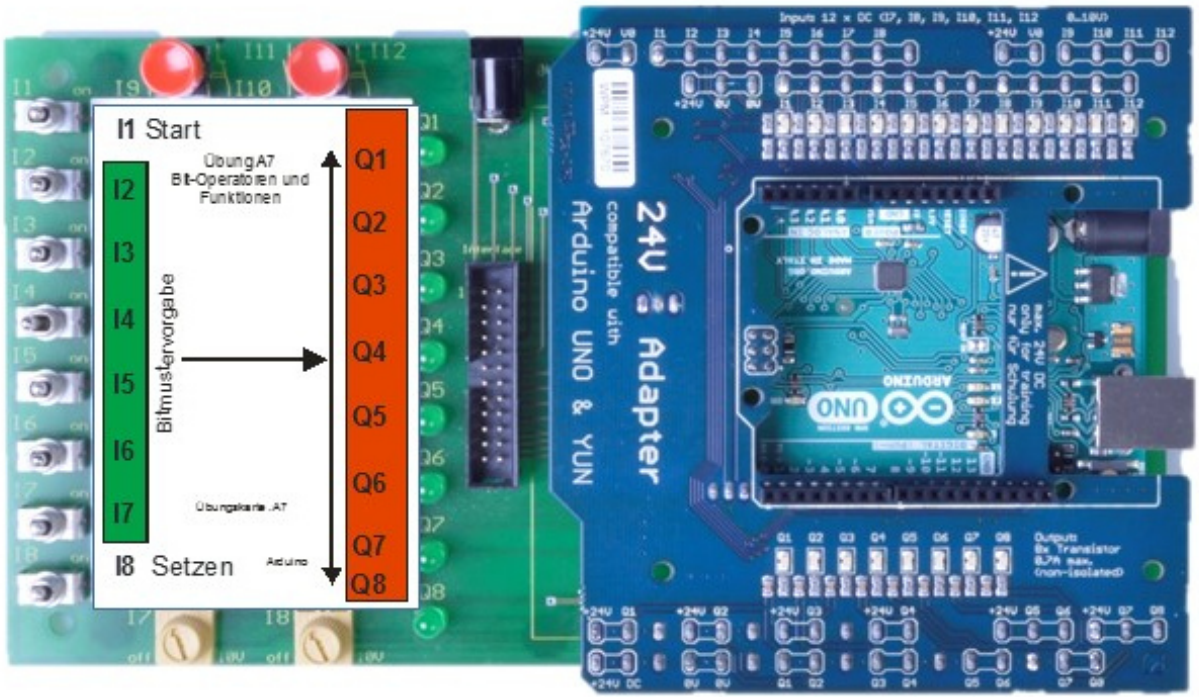
FELTRON ZEISSLER

MINITRAINERSCHULE

-BASIS-

für ***Arduino / Genuino***



Modul B

GRUNDSCHALTUNGEN DER ELEKTROTECHNIK MIT DEM ARDUINO/GENUINO-*MINITRAINER*

Reinhold Grewe / Klaus Machalek

Um die umfangreichen Möglichkeiten zu vermitteln und den unterschiedlichen Vorkenntnissen gerecht zu werden, wurde das Konzept der *MiniTrainerSchule* in mehrere Hefte aufgeteilt.

MiniTrainerSchule-Basis mit die *Module A* und *Modul B*:

Modul A : Arduino-*MiniTrainer* - und Grundlagen der Programmierung

Modul B : Grundschaltungen der Elektrotechnik

MiniTrainerSchule-Aufbau mit dem Modul C (separat zu bestellen!):

Modul C : Programmierung eigener Funktionen (Bausteinen)

Haftungsausschluss

Der Inhalt der *MiniTrainerSchule* ist ausschließlich zu Ausbildungszwecken erstellt. Wir haben den Inhalt und die Übungen sorgfältig getestet. Die Autoren übernehmen keinerlei Gewähr für die Aktualität, Richtigkeit und Vollständigkeit der bereitgestellten Informationen in dieser *MiniTrainerSchule*. Haftungsansprüche gegen die Autoren, welche sich auf Schäden materieller oder ideeller Art beziehen, die durch die Nutzung oder Nichtnutzung der dargebotenen Informationen bzw. durch die Nutzung fehlerhafter und unvollständiger Informationen verursacht wurden, sind grundsätzlich ausgeschlossen, sofern seitens der Autoren kein nachweislich vorsätzliches oder grob fahrlässiges Verschulden vorliegt. Die Autoren behalten es sich ausdrücklich vor, Teile der Seiten ohne gesonderte Ankündigung zu verändern, zu ergänzen oder zu löschen.

