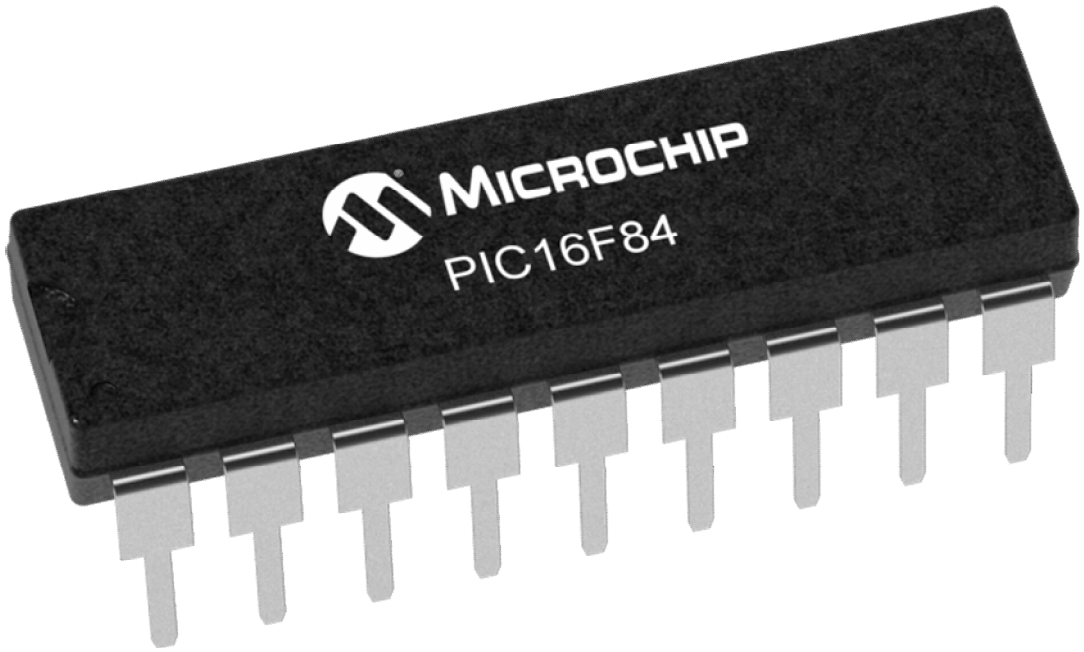
Dokumentation des Simulatorprojekts

(PIC 1816F84)



Sommersemester 2022

Fakultät: EMI

Studiengang: AI

Fach: Rechnerarchitektur

Dozent: Dipl. Ing. Stefan Lehmann

Alexander Kehl – 187957

Kai Penazzi - 188438

Hochschule Offenburg
offenburg.university

Inhaltsverzeichnis

1. Allgemeines

1.1 Grundsätzliche Arbeitsweise

Mit einem Simulator versucht man die Wirklichkeit bestmöglich mittels Algorithmen nachzubilden. Das Ergebnis ist abhängig von der Genauigkeit der Rechenregeln, sowie der Anzahl der berücksichtigten Randbedingungen. In diesem Projekt wird der Mikrocontroller PIC 16F84 simuliert. Der Fokus dabei liegt in der Funktionalität und nicht bei den internen Strukturen. Während die Simulation sich der logischen Ebene widmet, wird die elektrische Ebene ausgeblendet. Somit sind Einschwingverhalten, Schaltschwellen, ein realer Zeitbezug, Temperaturverhalten, Stromverbrauch, Verlustleistung, Spannungsschwankungen usw. in diesem Projekt zu vernachlässigen. Wenn dieser simulierte Mikrocontroller ein Programm abarbeitet, soll sich das gleiche Verhalten an den virtuellen Ein- und Ausgängen zeigen, wie bei einem echten Controller gleichen Typs.

1.2 Vor- und Nachteile einer Simulation

Eine Simulation wird in Erwägung gezogen, wenn Experimente und Messungen in der Realität zu langsam, zu schnell, zu gefährlich, zu teuer oder schlichtweg unmöglich sind. Sie ermöglichen uns eine Optimierung bestehender Systeme, dabei wird ein Augenmerk auf Effizienz und Effektivität gelegt.

