Digitālās ieejas, tas nemaz nav tik vienkārši

Ivars Driķis

2019. gada 14. septembrī

1 Ievads

Bērnībā uzmanīgi lasīju E. Aisberga grāmatas, kas stāstīja, ka radio, televīzija un tranzistors ir ļoti vienkārši, tikai krāsainā televīzija ir gandrīz vienkārši [1, 2, 3, 4]. Pat šodienas krāsu televizori ir ievērojami sarežģītāki par Arduino. Un kas varētu būt vienkāršāk par signāla nolasīšanu no digitālās ieejas? Izrādās, ka tas tā īsti nav. Tāpēc es saku, "digitālās ieejas, tas nemaz nav tik vienkārši".

2 Kā pareizi pacel kāju?

Strādāsim ar interaktīvo paneli, kas attēlots 1 attēlā. Uz tā izvietotas trīs nefiksējošās spiedpogas, trīs spīddiodes un vēl divas ķēdes, par kuram detalizēti runāsim vēlāk.

Saslēdzam shēmu, kā tas dots 2 attēlā un izveidojam vienkāršu programmu, kas lasa pogu stāvokli (nospiesta vai nenospiesta) un rezultātu attēlo uz spīddiodēm.

Interaktīvā paneļa spīddiodes ir pieslēgtas pie digitālajiem izvadiem 2, 4 un 6. Savukārt spiedpogas ir pieslēgtas digitālajām ieejām 3, 5 un 7. Šī programmas daļa kalpo kā dokumentācija shēmas slēgšanai. Tā kā mainīgie ir const nozīmē, ka kompilators dotās vērtības saliek visās vajadzīgajās vietās un paši mainīgie netiek veidot un ietaupīta Arduino atmiņa.

Listing 1: Panel01.ino

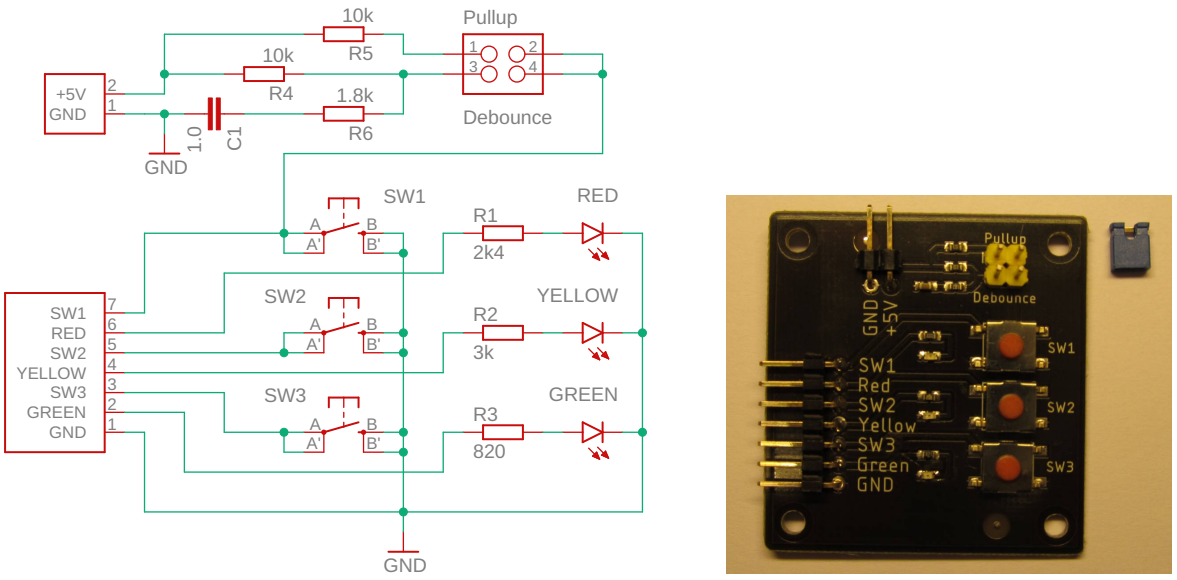
```
1  const int pinW1 = 7;
2  const int pinW2 = 5;
3  const int pinW3 = 3;
4  const int ledR = 6;
5  const int ledY = 4;
6  const int ledG = 2;
```

Inicializējam spīddiožu izejas un spiedpogu ieejas.

