Ein Voltmeter für die Digitaltechnik (0-5 Volt) – Aus der Reihe „Multitool“ Teil 3

Willkommen zum dritten und letzten Teil der Reihe „Mini Multitool“ in der die Funktion des Tools um ein 2-Kanal Voltmeter mit einem Messbereich von 0-5 Volt erweitert wird und der noch eingeschränkte Komfort in der Bedienung sowie der Funktionalität des im zweiten Teil der Reihe eingeführten Rechteck Frequenzgenerators verbessert wird. Das generierte Frequenzsignal wird nun nicht mehr über einen separaten Pin ausgegeben, sondern kann bei einem Modus Wechsel mit Hilfe der “Mode-Taste“ direkt an den für den I2C Scanner genutzten Pins 4 und 5 abgegriffen werden. Dabei ist das Ausgangssignal an Pin 4 nicht invertiert und an Pin 5 invertiert. Das Potential zwischen Pin 4 und Pin 5 beträgt im Frequenzgenerator Modus somit konstant 5 Volt. Die Ausgänge der Pins 4 und 5 wurden mithilfe zweier Widerstände von je 100 Ohm in Reihe und zwei Zener-Dioden mit je 5,1 Volt Durchbruchsspannung gegen kurzzeitige und geringe Überlastung resistenter gemacht. Das Potentiometer zur Frequenz Einstellung ist zugunsten von 2 Tastern zur genaueren und präziseren Einstellung ersetzt worden. Die ausgegebene Frequenz auf der jeweilig eingestellten Stufe (1- 254) im Bereich von 122 Hz bis 15,625 KHz wird nun auf dem O-Led Display angezeigt. Die Genauigkeit der in der O-Led angezeigten Frequenz hängt dabei von dem Systemquarz ab. In dem hier vorgestellten Code wird von einer genauen Systemquarzfrequenz von 16 MHz. ausgegangen. Sollte Ihr Quarz von dieser Systemquarzfrequenz abweichen, so tragen sie bitte Ihre ermittelte Systemquarzfrequenz in Hz in der Codezeile

```
#define CrystalFreq 16000000     // Systemquarzfrequenz des Arduinos
```

ein bzw. korrigieren diese nach oben oder unten. Für die Genauigkeit des Voltmeters ist zum einen die Ermittlung der genutzten Analog-Digital Referenzspannung wichtig als auch die Genauigkeit des AD Wandlers. Da wir auf den internen AD Wandler des Arduinos nicht direkt Einfluss nehmen können, bleibt nur die Kalibration der Referenzspannung. Bitte messen Sie dazu mit einem genügend genauen Voltmeter die Spannung an dem „URef“ Pins des Arduinos gegen Masse und tragen die gemessene Spannung in die Codezeile

```
#define UReference 5.0           // Spannungsreferenz für den Analogwandler
```

