Dokumentation PIC-Simulator

Gruppe: Stefan Stegmüller und Simon Hofmann

Inhalt

1.Grundlagen

Der PIC-Simulator wird in der .NET-Umgebung realisiert und basiert im Wesentlichen auf dem PIC-16F84-Microcontroller.

1.1 Programmiersprache

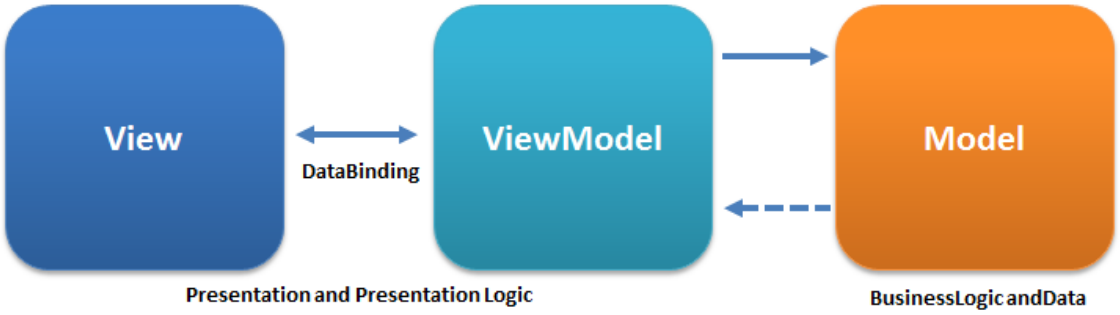
Als Programmiersprache wird C# verwendet. Diese ist eine aktuelle, objektorientierte Sprache, welche modernen Konzepten folgt. Da schon einige Erfahrungen mit C# bestanden, und im Vergleich mit C oder C++ viele Vorteile z.B. Garbage-Collection bestehen, wurde diese ausgewählt.

1.2 Entwicklungsumgebung

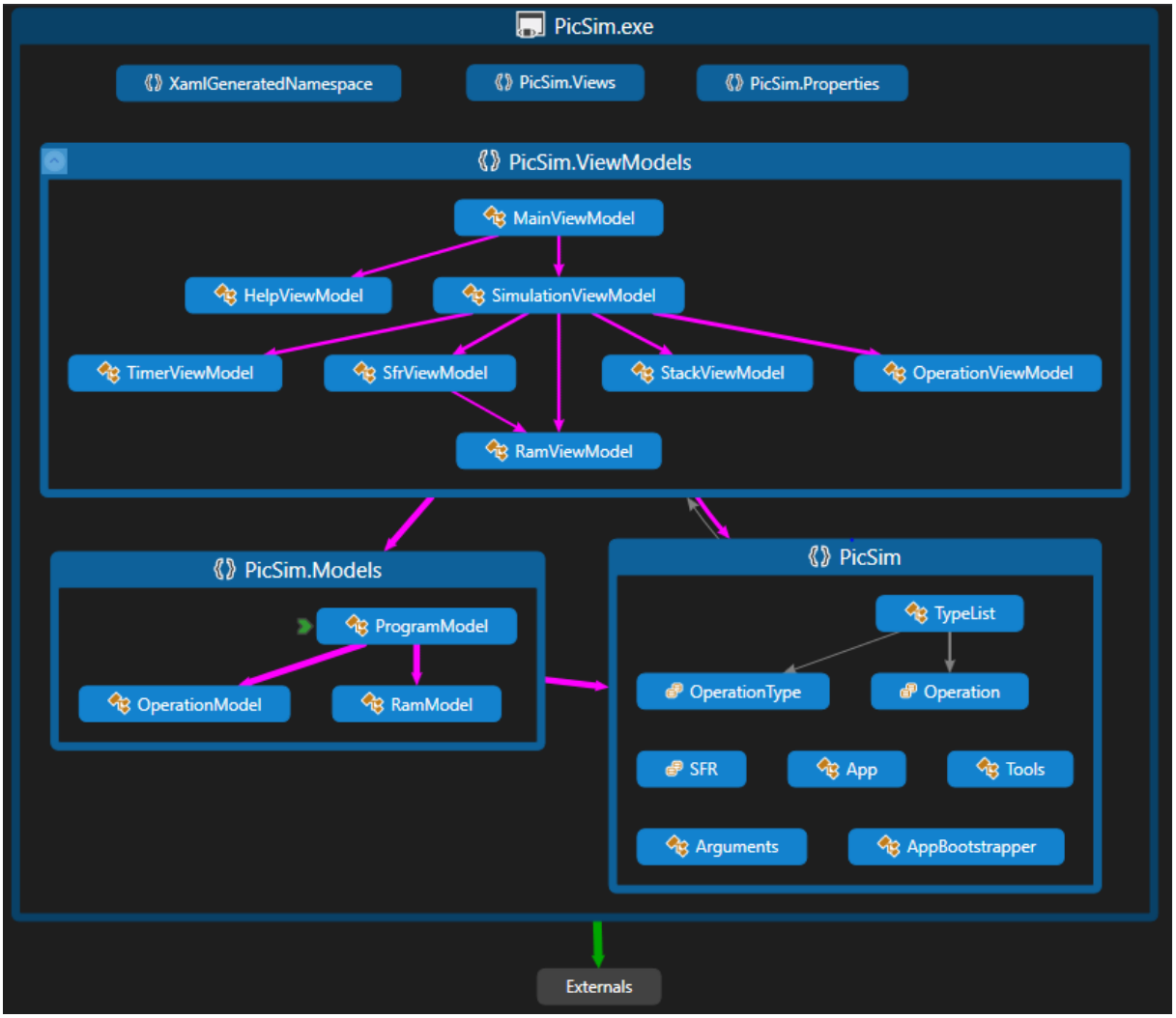
Als Entwicklungsumgebung für C# wird Visual Studio 2015 benutzt. Dies bietet sich an da Entwicklungsumgebung und Sprache von Mircosoft entwickelt wurden.

