JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

IndInf01-02: The Art of State Machines & Ampelsteuerung

Systemtechnik Labor
5BHITT 2015/16, Gruppe X

Stefan Erceg

Version 1.0

Note:

Betreuer: Prof. Weiser

Begonnen am 25. September 2015

Beendet am 1. Oktober 2015

Inhaltsverzeichnis

1 Einführung

Diese Übung zeigt wie eine Implementierung einer Ampelsteuerung in C basierend auf der Umsetzung von State Machines erfolgen kann.

1.1 Ziele

Das Ziel dieser Übung ist die Fähigkeit zu besitzen, die verschiedenen Arten der State Machines in C umsetzen zu können. Das State Pattern ist jedoch nicht in C umsetzbar, da diese State Machine eine objektorientierte Lösung besitzt.

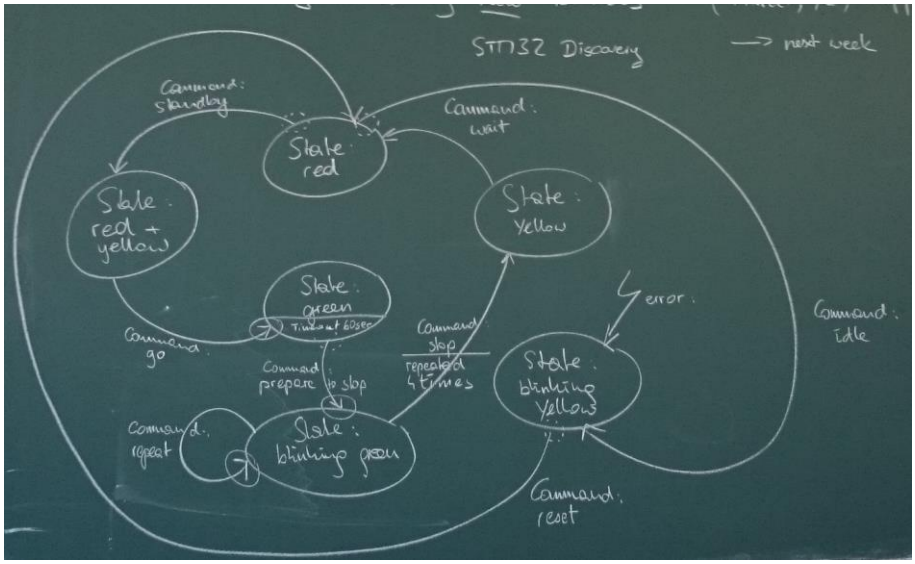
1.2 Voraussetzungen

- STM32F3-Discovery Mikrocontroller
- Grundlagen über Mikrocontroller-Programmierung

1.3 Aufgabenstellung

Implement a component based C-Programm to show the difference of the 5 types of state machines presented in the book of Mrs. Elicia White "Making Embedded Systems" with traffic light system we discussed in the lesson. To test your implementation you can use simple output functions (e.g. fprintf), but be prepared to implement it also on hardware (GPIO with Leds, Timers, etc.).

Don't forget to document the differences (advantages/disadvantages) in your protocol.



Installiere die auf Eclipse-CDT basierende Workbench von STMicroelectronics für die Familie von STM32-Boards. Implementiere auf unserem Testboard eine Ampel.

2 Ergebnisse

2.1 Arten von State Machines

2.1.1 State-Centric State Machine

Beschreibung

Bei dieser Art wird der Status überprüft und je nach Status das Kommando durchgeführt. Die State-Centric State Machine kann deren Inhalt ändern und zu einem neuen Status wechseln.

Aufbau

```
case (state):
     if event valid for this state
          handle event
          prepare for new state
          set new state
```

[1]

Vorteile

- einfacher Aufbau einer State Machine

Nachteile

- keine lose Kopplung (ein Status ist von dem anderem abhängig)
- jeder Status muss sich im Klaren sein, wie und wann er zum nächsten Status wechselt
- je mehr Status es gibt, desto kleiner ist die Effizienz

2.1.2 State-Centric State Machine with Hidden Transitions

Beschreibung

Hier wird ein bestimmter Status ausgeführt. Dieser wird nach einer bestimmten Änderung gewechselt und somit die nächste Status Funktion aufgerufen. Der Unterschied bei dieser Art von State Machine im Gegensatz zur State-Centric State Machine ist der, dass die Status mehr gekoppelt und weniger voneinander abhängig sind.

Aufbau

```
case (state):
     make sure current state is actively doing what it needs
     if event valid for this state
          call next state function
```

[1]

Vorteile

- gekoppelte Lösung
- weniger Abhängigkeiten

Nachteile

- genaue Ordnung der Status
- nicht geeignet bei Programmen, bei denen die Status viele Abhängigkeiten untereinander besitzen

2.1.3 Event-Centric State Machine

Beschreibung

Diese Art von State Machine wartet, bis ein bestimmter Event auftritt. Sobald dieser erfolgt, wird zu dem neuen Status gewechselt.

