Hochschule Hannover
University of Applied Sciences and Arts
-
Fakultät I
Elektro- und Informationstechnik

Projekt Bericht

Zeitabhängige Steuerung von Mikrocontrollern

Jan Wille

30.03.2020

Selbstständigkeitserklärung

Hiermit bestätige ich, dass die folgende Arbeit eigenständig von mir allein erstellt und unter Berücksichtigung der zur Verfügung gestellten Aufgabenstellung sowie dem Arbeitsmaterial unter Angabe aller verwendeten Quellen erarbeitet wurde. Die Regelungen und Konsequenzen eines Plagiats, inklusive disziplinarischer Maßnahmen, sind mir bewusst. Insbesondere wurden alle Zitate und gedanklichen Übernahmen als solche kenntlich gemacht.

Jan Wille

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Aufgabenstellung des Professors 1
1.2 erste Idee 2

2 Problem mit zeitabhängigen Abläufen **3**
2.1 Prototypenbau 4

3 Realisierung von Interrupts **5**
3.1 Prototypenbau 6

4 Zeitabhängige Interrupts **7**
4.1 Prototypenbau 8

5 Multiplexing mehrerer 7-Segment Anzeigen. **9**
5.1 Prototypenbau 10

6 Weiterführende Programmieraufgabe **11**

7 Anlagen **12**
7.1 Codebeispiel für Problem mit zeitabhängigen Abläufen 12
7.2 Codebeispiel für Realisierung von Interrupts 12
7.3 Codebeispiel für Zeitabhängige Interrupts 13
7.4 Codebeispiel für Aufgabe 4 13

Literatur **15**

Abbildungsverzeichnis **15**

Tabellenverzeichnis **15**

1 Einleitung

Aufgabe dieses Projektes soll es sein, einen Teilversuche des von Professor Steigenberger und Professor Patzke betreute Mikrocontroller Labor neu zu entwickeln. Parallel werden weiter Projekte durchgeführt, welche sich mit den verbleibenden Teilversuchen beschäftigen, sodass insgesamt das gesamte Labor überarbeitet und modernisiert wird.
Durch die Überarbeitung sollen die Versuche von den alten 8-Bit 8051-Mikrocontrollern auf modernere Arduino-Due Experimentierboards mit 32-Bit Prozessoren angepasst werden.
Dieser Teilversuch soll herausarbeiten, wie Zeitsteuerung und paralleles arbeiten auf einem solchen Mikrocontroller möglich ist und welche Besonderheiten und Herausforderungen dies mit sich bringt.

1.1 Aufgabenstellung des Professors

Die folgenden Rahmenbedingungen wurden von den Professoren basierend auf dem alten Versuch zur Verfügung gestellt.

Timing und Ablaufsteuerung, Inhalt:

- Timing: Realisieren von zeitlichen Bezügen
  - Hardware-Timer
    - Timer/Counter-Peripherals (Aufbau, Progr. Model)
    - Timer-Interrupts
    - Beispielhalft testen und mit Oszilloskop prüfen anhand einer geeigneten (Arduino-) Bibliothek
  - Software-Timer
    - Verwendung von micros() innerhalb loop()
    - Basis: Systick-Counter der CPU
    - Beispielhaft testen und mit Oszilloskop prüfen
- Ablaufsteuerung / Zustandsmaschinen
  - Realisierung von zwei Blinklichtern (LEDs) mit unterschiedlichen Frequenzen
    - (Verwendung SW-Timer, ohne FSM)
  - Realisierung von zwei Lauflichtern mit unterschiedlichen Frequenzen
    - SW-Zustandsmaschinen (C++-Klasse von Prof. R. Patzke)
    - Zwei Zustandsmaschinen verwenden, die einzelnen LEDs visualisieren dabei die einzelnen Zustände
- Optional:
  - PWM-Signal zur Helligkeitssteuerung einer LED

Sonstiges:

- Arduino Due (Atmel SAM3-MCU)
- IDE egal (Sloeber-Eclipse, VSCode-PlatformIO, Eclipse CDT (mit Platformio oder Sloeber PlugIn)

1.2 erste Idee

Durch ein erstes Brainstorming ergaben sich die folgenden fünf Ideen als erste Rahmenstruktur:

