Moving Average am ATtiny

Date: 2023-12-17

Contents

1 Das Programm 1

2 Der ATtiny85 hat 2
2.1 Fließkomma 2

3 Moving average 3
3.1 Wie stelle ich mir Moving Average vor 4
3.2 `python` Visualisierung 4
3.3 Samplerate 5
3.4 Schlussfolgerung 6

4 Small code cleanups 6

5 Das Programm nachher 6

1 Das Programm

Zum leichteren Test der Auswirkungen von Änderungen, testen wir das Programm am arduino UNO ... wir können die Daten über Serial zum PC schicken.

```
// heartRateMonitor arduino test program

//#include <DigisparkOLED.h>

int actualValue = 0;            //aktueller Sensorwert
unsigned int movAverage = 500;  // gleitender Mittelwert
unsigned long lastPulseTime;
int duration = 1000;  // Abstand (Zeit) zwischen zwei Pulssequenzen. Wert wird später
int pulse = 60;
//int movAveragePulse = 60;
int ThisTime;

bool state = false;

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

  pinMode(PB3, INPUT);
  pinMode(PB4, OUTPUT);
}

void loop() {
  actualValue = analogRead(3);

  ThisTime = millis();

  //***************** gleitender Mittelwert (Lowpass-Filter )**************
  movAverage = (movAverage * 92 + actualValue * 8) / 100;  // wobei n < 1 eigentlicher
```

```
  //wenn aktueller Wert über Mittelwert, dann starte den Puls
  if (actualValue > (movAverage + 15)) {
    if (state == false) {  // Pulsschlag erkannt
      state = true;
      duration = ThisTime - lastPulseTime;
      lastPulseTime = ThisTime;
      digitalWrite(PB4, HIGH);
      /* oled.setCursor(50, 4);
      oled.setFont(FONT8X16);
      oled.print("BEAT");*/
    }
  }
  // wenn Pulse detektiert (Peak) aber Wert wieder unter gleitenden Mittelwert fällt,
  if (state == true) {
    if (actualValue < (movAverage)) {
      state = false;
      digitalWrite(PB4, LOW);
      /* oled.setCursor(50, 4);
      oled.setFont(FONT8X16);
      oled.print("    ");*/
    }
  }

  pulse = 60000.0 / duration;

  // nochmalige Mittelwertbildung um Ausreißer in der Detektion zu glätten
  // movAveragePulse = (movAveragePulse * 0.85) + (pulse * (1.0 - 0.85));  // wobei n

  Serial.print(pulse);
  Serial.print(",");
  Serial.print(actualValue);
  Serial.print(",");
  Serial.println(movAverage + 15);

  delay(50);
}
```

2 Der ATtiny85 hat

- 8kB Flash/Programmspeicher
- 512 Byte RAM/Arbeitsspeicher
- und keinen Fließkommabefehle

2.1 Fließkomma

Das bedeutet Fließkommaberechnungen werden emuliert und brauchen viel Zeit (kein Problem) und viel Platz ... den wir nicht haben.

Es ist nur eine Fließkommaoperation im Code.

Codegröße (Sketch uses) 3264 bytes, wenn die Zeile mit FLießkomma rechnet

```
pulse = 60000.0 / duration;
```

2698 Bytes bei Ganzzahldivision

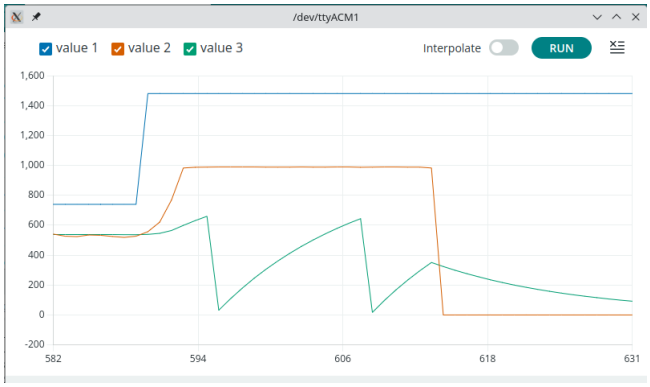
```
pulse = 60000 / duration;
```

3 Moving average

Mittelwertberechnung

```
movAverage = (movAverage * 92 + actualValue * 8) / 100;  // wobei n < 1 eigentlicher g
```