Autoren: Reinhold Grewe / Klaus Machalek

1. Auflage 2016 (Stand 07/16) Build 20160721

Modul B:

GRUNDSCHALTUNGEN DER ELEKTROTECHNIK

Lernziele:

Programmierung von Grundschaltungen der Elektrotechnik. Schwerpunkt ist es, logische Zusammenhänge zu verstehen und in ein Programm umzusetzen. Komplexe Funktionen stehen in diesem Modul fertig vorbereitet zur Verfügung und können in den Übungen verwendet werden, ohne die programmtechnische Umsetzung verstehen zu müssen. Das selbständige Gestalten komplexer Funktionen ist Gegenstand von Modul C.

Voraussetzungen : Kenntnisse aus Modul A

Wenn Sie dieses Modul bearbeitet haben, können Sie folgende Inhalte verstehen und programmieren:

- logische Verknüpfungen.
- elektrische Grundschaltungen
- Zeitbausteine und deren Zeitdiagramme
- Zählerbausteine und
- Datenbausteine wie FIFO und LIFO

Bezugsquellen:

1) FELTRON Elektronik-ZEISSLER & Co. GmbH,
   53842 Troisdorf, Auf dem Schellerod 22, Tel: 02241-4867-0,
   email feltron@feltron.de,
   Internet (online Shop) : www.feltron.de

2) Online Shop: www.minitrainer.de

INHALTSVERZEICHNIS MODUL B

1 Voraussetzungen aus Modul A 6
2 Die logischen Verknüpfungen und deren Umsetzung in den Programmcode 7
2.1 Spannungspegel und logische Zustände 7
2.2 Die Bit-Operatoren 7
2.3 Die UND-Verknüpfung 8
2.4 Die Oder-Verknüpfung 10
2.5 Die UND vor ODER-Verknüpfung 11
2.6 Die ODER vor UND -Verknüpfung 12
2.7 Die Exklusiv Oder-Verknüpfung 13
2.7.1 Übungen B1 … B6 - Verknüpfungen 14
2.8 Die Selbsthaltung 15
2.8.1 Übung B7 - Selbsthaltung 15
2.9 Flankererkennung 16
2.9.1 Übung B8 – Flankenerkennung 17
2.10 Kontaktprellen 17
3 Arbeiten mit Bausteinen und Bibliotheken 19
3.1 Baustein-Bibliotheken 19
3.2 Bausteine anwenden 20
4 Timer 21
4.1 Übersicht Timer 21
Tabelle B4 21
4.2 Ansprechverzögert (Einschaltverzögert) 22
4.2.1 Beispiel: Ansprechverzögert 6s 22
4.3 Rückfallverzögert (Ausschaltverzögert) 23
4.4 Impulsgeber 23
4.5 Asynchroner Taktgenerator 23
4.6 ÜBUNG B10 - TOILETTEN-LICHT-LÜFTER-STEUERUNG 24
4.7 ÜBUNG B11 - LEUCHTFEUER 24
4.8 Übung B12 - Baustellenampel 1 25
5 Schaltelemente 26
5.1 Stromstoßschalter 26
5.2 bistabile Kippstufe, FlipFlop 27

6 Zähler und Vergleicher 28
6.1 Zähler UP/DOWN 28
6.1.1 Übung Zähler 29
6.1.2 Übung B13 – Zähler 29
6.1.3 Übung B14 - Zähler und Timer 30
6.2 Vergleicher 30
6.2.1 Übung B15 - Zähler und Vergleicher 31
6.3 Übung B16 - Zähler und Vergleicher 31
7 Analogwerte 32
7.1.1 Übung B17 - Analog 33
7.1.2 Übung B18 - Schwellwertschalter 33
8 Datenbausteine 34
8.1 REGISTER 34
8.1.1 Übung Register 1 35
8.1.2 Übung B19 - Baustellenampel 2 36
8.2 LIFO 37
8.3 FIFO 39
8.3.1 Übungsaufgabe FIFO mit Fehlerbehandlung 41
8.3.2 Übung B20 - Produktionsanlage 41
9 Stichwortverzeichnis 42

