Simulation des PIC16F84

Sascha Moser und Jan Ulses

23. Juni 2014

Inhaltsverzeichnis

Kapitel 1

Vorwort

In der vorliegenden Dokumentation wird die Umsetzung des Projekts der Simulation eines Microcontrollers des Typs PIC16F84. Dieses Projekt wurde nur während dem Modul Rechnertechnik II bearbeitet. Dies umfasst sowohl die Planung des Projekt als auch die Durchführung. Es wurden Kenntnisse aus den Modulen Digitaltechnik, Software Engineering so wie Rechnertechnik I angewandt. Mit Hilfe dieses Projekts, sollen wir unsere Kenntnisse vom Aufbau und der Funktionsweise eines Microcontrollers erlernt und vertieft werden. Dieses Projekt wurde in Zweierteams bearbeitet. Die Dokumentation beschreibt den grundsätzlichen Aufbau der Simulatuion. Sie enthält außerdem Informationen zur Umsetzung von Assemblercode sowie Erläuterungen, wie die Benutzeroberfläche aufgebaut ist. Zusätzlich befindet sich am Ende des Dokuments ein Resümee über das Projekt.

Kapitel 2

Einleitung

Das Projekt der Pic-Simulation entstand im Rahmen des Studienfachs Rechnetechnik II an der Dualen Hochschule Karlsruhe im Studiengang Informationstechnik.

Ziel des Projektes ist es, am ende der Vorlesungszeit einen vollfunktionsfähigen Simulator geschrieben zu haben.

Die Programmiersprache war freiwählbar und wir einigten uns auf Java. Die Dauer des Projekts wurde auf eine Semsterlänge angekündigt.

Kapitel 3

Was ist eine Simulation?

Im Folgenden wird erläutert, was eine Simulation ist und was deren Vorteile bzw. was deren Nachteile sind.

Mit einer Simulation wird ein System dargestellt, dessen Analyse sehr komplex und unter Umständen schwer durchzuführen ist. Eine simulation stellt immer eine Abstraktion der Realität dar, und beschränkt sich deshalb auf das nötigste. Man führt an diesen Modellen Experimente durch,um dann Rückschlüsse auf das Verhalten des orginal Modells ziehen zu können. Bei der Implementation der Simulation muss beachtet werden, dass wesentliche Aspekte nicht unzureichend intgriert sind oder gar vergessen werden.

Im vorliegenden Fall soll eine Simulation für die Hardware des PIC16F84-Controllers geschrieben werden. Die Hardware muss nicht vorliegen. Die Simulation soll trotzdem so ablaufen als wäre die Hardware vorhanden. Somit muss die Simulation alle wichtigen Funktionalitäten der Hardware besitzen.

Eine Simulation besitzt mehrere Vor- und Nachteile:
Vorteile

- Gut geeingt bei Einsätzen für Testzwecke
- Senkt die Kosten für Tests von Programmen, da aufwändiges Beschreiben der Bausteine entfällt und ein Vorhanden sein der Hardware nicht nötig ist.
- Einfachere Fehlersuche in neuen Programmen, da man die Möglichkeit besitzt Arbeitsschritte, anhand der grafischen Nutzerobefläche, nachzufollziehen.
- Man hat keine probleme mit fehlerhafter Hardware, da alles softwareseitig gelöst ist. Dies spart Zeit bei der Fehlersuche, denn man weiß dass ein Hardwaredefekt auszuschließen ist.

Nachteile

- Fehler bei der Implementierung der Simulation wirken sich auf das Ergebniss des Tests aus und können so das Ergebnis beeinträchtigen.
- Je komplexer die Hardware ist, desto mehr Aufwand muss betrieben werden um die Simulation fehlerfrei zu gestallten. Dadurch steigt der Zeitaufwand und ebenso die Kosten für seine Simulation.
- Um Fehler zu vermeiden und die Richtigkeit der Simulation zu gewährleisten müssen aufwendige Tests implementiert werden.

Kapitel 4

Das Simulationsprogramm

Im nachfolgenden Kapitel wird die Obefläche des eigenen Simulationsprogrammes näher beschrieben. Dazu gehört eine Gesammtübersicht der GUI, sowie Detailbeschreibungen für die Menüleiste und Buttons.