1. LED-Blinklicht, $0{,}5\,\mathrm{Hz}$, verwenden von delay(), Schalter zum AN/AUS schalten
2. Lauflicht mit variabler Geschwindigkeit, Hardware Interrupt gesteuert
3. Sekundenzähler auf 7-Segment Anzeige, Hardware-Timer
4. Multiplexing mehrerer 7-Segemnt Anzeigen, Software-Timer (C++-Klasse)
5. weiterführende Programmieraufgabe mit Statemachine, folgende Ideen:
   - Ampelsteuerung mit Countdown
   - Schifffahrtsschleuse mit Füllstandanzeige

2 Problem mit zeitabhängigen Abläufen

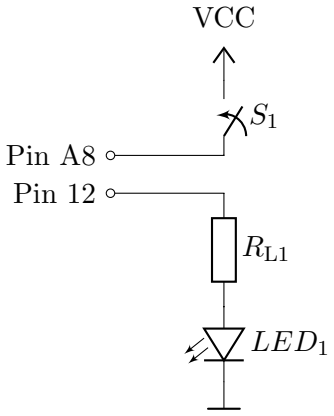


Abb. 2.1: Schaltplan für Aufgabe 1

Zu beginn soll gezeigt werden, wie es überhaupt zu Problemen durch zeitabhängige Vorgänge auf Mikrocontrollern kommen kann. Dazu wird ein Schalter S_1 und eine LED_1 über einen Arduino verschaltet (siehe Abbildung 2.1). Auf dem Arduino läuft dass im Anhang in Abschnitt 7.1 nachlesbare Programm, welches den Zustand des Schalters mittels einer IF-Abfrage testet und dann mithilfe der Arduino Library Funktion `delay()` die LED blinken lässt.
Testet man nun das Verhalten des Programmes, indem man den Schalter zu Verschiedenen Zeitpunkten im Blinkzyklus öffnet, wird deutlich, dass nur alle zwei Sekunden die IF-Abfrage tatsächlich durchgeführt wird. Öffnet man den Schalter nicht direkt zu beginn des Zyklus, wird dieser noch bis zum ende durchlaufen und erst dann schaltet der Arduino ab. Dies ist in den Impulsdiagrammen in Abbildung 2.2 noch einmal Graphisch dargestellt.

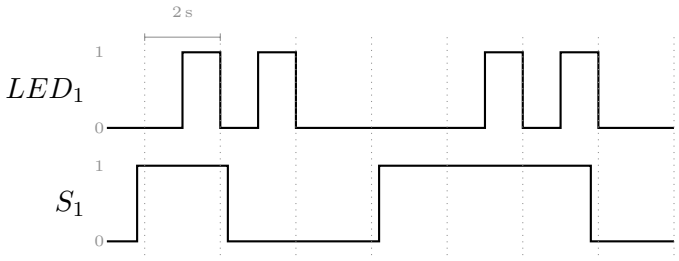


Abb. 2.2: Impulsdiagramme von S_1 und LED_1

Das Problem stellt hierbei die `delay()` Funktion dar, welche den Programmablauf blockiert und verhindert, dass andere Prozesse, wie eben das abfragen von Eingängen, durchgeführt werden.
Verschiedene Methoden, wie man diese Problem umgehen kann, soll in den folgenden Kapiteln erläutert werden.

