Dokumentation PIC16F84 Simulator

Version: 0.1

Autor: Noah Disch

Autor: Eduard Wayz

2 Einleitung

In dem vorliegenden Dokument wird die Entwicklung und Implementierung eines Simulators in Java für den Mikrocontroller PIC16F84 dargelegt. Der Fokus liegt hierbei auf den wesentlichen Funktionen und Eigenschaften des Simulators sowie den gewonnenen Erkenntnissen aus der Programmierung. Zudem werden die Herausforderungen aufgezeigt, denen wir bei der Entwicklung des Simulators begegnet sind.

3 Allgemeines

3.1 Funktionsweise eines Simulators

3.2 Vor- und Nachteile der Simulation

3.3 Beschreibung der Programmoberfläche

Bild einfügen von ganzem View mit den verschiedenen Bereichen gegliedert. (a bis ?)

Die komplette Programmoberfläche lässt sich in sieben Abschnitte aufteilen, diese werden im folgendem Abschnitt als a - f bezeichnet.

a) Der Bereich “a” zeigt die LST Datei in einer tabellarischen Ansicht (TableView) an. Oberhalb der Tabelle befinden sich zwei Buttons: “File..” und “Dokumentation”. Mit “File..” lässt sich das Dateiauswahl-Menü des eigenen Betriebssystems (das Bild in Abbildung x zeigt es unter macOS) und der zweite Button öffnet die Dokumentation zu diesem Projekt (diese Datei).
   Die Tabelle enthält sieben Spalten. Sechs dieser Spalten ("PC" bis “Comments”) sind unmittelbar aus der LST-Datei entnommen, während die Spalte “Breakpoint” ergänzt wurde, um dem Nutzer die Möglichkeit zu geben, Breakpoints zu setzen. (siehe Abbildung x). Bei jedem Schritt wird der nächste auszuführende Befehl in hellblau markiert.
   Unterhalb der Tabelle wird die Laufzeit in Mikrosekunden sowie die simulierte Quarz Frequenz in Megahertz angegeben.

b) Im Bereich “b” sind die primären Steuerelemente der Simulation untergebracht. Oberhalb der vier Knöpfe befinden sich Schieberegler, mit welchen sich die Schrittgeschwindigkeit im “Run”-Modus und die Quarzfrequenz einstellen lassen. Der beim Geschwindigkeitsregler eingestellte Wert gibt die Anzahl der in der Sekunde durchgeführten Schritte an.
   Der oben bereits erwähnte „RUN“-Button startet die kontinuierliche Ausführung der Steps des PIC, die in einstellbaren Intervallen erfolgen. Der gesamte View wird bei jedem Schritt neu geladen, um die Werte aktuell zu halten, wobei das Vorhandensein von Breakpoints hier überprüft wird. Wird ein Breakpoint erreicht, halten die Steps an, bis erneut auf den “RUN”-Button gedrückt wird.
   Der „STEP“-Button führt einen einzigen Schritt aus und aktualisiert daraufhin den gesamten View. Breakpoints werden hier nicht berücksichtigt und daher einfach übersprungen.
   Der Button “STOP” bewirkt das sofortige Beenden der “Run-Funktion”.
   Der Knopf “MCLR” steht für “Master-Clear” und sorgt für…

Ganz rechts befindet sich die Checkbox für den Watchdog, welcher hiermit aktiviert oder deaktiviert werden kann.

c) Im nächsten Bereich, der Bereich “c”, werden die Ports A und B des PICs mithilfe eines GridViews visualisiert. Die Farben zeigen an, ob der Port im Moment ein “Eingang” (“Input”) oder ein “Ausgang” (“Output”) ist, was durch das TRIS-Register definiert wird. Ein Ausgang lässt sich nicht manuell verändern, während die Eingänge durch einen Mausklick umschaltbar (0 -> 1 oder 1 -> 0) sind. Wobei im nächsten Schritt dann der Wert in den jeweiligen Port übernommen wird. Die TRIS-Register lassen sich nicht manuell einstellen.
d) Direkt unterhalb der Port-Darstellung befindet sich die Visualisierung des Stacks, Bereich “d”, welcher (gemäß der typischen Funktionsweise eines Stacks) von unten nach oben gefüllt wird. Der Stackpointer, welcher immer den aktuell “höchsten” Index angibt, wird, analog zur LST-Tabelle, ebenfalls durch hellblaue Markierung visualisiert. Sodass der Nutzer an welcher Stelle der Stack gerade verändert wird.
e) In einem weiteren Teilbereich werden die General Purpose Registers (GPR) und Special Function Registers (SFR) in einer Tab-Ansicht dargestellt,. Dieser ist aber auch wieder ein TableView. Beide Tabs enthalten die Spalten „Address", „Value (Hex)“ und „Value (Binary)“. Zusätzlich enthält der SFR-Tab die Spalte „Name“, welche als Alias für die Adressen dien
f) Zum Schluss werden im Bereich “f” noch einmal die wichtigsten Werte während des Programms angezeigt. Es ist zwar eine Wiederholung, von dem, was auch in den anderen Bereichen sichtbar ist, aber dennoch sehr hilfreich, da das die Werte sind, die am meisten während des Programms verändert werden.