1 VORAUSSETZUNGEN AUS MODUL A

Im Modul A werden die Grundlagen der Arduino-Programmierung behandelt. Um die Übersichtlichkeit und damit die Lesbarkeit der Programme zu erhöhen, wurde die Bibliotheksdatei "MyArduinoLib.h" entwickelt. In dieser Datei sind die Definitionen und Funktionen zusammengefasst, die für den Betrieb wichtig sind, aber nicht mehr verändert werden müssen.

Das sind im Detail:

- Definitionen zur Anpassung der Bezeichner für Eingänge I und Ausgänge Q. Damit kann im Programm mit I und Q statt mit den Pin-Bezeichnungen gearbeitet werden.
- Die Funktion "setPins( )" übernimmt das Setzen der Prozessorpins als Input oder Qutput.
- Die Funktion "MyWriteByte (byte, ByteToWrite)" dient zum Beschreiben eines kompletten Bytes an die Ausgänge.
- Die Funktion "MyReadByte( )" ist zum Lesen eines kompletten Bytes von den Eingängen.

Die "MyArduinoLib.h" wird auch im Modul B benutzt.

Hinweis
"MyArduinoLib.h" kann mit einem normalen Texteditor (Windows-Zubehör) geöffnet und geändert werden.

2 DIE LOGISCHEN VERKNÜPFUNGEN UND DEREN UMSETZUNG IN DEN PROGRAMMCODE

2.1 SPANNUNGSPEGEL UND LOGISCHE ZUSTÄNDE

Bei der Beschreibung von Zuständen begegnen uns verschiedene Begriffe wie HIGH, LOW, true und false, 1 und 0.

Zur Klärung:
"HIGH" und "**LOW"** beschreiben den **elektrischen** Zustand einer Schaltung. "High" ist der hohe und "LOW" der niedrige Spannungswert.

"1" und "**0"** beschreiben den **logischen** Zustand einer Schaltung. Mit dieser Bezeichnung arbeiten wir in Wahrheitstabellen. In der positiven Logik (nur diese verwenden wir hier) entspricht der elektrische Zustand "HIGH" einer logischen "1" und "LOW" der logischen "0".

"**true**" und "**false**" werden in der Programmierung benutzt um auszudrücken, ob ein Ausdruck (eine Aussage) "wahr" oder "falsch" ist. Wenn also ein logischer Zustand "1" per Programm abgefragt wird, ist die Aussage "true".

2.2 DIE BIT-OPERATOREN

Mit den Bit-Operatoren werden die Variablen nach den Regeln der **booleschen** Algebra verknüpft und das Ergebnis wird mit "=" an eine Variable übergeben. Variable können vordefinierte Eingänge "(I)" und Ausgänge "(Q)" sowie selbstdefinierte Variablen vom Datentyp "boolean" sein.

Schreibweise:
<Var_Ergebnis> = <Var 1> <Operator> <Var2> <Operator> <Var3>
Beispiel:
LichtAn = Hauptschalter & Lichtsensor & SchalterTor

&	UND	alle von n Bedingungen müssen "true" (wahr) sein
|	ODER	mindestens eine von n Bedingungen muss "true" (wahr) sein
~	NICHT	Negation: "true" wird "false " (falsch) und umgekehrt
^	Exklusives ODER	bei 2 Bedingungen müssen beide unterschiedlich sein.
=		Zuweisung an eine Variable

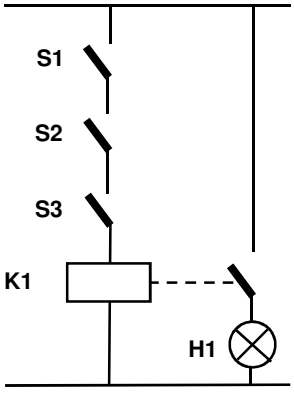
Tabelle B1

2.3 DIE UND-VERKNÜPFUNG

Die UND-Verknüpfung entspricht der Reihenschaltung in einer herkömmlichen, verdrahteten Schützsteuerung, d.h. der Ausgang ist dann eingeschaltet, wenn alle Schalter geschlossen sind.

