Übung Nr. 2

Jahrgang: BHME20
Gruppe: 3D

Protokollabgabe
Solldatum: 09.11.2023
Istdatum:
Note:

PROTOKOLL

Thema: Übung 9 & 10:

Tag: 19.10.2023
Zeit: 10:45-13:15 Uhr
Ort: HTBLA Kaindorf | PRR Labor
Anwesend: Traußnigg Jan
Abwesend: Ursnik Iwana
Schriftführer*in: Traußnigg Jan
Betreuer: Dipl.-Ing. Steiner Walter

Aufgabenstellung

Programmieren der Übung 9) Häufig verwendete Funktionen in das Modul „UTIL“ speichern. Erledigen der Übung 10) 1) eine Stoppuhr programmieren, welche durch Buttons gestartet/gestoppt etc. wird und den Zählstand auf den LEDS/7Seg-Anzeige ausgibt.

Resümee

Aus der heutigen Einheit können wir mitnehmen, wie man auch bei der Programmierung von Mikrocontrollern Code übersichtlich machen kann, indem man den Code auf einzelne Funktionen unterteilt und diese in der .h /.cpp Datei speichert. Auch konnten wir wieder einmal unser Verständnis für Abläufe, die simultan ablaufen müssen, erweitern. Dort ist es sehr wichtig, die Funktion millis () statt delay() zu verwenden. Möglich wäre es auch mit einer ISR, jedoch fehlte für diese Lösung leider die Zeit.

Unterschriften

Iwana Ursnik

Jan Traußnigg

Inhaltsverzeichnis

1 Zeitplan

10:45 – 11:00 Besprechung des letzten Protokolls & der neuen Aufgabenstellung
11:00 – 11:40 Erledigen der Aufgabe 9)
11:40 – 13:15 Durchdenken & programmieren der Aufgabe 10) 1)

2 Thema

2.1 Aufgabenstellung

Übung 9

1) Erstellen Sie aus den selbst programmierten Funktionen ein Modul "UTIL", bestehend aus UTIL.CPP und UTIL.H. In UTIL.H stehen nur die Funktionsprototypen und in UTIL.CPP die vollständigen Funktionen. Für die Verwendung des Moduls sind beide Dateien ins aktuelle Verzeichnis zu kopieren und UTIL.H ist zu inkludieren.

Übung 10

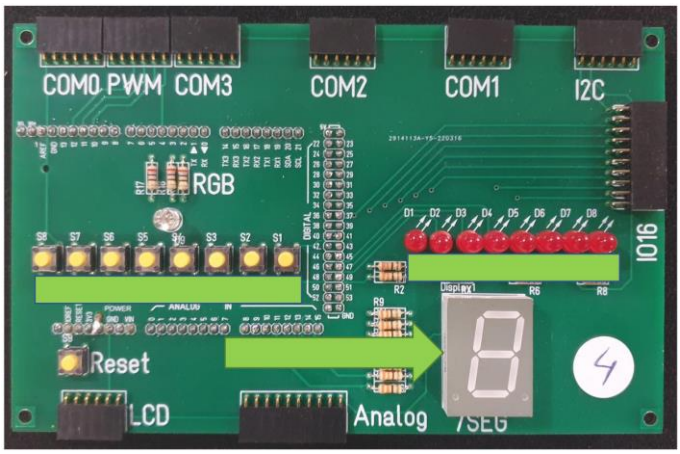

Abbildung 1: Angabe Arduino MEGA Shield. Die benötigten Pin-Nummer für LEDS und Buttons können hier abgelesen werden.

Programmieren Sie eine Stoppuhr mit folgenden Funktionen:

1) Die Auflösung der Stoppuhr ist 1/100 Sekunde Die 1/10 Sekunden werden auf der 7 Segmentanzeige und die Sekunden auf den Leds (binär oder BCD codiert) ausgegeben. Tasten:

START	**KEY0**	Starten der Stoppuhr
STOPP	**KEY1**	Stoppen der Stoppuhr
RESET	**KEY2**	Stoppen der Stoppuhr und zurücksetzten
ZZSKI	**KEY3**	Die Anzeige wird für 2 Sekunden angehalten, die Stoppuhr läuft weiter
ZZSTD	**KEY4**	Die Anzeige wird bis zum nächsten Drücken der Taste angehalten, die Stoppuhr läuft im Hintergrund weiter

Verwenden Sie das Modul UTIL für die Lösung der Probleme.
Die Genauigkeit der Stoppuhr kann durch Verwendung von millis() statt delay() erhöht werden.

