

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	4
2	Verwendete Materialien	4
2.1	LED	4
2.2	Widerstände	4
2.3	Taster	5
2.4	Mikrocontroller [1]	5
2.5	Logikbausteine	6
3	Begriffe	6
3.1	Pulsewellenmodulation (PWM)	6
3.2	PullUp	6
3.3	Baudrate	6
3.4	Wahrheitstabelle	6
4	Aufgabe 1 - Ampelsteuerung	7
4.1	Materialien	7
4.2	Vorbereitung	7
4.3	Praktikumsaufgabe	9
4.4	Fehlerdiskussion	13
4.5	Zusammenfassung	13
5	Aufgabe 2 - Reaktionsspiel	14
5.1	Materialien	14
5.2	Vorbereitung	15

5.3	Praktikumsaufgabe	15
5.4	Fehlerdiskussion	26
5.5	Zusammenfassung	26
6	Aufgabe 3 - Realisierung eines Logic Analyzers	27
6.1	Materialien	27
6.2	Vorbereitung	27
6.3	Praktikumsaufgabe	27
6.4	Fehlerdiskussion	34
6.5	Zusammenfassung	34
7	Aufgabe 4 - Wertbestimmung von Widerstand und Kondensator	35
7.1	Materialien	35
7.2	Vorbereitung	36
7.2.1	Aufgabe 1	36
7.2.2	Aufgabe 2	36
7.3	Praktikumsaufgabe	36
7.4	Fehlerdiskussion	36
7.5	Zusammenfassung	36
	References	37

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)
Silber	-	-	-	$\pm 10\%$
Gold	-	-	0.1	$\pm 5\%$
Braun	1	0	10	$\pm 1\%$
Rot	2	2	100	$\pm 2\%$
Gelb	4	4	10 000	-
Grün	5	5	100 000	$\pm 0.5\%$
Blau	6	6	1 000 000	$\pm 0.25\%$
Violett	7	7	10 000 000	$\pm 0.1\%$
Grau	8	8	100 000 000	$\pm 0.05\%$
Weiß	9	9	1 000 000 000	-

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.

2.5 Logikbausteine

Um verschiedene Logikgatter in Schaltungen verwenden zu können, ohne komplizierte Schaltkreise bauen zu müssen, werden Logikbausteine verwendet. Diese Bausteine implementieren verschiedene Gatter. D.h., bei zwei Eingängen a und b , wird durch das Anwenden der Funktion, die der Baustein implementiert, der Ausgang y erzeugt. Diese Funktionen bilden logische Operationen ab. Die Erklärung von logischen Funktionen würden über den Umfang dieses Protokolls hinausgehen und werden daher nicht weiter erklärt. In Tabelle 3 sind die verwendeten Bausteine und die Funktionen die sie implementieren gelistet.

Tabelle 3: Bausteine und deren logische Funktionen

Bausteinbezeichnung	Funktion
74HC00	NAND
74HC02	NOR
74HC08	AND
74HC32	OR
74HC86	XOR

3 Begriffe

3.1 Pulswellenmodulation (PWM)

3.2 PullUp

3.3 Baudrate

3.4 Wahrheitstabelle

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 4: 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.