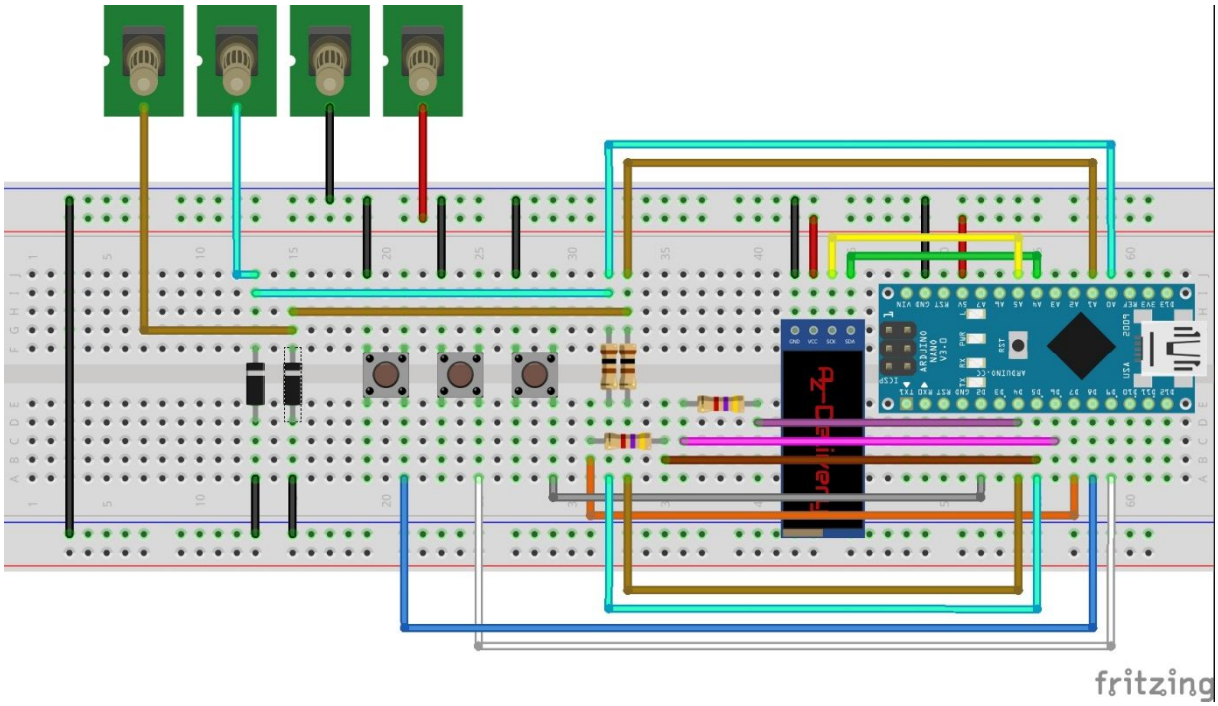
ein bzw. korrigieren diese. Im Normalfall sollte diese Spannung im Bereich von 4 bis maximal 5 Volt liegen.

Um die neuen Funktionen und Schutzmaßnahmen implementieren zu können, benötigen wir erneut eine leicht abgeänderte Hardware und damit eine veränderte Teileliste:

Anzahl	Bezeichnung	Anmerkung
1	Arduino Nano	
1	0,91 Zoll OLED I2C Display 128 x 32 Pixel	
2	4,7 KOhm Widerstände	
2	100 Ohm Widerstände	
3	Taster	
2	Zener-Dioden 5,1 Volt	

Danach kann die Hardware gemäß folgender Fritzing-Zeichnung Schaltung wie folgt erweitert werden:

