Reaktionsspiel

**Gesamtdokumentation**

**AP-Verantwortlicher:** Thomas Ammann, Ioannis Christodoulakis  
**Datum:** 31. Mai 2017

**Raumtemperatur:** 24.5 °C  
**Revision:** 01



**Inhaltsverzeichnis**

[1. Zusammenfassung 4](#_Toc483982832)

[2. Projektgesamtziel 4](#_Toc483982833)

[3. Grobplanung 4](#_Toc483982834)

[4. Feinplanung 5](#_Toc483982835)

[4.1 Funktionsprinzip 5](#_Toc483982836)

[4.2 Festlegungen 5](#_Toc483982837)

[4.3 Anforderungskatalog 6](#_Toc483982838)

[4.3.1 Hardware 6](#_Toc483982839)

[4.3.2 Software 7](#_Toc483982840)

[4.3.3 Zusammenführung 9](#_Toc483982841)

[4.3.4 Erweiterung Arbeitspaket 4. und Arbeitspaket 5. 10](#_Toc483982842)

[4.3.5 Begriffsverzeichnis 11](#_Toc483982843)

[5. Arbeitspakete 12](#_Toc483982844)

[5.1 Arbeitspaket 1: Eingabe 12](#_Toc483982845)

[5.1.1 Zusammenfassung 12](#_Toc483982846)

[5.1.2 Aufgabenstellung 12](#_Toc483982847)

[5.1.3 Vorgaben 12](#_Toc483982848)

[5.1.4 Blockschaltbild 12](#_Toc483982849)

[5.1.5 Lösungsansätze 12](#_Toc483982850)

[5.1.6 Berechnungen 12](#_Toc483982851)

[5.1.7 Schaltung 13](#_Toc483982852)

[5.1.8 Messungen 13](#_Toc483982853)

[5.1.9 Verwendete Geräte 14](#_Toc483982854)

[5.1.10 Schlussfolgerung 14](#_Toc483982855)

[5.2 Arbeitspaket 2: Ausgabe 14](#_Toc483982856)

[5.2.1 Zusammenfassung 14](#_Toc483982857)

[5.2.2 Aufgabenstellung 14](#_Toc483982858)

[5.2.3 Vorgaben 14](#_Toc483982859)

[5.2.4 Blockschaltbild 15](#_Toc483982860)

[5.2.5 Berechnungen 15](#_Toc483982861)

[5.2.6 Schaltung 17](#_Toc483982862)

[5.2.7 Messung 18](#_Toc483982863)

[5.2.8 Verwendete Geräte und Elektrokomponente 18](#_Toc483982864)

[5.2.9 Schlussfolgerung 18](#_Toc483982865)

[5.3 Arbeitspaket 3: Start 19](#_Toc483982866)

[5.3.1 Zusammenfassung 19](#_Toc483982867)

[5.3.2 Aufgabenstellung 19](#_Toc483982868)

[5.3.3 Begriffsverzeichnis 19](#_Toc483982869)

[5.3.4 Struktogramm 19](#_Toc483982870)

[5.3.5 Programm–Code 20](#_Toc483982871)

[5.3.6 Schlussfolgerung 21](#_Toc483982872)

[5.4 Arbeitspaket 4: Spielerauswahl 21](#_Toc483982873)

[5.4.1 Zusammenfassung 21](#_Toc483982874)

[5.4.2 Aufgabenstellung 21](#_Toc483982875)

[5.4.3 Struktogramm 22](#_Toc483982876)

[5.4.4 Programm–Code 22](#_Toc483982877)

[5.4.5 Schlussfolgerung 23](#_Toc483982878)

[5.5 Arbeitspaket 5: Zeitmessung 24](#_Toc483982879)

[5.5.1 Zusammenfassung 24](#_Toc483982880)

[5.5.2 Aufgabenstellung 24](#_Toc483982881)

[5.5.3 Struktogramm 25](#_Toc483982882)

[5.5.4 Programm – Code 25](#_Toc483982883)

[5.5.5 Schlussfolgerung 28](#_Toc483982884)

[5.6 Arbeitspaket 6: Zeitmessung-Auswertung 29](#_Toc483982885)

[5.6.1 Zusammenfassung 29](#_Toc483982886)

[5.6.2 Aufgabenstellung 29](#_Toc483982887)

[5.6.3 Struktogramm 29](#_Toc483982888)

[5.6.4 Programm–Code 30](#_Toc483982889)

[5.6.5 Schlussfolgerung 31](#_Toc483982890)

[5.7 Arbeitspaket 7: Spielende 31](#_Toc483982891)

[5.7.1 Zusammenfassung 31](#_Toc483982892)

[5.7.2 Aufgabenstellung 31](#_Toc483982893)

[5.7.3 Struktogramm 32](#_Toc483982894)

[5.7.4 Programm–Code 32](#_Toc483982895)

[5.7.5 Schlussfolgerung 33](#_Toc483982896)

[5.8 Arbeitspaket 8: Neustart 33](#_Toc483982897)

[5.8.1 Zusammenfassung 33](#_Toc483982898)

[5.8.2 Aufgabenstellung 33](#_Toc483982899)

[5.8.3 Struktogramm 34](#_Toc483982900)

[5.8.4 Programm–Code 34](#_Toc483982901)

[5.8.5 Schlussfolgerung 35](#_Toc483982902)

[5.9 Gesamt-Hardware 36](#_Toc483982903)

[5.10 Gesamt-Software 37](#_Toc483982904)

[6. Verwendete Geräte 42](#_Toc483982905)

[7. Stückliste und Kosten 42](#_Toc483982906)

[8. Schlussfolgerung 42](#_Toc483982907)

[9. Quellenverzeichnis 43](#_Toc483982908)

[10. Identifikation 43](#_Toc483982909)

# 1. Zusammenfassung

Es wurde, zu zweit, ein Projekt durchgeführt, in welchem ein Reaktionsspiel entwickelt wurde.

Das Reaktionsspiel vergleicht die Reaktionszeiten von zwei Spielern. Derjenige Spieler mit der kürzeren Reaktionszeit erhält einen Punkt. Auf 7-Segment-Anzeigen werden die Punktzahlen der Spieler angezeigt. Um das Spiel zu gewinnen, muss der Spieler neun Punkte erspielen.

Das Spiel wird von einem Arduino UNO Rev. 3 gesteuert. Die 7-Segment-Anzeigen werden mit BCD-UP-Counter und BCD-to-7-Segment-Converter angesteuert.

Das Projekt wurde in 10 Arbeitspakete unterteil. Diese wurden miteinander abgestimmt.   
Die entworfene Feinplanung musste im Verlauf des Projektes für AP4. & AP5. erweitert werden. Dies ermöglichte eine einfachere Durchführung der beiden Arbeitspakete.

Das Projekt ist im Zeitrahmen durchgeführt worden. Die Zeitplanung konnte eingehalten werden und ist plangemäss abgeschlossen worden.

# 2. Projektgesamtziel

Das Gerät soll die Reaktionszeit zweier Spieler überprüfen. Es besitzt zwei Taster mit denen die Reaktionszeit jedes Spielers nach einem optischen Signal gemessen wird. Der Spieler mit der kürzeren Reaktionszeit erhält einen Punkt. Die Punkte sind auf Anzeigen ersichtlich. Erzielt ein Spieler 9 Punkte wird ein optisches Signal ausgegeben und das Spiel ist beendet. Das Gerät soll spielfähig aufgebaut sein. Das Projekt soll am 28.06.2017 fertiggestellt sein.

# 3. Grobplanung

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Projektstruktur** | | | | | |
| **Teilziel** | **Arbeitspakete** | **Aufwand** | | | |
| Schätzung | | Ist | |
| Tag/e | Datum | Tag/e | Datum |
| Grobplanung | | ½ | 09.05.2017 | ½ | 10.05.2017 |
| Feinplanung | | 1 | 10.05.2017 | ½ | 10.05.2017 |
| Hardware | AP1. Eingabe | ½ | 15.05.2017 | ½ | 15.05.2017 |
| AP2. Ausgabe | 2 | 16.05.2017 | 1 | 16.05.2017 |
| Software | AP3. Start | ½ | 17.05.2017 | ¼ | 17.05.2017 |
| AP4. Spielerauswahl | ½ | 17.05.2017 | ½ | 22.05.2017 |
| AP5. Zeitmessung | 2 | 23.05.2017 | ½ | 22.05.2017 |
| AP6. Zeitmessung-Auswertung | 1 | 23.05.2017 | ¼ | 17.05.2017 |
| AP7. Spielende | ½ | 24.05.2017 | ¼ | 17.05.2017 |
| AP.8 Neustart | ½ | 24.05.2017 | ¼ | 17.05.2017 |
| Zusammen-Führung | AP9. Gesamtprogramm | 1 | 29.05.2017 | ½ | 29.05.2017 |
| AP10. Gesamt-Hardware | 1 | 29.05.2017 | ½ | 29.05.2017 |
| Anpassungen | 2 | / | ½ | 29.05.2017 |
| Bericht | Gesamtdokumentation | 2 | 31.05.2017 | 2 | 31.05.2017 |

# 4. Feinplanung

## 4.1 Funktionsprinzip

**Start**

Zum Starten wird die Start-Taste gedrückt. Die Lampen der Spieler blinken drei Mal auf.

