JOHANNES KEPLER
UNIVERSITÄT LINZ

Protokoll

Praktikum Digitale Schaltungen
LVA 336.003/19/20, SS20

Vorname	Nachname	Matrikelnummer
Michael	Rynkiewicz	k11736476

Ich bestätige hiermit, dieses Protokoll eigenhändig angefertigt zu haben. Plagiierte Protokolle werden nicht bewertet und bewirken einen negativen Abschluss des Praktikums mit entsprechenden Konsequenzen.

2020-10-08 ,
Datum


Unterschrift

Bewertung:

	A.1	A.2	A.3	A.4	A.5	A.6	A.7	Form	$\sum$
Max.:	9	9	9	9	9	9	9	7	70
Pkt.:									

Inhaltsverzeichnis

1 Einleitung

Das Wissen um die Funktionsweise eines Mikrocontrollers ist wesentlicher Bestandteil eines Informatik-Studiums. Im Zuge Dieses werden in diesem Protokoll verschiedene Experimente beschrieben und evaluiert. Insgesamt werden sieben Experimente begutachtet und diskutiert, welche im Zeitraum vom 28.September 2020 und 2.Oktober 2020 durchgeführt wurden.

2 Verwendete Materialien

Jede Sektion gibt die in ihr verwendeten Materialien mitsamt ihrer Stückzahl an. In dieser Sektion werden alle Materialien im Allgemeinen gelistet und beschrieben.

2.1 LED

Eine Leuchtdiode (LED) ist ein Bauelement der Elektronik. Wird sie von elektrischen Strom durchflossen beginnt sie Licht auszustrahlen. Die Wellenlänge, d.h., die Farbe des Lichts sowie ob es für das menschliche Auge sichtbar ist oder nicht, hängt von den benutzten Materialien im inneren der LED ab. Die in der LED verwendeten Materialien sind für dieses Protokoll nicht weiter von Bedeutung. Für die beschriebenen Aufgaben wurden die folgenden LEDs verwendet.

Tabelle 1: LEDs - Farben, Flussspannung, Maximalstrom [2]

Farbe	Durchflussspannung	Maximalstrom
Rot	1,6V - 2,2V	20mA
Gelb	1,9V - 2,5V	20mA
Grün	1,9V - 2,5V	20mA

2.2 Widerstände

Widerstände werden verwendet, um einen Spannungsabfall in einem Stromkreis zu verursachen. Damit kann die Stromstärke in einem Stromkreis begrenzt beziehungsweise verringert werden. Oft wird bei der Verwendung von

Tabelle 2: Farbcodierung von Widerständen

Farbe	1.Ring (10er Stelle)	2.Ring (1er Stelle)	3.Ring (Multiplikator)	4.Ring (Toleranz)
Braun	1	0	10	± 1
Grün	5	5	100 000	± 0.5
Gold	-	-	0.1	± 5

LEDs ein sogenannter Vorwiederstand verwendet, um die Stromstärke soweit abzusenken, dass die LED nicht beschädigt wird.

In den Aufgaben wurden Widerstände der sogenannten E6-Reihe verwendet. E-Reihen sind normierte Widerstandsgrößen, wobei die Zahl die Stufen zwischen den Potenzen angibt. D.h., bei der E6-Reihe sind sechs verschiedene Widerstandsgrößen zwischen 10Ω und 100Ω, zwischen 100Ω und 1000Ω, u.s.w. bis zum oberen Limit von $10M\Omega$.

Die verwendeten Widerstände sind Farbcodiert, um sie voneinander unterscheiden zu können. Die Farbkodierung von Widerständen wird in den Aufgaben, in denen sie verwendet werden, angegeben. Die Bedeutung der Codierung der verwendeten Widerstände kann in Tab. 2 abgelesen werden.

2.3 Taster

2.4 Mikrocontroller [1]

Mikrocontroller werden verwendet, um komplexere Logik in eine elektronische Schaltung zu integrieren. Die in diesem Protokoll verwendete Mikrocontroller sind Mikrocontroller vom Typ Arduino Uno Rev 3. Diese Mikrocontroller besitzen 14 digitale Pins, an denen eine Spannung von entweder 0V oder 5V angelegt oder ausgegeben werden kann. Des Weiteren besitzt es sechs analoge Pins, welche auch auf Spannungsschwankungen reagieren können. Weitere wichtige Pins sind In-Pins, welche eine Spannung von 5V liefern können und Out-Pins, welche eine Verbindung zur Erdung herstellen.

3 Begriffe

3.1 Pulsewellenmodulation (PWM)

3.2 PullUp

3.3 Baudrate

4 Aufgabe 1 - Ampelsteuerung

Es soll eine Ampelsteuerung implementiert und getestet werden. Die Ampel wird mithilfe von drei LEDs, in den Farben Rot, Gelb und Grün, aufgebaut. Weiters soll die Steuerung folgende Funktionsweise implementieren:

Phase 1 soll die Ampel auf Rot setzen. D.h., die rote LED wird eingeschaltet. Dieser Zustand soll vier Sekunden lang gehalten werden.

Phase 2 soll zusätzlich zur roten LED die gelbe einschalten. Dieser Zustand soll eine Sekunde lang gehalten werden.

Phase 3 soll die rote sowie die gelbe LED ausschalten, während die Grüne eingeschaltet wird. Dieser Zustand soll vier Selunden lang gehalten werden.

