

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	Σ
Max.:	9	9	9	9	9	9	9	7	70
Pkt.:									

Inhaltsverzeichnis

1	Einleitung	3
2	Verwendete Materialien	3
2.1	LED	3
2.2	Widerstände	3
2.3	Taster	4
2.4	Mikrocontroller [1]	4
3	Begriffe	5
3.1	Pulsewellenmodulation (PWM)	5
3.2	PullUp	5
4	Aufgabe 1 - Ampelsteuerung	6
4.1	Materialien	6
4.2	Vorbereitung	6
4.3	Praktikumsaufgabe	8
4.4	Fehlerdiskussion	12
4.5	Zusammenfassung	12
5	Aufgabe 2 - Reaktionsspiel	13
5.1	Materialien	13
5.2	Vorbereitung	14
5.3	Praktikumsaufgabe	14
	References	21

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 Vorwiderstand verwendet, um die Stromstärke so weit 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 Pulswellenmodulation (PWM)

3.2 PullUp

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 Sekunden 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 Braun - Grün - Braun - Gold	3
LED	Rot	1
LED	Gelb	1
LED	Grün	1
Mikrocontroller	Arduino Uno R3	1

4.2 Vorbereitung

Für den Schaltkreis müssen die Vorwiderstände 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.

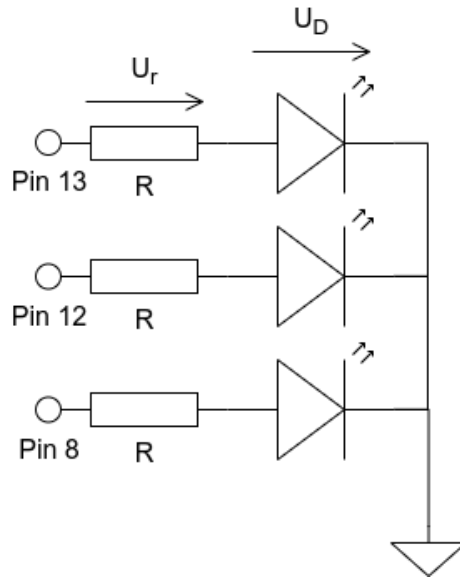


Abbildung 1: Stromkreis A1

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

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

Damit kann der benötigte Vorwiderstand berechnet werden.

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

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 (\sim) 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

