Inhaltsverzeichnis

Elektronik und computerunterstützte Messtechnik
LU PHY.M20 (SS 2022)

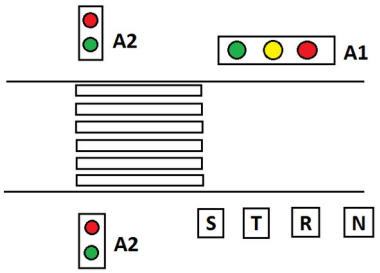
Übung 5: Mikrocontroller
1. Ampelschaltung
2. Reaktionsspiel

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



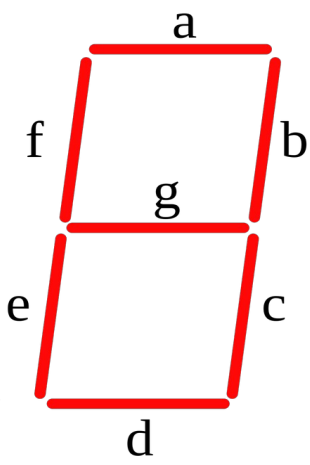
Das Ziel der Übung ist die Simulation einer geregelten Ampelschaltung an einem Fußgängerübergang (s. Abb. rechts) mit Hilfe eines Arduino UNO Boards. Programmiert wird in der Arduino Entwicklungsumgebung.

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 → b → g → e → d → c → g → f bzw. umgekehrt. Die Leuchtdauer der einzelnen Segmente beträgt zu Beginn des Spieles 500 ms und wird pro erzieltem Punkt kürzer (zB. T_{on} = 500 ms / ($n_{Gesamtpunke}$ +1)).

Wenn nun das Segment (a / d) leutet 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 erziehlten 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 Endpunktestand 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 Endpunktestand 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 werde. 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

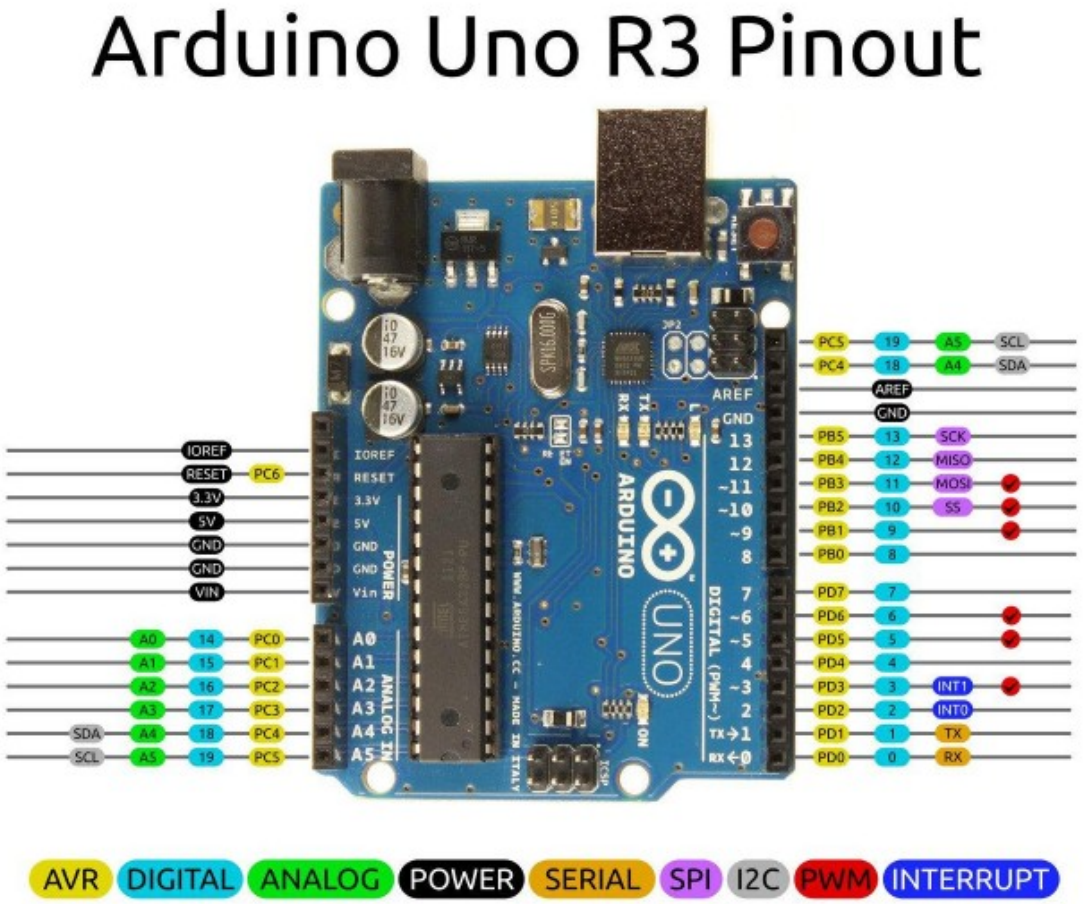
Für die Programmierung der Ampelsteuerung ist es empfehlenswerteinen der zwei verfügbaren Interrupt Pins für den Taster „R“ zu verwenden. Die „delay“ Funktion kann verwendet werden solange die Funktion der Ampelsteuerung nicht eingeschränkt wird.