**Spielerauswahl**

Zu Beginn jeder Runde wird zufällig ein Spieler gewählt, welcher beginnt. Danach kommt der andere Spieler.

Es darf nicht gezeigt werden welcher Spieler beginnt!

**Zeitmessung**

Zufällig nach zwei bis sechs Sekunden beginnt die Zeitmessung. Das Aufleuchten einer Lampe signalisiert dem Spieler den Messungsstart. Dieser soll nun so schnell als möglich seine Taste drücken. Mit dem Drücken der Taste wird die Zeitmessung gestoppt.   
Dieser Vorgang wird mit dem anderen Spieler wiederholt.

**Auswertung**

Die gemessenen Reaktionszeiten der Spieler werden verglichen. Der Spieler mit der kleineren Reaktionszeit erhält einen Punkt.

**Spielende**

Wenn ein Spieler neun Punkte erhält ist das Spiel beendet. Die Lampe des Gewinners blinkt.

**Neustart**

Zum Neustarten wird die Start-Taste gedrückt. Die Punkte werden zurückgesetzt.  
Das Spiel kann nur nach Spielende neu gestartet werden.

## 4.2 Festlegungen

**Hardwarefestlegungen**

* Das Reaktionsspiel wird mit einem Arduino Uno gesteuert.
* Die Speisespannung beträgt 4.5V DC.

**Softwarefestlegungen**

* Die Namen des Begriffsverzeichnisses müssen verwendet werden.
* Das gesamte Programm muss kleiner als 32kByte sein.

## 4.3 Anforderungskatalog

### 4.3.1 Hardware

**Arbeitspaket 1: Eingabe**

*Vorgaben:*Es soll für beide Spieler ein Taster zur Quittierung zur Verfügung stehen.  
 Mit einem Taster soll das Gerät gestartet bzw. zurückgesetzt werden.

*Schnittstelleninformationen:*

|  |  |  |
| --- | --- | --- |
| Speisung | Eingabe | Ausgabe |
|  | Quittierungs-Impuls1 | Quittierungs-Signal1 |
| 4.50V DC | Quittierungs-Impuls2 | Quittierungs-Signal2 |
|  | Start-Impuls | Start-Signal |

**Arbeitspaket 2: Ausgabe**

*Vorgaben:*Die Punktzahlen der Spieler sollen auf Anzeigen ersichtlich sein.  
 Jeder Spieler besitzt eine optische Meldung, die den Beginn der Zeitmessung signalisiert.   
 Jeder Spieler besitzt eine optische Meldung, die signalisiert, wer gewonnen hat.

*Erweiterungen:*  
 Die Punktzahl der Spieler wird aus je einem Controllerausgang bestimmt.   
 Die Punktzahlen werden einstellig ausgegeben.

*Schnittstelleninformationen:*

|  |  |  |
| --- | --- | --- |
| Speisung | Eingabe | Ausgabe |
|  | Punktzahl-Signal1 | Punktzahl1 |
|  | Punktzahl-Signal2 | Punktzahl2 |
| 4.50V DC | Optische-Meldung-Signal1 | Optische Meldung1 |
|  | Optische-Meldung-Signal2 | Optische Meldung2 |
|  | Reset-Signal (aktiv LOW) |  |

### 4.3.2 Software

**Arbeitspaket 3: Start**

*Vorgaben:*Zum Starten wird ein Start-Signal benötigt. Wenn das Programm startet soll ein optisches Start-Signal auftreten. Dieses Blinkt drei Mal.

*Schnittstelleninformationen:*

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Start-Signal | Optische Meldung1 |
|  | Optische Meldung2 |

**Arbeitspaket 4: Spielerauswahl**

*Vorgaben:*Zu Beginn jeder Runde wird zufällig ein Spieler gewählt, welcher beginnt. Danach kommt der andere Spieler.

*Schnittstelleninformationen:*

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| / | / |

**Arbeitspaket 5: Zeitmessung**

*Vorgaben:*Die Reaktionszeit wird zwischen dem optischen Zeitmessung-Start-Signal und dem Quittierungs-Signal gemessen.

*Erweiterungen:*  
 Um die Reaktion richtig zu testen wird die Reaktionszeitmessung zufällig 1 bis 5 Sekunden verspätet gestartet.

*Schnittstelleninformationen:*

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Quittierungs-Signal1 | Reaktionszeit1 |
|  | Reaktionszeit2 |
| Quittierungs-Signal2 | Optische Meldung1 |
|  | Optische Meldung2 |

**Arbeitspaket 6: Zeitmessung-Auswertung**

*Vorgaben:*Die Reaktionszeiten der Spieler werden verglichen. Der Spieler mit der kürzeren Reaktionszeit erhält einen Punkt.

*Erweiterungen:*  
 Die Punktzahl der Spieler wird aus je einem Controllerausgang bestimmt. Das Punktzahl- Signal ist ein HIGH-Impuls, welcher auftritt, wenn um die Punktzahl um einen Punkt erhöht werden soll.

*Schnittstelleninformationen:*

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Reaktionszeit1 | Punktzahl-Signal1 |
|  | Punktzahl-Signal2 |
| Reaktionszeit2 | Punktzahl1 |
|  | Punktzahl2 |

**Arbeitspaket 7: Spielende**

*Vorgaben:*Das Spiel ist beendet, sobald ein Spieler neun Punkte erhalten hat. Die optische Meldung «Gewinner» des Spielers geht an.

*Erweiterungen:*  
 Die optische Meldung «Gewinner» soll leuchten. Wenn die optische Meldung «Gewinner» ausgegeben wird soll ist Spielende abgeschlossen.

*Schnittstelleninformationen:*

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Punktzahl-Signal1 | Optische Meldung1 |
| Punktzahl-Signal2 | Optische Meldung2 |

**Arbeitspaket 8: Neustart**

*Vorgaben:*Nachdem das Spiel beendet wurde, kann das dieses mit einem Start-Signal zurückgesetzt werden.

*Schnittstelleninformationen:*

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Punktzahl1 | Reset-Signal (aktiv LOW) |
|  | Punktzahl-Signal1 |
| Punktzahl2 | Punktzahl-Signal2 |
|  | Optische Meldung1 |
|  | Optische Meldung2 |

### 4.3.3 Zusammenführung

**Arbeitspaket 9: Gesamtprogramm**

*Vorgaben:*Die AP3. Bis AP6. werden zum Gesamtprogramm zusammengeführt.   
 Das Programm soll wie in der Funktionsbeschreibung ablaufen.   
 (siehe «Funktionsbeschreibung»)

*Schnittstelleninformationen:*

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Quittieruns-Signal1 | Punktzahl-Signal1 |
| Quittieruns-Signal2 | Punktzahl-Signal2 |
| Start-Signal | Optische Meldung1 |
|  | Optische Meldung2 |
|  | Reset-Signal (aktiv LOW) |

**Arbeitspaket 10: Gesamt-Hardware**

*Vorgaben:*Die gesamte Hardware soll zusammengeführt werden.

*Schnittstelleninformationen:*

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Quittierungs-Impuls1 | Quittierungs-Signal1 |
| Quittierungs-Impuls2 | Quittierungs-Signal2 |
| Start-Impuls | Start-Signal |
| Punktzahl-Signal1 | Punktzahl1 |
| Punktzahl-Signal2 | Punktzahl2 |
| Optische-Meldung-Signal1 | Optische Meldung1 |
| Optische-Meldung-Signal2 | Optische Meldung2 |
| Reset-Signal (aktiv LOW) |  |

### 4.3.4 Erweiterung Arbeitspaket 4. und Arbeitspaket 5.

Um das Arbeitspaket 4 und das Arbeitspaket 5 miteinander abzustimmen, wurden die *Schnittstelleninformationen* erweitert.

