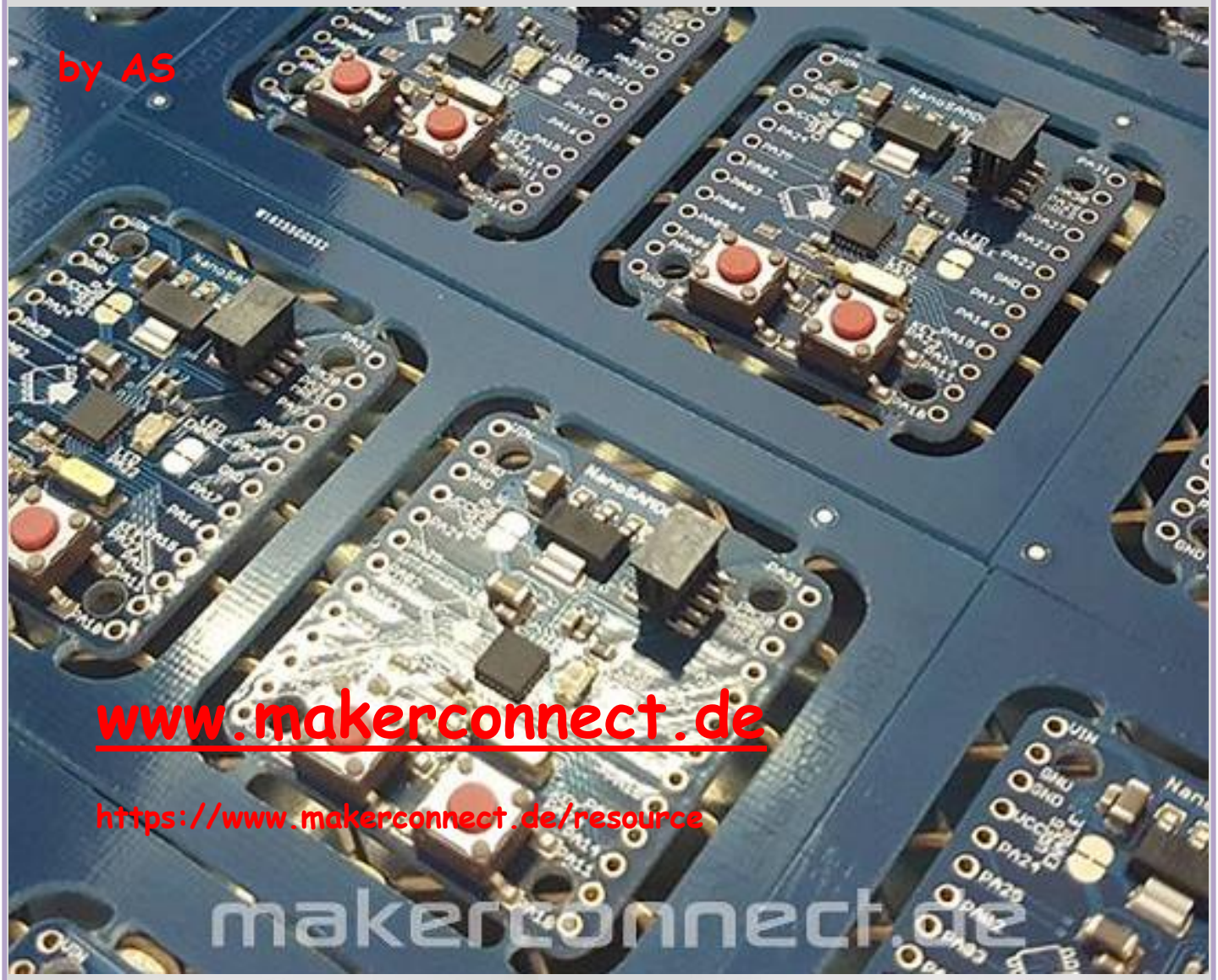
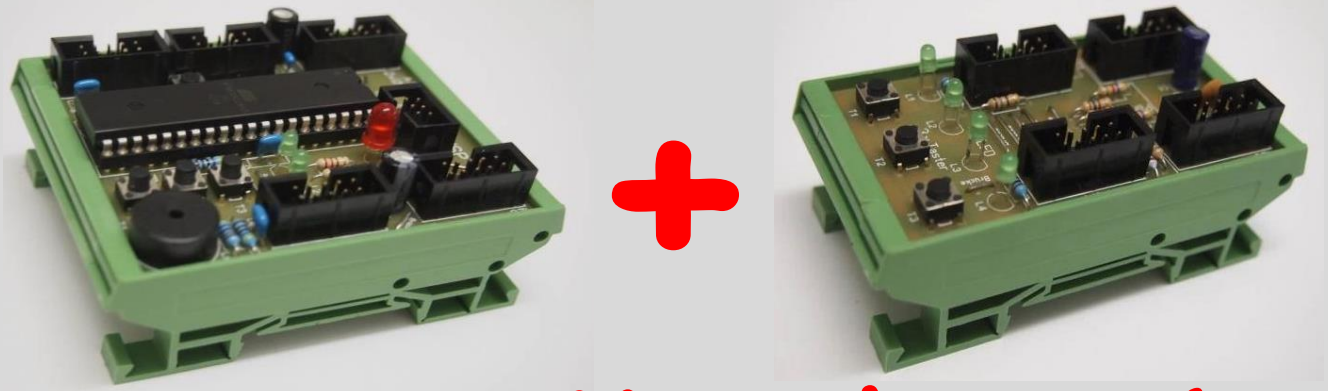
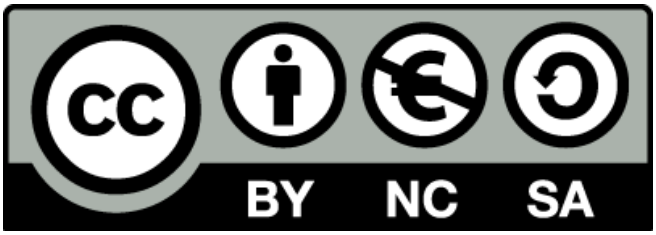
Mikrokontroller & I²C Bus