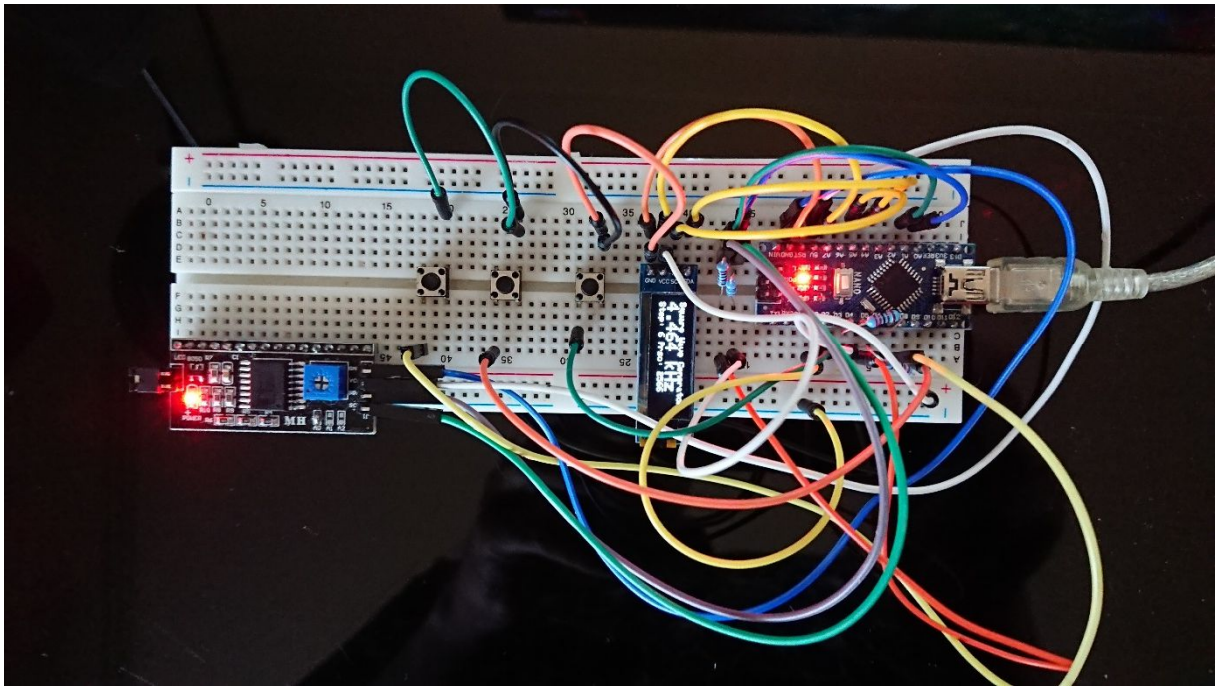
Anschluss Ports und 5 Volt Potential



An dieser Stelle möchte ich, wie versprochen, auf die im letzten Teil angesprochene geänderte Schaltung der beiden Pullup-Widerstände zu sprechen kommen. Durch den Anschluss an Ports statt an festen Potentialen wird der angeschlossene Widerstand „programmierbar“. d.h. durch gezieltes setzen der Zustände der Anschluss Ports der beiden Widerstände können diese wahlweise zu Pullup-Pulldown oder zu schaltungstechnisch neutralen Widerständen geschaltet werden. Dies nutze ich bei z.B beim Modus „I2C -Scanner“ (Teil 1) aus, indem die beiden 4,7 K Ohm Widerstände zu Pullup Widerständen werden, im Frequenzgenerator Modus (Teil 2) werden Sie dagegen deaktiviert. Im Voltmetermodus (Teil 3) werden sie zu Pulldown-Widerständen um ein definiertes Potential bei nicht Beschaltung der Port zu haben. Dies verhindert fehlerhafte Spanungsmessungen über den empfindlichen AD Wandler.

Bevor es nun an das Hochladen des Codes geht, stellen Sie bitte auch in diesem Teil sicher, dass bereits die Bibliothek „Soft Wire“ von Steve Marple in Ihrer IDE installiert ist. Die Bibliothek kann andernfalls von hier heruntergeladen werden. Weitere Informationen zu dieser Bibliothek finden sie auch auf dieser Seite.

Aufgebaut auf meinem Breadboard sieht das Multitool nun wie folgt aus:



Ein Hinweis zur Bedienung:

Bevor ein Modus umgeschaltet werden kann, ist der Prüfling oder die Last von den Ausgangspins zu trennen. Es darf niemals während eines laufenden Tests der Modus umgeschaltet werden, da es ansonsten zu Fehlfunktionen oder zur Beschädigung des Prüflings kommen kann.

