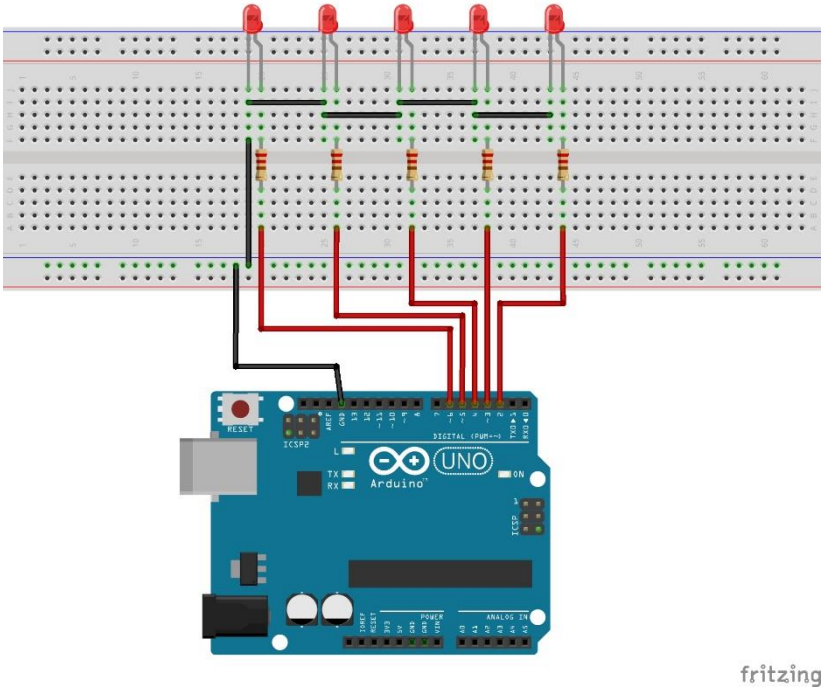
1. ჩავრთოთ პირველი შუქდიოდი;
2. პირველი შუქდიოდი ჩართულ მდგომარეობაში იყოს 0,5 წმ-ის განმავლობაში;

3. გამოვრთოთ პირველი შუქდიოდი;
4. ჩავრთოთ მეორე შუქდიოდი;
5. მეორე შუქდიოდი ჩართულ მდგომარეობაში იყოს 0,5 წმ-ის განმავლობაში;
6. გამოვრთოთ მეორე შუქდიოდი
7. ეს პროცედურა განმეორდეს დანარჩენი შუქდიოდებისთვისაც, შემდეგ იგივე მოქმედებები შესრულდეს უკუ მიმართულებით, მეოთხე შუქდიოდიდან პირველამდე.
8. გავიმეოროთ წინა პუნქტები უსასრულოდ.

ამოცანის განსახორციელებლად დაგვჭირდება ხუთი შუქდიოდი, შუქდიოდების სპეციფიკაციიდან (სამუშაო ძაბვა და დენი) გამომდინარე, შესაბამისი წინაღობის ხუთი რეზისტორი, სამაკეტო დაფა, გამტარები, არდუინოს დაფა და USB კაბელი.

შუქდიოდები უერთდება ციფრულ გამოსასვლელებს (მეორედან მეექვსეს ჩათვლით) ჩვენს შემთხვევაში, 220 ომი წინაღობის დენის შემზღუდველი რეზისტორების მეშვეობით.

როგორც სურ. 3.9-დან ჩანს, შუქდიოდები ასევე დაკავშირებულია ერთმანეთთან.



სურ. 3.9. ხუთი შუქდიოდის გამოყენებით სინათლის ტალღის მოძრაობის იმიტაციის სქემა

პროგრამის კოდი (სკეტჩი):

```
void setup()
{
//მეორე საკონტაქტო გამომყვანი, როგორც გამოსასვლელი
pinMode(2, OUTPUT);
//მესამე საკონტაქტო გამომყვანი, როგორც გამოსასვლელი
pinMode(3, OUTPUT);
//მეოთხე საკონტაქტო გამომყვანი, როგორც გამოსასვლელი
pinMode(4, OUTPUT);
//მეხუთე საკონტაქტო გამომყვანი, როგორც გამოსასვლელი
```

```
pinMode(5, OUTPUT);
//მეექვსე საკონტაქტო გამომყვანი, როგორც გამოსასვლელი
pinMode(6, OUTPUT);
}
void loop()
{
//მეორე საკონტაქტო გამომყვანთან დაკავშირებული შუქდიოდის ჩართვა
digitalWrite(2, HIGH);
delay(500); //დაყოვნება 500 მწმ

//მეორე საკონტაქტო გამომყვანთან დაკავშირებული შუქდიოდის გამორთვა
digitalWrite(2, LOW);
digitalWrite(3, HIGH);
delay(500);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
delay(500);

digitalWrite(4, LOW);
digitalWrite(5, HIGH);
delay(500);

digitalWrite(5, LOW);
digitalWrite(6, HIGH);
delay(500);

digitalWrite(6, LOW);
digitalWrite(5, HIGH);
```

```
delay(500);

digitalWrite(5, LOW);
digitalWrite(4, HIGH);
delay(500);

digitalWrite(4, LOW);
digitalWrite(3, HIGH);
delay(500);

digitalWrite(3, LOW);
}
```

პროგრამა მუშაობს, სინათლის ტალღის მოძრაობის იმიტაცია განვახორციელეთ შუქდიოდების ანთება/ჩაქრობის გზით, თუმცა სკეტჩი საკმარისად ოპტიმალური არ არის. თუ მოგვიანებით მასში ცვლილების შეტანა დაგვჭირდება, მაგალითად, თუ მოგვინდება, რომ სინათლის ტალღამ იმოძრაოს უფრო სწრაფად ან ნელა, მოგვიწევს შეცვალოთ დაყოვნების ფუნქციაში დროის მნიშვნელობა ამ ფუნქციის ყოველი გამოძახებისთვის. ჩვენს ამოცანაში დაყოვნების დრო არის 500 მილიწამი და ის მითითებულია რვაჯერ. გამოდის, რომ რვაჯერ, სათითაოდ მოგვიწევს დაყოვნების დროის მნიშვნელობის შეცვლა. ამ პრობლემის აღმოსაფხვრელად, სკეტჩში დავამატოთ ცვლადი, რომელიც წარმოადგენს დაყოვნების დროს. პროგრამის პირველივე სტრიქონში, void setup() ფუნქციამდე შემოვიტანოთ d მთელი

ტიპის ცვლადი და მივანიჭოთ მას მნიშვნელობა: int d=250; ამის შემდეგ, რიცხვი 500 მოცემულ სკეტჩში შევცვალოთ d ცვლადით. შეცვლილი სკეტჩის ატვირთვის შემდეგ, სინათლის ტალღა გაცილებით სწრაფად იმოძრავებს, რადგანაც დაყოვნების დროის მნიშვნელობა შემცირდა და გახდა 250. თუ მომავალში, ისევ მოგვინდება დაყოვნების დროის მნიშვნელობის შეცვლა, საკმარისი იქნება მისი შეცვლა მხოლოდ ერთხელ, სკეტჩის დასაწყისში, იქ სადაც ცვლადს მნიშვნელობა მივანიჭეთ.

ამ პროგრამის რეალიზება უფრო მარტივად for ოპერატორის გამოყენებითაა შესაძლებელი. დაყოვნების დრო 100 მილიწამამდე შევამციროთ.

for ოპერატორის გამოყენებით პროგრამას შემდეგი სახე ექნება:

```
int d=100; //დაყოვნების სიდიდე
void setup()
{
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
}
void loop()
{
  for (int a=2; a<7; a++)
  {
```

```
    digitalWrite(a, HIGH);
    delay(d);
    digitalWrite(a, LOW);
   }
   for (int a=5; a>2; a--)
   {
   digitalWrite(a, HIGH);
   delay(d);
   digitalWrite(a, LOW);
   }
}
```

ეკრანზე ბეჭდვა მიმდევრობითი პორტის გამოყენებით და სტრიქონებთან მუშაობა

განვიხილოთ, როგორ შეიძლება დავბეჭდოთ ინფორმაცია ეკრანზე არდუინოდან მიმდევრობითი პორტის მეშვეობით.

მიმდევრობითი პორტი (ინგლ. serial port, communications port (COM პორტი)) სტანდარტ RS-232-ის სლენგური სახელწოდებაა. პორტს მიმდევრობითი ეწოდება იმის გამო, რომ მისი გავლით ინფორმაციის გადაცემა ხდება მიმდევრობით, ისე რომ ინფორმაციის ერთ ბიტს მიჰყვება მეორე და ა.შ.

მიმდევრობითი პორტის გამოყენებით ეკრანზე ბეჭდვისთვის ვიყენებთ სურ. 3.7-ზე მოცემულ სქემას. შუქდიოდის ციმციმის გარდა, ცხადია, საინტერესო იქნებოდა

მომხმარებლისთვის ტექსტური სახითაც მიგვეწოდებინა ინფორმაცია. ამის გაკეთება შესაძლებელია მიმდევრობითი პორტისა და მონიტორის (serial monitor) მეშვეობით. იმისათვის რომ მიმდევრობითი პორტი გამოვიყენოთ, პირველ რიგში ის უნდა ჩავრთოთ ჩვენი პროგრამის მეშვეობით. რადგანაც ამის გაკეთება მხოლოდ ერთხელაა საჭირო, შესაბამისი ბრძანება Serial.begin (9600) უნდა ჩავწეროთ void setup()-ში. Serial.begin (9600) ბრძანება უზრუნველყოფს მიმდევრობითი პორტის ჩართვას. 9600 ნიშნავს რომ კომუნიკაციის სიჩქარეა 9600 ბოდი. ცხადია, რაც უფრო მაღალია ეს რიცხვი, მონაცემთა მიღება და გადაცემა მიმდევრობითი პორტის მეშვეობით მით უფრო სწრაფად ხდება. შესაძლებელია მიეთითოს სხვა რიცხვიც, მაგრამ თუ არდუინოსთვის განვსაზღვრავთ, რომ მისთვის დაყენებული იყოს 9600 ბოდი სიჩქარე, მაშინ იგივე რიცხვი უნდა იქნას მითითებული მიმდევრობითი მონიტორის-თვისაც, მიმდევრობითი მონიტორის ჩართვა ნიშნავს, რომ ჩვენ უკვე შეგვიძლია მისი საშუალებით მონაცემების გაგ-ზავნა და მიღება. მონაცემების გაგზავნა არდუინოს მიმდევ-რობით პორტზე შესაძლებელია Serial.print და Serial.println ბრძანებებით.

პროგრამის კოდი:

```
// წითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int redLEDPin=9;
// ყვითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int yellowLEDPin=10;
```

```
// წითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int redOnTime=250;
//წითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int redOffTime=250;
//ყვითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int yellowOnTime=250;
//ყვითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int yellowOffTime=250;
// ყვითელი შუქდიოდის ციმციმის რაოდენობა
int numYellowBlinks=5;
//წითელი შუქდიოდის ციმციმის რაოდენობა
int numRedBlinks=5;

void setup() {
  Serial.begin(9600); //მიმდევრობითი პორტის ჩართვა
  pinMode(redLEDPin,OUTPUT);
  pinMode(yellowLEDPin,OUTPUT);
}
void loop() {
  Serial.println("The Red LED is Blinking!");
  for(int j=1; j<=numRedBlinks;j=j+1){
    Serial.print ("  You are on Blink #: ");
    Serial.println(j);
    // ჩავრთოთ წითელი შუქდიოდი
    digitalWrite(redLEDPin, HIGH);
    //დაყოვნების დრო redOnTime
    delay(redOnTime);
    //გამოვრთოთ წითელი შუქდიოდი
    digitalWrite(redLEDPin,LOW);
    //დაყოვნების დრო redOffTime
     delay(redOffTime);
```

```
  }
Serial.println(" ");
Serial.println("The Yellow LED is Blinking!");
  for(int k=1; k<=numYellowBlinks; k++){
    Serial.print ("   You are on Blink #: ");
    Serial.println(k);
    //ჩავრთოთ ყვითელი შუქდიოდი
    digitalWrite(yellowLEDPin, HIGH);
   //დაყოვნების დრო yellowOnTime
    delay(yellowOnTime);
    // გამოვრთოთ ყვითელი შუქდიოდი
    digitalWrite(yellowLEDPin, LOW);
    //დაყოვნების დრო yellowOffTime
    delay(yellowOffTime);
  }
Serial.println(" ");
  }
```

ამ მაგალითში, ფაქტობრივად, ჩვენ უკვე ვიმუშავეთ სტრიქონებთან, როდესაც დავწერეთ Serial.println("The Red LED is Blinking") ბრძანება; სტრიქონს წარმოადგენს ბრჭყალებში მოთავსებული ტექსტი („The Red LED is Blinking"). შესაძლებელია ასევე შევქმნათ სტრიქონული ტიპის ცვლადი, მაგ.: String redMessage="The Red LED is Blinking"; გასაგებია, რომ Serial.println("The Red LED is Blinking") ბრძანება შეგვიძლია Serial.println(redMessage) ბრძანებით შევცვალოთ. შესაძლებელია სტრიქონული ტიპის ცვლადების კონკატენაცია ანუ გაერთიანება.

მაგ.. String wm1="Welcome to "

String wm2="My Project";

String wm3=wm1+wm2;

Wm3-ის მნიშვნელობა გახდება Welcome to My Project.

ზემოთ მოცემული კოდი გადავწეროთ ისე, რომ მასში გამოვიყენოთ სტრიქონებთან მუშაობის ჩვენთვის უკვე ცნობილი ხერხები.

```
// წითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int redLEDPin=9;
// ყვითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int yellowLEDPin=10;
//წითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int redOnTime=250;
//წითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int redOffTime=250;
//ყვითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int yellowOnTime=250;
//ყვითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int yellowOffTime=250;
// ყვითელი შუქდიოდის ციმციმის რაოდენობა
int numYellowBlinks=5;
//წითელი შუქდიოდის ციმციმის რაოდენობა
int numRedBlinks=5;
//ვაცხადებთ String ტიპის  redMessage  ცვლადს
String redMessage="The Red LED is Blinking";
//ვაცხადებთ String ტიპის yellowMessage ცვლადს
String yellowMessage="The yellow LED is
Blinking";
void setup() {
   //მიმდევრობითი პორტის ჩართვა
  Serial.begin(9600);
```

```
   //ვაცხადებთ String ტიპის wm1 ცვლადს და ვანიჭებთ მას მნიშვნელობას
   String wm1="Welcome to ";
   //ვაცხადებთ String ტიპის wm2 ცვლადს და ვანიჭებთ მას მნიშვნელობას
   String wm2="My Project";
   //ვაცხადებთ String ტიპის wm3 ცვლადს
   String wm3;
   //ვაერთიანებთ wm1 და wm2 სტრიქონებს wm3-ში
   wm3=wm1+wm2;
      Serial.println(wm3);
      pinMode(redLEDPin,OUTPUT);
      pinMode(yellowLEDPin, OUTPUT);
}
void loop() {
    Serial.println(redMessage);
    for(int j=1; j<=numRedBlinks; j=j+1){
      Serial.print ("  You are on Blink #: ");
      Serial.println(j);
      // ჩავრთოთ წითელი შუქდიოდი
      digitalWrite(redLEDPin,HIGH);
      //დაყოვნების დრო redOnTime
      delay(redOnTime);
      // გამოვრთოთ წითელი შუქდიოდი
      digitalWrite(redLEDPin, LOW);
      //დაყოვნების დრო redOffTime
      delay(redOffTime);
    }
    Serial.println(" ");
    Serial.println(yellowMessage);
    for(int k=1; k<=numYellowBlinks;k++){
      Serial.print ("   You are on Blink #: ");
```

```
      Serial.println(k);
      // ჩავრთოთ ყვითელი შუქდიოდი
      digitalWrite(yellowLEDPin,HIGH);
      //დაყოვნების დრო yellowOnTime
      delay(yellowOnTime);
      // გამოვრთოთ ყვითელი შუქდიოდი
      digitalWrite(yellowLEDPin,LOW);
      //დაყოვნების დრო yellowOffTime
      delay(yellowOffTime);
   }
   Serial.println(" ");
}
```

მონაცემების წაკითხვა მიმდევრობითი პორტიდან

დავუშვათ, გვინდა მომხმარებელმა არდუინოს გარკვეული სახის ინფორმაცია გადასცეს.

ვიყენებთ იგივე სქემას და კოდს, რაც წინა ამოცანაში გვქონდა (სურ. 3.7). ჩვენი პროგრამის თანახმად, 5-ჯერ ციმციმებს წითელი შუქდიოდი და ასევე, 5-ჯერ - ყვითელი. იმისათვის, რომ შევცვალოთ ეს მონაცემები (ანუ რამდენჯერ იციმციმოს წითელმა ან ყვითელმა შუქდიოდმა), პროგრამის დასაწყისში, სადაც გამოცხადებული გვაქვს შესაბამისი ცვლადები (numYellowBlinks, numRedBlinks), საჭიროა მათი მნიშვნელობების შეცვლა. მივანიჭოთ მათ მნიშვნელობები 15 და 1, შესაბამისად. კოდს ექნება შემდეგი სახე:

```
// წითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int redLEDPin=9;
//ყვითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
```

```
const int yellowLEDPin=10;
//წითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int redOnTime=250;
//წითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int redOffTime=250;
//ყვითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int yellowOnTime=250;
//ყვითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int yellowOffTime=250;
// ყვითელი შუქდიოდის ციმციმის რაოდენობა
int numYellowBlinks=15;
//წითელი შუქდიოდის ციმციმის რაოდენობა
int numRedBlinks=1;
void setup() {
  Serial.begin(9600);
  pinMode(redLEDPin,OUTPUT);
  pinMode(yellowLEDPin, OUTPUT);
}
void loop() {
  Serial.println("The Red LED is Blinking!");
  for(int j=1; j<=numRedBlinks; j=j+1){
    Serial.print ("     You are on Blink #: ");
    Serial.println(j);
    //წითელი შუქდიოდის ჩართვა
    digitalWrite(redLEDPin,HIGH);
    //დაყოვნების დრო redOnTime
    delay(redOnTime);
     //ყვითელი შუქდიოდის გამორთვა
    digitalWrite(redLEDPin,LOW);
    //დაყოვნების დრო redOffTime
     delay(redOffTime);
```

```
    }
  Serial.println(" ");
  Serial.println("The Yellow LED is
                  Blinking!");
  for(int k=1; k<=numYellowBlinks; k++){
    Serial.print ("    You are on Blink #: ");
    Serial.println(k);
    //ყვითელი შუქდიოდის ჩართვა
    digitalWrite(yellowLEDPin, HIGH);
    //დაყოვნების დრო yellowOnTime
    delay(yellowOnTime);
    //ყვითელი შუქდიოდის გამორთვა
    digitalWrite(yellowLEDPin,LOW);
    //დაყოვნების დრო yellowOffTime
    delay(yellowOffTime);
  }
    Serial.println(" ");
}
```

პროგრამა, როგორც წესი, მომხმარებლისთვის იწერება და მას კოდში ცვლილებების შეტანის უფლება არ უნდა ჰქონდეს. თუმცა, მომხმარებელს შეიძლება ჰქონდეს შესაძლებლობა თავად განსაზღვროს, რამდენჯერ აინთოს რომელიმე ფერის შუქდიოდი. ამისათვის, მან ეს რიცხვი უნდა შემოიტანოს. ამის გაკეთება შესაძლებელია მიმდევრობითი პორტის მეშვეობით. ისევე როგორც ჩვენ შეგვიძლია მომხმარებელს მივაწოდოთ (დავუბეჭდოთ ეკრანზე) ინფორმაცია მიმდევრობითი პორტის საშუალებით, ასევე შესაძლე-

ბელია მომხმარებლისგანაც მივიღოთ ინფორმაცია მიმდევრობითი პორტის მეშვეობით.

იმისათვის, რომ მომხმარებლისგან მივიღოთ ინფორმაცია, void setup()-ში ჩართული უნდა იყოს მიმდევრობითი პორტი. როგორც ზემოთ უკვე ავღნიშნეთ, ამის გაკეთება შესაძლებელია Serial.begin(9600) ბრძანებით. ეს ბრძანება ყოველთვის საჭიროა გვქონდეს void setup()-ში, თუ ვბეჭდავთ რაიმეს ეკრანზე ან ვკითხულობთ მონაცემებს მიმდევრობითი პორტის მეშვეობით.

პირველ რიგში, პროგრამის კოდში შევცვალოთ ის, რომ გამოვაცხადოთ numYellowBlinks, numRedBlinks ცვლადები მათთვის მნიშვნელობების მინიჭების გარეშე, რადგანაც ეს მნიშვნელობები მომხმარებლისგან უნდა მივიღოთ. ჩვენ შეგვიძლია მომხმარებელს ვკითხოთ, რამდენჯერ უნდა აციმციმდეს წითელი შუქდიოდი და რამდენჯერ - ყვითელი. ამისათვის გამოიყენება Serial.println ბრძანება, რომლის საშუალებითაც შეტყობინება ეგზავნება მიმდევრობით პორტს: Serial.println("How Many Times Do You Want the Red LED to Blink? "); ამ შემთხვევაში, ჩვენ შეკითხვას ვუსვამთ მომხმარებელს. ეს ნიშნავს, რომ უნდა დაველოდოთ, სანამ არ გაგვცემს პასუხს. ზოგი მომხმარებელი სწრაფად პასუხობს, ზოგს შეიძლება პასუხის გასაცემად რამდენიმე წუთიც კი დასჭირდეს, ამიტომ ჩვენ ზუსტად არ ვიცით, რამდენ ხანს უნდა ველოდოთ. უნდა ველოდოთ იმდენ ხანს, სანამ არ იქნება პასუხი. როგორ მივხვდეთ, რომ მომხმარებელმა

პასუხი გაგვცა? ამისათვის არსებობს Serial.available ფუნქცია. ეს ფუნქცია აბრუნებს 1-ს, თუ მომხმარებელმა შეიტანა მონაცემები, აბრუნებს 0-ს იმ შემთხვევაში, თუ მომხმარებელს მნიშვნელობა არ შეუტანია. ეს ნიშნავს, რომ თუ Serial.available-ის მნიშვნელობა არის 0, უნდა დაველოდოთ სანამ მომხმარებელი არ შემოიტანს მნიშვნელობას, თუ Serial.available-ის მნიშვნელობა არის 1, ესე იგი მომხმარებელს უკვე შეუტანია მნიშვნელობა და შემდეგ ნაბიჯზე უნდა გადავიდეთ.

როგორც ვხედავთ, იმისათვის რომ მივიღოთ ინფორმაცია მომხმარებლისგან, საჭიროა:

- მომხმარებელს ვთხოვოთ ინფორმაციის (მონაცემების) შემოტანა;
- დაველოდოთ, სანამ მომხმარებელი მონაცემებს შემოიტანს;
- წავიკითხოთ მონაცემები მიმდევრობითი პორტიდან.

ყოველივე ზემოთ თქმულის სარეალიზაციოდ, while ციკლი დაგვჭირდება. სანამ Serial.available() არის 0-ის ტოლი, მანამ while ციკლი მუშაობს, როგორც კი ფუნქციის მნიშვნელობა 1-ის ტოლი გახდება, გამოვდივართ ციკლიდან და სრულდება პროგრამის შემდეგი ეტაპები.

```
while (Serial.available()==0){
// დაველოდოთ
}
```

მას შემდეგ, რაც მომხმარებელი შეიტანს მონაცემს (გაგვცემს პასუხს), ჩვენ მისი წაკითხვა დაგვჭირდება:

```
numRedBlinks=Serial.parseInt();
```

Serial.parseInt() კითხულობს იმ რიცხვით მონაცემს, რომელიც შემოიტანა მომხმარებელმა და შემდეგ ამ რიცხვს ანიჭებს numRedBlinks ცვლადს. როდესაც მიმდევრობითი პორტიდან ვკითხულობთ მთელი (integer) ტიპის მონაცემს, უნდა გამოვიყენოთ Serial.parseInt(). სხვადასხვა ტიპის მონაცემის წასაკითხად სხვადასხვა ბრძანება არსებობს. თუ float ტიპის მონაცემის წაკითხვაა საჭირო, მაშინ გამოიყენება Serial.parseFloat(), ხოლო სტრიქონის (string) წასაკითხად - Serial.readString() ბრძანება.

