

Graz University of Technology

Institut für Materialphysik der Technischen Universität Graz

LABORÜBUNGEN: ELEKTRONIK UND COMPUTERUNTERSTÜTZTE MESSTECHNIK

Übungsnummer: 5

Übungstitel: Mikrocontroller

Gruppennummer: 7

Name: Maximilian PHILIPP

Name: Michael HINTERLEITNER

Mat. Nr.: 11839611

Mat. Nr.: 12002411

Datum der Übung: 22.06.2022

Sommersemester 2022

Inhaltsverzeichnis

1	Aufgabenstellung	3
2	Vorbereitung	8
3	Grundlagen	11
4	Versuchsdurchführung	11
4.1	Ampelschaltung	12
4.1.1	Simulation	12
4.1.2	Steckboard	13
4.2	Reaktionsspiel	15
4.2.1	Simulation	15
4.2.2	Steckboard	16
5	Auswertung	17
6	Diskussion und Zusammenfassung	18
6.1	Diskussion	18
6.2	Zusammenfassung	18
7	Appendix	19
7.1	Ampel Code	19
7.2	Game Code	22

Elektronik und computerunterstützte Messtechnik

LU PHY.M20 (SS 2022)

Übung 5: Mikrocontroller

1. Ampelschaltung
2. Reaktionsspiel

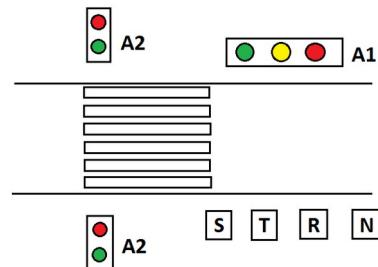
Abgabe der Vorbereitung: bis Freitag, 17. Juni 2022, 12:00 Uhr im TeachCenter

Praktikumstermin: KW25 (22.06 bis 24.06.2020)

Anmerkung: Zur Vorbereitung können eigene Arduinoprogramme mit dem Simulator „SimulIDE“ getestet werden. Dieser ist im TeachCenter unter „Laborübung“ verlinkt.

Aufgabe 1: Ampelschaltung mit Fußgängerübergang

Das Ziel der Übung ist die Simulation einer geregelten Ampelschaltung an einem Fußgängerübergang (s. Abb. rechts) mit Hilfe eines Arduino UNO Boards. Programmiert wird in der Arduino Entwicklungsumgebung.



Grundzustand: Der Verkehr fließt auf der Straße, Ampel A1 ist auf Grün und Ampel A2 ist auf Rot geschaltet.

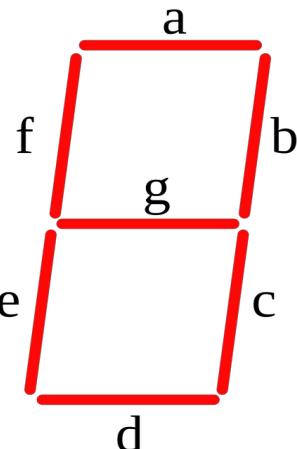
Schaltzustand: Die Taste „S“ wird gedrückt und die Ampeln wechseln das Signal. Dabei sollen die in Österreich üblichen Ampel Schaltregeln eingehalten werden. Nach 5s Grünphase der Ampel A2 soll in den Grundzustand zurückgewechselt werden.

Nachtfunktion: Durch einen Sensor „N“ (simuliert durch einen Photowiderstand/LDR) soll die Ampelschaltung beim Unterschreiten eines bestimmten Schwellwertes in einen Nachtmodus wechseln. Dabei soll die Ampel A1 auf gelb blinkend umschalten und die Ampel A2 ausgeschaltet werden.

Aufgabe 2: Reaktionsspiel

In diesem Teil der Übung wird ein einfaches Reaktionsspiel mithilfe einer Siebensegmentanzeige und zweie Tastern realisiert.

Es soll immer nur ein Segment leuchten. Das leuchtende Segment soll entlang einer 8-er Kurve laufen, also $a \rightarrow b \rightarrow g \rightarrow e \rightarrow d \rightarrow c \rightarrow g \rightarrow f$ bzw. umgekehrt. Die Leuchtdauer der einzelnen Segmente beträgt zu Beginn des Spiels 500 ms und wird pro erzieltem Punkt kürzer (zB. $T_{on} = 500 \text{ ms} / (n_{\text{Gesamtpunkte}} + 1)$).



Wenn nun das Segment (a / d) leuchtet muss Spieler (1 / 2) seinen Taster drücken. Geschieht dies innerhalb der Leuchtdauer vom entsprechenden Segment erhält der Spieler einen Punkt und der Punktestand wird per UART ausgegeben. Mit jedem erzielten Punkt läuft das leuchtende Segment schneller im Kreis. Wird das Zeitfenster verpasst oder zu einem ganz falschen Zeitpunkt gedrückt ändert der Leuchtpunkt seine Richtung. Sind insgesamt 10 Punkte erreicht ist das Spiel vorbei, alle Segmente blinken nun im Gleichtakt und der Endpunkttestand wird nochmals per UART ausgegeben. Werden nun beide Taster zeitgleich gedrückt beginnt das Spiel erneut.

Zusatz: Wird das Zeitfenster zum drücken des Tasters verpasst verliert der Spieler einen Punkt. Die Leuchtdauer des Segmentes verändert sich nicht und auch die erzielten Gesamtpunkte werden dadurch nicht beeinflusst. (Bei einem Fehler im Spiel wäre der Endpunkttestand zB. 3:6).

Zusatz: Der Punktestand soll in auf einem monochromen LC-Display angezeigt werden (HD44780). Hierfür steht die Library 'LiquidCrystal' im Arduino Library Manager zur Verfügung. Das Display soll im 6-Pin Modus betrieben werden.

Informationen zur Library:

<https://www.arduino.cc/reference/en/libraries/liquidcrystal/>

Minmales bsp. Programm der Library ist hier enthalten:

<https://www.arduino.cc/reference/en/libraries/liquidcrystal/liquidcrystal/>

Vorbereitung

Für beide Aufgaben ist ein Programmablaufplan (PAP) nach DIN 66001 anzufertigen. Hierzu stehen dieses Mal verschieden digitale Helfer zu Verfügung, wie zum Beispiel die von uns empfohlenen Programme PAPDesigner oder yEd von YWORKS. Es ist darauf zu achten dass die Reihenfolge der verschiedenen Abfragen bzw. Zustände auch die Prioritäten/Hierarchien im Programm beeinflussen.

Die Zusatzaufgaben müssen nicht in der Vorbereitung berücksichtigt werden und werden nur bei ausreichender Zeit im Labor durchgeführt.

Des Weiteren müssen zwei Prinzipskizzen für Aufgabe 1 entworfen werden. Diese sind jedoch mit der Hand & Lineal zu zeichnen und wie üblich sind alle Anschlüsse zu bezeichnen.

1. Skizze Vorwiderstände:

Bei der Dimensionierung der Vorwiderstände für die LEDs ist zu beachten dass der Strom die 10 mA nicht überschreitet.

2. Skizze Nachtsensor:

Zur Realisierung des Nachtsensors muss eine geeignete Beschaltung des LDR skizziert werden, so dass eine lichtabhängige Spannung gemessen werden kann.

Programmierung:

Programmiert wird in der Arduino Entwicklungsumgebung. Die Programmiersprache ist C/C++. Eine „Language Reference“ für Arduino ist unter nachfolgendem Link verfügbar. <https://www.arduino.cc/reference/en>

Für die Programmierung der Ampelsteuerung ist es empfehlenswerteinen der zwei verfügbaren Interrupt Pins für den Taster „R“ zu verwenden. Die „delay“ Funktion kann verwendet werden solange die Funktion der Ampelsteuerung nicht eingeschränkt wird.

