CT 2 − Praktikum 10

State-Event Technik: Ampelsteuerung

1 Einleitung

In diesem Praktikum werden Sie eine einfache Ampelsteuerung mit einem Zustandsautomaten (FSM[1]) als **State Event System** in UML beschreiben und anschliessend in C realisieren. Für die Verifikation steht ein Ampelmodell zur Verfügung.

2 Lernziele

Sie lernen in diesem Praktikum für ein reales Beispiel eine Steuerung mit einem Zustandsautomaten zu beschreiben und in Software umzusetzen. Gleichzeitig lernen Sie die Grundlagen der State-Event Technik kennen.

3 Vorgehen

Gehen Sie in folgenden Schritten vor:

- Zeichnen Sie ein UML-Diagramm für die Ampelsteuerung.
- Machen Sie sich mit den zur Verfügung gestellten Softwarekomponenten vertraut.
- Implementieren und testen Sie die einzelnen Module resp. Komponenten, sowie die gesamte Ampelsteuerung.

4 Ampelsteuerung

Die Ampelsteuerung regelt den Verkehr an einer Kreuzung mit zwei Fussgängerstreifen. Siehe Abbildung 1.

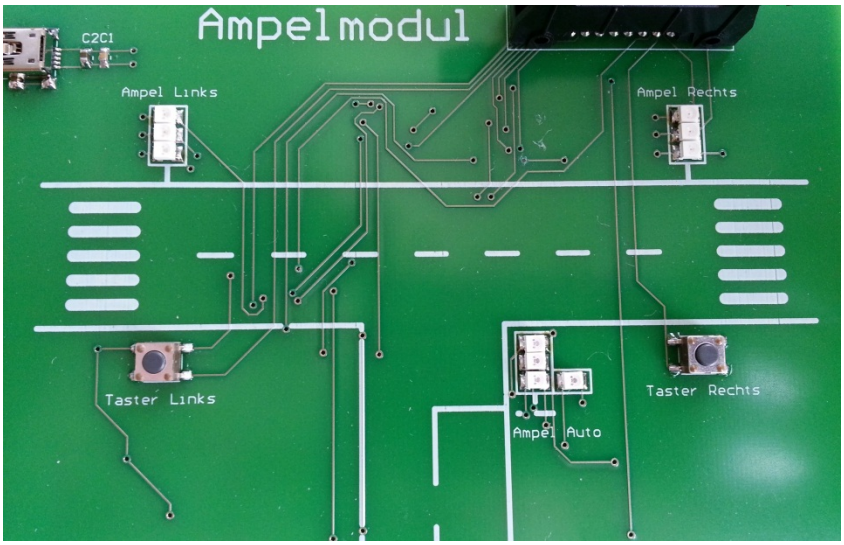

Abbildung 1: Ampelmodul

[1] Finite State Machine

In einem ersten Schritt soll eine Steuerung für den rechten Teil der Kreuzung entwickelt werden, d.h. für die Ampel Auto, Ampel Fussgänger rechts und den Taster rechts. Die Ampeln besitzen drei Signallampen: rot, gelb und grün.

Die Steuerung soll wie folgt arbeiten:

- Nach dem Aufstarten (Reset) des Systems werden die Ampeln für Autos und Fussgänger für 4 Sek. auf rot gestellt, dann wird für Autos die Fahrt freigegeben.
- Sobald der Taster gedrückt wird, geht die Ampel für Autos während 2 Sek. auf gelb, dann für Autos auf rot und für Fussgänger auf grün.
- Die Grünphase für die Fussgänger dauert 4 Sekunden. Dann wechselt die Ampel für 2 Sek. auf gelb und anschliessend auf rot sowie für Autos wieder zurück auf grün
- Wird während der Grünphase für Fussgänger der Taster erneut gedrückt, verlängert sich die Grünphase um 4 Sekunden.