**Arbeitspaket 4: Spielerauswahl**

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
|  | Spieler1 |
|  | Spieler2 |
| / | Zeitmessungstart1 |
|  | Zeitmessungstart2 |
|  | EingangSpieler1 |
|  | EingangSpieler2 |

|  |  |
| --- | --- |
| Variable | Speicher-Name |
| Spieler1 | Spieler[0] |
| Spieler2 | Spieler[1] |
| Zeitmessungstart1 | Zms[0] |
| Zeitmessungstart2 | Zms[1] |
| EingangSpieler1 | EingangSpieler[0] |
| EingangSpieler2 | EingangSpieler[1] |

**Arbeitspaket 5: Zeitmessung**

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Quittierungs-Signal1 | Reaktionszeit1 |
| Quittierungs-Signal2 |  |
| Zeitmessungstart1 | Reaktionszeit2 |
| Zeitmessungstart2 |  |
| EingangsSpieler1 | Optische Meldung1 |
| EingangsSpieler2 |  |
| Spieler1 | Optische Meldung2 |
| Spieler2 |  |

|  |  |
| --- | --- |
| Variable | Speicher-Name |
| Spieler1 | Spieler[0] |
| Spieler2 | Spieler[1] |
| Zeitmessungstart1 | Zms[0] |
| Zeitmessungstart2 | Zms[1] |
| EingangSpieler1 | EingangSpieler[0] |
| EingangSpieler2 | EingangSpieler[1] |

### 4.3.5 Begriffsverzeichnis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Eingabe | Speicher-Name | Pin-Name | Pin | Datentyp |
| Start-Signal | St\_Si | S\_S | 3 | bool |
| Quittieruns-Signal1 | Qu\_Si1 | Q\_S1 | 4 | char |
| Quittieruns-Signal2 | Qu\_Si2 | Q\_S2 | 5 | char |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ausgabe | Speicher-Name | Pin-Name | Pin | Datentyp |
| Punktzahl-Signal1 | Pu\_Si1 | P\_S1 | 6 | char |
| Punktzahl-Signal2 | Pu\_Si2 | P\_S2 | 7 | char |
| Optische Meldung1 | Op\_Me1 | O\_M1 | 8 | char |
| Optische Meldung2 | Op\_Me2 | O\_M2 | 9 | char |
| Reset-Signal (aktiv LOW) | Re\_Si | R\_S | 10 | char |

|  |  |  |
| --- | --- | --- |
| Variable | Speicher-Name | Datentyp |
| Punktzahl1 | Pu\_Za1 | char |
| Punktzahl2 | Pu\_Za2 | char |
| Reaktionszeit1 | Re\_Ze1 | char |
| Reaktionszeit2 | Re\_Ze2 | char |
| Spieler1 | Spieler[0] | char |
| Spieler2 | Spieler[1] | char |
| Zeitmessungstart1 | Zms[0] | char |
| Zeitmessungstart2 | Zms[1] | char |
| EingangSpieler1 | EingangSpieler[0] | char |
| EingangSpieler2 | EingangSpieler[1] | char |

# 5. Arbeitspakete

## 5.1 Arbeitspaket 1: Eingabe

### 5.1.1 Zusammenfassung

Es wurde eine funktionsfähige Schaltung zur Erzeugung dreier digitaler Signale entwickelt. Mit einem Tastendruck wird das jeweilige Signal logisch HIGH.

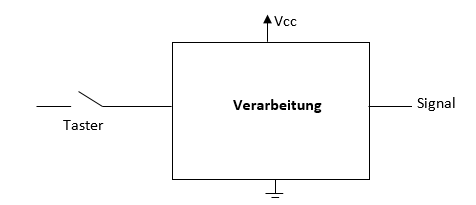
### 5.1.2 Aufgabenstellung

Der physikalische Druck soll in eine digitale Spannung umgewandelt werden. Es soll für beide Spieler ein Taster zur Quittierung zur Verfügung stehen. Mit einem Taster soll das Gerät gestartet werden.

### 5.1.3 Vorgaben

SchnittstelleninformationenSpeisung: 5V DC  
 Eingänge: Quittierungs-Impuls (Spieler1 und Spieler2, Druck), Start-Signal (Druck)  
 Ausgänge: Quittierungs-Signale (Spieler1 und Spieler2),   
 Start-Signal

### 5.1.4 Blockschaltbild



Start-Signal

Quittierungs-Signal 1

Quittierungs-Signal 2



Quittierungs-Impuls1

Quittierungs-Impuls2

Start-Signal (Druck)

Abbildung , Blockschaltbild

### 5.1.5 Lösungsansätze

Mit einem Taster und einem Pull-Down-Widerstand wird die digitale Spannung erzeugt.

### 

### 5.1.6 Berechnungen

**Gegebene Werte:**

5.00V

10.0kΩ

**Berechnungen:**

**1.50 mA**

### 5.1.7 Schaltung

Abbildung 2, Schaltung

Es wurde die obige Schaltung entwickelt. (Abb.2)

### 5.1.8 Messungen



Abbildung 3, Messschaltung

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Quittierung1 | Quittierung2 | Start | AM1 in mA | VM1 in V | VM2 in V | VM3 in V |
| 0 | 0 | 0 | 0.01 | 0 | 0 | 0 |
| gedrückt | 0 | 0 | 0.51 | 5 | 0 | 0 |
| 0 | gedrückt | 0 | 0.50 | 0 | 5 | 0 |
| 0 | 0 | gedrückt | 0.50 | 0 | 0 | 5 |
| gedrückt | gedrückt | gedrückt | 1.52 | 5 | 5 | 5 |

### 5.1.9 Verwendete Geräte

**Multimeter:** Agilent U1242A, BERUF Zug 03542

**Speisegerät:** DC Power Supply DF1730SB3A, BERUF Zug 03460

**Breadboard:** Besitzer Thomas Ammann

### 5.1.10 Schlussfolgerung

Es wurde eine funktionsfähige Schaltung entwickelt.

Die Ausgänge sind logisch LOW bis der jeweilige Taster gedrückt wird. Dabei wird der Ausgang logisch HIGH.

## 5.2 Arbeitspaket 2: Ausgabe

### 5.2.1 Zusammenfassung

Es wurde eine Schaltung entwickelt, welche die Punkzahlen der Spieler 1 und 2 auf je einer   
7-Segmentanzeigen anzeigt. Mit zwei LEDs werden die optische Signale der Spieler angezeigt.

### 5.2.2 Aufgabenstellung

Es muss eine Teilschaltung entwickelt werden, die die Punktzahlen der Spieler 1 und 2 auf Anzeigen anzeigt. Jeder Spieler besitzt eine optische Meldung, die den Beginn der Zeitmessung signalisiert. Jeder Spieler besitzt eine optische Meldung, die signalisiert, wer gewonnen hat.

### 5.2.3 Vorgaben

Erweiterungen:  
 Die Punktzahl der Spieler wird aus je einem Controllerausgang bestimmt.   
 Die Punktzahlen werden einstellig ausgegeben.

Schnittstelleninformationen:Speisung: 5V DC  
 Eingänge: Punktzahl-Signale (Spieler1 und Spieler2),   
 Zeitmessung-Start-Signale (Spieler1 und Spieler2),   
 Gewinner-Signale (Spieler1 und Spieler2),  
 Beginn-Signal (Spieler1 und Spieler2), Reset-Signal  
 Ausgänge: Punktzahlen (einstellig, Spieler1 und Spieler2),  
 optische Meldungen für «Zeitmessung-Start» (Spieler1 und Spieler2),  
 optische Meldungen für «Gewinner» (Spieler1 und Spieler2),  
 optisches Beginn-Signal (Spieler1 und Spieler2)

### 5.2.4 Blockschaltbild

optische Meldung Spieler 1

Vcc 5V

PZ-Signal Spieler 2

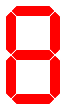
PZ-Signal Spieler 1

**Verarbeitung**

ZS-Signal Spieler 2

ZS-Signal Spieler 1

Punktzahl Spieler 1



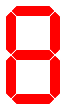
GW-Signal Spieler 2

GW-Signal Spieler 1

Beginn-Signal Spieler 2

Beginn-Signal Spieler 1

Punktzahl Spieler 2



Reset-Signal

optische Meldung Spieler 2

* ZS-Signale Spieler 1,2: Zeitmessung-Start-Signale Spieler 1,2
* PZ-Signale Spieler 1,2: Punktzahl-Signale Spieler 1,2
* GW-Signale Spieler 1,2: Gewinner-Signale Spieler 1,2

### 5.2.5 Berechnungen

Für LEDs müssen immer Vorwiderstände verwendet werden, welche dafür sorgen, dass an der LED nur die zulässige Spannung anliegt. Durch diese Vorwiderstände wird auch der Strom Idurch die LEDs begrenzt, sodass den Strom nicht über den maximalen zulässigen Strom der LED ansteigt. Für Inehme ich in meinen Berechnungen den Erfahrungswert 7mA.

**Kennwerte der roten LED:**

UF = 2.1V (Datenblatt, Quellenverzeichnis)

I = 7mA (Erfahrungswert)

\*1U0 (Speisung)= 5V (Vorgabe)

**414Ω**

Aus unserer E-24 Widerstandsreihe haben wir einen **R1,2=430 Ω** gewählt.

**6.74 mA**

**\*1** 5V Betriebsspannung von Arduino UNO Rev.3

**Pull-Down Widerstände:**

Da recht hohe Frequenzen zum Einsatz kommen die sich in unmittelbarer Nähe der Eingänge befinden ist es notwendig diese Eingänge mit einem definierten Potential zu versehen. Klassischer Weise wird ein Widerstand verwendet, welcher zum Beispiel auf Masse gelegt wird. Ein so genannter Pull Down Widerstand. Ich habe für Pull-down Widerstände R17=R18=R19=10kΩ entschieden.