Phase 4 soll die grüne LED ausschalten während die Gelbe eingeschaltet wird. Dieser Zustand soll eine Sekunde lang gehalten werden. Nach Ablauf der vier Sekunden soll die grüne LED erlöschen und der Ablauf bei Phase 1 neu gestartet werden.

4.1 Materialien

Tabelle 3: Aufgabe 1 - Verwendete Materialien

Bezeichnung	Eigenschaften	Menge
Widerstand	150Ω	3
	Braun - Grün - Braun - Gold	
LED	Rot	1
LED	Gelb	1
LED	Grün	1
Mikrocontroller	Arduino Uno R3	1

4.2 Vorbereitung

Für den Schaltkreis müssen die Vorwiederstände für die LEDs berechnet werden. Folgende Angaben sind bekannt.

- Ausgangsspannung der Pins des Mikrocontrollers: $V_{out} = 5V$

- Diodenspannung der LEDs: $U_D = 2V$
- Diodenstrom der LEDs: $I_D = 15mA$

Zur Berechnung wird folgender Stromkreis angenommen.

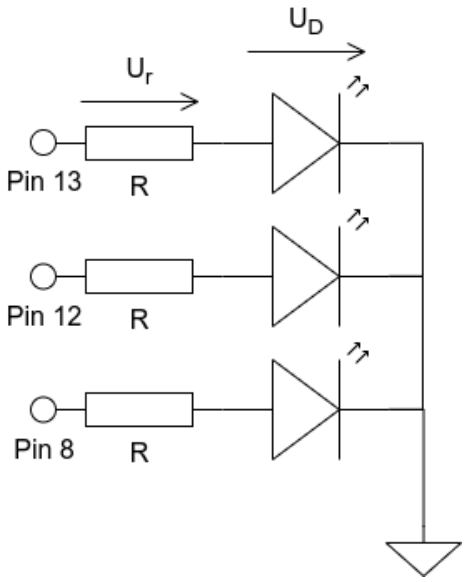

Abbildung 1: Stromkreis A1

Die Formel zur Berechnung des Stroms durch eine Diode ist bekannt.

$$I_D = \frac{1}{R_d} * (U - U_D) \tag{1}$$

Damit kann der benötigte Vorwiderstand berechnet werden.

$$\begin{aligned} I_D = \frac{1}{R_D} * (U - U_D) \Rightarrow \\ R_D = \frac{1}{I_D} * (U - U_D) \\ = \frac{1}{15mA} * (5V - 2V) \\ = 200\Omega \end{aligned} \tag{2}$$

Nachdem in der E6 Reihe keine 200Ω Widerstände vorhanden sind, wurden 150Ω gewählt. Diese Wahl erfolgt aus folgenden Überlegungen.

1. Laut angabe benötigt die LED $2V$ um zu schalten.
2. Es existieren die Widerstände 150Ω und 220Ω in der E6-Reihe
3. Bei einem Widerstand von 220Ω würden, laut Gleichung 1, $I_D = \frac{3V}{220\Omega} = 13,636mA$ durch die LED fließen.
4. Bei einem Widerstand von 150Ω würden, laut Gleichung 1, $I_D = \frac{3V}{150\Omega} = 20mA$ durch die LED fließen.

Da die Leuchtkraft einer LED von der Stromstärke abhängt und der Maximalstrom bei $20mA$ liegt, wurden 150Ω gewählt. Bei dieser Größe wird die LED bei theoretisch voller Leuchtkraft betrieben ohne die Lebensdauer markant zu verkürzen.

4.3 Praktikumsaufgabe

Der Stromkreis wurde laut Abbildung 1 implementiert. Die Implementierung wird in Abbildung 2 gezeigt.

Wie in Abbildung 2 gezeigt, wurden die Pins 13, 12 und 8 gewählt. Für die Wahl wurden die digitalen Pins herangezogen, da eine analoge Ausgabe nicht erforderlich war. Die Pins die mit einer Tilde (∼) markiert sind, sind in der Lage ein PWM-Signal zu liefern. Da ein solches Signal nicht benötigt wird, wurden auch diese ausgeschlossen. Von oben nach unten betrachte, wurden nun drei Pins ausgewählt, diese sind 13, 12, und 8. Die genannten Pins werden, im Code, als digitale Output-Pins konfiguriert.

Im nachfolgendem Text wird der Programmcode der Aufgabe erläutert. Einzelne Teile des Codes werden ausgewählt und beschrieben. Am Ende dieser Sektion befindet sich der vollständige Programmcode.

```
1 const int PIN_RED = 13;
2 const int PIN_YELLOW = 12;
3 const int PIN_GREEN = 8;
```

