FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Praktikum Rechnerstrukturen

Projektaufgabe 4.2 – Bedarfsampel mit Nachtmodus

Felix Lohse, Matrikelnummer:

Thomas Jürgensen, Matrikelnummer: 301880

Dokumentation zur Bedarfsampelimplementierung

Dozent: Prof. Dr. Ole Blaurock

Lübeck, 27.06.2018

Einleitung

Im Rahmen des Moduls “Rechnerstrukturen” sollte als Abschluss des Praktikums eine Ampelschaltung in Assembler unter Verwendung von Atmel Studio 6.2 programmiert werden. Hierbei war freigestellt, ob diese für den Debugger oder für das Board Atmel ATmega644PA funktionsfähig sein sollte.
Diese Dokumentation legt die Vorgehens- und Funktionsweise unserer Implementierung der Ampelschaltung für den Debugger dar.

Problemstellung

Die genaue Problemstellung ist aus dem Aufgabenblatt 4, Aufgabe 4-2 zu entnehmen.

Theoretische Vorleistung

Eine Fußgängerampel funktioniert folgendermaßen: Autos haben Grün, während Fußgänger Rot haben. Meldet ein Fußgänger nun Bedarf an, vergeht eine kurze Zeit und die Lampe für den Autoverkehr schaltet auf Gelb um. Danach wartet sie eine gewisse Zeit und schaltet auf Rot, während die Lampe für den Fußgängerverkehr noch immer Rot zeigt. Anschließend schaltet das rote Licht der Fußgänger auf Grün; während dieser Phase können Fußgänger die Straße überqueren, bis das grüne Licht der Fußgänger wieder dem roten weicht und nun alle Verkehrsbeteiligten wieder Rot haben. Letztlich wird für die Autos Rot-Gelb angezeigt, um von dort aus wieder Grün zu zeigen.

Die einzelnen Phasen können als Zustände aufgefasst werden, welche mit einem Automaten ausgedrückt werden können. Die Vorteile des Automaten liegen darin, dass er zustandsgetrieben und dadurch modularisiert und folglich leicht erweiterbar bzw. veränderbar ist.

Es ergibt sich folgender Zustandszyklus: Autos Grün, Fußgänger Rot; Autos Gelb, Fußgänger Rot; Autos Rot, Fußgänger Rot; Autos Rot, Fußgänger Grün; Autos Rot, Fußgänger Rot; Autos Rot-Gelb, Fußgänger Rot;

Auffallend ist das doppelte Auftreten des Zustandes, in dem alle Verkehrsteilnehmer Rot haben. Dies wird zum Vorteil genutzt und interne Vorgänge des zweiten Rot-Rot Zustandes werden anders ablaufen als die des ersten. Mehr hierzu wird in den folgenden Kapiteln erklärt. Zunächst wird davon ausgegangen, dass im zweiten Rot-Rot Zustand eine eventuelle neue Bedarfsanmeldung ausgewertet wird und im folgenden mit “AutosRotFußgängerRotRESET” bezeichnet wird.

Wird dies nun durch eine Nachtschaltung erweitert, ergibt sich ein siebter Zustand, in dem alle Lampen ausgeschaltet sind. Dieser soll ausschließlich durch den ersten Zustand erreicht werden, in dem die Autos Grün haben, wenn der Nachtmodus eingeschaltet ist. Dies geschieht zum Schutz der Fußgänger, damit die Ampel nicht während ihrer Grünphase plötzlich ausgeschaltet wird und sie im schlimmsten Fall in Gefahr durch den Straßenverkehr geraten könnten.

Die Ein- und Ausgaben des Automats sind durch die Aufgabenstellung gegeben: Es sollen die höchstwertigsten drei Bits des PORTA des Mikrocontrollers für die Eingabe und die restlichen fünf Bits für die Ausgabe verwendet werden.
Somit sind die Eingaben wie folgt kodiert:
x_0: “Nachtmodus An”, x_1: ”Nachtmodus Aus”, x_2: ”Bedarfstaster”
y_0: “Auto Grün”, y_1: ”Auto Gelb”, y_2: “Auto Rot”, y_3: “Fußgänger Grün”, y_4: “Fußgänger Rot”

