STRUKTUR in DATEN und CODE

Ein Versuch

engelbert gruber

20.04.2020
2ND GRADE SCIENCE

HINWEIS:

Meine Texte sind deshalb so lange, weil ich die schrittweise Entwicklung mit dokumentiere. Dieses schrittweise Vorgehen bringt es mit sich, dass man während dem Probieren konkrete Fakten über das Problem und die Lösung lernt.

INTRODUCTION

Ich nehme die Ausgabe des arduino-Voltmeter mit automatischer Bereichsumschaltung.

Die Umschaltung geschieht über zwei (oder mehrere) Anschlüsse die den Eingangsspannungsteiler verändern. Der Schalter ist offen, wenn der Anschluss INPUT ist und geschlossen wenn er OUTPUT ist und auf LOW.

EIN MESSBEREICH

Ohne etwas zu tun ist der arduino ein 0 bis 5V Messgerät.

```
void setup() {
  Serial.begin(9600);
}
const float scale_factor = 5.0 / 1024;
void loop() {
  int adc_in = analogRead(A0);
  Serial.println(adc_in * scale_factor);
}
```

Um uns beim Testen leichter zu tun machen wir uns eine globale Variable mit dem aktuellen Messwert.

```
const float scale_factor = 5.0 / 1024;
float measured;
void loop() {
  int adc_in = analogRead(A0);
  measured = adc_in * scale_factor;
  Serial.print(measured);
  Serial.println("V");
}
```

Das Testprogramm: test.cpp

```
// Testrahmen für ein arduino Programm
#include "arduino-simu.h"
#include "arduino-voltmeter-2/arduino-voltmeter-2.ino"


int main() {
   int tests = 0;
   int errors = 0;
   // test 1
   Tests++;
   simu_analogIn = 526;
   loop();
```

```
    if (measured != 2.5) {
      std::cout << "ERROR: 2.5 != " << measured << std::endl;
      errors++;
    }
    // summary
    std::cout << std::endl << "-------------" << std::endl;
    if (errors > 0) {
      std::cout << "ERROR. " << errors << " of " << tests
                << " tests failed." << std::endl;
      return 1;
    }
    std::cout << "OK. " << tests << " passed." << std::endl;
    return 0;
}
```

liefert

```
2.56836V
ERROR: 2.5 != 2.56836

-------------
ERROR. 1 of 1 tests failed.
```

Wenn wir anstatt 2.5 2.56836 einsetzen

```
2.56836V
ERROR: 2.56836 != 2.56836

-------------
ERROR. 1 of 1 tests failed.
```

Die floats unterscheiden sich auf Stellen hinter der sechsten. Wir korrigieren das

```
if (abs(measured - 2.56836) > 0.001) {
  std::cout << "ERROR: 2.56836 != " << measured << std::endl;
  errors++;
}
```

Dann “`OK. 1 passed.`”

Doppelten Code entfernen

Die 2.56836 im also doppelte Daten (Magic numbers).

```
int test_in[] = { 526 };
float test_out[] = { 2.56836 };
int main() {
   int tests = 0;
   int errors = 0;
   // tests
   for (int i=0; i<(sizeof(test_in)/sizeof(test_in[0])); i++) {
     tests++;
     simu_analogIn = test_in[i];
     loop();
     if (abs(measured - test_out[i]) > 0.001) {
       std::cout << "ERROR: " << test_out[i] << " != "
                 << measured << std::endl;
       errors++;
     }
   }
   …
```

ZWEI MESSBEREICHE

Wenn analogRead den Wert 1023 liefert, den höchsten Wert beim 10bit-Wandler kann das auch ein Overflow sein.

Offener Punkt: Overflow detection

Wir fügen den Testfall hinzu.

```
int test_in[] = { 526, 1023 };
float test_out[] = { 2.56836, 5.0 };
```

Runtest Ausgabe

```
ERROR: 5 != 4.99512

-------------
ERROR. 1 of 2 tests failed.
```

Das lassen wir einmal so stehen, weil wir da den Messbereich wechseln wollen..

```
const int max_adc = 1023;

const float scale_factor1 = 5.0 / 1024;
const float scale_factor2 = 10.0 / 1024;

float scale_factor = scale_factor1;
void loop() {
  int adc_in = analogRead(A0);
  if (adc_in == max_adc) {
    pinMode(Switch1, OUTPUT);
    digitalWrite(Switch1, LOW);
    scale_factor = scale_factor2;
  }
  else {
    Serial.println(adc_in * scale_factor);
  }
}
```

Testprogramm

Wir initialisieren measured auf -1, bei einem Overflow darf sich dieser wert nicht ändern.

```
int test_in[] = { 526, 1023 };
float test_out[] = { 2.56836, -1.0 };
int main() {
   int tests = 0;
   int errors = 0;
   // tests
   for (int i=0; i<(sizeof(test_in)/sizeof(test_in[0])); i++) {
     tests++;
     simu_analogIn = test_in[i];
     measured = -1;
     loop();
…
```