Aufbau

```
case (event):
     if state transition for this event
          go to new state
```

[1]

Vorteile

- weniger Codezeilen notwendig
- geeignet bei komplexen State Machine-Modellen (z.B. wenn viele Events auftreten müssen, damit der Status gewechselt wird)

Nachteile

- keine dynamische Lösung
- je mehr Events existieren, auf die reagiert werden muss, desto kleiner ist die Effizienz

2.1.4 State Pattern

Beschreibung

Hier handelt es sich um eine objektorientierte Lösung, da man jeden Status als Objekt betrachtet. In diesem Objekt werden Funktionen erstellt, damit diese auf einen bestimmten Event reagieren können. Folgende Funktionen werden für jedes Objekt angeboten:

- Enter – wird aufgerufen, wenn man dem Status beigetreten ist
- Exit – wird aufgerufen, wenn man den Status verlässt
- EventGo – verarbeitet den Start eines bestimmten Events
- EventStop – verarbeitet das Ende eines bestimmten Events
- Housekeeping – geeignet für Zustände, die periodisch überprüft werden (z.B. Timeout)

Aufbau

```
class Context {
      class State Red, Yellow, Green;
      class State Current;

constructor:
      Current = Red;
      Current.Enter();

destructor:
      Current.Exit();

Go:
      if (Current.Go() indicates a state change)
            NextState();

Stop:
      if (Current.Stop() indicates a state change)
            NextState();

Housekeeping:
      if (Current.Housekeeping() indicates a state change)
            NextState();

NextState:
      Current.Exit();
      if (Current is Red) Current = Green;
      if (Current is Yellow) Current = Red;
      if (Current is Green) Current = Yellow;
      Current.Enter();
}
```

[1]

Vorteile

- objektorientierte Lösung
- gekoppelte Lösung
- dynamische Lösung
- eine Funktion pro Event

Nachteile

- komplexeste Art einer State Machine
- Implementierung in C nicht einfach, da dort nur Structs und keine Objekte existieren

2.1.5 Table-Driven State Machine

Beschreibung

Wie der Name dieser State Machine bereits schon sagt, wird hier eine Tabelle verwendet.

Aufbau

states ▶ / events ▼	Inactive	Pause	FadeIn	Display	FadeOut
mouseover	cancel timer save cursor position start time next state is Pause				move tooltip to cursor next state is FadeIn
mousemove		do [Inactive, mouseover]	move tooltip to cursor	do [FadeIn, mousemove]	do [FadeIn, mousemove]
mouseout		cancel timer next state is Inactive	next state is FadeOut	do [Display, timeout]	do nothing
timeout		create tooltip at cursor start ticker next state is FadeIn		start ticker next state is FadeOut	
timetick			Increase opacity If opacity ≥ maximum cancel ticker start timer next state is Display		Decrease opacity If opacity ≤ 0 cancel ticker delete tooltip next state is Inactive

Bei der oberen Abbildung stellen die Zeilen die möglichen Events und die Spalten den Status dar. Der Code wird dadurch in 2 Teile aufgeteilt:

1. Teil: Darstellung der Tabelle, auf Grund welcher man weiß, welcher Status nach einem bestimmten Event ausgelöst werden soll
2. Teil: Maschine, welche die Tabelle ausliest

[1]

Vorteile

- Aufteilung des Codes in 2 Teile → gut lesbarer Code

Nachteile

- Fehler in der Tabelle können leicht übersehen werden

2.2 Einrichten von Eclipse & Konfiguration des Mikrocontrollers

Da diese Aufgabe mit Eclipse CDT Mars durchgeführt wird, wurde das Programm zuerst heruntergeladen [2] und in diesem das Plug-In von OpenSTM [3] installiert.

Damit der STM32F3-Discovery erkannt wird, wurde ein Treiber unter folgender Seite [4] heruntergeladen und installiert.

In Eclipse wurden folgende Schritte durchgeführt:

- New C Project
  - Project type: Empty Project
  - Toolchains: Ac6 STM32 MCU GCC
- MCU Configuration
  - Series: STM32F3
  - Board: STM32F3DISCOVERY
- Project Firmware configuration
  - "Hardware Abstraction Layer (Cube HAL)" ausgewählt
  - auf den Button "Download target firmware" geklickt

Nach Klicken des Buttons „Finish" sind 2 Fehlermeldungen aufgetaucht. Diese wurden gelöscht und danach das Projekt gecleant und gebuildet.