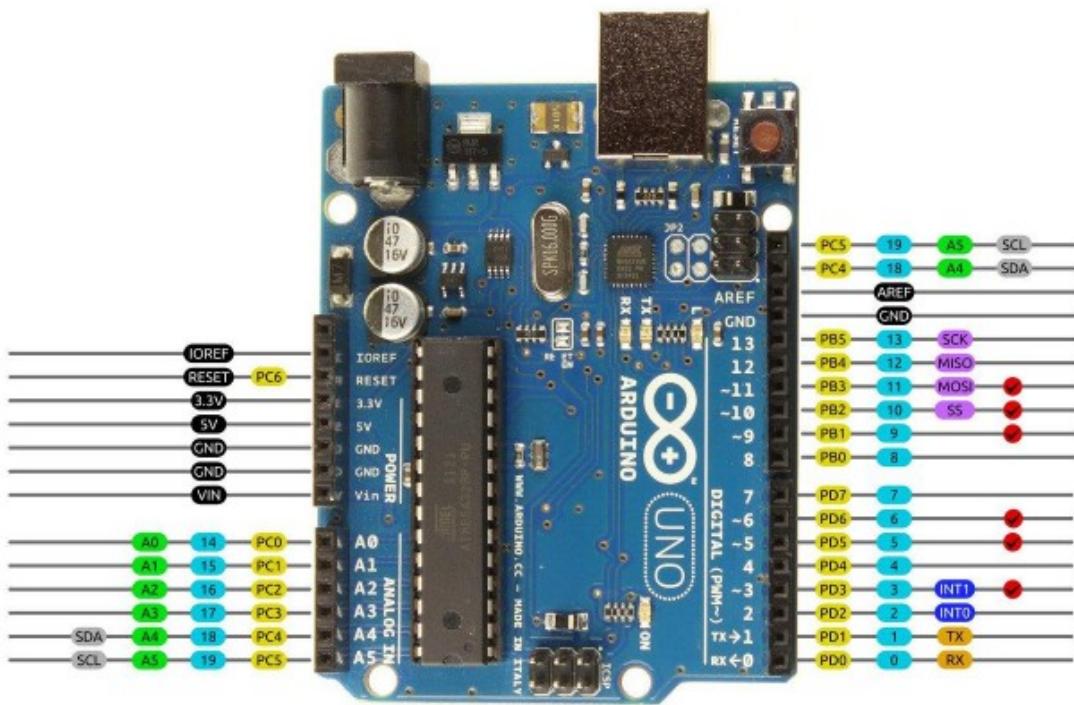
Coding Standard:

Zur besseren Lesbarkeit des Programms ist es notwendig einen „Coding Standard“ festzulegen der im gesamten Programm angewendet wird. Zum Beispiel:

- Konstanten ALL_UPPERCASE_WITH_UNDERSCORES
- Variablen firstWordLowerCaseRestCapitalizedWithoutUnderscores
- Funktionen firstWordLowerCaseRestCapitalizedEndsWithUnderscore_
- Kommentare //comments are necessary

Auf eine sinnvolle und AUSREICHENDE Kommentierung des erstellten Codes ist zu achten!

Arduino Uno R3 Pinout



Ein paar nützliche Funktionen in der Arduino-Umgebung:

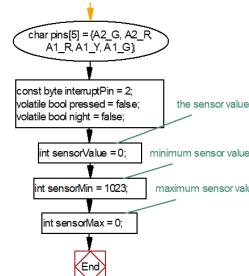
erzeugen des PWM-Signals	<code>analogWrite (pin, val);</code>
Seriellen Port verwenden	<code>Serial.begin(9600);</code>
Daten (Text) am Port schreiben	<code>Serial.print("Hallo World");</code>
Daten (Zahlen) am Port schreiben mit allen Nachkommastellen (bei Float/ Double)	<code>Serial.print(Variable, DEC);</code>
Wartefunktionen	<code>delay(val);</code> val in ms <code>delayMicroseconds(val);</code> in μ s
Einstellen der Pins	<code>pinMode(pin, Modus);</code>
Pin setzen	<code>digitalWrite(pin, Zustand);</code>

PAP Aufgabe 1

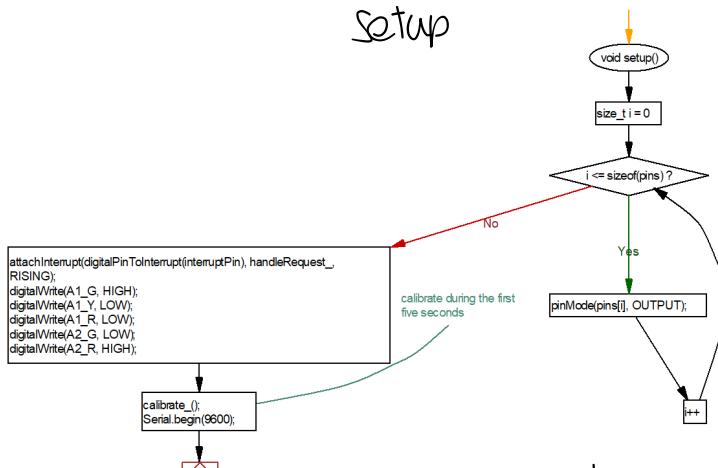
Maximilian PHILIPP 11839611 Mittwoch NM
Michael Hinterleitner 12002411 GRUPPE 7

Globals

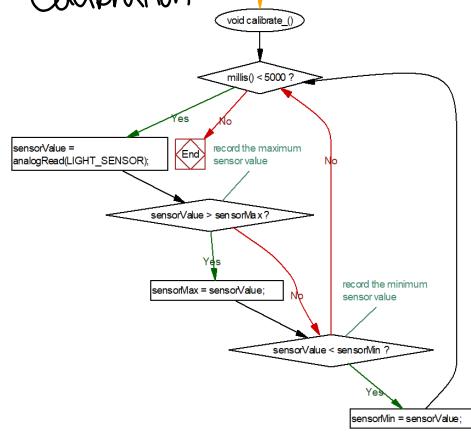
```
#define A2_G 10
#define A2_R9
#define A1_R13
#define A1_Y12
#define A1_G11
#define LIGHT_SENSOR 19
#define NUM_WARN_BLINKS 4
```



