JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Systemtechnik Labor
5BHITT 2015/16, Gruppe Y

Jakub Kopec

Version 0.1

Note:

Betreuer: Weiser Johann

Begonnen am 25. September 2015

Beendet am 10. Oktober 2015

Inhaltsverzeichnis

1 Einführung

Dieses Protokoll beschreibt wie die erste Aufgabe auf einem Mikrocontroller im Systemtechnik Labor realisiert wurde.

1.1 Ziele

Das Ziel der ersten Aufgabe am Mikrocontroller war es, eine Ampelsteuerung mittels State Machines zu realisieren.

1.2 Voraussetzungen

Um die Aufgabe erfolgreich abschließen zu können muss man im Besitz eines Mikrocontrollers sein. In unserem Fall ist das der STM32F3Discovery. In weiterer Folge ist es notwendig eine lauffähige IDE zu besitzen, welche mit dem Mikrocontroller kommunizieren kann. Sofern man diese Anforderungen erfüllt hat, muss man sich darüber im Klaren sein, was State Machines sind, beziehungsweise wie State Machines funktionieren. Diese Informationen können im Scriptum von Elica White „Making Embedded Systems“ im Kapitel 5 nachgelesen werden. Dort werden alle verschiedenen Arten von State Machines mit Vor- und Nachteilen so wie ihrer Funktionsweise beschrieben.

1.3 Aufgabenstellung

Hiermit bediene ich mich mit einem Screenshot aus Elearning:

INDINF02: AMPELSTEUERUNG

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

2 State Machines

Beim endlichen Zustandsautomaten (engl. finite state machine, kurz FSM) 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.

2.1 State-Centric State Machine

Beschreibung:
Für die State-Centric State Machine müssen zuvor Events und States für jeden möglichen Zustand definiert werden. In einem Switch-Case Konstrukt wird der aktuelle Status abgehandelt. In jedem Case Statement wird zusätzlich noch geprüft, ob das Event für den Status akzeptabel ist. Danach werden der nächste Status und das zugehörige Event gesetzt
(z.B. wenn Status Rot & Event Rot: setze Status Rot-Gelb & Event Rot-Gelb)

Vorteile:

- Einfache Möglichkeit eine State Machine zu programmieren.
- Unkompliziert aufgebaut

Nachteile:

- Komplett statisch - Neue Zustände müssen händisch hinzugefügt werden
- Jeder Zustand muss wissen WIE und WANN er zum Einsatz kommt
- Je mehr Zustände es gibt desto ineffizienter
- Daten sind nicht gekapselt und werden nicht von äußeren Zugriffen geschützt

2.2 State-Centric State Machine with Hidden Transitions

Beschreibung:
Wie der Name schon sagt, basiert diese State-Machine auf der vorherigen. Wieder wird das Switch-Case Konstrukt nach dem Status aufgelöst, aber hierbei wird auf die Abfrage des zugehörigen Events verzichtet. (z.B. Status Gelb -> Status Grün & Event wird ignoriert)

Vorteile:
- Mehr Verkapselung als bei der normalen State-Centric State Machine
- Geringere Abhängigkeiten als bei der normalen State-Centric State Machine
- Effizienter

Nachteile:
- Komplett statisch - Neue Zustände müssen händisch hinzugefügt werden

2.3 Event-Centric State Machine

Beschreibung:
Bei dieser State-Machine kommt wieder ein Switch-Case Konstrukt zum Einsatz. Im Gegensatz zu den Vorherigen, wird hierbei aber nach den Events aufgelöst (in den Switch Cases wird nach Events abgefragt). Wie bei der State-Centric State-Machine wird wieder überprüft ob der entsprechende Status passend ist.
(z.B. wenn Status Rot & Event Rot: setze Status Rot-Gelb & Event Rot-Gelb)

Vorteile:
- Weniger Codezeilen benötigt
- Für komplexe State Machines geeignet (z.B.: wenn viele Events auftreten müssen, damit der Status gewechselt wird)

Nachteile:
- Komplett statisch. Nicht dynamisch
- Je mehr Zustände es gibt desto ineffizienter

2.4 State-Pattern State Machine

Beschreibung:
Dies ist eine objektorientierte Umsetzung einer State Machine. Es wird eine Klasse erstellt welche verschiedene Methoden besitzt die jeweils auf ein Event reagieren:

- 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)

Vorteile:
- Dynamisch
- Objekt orientiert
- Verkapselt
- Eine Methode um jedes Event zu verwalten

Nachteile:
- Viel komplexer als die vorherigen Lösungen
- Nicht einfach zu verstehen
- Es ist nicht möglich dieses Pattern in C umzusetzen, da C nicht Objekt-Orientiert arbeitet.