Bevor man das Programm ausführt, muss man noch bei den „Debug Configurations" eine neue „Ac6 STM32 Debugging"-Konfiguration erstellen. Nach Klick auf den Button „Debug" wird das Programm dann erfolgreich auf den STM32-Mikrocontroller geflasht und ausgeführt.

2.3 Implementierung der State Machines

In einem c-File habe ich zunächst alle Funktionen für die verschiedenen Zustände der LEDs programmiert (rot, grün, grün blinkend, usw.). In diesem File existiert ebenfalls eine Funktion zum Reseten der LEDs, die zu Beginn jeder Set-Funktion aufgerufen wird.

Möchte man z.B. die rote LED aufleuchten lassen, muss dieser Befehl geschrieben werden:

```
BSP_LED_On(LED_RED);
```

Um die LED auszuschalten, wird dieser Befehl verwendet:

```
BSP_LED_Off(LED_RED);
```

Weiters habe ich dann für jede einzelne State Machine (bis auf das Table Driven und das State Pattern, da dies auf einer objektorientierten Sprache basiert und daher in C nicht umgesetzt werden kann) jeweils ein eigenes c-File geschrieben, um die State Machines möglichst gut voneinander zu kapseln.

Das Header-File „state_machine.h" wurde erstellt, um dort die Enums für die Zustände der LEDs und die Events zu lagern:

```
typedef enum {
     RED,
     RED_YELLOW,
     GREEN,
     GREEN_BLINK,
     YELLOW,
     YELLOW_BLINK
} LEDstate;

typedef enum {
     // yellow to red
     STOP,
     // red to red-yellow
     PREPAREFORGOING,
     // yellow-red to green
     GO,
     // green to green-blink
     PREPAREFORWAITING,
     // green-blink to yellow
     CAUTION,
     // error state in case something goes wrong
     ERR
} LEDevent;
```

Ebenfalls existiert dort ein Struct, in dem der aktuelle Zustand und das aktuelle Event gespeichert werden:

```
typedef struct {
     LEDstate currentState;
     LEDevent currentEvent;
} currentTrafficLight;
```

In der main-Funktion wird dann die Funktion aufgerufen, welche in den jeweiligen Files der State Machines implementiert wurden. Vor dem Aufruf werden der STM und die LEDs folgendermaßen initialisiert:

```
SystemInit();
SystemCoreClockUpdate();

SysTick_Config(SystemCoreClock / 1000);

BSP_LED_Init(LED_RED);
BSP_LED_Init(LED_ORANGE);
BSP_LED_Init(LED_GREEN_2);
```

3 Zeitaufwand

3.1 Zeitabschätzung

Teilaufgabe	benötigte Zeit
Eclipse einrichten & Board konfigurieren	50 Minuten
Programm implementieren	100 Minuten
Protokoll schreiben	60 Minuten
Gesamt	**210 Minuten (3 h 30 min)**

3.2 Tatsächlicher Zeitaufwand

Teilaufgabe	benötigte Zeit
Eclipse einrichten & Board konfigurieren	90 Minuten
Programm implementieren	125 Minuten
Protokoll schreiben	70 Minuten
Gesamt	**285 Minuten (4 h 45 min)**

Es wurde geschätzt, dass diese Übung im Labor-Unterricht, der von 8 Uhr bis 11:30 Uhr stattfindet, fertiggestellt wird. Da zu diesem Zeitpunkt in der Schule das WLAN keine starke Verbindung hatte, haben das Einrichten von Eclipse und das Konfigurieren des Boards länger gedauert als gedacht. Daher kam es zu Hause zu einem Zusatzaufwand von einer Stunde und 15 Minuten.

4 Lessons learned

- gelernt, bei welchen Anwendungsfällen man die jeweilige Art von State Machine verwenden soll
- Vertiefung der Kenntnisse bei der Mikrocontrollerprogrammierung

5 Quellenangaben

[1] Elecia White (2011). Making Embedded Systems [Online].
Available at: http://it-ebooks.info/book/549/
[zuletzt abgerufen am 25.09.2015]

[2] The Eclipse Foundation (2015).
C/C++ Development Tooling (CDT) [Online].
Available at: https://projects.eclipse.org/projects/tools.cdt
[zuletzt abgerufen am 25.09.2015]

[3] OpenSTM32 (2013). OpenSTM32 Community [Online].
Available at: http://www.openstm32.org/
[zuletzt abgerufen am 25.09.2015]

[4] STMicroelectronics (2015). ST-LINK/V2-1 USB driver on
Windows Vista, 7 and 8 [Online]. Available at:
http://www.st.com/web/catalog/tools/FM147/SC1887/PF260218
[zuletzt abgerufen am 25.09.2015]