PIC16F84-SIMULATOR-DOKUMENATION

Rechnerarchitekturen-Labor (SS 2019)

Marcel Grüßinger

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

Im Rahmen des Labors zu der Lehrveranstaltung „Rechnerarchitekturen“ sollte ein Simulator für den Microcontroller „PIC16F84“ umgesetzt werden. Dabei wurde die Programmiersprache mit Java vorgegeben. Welches Framework für die Implementierung der graphischen Benutzeroberfläche verwendet werden soll wurde allerdings nicht vorgegeben.

Diese Dokumentation soll die Grundlegenden Softwarearchitekturentscheidungen und abstrakte Funktionsweise des Simulators näher beschreiben.

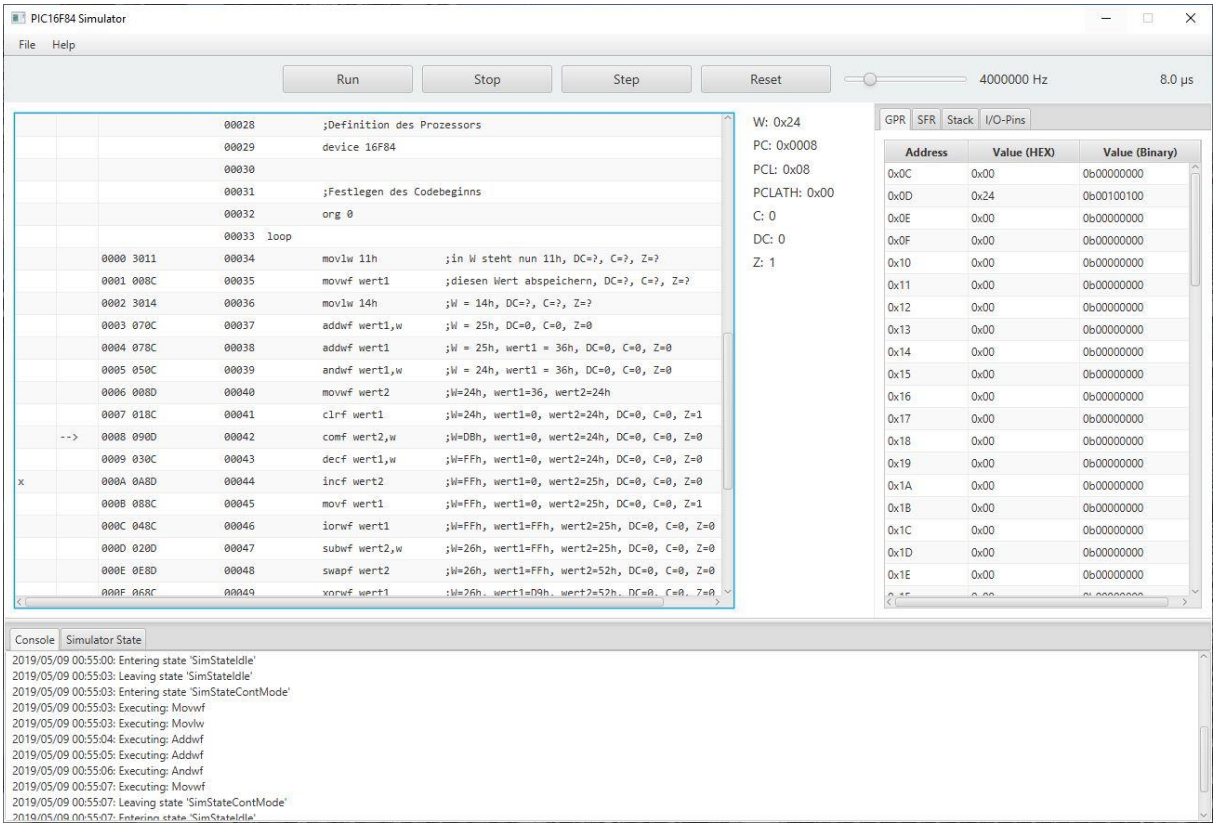

