Simulator für den Microcontroller PIC16F84

Dokumentation

von
Dennis Alles und Steffen Kurstak

TINF11B3

Studiengang Informationstechnik
an der Dualen Hochschule Karlsruhe

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Begriffserklärung Simulation und Simulator

Eine Simulation dient der Analyse, um das Verhaltens eines dynamischen Systems unter bestimmten Vorgaben und Umständen zu testen. Ein Simulator stellt ein komplettes System aus der Realität dar. Der Simulator wird oft für Testzwecke, Schulung oder Entwicklung verwendet oder wenn ein reales System sich nicht direkt beobachten lässt. Aus den Experimentierergebnissen des Simulators kann man Erkenntnisse über das reale System gewinnen. Im Idealfall sollte sich der Simulator wie das reale System verhalten. Je näher die Kopie an das Original heranreicht, desto aussagekräftiger sind die dabei entstehenden Ergebnisse.

1.2 Vorteile einer Simulation

Experimente auf einem Simulator durchzuführen ist kostengünstiger und mit weniger Aufwand verbunden als auf einem realen System. Bei einem Simulator können keine Hardwareschäden durch falsche Eingaben wie bei einem Microcontroller entstehen.
Bei einem Simulator kann jederzeit verschiedene Rahmenbedingungen und Einstellungen leicht geändert werden. Die einzelnen Vorgänge bzw. Schritte können gut beobachtet werden. Das Programm kann durch den Benutzer gestoppt und der Schritt nochmals im Detail angeschaut werden. Fehlerfälle können so rekonstruiert und beseitigt werden. Für Anfänger kann ein Simulator zum Verstehen eines Microcontrollers beitragen.

1.3 Nachteile einer Simulation

Oft ist der Simulator eine Vereinfachung des originalen Models und spiegelt die Messergebnisse nur ungenau wider. So ist es auch möglich, dass der Simulator fehlerhaften Code ausführen kann und so später die Orginalware durch den Code zerstört wird. Ein Modell liefert nur in einem bestimmten Kontext Ergebnisse, die sich auf die Realität übertragen lassen. In anderen Parameterbereichen können die Resultate schlichtweg falsch sein. Daher ist die Verifikation dieser Ergebnisse sehr

wichtig. Wenn nur wenige kostengünstige Bauteile benötigt werden, lohnt sich die Simulation oft nicht. So kann die Simulation doch teurer sein, als der wirkliche Nutzen hiervon.

1.4 Programmoberfläche und Bedinung(siehe Dokumentation.pdf)

Bei Starten des Programms öffnet sich die Benutzeroberfläche. Der Benutzer kann hier die Funktionen des PIC16F nachvollziehen.