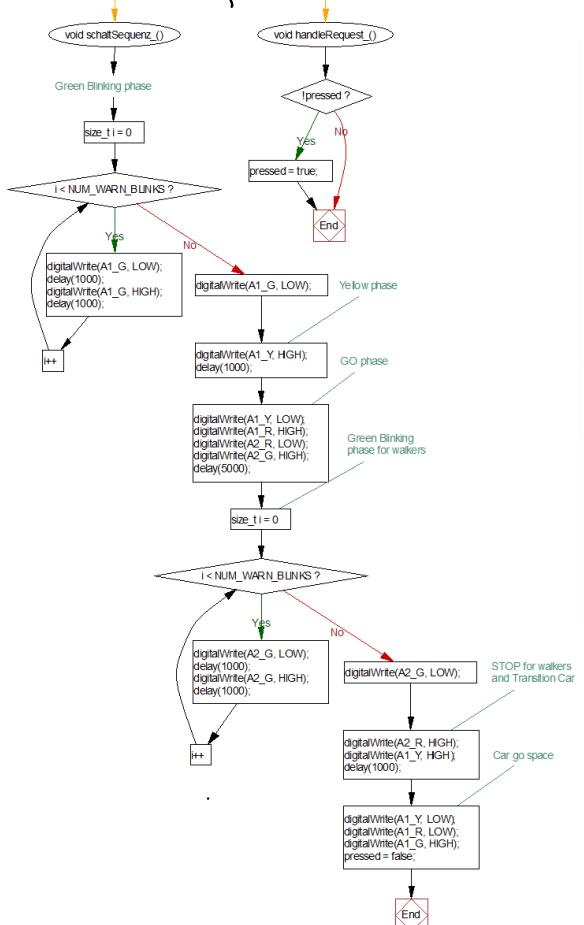
Setup



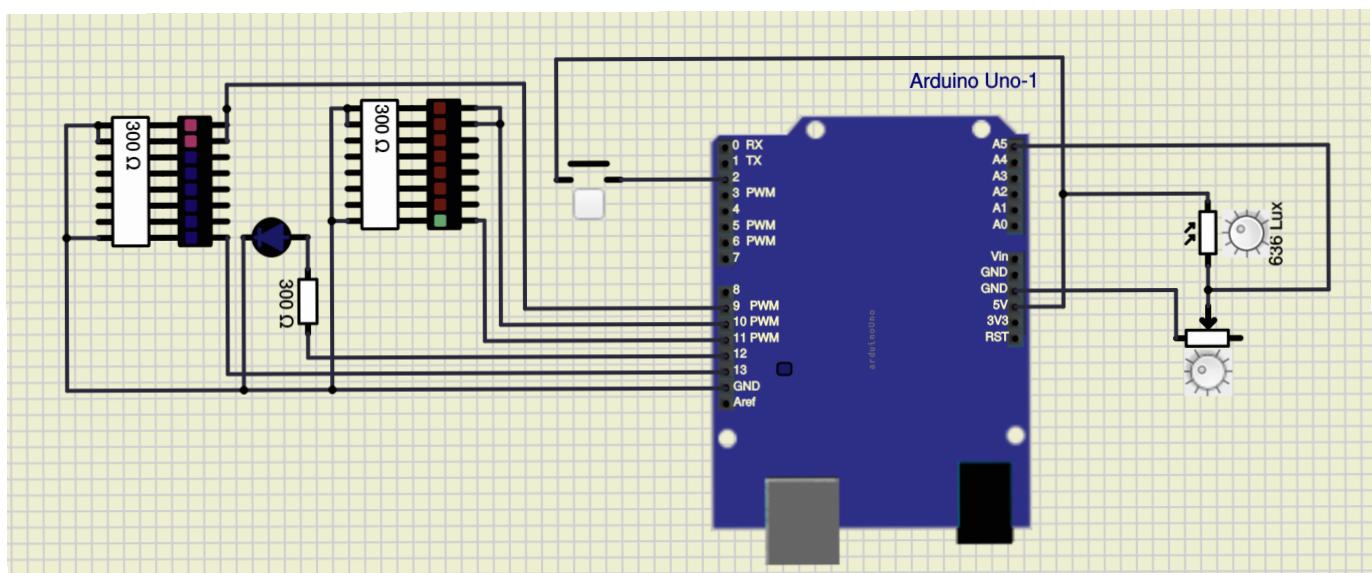
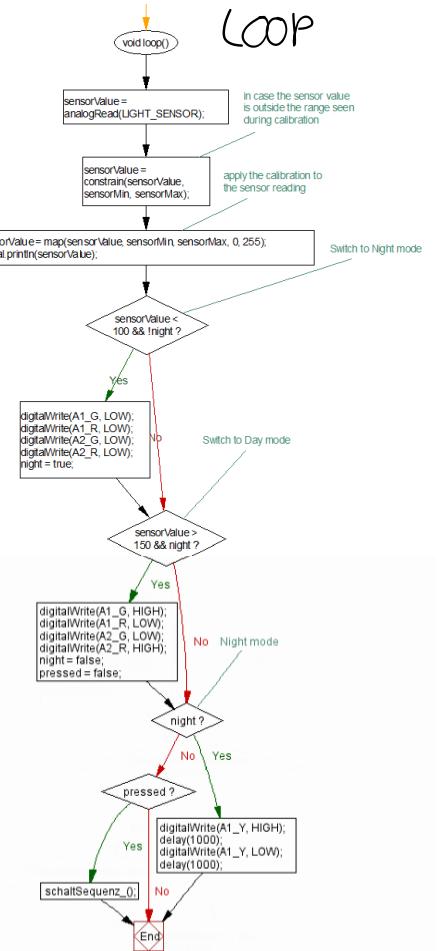
Calibration



Schaltseq.



intrp.

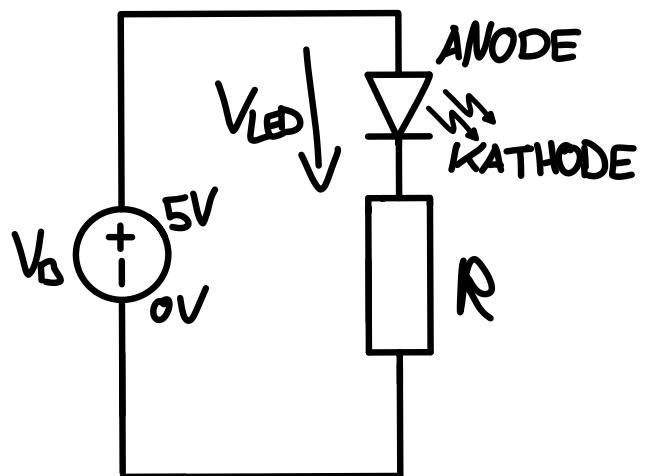


1. Skizze Vormwiderstände:

$$\text{Formel } R = \frac{V_B - V_{LED}}{I_{LED}}$$

	Yellow	Green	Red
Forward Voltage	2,15V	2,95V	1,8V
Vormwiderstand	285Ω	205Ω	320Ω
E12	270 + 15	180 + 27	270 + 56

Damit 10mA nicht überschritten wird
sollten die Widerstände größer als die
errechneten sein.



2. Skizze Nachtsensor

LDR variiert von $R(E_v)$ ca. 100Ω - $10M\Omega$ (viel Lux \rightarrow 0 Lux)

wir messen 0 - 5V im Analogpin 0 - 1023

Somit kann mit einem $10k\Omega$ Poti ein einstellbarer

Spannungsteiler gebaut werden. Mit dem Poti
lässt sich das Verhalten finetunen, so dass

bei $\sim 100\Omega$ beim LDR fast kein Spannungsabfall ist
 $I \cdot R_{LDR} \ll I \cdot R \approx 5V$ und bei $\sim 10M\Omega$ beim LDR

quasi die komplette Spannung abfällt also

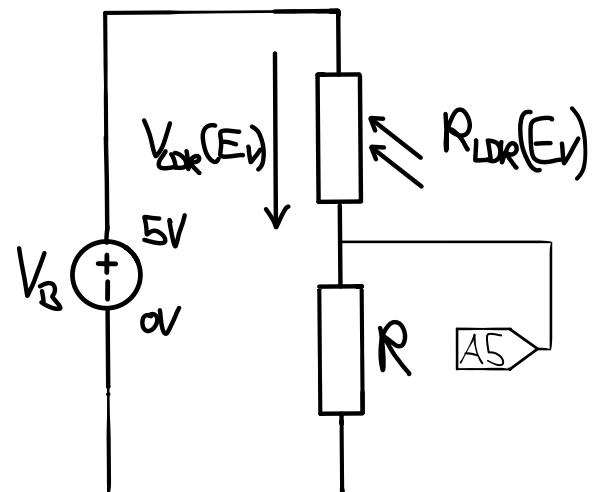
$$5V \approx I \cdot R_{LDR} \gg I \cdot R$$

$$\Rightarrow R_{LDR\max} \gg R \wedge R_{LDR\min} \ll R$$

$$1 \gg 10^{-3} \quad 1 \gg 10^{-2} \text{ mit } R = 10^4\Omega$$

$$\Rightarrow R \approx 10k\Omega \text{ gute Wahl da jedoch nie } I \cdot R_{LDR\min} = 0$$

& $I \cdot R_{LDR\max} = 5V$ wird in der Software so kalibriert, dass der minimale Wert wirklich 0 ist & der maximale 255. Damit
Sinnvolles Schalten in den Nachtmodus ermöglicht wird.