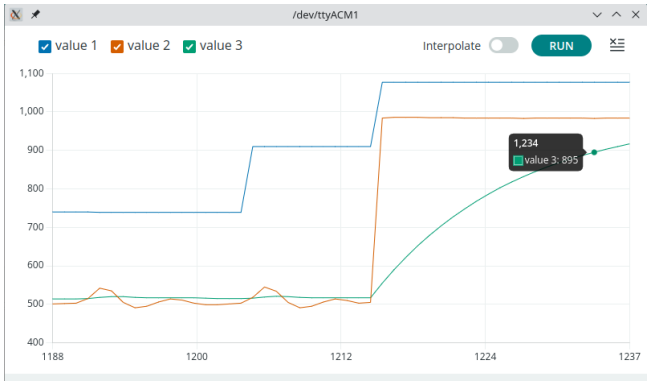
Gefährlich weil 1023 der Maximalwert beim 10ADC ist und 65535 das Maximum für einen `unsigned int` 16 bit.



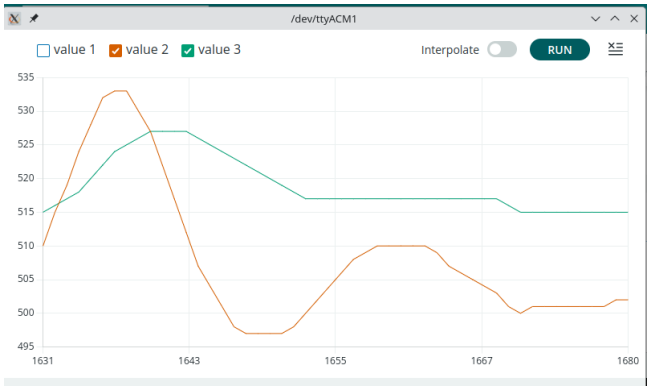
Wir verkleinern auf Multiplikation mit 23

```
movAverage = (movAverage * 23 + actualValue * 2) / 25;  // wobei n < 1 eigentlicher ge
```

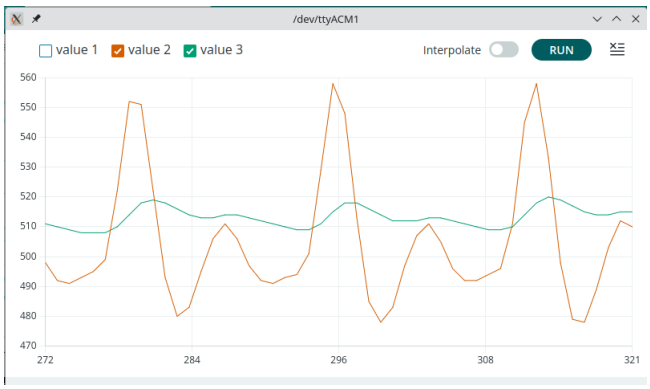
Die Steigung der Kurve (Pseudoladekurve) sieht ähnlich aus.



Und das delay am Ende mit 10 ms hat auch Einfluß auf den Mittelwert.



bei 40ms reagiert der Mittelwert langsamer.