2.1 Prototypenbau

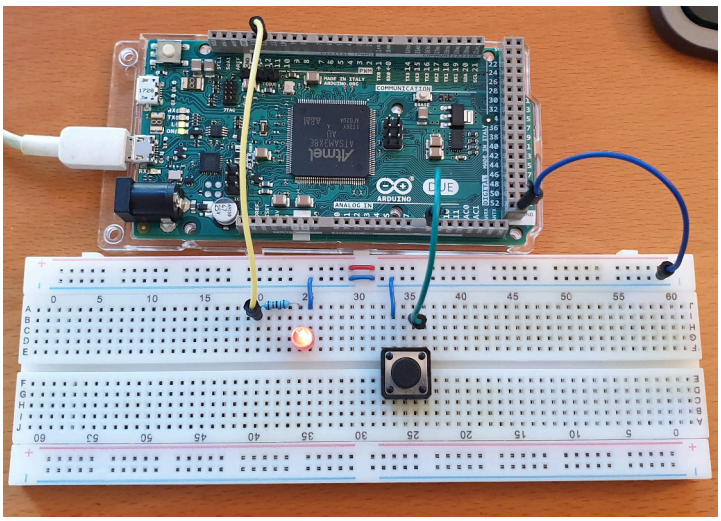


Abb. 2.3: Breadboardaufbau der LED und des Taster

Um das verhalten des Programmes zu testen, wurde der Aufbaus aus Abbildung 2.1 auf einem Breadboard umgesetzt und dieses mit dem Arduino verbunden. Da der Arduino $3{,}3\,\mathrm{V}$ an seinen Pins ausgibt, wurde für die LED ein Vorwiederstand von $100\,\Omega$ gewählt.
Ein eigens Platinenlayout wurde für diesen Versuch nicht erstellt, da der simple Aufbau problemlos als Teil folgender Aufbauten geplant werden kann.
Test mit dem Board und durchführen der geplanten Aufgabe zeigen, das die Funktion dem Konzept entspricht und die Planung fortgesetzt werden kann.

3 Realisierung von Interrupts

Eine erste Möglichkeit ohne Verzögerung auf Signale an Eingängen zu reagieren, stellt der sogenannte Interrupt dar. Wird ein Pin des Arduinos mit einem Interrupt ausgestattet, muss diesem Pin eine *Interrupt Service Routine* (kurz ISR) zugeordnet werden. Diese wird automatisch aufgerufen und abgearbeitet, wenn die Interrupt Bedingung am Pin erkannt wird. Dabei wird das Hauptprogramm pausiert und nach Beendigung der ISR normal weiter bearbeitet.
Sollen Interrupts auf einem Mikrocontroller verwendet werden, sind allerdings einige Besonderheiten zu beachten. Je nach dem welcher spezifische Mikrocontroller verwendet wir, ist zuerst einmal zu überprüfen, welche Pins überhaupt Interrupt fähig sind. Auf dem hier verwendeten Arduino Due sind das alle GpIO-Pins [Cor15]. Außerdem können Prozess bedingt keine zeitabhängigen Funktionen wie `delay()` oder `millisecend()` innerhalb einer ISR benutzt werden. Um Datenübertragung zwischen dem Hauptprogramm und der ISR zu ermöglichen, sollten Variablen global und volatile deklariert werden. [Com19]
Die Arduino Library bietet zum nutzen von Interrupts die fertige Funktion `attachInterrupt()`, welche automatisch alle nötig Einstellungen im spezifischen Arduino Vornimmt. Die Funktion benötigt die Parameter *Interrupt-Bezeichnung*, *Name der ISR* und *Modus*. Der Modus gibt dabei an, ob der Interrupt bei steigender, fallender oder jeder Flanke reagieren soll. Zusätzlich gibt es noch die Funktion `digitalPinToInterrupt(pin)`, welche eine Pinbezeichnung in die benötigte Interrupt Bezeichnung umwandelt. Weitere Informationen können der Arduino Website entnommen werden. [Com19]

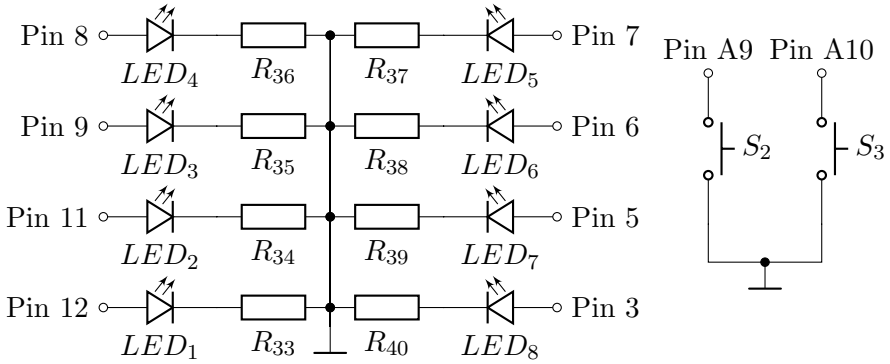


Abb. 3.1: Schaltplan für Aufgabe 2

Als Anwendungsbeispiel wird ein Lauflicht aus acht LED mit den zugehörigen Vorwiederständen aufgebaut. Auch hier werden wider $100\,\Omega$ Widerstände gewählt. Die Geschwindigkeit des Lauflichtes soll über zwei Knöpfe regelbar sein. Der Aufbau ist in Abbildung 3.1 Skizziert.
Für die Funktion werden nacheinander die einzelnen LEDs ein und wider ausgeschaltet und die Zeitverzögerung wird mittels der `delay()` Funktion gelöst. Der Wert der Zeitverzögerung wird durch eine globale Variable vorgegeben.

