Daudzpogu klaviatūra

Ivars Driķis

2019. gada 10. oktobrī

1 Ievads

Lai tiktu galā ar lielu skaitu spīddiožu, lampiņu un spiedpogu, lieto dinamisko dimanisko vadīšanas metodi. Par to, kā tas strādā, stāstīju materialā par 8x8 spīddiožu matricu [1]. Spīddiožu matricu veido horizontālu un vertikālu vadu grupas, kuru krustpunktā ir ievietotas spīddiodes. Līdzīgi ir jārīkojas arī lampiņu un spiedpogu matricas gadījumā. Pie kam, svarīgi lai kopā ar lampiņu un spiedpogu virknē būtu ieslēgta diode, kā tas redzams 1 zīmējumā. Lampiņu matricas gadījumā tas ir kritiski, jo pretējā gadījumā, mēģinot tās vadīt, spīdēs ne tikai tās lampiņas, kuras gribējām ieslēgt, bet gan visas! Tiesa, dažādā spilgtumā. Spiedpogu matricā diode nav tik kritiska, jo parasti ir nospiesta tikai viena no podziņām. Šādas klaviatūras tad arī ķīnieši ražo un tirgo, kā paratsi viņi taupa visur kur vien var. Ja nospiedīs vienlaikus 2 vai vairāk pogas, tad ne tikai nevarēs pareizi noteikt kuras no tām ir nospiestas, bet arī veiksmes gadījumā var nosvilināt kādu no izejām. Kādai ir jābūt korekti izveidotai klaviatūrai un kā ar to strādāt, par to tad arī šajā dokumentā.

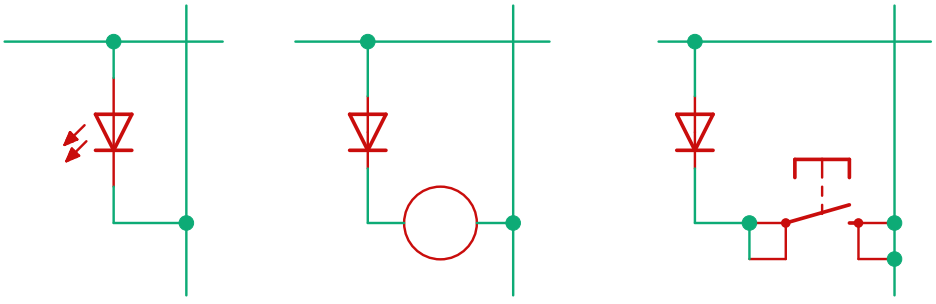
2 Soli pa solim

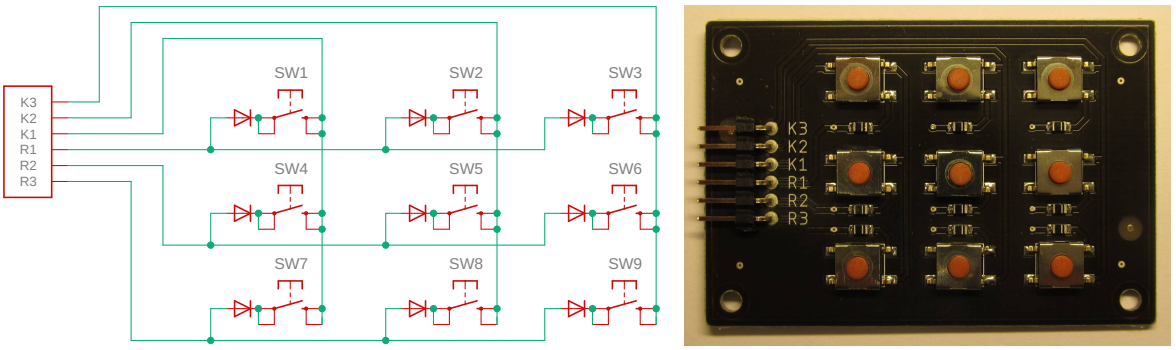
Apskatam pareizu dinamisko klaviatūru, kuras shēma dota 2. zīmējumā un kura pieslēgta Arduino savietojamam mikrodatoram atbilstoši 3. zīmējumam. Iesim soli pa solim sakot ar vienas pogas apprasīšanu, tālāk skatīsimies kā vajag un kā nevajag darīt, lai strādātu ar spiedpogu matricu.