Beispiel: Logische UND-Verknüpfung mit drei Schaltern

Realisierung in verdrahteter Schützschaltung

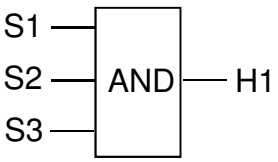


Das Ausgangssignal H1 einer UND-Verknüpfung hat dann den logischen Zustand "1" wenn alle Eingangssignale (S...) den Zustand "1" haben. Die "1" steht für einen betätigten Schließer- oder einen nicht betätigten Öffnerkontakt.

Wahrheitstabelle:

S1	S2	S3	H1
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	0
0	0	1	0
1	0	1	0
0	1	1	0
1	1	1	1

Funktionsbausteinsprache:



Schaltalgebra:

H1 = S1 & S2 & S3

Beispielprogramm B20

```
#include "MyArduinoLib.h"      //Bindet die Erweiterungen für den MiniTrainer ein

void setup()
{                     //wird  Programmstart einmal durchlaufen
  setPins();          //Modus für Eingänge und Ausgänge setzen
}
  //Variablen-Deklaration
  boolean S1, S2, S3, H1;

void loop()
{
  S1=digitalRead(I1);
  S2=digitalRead(I2);
  S3=digitalRead(I3);

  H1=S1 & S2 & S3;    //UND-Verknüpfung

  digitalWrite(Q1, H1);
}
```

Alternativ kann das Programm ohne Nutzung von Variablen auch so geschrieben werden:

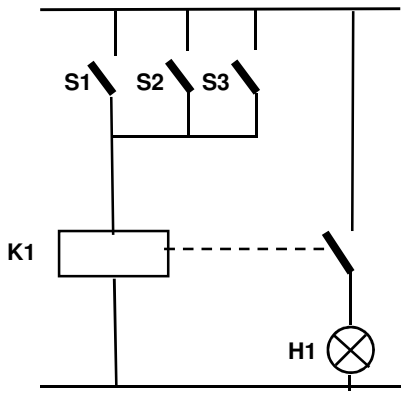
Beispielprogramm B21

```
#include "MyArduinoLib.h"   //  Bindet die Erweiterungen für den MiniTrainer ein

void setup()
{                     //wird Programmstart einmal durchlaufen
  setPins();          // Modus für Eingänge und Ausgänge setzen
}

void loop()
{
  digitalWrite(Q1, digitalRead(I1) & digitalRead(I2) &
  digitalRead(I3));
}
```

2.4 DIE ODER-VERKNÜPFUNG

Die ODER-Verknüpfung entspricht der Parallelschaltung in einer herkömmlichen, verdrahteten Schützsteuerung, d.h. der Ausgang H1 ist dann eingeschaltet, wenn mindestens einer der Schalter geschlossen ist. Im folgenden Beispiel wird zum besseren Verständnis gezeigt, wie eine ODER-Verknüpfung in verdrahteter Schütztechnik aussieht.
Beispiel: Logische ODER-Verknüpfung mit drei Schaltern

Realisierung in verdrahteter Schützschaltung

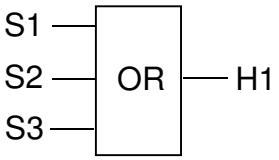


Das Ausgangssignal (H1) einer ODER-Verknüpfung hat dann den logischen Zustand „1“ wenn mindestens ein oder mehrere Eingangssignale (S...) den Zustand „1“ haben. Die „1“ steht für einen betätigten Schließerkontakt oder einen nicht betätigten Öffnerkontakt.

Wahrheitstabelle:

S1	S2	S3	H1
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

Funktionsbausteinsprache:



Schaltalgebr

$H1 = S1 \vee S2 \vee S3$

Beispielprogramm B22

```
#include "MyArduinoLib.h"   //Bindet die Erweiterungen für den MiniTrainer ein

void setup()
{                           //wird Programmstart einmal durchlaufen
  setPins();                //Modus für Eingänge und Ausgänge setzen
}
void loop()
{
digitalWrite(Q1, digitalRead(I1) | digitalRead(I2) |
digitalRead(I3));
}
```

2.5 DIE UND VOR ODER-VERKNÜPFUNG

Die drei Schalter sollen logisch in UND und ODER verknüpft werden:
Die Eingabe dieser Logik erfolgt nach der grundsätzlichen Regel der Booleschen Algebra:

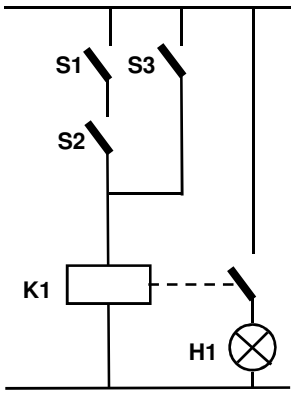
PUNKTRECHNUNG GEHT VOR STRICHRECHNUNG

In der Umsetzung in die Verdrahtungslogik heißt dies:

REIHENSCHALTUNG GEHT VOR PARALLELSCHALTUNG

Das folgende Beispiel zeigt die Umsetzung einer kombinierten UND-ODER-Schaltung nach der oben angeführten Regel:

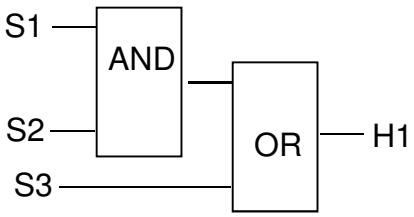
Realisierung in verdrahteter Schützschaltung



Wahrheitstabelle:

S1	S2	S3	H1
0	0	0	0
1	0	0	0
0	1	0	0
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

Funktionsbausteinsprache:



Schaltalgebra

H1 = S1 & S2 v S3

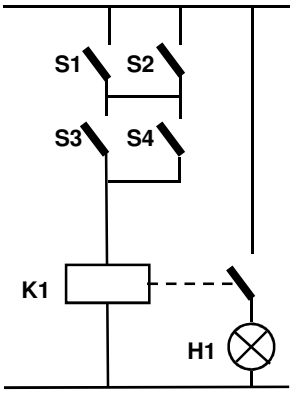
Beispielprogramm B23

```
#include "MyArduinoLib.h"//Bindet die Erweiterungen für den MiniTrainer ein

void setup()              //wird Programmstart einmal durchlaufen
{
setPins();                //Modus für Eingänge und Ausgänge setzen
}
void loop()
{
digitalWrite(Q1, (digitalRead(I1) & digitalRead(I2)) |
digitalRead(I3));
}
```

2.6 DIE ODER VOR UND -VERKNÜPFUNG

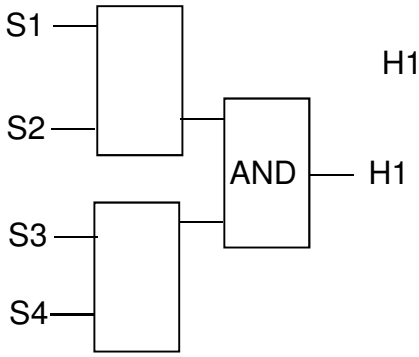
Die vier Schalter sollen logisch in ODER und UND verknüpft werden:
Die Eingabe dieser Logik nennt man auch *Konjunkive Form.* Da die Boolesche Regel auch hier gilt, muss mit Klammern gearbeitet werden. Somit legt man fest, dass die ODER -Verknüpfung zuerst verarbeitet wird.
Das folgende Beispiel zeigt die Umsetzung einer kombinierten UND – ODER -Schaltung nach der oben angeführten Regel:

Realisierung in verdrahteter Schützschaltung



Wahrheitstabelle: *Funktionsbausteinsprache:* *Schaltalgebra:*

S1	S2	S3	S4	H1
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
1	1	0	0	0
0	0	1	0	0
1	0	1	0	1
0	1	1	0	1
1	1	1	0	1
0	0	0	1	0
1	0	0	1	1
0	1	0	1	1
1	1	0	1	1
0	0	1	1	0
1	0	1	1	1
0	1	1	1	1
1	1	1	1	1



H1 = (S1 V S2) & (S3 V S4)