Zwar wird auch in diesem Fall das Hauptprogramm durch die `delay()`'s blockiert, jedoch werden Interrupts extern von diesem bearbeitet und können so jederzeit das Programm beeinflussen. Dazu wird den beiden Pins an denen die Schalter S_1 und S_2 angeschlossen sind, wären des Setups ein Interrupt zugeordnet und innerhalb der jeweiligen ISR wird die Verzögerungsvariable verändert. Ein Beispielprogramm kann in Abschnitt 7.2 nachgelesen werden.

3.1 Prototypenbau

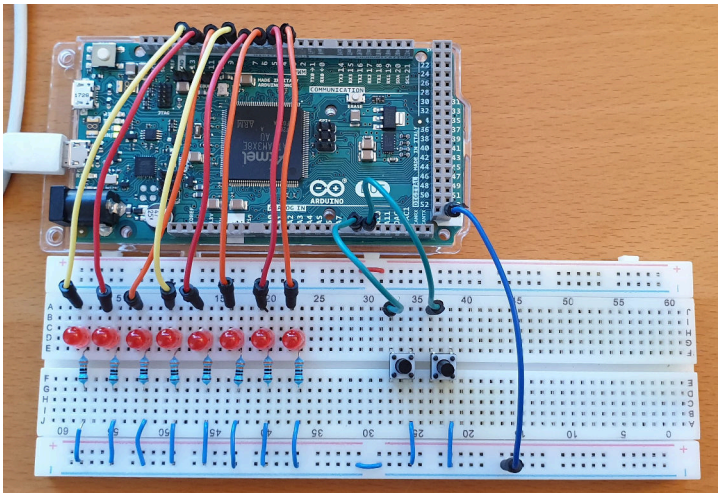


Abb. 3.2: Lauflicht und 2 Taster auf einem Breadbord

Der geplante Aufbau wird auf einem Breadboard gesteckt. Dabei hat sich herausgestellt, das es am einfachsten ist den internen Pullup-Widerstand des Arduino Dues mit der Funktion `pinModer (pin,INPUT_PULLUP)` zu aktivieren und die Schalter auf Ground zu verbinden. Dadurch wird der Verdrahtungsaufwand geringer, alledings ist zu beachten dass grundsätzlich ein HIGH Signal am Pin anliegt und das drücken des Schalters ein LOW Signal erzeugt.
Um die code Lesbarkeit zu erhöhen, wurde eine Funktion ergänzt, welche die einzelnen Bits eines Bytes auf acht Pins mapt. Dadurch kann das Lauflicht als simples schiebe Register realisiert werden. Beim testen haben sich einige Probleme ergeben. Die `delay()` Funktion hängt sich auf wenn der Eingabeparameter null erreicht. Dies kann durch eine simple IF-Abfrage verhindert werden. Außerdem ist eine lineare Änderung der Geschwindigkeit eher unpraktisch, reicht aber für Demonstrationszwecke aus.

4 Zeitabhängige Interrupts

Um das Hauptprogramm vollständig frei von Blockaden zu halten, ist es möglich, zeitabhängige Abläufe durch sogenannte *Timer Interrupts* zu lösen. Wie bereits in Kapitel 3 beschrieben, wird auch einem solchem Interrupt eine ISR zugeordnet. Diese wird automatisch in festen zeitlichen Intervallen aufgerufen und abgearbeitet.
Von Mikrocontroller zu Mikrocontroller unterscheidet sich die Verfügbarkeit solcher Timer stark und auch wie sie genau programmiert und konfiguriert werden, ist je nach Anwendungsfall im Datenblatt nachzulesen.