Listing 1: Konstanten für Aufgabe 1

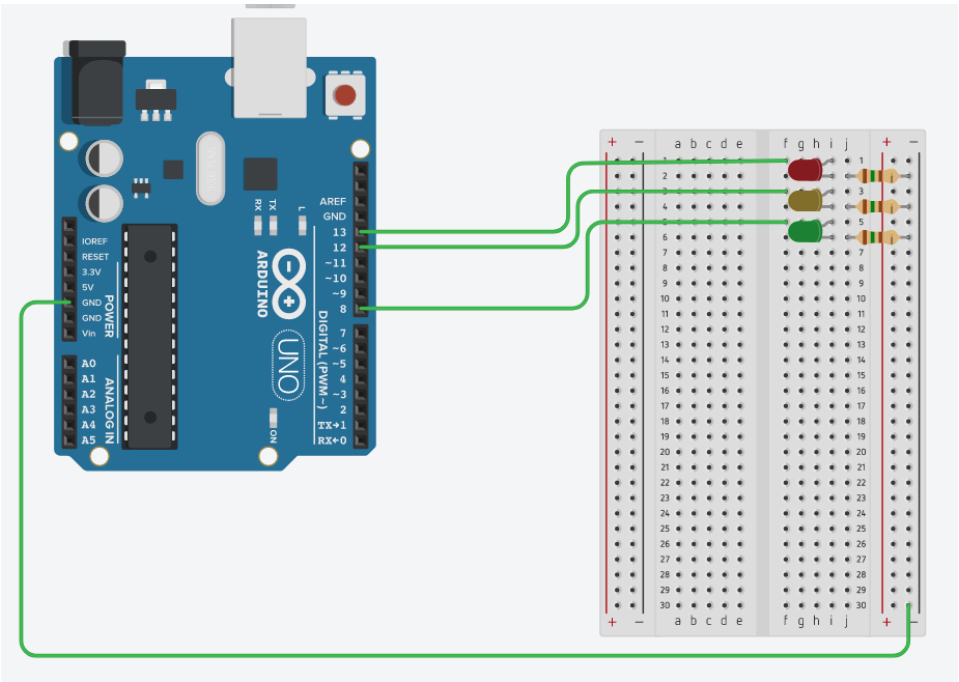

Abbildung 2: Implementiert Stromkreis von Aufgabe 1

In Listing 1 werden die Pins definiert. Durch das Anlegen von Konstanten für die Pins, kann der Code aussagekräftiger gestaltet werden. Des weiteren entsteht dadurch die Möglichkeit, andere Pins zu verwenden, ohne den gesamtem Code durchgehen zu müssen. Es reicht die Nummer an einer Stelle zu ändern.

```
void setup()
{
  pinMode(PIN_RED, OUTPUT);
  pinMode(PIN_YELLOW, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);
}
```

Listing 2: Setup für Aufgabe 1

In Listing 2 werden die Pins angelegt und konfiguriert. Wie oben beschrieben werden die Pins als Output-Pins, d.h., als Spannungsquelle, angelegt.

```
void off(int pin) {
  digitalWrite(pin, LOW);
}
```

```
4
5  void on(int pin) {
6    digitalWrite(pin, HIGH);
7  }
```

Listing 3: On- und Off Methoden für Aufgabe 1

In Listing 3 werden die Pins entweder ausgeschalten (LOW) oder eingeschalten (HIGH). Durch das Verwenden dieser Methoden ist es einfacher, die Phasen zu definieren bzw., im Code zu erkennen.

```
1  void loop()
2  {
3    off(PIN_YELLOW);
4    on(PIN_RED);
5    delay(4 * 1000);
6
7    on(PIN_YELLOW);
8    delay(1 * 1000);
9
10   off(PIN_RED);
11   off(PIN_YELLOW);
12   on(PIN_GREEN);
13   delay(4 * 1000);
14
15   off(PIN_GREEN);
16   on(PIN_YELLOW);
17   delay(1 * 1000);
18 }
```

Listing 4: Programmschleife für Aufgabe 1

In Listing 17 werden die einzelnen Phasen implementiert. Nach jeder Leerzeile, d.h., nach Zeile 6, 9, und 14 beginnt jeweils eine neue Phase.

Zu Beginn wird Phase 1 implementiert. Diese Schaltet die gelbe LED, Pin 12, aus, falls vorher Phase 4 aktiv war und schaltet die rote LED, Pin 13, an. Danach wird die $delay(x)$ Funktion aufgerufen, welche die Programmausführung für x Millisekunden unterbricht. Die Angabe von x als Berechnung aus $Sekunde * 1000$ wurde gewählt, um im Code besser erkenntlich zu machen, dass es sich um Millisekunden handelt. Der Programmfluss wird daher für vier Sekunden unterbrochen.

Dann folgt Phase 2, welche zusätzlich zur Roten auch die gelbe LED einschaltet. Der Programmfluss wird für eine Sekunde unterbrochen.

Es folgt Phase 3. Die Rote und die gelbe LED werden ausgeschaltet. Die

Grüne, Pin 8, wird eingeschaltet. Der Programmfluss wird für weitere vier Sekunden unterbrochen.

Schlussendlich folgt Phase 4. Es wird die grüne LED wieder ausgeschaltet, während die Gelbe aktiv wird. Es folgt wieder eine Unterbrechung des Programms für eine Sekunde. Am Ende der *loop*() Funktion wird sie wieder von Beginn an, d.h., von Phase 1 aus, ausgeführt.

```
const int PIN_RED = 13;
const int PIN_YELLOW = 12;
const int PIN_GREEN = 8;

void setup()
{
  pinMode(PIN_RED, OUTPUT);
  pinMode(PIN_YELLOW, OUTPUT);
  pinMode(PIN_GREEN, OUTPUT);
}

void loop()
{
  off(PIN_YELLOW);
  on(PIN_RED);
  delay(4 * 1000);

  on(PIN_YELLOW);
  delay(1 * 1000);

  off(PIN_RED);
  off(PIN_YELLOW);
  on(PIN_GREEN);
  delay(4 * 1000);

  off(PIN_GREEN);
  on(PIN_YELLOW);
  delay(1 * 1000);
}

void off(int pin) {
  digitalWrite(pin, LOW);
}

void on(int pin) {
  digitalWrite(pin, HIGH);
}
```

Listing 5: Vollständiger Programmcode für Aufgabe 1

4.4 Fehlerdiskussion

Es wurden während dieser Aufgabe keine Fehler gefunden.