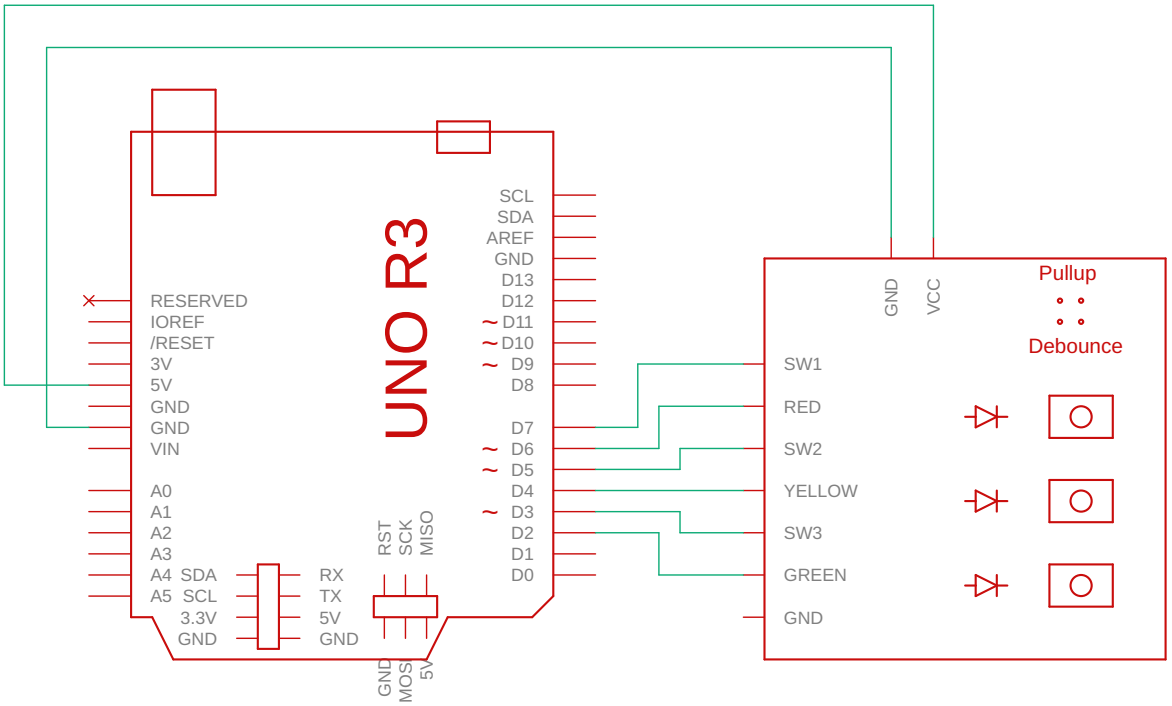
```
8  void setup()
9  {
10   pinMode(ledR, OUTPUT);
11   pinMode(ledY, OUTPUT);
12   pinMode(ledG, OUTPUT);
13   pinMode(pinW1, INPUT);
14   pinMode(pinW2, INPUT);
15   pinMode(pinW3, INPUT);
16 }
```

Tālāk nolasām pogu stāvokli un attēlojam to uz spīddiodēm.

```
18 void loop()
19 {
20   digitalWrite(ledR, digitalRead(pinW1));
```



Att. 1: Interaktīvais panelis, shēma un foto.



Att. 2: Interaktīvā paneļa savienojums ar Arduino

```
21    digitalWrite(ledY, digitalRead(pinW2));
22    digitalWrite(ledG, digitalRead(pinW3));
23    delay(100);
24 }
```

Šajā programmā tika izmantotas četras funkcijas. Ļoti iesaku sākt radināties arī lasīt oriģinālo dokumentāciju

pinMode(pin, mode) - konfigurē norādīto digitālo līniju vai nu par ieeju vai arī par izeju, [5]:

pin: digitālās līnijas numurs

mode: ja OUTPUT, tad izeja, ja INPUT vai INPUT_PULLUP tad ieeja.

digitalRead(pin) - lasa signāla līmeni no digitālās ieejas, [6]. Atgriež HIGH vai LOW:

pin: digitālās līnijas numurs

digitalWrite(pin, value) - norādīto pinu uzstāda zemā vai augstā līmenī, [7]

pin: digitālās līnijas numurs

value: spriegums digitālās līnijas izejā, LOW vai HIGH

delay(ms) - aptur programmas izpildi uz noteiktu laiku [8];

ms: laiks milisekundēs no 0 līdz $2^{32} - 1$. Tātad ilgākais laiks ir apmēram 50 diennaktis!