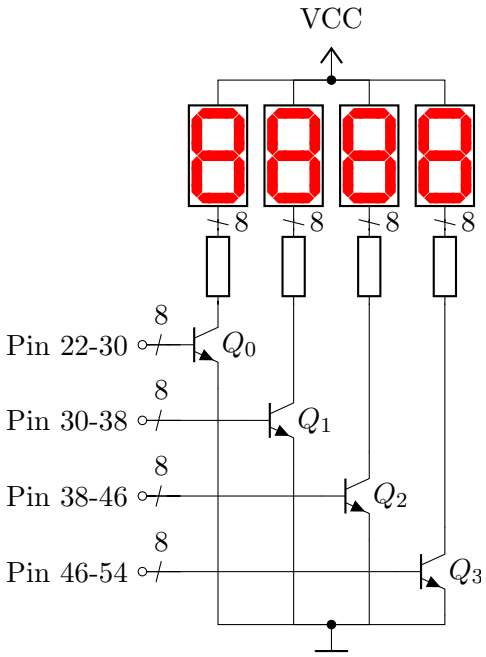


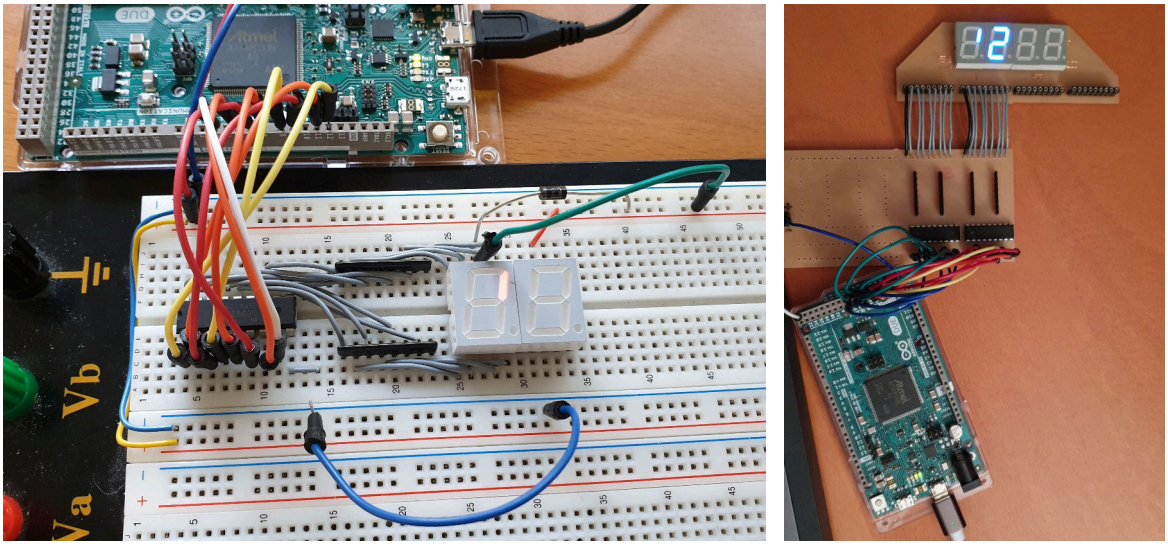
Abb. 4.1: Schaltplan für Aufgabe 3

Als Anwendungsbeispiel wird nun ein Timer auf mehreren 7-Segment Anzeigen erstellt. Dieser soll im Sekunden Takt rauf oder runter zählen können, dabei soll allerdings das Hauptprogramm für zukünftige Anwendungen frei bleiben. Die `loop()` Funktion des Arduinos muss also vollständig leer bleiben.
Um die Timer Interrupts des Arduino Dues zu konfigurieren, wird eine fertige Funktion aus einem Comunity Forum verwendet.[ss12] Diese `startTimer()` Funktion benötigt als Parameter, welcher der drei Timer verwendet werden soll, welcher der drei Channel pro Timer verwendet werden soll und mit welcher Frequenz der Timer laufen soll. Der Name der ISR des jeweiligen Timers kann der Tabelle 4.1 entnommen werden.
Wird ein solcher Timer mit der Frequenz von $1\,\mathrm{Hz}$ programmiert, wird seine ISR einmal pro Sekunde aufgerufen. Nun kann in dieser eine Variable in- oder dekrementiert werden und auf einer der 7-Segement Anzeigen dargestellt werden.
Ein Beispiel Lösungsprogramm kann in Abschnitt 7.4 nachgelesen werden.