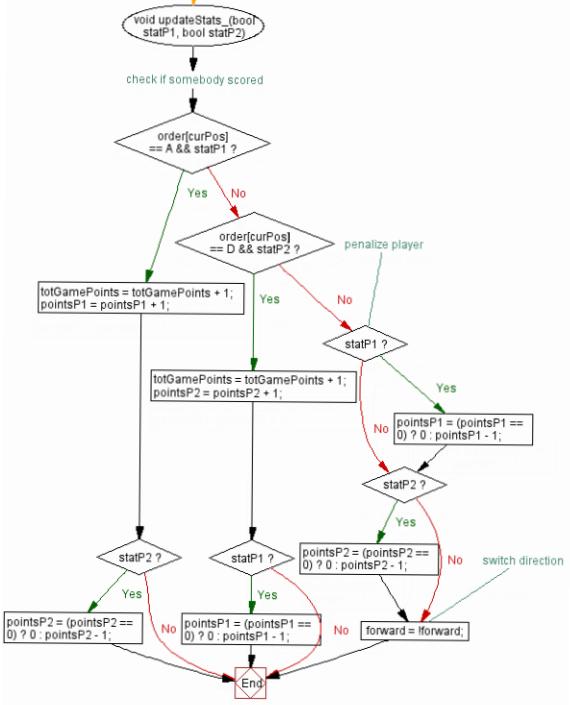


PAP Aufgabe 2

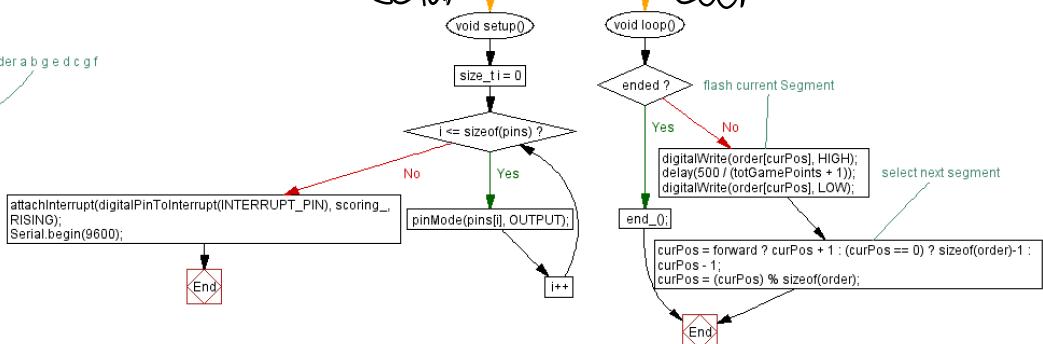
Globals

```
#define A13
#define B12
#define C11
#define D10
#define E9
#define F8
#define G7
#define POINTS_TO_END_THE_GAME 10
char order[8] = (A, B, G, E, D, C, G, F);
order a b g e d c g f
char pins[7] = (A, B, C, D, E, F, G);
const byte INTERRUPT_PIN = 2;
const byte P1_PIN = 5;
const byte P2_PIN = 6;
size_t totGamePoints = 0;
size_t pointsP1 = 0;
size_t pointsP2 = 0;
volatile bool ended = false;
volatile bool forward = true;
volatile size_t curPos = 0;
```

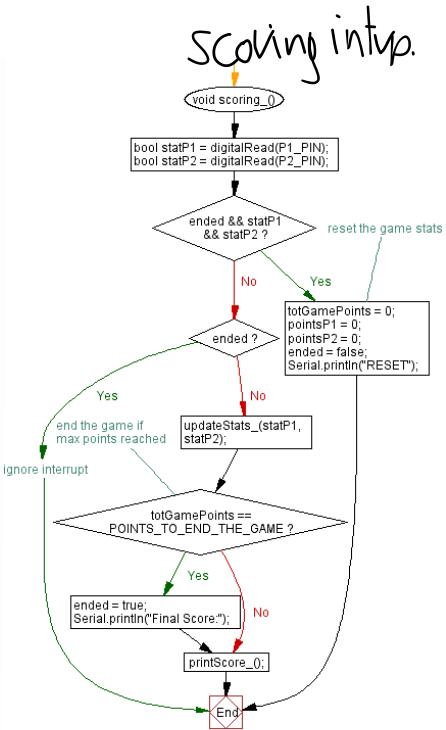
update Score



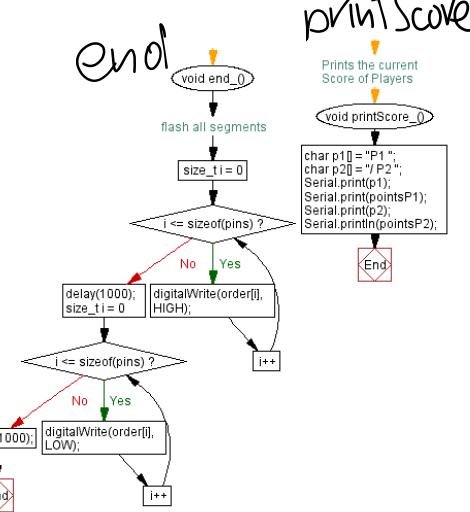
Setup



Scoring intup.



end_0



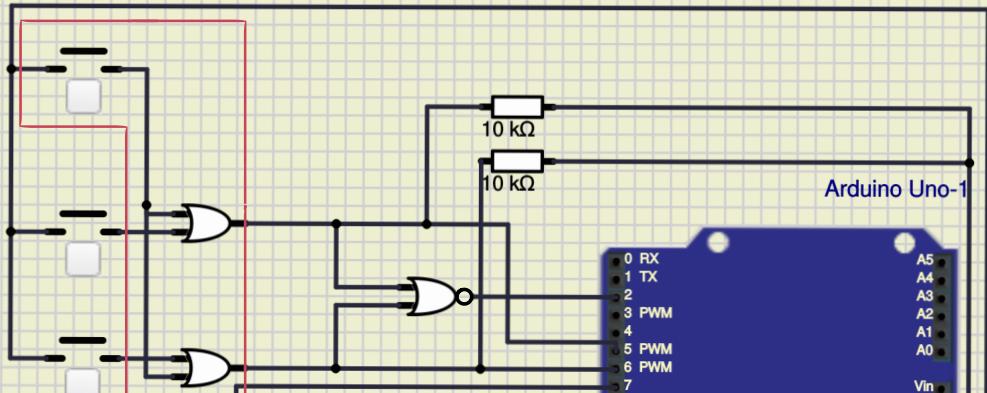
printScore

```
Prints the current Score of Players
void printScore_0() {
    char p1[] = "P1 ";
    char p2[] = "/P2 ";
    Serial.print(p1);
    Serial.print(pointsP1);
    Serial.print(p2);
    Serial.print(pointsP2);
}
End
```

Nur für
Simulation
Taster & Gatter
können entfernt
werden.

Reset

Player 1 A
Player 2 D



Arduino Uno-1

300 Ω

Pallup Intern

3 Grundlagen

Die Grundlage für diese Übung bietet die Plattform Arduino mit deren Soft- und Hardware. Die Hardware, also das Mikrocontroller-Board (*Arduino Uno R3*), ist mitsamt der zugehörigen Pinbelegung in Abschnitt 1 (Aufgabenstellung) dargestellt. Neben der Rechenfunktion einer CPU, was alleinstehend einem Mikroprozessor entspricht, verfügt er wie andere Mikrocontroller über weitere Elemente wie Speicher respektive einem eigenen Schwingquarz zur Taktung, was ihn zu einem kleinen, eigenständigen Computer macht. Die integrierte Entwicklungsumgebung (IDE) der Arduino-Software erlaubt freien Zugang (open-source). Die Ansteuerung des Boards über die IDE erfolgt in einer C/C++ ähnlichen Programmiersprache.

Zur Auswertung sind keine Formeln notwendig, die in diesem Kapitel andererseits zu erwähnen wären.

4 Versuchsdurchführung

Die verwendeten Geräte sind Tabelle 1 zu entnehmen.

Tabelle 1: Tabelle der verwendeten Geräte
Geräteliste

Gerät/Bauelement	Typ
Mikrocontroller-Board	<i>Arduino Uno R3</i>
Widerstandsleiste	
7-Segment-Anzeige	
2x-NOR-Gatter	74LS02[1]

Entprellter Schalter Der entprellter Schalter wird als Signalgeber für die Schaltungen verwendet. Der Aufbau dieses Schalters ist in Abbildung 1 ersichtlich, jedoch ist noch der *GND* mit Ground und *VCC* mit 5 V zu beschalten, damit das Signal von den Schaltern für die jeweilige Betriebsart abgegriffen werden kann.

Jeder dieser Schalter hat hierbei eine Standard-*HIGH*- bzw. *LOW*-Betriebsart.