პროგრამის კოდი:

```
// წითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int redLEDPin=9;
// ყვითელ შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int yellowLEDPin=10;
//წითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int redOnTime=250;
//წითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int redOffTime=250;
//ყვითელი შუქდიოდის ჩართულ მდგომარეობაში ყოფნის დრო
int yellowOnTime=250;
//ყვითელი შუქდიოდის გამორთულ მდგომარეობაში ყოფნის დრო
int yellowOffTime=250;

String redMessage="The Red LED is Blinking";
```

```
String yellowMessage = "The Yellow LED is
Blinking";
int numYellowBlinks;
int numRedBlinks;
void setup() {
  Serial.begin(9600);
  pinMode(redLEDPin,OUTPUT);
  pinMode(yellowLEDPin, OUTPUT);
  Serial.println("How Many Times Do You Want
  the Red LED to Blink? ");
  while (Serial.available()==0){
     //ველოდებით, სანამ მომხმარებელი არ შეიტანს რაიმე ინფორმაციას
  }
  numRedBlinks=Serial.parseInt();

  Serial.println("How Many Times Do You Want
  the Yellow LED to Blink? ");
  while (Serial.available()==0){
     //ველოდებით სანამ მომხმარებელი არ შეიტანს რაიმე ინფორმაციას
  }
   //ვკითხულობთ მომხმარებლის მიერ შეტანილ ინფორმაციას
   numYellowBlinks=Serial.parseInt();
 }

void loop() {
   Serial.println(redMessage);
   for(int j=1; j<=numRedBlinks; j=j+1){
     Serial.print ("    You are on Blink #: ");
     Serial.println(j);
     //ჩავრთოთ წითელი შუქდიოდი
```

```
    digitalWrite(redLEDPin, HIGH);
    //დაყოვნების დრო redOnTime
    delay(redOnTime);
    // გამოვრთოთ წითელი შუქდიოდი
    digitalWrite(redLEDPin, LOW);
    //დაყოვნების დრო redOffTime
    delay(redOffTime);
  }
  Serial.println(" ");
  Serial.println(yellowMessage);
  for(int k=1; k<=numYellowBlinks; k++){
    Serial.print ("    You are on Blink #: ");
    Serial.println(k);
    //ჩავრთოთ ყვითელი შუქდიოდი
    digitalWrite(yellowLEDPin, HIGH);
    //დაყოვნების დრო yellowOnTime
    delay(yellowOnTime);
    // გამოვრთოთ ყვითელი შუქდიოდი
    digitalWrite(yellowLEDPin, LOW);
    //დაყოვნების დრო yellowOffTime
    delay(yellowOffTime);
  }
    Serial.println(" ");
}
```

სტრიქონული, მთელი და მცოცავმძიმიანი მნიშვნელობების მიმდევრობითი პორტიდან წაკითხვის მარტივი გზები

იმისათვის, რომ წავიკითხოთ მონაცემები მიმდევრობითი პორტის მეშვეობით, პირველ რიგში, უნდა განვსაზღვროთ რომელი ტიპის მონაცემებს ველოდებით. უმეტეს შემთხვევაში, საქმე გვაქვს სტრიქონულ (string), მცოცავმძიმიან (float) და მთელი (int) ტიპის მონაცემებთან. სტრიქონის ან ტექსტის წაკითხვა ხდება Serial.readString() ბრძანებით, Serial.parseFloat() მცოცავმძიმიანი, Serial.parseInt() ბრძანება კი მთელი ტიპის მნიშვნელობების წასაკითხად გამოიყენება.

დავწეროთ პროგრამა, რომელიც მომხმარებლისგან მოითხოვს ინფორმაციას მისი ასაკის, წონისა და სიმაღლის შესახებ. ეკრანზე შედეგის სახით სწორედ ეს ინფორმაცია უნდა გამოვიდეს. პირველ რიგში უნდა გადავწყვიტოთ ცვლადების რომელ ტიპს ვიყენებთ. მომხმარებლის სახელის აღსანიშნავად გამოვიყენებთ String ტიპს, წონის და სიმაღლის (დავუშვათ, ვამრგვალებთ მათ უახლოეს მთელ რიცხვამდე) აღსანიშნავად შესაძლებელია int ტიპის გამოყენება. თუ გვინდა, რომ ვინმეს სიმაღლე არამთელ რიცხვებში გამოვხატოთ, ცხადია ამ შემთხვევაში float ტიპი უნდა გამოვიყენოთ. ცვლადების ტიპის შერჩევის შემდეგ ხდება მათი გამოცხადება და წაკითხვა შესაბამისი ბრძანებით.

როგორც ზემოთ ავღნიშნეთ, მომხმარებლისგან ინფორმაციის მისაღებად საჭიროა:

- მომხმარებელს ვთხოვოთ ინფორმაციის (მონაცემების) შემოტანა;
- დაველოდოთ, სანამ მომხმარებელი მონაცემებს შემოიტანს;
- წავიკითხოთ მონაცემები მიმდევრობითი პორტიდან.

პროგრამის კოდს ექნება შემდეგი სახე:

```
//ვაცხადებთ String  ცვლადს სახელისთვის
String myName;
//ვაცხადებთ Int ცვლადს ასაკისთვის
int age;
//ვაცხადებთ float ტიპის ცვლადს სიმაღლისთვის
float height;
void setup() {
  //ჩავრთოთ მიმდევრობითი პორტი
  Serial.begin(9600);
}
void loop() {
  //მომხმარებელს ვთხოვთ, რომ შეიყვანოს მონაცემი
  Serial.println("Please enter your name: ");
  //ველოდებით მანამ, სანამ მომხმარებელი შეიყვანს მონაცემს
  while (Serial.available()==0) {
  }
  //ვკითხულობთ მომხმარებლის მიერ შეყვანილ მონაცემს და ვანიჭებთ მას myName-ს
  myName=Serial.readString();
  //მომხმარებელს ვთხოვთ, რომ შეიყვანოს მონაცემი
```

```
  Serial.println("How old are you? ");
   while (Serial.available()==0)  {
    }
   //ვკითხულობთ მომხმარებლის მიერ შეყვანილ მონაცემს და ვანიჭებთ
   მას age-ს
   age=Serial.parseInt();
   Serial.println("How tall are you? ");
   //ვთხოვთ მომხმარებელს, რომ შეიყვანოს მონაცემი
   while (Serial.available()==0)  {
   }
   //ვკითხულობთ მომხმარებლის მიერ შეყვანილ მონაცემს და ვანიჭებთ მას
   height-ს
   height=Serial.parseFloat();
  //წაკითხული მნიშვნელობების გამოტანა მიმდევრობით მონიტორზე
   Serial.print("Hello ");
   Serial.print(myName);
   Serial.println(", you are ");
   Serial.print(age);
   Serial.println(" years old,");
   Serial.print("and you are ");
   Serial.println(height);
   Serial.print(" feet tall.");
   Serial.println("");
}
```

პროგრამის შესრულებით მიღებული შედეგის ნახვა შესაძლებელია მიმდევრობითი მონიტორის ფანჯარაში.

შუქდიოდის სიკაშკაშის ცვლილება განივ-იმპულსური მოდულაციის (PWM) გამოყენებით

ჩვენ უკვე ვისწავლეთ, რომ შუქდიოდების ჩართვა/გამორთვა შესაძლებელია digitalWrite() ფუნქციის საშუალებით. შესაძლებელია ასევე შუქდიოდების ნათების სიკაშკაშის რეგულირება, დროის იმ შუალედის ცვლილებით, როდესაც შუქდიოდი ჩართულ ან გამორთულ მდგომარეობაშია. ამისათვის გამოიყენება განივ-იმპულსური მოდულაცია (Pulse-Width Modulation,PWM).

განივ-იმპულსური მოდულაცია შეიძლება გამოვიყენოთ შუქდიოდის ნათების სიკაშკაშის ცვლილების ილუზიის შესაქმნელად, მისი ვთქვათ, წამში 500-ჯერ ჩართვა-გამორთვით. აღქმული სიკაშკაშე განისაზღვრება დროის იმ ინტერვალების სიდიდით, როდესაც შუქდიოდი ჩართულია ან გამორთულია. ჩვენს თვალს არ შეუძლია დაინახოს ციმციმი წამში 50-ზე მეტი სიხშირით, სწორედ ამიტომ იქმნება შთაბეჭდილება, რომ შუქდიოდი უწყვეტად ანათებს.

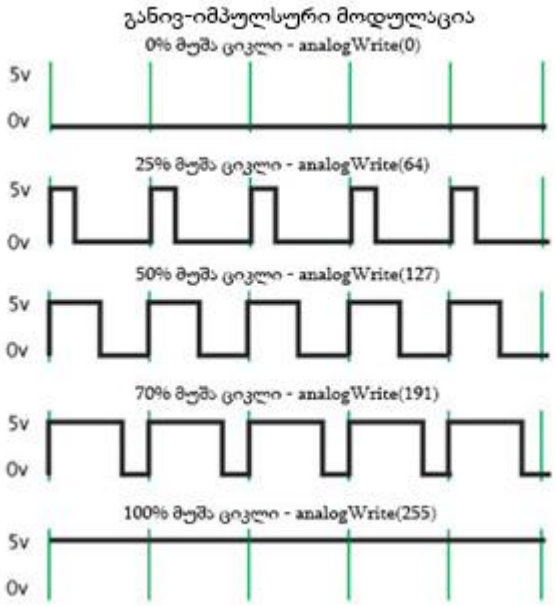
3, 5, 6, 9, 10 და 11 ციფრულ გამოსასვლელებს PWM-ის მხარდაჭერა აქვთ. ისინი არდუინოს დაფაზე „ტილდა“(~)-თი არიან აღნიშნული. განივ-იმპულსური მოდულაცია (PWM) გულისხმობს ანალოგური სიგნალის მოქმედების მსგავსი ეფექტის მიღებას ციფრული სიგნალით. იგი ორი ძირითადი პარამეტრით განისაზღვრება: შევსების კოეფიციენტი და სიხშირე. რაც უფრო მაღალია შევსების კოეფი-

ციენტი (თითოეულ ციკლში, დროის იმ მონაკვეთის დამოკიდებულება, როცა კონტაქტის გავლით დენი გადის, დროის მონაკვეთთან, როდესაც დენი არ გადის), მით მაღალია ციფრულ გამოსასვლელთან შეერთებული შუქდიოდის აღქმადი სიკაშკაშე. შევსების კოეფიციენტს ხშირად მუშა ციკლსაც უწოდებენ (Duty cycle). მუშა ციკლი წარმოადგენს იმ დროის ხანგრძლივობას, როცა სიგნალის მნიშვნელობაა ლოგიკური 1 და გამოისახება პროცენტებში, ხოლო სიხშირე მიუთითებს, რამდენი ციკლი სრულდება 1 წამის განმავლობაში (მაგ. 1000 Hz არის 1000 ციკლი წამში) და ეს თავისთავად განსაზღვრავს თუ რა სიხშირით ხდება ლოგიკურ 1-ს და ლოგიკურ 0-ს შორის მონაცვლეობა. საბოლოოდ, პრაქტიკაში ციფრული სიგნალის ხშირი და სწრაფი ჩართვა-გამორთვა შესაბამისი მუშა ციკლის დახმარებით შუქდიოდზე გვაძლევს ანალოგური სიგნალის იმიტაციას, მოცემულ მიკროკონტროლერში ამ ფუნქციის უქონლობის გამო.

სურ. 3.10-ზე გამოსახულია PWM ციკლები შევსების სხვადასხვა კოეფიციენტით. როგორც ვხედავთ, შუქდიოდის ნათების პერიოდის ხანგრძლივობა მით მეტია, რაც მეტია მუშა ციკლი (შევსების კოეფიციენტი).

PWM სიგნალის მისაღებად analogWrite(x,y) ფუნქცია გამოიყენება, სადაც x - ციფრული გამოსასვლელის ნომერია, ხოლო y - შევსების კოეფიციენტის მნიშვნელობა 0-255 დია-

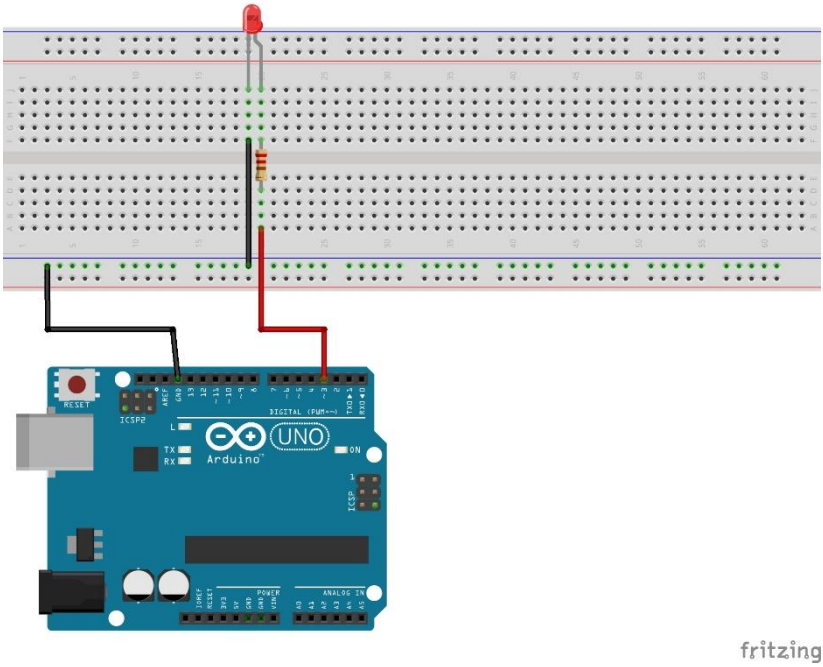
პაზონში; 0 შეესაბამება 0% შევსების კოეფიციენტს, ხოლო 255- 100%-ს.



სურ. 3.10. PWM შევსების სხვადასხვა კოეფიციენტით

PWM გამოსასვლელზე analogWrite()-ის საშუალებით 0-ის გადაცემის შემთხვევაში, ჩართული მდგომარეობის ანუ ძაბვის მიწოდების ხანგრძლივობა იქნება ნული, ანუ სამუშაო ციკლის 0%; 64-ის გადაცემის შემთხვევაში (რაც შეადგენს 255-ის ≈25%(25,1)-ს, 5ვ ძაბვა მიწოდებული იქნება დროის 25%-ის განმავლობაში, ძაბვის მიწოდება შეწყვეტილი იქნება დანარჩენი დროის (75%) განმავლობაში. 191-ის გადაცემის შემთხვევაში, დროის ≈75% (74,9)-ის განმავლობაში

მიეწოდება 5ვ, ძაბვის მიწოდება კი შეწყვეტილი იქნება დანარჩენი დროის (25%) განმავლობაში. 255-ის გადაცემა კი ნიშნავს, რომ ძაბვის მიწოდება ხდება სრული დროის (100%) განმავლობაში. საბოლოოდ, შეგვიძლია ვთქვათ, რომ ციფრულ გამომსვლელ კონტაქტზე ხდება ციფრულ ანალოგური გარდაქმნის სიმულირება.



სურ.3.11. განივ-იმპულსური მოდულაციის მაგალითი შუქდიოდით

ავაწყოთ სურ.3.11-ზე მოცემული სქემა და დავწეროთ სკეტჩი:

```
int d=5;
```

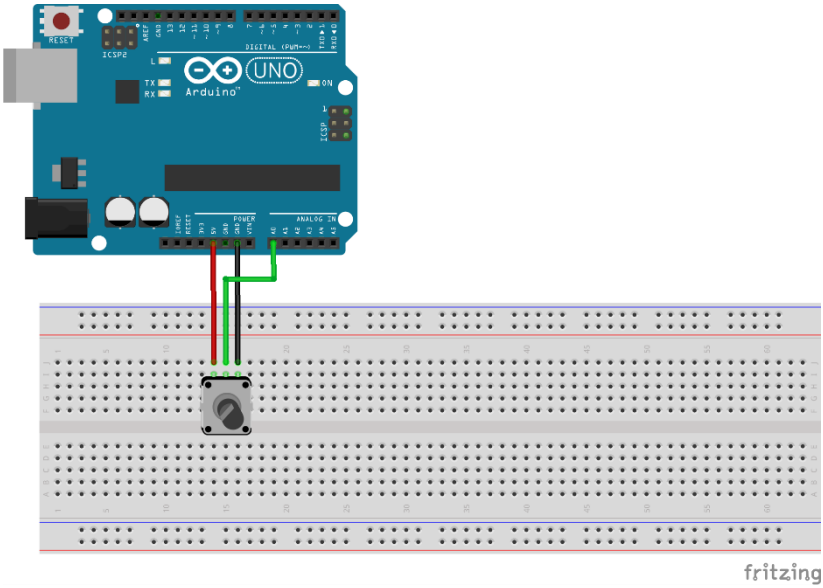
```
void setup() {
  pinMode (3, OUTPUT);
}
void loop() {
  for (int a=0; a<256; a++){
    analogWrite(3,a);
    delay(d);
  }
  for (int a=255; a>=0; a--){
    analogWrite(3,a);
    delay(d);
  }
    delay(200);
 }
```

შუქდიოდის (ამ შემთხვევაში, ის მიერთებულია მესამე ციფრულ გამოსასვლელთან) ნათების სიკაშკაშე, ნელ-ნელა იზრდება და მცირდება შევსების კოეფიციენტის ცვლილების მიხედვით. სხვა სიტყვებით, შუქდიოდის სიკაშკაშე იზრდება, აღწევს მაქსიმუმს, შემდეგ კი ნელ-ნელა მცირდება.

ანალოგური ძაბვის მნიშვნელობების წაკითხვა

განვიხილოთ, როგორ გამოვიყენოთ არდუინოს ანალოგური საკონტაქტო გამომყვანები ძაბვის მნიშვნელობების წასაკითხად. შეგახსენებთ, რომ Arduino Uno-ს შემთხვევაში ანალოგური საკონტაქტო გამომყვანები მოწყობილობაზე აღნიშნულია A0.. A5-ით.

სქემა, რომელსაც გამოვიყენებთ, წარმოადგენს მარტივ ძაბვის გამყოფს პოტენციომეტრით. პოტენციომეტრის მოქმედების პრინციპები ახსნილია წიგნის მეორე თავში.



სურ. 3.12. მარტივი ძაბვის გამყოფი პოტენციომეტრის გამოყენებით

პოტენციომეტრს სამი გამომყვანი აქვს. ე.წ. „ზედა“ გამომყვანს მიეწოდება კვება, შუა უერთდება A0-ს, ხოლო „ქვედა“ უერთდება მიწას. U_{out} გამოსასვლელი ძაბვა ეს არის ძაბვა A0-ზე (სურ. 2.10; სურ. 3.12). ჩვენ სწორედ ძაბვის ეს მნიშვნელობები უნდა წავიკითხოთ.

პოტენციომეტრის რეგულატორის მოტრიალებით იცვლება წინაღობის მნიშვნელობები შუა და ორ დანარჩენს გამომყვანს შორის.

როდესაც შესასვლელზე 5V ძაბვა გვაქვს, პოტენციომეტრის გარე გამომყვანებს შორის ძაბვაც არის 5V, პოტენციომეტრის რეგულატორის ტრიალით შესაძლებელია ძაბვის ნელ-ნელა გაზრდა/შემცირება 0-დან 5V-მდე.

წინა ამოცანაში ჩვენ განვიხილეთ PWM-ის საშუალებით ციფრულ ანალოგური გარდაქმნის იმიტაცია. როგორც ვნახეთ, ძაბვის მნიშვნელობების მითითება შესაძლებელი იყო 0-დან 255-ის ჩათვლით დიაპაზონში: 0 შეესაბამება 0 ვოლტს და 255 – 5 ვოლტს. როდესაც საუბარია ანალოგური ძაბვის მნიშვნელობების წაკითხვაზე, აქ საქმე ცოტა სხვაგვარადაა. ამ შემთხვევაში, დიაპაზონია 0-დან 1023-ის ჩათვლით, სადაც 0 შეესაბამება 0 ვოლტს და 1023 – 5 ვოლტს. ის, რომ ეს ნამდვილად ასეა, შევამოწმოთ პრაქტიკული ამოცანის საშუალებით, სადაც გამოვიყენებთ პოტენციომეტრს.

```
// პოტენციომეტრთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int  potPin=A0;
//ვაცხადებთ readValue ცვლადს, როგორც int ტიპის სიდიდეს
int readValue;
```

```
void setup() {
   //ვრთავთ მიმდევრობით პორტს
   Serial.begin(9600);
}
void loop() {
    //ვკითხულობთ potPin მნიშვნელობას და ვანიჭებთ მას readValue-ს
    readValue=analogRead(potPin);
    //მიმდევრობით პორტში ვწერთ readValue-ს მნიშვნელობას
    Serial.println(readValue);
    //დაყოვნების დრო 250 მწმ
    delay(250);
}
```

თუ პოტენციომეტრის სახელურს მოვატრიალებთ პირობითად მარჯვნივ და შედეგებს შევხედავთ მიმდევრობითი მონიტორის ფანჯარაში, დავინახავთ რიცხვებს, რომელთა სიდიდე მატულობს, მაქსიმალური რიცხვის მნიშვნელობა 1023-ია, თუ სახელურს მოვატრიალებთ მარცხნივ, რიცხვების მნიშვნელობები იკლებს, მინიმალური მნიშვნელობა ნულის ტოლი ხდება. ეს იმიტომ ხდება, რომ როგორც ზემოთ უკვე ავღნიშნეთ, პოტენციომეტრის რეგულატორის საშუალებით შესაძლებელია ძაბვის შემცირება ან გაზრდა. როდესაც ეკრანზე ვხედავთ 1023-ს, ეს ნიშნავს, რომ ამ შემთხვევაში ძაბვის მნიშვნელობა არის 5V.

არსებობს ფორმულა, რომლის საშუალებითაც შესაძლებელია წაკითხული ანალოგური ძაბვის ციფრული მნიშვნელობის გადაყვანა ვოლტებში, ამ ფორმულას შემდეგი სახე აქვს: Voltage=5/1023*ReadValue;

ახლა გადავაკეთოთ კოდი ისე, რომ დავაბეჭდინოთ ანალოგური ძაბვის მნიშვნელობები, ასე ვთქვათ, ჩვენთვის „გასაგებ“ ფორმატში, ანუ 0-დან 1023 დიაპაზონში კი არა, არამედ მნიშვნელობები 0-დან 5 ვოლტის ფარგლებში.

ჩვენი პროგრამის კოდი მიიღებს შემდეგ სახეს:

```
// პოტენციომეტრთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int potPin=A0;
//ვაცხადებთ readValue ცვლადს, როგორც int ტიპის სიდიდეს
int readValue;
//ვაცხადებთ Voltage ცვლადს, როგორც float ტიპის სიდიდეს
float Voltage;
void setup() {
  //ვრთავთ მიმდევრობით პორტს
  Serial.begin(9600);
}
void loop() {
    //ვკითხულობთ potPin მნიშვნელობას და ვანიჭებთ მას readValue-ს
    readValue=analogRead(potPin);
    //ანალოგური potPin პორტიდან წაკითხული ძაბვის ციფრული
     მნიშვნელობა readValue გადაგვყავს ვოლტებში
    Voltage=(5.0/1023.0)*readValue
     //ეს მნიშვნელობა გამოგვაქვს მიმდევრობით პორტზე
     Serial.println(Voltage);
     //დაყოვნება 250 მწმ
     delay(250);
     }
```

პოტენციომეტრის რეგულატორის მოტრიალების შესაბამისად, მიმდევრობით მონიტორში ძაბვის მნიშვნელო-

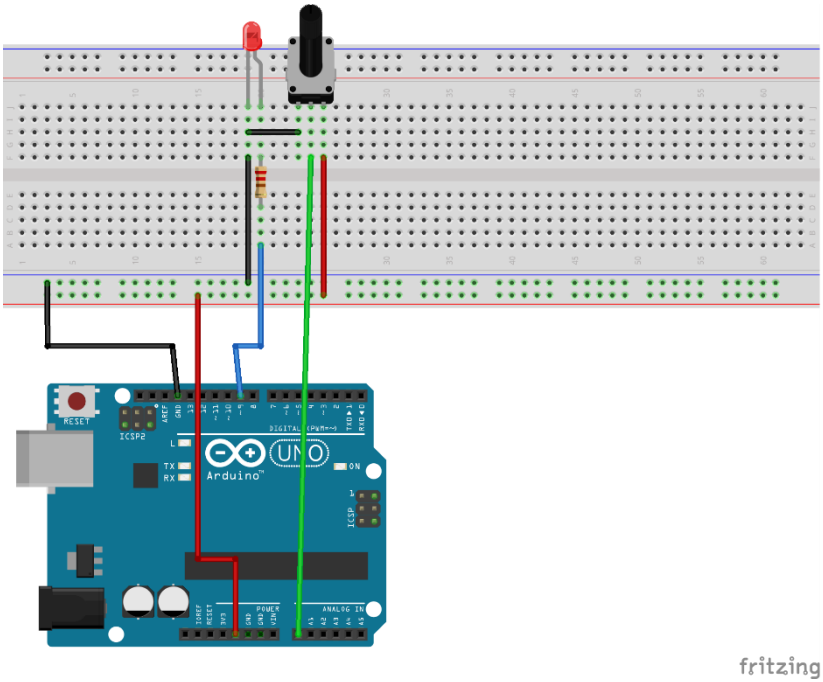
ბები გამოჩნდება 0-დან 5 ვოლტი მნიშვნელობების დია-პაზონში. ცხადია, ჩვენთვის ძაბვის მნიშვნელობების ამგვარი სახით წარმოდგენა ბევრად უფრო მოხერხებულია აღქმის თვალსაზრისით.

შუქდიოდის სიკაშკაშის ცვლილება პოტენციომეტრის გამოყენებით

ზემოთ განვიხილეთ ძაბვის მნიშვნელობების ჩაწერისა და წაკითხვის საკითხები. მიღებული ცოდნის საფუძველზე უკვე შეგვიძლია შევცვალოთ შუქდიოდის სიკაშკაშე პოტენციომეტრის რეგულატორის პოზიციის ცვლილებით.

იმისათვის, რომ ეს გავაკეთოთ, საჭიროა პოტენციომეტრი გამოყენებული იყოს, როგორც ძაბვის გამყოფი. შუქდიოდი მივუერთოთ რომელიმე PWM გამოსასვლელს. ჩვენს შემთხვევაში, შუქდიოდის მართვა ხდება მეცხრე საკონტაქტო გამომყვანიდან. სქემა, რომელსაც ავაგებთ, ნაჩვენებია სურ. 3.13-ზე.

როგორც სურ. 3.13-დან ჩანს, რეზისტორი უერთდება შუქდიოდის ანოდს, შუქდიოდის კათოდი მიერთებულია მიწასთან. ჩვენ ვიყენებთ 220 ომი წინაღობის მქონე რეზისტორს, შუქდიოდში გამავალი დენის შეზღუდვის მიზნით.