Eine Simulation veranschaulicht uns einen Vorgang oder Mechanismus, um auf seine potenziellen Fehler aufmerksam zu werden. In der Realität kommen Aspekte dazu, die Probleme aufwerfen können, die bei der Simulation nicht realisierbar waren. Falls eine nahezu komplette Simulation möglich ist, sollte darauf geachtet werden, ob der Aufwand sich überhaupt lohnt, oder ob Teilelemente außenvor gelassen werden können, damit beim Erstellen der Simulation der größte Ertrag rausspringt.

1.3 Programmoberfläche und deren Handhabung

Die benutzte Programmoberfläche ist JavaFX und stellt die Visualisierung unseres Projektes dar. Hauptsächlich wurde die Oberfläche mit dem JavaFX Scene Builder erstellt, welcher uns das Codegerüst nach vollenden der Oberfläche zur Verfügung stellte. Im Scene Builder können Buttons, Checkboxen usw. eingebaut werden, welche in der Entwicklungsumgebung „Visual Studio Code“ editiert und angepasst werden können.

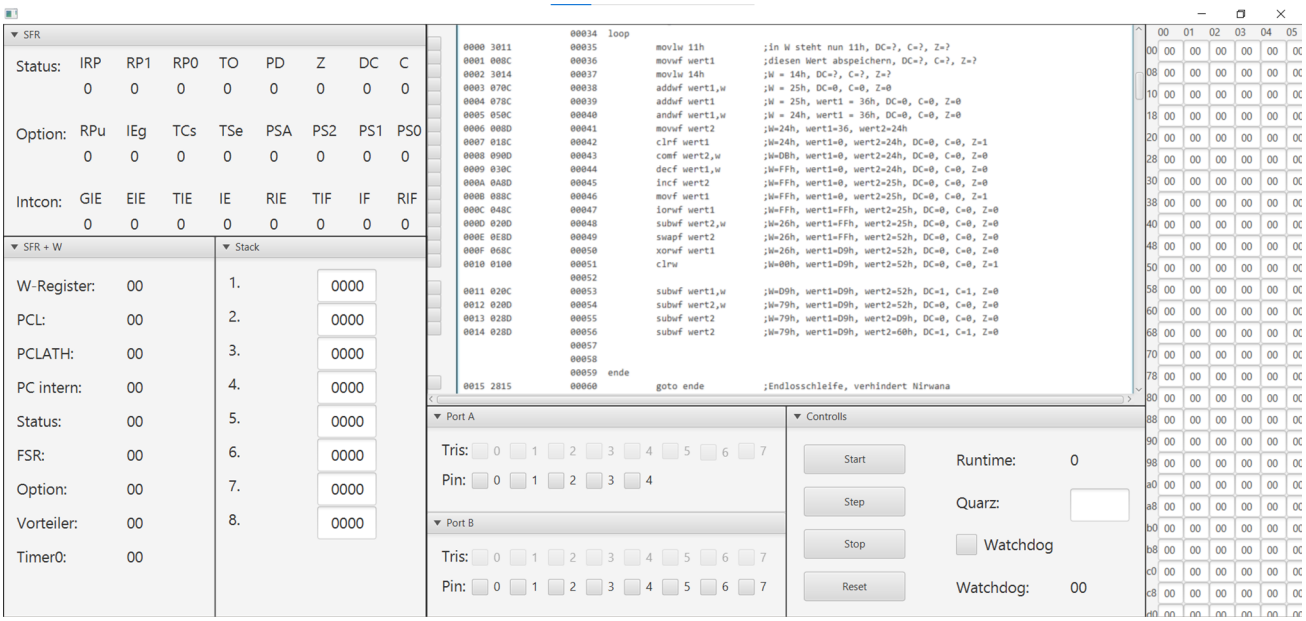


Abbildung 1: Screenshot der Programmoberfläche incl. Testprogramm3

Die Programmoberfläche wird in folgende Elemente geteilt:

„SFR“: Hier werden das Status-, Option- und Intconregister auf Bit-Ebene gezeigt,

„SFR+W“: Hier werden die Werte einzelner Register, sowie von Komponenten dargestellt,

„Stack“: Auf den Stack werden hexadezimale Zahlen gelegt,

„Port A“ und „Port B“: Portpins können ausschließlich manuell aktiviert oder deaktiviert werden,

„Programm (Testfiles)“: Hier werden die Testprogramme angezeigt. Neben jedem Befehl wird eine Checkbox angelegt, die als Breakpoint dient. Durch rote Striche wird gekennzeichnet, in welcher Befehlszeile wir uns befinden,

„Controlls“: In den Controlls kann das Programm mithilfe des „Start“-Buttons gestartet werden, sodass das Programm entweder durchläuft oder bis man an einem Breakpoint angelangt ist. Der „Stop“- Button hält das Programm an. Der „Step“- Button führt zum nächsten Befehl aus. „Reset“ lässt das Programm zurück an den Anfang setzten, sowie alle Elemente werden auf ihren Startwert gesetzt. Man kann die Quarzfrequenz in der Textbox bestimmen, sowie entscheiden, ob der Watchdog aktiviert sein soll oder nicht. Dieser funktioniert nicht wenn man mit Steps durch das Programm geht.,

„RAM“: Hier werden die Ergebnisse der Operationen gespeichert.

