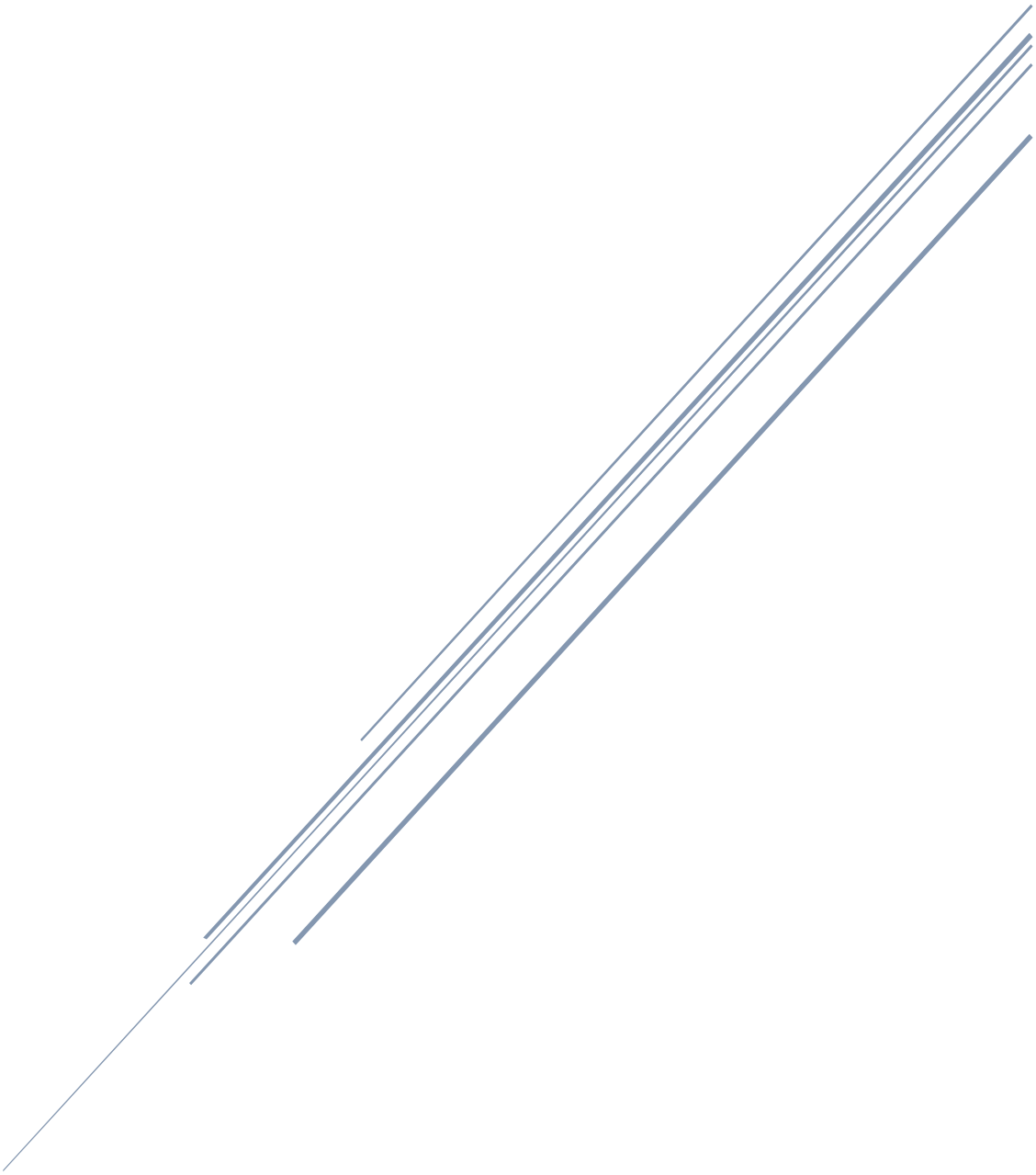
PIC16F84-Simulator

Author: André Schmitt, Dominik Vogel



DHBW Karlsruhe
TINF15B3

Inhaltsverzeichnis

Arbeitsweiße

Laden eines Hex oder Assambler files

Um eine Datei zu laden, muss diese entweder in den Simulator gezogen, oder über den open Button Geöffnet werden. Das Geladene File wird im Editor angezeigt und wenn nötig kompiliert.

> Es werden nur die Assambler Befehle angezeigt, nicht die Hex Befehle.

Ändern sie das Programm im Editor, wird es nach Speicherung des Programms erneut kompiliert.