Runtest “OK. 2 passed.”

```
int test_in[] = { 526, 1023, 526 };
float test_out[] = { 2.56836, -1.0, 5.0 };
```

Runtest:

```
2.56836V
ERROR: 5 != -1
```

ACHTUNG: der arduino-simu.h vom vorigen Projekt liefert ab dem dritten analogRead immer 1023. Das muss man zurücksetzen.

```
simu_analogIn = test_in[i];
simu_cnt = 0;  // simulator liefert sonst ab dritten mal 1023
measured = -1;
loop();
```

Besser:

```
2.56836V
5.13672V
ERROR: 5 != 5.13672

-------------
ERROR. 1 of 3 tests failed.
```

Die 5.13672 in test_out einfügen: “OK. 3 tests passed”

Refactor Code

Variablennamen mit Zahlen sind … ein Hinweis es mit einem Array zu versuchen.

```
const float scale_factors[] = { 5.0 / 1024, 10.0 / 1024 };

float scale_factor = scale_factors[0];
float measured;
void loop() {
  int adc_in = analogRead(A0);
  if (adc_in == max_adc) {
     pinMode(Switch1, OUTPUT);
     digitalWrite(Switch1, 0);
     scale_factor = scale_factors[1];
```

```
    }
```

DREI MESBBEREICHE

Wenn wir jetzt den Test erweitern.

```
int test_in[] = { 526, 1023, 526, 1023, 526 };
float test_out[] = { 2.56836, -1.0, 5.13672, -1, 10 };
```

Sollte das zweite 1023 auf den nächsten Messbereich umschalten, den 20V. Aber den Code haben wir noch nicht geschrieben, deshalb

```
2.56836V
5.13672V
5.13672V
ERROR: 10 != 5.13672
```

Der dritte scale factor ist einfach, aber die Umschaltung ist hart kodiert.

```
if (adc_in == max_adc) {
    pinMode(Switch1, OUTPUT);
    digitalWrite(Switch1, 0);
    scale_factor = scale_factors[1];
}
```

Anstatt “1” brauchen wir eine Messbereichsindex.

```
const float scale_factors[] = {
   5.0 / 1024,
   10.0 / 1024,
   20.0 / 1024 };

int mb_index = 0;   // messbereichsindex
float scale_factor = scale_factors[mb_index];
float measured;
void loop() {
  int adc_in = analogRead(A0);
  if (adc_in == max_adc) {
      if (mb_index >= sizeof(scale_factors)/sizeof(scale_factors[0])) {
         Serial.println("OVERFLOW");
      }
```

```
        else {
           mb_index++;
           // TODO switch setting
           pinMode(Switch1, OUTPUT);
           digitalWrite(Switch1, 0);
           scale_factor = scale_factors[mb_index];
        }
     }
```

Runtest:

```
2.56836V
5.13672V
10.2734V
ERROR: 10 != 10.2734
```

Den Test korrigieren:

```
int test_in[] = { 526, 1023, 526, 1023, 526 };
float test_out[] = { 2.56836, -1.0, 5.13672, -1, 10.2734 };
```

Dann sind wir bei “`OK. 5 passed.`”.

Messbereichsdefinition

Das “TODO switch setting” weist darauf hin, dass ein Messbereich durch scale_factor und pin-Konfiguration definiert ist. Wir brauchen also mehr als nur

```
const float scale_factors[] = {
   5.0 / 1024,
   10.0 / 1024,
…
```

Ein Konstrukt das ein float (scale_factor) und ein int (pinMode) speichert. In C ist das ein struct (für den dritten Messbereich brauchen wir zwei pinMode):

```
struct MBstruct {
   float scale_factor;
   int sw1mode;
   int sw2mode;
};
```

Und davon ein Array:

```
const MBstruct MB[] = {
   MBstruct{ 5.0 / 1024, INPUT, INPUT },
   MBstruct{ 10.0 / 1024, OUTPUT, INPUT },
   MBstruct{ 20.0 / 1024, OUTPUT, OUTPUT }
   };
```

Und dann im Code verwenden

```
int mb_index = 0;   // messbereichsindex
float scale_factor = MB[mb_index].scale_factor;
float measured;
void loop() {
  int adc_in = analogRead(A0);
  if (adc_in == max_adc) {
     if (mb_index >= sizeof(MB)/sizeof(MB[0])) {
       Serial.println("OVERFLOW");
     }
     else {
       mb_index++;
       pinMode(Switch1, MB[mb_index].sw1mode);
       if (MB[mb_index].sw1mode == OUTPUT) {
         digitalWrite(Switch1, 0);
       }
       scale_factor = MB[mb_index].scale_factor;
     }
  }
  else {
     measured = adc_in * scale_factor;
     Serial.print(measured);
     Serial.println("V");
  }
}
```

Funktioniert.