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!



Ein paar nützliche Funktionen in der Arduino-Umgebung:

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

2 Grundlagen

Schaltwerke (sequentielle Logik) können im Gegensatz zu Schaltnetzen (kombinatorische Logik) deren Zustände speichern. Als Grundlage dienen hierfür Flip-Flops, also bistabile Kippstufen, die es in diversen Ausführungen gibt - beispielsweise existieren unterschiedliche Steuerungsarten (zustands- oder flankengesteuert). Elementar ist hierbei das RS-Flip-Flop wie es in Abbildung 1 zu sehen ist. Hierbei wurde jenes mittels NOR-Gattern realisiert; anstelle hätten auch NAND-Gatter verwendet werden können.

Abbildung 1: Diese Schaltung zeigt den Aufbau eines RS-Flip-Flops aus NOR-Gattern mit Eingängen R und S sowie Ausgängen Q und $\bar{Q}$ [7]

Dabei handelt es sich um ein zustandsgesteuertes Flip-Flop mit den Eingängen R und S sowie Ausgängen Q und $\bar{Q}$, wobei dieses transparent ist, da Änderungen vom Ein- auf den Ausgang sofort übertragen werden. Ein Flip-Flop kann nun mit einem zusätzlichen Eingang, dem Takt, auch Clock (C), angesteuert werden. Wenn nun zusätzlich zwei Flips-Flops miteinander seriell verbunden werden und der Takt am Eingang des zweiten Flip-Flops negiert wird, kann man ein Master-Slave Flip-Flop erhalten, welches zweiflankengesteuert und nicht-transparent ist; d.h. dass am Master, dem erste Flip-Flop, die Signale der Eingänge bei steigender Flanke eingelesen/zwischengespeichert und an den Slave, den hinteren Flip-Flop, bei fallender Flanke übertragen werden, wobei der Eingang dabei verriegelt ist. Somit sind Eingang und Ausgang also getrennt. Wenn weiters die Ausgänge auf die Eingänge rückgekoppelt werden, erhält man ein zweiflankengesteuertes JK-Master-Slave Flip-Flop mit Eingängen J und K, wie es in **??** (Vorbereitung) inklusive der zugehörigen Wahrheitstabelle dargestellt wird. Aufgrund der Rückkopplung tritt bei $J = K = 1$ Togglen auf (an der negativen Flanke des Takts), was das Kippen des vorherigen Zustands beschreibt. Diese Konstellation der Eingangszustände war ohne Rückkopplung nicht definiert beziehungsweise verboten.

Anwendungen finden Flip-Flops unter anderem für Zähler (wie in dieser Laborübung) oder für Schieberegister. Dabei stellen die Flip-Flops Speicherelemente für die Zustände dar, welche mittels der anliegenden Taktung verändert werden können.

3 Versuchsdurchführung

Da die exakten Komponenten für die logischen Gatter nicht in *LTspice* zur Verfügung standen, wurden funktionstüchtig-äquivalente Bauteile in der Simulation verwendet.

Tabelle 1: Tabelle der verwendeten Geräte

Geräteliste		
Gerät/Bauelement	**Typ**	**In Simulation**
Netzgerät	nicht bestimmbar	
NOT-Gatter	*74LS04*[1]	74HCT04
2x-NAND-Gatter	*74LS00*[5]	74HCT00
3x-NAND-Gatter	*74LS10*[6]	74HCT10
JK-MS-FF	*74LS109*[2]	74HCT109

Die Gegenstücke in der Simulation wurden unten angeführt. Die verwendeten Geräte sind Tabelle 1 zu entnehmen.