**Vorwiderstände 7-Segment-Anzeige:**

U’F = 2.1V (Datenblatt, Quellenverzeichnis)

I’ = 5mA (Erfahrungswert)

\*1U0 (Speisung)= 5V (Vorgabe)

**580Ω**

Aus unserer E-24 Widerstandsreihe haben wir einen **R3-16=620 Ω** gewählt.

**4.67 mA**

### 5.2.6 Schaltung



### 5.2.7 Messung

|  |  |  |
| --- | --- | --- |
| Clock-Signal | 7-Segment Zahl Spieler 1 | 7-Segment Zahl Spieler 2 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 1 | 2 | 2 |
| 0 | 2 | 2 |
| 1 | 3 | 3 |
| 0 | 3 | 3 |
| 1 | 4 | 4 |
| 0 | 4 | 4 |
| 1 | 5 | 5 |
| 0 | 5 | 5 |
| 1 | 6 | 6 |
| 0 | 6 | 6 |
| 1 | 7 | 7 |
| 0 | 7 | 7 |
| 1 | 8 | 8 |
| 0 | 8 | 8 |
| 1 | 9 | 9 |
| 0 | 9 | 9 |
| 1 | 0 | 0 |

### 5.2.8 Verwendete Geräte und Elektrokomponente

|  |  |  |
| --- | --- | --- |
| Digitrainer 6 BERUF Zug 00878 | 2x 7-Segment Anzeigen | 2x BCD-Decoder 74HC4511 |
| 2x LEDs rot 5mm | 2x 74HC162 | 3x Taster |
| 2x R1=R2=430Ω | 14x R3-R16=620Ω | 3x Pull-Down Widerstände R17-R19=10kΩ |

### 5.2.9 Schlussfolgerung

Die entwickelte Schaltung erfüllt die Vorgaben (siehe 3. Vorgaben/Seite 14).   
Sie wurde mit einem Digitrainer komplett getestet.

## 5.3 Arbeitspaket 3: Start

### 5.3.1 Zusammenfassung

Es wurde ein Software-Programm entwickelt, welches ein Start-Signal benötigt. Wenn das Programm startet, blinken zwei LEDs drei Mal.

### 5.3.2 Aufgabenstellung

Zum Starten wird ein Start-Signal benötigt. Wenn das Programm startet soll ein optisches Start-Signal auftreten. Dieses Blinkt drei Mal.

Schnittstelleninformationen  
Eingänge: Start-Signal (aktiv LOW)  
Ausgänge: Beginn-Signal (Spieler1 und Spieler2)

### 5.3.3 Begriffsverzeichnis

|  |  |  |  |
| --- | --- | --- | --- |
| Eingabe | Speicher-Name | Pin-Name | Pin |
| Start-Signal | St\_Si | S\_S | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
| Ausgabe | Speicher-Name | Pin-Name | Pin |
| Optische Meldung1 | Op\_Me1 | O\_M1 | 8 |
| Optische Meldung2 | Op\_Me2 | O\_M2 | 9 |

### 5.3.4 Struktogramm

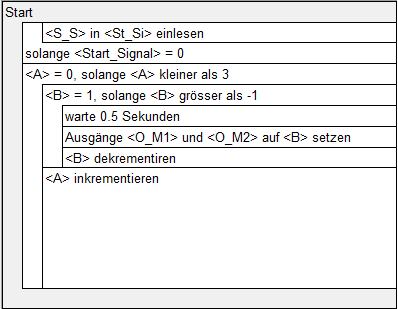


Abbildung , Struktogramm

### 5.3.5 Programm–Code

Um eine Variable zu erstellen, müssen wir diese zuerst deklarieren. Wir haben für die Variablen  
Start-Signal, A und B den Datentyp **char** gewählt, da die Variablen nicht mehr Speicherplatz benötigen (1 Byte).

**bool** St\_Si ; // Variablen

**char** A,

B;

Die Pin-Nummern der Ein- und Ausgänge sind den Pin-Namen gespeichert. Diese sind unveränderbar.

**const char** S\_S = 3, // Konstanten

O\_M1 = 8,

O\_M2 = 9;

**setup()** wird einmalig beim Einschalten des Arduino ausgeführt. Dort definieren wir mit der Funktion **pinMode()**, ob die digitalen Pins Ein- oder Ausgänge sind.

**void setup()** { // I/O

**pinMode** (S\_S, INPUT); // Eingaenge

**pinMode** (O\_M1, OUTPUT); // Ausgaenge

**pinMode** (O\_M2, OUTPUT);

}

**loop()** wird wiederholt ausgeführt. In der Funktion wird das Hauptprogramm ausgeführt.   
Die **do-while**-Schleife liest den Digitalwert am Eingang «S\_S» ein und speichert ihn in «St\_Si». Danach wartet das Programm 50ms, bevor es überprüft, ob «St\_Si» 0 entspricht. Trifft dies zu, wird die Schleife wiederholt.

**void loop()** { // Hauptprogramm

**do** { // Start-Signal einlesen

St\_Si = **digitalRead**(S\_S);

**delay** (50);

} **while** (St\_Si == 0);

Die erste **for**-Schleife wird dreimal durchgeführt. Die zweite Schleife wird zweimal durchgeführt. Im erstem Durchlauf werden die Ausgänge «O\_M1» und «O\_M2» eingeschaltet. Im zweiten Durchlauf werden diese ausgeschaltet. Die **delay()**–Funktion lässt den Arduino die Ausführung des Programms für eine gewisse Zeitspanne anhalten. Damit wird der Logikwechsel der Ausgänge verlangsamt.

**for** (A = 0; A < 3; A++) { // Optische Meldung

**for** (B = 1; B > -1; B--) {

**delay** (500);

**digitalWrite** (O\_M1, B);

**digitalWrite** (O\_M2, B);

}

}

}

### 5.3.6 Schlussfolgerung

Das entwickelte Software-Programm erfüllt die Anforderungen der Aufgabenstellung.  
(2. Aufgabenstellung/Seite 19).

## 5.4 Arbeitspaket 4: Spielerauswahl

### 5.4.1 Zusammenfassung

Es wurde ein Software-Programm entwickelt, welches zufällig einen Spieler wählt. Dieser beginnt die Runde.

### 5.4.2 Aufgabenstellung

Zu Beginn jeder Runde wird zufällig ein Spieler gewählt, welcher beginnt. Danach kommt der andere Spieler.

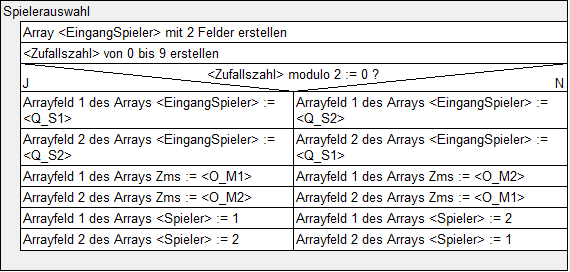
Schnittstelleninformationen

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| / | Spieler1 |
| / | Spieler2 |
| / | Zeitmessungstart1 |
| / | Zeitmessungstart2 |
| / | EingangSpieler1 |
| / | EingangSpieler2 |

Begriffserklärung

|  |  |
| --- | --- |
| Variable | Speicher-Name |
| Spieler1 | Spieler[0] |
| Spieler2 | Spieler[1] |
| Zeitmessungstart1 | Zms[0] |
| Zeitmessungstart2 | Zms[1] |
| EingangSpieler1 | EingangSpieler[0] |
| EingangSpieler2 | EingangSpieler[1] |

### 5.4.3 Struktogramm



### 5.4.4 Programm–Code

Um eine Variable zu erstellen, müssen wir diese zuerst deklarieren. Wir haben für die Variablen  
den Datentyp **char** gewählt, da die Variablen nicht mehr Speicherplatz benötigen (1 Byte).

**char** EingangSpieler[2],// Variablen

Zms[2],

Spieler[2];

**unsigned long** Zufallszahl;

Die Pin-Nummern der Ein- und Ausgänge sind den Pin-Namen gespeichert. Diese sind unveränderbar.

**const char** Q\_S1 = 4,// Konstanten

Q\_S2 = 5,

O\_M1 = 8,

O\_M2 = 9;

**setup()** wird einmalig beim Einschalten des Arduino ausgeführt. Mit der Funktion **randomSeed()** wird der Analogeingang definiert, welcher für die Erstellung des «zufälligen» Werts verwendet wird. Dieser Eingang muss logisch unbestimmt sein, darf also nicht angeschlossen sein.

**void setup()** { // I/O

**randomSeed**(**analogRead**(0)); // Der analog-Input 0 darf nicht angeschlossen werden.

}

**loop()** wird wiederholt ausgeführt. In der Funktion wird das Hauptprogramm ausgeführt.   
Die Funktion **random()** generiert einen «zufälligen» Wert von 0 bis 99. Dieser wird in «Zufallszahl» gespeichert. Mit der Variable «Zufallszahl» wird einem modulo-2-Rechnung durchgeführt. Ist das Ergebnis 0 beginnt Spieler 1. Ist das Ergebnis 1 beginnt Spieler 2.