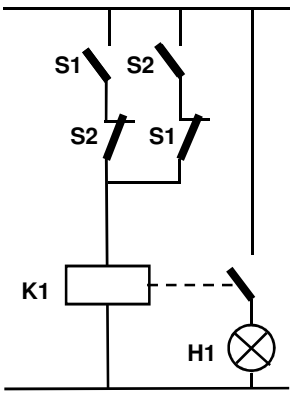
Beispielprogramm B24

```
#include "MyArduinoLib.h"   //Bindet die Erweiterungen für den MiniTrainer ein
void setup()
{                           //wird Programmstart einmal durchlaufen
  setPins();                //Modus für Eingänge und Ausgänge setzen
}
void loop() {
digitalWrite(Q1, (digitalRead(I1) | digitalRead(I3)) &
(digitalRead(I2) | digitalRead(I4)));
}
```

2.7 DIE EXKLUSIV ODER-VERKNÜPFUNG

Die meist genutzte *Exklusiv-ODER*-Schaltung ist die *XOR*-Schaltung mit zwei Eingängen. Sie unterscheidet sich zu der *ODER*-Verknüpfung dahingehend, dass der Ausgang nur dann „1“ ist, wenn einer der beiden Eingänge den Zustand „1“ hat. Sind beide Eingänge „1“, dann ist der Ausgang „0“.
Mit einer solchen Schaltung kann man eine gegenseitige Verriegelung von zwei Eingangssignalen einfach realisieren.

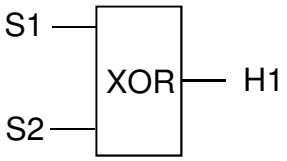
Realisierung in verdrahteter Schützschaltung



Wahrheitstabelle:

S1	S2	H1
0	0	0
1	0	1
0	1	1
1	1	0

Funktionsbausteinsprache:



Schaltalgebra:

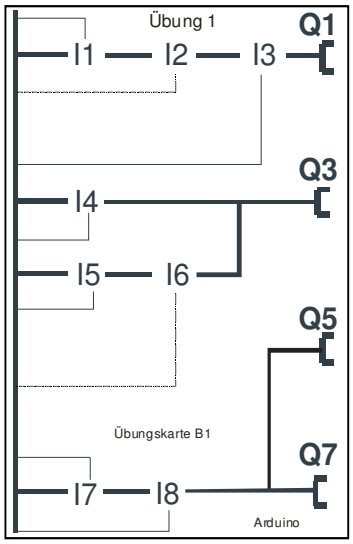
$H1 = (S1 \,\&\, \overline{S2}) \vee (S2 \,\&\, \overline{S1})$

Beispielprogramm B25

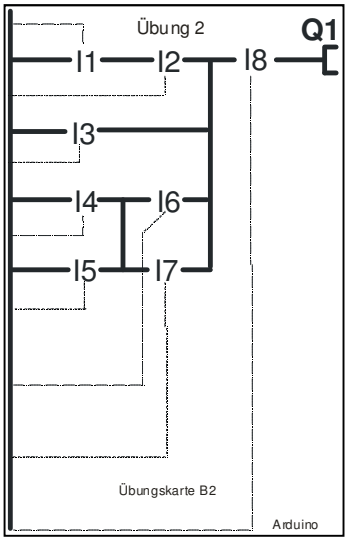
```
#include "MyArduinoLib.h"   //Bindet die Erweiterungen für den MiniTrainer ein

void setup()
{                           //wird Programmstart einmal durchlaufen
  setPins();                //Modus für Eingänge und Ausgänge setzen
}
void loop()
{
digitalWrite(Q1, digitalRead(I1) ^ digitalRead(I2));

}
```

