Rechnerarchitektur
Sommersemester 24

ertis embedded real-time systems ulm university institute

Laborprojekt – Versuch 1

In der Veranstaltung *"Grundlagen der Technischen Informatik - Projekt"* haben sie eine Arithmetisch Logische Einheit (ALU) mit VHDL entwickelt und getestet. Der Schwerpunkt lag auf dem Erlernen von VHDL und dessen Anwendungsmöglichkeiten. Im Rahmen des Labor-Projektes der Veranstaltung *"Rechnerachitektur"* werden sie einen vollständigen 32-Bit-RISC-V-Prozessor entwickeln und testen.

Die wichtigste Ihnen zur Verfügung stehende Referenz ist das Handbuch:
`https://riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf`

Es wird empfohlen, das Handbuch herunterzuladen, damit Sie jederzeit schnell darauf zugreifen können. Für dieses Projekt sind insbesondere die Abschnitte 2.1 und 2.4 bis 2.6 sowie Kapitel 19 mit einem Überblick über die Befehlskodierungen (Seite 104) relevant.

Die Aufgabenblätter der Labor-Projekte sind immer dreiphasig aufgebaut. Der erste Bereich dient zur Hinleitung und Einführung des aktuellen Themengebietes und bietet allgemeine Informationen. Im zweiten Bereich werden die Ziele des jeweiligen Termins zusammengefasst sowie Quellen und Hinweise für spezifisches Fachwissen für Ihre Vorbereitung zur Verfügung gestellt. Der dritte Bereich beschreibt die Aufgaben, Einschränkungen und Abnahmekriterien des Termins. Die Aufgabe beinhaltet, in den meisten Fällen, eine Beschreibung der Entity, Ports und weiteren Eigenschaften, die von Ihnen entsprechend umgesetzt werden sollen. Eine erfolgreiche Abnahme durch die Tutoren beinhaltet die vollständige, lauffähige Umsetzung der Aufgabe unter Einhaltung der Programmierkonvention. Optional müssen Sie ein Kolloquium, welches die Themengebiete der Vorbereitung abdeckt, erfolgreich absolvieren.

Ziele des 1. Versuchs

- Verständnis der Prozessorspezifikation von RISC-V
- Einführung der Programmierkonventionen für den RISC-V
- Anpassung und Erweiterung der ALU und dem Constant_Package aus dem GdTi-Labor
- Konzeptionierung, Umsetzung und Validierung eines generischen Multiplexers
- Konzeptionierung, Umsetzung und Validierung eines generischen Pipeline-Registers

Vorbereitung

GHDL

Wie im vorherigen GdTI-Projekt wird auch in diesem Projekt `ghdl` verwendet. Erarbeiten Sie sich, falls nötig, die grundlegenden Schritte für die Arbeit mit `ghdl`:

Im Folgenden gibt es eine kurze Einführung in die Kommandos, die man – in der gegebenen Reihenfolge – für die typische Arbeit mit `ghdl` und `gtkwave` im Labor-Projekt benötigt. Speziell für `ghdl` existiert eine gute Dokumentation unter `https://ghdl.github.io/ghdl/`

VHDL-Compiler wie `ghdl` speichern bereits analysierte Quelltexte in einer sogenannten *workbench*. Beim Schritt *elaborate* wird das Hauptprogramm mit bereits analysierten Entitäten automatisch aufgefüllt. Meist ist dies sehr praktisch, kann aber auch zu unerwünschten Nebeneffekten führen, weshalb Sie diese ggf. zuerst leeren sollten:

```
rm work-*.cf
```

Das hier formulierte `rm`-Kommando stellt sicher, dass `ghdl` im aktuellen Verzeichnis vergisst, was schon analysiert wurde. Danach folgt der eigentliche Ablauf, bei dem man *frisch* alle benötigten Entitäten (engl. entities) (i) analysiert, (ii) elaboriert und wahrscheinlich (iii) ausführt und (iv) visualisiert.

Analysieren Die Option `-a` analysiert den gegebenen Quelltext hinsichtlich Syntax und Vollständigkeit der verknüpften Entitäten, bindet alle benötigten Quelldateien und Bibliotheken in das Projekt ein und verknüpft diese.

```
ghdl -a <name>.vhdl
```

Elaborieren Die Option `-e` elaboriert bereits analysierte Quelltexte.

```
ghdl -e <name>
```

Beim Elaborieren wird eine Liste aller im Design verwendeten Entitäten erstellt, auf Architekturebene verknüpft und die Top-Entity des Designs identifiziert und durch den Parameter `<name>` festgelegt. Diese Entität wird als Entwurfsentitätseintragspunkt (engl. design entity instantiation) bezeichnet; in unserem Fall ist dies typischerweise eine *Testbench*. Durch das Elaborieren wird eine ausführbare Datei erstellt, die den Entwurfsentitätseintragspunkt enthält und zur Simulation des Designs verwendet werden kann. Wichtig: Alle benötigten Quelltexte bzw. Entities müssen vorher analysiert worden sein!

