Arduinoøving 2

IELET1002 - Datateknikk

Gunnar Myhre, BIELEKTRO

3. februar 2022

Oppgåve 1 - Fleirvalgsoppgåver

1.

Integrated Development Enviroment

2.

a) Fila er endra sidan sist lagring.

3.

b) Nei, men programmet vert kompilert av avr-gcc og eventuelle feilmeldinger frå kompilatoren vil visast i loggen.

4.

c) Strengt tatt usann. *Upload* er som *Verify* men med eitt ekstra steg: å laste det kompilerte resultatet over på arduinoen.

5.

b) Mappe og fil.

6.

a) Kodelinje som markøren er på.

7.

a) Boards manager.

8.

c) CTRL + R

9.

c) CTRL + SHIFT + M

10.

a) Det stemmer.

11.

d)

12.

c) *Syntax Highlighting*/syntaksgjenkjenning.

Oppgåve 2 - *analogRead()* og spenningsdeling

a)

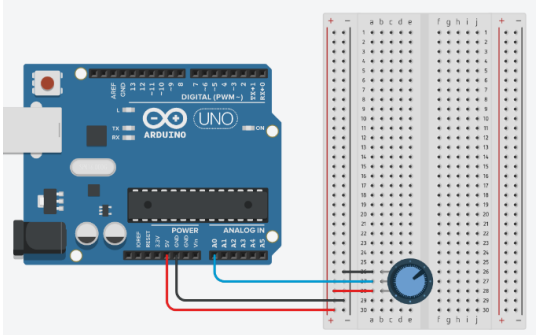
I øving 1 (oppg. 2b) diskuterte eg tilfellet der lasta forbruker høgare effekt enn arduinoen er i stand til å forsyne, og eg rekna ut at denne effekten i tilfellet med $5V$ over $10\text{k}\Omega$ er $2,5\text{mW}$. Eg tester eksperimentelt kva som er verdiene avlest av arduinoen for fire forskjellige potmetere dreid til sine ytterpunkter.

Potmeter nr.	Lågaste verdi	Høgaste verdi
$10K_1$	0	1023
$10K_2$	0	1023
$10K_3$	0	1023
$10K_4$	0	1023

eg gjentar eksperimentet med forskjellige USB-portar og med ladaren til laptopen dratt ut, og får dei same resultata som i tabellen over.

Eg gjetter på at dette skyldast at mine potmetereksemplarer er av tilstrekkeleg kvalitet til å gje korrekt verdi i ytterpunktene.

```
#define POTMETER_PIN A0

void setup(){
  pinMode(POTMETER_PIN, INPUT);
  Serial.begin(9600);
}

int potMeterReading;

void loop(){
  potMeterReading = analogRead(POTMETER_PIN);
  Serial.println(potMeterReading);
}
```



b)

Eg gjer som i øving 1 (oppg. 2c) og finner eit forholdstal for å rekne om til spenning

```
//Rekner ut spenning i A0 og printer til Serial
pinVoltage = potMeterReading * 0.004887586;
Serial.println(pinVoltage);
```

Det vi måler her er spenningsfallet over den delen av R_{pot} som til ein kvar tid befinner seg mellom $A0$ og $5V$.

c)

Tilsvarande som i øving 1 (oppg. 2d) finner eg spenninga vha. spenningsdeling.

```
const int potMeterPin = A0;
const int potMeterTotalResistance = 10000;

int potMeterReading;
float pinVoltage;
float potMeterResistance;


void setup(){
  pinMode(potMeterPin, INPUT);
  Serial.begin(9600);
  }

void loop() {
  potMeterReading = analogRead(potMeterPin);

  // Calculate voltage from 10 bit value
  // Constant 0.004887586 is derived from
  // k = 5V / 1023 (value span)
  pinVoltage = potMeterReading * 0.004887586;

  // Calculate resistance from 10 bit value
  potMeterResistance = potMeterTotalResistance - (
      pinVoltage * potMeterTotalResistance) / 5;
  Serial.print("voltage: ");
  Serial.print(pinVoltage);
  Serial.print(" resistance: ");
  Serial.println(potMeterResistance);

  delay(10);
}
```

d)