Pārbaudot šīs programmas darbību nākas secināt, ka tā nestrādā. Spaidot pogas nekas nenotiek. Reizēm kāda no mirdzdiodēm iespīdās, un tas notiek ja ar pirkstu pieskarās pareizā vietā, piemēram pie pogas augšējiem izvadiem.

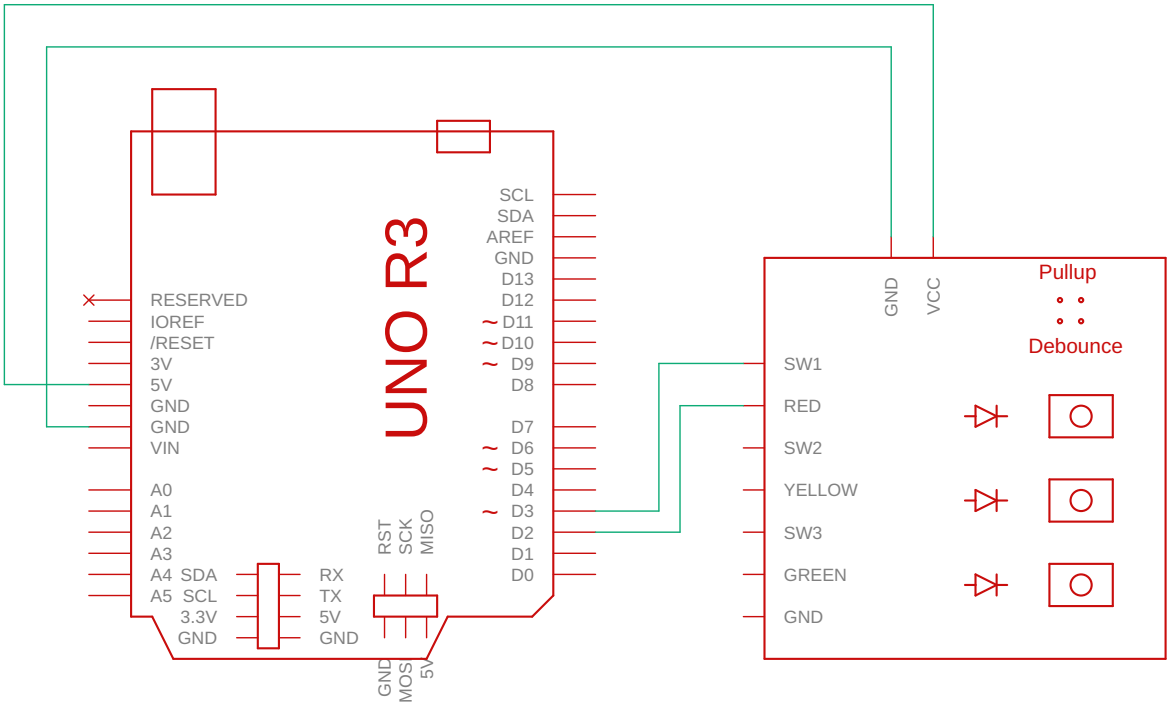
Situācija daļēji atrisinās, ja ar savienotājelelentu savieno "Pullup" kontaktus. Tad sarkanā spīddiode sāk strādāt tā, kā tam vajadzētu būt. Kas notiek? Uz digitālo ieeju D7 caur rezistoru R5 tiek padots spriegums 5V. Ja poga SW1 tiek nospiesta, tad spriegums uz digitālo ieeju D7 ir 0V. Tātad, nospiežot pogu, sarkanā spīdiode dziest. Rezistoru R5 sauc par "pull-up" rezistoru. Izrādās, ka "pull-up" rezistori ir iebūvēti mikrokontrolierī un tos var ieslēgt izmantojot INPUT_PULLUP opciju funkcijā pinMode(pin, mode).

Listing 2: Panel02.ino

```
8  void setup()
9  {
10   pinMode(ledR, OUTPUT);
11   pinMode(ledY, OUTPUT);
12   pinMode(ledG, OUTPUT);
13   pinMode(pinW1, INPUT_PULLUP);
14   pinMode(pinW2, INPUT_PULLUP);
15   pinMode(pinW3, INPUT_PULLUP);
16 }
```

3 Kā pārslēgt spīddiodi ar spiedpogu

Vispirms saslēgsim shēmu, kas dota 3 zīmējumā. Tur pie Arduino pieslēgta gan augšējā spiedpoga gan arī augšējā sarkanā spīddiode. Izveidosim programmu 3, kas



Att. 3: Interaktīvā paneļa savienojums ar Arduino

- ieslēdz spīddiodi, ja nospiež pogu,
- izslēdz spīddiodi, ja pogu nospiež otrreiz,
- un atkal no sākuma

Vispirms nodefinējam Arduino izeju, pie kuras ir pieslēgta spīddiode un ieeju, pie kuras savukārt ir pieslēgta spiedpoga.