2. Programmstruktur

Das Projekt entspricht dem MVVM-Entwurfsmuster. Dies ist hilfreich, da Darstellung und Logik getrennt sind. Für die angewandte Programmiersprache C# bietet sich MVVM an.



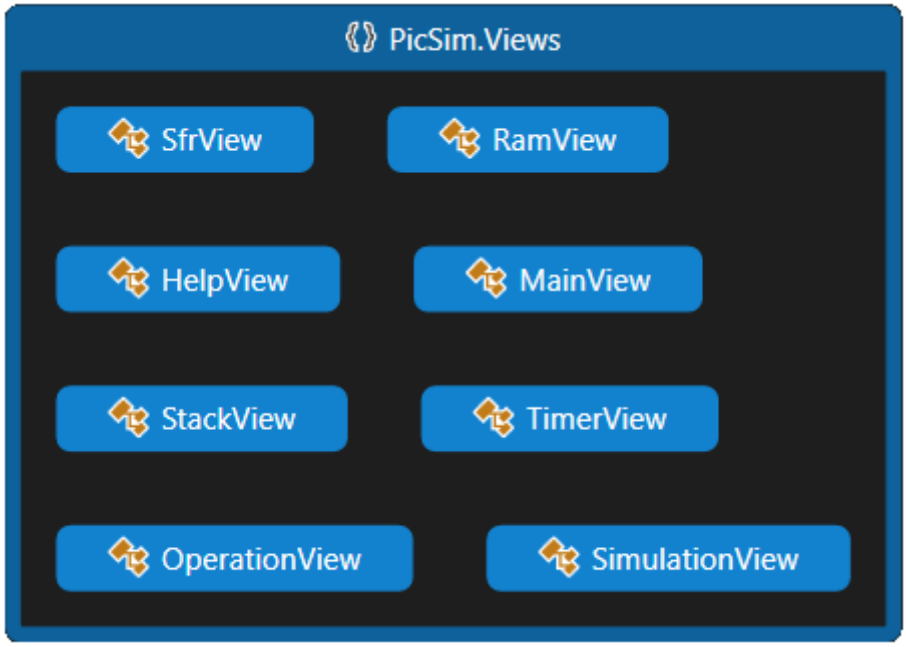
Die View dient zur Darstellung von Daten. Hier kann der Benutzer eingaben tätigen und Ausgaben betrachten. Die View beinhaltet aber keinerlei Daten. Das ViewModel stellt der View Daten über eine Datenbindung bereit. Außerdem ist es zuständig für die Anzeigelogik und das Verarbeiten von Aktionen der Benutzers an der Oberfläche. Empfangene Daten werden im vom ViewModel im Model abgelegt. Dort befindet sich dann die Geschäftslogik. Falls Daten benötigt werden holt sich das ViewModel die Daten vom Model.



Wie in diesem Klassendiagramm zu sehen ist, haben Views und ViewModels keine Klassenbeziehungen.

