DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Dokumentation PIC Simulator

Abgabedatum	12. Mai 2025
Kurs	TINF23B3
Autoren	Luca Müller
	Leander Gantert

Zusammenfassung

Dieses LaTeX-Dokument kann als Vorlage für einen Praxis- oder Projektbericht, eine Studien- oder Bachelorarbeit dienen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Liste der Algorithmen

Abkürzungsverzeichnis

Kapitel 1

Grundlagen

1.1 Grundsätzliche Arbeitsweise eines Simulators

Ein Simulator ist eine Software, welche das Verhalten eines realen Systems nachbildet. Im Fall eines Mikrocontrollers wie dem PIC wird die Hardwarearchitektur virtuell nachgebildet, um Programme auszuführen. Die grundsätzliche Arbeitsweise eines Simulators im Falle eines PICs umfasst folgende Schritte:

- **Laden des Programms**: Der Simulator lädt das zu testende Programm, typischerweise in Form einer Assemblerdatei (.LST)
- **Dekodieren der Instruktionen**: Jede Instruktion wird analysiert und in eine entsprechende Softwareoperation übersetzt.
- **Ausführen der Instruktionen**: Die simulierte CPU führt die Instruktionen aus, wobei Speicher, Register und Flags entsprechend aktualisiert werden.
- **Interaktion mit Peripherie**: Simulierte Peripheriegeräte wie Timer, Interrupts oder I/O-Ports werden entsprechend der Programmlogik angesprochen.
- **Visualisierung**: Der aktuelle Zustand des Systems wird in der Benutzeroberfläche dargestellt, um den Programmablauf nachvollziehbar zu machen.

1.2 Vor- und Nachteile einer Simulation

Simulationen bieten zahlreiche Vorteile, haben jedoch auch einige Einschränkungen:

Vorteile

- **Kosteneffizienz**: Es wird keine physische Hardware benötigt, was Kosten für Entwicklung und Tests reduziert.

- **Flexibilität**: Änderungen am Programm oder an der Hardwarekonfiguration können schnell und einfach vorgenommen werden.
- **Debugging**: Der Simulator ermöglicht detaillierte Einblicke in den Programmablauf, einschließlich Register- und Speicherinhalte.
- **Sicherheit**: Fehlerhafte Programme können getestet werden, ohne reale Hardware zu beschädigen.

Nachteile

- **Eingeschränkte Genauigkeit**: Timing-kritische Anwendungen oder Hardware-spezifische Eigenschaften können nur begrenzt simuliert werden.
- **Leistungsanforderungen**: Simulationen können ressourcenintensiv sein und erfordern leistungsfähige Computer.
- **Abweichungen zur Realität**: Nicht alle Hardwaredetails können exakt nachgebildet werden, was zu Abweichungen im Verhalten führen kann.

1.3 Programmoberfläche und deren Handhabung

Die Benutzeroberfläche eines Simulators ist entscheidend für eine effiziente Nutzung. Sie sollte intuitiv gestaltet sein und alle relevanten Funktionen bereitstellen. Typische Elemente und deren Handhabung umfassen:

- **Code-Editor**: Ermöglicht das Laden und Bearbeiten von Assemblerdateien. Breakpoints können durch Anklicken der Zeilennummer gesetzt werden.
- **Steuerelemente**: Buttons wie `Run`, `Step In`, `Step Over` und `Reset` steuern die Programmausführung.
- **Register- und Speicheranzeige**: Zeigt den aktuellen Zustand der Register, Flags und des Speichers in Echtzeit an.
- **I/O-Visualisierung**: Interaktive Darstellung der I/O-Pins, die durch Anklicken manipuliert werden können.
- **Debugging-Tools**: Funktionen wie das Setzen von Breakpoints, das Überwachen von Variablen und das manuelle Überschreiben von Speicherinhalten erleichtern die Fehlersuche.

Die Kombination dieser Elemente ermöglicht eine effiziente Entwicklung, Analyse und Optimierung von Programmen für Mikrocontroller.