Listing 3: SpParslegt.ino

```
1 const byte ledPin = 3;
2 const byte swiPin = 2;
```

Tālāk definējam globālos mainīgos, kuros glabāsies pašreizējais spīddiodes stāvoklis kā arī iepriekšējais spiedpogas stāvoklis. Šeit mums spīddiode sākumā būs izslēgta un spiedpoga nenospiesta. Kā redzēsim tālāk, šo pirmo spiedpogas stāvokļa vērtību mēs nekad nelietosim, bet es tomēr ieteiktu rīkoties aprakstītajā veidā jo tādējādi mēs paši sev atgādinām, ka nenospiests pogas stāvoklis nozīmē HIGH signālu Arduino ieejā.

```
4 byte ledState = LOW;
5 byte swiState = HIGH;
```

Vispirms inicializējam spīddiodes izeju un uzstādam tas stāvokli atbilstoši mainīgā ledState vērtībai.

```
7 void setup( void )
8 {
9   pinMode(ledPin, OUTPUT);
10  digitalWrite(ledPin, ledState);
```

Tālāk inicializējam spiedpogas ieeju neaizmirstot ieslēgt pull-up rezistoru. Lai programma korekti strādātu ari gadījumā, kad Arduino pārstartēšanās laikā spiedoga ir nospiesta, nolasām un noglabājam spiedpogas stāvokli.

```
12   pinMode(swiPin, INPUT_PULLUP);
13   swiState = digitalRead( swiPin );
14 }
```

Visbeidzot programmas galvenā daļa. Vispirms nolasām spiedpogas stāvokli un noglabājam to pagaidu mainīgajā curSwiState.

```
16 void loop( void )
17 {
18   byte curSwiState = digitalRead( swiPin );
```

Tālāk ķeram to brīdi, kad spiedpoga tiko ir nospiesta: iepriekšējais pogas stāvoklis bija nenospiesta, tagad jāpārbauda vai poga ir nospiesta. Ja jā, tad pārslēdz spīddiodi.

```
20   if( (swiState==HIGH) && (curSwiState==LOW) ) {
21     ledState = !ledState;
22     digitalWrite(ledPin, ledState);
23   }
```

Acīmredzot, vajadzēja arī atjaunināt swiState vērtību. Taču jāatceras, ka jāapstrādā arī gadījums, kad spiedpoga tiek atlaista. Apvienojam šos abus uzdevumus un atjaunināms swiState vērtību tikai tad, ja spiedpogas stāvoklis ir mainījies

```
25   if( swiState != curSwiState )
26     swiState != curSwiState;
27 }
```

Izmēģinām programmu, un atkal strādā ne tā kā gribējām. Šoreiz savienojam ”Debounce” kontaktus un redzam, ka programma strādā ievērojami labāk. Iemesls problēmām ir tas, ka nospiežot pogu vienu reizi kontaktplāksnes savienojas un atvienojas vairākkārt, pie kam tas ne vienmēr notiek tieši tāpat kā iepriekšējā reizē. Šo parādību novērš kondensators C1 ķēdē ar rezistoriem R4 un R6, sk. 1 zīmējumu.

Nevēlamo vairākkārtīgo ieslēgšanos var novērst programmātiski vienkārši nogaidot brīdi pēc pogas nospiešanas

Listing 4: SoftDebounce.ino

```
20   if( (swiState==HIGH) && (curSwiState==LOW) ) {
21     ledState = !ledState;
22     digitalWrite(ledPin, ledState);
23     delay( 100 );
24   }
```

4 Cik reižu nospiežam pogu?

Varbūt kādam ir interesanti cik reižu pogas nospiešanas laikā savienojas atvienojas kontakti. Apskatīsim programmu, kas ļauj to noskaidrot. Skaitīsim un izvadīsim rezultātus uz datora ekrāna. Konstatēt to brīdi, kad poga tiek nospiesta un var konstatēt aparātiski. Tas ir, mēs Arduino ieeju inicializējam tā, lai brīdī kad signāls uz ieejas mainās no HIGH uz LOW, tiktu izsaukta funkcija no programmas, kas veic visas ar šo notikumu saistītājās operācijas. Šādu risinājumu sauc par pārtraukuma apstrādi [9, 10] un lieto tad, ja notikums ir jāapstrādā nekavējoties.