სურ. 3.13. შუქდიოდის სიკაშკაშის ცვლილება პოტენციომეტრის გამოყენებით

ჩვენი მიზანია წავიკითხოთ ძაბვის მნიშვნელობა პოტენციომეტრიდან, შემდეგ კი ეს მნიშვნელობა ჩავწეროთ შუქდიოდში. როდესაც ვკითხულობთ ძაბვას 0-დან 5 ვოლტის ჩათვლით, მიმდევრობითი მონიტორის (serial monitor) ფანჯარაში ვხედავთ რიცხვებს 0-დან 1023-ის ჩათვლით. ამ შემთხვევაში, 0-ს შეესამება 0, 1023-ს - 5 ვოლტი.

ჩვენს ამოცანაში, შუქდიოდის სიკაშკაშეს ვცვლით პოტენციომეტრიდან წაკითხული მნიშვნელობის საფუძ-

ველზე. თუ პოტენციომეტრიდან წავიკითხავთ 0-ის ტოლ მნიშვნელობას, მაშინ ჩასაწერი მნიშვნელობაც იქნება 0, რაც შეესაბამება ძაბვის ნულოვან მნიშვნელობას. თუ ჩვენ პოტენციომეტრიდან ვკითხულობთ მნიშვნელობას 1023, ეს ნიშნავს რომ უნდა ჩავწეროთ ძაბვის მაქსიმალური მნიშვნელობა 5 ვოლტი, ამისათვის კი უნდა მივუთითოთ 255-ის ტოლი მნიშვნელობა. ფაქტიურად, ჩვენ ახლა უნდა დავადგინოთ შესაბამისობა წაკითხვისა (0-დან 1023-ის ჩათვლით) და ჩაწერის (0-დან 255-ის ჩათვლით) მნიშვნელობებს შორის. შუქდიოდში ჩასაწერი მნიშვნელობის სიდიდე იქნება პოტენციომეტრიდან მიღებული მნიშვნელობის 255/1023-ზე ნამრავლი: WriteValue=(255/1023) * ReadValue

პროგრამის კოდი:

```
// პოტენციომეტრთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int potPin= A0;
// შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int LEDPin= 9;
// ეს ცვლადი კითხულობს მნიშვნელობას პოტენციომეტრიდან
int readValue;
//ამ ცვლადს ვიყენებთ შუქდიოდში მნიშვნელობის ჩასაწერად
int writeValue;
void setup() {
  pinMode(LEDPin, OUTPUT);
  //ვრთავთ მიმდევრობით პორტს
  Serial.begin(9600);
}
void loop() {
```

```
  //ვკითხულობთ ძაბვას პოტენციომეტრზე
  readValue = analogRead(potPin);
  //ვანგარიშობთ writeValue-ს მნიშვნელობას შუქდიოდისთვის
  writeValue = (255.0/1023.0)*readValue;
  //ვწერთ writeValue-ს შუქდიოდში
  analogWrite(LEDPin,writeValue);
  Serial.print("Writing a value of ");
  Serial.println(writeValue);
}
```

მიაქციეთ ყურადღება, რომ აქ 255 და 1023 ისე წერია თითქოს ისინი float ტიპის იყვნენ, writeValue კი ამ დროს აღწერილია, როგორც მთელი ტიპის ცვლადი. 255 და 1023 რომ დაგვეწერა, როგორც მთელი რიცხვები, გაყოფის შედეგი ნულის ტოლი იქნებოდა და შესაბამისად, writeValue-ც ვერასოდეს ვერ მიიღებდა სხვა მნიშვნელობას, გარდა ნულისა.

მოცემული კოდის საშუალებით, პოტენციომეტრის რეგულატორის დატრიალების გზით, ჩვენ შევძლებთ შუქდიოდის სიკაშკაშის ცვლილებას. ძაბვის მნიშვნელობის წაკითხვა ხდება პოტენციომეტრიდან, შემდეგ კი - ამ მნიშვნელობის ჩაწერა შუქდიოდში. შედეგების ნახვა შესაძლებელია მიმდევრობითი მონიტორის ფანჯარაში.

რელეს მოდულის მართვა არდუინოს საშუალებით

განვიხილოთ, როგორ შეიძლება ვმართოთ რელეს მოდული არდუინოს საშუალებით. ვაჩვენოთ ეს ნათურას მაგალითზე. არდუინოს შეუძლია მართოს მოწყობილობები, რომლებიც მუშაობენ 5V-მდე ძაბვაზე, თუ საჭიროა 5V-ზე უფრო მაღალ ძაბვაზე ან ცვლად დენზე მომუშავე მოწყობილობების მართვა, ასეთ შემთხვევაში რელეს მოდული უნდა გამოვიყენოთ. მისი საშუალებით შესაძლებელია როგორც მუდმივ, ასევე ცვლად დენზე მომუშავე მოწყობილობების მართვა.

სურ. 3. 14. 5 ვოლტიანი რელეს მოდული

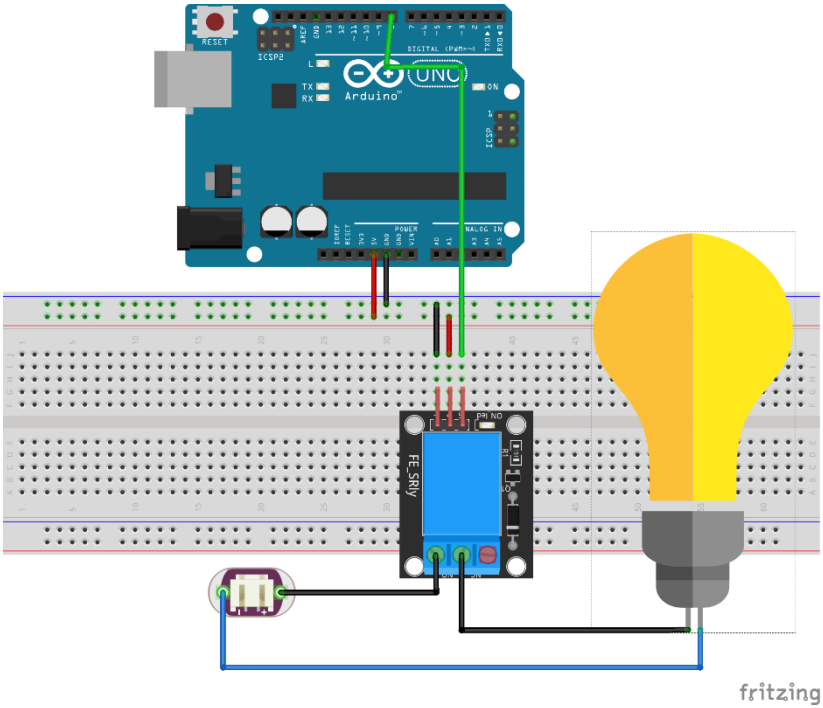
ჩვენ გამოვიყენებთ SRD-05VDC-SL-C რელეს მოდულს. ის მუშაობს 5V-ზე და მისი მართვა შესაძლებელია ნებისმიერი მიკროკონტროლერით. ჩვენ არდუინოს გამოვიყენებთ.

რელეს მოდულს 6 საკონტაქტო გამომყვანი აქვს: სამი ერთ მხარეს და სამი - მეორე მხარეს. ქვედა მხარეს განლაგებულია სიგნალის, 5V და მიწის შესაბამისი საკონტაქტო გამომყვანები. ისინი არდუინოს უნდა მივუერთოთ. მეორე მხარეს განლაგებულია NC (ნორმალურად ჩაკეტილი), C (საერთო) და NO (ნორმალურად ღია) გამომყვანები, ისინი წარმოადგენენ 5V-იანი რელეს გამოსასვლელ საკონტაქტო გამომყვანებს და გამოსასვლელ მოწყობილობას უნდა მივუერთოთ.

იმისათვის რომ განვახორციელოთ მაღალ ძაბვაზე მომუშავე მოწყობილობის მართვა რელეს მოდულის საშუალებით, უსაფრთხოების წესების ცოდნაა აუცილებელი. მაღალი ძაბვა საშიშია და მასთან არასწორად მუშაობის შემთხვევაში მან ადამიანის ჯანმრთელობას შეიძლება სერიოზული ზიანი მიაყენოს.

მაღალ ძაბვაზე მომუშავე მოწყობილობის სამართავად, საჭიროა გარე კვების წყარო. მაგალითად, 220 ვოლტიანი ნათურის ასანთებად ის უნდა მივუერთოთ სახლის 220 ვოლტიან ქსელს. დავუკავშიროთ VCC, მიწა და სიგნალი შესაბამისად, 5V, მიწას და მე-8 საკონტაქტო გამომყვანებს არდუინოზე. მეორე მხარეს, 220 ვოლტიანი ცვლადი დენის წყაროს

ერთი გამტარი მივუერთოთ ნათურას ერთ რომელიმე ბოლოს, მეორე გამტარი - რელეს საერთო (C) გამომყვანს. ნორმალურად ჩაკეტილი საკონტაქტო გამომყვანი (NO) მივუერთოთ ნათურის მეორე ბოლოს.



სურ. 3. 15. რელეს მოდულის მიერთება არდუინოსთან

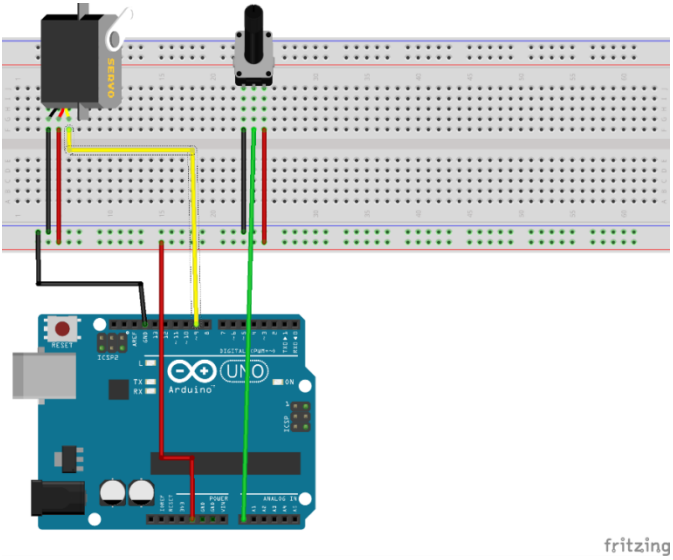
კოდი:

```
//რელეს მოდულზე მიერთებული საკონტაქტო გამომყვანის ნომერი
const int relay_pin = 8;
void setup(){
  //relay_pin როგორც გამომყვანი
  pinMode(relay_pin,OUTPUT);
```

```
void loop(){
    digitalWrite(relay_pin,HIGH); //ჩავრთოთ რელე
    delay(5000); //დაყოვნება 5 წამი
    digitalWrite(relay_pin,LOW); //გამოვრთოთ რელე
    delay(5000); //დაყოვნება 5 წამი
   }
```

სერვოძრავას მართვა არდუინოს საშუალებით

სერვოძრავას პოზიციის მართვა შესაძლებელია პოტენციომეტრის მეშვეობით. სერვოძრავა უნდა „მიჰყვებოდეს“ პოტენციომეტრის ღერძის პოზიციას. არდუინოს მეშვეობით სერვოძრავას სამართავად გამოვიყენებთ სურ.3.16-ზე მოცემულ სქემას.



სურ. 3.16. სერვოძრავას და პოტენციომეტრის არდუინოსთან მიერთების სქემა

სტანდარტულად, სერვოძრავას აქვს სამი (წითელი, ყვითელი და შავი) გამტარი. სხვადასხვა მწარმოებლის მიერ გამოშვებულ სერვოძრავებს შეიძლება განსხვავებული ფერის გამტარები ჰქონდეს. წითელი ფერის გამტარი არის კვებისთვის, ის უნდა მივუერთოთ არდუინოს კვებას (5V), შავი ფერის გამტარი - მიწას (GND), ხოლო ყვითელი ფერის გამტარი უნდა მიუერთდეს არდუინოს ერთ-ერთ საკონტაქტო გამომყვანს, რომელიც ასრულებს განივ-იმპულსური მოდულაციის ფუნქციას. ჩვენს შემთხვევაში, ამისათვის გამოვიყენოთ მეცხრე საკონტაქტო გამომყვანი. მცირე სიმძლავრის სერვოძრავას კვება შეიძლება პირდაპირ არდუინოდან მიეწოდოს. სერვოძრავების უმეტესობა დიდ სიმძლავრეს მოითხოვს, ასეთ შემთხვევაში საჭირო ხდება ცალკე კვების წყაროს არსებობა. დიდი სიმძლავრის სერვოძრავას მიერთებამ შეიძლება არდუინო დააზიანოს. სერვოძრავების უმეტესობა გათვლილია 0-180 გრადუსის დიაპაზონში სამუშაოდ. თუმცა, სიმართლე ისაა, რომ მათი უმეტესობა ამ სრულ დიაპაზონში ვერ მუშაობს. გარდა ამისა, უნდა გვახსოვდეს ისიც, რომ სერვოძრავას ამუშავების მცდელობას იმ დიაპაზონის გარეთ, რომლის ფარგლებშიც მას შეუძლია მოქმედება, შესაძლებელია მოჰყვეს როგორც არდუინოს, ასევე სერვოძრავას დაზიანება. თითოეული სერვოძრავა განსხვავებულია და ზოგჯერ ორ სერვოძრავას, ერთი და იგივე სამოდელო ნომრით და ერთი და იმავე მწარმოებლის მიერ წარმოებულს, შეიძლება სხვადასხვა სამოძრაო დია-

პაზონი ჰქონდეს. ეს განსაკუთრებით იაფფასიან მოდელებში იგრძნობა.

იმისათვის, რომ სერვოძრავასთან მუშაობა შევძლოთ, შესაბამისი ბიბლიოთეკა უნდა ჩავტვირთოთ. ამისათვის, პროგრამის კოდში შემდეგი სტრიქონი მოვათავსოთ: #include <Servo.h>. ამის შემდეგ, საჭიროა შეიქმნას Servo კლასის ობიექტი. ეს ობიექტი იქნება ის რაღაც, რომლის საშუალებითაც ჩვენ სერვოძრავას ავამოძრავებთ. ობიექტის შექმნა შემდეგნაირად შეიძლება:

```
Servo myPointer;
```

იმისათვის, რომ სერვოძრავა გარკვეულ პოზიციაში გადაადგილდეს, პროგრამის კოდში ჩავწეროთ:

```
myPointer.write(pos);
delay(15);
```

pos ცვლადმა შეიძლება მიიღოს მნიშვნელობა 0-დან 180 გრადუსის დიაპაზონში. თუმცა, როგორც ზემოთ ავღნიშნეთ, სერვოძრავების უმეტესობას არ შეუძლია იმოქმედოს მთლიან არეში, ამიტომ დაგვჭირდება ექსპერიმენტების ჩატარება იმის გასარკვევად, თუ რა დიაპაზონში შეიძლება იმუშაოს ჩვენმა სერვოძრავამ უსაფრთხოდ.

შემდეგი მოქმედება, რაც უნდა გავაკეთოთ, ის არის რომ არდუინოს „გავაგებინოთ“, რომელ საკონტაქტო გამომყვანთან არის მიერთებული სერვოძრავა. ჩვენს შემთხვევაში, ეს იქნება მეცხრე საკონტაქტო გამომყვანი, თუმცა ის შეიძლება იყოს ნებისმიერი საკონტაქტო გამომყვანი,

რომელიც ასრულებს განივ-იმპულსური მოდულაციის ფუნ-ქციას. პროგრამის კოდის დასაწყისში გამოვაცხადოთ ცვლადი servoPin: int servoPin=9; ჩვენ ამით არდუინოს გავაგებინეთ, რომ მეცხრე საკონტაქტო გამომყვანს ვიყენებთ, ახლა უკვე საჭიროა კოდში ვაჩვენოთ, რომ ჩვენი სერვოძრავა (რომელსაც myPointer დავარქვით) ამ კონკრეტულ გამომყვანთანაა (servoPin) მიერთებული. ამისათვის კოდის void setup ნაწილში ვწერთ:

myPointer.attach(servoPin);

ამის შემდეგ შეგვიძლია დავწეროთ პროგრამა, რომელიც მომხმარებელს საშუალებას მისცემს სერვოძრავას პოზიცია პოტენციომეტრის ღერძის ბრუნვის მიხედვით შეცვალოს.

```
//გამოვიყენოთ servo ბიბლიოთეკა
#include <Servo.h>
//int ტიპის ცვლადი, რომელშიც სერვოძრავას პოზიცია ინახება
int pos = 0;
// პოტენციომეტრთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int potPin= A0;
// ეს ცვლადი კითხულობს მნიშვნელობას პოტენციომეტრიდან
int readValue;
//არდუინოს საკონტაქტო გამომყვანი, რომელთანაც სერვოძრავაა მიერთებული
const int servoPin=9;
//15 მწმ დაყოვნება
int servoDelay=15;
//ობიექტის შექმნა
Servo myPointer;
```

```
void setup()
{
  //ვეუბნებით პროგრამას, რომ სერვოძრავა მიერთებულია servoPin-ზე,
  რომელიც ფიზიკურად არდუინოს მე-9 გამომყვანია
   myPointer.attach(servoPin);
}
void loop() {
   //ვკითხულობთ ძაბვას პოტენციომეტრზე
   readValue = analogRead(potPin);
   //ვახდენთ წაკითხული ძაბვის შესაბამისი ციფრული მნიშვნელობის
   მასშტაბირებას სერვოს პოზიციების მნიშვნელობებთან
   readValue = map(val, 0, 1023, 0, 179);
   //ვანიჭებთ სერვოძრავას readValue ცვლადის მნიშვნელობას
   myPointer.write(readValue);
   //ველოდებით 15 მწმ, რომ სერვოძრავა მივიდეს ბრძანებაში მითითებულ
   პოზიციამდე
   delay(servoDelay);
}
```

დროის ათვლა არდუინოში millis() ფუნქციის გამოყენებით

millis() ფუნქცია აბრუნებს იმ მილიწამების რაოდენობას, რომელიც პროგრამის გაშვებიდან (არდუინოზე კვების მიწოდებიდან) გავიდა. ამ რაოდენობის განულება ხდება დაახლოებით 50 დღის შემდეგ, არდუინოს სისტემის თავისებურებიდან გამომდინარე. millis() ფუნქციას პარამეტრები არა აქვს. სინტაქსი შემდეგნაირად გამოიყურება:

time = millis().

მიაქციეთ ყურადღება, millis() ფუნქცია აბრუნებს unsigned long ტიპს. ლოგიკური შეცდომა წარმოიშობა, თუ პროგრამისტი შეეცდება შეასრულოს გარკვეული არითმეტიკული მოქმედებები უფრო მცირე ზომის მონაცემთა ტიპებთან (მაგ. int). Signed long-ის შემთხვევაშიც კი შეიძლება მივიღოთ ლოგიკური შეცდომა, რადგანაც მისი მაქსიმალური სიდიდე ორჯერ ნაკლებია unsigned long-ზე.

ქვემოთ მოცემული კოდის საშუალებით შესაძლებელია ეკრანზე დაიბეჭდოს პროგრამის დაწყებიდან გასული დრო.

```
unsigned long time;
void setup(){
  Serial.begin(9600);
}
void loop(){
  Serial.print("Time: ");
  time = millis();
  //პროგრამის გაშვებიდან დაწყებული დრო
  Serial.println(time);
  //დავლოდოთ 1 წმ
  delay(1000);
}
```

ქვემოთ მოყვანილ მაგალითში შუქდიოდი ინთება ან ქრება 1 წამის შუალედით millis() ფუნქციის გამოყენებით.

```
//შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int ledPin =  13;
//int ტიპის ცვლადი, რომელიც შუქდიოდის მდგომარეობას ინახავს
int ledState = LOW;
```

```
//დროის მნიშვნელობის შესანახად უნდა გამოვიყენოთ unsigned long ტიპის ცვლადი, რადგან ეს სიდიდე იმდენად დიდია, რომ int ტიპის ცვლადში არ შეინახება; ვინახავთ დროის იმ მნიშვნელობას, როცა ბოლოს მოხდა შუქდიოდის მდგომარეობის ცვლილება
unsigned long previousMillis = 0;
//long ტიპის კონსტანტა, რომელშიც მოთავსებულია დროის ის სიდიდე, რა დროშიც უნდა მოხდეს შუქდიოდის ციმციმი. ჩვენს შემთხვევაში, ეს არის 1000 მილიწამი ანუ 1 წამი.
const long interval = 1000;
void setup() {
   pinMode(ledPin, OUTPUT);
}
void loop() {
  //შევამოწმოთ მიმდინარე დრო
  unsigned long currentMillis= millis();
  //შევამოწმოთ მიმდინარე დროის და ბოლოს აღებული მისი მნიშვნელობის სიდიდეთა სხვაობა თუ არის მეტი ან ტოლი 1000-ის, ანუ გავიდა თუ არა 1000 მწმ
  if(currentMillis-previousMillis>= interval){
    //შევინახოთ მიმდინარე დროის მნიშვნელობა
   previousMillis = currentMillis;
   //თუ შუქდიოდი გამორთულია- ჩავრთოთ, თუ არადა-გამოვრთოთ
   if (ledState == LOW) {
    ledState = HIGH;
   }else{
    ledState = LOW;
   }
   //მივანიჭოთ შუქდიოდის საკონტაქტო გამომყვანს ledState ცვლადის მნიშვნელობა
   digitalWrite(ledPin, ledState);
 }
}
```

თხევადკრისტალური დისპლეის (LCD) მართვა

განვიხილოთ თხევადკრისტალური დისპლეი პარალელური ინტერფეისით. პარალელური ინტერფეისის შემთხვევაში, მიკროკონტროლერი დისპლეის მართვის მიზნით, ერთდროულად ურთიერთქმედებს LCD-ს ინტერფეისის საკონტაქტო გამომყვანებთან. LCD-ს ინტერფეისი შემდეგი გამომყვანებისგან შედგება:

- რეგისტრის ამორჩევის (RS) საკონტაქტო გამომყვანი;
- ჩაწერა/წაკითხვის (Read/Write (R/W)) საკონტაქტო გამომყვანი, რომელიც კითხვის ან ჩაწერის რეჟიმს განსაზღვრავს.
- Enable საკონტაქტო გამომყვანი;
- მონაცემთა რვა (D0-D7) საკონტაქტო გამომყვანი. ამ საკონტაქტო გამომყვანების მდგომარეობები (მაღალი ან დაბალი) არის ის ბიტები, რომელსაც ვწერთ ან ვკითხულობთ LCD-ში.

გარდა ამისა, არის კიდევ დისპლეის კონტრასტის რეგულირების გამომყვანი (Vo), კვების წყაროს საკონტაქტო გამომყვანები (+5V და GND).

დისპლეის მართვის პროცესი მოიცავს მონაცემების შეტანას მონაცემების რეგისტრში, ამ მონაცემებით ფორმირდება ის გამოსახულება, რომლის გამოტანაც ეკრანზე გვსურს. LiquidCrystal ბიბლიოთეკა ამ პროცესს ისე ამარტივებს, რომ ჩვენ არ დაგვჭირდება დაბალი დონის ბრძანებების ცოდნა.

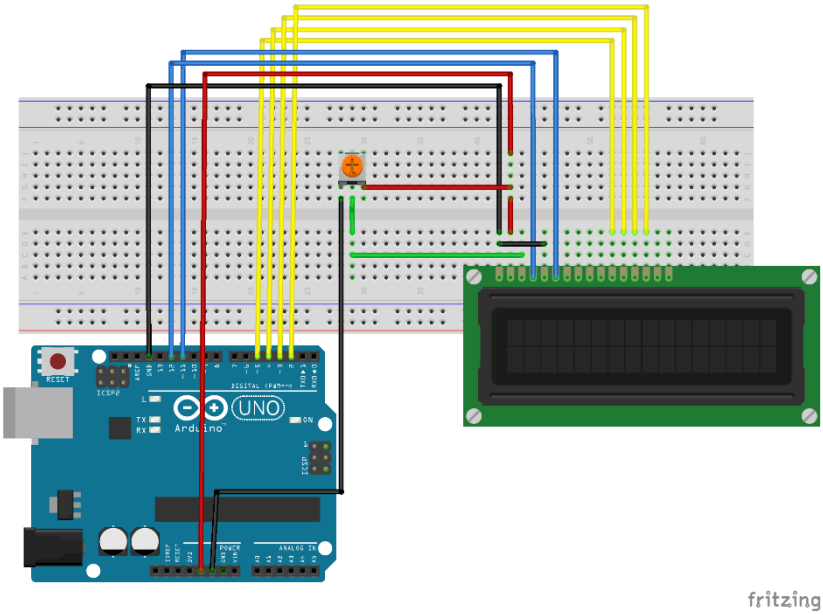
LiquidCrystal ბიბლიოთეკა მუშაობს ყველა იმ LCD დისპლეისთან, რომელიც Hitachi HD44780 დრაივერთანაა თავსებადი.

Hitachi-სთან თავსებადი LCD-ების მართვა შეიძლება 4 და 8-ბიტიან რეჟიმებში. 4-ბიტიანი რეჟიმი არდუინოს 7 საკონტაქტო გამომყვანს მოითხოვს, 8-ბიტიანი რეჟიმის შემთხვევაში - 11 საკონტაქტო გამომყვანია საჭირო. ტექსტის ეკრანზე გამოსატანად, 4-ბიტიან რეჟიმში თითქმის ყველაფრის გაკეთებაა შესაძლებელი. ქვემოთ მოცემულ მაგალითში ნაჩვენებია 2x16LCD-ის მართვა 4-ბიტიან რეჟიმში.

არდუინოს LCD-სთან მიერთების სქემა ნაჩვენებია სურ. 3.17-ზე. LCD-ს მისაერთებლად არდუინოს დაფასთან, შემდეგი საკონტაქტო გამომყვანები უნდა დავაკავშიროთ:

- LCD RS და 12
- LCD Enable და 11
- LCD D4 და 5
- LCD D5 და 4
- LCD D6 და 3
- LCD D7 და 2

გარდა ამისა, გამტარების საშუალებით მივაერთოთ 10kΩ წინაღობის მქონე პოტენციომეტრი +5V და GND-ს, პოტენციომეტრის შუა საკონტაქტო გამომყვანი მივუერთოთ LCD-ს V0-ს (მესამე საკონტაქტო გამომყვანი).