5 Software Design

Die gesamte Ampelsteuerung kann mit dem Struktogramm in Abbildung 2 beschrieben werden:

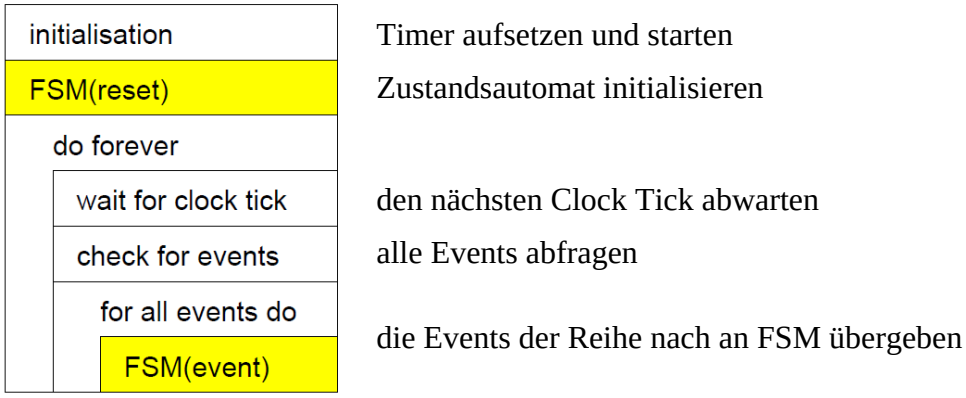

Abbildung 2: Struktogramm Ampelsteuerung

Der Zustandsautomat wird mit der Funktion `FSM(event)` realisiert, der als Parameter ein Event (Ereignis) übergeben wird, die sonst aber vollständig autonom arbeitet. Z.B. wird der Zustand in der Funktion selbst verwaltet und es werden Aktionen aufgerufen resp. Meldungen an die Ampel gesendet.

Das Hauptprogramm und die Infrastruktur für die Realisierung des Timers stellen wir Ihnen vollständig zur Verfügung. Die Funktionen für das Erfassen der Events, den Zustandsautomaten und das Auslösen der Aktionen (Setzen der Ampeln) müssen Sie selber implementieren, dazu stehen aber Rahmenprogramme zur Verfügung.

Das gesamte Programm ist in folgende Module aufgeteilt:

main*	implementiert den gesamten Ablauf nach obigem Struktogramm
fsm	implementiert den Zustandsautomaten, reagiert auf Events und löst entsprechende Aktionen (Meldungen) aus
events	bestimmt die Events, die durch Drücken eines Tasters oder den Ablauf des Timers erzeugt werden
actions	implementiert die Aktionen zum Setzen der Ampel für die Autos und für die Fussgänger

timer* implementiert die Funktionen für die Initialisierung, das Starten und Auslesen des Timers

clock* initialisiert und startet den Timer des Prozessors. Wird durch das Modul timer verwendet.

Die mit * bezeichneten Module sowie alle Header Files stellen wir Ihnen vollständig zur Verfügung.

5.1 Das Modul main (`main.c`)

Im Hauptprogramm wird zuerst der Timer initialisiert. Anschliessend wird der Zustandsautomat in den Startzustand versetzt, indem das Event Reset übergeben wird. In der nachfolgenden unendlichen Schleife wird zuerst auf das Eintreffen des nächsten Timer-Tics gewartet. Dann werden alle möglichen Ereignisse evaluiert resp. abgefragt und der Reihe nach dem Zustandsautomaten übergeben.

5.2 Zustandsautomat (`fsm.h, fsm.c`)

Der Zustandsautomat ist vordefiniert und besteht aus einer Funktion, der als Parameter jeweils ein Event übergeben werden muss:

```
FSM(unsigned char event)
```

Die Events sind als `unsigned char` Konstanten in `events.h` definiert.

Definieren Sie die Zustände für den Zustandsautomaten in `fsm.c` als Konstanten und vergessen Sie nicht, dass der aktuelle Zustand in einer statischen Variable abgelegt werden muss.

5.3 Das Modul events (`events.h, events.c`)

Events entsprechen Änderungen der Eingangssignale und nicht Signalen, die dauernd anliegen. Events sind aus dieser Sicht etwas Einmaliges, können aber natürlich auch repetitiv sein.

5.3.1 Taster Events

Ein Taster Event liegt dann vor, wenn der Taster gedrückt wurde, d.h. wenn er bei der letzten Abfrage noch nicht gedrückt war, bei der aktuellen Abfrage aber gedrückt ist (das Loslassen des Tasters ist damit kein Event). Das Abfragen resp. Evaluieren des Taster Events geschieht mit der Funktion:

```
unsigned char getRightKeyEvent();
```

Rückgabewert ist entweder `NOEVENT` oder `KEYPRESSED_R`.