Kapitel 2

Benutzeroberfläche

Die Benutzeroberfläche des PIC Simulators ist so gestaltet, dass sie eine intuitive und effiziente Nutzung ermöglicht. Im Folgenden werden die wichtigsten Elemente der Benutzeroberfläche und deren Handhabung detailliert beschrieben:

2.1 Hauptfenster

Das Hauptfenster des Simulators ist in mehrere Bereiche unterteilt, die jeweils spezifische Informationen und Funktionen bereitstellen:

- **Code-Editor**: Der integrierte Code-Editor ermöglicht das Laden, Bearbeiten und Speichern von Assemblerdateien. Breakpoints können durch einfaches Anklicken der Zeilennummer gesetzt werden, um den Programmablauf gezielt zu analysieren.
- **Steuerelemente**: Buttons wie `Run`, `Step In`, `Step Over` und `Reset` steuern die Programmausführung. Diese Steuerelemente sind übersichtlich angeordnet und leicht zugänglich.
- **Register- und Speicheranzeige**: Dieser Bereich zeigt den aktuellen Zustand der Register, Flags und des Speichers in Echtzeit an. Änderungen werden sofort visualisiert, um den Programmablauf nachvollziehbar zu machen.
- **I/O-Visualisierung**: Die I/O-Pins des Mikrocontrollers werden interaktiv dargestellt. Benutzer können die Zustände der Pins durch Anklicken ändern, um verschiedene Szenarien zu simulieren.
- **Statusleiste**: Die Statusleiste zeigt wichtige Informationen wie den aktuellen Program Counter (PC), den Stackpointer und die Ausführungszeit an.

2.2 Menüleiste

Die Menüleiste bietet Zugriff auf grundlegende Funktionen des Simulators:

- **Datei**: Optionen zum Laden, Speichern und Schließen von Projekten.
- **Ansicht**: Anpassung der Benutzeroberfläche, z. B. das Ein- oder Ausblenden bestimmter Bereiche.
- **Hilfe**: Zugriff auf die Dokumentation und Tutorials.

2.3 Debugging-Tools

Die Benutzeroberfläche enthält leistungsstarke Debugging-Tools, die die Fehlersuche erleichtern:

- **Breakpoints**: Benutzer können Breakpoints setzen, um die Programmausführung an bestimmten Stellen zu pausieren und schrittweise fortzusetzen.
- **Speicherüberwachung**: Der Speicherinhalt kann in Echtzeit überwacht und bei Bedarf manuell geändert werden.
- **Registeranzeige**: Alle Register des Mikrocontrollers werden angezeigt, einschließlich ihrer aktuellen Werte.

2.4 Visualisierung und Interaktivität

Die Benutzeroberfläche bietet eine visuelle Darstellung des Mikrocontrollers und seiner Peripherie:

- **Simulation von Peripheriegeräten**: Timer, Interrupts und andere Peripheriegeräte können simuliert und überwacht werden.
- **Echtzeit-Updates**: Änderungen im Programm oder in der Hardwarekonfiguration werden sofort in der Benutzeroberfläche reflektiert.
- **Interaktive Elemente**: Benutzer können direkt mit der Simulation interagieren, z. B. durch das Ändern von I/O-Pin-Zuständen.
- **Konfiguration**: Benutzer können die Taktfrequenz des Simulators anpassen.

2.5 Zusammenfassung

Die Benutzeroberfläche des PIC Simulators kombiniert Funktionalität und Benutzerfreundlichkeit. Sie bietet alle notwendigen Werkzeuge, um Programme effizient zu entwickeln, zu testen und zu debuggen. Durch die klare Struktur und die interaktiven Elemente wird die Arbeit mit dem Simulator erheblich erleichtert.

Kapitel 3

Technische Details und Funktionsweise

3.1 Beschreibung des Grundkonzepts

Der Simulator bildet die Architektur des Mikrocontrollers nach, einschließlich CPU, Speicher, Register und Peripheriegeräten. Dabei werden alle Befehle des Mikrocontrollers interpretiert und ausgeführt, während der Zustand des Systems in Echtzeit visualisiert wird.

3.2 Warum Python gewählt wurde

Zur Umsetzung des Projekts wurde Python als Programmiersprache gewählt, da es mehrere Vorteile bietet:

- **Einfache Syntax**: Python ermöglicht eine klare und übersichtliche Implementierung, was die Entwicklung erleichtert.
- **Schnelle Entwicklungszeit**: Dank der hohen Abstraktionsebene können Prototypen schnell erstellt und getestet werden.
- **Umfangreiche Bibliotheken**: Python bietet zahlreiche Bibliotheken, wie `PyQt` für die GUI, welche die Entwicklung beschleunigen.
- **Plattformunabhängigkeit**: Python-Programme können auf verschiedenen Betriebssystemen ausgeführt werden.