სურ. 3.17. LCD-ს მიერთება არდუინოსთან

ქვემოთ მოცემულია პროგრამა, რომლის საშუალებითაც ეკრანზე შეგვიძლია დავწეროთ ტექსტი „Hello World”.

```
//გამოვიყენოთ LiquidCrystal ბიბლიოთეკა:
#include <LiquidCrystal.h>
//ხდება ბიბლიოთეკის ინიციალიზაცია, რაც მდგომარეობს იმაში, რომ
ვაცხადებთ დისპლეის და მასთან მიერთებულ არდუინოს საკონტაქტო გამომ-
ყვანებს
const int rs=12, en=11, d4=5, d5=4,d6=3, d7=2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  //ვაცხადებთ რამდენი სტრიქონისა და სვეტისგან შედგება დისპლეი
  (დამოკიდებულია დისპლეის ტიპზე)
```

```
  lcd.begin(16,2);
  //ვწერთ შეტყობინებას დისპლეის ეკრანზე
  lcd.print("hello, world!");
}
void loop() {
//კურსორის პოზიცია ეკრანზე
lcd.setCursor(0, 1);
//მოცემულ პოზიციაზე გამოგვყავს არდუინოს ჩართვიდან გასული დრო
lcd.print(millis() / 1000);
}
```

შუქდიოდების მართვა მასივის გამოყენებით

მასივი არის ერთი ტიპის ცვლადების ერთობლიობა, რომლის ცალკეულ ელემენტებთან წვდომა ინდექსების საშუალებით ხორციელდება.

ქვემოთ მოცემულია მასივის გამოცხადების რამდენიმე ხერხი:

```
int myInts[5];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[5] = {2, 4, -8, 3, 2};
char message[5] = "hello";
```

მასივი მისი ინიციალიზაციის გარეშეც შეგვიძლია გამოვაცხადოთ (int myInts[5]); MyPins მასივის შემთხვევაში, მასივს ზომის მითითების გარეშე ვაცხადებთ. ამ დროს, კომპილატორი ითვლის ელემენტების რაოდენობას და ქმნის შესაბამისი ზომის მასივს. ბოლო ორ მაგალითში, მასივები ინიციალიზებულია და ზომაც მითითებულია.

მასივის ელემენტებისადმი წვდომა ინდექსების საშუალებით ხდება, ინდექსაცია იწყება 0-დან. შესაბამისად, mySensVals[0] = 2, mySensVals[1] = 4 და ა.შ. ათელემენტიან მასივში, ბოლო ელემენტის ინდექსი იქნება 9, მაგალითად:
int myArray[10]={9,3,2,4,3,2,7,8,9,11};

myArray[9]-ს მნიშვნელობა არის 11. myArray[10] ელემენტი არ არსებობს, თუმცა თუ მაინც შევეცდებით მას მივმართოთ, პროგრამა რაღაც შემთხვევით სიდიდეს აიღებს მნიშვნელობად, ამიტომ მასივის ელემენტებისადმი მიმართვის დროს აუცილებელია განსაკუთრებით ყურადღებით ვიყოთ.

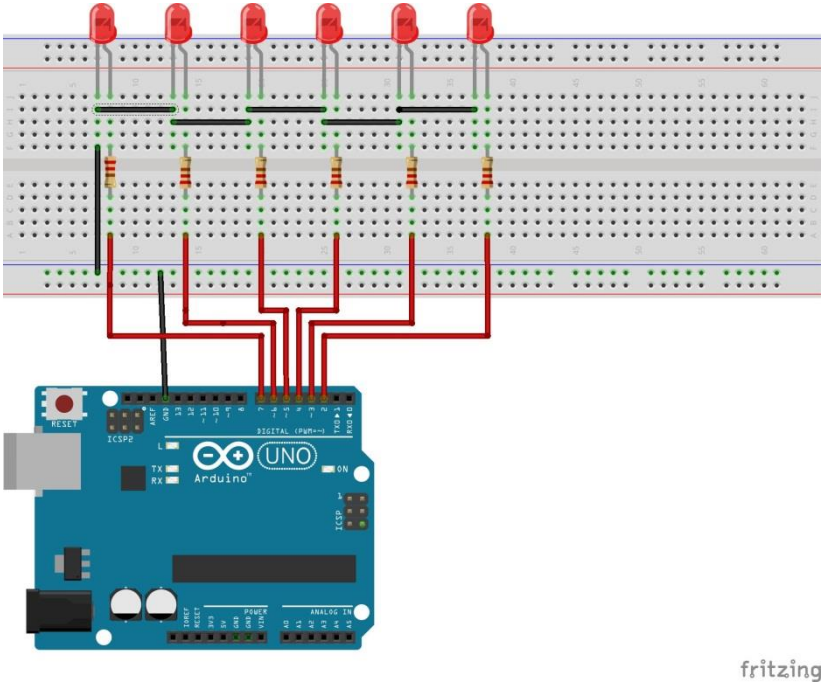
მასივის ელემენტისთვის მნიშვნელობის მინიჭება შესაძლებელია შემდეგნაირად: mySensVals[0]=10; შესაძლებელია ასევე ცვლადს მივანიჭოთ მასივის ელემენტის მნიშვნელობა: x = mySensVals[4];

მასივები ხშირად გამოიყენება for ციკლის შიგნით, სადაც ციკლის მმართველი ცვლადი წარმოადგენს ინდექსს მასივის თითოეული ელემენტისთვის. მაგალითად, მიმდევრობითი პორტის მეშვეობით მასივის ელემენტების გამოსატანად, შესაძლებელია გამოვიყენოთ შემდეგი კოდი:

```
int i;
for (i = 0; i < 5; i = i + 1) {
  Serial.println(myPins[i]);
}
```

ქვემოთ განხილული მაგალითი აჩვენებს, როგორ შეიძლება ვმართოთ არამეზობლად მდებარე საკონტაქტო

გამომყვანები, მათი ნომრების მასივში მოთავსების გზით. ამისათვის შეგვიძლია for ციკლის ოპერატორი გამოვიყენოთ. ამოცანის შესასრულებლად დაგვჭირდება 6 შუქდიოდი, რომლებიც არდუინოს საკონტაქტო გამომყვანებთან იქნება დაკავშირებული, ასევე შუქდიოდების სპეციფიკაციის შესაბამისი წინაღობის რეზისტორები. შუქდიოდების მართვა ხორციელდება მასივში მათი თანმიმდევრობის და არა ფიზიკურად სქემაში ჩართვის მიმდევრობის მიხედვით.



სურ.3.18. შუქდიოდების მართვა მასივის გამოყენებით

```
int timer = 100;
//არდუინოს იმ საკონტაქტო გამომყვანების ნომრების მასივი, რომლებზეც შუქდიოდებია მიერთებული
int ledPins[] = {2,7,4,6,5,3};
//საკონტაქტო გამომყვანების რაოდენობა (მასივის სიგრძე)
int pinCount = 6;
void setup() {
//მასივის ელემენტების ინდექსაცია იწყება 0-დან და მთავრდება pinCount-ით. გამოვიყენოთ for ციკლი იმისთვის, რომ თითოეული საკონტაქტო გამომყვანის ინიციალიზაცია მოვახდინოთ გამოსასვლელის სახით
for (int thisPin = 0; thisPin < pinCount;
thisPin++) {
  pinMode(ledPins[thisPin], OUTPUT);
}
}
void loop() {
for (int thisPin=0;thisPin<pinCount;thisPin++)
{
  //ჩავრთოთ შუქდიოდი
  digitalWrite(ledPins[thisPin],HIGH);
  //დავველოდოთ timer დრო
  delay(timer);
  //გამოვრთოთ შუქდიოდი
  digitalWrite(ledPins[thisPin],LOW);
  }
for(int thisPin=pinCount-1; thisPin>=0;
thisPin--)
{
  //ჩავრთოთ შუქდიოდი
  digitalWrite(ledPins[thisPin],HIGH);
```

```
    //დაველოდოთ timer დრო
    delay(timer);
     //გამოვრთოთ შუქდიოდი
    digitalWrite(ledPins[thisPin], LOW);
  }
}
```

წყვეტები

წყვეტა (interrupt) - სიგნალი პროგრამული ან აპარატურული უზრუნველყოფიდან, რომლის დანიშნულებაა მიკროკონტროლერს შეატყობინოს რაიმე მოვლენის დაწყება, რომელიც დაუყოვნებლივ ყურადღებას საჭიროებს.

წყვეტა აფრთხილებს მიკროკონტროლერს მაღალი პრიორიტეტის მქონე მოვლენის დაწყების თაობაზე, რის შედეგადაც მიკროკონტროლერი ძირითადი კოდის შესრულებას წყვეტს, ინახავს ამ მდგომარეობას და გადადის წყვეტის დამუშავების ფუნქციაზე. ამ ფუნქციას (რუტინას) ინგლისურად interrupt service routine, ISR ეწოდება. ეს ფუნქცია დროებითია და მისი დასრულების შემდეგ მიკროკონტროლერი ძირითადი კოდის შესრულებას იმ ადგილიდან აგრძელებს, საიდანაც გამომახება მოხდა წყვეტის სიგნალის გაჩენისას.

წყვეტა გამოიყენება მიკროკონტროლერის მიერ, რათა ყურადღება მიაქციოს გარე, პერიფერიულ სიგნალებს, მაგალითად, ციფრულ შესასვლელზე ღილაკის დაჭერას, რაიმე გადამწოდიდან სიგნალის გამოჩენას, ანალოგურ შესასვლელზე ანალოგური სიგნალის მიღების შემდეგ მისი

დამუშავების დასრულებას, შიგა ტაიმერების მთვლელის გადაჭარბებას და ა.შ.

არდუინოს ბაზურ მოდელებს (Uno, Nano, Mini და ა.შ. ATmega 168/328-ის ბაზაზე) ორი ტიპის წყვეტა გააჩნია: შიგა (Internal) და გარე (external). გარე წყვეტები გამოწვეულია შეტანა-გამოტანის პორტებზე სიგნალის ცვლილებით, ხოლო შიგა წყვეტები - მიკროკონტროლერის ტაიმერებით. გარე წყვეტები, არდუინოს შემთხვევაში, განხორციელებულია ორ ციფრულ შესასვლელზე INT0 და INT1 და ისინი შეესაბამება არდუინოს დაფაზე მეორე და მესამე გამომყვანს. თვითონ მიკროკონტროლერს, რომელიც გამოყენებულია არდუინოში (ATmega168/328) წყვეტების მეტი წყაროები აქვს, მაგრამ არდუინოს თავისებურების გამო მხოლოდ ზემოთ აღნიშნული ორი წყვეტა გამოიყენება. ენთუზიასტების მიერ დაწერილია ბიბლიოთეკები, რომლებიც ემსახურება როგორც ტაიმერის, ასევე ციფრულ საკონტაქტო გამომყვანებზე სიგნალის ცვლილების წყვეტებს. მათი გამოყენება შესაძლებელია ამ ბიბლიოთეკების ინსტალაციის შედეგად.

წყვეტების მომსახურებისთვის არდუინოს თავისი მზა ფუნქციები აქვს: attachInterrupt და DetachInterrupt.

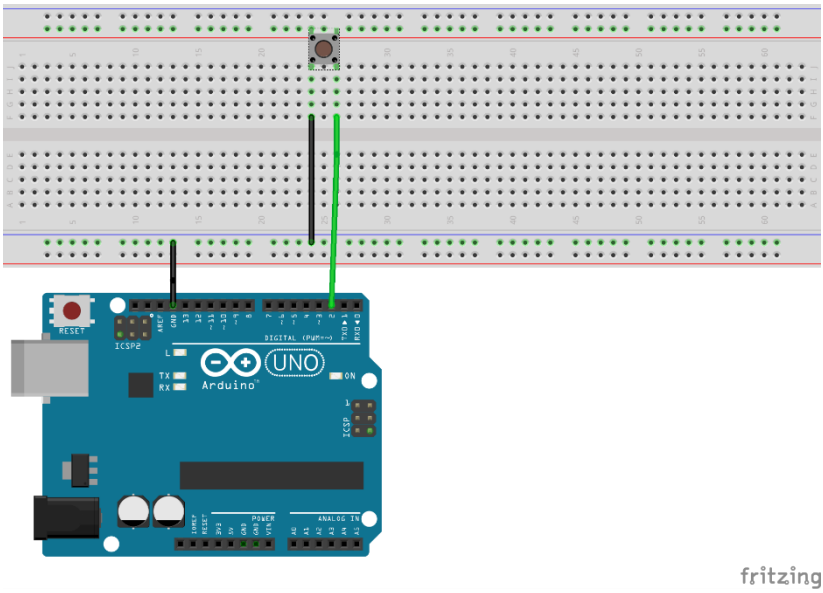
სინტაქსი ბაზური მოდელებისთვის ასეთია: attachInterrupt(digitalPinToInterrupt(pin), ISR, mode); ეს ფუნქცია წყვეტას შესაბამის შეტანა-გამოტანის პორტს აბამს.

attachInterrupt ფუნქციის პარამეტრებია:

- digitalPinToInterrupt(pin) - pin არის მე-2 ან მე-3 საკონტაქტო გამომყვანის ნომერი. წყვეტა შესაძლებელია მხოლოდ ორ ციფრულ შესასვლელზე: 2 და 3.
- ISR - ფუნქციის სახელი, რომელიც უნდა გამოიძახოს მიკროკონტროლერმა წყვეტის გამოჩენისას pin შესასვლელზე.
- Mode - მიუთითებს, თუ როდის უნდა გაჩნდეს წყვეტა:
- LOW- წყვეტა ჩნდება, როდესაც pin შესასვლელზე ნულია ანუ შესაბამის საკონტაქტო გამომყვანზე ნულოვანი პოტენციალი ჩნდება.
- CHANGE - წყვეტა ჩნდება, როდესაც pin შესასვლელზე არის ლოგიკური დონეების ცვლილება, ანუ გადასვლა ნულიდან ერთზე ან - პირიქით.
- RISING - წყვეტა ჩნდება, როდესაც pin შესასვლელზე ლოგიკური დონე ნულიდან ერთზე ადის.
- FALLING წყვეტა ჩნდება, როდესაც pin შესასვლელზე ლოგიკური დონე ერთიდან ნულზე ჩამოდის.

detachInterrupt(digitalPinToInterrupt(pin)) ფუნქცია თიშავს attachInterrupt-ით დაშვებულ წყვეტას.

detachInterrupt ფუნქციის pin პარამეტრი - ეს არის ციფრული შესასვლელი, რომლისთვისაც დაშვებული იყო წყვეტა.



სურ. 3.19. წყვეტა ღილაკის გამოყენებით

```
//შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int ledPin=13;
//არდუინოს საკონტაქტო გამომყვანი, რომელთანაც ღილაკია მიერთებული
const int interruptPin = 2;
//ცვლადი, რომელიც ინახავს ღილაკის მნიშვნელობას
volatile byte state = LOW;

void setup() {
    //გამოვაცხადოთ ledPin, როგორც გამოსასვლელი
   pinMode(ledPin, OUTPUT);
    //გამოვაცხადოთ interruptPin, როგორც შესასვლელი ჩაშენებული PULLUP
    მომჭიმავი წინაღობის აქტივაციით
   pinMode(interruptPin, INPUT_PULLUP);
// მივაბათ წყვეტა  interruptPin-ს, წყვეტის დამუშავების ფუნქციის სახელია blink,
აქტიურდება  interruptPin-ზე  სიგნალის  დონის  ცვლილებისას
```

```
attachInterrupt(digitalPinToInterrupt(interrup
tPin),blink,CHANGE);
}
void loop() {
   //ჩავწეროთ შუქდიოდში მნიშვნელობა
  digitalWrite(ledPin, state);
}
void blink() {
   //წყვეტის წარმოქმნისას  state-ის მნიშვნელობის ცვლილება საპირისპიროთი
  state = !state;
}
```

ამ მაგალითში, მე-13 საკონტაქტო გამომყვანთან დაკავშირებული შუქდიოდი ინთება იმ შემთხვევაში, როდესაც წყვეტა ხდება მეორე საკონტაქტო გამომყვანზე. წყვეტას იწვევს მეორე შესასვლელზე ლოგიკური დონის ყოველი ცვლილება (0–დან 5 ან 5-დან 0 ვოლტი)

უნდა გავითვალისწინოთ, რომ წყვეტის დამუშავების ფუნქცია იყოს რაც შეიძლება მოკლე და არ უნდა გამოვიყენოთ delay() და millis() ფუნქციები. ისინი არ იმუშავებს ამ შემთხვევაში იმის გამო, რომ თვითონ ეს ფუნქციები იყენებს შიგა ტაიმერის წყვეტას.

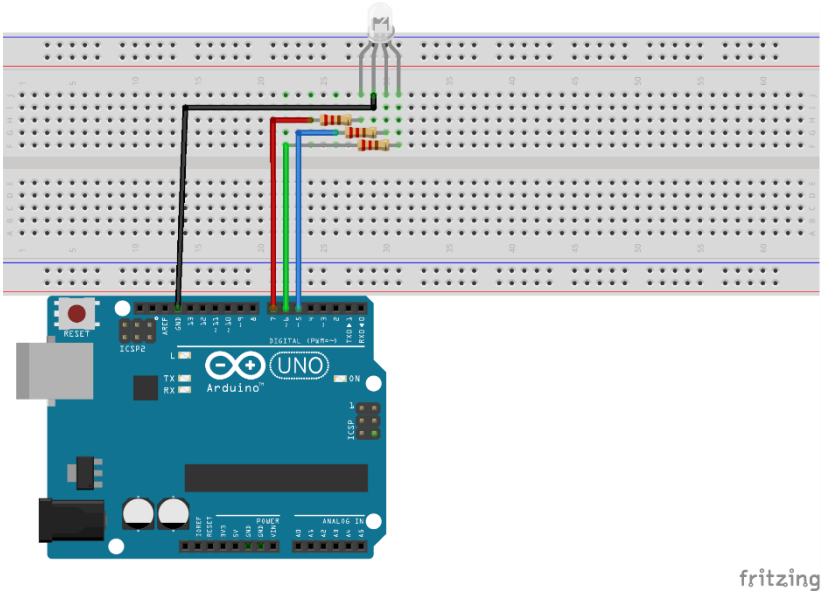
ცვლადები, რომლებიც გამოიყენება წყვეტების დამუშავების ფუნქციებში, უნდა ავღნიშნოთ როგორც volatile. volatile გამოიყენება იმისთვის, რომ ძირითად პროგრამაში და წყვეტის ფუნქციაში არსებული ცვლადების განახლება წყვეტის დამუშავებისას კორექტულად მოხდეს.

უფრო ვრცლად წყვეტების და წყვეტების სხვა ბიბლიოთეკების შესახებ შეგიძლიათ იხილოთ http://playground.arduino.cc/code/interrupts

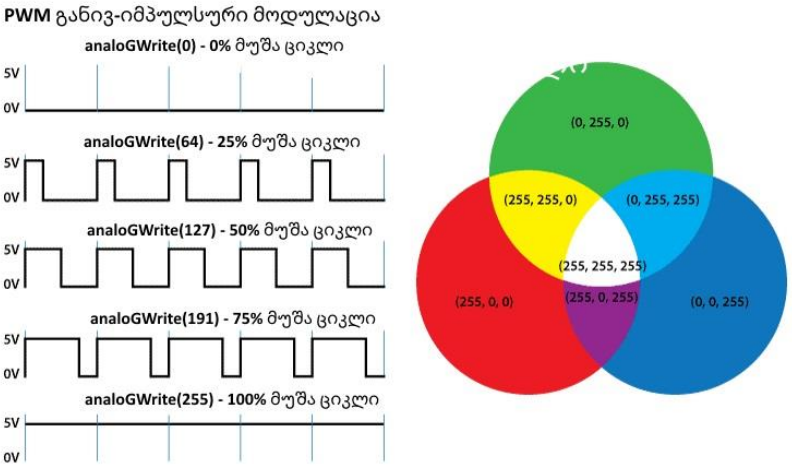
RGB შუქდიოდის ფერების მართვა არდუინოს საშუალებით

RGB შუქდიოდს შეუძლია გამოასხივოს სხვადასხვა ფერი სამი ძირითადი ფერის (წითელი, მწვანე და ლურჯი) შერევით. მასში მოთავსებულია სამი შუქდიოდი. RGB შუქდიოდს ოთხი გამომყვანი აქვს, აქიდან სამი შეესაბამება წითელ, მწვანე და ლურჯ ფერებს და ერთი არის საერთო ანოდი ან კათოდი, იმის მიხედვით თუ რა ტიპის RGB შუქდიოდი გვაქვს. ჩვენი მაგალითის შემთხვევაში, გვაქვს საერთო კათოდი. სურ. 3.20-ზე ნაჩვენებია RGB შუქდიოდის არდუინოსთან შეერთების სქემა.

ანალოგური გამოსასვლელის სიმულაციისთვის გამოვიყენებთ განივ-იმპულსურ მოდულაციას (PWM), რომელიც შუქდიოდისთვის სხვადასხვა დონის ძაბვის მიწოდებას უზრუნველყოფს. ამგვარად, ჩვენ შეგვიძლია მივიღოთ სასურველი ფერი. სურ. 3.21-ზე მოცემულია PWM-ის მნიშვნელობებისა და შუქდიოდის ფერების შესაბამისობა.



სურ.3.20. RGB შუქდიოდის ფერების მართვა



სურ. 3.21. PWM-ის მნიშვნელობებისა და შუქდიოდის ფერების შესაბამისობა

განვიხილოთ პროგრამის კოდი. ჩვენ ვიყენებთ მე-7, მე-6 და მე-5 (redPin, greenPin და bluePin) საკონტაქტო გამომყვანებს. setup განყოფილებაში მათ განვსაზღვრავთ, როგორც გამოსასვლელებს. სკეტჩში გვაქვს ფუნქცია setColor(), რომელსაც აქვს სამი არგუმენტი (redValue, greenValue და blueValue). არგუმენტები წარმოადგენს შუქდიოდების სიკაშკაშეს ანუ PWM სიგნალის მუშა ციკლს, რომელიც იქმნება analogWrite() ფუნქციის გამოყენებით. არგუმენტების მნიშვნელობები იცვლება 0-დან 255 დიაპაზონში. ეს უკანასკნელი წარმოადგენს PWM სიგნალის 100%-იან მუშა ციკლს ან შუქდიოდის სიკაშკაშის მაქსიმუმს.

```
// შუქდიოდებთან მიერთებული საკონტაქტო გამომყვანის ნომრები
const int redPin= 7;
const int greenPin = 6;
const int bluePin = 5;

void setup() {
   // გამოვაცხადოთ redPin, greenPin და bluePin, როგორც გამოსასვლელები
  pinMode(redPin,OUTPUT);
  pinMode(greenPin,OUTPUT);
  pinMode(bluePin,OUTPUT);
}

void loop() {
  setColor(255, 0, 0); //წითელი ფერი
  delay(1000);
  setColor(0, 255, 0); //მწვანე ფერი
  delay(1000);
```

```
  setColor(0, 0, 255); //ლურჯი ფერი
  delay(1000);
  setColor(255, 255, 255);//თეთრი ფერი
  delay(1000);
  setColor(170, 0, 255); //მეწამული ფერი
  delay(1000);
}
void setColor(int redValue,int greenValue,int
blueValue) {
   //ჩავწეროთ redPin, greenPin და bluePin გამომყვანებში მნიშვნელობები
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}
```

loop() ფუნქცია უზრუნველყოფს შუქდიოდის ფერის ყოველ წამში ცვლილებას. იმისათვის, რომ შუქდიოდმა წითლად გაანათოს, setColor() ფუნქცია უნდა გამოვიძახოთ და redValue არგუმენტის მნიშვნელობად დავაყენოთ 255, greenValue-სა და blueValue-ს მნიშვნელობა ამ დროს უნდა იყოს ნული. მსგავსად, შეიძლება მივიღოთ ორი დანარჩენი ძირითადი ფერიც (მწვანე და ლურჯი). სხვა ფერების მიღება შესაძლებელია არგუმენტების მნიშვნელობების ცვლილებით. ასე მაგალითად, თუ სამივე შუქდიოდის სიკაშკაშისთვის ავიღებთ მაქსიმალურ მნიშვნელობებს (255), შუქდიოდზე გვექნება თეთრი ფერი; როგორც პროგრამის კოდიდან შეგიძლიათ ნახოთ, მეწამული ფერის მისაღებად

საჭიროა შემდეგი მნიშვნელობების მითითება: 170-redValue, 0 – greenValue და 255 – blueValue.