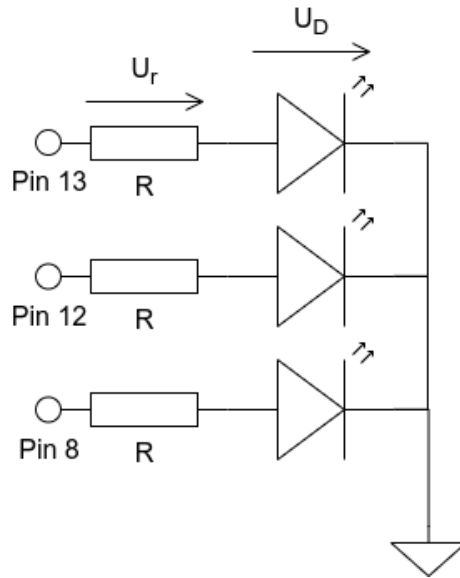


Abbildung 1: Stromkreis A1

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

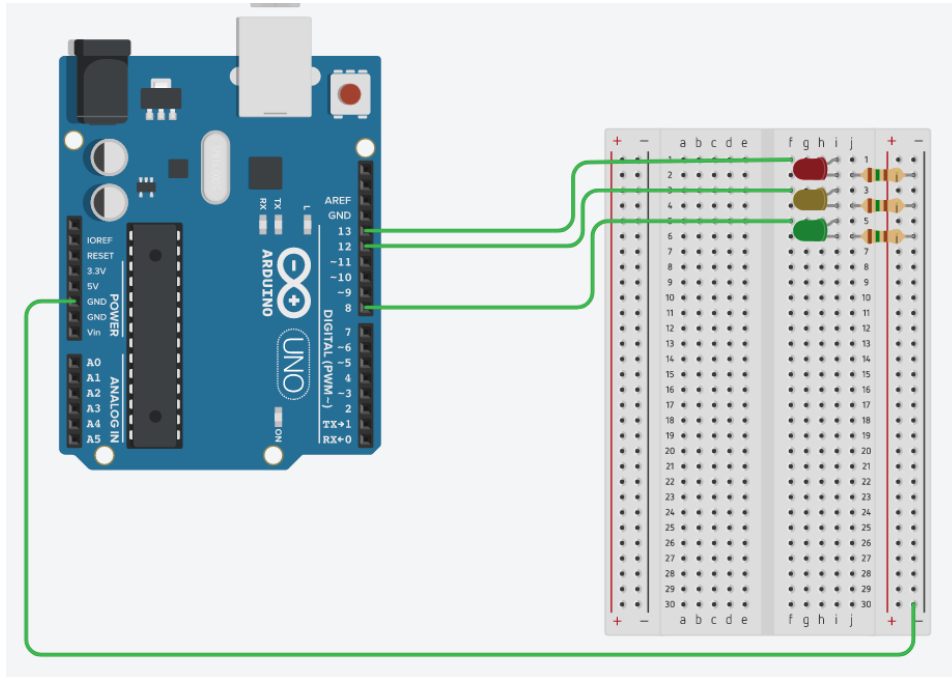


Abbildung 2: Implementiert Stromkreis von Aufgabe 1

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 4 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 5: 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

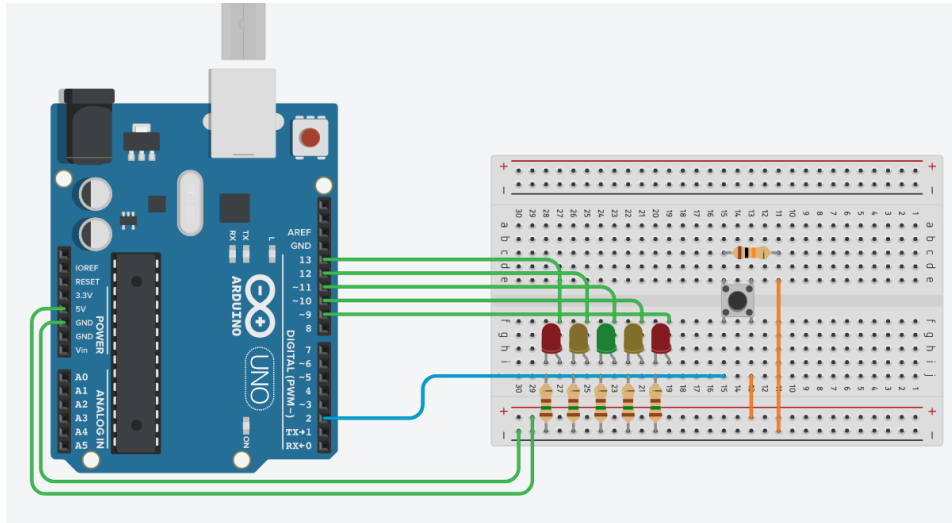


Abbildung 3: Implementierter Stromkreis von Aufgabe 2

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.