2.2 Verwendete Geräte und Hilfsmittel

- Visual Studio Code (mit Plugin PlatformIO)
- Arduino IDE (privat-Laptop oder Schul-PC)
- Arduino UNO

2.3 Vorgangsweise

Feedback zum letzten Protokoll:

Die Aufgabenstellung genau lesen: die Funktion void setSeg7(uint8_t nr) soll nur die Zahl auf der 7 Segmentanzeige ausgeben, da gehört kein delay() oder sonstiges rein. Nur dass die Zahl ausgegeben wird.

Aufgabe 9)

Definieren der Funktionsprototypen & Pin-Nummern der 7-Segment Anzeige, LEDs und Buttons.

File: UTIL.h

```
#ifndef _UTIL_H
#define _UTIL_H
#include <stdint.h>
#include <Arduino.h>

const int segmentPins[] = {62, 63, 64, 65, 66, 67, 68, 69};
const int buttonPins[] = {9, 8, 7, 6, 38, 39, 40, 41};
const int ledPins[] = {49, 48, 47, 46, 45, 44, 43, 42};
const int numbersSeg7[] = {
  B11000000,  // 0
  B11111001,  // 1
  B10100100,  // 2
  B10110000,  // 3
  B10011001,  // 4
  B10010010,  // 5
  B10000010,  // 6
  B11111000,  // 7
  B10000000,  // 8
  B10010000   // 9
};
const int numSegments = sizeof(segmentPins) / sizeof(segmentPins[0]);
const int numButtons = sizeof(buttonPins) / sizeof(buttonPins[0]);
const int numLeds = sizeof(ledPins) / sizeof(ledPins[0]);

void initLed(void);
void initSeg7(void);
void initButtons(void);
void initAll(void);
bool kbhit(void);
uint8_t getch(void);
void clearLed(void);
void clearSeg7(void);
void setSeg7(uint8_t nr);
int binaryToBcd(int binaryValue);

#endif
```

Ausschreiben der vollständigen Funktionen

File: UTIL.cpp

```
#include "UTIL.h" //Path: src/UTIL.h

void initSeg7(void)
{
  for (int i = 0; i < numSegments; i++)
  {
    pinMode(segmentPins[i], OUTPUT);
  }
}

void initLed(void)
{
  for (int i = 0; i < numLeds; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
}

void initButtons(void)
{
  for (int i = 0; i < numButtons; i++)
  {
    pinMode(buttonPins[i], INPUT_PULLUP);
  }
}

void initAll(void)
{
  initLed();
  initSeg7();
  initButtons();
}

bool kbhit(void)
{
  for (int i = 0; i < 8; i++)
  {
    if (digitalRead(buttonPins[i]) == LOW)
    return true;
  }
  return false;
}

uint8_t getch(void)
{
  while (!kbhit())
    ;
```

```
  for (int i = 0; i < 8; i++)
  {
    if (!digitalRead(buttonPins[i]))
    {
      return i;
    }
  }
  return 0;
}

void clearLed(void)
{
  for (int i = 0; i < numLeds; i++)
  {
    digitalWrite(ledPins[i], LOW);
  }
}

void clearSeg7(void)
{
  for (int i = 0; i < numSegments; i++)
  {
    digitalWrite(segmentPins[i], LOW);
  }
}

void setSeg7(uint8_t nr)
{
  for (int j = 0; j < 8; j++) {
    digitalWrite(segmentPins[j], bitRead(numbersSeg7[nr], j));
  }
}

int binaryToBcd(int binaryValue)
{
  if(binaryValue < 0 || binaryValue > 9999)
  {
    return -1;
  }
  int bcdValue = 0;
  int multiplier = 1;
  while(binaryValue > 0)
  {
    bcdValue += (binaryValue % 10) * multiplier;
    binaryValue /= 10;
    multiplier *= 16;
  }

  return bcdValue;
}
```

Aufgabe 10) 1) Programmieren der main.cpp

main.cpp

Schritt 1) Inkludieren des UTIL-Moduls

```
#include "UTIL.h"
#include <stdint.h>
```