Nun kann folgender Code auf den Arduino Nano geladen werden:

```
// Tobias Kuch 2020 GPL 3.0  tobias.kuch@googlemail.com
// https://github.com/kuchto

#include <Wire.h>
#include <SoftWire.h>
//#include <AsyncDelay.h>
#include "SSD1306Ascii.h"
#include "SSD1306AsciiWire.h"

#define oLed_I2C_ADDRESS 0x3C
```

```
#define VSensor0 A0
#define VSensor1 A1

#define UniIOA 4
#define UniIOB 5

#define POTAConfig 6
#define POTBConfig 7

#define UpSwitch 8              // Pin Up Taster
#define DownSwitch 9            // Pin Down Taster
#define ModeSwitch 2            // Pin Modus Taster

#define Prescaler 256           // Mögliche Werte: 1,8,64,256,1024
#define CrystalFreq 16000000    // Quarzgeschwindikeit des Arduinos: Standard: 16 mhz
#define UReference 4.5          // Spannungsreferenz für den Analogwandler (hier: 4.5 Volt)
#define ADResolution 1023
#define interval1 500           // Zeitraum in ms zwischen 2 analog Messungen der Spannung an A0
und A1

const uint8_t firstAddr = 1;
const uint8_t lastAddr = 0x7F;

SSD1306AsciiWire oled;
SoftWire sw(UniIOA, UniIOB);
SoftWire sw_rev(UniIOB, UniIOA);

//------------------------------------------------------------------------------

uint8_t I2C_Device_found_at_Address = 0;
uint16_t Round = 0;
uint16_t IntFreqStepOld = 0;

bool AlreadyScan = false;
bool Alreadypressed1 = false;
bool Alreadypressed2 = false;
bool Alreadypressed3 = false;
bool Modeswitched = false;

byte SelectedMode = 0;

unsigned int IntFreqStep = 1;

unsigned long previousMillis1 = 0;
unsigned long Frequency = 0;

float VoltageMultiplicator = 0;

ISR(TIMER1_COMPA_vect) { //Timer1 interrupt.
  // Weitere Infos auf: https://www.mikrocontroller.net/articles/AVR-Tutorial:_Timer
  digitalWrite(UniIOA, !(digitalRead(UniIOA)));
  digitalWrite(UniIOB, !(digitalRead(UniIOB)));
```

```
}

void setup()
{
  pinMode(ModeSwitch, INPUT_PULLUP);
  pinMode(UpSwitch, INPUT_PULLUP);
  pinMode(DownSwitch, INPUT_PULLUP);
  pinMode(UniIOA, OUTPUT);
  pinMode(UniIOB, OUTPUT);
  pinMode(POTAConfig, OUTPUT);
  pinMode(POTBConfig, OUTPUT);
  digitalWrite (POTAConfig, HIGH);
  digitalWrite (POTBConfig, HIGH);
  cli();                          // stoppe alle Interrupts
  TCCR1A = 0;                     // set entire TCCR1A register to 0 TCCR - Timer/Counter Control
Register
  TCCR1B = 0;                     // Setze Timer/Counter Control Register TCCR1B auf 0
  TCCR1B |= (1 << WGM12);           // Schalte Clear Timer on Compare (CTC) Modus ein
  if (Prescaler == 0)
  {
    TCCR1B |= (1 << CS10);          // Setze Prescaler auf 0.
  }
  if (Prescaler == 8)
  {
    TCCR1B |= (1 << CS11);          // Setze CS11 Bit auf 1 für den 8 Prescaler.
  }
  if (Prescaler == 64)
  {
    TCCR1B |= (1 << CS11) | (1 << CS10);    // Setze CS10 und CS11 Bit auf 1 für den 64 Prescaler.
  }
  if (Prescaler == 256)
  {
    TCCR1B |= (1 << CS12);          // Setze CS12 Bit auf 1 für den 256 Prescaler.
  }
  if (Prescaler == 1024)
  {
    TCCR1B |= (1 << CS12) | (1 << CS10);    // Setze CS10 und CS12 Bit auf 1 für den 1024 Prescaler.
  }
  TCNT1  = 0;                     // Initialisiere Zähler/Zeitgeber Register Wert auf 0
  OCR1A = 130;                      // Aufruffrequenz Timer 1  241 Hz * 2