Die Funktion muss den Tasterwert aus dem letzten Aufruf kennen, dieser kann in einer statisch deklarierten Variable abgelegt werden. Damit lässt sich eine globale Variable vermeiden.

```
unsigned char getKeyEvent(void) {
    static unsigned char lastKey = ?;
    unsigned char currentKey;
...
}
```

Verwenden Sie für das Einlesen des Tasters die Funktion für 8-Bit Variablen aus <`dos.h`>:

```
unsigned char inportb(unsigned int PORT_ADDR);
```

Die Taster liegen am Input-Port 707h. Der „Taster Rechts“ belegt dabei das Bit 1 und der „Taster Links“ das Bit 0. Ist der Taster gedrückt, steht das entsprechende Bit auf ‚1‘.

5.3.2 Timer events

Im Modul timer wird ein Timer gestartet, indem ein Zähler auf einen vordefinierten Wert gesetzt wird. Dieser wird in einer Timer ISR bis auf 0 dekrementiert. Ein Timer Event liegt also vor, wenn bei der letzten Abfrage des Timers der Zählerstand noch nicht 0 war, neu aber gleich 0 ist (ein abgelaufener Timer kann damit kein Event mehr erzeugen). Der Timerstand kann mit der Funktion `getTimerValue()` aus dem Modul timer abgefragt werden.

Das Abfragen resp. Evaluieren des Timer Events geschieht mit der Funktion:

```
unsigned char getTimerEvent(void);
```

Rückgabewert ist entweder `NOEVENT` oder `TIMEOUT`.

Hinweis: das Verwalten des letzten Timerstands lässt sich gleich handhaben, wie der letzte Tastendruck beim Taster Event.

5.4 Das Modul actions (`actions.h, action.c`)

Aktionen oder Meldungen sind in unserem Fall das Setzen der entsprechenden Ampel-Lampen. Die Funktionen für das Auslösen der entsprechenden Aktionen sind vordefiniert und müssen von Ihnen noch programmiert werden:

```
void setAmpelAuto(unsigned char lights);
void setAmpelFussR(unsigned char lights);
```

Die Lampen werden über das Output-Port `0x703` wie folgt angesteuert:

Bit	**Bedeutung**
7 .. 6	Warnsignal für Rechtsabbieger 00 aus 01 aus 10 orange ein 11 orange ein
5 .. 4	Ampel Fussgänger links 00 rot 01 gelb 10 gelb 11 grün
3 .. 2	Ampel Fussgänger rechts Farbzuordnung wie Ampel Fussgänger links
1 .. 0	Ampel Auto Farbzuordnung wie Ampel Fussgänger links

In `actions.h` sind Konstanten für die Farben vordefiniert. Diese dienen als Bit-Masken und können direkt beim Aufruf der entsprechenden Funktion verwendet werden, z.B.

```
setAmpelAuto(RED);
```

Verwenden Sie für die Ausgabe auf das Output-Port die Funktion für 8-Bit Variablen aus `<dos.h>`:

```
void outportb(unsigned int PORT_ADDR, unsigned char value);
```

5.5 Das Modul timer (`timer.h, timer.c`)

Das Modul wird vollständig zur Verfügung gestellt. Es stellt nach aussen drei Funktionen zur Verfügung:

```
void initTimer(void);
void startTimer(unsigned int ticks);
unsigned int getTimerValue(void),
```

`startTimer()` setzt einen Abwärtszähler auf die benötigte Dauer (Dauer = Anzahl Tics x 200ms)

`getTimerValue()` gibt den aktuellen Stand des Abwärtszählers zurück.

6 Falls Sie noch Zeit haben …

Erweitern Sie Ihre Ampelsteuerung für die ganze Kreuzung. Definieren Sie dazu zuerst das genaue Verhalten mit einem UML Diagramm. Wird der Taster Rechts gedrückt, so bleibt die Autoampel auf grün, zusätzlich soll aber das Warnsignal für Rechtsabbieger blinken. Wird der Taster links gedrückt, haben die Autos rot. Erweitern Sie Ihren Code entsprechend.