Abbildung 1: Screenshot des Simulators

Klassendiagramm

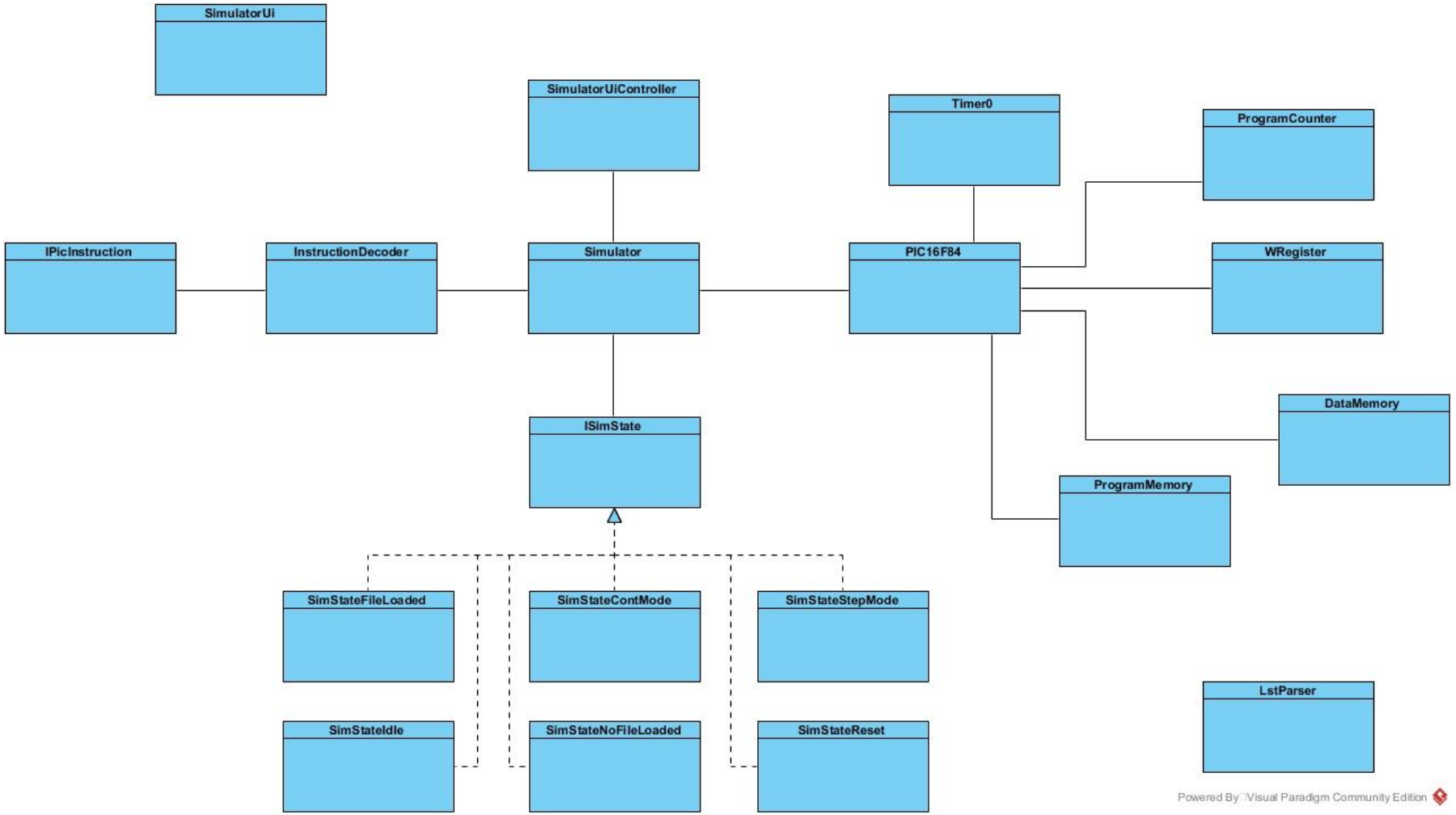

Abbildung 2: Klassendiagramm

Die Abbildung 2 zeigt die essenziellsten Klassen des PIC16F84-Simulators. Nicht zu sehen sind unter anderem die Klassen der 35 Befehle, welche von dem Microcontroller unterstützt werden. Die zentralen Klassen des Projekts sind die folgende:

- Simulator
- PIC16F84

Die Klasse „Simulator“ verwaltet, wie der Name bereits vermuten lässt, den Zustand (siehe Kapitel „Zustandsautomat“) des Simulators. Dazu zählt allerdings auch die Aktualisierung des Laufzeitzählers und die Ausführung der Befehle.