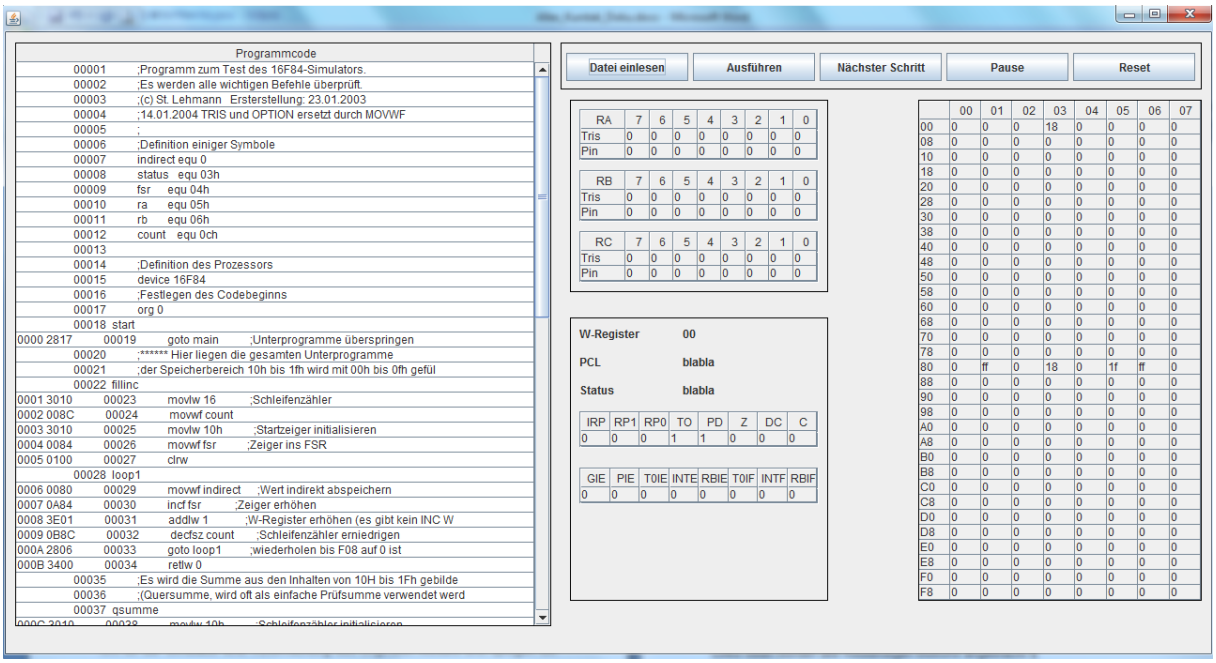

Abbildung 1: komplette Programmoberfläche

Die Programmoberfläche wurde in 5 verschiedene Bereiche untergliedert.

Rechts oben befinden sich insgesamt fünf Buttons, die das Programm steuern:

Datei einlesen: Es erscheint ein Dialogfenster mit der Aufforderung eine Datei auszuwählen. Diese Datei wird nun in die auf der linken Seite angebrachte Feld „Programmcode“ geladen.

Ausführen: Alle Befehle werden nacheinander ausgeführt, bis er an das Ende der Datei gelangt, der Button „Pause“ getätigt wird oder an einem Breakpoint angekommen ist.

Nächster Schritt: Es wird der nächste Befehl ausgeführt.

Pause: Der aktuelle Run-Vorgang lässt sich stoppen und mit einem erneuten „Ausführen“ wieder von aktueller Stelle fortfahren.

Reset: Setzt alle Ports, Stack, Bank0 und Bank1 auf ihre Start-Werte zurück. Und bringt das geladene Programm wieder zurück zum Anfang.

Auf der linken Seite befindet ein Textfeld, in dem der Programmcode, der eingelesen wurde, angezeigt wird.
In der Mitte – unterhalb der Buttons – sind die Ports zu sehen. Die einzelnen Pins der Ports können entweder durch einen Klick auf 1 gesetzt oder genullt werden.
Unter den Ports sind die Spezialfunktionsregister angebracht. Hier sind unter anderem die Werte des W-Registers, des PC oder des FSR zu sehen.

Im fünften Bereich - rechts - ist eine Tabelle zu sehen, welche die Werte aller Register zeigt.

Außerdem gibt es noch eine Menüleiste, in der die Hilfe aufgerufen werden kann. Dann wird diese Dokumentation geöffnet.

Die Checkbox am
Anfang jeder Zeile zeigt an ob diese ein Breakpoint ist oder nicht. Bei einem Klick auf die entsprechende Zeile wird diese Checkbox verändert. Die gelbe Zeile ist die aktuelle Zeile.
Auf der rechten Seite wird der Inhalt des Speichers angezeigt. Außerdem kann über die Schaltflächen der Ablauf des Programms eingestellt sowie die Eingänge belegt werden.

Der Inhalt des W Registers wird Dezimal und Hexadezimal angezeigt.
Die Checkboxes stellen die zugehörigen Register da, wobei das Status Register nicht verändert werden kann.
Die Register RA und RB können nur verändert werden, wenn sie auf Eingabe stehen. Dies wird durch ein i unterhalb der Checkbox dargestellt. Sind die Bits auf Ausgabe so ersetzt ein o die Stelle des i.

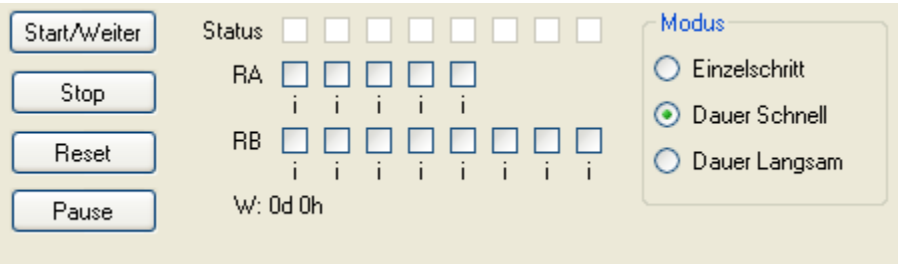

Abb.5 Schaltflächen

Eine Matrix und eine ListBox (Abb.6) zeigen den Inhalt des Speichers und den Stack an. Die Matrix zeigt die Speicherstellen in Hexadezimal an. Die Speicherbank wird über einen Reiter ausgewählt. Die Listbox zeigt die Rücksprungadressen an, wobei die oberste Adresse immer die nächste Rücksprungadresse ist.

In diesem Kapitel werden ein paar Hinweise zur Anordnung und Handhabung der Programmoberfläche
gegeben.