Wir haben noch keinen Test auf die pinModes gemacht.

```
     // Testrahmen für ein arduino Programm
     #include "arduino-simu.h"
```

```
#include "arduino-voltmeter-2/arduino-voltmeter-2.ino"

const int sw1 = A1;

int test_in[] = { 526, 1023, 526, 1023, 526 };
float test_out[] = { 2.56836, -1.0, 5.13672, -1, 10.2734 };
int test_mode1[] = { INPUT, OUTPUT, OUTPUT, OUTPUT, OUTPUT };
int main() {
   int tests = 0;
   int errors = 0;
   // tests
   for (int i=0; i<(sizeof(test_in)/sizeof(test_in[0])); i++) {
     tests++;
     simu_analogIn = test_in[i];
     simu_cnt = 0;  // simulator liefert sonst ab dritten mal 1023
     measured = -1;
     loop();
     if (abs(measured - test_out[i]) > 0.001) {
       std::cout << "ERROR: " << test_out[i] << " != "
                 << measured << std::endl;
       errors++;
     }
     if (test_mode1[i] != simu_pinMode[sw1]) {
        std::cout << "ERROR: mode1 " << test_mode1[i] << " != "
                << simu_pinMode[sw1] << std::endl;
        errors++;
     }
   }
   // summary
   std::cout << std::endl << "-------------" << std::endl;
   if (errors > 0) {
     std::cout << "ERROR. " << errors << " of " << tests
               << " tests failed." << std::endl;
     return 1;
   }
   std::cout << "OK. " << tests << " passed." << std::endl;
   return 0;
```

```
}
```

sw1mode, sw2mode riecht nach refactoring.

Um es etwas leichter lesbar zu machen könne wir auf die Schnelle aus dem struct ein class machen und das pinMode-Setzen dort machen. Eine class ist wie ein struct, kann aber Funktionen/Methoden enthalten (und in der class ist alles private, deshalb die `public:` Angabe).

```
class MBclass {
public:
   float scale_factor;
   int sw1mode;
   int sw2mode;
   void set_pinMode()
   {
/*    pinMode(Switch1, MB[mb_index].sw1mode);
      if (MB[mb_index].sw1mode == OUTPUT) {
           digitalWrite(Switch1, 0);
      }*/
   }
};

MBclass MB[] = {
   MBclass{ 5.0 / 1024, INPUT, INPUT },
   MBclass{ 10.0 / 1024, OUTPUT, INPUT },
   MBclass{ 20.0 / 1024, OUTPUT, OUTPUT }
   };

int mb_index = 0;   // messbereichsindex
float scale_factor = MB[mb_index].scale_factor;
float measured;
void loop() {
  int adc_in = analogRead(A0);
  if (adc_in == max_adc) {
    if (mb_index >= sizeof(MB)/sizeof(MB[0])) {
     Serial.println("OVERFLOW");
    }
    else {
```

```
            mb_index++;
            MB[mb_index].set_pinMode();
            scale_factor = MB[mb_index].scale_factor;
          }
        }
```

set_pinMode tut noch nichts. Das Programm compiliert und wir bekommen die zu erwartenden Fehler wegen des nicht veränderten pinModes.

```
2.56836V
ERROR: mode1 1 != 0
5.13672V
ERROR: mode1 1 != 0
ERROR: mode1 1 != 0
10.2734V
ERROR: mode1 1 != 0

-------------
ERROR. 4 of 5 tests failed.
```

Mit :

```
void set_pinMode()
{
  pinMode(Switch1, this->sw1mode);
  if (this->sw1mode == OUTPUT) {
        digitalWrite(Switch1, 0);
  }
}
```

Funktionieren die Tests wieder. Wir erweitern um Switch2 und set_pinMode ...

OVERFLOW

Wenn wir noch ens weiter schalten mit sollte OVERFLOW ausgegeben werden.

```
int test_in[] = { 526, 1023, 526, 1023, 526, 1023 };
float test_out[] = { 2.56836, -1.0, 5.13672, -1, 10.2734, -1 };
int test_mode1[] = { INPUT, OUTPUT, OUTPUT, OUTPUT, OUTPUT, OUTPUT };
```

Tut es aber nicht.

```
2.56836V
5.13672V
10.2734V
ERROR: mode1 1 != 0
```

Warum das, wo es schon so schön lief ? Wenn wir mb_index mitausgeben

```
2.56836V
0
1
5.13672V
1
2
10.2734V
2
3
ERROR: mode1 1 != 0
```

Er zählt zu weit. Fix it.

```
if (mb_index >= (sizeof(MB)/sizeof(MB[0])-1)) {
  Serial.println("OVERFLOW");
}
```