Tab. 4.1: Zuordnung der Timer-Channel zu ISR-Namen

Timer	Channel	ISR-Name
TC0	0	TC0_Handler()
	1	TC1_Handler()
	2	TC2_Handler()
TC1	0	TC3_Handler()
	1	TC4_Handler()
	2	TC5_Handler()
TC2	0	TC6_Handler()
	1	TC7_Handler()
	2	TC8_Handler()

4.1 Prototypenbau



(a) Breadboard (b) Platine

Abb. 4.2: Aufbau und Verdrahtung von einer 7-Segemnt Anzeigen

Da die Pins des Arduino Duos nur maximal $15\,\mathrm{mA}$ und das noch nicht mal an allen Pins, liefern können, werden zusätzlich Darlington-Arrays als Treiber verbaut. Diese sorgen außerdem für eine wählbare Quellspannung. Daher können für diesen Aufbau $5\,\mathrm{V}$ verwendet werden.

5 Multiplexing mehrerer 7-Segment Anzeigen.

Da der Verdrahtungsaufwand und Pinbedarf in Kapitel 4 relativ groß war, soll dieser nun durch Verbesserung der Software reduziert werden.
Dazu wird die Technik des Multiplexings verwendet. Dabei werden alle 7-Segment Anzeigen an die Selben Pins am Arduino angeschlossen, zusätzlich werden aber noch Transistoren in den einzelnen Spannungsversorgungen verbaut, sodass sich jede Anzeige nach bedarf ein- und ausschalten lässt. Werden nun die anzeigen schnell genug aufeinanderfolgend geschaltet, ist es möglich, auf jeder Anzeige eine andere Zahl darzustellen, obwohl alle an den selben Pins angeschlossen sind.
Vergleich man die Schaltpläne in Abbildung 4.1 und Abbildung 5.1 wird der unterschied schnell deutlich. Allerdings muss die Software entsprechen angepasst werden.

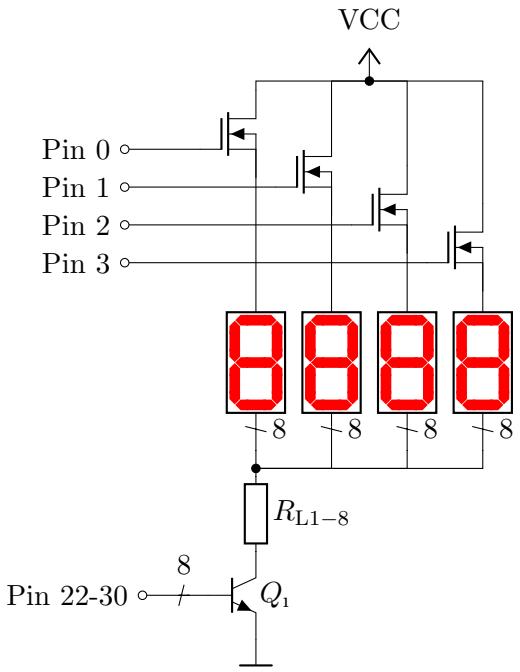


Abb. 5.1: Schaltplan für Aufgabe 4

Ziel ist es also, ein Programm zu schreiben, das kontinuierlich die Anzeigen durchschaltet und so eine beliebige vierstellige Zahl darstellen kann. Dies wäre mithilfe von Timer Interrupts zwar möglich, jedoch wird es dabei schwierig den Überblick zu behalten. Daher wird hier eine weitere Möglichkeit zur zeitabhängigen Steuerung vorgestellt, die Software Timer.
Professor R. Patzke hat eine C++ Klasse geschrieben und auf GitHub veröffentlicht [Pat18], die unter anderem eine Methode enthält, welche in festlegbaren Intervallen ein TRUE zurück gibt. Dies ermöglicht das schreiben von IF-Abfragen, die zeitliche Bedingungen haben können:

```
if (loopCheck.timerMilli(0, 1000, 0)) {
}
```

Daraus ergibt sich ein sehr leserlicher und nachvollziehbarer Code.