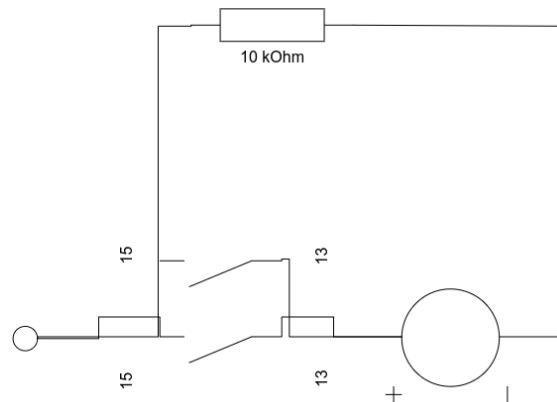


Abbildung 4: Schaltung des Tasters mit Pull-Up Widerstand

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.

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.

```
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 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.

```
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 5.2 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..

```

1 void setup() {
2   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

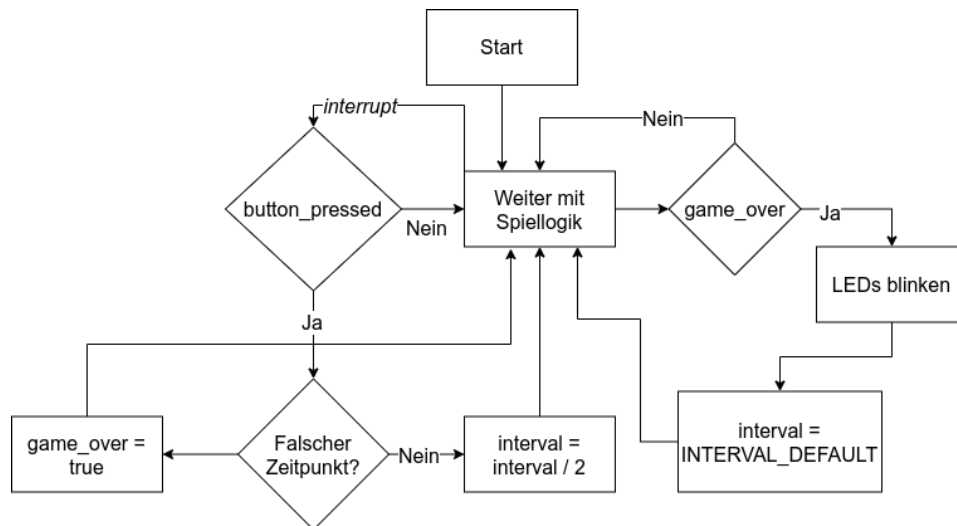


Abbildung 5: Annäherung des Spielverlaufs

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

```

1 int i;
2 for (i = 0; i < NUM_PINS; i++) {
3     pinMode(PINS[i], OUTPUT);
4 }

```

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.

```

1 pinMode(PIN_BUTTON, INPUT_PULLUP);
2
3 attachInterrupt(
4     digitalPinToInterrupt(PIN_BUTTON),
5     on_button_change,
6     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* bedeutet, 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 verschiedenen 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 Beginn 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 der 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(...)* eingeschaltet.

```

1 void on_button_change() {
2     int button_state = digitalRead(PIN_BUTTON);
3     if (button_state == HIGH) {
4         on_button_rising();
5     } else {
6         on_button_falling();
7     }
8 }
9
10 void on_button_rising() {
11     button_pressed = true;
12
13     if (pin_index == WINNING_PIN_INDEX &&
14         !game_over) {
15         interval = interval / 2;
16     } else {
17         game_over = true;
18     }
19 }
20
21 void on_button_falling() {
22     button_pressed = false;
23 }

```

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()* verwendet. 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 entsprechende Variable auf *true* gesetzt wird.

```
1 void blink_all() {
2     int j;
3     for (j = 0; j < 3; j++) {
4         int i;
5         write_to_all(HIGH);
6         delay(500);
7         write_to_all(LOW);
8         delay(500);
9     }
10 }
11
12 void write_to_all(int state) {
13     int i;
14     for (i = 0; i < NUM_PINS; i++) {
15         digitalWrite(PINS[i], state);
16     }
17 }
18
19 void increase_pin_index() {
20     pin_index++;
21     if (pin_index >= NUM_PINS){
22         pin_index = 0;
23     }
24 }
```

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.

```
1 void loop()
2 {
3     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 eingeschaltet. 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