Atbilstoši dokumentācijai, Arduino Uno savietojamajiem mikrodatoriem tikai divas digitālās ieejas 2 un 3 var tikt inicializētas ārējo pārtraukumu uztveršanai. Programmu sākam nodefinējot kuru no šīm ieejām izmantosim

Listing 5: SpSkaiti.ino

```
1 const byte interruptPin = 2;
```

Tālāk pārtraukuma apstrādes daļa. Vispirms definējam mainīgo, kas skaitīs kontaktu savienošanās - atvienošanās reizes int count. Atslēgas vārds volatile norāda kompilatoram ka nevajag pārcensties ar šā mainīgā vietas programmā optimizāciju. Definējam ar funkciju add, kas katra pārtraukuma reizē palielinās mainīgā count vērtību par 1.

```
3 volatile int count = 0;
4 void add( void ) { count++; }
```

Izveidosim seriālo savienojumu ar datoru

```
6 void setup( void )
7 {
8   Serial.begin( 9600 );
```

Inicializējam pārtraukuma apstrādes ieeju. Vispirms tā ir ieeja ar pull-up rezistoru. Izrādās, ka ieejai 2 atbilst pārtraukums 0 bet ieejai 3 pārtraukums 1. Lai ieejas pārslēgšanu varētu veikt mainot tikai vienu vietu programmā, izmantojam funkciju digitalPinToInterrupt kas veic šo pārkodēšanu. Visbeidzot uzstādām pārtraukuma apstrādes funkciju un norādām ka tas nostrādās ja signāls uz ieejas mainās no HIGH uz LOW.

```
9   pinMode(interruptPin, INPUT_PULLUP);
10  int ic = digitalPinToInterrupt(interruptPin);
11  attachInterrupt(ic, add, FALLING);
12 }
```

Tālāk programmas galvenā daļa. Vispirms definējam mainīgo lastCount, kas glabā iepriekšējo skaitītāja vērtību. Tas varēja būt globālais mainīgais, taču tā kā tas tiek izmantots tikai funkcijā loop, tāpēc pareizāk to kā lokālo statisko mainīgo.

```
14 void loop( void )
15 {
16 static int lastCount = -1;
```

Ja skaitītāja vērtība ir mainījusies spiedpogas nospiešanas rezultātā, tad izmaiņas tiek aizsūtītas uz datoru izmantojot seriālo savienojumu.

```
18   if( count != lastCount ) {
19     Serial.println( count );
20     lastCount = count;
21     delay( 100 );
22   }
23 }
```

Atveram uz datora seriālo monitoru un novērojam programmas darbību gan ar savienotiem gan ar atvienotiem "Debounce" kontaktiem.

Pateicības

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

Literatūras saraksts

[1] E. Aisbergs. *Radio? Tas ir ļoti vienkārši!* Liesma, Rīga, 1968.

[2] E. Aisbergs. *Tranzistors? Tas ir ļoti vienkārši!* Liesma, Rīga, 1967.

[3] E. Aisbergs. *Televīzija? Tas ir ļoti vienkārši.* Liesma, Rīga, 1967.

[4] E. Aisbergs and Ž. Durī. *Krāsainā televīzija? Tas ir gandrīz vienkārši.* Liesma, Rīga, 1971.

[5] Arduino Reference. pinMode(). https://www.arduino.cc/en/Reference/PinMode. [Online; accessed 2016.09.18].

[6] Arduino Reference. digitalRead(). https://www.arduino.cc/en/Reference/DigitalRead. [Online; accessed 2016.11.04].

[7] Arduino Reference. digitalWrite(). https://www.arduino.cc/en/Reference/DigitalWrite. [Online; accessed 2016.09.18].

[8] Arduino Reference. delay(). https://www.arduino.cc/en/Reference/Delay. [Online; accessed 2016.09.18].

[9] Wikipedia, Interrupt. https://en.wikipedia.org/wiki/Interrupt. [Online; accessed 2019.09.14].

[10] Nick Gammon. Interrupts. http://www.gammon.com.au/interrupts. [Online; accessed 2019.09.14].