Hieraus ergibt sich der Automat A={X, Y, Z ,δ, μ}, mit
X = {000, 100, 110, 101, 010, 001}
Y = {00000, 00001, 10010, 10100, 01100, 10110}
Z = {0,1,2,3,4,5,6,7}
δ: Z x X -> Z, $\delta(x_1 , x_0 , z_0) = (z_0^+)$
μ: Z -> Y, $\mu(z_0) = (y_4 , y_3 , y_2 , y_1 , y_0)$
Die Berechnungsvorschriften für δ und μ ergeben sich aus dem folgenden Zustandsdiagramm:

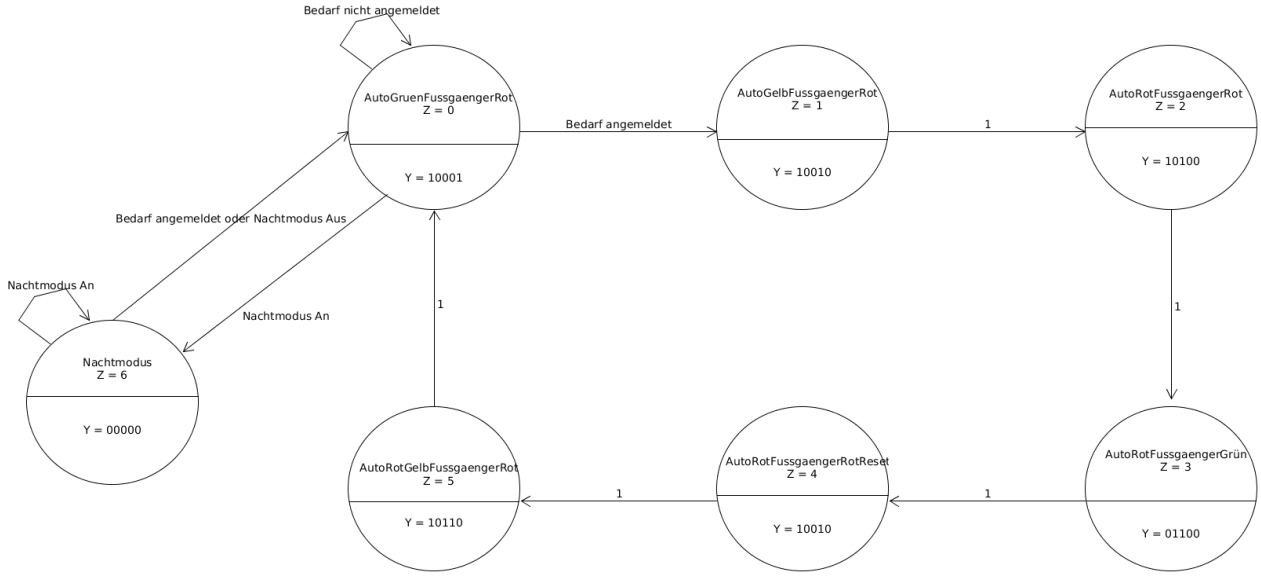

Abbildung 1: Zustandsübergangsdiagramm

Praktische Lösung

In diesem Abschnitt werden die Designentscheidungen erläutert, welche für die Implementierung getroffen wurden, sowie die Umsetzung erklärt und die Limitierungen der Umsetzung dargestellt. Die Umsetzung basiert auf dem WaitTimer, welcher von Prof. Dr. Ole Blaurock zur Verfügung gestellt wurde. Sämtliche Änderungen wurden mit Kommentaren kenntlich gemacht.

Designentscheidungen

Vor dem eigentlichen Programmieren müssen noch einige Entscheidungen für die Umsetzung getroffen werden. Im folgenden Abschnitt werden unsere Entscheidungen aufgezeigt und begründet.

Die Abfrage der Tasten geschieht über Polling innerhalb der Aufrufe des WaitTimers. Die Entscheidung gegen Interrupts begründet sich dadurch, dass der Programmfluss durch einen Interrupt unterbrochen wird und Service-Routinen ausgeführt werden. Da jedoch in der Ampelschaltung hauptsächlich gewartet wird, bis eine bestimmte Zeit abgelaufen ist, ist eine

Interrupt getriebene Umsetzung überflüssig. Somit ist auch eine bessere Kontrolle über den Programmfluss gewährleistet, da nur die Interrupts des WaitTimers ausgelöst werden.