Alle dargestellten Bereiche sind miteinander verknüpft und ermöglichen so eine kontinuierliche Aktualisierung der Benutzeroberfläche während der Simulation.

4 Realisierung

4.1 Grundkonzept des Simulators

4.2 Struktur und Gliederung

4.3 Programmstruktur und Ablaufdiagramme

4.4 Beschreibung der verwendeten Variablen

4.5 (Auswahl und Begründung der Programmiersprache)

4.6 Detailbeschreibung ausgewählter Funktionen BTFSx, CALL, MOVF, RRF, SUBWF, DECFSZ, XORLW

4.7 Flags und deren Implementierung

4.8 Interrupts (Auszu aus dem Listing)

Beschreibung und Diagramme

4.9 TRIS-Register und Latchfunktion

4.10 Hardwareansteuerung

4.11 State-Machine (EEPROM)

5 Zusammenfassung und persönliches Fazit

5.1 Praktische Nachbildung

5.2 Probleme und deren Lösungen

Eine wesentliche Herausforderung bei der Erstellung der grafischen Benutzeroberfläche (GUI) war die notwendige kontinuierliche Aktualisierung der Oberfläche nach jedem einzelnen Schritt. Um den gewünschten Ablauf zu gewährleisten, war es erforderlich, eine öffentlich zugängliche, statische Instanz innerhalb der Klasse zu initialisieren, welche auf sich selbst referenziert (z. B. im PortController). Diese Instanz musste einmalig initialisiert werden, um anschließend im ButtonsController die Methode `buildUI()` des PortControllers aufzurufen. Ohne diese spezifische Struktur wäre eine fortlaufende Aktualisierung des Views nach jedem Schritt nicht möglich gewesen. Die Identifikation und Implementierung dieser Logik war daher für alle betroffenen Controller zwingend erforderlich, was jedoch ein tiefgehendes Wissen in der Strukturierung und im Zusammenspiel von JavaFX-Komponenten erforderte.

Ene weitere signifikante Herausforderung stellte das korrekte Größenmanagement der GUI-Elemente dar. Wie im nachfolgenden Kapitel detailliert beschrieben wird, entschieden wir uns bewusst gegen die Verwendung eines UI-Builders und gestalteten das Projekt eigenständig. Dies erforderte eine gut abgestimmte Kombination von Eigenschaften innerhalb der `.fxml`-Dateien, der verknüpften CSS-Dateien sowie der Controller-Klassen. Dabei war es insbesondere herausfordernd, die geeigneten Stellen für bestimmte Attribute zu identifizieren. Oftmals funktionierten Attribute wie `vgrow="ALWAYS"` oder `fillHeight="true"` zunächst nicht wie erwartet, da sich später herausstellte, dass sie an insgesamt drei verschiedenen Stellen definiert werden mussten. Die systematische Ermittlung dieser Zusammenhänge erwies sich als zeitaufwendig und erforderte mehrere Stunden intensiver Fehlersuche und Experimente. Zusätzlich bestand die Anforderung von uns, ein responsives Layout umzusetzen, um sicherzustellen, dass die Benutzeroberfläche auf unterschiedlichen Bildschirmauflösungen und variablen Fenstergrößen richtig dargestellt wird. Dies erforderte den gezielten Einsatz von JavaFX-Komponenten wie z. B. GridPane, wobei wir Layout-Attribute wie GridPane.rowSpan="4" und percentWidth="10" nutzen mussten, um eine flexible und anpassungsfähige Darstellung der GUI zu erreichen