Übung Nr. 4

Jahrgang: BHME20
Gruppe: 3D

Protokollabgabe
Solldatum: 23.11.2023
Istdatum:
Note:

PROTOKOLL

Thema: Fertigstellung der Aufgabe 10

Tag: 16.11.2023
Zeit: 10:45-13:15 Uhr
Ort: HTBLA Kaindorf | PRR Labor
Anwesend: Traußnigg Jan, Ursnik Iwana
Abwesend:
Schriftführer*in: Traußnigg Jan
Betreuer: Dipl.-Ing. Steiner Walter

Aufgabenstellung

Die Fehler, welche im letzten Protokoll dokumentiert wurden sollen behoben werden, wofür eine genauere Problemdiagnose nötig ist. Nach Behebung des Fehlers soll dann die Aufgabe 10) b) programmiert werden, bei welcher die Zeit auf dem LCD-Bildschirm ausgegeben wird.

Resümee

Durch die heutige Fehleranalyse unseres Programes, konnten wir tieferes Verständnis für die Wichtigkeit der Laufzeit und die getch() Funktion gewinnen. Zudem haben wir eine intuitive Lösung kennengelernt, einen Flankenanstieg Software mäßig simpel zu erkennen. Bei der weiteren Programmierung der Aufgabe 10) b) konnten wir den LCD-Bildschirm kennenlernt und wie man mit Modulen von anderen Autoren umgeht und diese verstehen lernt.

Unterschriften

Iwana Ursnik

Jan Traußnigg

Inhaltsverzeichnis

1 Zeitplan

10:45 – 12:00 Mit Seriellem Monitor Rückgabewerte der getch() Funktion ausgegeben (getestet)
12:00 – 12:50 Die gefundene Schwachstelle im Code behoben
12:50 – 13:15 Das auf Elearn verfügbare Modul für die LCD Ausgabe in unser Programm implementiert

2 Thema

2.1 Aufgabenstellung

- Beheben der Fehler, welche im letzten Protokoll dokumentiert wurden.
- Programmieren der Aufgabe 10) b)

Erweitern Sie die Stoppuhr so, dass bei ZZSKI die Zeit (mm:ss.hh) am LCD ausgegeben wird.

2.2 Verwendete Geräte und Hilfsmittel

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

2.3 Vorgangsweise

Fehlersuche mithilfe des seriellen Monitors

Fehler: Nach Testung des Programmes auf dem Arduino, konnten folgende Kenntnisse gezogen werden:

- Bei mehrmaligem Betätigen der „Anzeige aus, bis Button wieder gedrückt wird“-Taste wird der Counter rückgesetzt.
- Bei langem Halten der „Anzeige aus, bis Button wieder gedrückt wird“, wird die Anzeige nicht gestoppt. Dies muss genauer behandelt werden.

Die Vermutung der Fehlerursache der Rücksetzung des Counters ist, dass die getch() Funktion teilweise einen fehlerhaften Rückgabe-Wert geben könnte.

Um das zu überprüfen, verwenden wir den seriellen Monitor zur Ausgabe der Rückgabewerte der getch() Funktion:

Schritt 1: Seriellen Monitor initialisieren

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

Schritt 2: Rückgabewert am seriellen Monitor ausgeben

```
int returnValue;
int lineCount;

void handleInput()
{
  if(kbhit())
  {
    switch(returnvalue = getch())
    {
      case 0:
        ...
      case 1:
        ...
      case 2:
        ...
      case 3:
        ...
      case 4:
        ...
      default:
        stopCounting();
        break;
    }
  }
  Serial.print(returnValue);
  Serial.print(" ");
  lineCount++;
  if(lineCount>20)
  {
    lineCount = 0;
    Serial.println();
  }
}
```

Um eine übersichtlichere Ausgabe zu erreichen, wurden die Rückgabewerte nebeneinander geschrieben, nach 20 Zeilen wurde dann ein Zeilenumbruch gemacht (deswegen lineCount)

Erkenntnisgewinn nach der Fehlersuche

Nach der Fehlersuche konnte festgestellt werden, dass bei betätigen eines Tasters die getch() Funktion so schnell arbeitet, dass sie auch bei kurzer Betätigung ca. 50 Rückgabewerte liefert.

Problembehebung der Fehler

Um diesen Fehler zu beheben, soll nur bei Flankenanstieg die switch-Case Funktion behandelt werden.

Dazu wird eine flag eingeführt. Diese ist standartmäßig true und soll nach erstem Behandeln eines cases auf false gesetzt werden. Jetzt machen wir uns die if(kbhit()) Abfrage zu nützen. Diese wird währen man die Taste gedrückt hält behandelt, jedoch wird durch ein if(firsthit) das switch case nur beim ersten Mal behandelt. Wenn jetzt if(kbhit()) nicht behandelt wird, weil die Taste nicht mehr gedrückt wird, soll in einem else {} die flag wieder auf true gesetzt werden. Ausprogrammiert sieht das ganze dann folgendermaßen aus:

Schritt 1: Erstellen der flag firstHit

```
bool firstHit = true;
```

Schritt 2: Überarbeiten der handleinput() Funktion

```
void handleInput()
{
  if(kbhit())
  {
    if(firstHit)
    {
      firstHit=false;
      switch(getch())
      {
        case 0:
          ...
        case 1:
          ...
        case 2:
          ...
        case 3:
          ...
          break;
        case 4:
          ...
        default:
          stopCounting();
          break;
      }
    }
    Serial.print(returnValue);
    Serial.print(" ");
    lineCount++;
    if(lineCount>20)
    {
      lineCount = 0;
      Serial.println();
    }
  }
  else
  {
    firstHit=true;
  }
}
```

Anzeige der Uhrzeit auf dem LCD-Bildschirm

Um den LCD-Bildschirm zu verwenden, ist das bereits zur Verfügung gestellte Modul von Elearn zu verwenden. Um dieses zu verwenden, müssen die .h und .cpp Dateien als erstes in den src-Ordner des Projekt-Ordners der Aufgabe 10 kopiert werden. Dann muss in der main.cpp die lcd12864.h Datei inkludiert werden.