Starten von geladenen Programmen

Um das geladene Programm zu starten müssen sie lediglich den ‚Start' Button auf der Oberfläche drücken. Alternativ kann auch ‚Step in' oder ‚Step over' gedrückt werden, um das Programm direkt in einzelschritten zu durchlaufen.

Öffnen von Peripherie

Um z. B. ein Taster Feld zu öffnen, wählen sie bei Peripherals die Gewünschte aus, und klicken sie auf ‚create Peripherals'.

In allen Peripherals kann über einen rechtsklick verschiedene Optionen aufgerufen werden.

Vor- und Nachteile

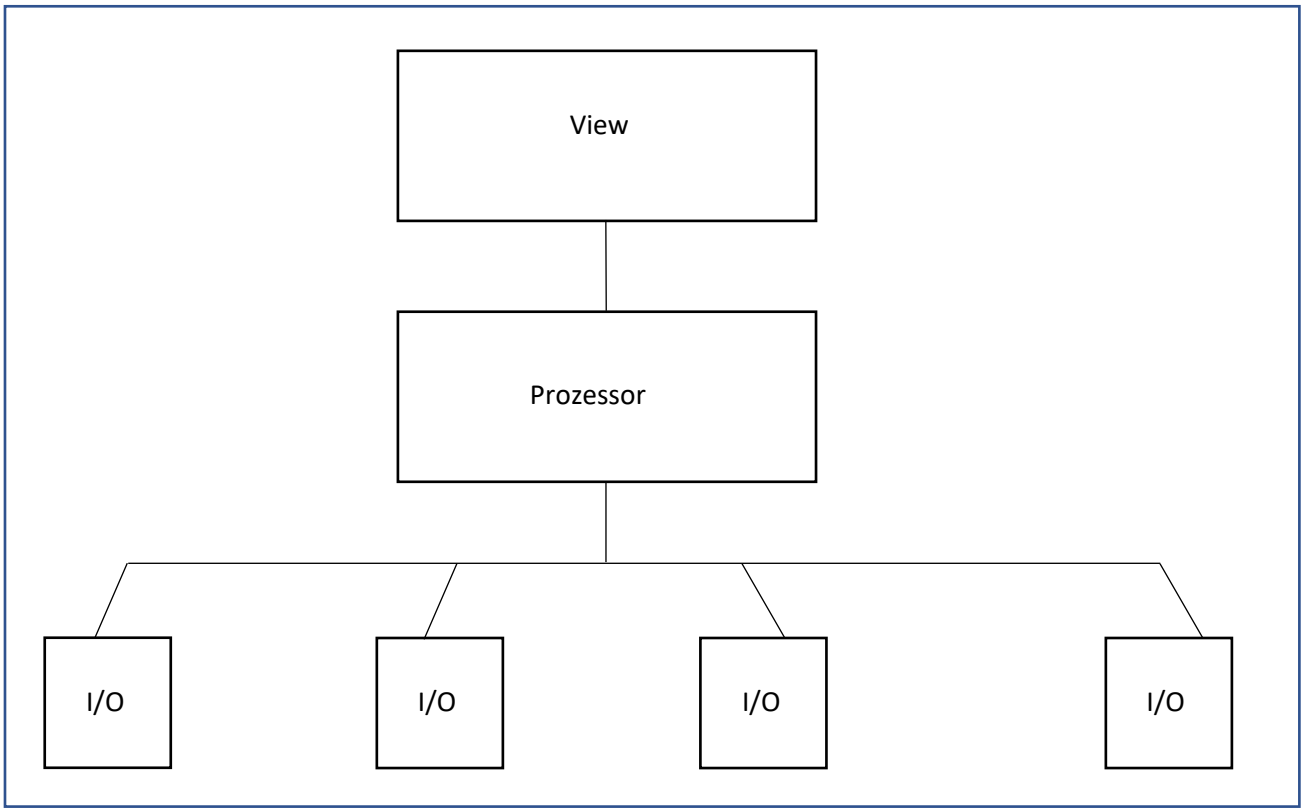
Programmoberfläche und deren Handhabung

TODO: 3

Realisierung

Grundkonzept

Gliederung



Programmstruktur

Das Programm untergliedert sich grob in drei Teile. Dem Hauptfenster, dem emulierten Prozessor und der Verknüpfung des Prozessors mit externer Peripherie.

Hauptfenster

Das Hauptfenster wird von der Klasse **TfrmMain** repräsentiert. Sie beinhaltet neben der Benutzeroberfläche auch eine Instanz der **TProcessor** Klasse. Des Weiteren, können über das Hauptfenster diverse Peripherie-Emulationen geöffnet werden.