5.1 Prototypenbau

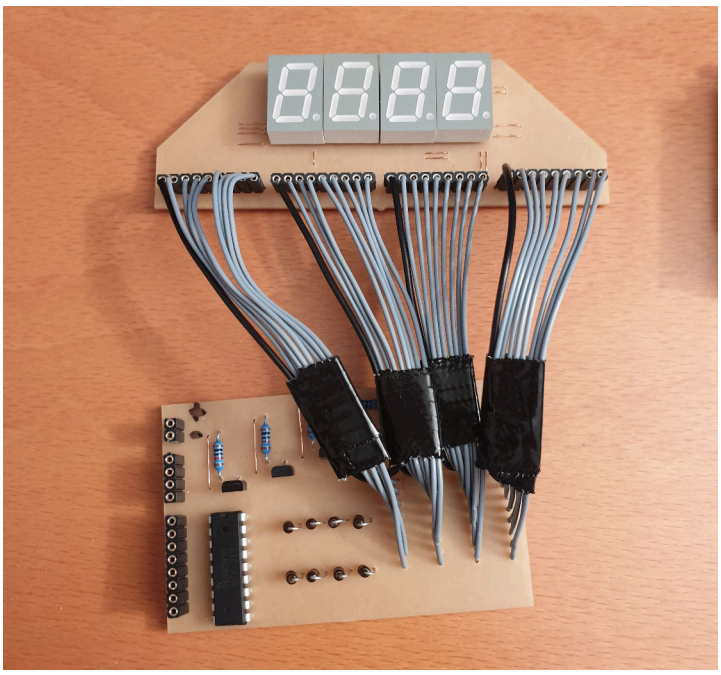


Abb. 5.2: Platinenprototyp für Multiplexing von vier Anzeigen

6 Weiterführende Programmieraufgabe

Hier sollen mehrere Vorschläge für Aufgaben gemacht werden, die mit dem Wissen der vorherigen Kapiteln gelöst werden sollen. Folgende Vorschläge sind denkbar:

1. **Ampelsteuerung**, mit zwei Richtungen, Autos und Fußgänger. Die Autoample schaltet zyklisch zwischen rot und grün, wird ein Knopf gedrückt wechselt die Ampel wenige Sekunden später auf rot, egal wie lange sie vorher grün war.

2. **Bootschleuse**, diese soll zwei Tore haben, symbolisiert durch Segmente einer Anzeige. wird an einer der Seiten ein Knopf gedrückt, wird das entsprechende Tor geöffnet, nach bestimmter Zeit wieder geschlossen und dann der Wasserstand angepasst(symbolisiert durch mehrere LEDs). Tore können nur geöffnet werden, wenn der Wasserstand auf ihrem Level ist!

Interessant ist hierfür auch die weitere Bibliothek von Robert Patske, die das anwenden von Statemachines vereinfacht.

7 Anlagen

7.1 Codebeispiel für Problem mit zeitabhängigen Abläufen

```
#include "Arduino.h"

#define SWITCH A8   //Pin number the Switch is connected to
#define LED 12      //Pin number the LED is connected to

//The setup function is called once at startup of the sketch
void setup() {
    pinMode(SWITCH, INPUT_PULLUP);
    pinMode(LED, OUTPUT);
}

// The loop function is called in an endless loop
void loop() {
    if (digitalRead(SWITCH)==1) {   //decect if Switch is ON or OFF
        delay(1000);                //blink LED wit .5Hz
        digitalWrite(LED,1);
        delay(1000);
        digitalWrite(LED,0);
    } else {
        digitalWrite(LED,0);        //turn LED Off
    }
}
```

7.2 Codebeispiel für Realisierung von Interrupts

```
/* Versuch 2 - Aufgabe 1
 * LED-Blink-Speed
 * using two button-triggered interrupts to modify blink speed
 */

#include "V2A1.h"

#define BUTTON1 A8
#define BUTTON2 A9
volatile int prct = 100;    //a value in Percent
char ledReg = 0x00;         //byte represents the 8 Outputs

//The setup function is called once at startup of the sketch
void setup() {
    setupPorts();
    pinMode(BUTTON1, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(BUTTON1),ISR_ButtonUp,FALLING
        );
    pinMode(BUTTON2, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(BUTTON2),ISR_ButtonDown,
```

```
        FALLING);
}

// The loop function is called in an endless loop
void loop() {
    if (ledReg==0x00)       //check if bit has bin shifted out
        ledReg=0x01;        //reset to start condition
    digitalWriteByte(ledReg);//display current byte on the LEDs
    ledReg<<=1;             //shift byte one digit to the left
    delay(10*prct);         //delay, percentage of 1s
}

void ISR_ButtonUp(void) {
    prct+=10;               //increase percentage
}

void ISR_ButtonDown(void) {
    prct-=10;               //decrease percentage
    if (prct<1) prct=1;     //do not fall under 1
}
```

