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 28. Oktober 2015

Beendet am 30. November 2015

Inhaltsverzeichnis

1 Einführung

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

1.1 Ziele

Das Ziel der zweiten Aufgabe am Mikrocontroller war es, eine Ampelsteuerung mittels State Machines und Timerinterrupts zu realisieren, welche durch drücken des User-Buttons in den Zustand der Nachtschaltung versetzt werden soll.

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:

INDINF03: AMPELSTEUERUNG MIT INTERRUPTS

Implementiere eine Ampel, welche rein mit Interrupts gesteuert wird:

a) Die Ampel möge von rot-orange-grün auf orange-blinken umschalten, wenn der Userbutton gedrückt wird.

b) Sowohl die rot/orange/grün-Phasen als auch die Phasen des Orange-Blinkens sollen mittels Timerinterrupts gesteuert werden.

Das Hauptprogramm besteht dann nur noch aus der Konfiguration des Systems, hernach folgt eine "Idle"-Phase (funktionslose Endlosschleife).

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 Ausgangspunkt für diese Aufgabe ging ich von INDINF02 aus, wo nur eine Ampelsteuerung mittels einer State Machine zu realisieren war. Diese Aufgabe erweiterte ich auf die gewünschten Interrupts.

3.2 Projekterstellung

Hierfür habe ich, mit der selben Prozedur wie bei der letzten Aufgabe, ein neues Projekt erstellt und die src-Files aus der letzten Aufgabe hineinkopiert, um nicht alles von neu schreiben zu müssen.

3.3 Auswahl der State Machine Implementation

Wie auch bei der letzten Aufgabe (INDINF02) habe ich 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 Implementieren des Timerinterrupts

Was sich zur vorherigen Aufgabe geändert hat ist dass man nun das Programm nicht in einer endlosen Schleife ablaufen lässt, sondern die State Machine jede Millisekunde aufgerufen wird. Deshalb musste ich in der State Machine einen „counter“ einbauen, welcher die vergangenen Millisekunden mitzählt. Den „counter“ brauche ich um den LEDs sagen zu können wie lange sie leuchten sollen (wie viele Millisekunden lang).

3.5 Implementieren des Userbuttons

Damit auch etwas passiert, wenn man auf den User-Button drückt, musste dieser Interrupt zuerst konfiguriert und danach aktiviert werden. Dadurch wird eine „Callback“-Methode aufgerufen, welche man auch selbst implementieren musste. In der „Callback“-Methode steht drinnen, was passieren soll, wenn ein bestimmter Interrupt eintritt. In meinem Fall, wird in dieser Methode die Nachtschaltung aktiviert.

3.6 Probleme

Ein großes Problem, welches ich hatte war, dass ich vergessen habe in das File „stm32f3xx_it.c“ die Funktion „EXTI0_IRQHandler(void)“ zu schreiben, welche die Interrupts überhaupt ermöglicht. Dadurch hat sich mein Mikrocontroller immer aufgehängt und ich wusste den Grund nicht. Auf den Fehler bin ich mit Hilfe meiner sozialen Mitschüler gekommen.