Prozessor

Der Prozessor wird durch die **TProcessor** Klasse dargestellt. In dieser Klasse sind alle, für die vollständige Emulation eines PIC16F84-Prozessors, relevanten Felder und Funktionen implementiert.

Darunter fallen unter anderem:

- Programmspeicher inklusive Programmzähler und Stack
- Working-Register
- File Map inklusive Special-Function-Register
- Externer EEPROM inklusive State-Machine
- Pins (mithilfe der **TPin**/**TPinArray** Klassen)
- WatchDog-Timer
- Breakpoints
- Interrupts

Peripherie

Verschiedene Peripherien können einfach hinzugefügt werden, indem man von der **TPeripheralForm** Klasse erbt und die virtuellen Funktionen für Benutzereingaben, Visualisierung und Logik überschreibt.

In der aktuellen Version der Anwendung sind zwei Peripherien verfügbar. Ein Button-Panel, für externe Eingaben am Prozessor und ein LED-Array, um eine vom Prozessor generierte Ausgabe anzuzeigen.

Die Verbindung zwischen den Pins am Prozessor und den Pins eines Peripherie-Objekts können einfach per „drag'n'drop" hergestellt werden. Es ist sowohl möglich, Pins einzeln zu verbinden, aber es können auch direkt alle Pins auf einmal verbunden werden.

Programmiersprache

Beschreibung der Funktionen

Rechen Operationen

Es werden Addition (**addwf**/**addlw**) und Subtraktion (**subwf**/**sublw**) unterstützt. Beide nutzen eine weitere Funktion, welche eine Addition durchführt, das Ergebnis zurückliefert und dabei auch die Status-Flags setzt.

Logische Operationen

Für logische Operationen werden die nativ von Lazarus unterstützen Operatoren benutzt:

- **and** → **andwf**, **andlw**
- **or** → **iorwf**, **iorlw**
- **xor** → **xorwf**, **xorlw**
- **not** → **comf**
- **shl/shr** → **rlf**, **rrf** (rotate passiert nicht automatisch)

Bit Operationen

Für Bitorientiere Operationen wurde eine Property angelegt, welche es erlaubt auf die einzelnen Bits der innerhalb der Filemap direkt zuzugreifen.

- Set → **bfs**
- Clear → **bfc**

Andere zuweisende Operationen

Folgende Befehle nutzen lediglich die Lazarus eigene Zuweisung, um ihren Zweck zu erfüllen:

- Nullsetzend → **clrf**, **clrw**
- Kopierend → **movwf**, **movlw**, **movf**
- Inc/Decrement → **incf**, **decf**
- Swap → **swapf**

Sprung Operationen

- Direkter Sprung → **goto**, **call**, **return**, **retlw**, **retfie**
- Bedingter Sprung → **incfsz**, **decfsz**¸ **btfsc**, **btfss**

Sleep

Der Sleep-Befehl ist mittels eines Flags implementiert, welches während einem Sleep-Befehl gesetzt wird. Dieses Flag verhindert jegliche Ausführung von Befehlen. Wird jedoch ein Interrupt erkannt, so wird dieses Flag wieder zurückgesetzt, wodurch Befehle wieder ausgeführt werden.

Auch der Watchdog-Timer kann den Sleep-Modus beenden. Dieser Timer kann mittels des Befehls **clrwdt** zurückgesetzt werden.

Flags

Über eine Funktion werden Flags aus dem Ram gelesen und als boolean zurückgegeben.

Interrupts

Interrupts wurden über den Ram Implementiert. Vor jeder Ausführung eines Befehls werden die verschiedenen Interrupts abgeprüft. Ist ein Interrupt aufgetreten, wird der Programm Counter auf den Interrupt Vektor gesetzt.

TRIS-Register

Breakpoints

Hardware Ansteuerung

n/a

State-Machine / EEPROM

Die State-Machine wurde eine Variable angelegt, indem der Status gespeichert werden kann. Wird ein init zyklus erkannt, wird diese Variable auf 'esReady' gesetzt. Bei einem zugriff auf den EEPROM wird zuerst auf diese Variable geprüft. Ist sie nicht 'esReady' wird kein wert gelesen.

Fazit

André:

Durch dieses Projekt konnte ich Einblicke in die Sprache Lazarus bekommen. Ich konnte mir ein Bild über die Möglichkeiten und Tücken von der Programmiersprache machen. Des weiteren konnte ich die Funktionsweiße des PIC 16F84 besser verstehen.