Entprellter Schalter Der entprellter Schalter wird als Signalgeber für die logischen Schaltungen und Gatter verwendet. Der Aufbau dieses Schalters ist in Abbildung 2 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 2: Dies sind die zwei entprellten Schalterplatinen mit je zwei Schalter ($S1$,$S2$) die entweder mit Standard-*HIGH* oder -*LOW* verwendet werden können. Diese werden durch Beschalten des *GND* und der 5 V Betriebsspannung in Betrieb genommen

LED Leiste Damit die Eingangssignale und Ausgangssignale der logischen Schaltungen dargestellt werden können, wird eine LED-Leiste verwendet. Diese besteht aus mehreren LEDs mit Vorwiderständen und ist in Abbildung 3 dargestellt und unterstützt bis zu 8 Aus- bzw. Eingangssignale. Die hier verwendete LED-Leiste hat einen Common-Ground und somit werden positiven Spannungssignale direkt an den verschiedenen Anschlüssen angelegt um die LED zum Leuchten zu bringen.

Abbildung 3: Dies ist die LED-Leiste, welche 8 LEDs mit je einem Vorwiderstand hat. Sie ist in Common-Ground-Konfiguration und wird verwendet, um Signal anzeigen zu können.

Damit die Aufnahmen der Schaltungen übersichtlich bleiben, wurden diese meistens von den Aufnahmen weggeschnitten und deren Funktion in der jeweiligen Grafik mittels *LED* dargestellt oder in der Beschriftung gekennzeichnet bzw. erwähnt.

3.1 Ampelschaltung

3.1.1 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 folgernder Abbildung 4 ist der Programmablauf nochmals ersichtlich des weiteren wurde der verwendete Code im Appendix 6.1 hinzugefügt.

Abbildung 4: Diese Grafik beinhaltet den Programmablauf einer, nach der Aufgabenstellung entworfenen, Ampelschaltung mit einem Fußgängerübergang. Hier wurde der *DIN 66001* verwendet um den Programmablauf darzustellen.

Beim erstellen des Code wurde auf die korrekte Beschaltung wie in der kleinen österreichischen Ampelkunde beschrieben wird.

3.1.2 Steckboard

Aufbau des JK-Master-Slave-Flip-Flop Zunächst wird der CMOS-Inverter mittels zweier MOSFETs (einem p-MOSFET [4] und einem n-MOSFET [3]) wie nach dem Schaltbild (**??**) aufgebaut. Zur Visualisierung des Eingangszustands U_l und des Ausgangszustands U_q wurden die LEDs der LED-Leiste parallel dazu geschaltet. Das Eingangssignal wurde durch einen entprellten Schalter, im Standardzustand *LOW*, gegeben. Der Aufbau wird in Absatz 3 dargestellt. Als Spannungsquelle wurde ein Netzgerät verwendet und auf 5 V eingestellt.

Abbildung 5: Dies ist der Aufbau eines JK-Master-Slave-Flip-Flops nach dem Schaltplan aus **??**, wobei:
J... normale Set Eingang
K... normale Reset Eingang
C... Clocksignal Eingang
R... CLEAR Eingang
S... PRESET Eingang
Q... Ausgang
$\bar{Q}$... Invertierte Ausgang
ist . Der Zustand beider kann anhand einer LED in der LED-Leiste abgelesen werden.

Um die Funktionstüchtigkeit des JK-MS-FF's zu überprüfen, wurde die Wahrheitstafel alle möglichen Schaltsignale der J K Eingänge durchgeschaltet. Die Resultate sind in Abbildung 6 ersichtlich.

Abbildung 6: Diese Abbildung beinhaltet die gemessenen Eingangs- und Ausgangssignale des gebauten JK-Master-Slave-Flip-Flops. Eine leuchtende LED entspricht einem *HIGH* Signal, eine nicht leuchtende entspricht *LOW*. Zudem ist:
J... normale Set Eingang
K... normale Reset Eingang
C... Clocksignal Eingang
R... CLEAR Eingang
S... PRESET Eingang
Q... Ausgang
$\bar{Q}$... Invertierte Ausgang

3.2 Reaktionsspiel

3.2.1 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 folgernder Abbildung 7 ist der Programmablauf nochmals ersichtlich desweiteren wurde der verwendete Code im Appendix 6.2 hinzugefügt.