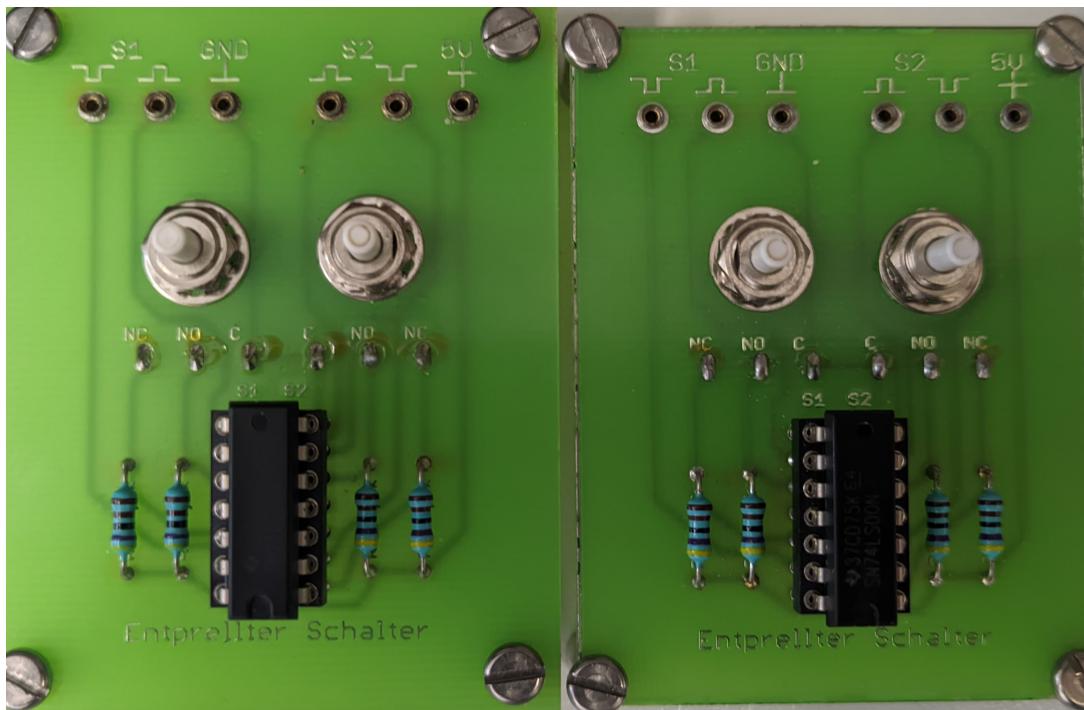


Abbildung 1: Dies sind die zwei entprellten Schalterplatinen mit je zwei Schaltern (S_1, S_2) die entweder mit Standard-HIGH oder -LOW verwendet werden können. Diese werden durch Beschalten des GND und der 5 V Betriebsspannung in Betrieb genommen

4.1 Ampelschaltung

4.1.1 Simulation

Da in der Vorbereitung der Programmablauf entworfen wurde, wurde zunächst nur das Programm geschrieben und in der Simulationssoftware *SimulIDE* simuliert. Da die Funktionstüchtigkeit in der Simulation gewährleistet wurde, konnte die Ampelschaltung aufgebaut werden. In folgender Abbildung 2 ist der Programmablauf nochmals ersichtlich. Des weiteren wurde der verwendete Code im Appendix 7.1 hinzugefügt.

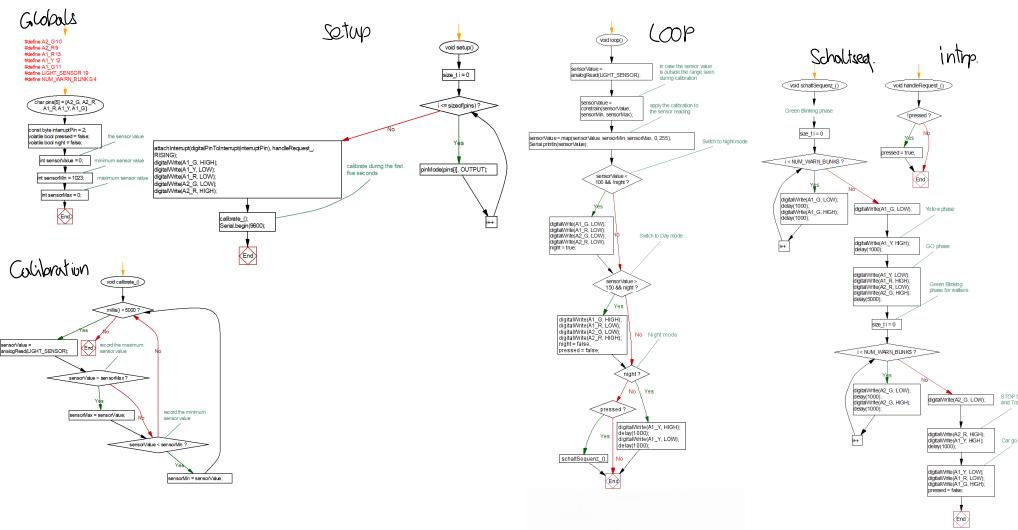


Abbildung 2: Diese Grafik beinhaltet den Programmablauf einer, nach der Aufgabenstellung entworfenen, Ampelschaltung mit einem Fußgängerübergang. Hier wurde der DIN 66001 verwendet, um den Programmablauf darzustellen. Falls der Plan zu klein ist, ist dieser in der Vorbereitung nochmals größer ersichtlich.

Beim Erstellen des Codes wurde auf die korrekte Beschaltung, wie in der kleinen österreichischen Ampelkunde beschrieben, geachtet.

4.1.2 Steckboard

Die Schaltung ist mittels der *SimulIDE* entworfen worden. In Abbildung 3 ist der Aufbau der simulierten Schaltung ersichtlich. Die verwendeten Widerstände sind in der Schaltung ersichtlich, wobei der Spannungsteiler aus dem LDR und einem Pull-Down-Widerstand besteht.

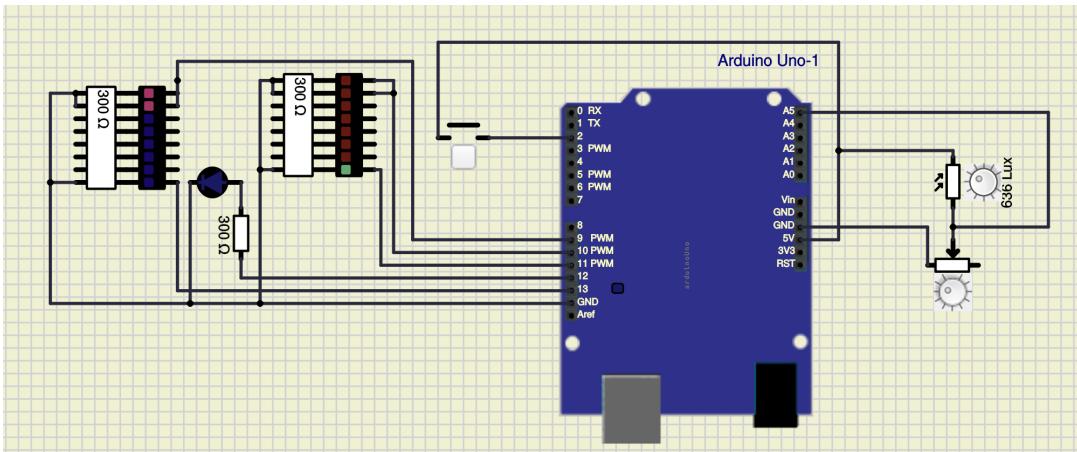


Abbildung 3: Dies ist der skizzierte Aufbau der Ampelschaltung, welche in der Aufgabenstellung beschrieben worden ist.

Mit diesem als Vorbild wurde die eigentliche Schaltung aufgebaut. Der Aufbau am Steckbrett ist in Abbildung 4 ersichtlich. Die Spezifikationen der Schaltung können dieser entnommen werden.

