Rechnerarchitektur
Sommersemester 25

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 Fokus lag dabei auf dem Erlernen von VHDL und dessen vielfältigen Anwendungsmöglichkeiten. Im Rahmen des Laborprojekts der Veranstaltung *"Rechnerarchitektur"* werden Sie nun einen vollständigen 32-Bit-RISC-V-Prozessor entwickeln und testen. Die wichtigste Referenzquelle für dieses Projekt ist das offizielle Handbuch.

Es wird empfohlen, das Handbuch herunterzuladen, um jederzeit schnell darauf zugreifen zu können. Für dieses Projekt sind insbesondere Kapitel 1 und 2 von Bedeutung.

Die Aufgabenblätter der Laborprojekte sind in der Regel dreiphasig aufgebaut:

1. **Einführungsphase**: Diese Phase führt in das aktuelle Themengebiet ein und stellt allgemeine Informationen bereit.

2. **Zielsetzung und Vorbereitung**: Hier werden die Ziele des jeweiligen Termins zusammengefasst. Zudem finden Sie Quellen und Hinweise für spezifisches Fachwissen, das Sie zur Vorbereitung benötigen.

3. **Aufgabenbeschreibung und Abnahme**: In diesem Abschnitt werden die konkreten Aufgaben, Einschränkungen und Abnahmekriterien des Termins detailliert beschrieben. Die Aufgaben beinhalten in den meisten Fällen eine Beschreibung der Entity, der Ports sowie weiterer Eigenschaften, die Sie entsprechend umsetzen sollen. Eine erfolgreiche Abnahme durch die Tutoren setzt die vollständige und lauffähige Umsetzung der Aufgabe unter Einhaltung der Programmierkonventionen voraus.

Optional müssen Sie ein Kolloquium ablegen, in dem die Themengebiete der Vorbereitung abgefragt werden. Das Kolloquium dient dazu, Ihr Verständnis und Ihre Fähigkeit, die im Labor behandelten Konzepte anzuwenden, zu überprüfen. In diesem Gespräch werden Sie aufgefordert, die wesentlichen Aspekte der RISC-V-Architektur, der Implementierung von Multiplexern, Pipeline-Registern und der Anpassung von ALUs zu erläutern. Darüber hinaus werden praktische Fragestellungen gestellt, die Ihre Fähigkeiten in der Problemanalyse und -lösung unter Beweis stellen sollen. Das Kolloquium bietet Ihnen auch die Gelegenheit, tiefer in spezifische Themen einzutauchen und eventuelle Unklarheiten zu klären, bevor die endgültige Abnahme des Projekts erfolgt.

Ziele des 1. Versuchs

Im Rahmen des ersten Laborversuchs wird ein vertieftes Verständnis der Prozessorspezifikation von RISC-V erarbeitet. Zunächst werden die grundlegenden Prinzipien und die Architektur des RISC-V-Prozessors vermittelt, um eine solide Grundlage für die Weiterentwicklung und Anpassung des Systems zu schaffen.

Ein weiterer wichtiger Aspekt dieses Versuchs ist die Einführung der Programmierkonventionen für den RISC-V-Prozessor. Hierbei werden die spezifischen Anforderungen und Konventionen für die Programmierung auf dieser Architektur erläutert, um eine korrekte und effiziente Implementierung sicherzustellen.

Darüber hinaus werden Sie die Möglichkeit haben, die bereits im GdTI-Labor erstellte Arithmetisch-Logische Einheit (ALU) sowie das Constant_Package zu erweitern und anzupassen. Diese Erweiterungen sind notwendig, um den RISC-V-Prozessor effizient und gemäß den Spezifikationen zu implementieren.

Im weiteren Verlauf des Versuchs werden Sie einen generischen Multiplexer konzipieren, um die erforderlichen Daten zwischen verschiedenen Komponenten des Prozessors weiterzuleiten. Die Umsetzung dieses Multiplexers wird es Ihnen ermöglichen, die Funktionsweise von Prozessorarchitekturen besser zu verstehen und anzuwenden.

Abschließend steht die Konzeptionierung, Umsetzung und Validierung eines generischen Pipeline-Registers auf dem Programm. Pipeline-Register sind ein wesentlicher Bestandteil der modernen Prozessorarchitekturen, da sie den parallelen Betrieb von Prozessen ermöglichen und die Leistung des Prozessors erheblich steigern.

Durch die Bearbeitung dieser Ziele werden Sie nicht nur praktische Erfahrungen mit der RISC-V-Architektur sammeln, sondern auch Ihre Fähigkeiten in der Hardwarebeschreibung und -implementation vertiefen.

Vorbereitung

GHDL

Wie bereits im vorherigen GdTI-Projekt wird auch in diesem Projekt `ghdl` verwendet. Falls notwendig, erarbeiten Sie sich die grundlegenden Schritte für die Arbeit mit `ghdl`.

Im Folgenden finden Sie eine kurze Einführung in die Kommandos, die Sie – in der angegebenen Reihenfolge – für die typische Arbeit mit `ghdl` und `gtkwave` im Labor-Projekt benötigen. Eine ausführliche Dokumentation für `ghdl` finden Sie 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. Dies ist in den meisten Fällen sehr praktisch, kann jedoch zu unerwünschten Nebeneffekten führen, weshalb Sie diese leeren sollten:

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

Das `rm`-Kommando sorgt dafür, dass `ghdl` im aktuellen Verzeichnis vergisst, welche Entitäten bereits analysiert wurden. Danach folgt der eigentliche Ablauf, bei dem alle benötigten Entitäten (engl. *entities*) (i) analysiert, (ii) elaboriert und höchstwahrscheinlich (iii) ausgeführt sowie (iv) visualisiert werden.