2.7.1 ÜBUNGEN B1 … B6 - VERKNÜPFUNGEN

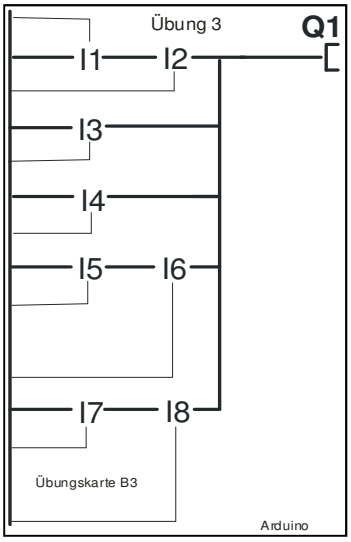
Übungskarte B1



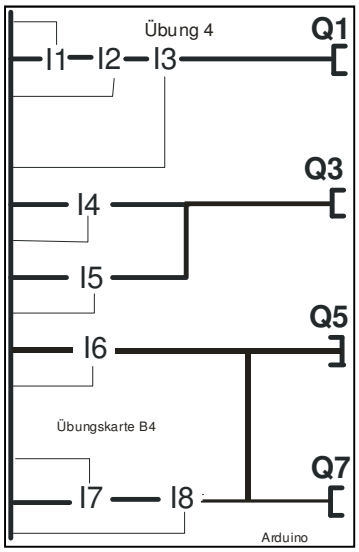
Übungskarte B2



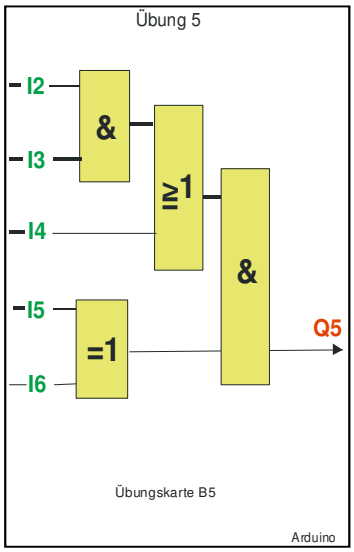
Übungskarte B3



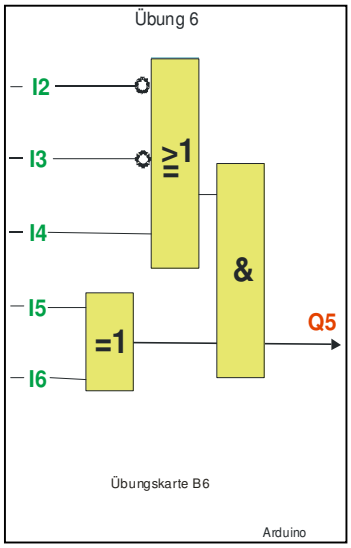
Übungskarte B4



Übungskarte B5



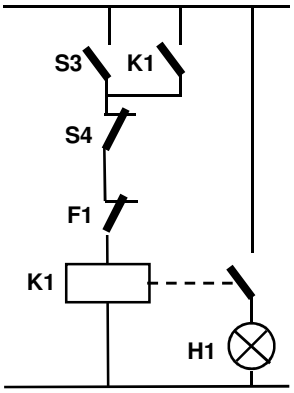
Übungskarte B6



2.8 DIE SELBSTHALTUNG

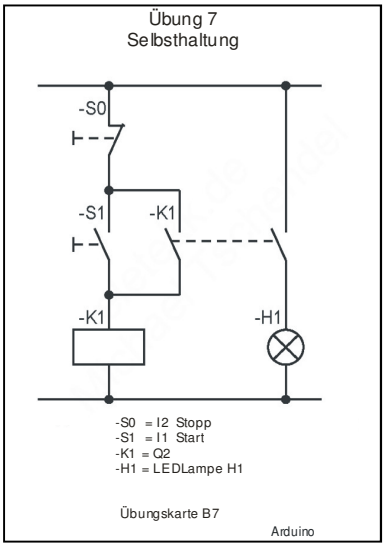
Das folgende Beispiel zeigt eine drahtbruchsichere Schaltplanerstellung einer Selbsthaltung:

Realisierung in verdrahteter Schützschaltung



Die verdrahteten Öffner S4 und F1 werden als Schliesserkontakte im Schaltplan programmiert und auf logisch „1“abfragen. Das heißt die Eingangssignale von S4 und F1 werden nicht negiert!

2.8.1 ÜBUNG B7 - SELBSTHALTUNG

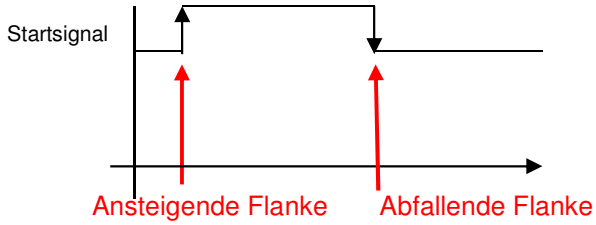


Diese Übungskarte stellt den Stromlaufplan für eine Selbsthaltung dar. Sie soll drahtbruchsicher in ein Programm umgesetzt werden.