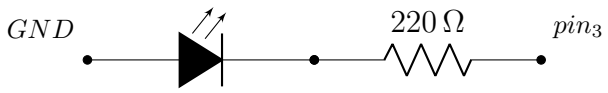
Dette løyste eg med å lage ein funksjon.

```
1 const int potMeterPin = A0;
2 const int potMeterTotalResistance = 10000;
3
4
5 int valueToRange(float value, int numOfDivisions, int
    valueSpan){
6   /* Divides valueSpan into numOfDivisionsand check which
7    *  division the provided value is in => (0 -
       numOfDivisions).
8    */
9   int valueInRange;
10  int division = valueSpan / numOfDivisions;
11
12  for (int i = 0; i < numOfDivisions; i++){
13    if (value < division * (i+1)){
14      valueInDivision = i;
15      break;
16    }
17  }
18  return valueInDivision;
19 }
20
21
22 void setup(){
23   pinMode(potMeterPin, INPUT);
24   Serial.begin(9600);
25   }
26
27 void loop() {
28   int potMeterReading = analogRead(potMeterPin);
29
30   // Find which range the value is in
31   int range = valueToRange(potMeterReading, 4, 1024);
32
33   Serial.println(potMeterReading);
34   Serial.println(range);
35
36
37   delay(100);
38 }
```

Oppgåve 3 - *analogWrite()* og PWM

a)

Eg kopler LEDen i serie med ein 220Ω-motstand mellom jord og pin 3.

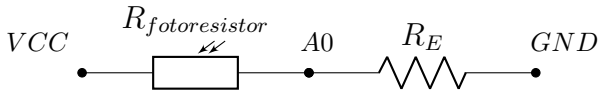



```
1 #define LED_PIN = 3;
2
3 int ledPinState = 0;
4 bool waveIsRising = true;
5
6 void setup(){
7   pinMode(LED_PIN, OUTPUT);
8 }
9
10
11 void loop() {
12
13   // ledPinState triangle wave
14   if (waveIsRising) {
15     if (ledPinState < 255){
16       ledPinState ++;
17     } else {
18       waveIsRising = false;
19     }
20   } else {
21     if (ledPinState > 0) {
22       ledPinState --;
23     } else {
24       waveIsRising = true;
25     }
26   }
27
28   analogWrite(LED_PIN, ledPinState);
29   delay(5);
30 }
```

Denne koden produserer ein jamn triangelbølgeform på pin 3 vha. PWM.

b)

Eg måler motstanden over fotoresistoren og ser at den har eit verdispenn på omtrent 30kΩ. Under normal belysning i rommet har fotoresistoren ein motstand $1,8\text{k}\Omega$



spenninga i A0 er dermed gitt som

$$v_{A0} = \frac{R_{fotoresistor}}{R_E + R_{fotoresistor}} 5V \tag{1}$$

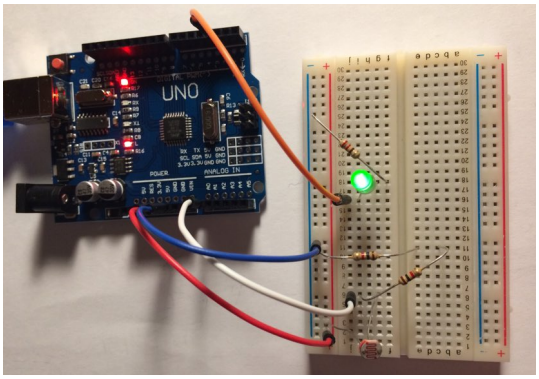
Om R_E er for stor eller for liten vil verdispennet i $A0$ gå mot 0. Derfor velger eg $R = 2\text{k}\Omega$ som eit utgongspunkt sidan det tilsvarar ein verdi ein kan forvente i eit normalt opplyst rom.

For å tilpasse verdien til lysdioden lager eg meg to skalaer. Botnpunktet til fotoresistorverdien setter eg til 150, som er ein verdi det er lett å halde seg under når ein blokkerer lyset inn til fotoresistoren med tri fingrar.