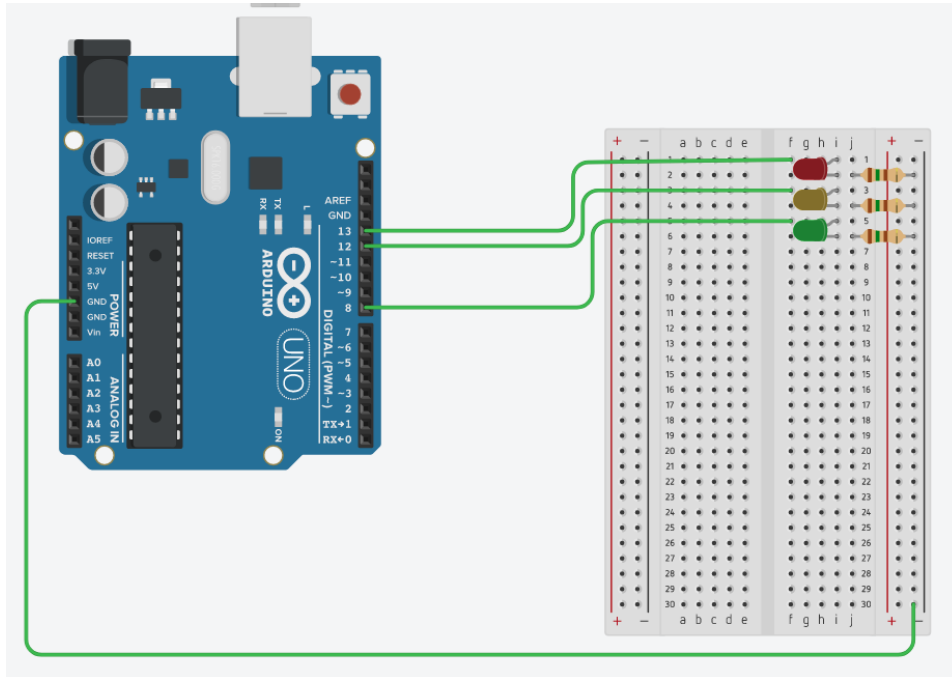


Abbildung 2: Implementiert Stromkreis von Aufgabe 1

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 gesamten Code durchgehen zu müssen. Es reicht die Nummer an einer Stelle zu ändern.

```

1 void setup()
2 {
3     pinMode(PIN_RED, OUTPUT);
4     pinMode(PIN_YELLOW, OUTPUT);
5     pinMode(PIN_GREEN, OUTPUT);
6 }

```

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.

```

1 void off(int pin) {
2     digitalWrite(pin, LOW);
3 }

```

```

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 10 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.

```
1  const int PIN_RED = 13;
2  const int PIN_YELLOW = 12;
3  const int PIN_GREEN = 8;
4
5  void setup()
6  {
7      pinMode(PIN_RED, OUTPUT);
8      pinMode(PIN_YELLOW, OUTPUT);
9      pinMode(PIN_GREEN, OUTPUT);
10 }
11
12 void loop()
13 {
14     off(PIN_YELLOW);
15     on(PIN_RED);
16     delay(4 * 1000);
17
18     on(PIN_YELLOW);
19     delay(1 * 1000);
20
21     off(PIN_RED);
22     off(PIN_YELLOW);
23     on(PIN_GREEN);
24     delay(4 * 1000);
25
26     off(PIN_GREEN);
27     on(PIN_YELLOW);
28     delay(1 * 1000);
29 }
30
31 void off(int pin) {
32     digitalWrite(pin, LOW);
33 }
34
35 void on(int pin) {
36     digitalWrite(pin, HIGH);
37 }
```

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 leuchtet, 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	150 Ω	5
	Braun - Grün - Braun - Gold	
Widerstand	10k Ω	1
	Braun - Schwarz - Orange - Gold	
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 richtigen 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 fiel 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

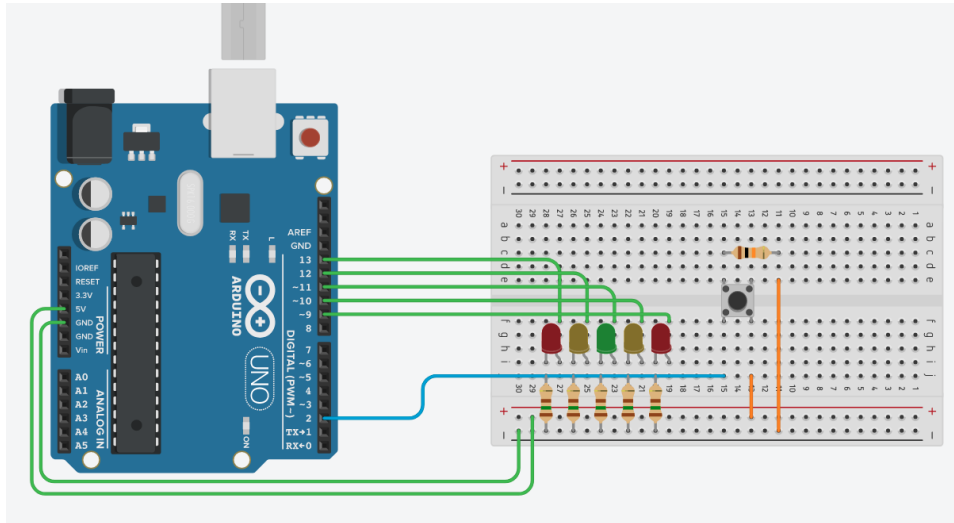


Abbildung 3: Implementierter Stromkreis von Aufgabe 2

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 vorhandenen Terminals des echten Schalters. Bei Tastendruck werden beide gleichzeitig geschlossen.

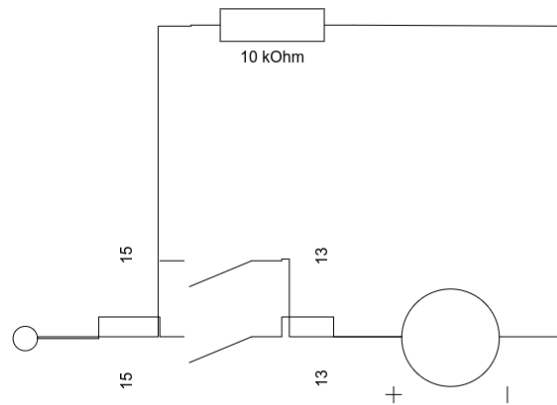


Abbildung 4: Schaltung des Tasters mit Pull-Up Widerstand

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.