4.5 Zusammenfassung

5 Aufgabe 2 - Reaktionsspiel

Es soll eine Schaltung implementiert werden, welche ein lichtbasiertes Reaktionsspiel darstellt. Die Schaltung soll aus fünf LEDs bestehen, welche der Reihe nach leuchten und erlöschen. D.h., LED 1 leuchtet, LED 1 erlischt und LED 2 leuchtet, LED 2 erlischt und LED 3 leuchtet, u.s.w.. Sobald die letzte LED erlischt soll die erste LED wieder leuchten und der Rythmus von neuen beginnen.

Die LEDs werden in der Reihenfolge Rot - Gelb - Grün - Gelb - Rot geschaltet und angeordnet. Des weiteren wird ein Taster eingebaut. Wird der Taster gedrückt, wenn die grüne LED leuchtet, halbiert sich die Zeit zwischen der die LEDs geschaltet werden. D.h., bleibt eine LED fünf Sekunden eingeschaltet, bevor die nächste LED geschaltet wird, so bleibt sie danach nur 2,5 Sekunden lang eingeschaltet. Wird der Taster erneut gedrückt, wenn die grüne LED leichtet, halbiert sich die Zeit erneut auf 1,25 Sekunden, u.s.w..

Wird der Taster gedrückt, wenn die grüne LED nicht leuchtet, wird die Zeit wieder auf den ursprünglichen Wert gesetzt. In diesem Beispiel also zurück auf fünf Sekunden. In diesem Fall sollen auch alle LEDs gleichzeitig drei mal blinken, bevor das Spiel schlussendlich von vorne beginnt.

5.1 Materialien

Tabelle 4: Aufgabe 2 - Verwendete Materialien

Bezeichnung	Eigenschaften	Menge
Widerstand	150Ω Braun - Grün - Braun - Gold	5
Widerstand	$10k\Omega$ Braun - Schwarz - Orange - Gold	1
LED	Rot	2
LED	Gelb	2
LED	Grün	1
Taster	4 Polig	1
Mikrocontroller	Arduino Uno R3	1

5.2 Vorbereitung

Für den ersten Teil muss wieder der Vorwiderstand der LEDs berechnet werden. Die Berechnung und des Widerstands und die reale Auswahl aus der E6-Reihe erfolgt wie in Sektion 4.2. D.h., es wurden 200Ω berechnet und der 150Ω Widerstand aus der E6-Reihe ausgewählt.

Der zweite Teil bestand aus zwei Fragestellungen. Erstens, wie kann ein Mikrocontroller-Programm unterbrochen werden, um ,z.B., einen Druck auf einen Taster zu erkennen. Dies kann mithilfe eines sogenannten *Interrupts* implementiert werden. Dafür wird ein Pin als Input-PullUp-Pin angelegt und für diesen Pin eine Funktion definiert, welche aufgerufen wird, sobald der Taster gedrückt wird.

Zweitens, wie kann ermittelt werden, ob der Taster zum richtigen oder falschen Zeitpunkt gedrückt wurde. Dafür kann eine Variable mit dem *volatile*-Schlüsselwort angelegt werden. Damit wird die Variable nicht in einem Zwischenspeicher behalten, sondern wird immer vom Hauptspeicher gelesen, wenn sie verwendet wird. Diese Variable kann die Werte "wahr", bzw. true", und falsch", bzw false", annehmen. Wird der Button gedrückt kann die Variable auf "wahr"gesetzt werden, ansonsten auf falsch". In der Programmlogik selbst kann dann überprüft werden, welchen Wert die Variable momentan besitzt. Damit kann ausgewertet werden, ob sie zum richtigenS-Zeitpunkt den Wert "wahr"hat, oder nicht.

5.3 Praktikumsaufgabe

In Abbildung 3 ist die implementierte Schaltung zu sehen. Zur Klärung sei gesagt, dass das blaue Kabel nur mit Spalte 15 verbunden ist und nicht mit einem der Widerstände, obwohl das Bild es vermuten lässt.

Weiters ist die Wahl der Pins zu erkennen. Die Wahl viel auf die Pins 9 bis 13 zur Steuerung der LEDs und auf Pin 2 zur Erkennung des Tastendrucks. Die Pins zur Ansteuerung der LEDs wurde willkürlich getroffen. Es handelt sich nur von oben nach unten um die ersten, zur Verfügung stehenden Pins. Pin 2 wurde gewählt, da nur Pin 2 und 3 in der Lage sind, an einen Interrupt gekoppelt zu werden.