Lysverdi	Lågast	Høgast
fotoresistor	150	1023
LED	0	255

Til dette kan vi bruke arduinos *map()*-funksjon.

```
1  lightSensorReading = analogRead(PHOTORESISTOR_PIN);
2  if (lightSensorReading < DARK_THRESHOLD) {
3    ledValue = 0;
4  } else {
5    ledValue = map(lightSensorReading, DARK_THRESHOLD,
         1023, 0, 255);
6  };
```



Oppgåve 4 - Random

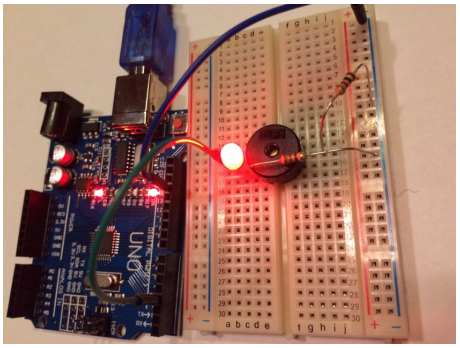
a)

```
#define RANDOM_SEED_PIN 2 // Pin 0 must not be used

int randomNumber;

void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(RANDOM_SEED_PIN));
}

void loop(){
  randomNumber = random(1,7);

  Serial.println(randomNumber);
  delay(1000);
}
```

b)

Eg velger å bruke *random*-generatoren på ein litt annan måte.

```
const int randomSeedPin = 2; // This pin must remain unused
const int buzzerPin = 3;
const int fundamentalFreq = 10;
const int toneDuration = 100;


int randomTone;
int currentFreq;


void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(randomSeedPin));
  pinMode(buzzerPin, OUTPUT);
}


void loop(){
  /* Plays randomly generated music on a piezoelectric buzzer.
   *  The default fundamental frequency of 10 Hz puts about 10-20%
   *  of the notes generated below the audible range, which makes sounds
   *  reminiscent of drums.
   */
  randomTone = random(1,13);

  // Random pure intonation harmonics based on random integer
  currentFreq = fundamentalFreq * randomTone;

  tone(buzzerPin, currentFreq, toneDuration);
  Serial.println(currentFreq);
  delay(toneDuration);
}
```



c)

Dette løyste eg på denne måten.

```
1  const int randomSeedPin = 2; // This pin must remain unused
2  const int testDuration = 10000;
3
4  int randomToss;
5  int tossAmounts[6];
6  char strBuf[64];
7
8  void setup(){
9    Serial.begin(9600);
10   randomSeed(analogRead(randomSeedPin));
11 }
12
13
14 void loop(){
15   // Perform a test of the random generator
16   for (int i=0; i<testDuration; i++){
17     randomToss = random(0,6);
18     tossAmounts[randomToss]++;
19   }
20
21   // Print the test results
22   sprintf(strBuf, "Test concluded after %d iterations",
       testDuration);
23   Serial.println(strBuf);
24   for (int i = 0; i < 6; i++){
25     sprintf(strBuf, " num %d: ", i+1);
26     Serial.print(strBuf);
27     Serial.println(tossAmounts[i]);
28   }
29
30   // Reset the tossAmounts array
31   memset(tossAmounts, 0, sizeof(tossAmounts));
32
33   delay(1000);
34 }
```

Dersom eg ville hatt svara som prosent ville eg gjort om

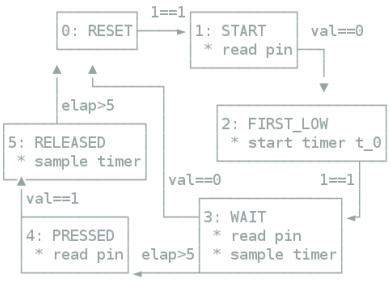
```
1      Serial.println(tossAmounts[i] / testDuration);
```

d)

Først implementerte eg ein versjon med *delay*-kall:

```
const int randomSeedPin = 2; // This pin must remain unused
const int buttonPin = 4;
const int testDuration = 10000;

int randomToss;
int tossAmounts[6];
char strBuf[64];


void buttonWait(int pin){
  bool isPressed;

  for (;;){
    isPressed = digitalRead(pin);
    if (isPressed) {
      break;
    }
    delay(10);
  }
}


void setup(){
  Serial.begin(9600);
  randomSeed(analogRead(randomSeedPin));
  pinMode(buttonPin, INPUT);
}


void loop(){
  // Wait for user to press button
  Serial.println("Press button to perform test.");
  buttonWait(buttonPin);
  Serial.println("   testing...");

  // Perform a test of the random generator
  for (int i=0; i<testDuration; i++){
    randomToss = random(0,6);
    tossAmounts[randomToss]++;
  }

  // Print the test results
  sprintf(strBuf, "Test concluded after %d iterations", testDuration);
  Serial.println(strBuf);
  for (int i = 0; i < 6; i++){
    sprintf(strBuf, " num %d: ", i+1);
    Serial.print(strBuf);
    Serial.println(tossAmounts[i]);
  }

  // Reset the tossAmounts array
  memset(tossAmounts, 0, sizeof(tossAmounts));
}
```

Når eg implementerte trykknappkretsen vart eg klar over problemet med knappeprell. Ein måte å løyse dette på som har mange andre fordelar er å implementere programmet vha. separate tilstandsmaskiner. Eg teikna eit tilstandsdiagram for knappen som tar hensyn til at transienten har mykje støy dei første 5 millisekunda.