2.1 Lasām vienu pogu

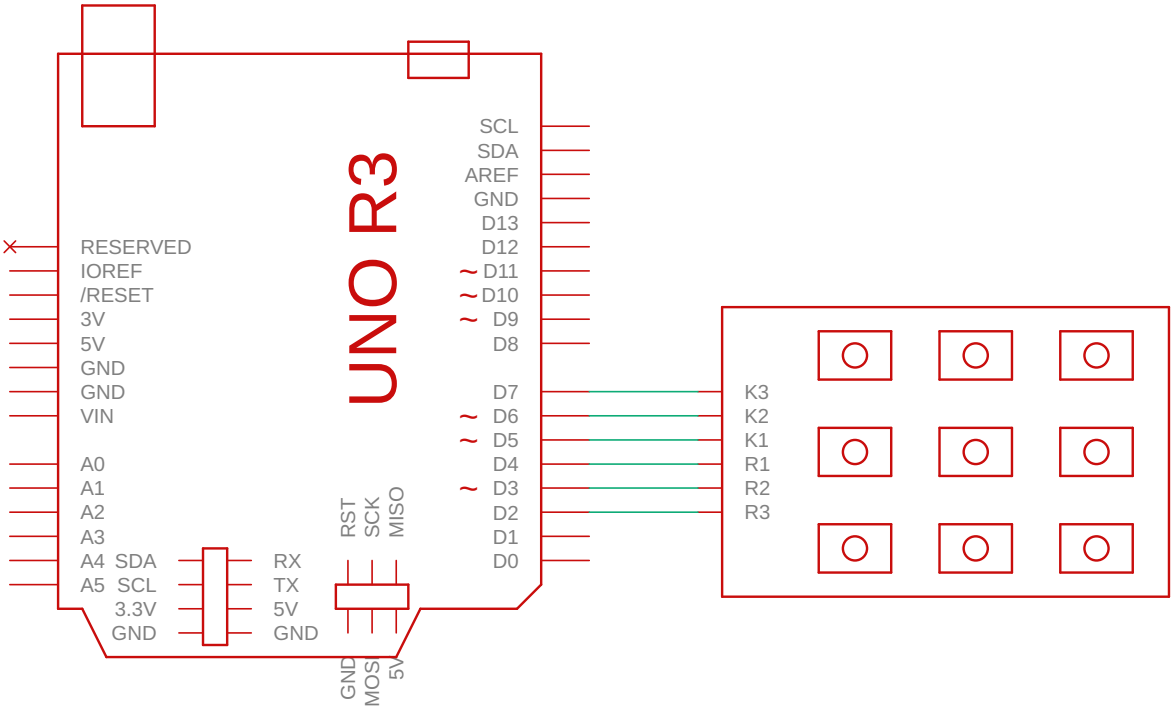
Sākumā strādāsim ar tikai vienu pogu par pamatu izmantosim Panel01 programmu no [2]. Tieši tāpat uz ieejas, kurai caur diodi ir pieslēgta spiedpoga, izmantojot iebūvēto PULL_UP

Att. 1: Spīddiožu, lampiņu un spiedpogu matricas elements





Att. 2: Dinamiskās klaviatūras modulis



Att. 3: Dinamiskā klaviatūras mooduļa pieslēgums

rezistoru, ir uzstādīts HIGH signāla līmenis. Savukārt spiedpogas otrā puse ir pieslēgta nevis GND, kā tas bija Panel01 programmā no [2], bet gan mikrodatora izejai. Tas nozīmē, ka ieejā signāla līmenis ir LOW tikai tad, ja spiedpoga ir nospiesta un atbilstošajā mikrodatora izejā ir LOW signāla līmenis. Tas ļauj, manipulējot ar izejas signālu līmeniem, vienai ieejai nolasīt vairāku spiedpogu stāvokli.

Programmu sākam definējot pie krām no mikrodatora līnijām ko pieslēdzam

Listing 1: oneKey.ino

```
1 const int pinLed = 13;
2 const int pinR = 4;
3 const int pinK = 7;
```

Sagatavojam darbam izeju, pie kuras pieslēgta spīddiode.

```
5 void setup( void )
6 {
```

```
7    pinMode(pinLed, OUTPUT);
```

Klaviatūras rindu R pieslēdzam mikrodatora ieejai un kolonnu K pieslēdzam mikrodatora izejai. Tādējādi mēs aptaujāsim spiedpogu SW3. Ieejas signāla līmeni paceļam augšā ar pull-up rezistoru, bet izejā iestādām zemu signāla līmeni.

```
9    pinMode(pinR, INPUT_PULLUP);
10   pinMode(pinK, OUTPUT);
11   digitalWrite(pinK, LOW);
12 }
```

Programmas galvenajā daļā nolasām signālu no ieejas un padodam to uz spīddiodi. Pēc tam nedaudz uzgaidam.

```
14 void loop( void )
15 {
16   digitalWrite(pinLed, digitalRead(pinR) );
17   delay( 100 );
18 }
```

Nu ko, spīddiode atkārto spiedpogas SW3 stāvokli, pārējās pogas nejūt.