4.1 Übersicht

Die Abbildung 4.1 zeigt eine Übersicht der Simulation. Im oberen Bereich befindet sich die Menüleiste auf die wir später noch genauer eingehen. Die Tabelle auf der linken Seite des Bildes repräsentiert die Registerübersicht. Diese aktualisiert nur die Felder, welche sich während eines Schritts verändert haben könnten. Im

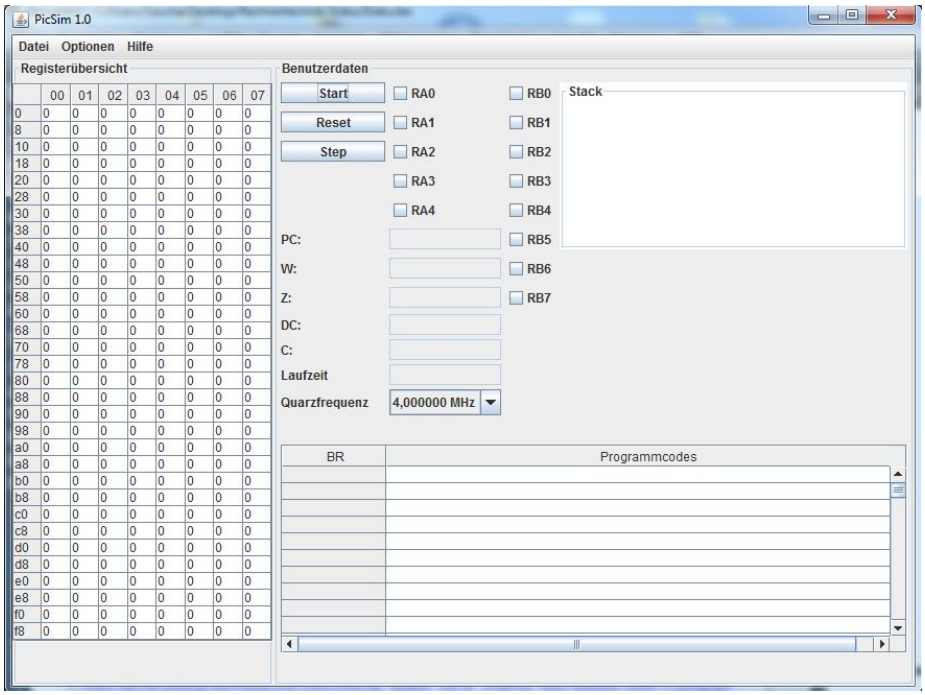

Abbildung 4.1: Übersicht der Simulation

unteren Bereich befindet ebenfalls eine Tabelle. Sie dient dazu den Assembler darzustellen. Ebenso sollte bei ihr die Möglichkeit bestehen Breakpoints zu setzen. Darüber befinden sich verschiedene Textfelder die mit den jeweiligen Labels versehen sind. Daneben befinden sich mehrere JCheckboxen, die die Ports des Microcontroller darstellen sollen. Ebenso befinden sich in diesem Bereich mehrere Buttons. Diese werden im weiteren Verlauf der Dokumentation genauer erläutert.1 Auf der rechten Seite befindet sich ein größeres Textfeld. Dieses Textfeld zeigt Ereignisse an, wenn sich der Stack des Controllers ändert. Dort werden die Aussprungadressen angezeigt.

4.2 Menüleiste

Die Menüleste befindet sich wie in Abbildung 4.2 zu sehen im oberen Bereich der Anwendung. Man hat die Möglichkeit auf die Schaltflächen Datei, Optionen und Hilfe zu klicken. Unter der Schaltfläche Datei hat man die Möglichkeit eine Datei zu öffnen oder das Programm zu schließen. Auf dem Bild 4.3 ist diese Möglichkeit zu sehen. Unter der Schaltfläche Hilfe verbergen sich die Möglichkeit

Abbildung 4.2: Übersicht der Menüleiste

diese PDF zu öffnen oder man erhält eine kurze Info über die Programmierer der Simulation.

Abbildung 4.3: Ansicht zum Schließen der Simulation

4.3 Buttons

In diesem Abschnitt wird auf die Funktionalität der Buttons näher eingegangen. In Abbildung 4.4 sind die drei Buttons, aus der GUI, mit den Namen Start/-Stop, Step und Reset zu sehen. Der Start/Stop- Button ändert je nach Zustand seinen Namen. Die Buttons Start/Stop und Step sind außer Funktion solange kein LST-File geladen ist. Ist ein File geladen, werden beim anklicken des Startbuttons alle anderen Buttons gesperrt. Der Button erhält die Bezeicnung Stop. Die Simulation beginnt nun und läuft solange man nicht auf erneut auf den Stop-Button klickt. Der Step-Button führt nur einen Schritt pro Klick aus. Hier werden keine Buttons gesperrt. Der Reset-Button stetzt die Software, auf die in der Controllerspezifikation, festgelegten Werte zurück.