**void loop()** { // Hauptprogramm

Zufallszahl = **random**(100);

**if**(Zufallszahl % 2 == 0){// Auswertung

EingangSpieler[0] = Q\_S1;

EingangSpieler[1] = Q\_S2;

Zms[0] = O\_M1;

Zms[1] = O\_M2;

Spieler[0] = 1;

Spieler[1] = 2;

}

**else**{

EingangSpieler[0] = Q\_S2;

EingangSpieler[1] = Q\_S1;

Zms[0] = O\_M2;

Zms[1] = O\_M1;

Spieler[0] = 2;

Spieler[1] = 1;

}

}

### 5.4.5 Schlussfolgerung

Das entwickelte Software-Programm erfüllt die Anforderungen der Aufgabenstellung  
(2. Aufgabenstellung/Seite 21).

## 5.5 Arbeitspaket 5: Zeitmessung

### 5.5.1 Zusammenfassung

Es wurde ein Software-Programm entwickelt, welches die Reaktionszeit der beiden Spieler misst.

### 5.5.2 Aufgabenstellung

Die Reaktionszeit wird zwischen dem optischen Zeitmessung-Start-Signal und dem Quittierungs-Signal gemessen.

Um die Reaktion richtig zu testen wird die Reaktionszeitmessung zufällig 1 bis 5 Sekundenverspätet gestartet.

Schnittstelleninformationen

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Quittierungs-Signal1 | Reaktionszeit1 |
| Quittierungs-Signal2 | Reaktionszeit2 |
| Zeitmessungstart1 | Optische Meldung1 |
| Zeitmessungstart2 | Optische Meldung2 |
| EingangsSpieler1 |  |
| EingangsSpieler2 |  |
| Spieler1 |  |
| Spieler2 |  |

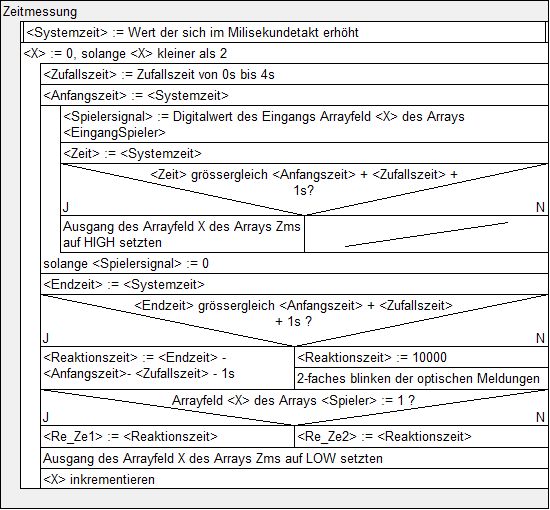
Begriffserklärung

|  |  |
| --- | --- |
| Variable | Speicher-Name |
| Spieler1 | Spieler[0] |
| Spieler2 | Spieler[1] |
| Zeitmessungstart1 | Zms[0] |
| Zeitmessungstart2 | Zms[1] |
| EingangSpieler1 | EingangSpieler[0] |
| EingangSpieler2 | EingangSpieler[1] |
| Reaktionszeit1 | Re\_Ze1 |
| Reaktionszeit2 | Re\_Ze2 |

|  |  |  |  |
| --- | --- | --- | --- |
| Eingabe | Speicher-Name | Pin-Name | Pin |
| Quittieruns-Signal1 | Qu\_Si1 | Q\_S1 | 4 |
| Quittieruns-Signal2 | Qu\_Si2 | Q\_S2 | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
| Ausgabe | Speicher-Name | Pin-Name | Pin |
| Optische Meldung1 | Op\_Me1 | O\_M1 | 8 |
| Optische Meldung2 | Op\_Me2 | O\_M2 | 9 |

### 5.5.3 Struktogramm



### 5.5.4 Programm – Code

Zunächst werden die Variablen definiert.

**char** X,// Variablen

Spielersignal,

Zms[2], // Zeitmessungstart

Spieler[2];

**int** Zufallszeit,

Re\_Ze1,

Re\_Ze2;

**unsigned long** Anfangszeit, // Anfangszeit der Zeitmessung

Endzeit, // Endzeit der Zeitmessung

Zeit;

Die Pin-Nummern der Ein- und Ausgänge sind den Pin-Namen gespeichert. Diese sind unveränderbar.

**const char** Q\_S1 = 4,// Konstanten

Q\_S2 = 5,

O\_M1 = 8,

O\_M2 = 9;

**setup()** wird einmalig beim Einschalten des Arduino ausgeführt. Dort definieren wir mit der Funktion **pinMode()**, ob die digitalen Pins Ein- oder Ausgänge sind. Mit der Funktion **randomSeed()** wird der Analogeingang definiert, welcher für die Erstellung des «zufälligen» Werts verwendet wird. Dieser Eingang muss logisch unbestimmt sein, darf also nicht angeschlossen sein.

**void setup()** { // I/O

**pinMode**(Q\_S1, INPUT); // Eingaenge

**pinMode**(Q\_S2, INPUT);

**pinMode**(O\_M1, OUTPUT); // Ausgaenge

**pinMode**(O\_M2, OUTPUT);

**randomSeed**(**analogRead**(0)); // Der analogInput 0 darf nicht angeschlossen werden.

}

**loop()** wird wiederholt ausgeführt. In der Funktion wird das Hauptprogramm ausgeführt.   
Die **for**-Schleife wird zweimal durchgeführt. Die Funktion **random()** erzeugt einen «zufälligen» Wert von 0 bis 4000. Die Funktion **millis()** erzeugt einen Wert, welcher jede Millisekunde inkrementiert wird. «Anfangszeit» entspricht dem Wert von **millis()**.

**void loop()** { // Hauptprogramm

**for**(X = 0; X < 2; X++){ // Auswertung

Zufallszeit = **random**(4001);

Anfangszeit = **millis**();

«Zeit» entspricht dem aktuellen Wert von **millis()**. Die **if**-Verzweigung prüft, ob «Zeit» grössergleich «Anfangszeit» + «Zufallszeit» + 1000 entspricht. Trifft dies zu wird ein Zeitmessungsstart-Signal ausgegeben. Die **do-while**-Schleife wird wiederholt solange «Spielersignal» 0 entspricht. Danach wird der Wert von **millis()** «Endzeit» zugeordnet.

**do**{

Spielersignal = **digitalRead**(EingangSpieler[X]);

Zeit = **millis**();

if(Zeit >= Anfangszeit + Zufallszeit + 1000){

**digitalWrite**(Zms[X], HIGH);

}

}**while**(Spielersignal == 0);

Endzeit = **millis**();

Die **if**-Verzweigung prüft, ob «Endzeit» grössergleich «Anfangszeit» + «Zufallszeit» + 1000. Trifft dies zu wird die Reaktionszeit ausgerechnet und in «Reaktionszeit» gespeichert. Trifft der Vergleich nicht zu wird «Reaktionszeit» auf 10000 (10s) gesetzt. Dabei werden zwei optische Signale ausgegeben.

**if**(Endzeit >= Anfangszeit + Zufallszeit + 1000){

Reaktionszeit = Endzeit - Anfangszeit - Zufallszeit - 1000;

}

**else**{

Reaktionszeit = 10000;

**digitalWrite**(O\_M1, HIGH);

**digitalWrite**(O\_M2, HIGH);

**delay**(300);

**digitalWrite**(O\_M1, LOW);

**digitalWrite**(O\_M2, LOW);

**delay**(300);

**digitalWrite**(O\_M1, HIGH);

**digitalWrite**(O\_M2, HIGH);

**delay**(300);

**digitalWrite**(O\_M1, LOW);

**digitalWrite**(O\_M2, LOW);

}

Die **if**-Verzweigung prüft welchem Spieler «Reaktionszeit» zuzuordnen ist und speichert diese in die entsprechende Variable. Danach wird das Zeitmessungsstart-Signal auf logisch LOW gesetzt.

**if**(Spieler[X] = 1){

Re\_Ze1 = Reaktionszeit;

}

**else**{

Re\_Ze2 = Reaktionszeit;

}

**digitalWrite**(Zms[X], LOW);

}

}

### 5.5.5 Schlussfolgerung

Das entwickelte Software-Programm erfüllt die Anforderungen der Aufgabenstellung.

## 5.6 Arbeitspaket 6: Zeitmessung-Auswertung

### 5.6.1 Zusammenfassung

Es wurde ein Software-Programm entwickelt, welches die Reaktionszeiten zweier Spieler vergleicht und auswertet. Derjenige Spieler mit der kleineren Reaktionszeit erhält einen Punkt.

### 5.6.2 Aufgabenstellung

Die Reaktionszeiten der Spieler werden verglichen. Der Spieler mit der kürzeren Reaktionszeit erhält einen Punkt.

Die Punktzahl der Spieler wird aus je einem Controllerausgang bestimmt. Das Punktzahl-Signal ist ein HIGH-Impuls, welcher auftritt, wenn die Punktzahl um einen Punkt erhöht werden soll.