Ausführen Die Option `-r` für *run* führt eine Simulation durch.

```
ghdl -r <name> --vcd=<name>.vcd --stop-time=100ns
```

`<name>` muss dabei dem des vorher durchgeführten Elaborate-Schritts entsprechen. Die Option `--vcd=<name>` wird nur benötigt, wenn man die Ergebnisse der Simulation mit `gtkwave` anschauen will; für gewöhnlich schadet sie im Labor-Projekt nicht. Außerdem sollten Sie die Simulation sinnvoll zeitlich beschränken – insbesondere, wenn Sie zur Visualisierung `gtkwave` verwenden möchten. Die Option `--stop-time` erlaubt dies. Im Beispielfall wird nach $100\,ns$ simulierter Zeit abgebrochen.

Visualisieren Zur Visualisierung verwenden wir im Labor-Projekt das Werkzeug *GTKWave*.

```
gtkwave <name>.vcd
```

Das Kommando öffnet `gtkwave` mit den Daten aus dem Ausführungsschritt. In GTKWave muss man typischerweise zuerst links oben eine Entity auswählen. Wann muss man im linken, unteren Fenster die gewünschten Signale der gewählten *Entity appenden*. Nun hat man im großen Fenster rechts das Spannungs-Zeit-Diagramm der gewählten Signale. Gegebenenfalls muss man noch die Zeitskalierung des Diagramms mit den Lupen-Icons geeignet einstellen. Durch die Art des Aufrufs muss man `gtkwave` beenden, ehe man auf der Kommandozeile weiterarbeiten kann.

Spezifikation RISC-V

Lesen Sie sich Abschnitte 2.1 und 2.4 bis 2.6 der Referenz (RV32I Base Integer Instruction Set, Version 2.0) durch und eignen Sie sich ein Grundverständnis der Befehlsstruktur so an, dass sie den Überblick der Befehlskodierungen (Seite 104) erklären und beschreiben können.

ALU

Überprüfen Sie Ihre Umsetzung der ALU aus dem GdTi-Labor auf Lauffähigkeit und überarbeiten Sie diese oder erstellen Sie diese gegebenenfalls neu.

Multiplexer und Register

Frischen Sie Ihre erworbenen Grundkenntnisse über generische Entities, sowie dem Verhalten von Registern und Multiplexern auf. Überlegen Sie sich ein theoretisches Konzept zur

Umsetzung eines generischen Register und Multiplexer sowie deren Testbenches.

Aufgaben

Aufgabe 1: ALU

Überarbeiten Sie Ihre ALU und die Testbench aus dem Labor-Projekt in GdTi.

- Erstellen Sie eine Ordnerstruktur für das Labor-Projekt mit vier Unterordnern Packages, Komponenten, Testbenches und Risc-V.
- Kopieren Sie die Dateien der ALU aus GdTi in den Unterordner ALU von Komponenten und die dazugehörige Testbench in den Unterordner ALU in Testbenches.
- Erstellen Sie ein Bash-Skriptim Ordner *<Projektordner>/Testbenches/ALU*, mit dem Sie die Simulation der ALU-Testbench inklusive der Analyse und Elaboration automatisch und fehlerfrei ausführen können.
- Passen Sie Ihre Umsetzung der ALU an die Programmierkonvention in der Datei *"CodingConventions.vhdl"* an und nutzen Sie die Konstanten, welche in der aktualisierte Datei *"Constant_Packages.vhdl"* definiert sind.
- Analysieren, Elaborieren und Simulieren Sie erfolgreich die angepasste ALU mit der zur Verfügung gestellten Testbench.
- Laden Sie die erstellte VHDL-Funktion und die dazugehörigen Dateien als Zip-File:

  Name_Vorname_Versuch1_Aufgabe1.zip in Moodle hoch.

  Anmerkung: Sollten Sie die ALU bislang nicht erstellt haben, sind die Spezifikationen der ALU sowie aller benötigten Komponenten und deren Rümpfe und Beschreibungen unter dem Bereich 'Zusätzliche Daten' im Moodle-Kurs hinterlegt.