  TIMSK1 &= (0 << OCIE1A);          // Sperre Timer Compare interrupt TIMSK - Timer/Counter
Interrupt Mask Register
  sei();                          // erlaube interrupts
  VoltageMultiplicator = float(UReference)/float(ADResolution);
  Wire.begin();
  Wire.setClock(400000L);
  sw.setTimeout_ms(200);          // Software I2C Bus Objekt 1 TimeOut
  sw_rev.setTimeout_ms(200);        // Software I2C Bus Objekt 2 TimeOut
  oled.begin(&Adafruit128x32, oLed_I2C_ADDRESS);
  oled.setFont(Adafruit5x7);
  oled.clear();
  oled.set1X();
  oled.clear();
```

```
  oled.setCursor(0, 0);
  oled.print("Scanning I2C Bus...");
  oled.setCursor(0, 1);
  oled.print("Initital Scan.");
}
//------------------------------------------------------------------------------

bool scanI2C()
{
  bool detected = false;
  for (uint8_t addr = firstAddr; addr <= lastAddr; addr++)
  {
    uint8_t startResult = sw.llStart((addr << 1) + 1); // Signal a read
    sw.stop();
    if ((startResult == 0) & (I2C_Device_found_at_Address != addr)) // New I2C Device found
    {
      Round = 0;
      detected = true;
      I2C_Device_found_at_Address = addr;
      AlreadyScan = false;
      oled.set1X();
      oled.setCursor(0, 0);
      oled.print("I2C Device found at:  ");
      oled.setCursor(0, 1);
      oled.set2X();
      oled.setCursor(0, 1);
      oled.print("-");
      oled.print(addr, BIN);
      oled.print("-b    ");
      oled.set1X();
      oled.setCursor(0, 3);
      oled.print("0x");
      if (addr < 16) oled.print("0");
      oled.print(addr, HEX);
      oled.print(" HEX  --  ");
      if (addr < 10) oled.print("0");
      oled.print(addr, DEC);
      oled.print(" DEC ");
      break;
    }  // Device Found END
  }  // Scan Round END
  if (!detected)
  {
    I2C_Device_found_at_Address = 0;
    Round++;
  }
  return detected;
} // Function END

bool scanI2C_rev()
{
  bool detected = false;
  for (uint8_t addr = firstAddr; addr <= lastAddr; addr++)
```

```
  {
   uint8_t startResult = sw_rev.llStart((addr << 1) + 1); // Signal a read
   sw_rev.stop();
   if ((startResult == 0) & (I2C_Device_found_at_Address != addr)) // New I2C Device found
   {
    Round = 0;
    detected = true;
    I2C_Device_found_at_Address = addr;
    AlreadyScan = false;
    oled.set1X();
    oled.setCursor(0, 0);
    oled.print("I2C Device found at:  ");
    oled.setCursor(0, 1);
    oled.set2X();
    oled.setCursor(0, 1);
    oled.print("-");
    oled.print(addr, BIN);
    oled.print("-b    ");
    oled.set1X();
    oled.setCursor(0, 3);
    oled.print("0x");
    if (addr < 16) oled.print("0");
    oled.print(addr, HEX);
    oled.print(" HEX  --  ");
    if (addr < 10) oled.print("0");
    oled.print(addr, DEC);
    oled.print(" DEC ");
    break;
   }  // Device Found END
  }  // Scan Round END

  if (!detected)
  {
   I2C_Device_found_at_Address = 0;
   Round++;
  }
  return detected;
} // Function END

void CheckMode()
{
  bool PinStatus1 = digitalRead(UpSwitch);
  if ((PinStatus1 == LOW) && !(Alreadypressed1))
  {
   delay(200);
   IntFreqStep = IntFreqStep + 1;
   if  (IntFreqStep > 254)
    {
    IntFreqStep = 254;
    }
   Alreadypressed1 = true;
  } else if ((PinStatus1 == HIGH) && (Alreadypressed1))
  {
```

```