Abbildung 4.4: Übersich der Buttons

Kapitel 5

Datenverarbeitung

5.1 Event Listener

Damit Veränderungen in der grafischen Nutzeroberfläche oder in den Registern bemerkt werden und dadurch das entsprechende Informationsfeld aktualisiert wird, wurden Event Listener benutzt.

5.1.1 Was sind Event Listener?

Um nicht dauerhaft die GUI sowie unsere Daten auf Änderungen abzufragen, haben wir uns für die Lauschmethode"der Eventlistener entschieden. Es gibt Ereignissauslöser und Interessenten der Ereignisse. Diese Interessenten melden sich bei einem Auslöser an und ab. Die Listener enthalten alle Methoden, welche ausgeführt werden sollen sobald sich eine Änderung ergibt.
Ein Sonderfall der Eventlistener ist der Actionlistener der Buttons. Er stellt fest, welcher Button gedrückt wurde und führt anhand des Ergebnisses die entsprechenden Anweisungen aus.

5.1.2 Beispiel Quellcode der Simulation

Im folgenden wird anhand eines Beispiels beschrieben wie wir die Eventlistener eingessetzt haben.

Listing 5.1: Auszug GUI-Klasse

```
@Override
public void update(UpdateGUIInfoField event) {
  int value = event.getValue();
  switch (event.getField()) {
    case PC:
      pc.setText(String.valueOf(Integer.toHexString(value)));
      break;
    case W:
      wreg.setText(String.valueOf(Integer.toHexString(value)));
      break;
    case STATUS:
```

```
        carry.setText(String.valueOf(menuBar.register.testBit(value, 0)));
        dc.setText(String.valueOf(menuBar.register.testBit(value, 1)));
        zerobit.setText(String.valueOf(menuBar.register.testBit(value, 2)));
        break;
    }
}
```

Listing 5.2: Auszug Eventklasse

```
public class UpdateGUIInfoField extends EventObject {
  private final int field;
  private final int value;

  public UpdateGUIInfoField( Object source, int field, int value ){
    super( source );
    this.field = field;
    this.value = value;
  }

  public int getField() {
    return field;
  }

  public int getValue() {
    return value;
  }
}
```

Listing 5.3: Auszug Eventinterface

```
public interface GUIListener extends EventListener{
    void update ( UpdateGUIInfoField event);
}
```

5.2 Einlesen des LST-Files

Das LST-File wird insgesammt zweimal eingelesen. In der Funktion reader() der Klasse Scan.java wird ein FileReader initialisiert. Danach wird ein BufferdReader erstellt, welcher die eingelesenen Daten puffert bzw. zwischenspeichert.

Listing 5.4: LST-File einlesen

```
FileReader fr = new FileReader(String.valueOf(pathToLSTFile));
BufferedReader br = new BufferedReader(fr);
```

Die Daten werden in einen String eingelesen und dieser wird sofort auf die wichtigen Informationen beschnitten und überprüft, ob es sich dabei tatsächlich um Maschinencode im Hexformat oder eine leer Zeile handelt. Ist es Maschinencode so wird dieser direkt in eine ArrayList gespeichert.

Listing 5.5: LST-File einlesen-Abfrage ob Maschinencode

```
if (!(zeile.equals("␣␣␣␣") || !isHexValue(zeile))) {
```

```
                hexCode.add(zeile);
        }
```

Diese Arraylist enthält am Ende den kompletten Maschinencode der innerhalb eines LST-Files zu finden ist. Mit diesen Daten simulieren wir das eigentliche Programm. Anhand des Programmablaufs aus Bild bla bla blubb ist der genaue Ablauf zu erkennen. Abbildung 5.1 zeigt den Ablauf zum Einlesen des Dokuments.

5.3 Öffnen der Hilfe PDF

Eine Funktion der grafischen Oberfläche sollte sein, dass über eine Schaltfläche dieses Dokument aufgerufen werden kann. Diese Funktion findet sich in der Simulation im Chapter Hilfe der Menübar. Durch unsere spezielle Implemation, welche unter Abbildung bla bla blub zu sehen ist, ist es nur Windows-Usern möglich die Funktion zu nutzen. Sie ruft das Standartausgabeprogramm für PDFs auf und öffnet das angegebene Dokument.