Nach der Initialisierung kann dann dem Bildschirm dann ein String gesendet werden. Dieser soll im Format (mm:ss.hh) am LCD ausgegeben wird. Dazu muss eine Funktion geschrieben werden, die den String in dieses Format formatiert

Schritt 1: Modul inkludieren & Initialisierung

```
#include "lcd12864.h"

void setup()
{
  ...
  initLcd12864();
}
```

Schritt 2: Funktion zur Formatierung des Strings programmieren

```
char* formatTime(unsigned long secCounter, unsigned long milCounter)
{
  unsigned long totalSeconds = secCounter;
  unsigned long minutes = totalSeconds / 60;
  unsigned long seconds = totalSeconds % 60;
  unsigned long hundredths = milCounter%100;

  static char returnString[9]; // Format: "mm:ss.hh\0"
  sprintf(returnString, "%02lu:%02lu.%02lu", minutes, seconds, hundredths);

  return returnString;
}
```

Schritt 3: String erstellen, mit Funktion formatieren und ausgeben

```
void updateDisplay()
{
  timeStringLCD = formatTime(secCounter, milCounter);
  blankLcd12864();
  printLcd12864(0, 0, timeStringLCD);
  ...
}
```

Endgültiger Code

```
#include "UTIL.h"
#include <stdint.h>
#include "lcd12864.h"

const unsigned long TIMER_HIDE_INTERVAL = 2000;
const unsigned long SEC_INTERVAL = 1000;

unsigned long timerStart;
unsigned long timerHideStart;

unsigned long secCounter;
unsigned long milCounter;

int returnValue;
int lineCount;

char* timeStringLCD;

bool secCounterFlag;
bool showCounterFlag;
bool timerHideFlag;
bool firstHit = true;

  void stopCounting();
  void updateDisplay();
  void handleInput();
  void handleCounter();
  void resetTimer();
  char* formatTime(unsigned long secCounter, unsigned long milCounter);

void setup()
{
  initAll();
  resetTimer();
  //Serial.begin(9600);
  initLcd12864();
}

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

```
void handleInput()
{
  if(kbhit())
  {
    if(firstHit)
    {
      firstHit=false;
      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:
          showCounterFlag = !showCounterFlag;
          break;
        default:
          stopCounting();
          break;
      }
    }
    // Serial Monitor input
    /*
    Serial.print(returnValue);
    Serial.print(" ");
    lineCount++;
    if(lineCount>20)
    {
      lineCount = 0;
      Serial.println();
    }
  */
```

```
  }
  else
  {
    firstHit=true;
  }
}

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

void updateDisplay()
{
  timeStringLCD = formatTime(secCounter, milCounter);
  //Serial.println(timeStringLCD);
  blankLcd12864();
  printLcd12864(0, 0, timeStringLCD);
  if (showCounterFlag)
  {
    for (int i = 0; i < numLeds; i++)
    {
      digitalWrite(ledPins[i], bitRead(secCounter, i));
    }
    setSeg7((milCounter / 100)%10);
  }
}

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

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

  secCounter = 0;
  milCounter = 0;

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

  clearLed();
  clearSeg7();
  blankLcd12864();
}

char* formatTime(unsigned long secCounter, unsigned long milCounter)
{
  unsigned long totalSeconds = secCounter;
  unsigned long minutes = totalSeconds / 60;
  unsigned long seconds = totalSeconds % 60;
  unsigned long hundredths = milCounter%100;

  static char returnString[9]; // Format: "mm:ss.hh\0"
  sprintf(returnString, "%02lu:%02lu.%02lu", minutes, seconds, hundredths);

  return returnString;
}

/*
//updateDisplay() mit BCD counter
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);
  }
} */
```

3 Messergebnisse

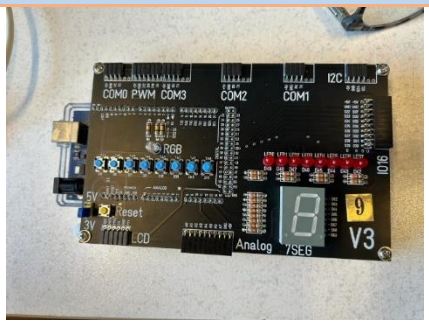

Abbildung 1: Arduino Mega Shield
(Foto von Kollegin Anna-Aurora Schreiner)

Dieser überarbeitete Code muss in der nächsten Einheit noch getestet werden. Der Ausgang am LCD Bildschirm wurde bisher nur programmiert und nicht überprüft, wobei der Code für den String selber bereits auf WokWi mit dem seriellen Monitor getestet wurde.