   Alreadypressed1 = false;
  }

  bool PinStatus2 = digitalRead(DownSwitch);
  if ((PinStatus2 == LOW) && !(Alreadypressed2))
  {
   delay(200);
   IntFreqStep = IntFreqStep - 1;
   if  (IntFreqStep < 1)
    {
    IntFreqStep = 1;
    }
   Alreadypressed2 = true;
  } else if ((PinStatus2 == HIGH) && (Alreadypressed2))
  {
   Alreadypressed2 = false;
  }

  bool PinStatus3 = digitalRead(ModeSwitch);
  if ((PinStatus3 == LOW) && !(Alreadypressed3))
  {
   delay(200);
   SelectedMode++;
   if  (SelectedMode > 2) {
    SelectedMode = 0;
   }
   Alreadypressed3 = true;
   Modeswitched = true;
  } else if ((PinStatus3 == HIGH) && (Alreadypressed3))
  {
   Alreadypressed3 = false;
  }
}

void loop()
{
  CheckMode();
  if (SelectedMode == 0) // I2C Scanner Modus
  {
   if (Modeswitched)
   {
    cli();  //disable interrupts
    TIMSK1 &= (0 << OCIE1A); // Sperre Timer Compare interrupt TIMSK - Timer/Counter Interrupt
Mask Register
    sei();  //allow interrupts
    pinMode(POTAConfig, OUTPUT);
    pinMode(POTBConfig, OUTPUT);
    digitalWrite (POTAConfig, HIGH);
    digitalWrite (POTBConfig, HIGH);
    oled.clear();
    oled.setCursor(0, 0);
    oled.print("Scanning I2C Bus...");
   }
```

```
   Modeswitched = false;
   if ((!scanI2C()) && (Round > 2))
   {
    if (!AlreadyScan)
    {
     AlreadyScan = true;
     oled.clear();
     oled.setCursor(0, 0);
     oled.print("Scanning I2C Bus...");
    }
    Round = 0;
    I2C_Device_found_at_Address = 0;
   }
  }
  CheckMode();
  if (SelectedMode == 0) // I2C Scanner Modus
  {
   if (Modeswitched)
   {
    cli();  //disable interrupts
    TIMSK1 &= (0 << OCIE1A); // Sperre Timer Compare interrupt TIMSK - Timer/Counter Interrupt
Mask Register
    TCNT1  = 0;            // Lösche Counter Value
    sei();  //allow interrupts
    pinMode(POTAConfig, OUTPUT);
    pinMode(POTBConfig, OUTPUT);
    digitalWrite (POTAConfig, HIGH);
    digitalWrite (POTBConfig, HIGH);
    oled.clear();
    oled.setCursor(0, 0);
    oled.print("Scanning I2C Bus...");
   }
   Modeswitched = false;
   if ((!scanI2C_rev()) && (Round > 2))
   {
    if (!AlreadyScan)
    {
     AlreadyScan = true;
     oled.clear();
     oled.setCursor(0, 0);
     oled.print("Scanning I2C Bus...");
    }
    Round = 0;
    I2C_Device_found_at_Address = 0;
   }
  }
  CheckMode();
  if (SelectedMode == 1) // Square Wave Generator
  {
   if (Modeswitched)
   {

    digitalWrite(POTAConfig, LOW);
```

```
    digitalWrite(POTBConfig, LOW);
    pinMode(POTAConfig, INPUT);
    pinMode(POTBConfig, INPUT);
    pinMode(UniIOA, OUTPUT);
    pinMode(UniIOB, OUTPUT);
    digitalWrite(UniIOA, LOW);
    digitalWrite(UniIOB, HIGH);
    cli();  //disable interrupts
    TCNT1  = 0;            // Lösche Counter Value
    TIMSK1 |= (1 << OCIE1A); // Erlaube Timer compare interrupt TIMSK - Timer/Counter Interrupt
Mask Register
    sei();  //allow interrupts
    oled.clear();
    oled.setCursor(0, 0);
    oled.print("Square Wave Generator");
    oled.set2X();
    oled.setCursor(0, 1);
    float Result = float(Frequency) / 1000;
    if (Result > 1)
    {
    oled.print(Result,3);
    oled.print(" kHz ");
    } else
    {
    oled.print(float(Frequency),2);
    oled.print(" Hz   ");
    }