State Maschine 1

Copyright



Sicherheitshinweise

Lesen Sie diese Gebrauchsanleitung, bevor Sie diesen Bausatz in Betrieb nehmen und bewahren Sie diese an einem für alle Benutzer jederzeit zugänglichen Platz auf. Bei Schäden, die durch Nichtbeachtung dieser Bedienungsanleitung verursacht werden, erlischt die Gewährleistung/Garantie. Für Folgeschäden übernehmen wir keine Haftung! Bei allen Geräten, die zu ihrem Betrieb eine elektrische Spannung benötigen, müssen die gültigen VDE-Vorschriften beachtet werden. Besonders relevant sind für diesen Bausatz die VDE-Richtlinien VDE 0100, VDE 0550/0551, VDE 0700, VDE 0711 und VDE 0860. Bitte beachten Sie auch nachfolgende Sicherheitshinweise:

- Nehmen Sie diesen Bausatz nur dann in Betrieb, wenn er zuvor berührungssicher in ein Gehäuse eingebaut wurde. Erst danach darf dieser an eine Spannungsversorgung angeschlossen werden.
- Lassen Sie Geräte, die mit einer Versorgungsspannung größer als 24 V- betrieben werden, nur durch eine fachkundige Person anschließen.
- In Schulen, Ausbildungseinrichtungen, Hobby- und Selbsthilfewerkstätten ist das Betreiben dieser Baugruppe durch geschultes Personal verantwortlich zu überwachen.
- In einer Umgebung in der brennbare Gase, Dämpfe oder Stäube vorhanden sind oder vorhanden sein können, darf diese Baugruppe nicht betrieben werden.
- Im Falle eine Reparatur dieser Baugruppe, dürfen nur Original-Ersatzteile verwendet werden! Die Verwendung abweichender Ersatzteile kann zu ernsthaften Sach- und Personenschäden führen. Eine Reparatur des Gerätes darf nur von fachkundigen Personen durchgeführt werden.
- Spannungsführende Teile an dieser Baugruppe dürfen nur dann berührt werden (gilt auch für Werkzeuge, Messinstrumente o.ä.), wenn sichergestellt ist, dass die Baugruppe von der Versorgungsspannung getrennt wurde und elektrische Ladungen, die in den in der Baugruppe befindlichen Bauteilen gespeichert sind, vorher entladen wurden.
- Sind Messungen bei geöffnetem Gehäuse unumgänglich, muss ein Trenntrafo zur Spannungsversorgung verwendet werden
- Spannungsführende Kabel oder Leitungen, mit denen die Baugruppe verbunden ist, müssen immer auf Isolationsfehler oder Bruchstellen kontrolliert werden. Bei einem Fehlers muss das Gerät unverzüglich ausser Betrieb genommen werden, bis die defekte Leitung ausgewechselt worden ist.
- Es ist auf die genaue Einhaltung der genannten Kenndaten der Baugruppe und der in der Baugruppe verwendeten Bauteile zu achten. Gehen diese aus der beiliegenden Beschreibung nicht hervor, so ist eine fachkundige Person hinzuzuziehen

Bestimmungsgemäße Verwendung

- Auf keinen Fall darf 230 V~ Netzspannung angeschlossen werden. Es besteht dann Lebensgefahr!
- Dieser Bausatz ist nur zum Einsatz unter Lern- und Laborbedingungen konzipiert worden. Er ist nicht geeignet, reale Steuerungsaufgaben jeglicher Art zu übernehmen. Ein anderer Einsatz als angegeben ist nicht zulässig!
- Der Bausatz ist nur für den Gebrauch in trockenen und sauberen Räumen bestimmt.
- Wird dieser Bausatz nicht bestimmungsgemäß eingesetzt kann er beschädigt werden, was mit Gefahren, wie z.B. Kurzschluss, Brand, elektrischer Schlag etc. verbunden ist. Der Bausatz darf nicht geändert bzw. umgebaut werden!
- Für alle Personen- und Sachschäden, die aus nicht bestimmungsgemäßer Verwendung entstehen, ist nicht der Hersteller, sondern der Betreiber verantwortlich. Bitte beachten Sie, dass Bedien- und /oder Anschlussfehler außerhalb unseres Einflussbereiches liegen. Verständlicherweise können wir für Schäden, die daraus entstehen, keinerlei Haftung übernehmen.
- Der Autor dieses Tutorials übernimmt keine Haftung für Schäden. Die Nutzung der Hard- und Software erfolgt auf eigenes Risiko.

State Maschine 1 (Der Anfang …)

Sicher wurden über dieses Thema schon viele Bücher und Beiträge geschrieben. Eigentlich möchte ich keinen neuen Beitrag dieser Sammlung zufügen.
Leider stört mich an den vielen Beiträgen etwas. Sie sind alle recht kompliziert und lang geschrieben. Es gibt kaum Beiträge, die kurz und bündig sind und man auch etwas versteht.
Ich versuche es trotzdem zu machen. Leider muss ich mit etwas Theorie anfangen.