2.5 Table Driven State Machine

Beschreibung:

Hierbei wird eine Tabelle verwendet um die verschiedenen Status darzustellen.

Vorteile:

- Eine gute Möglichkeit für komplexe State Machines
- Wiederverwendbar
- Man hat zwei Codes anstatt von einem komplexen Code
- Übersichtlich

Nachteile:

- Es passiert schnell, dass man Fehler in der Tabelle macht.

3 Ergebnisse

3.1 Vorbereitung

Als allererstes müssen alle Anforderungen erfüllt werden um in weiterer Folge die Ziele dieser Aufgabe realisieren zu können.

Einen Mikrocontroller hatte ich und über die State Machines war ich mir nach der in den Voraussetzungen erwähnten Lektüre im Klaren.

Ein sehr großes Problem stellte für mich eine lauffähige Toolchain dar, da ich von meinem Host-System aus arbeiten wollte. Ich habe ein MacBook und daher ist mein OS logischer weise OSX. Das stellte mich vor mehrere Herausforderungen, da Ac6 nur Windows und Linux unterstützt. Ich habe also mehrere Stunden damit verbracht, eine Lösung für dieses Problem zu finden. Schlussendlich habe ich es auch geschafft und eine eigene Anleitung dafür geschrieben. Ich habe die Anleitung mit Absicht von diesem Protokoll getrennt, damit meine Anleitung verbreitet werden kann für alle Mac-User, die auch dieses Problem vielleicht in Zukunft haben werden.

3.2 Projekterstellung

Sobald ich endlich eine lauffähige IDE hatte konnte ich beim Wizard die Einstellungen der neu installierten Software benutzen und die STM Einstellungen anpassen. Dafür verwende ich die Toolchain Ac6 STM32 MCU GCC und für das Debuggen die Debug-Configuration von Ac6.

3.3 Auswahl der State Machine Implementation

Ich habe mich entschieden diese Aufgabe mit der „**State Centric State Machine with Hidden Transitions**“ zu lösen und das aus dem Grund, dass sie weniger Abhängigkeiten hat als die ursprüngliche **State Centric State Machine** und effizienter ist. Die anderen State Machines haben mir nicht so zugesagt, da die **Event Centric State Machine** wieder mehr Abhängigkeiten haben müsste und eher für komplexe State Machines geeignet ist und ich mit der **Table Driven State Machine** Probleme bei der Umsetzung hatte.

3.4 LED Funktionen auslagern

Da ich während der Implementierung darauf gekommen bin, dass die LED – Aufrufe oft doppelt oder mehrfach vorkommen, habe ich diese als erstes in Funktionen ausgelagert und nach Farben und Funktionalität sortiert. Da es aber mit der Zeit sehr viele wurden, habe ich mich entschieden diese in ein eigenes .c-File zu schreiben.

3.5 Implementieren der State Machine

Den Ablauf der State Machine ist mit Hilfe von Doxygen dokumentiert, deshalb werde ich hier nicht näher darauf eingehen. Allerdings musste ich mir überlegen, wo ich den aktuellen Zustand meiner Ampelsteuerung speichere. Als Lösung dafür, habe ich ein Enum mit allen Möglichen Status definiert und einen Struct, welcher den Zustand speichert. Die State Machine wurde natürlich auch in ein eigenes .c-File geschrieben und includet ein .h-File.

3.6 Doxygen Dokumentation

Um mit Doxygen dokumentieren zu können, muss man das auch installieren. Glücklicherweise ging das viel einfacher als die Toolchain-Installation. Ich habe folgenden Link befolgt und hatte garkeine Probleme:
https://github.com/theolind/mahm3lib/wiki/Integrating-Doxygen-with-Eclipse