Im Schaltkreis ist weiters der Taster zu sehen, der mit einem Pull-Up-Widerstand, wie in Sektion 3.2 zu sehen, versehen ist. Die Terminals an

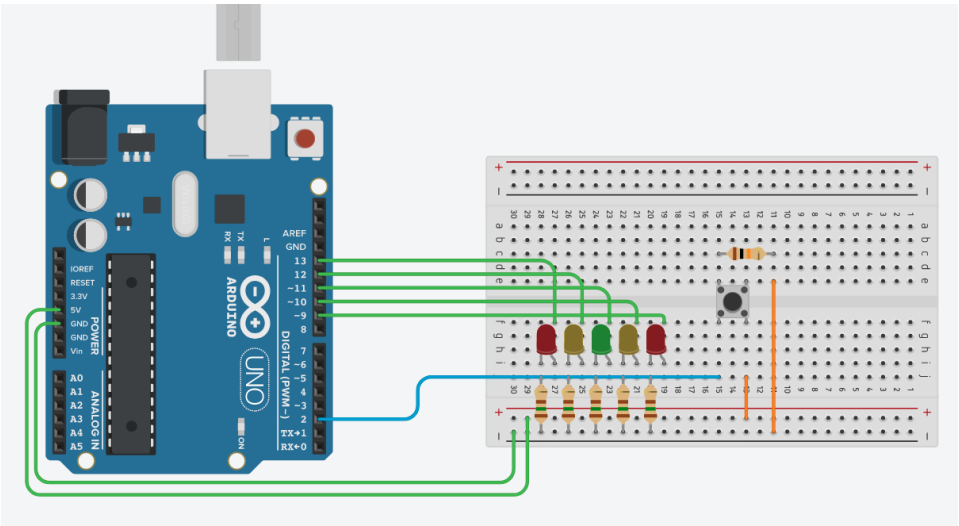

Abbildung 3: Implementierter Stromkreis von Aufgabe 2

der Spalte 15 auf beiden Seiten sind verbunden, die Terminal an den Spalten 15 und 13 werden bei einem Tastendruck verbunden. Damit ergibt sich das Teilschaltbild des Tasters, dass in Abbildung 4 zu sehen ist. Die alleinstehenden Zahlen geben die zugehörige Spalte des Steckbrettes an. Des weiteren sei gesagt, dass in der Abbildung zwei Schalter zu sehen sind, diese repräsentieren die zwei vorhanden Terminals des echten Schalters. Bei Tastendruck werden beide gleichzeitig geschlossen.

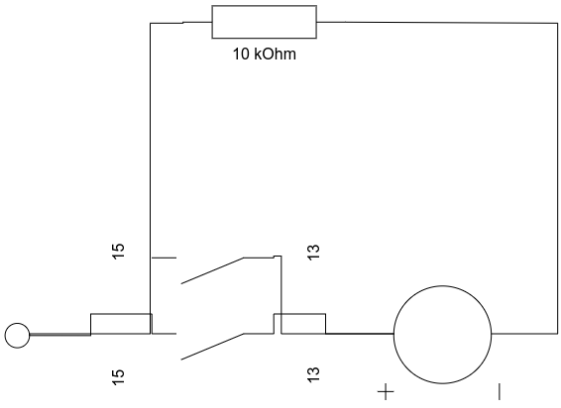

Abbildung 4: Schaltung des Tasters mit Pull-Up Widerstand

Im nachfolgendem Text wird der Programmcode der Aufgabe erläutert. Einzelne Teile des Codes werden ausgewählt und beschrieben. Am Ende dieser Sektion befindet sich der vollständige Programmcode.

```
onst int NUM_PINS = 5;
const int WINNING_PIN_INDEX = 2;
const int PINS[] = {
  13, 12, 11, 10, 9
};
const int PIN_BUTTON = 2;
```

Listing 6: Setzen der Pin-Konstanten

In Listing 6 werden die Nummern der Pins angelegt. Die Besonderheit gegenüber der vorherigen Sektion, Sektion 4, ist die Verwendung eines Arrays. In diesem werden die verwendeten Pinnummern zu Ansteuerung der LEDs angegeben. Mit NUM_PINS wird die Länge des Arrays angegeben. Solch eine Angabe ist nützlich, da die Länge eines Arrays in der gegebenen Programmiersprache, C, nicht trivial findbar ist.

Um in C die Länge eines Arrays zu bestimmen, müsste zuerst mittels $sizeof(PINS)$ die Menge an Speicher ermittelt werden, die das Arrays belegt. Dieser Wert müsste sodann durch das Ergebnis von $sizeof(int)$, die Menge an Speicher die ein Integer-Wert im Speicher belegt, dividiert werden. Das Ergebnis aus $sizeof(PINS)/sizeof(int)$ würde dann die Länge des Arrays ergeben. Einfacher ist es daher, die Größe als Konstante anzugeben, wenn diese, wie in diesem Fall, bekannt ist. Mit der Konstante PIN_BUTTON wird die Pinnummer des Pins angegeben, welcher später den Zustand des Tasters erkennt.

Die Konstante $WINNING\_PIN\_INDEX$ gibt an, welcher Index des Pin-Arrays die grüne LED darstellt, D.h., die der Taster gedrückt, wenn der Pin von $PINS[WINNING\_PIN\_INDEX]$ eingeschaltet ist, ist zum richtigen Zeitpunkt gedrückt worden.

```
const unsigned long INTERVAL_DEFAULT = 5000;
```

Listing 7: Standard Interval zum LED wechsel

Mit der Konstanten die in Listing 6 angegeben ist, wird das Standardinterval angegeben, mit dem bei den LEDs weitergesprungen wird. D.h., nach Ablauf der Zeit von $INTERVAL\_DEFAULT$, in Millisekunden, wird zur nächsten LED gesprungen, wenn das Spiel neu gestartet wurde.

```
unsigned long current_millis = 0;
```

```
unsigned long previous_millis = 0;
unsigned long interval = INTERVAL_DEFAULT;
```

