Microcontroller Teil 4 Programmiertechnik

DI (FH) Andreas Pötscher

HTL Litec

Verknüpfungssteuerungen

Verknüpfungssteuerungen sind die einfachsten Steuerungen. Dabei wird nur aufgrund von Eingangszuständen ein Ausgangszustand eingestellt. Die Reihenfolge oder die zeitliche Abfolge von Eingängen hat keinen Einfluss auf die Ausgänge.

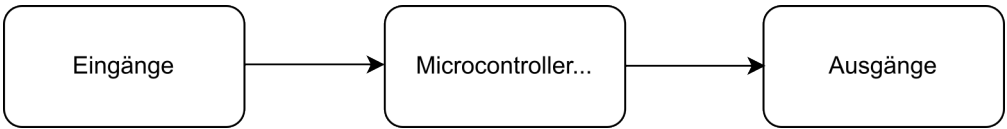

Figure 1: Verknüpfungssteuerungen

Verknüpfungssteuerungen Programmaufbau

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.

Verknüpfungssteuerungen Programmaufbau

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 Eingänge lesen. Am Anfang des Main Loop werden alle Eingänge gelesen und in Variablen gespeichert

Verknüpfungssteuerungen Programmaufbau

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 Eingänge lesen. Am Anfang des Main Loop werden alle Eingänge gelesen und in Variablen gespeichert
- 3 Ausgänge setzen. Aufgrund der Eingänge werden alle Ausgänge gesetzt. Dabei müssen in jeder Verzweigung alle Ausgänge gesetzt werden.

Verknüpfungssteuerungen Programmaufbau

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 Eingänge lesen. Am Anfang des Main Loop werden alle Eingänge gelesen und in Variablen gespeichert
- 3 Ausgänge setzen. Aufgrund der Eingänge werden alle Ausgänge gesetzt. Dabei müssen in jeder Verzweigung alle Ausgänge gesetzt werden.
- GOTO 2

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 Eingänge lesen. Am Anfang des Main Loop werden alle Eingänge gelesen und in Variablen gespeichert
- 3 Ausgänge setzen. Aufgrund der Eingänge werden alle Ausgänge gesetzt. Dabei müssen in jeder Verzweigung alle Ausgänge gesetzt werden.
- GOTO 2
- **Beispiel**

Zur Vereinfachung und Vermeidung von Redundanzen kann das Ansteuern der Ausgänge mit `#define` Präprozessormakros programmiert werden. Der Präprozessor kopiert den Code vor dem eigentlichen Kompilieren an die richtigen Stellen.

```
#define PINLED1 1
PORTA |= 0x01 << PINLED1;
```

Beim Übersetzten des Programms wird dabei als erstes der Präprozessor ausgeführt. Der Code sieht dann so aus.

```
PORTA |= 0x01 << 1;
```

Beispiel

Wahrheitstabellen

Die Logik für eine Verknüpfungssteuerung lässt sich sehr gut mit einer Wahrheitstabelle darstellen. Dabei wird für jeden Ein- und Ausgang eine Spalte und für jede Eingangskombination eine Zeile angelegt. Bei drei Sensoren und zwei Leuchten sieht das dann folgendermaßen aus:

S1	S2	S3	L1	L2
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Wahrheitstabellen

Beispiel:

- Leuchte1 soll leuchten, wenn einer oder zwei von den drei Sensoren logisch 1 melden. Ansonsten nicht:
- Leuchte2 soll leuchten, wenn alle drei Sensoren logisch 1 melden.

S1	S2	S3	L1	L2
0	0	0	**0**	**0**
0	0	1	**1**	**0**
0	1	0	**1**	**0**
0	1	1	**1**	**0**
1	0	0	**1**	**0**
1	0	1	**1**	**0**
1	1	0	**1**	**0**
1	1	1	**0**	**1**

Ablaufsteuerungen

Bei Ablaufsteuerungen werden nicht nur Eingangszustände, sondern auch interne Zustände berücksichtigt. Damit sind Abfolgesteuerungen möglich.