3.1 Wie stelle ich mir Moving Average vor

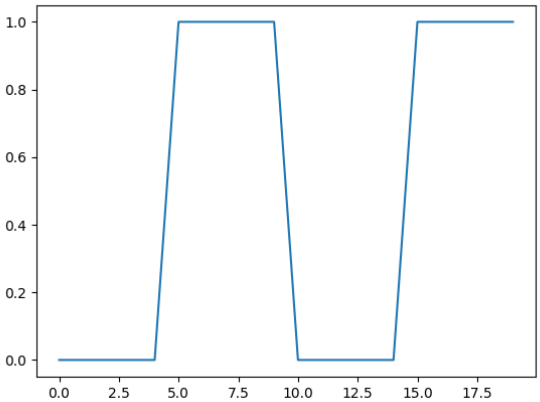
Wenn ich ein Signal habe, dass von `0` auf `1` springt ... was macht der `moving average` daraus ?

```
movavg = (sum + actualvalue / 2)
```

val	sum	movaverage
0	0	0
1	0	0.5
1	0.5	0.75
1	0.75	0.875
1	0.875	0.9375
1	0.9375	0.96875

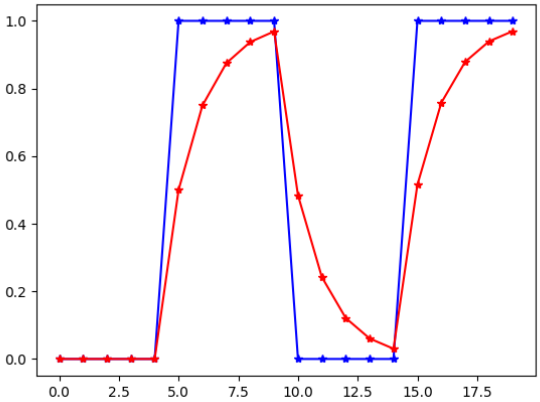
3.2 `python` Visualisierung

```
x = list(range(20))
# [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

y = [(a % 10)//5 for a in x]
# [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```



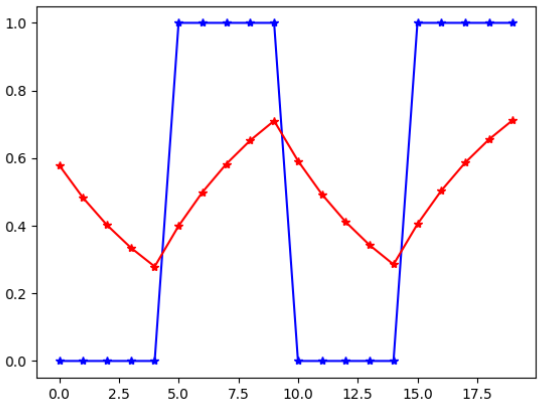

```
def movavg(a):
    global avg
    avg = (avg + a)/2
    return avg
```

```
avg = 0
z = [movavg(a) for a in x]
# [0.0, 0.0, 0.0, 0.0, 0.0,
#  0.5, 0.75, 0.875, 0.937, 0.968,
#  0.484, 0.242, 0.121, 0.060, 0.030,
#  0.515, 0.757, 0.878, 0.939, 0.969]
fig, ax = plt.subplots()
ax.plot(x, y,'b*-', z, 'r*-')
plt.show()
```



Wenn wir die Änderung des Mittelwerts klein halten wollen müssen wir das Gewicht der bisherigen Werte erhöhen.

```
def movavg(a):
    global avg
    avg = (avg*5 + a)/6
    return avg

avg = 0
z = [movavg(a) for a in x]
# [0.0, 0.0, 0.0, 0.0, 0.0,
# 0.166, 0.305, 0.421, 0.517, 0.598,
# 0.498, 0.415, 0.346, 0.288, 0.240,
# 0.366, 0.472, 0.560, 0.633, 0.694]
```



3.3 Samplerate

Das `delay` am Ende der Funktion `loop` bestimmt wieviele Messwerte der Puls breit ist und die Gewichtung im moving average muss daran angepasst werden damit die Baseline nicht zu sehr mitgeht.

ABER wenn das `delay` 50ms ist bekommen wir bei einer Herzfrequenz von 60 Schlägen die Minute eine Periodendauer von 1000 plus/minus 50