Listing 8: Variablen zur Zeitmessung

Die Variablen in Listing 8 werden dazu verwendet, die vergangene Zeit im Programm mitzuverfolgen. In *current_millis* wird gespeichert, wie viel Zeit seit dem Programmstart vergangen ist. Der Wert in *previous_millis* gibt an, zu welchem Zeitpunkt die letzte Aktion durchgeführt wird. Die Bedeutung dieser Variable wird im Verlauf des Programmcodes klarer. In *interval* ist das derzeitige Interval zwischen Wechseln der LEDs gespeichert. Zu Beginn wird sie mit der Konstante aus Listing 7 initialisiert. Sie wird bei jedem Neustart des Spiels auf diesen Wert zurückgesetzt.

```
int pin_index = 0;

volatile bool button_pressed = false;
volatile bool game_over = false;
```

Listing 9: Variablen zur Zustandsbestimmung des Spiels

Durch die Variablen in Listing 9 wird der derzeitige Zustand des Spiels mitverfolgt. Sie haben zusätzlich zum Typ noch das Schlüsselwort *volatile*, welches im zweiten Teil der Sektion **??** beschrieben wurde. Diese Schlüsselwort benötigen sie, da die Variablen in einem Interrupt verändert werden und dadurch direkt vom Speicher ausgelesen werden müssen. Eine Annäherung des mit diesen Variablen bewirkten Zustandsverlauf kann in Abbildung 5 abgelesen werden.

Die Variable *pin_index* ist nicht mittels *volatile* markiert, da sie sich nur in der Hauptschleife des Programms ändert. Nach Ablauf von *interval* wird sie um 1 erhöht bis zu einem Maximum von $NUM\_PINS - 1$. Würde sie den Wert NUM_PINS annehmen, wird sie wieder auf 0 zurückgesetzt. Diese Variable gibt an, welcher Index, von 0 beginnend, aus $PINS$ verwendet wird, um eine LED anzusteuern. D.h., $pin\_index = 0 \Rightarrow Pin13 = HIGH$, $pin\_index = 1 \Rightarrow Pin12 = HIGH$, u.s.w..

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

Listing 10: Einstellen der seriellen Schnittstelle

Mit dem Kommando in Listing 10 wird eine serielle Verbindung zu einem verbundenem Computer hergestellt. Damit kann später im Programm Text

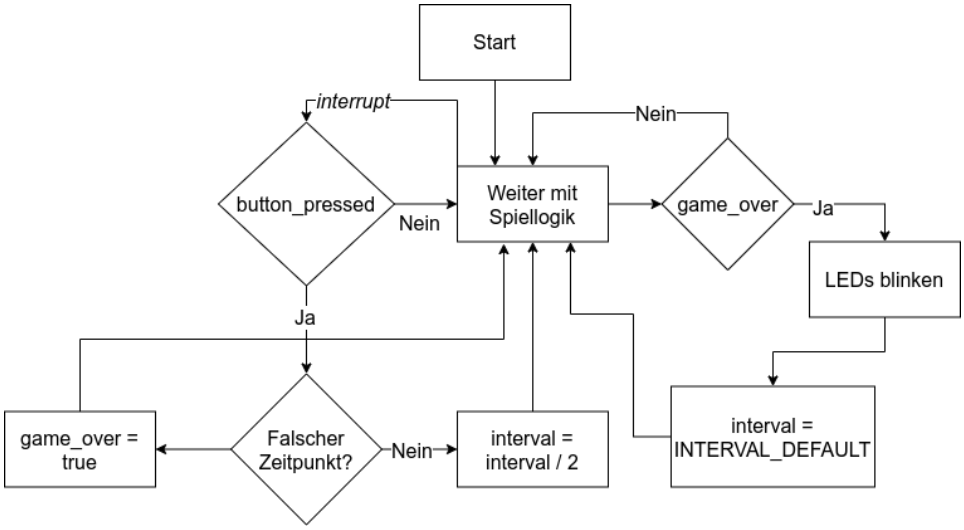

Abbildung 5: Annäherung des Spielverlaufs

an den Computer übertragen oder vom Computer auf den Mikrocontroller gesendet werden. Das Argument der Methode, 9600 gibt die Baudrate an.

```
int i;
for (i = 0; i < NUM_PINS; i++) {
    pinMode(PINS[i], OUTPUT);
}
```

Listing 11: Konfiguration der Pins

In Listing 11 werden die Pins durchgegangen. Hier wird das in Listing 6 angelegte Array zum ersten Mal verwendet. Es werden hier alle Pinnummern in dem Arrays durchgelaufen und als Output eingestellt. Dadurch muss nicht jede Pinnummer einzeln eingerichtet werden.

```
pinMode(PIN_BUTTON, INPUT_PULLUP);

attachInterrupt(
    digitalPinToInterrupt(PIN_BUTTON),
    on_button_change,
    CHANGE);
```

Listing 12: Einstellen des Buttons

Der Code in Listing 12 setzt nun den Pin, der an dem Taster angeschlossen ist, auf Input-PullUp und verknüpft ihn mit einem Interrupt. Die Funktion

on_button_change wird nun aufgerufen, jedes mal wenn der Taster manipuliert wird. Der Parameter dritte Parameter der *attachInterrupt*-Methode, in diesem Fall $CHANGE$, gibt an was passieren muss damit die Methode aufgerufen wird. $CHANGE$ beduetet, dass jedes mal wenn sich das Potenzial ändert, daher entweder von $5V$ auf $0V$ fällt oder von $0V$ auf $5V$ steigt, der Interrupt ausgeführt wird.