4.1 Anordnung der Programmoberfläche

. Der Stack ist als Liste unten dargestellt. welcher nach
unten wächst, wenn darauf gepushed wird und dementsprechend nach oben, wenn er gepopt wird. Die unterste Stelle ist immer der aktuelle Programmzähler. Die beiden Ports sind auf der unteren rechte dargestellt, wenn sie ausgegraut sind, dann ist der Port ein Ausgang. Mittels des Anklicken der Checkboxes kann man den Zustand ändern, sofern

es sich um eine Eingang haltet. Auf der unteren linken Seite werden noch die Interrupt Flags, der Carry, der Digitcarry, der Zero, die Anzahl der vergangen CPU-Zyklen, das WRegister
und der Prescaler angezeigt. Die Watchdog-Anzeige befindet sich dort ebenfalls, welcher sich dort auch aktivieren lässt.

4.2 Handhabung der Programmoberfläche

Neuer Programmcode für den Simulator lässt sich über „Programm“ -> „Programm laden ...“ einlesen. Wenn ein Programm eingelesen ist, kann man durch anklicken des Knopfes „Next Step“ den nächsten Opcode ausführen und somit den Programmablauf Schritt für Schritt nachvollziehen. Mittels des Knopfes „Run“ läuft das Programm bis es auf einen Breakpoint trifft. Ein Breakpoint lässt sich durch das Anklicken der jeweiligen Checkbox vor der Opcode-Zeile setzen. Die serielle Schnittstelle lässt sich über „Serielle Schnittstelle“ -> „Aktivieren“ aktivieren und im selben Menü über „Einstellungen“ konfigurieren

2 Realisation

2.1 Beschreibung des Grundkonzepts

Das Grundkonzept dieses Programms ist die Nachbildung eines PIC16F84 Mikroprozessors. Es soll möglich sein diesen Mikroprozessor zu simulieren, dass heißt die komplette Funktion des Prozessors so abzubilden, dass Programme die im Hexformat vorliegen, korrekt interpretiert und ausgeführt werden können. Dadurch wird das Testen von Assemblerprogrammen drastisch vereinfacht, es ist somit möglich das Programm ohne Hardware korrekt auszuführen und zu testen.

2.2 Beschreibung der Gliederung

Der Simulator wurde in die Klassen

- Befehle,
- FileChooser,
- FileHandler,
- Main,
- Oberflaeche,
- Programm,
- Register,
- Steuerung,
- Takt

Aufgeteilt.

Die Klasse **Main** erzeugt ein neues Objekt **Steuerung**, die das ganze Programm, also die Kommunikation zwischen Oberflaeche und Ausführung steuert. In der Klasse **Oberflaeche** wird die Programmoberfläche erstellt. Des Weiteren werden hier die Benutzertätigkeiten erfasst und an die Steuerung weitergeleitet. . Wenn eine

neue Quelldatei eingelesen wird, kommen die beiden Klassen **FileChooser** und **FileHandler** zum Einsatz. Sie werden für das Laden eines Programms benötigt. Die Klasse **Befehle** ist für das Abarbeiten der einzelnen Befehle zuständig. Alle Befehle werden von der Steuerung aufgerufen und liefern einen Opcode zurück. Die Klasse **Programm** verarbeitet den gesamten Programmcode. Die Klasse **Register** ist für die Speicherung der Werte und Register zuständig. Für die Run-Funktion und den damit verbunden Taktgeber wird die Klasse **Takt** benutzt.
Das Grundkonzept ist so gewählt worden, dass die interne Abarbeitung in der Klasse **Steuerung** zusammenläuft. Sie ist somit als Schnittstelle zwischen User und Internen Strukturen zu sehen.

2.3 Programmstruktur, Ablaufdiagramme, Variablen usw.

2.4 Welche Programmiersprache wurde gewählt? Warum?

Zur Implementierung des Simulators haben wir uns ziemlich schnell auf die Programmiersprache Java entschieden. Die Entscheidung wurde auf Grund der Vorkenntnisse und Erfahrung getroffen. Außerdem hat Java den Vorteil, dass sich die Entwicklungsarbeit durch ihre objektorientierte Struktur leicht aufteilen lässt. Des Weiteren ist Java plattformunabhängig und erleichtert so die Portierung in andere Systeme. Die einfache Syntax und die Möglichkeit der Thread-Programmierung, Nebenläufigkeit von Prozessen, bestärkte unsere Entscheidung. Die Programmoberfläche kann in Java sehr gut gestaltet und gelöst werden, was auch wiederum ein Pluspunkt darstellt.