Um abzufragen, ob eine Taste gedrückt wurde, muss dies gespeichert werden. Diese Speicherung findet in einer Variable statt, welche im SRAM abgelegt wird. Die Alternative wäre, die Information direkt in einem Register zu speichern, da diese jedoch rar gesäht sind und in komplexeren Programmen zu Speicherknappheit führen, wurde sich für die Speicherung in einer Variablen entschieden.
Durch die Speicherung ist es möglich, dass Bedarf aus jeder Phase heraus angemeldet werden kann. Die Auswertung des Bedarfs findet jedoch erst nach dem Zustand “AutosRotFußgängerRotRESET” statt, da hier die gespeicherten Variablen verändert werden.

Es werden Pullups verwendet, um den Ruhepegel der Eingangspins auf 1 zu setzen. Dies ist eigentlich nicht notwendig, da das Programm für den Simulator geschrieben wurde, würde aber eine etwaige Portierung auf die konkrete Hardware erleichtern.

Anstatt jeweils eine Wartezeitvariable für jeden Zustand zu speichern, werden ähnliche Zustände in Gruppen zusammengefasst. Somit sind weniger Variablen notwendig und Speicherplatz wird eingespart.

Der Nachtmodus kann aus jedem Zustand heraus angemeldet werden, jedoch wird er nur aus dem ersten Zustand heraus erreicht. Somit wird erst der restliche Phasenzyklus durchgelaufen, bevor der Nachtmodus letzendlich aktiviert wird.

Implementierung mit Visualisierung

In diesem Abschnitt wird die Umsetzung grafisch dargestellt und erläutert. Auf genaue Einzelheiten im Code wird nicht eingegangen, lediglich die Entscheidungen und der Programmablauf sollen hier dargestellt werden.