მონაცემების შენახვა ენერგოდამოუკიდებელ მეხსიერებაში (EEPROM)

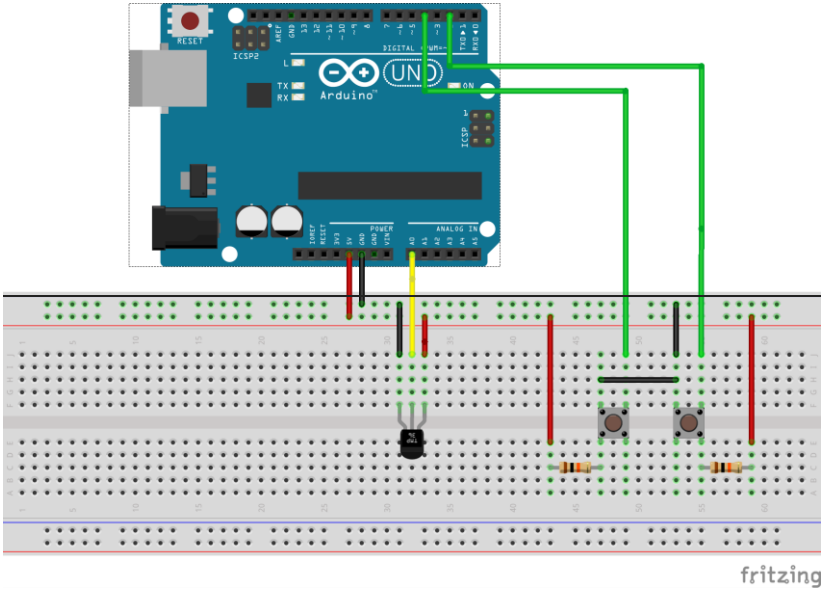
ამ პარაგრაფში ჩვენ განვიხილავთ ენერგოდამოუკიდებელი მეხსიერების ერთ-ერთ სახეს - EEPROM-ს. EEPROM (Electrically Erasable Programmable Read-Only Memory) ელექტრულად წაშლადი დაპროგრამებადი მუდმივი დამმახსოვრებელი მოწყობილობაა. არდუინოს დაფების უმრავლესობაში გამოყენებულ მიკროკონტროლერებს აქვთ 512, 1024 ან 4096 ბაიტი EEPROM მეხსიერება. ეს მეხსიერება ენერგოდამოუკიდებელია, რაც იმას ნიშნავს, რომ მონაცემები არ იშლება, როდესაც არდუინოს დაფას კვება შეუწყდება.

არდუინოს EEPROM შეიძლება განვიხილოთ, როგორც მასივი, სადაც თითოეული ელემენტი არის ერთი ბაიტი. ამ მეხსიერებიდან წაკითხვის და ჩაწერის დროს, მისამართს მასივის ინდექსის სახით ვუთითებთ.

EEPROM-ში მონაცემების ჩაწერის/წაშლის (Write/Erase ციკლი) შესაძლო რაოდენობა შეზღუდულია და ის დაახლოებით 100000-ის ტოლია. ცხადია, EEPROM-ის გამოყენების დროს ამ შეზღუდვის გათვალისწინება აუცილებელია. უნდა ვეცადოთ, EEPROM-ში კოდი ძალიან ხშირად არ

ჩავწეროთ, რადგანაც წაშლის ოპერაციაც ჩაწერის ოპერაციად ითვლება.

ქვემოთ მოცემულია პროგრამა, რომელშიც EEPROM-თან სამუშაო რამდენიმე ფუნქციაა განხილული. ეს პროგრამა კითხულობს მონაცემებს ანალოგურ შესასვლელზე მიერთებული TMP36 ტემპერატურის სენსორიდან და ამ მნიშვნელობებს EEPROM-ში ინახავს ყოველ 2 წამში ერთხელ. ერთი ღილაკის საშუალებით ხდება შენახული მნიშვნელობების მიმდევრობით მონიტორზე გამოტანა, მეორე ღილაკი EEPROM-ში ამ მნიშვნელობებს შლის.



სურ. 3.22. მონაცემების EEPROM-ში შენახვა და წაშლა

```
#include <EEPROM.h>
// EEPROM-დან წაკითხვის ფუნქციის გამოძახებებს შორის დროის მნიშვნელობა
#define SAMPLE_TIME 2000
//ანალოგური საკონტაქტო გამომყვანის ნომერი
const int analogPin = 0;
//ბეჭდვის ღილაკის  საკონტაქტო გამომყვანის ნომერი
const int printPin = 2;
//წაშლის ღილაკის საკონტაქტო გამომყვანის ნომერი
const int erasePin = 4;
//int ტიპის ცვლადი, რომელიც EEPROM-ის მისამართს ინახავს
int address = 0;
//long ტიპის ცვლადი, რომელიც mills() ფუნქციის მნიშვნელობას ინახავს
unsigned long timer;

//ვაცხადებთ ფუნქციების სახელებს
void printAnalog ();
void clearEEPROM();
void writeAnalog();

void setup(){
  //ჩავრთოთ მიმდევრობითი პორტი
  Serial.begin(9600);
  //millis() პროგრამის გაშვებიდან გასულ დროს გვიბრუნებს
  timer=millis();
}

void loop(){
   //შევამოწმოთ, მოვიდა თუ არა ანალოგური საკონტაქტო გამომყვანის
   მნიშვნელობის წაკითხვის  დრო
  if(millis()-timer > SAMPLE_TIME)
```

```
  {
      // ვიძახებთ მეხსიერებაში ჩაწერის ფუნქციას
      writeAnalog();
      // timer -ს ვანიჭებთ მიმდინარე დროის მნიშვნელობას.
      timer = millis();
  }
  // შევამოწმოთ, დაჭერილია თუ არა წაკითხვის ღილაკი
  if(!digitalRead(printPin))
        {
    printAnalog();
    delay(500);
  }
  //შევამოწმოთ, დაჭერილია თუ არა წაშლის ღილაკი
  if(!digitalRead(erasePin))
  {
    clearEEPROM();
    delay(500);
  }
    delay(1);
}

void printAnalog()
{
  for (int i=0 ; i< EEPROM.length(); i++) {
   // წავიკითხოთ EEPROM-ის მნიშვნელობა i მისამართიდან
   byte value = EEPROM.read(i);
   //გამოვტოვოთ ის მისამართები, სადაც არაფერი წერია
   if(value!=0)
    {
```

```
      //გადავიყვანოთ შენახული მნიშვნელობა ტემპერატურაში
      float temp = value * (5.0 / 1024.0);
      temp = (temp - 0.5) * 100;
      // გამოგვყავს მნიშვნელობები მიმდევრობით პორტზე
      Serial.println(temp);
    }
  }
 }

void clearEEPROM()
{
  for (int i = 0; i<EEPROM.length(); i++){
    //გამოვტოვოთ ის მისამართები, სადაც არაფერი წერია
   if(EEPROM.read(i)!=0){
    //ვწერთ 0-ს i მისამართზე
    EEPROM.write(i,0);
    }
}
Serial.println("EEPROM erased");
// გავანულოთ მისამართების მთვლელი
address=0;
}

void writeAnalog (){
  //ვკითხულობთ ანალოგური საკონტაქტო გამომყვანის მნიშვნელობას
  byte value = analogRead(analogPin);
  //ვწერთ წაკითხულ მნიშვნელობას EEPROM-ში მოცემულ მისამართზე
  EEPROM.write(address,value);
  // გამოგვყავს მნიშვნელობები მიმდევრობით პორტზე
  Serial.print("Value stored at address ");
```

```
  Serial.println(address);
  // მისამართის შემცველი ცვლადის ინკრემენტირება
  address++;
  //შევამოწმოთ, მიაღწია თუ არა EEPROM-ის მისამართმა მაქსიმუმს
  if(address == EEPROM.length())
  {
  address = 0;  //თუ კი, მაშინ ვანულებთ მისამართს
  }
}
```

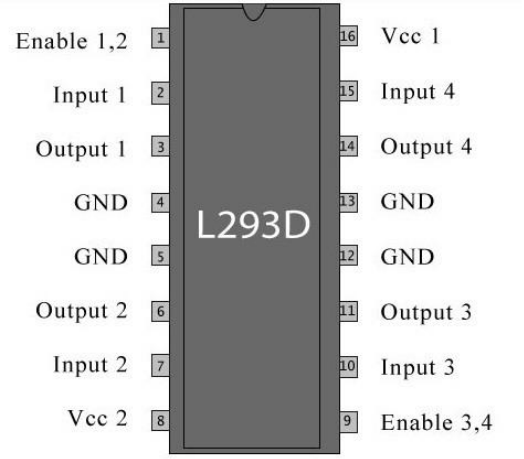
ამ მაგალითში ჩვენთვის საინტერესოა ორი ფუნქცია EEPROM.read() და EEPROM.write(). EEPROM.read() ფუნქცია აბრუნებს int ტიპის მონაცემს, რომელიც შესაბამის მისამართშია მითითებული.

EEPROM.write() ფუნქცია წერს int ან byte მონაცემთა ტიპს მითითებულ მისამართზე. თუ ჩვენ გვინდა, რომ შევინახოთ უფრო მეტი მთელი ტიპის მნიშვნელობები, ვიდრე ეს არის 0-255 დიაპაზონში, თითოეული ჩაწერა/წაკითხვისთვის რამდენიმე მისამართის გამოყენება დაგვჭირდება.

მუდმივი დენის ძრავას მართვა არდუინოს საშუალებით

განვიხილოთ, მუდმივი დენის ძრავას (DC Motor) არდუინოსთან მიერთებისა და მუდმივი დენის ძრავას მართვის საკითხები.

ელექტროძრავას ორი კონტაქტი აქვს. ელექტროძრავას ასამუშავებლად, საჭიროა ერთი კონტაქტი მივუერთოთ კვებას, მეორე - მიწას. ამასთან, თუ პოლაროზას შევცვლით, ძრავა საწინააღმდეგო მხარეს დაიწყებს ტრიალს. ძრავას მიერთება პირდაპირ არდუინოს გამომყვანებზე არ შეიძლება. ამისთვის ძრავის დრაივერია საჭირო. ჩვენ გამოვიყენებთ L293D დრაივერს. იმისათვის, რომ სწორად ჩავრთოთ დრაივერი, მისი საკონტაქტო გამომყვანების დანიშნულება უნდა ვიცოდეთ.



სურ.3.23. L293D დრაივერის საკონტაქტო გამომყვანები

მიაქციეთ ყურადღება, რომ დრაივერს ზედა მხარეს ჩაღრმავება აქვს. ის მიკროსქემის პოზიციონირებისთვისაა საჭირო.

L293D-ს ორი ძრავას მართვა შეუძლია. პირობითად ერთი ნახევარი განკუთვნილია ერთი ძრავისათვის, ხოლო მეორე - მეორისათვის. განვიხილოთ დრაივერის თითოეული საკონტაქტო გამომყვანის დანიშნულება:

Enable 1, 2 - ჩართავს/გამორთავს ძრავას მარცხენა მხრიდან მართვის შესაძლებლობას. უმეტესწილად აზრი აქვს, რომ ის ჩართულ მდგომარეობაში იყოს (მივაწოდოთ 5V).

Input 1 და 2 - უერთდება არდუინოს საკონტაქტო გამომყვანებს და მართავს ძრავას შესაბამისი მხრიდან.

Output 1 და 2 – ამ საკონტაქტო გამომყვანებს უერთდება პირველი ძრავას კონტაქტები.

GND - მიწა.

Vcc 2 - ძრავების კვება (5V ან იმდენი, რამდენიც საჭიროა ძრავებისთვის)

Vcc 1 — დრაივერის კვება (5V).

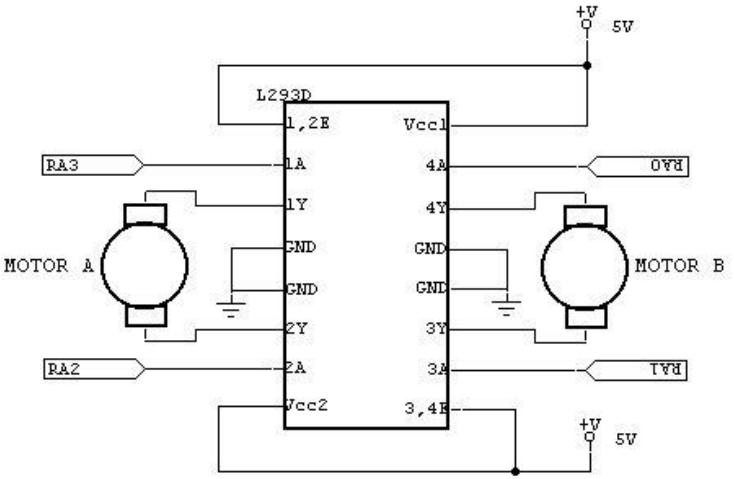
Input 3 და 4 - უერთდება არდუინოს საკონტაქტო გამოსასვლელებს და მართავს ძრავას მუშაობას შესაბამისი მხრიდან.

Output 3 და 4 - ამ საკონტაქტო გამომყვანებს უერთდება მეორე ძრავას კონტაქტები.

Enable 3,4 - ჩართავს/გამორთავს ძრავას მართვის შესაძლებლობას მარჯვენა მხრიდან. უმეტესწილად აზრი აქვს რომ ის ჩართულ მდგომარეობაში იყოს (მივაწოდოთ 5V).

RA 0, 1, 2, 3 - არდუინოს საკონტაქტო გამომყვანებს უერთდება.

სურ. 3.24-ზე ნაჩვენებია L293D-ის ჩართვის სქემა.



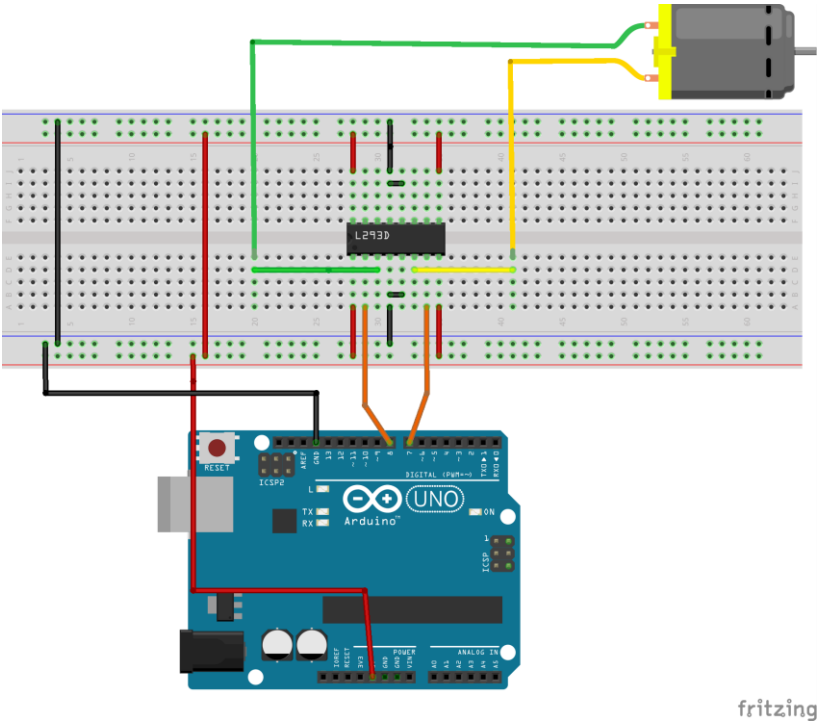
სურ.3.24. L293D-ის ჩართვის სქემა (შეიძლება როგორც მხოლოდ ერთი, ასევე ორი ძრავას ჩართვა)

განვიხილოთ, როგორ ხდება შეერთება და ჯერ ერთი, ხოლო შემდეგ - ორი ძრავას მართვა.

ძრავა შევაერთოთ მე-7 და მე-8 საკონტაქტო გამომყვანებთან. ძრავას ბრუნვის სამართავად გამოიყენება ჩვენთვის უკვე კარგად ნაცნობი digitalWrite ბრძანება, სიგნალი უნდა გავუგზავნოთ თითოეულ საკონტაქტო

გამომყვანს, ამასთან ერთი უნდა იყოს HIGH, ხოლო მეორე - LOW. თუ საჭიროა ძრავას გაჩერება, მაშინ ორივე საკონტაქტო გამომყვანზე უნდა გავაგზავნოთ LOW.

სურ. 3.25-ზე მოცემულია არდუინოს საშუალებით ერთი ძრავას მართვის სქემა.

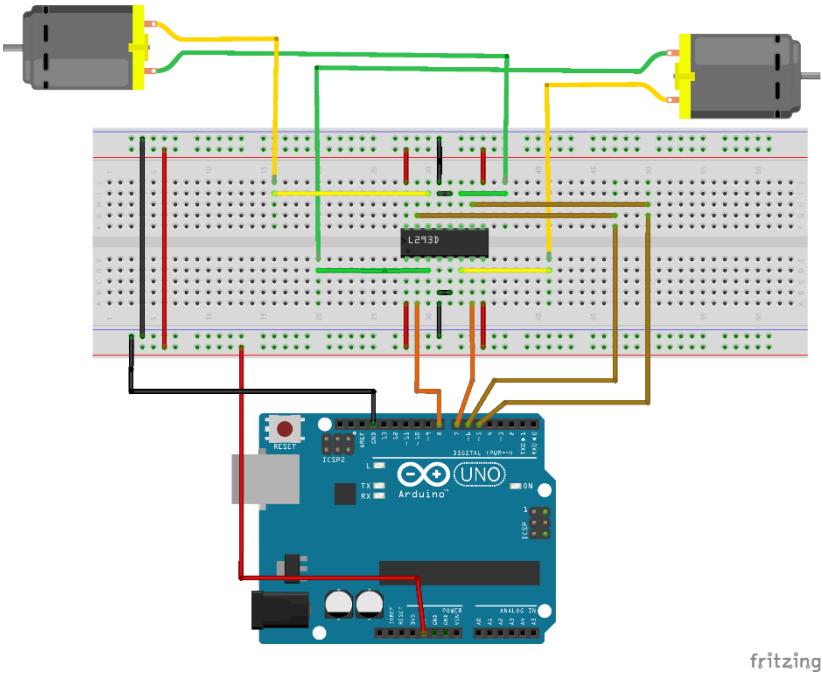


სურ. 3.25. ერთი ძრავას მართვის სქემა

დავწეროთ პროგრამა, რომელიც უზრუნველყოფს ძრავას მობრუნებას რომელიმე ერთი მიმართულებით, შემდეგ აჩერებს მას 1 წამით, შემდეგ აბრუნებს სხვა მიმართულებით და შემდეგ ისევ აჩერებს 1 წამით.

```
const int m1 = 7; //ძრავას კონტაქტი 1
const int m2 = 8; //ძრავას კონტაქტი 2

void setup() {
  // ვაცხადებთ საკონტაქტო გამომყვანებს, როგორც გამოსასვლელებს
  pinMode (m1,OUTPUT);
  pinMode (m2,OUTPUT);
}

void loop() {
  // ვაბრუნებთ ძრავას 1 წამით, ერთი მიმართულებით
  digitalWrite(m1,HIGH);
  digitalWrite(m2,LOW);
  delay(1000);

  //ვაჩერებთ ძრავას 1 წამით
  digitalWrite(m1, LOW);
  digitalWrite(m2, LOW);
  delay(1000);

  // ვაბრუნებთ ძრავას 1 წამით, მეორე მიმართულებით
  digitalWrite(m1, LOW);
  digitalWrite(m2, HIGH);
  delay(1000);

  //ვაჩერებთ ძრავას 1 წამით
  digitalWrite(m1, LOW);
  digitalWrite(m2, LOW);
  delay(1000);
}
```

ახლა მივაერთოთ 2 ძრავა:



სურ. 3.26. ორი ძრავას მართვის სქემა

დავწეროთ პროგრამა, რომლის შესრულების შედეგად ძრავა 2 წამი მოძრაობს ერთი მიმართულებით და შემდეგ საპირისპირო მიმართულებით მოძრაობს 1 წამის განმავლობაში.

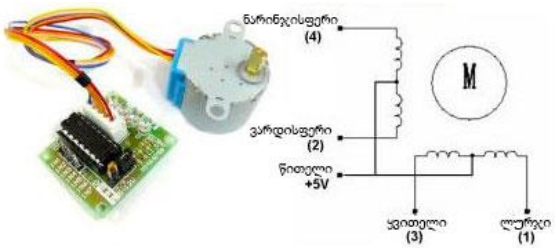
```
const int m11 = 7; // პირველი ძრავას კონტაქტი 1
const int m12 = 8; // პირველი ძრავას კონტაქტი 2
const int m21 = 5; // მეორე ძრავას კონტაქტი 1
const int m22 = 6; // მეორე ძრავას კონტაქტი 2
```

```
void setup() {
  // ვაცხადებთ საკონტაქტო გამომყვანებს, როგორც გამოსასვლელებს
  pinMode (m11, OUTPUT);
  pinMode (m12, OUTPUT);
  pinMode (m21, OUTPUT);
  pinMode (m22, OUTPUT);
}

void loop() {
  // ვაბრუნებთ ორივე ძრავას 2 წამით, ერთი მიმართულებით
  digitalWrite(m11, HIGH);
  digitalWrite(m12, LOW);
  digitalWrite(m21, HIGH);
  digitalWrite(m22, LOW);
  delay(2000);

   // ვაბრუნებთ ორივე ძრავას 1 წამით, მეორე  მიმართულებით
  digitalWrite(m11, LOW);
  digitalWrite(m12, HIGH);
  digitalWrite(m21, LOW);
  digitalWrite(m22, HIGH);
  delay(1000);
 }
```

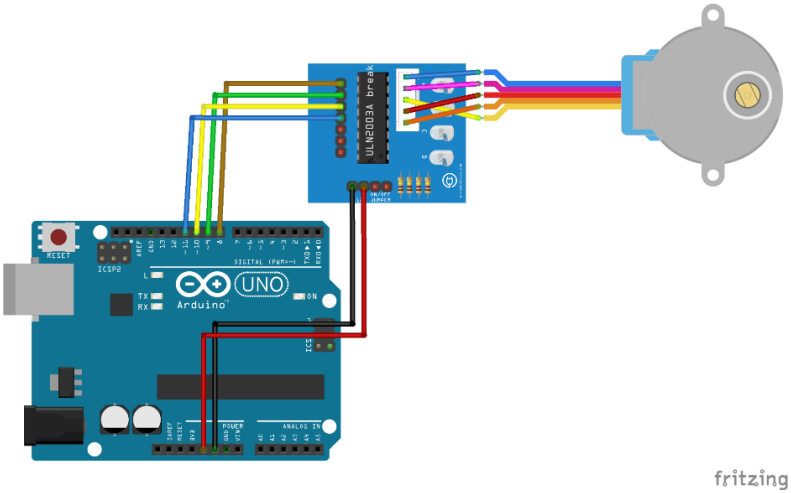
თუ ძრავას ბრუნვის მიმართულება არ არის ისეთი როგორიც ჩვენ გვინდა, მაშინ ძრავას საკონტაქტო გამომყვანებს ადგილები უნდა შევუცვალოთ.

ბიჯური ძრავას მართვა არდუინოს საშუალებით

ბიჯური ძრავა წარმოადგენს რაღაც შუალედურს მუდმივი დენის ძრავასა და სერვოძრავას შორის. ბიჯური ძრავების დიდი უპირატესობაა ის, რომ მათი ზუსტად პოზიციონირება შეიძლება. განვიხილოთ, 28byj-48 ბიჯური ძრავას მართვა არდუინოსა და ULN2003 Breakout ძრავის მართვის დაფის საშუალებით.



სურ.3.27. ბიჯური ძრავა და ძრავას მართვის კონტროლერი



სურ. 3.28. ბიჯური ძრავას მართვა არდუინოს და ძრავას მართვის კონტროლერის საშუალებით

ბიჯურ ძრავას 5 გამომყვანი აქვს. ძრავა მივაერთოთ არდუინოსთან სურ.3.28-ზე მოცემული სქემის მიხედვით. გავითვალისწინოთ, რომ ბიჯურ ძრავას აქვს რედუქტორი კოეფიციენტით 63.68395:1. ძრავას სპეციფიკაციიდან გამომდინარე, მისი ერთი ბიჯით გადაადგილების კუთხე შეადგენს 5.625°-ს, მართვის 8 საფეხურიანი თანმიმდევრობისთვის და 11.25°-ს, მართვის 4 საფეხურიანი თანმიმდევრობისთვის. ბიჯების გამოთვლისთვის საჭიროა ერთი სრული ბრუნის განმავლობაში ბიჯების რაოდენობა გავამრავლოთ რედუქტორის შესაბამის კოეფიციენტზე.

დაფაზე აღნიშვნა	ძრავას გამომყვანები	ბიჯი	ბიჯი	ბიჯი	ბიჯი
საფეხურები		1	2	3	4
-	5 - წითელი	+5v	+5v	+5v	+5v
D	4 - ნარინჯისფერი	0	0	1	1
C	3 - ყვითელი	0	1	1	0
B	2 - ვარდისფერი	1	1	0	0
A	1 - ლურჯი	1	0	0	1

ცხრილი 3.1. მართვის 4 საფეხურიანი თანმიმდევრობა

ზემოთ აღნიშნულიდან გამომდინარე, მართვის 8 საფეხურიანი თანმიმდევრობისთვის ბიჯების რაოდენობა ტოლი

იქნება (360°/5.625°)*63.68395=64*63.68395=4075.77; მართვის 4 საფეხურიანი თანმიმდევრობისთვის, ბიჯების რაოდენობა ტოლი იქნება (360°/11.25°)*63.68395=32 * 63.68395 =2037.88, დავამრგვალოთ მიღებული მნიშვნელობები 4076-მდე და 2038-მდე, შესაბამისად. ბიბლიოთეკა, რომელსაც ჩვენ გამოვიყენებთ ძრავას მართვისთვის, 4 საფეხურიანი მართვის თანმიმდევრობისთვისაა განკუთვნილი.