7.3 Codebeispiel für Zeitabhängige Interrupts

```
/* Versuch 2 - Aufgabe 3
 * Timer & Interrupt
 */

// Do not remove the include below
#include "V2A3_Timer_Interrupt.h"

volatile int number=0;

void setup() {
    Serial.begin(57600);
    setupPorts();
    //TC1 channel 0 and the desired frequency
    startTimer(TC1, 0, 10);
}

void loop() {

}

//This function will be call once per second
void TC3_Handler() {
    TC_GetStatus(TC1, 0); // You must do TC_GetStatus to "accept"
       interrupt
    displayNumber(number++);
    if (number>=99) number=0;
}
```

7.4 Codebeispiel für Aufgabe 4

```
/* Versuch 2 - Aufgabe 4
 * Ablaufsteuerung mit LoopCheck (R. Patzke)
```

```
 */

#include "V2A4_Loopcheck.h"

LoopCheck loopCheck;        //instance of loopCheck
#define DISPLAY_NUMBER 4    //number of displays connected via
    multiplexing

int displayPowerPins[DISPLAY_NUMBER] = {42,43,44,45};   //pins VCC of
    the displays is connected to
int whitchDisplay = DISPLAY_NUMBER-1;                   //stores which
    of the displays is selected
int Number = 0;                                         //number to
    display

//The setup function is called once at startup of the sketch
void setup() {
    setupPorts();
    for (int i=0;i<DISPLAY_NUMBER;i++)
        pinMode(displayPowerPins[i], OUTPUT);
}

// The loop function is called in an endless loop
void loop() {
    loopCheck.begin();

    if (loopCheck.timerMilli(0, 1, 0)) {
        digitalWrite(displayPowerPins[whitchDisplay], 1);   //disable
            display of last cycle, invert because p-Channel
        whitchDisplay--;                                    //select
            next display

        displayNumber(Number/((int) pow(10,whitchDisplay)) % 10);//
            display appropriate digit
        digitalWrite(displayPowerPins[whitchDisplay], 0);   //enable
            current display, invert because p-Channel

        if (whitchDisplay<=0) whitchDisplay=DISPLAY_NUMBER-1;//if
            lowest display, revert to highest
    }

    if (loopCheck.timerMilli(1, 1000, 0)) { //increment number once per
        second
        Number++;
    }

    loopCheck.end();
}
```

Literatur

[Com19] Arduino Community. *Arduino Reference Guide - attachInterrupt()*. ENG. 4. Dez. 2019. URL: www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/ (besucht am 26. 03. 2020).

[Cor15] Atmel Corporation. *SAM3X/SAM3A Series Datasheet*. ENG. 23. März 2015. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf (besucht am 14. 05. 2020).

[Pat18] Robert Patzke. *Decentral Homeautomation*. GER. 17. Feb. 2018. URL: github.com/RobertPatzke/homeautomation (besucht am 26. 03. 2020).

[ss12] user: stimmer und user: seavik. *Arduino Forum - Timer Interrupts on Due*. ENG. 4. Nov. 2012. URL: https://forum.arduino.cc/index.php?topic=130423.0 (besucht am 26. 03. 2020).

Abbildungsverzeichnis

2.1 Schaltplan für Aufgabe 1 . . . 3
2.2 Impulsdiagramme von S_1 und LED_1 . . . 3
2.3 Breadboardaufbau der LED und des Taster . . . 4

3.1 Schaltplan für Aufgabe 2 . . . 5
3.2 Lauflicht und 2 Taster auf einem Breadbord . . . 6

4.1 Schaltplan für Aufgabe 3 . . . 7
4.2 Aufbau und Verdrahtung von einer 7-Segemnt Anzeigen . . . 8

5.1 Schaltplan für Aufgabe 4 . . . 9
5.2 Platinenprototyp für Multiplexing von vier Anzeigen . . . 10

Tabellenverzeichnis

4.1 Zuordnung der Timer-Channel zu ISR-Namen . . . 8