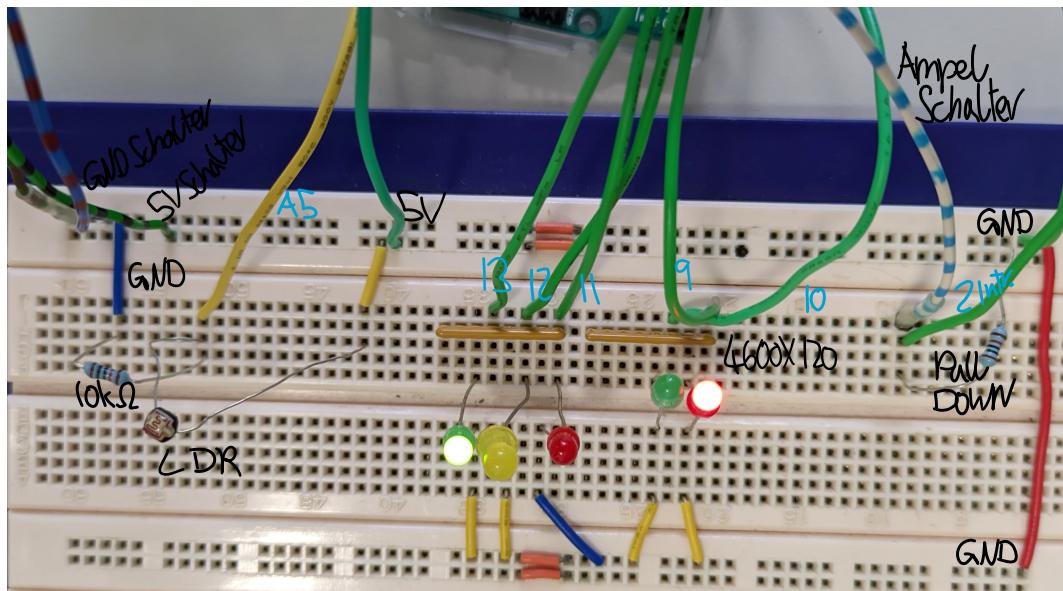


Abbildung 4: Diese Abbildung beinhaltet den Aufbau am Steckbrett zur Ampelschaltung, wie in Abbildung 3 ersichtlich. Die Beschriftungen in blau bezeichnen die am *Arduino Uno R3* verwendeten Pins.

4.2 Reaktionsspiel

4.2.1 Simulation

Da in der Vorbereitung der Programmablauf entworfen wurde, wurde zunächst nur das Programm geschrieben und in der Simulationssoftware *SimulIDE* simuliert. Da die Funktionstüchtigkeit in der Simulation gewährleistet wurde, konnte das Reaktionsspiel aufgebaut werden. In folgender Abbildung 5 ist der Programmablauf nochmals ersichtlich. Des Weiteren wurde der verwendete Code im Appendix 7.2 hinzugefügt.

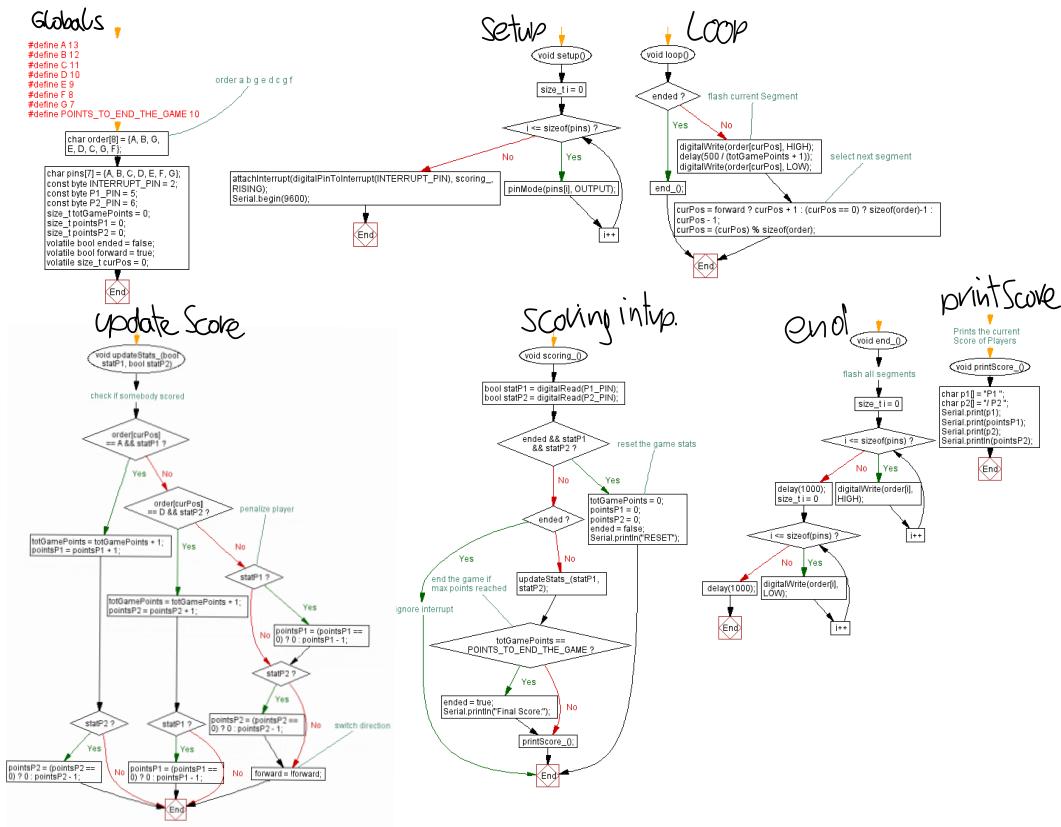


Abbildung 5: Diese Grafik beinhaltet den Programmablauf eines, nach der Aufgabenstellung entworfenen, Reaktionsspiels. Hier wurde der *DIN 66001* verwendet, um den Programmablauf darzustellen. Falls dieser zu klein sein sollte, ist dieser in der Vorbereitung nochmals größer ersichtlich.

4.2.2 Steckboard

Die Schaltung ist mittels der *SimulIDE* entworfen worden, in Abbildung 6 ist der Aufbau der simulierten Schaltung ersichtlich.

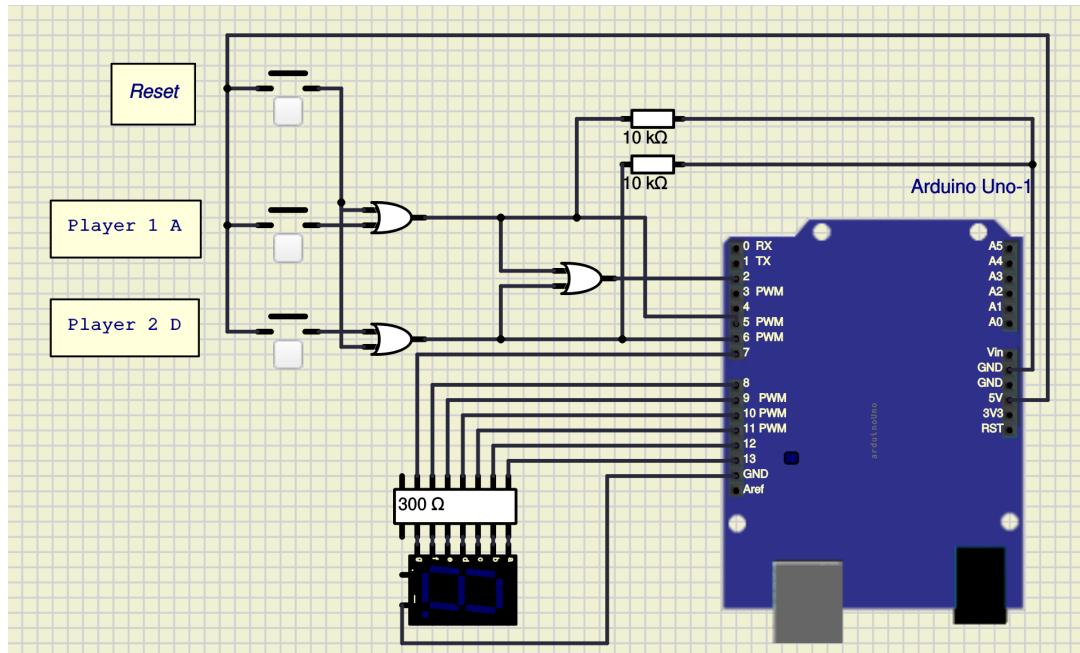


Abbildung 6: Dies ist der skizzierte Aufbau des Reaktionsspiels, welche in der Aufgabenstellung beschrieben worden ist.