Die Klasse „PIC16F84“ beinhaltet alle Referenzen zu den Klassen die zum Beispiel die Logik des Programmspeichers oder des RAMs umsetzen.

Zustandsautomat

Zu Grunde des Simulators liegt der in Abbildung 3 visualisierte Zustandsautomat. Dieser sorgt für die korrekte Abarbeitung von Benutzereingaben (bspw. durch den Klick auf einen Button).

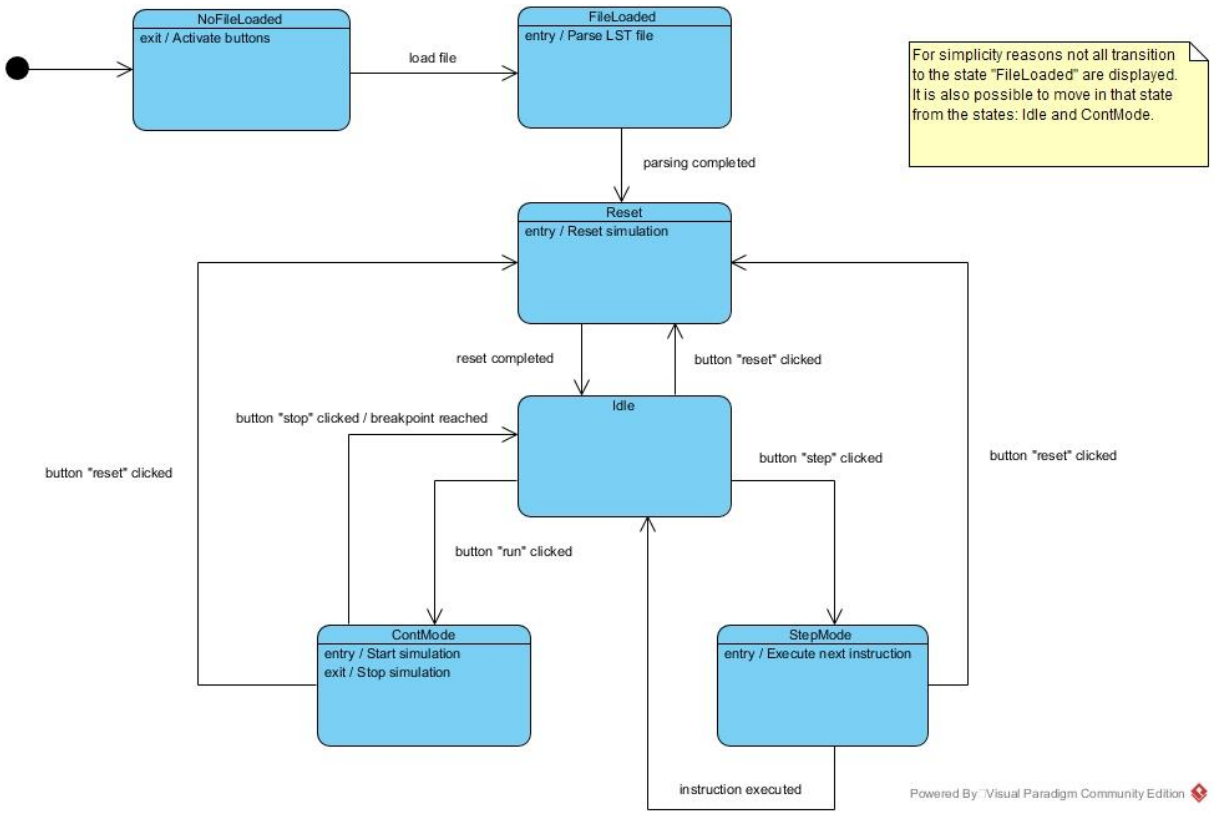

Abbildung 3: Diagramm des Zustandsautomaten

Befehlsausführung

Die Methode „executeSingleInstruction()“ aus der Klasse „Simulator“ ist für die Ausführung der Befehle verantwortlich. Diese arbeitet die Instruktionen nach dem identischen Prinzip der Pipeline ab (Fetch -> Decode -> Execute. Des Weiteren wird zu Beginn überprüft, ob ein Interrupt des Timer0 aufgetreten ist und ggfs. die Ausführung der ISR eingeleitet. Ebenso wird in dieser Methode der Laufzeitzähler, der Programmzähler und der Zähler des Timer0 erhöht.

```
public void executeSingleInstruction() {
   if(picController.getTimer0().checkForTimer0Interrupt()) {
      byte intconValue = picController.getDataMemory().load((byte) 0x0B);
      intconValue = (byte) (intconValue & 0b01111111);
      picController.getDataMemory().store((byte) 0x0B, intconValue);

      new Call((short) 0x0004).execute(picController);
   }

   // Fetch
   short opcode = picController.getProgramMemory().getNextInstruction();
   picController.getProgramCounter().incrementProgramCounter();

   // Decode
   IPicInstruction instruction = decoder.decode(opcode);

   // Execute
   LOGGER.info("Executing: " + instruction.getClass().getSimpleName());
   notifyDebugConsole("Executing: " + instruction.getClass().getSimpleName());
   int cycles = instruction.execute(picController);

   updateRuntimeCounter(cycles);
   picController.getTimer0().increaseTimerCounter();

   notifyCurrentExecutedCode();
}
```

Fazit