```

```

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 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 Laufflicht 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

6 Aufgabe 3 - Realisierung eines Logic Analyzers

Es ist eine Schaltung zu entwerfen, welche Logikbausteine erkennt. Da bei zwei Signalen a und b als Eingänge je nach Baustein, bzw. Gatter, ein anderes Signal y am Ausgang anliegt, kann ein unbekanntes Gatter analysiert und benannt werden.

6.1 Materialien

Tabelle 6: Aufgabe 3 - Verwendete Materialien

Bezeichnung	Eigenschaften	Menge
Widerstand	150 Ω	2
	Braun - Grün - Braun - Gold	
74HC00	NAND	1
74HC02	NOR	1
74HC08	AND	1
74HC32	OR	1
74HC86	XOR	1
Mikrocontroller	Arduino Uno R3	1

6.2 Vorbereitung

Zur Vorbereitung dieser Aufgabe, müssen die Wahrheitstabellen der zu untersuchenden Funktionen bekannt sein. Dafür werden die Tabellen für die verwendeten Funktionen aufgestellt. Es gilt die Annahme, dass die Funktionen bekannt sind und werden daher nicht näher erklärt. Eine Erklärung der logischen Funktionen würde über den Umfang dieses Protokolls hinaus gehen. Die Tabellen können in den Tabellen 7 angelesen werden.

6.3 Praktikumsaufgabe

Für die Aufgabe wurde nun die in Abbildung 6 gezeigte Schaltung implementiert. Die Pins 13 und 12 bilden hierfür jeweils die Eingänge für das Logikgatter. Der Pin 8 liest dann den Ausgang aus. Die Widerstände von 150 Ω wurden verwendet, um die Logikbausteine vor zu hohen Stromflüssen

Tabelle 7: Wahrheitstabellen verwendeter Logikfunktionen

AND			OR			NAND		
A	B	Y	A	B	Y	A	B	Y
0	0	0	0	0	0	0	0	1
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

NOR			XOR		
A	B	Y	A	B	Y
0	0	1	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	0	1	1	0

zu schützen. Mittels des angeschlossenen Multimeters kann der Ausgang manuell ausgelesen werden und das Ergebnis des Programmcodes überprüft werden.

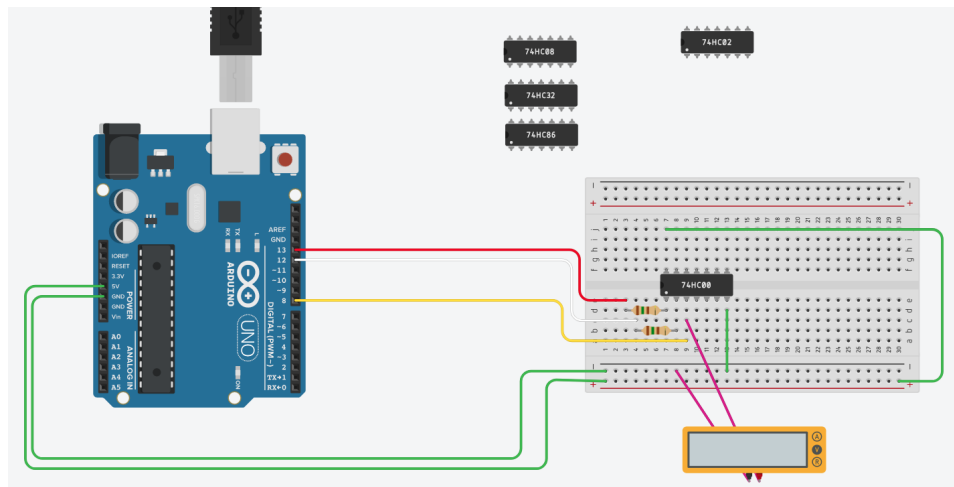


Abbildung 6: Implementierter Stromkreis von Aufgabe 3

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 /*

```

2 (Input1, Input2) => [Index]
3 (0, 0) => [0]
4 (0, 1) => [1]
5 (1, 0) => [2]
6 (1, 1) => [3]
7 */
8 bool TRUTH_TABLE[] = { false, false, false, false };

```

Listing 18: Variable für die Wahrheitstabelle

In Listing 18 wird ein Array angelegt, welches die Ergebnisse der Evaluierung speichert. Im Kommentar sind die Abbildungen der Eingänge auf den jeweiligen Index zu sehen. D.h., werden beide Eingänge auf 0, bzw. *LOW*, geschaltet, so wird das Ergebnis an Index 0 gespeichert.