3.3 Ausführung eines Befehlszyklus

Der Befehlszyklus im PIC Simulator folgt einem klar strukturierten Ablauf. Zunächst wird die Assemblerdatei analysiert, und die darin enthaltenen Befehle in ein Array geladen, das als Programmspeicher dient.

Zu Beginn eines Befehls Zyklus wird überprüft, ob sich der Mikrocontroller im Sleep-Modus befindet, die Ausführung pausiert wurde oder der aktuelle Befehl ignoriert werden soll. Anschließend wird der nächste Befehl aus dem Programmspeicher eingelesen und dekodiert. Der Programmzähler (Program Counter) wird daraufhin inkrementiert, um auf den nächsten Befehl zu zeigen. Basierend auf der Dekodierung wird die entsprechende Funktion ausgeführt, welche den Befehl implementiert und dabei alle relevanten Statusflags (z. B. Carry, Zero) setzt. Nach der Ausführung des Befehls wird die Laufzeit des Programms entsprechend der Befehlslänge inkrementiert. Gleichzeitig wird der Watchdog-Timer erhöht, um dessen Überwachung zu simulieren. Zum Abschluss eines Zyklus werden anstehende Interrupts geprüft und verarbeitet. Schließlich wird die Benutzeroberfläche aktualisiert, um den aktuellen Zustand des Mikrocontrollers visuell darzustellen.