Schnittstelleninformationen

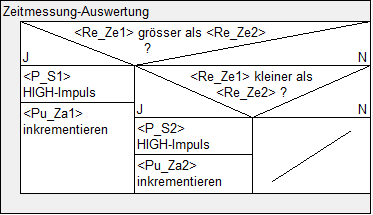
|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Reaktionszeit1 | Punktzahl-Signal1 |
| Reaktionszeit2 | Punktzahl-Signal2 |
|  | Punktzahl1 |
|  | Punktzahl2 |

Begriffserklärung

|  |  |
| --- | --- |
| Variable | Speicher-Name |
| Punktzahl1 | Pu\_Za1 |
| Punktzahl2 | Pu\_Za2 |
| Reaktionszeit1 | Re\_Ze1 |
| Reaktionszeit2 | Re\_Ze2 |

|  |  |  |  |
| --- | --- | --- | --- |
| Ausgabe | Speicher-Name | Pin-Name | Pin |
| Punktzahl-Signal1 | Pu\_Si1 | P\_S1 | 6 |
| Punktzahl-Signal2 | Pu\_Si2 | P\_S2 | 7 |

### 5.6.3 Struktogramm



### 5.6.4 Programm–Code

Zunächst werden die Variablen definiert.

**int** Re\_Ze1, // Variablen

Re\_Ze2 ;

**char** Pu\_Za1 = 0,

Pu\_Za2 = 0 ;

Die Pin-Nummern der Ein- und Ausgänge sind den Pin-Namen gespeichert. Diese sind unveränderbar.

**const char** P\_S1 = 6, // Konstanten

P\_S2 = 7;

**setup()** wird einmalig beim Einschalten des Arduino ausgeführt. Dort definieren wir mit der Funktion **pinMode()**, ob die digitalen Pins Ein- oder Ausgänge sind.

**void setup()** { // I/O

**pinMode**(P\_S1, OUTPUT); // Ausgaenge

**pinMode**(P\_S2, OUTPUT);

}

**loop()** wird wiederholt ausgeführt. In der Funktion wird das Hauptprogramm ausgeführt.   
Die **if**-Verzweigung prüft, ob «Re\_Ze1» grösser als «Re\_Ze2» ist. Trifft dies zu wird ein Punktzahl-Signal ausgegeben und die Punktzahl des ersten Spielers wird inkrementiert. Die **else-if**-Verzweigung prüft, ob «Re\_Ze1» kleiner als «Re\_Ze2» ist. Trifft dies zu wird ein Punktzahl-Signal ausgegeben und die Punktzahl des zweiten Spielers wird inkrementiert.

**void loop()** { // Hauptprogramm

**if**(Re\_Ze1 > Re\_Ze2){ // Auswertung

**digitalWrite**(P\_S1, HIGH);

**delay**(10);

**digitalWrite**(P\_S1, LOW);

Pu\_Za2++;

}

**else if**(Re\_Ze1 < Re\_Ze2){

**digitalWrite**(P\_S2, HIGH);

**delay**(10);

**digitalWrite**(P\_S2, LOW);

Pu\_Za1++;

}

### 5.6.5 Schlussfolgerung

Das entwickelte Software-Programm erfüllt die Anforderungen der Aufgabenstellung  
(2. Aufgabenstellung/Seite 29).

## 5.7 Arbeitspaket 7: Spielende

### 5.7.1 Zusammenfassung

Es wurde ein Software-Programm entwickelt, welches das Spielende auswertet. Das Programm gibt eine optische Meldung aus, wenn ein Spieler neun Punkte erhält.

### 5.7.2 Aufgabenstellung

Das Spiel ist beendet, sobald ein Spieler neun Punkte erhalten hat. Die optische Meldung «Gewinner» des Spielers geht an.

Schnittstelleninformationen  
Die optische Meldung «Gewinner» soll leuchten. Wenn die optische Meldung «Gewinner» ausgegeben wird soll ist Spielende abgeschlossen.

|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Punktzahl-Signal1 | Optische Meldung1 |
| Punktzahl-Signal2 | Optische Meldung2 |

Begriffsverzeichnis

|  |  |
| --- | --- |
| Variable | Speicher-Name |
| Punktzahl1 | Pu\_Za1 |
| Punktzahl2 | Pu\_Za2 |

|  |  |  |  |
| --- | --- | --- | --- |
| Ausgabe | Speicher-Name | Pin-Name | Pin |
| Optische Meldung1 | Op\_Me1 | O\_M1 | 8 |
| Optische Meldung2 | Op\_Me2 | O\_M2 | 9 |

### 5.7.3 Struktogramm

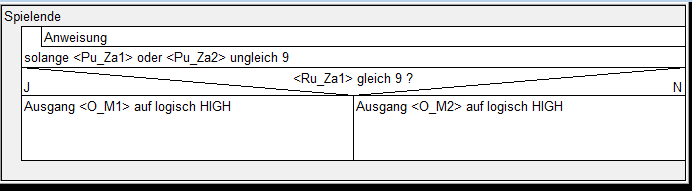


Abbildung , Struktogramm

### 5.7.4 Programm–Code

In den Variablen «Pu\_Za1» und «Pu\_Za1» werden die Punktzahlen der Spieler gespeichert. Da diese minimal 0 und maximal 9 sein können genügen Variablen vom Datentyp **char**.

**char** Pu\_Za1, // Variablen

Pu\_Za2;

Die Pin-Nummern der Ein- und Ausgänge sind den Pin-Namen gespeichert. Diese sind unveränderbar.

**const char** O\_M1 = 8, // Konstanten

O\_M2 = 9;

**setup()** wird einmalig beim Einschalten des Arduino ausgeführt. Dort definieren wir mit der Funktion **pinMode()**, ob die digitalen Pins Ein- oder Ausgänge sind.

**void setup ()** { // I/O

**pinMode** (O\_M1, OUTPUT); // Ausgaenge

**pinMode** (O\_M2, OUTPUT);

}

**loop()** wird wiederholt ausgeführt. In der Funktion wird das Hauptprogramm ausgeführt.   
Die **do-while**-Schleife überprüft, ob «Pu-Za1» oder «Pu\_Za2» 9 entspricht. Trifft dies zu, fragt die **if**-Verzweigung, welche der beiden Variablen 9 entspricht. Danach wird eine optische Meldung ausgegeben.

**void loop()** { / // Hauptprogramm

**do**{ // Auswertung

// Spielerauswahl, Zeitmessung, Zeitmessung-Auswertung

}**while**(Pu\_Za1 == 9 || Pu\_Za2 == 9);

**if**(Pu\_Za1 == 9){

**digitalWrite**(O\_M1, HIGH);

}

**else**{

**digitalWrite**(O\_M2, HIGH);

}

}

### 5.7.5 Schlussfolgerung

Das entwickelte Software-Programm erfüllt die Anforderungen der Aufgabenstellung.  
(2. Aufgabenstellung/ Seite 31).

## 5.8 Arbeitspaket 8: Neustart

### 5.8.1 Zusammenfassung

Es wurde ein Software-Programm entwickelt, welches einen Neustart durchführt. Das Programm setzt die Punktzahlen und die Anzeigen der Spieler auf 0 zurück.

### 5.8.2 Aufgabenstellung

Nachdem das Spiel beendet wurde, kann das dieses mit einem Start-Signal zurückgesetzt werden.

Schnittstelleninformationen

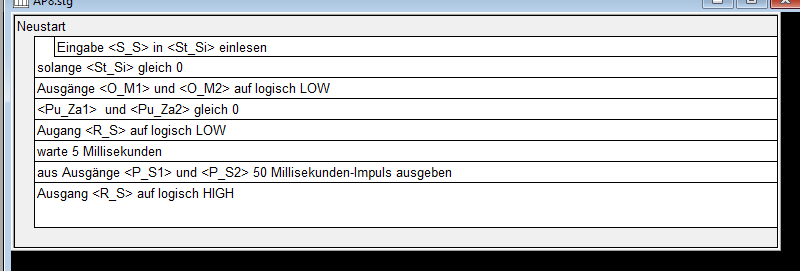
|  |  |
| --- | --- |
| Eingabe | Ausgabe |
| Punktzahl1 | Reset-Signal (aktiv LOW) |
| Punktzahl2 | Punktzahl-Signal1 |
|  | Punktzahl-Signal2 |
|  | Optische Meldung1 |
|  | Optische Meldung2 |

Begriffsverzeichnis

|  |  |  |  |
| --- | --- | --- | --- |
| Ausgabe | Speicher-Name | Pin-Name | Pin |
| Punktzahl-Signal1 | Pu\_Si1 | P\_S1 | 6 |
| Punktzahl-Signal2 | Pu\_Si2 | P\_S2 | 7 |
| Optische Meldung1 | Op\_Me1 | O\_M1 | 8 |
| Optische Meldung2 | Op\_Me2 | O\_M2 | 9 |
| Reset-Signal (aktiv LOW) | Re\_Si | R\_S | 10 |

|  |  |
| --- | --- |
| Variable | Speicher-Name |
| Punktzahl1 | Pu\_Za1 |
| Punktzahl2 | Pu\_Za2 |

### 5.8.3 Struktogramm



### 5.8.4 Programm–Code

Um eine Variable zu erstellen, müssen wir diese zuerst deklarieren. Wir haben für die Variablen  
Start-Signal, Punktzahl1 und B den Datentyp **char** gewählt, da die Variablen nicht mehr Speicherplatz benötigen (1 Byte).