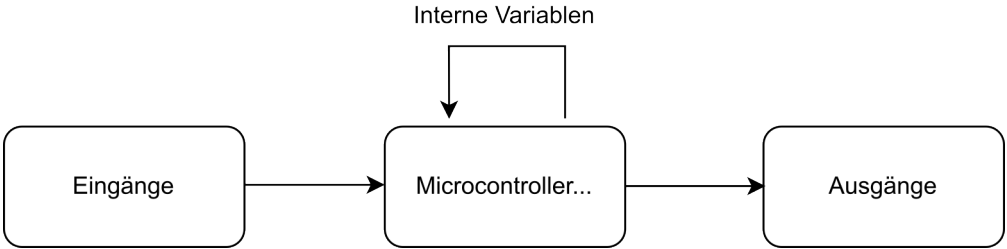

Figure 2: Ablaufsteuerungen

State Machine

Ein solches Verhalten wird in der Informatik als Statemachine oder Zustandsautomat dargestellt. Im einfachsten Fall gibt es dabei 2 States oder Zustände. Es können aber natürlich auch mehr sein.

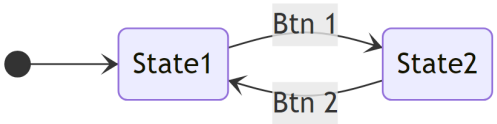

Figure 3: Statemachine

Der State wird im Programm in einer eigenen Variablen gespeichert.

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 **Initialsieren der Variable in der der Zustand (State) gespeichert wird.**

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 **Initialsieren der Variable in der der Zustand (State) gespeichert wird.**
- 3 Eingänge lesen. Am Anfang des Main Loop werden alle Eingänge erfasst und in Variablen gespeichert.

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 **Initialsieren der Variable in der der Zustand (State) gespeichert wird.**
- 3 Eingänge lesen. Am Anfang des Main Loop werden alle Eingänge erfasst und in Variablen gespeichert.
- 4 **Setzen der Zustandsvariablen.**

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 **Initialsieren der Variable in der der Zustand (State) gespeichert wird.**
- 3 Eingänge lesen. Am Anfang des Main Loop werden alle Eingänge erfasst und in Variablen gespeichert.
- 4 **Setzen der Zustandsvariablen.**
- 5 Ausgänge setzen. Aufgrund der Eingänge **und der Zustandsvariablen** werden alle Ausgänge gesetzt. Dabei müssen in jeder Verzweigung alle Ausgänge gesetzt werden.

Das Programm kann dabei immer nach folgendem Schema einfach aufgebaut werden.

- 1 Initialisierung. Hier werden die Ein- und Ausgänge definiert.
- 2 **Initialsieren der Variable in der der Zustand (State) gespeichert wird.**
- 3 Eingänge lesen. Am Anfang des Main Loop werden alle Eingänge erfasst und in Variablen gespeichert.
- 4 **Setzen der Zustandsvariablen.**
- 5 Ausgänge setzen. Aufgrund der Eingänge **und der Zustandsvariablen** werden alle Ausgänge gesetzt. Dabei müssen in jeder Verzweigung alle Ausgänge gesetzt werden.
- GOTO 3

Ablaufsteuerung Beispiel

Damit kann zum **Beispiel** eine Steuerung mit einem Ein- und einem Austaster realisiert werden. Ein Led lässt sich dann nur einschalten, wenn sich die Steuerung im Ein-Modus befindet.

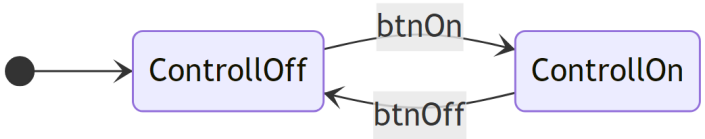

Figure 4: Statemachine des Beispiels

Dabei kommt das Ergebnis nicht nur auf die Zustände der Eingänge an, sondern auch auf die Reihenfolge in der diese Betätigt werden. Werden z. B. folgende Eingänge in der gegeben Reihefolge gesetzt:

```
1. PINB2 (btnLed) = true
```

Ist die LED an PA1 **nicht** eingeschaltet.

Werden aber die Eingänge in folgender Reihenfolge betätigt.

1. `PINB0 (btnOn) = true`
2. `PINB0 (btnOn) = false`
3. `PINB2 (btnLed) = true`

ist die LED an PA1 eingeschaltet. Dadurch, dass der Eingang PINB0 auf true gesetzt wird, wird interne Variable `controlOn` auf true gesetzt. Das Ergebnis wenn PINB2 auf true gesetzt wird verändert sich dadurch.

Ablaufsteuerung

Der zeitliche Ablauf kann in einem Signal-Zeit-Diagramm dargestellt werden.