Kooperatives Multitasking

Kooperatives Multitasking beruht darauf, dass jeder Task sich selbst unterbricht, wenn er dazu bereit ist. Man sagt, dass die Tasks kooperieren, weil sie sich gegenseitig Rechenzeit abgeben. Der Haken ist: Wenn ein Task aus irgendeinem Grund überhaupt keine Rechenzeit abgibt, dann bleiben alle anderen Tasks hängen. Da kooperatives Multitasking ohne besondere Hardware auskommt, findet man in Programmen für Mikrocontroller diverse Implementierungen dieser Variante.
Eine klassische Implementierung ist der **Zustandsautomat (Statemaschine).**

Präemptives Multitasking

Die großen Betriebssysteme (Windows, Linux, etc) benutzen präemptives Multitasking. Dabei unterbricht ein Timer in regelmäßigen Intervallen den gerade laufenden Task, um andere Tasks auszuführen. Die Tasks werden also nicht wirklich gleichzeitig ausgeführt, sondern immer abwechselnd ein Stück von jedem Task. (Mehrkernprozessor nicht berücksichtigt)
Damit das Betriebssystem seine Tasks aktiv unterbrechen kann, braucht es einen Timer und einen Prozessor, der dazu spezielle Funktionen hat. Mikrocontroller sind für präventives Multitasking in der Regel nicht geeignet.

Definition Statemaschine (Zustandsautomat)

Endlicher Zustandsautomat (engl. finite state machine, kurz FSM)
Dabei handelt es sich um ein Steuerkonzept, welches eine abstrakte Maschine zum Vorbild nimmt, die über einen internen Zustand verfügt. Dieser Zustand beinhaltet alle Informationen zum Betrieb der Maschine. Die Maschine arbeitet, indem sie von einem Zustand in einen anderen Zustand übergeht und bei derartigen Zustandsübergängen Aktionen ausführt. Dabei ergibt sich der Folgezustand aus dem momentanen Zustand und einem Ereignis, z. B. einem Tastendruck. Eine FSM kann in Software oder auch Hardware aufgebaut werden.
Die FSM selbst wird über eine Taktung angetrieben, kann also nicht in beliebigen kurzen Zeitspannen auf Ereignisse reagieren. In jedem Takt wird anhand des vorliegenden Zustands und dem Status der Eingabekanäle entschieden, welcher Zustand als nächstes vorliegen soll und welche Aktionen auszuführen sind.
Die abstrakte Beschreibung einer FSM ist auf mehrere Arten möglich. Zum einen kann sie in Form einer Tabelle beschrieben werden, aber auch eine graphische Darstellung der Zustände und deren Abhängigkeiten in Form eines Zustandsdiagramms sind möglich.

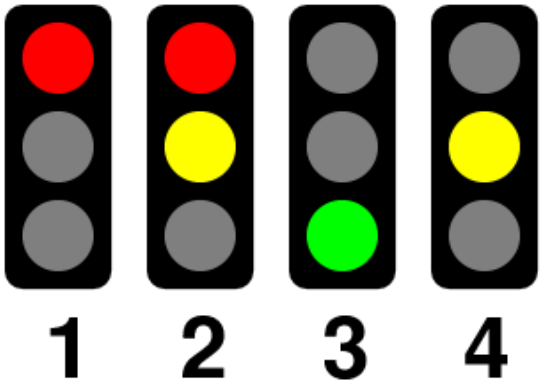
Eigentlich war es das schon mit der Theorie. Ein paar Erklärungen werden zu den einzelnen Programmen noch kommen. Fangen wir mit was leichten an.

Die Ampel – ein Klassiker

Ampelsteuerung

Als erstes wollen wir an einem Beispiel einer Ampelsteuerung die Funktion demonstrieren.

Auf dem Bild ist der Ablauf einer einfachen Ampel dargestellt.
Die Funktion ist in 4 Schritten untergliedert:



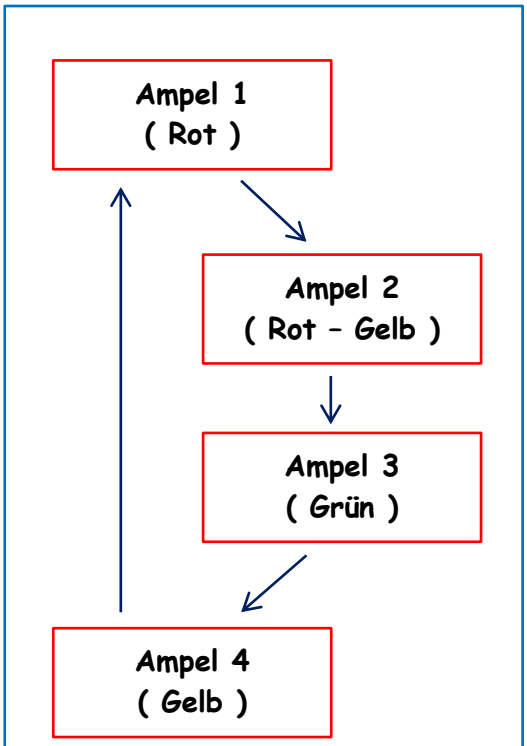
Schritt 1 – Rot
Schritt 2 – Rot – Gelb
Schritt 3 – Grün
Schritt 4 – Gelb

(Rücksprung zu Schritt 1)

Die Funktion habe ich einmal als Tabelle und in einer graphisch Darstellung beschrieben. Dabei habe ich zu Anfang bewusst einen einfachen Ablauf gewählt.

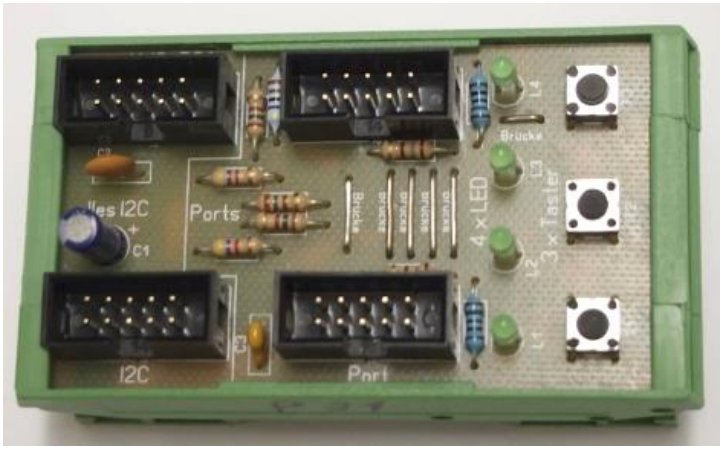
Zustandstabelle Ampel		
Zustand	Ampel	nächster Zustand
1	Rot	2
2	Rot - Gelb	3
3	Grün	4
4	Gelb	1

Bei dieser Funktion wiederholt sich der Ablauf ständig. Man kann sehr schön dabei den Ablauf erkennen.



„Einfache" Ampelsteuerung

Zur Demonstration nutze ich dieses Modul. Dabei ordne ich die LED folgenden Funktionen zu:



LED 3 – Rotes Licht

LED 2 – Gelbes Licht

LED 1 – Grünes Licht

Sehen wir uns die Funktion einer Ampel in einem Programm an:

```
/* ATB_Stm_1.c Created: 06.10.2014 18:25:28 Author: AS */

#define F_CPU 16000000UL        // Angabe der Quarzfrequenz, wichtig für die Zeit
#include <util/delay.h>         // Einbindung Datei Pause
#include <avr/io.h>             // Einbindung Datei Ausgänge
int main(void)                  // AT 1284p
 {
   DDRA=0b01110000;             // Port A auf Ausgang schalten
   PORTA |= (1<<PA6);           // Schaltet Pin 6 aus
   PORTA |= (1<<PA5);           // Schaltet Pin 5 aus
   PORTA |= (1<<PA4);           // Schaltet Pin 4 aus
   // PA 6 = Rot                Zuordnung der LED
   // PA 5 = Gelb
   // PA 4 = Grün
   while(1)                     // Programmschleife
    {
      // 1. Rot ein
      PORTA &= ~(1<<PA6);       // Schaltet Pin 6 ein
      _delay_ms(3000);          // Pause 3000 ms

      // 2. Gelb ein
      PORTA &= ~(1<<PA5);       // Schaltet Pin 5 ein
      _delay_ms(1000);          // Pause 1000 ms

      // 3. Rot aus, Gelb aus, Grün ein
      PORTA |= (1<<PA6);        // Schaltet Pin 6 aus
      PORTA |= (1<<PA5);        // Schaltet Pin 5 aus
      PORTA &= ~(1<<PA4);       // Schaltet Pin 4 ein
      _delay_ms(2000);          // Pause 2000 ms

      // 4. Grün aus, Gelb ein
      PORTA |= (1<<PA4);        // Schaltet Pin 4 aus
      PORTA &= ~(1<<PA5);       // Schaltet Pin 5 ein
      _delay_ms(1500);          // Pause 1500 ms

      // 5. Gelb aus
      PORTA |= (1<<PA5);        // Schaltet Pin 5 aus
      _delay_ms(50);            // Pause 50 ms
    }
 }
```