```

1 digitalWrite(PIN_OUTPUT_1, LOW);
2 digitalWrite(PIN_OUTPUT_2, LOW);
3
4 delay(2000);
5 TRUTH_TABLE[0] = digitalRead(PIN_INPUT) == HIGH;
6
7 digitalWrite(PIN_OUTPUT_1, LOW);
8 digitalWrite(PIN_OUTPUT_2, HIGH);
9
10 delay(2000);
11 TRUTH_TABLE[1] = digitalRead(PIN_INPUT) == HIGH;
12
13 digitalWrite(PIN_OUTPUT_1, HIGH);
14 digitalWrite(PIN_OUTPUT_2, LOW);
15
16 delay(2000);
17 TRUTH_TABLE[2] = digitalRead(PIN_INPUT) == HIGH;
18
19 digitalWrite(PIN_OUTPUT_1, HIGH);
20 digitalWrite(PIN_OUTPUT_2, HIGH);
21
22 delay(2000);
23 TRUTH_TABLE[3] = digitalRead(PIN_INPUT) == HIGH;

```

Listing 19: Ausführen der Logikfunktionen

Der Code in Listing 19 setzt nacheinander die Eingänge des Gatters auf die Werte der Wahrheitstabellen. D.h., es werden beide Eingänge auf 0 gesetzt und das Ergebnis im Array gespeichert. Danach wird ein Eingang auf 1 und der andere auf 0 gesetzt und das Ergebnis wieder im Array gespeichert, u.s.w..

Zwischen den einzelnen Schaltungen der Eingänge wird mittels Aufruf von

delay(2000) zwei Sekunden lang gewartet. Dieser Aufruf ist dazu da, um ein manuelles Ablesen des Multimeters zu ermöglichen und ist für die korrekte Ausführung des Programms nicht notwendig.

```
1 Serial.println("");
2 Serial.println("|---|---|---|");
3 Serial.println("| a | b | y |");
4 Serial.println("|---|---|---|");
5
6 Serial.print("| 0 | 0 | ");
7 Serial.print(TRUTH_TABLE[0]);
8 Serial.println(" |");
9
10 Serial.print("| 0 | 1 | ");
11 Serial.print(TRUTH_TABLE[1]);
12 Serial.println(" |");
13
14 Serial.print("| 1 | 0 | ");
15 Serial.print(TRUTH_TABLE[2]);
16 Serial.println(" |");
17
18 Serial.print("| 1 | 1 | ");
19 Serial.print(TRUTH_TABLE[3]);
20 Serial.println(" |");
21 Serial.println("|---|---|---|");
22 Serial.println("");
```

Listing 20: Ausgabe der Wahrheitstabelle

Die Ergebnisse die in der Variable aus Listing 18 gespeichert wurden, werden in Listing 20 ausgegeben. Diese Ausgabe wird gemacht, um eine manuelle Überprüfung des Ergebnisses machen zu können. D.h., sie wird gemacht um ein Debuggen des Codes zu ermöglichen, ist aber für eine korrekte Ausführung des Programms nicht notwendig. Das Programm gibt hier die resultierende Wahrheitstabelle, in der Form einer Tabelle wie in Tabellen 7 dargestellt, aus.

```
1 if (!TRUTH_TABLE[0]
2 && !TRUTH_TABLE[1]
3 && !TRUTH_TABLE[2]
4 && TRUTH_TABLE[3]) {
5     Serial.println("AND - Gate");
6 } else if (!TRUTH_TABLE[0]
7 && TRUTH_TABLE[1]
8 && TRUTH_TABLE[2]
9 && TRUTH_TABLE[3]) {
10     Serial.println("OR - Gate");
11 } else if (TRUTH_TABLE[0]
```