Abnahme *Funktion der ALU*, Programmierkonventionen, RISC-V Spezifikation des Befehlssatzes und generische Parameter.

Aufgabe 2: Pipeline-Registers mit generischer Datenbreite

Ein Pipeline-Register ist ein grundlegendes Element in einem Prozessor mit Pipelining. Pipelining ist eine Technik zur Verbesserung der Verarbeitungsgeschwindigkeit von Befehlen in einem Prozessor, indem die Ausführung eines Befehls in mehrere Stufen oder Phasen aufgeteilt wird. Jede Stufe bearbeitet einen Teil des Befehls.

Das Pipeline-Register mit generischer Datenbreite `registerWidth` dient daher als Zwischenspeicher für die Daten, während sie zwischen den verschiedenen Stufen des Pipeline-Verarbeitungszyklus bewegt werden. Es puffert die Daten, um sicherzustellen, dass sie konsistent und stabil von einer Stufe zur nächsten übertragen werden können und spielt eine wichtige Rolle bei der Synchronisierung.

Das Verhalten des Pipeline-Registers kann durch einen Prozess beschrieben, der auf den Takt `pi_clk` und das Rücksetzsignal `pi_rst` reagiert. Bei fallender Flanke des Takts wird das Eingangssignal `pi_data` in das Register geschrieben. Wenn das Rücksetzsignal aktiviert ist, wird das Register auf einen definierten Startwert zurückgesetzt.

Das Ausgangssignal des Pipeline-Registers `po_data` entspricht immer dem aktuellen Wert des Registers. Dies ermöglicht eine synchrone und konsistente Übertragung der Daten zwischen den Pipeline-Stufen.

- Erstellen Sie den Unterordnern *<Projektordner>/Komponenten/Register*.
- Erstellen Sie die Entity PipelineRegister in der Datei *PipelineRegister.vhdl* und setzten Sie das beschriebene Verhalten um.
- Erstellen Sie die Testbench mit Entity PipelineRegister_tb in der Datei *PipelineRegister_tb.vhdl* im Unterordner *<Projektordner>/Testbenches/Register*, welche das Register für die Datenbreite $5, 6, 8, 16, 32$ für zufällige Binärwerte testet.
- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Testbenches/Register*, mit dem Sie die Simulation der Register-Testbench inklusive der Analyse und Elaboration automatisch und erfolgreich ausführen können.
- Laden Sie die erstellte VHDL-Funktion, die Testbench und die dazugehörigen Dateien als Zip-File:

  Name_Vorname_Versuch1_Aufgabe2.zip in Moodle hoch.

Abnahme *Funktion der Pipeline-Register*, Programmierkonventionen, RISC-V Spezifikation des Befehlssatzes und generische Parameter.

Aufgabe 3: 2-1-Multiplexers mit generischer Datenbreite .....

Ein 2-1-Multiplexer ist ein grundlegendes Bauelement in der digitalen Schaltungstechnik. Er wird verwendet, um zwischen zwei Eingangssignalen zu wählen und eines davon als Ausgangssignal auszugeben, basierend auf einem Steuersignal.

Das Verhalten des 2-1-Multiplexers kann durch eine einfache logische Funktion beschrieben werden: Wenn das Steuersignal `pi_selector` auf "0"gesetzt ist, wird das erste Eingangssi-

gnal `pi_first` mit generischer Breite `dataWidth` auf den Ausgang `po_output` geschaltet, andernfalls wird das zweite Eingangssignal `pi_second` ausgegeben.

- Erstellen Sie den Unterordner *<Projektordner>/Komponenten/Multiplexer*.
- Erstellen Sie die Entity Multiplexer in der Datei *gen_mux.vhdl* und setzen Sie das beschriebene Verhalten um.
- Erstellen Sie die Testbench mit der Entity *gen_mux_tb* in der Datei *gen_mux_tb.vhdl* im Unterordner *<Projektordner>/Testbenches/Multiplexer*, welche den Multiplexer für die Datenbreite $5, 6, 8, 16, 32$ für zufällige Binärwerte an den Eingängen testet.
- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Testbenches/Multiplexer*, mit dem Sie die Simulation der Multiplexer-Testbench inklusive der Analyse und Elaboration automatisch und erfolgreich ausführen können.
- Laden Sie die erstellte VHDL-Funktion, die Testbench und die dazugehörigen Dateien als Zip-Datei unter dem Namen

  Name_Vorname_Versuch1_Aufgabe3.zip in Moodle hoch.

Abnahme *Funktion des Multiplexers*, Programmierkonventionen, RISC-V Spezifikation des Befehlssatzes und generische Parameter.