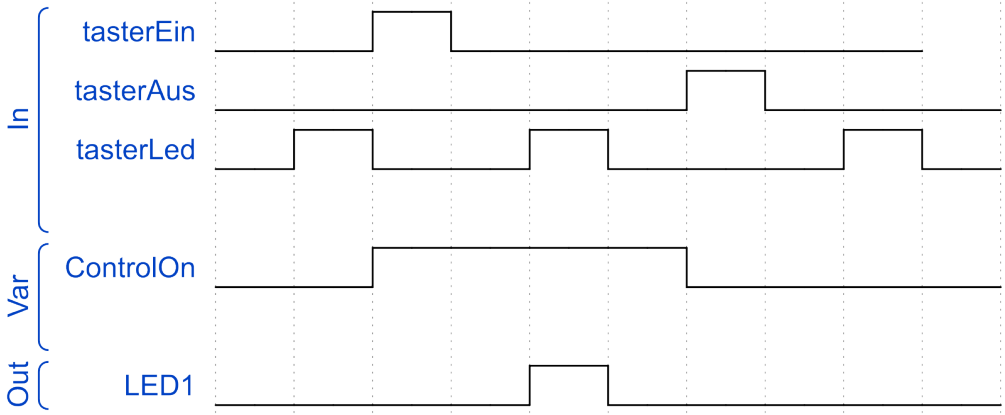

Figure 5: Signal Zeitverlauf

Signalflanken

Wann Signalflanken beachtet werden müssen, lässt sich am besten anhand eines Beispiels erklären.

Beispiel

Signalflanken

Im vorherigen Beispiel schaltet sich die LED sehr schnell Ein und Aus. Je nach dem wann der Taster losgelassen wird leuchtet die LED dann oder nicht. Dieses Verhalten lässt sich mit einer Statemachine gut zeigen.

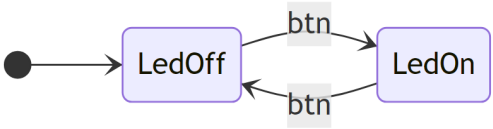

Figure 6: Statemachine Flanken

Dadurch, dass das gleiche Signal für beide Zustandswechsel verwendet wird, wechselt beim jedem Schleifendurchlauf der Zustand.

Für solche Steuerungsaufgaben ist es notwendig, die Änderung des Eingangssignals zu detektieren. Z.B. Wenn wir im vorigen Beispiel die Led genau einmal umschalten wollen. Sehen wir uns dazu den zeitlichen Signalverlauf des Tasters an.

Figure 7: Signal Zeitverlauf Taster

Dabei wird der bei einmaligem Drücken des Tasters fünfmal ein High-Pegel eingelesen. Die Led wird also 5 mal umgeschaltet.

Steigende und fallende Flanke

Damit nur einmal gezählt wird können wir die Flanken des Signals verwenden. Es gibt eine steigende und eine fallende Flanke.

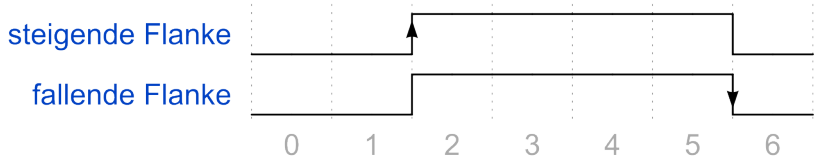

Figure 8: Steigende und fallende Flanke

Steigende und fallende Flanke

Die steigende Flanke tritt auf wenn sich das Signal von Low auf High ändert. Also wenn der Taster gedrückt wird. Die fallende Flanke tritt auf wenn sich das Signal von High auf Low ändert. Also wenn der Taster losgelassen wird.

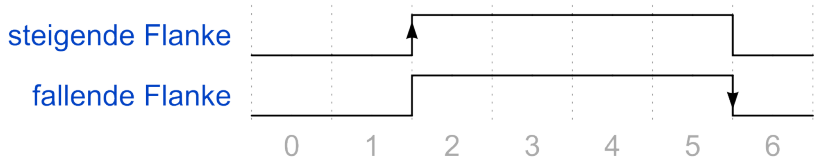

Figure 9: Steigende und fallende Flanke

Flanken detektieren

Dazu wird vor der `while(1)` Schleife eine Variable deklariert. Am Ende der Schleife wird der Wert des Eingangs in die Variable gespeichert.

```
uint8_t lastBtn = 0;
while(1)
{
    //Hier können die Flanken detektiert werden
    lastBtn = btn;
}
```