Abbildung 7: Diese Grafik beinhaltet den Programmablauf eines, nach der Aufgabenstellung entworfenen, Reaktionsspiels. Hier wurde der *DIN 66001* verwendet um den Programmablauf darzustellen.

3.2.2 Steckboard

Wie in Aufgabenstellung gefordert, galt es eine Schaltung zu entwickeln, welche durch Anschlagen mindestens zweier Sensoren einen Alarm auslösen sollte. Diese Schaltung wurde, um die Fehlerquellen zu reduzieren, jedoch aus der Musterlösung entnommen und nicht die selbst Entworfene verwendet. Die vier Eingangsgrößen ($x1$, $x2$, $x3$, $x4$) wurden durch vier entprellte Schalter realisiert. Dabei sind $x1$ & $x3$ im Standard LOW und $x2$ & $x4$ Standard $HIGH$ Betrieb geschaltet worden, damit die Alarmanlage bei Signalunterbrechung $x2$ & $x4$ und einem Signal bei $x1$ & $x3$ anschlägt. Der Aufbau der Schaltung kann Dies ist der Aufbau des dekadischen synchron 4Bit-Zähler nach dem Schaltplan aus **??**. Dabei sind QN die $(N-1)$ten Bits des Zählers und C das zuzählende Eingangssignal. Die verwendeten Komponenten sind der Tabelle 1 zu entnehmen. entnommen werden.

Abbildung 8: Dies ist der Aufbau des dekadischen synchron 4Bit-Zähler nach dem Schaltplan aus **??**. Dabei sind QN die $(N-1)$ten Bits des Zählers und C das zuzählende Eingangssignal. Die verwendeten Komponenten sind der Tabelle 1 zu entnehmen.

Um die Funktionstüchtigkeit des 4Bit dekadischen synchron Zählers zu überprüfen, wurde so viele Signale bei CLK damit der Zähler bis 10 zählt und dann sich resetet. Somit konnte die Zählfähigkeit der Schaltung mittels LEDs an den Ausgängen QN dargestellt werden. Die Kombinationen sind in Abbildung 9 ersichtlich.

Abbildung 9: Diese Abbildung beinhaltet die gemessenen logischen Ausgangszustände der QN Ausgänge bei einer Serie an Eingangspulsen. Dabei sind QN die $(N-1)$ten Bits des gebauten 4Bit dekadischen synchron Zählers. Die Zahlen in der rechten Spalte entsprechen dem j-ten gegebenen Puls im Eingangssignal CLK.Eine leuchtende LED entspricht einem $HIGH$-, eine nicht leuchtende entspricht einem LOW-Signal

4 Auswertung

In diesem Protokoll ist keine Auswertung von Nöten, da die geforderten Resultate direkt aus den Ergebnissen der Laborübung folgen. Dennoch wurde für bessere Nachvollziehbarkeit die Daten aus **??** nochmals mit Farben hinterlegt und die *HIGH* Zustände mit einer 1 markiert.

Abbildung 10: Diese Grafik spiegelt das Verhalten der Einbruchssicherungsschaltung (aus **??**) wider, indem alle möglichen Eingangssignale durchgeschaltet worden sind und die Response an den Ausgängen aufgezeichnet wurde (entsprechend Wahrheitstafel).Dabei sind $x1$, $x2$, $x3$, $x4$ & der *Master-Switch* die Eingangssignale und y das Ausgangssignal sowie $y_m s$ das Ausgangssignal mit *Master-Switch*. Die SPICE-Directive der Simulation ist `.tran 0 4 0`; die Einstellungen der Eingangssignale können der **??** entnommen werden. Zudem wurden hier die Ticks von einander mittels orange gepunkteten Linien getrennt und die *HIGH* mit 1er markiert.