2.2 Lasām divas pogas

Papildinām iepriekš uzrakstīto programmu 1. tā, lai varētu aptaujāt divas pogas.

Listing 2: twoKeys.ino

```
1 const int pinR1 = 4;
2 const int pinR2 = 3;
3 const int pinK3 = 7;
4
5 byte swiState3 = HIGH;
6 byte swiState9 = HIGH;
7
8 void setup( void )
9 {
10   Serial.begin( 9600 );
11
12   pinMode(pinR1, INPUT_PULLUP);
13   pinMode(pinR2, INPUT_PULLUP);
14   pinMode(pinK3, OUTPUT);
15
16   digitalWrite(pinK3, LOW);
17
18   swiState3 = digitalRead( pinR1 );
19   swiState9 = digitalRead( pinR2 );
20 }
21
22 void loop( void )
23 {
24   byte curSwiState3 = digitalRead( pinR1 );
25   byte curSwiState9 = digitalRead( pinR2 );
26
27   if( (swiState3==HIGH) && (curSwiState3==LOW) )
28     Serial.println( "Swi3" );
29
30   if( (swiState9==HIGH) && (curSwiState9==LOW) )
31     Serial.println( "Swi9" );
```

```
32
33    swiState3 = curSwiState3;
34    swiState9 = curSwiState9;
35    delay( 100 );
36 }
```

3 Lasām vairākas pogas vienlaikus

Apskatīsim vienkāršu programmu, kas ļauj noteikt visas nospiestās spiedpogas. Kā ierasts, sākam ar to kas kur pieslegts

Listing 3: kbdMulti.ino

```
1 const int pinR1 = 4;
2 const int pinR2 = 3;
3 const int pinR3 = 2;
4 const int pinK1 = 5;
5 const int pinK2 = 6;
6 const int pinK3 = 7;
```

Lai programmā varētu lietot ciklus, izmantotās ieejas un izejas ir jāsakārto masīvos

```
8 const int pinK[3] = {pinK1,pinK2,pinK3};
9 const int pinR[3] = {pinR1,pinR2,pinR3};
```

Inicializējam seriālo savienojumu ar datoru

```
11 void setup( void )
12 {
13    Serial.begin( 9600 );
```

un Arduino ieejas un izejas. Pie kam uz izejam uzliekam HIGH.

```
15    for(int i=0; i<3; i++) {
16      pinMode(pinR[i], INPUT_PULLUP);
17      pinMode(pinK[i], OUTPUT);
18      digitalWrite(pinK[i], HIGH);
19    }
20 }
```

Tālāk programmas cikla daļa. Inicializējam statisko mainīgo, kurā glabāsim mainīgo, kurā iekodētas iepriekš nospiestās pogas kā arī virkni lokālo mainīgo.

```
22 void loop( void )
23 {
24 static int prew = 0;
25 int i, j, atb = 0;
```

Apstrādājam katru no 3 kolonnām. Vispirms izvēlētajai kolonnai atbilstošo izeju pārslēdzam LOW stāvoklī. Pēc tam apjautājam katru no rindām atbilstošajām ieejām. Ja kādā no ieejām nolasām LOW, tad spiedpoga, kura atrodas uz izvēlētās rindas un izvēlētās kolonnas līniju krustpunkta, ir nospiesta. To atzīmējam uzstādot atbilstošo bitu mainīgajā atb. Piemēram, ja LOW ir ieejā pin2R kad LOW uzdots izejā pin1K, tad uzstādam 3*0+1=1 mainīgā atb bitu. Savukārt, ja LOW ir ieejā pin2R kad LOW uzdots izejā pin3K, tad uzstādam 3*2+2=8 mainīgā atb bitu. Tā kā apakšprogrammas darbības sākumā visi atb biti bija 0, tad šadi iezīmējas nospiestās spiedpogas. Kad visas rindām atbisltošas ieejas ir pārbaudītas, kolonnai atbilstošajā izejā atjauno HIGH līmeni.

```
27    for(i=0; i<3; i++) {
28      digitalWrite(pinK[i], LOW);
29
30      for(j=0; j<3; j++) {
31        byte cur = digitalRead(pinR[j]);
32        if( cur == LOW ) bitSet(atb, 3*i+j);
33      }
34
35      digitalWrite(pinK[i], HIGH);
36    }
```

Kad klaviatūras nolasīta un rezultāts atrodams mainīgajā atb, to nosūtīt uz datora ekrānu. Bet tikai vienu reizi. Ja neviens no taustiņiem nav nospiests (atb==0), tad neko nedrukā.

```
38    if( (atb!=0) && (atb!=prew) ) {
```