Dette implementerte eg på denne måten:

```
#define BUTTON_1_PIN 6
#define BUTTON_2_PIN 5
#define BUTTON_3_PIN 4

const bool WRITE_TO_SERIAL = true;

class ButtonStateMachine {
  /* Button state machine class with debouncing
   *  on both push and release transients.
   *  Pull down switch: vcc--[SWITCH]--sample--[R=220]--gnd */
  private:
    bool val;
    uint8_t pin;
    unsigned long t;
    unsigned long t_0;
    const uint8_t bounce_delay = 5;

  public:
    uint8_t state_prev = State::RESET;
    bool state_has_changed;

    enum State : uint8_t {
      RESET = 0,
      START,
      FIRST_LOW,
      WAIT,
      PRESSED,
      RELEASED
    };
    State state = State::RESET;

    ButtonStateMachine(uint8_t pin) {
      this->pin = pin;
      Init();
    }

    void Init(){
      pinMode(pin, INPUT);
    }

    void NextState(){
      state_prev = state;

      switch (state){
        case State::RESET:
          state = State::START;
        break;

        case State::START:
          val = digitalRead(pin);
          if (val == LOW) {state = State::FIRST_LOW;}
        break;

        case State::FIRST_LOW:
          t_0 = millis();
          state = State::WAIT;
        break;

        case State::WAIT:
          val = digitalRead(pin);
          t = millis();
          if (val == LOW) {state = State::RESET;}
          if (t - t_0 > bounce_delay){state = State::PRESSED;}
        break;

```

```
66          case State::PRESSED:
67            val = digitalRead(pin);
68            t_0 = millis();
69            if (val == HIGH) {state = State::RELEASED;}
70          break;
71
72          case State::RELEASED:
73            t = millis();
74            if (t - t_0 > bounce_delay){state = State::RESET;}
75          break;
76        }
77        state_has_changed = (state_prev != state);
78      }
79  };
80
81
82  // Global object instantiations
83  ButtonStateMachine button_1(BUTTON_1_PIN);
84  ButtonStateMachine button_2(BUTTON_2_PIN);
85  ButtonStateMachine button_3(BUTTON_3_PIN);
86
87
88  void setup(){
89    if (WRITE_TO_SERIAL) {Serial.begin(9600);}
90  }
91
92
93  void loop(){
94    button_1.NextState();
95    button_2.NextState();
96    button_3.NextState();
97
98    // Debug
99    if (WRITE_TO_SERIAL) {
100     if (button_1.state_has_changed){
101       if (button_1.state == ButtonStateMachine::State::PRESSED) {Serial.println("BUTTON_
              ONE");}
102       Serial.print("_state_s1:_");
103       Serial.println(button_1.state);
104     }
105     if (button_2.state_has_changed){
106       if (button_2.state == ButtonStateMachine::State::PRESSED) {Serial.println("BUTTON_
              TWO");}
107       Serial.print("_state_s2:_");
108       Serial.println(button_2.state);
109     }
110     if (button_3.state_has_changed){
111       if (button_3.state == ButtonStateMachine::State::PRESSED) {Serial.println("BUTTON_
              THREE");}
112       Serial.print("_state_s3:_");
113       Serial.println(button_3.state);
114     }
115   }
116 }
```

Dette designmønsteret har mange fordelar

- Det er nå mogleg å ha fleire samtidige prosessar, i motsetning til om ein bruker *delay*-kall
- Systemet har ein endeleg mengde tilstandar, noko som gjer det mogleg å rapportere heile systemets tilstand i korte trekk.
- Dette gjer det også lettare å lese av ein tidsbruksprofil av systemet for å optimalisere overgangane og gjere systemet raskare.
- Sidan *ButtonStateMachine* er ein objektdefinisjon er det lett å gjenbruke koden og instansiere så mange knappeobjekter ein vil.