Man kann den Ablauf der Ampelsteuerung in 5 Schritte untergliedern:

1. **Schritt – Rot ein**
2. **Schritt – Gelb ein**
3. **Schritt – Rot aus, Gelb aus, Grün ein**
4. **Schritt – Grün aus, Gelb ein**
5. **Schritt – Gelb aus**

Damit haben wir einen kompletten Durchlauf. Als nächstes folgt wieder der 1. Schritt.

Eine kurze Erklärung zum Programm:

```
#define F_CPU 16000000UL        // Angabe der Quarzfrequenz, wichtig für die Zeit
#include <util/delay.h>         // Einbindung Datei Pause
#include <avr/io.h>             // Einbindung Datei Ausgänge
```

Es werden die notwendigen Dateien geladen und die Frequenz angegeben

```
int main(void)                  // AT 1284p
 {
```

Start des Programmes

```
   DDRA=0b01110000;             // Port A auf Ausgang schalten
   PORTA |= (1<<PA6);           // Schaltet Pin 6 aus
   PORTA |= (1<<PA5);           // Schaltet Pin 5 aus
   PORTA |= (1<<PA4);           // Schaltet Pin 4 aus
   // PA 6 = Rot                Zuordnung der LED
   // PA 5 = Gelb
   // PA 4 = Grün
```

Angabe PortA, Freigabe der Pins, PA4, PA5, PA6 ausschalten, Farben der LED zuordnen

```
   while(1)                     // Programmschleife
    {
```

Beginn Programmschleife

```
      // 1. Rot ein
      PORTA &= ~(1<<PA6);       // Schaltet Pin 6 ein
      _delay_ms(3000);          // Pause 3000 ms

      // 2. Gelb ein
      PORTA &= ~(1<<PA5);       // Schaltet Pin 5 ein
      _delay_ms(1000);          // Pause 1000 ms

      // 3. Rot aus, Gelb aus, Grün ein
      PORTA |= (1<<PA6);        // Schaltet Pin 6 aus
      PORTA |= (1<<PA5);        // Schaltet Pin 5 aus
      PORTA &= ~(1<<PA4);       // Schaltet Pin 4 ein
      _delay_ms(2000);          // Pause 2000 ms

      // 4. Grün aus, Gelb ein
      PORTA |= (1<<PA4);        // Schaltet Pin 4 aus
      PORTA &= ~(1<<PA5);       // Schaltet Pin 5 ein
      _delay_ms(1500);          // Pause 1500 ms

      // 5. Gelb aus
      PORTA |= (1<<PA5);        // Schaltet Pin 5 aus
      _delay_ms(50);            // Pause 50 ms
    }
 }
```

Ein- und Ausschalten der Pins

Dieses Beispielprogramm ist natürlich keine State Maschine und nicht Multitasking. Es ist eigentlich ganz einfaches Programm, das nur die Funktion demonstriert. Als nächstes wollen wir es so verändern, das es unseren Forderungen entspricht.