Tā kā teksts uz datora ekrāna tiek drukāts pa rindām, tad arī mainīgais atb ir jāatkodē atbilstoši klaviatūras spiedpogu rindām. Ja poga nospiesta, tad drukā simbolu '*', ja nav nospiesta tad simbolu '+'. Kad visa klaviatūras rinda nodrukāta, tad pāriet uz jaunu rindu.

```
39      for(i=0; j<3; j++) {
40        for(j=0; i<3; i++) {
41          if( bitRead(atb,3*i+j)==1 ) Serial.print('*');
42          else Serial.print('+');
43        }
44
45        Serial.println();
46      }
```

Beigās nodrukā tukšu rindu, lai klaviatūru attēli atdalītos viens no otra un liek datoram nedaudz pagaidīt.

```
48      Serial.println();
49      delay( 100 );
50    }
```

Pašās beigās noglabā klaviatūras stāvokli. To dara vienmēr.

```
52    prew = atb;
53  }
```

Pamēģinām un priecajamies.

4 Izmantojam standartabiblioteku

Apskatīsim kā noprogrammēt dinamisko klaviatūru, kas pieslegta kā 3 ar standarta bibliotēku Keypad [3]. Kā pievieno bibliotēkas, lasiet vienā no iepriekšējiem dokumentiem [4]. Sākam ar bibliotējas headera faila norādīšanu

Listing 4: kbdMinimal.ino

```
1 #include <Keypad.h>
```

Tālāk, kā jau ierasts, definējam kur ko pieslēgt.

```
3 const int pinR1 = 4;
4 const int pinR2 = 3;
5 const int pinR3 = 2;
```

```
6 const int pinK1 = 5;
7 const int pinK2 = 6;
8 const int pinK3 = 7;
```

Lai varētu definēt klaviatūras objektu, ir jāizveido virkne mainīgo. Vispirms norādām klaviatūras rindu un kolonnu skaitu. Pēc tam sakārtojam masīvos rindu un kolonnu pieslēguma līnijas. Tālāk nepieciešami pogu izkārtojuma masīvs. Un visbeidzot varam definēt klaviatūras objektu.

```
10 const int kbdRows = 3;
11 const int kbdCols = 3;
12
13 byte rowPins[kbdRows] = {pinR1,pinR2,pinR3};
14 byte colPins[kbdCols] = {pinK1,pinK2,pinK3};
15
16 char keys[kbdRows][kbdCols] = {
17     {'1','2','3'},
18     {'4','5','6'},
19     {'7','8','9'},
20 };
21
22 Keypad kpd = Keypad(makeKeymap(keys), rowPins, colPins, kbdRows, kbdCols);
```

Perifērijas iekārtu inicializācijas sadaļā atveram seriālo savienojumu.

```
24 void setup( void )
25 {
26   Serial.begin( 9600 );
27 }
```

Programmas cikla daļā definējam statisko mainīgo, kurā tiks saglabāts iepriekšējam nospiestajam taustiņam atbilstošais simbols. Sākumā ieliekam tādu simbolu, kurš nav klaviatūrā.

```
29 void loop( void )
30 {
31 static char prewKey = ' ';
```

Tagad nolasām simbolu no klaviatūras. Ja neviens no taustiņiem nav nospiests, tad šī funkcija getKey atgriež 0.

```
33   char key = kpd.getKey();
```

Taustiņa kodu uz datoru sūtam tikai tad, ja kāds no taustiņiem nospiests un tikai tad, ja iepriekš bija nospiests cits simbols.

```
35   if( (key!=0) && (prewKey!=key) )
36     Serial.println( key );
```

Visbeidzot saglabājam nospiestajam taustiņam atbilstošo simbolu un nedaudz uzgaidam.

```
38   prewKey = key;
39   delay( 10 );
40 }
```

Darbiniet programmu un pārliecinieties, ka nospiestā taustiņa simbola kodu datorā redzam tikai vienreiz pat tad, ja turam to nospiestu.

Pateicības

Platīte izgatavota Latvijas Universitātes Fonda projekta "Programmējam ar prieku" ietvaros un to finansē SIA Mikrotīkls.

Literatūras saraksts

[1] Ivars Driķis. Ļoti ļoti daudz mirdzdiožu! http://home.lu.lv/~drikis/BlogaFaili/G01-Arduino/2018/a05-led8x8.pdf.

[2] Ivars Driķis. Digitālās ieejas, tas nemaz nav tik vienkārši. http://home.lu.lv/~drikis/BlogaFaili/G01-Arduino/2018/a06-ieejas.pdf.

[3] Arduino Reference. Keypad library. https://playground.arduino.cc/Code/Keypad/. [Online; accessed 2019.09.16].

[4] Ivars Driķis. Attāluma mērīšana, arduino bibliotēkas. http://home.lu.lv/~drikis/BlogaFaili/G01-Arduino/2018/a01-arduinoLibs.pdf.