Die Durchführung des Projekts verlief ohne größere Probleme. Dieser Erfolg ist vor allem auf die einwöchige Analyse- und Designphase zurückzuführen. Während der Analysephase wurden die Anforderungen so evaluiert, dass diese Softwarekomponenten (wie bspw. dem Parser für LST-Files) zugeordnet werden konnten. Deshalb musste in der darauffolgenden Designphase nur noch bestimmt werden in welcher Weise die verschiedenen Komponenten miteinander interagieren. Resultate aus diesen beiden Projektphasen sind das Klassendiagramm (Abbildung 2) und der Zustandsautomat (Abbildung 3). Zusätzlich wurden die Anforderungen aus dem Lastenheft als Issues innerhalb von GitHub abgebildet. Dies ermöglichte eine strukturierte Abarbeitung der Anforderungen, da hier die Abhängigkeiten zwischen den Issues klar dargestellt werden konnte.

Somit mussten in der Implementierungsphase nur noch die Diagramme in Quellcode umgewandelt werden. Die restlichen Implementierungsdetails wurden dem Datenblatt des Microcontrollers und dem Themenblatt von Herrn Lehmann entnommen. Das Setzen der Flags C, DC und Z bei den Addier- und Subtrahierbefehlen stellte in dieser Phase eine gewisse Herausforderung dar. Hier wurde vergessen nach der Konvertierung der Registerwerte von Byte zu Integer die oberen 24 Bit mit einer bitweise Verundung zu löschen.

Grundsätzlich lässt sich festhalten, dass die Vorkenntnisse aus der Lehrveranstaltung „Maschinennahe Programmierung“, welche regulär im vierten Semester des Studiums stattfindet, sehr von Vorteil waren und die Umsetzung des Simulators deutlich beschleunigt hat. Dennoch könnte an der Art und Weise, wie dieses Projekts durchgeführt wurde, eine Verbesserung vorgenommen werden. Die graphische Benutzeroberfläche wurde erst spät in der Entwicklung fertiggestellt, da diese mit der Implementierung von Features weitergewachsen ist und später (aufgrund von Unübersichtlichkeit) neu entworfen werden musste. Diesen zusätzlichen Aufwand hätte man durch die Erstellung von Storyboards oder eines einfachen Mockups in der Designphase einsparen können.

Letzen Endes hinterlässt dieses Projekt einen positiven Eindruck, da die Implementierung zum einen Spaß gemacht hat und zum anderen auch sehr lehrreich war.