Des weiteren gibt es noch weitere Werte, die übergeben werden können, welche verschieden Zustände abbilden.

- LOW, der Interrupt wird ausgeführt, wenn am Pin $0V$ anliegen
- $CHANGE$, der Interrupt wird ausgeführt, wenn sich das Potenzial am Pin ändert
- $RISING$, der Interrupt wird ausgeführt, wenn die Spannung am Pin von $0V$ auf $5V$ steigt
- $FALLING$, der Interrupt wird ausgeführt, wenn die Spannung am Pin von $5V$ auf $0V$ fällt

```
1   init_game();
2 }
3
4 void init_game() {
5   game_over = false;
6   interval = INTERVAL_DEFAULT;
7   pin_index = 0;
8   digitalWrite(PINS[0], HIGH);
9 }
```

Listing 13: Setzen der Standardwerte

Listing 13 zeigt die Methode, die aufgerufen wird, wenn das Spiel neu gestartet wird. Der Aufruf von *init_game*() zu Begin des Listings ist noch in der *setup*()-Methode zu finden. Dadurch werden noch vor Spielbeginn die Werte richtig gesetzt.

In der *init_game*()-Methode wird des Zustand mittels *game_over* zurückgesetzt. Das Intervall zwischen den LED wechseln wird auf den Standardwert gesetzt. Die aktive LED wird mittels $pin\_index = 0$ auf die Erste gesetzt und mit dem Aufruf von $digitalWrite(\dots)$ eingeschaltet.

```
void on_button_change() {
  int button_state = digitalRead(PIN_BUTTON);
  if (button_state == HIGH) {
    on_button_rising();
  } else {
    on_button_falling();
  }
}

void on_button_rising() {
  button_pressed = true;

  if (pin_index == WINNING_PIN_INDEX &&
     !game_over) {
    interval = interval / 2;
  } else {
    game_over = true;
  }
}

void on_button_falling() {
  button_pressed = false;
}
```

Listing 14: Interrupt Methoden des Tasters

Obwohl die Methoden, welche die Logik des Interrupts darstellen, am Ende des Codes liegen, werden sie hier besprochen. Damit kann ein Kontext für andere Codeteile erzeugt werden.

In Listing 14 wird die in Listing 12 verwendete Methode gezeigt. Zu Beginn von *on_button_change*() wird der Zustand des Tasters ausgelesen. Liegen zum Zeitpunkt des Aufrufs $5V$ am Pin an, so wird *on_button_rising*() aufgerufen, ansonsten wird *on_button_falling*() verwendetet. Dieser Distinktion muss verwendet werden, da nur ein Interrupt pro Pin angelegt werden kann. Es können die entsprechenden Methoden nicht gleichzeitig direkt an den Pin angehängt werden. Daher muss die Methode *on_button_change*() diese Unterscheidung durchführen.

In *on_button_rising*() wird zuerst der Zustand von *button_pressed* auf *true* gesetzt. Damit kann in der Programmschleife erkannt werden, ob der Taster gedrückt wird oder nicht. Danach wird überprüft welche LED zurzeit leuchtet, indem der Wert in *pin_index* überprüft wird und ob das Spiel bereits vorbei ist. Die letztere Überprüfung wird gemacht, da der Taster auch dann getätigt werden kann, wenn, z.b., alle LEDs blinken. Wurde der Taster nicht

zum richtigen Zeitpunkt getätigt, so wird der Zustand des Spiels auf "Game Over"gesetzt, indem die enstprechende Variable auf *true* gesetzt wird.

```
void blink_all() {
  int j;
  for (j = 0; j < 3; j++) {
    int i;
    write_to_all(HIGH);
    delay(500);
    write_to_all(LOW);
    delay(500);
  }
}

void write_to_all(int state) {
  int i;
  for (i = 0; i < NUM_PINS; i++) {
    digitalWrite(PINS[i], state);
  }
}

void increase_pin_index() {
    pin_index++;
    if (pin_index >= NUM_PINS){
      pin_index = 0;
    }
}
```

Listing 15: Hilfsmethoden

In Listing 15 werden die verwendeten Hilfsmethoden gezeigt. Für *write_to_all* kann der gewünschte Zustand übergeben werden, $HIGH$ oder LOW, und dieser wird, ähnlich wie in *setup*(), mittels einer Schleife an alle Pins im $PINS$-Array übernommen. *blink_all* wird verwendet, um das Blinken der LEDs am Ende des Spiels zu implementieren. Es werden hier drei mal alle Pins im $PINS$-Array erst mittels $write\_to\_all(HIGH)$ eingeschaltet. Danach wird 500ms lang mit $delay(500)$ gewartet. Mit $write\_to\_all(LOW)$ und anschließenden $delay(500)$ werden die LEDs wieder ausgeschaltet und 500ms lang gewartet.

increase_pin_index() hat die Aufgabe, die Variable *pin_index* zu erhöhen. Würde die Variable das Limit von $NUM\_PINS - 1$ überschreiten, wird die Variable wieder auf 0 gesetzt.

```
void loop()
{
  current_millis = millis();
```

```
4
5   if (current_millis - previous_millis >= interval
6       && !button_pressed
7       && !game_over) {
8     digitalWrite(PINS[pin_index], LOW);
9     increase_pin_index();
10    digitalWrite(PINS[pin_index], HIGH);
11  previous_millis = current_millis;
12  } else if (game_over) {
13    blink_all();
14    init_game();
15    previous_millis = current_millis;
16  }
17 }
```