    oled.set1X();
    oled.setCursor(0, 3);
    oled.print("Step: ");
    oled.print(IntFreqStep);
    oled.print(" Prsc: ");
    oled.println(Prescaler);
   }
   Modeswitched = false;

   if (IntFreqStep != IntFreqStepOld)
   {
    IntFreqStepOld = IntFreqStep;
    cli();//stop interrupts
    OCR1A = IntFreqStep;
    if ( TCNT1 >= OCR1A )
    {
      TCNT1 = OCR1A - 1; //initialize counter value to 0
    }
    sei();//allow interrupts
    Frequency = (CrystalFreq / Prescaler) / (IntFreqStep + 1) / 2;
    oled.set2X();
    oled.setCursor(0, 1);
    double Result = float(Frequency) / 1000;
    Serial.println(Result);
    if (Result > 1)
    {
```

```
    oled.print(Result,3);
    oled.print(" kHz ");
    } else
    {
    oled.print(Frequency);
    oled.print(" Hz   ");
    }
    oled.set1X();
    oled.setCursor(0, 3);
    oled.print("Step: ");
    oled.print(IntFreqStep);
    oled.print(" Prsc: ");
    oled.println(Prescaler);
   // oled.print(" ");
   }
 }
 CheckMode();
 if (SelectedMode == 2) // 2 Channel Voltmeter 0- 5 Volt 2 stellen genau. Vref ist mit 5 Volt
angegeben.
 {
  if (Modeswitched)
  {
   cli();  //disable interrupts
   TIMSK1 &= (0 << OCIE1A); // Verbiete Timer compare interrupt TIMSK - Timer/Counter
Interrupt Mask Register
    TCNT1  = 0;            // Lösche Counter Value
    sei();  //allow interrupts
    digitalWrite(POTAConfig, LOW);
    digitalWrite(POTBConfig, LOW);
    pinMode(POTAConfig, OUTPUT);
    pinMode(POTBConfig, OUTPUT);
    digitalWrite(UniIOB, LOW);
    digitalWrite(UniIOA, LOW);
    pinMode(UniIOB, INPUT);
    pinMode(UniIOA, INPUT);
    oled.clear();
    oled.setCursor(0, 0);
    oled.print("2 Ch. Voltage Sensor");
    oled.setCursor(0, 1);
    oled.print("Channel 1  Channel 2 ");
  }
  Modeswitched = false;
  if (millis() - previousMillis1 > interval1)
   {
   previousMillis1 = millis();   // aktuelle Zeit abspeichern
   oled.set2X();
   int VS0 = analogRead(VSensor0);  // read the input pin
   int VS1 = analogRead(VSensor1);  // read the input pin
   float Voltage1 = float(VS0) * VoltageMultiplicator;
   float Voltage2 = float(VS1) * VoltageMultiplicator;
   oled.setCursor(0, 2);
   oled.print(Voltage1,2);
   oled.print("V");
```

```
    oled.setCursor(66, 2);
    oled.print(Voltage2,2);
    oled.print("V");
    oled.set1X();
    }
 }
  // END MainLoop
}
```

Das Tool lässt sich durch den flexiblen Hardwareaufbau noch beliebig um weitere Funktionen erweitern, beispielsweise um einen Logiktester, der im TTL Bereich von 0-5 V den logischen Zustand der Ports 4 und 5auf dem Display anzeigt. Oder z.B. auch um eine sendende serielle Schnittstelle, die ständig Zufallszeichen mit einer variablen Baudrate auf den Pins sendet.

Dank des speichersparenden und optimierten Codes verwendet der gesamte Sketch nur 11818 Bytes (38%) des Programmspeicherplatzes. Das Maximum des Arduino Nanos ist 30720 Bytes. Der globalen Variablen verwenden 757 Bytes (36%) des dynamischen Speichers, 1291 Bytes für lokale Variablen verbleiben für Ihre Erweiterungen.

Alle Projekte und Informationen meiner Blogs finden Sie, wie immer, auch unter https://github.com/kuchto.

Ich wünsche Ihnen viel Spaß beim Bauen und beim Verwenden des kleinen Tools.

Tobias Kuch