Listing 5.6: PDF-Öffnen

```
 if (object.getSource() == doku) {
    System.out.println("doku wurde angeklickt");
    if (Desktop.isDesktopSupported()) {
    try { Desktop.getDesktop().open(new File("./Dokumente/
   pic16f84A.pdf")); }
    catch (IOException e1) { e1.printStackTrace(); }
    }
}
```

Die ausschlaggebende Zeile für eine reine Windowsverwendung ist Zeile 3. Es wird überprüft ob die Funktion PDF-Öffnungsfunktion ausführbar ist.

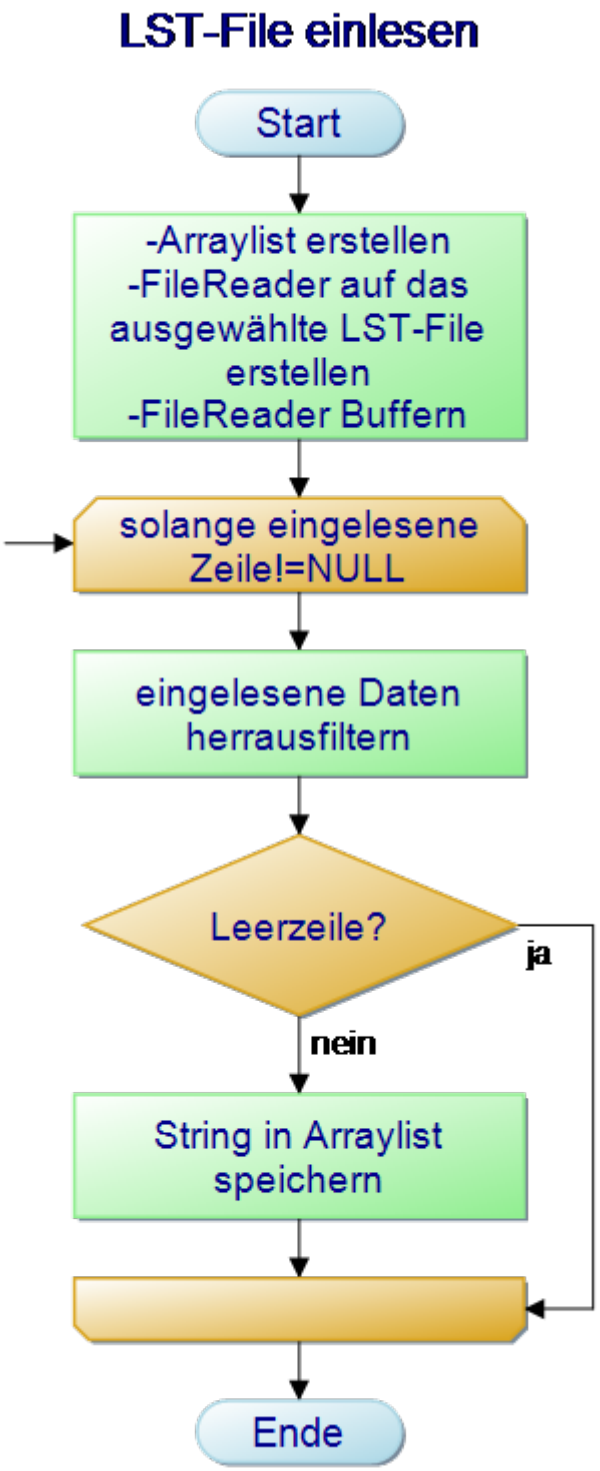

Abbildung 5.1: Ablaufdiagramm für das Einlesen von LST-Files

Kapitel 6

Befehlscode

In diesem Kapitel wird anhand von zwei Befehlen erklärt wie wir die Befehle umgesetzt haben.

6.1 Umsetzung des Befehls MOVWF

Dieser Befehl ist einer der wichtigsten Befehle für den Mircocontroller.Dieser Befehl schreibt Daten vom Arbeitsregister W in ein ausgewähltes Register 'f'. Im folgenden Listing erkennt man unseren Code, wie der Befehl umgesetzt wurde. Dieser Befehl, wurde wie alle anderen auch, als Funktion implementiert. Die Funktion erhält, für die Berechnung, alle notwendigen Parameter. Dies ist im folgenden Listing zu sehen. Für ein besseres Verständin dient der Programmablaufplan 6.1.