Listing 16: Programmschleife

In Listing 16 wird die Programmschleife beschrieben. Hier wird nun die Variable *current_millis* verwendet, um die vergangene Zeit zu speichern. Ist der Abstand zwischen *current_millis* und *previous_millis*, zu Beginn 0, größer oder gleich dem derzeitigen Intervall, so wird zuerst die momentan aktive LED ausgeschaltet. Danach wird die Methode *increase_pin_index* ausgeführt, um den verwendeten Index auf den nächsten Pin zu setzen. Die LED die als nächstes leuchten soll, wird sodann eingeschalten. Schlussendlich wird der Wert von *previous_millis* auf den Wert von *current_millis* gesetzt, um wieder den Zustand $current\_millis - previous\_millis < interval$ zu erreichen. Dieser Ablauf wird allerdings nur dann durchgeführt, wenn weder der Taster gedrückt, noch das Spiel vorbei ist.

Ist das Spiel vorbei so wird die vorher beschriebene Methode *blink_all*() aufgerufen, um alle LEDs blinken zu lassen. Mittels *init_game*() werden alle globalen Werte wieder auf ihren Standard zurückgesetzt. Schlussendlich wird der Wert von *previous_millis* auf den Wert von *current_millis* gesetzt, um das selbe Intervall wie bei einem Spielstart zu haben. Würde dies nicht gemacht, könnte das Spiel früher von der ersten LED auf die Zweite überspringen als gewollt.

```
1 const int NUM_PINS = 5;
2 const int WINNING_PIN_INDEX = 2;
3 const int PINS[] = {
4   13, 12, 11, 10, 9
5 };
6 const int PIN_BUTTON = 2;
7 const unsigned long INTERVAL_DEFAULT = 5000;
8
```

```
unsigned long current_millis = 0;
unsigned long previous_millis = 0;
unsigned long interval = INTERVAL_DEFAULT;

int pin_index = 0;

volatile bool button_pressed = false;
volatile bool game_over = false;

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

  int i;
  for (i = 0; i < NUM_PINS; i++) {
    pinMode(PINS[i], OUTPUT);
  }

  pinMode(PIN_BUTTON, INPUT_PULLUP);

  attachInterrupt(
    digitalPinToInterrupt(PIN_BUTTON),
    on_button_change,
    CHANGE);

  init_game();
}

void init_game() {
  game_over = false;
  interval = INTERVAL_DEFAULT;
  pin_index = 0;
  digitalWrite(PINS[0], HIGH);
}

void loop()
{
  current_millis = millis();

  if (current_millis - previous_millis >= interval
       && !button_pressed
       && !game_over) {
    digitalWrite(PINS[pin_index], LOW);
    increase_pin_index();
    digitalWrite(PINS[pin_index], HIGH);
  previous_millis = current_millis;
  } else if (game_over) {
    blink_all();
    init_game();
```

```
58      previous_millis = current_millis;
59    }
60  }
61
62  void blink_all() {
63    int j;
64    for (j = 0; j < 3; j++) {
65      int i;
66      write_to_all(HIGH);
67      delay(500);
68      write_to_all(LOW);
69      delay(500);
70    }
71  }
72
73  void write_to_all(int state) {
74    int i;
75    for (i = 0; i < NUM_PINS; i++) {
76      digitalWrite(PINS[i], state);
77    }
78  }
79
80  void increase_pin_index() {
81      pin_index++;
82      if (pin_index >= NUM_PINS){
83        pin_index = 0;
84      }
85  }
86
87  void on_button_change() {
88    int button_state = digitalRead(PIN_BUTTON);
89    if (button_state == HIGH) {
90      on_button_rising();
91    } else {
92      on_button_falling();
93    }
94  }
95
96  void on_button_rising() {
97    button_pressed = true;
98
99    if (pin_index == WINNING_PIN_INDEX &&
100       !game_over) {
101     interval = interval / 2;
102   } else {
103     game_over = true;
104   }
105 }
106
```

```
107 void on_button_falling() {
108   button_pressed = false;
109 }
```

Listing 17: Vollständiger Programmcode für Aufgabe 2

5.4 Fehlerdiskussion

Es wurde der Fehler gemacht, dass in Listing 16 auch auf den Zustand des Tasters geprüft wird. D.h., der Taster darf nicht gedrückt sein, damit das Spiel weiter läuft. Dieser Fehler verursacht, dass das Lauflicht pausiert und nicht weiterläuft, wenn der Taster nicht mehr losgelassen wird.

Ein weiterer glücklicher Zufall konnte einen weiteren Fehler verhindern. Es wurde nämlich die Funktionsweise des Tasters falsch verstanden. Die Annahme war nicht, dass wie in Abbildung 4 dargestellt, die Spalte 15, bzw. die Spalte 13, verbunden ist, sondern dass Spalte 15 mit Spalte 13 verbunden ist. Würde dies zutreffen, so würde die Schaltung nicht funktionieren. Nachdem jedoch auch die gegebene Schaltung eines Pull-Down Widerstands falsch gelesen wurde, wurde sie unbeabsichtigt richtig implementiert.

5.5 Zusammenfassung

Literatur

[1] Arduino. *Arduino Uno Rev 3*. https://store.arduino.cc/arduino-uno-rev3.

[2] Wikipedia. *Leuchtdiode - Elektrische Eigenschaften*. https://de.wikipedia.org/wiki/Leuchtdiode#Elektrische_Eigenschaften.