```
60000 /  950 = 63.16
60000 / 1000 = 60.00
60000 / 1050 = 57.14
```

bei 120 Schlägen die Minute

```
60000 / 450 = 133.33
60000 / 500 = 120.00
60000 / 550 = 109.09
```

3.4 Schlussfolgerung

Das bedeutet ein kleineres `delay` gibt feinere Stufen.

Ob mehr oder weniger Glättung gut ist ?

- Wenn die zweite Spitze groß ist ist mehr Glättung .. schlecht.
- Wenn nur 16-bit Zahlen verwendet werden sind folgende Werte an der Grenze vor einem Overflow.

  ```
  movAverage = (movAverage * 63 + actualValue * 1) / 64;
  ```

  Unter der Annahme, dass eine Messung nicht übersteuert ist, wir nicht nur Werte von 950 bis 1023 bekommen könnten wir auch noch etwas höher gehen.

  `(movAverage * 63 + actualValue * 1)` kann nicht über 65535 werden, ansonst müssen wir einen längeren Integertyp (uint32_t) verwenden.

4 Small code cleanups

- actualValue local und unsigned int kommen wir von 2692 auf 2622 Byte herunter.
- auch alle anderen globalen Variablen in die `loop`-Funktion verschieben und wenn notwendig (dass sie das Funktionsende überleben müssen) `static` davor schreiben.

Nachher 2604 Bytes.

Die Typumwandlungen von `unsigned` auf `signed integer` benötigen auch Codespace.

Wenn wir die Zeile

```
pulse = 60000 / duration;
```

anschauen, welcher Typ ist `60000`? Der Compiler verwendet als default 16-bit Integer mit Voreichen, 60000 ist zu groß dafür.

Wenn wir den Typ auf `unsigned 16-bit integer` festlegen

```
pulse = uint16_t(60000) / duration;
```

Sinkt die Codegröße von 5310 auf 5232

5 Das Programm nachher

```
// heartRateMonitor arduino test program

//#include <DigisparkOLED.h>
const uint8_t analogIn = A0;
const uint8_t Led = LED_BUILTIN;

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

  pinMode(analogIn, INPUT);
  pinMode(Led, OUTPUT);
```

```
}

void loop() {
  static unsigned int movAverage = 500;  // gleitender Mittelwert
  static unsigned long lastPulseTime;
  static int pulse = 60;
  static bool state = false;

  unsigned int actualValue = analogRead(analogIn);

  unsigned long ThisTime = millis();

  //***************** gleitender Mittelwert (Lowpass-Filter )***************
  movAverage = (movAverage * 23 + actualValue * 2) / 25;  // wobei n < 1 eigentlicher

  //wenn aktueller Wert über Mittelwert, dann starte den Puls
  if (state == false) {  // Pulsschlag erkannt
    if (actualValue > (movAverage + 15)) {
      state = true;
      int duration = ThisTime - lastPulseTime;
      pulse = uint16_t(60000) / duration;
      lastPulseTime = ThisTime;
      digitalWrite(Led, HIGH);
      /* oled.setCursor(50, 4);
      oled.setFont(FONT8X16);
      oled.print("BEAT");*/
    }
  }
  // wenn Pulse detektiert (Peak) aber Wert wieder unter gleitenden Mittelwert fällt,
  else {
    if (actualValue < (movAverage)) {
      state = false;
      digitalWrite(Led, LOW);
      /* oled.setCursor(50, 4);
      oled.setFont(FONT8X16);
      oled.print("    ");*/
    }
  }

  // nochmalige Mittelwertbildung um Ausreißer in der Detektion zu glätten
  // movAveragePulse = (movAveragePulse * 0.85) + (pulse * (1.0 - 0.85));  // wobei n

  Serial.print(pulse);
  Serial.print(",");
  Serial.print(actualValue);
  Serial.print(",");
  Serial.println(movAverage + 15);

  delay(40);
}
```