Mit diesem als Vorbild wurde die eigentliche Schaltung aufgebaut. Der Aufbau am Steckbrett ist in Abbildung 7 ersichtlich. Die Spezifikationen der Schaltung können dieser ebenso entnommen werden.

Jedoch musste aus Verfügbarkeitsgründen ein NOR-Gatter statt einem OR-Gatter verwendet werden, wodurch im Code eine kleine Anpassung gemacht werden musste, dass nun mit einer *FALLING-edge* statt einer *RISING-edge* getriggert wurde. Dies ist im Code im Appendix ersichtlich.

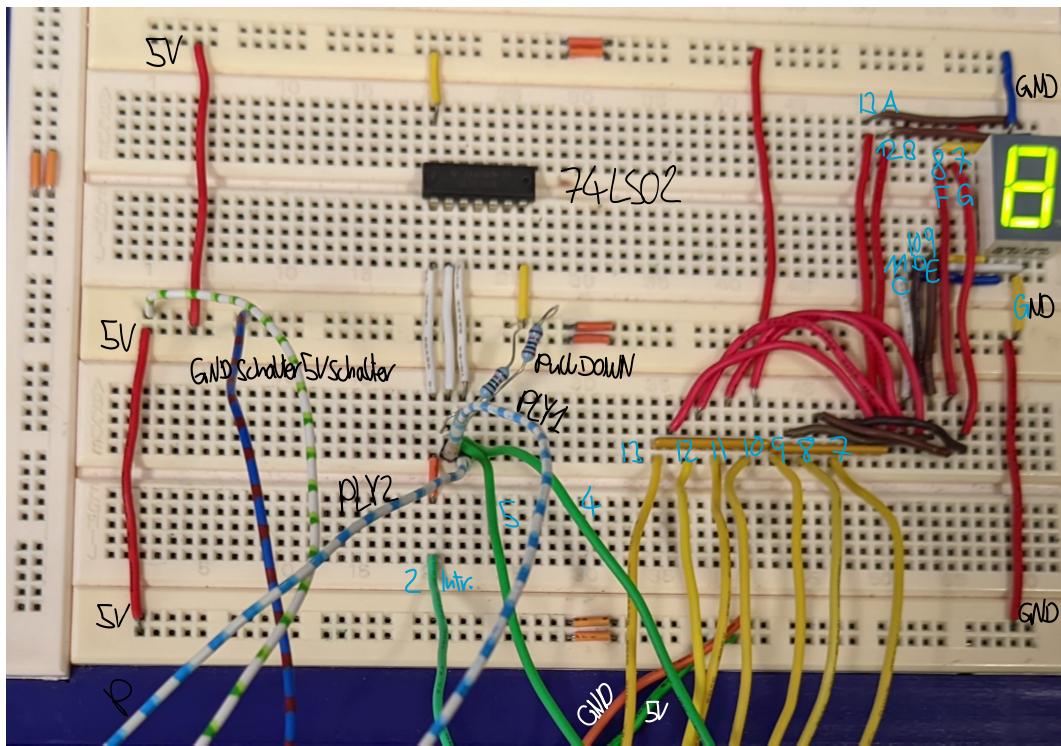


Abbildung 7: Diese Abbildung beinhaltet den Aufbau am Steckbrett zum Reaktionsspiel, wie in Abbildung 6 ersichtlich. Die Beschriftungen in blau bezeichnen die am *Arduino Uno R3* verwendeten Pins.

5 Auswertung

In diesem Protokoll ist keine Auswertung von Nöten, da die geforderten Resultate direkt aus den Ergebnissen der Laborübung folgen. Die Funktionstüchtigkeit konnte festgestellt werden indem, dass Spiel gespielt wurde und die Ampelschaltung in Tag- und Nachtbetrieb verwendet worden ist. Das gewollte Verhalten wurde von den Betreuern bestätigt. Hier sind Links zu Video der Schaltungen: hier die Ampel und hier das Reaktionsspiel.

6 Diskussion und Zusammenfassung

6.1 Diskussion

Die Simulation in *SimulIDE* wie auch der Aufbau am Steckbrett haben den zu erwartenden Abläufen, die in Abschnitt 1 (Aufgabenstellung) gefordert sind, entsprochen.

6.2 Zusammenfassung

Im Rahmen dieser Laborübung wurden eine Ampelschaltung und ein Reaktionsspiel mithilfe der Arduino-Plattform konzipiert, deren Funktionstüchtigkeit mit *SimulIDE* und am Steckbrett mit einem Arduino-Board verifiziert wurde.

7 Appendix

7.1 Ampel Code

```
1
2 #define A2_G 10
3 #define A2_R 9
4 #define A1_R 13
5 #define A1_Y 12
6 #define A1_G 11
7 #define LIGHT_SENSOR 19
8 #define NUM_WARN_BLINKS 4
9
10 char pins [5] = { A2_G, A2_R, A1_R, A1_Y, A1_G };
11 const byte interruptPin = 2;
12 volatile bool pressed = false;
13 volatile bool night = false;
14
15 int sensorValue = 0;      // the sensor value
16 int sensorMin = 1023;    // minimum sensor value
17 int sensorMax = 0;        // maximum sensor value
18
19 void setup() {
20     for (size_t i = 0; i <= sizeof(pins); i++) {
21         pinMode(pins[i], OUTPUT);
22     }
23     attachInterrupt(digitalPinToInterrupt(interruptPin),
24                     handleRequest_, RISING);
25     // set default state
26     digitalWrite(A1_G, HIGH);
27     digitalWrite(A1_Y, LOW);
28     digitalWrite(A1_R, LOW);
29     digitalWrite(A2_G, LOW);
30     digitalWrite(A2_R, HIGH);
31
32     // calibrate during the first five seconds
33     calibrate_();
34     Serial.begin(9600);
35 }
36 void calibrate_() {
```

```
37  while ( millis () < 5000) {  
38      sensorValue = analogRead (LIGHT_SENSOR);  
39  
40      // record the maximum sensor value  
41      if (sensorValue > sensorMax) {  
42          sensorMax = sensorValue;  
43      }  
44  
45      // record the minimum sensor value  
46      if (sensorValue < sensorMin) {  
47          sensorMin = sensorValue;  
48      }  
49  }  
50 }  
51  
52 void loop () {  
53     sensorValue = analogRead (LIGHT_SENSOR);  
54     // in case the sensor value is outside the range seen  
     during calibration  
55     sensorValue = constrain (sensorValue , sensorMin , sensorMax  
    );  
56  
57     // apply the calibration to the sensor reading  
58     sensorValue = map (sensorValue , sensorMin , sensorMax , 0 ,  
    255);  
59     Serial . println (sensorValue);  
60     // Switch to Night mode  
61     if (sensorValue < 100 && !night) {  
62         digitalWrite (A1_G , LOW);  
63         digitalWrite (A1_R , LOW);  
64         digitalWrite (A2_G , LOW);  
65         digitalWrite (A2_R , LOW);  
66         night = true ;  
67     }  
68     // Switch to Day mode  
69     if (sensorValue > 150 && night) {  
70         digitalWrite (A1_G , HIGH);  
71         digitalWrite (A1_R , LOW);  
72         digitalWrite (A2_G , LOW);  
73         digitalWrite (A2_R , HIGH);  
74         night = false ;
```

```
75     pressed = false;  
76 }  
77 // Night mode  
78 if (night) {  
79     digitalWrite(A1_Y, HIGH);  
80     delay(1000);  
81     digitalWrite(A1_Y, LOW);  
82     delay(1000);  
83     return;  
84 }  
85  
86 if (pressed) {  
87     schaltSequenz_();  
88 }  
89 }  
90  
91 void schaltSequenz_() {  
92     // Green Blinking phase  
93     for (size_t i = 0; i < NUM_WARN_BLINKS; i++) {  
94         digitalWrite(A1_G, LOW);  
95         delay(1000);  
96         digitalWrite(A1_G, HIGH);  
97         delay(1000);  
98     }  
99     digitalWrite(A1_G, LOW);  
100    // Yellow phase  
101    digitalWrite(A1_Y, HIGH);  
102    delay(1000);  
103    // GO phase  
104    digitalWrite(A1_Y, LOW);  
105    digitalWrite(A1_R, HIGH);  
106    digitalWrite(A2_R, LOW);  
107    digitalWrite(A2_G, HIGH);  
108    delay(5000);  
109    // Green Blinking phase for walkers  
110    for (size_t i = 0; i < NUM_WARN_BLINKS; i++) {  
111        digitalWrite(A2_G, LOW);  
112        delay(1000);  
113        digitalWrite(A2_G, HIGH);  
114        delay(1000);  
115    }
```

```
116 digitalWrite(A2_G, LOW);  
117 // STOP for walkers and Transition Car  
118 digitalWrite(A2_R, HIGH);  
119 digitalWrite(A1_Y, HIGH);  
120 delay(1000);  
121 // Car go space  
122 digitalWrite(A1_Y, LOW);  
123 digitalWrite(A1_R, LOW);  
124 digitalWrite(A1_G, HIGH);  
125 pressed = false;  
126 }  
127  
128 void handleRequest_() {  
129     if (!pressed) {  
130         pressed = true;  
131     }  
132 }
```