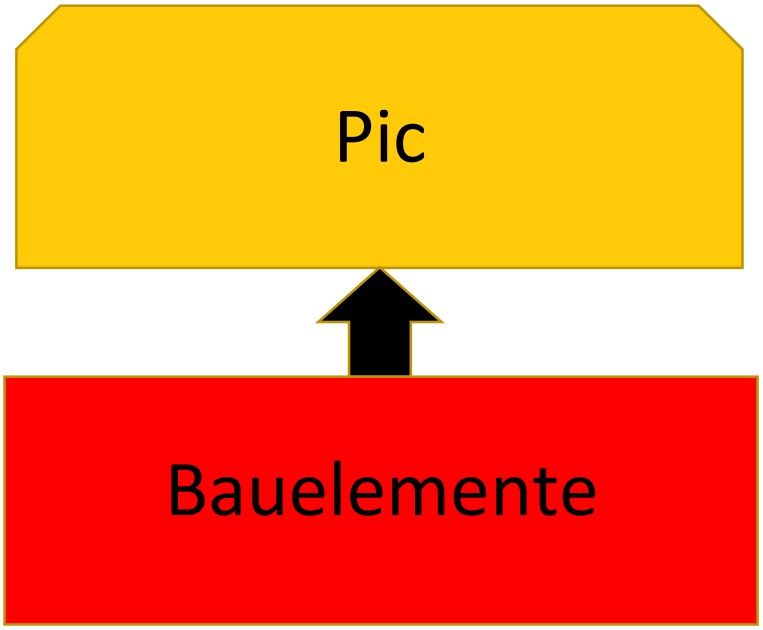
2. Realisation

2.1 Das Grundkonzept des Mikrocontrollers PIC 16F84

Der Mikrocontroller wurde mit der Programmiersprache Java realisiert. Teilweise wurden die Komponenten in eigene Klassen ausgelagert, damit die Übersicht nicht komplett ignoriert wird. Um die GUI zu realisieren, benutzten wir JavaFX mithilfe des JavaFX Scene Builders, welcher eine einfache Implementierung ermöglicht.

Die einzelnen Bauelemente werden in einer großen Klasse verknüpft, welche mit Methoden realisiert wurden. Die Methoden kommunizieren so miteinander, dass ein logisches Ergebnis entsteht.

Im Prinzip wurde der PIC in zwei Bereiche aufgeteilt:



Die einzelnen Bauelemente im untersten Bereich werden mit der Hauptklasse gekoppelt. Die Hauptklasse im Projekt entspricht mit der GUI den gesamten PIC Controller.

2.2 Tiefergehende Beschreibung der Funktionen anhand von Befehlen

2.2.1 BTFSS

Prüft das n-te Bit eines Werts. Wenn das n-te Bit eine 0 ist, wird der nächste Befehl ausgeführt, ansonsten wird der NOP Befehl ausgeführt und der Programmzähler um 1 erhöht + 1 weil der BTFSS – Befehl ausgeführt wurde. In Summe dann +2.

2.2.2 CALL

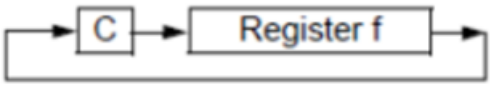
Legt Rücksprungadresse auf den Stack (Stackpointer um 1 erhöht) und der Programmzähler wird auf einen 13 – Bit Wert gesetzt.

2.2.3 MOVF

Der Wert des File-Registers wird in Abhängigkeit des Destinationbits bei 1 in sich selbst verschoben, bei 0 ins W – Register.

2.2.4 RRF

Der Wert einer Adresse im RAM wird durch das Carryflag – Bit um 1 Bit nach rechts rotiert. Das Ergebnis wird bei einer 1 als Destination - Bit im Fileregister, bei einer 0 im W – Register gespeichert.