```

12 && TRUTH_TABLE[1]
13 && TRUTH_TABLE[2]
14 && !TRUTH_TABLE[3]) {
15     Serial.println("NAND - Gate");
16 } else if (TRUTH_TABLE[0]
17 && !TRUTH_TABLE[1]
18 && !TRUTH_TABLE[2]
19 && !TRUTH_TABLE[3]) {
20     Serial.println("NOR - Gate");
21 } else if (!TRUTH_TABLE[0]
22 && TRUTH_TABLE[1]
23 && TRUTH_TABLE[2]
24 && !TRUTH_TABLE[3]) {
25     Serial.println("XOR - Gate");
26 } else {
27     Serial.println("Unrecognized Gate");
28 }

```

Listing 21: Evaluierung der Ergebnisse

Die Ergebnisse werden in Listing 21 evaluiert. Je nachdem welche Ausgaben der Logikbaustein getätigt hat, kann nun die entsprechende Funktion gefunden werden. Dadurch wird das Ergebnis mit den Ergebnissen aus den Tabellen 7 verglichen. Gleichen die Ergebnisse denen der Y-Spalte einer Tabelle, so wurde das jeweilige Gatter erkannt und der Name wird ausgegeben. Sind die Ergebnisse nicht zuzuordnen, so wird die Meldung „Unrecognized Gate“, zu Deutsch „Unbekanntes Gatter“, ausgegeben.

```

1 void loop()
2 {
3 }

```

Listing 22: Programmschleife

Die Programmschleife in Listing 22 ist leer, da der Code für jedes Logikgatter nur einmal ausgeführt werden muss. Da für jedes Logikgatter die Schaltung umgebaut werden muss, sprich, es muss das Gatter ausgetauscht werden, ist das Ausführen des Codes in einer Schleife nicht zielführend.

```

1 const int PIN_OUTPUT_1 = 13;
2 const int PIN_OUTPUT_2 = 12;
3 const int PIN_INPUT = 8;
4
5 /*
6 (Input1, Input2) => [Index]
7 (0, 0) => [0]
8 (0, 1) => [1]

```

```

9  (1, 0) => [2]
10 (1, 1) => [3]
11 */
12 bool TRUTH_TABLE[] = { false, false, false, false };
13
14 void setup()
15 {
16     Serial.begin(9600);
17
18     pinMode(PIN_OUTPUT_1, OUTPUT);
19     pinMode(PIN_OUTPUT_2, OUTPUT);
20     pinMode(PIN_INPUT, INPUT);
21
22     digitalWrite(PIN_OUTPUT_1, LOW);
23     digitalWrite(PIN_OUTPUT_2, LOW);
24
25     delay(2000);
26     TRUTH_TABLE[0] = digitalRead(PIN_INPUT) == HIGH;
27
28     digitalWrite(PIN_OUTPUT_1, LOW);
29     digitalWrite(PIN_OUTPUT_2, HIGH);
30
31     delay(2000);
32     TRUTH_TABLE[1] = digitalRead(PIN_INPUT) == HIGH;
33
34     digitalWrite(PIN_OUTPUT_1, HIGH);
35     digitalWrite(PIN_OUTPUT_2, LOW);
36
37     delay(2000);
38     TRUTH_TABLE[2] = digitalRead(PIN_INPUT) == HIGH;
39
40     digitalWrite(PIN_OUTPUT_1, HIGH);
41     digitalWrite(PIN_OUTPUT_2, HIGH);
42
43     delay(2000);
44     TRUTH_TABLE[3] = digitalRead(PIN_INPUT) == HIGH;
45
46     Serial.println("");
47     Serial.println("|---|---|---|");
48     Serial.println("| a | b | y |");
49     Serial.println("|---|---|---|");
50
51     Serial.print("| 0 | 0 | ");
52     Serial.print(TRUTH_TABLE[0]);
53     Serial.println(" |");
54
55     Serial.print("| 0 | 1 | ");
56     Serial.print(TRUTH_TABLE[1]);
57     Serial.println(" |");

```