2.5 tiefergehende Beschreibung der Funktionen an Hand ausgewählter Beispiele

(BTFSx, CALL, MOVF, RRF, SUBWF, DECFSZ, XORLW). Diese und ggf. weitere Befehle anhand von kurzen Programmsequenzen und Ablaufdiagrammen erläutern.

2.6 Realisierung der Flags und deren Wirkungsmechanismen.

2.7 Wie wurden Interrupts implementiert? (Auszug aus dem Listing)

2.8 Wie wurde die Funktion des TRIS-Registers realisiert? (Latchfunktion?)

2.9 Alle Register müssen manuell veränderbar sein.

2.10 Breakpoints sollten implementiert sein

2.11 Hardwareansteuerung

2.12 Helpfunktion ruft nur die Dokumentation auf.

2.13 Diagramme und Beschreibung der Interruptfunktion.

Diagramm und Beschreibung mittels State-Machine z.B. für EEPROM

3 Fazit

3.1 Wie weit konnten die Funktionen des Bausteins per Software nachgebildet werden?

3.2 Fazit, persönliche Erfahrung und Erkenntnis.

Was passierte während der Entwicklung des Projektes? Welche Probleme tauchten auf und wie wurden Sie gelöst. Vermeiden Sie dabei negative Formulierungen. Was würde ich anders machen, wenn ich das Projekt nochmals realisieren müsste? (Umfang des Fazits ca. 10% der Ausarbeitung)

Zu Beginn standen wir vor viele Fragen und Problemen, wie wir anfangen soll und wie das letztendlich realisisiert werden kann. Diese Fragen und Probleme wurden jedoch mit der Zeit gelöst und dadurch immer weniger. Je weiter wir kamen um so komplexer wurde unser Programm und wir mussten an immer mehr Sachen denken. Beispielsweise bei der Implementierung der einzelnen Befehle musste darauf geachtet werden, dass die richtigen Bits gesetzt oder gelöscht wurden, dass man sich in der richtigen Bank befindet usw. Durch das Beachten bestimmter Kleinigkeiten und das erfolgte Abändern und Vereinfachen im Programmcode wurde unser Programm sehr übersichtlich und gut lesbar gestaltet.
Am Anfang der Vorlesung hatten wir ein Pflichtenheft erstellt mit bestimmten Muss-Zielen, Kann-Zielen und einer Abgrenzung. In der Endversion wird der Quellcode mit einer grafischen Hervorhebung der aktuellen Zeile angezeigt. Dieser Quellcode ist über die Buttons Ausführen oder nächster Schritt ausführbar. Die Run-Funktion kann über den Button Pause wiederum gestoppt werden. Außerdem werden die Register, Flags und Ports in der Programmoberfläche angezeigt und können auch geändert werden. Weitere Muss-Ziele waren zum einen die Interrupt-Funktion und die Dokumentation als Hilfe anzeigen. Diese beiden Punkte wurden auch abgedeckt und somit sind alle Muss-Ziele und somit wichtige Funktionen implementiert, die für den Simulator von Bedeutung sind.
Durch die gemeinsame Entwicklung am Projekt wurde stets auf eine gute Absprache untereinander geachtet. Es wurde stets geklärt, welcher Entwicklungsschritt als nächster angestrebt wird und wie wir gemeinsam vorgehen. Während dieses Projekts lernten wir viele Einzelheiten der Programmiersprache Java und wir merkten diesbezüglich auch schnell kleinere Erfolge. Ebenso wurde das Abarbeiten des Programmcodes im Laufe der Zeit auch immer klarer und der Ablauf komplett verständlich.
Nachteil bei so einem komplexen Projekt liegt in dem Zeitmanagement. Am Anfang lässt sich die Komplexität der Aufgabenstellung nicht abschätzen. Auch im Laufe des Projekts weiß man nie genau, wie viel Zeit noch benötigt wird und welche Probleme noch auftreten und welche Ursachen daraus entstehen können. Das Projekt hatte am Schluss einen sehr hohen Zeitaufwand, der unserer Meinung nach intensiver war als der Lernaufwand für eine Klausur. Die Programmierung des Simulators erforderte

viel Zeit, Geschick, Geduld und Engagement, die Arbeit zufrieden stellend zu lösen und pünktlich abzuliefern.
Alles in allem war es eine lehrreiche Erfahrung sich mit einem Thema im Detail auseinanderzusetzen. Dabei hatten wir wichtige Erkenntnisse und Fertigkeiten erlernt, die man später gut gebrauchen kann.

Button	Lösung zum automatisierten Auslesen
Datei einlesen:	uname -r
Ausführen:	uname -p
Nächste	getconf LONG_BIT

Tabelle 1: Auslesen bestimmter Parameter bei Solaris-Servern