Notwendige Änderungen

Um dieses Programm Multitasking fähig zu machen, muss die while Schleife in regelmäßigen Zeitabständen unterbrochen werden, um andere Tasks auszuführen. Eine mögliche Umsetzungsmethode für kooperatives Multitasking ist der Zustandsautomat.
Der Zustandsautomat besteht aus einer (meist) endlosen Programmschleife, in der die einzelnen Tasks immer wieder nacheinander aufgerufen werden.

```
int main()
  {
    while(1)                          // Endlos-Schleife
     {
        mache ein bisschen von Task 1;
        mache ein bisschen von Task 2;
      }
    }
```

Damit auch wirklich jeder Task oft genug an die Reihe kommt, darf keiner der Tasks stehen bleiben. Jeder Task darf pro Schleifendurchlauf nur eine kleine kurze Aktion durchführen. Aufgaben, die länger dauern (z.B. das Warten auf eine Eingabe mit einem Taster) müssen über mehrere Schleifendurchläufe verteilt werden.

Regeln State Maschine

Die Zustandsnummer ist in diesem Fall einfach die Taktung der FSM. Verfolgt man die Zustände von einem Zustand zum nächsten, dann kann man sich sehr leicht davon überzeugen, dass die Lichtfolge der Ampel tatsächlich der gewünschten Abfolge entspricht.
Hat man die Funktionalität einer FSM erst mal soweit in Tabellenform festgelegt, dann ist es sehr einfach daraus ein Programm in einer Programmiersprache wie z.B. C abzuleiten, welches diese State Maschine implementiert. Kochrezeptartig kann man dabei folgenden Aufbau vornehmen:

- Es gibt eine globale Variable, die den aktuellen Zustand der Maschine repräsentiert. Die Zustände wurden in obiger Tabelle bereits durchnummeriert, so dass es sich anbietet, Zustände innerhalb der Maschine durch ebendiese Zahlen darzustellen.
- Die FSM wird als Funktion implementiert, die für jeden einzelnen Takt aufgerufen wird.
- Jeder Zustand wird innerhalb der Funktion durch einen case innerhalb einer switch Anweisung dargestellt.
- Jeder Zustand kann vor verlassen der Funktion den aktuellen Zustand der FSM beim nächsten Aufruf der Funktion festlegen, indem er an die globale Variable die Nummer des nächsten Zustands zuweist.
- Jegliche Form von Warteschleifen innerhalb der FSM ist verboten. Wenn die FSM auf ein Ereignis warten müsste, dann ist dafür ein eigener Zustand vorzusehen, der auf das Eintreten des Ereignisses prüft und nur dann den nächsten Zustand auswählt,

wenn das Ereignis tatsächlich eingetreten ist. Damit erreicht man Multitasking.

- Es ist sinnvoll, den Zuständen Namen in Form eines #define oder enums zu geben, damit wird das Konstrukt deutlich leichter lesbar.

Die Task-Funktionen des Zustandsautomaten sind so geschrieben, dass sie bei jedem Schleifendurchlauf Bescheid wissen, welchen kleinen Arbeitsschritt sie als nächstes zu tun haben. Diese Information merkt sich der Task in seiner Zustands-Variable.

- Was muss ich jetzt tun? Das tu ich jetzt!
- Was will ich beim nächsten Schleifendurchlauf tun? Das merke ich mir!

Damit haben wir den ersten Teil der State Maschine geschafft. Eigentlich ist noch nicht viel zu sehen. Ein paar Grundlagen, ein paar einfache Beispiele, ein paar Erklärungen.
Im nächsten Teil geht es weiter.

Einige Teile des Textes wurden zur besseren Übersicht farblich gestaltet.
Die Nutzung erfolgt auf eigenes Risiko.
Ich wünsche viel Spaß beim Bauen und programmieren
Achim

myroboter@web.de