**char** St\_Si, // Variablen

Pu\_Za1,

Pu\_Za2;

Die Pin-Nummern der Ein- und Ausgänge sind den Pin-Namen gespeichert. Diese sind unveränderbar.

**const char** S\_S = 3, // Konstanten

P\_S1 = 6,

P\_S2 = 7,

O\_M1 = 8,

O\_M2 = 9,

R\_S = 10;

**setup()** wird einmalig beim Einschalten des Arduino ausgeführt. Dort definieren wir mit der Funktion **pinMode()**, ob die digitalen Pins Ein- oder Ausgänge sind.

**void setup ()** { // I/O

**pinMode** (S\_S, INPUT); // Eingaenge

**pinMode** (P\_S1, OUTPUT); // Ausgaenge

**pinMode** (P\_S2, OUTPUT);

**pinMode** (O\_M1, OUTPUT);

**pinMode** (O\_M2, OUTPUT);

**pinMode** (R\_S, OUTPUT);

}

**loop()** wird wiederholt ausgeführt. In der Funktion wird das Hauptprogramm ausgeführt.   
Der Inhalt der **do-while**-Schleife liest den Digitalwert am Eingang «S\_S» ein und speichert ihn in «St\_Si». Dies wird solange ausgeführt, bis «St\_Si» 0 entspricht. Danach werden die optischen Meldungen ausgeschaltet und die Punktzahlen und die Anzeigen auf 0 zurückgesetzt.

**void loop()** { // Hauptprogramm

**do**{ // Auswertung

St\_Si = **digitalRead(**S\_S);

}**while** (St\_Si == 0);

**digitalWrite** (O\_M1, LOW); // Optische Meldung ausschalten

**digitalWrite** (O\_M2, LOW);

Pu\_Za1 = 0; // Punktzahl1 zurücksetzten

Pu\_Za2 = 0; // Punktzahl2 zurücksetzten

**DigitalWrite** (R\_S, LOW); // Anzeigen zurücksetzten

**delay** (5);

**digitalWrite** (P\_S1, HIGH);

**digitalWrite** (P\_S2, HIGH);

**delay** (50);

**digitalWrite** (P\_S1, LOW);

**digitalWrite** (P\_S2, LOW);

**digitalWrite** (R\_S, HIGH);

}

### 5.8.5 Schlussfolgerung

Das entwickelte Software-Programm erfüllt die Anforderungen der Aufgabenstellung  
(2. Aufgabenstellung/ Seite 33).

## 5.9 Gesamt-Hardware



Die Zusammenführung der Hardware verlief problemlos.

## 5.10 Gesamt-Software

**Anpassungen**

Bei der Zusammenführung der Arbeitspakete AP3. bis AP8. wurden Änderungen vorgenommen.   
Das Programm besitzt nun drei Unterprogramme («Start\_Taster()»,«Blinken()»,«Zuruecksetzung()»). Diese verkürzen den Programm-Code.

|  |  |
| --- | --- |
| /\* Projektnamen: Reaktionsspiel  \* Dateinamen: Gesamtprogramm\_Reaktionsspiel.ino  \* Entwickler: T. Ammann, I. Christodoulakis  \* Datum: 29.05.2017  \* Ort: 6301 Zug, CH  \* Entwicklungsschritt: Zusammenführung, 29.05.2017  \* Formatierung, 29.05.2017  \* Erweiterung um Unterprogramm  \* "Start\_Taster", "Blinken" &  \* "Zuruecksetzung"  \* Letztes Aenderungsdatum: 29.05.2017  \*/  /\* Variablen ------------------------------------------------------------------- \*/  **bool** St\_Si; // Start-Signal  **char** EingangSpieler[2],  Spieler[2],  Zms[2],  Spielersignal;  **char**  Pu\_Za1 = 0, // Punktzahl1  Pu\_Za2 = 0; // Punktzahl2  **char** a, // fuer for-Schleifen {  b,  c,  x,  y; // }  **unsigned long** Anfangszeit, // Anfangszeit der Zeitmessung  Endzeit, // Endzeit der Zeitmessung  Zeit,  Zufallszahl, // zufaellig generierte Zahl  Re\_Ze1, // Reaktionszeit1  Re\_Ze2, // Reaktionszeit2  Zufallszeit; // zufaellig generierte Zeit (0 <= Wert < 4001)  /\* -------------------------------------------------------------------------------- \*/ | Die Variable «St\_Si» ist vom Datentyp «bool», da sie nur die Werte 0 & 1 besitzen kann.  Die Variablen vom Daten-Typ «char», da sie nur einstellige Dezimal-Werte annehmen können.  Die Variablen vom Datentyp «unsigned long» speichern Zeiten, welche von der Funktion «millis()» bestimmt werden. «millis()» gibt den Wert in «unsigned long» aus. |

|  |  |
| --- | --- |
| /\* Konstanten ---------------------------------------------------------------- \*/  **const char** S\_S = 3, // Eingaenge {  Q\_S1 = 4,  Q\_S2 = 5, // }  P\_S1 = 6, // Ausgaenge {  P\_S2 = 7,  O\_M1 = 8,  O\_M2 = 9,  R\_S = 10; // }  /\* -------------------------------------------------------------------------------- \*/  /\* Setup ----------------------------------------------------------------------- \*/  **void setup()**{  /\* Eingaenge ----------------------------------------------------------------- \*/  **pinMode**(S\_S, INPUT);  **pinMode**(Q\_S1, INPUT);  **pinMode**(Q\_S2, INPUT);  /\* Ausgaenge ----------------------------------------------------------------- \*/  **pinMode**(P\_S1, OUTPUT);  **pinMode**(P\_S2, OUTPUT);  **pinMode**(O\_M1, OUTPUT);  **pinMode**(O\_M2, OUTPUT);  **pinMode**(R\_S, OUTPUT);  /\* Pin fuer Zufallszahl ------------------------------------------------------ \*/  **randomSeed**(**analogRead**(0)); // analogPin0 nicht anschliessen!  }  /\* -------------------------------------------------------------------------------- \*/ | Die Konstanten sind im Datentyp «char», da sie einfache Werte (Charakter) speichern.  Im Setup werden die Ein- und die Ausgänge definiert.  Mit der Funktion «randomSeed()» wird die Referenz für die Zufallszahlgenerierung definiert. |

|  |  |
| --- | --- |
| /\* Unterprogrammdeklaration ------------------------------------------ \*/  // Start-Signal einlesen  **void** Start\_Taster(**char** Start\_Pin);  // Optische Meldung ein- & ausschalten  **void** Blinken(**void**);  //Anzeigen zuruecksetzten  **void** Zuruecksetzung(**char** Punktzahl\_Pin1, **char** Punktzahl\_Pin2, **char** Reset\_Pin);  /\* -------------------------------------------------------------------------------- \*/  /\* Hauptprogramm =================================== \*/  **void loop()**{  Zuruecksetzung(P\_S1, P\_S2, R\_S);  /\* AP3. ------------------------------------------------------------------------- \*/  Start\_Taster(S\_S);  **for**(a = 0; a < 3; a++){ // Optische Meldung ausgeben  Blinken();  }  /\* -------------------------------------------------------------------------------- \*/  /\* AP7. ------------------------------------------------------------------------- \*/  **do**{  /\* AP4. ------------------------------------------------------------------------- \*/  Zufallszahl = **random**(100);  **if**(Zufallszahl %2 == 0){  EingangSpieler[0] = Q\_S1;  EingangSpieler[1] = Q\_S2;  Zms[0] = O\_M1;  Zms[1] = O\_M2;  Spieler[0] = 1;  Spieler[1] = 2;  }  **else**{  EingangSpieler[0] = Q\_S2;  EingangSpieler[1] = Q\_S1;  Zms[0] = O\_M2;  Zms[1] = O\_M1;  Spieler[0] = 2;  Spieler[1] = 1;  }  /\* -------------------------------------------------------------------------------- \*/ | Die Unterprogramme «Start\_Taster», «Blinken» & «Zuruecksetzung» werden deklariert.  Diese Funktionen werden im Programm mehrmals verwendet. Um den benötigten Speicher zu verkleinern werden Unterprogramme erstellt.  Das Hauptprogramm befindet sich in der Funktion «loop()». Diese wird ständig wiederholt.  AP3. führt «Start\_Taster» aus. Danach wird «Blinken» drei mal ausgeführt.  AP4. generiert zunächst eine Zufallszahl zwischen 0 und 99.  Abhängig davon, ob die Zahl gerade oder ungerade ist werden die Arrays «EingangSpieler», «Zms» & «Spieler» einem Wert zugeordnet. |