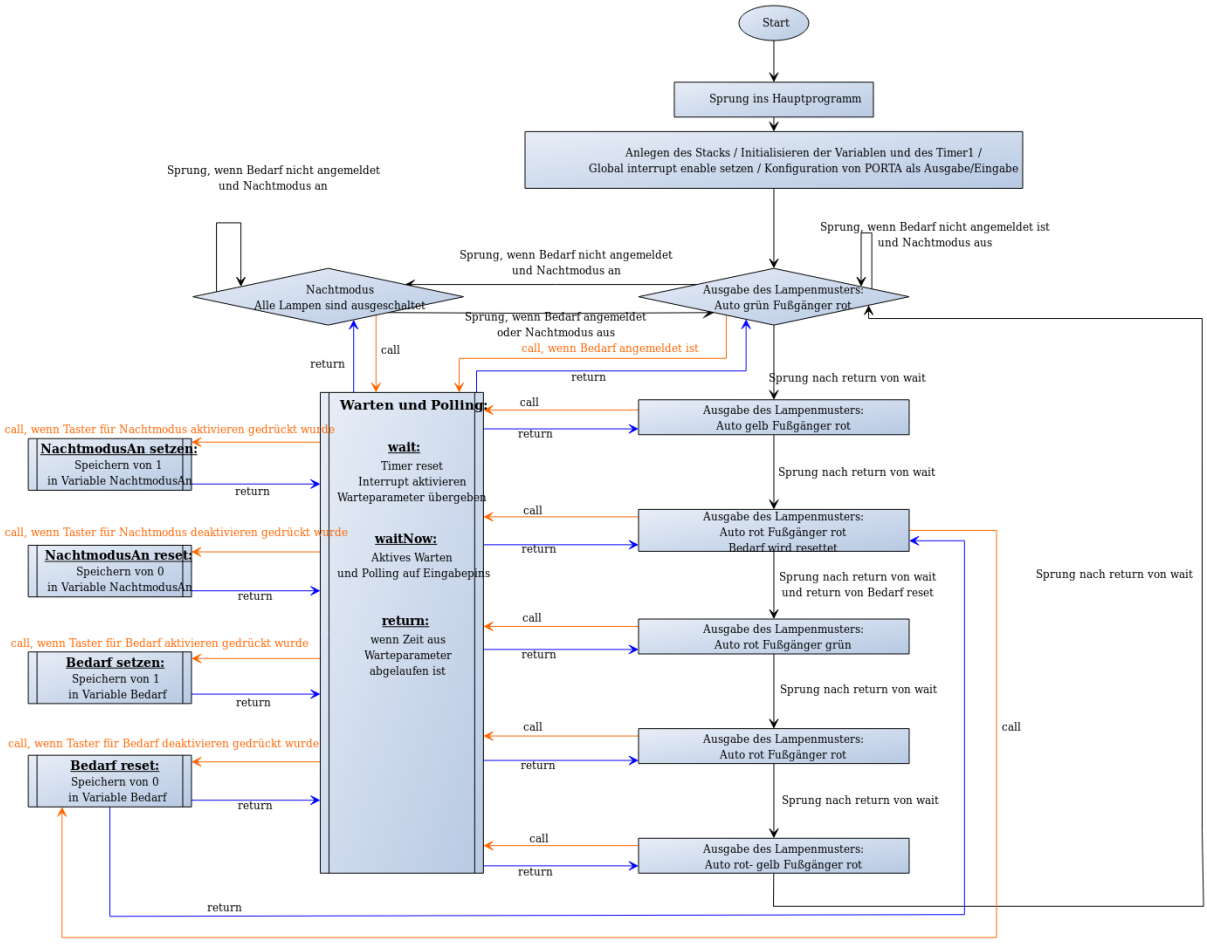

Abbildung 2: Flussdiagramm

Nach dem Sprung ins Hauptprogramm folgt der Abschnitt der Initialisierung, in der Variablen, Timer1, Global Interrupts, Stack und PORTA initialisiert bzw. konfiguriert werden.
Die Wartezeiten sind in drei Kategorien eingeteilt und es werden drei Variablen initialisiert – die Hauptphase wird 20 Sekunden Wartezeit haben, die Zwischenphasen 5 Sekunden und der Nachtmodus 0.1 Sekunden. Die kurze Wartezeit im Nachtmodus resultiert aus dem Polling während der Wartezeit. Die WaitNow aus dem WaitTimer ist sehr kurz gehalten, damit oft gepollt werden kann.
Somit ist eine hohe Abtastrate gegeben, die Reaktion auf den Knopfdruck erscheint für den Menschen unmittelbar und es wird garantiert, dass ein Tastendruck erkannt wird, da ein menschlicher Tastendruck unter allen Umständen länger als ein Taktzyklus des Mikrocontrollers andauert.

Wie es auch das Flussdiagramm zeigt, gibt es fünf Unterprogramme, wovon vier lediglich für die Speicherung von Variablen zuständig sind.
Das fünfte Unterprogramm ist der angepasste WaitTimer, welcher von jedem Zustand aus aufgerufen wird. Spätestens jetzt wird ersichtlich, dass wir uns für das Polling innerhalb des WaitTimers entschieden haben, weil der WaitTimer so oft aufgerufen wird.

Es gibt sieben Zustände, welche als Labels realisert wurden, um Sprunganweisungen benutzen zu können. Diese Zustände entsprechen den Ampelphasen, wovon zwei Phasen bedingte Sprünge enthalten. Dies ist zum Einen der Nachtmodus, in dem verweilt wird, so lange kein Bedarf angemeldet und der Nachtmodus aktiv ist, und zum Anderen die Phase, in der Autos Grün haben. Der Letztere wird erst verlassen, wenn Bedarf angemeldet oder der Nachtmodus aktiviert wird.

Resultierender Funktionsumfang und Limitierungen

Aus der Implementierung ergibt sich eine Bedarfsampel, die einen Zyklus nur dann durchläuft, wenn der Bedarfsknopf betätigt wurde.
Der Zyklus wird einmal vollständig durchlaufen und es kann jederzeit ein erneuter Bedarf angemeldet werden. Da die Grünphase der Autos mindestens 20 Sekunden anhält, ist gewährleistet, dass auch bei erneuter Anmeldung eine Grünphase gegeben ist.
Wenn die Bedarfstaste durchgehend gehalten wird, wiederholt sich der Zyklus so lange, bis die Taste losgelassen wird oder maximal einmal mehr, falls die Taste während oder nach der Rotphase für die Fußgänger losgelassen wird.
Der Nachtmodus ist immer aktivierbar und tritt erst nach einer Grünphase der Autos ein.
Aus dem Nachtmodus heraus kann jederzeit Bedarf angemeldet werden, woraufhin ein Zyklus durchlaufen und in den Nachtmodus zurückgekehrt wird.

Die gleichzeite Betätigung der “Nachtmodus Ein” und “Nachtmodus Aus” Tasten ist nicht vorgesehen und kann zu fehlerhaftem Verhalten führen.

Betriebsbedingungen

Das Programm ist für den Debugger des AtmelStudio 6.2 geschrieben. In diesem muss das richtige Board eingestellt sein und nach Ausführung des Programms muss das Pullup manuell gesetzt werden.