```
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;
```

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.

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

Listing 7: Standard Interval zum LED wechsel

Mit der Konstanten die in Listing 6 angegeben ist, wird das Standardintervall 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.

```
1 unsigned long current_millis = 0;
2 unsigned long previous_millis = 0;
3 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.

```

1 int pin_index = 0;
2
3 volatile bool button_pressed = false;
4 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.

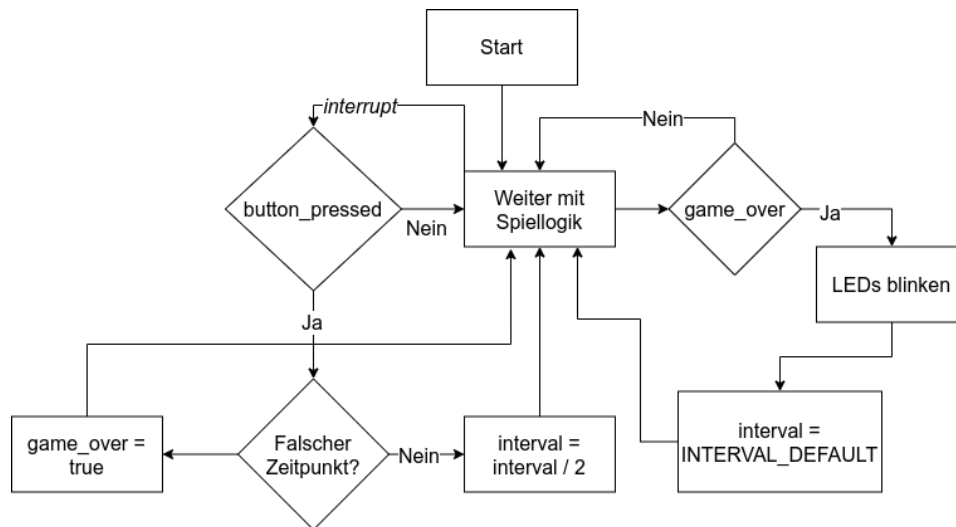


Abbildung 5: Annäherung des Spielverlaufs

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..

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

Listing 10: Einstellen der seriellen Schnittstelle

```
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
9 unsigned long current_millis = 0;
10 unsigned long previous_millis = 0;
11 unsigned long interval = INTERVAL_DEFAULT;
12
13 int pin_index = 0;
14
15 volatile bool button_pressed = false;
16 volatile bool game_over = false;
17
18 void setup()
19 {
20   Serial.begin(9600);
21
22   int i;
23   for (i = 0; i < NUM_PINS; i++) {
24     pinMode(PINS[i], OUTPUT);
25   }
26
27   pinMode(PIN_BUTTON, INPUT_PULLUP);
28
29   attachInterrupt(
30     digitalPinToInterrupt(PIN_BUTTON),
31     on_button_change,
32     CHANGE);
33
34   init_game();
35 }
36
37 void init_game() {
```

```

38 game_over = false;
39 interval = INTERVAL_DEFAULT;
40 pin_index = 0;
41 digitalWrite(PINS[0], HIGH);
42 }
43
44 void loop()
45 {
46     current_millis = millis();
47
48     if (current_millis - previous_millis >= interval
49         && !button_pressed
50         && !game_over) {
51         digitalWrite(PINS[pin_index], LOW);
52         increase_pin_index();
53         digitalWrite(PINS[pin_index], HIGH);
54         previous_millis = current_millis;
55     } else if (game_over) {
56         blink_all();
57         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 11: Vollständiger Programmcode für Aufgabe 2

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.