|  |  |
| --- | --- |
| /\* AP5. ------------------------------------------------------------------------- \*/  **for**(x = 0; x < 2; x++){  Zufallszeit = **random**(4001);  Anfangszeit = **millis()**;  **do**{  Spielersignal = **digitalRead**(EingangSpieler[x]);  Zeit = **millis()**;  **if**(Zeit >= Anfangszeit + Zufallszeit+ 1000){  **digitalWrite**(Zms[x], HIGH);  }  }**while**(Spielersignal == 0);  Endzeit = **millis()**;  **if**(Endzeit >= Anfangszeit+ Zufallszeit + 1000){  Reaktionszeit = Endzeit – Anfangszeit – Zufallszeit – 1000;  }  **else**{  Reaktionszeit = 10000;  **for**(c = 0; c <2; c++){  Blinken();  }  }  **if**(Spieler[x] == 1){  Re\_Ze1 = Reaktionszeit;  }  **else**{  Re\_Ze2 = Reaktionszeit;  }  **digitalWrite**(Zms[x], LOW);  }  /\* -------------------------------------------------------------------------------- \*/ | AP5. führt die Zeitmessung durch. Diese wird zwei Mal wiederholt.  Zunächst wird eine Zufällige Zahl zwischen 0 und 4000 generiert.  Danach wird der Messvorgang durchgeführt..  Zuletzt wird die Reaktionszeit dem entsprechenden Spieler zugeordnet. |

|  |  |
| --- | --- |
| /\* AP6. ------------------------------------------------------------------------- \*/  **if**(Re\_Ze1 > Re\_Ze2){  **digitalWrite**(P\_S2, HIGH);  **delay**(100);  **digitalWrite**(P\_S2, LOW);  Pu\_Za2++;  }  **else** **if**(Re\_Ze2 > Re\_Ze1){  **digitalWrite**(P\_S1, HIGH);  **delay**(100);  **digitalWrite**(P\_S1, LOW);  Pu\_Za1++;  }  /\* -------------------------------------------------------------------------------- \*/  /\* AP7. ------------------------------------------------------------------------- \*/  }**while**(Pu\_Za1 < 9 && Pu\_Za2 < 9);  **if**(Pu\_Za1 == 9){  **digitalWrite**(O\_M1, HIGH);  }  **else**{  **digitalWrite**(O\_M2, HIGH);  }  /\* -------------------------------------------------------------------------------- \*/  /\* AP8. ------------------------------------------------------------------------- \*/  Start\_Taster(S\_S);  **digitalWrite**(O\_M1, LOW);  **digitalWrite**(O\_M2, LOW);  Zuruecksetzung(P\_S1, P\_S2, R\_S);  Pu\_Za1 = 0; // Punktzahl1 zuruecksetzen  Pu\_Za2 = 0; // Punktzahl2 zuruecksetzen  /\* -------------------------------------------------------------------------------- \*/  }  /\* ================================================= \*/ | AP6. wertet die Reaktionszeiten aus und vergibt dem Spieler mit der kleineren Reaktionszeit einen Punkt.  AP7. führt den gesamten Ablauf durch, bis einer der Spieler neun Punkte besitzt.  AP8. führt zunächst «Start\_Taster» aus. Danach werden die Ausgänge «O\_M1» & «O\_M2» auf logisch LOW gesetzt. Anschliessend wird «Zuruecksetzung» ausgeführt und die Punkte der Spieler werden genullt. |

|  |  |
| --- | --- |
| /\* Unterprogrammdefinition --------------------------------------------- \*/  // Start-Signal einlesen  **void** Start\_Taster(**char** Start\_Pin){  **do**{  St\_Si = **digitalRead**(Start\_Pin);  **delay**(50);  }**while**(St\_Si == 0);  }  // Optische Meldung ein- & ausschalten  **void** Blinke(**void**){  **for**(b = 1; b > -1; b--){  **delay**(500);  **digitalWrite**(O\_M1, b);  **digitalWrite**(O\_M2, b);  }  }  // Anzeigen zuruecksetzen  **void** Zuruecksetztung(**char** Punktzahl\_Pin1, **char** Punktzahl\_Pin2, **cha**r Reset\_Pin){  **digitalWrite**(Reset\_Pin, LOW);  **for**(y = 1; y > -1; y--){  **delay**(100);  **digitalWrite**(Punktzahl\_Pin1, y);  **digitalWrite**(Punktzahl\_Pin2, y);  }  **digitalWrite**(Reset\_Pin, HIGH);  }  /\* -------------------------------------------------------------------------------- \*/ | «Start\_Taster» liest den Wert an «Start\_Pin» solange ein, solange der Wert null entspricht. Danach wird das Programm verlassen.  «Blinken» schaltet die Ausgänge «O\_M1» und «O\_M2» zunächst ein, wartet 500 Milli-Sekunden und schaltet diese wieder aus.  «Zuruecksetzung» setzt den Ausgang «Reset\_Pin» auf logisch LOW. Danach wird aus den Ausgängen «Punktzahl\_Pin1» & «Punktzahl\_Pin2» ein HIGH-Impuls ausgegeben. Zuletzt wird der «Reset\_Pin» auf logisch HIGH gesetzt. |

# 6. Verwendete Geräte

**Multimeter:** Agilent U1242A, BERUF Zug 03542

**Speisegerät:** DC Power Supply DF1730SB3A, BERUF Zug 03460

**Breadboard:** Besitzer Thomas Ammann

Digitrainer 6 BERUF Zug 00878

Arduino Uno Rev.3

# 7. Stückliste und Kosten

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bauteil | Typ/Wert | Menge | Kosten pro Stk. in Fr. | Artikelnummer | Lieferant |
| LED | Rot LED 5mm | 2 | 0.29 | LED 5MM 12V RT | Reichelt |
| Taster | Drucktaster | 3 | 1.30 | SDT 21SP |
| IC | 74HC163 | 2 | 0.28 | 74HCT 163 |
| 74HC4511 | 2 | 0.68 | 74HC 4511 |
| 7-Segment Anzeige | HDSP 5503 | 2 | 0.78 | SC 52-11 GE |
| Widerstand | Metallschicht Widerstände R1=R2=430Ω | 2 | 0.08 | METALL 430 |
| Metallschicht Widerstände  R3-R16=620Ω | 14 | 0.08 | METALL 620 |
| Metallschicht Widerstände  R17-R19=10kΩ | 3 | 0.08 | METALL 10,0K |
| Mikrokontroller | Arduino UNO Rev.3 | 1 | 18.05 | ARDUINO UNO |
| Rasterplatine | Streifenrasterplatine | 1 | 1.48 | H25SR160 |

\*Die Versandkosten von Reichelt betragen 7.16 Franken.

**Total: 29.01 CHF**

# 8. Schlussfolgerung

Das Projekt ist im Zeitrahmen durchgeführt worden. Die Zeitplanung konnte grösstenteils eingehalten werden und ist plangemäss abgeschlossen worden.   
Die entworfene Feinplanung musste für AP4. & AP5. erweitert werden. Dies ermöglichte eine einfachere Durchführung der beiden Arbeitspakete.   
Die 7-Segment-Anzeigen werden mit BCD-UP-Counter und BCD-to-7-Segment-Converter angesteuert.

## 8.1 Verbesserungsmöglichkeiten

**1. 7-Segment-Ansteuerung**

Die Ansteuerung der 7-Segment-Anzeigen erfolgt mit BCD-UP-Counter und BCD-to-7-Segment-Converter. Dies schränkt die Anzeige-Möglichkeiten ein. Mit, vom Arduino angesteuerten, Schieberegistern könnten die 7-Segment-Anzeigen vielseitiger genutzt werden.

# 9. Quellenverzeichnis

**Hardware:**

*LED rot 5mm:*  
http://www.produktinfo.conrad.com/datenblaetter/175000-199999/184543-da-01-en-LED\_20\_5mm\_rot\_WU\_8\_53\_HD.pdf, 15.05.2017

*Arduino UNO Rev. 3 Spezifikationen:*

https://www.arduino.cc/en/Main/arduinoBoardUno, 15.05.2017

*74HC162:*

http://pdf.datasheetcatalog.com/datasheet/philips/74HC162.pdf, 15.05.2017

*74HC4511:*

https://www.sparkfun.com/datasheets/IC/74HC4511.pdf, 15.05.2017

*7-Segment:*

http://www.alldatasheet.com/datasheet-pdf/pdf/64675/HP/HDSP-5503.html, 15.05.2017

**Software:**

*Struktogrammer:*

https://www.zg.ch/behoerden/volkswirtschaftsdirektion/gibz/grundbildung/02-technik-informatik/informatiker-in-efz

*Arduino-Software:*

https://www.arduino.cc/en/Main/Software

# 10. Identifikation

|  |
| --- |
| Zug, 30.05.2017 |
| Unterschriften: |