Flanken detektieren

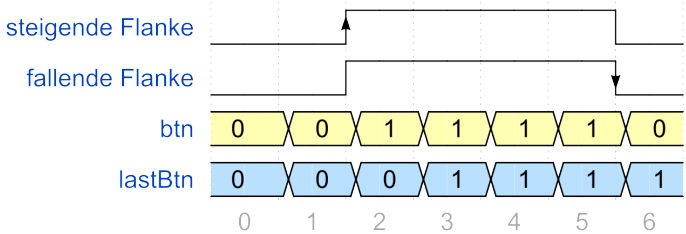

Figure 10: Werte der Variablen btn und lastBtn

```
uint8_t lastBtn = 0;
while(1)
{
    lastBtn = btn;
}
```

Steigende Flanke detektieren

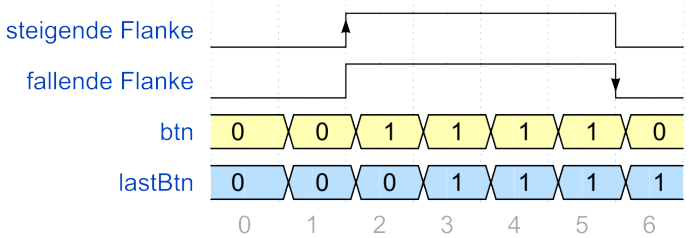

Figure 11: Werte der Variablen btn und lastBtn

Bei einer steigenden Flanke (*engl. rising Edge*) ist der aktuelle Wert `btn` dann 1 und der vorherige Wert `lastBtn` dann 0.

Steigende Flanke detektieren

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

```
uint8_t lastBtn = 0;
while(1)
{
    uint8_t btn = PINB >> PINBTN & 0x01;
    if(btn == 1 && lastBtn == 0)
    {
        //Steigende Flanke
    }
    lastBtn = btn;
}
```

Fallende Flanke detektieren

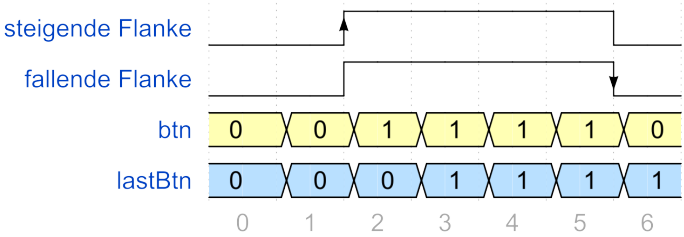

Figure 12: Werte der Variablen btn und lastBtn

Bei einer fallenden Flanke (*engl. falling Edge*) ist der aktuelle Wert `btn` dann 0 und der vorherige Wert `lastBtn` dann 1.

Fallende Flanke detektieren

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

```
uint8_t lastBtn = 0;
while(1)
{
    uint8_t btn = PINB >> PINBTN & 0x01;
    if(btn == 0 && lastBtn == 1)
    {
        //Fallende Flanke
    }
    lastBtn = btn;
}
```

Beliebige Flanke detektieren

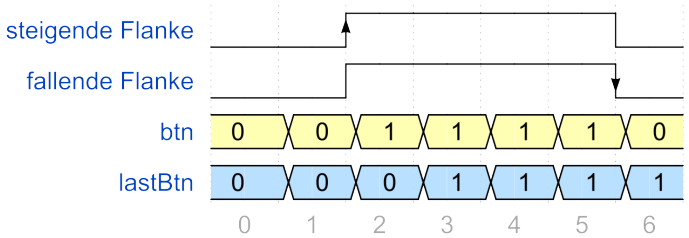

Figure 13: Werte der Variablen btn und lastBtn

Als dritte Möglichkeit kann auch jede beliebige Flanke (*engl. any Edge*) einfach detektiert werden. Dazu müssen die Werte von `btn` und `lastBtn` ungleich sein.

Beliebige Flanke detektieren

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

```
uint8_t lastBtn = 0;
while(1)
{
    uint8_t btn = PINB >> PINBTN & 0x01;
    if(btn != lastBtn)
    {
        //Beliebige Flanke
    }
    lastBtn = btn;
}
```

Beispiel Led umschalten

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

Beispiel: Led-Umschaltung mit Flankenerkennung.

Beispiel Lösung