Schritt 2) Definieren der benötigten Variablen

```
const unsigned long TIMER_HIDE_INTERVAL = 2000;
const unsigned long SEC_INTERVAL = 1000;
const unsigned long MIL_INTERVAL = 100;

unsigned long timerStart;
unsigned long timerCurrent;
unsigned long timerHideStart;

unsigned long secCurrentTime;
unsigned long secCounter;
unsigned long secRemainder;

unsigned long milCurrentTime;
unsigned long milCounter;

bool secCounterFlag;
bool showCounterFlag;
bool timerHideFlag;
```

Schritt 3) Definieren der Funktionsprototypen

```
void stopCounting();
void updateDisplay();
void handleInput();
void handleCounter();
void resetTimer();
```

Schritt 4) Ausprogrammieren der Funktionen

```
void stopCounting(void)
{
  secCounterFlag = false;
  timerHideFlag = false;
  updateDisplay();
}
```

```
void handleInput()
{
  if(kbhit())
  {
    switch(getch())
    {
      case 0:
        resetTimer();
        secCounterFlag = true;
        showCounterFlag = true;
        break;
      case 1:
        showCounterFlag = true;
        stopCounting();
        break;
      case 2:
        showCounterFlag = false;
        stopCounting();
        resetTimer();
        break;
      case 3:
        if(secCounterFlag)
        {
          showCounterFlag = false;
          timerHideStart = millis();
          timerHideFlag = true;
        }
        break;
      case 4:
        if(secCounterFlag)
        {
          showCounterFlag = !showCounterFlag;
        }
        break;
      default:
        stopCounting();
        break;
    }
  }
}
```

```
void updateDisplay()
{
  if (showCounterFlag)
  {
    int bcdValue = binaryToBcd(secCounter);
    int reversedBcdValue = 0;
    for (int i = 0; i < numLeds; i++)
    {
      bitWrite(reversedBcdValue, i, bitRead(bcdValue, numLeds - 1 - i));
    }
    for (int i = 0; i < numLeds; i++)
    {
      digitalWrite(ledPins[i], bitRead(reversedBcdValue, i));
    }
    setSeg7(milCounter % 10);
  }
}

void handleCounter()
{
  timerCurrent = millis() - timerStart;
  updateDisplay();
  if (timerHideFlag && (millis() >= (timerHideStart + TIMER_HIDE_INTERVAL)))
  {
    timerHideFlag = false;
    showCounterFlag = true;
  }
  if (timerCurrent >= secCurrentTime)
  {
    secCurrentTime += SEC_INTERVAL;
    secCounter++;
    secRemainder = secCounter;
  }
  if (timerCurrent >= milCurrentTime)
  {
    milCurrentTime += MIL_INTERVAL;
    milCounter++;
  }
}
```

```
void resetTimer()
{
  timerStart = millis();
  timerCurrent = 0;
  timerHideStart = 0;

  secCurrentTime = SEC_INTERVAL;
  secCounter = 0;
  secRemainder = 0;

  milCurrentTime = MIL_INTERVAL;
  milCounter = 0;

  secCounterFlag = false;
  showCounterFlag = false;
  timerHideFlag = false;

  clearLed();
  clearSeg7();
}
```

Schritt 5) loop() und setup() schreiben

```
void setup()
{
  initAll();
  resetTimer();
}

void loop()
{
  handleInput();
  if (secCounterFlag && (secCounter < 256))
  {
    handleCounter();
  }
}
```

3 Messergebnisse

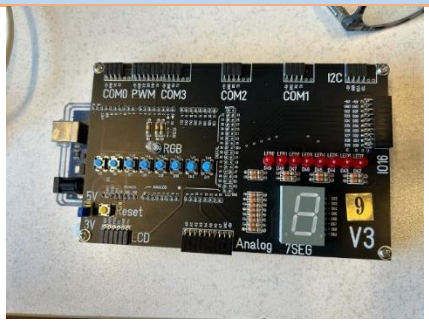

Abbildung 2: Arduino Mega Shield
(Foto von Kollegin Anna-Aurora Schreiner)

Anmerkungen: Da die Zeit leider nicht reichte, muss das Programm in der nächsten Einheit auf Funktionalität geprüft werden. Bei der Simulation in WokWi hat es leider nicht zuverlässig funktioniert, was aber nicht bedeutet dass beim echten Aufbau nicht funktionieren wird.