2.2.5 SUBWF

Subtrahiert den Wert im File – Register vom Wert im W – Register und speichert in Abhängigkeit des Destinationbits. Bei einer 1 im File-Register, bei 0 im W – Register.

2.2.6 DECFSZ

Der Wert im File – Register wird einmal dekrementiert. Wenn das Destinationbit 0 ist so wird das Ergebnis ins W – Register geschrieben, wenn es 1 ist, dann wird das Ergebnis zurück ins File – Register geschrieben. Ist das Ergebnis nicht 0 so wird der nächste Befehl ausgeführt, ansonsten wird der NOP – Befehl ausgeführt und der Programmzähler um 1 erhöht + weil DECFSZ – Befehl (auch + 1). Insgesamt dann Programmzähler + 2;

2.2.7 XORLW

Der Wert im W – Register wird mit einer 8 – Bit Literalen XOR verknüpft und ins W – Register gespeichert.

2.3 Flags

Sowohl das Zeroflag, Carryflag und Digitcarryflag wurden als integer Variable realisiert.
Das Zeroflag nimmt den Wert 1 an und wird somit gesetzt, wenn das Ergebnis einer Operation 0 ist, ansonsten wird es nicht gesetzt.

Das Carryflag bekommt zuerst mitgeteilt, ob es sich bei der Operation um eine Subtraktion oder Addition handelt. Anschließend wird geprüft, ob vom ersten Wert eine 0 addiert oder subtrahiert wird. Die beiden Faktoren entscheiden unteranderem, ob das Carryflag gesetzt wird. Der entscheidendste Faktor ist der Überlauf, bei einer Addition (Ergebnis > 255) bzw. kein Überlauf bei einer Subtraktion (Ergebnis > 0). In beiden Fällen wird das Carryflag gesetzt.

Das Digitcarryflag ähnelt dem Carryflag, da beide bei Überläufen gesetzt werden. Das Digitcarryflag wird nämlich dann gesetzt, wenn das 5te – Bit einen Übertrag erzeugt.

Alle Flags befinden sich in der Klasse Statusregister.

2.4 Interrupts

Damit ein Interrupt auftreten kann muss das „Global - Interrupt - Enable - Bit“ gesetzt sein.
Jede Interruptquelle hat ein eigenes Interrupt - Enable – Bit, sowie ein Interuptflag. Einzelne quellen können so aktiviert oder deaktiviert werden. Die Flag wird unabhängig vom Global - Interrupt - Enable – Bit gesetzt. …….. . Folgende Abbildung zeigt die Interruptlogik.

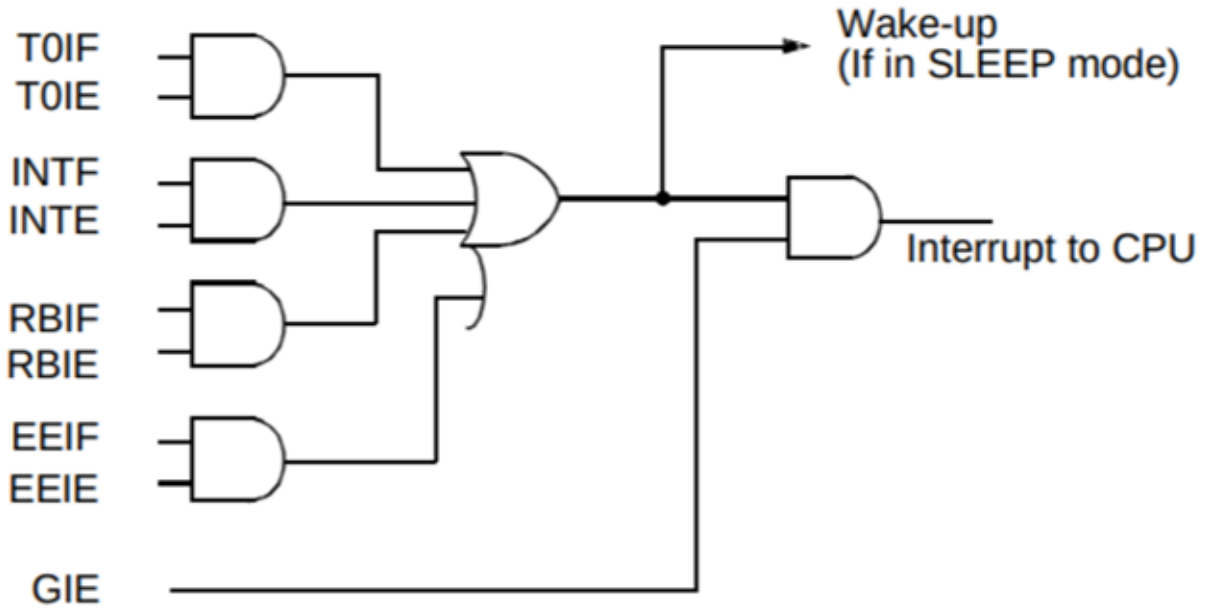


Abbildung 3: Interruptlogik (Quelle: Datenblatt PIC16F8x_20210412)