```

58
59 Serial.print("| 1 | 0 | ");
60 Serial.print(TRUTH_TABLE[2]);
61 Serial.println(" |");
62
63 Serial.print("| 1 | 1 | ");
64 Serial.print(TRUTH_TABLE[3]);
65 Serial.println(" |");
66 Serial.println("|---|---|---|");
67 Serial.println("");
68
69 Serial.println("Result: ");
70
71 if (!TRUTH_TABLE[0]
72     && !TRUTH_TABLE[1]
73     && !TRUTH_TABLE[2]
74     && TRUTH_TABLE[3]) {
75     Serial.println("AND - Gate");
76 } else if (!TRUTH_TABLE[0]
77     && TRUTH_TABLE[1]
78     && TRUTH_TABLE[2]
79     && TRUTH_TABLE[3]) {
80     Serial.println("OR - Gate");
81 } else if (TRUTH_TABLE[0]
82     && TRUTH_TABLE[1]
83     && TRUTH_TABLE[2]
84     && !TRUTH_TABLE[3]) {
85     Serial.println("NAND - Gate");
86 } else if (TRUTH_TABLE[0]
87     && !TRUTH_TABLE[1]
88     && !TRUTH_TABLE[2]
89     && !TRUTH_TABLE[3]) {
90     Serial.println("NOR - Gate");
91 } else if (!TRUTH_TABLE[0]
92     && TRUTH_TABLE[1]
93     && TRUTH_TABLE[2]
94     && !TRUTH_TABLE[3]) {
95     Serial.println("XOR - Gate");
96 } else {
97     Serial.println("Unrecognized Gate");
98 }
99 }
100
101 void loop()
102 {
103 }

```

Listing 23: Vollständiger Programmcode der Aufgabe 3

6.4 Fehlerdiskussion

Während des Testens des Programms konnten alle Gatter bis auf das NOR-Gatter nicht erkannt werden. Der Grund dafür war, dass bei gleicher Ausrichtung des NOR-Gatters, die Ein- und Ausgänge vertauscht waren. D.h., die Reihenfolge der Pins der Bausteine war "Eingänge 1 - Eingang 2 - Ausgang", außer am NOR-Gatter. Hier ist die Reihenfolge "Ausgang - Eingang 1 - Eingang 2". Um das NOR-Gatter richtig zu erkennen, muss daher die Schaltung entsprechend angepasst werden. Da die Schaltung theoretisch ein unbekanntes Gatter erkennen soll, müsste daher die Schaltung geändert werden, wenn das Gatter nicht erkannt wurde. Wird das Gatter danach immer noch nicht erkannt, handelt es sich tatsächlich um ein unbekanntes Gatter, ansonsten um ein NOR-Gatter.

6.5 Zusammenfassung

7 Aufgabe 4 - Wertbestimmung von Widerstand und Kondensator

In dieser Aufgabe geht es um Widerstände und Kondensatoren. Diese sind Grundbausteine der Elektrotechnik. Neben diversen Berechnungen sind folgende Fragen zu beantworten.

- Können Unterschiede zwischen Messungen am Mikrocontroller und manuellen Methoden gefunden werden?
- Wenn Ja, welche und warum?
- Mit welchen Prinzipien und Überlegungen wurden die Widerstände gemessen?
- Welche Abweichungen ergeben sich beim Messen bekannter Kondensatoren?

7.1 Materialien

Tabelle 8: Aufgabe 4 - Verwendete Materialien

Bezeichnung	Eigenschaften	Menge
Widerstand	10k Ω Braun - Schwarz - Orange - Gold	2
Widerstand	unbekannt	k.A.
Mikrocontroller	Arduino Uno R3	1

In der Tabelle 8 sind unbekannte Widerstände in nicht angegebener Menge gelistet. Damit sind Testwiderstände gemeint, welche durch das Testen an der Schaltung gemessen werden können. Beim Versuch an dieser Schaltung sind die Widerstandswerte normalerweise bekannt, um die Richtigkeit der Schaltung zu bestätigen. Die Werte und Anzahl der verwendeten Widerstände sind jedoch nicht relevant.

7.2 Vorbereitung

7.2.1 Aufgabe 1

In dieser Aufgabe sind Widerstände an Hand ihrer Farbcodes zu bestimmen. Drei Widerstände sind gegeben. Anhand der Tabelle in Sektion 2 kann der Wert bestimmt werden.

Tabelle 9: Widerstandswerte

Ring 1	Ring 2	Ring 3	Ring 4	Widerstandswert	Toleranz
Gelb	Violett	Rot	Gold	$4,7k\Omega$	$\pm 5\%$
Rot	Weiß	Grün	Gold	$3,9M\Omega$	$\pm 5\%$
Blau	Grau	Rot	Silber	$6,8k\Omega$	$\pm 10\%$

7.2.2 Aufgabe 2

7.3 Praktikumsaufgabe

7.4 Fehlerdiskussion

7.5 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".