ქვემოთ მოცემულ სკეტჩში ძრავა ასრულებს სრულ ბრუნს ჯერ საათის ისრის და შემდეგ საწინააღმდეგო მიმართულებით და დაუსრულებლად იმეორებს ამ ციკლს.

```
// ბიჯური ძრავას სტანდარტული ბიბლიოთეკა
#include <Stepper.h>
// ბიჯური ძრავას მართვის დაფასთან მისაერთებელი გამომყვანები
const int in1Pin = 8;
const int in2Pin = 9;
const int in3Pin = 10;
const int in4Pin = 11;
// ბიჯური ძრავას ბიჯების რაოდენობა 1 სრული ბრუნისთვის
const int stepsPerRevolution = 2038;
// ბიჯური ძრავას პარამეტრების განსაზღვრა
Stepper
motor(stepsPerRevolution,in1Pin,in2Pin,in3Pin,
in4Pin);
void setup()
{
  // ვაცხადებთ საკონტაქტო გამომყვანებს, როგორც გამოსასვლელებს
  pinMode(in1Pin,OUTPUT);
  pinMode(in2Pin,OUTPUT);
```

```
  pinMode(in3Pin,OUTPUT);
  pinMode(in4Pin,OUTPUT);
 // მიმდევრობითი პორტის პარამეტრები
  Serial.begin(9600);
 // ძრავას სიჩქარე (ბრუნი /წუთში)
  motor.setSpeed(60);
}
 void loop() {
    // ძრავა ასრულებს ერთ სრულ ბრუნს საათის ისრის მიმართულებით.
     Serial.println("clockwise");
     myStepper.step(stepsPerRevolution);
     //დაყოვნება 500 მწმ
     delay(500);
     //ძრავა ასრულებს ერთ სრულ ბრუნს საათის ისრის მოძრაობის
     საწინააღმდეგო მიმართულებით
     Serial.println("counterclockwise");
     myStepper.step(-stepsPerRevolution);
     //დაყოვნება 500 მწმ
     delay(500);
     }
```

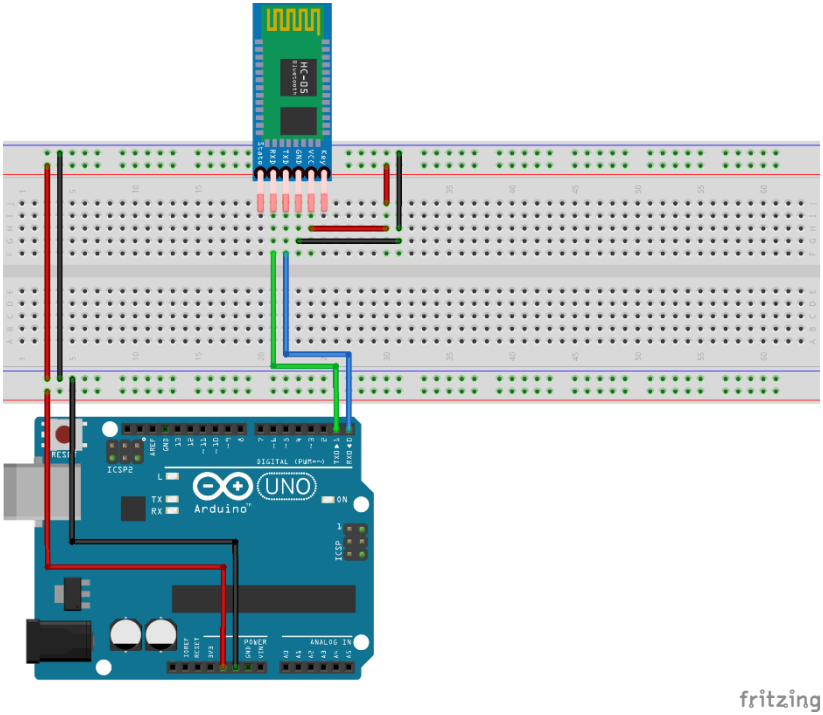
პროგრამის კოდში Stepper ბიბლიოთეკის ჩართვის შემდეგ, განისაზღვრება მართვის ოთხი საკონტაქტო გამომყვანი - in1Pin, in2Pin, in3Pin, in4Pin. იმისათვის, რომ არდუინოს Stepper ბიბლიოთეკას გავაგებინოთ, რომელი საკონტაქტო გამომყვანია მიერთებული ძრავას კონტრო-ლერთან, შემდეგი ბრძანება უნდა დავწეროთ:

Stepper motor(2038, in1Pin, in2Pin, in3Pin, in4Pin); პირველი პარამეტრი არის ბიჯების რაოდენობა, რომელსაც ძრავა

ასრულებს ერთი ბრუნის დასასრულებლად. ბიჯური ძრავას მოძრაობის სიჩქარეს ვაყენებთ motor.setSpeed(60) ბრძანებით. იმის გამო, რომ ძრავას აქვს რედუქტორი სიჩქარე გამოცხადებულზე მცირე იქნება.

HC-06 ბლუთუზ მოდულის მიერთება არდუინოსთან

სხვადასხვა ამოცანის განხორციელებისას, ხშირად წარმოიქმნება დისტანციურად მართვის ან სატელეფონო გაჯეტებიდან მონაცემების გადაცემის აუცილებლობა.



სურ. 3.29. ბლუთუზ მოდულის მიერთება არდუინოსთან

მონაცემების მიმოცვლისა და გავრცელების ერთ-ერთი ყველაზე პოპულარული მეთოდია მონაცემების გადაცემა ბლუთუზის მეშვეობით.

ჩვენ განვიხილავთ მარტივ მაგალითს, როგორ შეიძლება მივუერთოთ ბლუთუზის მოდული არდუინოს და განვახორციელოთ დისტანციური მართვა ტელეფონიდან. ამისათვის დაგვჭირდება Arduino Uno, გამტარების ნაკრები, HC-06 ბლუთუზი.

ბლუთუზის არდუინოსთან შეერთება შემდეგნაირად ხდება:

არდუინო	ბლუთუზი
საკონტაქტო გამომყვანი 1 (TX)	RXD
საკონტაქტო გამომყვანი 0 (RX)	TXD
GND	GND
5V	VCC

ცხრილი 3.2. ბლუთუზის არდუინოსთან შეერთება

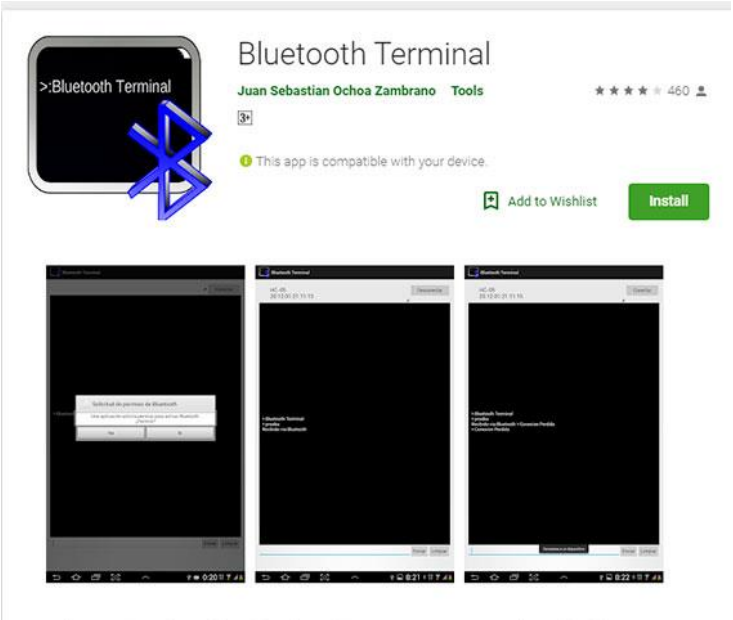
სკეტჩის ჩატვირთვის დროს ბლუთუზის მოდული არდუინოსთან შეერთებული არ უნდა იყოს. წინააღმდეგ შემთხვევაში, სკეტჩი არ ჩაიწერება მიკროკონტროლერში. მიზეზი ის არის, რომ ბლუთუზ მოდულთან კავშირი

ხორციელდება RX და TX გამომყვანების საშუალებით, რომელსაც ასევე იყენებს Arduino IDE სკეტჩის ჩასაწერად.

```
int val;
// შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int LED = 13;
void setup()
{
  // მიმდევრობითი პორტის ჩართვა
   Serial.begin(9600);
  // საკონტაქტო გამომყვანი LED-ი, როგორც  გამოსასვლელი,  მასში ვწერთ
   ლოგიკურ 1-ს
   pinMode(LED,OUTPUT);
    digitalWrite(LED,HIGH);
}
void loop()
{
  //თუ მიმდევრობით პორტში გვაქვს ინფორმაცია
  if (Serial.available())
   {
    // ვანიჭებთ მიმდევრობითი პორტით მიღებულ სიდიდეს val
    ცვლადს და ვადარებთ  მის მნიშვნელობას 1-ს და 0-ს
    val = Serial.read();
     if (val == '1')
      {
     //თუ 1-ის ტოლია, ვწერთ მასში ლოგიკურ 1-ს (შუქდიოდი ინთება)
      digitalWrite(LED, HIGH);
      }
      if ( val == '0')
      {
```

```
    //თუ 0-ის ტოლია, ვწერთ მასში ლოგიკურ 0-ს (შუქდიოდი ქრება)
     digitalWrite(LED, LOW);
     }
   }
}
```

მას შემდეგ, რაც სკეტჩი ჩაიწერება და ბლუთუზ მოდული არდუინოს მიუერთდება, შეიძლება შემდეგ ეტაპზე გადავიდეთ. ვრთავთ ბლუთუზს ტელეფონში და ახალ მოწყობილობას ვეძებთ, სიაში ვპოულობთ HC-06-ს და ვუერთდებით მას. ტელეფონი მოითხოვს პინ-კოდს, შეგვყავს „1234“ ან „0000“. ამის შემდეგ, საჭიროა Bluetooth Terminal აპლიკაციის ტელეფონში ჩამოტვირთვა.



სურ.3.30. Bluetooth Terminal-ის ჩამოტვირთვა (Android პლატფორმა)

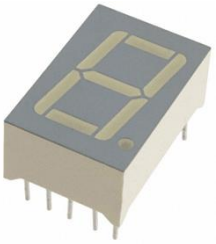
Bluetooth Terminal-ის რამდენიმე ვერსია არსებობს. როგორც წესი, ისინი განსხვავდება დიზაინით, ფუნქციონალი კი ერთი და იგივე აქვთ.

ტერმინალის დაყენების შემდეგ, მას ვუშვებთ, ვირჩევთ ბლუთუზ მოდულს (HC-06) და ვამყარებთ კონტაქტს.

ახლა უკვე შეგვიძლია შევამოწმოთ, მუშაობს თუ არა პროგრამა. ვწერთ ტერმინალში ციფრ „0“-ს და ვაგზავნით. L შუქდიოდი, რომელიც არდუინოს დაფაზე მე-13 საკონტაქტო გამომყვანთანაა დაკავშირებული, უნდა ჩაქრეს. თუ გავაგზავნით ტერმინალიდან ციფრ „1“-ს, მაშინ L შუქდიოდი ისევ უნდა აინთოს.

7-სეგმენტა ინდიკატორის მართვა არდუინოს საშუალებით

7-სეგმენტა ინდიკატორი წარმოადგენს შუქდიოდების ნაკრებს, სულ არის 8 შუქდიოდი. ისინი განლაგებულია განსაზღვრული მიმდევრობით, რაც საშუალებას იძლევა ინდიკატორზე გამოტანილ იქნეს ციფრები 0-დან 9-ის ჩათვლით.



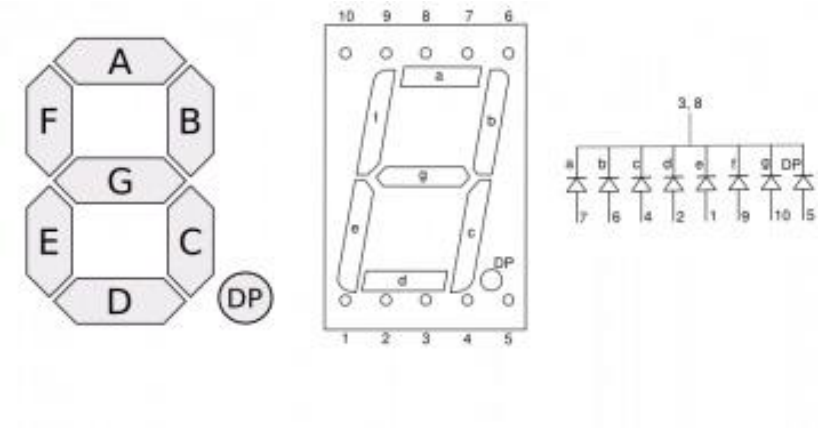
სურ. 3.31. 7-სეგმენტა ინდიკატორი

ინდიკატორს აქვს საერთო საკონტაქტო გამომყვანები (მე-3 და მე-8) კათოდისთვის ან ანოდისთვის. ეს ძალიან მოსახერხებელია, თითოეული კათოდიდან ან ანოდიდან ცალკე გამტარის მიწასთან ან +5V-თან მიერთება არ დაგვჭირდება. საკმარისი იქნება ავირჩიოთ რომელიმე ერთი საერთო კათოდიდან ან ანოდიდან და მივუერთოთ შესაბამის კვების წყაროს. საერთო ანოდიან ინდიკატორს აქვს ცალ-ცალკე კათოდების გამომყვანები და პირიქით, საერთო კათოდიან ინდიკატორს აქვს ცალ-ცალკე ანოდების გამომ-ყვანები.

სურვილისამებრ, ჩვენ შეგვიძლია დავაყენოთ რამდე-ნიმე ასეთი ინდიკატორი მიმდევრობით ორნიშნა, სამნიშნა და ა.შ. რიცხვების გამოსატანად. თუმცა, ასეთი მიზნე-ბისთვის მზა კომპაქტური ნაკრებები არსებობს.

7-სეგმენტა ინდიკატორზე იგივე წესები ვრცელდება, რაც სტანდარტულ შუქდიოდებზე - თითოეულს თავისი რეზისტორი უნდა ჰქონდეს. წერტილის შესაბამის სეგმენტსაც სჭირდება რეზისტორი და სწორედ ამიტომ, დამატებით კიდევ ერთი რეზისტორი დაგვჭირდება.

სქემატურად 7-სეგმენტა ინდიკატორი შეიძლება წარმოვადგინოთ სურ. 3.32-ზე მოცემული სახით:

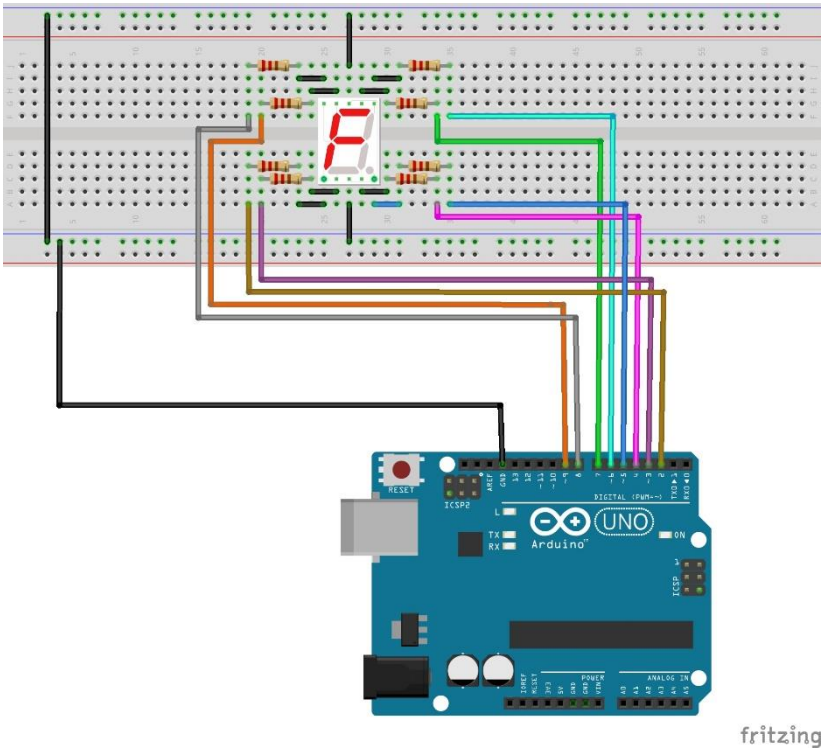


სურ. 3.32. 7-სეგმენტა ინდიკატორის სქემატური გამოსახულება და პრინციპიალური სქემა

არდუინო	ინდიკატორი
2	1(E)
3	2(D)
4	4(C)
5	5(DP)
6	6(B)
7	7(A)
8	9(F)
9	10(G)

ცხრილი 3.3. 7-სეგმენტა ინდიკატორისა და არდუინოს გამომყვანების შეერთება

ავაწყოთ სქემა:



სურ.3.33. 7-სეგმენტა საერთო კათოდიანი ინდიკატორის არდუინოსთან მიერთების სქემა

შემოწმების მიზნით შეიძლება გავუშვათ სტანდარტული მაგალითი Blink. მაგალითად, ავაციმციმოთ მეხუთე საკონტაქტო გამომყვანზე მიერთებული სეგმენტი (წერტილი).

თუ ჩვენ გვინდა ვაციმციმოთ ციფრი 1, მაშინ უნდა გამოვიყენოთ მე-4 და მე-6 შუქდიოდები, ისინი უერთდება არდუინოს მე-4 და მე-6 საკონტაქტო გამომყვანებს.

```
int led4 = 4;
int led6 = 6;
```

თუ ჩვენ გვინდა გამოვიტანოთ ციფრი 5, მაშინ უნდა ავანთოთ 5 შუქდიოდი, 8-ის გამოსატანად კი - 7 შუქდიოდის ანთება მოგვიწევს. რთული ამოცანების შემთხვევაში, ასეთ რაოდენობასთან მუშაობა ცოტა რთულია. მოგვიწევს, ყოველ ჯერზე შევხედოთ სქემას, რათა გავიხსენოთ, რომელი შუქდიოდები უნდა ჩავრთოთ თითოეული ციფრის გამოსატანად.

არსებობს ამოცანის გადაჭრის უფრო მარტივი გზა. ამისთვის გამოვიყენებთ ინფორმაციის ერთეულს - ბაიტს. ბაიტი თავის ორობით წარმოდგენაში შედგება 8 ბიტისგან. თითოეულ ბიტს შეუძლია მიიღოს მნიშვნელობა 0 ან 1. ჩვენი შუქდიოდური ინდიკატორი სწორედ რვა შუქდიოდისგან შედგება. ამგვარად, შეგვიძლია ციფრი ინდიკატორზე წარმოვიდგინოთ ბიტების ნაკრების სახით, სადაც ერთიანი ნიშნავს ჩართულ დიოდს, ხოლო ნული - გამორთულს, საერთო კათოდიანი 7-სეგმენტა ინდიკატორისთვის. რიცხვი ორობით კოდში ჩაიწერება შემდეგნაირად: 0b00000000. პირველი ორი სიმბოლო 0b გვიჩვენებს, რომ საუბარია ორობით თვლის სისტემაზე. დანარჩენი ნულები აღნიშნავს, რომ ყველა შუქდიოდი გამორთულია. ეს

სამართლიანია საერთო კათოდის მქონე 7 სეგმენტა ინდიკატორისთვის. საერთო ანოდის მქონე ინდიკატორისთვის ნულები ნიშნავს, რომ ყველა სეგმენტი ჩართულია. ჩვენ შემთხვევაში, ვიყენებთ საერთო კათოდის მქონე ინდიკატორს.

ჩვენ ვიყენებთ საკონტაქტო გამომყვანებს მეორედან მეცხრეს ჩათვლით. მეორე საკონტაქტო გამომყვანი იწერება ყველაზე მარჯვნივ მდებარე პოზიციაში. მის ჩასართავად, დავწეროთ ერთიანი შესაბამის ადგილას: 0b00000001. შესაძლებელია დამოუკიდებლად ჩავრთოთ ცალ-ცალკე თითოეული სეგმენტი (დიოდი), ერთიანის წანაცვლების გზით, წარმოდგენილ ბაიტში. თუ დავაკვირდებით, მივხვდებით, წერტილის შესაბამისი სეგმენტის ასანთებად, მარჯვნიდან მეოთხე ბიტის ადგილას უნდა დავწეროთ ერთიანი. თუ ჩვენ მას არ გამოვიყენებთ, მაშინ ის ყოველთვის იქნება 0. 7-სეგმენტა ინდიკატორის შუა სეგმენტის გასანათებლად მარცხნიდან პირველ თანრიგში უნდა ჩაიწეროს ერთიანი. ნულებისა და ერთების კომბინაციით, შეიძლება შევქმნათ ჩვენთვის საჭირო ციფრი. მაგალითად, ციფრი 0 წარმოიდგინება, როგორც 0b01110111.

დავწეროთ პროგრამა, რომლის საშუალებითაც ინდიკატორზე გამოვა ციფრი 0.

```
#define FIRST_SEGMENT_PIN 2
#define SEGMENT_COUNT 8
byte number0 = 0b01110111;
```

```
void setup() {
  for (int i = 0; i < SEGMENT_COUNT; ++i)
    pinMode(i + FIRST_SEGMENT_PIN, OUTPUT);
}

void loop() {
  //ყოველი სეგმენტისთვის ვაცხადებთ: უნდა იყოს თუ არა ეს სეგმენტი
  ჩართული. ამისთვის ვიღებთ ბიტს, რომელიც შეესაბამება მიმდინარე  i
  სეგმენტს და ვწერთ მასში 1 -ს, თუ  ჩართულია და 0-ს, თუ გამორთულია
  for (int i = 0; i < SEGMENT_COUNT; ++i) {
  // ვრთავთ ან ვთიშავთ სეგმენტს enableSegment-ის მნიშვნელობის მიხედვით
  digitalWrite(i+FIRST_SEGMENT_PIN,
                  enableSegment);
  }
}
```

როგორც პროგრამის კოდიდან ჩანს, ციკლის მმართველ ცვლადად გამოყენებულია შუქდიოდების რაოდენობა და თითოეული შუქდიოდისთვის შესაბამისი საკონტაქტო გამომყვანები დაყენებულია OUTPUT რეჟიმში. ამის შემდეგ, ისევ ციკლში გავდივართ ყველა შუქდიოდს, ვანთებთ შესაბამის სეგმენტებს და ინდიკატორზე ვიღებთ ციფრ 0-ს.

დანარჩენი ციფრებისთვის ასევე შეგვიძლია განვსაზღვროთ ბიტების საჭირო კომბინაცია.

byte number1 = 0b00010100;

byte number2 = 0b10110011;

... და ასე შემდეგ.

7-სეგმენტა ინდიკატორის საშუალებით შევქმნათ უმარტივესი წამმზომი.

ამოცანის გადაწყვეტა მარტივდება მასივის გამოყენებით. millis მეთოდის საშუალებით მივიღებთ წამების იმ რაოდენობას, რომელიც პროგრამის გაშვების მომენტიდან გავიდა, გამოტანით კი გასული წამების მხოლოდ ბოლო ციფრს გამოვიტანთ.

```
#define FIRST_SEGMENT_PIN    2
#define SEGMENT_COUNT        8

//სულ 10 ციფრია, ამიტომაც მასივიც 10 ციფრს შეიცავს
byte numberSegments[10] = {
  0b01110111,
  0b00010100,
  0b10110011,
  0b10110110,
  0b11010100,
  0b11100110,
  0b11100111,
  0b00110100,
  0b11110111,
  0b11110110,
};

void setup()
{
  for (int i = 0; i < SEGMENT_COUNT; ++i)
    pinMode(i + FIRST_SEGMENT_PIN, OUTPUT);
}
```

```
void loop()
{
  //განვსაზღვრავთ ციფრს, რომელიც უნდა გამოჩნდეს ინდიკატორზე
  int number = (millis() / 1000) % 10;
  //მასივიდან ვიღებთ მიღებული ციფრის მნიშვნელობას
  int mask = numberSegments[number];
  // შვიდივე სეგმენტისთვის ვსაზღვრავთ ჩართული უნდა იყოს ეს სეგმენტი თუ გამორთული
  for (int i = 0; i < SEGMENT_COUNT; ++i) {
    //მიღებული მნიშვნელობის მიხედვით ვრთავთ ან ვთიშავთ სეგმენტს
    boolean enableSegment = bitRead(mask, i);
    digitalWrite(i+FIRST_SEGMENT_PIN,
                 enableSegment);
    byte currentPinMask = b10000000;
  for (int i = 2; i <= 8; i++) {
    if(mask&currentPinMask)
      digitalWrite(i,HIGH);
     else
       digitalWrite(i, LOW);
  currentPinMask = currentPinMask >> 1;
  }
 }
}
```

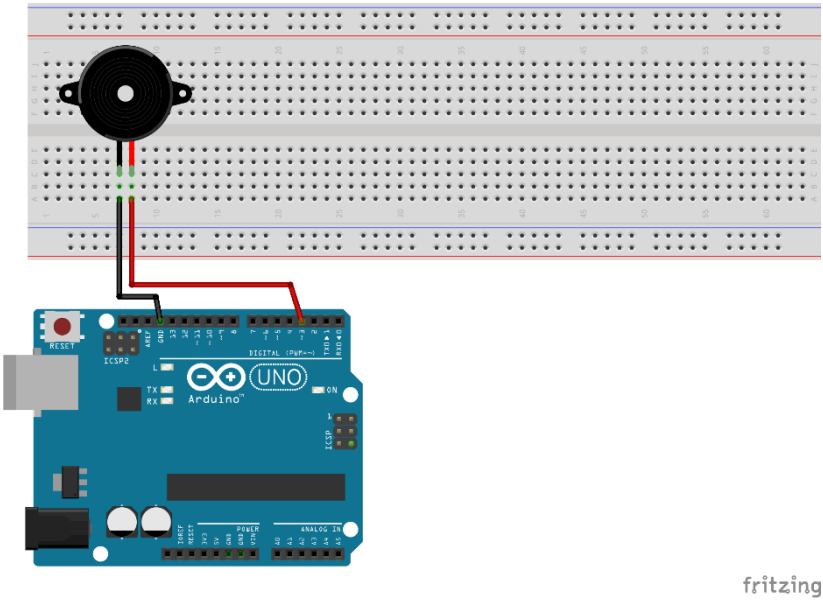
პროგრამის კოდის გაშვების შედეგად, მივიღებთ რეალურ წამზომს.