2.5 TRIS- Register

Das TRIS-Register wurde so realisiert, dass die RB4 – 7 Bits ein Interrupt auslösen können, sobald p- und n-Transistor auf Eingang gestellt sind.

3. Zusammenfassung

3.1 Inwieweit war die Realisierung möglich?

Theoretisch wäre es möglich, anhand dessen was Java bietet, jedes Bauelement in eine extra Klasse auszulagern. So hätte das große „Problem“ in viele kleine „Teilprobleme“ zerlegt werden können. So würde auch eine gewisse Realitätsnähe herangeführt werden. Problematisch wurde es bei dem RA4 – Bit. Dies war nicht zu realisieren, denn das Bit würde theoretisch nur mit einem Pull – Up Widerstand aktiv werden. Die Realisierung erfolgt nun mit manueller Betätigung des 4ten Portpins im PortA Bereich.

3.2 Fazit

Anfangs waren wir etwas überfordert mit der ganzen Sache. Für uns beide war es schließlich das erste richtige „große“ Projekt gewesen. Es war viel Arbeit sich in die ganze Geschichte reinzulesen und es auch zu verstehen. Wir starteten mit einer groben Gliederung was zu tun war und teilten dann die Arbeiten auf. Wir begannen mit der GUI und machten uns gleichzeitig an die ersten Komponenten. Was sich als suboptimal herausstellte, denn, während die ersten Komponenten fertig zu schienen waren, konnten wir sie nicht testen, da die GUI noch nicht fertig war. Das Ziel zu dem Zeitpunkt lag dann bei der Fertigstellung der GUI. Als die GUI fertig war, ging es so richtig los mit der Programmierung des PIC. Nach und nach stieg man hinter den Microcontroller und verstand mehr und mehr. Die ersten Fortschritte waren ein kleiner Meilenstein für uns.

Das große Problem war, dass wir zu spät angefangen haben die einzelnen Komponenten zu testen, sodass wir Fehler entdeckten, die wir nicht direkt lösen konnten. Dies warf uns ein gutes Stück zurück. So war die erste Hälfte des Semesters von keinen großen Erfolgen gekürt (bis auf die GUI).

Die größte Aufgabe war es, alle Komponenten zusammenzufügen und die Testprogramme zum Laufen zu bringen, was etwas herausfordernd war, weil wir die gesamten Methoden nicht getestet hatten. Es gab Fehler, die einige Stunden in Anspruch genommen haben, sie zu korrigieren. Des Weiteren fanden wir heraus, dass manche Komponenten nicht so funktionieren wie sie implementiert wurden. Also mussten die fehlerhaften Komponenten teilweise komplett umgeschrieben werden.

Zusammenfassend kann man sagen, dass die Planung besser hätte, strukturiert und durchdacht werden müssen. Die anfangs erwähnte grobe Planung war viel zu grob, weshalb wir mehrere Rückschläge erfahren mussten. Das Zeitmanagement war trotz einigen Wochenendstunden nicht ausgereift. Wir hängten uns an Kleinigkeiten zu sehr auf, was unteranderem dem schlechte Zeitmanagement und den unausgereiften Vorüberlegungen geschuldet war.

Um nicht nur negativ zu reflektieren: Wir hatten eine sehr gute Absprache bezüglich des Programmierens. Denn es kam vielleicht einmal vor, dass sich beim zusammenführen beider Projektteile die Codes überschnitten. So haben wir im positiven Sinne aneinander vorbei programmiert. Auch die Programmierart überschnitt sich oftmals, sodass nicht lange überlegt werden musste was der jeweilig andere gemacht hatte.

Bei einer nochmaligen Realisierung eines solchen Projekts, würden wir definitiv mehr Zeit für die Planung aufbringen, sowie allgemein strukturierter an das Ganze herangehen.