7.2 Game Code

```
1 #define A 13  
2 #define B 12  
3 #define C 11  
4 #define D 10  
5 #define E 9  
6 #define F 8  
7 #define G 7  
8 #define POINTS_TO_END_THE_GAME 10  
9 // order a b g e d c g f  
10 char order[8] = {A, B, G, E, D, C, G, F};  
11  
12 char pins[7] = {A, B, C, D, E, F, G};  
13 const byte INTERRUPT_PIN = 2;  
14 const byte P1_PIN = 5;  
15 const byte P2_PIN = 6;  
16 size_t totGamePoints = 0;  
17 size_t pointsP1 = 0;  
18 size_t pointsP2 = 0;  
19 volatile bool ended = false;  
20 volatile bool forward = true;  
21 volatile size_t curPos = 0;
```

```
22
23 void setup() {
24     for (size_t i = 0; i < sizeof(pins); i++) {
25         pinMode(pins[i], OUTPUT);
26     }
27     pinMode(INTERRUPT_PIN, INPUT_PULLUP);
28     pinMode(P1_PIN, INPUT);
29     pinMode(P2_PIN, INPUT);
30
31     attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN) ,
32                     scoring_ , FALLING);
33     Serial.begin(9600);
34 }
35 void loop() {
36     if (ended) {
37         end_();
38         return;
39     }
40     // flash current Segment
41     digitalWrite(order[curPos] , HIGH);
42     delay(500 / (totGamePoints + 1));
43     digitalWrite(order[curPos] , LOW);
44     // select next segment
45     curPos = forward ? curPos + 1 : (curPos == 0) ? sizeof(
46                                         order)-1 : curPos - 1;
46     curPos = (curPos) % sizeof(order);
47 }
48
49 void end_() {
50     // flash all segments
51     for (size_t i = 0; i <= sizeof(pins); i++) {
52         digitalWrite(order[i] , HIGH);
53     }
54     delay(1000);
55     for (size_t i = 0; i <= sizeof(pins); i++) {
56         digitalWrite(order[i] , LOW);
57     }
58     delay(1000);
59 }
```

60

```
61 // Prints the current Score of Players
62 void printScore_() {
63     char p1[] = "P1 ";
64     char p2[] = "/ P2 ";
65     Serial.print(p1);
66     Serial.print(pointsP1);
67     Serial.print(p2);
68     Serial.println(pointsP2);
69 }
70
71 void updateStats_(bool statP1, bool statP2) {
72     // check if somebody scored
73     if (order[curPos] == A && statP1) {
74         totGamePoints = totGamePoints + 1;
75         pointsP1 = pointsP1 + 1;
76         if (statP2) {
77             pointsP2 = (pointsP2 == 0) ? 0 : pointsP2 - 1;
78         }
79     } else if (order[curPos] == D && statP2) {
80         totGamePoints = totGamePoints + 1;
81         pointsP2 = pointsP2 + 1;
82         if (statP1) {
83             pointsP1 = (pointsP1 == 0) ? 0 : pointsP1 - 1;
84         }
85     } else {
86         // penalize player
87         if (statP1) {
88             pointsP1 = (pointsP1 == 0) ? 0 : pointsP1 - 1;
89         }
90         if (statP2) {
91             pointsP2 = (pointsP2 == 0) ? 0 : pointsP2 - 1;
92         }
93         // switch direction
94         forward = !forward;
95     }
96 }
97
98 void scoring_() {
99     if (ended){
100         delay(200);
101     }
}
```

```
102  bool statP1 = digitalRead(P1_PIN);  
103  bool statP2 = digitalRead(P2_PIN);  
104  if (ended && statP1 && statP2) {  
105      // reset the game stats  
106      totGamePoints = 0;  
107      pointsP1 = 0;  
108      pointsP2 = 0;  
109      ended = false;  
110      Serial.println("RESET");  
111      return;  
112  } else if (ended) {  
113      // ignore interrupt  
114      return;  
115  }  
116  updateStats_(statP1, statP2);  
117  // end the game if max points reached  
118  if (totGamePoints == POINTS_TO_END_THE_GAME) {  
119      ended = true;  
120      Serial.println("Final Score:");  
121  }  
122  printScore_();  
123 }  
124 }
```

Literaturverzeichnis

- [1] *DM74LS02 Quad 2-Input NOR Gate.* en. 2000. URL: <https://www.futurlec.com/74LS/74LS02.shtml> (besucht am 22.05.2022).

Abbildungsverzeichnis

1	Dies sind die zwei entprellten Schalterplatinen mit je zwei Schalter (S_1, S_2) die entweder mit Standard- <i>HIGH</i> oder - <i>LOW</i> verwendet werden können. Diese werden durch Beschalten des <i>GND</i> und der 5 V Betriebsspannung in Betrieb genommen	12
2	Diese Grafik beinhaltet den Programmablauf einer, nach der Aufgabenstellung entworfenen, Ampelschaltung mit einem Fußgängerübergang. Hier wurde der <i>DIN 66001</i> verwendet, um den Programmablauf darzustellen. Falls der Plan zu klein ist, ist dieser in der Vorbereitung nochmals größer ersichtlich.	13
3	Dies ist der skizzierte Aufbau der Ampelschaltung, welche in der Aufgabenstellung beschrieben worden ist.	14
4	Diese Abbildung beinhaltet den Aufbau am Steckbrett zur Ampelschaltung, wie in Abbildung 3 ersichtlich. Die Beschriftungen in blau bezeichnen die am <i>Arduino Uno R3</i> verwendeten Pins.	14
5	Diese Grafik beinhaltet den Programmablauf eines, nach der Aufgabenstellung entworfenen, Reaktionsspiels. Hier wurde der <i>DIN 66001</i> verwendet, um den Programmablauf darzustellen. Falls dieser zu klein sein sollte, ist dieser in der Vorbereitung nochmals größer ersichtlich.	15
6	Dies ist der skizzierte Aufbau des Reaktionsspiels, welche in der Aufgabenstellung beschrieben worden ist.	16
7	Diese Abbildung beinhaltet den Aufbau am Steckbrett zum Reaktionsspiel, wie in Abbildung 6 ersichtlich. Die Beschriftungen in blau bezeichnen die am <i>Arduino Uno R3</i> verwendeten Pins.	17

Tabellenverzeichnis

1	Tabelle der verwendeten Geräte	11
---	--	----