პიეზოდინამიკის მიერთება არდუინოსთან

პიეზოდინამიკი - ელექტრომექანიკური გარდამქნელია, რომლის ერთ-ერთ სახესხვაობას ბგერის პიეზოგამომსხივებელი წარმოადგენს. მას ხშირად პიეზოდინამიკსაც (ინგლ. buzzer) უწოდებენ. პიეზოდინამიკი ელექტრულ ძაბვას მემბრანის რხევებში გარდაქმნის. სწორედ ეს რხევები ქმნიან ბგერით ტალღას.

ჩვენს ამოცანას წარმოადგენს პიეზოდინამიკის არდუინოსთან მიერთება და ბგერის სიხშირის რეგულირება, პროგრამაში შესაბამისი პარამეტრების ცვლილების გზით.

პიეზოდინამიკის არდუინოსთან მიერთების სქემა მოცემულია სურ. 3.34-ზე:



სურ. 3.34. პიეზოდინამიკის არდუინოსთან მიერთების სქემა

ამ სქემის ასამუშავებლად დავწეროთ პროგრამა:

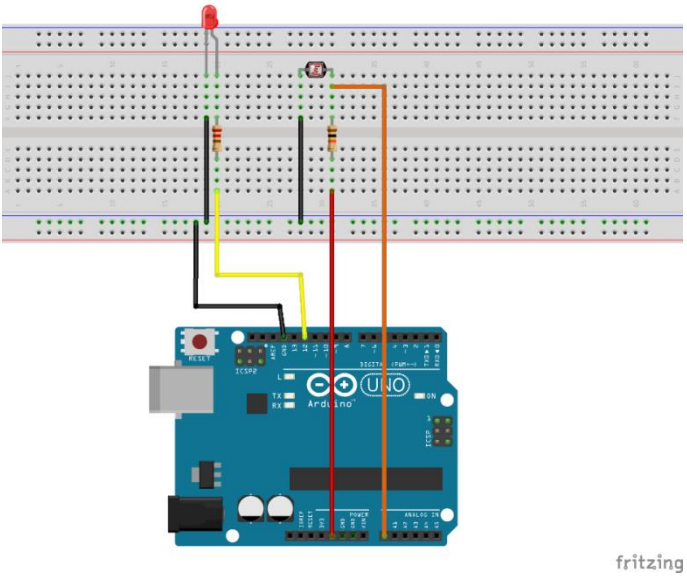
```
// პიეზოდინამიკთან მიერთებული გამომყვანის ნომერი
const int p=3;

void setup(){
  pinMode(p, OUTPUT);
}
void loop()
{
//ვრთავთ 500 ჰც სიხშირით
tone (p, 500);
//ველოდებით 100 მილიწამი
delay(100);
//ვრთავთ 1000 ჰც სიხშირით
tone(p,1000);
//ველოდებით 100 მილიწამი
delay(100);
}
```

აქ ვიყენებთ არდუინოს `tone(pin,frequency)` ფუნქციას, რომელიც `pin` გამომყვანზე ახდენს სწორკუთხა ტალღების გენერაციას, რომლის სიხშირეც არის frequency.

პროგრამიდან ჩანს, რომ იგი ახდენს ორი სხვადასხვა ტონის გენერაციას 100-100 მილიწამი დაყოვნებით.

ფოტორეზისტორის მიერთება არდუინოსთან

ფოტორეზისტორი წარმოადგენს რეზისტორს, რომლის წინაღობა დამოკიდებულია მასზე დაცემული სინათლის სიკაშკაშეზე. ის ფაქტიურად, განათებულობის სენსორს წარმოადგენს. ქვემოთ განხილულ ამოცანაში შუქდიოდი ანათებს, მაშინ როდესაც ფოტორეზისტორზე სინათლის სიკაშკაშე რაიმე წინასწარ განსაზღვრულ მნიშვნელობაზე ნაკლებია. ამ სიკაშკაშის რეგულირება შესაძლებელია პროგრამულად. ამოცანის რეალიზებისათვის დაგვჭირდება არდუინოს დაფა, 6 გამტარი, ფოტორეზისტორი, შუქდიოდი, 220Ω და 10kΩ წინაღობის რეზისტორები. არდუინოსთან ფოტორეზისტორის შეერთების სქემა მოცემულია სურ. 3.35-ზე.



სურ.3.35. ფოტორეზისტორის არდუინოსთან შეერთების სქემა

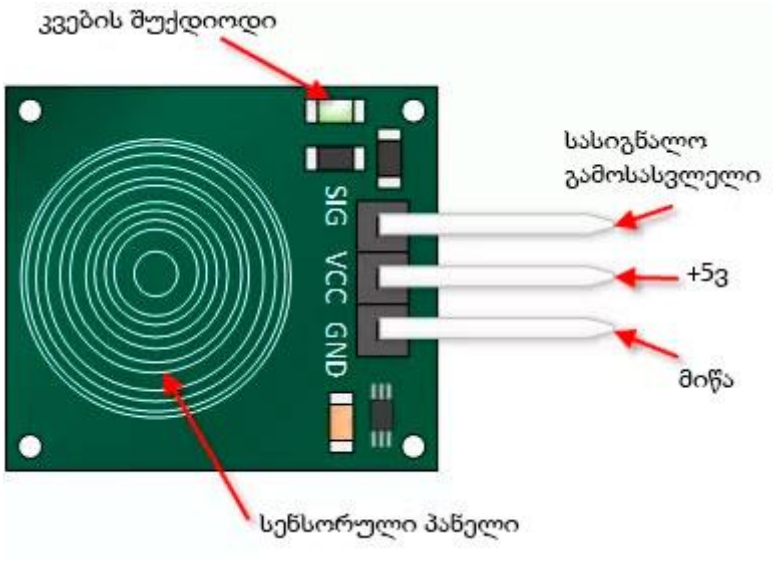
ამ სქემის ასამუშავებლად დავწეროთ პროგრამა:

```
// შუქდიოდთან მიერთებული გამომყვანის ნომერი
const int led = 12;
// ფოტორეზისტორთან მიერთებული გამომყვანის ნომერი
const int ldr = 0;
 void setup()
{
  pinMode(led, OUTPUT);
}

void loop()
{
   //თუ განათებულობის მაჩვენებელი 800-ზე ნაკლებია, მაშინ შუქდიოდს ვრთავთ
   if (analogRead(ldr)<800)
    digitalWrite(led, HIGH);
   else
     //წინააღმდეგ შემთხვევაში, შუქდიოდს ვთიშავთ
     digitalWrite(led,LOW);
}
```

თუ შუქდიოდი განათებულობის ცვლილებაზე არ რეაგირებს, პროგრამაში რიცხვი 800 სხვა რიცხვით უნდა შევცვალოთ. თუ შუქდიოდი მუდმივად ანათებს - ეს რიცხვი უნდა შევამციროთ, ხოლო თუ არ ანათებს, მაშინ - გავზარდოთ.

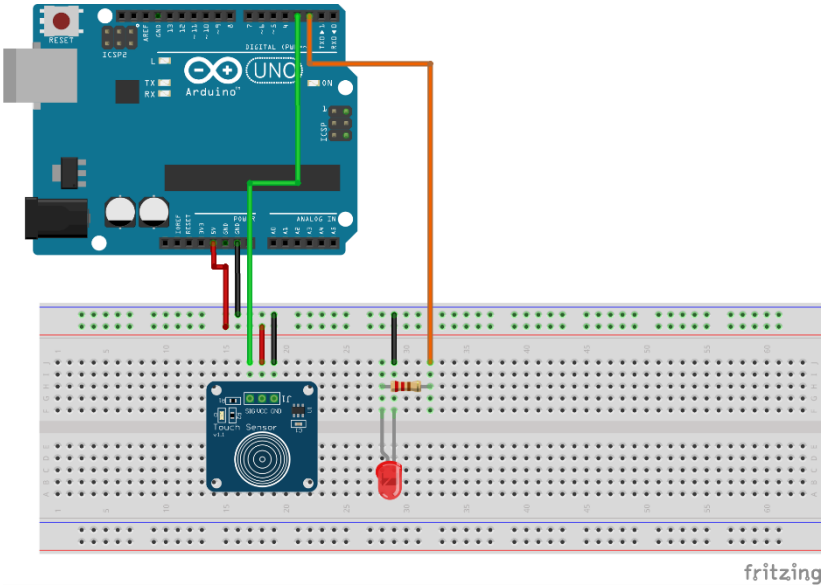
შეხების სენსორი

განვიხილოთ TTP223B შეხების ციფრული ტევადური სენსორი. ის ნებისმიერი სახის მიკროკონტროლერს შეგვიძლია მარტივად მივუერთოთ, შეიცავს მხოლოდ სამ ტერმინალს გარე ინტერფეისისთვის. შეხების სენსორი ადამიანის სხეულს იყენებს, როგორც სქემის ნაწილს. როდესაც სენსორის პანელთან ხდება შეხება, წრედის ტევადობა იცვლება და სწორედ, ამის დაფიქსირება ხდება. ტევადობის ეს ცვლილება იწვევს გამოსასვლელზე მდგომარეობის ცვლილებას.



სურ.3.36. შეხების სენსორის სქემა

ქვემოთ განხილულ ამოცანაში შუქდიოდი ანათებს, მაშინ როდესაც შეხების სენსორზე შეხება ფიქსირდება.



სურ.3.37. შეხების სენსორის არდუინოსთან შეერთების სქემა

კოდი:

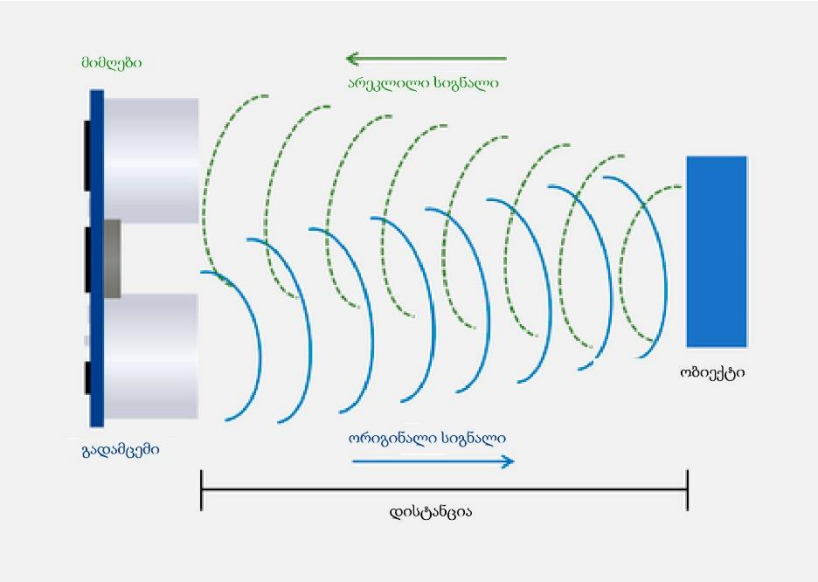
```
/*
არდუინო        შეხების სენსორი
   5V             VCC
   GND            GND
   D3             SIG
*/
const int TouchSensor=3;//შეხების სენსორთან მიერთებული საკონტაქტო გამომყვანის ნომერი
const int led=2; //შუქდიოდთან მიერთებული საკონტაქტო გამომყვანის ნომერი

void setup(){
   // მიმდევრობითი პორტის ჩართვა
   Serial.begin(9600);
```

```
  // საკონტაქტო გამომყვანი LED,  როგორც გამოსასვლელი
   pinMode(led, OUTPUT);
  // საკონტაქტო გამომყვანი TouchSensor, როგორც შესავლელი
   pinMode(TouchSensor, INPUT);
   }

void loop(){
   //ვკითხულობთ შეხების სენსორის გამომყვანის მნიშვნელობას
   if(digitalRead(TouchSensor)==HIGH)
   {
   digitalWrite(led, HIGH);
    //თუ გამომყვანზე მაღალი ლოგიკური დონეა, მაშინ შეხებაა დაფიქსირებული
    და ავანთოთ შუქდიოდი
     Serial.println("Led ON");
   }
   else
   {
     digitalWrite(led, LOW);
     //თუ გამომყვანზე დაბალი ლოგიკური დონეა, მაშინ შეხება არ არის
     დაფიქსირებული და ჩავაქროთ შუქდიოდი
      Serial.println("Led OFF");
    }
   //დაყოვნება შეხების სენსორის შემოწმებებს შორის
   delay(50);
}
```

HC-SR04 ულტრაბგერითი სენსორი და არდუინო

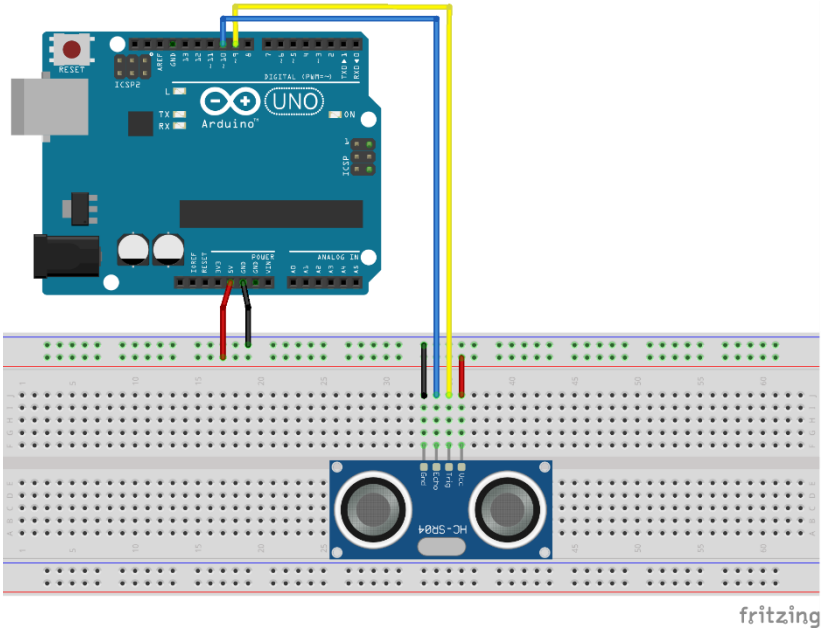
განვიხილოთ, HC-SR04 ულტრაბგერითი სენსორის მუშაობის პრინციპები და მისი გამოყენება არდუინოს დაფასთან მიმართებაში.

ულტრაბგერითი სენსორი ასხივებს 40kHz სიხშირის ულტრაბგერას, თუ ის თავის გზაზე წააწყდება რაიმე ობიექტს ან დაბრკოლებას, უკან ბრუნდება. სენსორიდან ობიექტამდე მანძილის გამოსათვლელად საჭიროა ბგერის სიჩქარისა და იმ დროის გათვალისწინება, რომელიც ულტრაბგერას უკან მოსაბრუნებლად სჭირდება.



სურ.3.38. ულტრაბგერის გავრცელება გარემოში

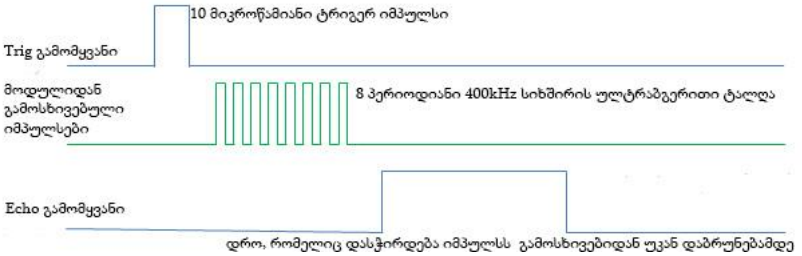
HC-SR04 ულტრაბგერით სენსორს ოთხი საკონტაქტო გამომყვანი აქვს: VCC, GND, Trig და Echo. VCC და GND უნდა მიეერთდეს არდუინოს დაფაზე 5V და GND გამომყვანებს შესაბამისად, trig და echo შეიძლება შეეერთდეს ნებისმიერ საკონტაქტო გამომყვანს არდუინოს დაფაზე.



სურ. 3.39. ულტრაბგერითი სენსორის მიერთება არდუინოსთან

ულტრაბგერის მისაღებად საჭიროა მოდულს Trig გამომყვანზე მივაწოდოთ 10μs იმპულსი. ამის შემდეგ მოდული გამოასხივებს 40kHz სიხშირის 8 პერიოდიან ულტრაბგერით ტალღას. ამ მომენტიდან Echo გამომყვანი გადადის მაღალ ლოგიკურ მდგომარეობაში და რჩება ასე, სანამ გამოსხივებული ტალღა უკან არ დაბრუნდება. აქედან გამომდინარე, Echo გამომყვანით ითვლება ის დრო,

რომელიც სიგნალს გამოსხივებიდან უკან დაბრუნებამდე დასჭირდა.



სურ. 3.40. HC-SR04 ულტრაბგერითი მოდულის დროითი დიაგრამა

მაგალითად, თუ ბგერის სიჩქარეა 340 მ/წმ (0.034 სმ/მკწმ) და ბგერით ტალღას სენსორთან დასაბრუნებლად სჭირდება 294 მკწმ, ობიექტის სენსორამდე დაშორება გამოითვლება S=vt/2 ფორმულით, რადგანაც ბგერითი ტალღა გადის მანძილს ობიექტამდე და შემდეგ უბრუნდება ისევ სენსორს.

სენსორიდან აღმოჩენილ ობიექტამდე მანძილის გამოსათვლელი პროგრამის დასაწერად, უპირველეს ყოვლისა, განვსაზღვროთ ის საკონტაქტო გამომყვანები, რომელსაც ავირჩევთ Trig-ისა და Echo-სთვის. ჩვენს შემთხვევაში, ეს იქნება მე-9 და მე-10 საკონტაქტო გამომყვანები არდუინოს დაფაზე, პროგრამაში შესაბამის ცვლადებს ვარქმევთ trigPin და echoPin-ს. ამის შემდეგ, დაგვჭირდება ასევე ცვლადები ტალღის გავრცელების დროისა და გავლილი მანძილის აღსანიშნავად. setup

ფუნქციაში უნდა განვსაზღვროთ trigPin, როგორც OUTPUT და echoPin, როგორც INPUT, ასევე უნდა დავუკავშირდეთ მიმდევრობით პორტს, იმისათვის რომ მიმდევრობით მონიტორზე შედეგები დავინახოთ.

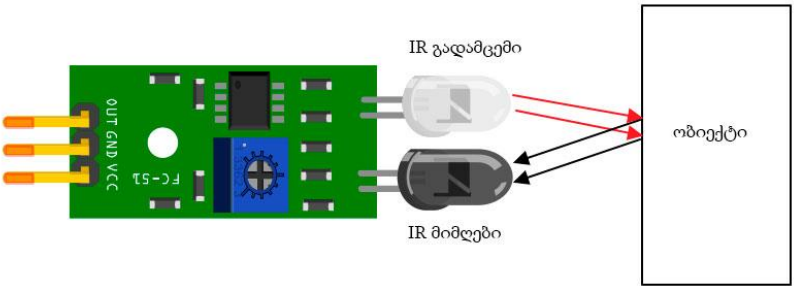
loop ფუნქციაში, TrigPin უნდა იყოს დაბალ ლოგიკურ მდგომარეობაში (LOW). ამის შემდეგ, ულტრაბგერის გენერირებისთვის trigPin გამომყვანი 10 μs-ის განმავლობაში უნდა დავაყენოთ HIGH მდგომარეობაში. pulseIn() ფუნქციის საშუალებით შეგვიძლია წავიკითხოთ ულტრაბგერის გავრცელების დრო და მოვათავსოთ ის duration ცვლადში. ამ ფუნქციას ორი პარამეტრი აქვს, პირველი პარამეტრი არის echoPin, მეორე შეიძლება იყოს HIGH ან LOW. ამ შემთხვევაში, HIGH ნიშნავს, რომ PulseIn() ფუნქცია დაელოდება საკონტაქტო გამომყვანის HIGH მდგომარეობაში გადასვლას, რომელიც გამოწვეულია ბგერითი ტალღით და დაიწყებს დროის ათვლას, შემდეგ ის ელოდება საკონტაქტო გამომყვანის LOW მდგომარეობაში გადასვლას, ეს იმის მაჩვენებელია, რომ არეკლილი ტალღა დაბრუნდა და დროის ათვლაც შეწყდება. საბოლოოდ, pulseIn() ფუნქცია დააბრუნებს იმპულსის ხანგრძლივობას მიკროწამებში. როგორც ზემოთ უკვე ავღნიშნეთ, მანძილის გამოსათვლელად დრო უნდა გავამრავლოთ სიჩქარეზე (0.034) და გავყოთ ორზე. სენსორიდან ობიექტამდე მანძილის მნიშვნელობას დავინახავთ მიმდევრობითი მონიტორის ფანჯარაში.

```
// არდუინოს საკონტაქტო გამომყვანების ნომრები, რომელთანაც ულტრაგერითი სენსორია მიერთებული
const int trigPin = 9;
const int echoPin = 10;

//ვაცხადებთ ცვლადებს
long duration;
int distance;

void setup() {
  pinMode(trigPin,OUTPUT); //trigPin, როგორც გამოსასვლელი
  pinMode(echoPin,INPUT); //echoPin, როგორც შესასვლელი
  Serial.begin(9600);
}

void loop() {
  //trigPin დაბალ ლოგიკურ მდგომარეობაში გადაგვყავს
 digitalWrite(trigPin, LOW);
 //ველოდებით, სანამ trigPin-ზე ლოგიკური ნული არ გვექნება
 delayMicroseconds(2);
  //trigPin გადაგვყავს მაღალ ლოგიკურ მდგომარეობაში 10 მკწმ-ის განმავლობაში
 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);
 digitalWrite(trigPin, LOW);
 //ვკითხულობთ echoPin-ს, რომელიც ტალღის მოძრაობის დროს აბრუნებს მიკროწამებში
 duration = pulseIn(echoPin, HIGH);
 // გამოვთვალოთ მანძილი
 distance= duration*0.034/2;
//მიღებული შედეგი დავბეჭდოთ მიმდევრობით მონიტორში
 Serial.print("Distance: ");
 Serial.println(distance);
}
```

მიახლოების ინფრაწითელი (IR) სენსორი და არდუინო

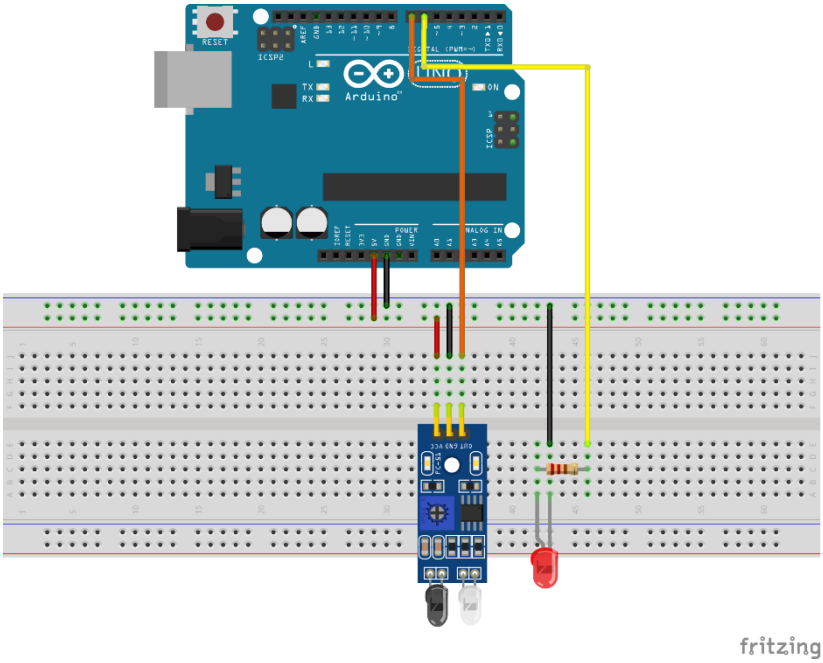
მიახლოების ინფრაწითელი სენსორის საშუალებით შესაძლებელია დაბრკოლების აღმოჩენა. მისი გამოყენება მრავალფუნქციურად შეიძლება, ის განსაკუთრებით პოპულარულია სხვადასხვა სახის, რობოტებთან დაკავშირებული, პროექტების განსახორციელებლად, როგორიცაა, მაგალითად: რობოტი, რომელიც გვერდს უვლის დაბრკოლებას ან რობოტი, რომელიც რაიმე წინასწარ განსაზღვრულ ხაზს მიუყვება. ეს არის აქტიური ინფრაწითელი სენსორი. IR სენსორი ასხივებს და შემდეგ ახდენს თავისივე ინფრაწითელი სხივების აღმოჩენას.

განვიხილოთ ამ სენსორის მოქმედების პრინციპი. მასზე არსებული ინფრაწითელი შუქდიოდი მუდმივად ასხივებს ვიწროდ მიმართულ ინფრაწითელ სხივს, რომელიც დაბრკოლებაზე მოხვედრისას აირეკლება და მისი რაღაც ნაწილი ბრუნდება უკან. თუ ეს სხივები მოხვდა სენსორზე არსებულ ინფრაწითელ დეტექტორზე, მაშინ მოწყობილობის გამოსასვლელზე გვექნება დადებითი იმპულსი.



სურ. 3.41. ინფრაწითელი სენსორის მოქმედების პრონციპი

გასათვალისწინებელია ის, რომ შავი ზედაპირი თითქმის მთლიანად შთანთქავს შუქს და თუ შავ ზედაპირიან ობიექტს მივუშვერთ სენსორს, მაშინ მიმღებზე გამოსხივებული ინფრაწითელი ტალღების მხოლოდ მცირე ნაწილი მოხვდება და შეიძლება სენსორის დაფას არეკლილ სხივზე რეაქცია არც ჰქონდეს.