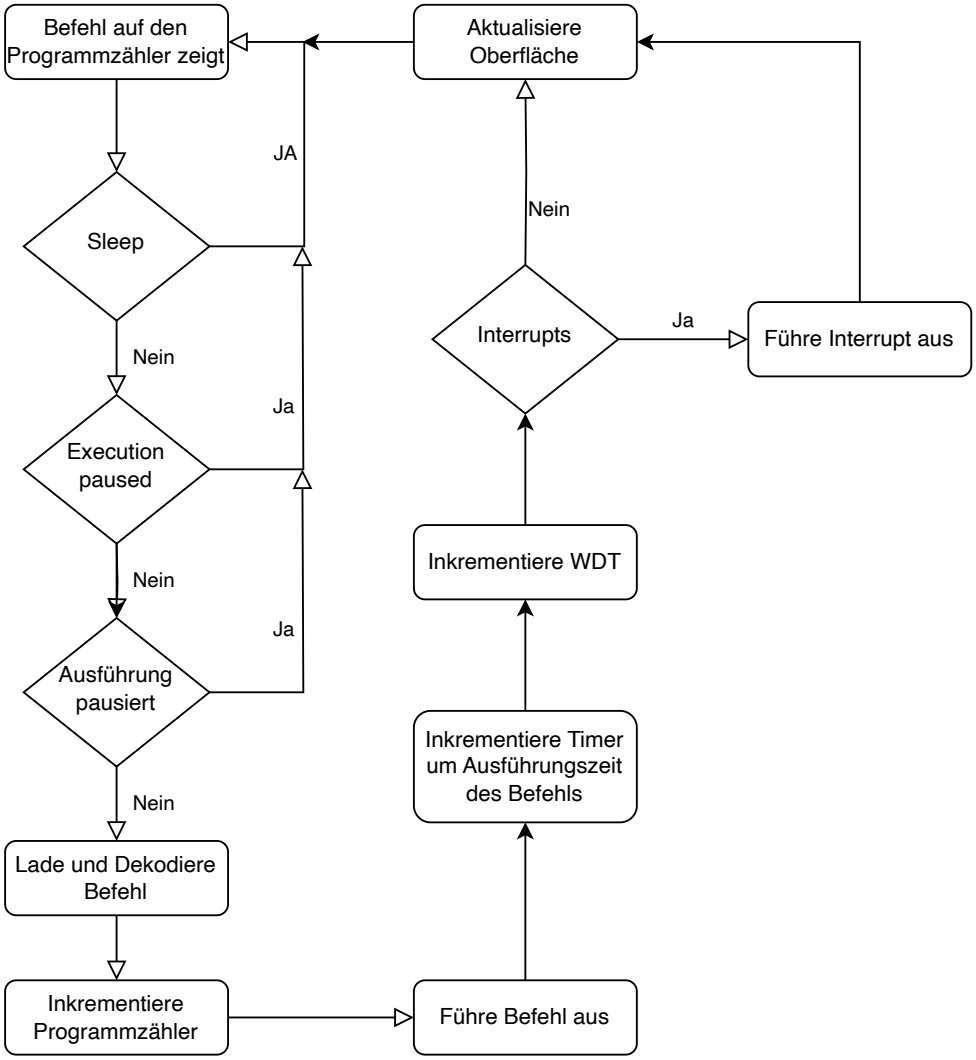

Abbildung 3.1: Programmablauf einer Befehlsausführung

Kapitel 4

Fazit

Nach Abschluss des PIC Simulator-Projekts können wir mit Stolz festhalten, dass alle ursprünglich definierten Anforderungen erfolgreich umgesetzt wurden. Der entwickelte Simulator bietet eine umfassende und benutzerfreundliche Lösung zur Simulation von PIC-Mikrocontroller-Programmen und stellt damit eine wertvolle Alternative zur physischen Hardware dar.

Die zu begin des Projekts definierten Ziele, konnten entsprechend umgesetzt werden. Der PIC Simulator ermöglicht es Entwicklern, Programme zu testen und zu debuggen, ohne auf physische Hardware angewiesen zu sein.

4.1 Herausforderungen im Entwicklungsprozess

Trotz des erfolgreichen Abschlusses des Projekts gab es während der Entwicklung einige nennenswerte Herausforderungen:

4.1.1 Integration von UI und Backend

Eine der größten Herausforderungen war die nahtlose Integration der Benutzeroberfläche mit dem Backend-System. Die Kommunikation zwischen diesen beiden Komponenten erwies sich als komplexer als zunächst angenommen:

- **Asynchrone Aktualisierungen**: Die Notwendigkeit, die UI in Echtzeit zu aktualisieren, während das Backend die Simulation durchführt, erforderte eine sorgfältige Implementierung von asynchronen Mechanismen.
- **Datenfluss**: Die effiziente Übertragung von Daten zwischen Backend und Frontend musste optimiert werden, um eine flüssige Benutzererfahrung zu gewährleisten.
- **Ereignisbehandlung**: Die Verarbeitung von Benutzereingaben und deren korrekte Weiterleitung an das Backend erforderte ein durchdachtes Ereignisbehandlungssystem.

Die Lösung dieser Herausforderungen wurde durch den Einsatz von Python und Qt erreicht, die ein robustes Signal-Slot-System für die Kommunikation zwischen Komponenten bereitstellen.

4.1.2 Kontinuierliche Weiterentwicklung

Ein weiterer anspruchsvoller Aspekt war die kontinuierliche Weiterentwicklung des Projekts. Während der Implementierung entstanden immer wieder neue Anforderungen und Herausforderungen:

- **Erweiterung der Funktionalität**: Mit fortschreitender Entwicklung wurden zusätzliche Funktionen identifiziert, die integriert werden mussten, ohne die bestehende Codebasis zu beeinträchtigen.
- **Refactoring und Optimierung**: Um die Wartbarkeit des Codes zu verbessern, waren regelmäßige Refactoring-Maßnahmen erforderlich.
- **Fehlerbehandlung**: Mit zunehmender Komplexität des Systems musste ein robustes Fehlerbehandlungssystem entwickelt werden, um mit unerwarteten Situationen umgehen zu können.
- **Leistungsoptimierung**: Die Simulation musste effizient genug sein, um auch komplexe Programme in angemessener Zeit ausführen zu können.

Der modulare Aufbau des Simulators erwies sich hier als entscheidender Vorteil, da er die schrittweise Erweiterung und Anpassung des Systems ermöglichte, ohne die Gesamtstabilität zu gefährden.

4.2 Lessons Learned

Aus diesem Projekt haben wir wertvolle Erkenntnisse gewonnen:

- **Vorausschauende Planung**: Eine gründliche Anforderungsanalyse und architektonische Planung zu Beginn des Projekts zahlen sich in späteren Phasen aus.
- **Modularität**: Ein modularer Ansatz ermöglicht flexiblere Anpassungen und Erweiterungen während der Entwicklung.
- **Testgetriebene Entwicklung**: Regelmäßige Tests sind unerlässlich, um die Funktionalität und Stabilität des Systems zu gewährleisten.
- **Dokumentation**: Eine umfassende Dokumentation erleichtert die Wartung und Weiterentwicklung des Systems.