Listing 6.1: Befehl MOVWF

```
public void movwf (int instruction) throws NoRegisterAddressException {
int address = instruction & 0x007F;
int value = register.getW();
register.setRegValue(address, value);
register.incPC();
register.incCycles();
    }
```

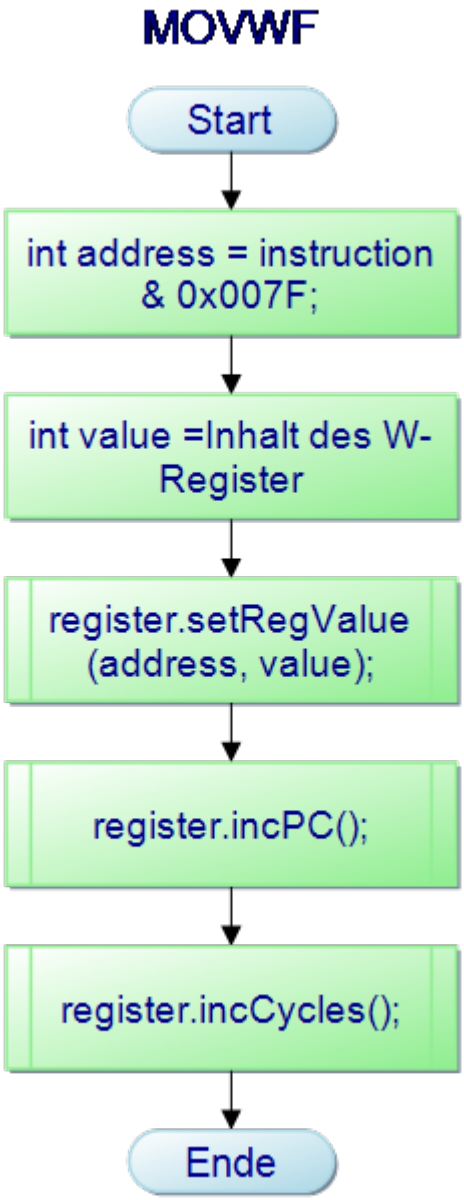

Abbildung 6.1: Programmablaufplan des Befehls MOVWF

6.2 Umsetzung des Befehls RRF

Dieser Befehl rotiert ein Bit nach rechts. Dabei wird das Carrybit mitbenutzt. Fällt ein Bit nach rechts aus der Darstellung wandert dieses Bit in den Carry. Das Bit aus dem Carry wandert an die höchste Stelle des Registers. Mit dem Bit 'd' des fertigen Befehls entscheidet sich wo das Ergebnis abgespeichert wird. Ist d=0 wird das Ergebnis in das Arbeitsregister W geschrieben, falls d=1 wird das Ergebnis in das festgelegte Register gespeichert. Diese Abfrage erkennt man im folgenden Listing.

Listing 6.2: Befehl MOVWF

```
public void rrf(int instruction) throws NoRegisterAddressException {
instruction = instruction & 0x00FF;
int address = instruction & 0x007F;
int f = register.getRegValue(address);
// aktuelles Carrybit speichern
int c = (register.getRegValue(STATUS)) & 0x01;
// neues Carrybit setzen
register.setRegValue(STATUS, (f >>> 0) & 0x0001);
// nach rechts shiften und c um 7 stellen nach links geshiftet addieren
int value = (f >> 1) + (c << 7);
//setRegValue to destination
if(instruction < 0x007F) {
register.setW(value);
        } else {register.setRegValue(address, value);}
register.incPC();
register.incCycles();
}
```

Der Programmablaufplan dieser Funktion ist in der folgenden Abbildung ersichtlich.

Kapitel 7

Zusammenfassung

Abschließend gilt zusagen, dass das Projekt ansich eine gute Idee ist. Jedoch nimmt es im Verhältnis zu anderen Porjekten viel Zeit in Anspruch. Wir konnten unsere Kenntnisse im Bereich der Java-Programmierung vertiefen und erweitern. Dies trifft insbesondere auf das Entwickeln einer grafischen Oberfläche oder die von uns verwendeten EventListener zu.

Ebenso konnten wir, Werkzeuge aus dem Fach Software Engineering verwenden. Hierzu zählen die Mergeprogramme die das zusammenarbeiten vereinfacht haben. Zusätzlich nutzen wir noch eine Versionskontrolle, die vorallem für das Entickeln der Gui wichtig wurde.

Aus dem Fach Rechnertechnik kannten wir schon einige Befehle und Funktionsweißen des Microcontrollers. Diese Kenntnisse konnten ebenfalls vertieft und erweitert werden.