სურ. 3.42. ინფრაწითელი სენსორის მიერთება არდუინოსთან

კოდი:

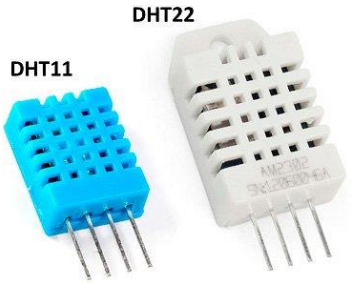
```
// არდუინოს გამომყვანის ნომერი, რომელიც შუქდიოდს უერთდება
Const int LED = 6;
// არდუინოს გამომყვანის ნომერი, რომელიც  ინფრაწითელ სენსორს უერთდება
Const int irPin = 7;
// ცვლადი, რომელიც ინახავს სენსორის მდომარეობას
int sensorOut = HIGH;


void setup() {
  // შუქდიოდის შესაბამისი საკონტაქტო გამომყვანი,  როგორც გამოსასვლელი
   pinMode(LED,OUTPUT);
  // საკონტაქტო გამომყვანი irPin, როგორც შესასვლელი
   pinMode(irPin, INPUT);
 //მიმდევრობითი პორტის ჩართვა
 Serial.begin(9600);
 }

void loop() {
   // ვკითხულობთ სენსორის მნიშვნელობას
   sensorOut = digitalRead(irPin);
   if (sensorOut == HIGH)
   {
   //თუ დაბალი ლოგიკური დონეა, მაშინ სენსორის წინ დაბრკოლებაა
   Serial.println("Obstacle!!!");
   //ვანთებთ შუქდიოდს
    digitalWrite(LED, HIGH);
     }
     else
     {
    //თუ მაღალი ლოგიკური დონეა, მაშინ სენსორის წინ  დაბრკოლება არ არის
   Serial.println("No Obstacle");
```

```
    //ვაქრობთ შუქდიოდს
    digitalWrite(LED, LOW);
     }
   //დაყოვნება ინფრაწითელი სენსორის წაკითხვებს შორის
    delay(200);
  }
```

თუ სენსორის წინ დაბრკოლება არ არის, მაშინ მიმდევრობით მონიტორში იბეჭდება `No Obstacle` და შუქდიოდი ირთვება. თუ სენსორის წინ რაიმე ობიექტს მოვათავსებთ, მაშინ პროგრამა დაბეჭდავს ტექსტს „`Obstacle!!!`“ და შუქდიოდი გამოირთვება.

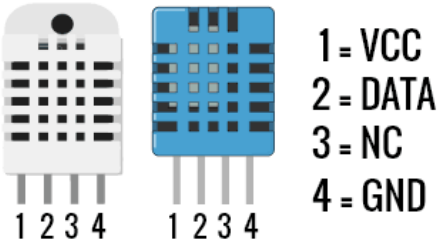
ტემპერატურისა და ტენიანობის DHT11 და DHT22 სენსორები და არდუინო

DHT11 და DHT22 ტემპერატურისა და ტენიანობის სენსორებს შორის DHT22 უფრო ძვირადღირებულ ვერსიას წარმოადგენს, გაცილებით უკეთესი მახასიათებლებით. DHT22-სთვის ტემპერატურის გაზომვის დიაპაზონი -40-დან 125°C-მდეა ±0.5 გრადუსის სიზუსტით, მაშინ როდესაც DHT11-სთვის ეს დიაპაზონი არის 0-დან 50°C-მდე ±2 გრადუსის სიზუსტით. გარდა ამისა, DHT22 სენსორს ტენიანობის გაზომვის უკეთესი დიაპაზონი აქვს (0÷100%), სიზუსტით 2-5%. შედარებისთვის, DHT11 სენსორს შეუძლია გაზომოს ტენიანობა 20÷80%-ის საზღვრებში, 5% სიზუსტით.



სურ. 3.43. ტემპერატურისა და ტენიანობის სენსორები

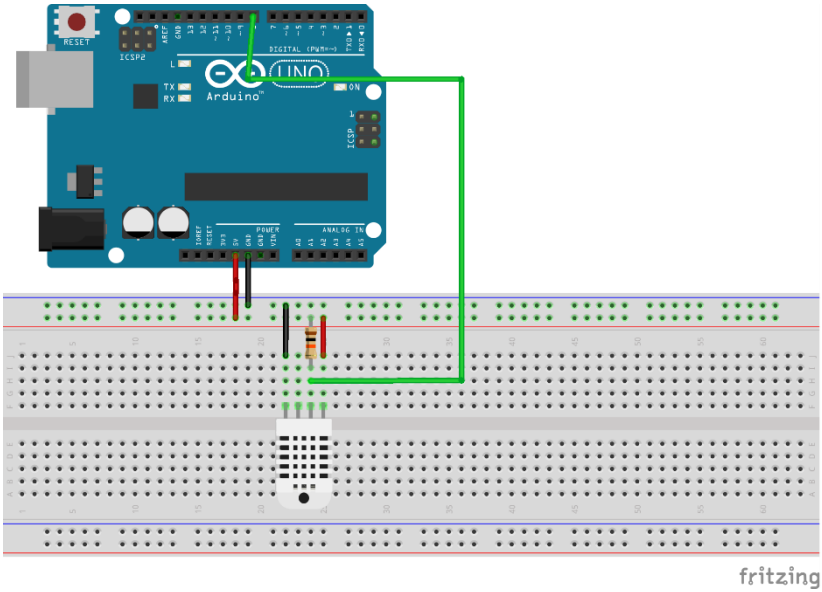
თუმცა, არის ზოგი მახასიათებელი, რომელიც DHT11-ს DHT22-ზე უკეთესი აქვს. მაგალითად, დისკრეტიზაციის სიჩქარე DHT11-სთვის 1Hz-ია, ანუ ეს არის ერთი წაკითხვა ყოველ წამში, მაშინ როდესაც იგივე მახასიათებლის მნიშვნელობა DHT22-თვის არის 0,5Hz ანუ ერთი წაკითხვა ყოველ 2 წამში. გარდა ამისა, DHT11 ზომით უფრო პატარაა DHT22-ზე. ორივე სენსორის მუშა ძაბვა არის 3-დან 5 ვოლტამდე, მაშინ როდესაც გაზომვისას დენის მაქსიმალური მნიშვნელობაა 2,5 mA.



სურ. 3.44. DHT11 და DHT22 სენსორების საკონტაქტო გამომყვანები (NC არ გამოიყენება)

განვიხილოთ, როგორ მუშაობენ DHT11 და DHT22 სენსორები. მათ შემადგენლობაში შედის ტენიანობის განსაზღვრის სენსორული კომპონენტი, NTC ტემპერატურის სენსორი (თერმისტორი) და ინტეგრირებული სქემა სენსორის უკანა მხარეს. ტენიანობის გასაზომად ისინი იყენებენ ტენიანობისადმი მგრძნობიარე კომპონენტს, რომელსაც აქვს ორი ელექტროდი, მათ შორის მოთავსებული ტენის შემაკავებელი ნივთიერებით. ტენიანობის შეცვლისას, იცვლება ელექტროდებს შორის მოთავსებული ნივთიერების გამტარობა, რაც გულისხმობს ელექტროდებს შორის წინაღობის ცვლილებას. წინაღობის ეს ცვლილება იზომება და მუშავდება ინტეგრირებული სქემის მიერ, რის შემდეგაც მიკროკონტროლერს უკვე შეუძლია მისი მნიშვნელობის წაკითხვა.

DHTxx სენსორს აქვს ოთხი საკონტაქტო გამომყვანი: VCC, GND, მონაცემთა გამომყვანი (data pin) და NC საკონტაქტო გამომყვანი, რომელსაც არაფერთან აერთებენ. სენსორსა და არდუინოს დაფას შორის კომუნიკაციის უზრუნველსაყოფად მომჭიმავი 10 კილოომი რეზისტორი გამოიყენება. არსებობს ამ სენსორების ისეთი ვერსიებიც, რომლებშიც მომჭიმავი რეზისტორები შიგნით არის ჩაშენებული.



სურ. 3.45. DHTXX სენსორის მიერთება არდუინოსთან.

DHTXX სენსორები მონაცემების გადასაცემად იყენებენ თავის 1-wire პროტოკოლს. ეს პროტოკოლი ზუსტ სინქრონიზაციას მოითხოვს, სენსორებიდან მონაცემების მიღების დროითი დიაგრამები შეგიძლიათ იხილოთ მათი სპეციფიკაციების ცხრილში. თუმცა, დროით დიაგრამებზე განსაკუთრებული ფიქრი არცაა საჭირო, რადგანაც DHT ბიბლიოთეკა, რომელსაც ჩვენ გამოვიყენებთ, თვითონ აგვარებს ამ საკითხს.

DHT ბიბლიოთეკის ნახვა არდუინოს ოფიციალურ საიტზეა შესაძლებელი, ეს ბიბლიოთეკა პროგრამის კოდში უნდა ჩავრთოთ. ასევე უნდა განვსაზღვროთ იმ საკონტაქტო გამომყვანის ნომერი, რომელსაც მივუერთებთ სენსორს და

შევქმნათ DHT ობიექტი. როგორც ქვემოთ მოცემული პროგრამის კოდიდან ჩანს, read22() ფუნქციის საშუალებით სენსორიდან მონაცემების წაკითხვა და მათი მნიშვნელობების (ტემპერატურა და ტენიანობა) t და h ცვლადებში მოთავსება ხდება. თუ ვიყენებთ DHT11 სენსორს, ასეთ შემთხვევაში, read11() ფუნქცია უნდა გამოვიყენოთ. პროგრამის გაშვების შემდეგ, ტემპერატურისა და ტენიანობის მნიშვნელობებს მიმდევრობითი მონიტორის ფანჯარაში დავინახავთ.

```
#include <dht.h>
//განვსაზღვრავთ არდუინოს იმ საკონტაქტო გამომყვანის ნომერს, რომელსაც სენსორი უერთდება
#define dataPin 8
//ვქმნით DHT ობიექტს
dht DHT;

void setup() {
  Serial.begin(9600);
}
void loop() {
   //ვკითხულობთ მონაცემებს სენსორიდან
  int readData=DHT.read22(dataPin);
  //ვიღებთ ტემპერატურის მნიშვნელობას
  float t=DHT.temperature;
  //ვიღებთ ტენიანობის მნიშვნელობას
  float h = DHT.humidity;
  //ვბეჭდავთ შედეგებს მიმდევრობით მონიტორში
  Serial.print("Temperature = ");
  Serial.print(t);
  Serial.print(" *C ");
  Serial.print("    Humidity = ");
```

```
  Serial.print(h);
  Serial.println(" % ");
  //2 წამიანი დაყოვნება
  delay(2000);
}
```

არდუინო და SD-ბარათი

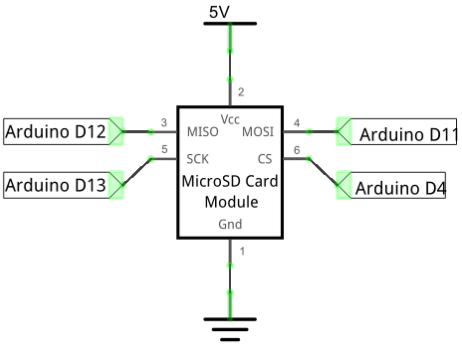
განვიხილოთ, როგორ შეიძლება გამოვიყენოთ SD ბარათის მოდული არდუინოს დაფასთან. SD ბარათის მოდული სტანდარტულ MicroSD ბარათებთან მუშაობს, რომელთა მუშა ძაბვაა 3.3V. მოდულს აქვს ძაბვის რეგულატორი და ძაბვის დონის წამნაცვლებელი, ასე რომ მისი გამოყენება შესაძლებელია არდუინოს დაფის 5V საკონტაქტო გამომყვანებთან.

სურ.3.46. SD მოდულები

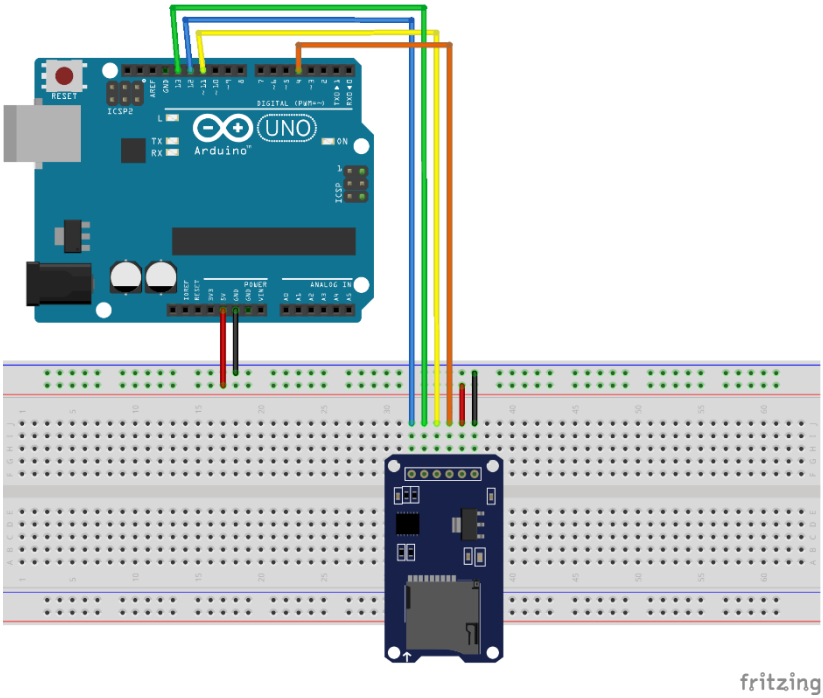
SD ბარათის მოდულს ექვსი საკონტაქტო გამომყვანი აქვს: VCC, GND და კიდევ ოთხი გამომყვანი SPI კომუნიკაციისთვის.

SD ბარათის მოდული	GND	VCC	CS	MOSI	MISO	SCK
Arduino Uno	GND	+5V	4	11	12	13
Arduino Mega	GND	+5V	53	51	50	52

ცხრილი 3.4. SD ბარათის მოდულის არდუინოსთან შეერთება Arduino Uno-სა და Arduino Mega-სთვის



სურ. 3.47. SD ბარათის არდუინოს მოდულთან შეერთების პრინციპიალური სქემა



სურ. 3.48. SD ბარათის მოდულის შეერთება არდუინოს დაფასთან

ქვემოთ მოცემულია პროგრამა, რომლის საშუალებითაც ვწერთ ინფორმაციას SD ბარათზე და შემდეგ ვკითხულობთ მას.

```
// საჭირო ბიბილიოთეკები
#include <SD.h>
#include <SPI.h>
// ფაილის ობიექტი
File myFile;
//არდუინოს საკონტაქტო გამომყვანის ნომერი, რომელიც CS -თან არის მიერთებული
const int pinCS = 4;
```

```
void setup() {
  Serial.begin(9600);
  pinMode(pinCS, OUTPUT);

  // SD ბარათის ინიციალიზაცია
  if (SD.begin())
  {
  Serial.println("SD card is ready to use.");
  } else
  {
  Serial.println("SD card initialization
  failed");
  return;
  }
  // ფაილის შექმნა ან გახსნა
  myFile = SD.open("test.txt", FILE_WRITE);
  // თუ ფაილი გაიხსნა, მაშინ ჩავწეროთ:
  if (myFile) {
    Serial.println("Writing to file...");
    // ჩავწეროთ ტექსტი
    myFile.println("Testing text 1, 2 ,3...");
    // ფაილის დახურვა
    myFile.close();
    Serial.println("Done.");
  }
  //თუ ფაილი არ გაიხსნა, მაშინ შეცდომაა
  else {
    Serial.println("error opening test.txt");
  }
  // ფაილის წაკითხვა
  myFile = SD.open("test.txt");
```

```
  if (myFile) {
   Serial.println("Read:");
    // წავიკითხოთ მთლიანი ფაილი
    while (myFile.available()) {
    Serial.write(myFile.read());
    }
    myFile.close();
  }
  else {
    Serial.println("error opening test.txt");
  }
}
void loop() {
  //ციკლი ცარიელია
}
```

როგორც ზემოთ მოყვანილი პროგრამიდან ჩანს, პირველ რიგში საჭიროა SD მეხსიერების ბარათისა და SPI ბიბლიოთეკის ჩართვა, myFile ობიექტის შექმნა და pinCs საკონტაქტო გამომყვანის განსაზღვრა SPI სალტეზე, ეს არის 53-ე საკონტაქტო გამომყვანი Arduino Mega-სთვის და მე-4 საკონტაქტო გამომყვანი Arduino Uno-სთვის. ამ მაგალითში ჩვენ გვჭირდება, რომ პროგრამა შესრულდეს მხოლოდ ერთხელ, ამიტომ მთელი კოდი მოქცეულია setup ფუნქციის შიგნით, loop-ში არაფერი წერია. როგორც კოდიდან ჩანს, pinCs საკონტაქტო გამომყვანი განსაზღვრულია, როგორც output. pinCs საკონტაქტო გამომყვანი უნდა იყოს LOW

მდგომარეობაში, იმისათვის რომ SD მოდულსა და არდუინოს შორის კომუნიკაცია არსებობდეს.

SD.begin() ფუნქცია SD ბარათის ინიციალიზაციის საშუალებას იძლევა. თუ ინიციალიზაცია წარმატებით განხორციელდება, ეს ნიშნავს, რომ if პირობა სრულდება და მიმდევრობითი მონიტორის ფანჯარაში გამოჩნდება "SD card is ready to use" სტრიქონი; წინააღმდეგ შემთხვევაში, ეკრანზე დაიბეჭდება "SD card initialization failed" და პროგრამის შესრულება შეწყდება.

SD.open() ფუნქცია ქმნის ახალ ფაილს "test.txt" სახელწოდებით. ფუნქცია ასევე შეიცავს FILE_WRITE არგუმენტს, რომლის გამოყენებითაც ჩვენ შეგვიძლია ფაილის როგორც ჩაწერა, ასევე წაკითხვა. თუ ფაილი უკვე არსებობს, SD.open() ფუნქცია მას უბრალოდ გახსნის.

ფაილის შექმნის შემდეგ, მიმდევრობითი მონიტორის ფანჯარაში ვბეჭდავთ ტექსტს "Writing to file", ფაილში ტექსტის ("Testing text 1, 2 ,3…") ჩაწერა ხდება myFile.println() ფუნქციის საშუალებით. ამის შემდეგ უკვე საჭიროა close() ფუნქციის გამოყენება, იმისათვის რომ ფაილში ჩაწერილი მონაცემები SD ბარათში იყოს შენახული.

კოდის არდუინოში ჩატვირთვის შემდეგ, მიმდევრობითი მონიტორის ფანჯარაში მივიღებთ:

```
SD card is ready to use.
Writing to file...
Done.
Read:
Testing text 1, 2 ,3...
Testing text 1, 2 ,3...
```

სურ. 3.49. პროგრამის შესრულების შედეგი

ამგვარად, ჩვენ წარმატებით მოვახდინეთ SD ბარათის ინიციალიზაცია, ჩავწერეთ მასში ტექსტი “Testing text 1, 2 ,3…” და შემდეგ წავიკითხეთ ის. თუ SD ბარათს გავხსნით კომპიუტერზე, დავინახავთ ჩვენს მიერ შექმნილ „test.txt” ფაილს და მასში ჩაწერილ ტექსტს.

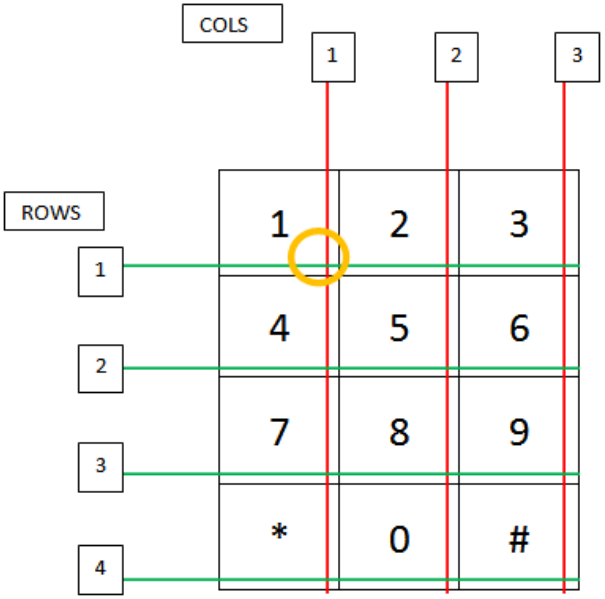
მატრიცული კლავიატურის გამოყენება

მატრიცული კლავიატურა მომხმარებელს პროგრამის მუშაობის პროცესში მონაცემების შეტანის საშუალებას აძლევს. განვიხილოთ 12 ღილაკიანი მატრიცული კლავიატურის არდუინოსთან შეერთებისა და Keypad.h ბიბლიოთეკის გამოყენების საკითხები.

მატრიცული კლავიატურა ხშირად გამოიყენება არდუინოს სისტემაში მონაცემების შესატანად, მემბრანული ტიპის კლავიშების პანელი საკმაოდ თხელია და გამოსაყენებლად მარტივი. 12-ღილაკიან პანელს სამი სვეტი და

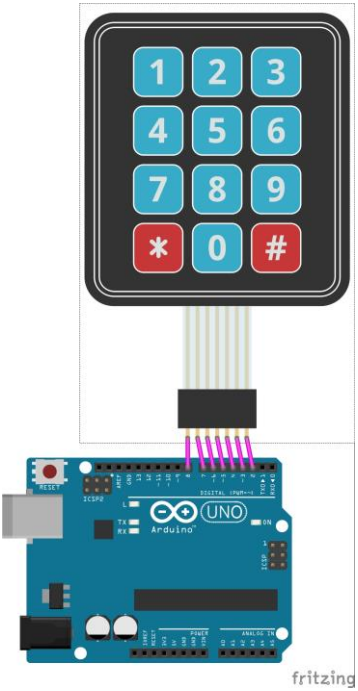
ოთხი სტრიქონი აქვს. ღილაკის დაჭერა ერთ-ერთი სტრიქონის გამოსასვლელს აკავშირებს ერთ-ერთი სვეტის გამოსასვლელთან. ამ ინფორმაციის მიხედვით, შეიძლება განვსაზღვროთ, რომელი ღილაკი იყოს დაჭერილი. მაგალითად, როდესაც ვაჭერთ 1 ღილაკს, ერთმანეთს უკავშირდება პირველი სვეტი და პირველი სტრიქონი. არდუინო განსაზღვრავს ამას და პროგრამაში შეჰყავს ერთიანი.

ქვემოთ მოცემულ სურათზე ნაჩვენებია, კლავიატურის შიგნით როგორ არის განლაგებული სვეტები და სტრიქონები.



სურ.3.50. მატრიცული კლავიატურა

განვიხილოთ keypad.h ბიბლიოთეკის მუშაობის პრინციპი. როდესაც მომხმარებელი აჭერს კლავიატურის ღილაკზე, პროგრამა მიმდევრობითი მონიტორის ფანჯარაში შესაბამის მნიშვნელობას გვიჩვენებს.



სურ. 3.51. მატრიცული კლავიატურის არდუინოსთან შეერთება

```
#include <Keypad.h>
const byte Rows= 4; //სტრიქონების რაოდენობა
const byte Cols= 3; //სვეტების რაოდენობა
// გამოვაცხადოთ keymap სიმბოლური მასივი
char keymap[Rows][Cols]=
{
```

```
{'1', '2', '3'},
{'4', '5', '6'},
{'7', '8', '9'},
{'*', '0', '#'}
};
```

//კლავიატურის მიერთება არდუინოსთან

```
byte rPins[Rows]={8,7,6,5}; //სტრიქონები 0-3
byte cPins[Cols]={4,3,2}; //სვეტები 0-2
// ბრძანება keypad ბიბლიოთეკისთვის, Keypad კლასის ინიციალიზაცია
Keypad kpd= Keypad(makeKeymap(keymap),rPins,
cPins,Rows,Cols);

void setup()
{
     Serial.begin(9600);
}
```

// თუ ღილაკი დაჭერილია, ამ ღილაკის მნიშვნელობა ინახება 'keypressed'ცვლადში; თუ ღილაკის მნიშვნელობა NO_KEY-ს ტოლი არ არის, მაშინ ის იბეჭდება მიმდევრობით მონიტორში

```
void loop()
{
     char keypressed = kpd.getKey();
     if (keypressed != NO_KEY)
     {
          Serial.println(keypressed);
     }
}
```

გამოყენებული ლიტერატურა

1. ი. მოსაშვილი, ს. ონიანი. Arduino. პროგრამირების საფუძვლები. „ტექნიკური უნივერსიტეტი“. თბილისი. 2016.
2. ჯ. გრიგალაშვილი. Arduino-ს ვიზუალური დაპროგრამება FLProg გარემოში. საქართველოს ტექნიკური უნივერსიტეტი. თბილისი. 2015
3. ჯ. გრიგალაშვილი. მეთოდური მითითებები Arduino-ს შესწავლისათვის. საქართველოს ტექნიკური უნივერსიტეტი. თბილისი. 2016;
4. P. Horovitz. W. Hill. The art of electronics. 2nd edition. Cambridge University Press. 1994;
5. J. Boxall. Arduino Workshop. A Hands-on Introduction with 65 Projects. San Franciso. 2013.
6. J. Bloom. Exploring Arduino. Tools and Techniques for Engineering Wizardry. John Willey & Sons, Inc. 2013
7. https://www.arduino.cc/
8. www.toptechboy.com
9. https://www.youtube.com/user/mcwhorpj/videos
10. http://playground.arduino.cc
11. https://giomjava.com/
12. https://georobot.wordpress.com/
13. https://www.norwegiancreations.com
14. https://learn.adafruit.com
15. https://howtomechatronics.com/
16. http://fritzing.org/home/

სარჩევი