2.1 WPF

WPF (Windows Presentation Forms) ist ein Grafikframework in .NET. In Verbindung mit XAML-Dateien kann die Visualisierung einfach gestaltet werden. Im Projekt selbst sind die einzelnen Views unterteilt.



2.2 Frameworks

Als Framework wird Caliburn.micro eingesetzt. Es verringert den Aufwand für die Implementierung des MVVM-Entwurfsmuster. Es ermöglicht eine einfach zu implementierende Datenbindungen zwischen den Views und ViewModels.

3. Implementierung

Im Folgenden werden die entsprechenden Implementierungen erläutert.

3.1 Quellcodeübernahme

Nur LST.-Dateien können eingelesen werden. Die Befehle, Argumente und ihr Index werden anhand von Operationscodes erkannt und in einem Objekt festgehalten.

3.2 Programmablauf

Ein Programm kann mit Buttons gestartet, gestoppt oder schrittweise ausgeführt werden. Durch einen Thread wird sicher gestellt das andere Bedienelemente noch während des Programmablaufs reagieren. Der Programmablauf kann mit Breakpoints an bestimmtem Stellen gestoppt werden. Das Programm kann nur gestartet werden, falls eine LST-Datei eingelesen wurde und der Programmcounter nicht auf dem letzten Befehl steht.

3.2 Ram und Special Function Register

Der Ram ist als Byte-Array implementiert und Special Function Register können einfach über eine Enumeration mit bestimmten Adressen angesprochen werden.

3.3 Interrupts

Implementiert sind der Timerinterrupt mit Timermodus und Clockmodus, der RB0-Interrupt und der RB4-7-Interrupt. Die Interrupts an PortB werden mit dem Vergleich von temporären Variablen des letzten Befehls, realisiert. Der Timerinterrupt beachtet gegebenenfalls den Prescaler welcher mit folgender Formel realisert ist:

$$2^{(\text{OPTION\_REGISTER}(0\text{-}2) + 1)}$$

Der Clockmodus des Timers wird durch einen weiteren Thread realisiert. Da nun der Thread des Programmablauf und der Clockmodus-Thread gleichzeitig auf das Ram-Objekt zugreifen, ist das RamModel threadsicher implementiert.

3.4 Watchdog

Der Watchdog-Timer ist nicht unabhängig wie in einem echten PIC, sondern abhängig von der Lauftzeit, welche sich aus Maschinenzyklen, Quarzfrequenz und ggf. Prescaler errechnet. Bei einem Sleep-Befehl wird der Programmablauf in einen speziellen Modus umgeschaltet, sodass der Thread weiter läuft ohne Befehle auszuführen.

4. Fazit

4.1 Bewertung des Simulationsprogramms

Der Simulator konnte annähernd in vollem Umfang nachgebildet werden. Die EEPROM-Funktion mit Interrupt wurde nicht implementiert.

4.2 Allgemeines Fazit

Die Aufgabe einen Simulator für einen PIC-Microcontroller zu entwickeln, hört sich im ersten Moment enorm an. Nach einiger Arbeitszeit an diesem Projekt ergeben sich einige Fragestellungen, die mit Recherche, Teamarbeit und Eigeninitiative zu bewältigen sind.

Mit dem Lösen von Problemen den Mikrocontroller betreffend, sind neue Erkenntnisse erlangt worden. Diese betreffen sowohl die Programmierung in Assembler als auch den eigentlichen Mikrocontroller in seiner Funktionsweise. Die Entwicklung des Simulators erfordert ein sehr gutes Verständnis des realen Mikrocontrollers. Die Funktionen und das Verhalten z.B. nach Eintreffen eines Interrupts, müssen exakt wiedergegeben werden. Die einzelnen Assemblerbefehle sind ebenso mit ihren entsprechenden Auswirkungen auf den RAM und speziell das Statusregister implementiert. Um die einzelnen Zugriffe im Simulator trennen zu können, wurden Threads verwendet. Diese werden unter anderem beim Timer, Watchdog und START/STOP verwendet. Auch die Visualisierung spielt eine enorme Rolle. Hierbei hat das Design Pattern MVVM geholfen. Die Ansicht wurde von der Logik komplett getrennt. So konnte die Logik nahezu unabhängig implementiert werden.

Nicht nur bei dem Thema Assembler und PIC verbesserten sich Kenntnisse, sondern auch bei der .NET-Entwicklung und der Implementierung von Design Patterns. Hier konnten auch Inhalte anderer Vorlesungen verwendet werden. Mit C# als verwendete Programmiersprache konnten neue Erfahrungen gesammelt werden.