Runtest

```
2.56836V
5.13672V
10.2734V
OVERFLOW

-------------
OK. 6 passed.
```

MESSBEREICH NACH UNTEN SCHALTEN

Fehlt uns noch. Ein Test mehr (test_in, test_out, test_mode1):

```
100, 1, OUTPUT
```

Wir bekommen:

```
OVERFLOW
1.95312V
ERROR: 1 != 1.95312
```

Die 1.95312 korrigieren. Wenn wir noch einen Test hinzufügen (test_in, test_out, test_mode1):

```
200, 1.95313, OUTPUT
```

Müsste wenn der Messbereich nach unten gewechselt wurde dasselbe herauskommen.

```
1.95312V
3.90625V
ERROR: 1 != 3.90625
```

Tut es noch nicht. Das muss irgendwie hier im “else” passieren:

```
  if (adc_in == max_adc) {
    if (mb_index >= (sizeof(MB)/sizeof(MB[0])-1)) {
      Serial.println("OVERFLOW");
    }
    else {
      mb_index++;
      MB[mb_index].set_pinMode();
      scale_factor = MB[mb_index].scale_factor;
    }
  }
  else {
// TODO maybe to lower MB
      measured = adc_in * scale_factor;
      Serial.print(measured);
      Serial.println("V");
```

```
}
```

Wechseln könne wir, wenn der gemessene Wert “measured” kleiner als der Maximalwert des darunter liegenden Messbereiches ist, den … packen wir in die class.

```
class MBclass {
public:
   float scale_factor;
   float max_measured;
   int sw1mode;
   int sw2mode;
```

Initialisieren den Wert

```
MBclass MB[] = {
   MBclass{ 5.0 / 1024, 5.0, INPUT, INPUT },
   MBclass{ 10.0 / 1024, 10.0, OUTPUT, INPUT },
   MBclass{ 20.0 / 1024, 20.0, OUTPUT, OUTPUT }
   };
```

Das riecht schon wieder (aber ich bin jetzt dann müde). Wir brauchen noch den Code

```
else {
  measured = adc_in * scale_factor;
  Serial.print(measured);
  Serial.println("V");
  if ((mb_index > 0) && (MB[mb_index-1].max_measured > measured)) {
   mb_index--;
   MB[mb_index].set_pinMode();
   scale_factor = MB[mb_index].scale_factor;
  }
}
```

Und der runtest sagt

```
OVERFLOW
1.95312V
1.95312V
1.95312V
ERROR: mode1 1 != 0
```

Weil im 20V Messbereich 1.9V einen herunter schaltet dann auf den 10V und dort die 1.9V noch einmal herunter schalten, wir sind jetzt also im 5V Bereich.

Test, wir hängen das Triple vom Anfang noch an die Testreihe:

```
int test_in[] = { 526, 1023, 526, 1023, 526, 1023, 100, 200,
526 };
float test_out[] = { 2.56836, -1.0, 5.13672, -1, 10.2734, -1,
1.95312, 1.95312, 2.568 };
int test_mode1[] = { INPUT, OUTPUT, OUTPUT, OUTPUT, OUTPUT,
OUTPUT, OUTPUT, INPUT, INPUT };
```

Runtest

```
2.56836V
5.13672V
10.2734V
OVERFLOW
1.95312V
1.95312V
2.56836V

-------------
OK. 9 passed.
```

Fertig …

Refactor Initialisierung

```
MBclass{ 5.0 / 1024, 5.0, INPUT, INPUT }
```

Der wird wohl selber dividieren können.

Bei Klassen gibt es einen Konstruktor, der beim Erstellen des Objektes aufgerufen wird.

Die sw1/2mode werden nur noch intern verwendet und müssen deshalb nicht mehr “public” sein.

```
class MBclass {
   int sw1mode;
   int sw2mode;
public:
```

```
    float scale_factor;
    float max_measured;
    MBclass (float max_measured, int sw1mode, int sw2mode)
    {
      this->max_measured = max_measured;
      this->sw1mode = sw1mode;
      this->sw2mode = sw2mode;
      this->scale_factor = max_measured / (max_adc + 1);
    }
    void set_pinMode() …
```

Die Erstellung des Arrays vereinfacht sich dadurch:

```
MBclass MB[] = {
   MBclass( 5.0, INPUT, INPUT ),
   MBclass( 10.0, OUTPUT, INPUT ),
   MBclass( 20.0, OUTPUT, OUTPUT )
   };
```

Achtung: Vorher waren da geschwungene Klammern.

```
MBclass MB[] = {
   MBclass{ 5.0 / 1024, INPUT, INPUT },
```

RESULTS

Und es kompiliert am arduino … fehlt noch der Test mit Hardware.

CONCLUSION

Feedback ?

REFERENCES

1. TDD
2. Unit testing