Übungskarte B7

2.9 FLANKERERKENNUNG

Manchmal ist es wichtig, Programmteile beim Wechseln eines Zustandes, genau einmal, zu starten. Dazu müssen wir den Zustandswechsel erkennen. In dem Diagramm ist es der steile Anstieg von "LOW" nach "HIGH" oder der Abfall von "HIGH" nach "LOW". Man spricht deshalb von "ansteigender" bzw. "abfallender" Flanke.



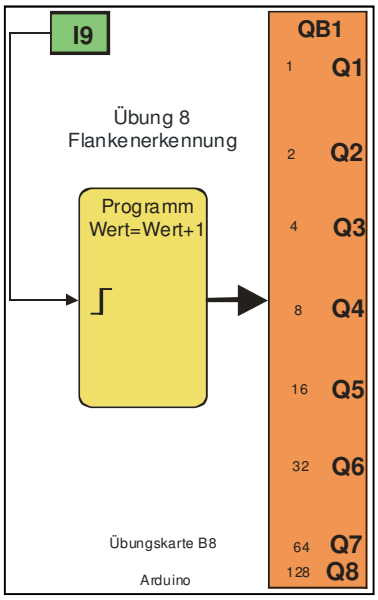
Die programmtechnische Erkennung von Flanken ist mit Hilfe einer Variablen einfach zu gestalten. Diese Variable merkt sich zyklisch den Zustand des zu überwachenden Bits. Im Beispiel der ansteigenden Flanke ist es "false" (logische "0"). Dann vergleicht das Programm den aktuellen Zustand, im Beispiel "true" (logische "1") mit dem zuletzt gespeicherten Wert "false" (logische "0"). Weil der Zustand vorher "false" war und jetzt "true" ist, hat das Programm eine ansteigende Flanke erkannt.
Die abfallende Flanke wird sinngemäß genauso erkannt. Hier ist der letzte gemerkte Zustand "true" und der aktuelle ist "false".

Beispielprogramm B26

```
#include "MyArduinoLib.h"
void setup()    //setup wird beim Programmstart einmal durchlaufen
{
setPins();      //Modus für Eingänge und Ausgänge setzen
}
  // Deklaration
  boolean I9_letzterWert;             //Wert aus letztem Zyklus
  boolean I11_letzterWert;            //Wert aus letztem Zyklus
  byte n;                             //Zähl-Variable
void loop()
{
  if(digitalRead(I9) == true && I9_letzterWert == false)
{
  n=n+1;              //I9 hat von LOW nach HIGH gewechselt -> ansteigende Flanke
}
  if(digitalRead(I11) == false && I11_letzterWert == true) {
  n=n-1;              //I11 hat von HIGH nach LOW gewechselt -> abfallende Flanke
}
  I9_letzterWert = digitalRead(I9);          // Merken des letzten  Zustands
  I11_letzterWert = digitalRead(I11);        // Merken des letzten Zustands
  MyWriteByte(n);                            // Ausgabe zur Kontrolle
}
```

2.9.1 Übung B8 – Flankenerkennung

Schreiben sie ein Programm, dass jede **steigende** Flanke von I9 zählt. Geben Sie den Zählerstand an die LEDs zu Kontrolle aus.



Übungskarte B8

2.10 Kontaktprellen

Kontakte haben die Eigenschaft, beim Schließen zu prellen. Das bedeutet, dass sie im Bereich von einigen Millisekunden sich mehrfach schließen und wieder öffnen. Wird der Kontakt mit einer schnellen Elektronik abgefragt, kann jeder Preller als eigener Impuls erkannt werden und so zu Fehlern führen. Je nach Kontaktart und Material kann das Prellen unterschiedlich stark ausgeprägt sein. Mit Hilfe eines kleinen Programms kann die Wirkung des Prellens ausgeblendet werden.
Versuch mit einen Programm ohne Entprellung und ohne Flankenerkennung:

Beispielprogramm B27

```
#include <MT_LIB_REGISTER.h>
#include <MT_LIB_TIMER.h>
#include "MyArduinoLib.h"
  // Deklaration
  int i;              //Zählvariable
  boolean letzterTakt;
  byte Counter;

void setup()          //wird beim Programmstart einmal durchlaufen
{
  setPins();          //Modus für Eingänge und Ausgänge setzen
  Counter =0;
  MyWriteByte(Counter);
}
void loop()
```