5 Diskussion und Zusammenfassung

5.1 Diskussion

In **??** von der Simulation und Abbildung 6, welche die LEDs vom Steckbrett zeigt, ist das Verhalten eines CMOS-Inverters gut zu erkennen; dabei wird das Signal am Ausgang relativ zu jenem am Eingang negiert. Weiters ist anhand von **??** aus der Simulation ersichtlich, dass der Schaltvorgang in CMOS endlich schnell erfolgt. Dabei ist ein deutliches Maximum des Stroms von 4,21 mA zu erkennen (siehe **??**), wenn (im Falle des Inverters) beide MOSFETs leitfähig sind, nämlich am Schnittpunkt der Spannungsverläufe. Die Spannung an diesen Schnittpunkten beträgt 1,63 V (siehe **??**) und liegt somit innerhalb der für die Gate-Source-Spannung toleranten Intervalle von 0,8 V bis 2,4 V beim *ZVN2106A* und von −3,5 V bis −1,5 V beim *ZVP2106A* nach den zugehörigen Datenblättern.

Das aus MOSFETs konstruierte NAND liefert als Ergebnis aller möglichen Kombinationen der beiden Eingangssignale **??** für die Simulation und **??** für die Steckbrettschaltung. Somit folgt der Verhalt jenem, der in der Wahrheitstabelle in **??** (Vorbereitung) notiert wurde - der Ausgang ist stets `HIGH`, außer wenn beide Eingänge auf `High` sind.

Genauso folgen für die Einbruchsicherungsschaltung **??** und Abbildung 9 der konstruierten Wahrheitstabelle in **??** (Vorbereitung). Demnach löst der Alarm aus, sofern der *Master-Switch* eingeschaltet ist, wenn mindestens zwei Eingangsvariablen auf `HIGH` sind.

5.2 Zusammenfassung

Im Rahmen dieser Laborübung wurde das Verhalten von einem CMOS-Inverter (siehe **??** respektive Abbildung 6), einem aus CMOS konstruierten NAND (siehe **??** respektive **??**) und einer Einbruchsicherungsschaltung (=Schaltnetz mit 4 Eingangsvariablen und *Master-Switch*; siehe Abbildung 10 respektive Abbildung 9) erfolgreich mittels Simulation und mithilfe von LEDs an Steckbrettschaltungen verifiziert.

6 Appendix

6.1 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), handleRequest_,
      RISING);
24   // set default state
25   digitalWrite(A1_G, HIGH);
26   digitalWrite(A1_Y, LOW);
27   digitalWrite(A1_R, LOW);
28   digitalWrite(A2_G, LOW);
29   digitalWrite(A2_R, HIGH);
30
31   // calibrate during the first five seconds
32   calibrate_();
33   Serial.begin(9600);
34 }
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 }
```

6.2 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), scoring_,
      FALLING);
32   Serial.begin(9600);
33 }
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(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
123   printScore_();
124 }
```

Literaturverzeichnis

[1] *DM74LS04 Hex Inverting Gates*. en. 2000. URL: https://www.futurlec.com/74LS/74LS04.shtml (besucht am 22. 05. 2022).

[2] *Dual J-K Positive-Edge-Triggered Flip-Flops With Preset And Clear datasheet*. en. 1988. URL: https://www.ti.com/lit/ds/symlink/sn54ls109a.pdf (besucht am 08. 06. 2022).

[3] *N-CHANNEL ENHANCEMENTMODE VERTICAL DMOS FET ZVN2106A*. en. 1994. URL: https://www.diodes.com/assets/Datasheets/ZVN2106A.pdf (besucht am 22. 05. 2022).

[4] *P-CHANNEL ENHANCEMENTMODE VERTICAL DMOS FET ZVP2106A*. en. 1994. URL: https://www.diodes.com/assets/Datasheets/ZVP2106A.pdf (besucht am 22. 05. 2022).

[5] *SNx400, SNx4LS00, and SNx4S00 Quadruple 2-Input Positive-NAND Gates*. en. 2017. URL: https://www.ti.com/lit/ds/symlink/sn74ls00.pdf (besucht am 08. 06. 2022).

[6] *Technical Information - Fairchild Semiconductor 74LS10 Datasheet*. en. 2000. URL: https://www.futurlec.com/74LS/74LS10.shtml (besucht am 08. 06. 2022).

[7] Ulrich Tietze, Christoph Schenk und Eberhard Gamm. *Halbleiter-Schaltungstechnik*. Springer-Verlag GmbH, 5. Juli 2019. ISBN: 3662485532. URL: https://www.ebook.de/de/product/37063433/ulrich_tietze_christoph_schenk_eberhard_gamm_halbleiter_schaltungstechnik.html.

Abbildungsverzeichnis

Tabellenverzeichnis