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.

```
\paragraph{Elaborieren}
Die Option \texttt{-e} elaboriert bereits analysierte Quelltexte.

\begin{lstlisting}[style=sh]  ghdl -e <name>
```

Beim Elaborieren wird eine Liste aller im Design verwendeten Entitäten erstellt, die auf Architekturebene verknüpft werden. Die Top-Entity des Designs wird 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` führt eine Simulation durch.

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

`<name>` muss dabei dem Namen des vorher durchgeführten Elaborate-Schritts entsprechen. Die Option `--vcd=<name>` ist nur erforderlich, wenn Sie die Ergebnisse der Simulation mit `gtkwave` anschauen möchten; sie ist im Labor-Projekt jedoch in der Regel nützlich. Außerdem sollten Sie die Simulation sinnvoll zeitlich begrenzen – insbesondere, wenn Sie zur Visualisierung `gtkwave` verwenden möchten. Die Option `--stop-time` erlaubt dies. Im Beispiel wird nach $100\,ns$ simulierter Zeit abgebrochen.

Visualisieren Zur Visualisierung verwenden wir im Labor-Projekt das Programm *GTKWave*:

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

Dieses Kommando öffnet `GTKWave` mit den Daten aus dem Ausführungsschritt. In GTKWave müssen Sie typischerweise zuerst links oben eine Entity auswählen. Im linken unteren Fenster wählen Sie die gewünschten Signale der gewählten *Entity* aus. Danach haben Sie im rechten Fenster das Zeit-Diagramm der gewählten Signale. Möglicherweise müssen Sie noch die Zeitskalierung des Diagramms mit den Lupen-Icons anpassen.

Spezifikation RISC-V

Lesen Sie sich die Abschnitte 2.1 und 2.4 bis 2.6 der Referenz (RV32I Base Integer Instruction Set, Version 2.0) durch und erwerben Sie ein Grundverständnis der Befehlsstruktur. So sollten Sie in der Lage sein, den Überblick über die Befehlskodierungen (Seite 104) zu erklären und zu beschreiben.

ALU

Überprüfen Sie ihre Umsetzung der ALU aus dem GdTI-Labor auf ihre Funktionsfähigkeit. Falls notwendig, überarbeiten oder erstellen Sie die ALU neu.

Multiplexer und Register

Frischen Sie Ihre Grundkenntnisse über generische Entities sowie das Verhalten von Registern und Multiplexern auf. Überlegen Sie sich ein theoretisches Konzept zur Umsetzung eines generischen Registers und Multiplexers sowie deren Testbenches.

Aufgaben

Aufgabe 1: ALU

Überarbeiten Sie Ihre ALU und die zugehörige Testbench aus dem Labor-Projekt in GdTI gemäß den nachfolgenden Anforderungen.

- Erstellen Sie eine strukturierte Projektumgebung mit folgender Ordnerstruktur:
  - **Packages** – Enthält allgemeine Pakete, z. B. *Constant_Packages.vhdl*.
  - **Komponenten** – Enthält alle VHDL-Komponenten, z. B. den Unterordner *ALU*.
  - **Testbenches** – Enthält die Testumgebungen, z. B. den Unterordner *ALU*.
  - **Risc-V** – Beinhaltet die Implementierung des Prozessors.
- Kopieren Sie die Dateien der ALU aus GdTI in den Unterordner *Komponenten/ALU* und die dazugehörige Testbench in *Testbenches/ALU*.
- Erstellen Sie ein Bash-Skript im Verzeichnis *Skripts/ALU*, das folgende Schritte automatisiert:
  - Analyse der VHDL-Dateien (`ghdl -a`).
  - Elaboration (`ghdl -e`).
  - Simulation der ALU-Testbench mit Ausgabe der Wellenform (`ghdl -r`).

  Das Skript soll sicherstellen, dass die Simulation fehlerfrei durchläuft.
- Passen Sie Ihre ALU-Implementierung an die in *CodingConventions.vhdl* definierten Programmierkonventionen an.
- Verwenden Sie für alle Konstanten die aktualisierte Datei *Constant_Packages.vhdl*.
- Führen Sie eine vollständige Analyse, Elaboration und Simulation der angepassten ALU durch.
- Erstellen Sie eine ZIP-Datei (**Name_Vorname_Versuch1_Aufgabe1.zip**) mit Ihrer ALU-Implementierung und allen relevanten VHDL-Dateien, der Testbench-Datei, dem Bash-Skript und einer README-Datei welche eine kurze Anleitung zur Ausführung der Simulation beinhaltet. Laden Sie die ZIP-Datei in Moodle hoch.

Abnahmekriterien Die Abnahme erfolgt auf Basis folgender Punkte:

- Korrekte und lauffähige ALU-Implementierung.
- Einhaltung der Programmierkonventionen (*CodingConventions.vhdl*).
- Nutzung der vorgegebenen Konstanten aus *Constant_Packages.vhdl*.
- Simulation der ALU anhand einer Testbench mit definierten Testfällen.
- Die ZIP-Datei enthält alle erforderlichen Dateien und ist korrekt strukturiert.

Aufgabe 2: Pipeline-Register mit generischer Datenbreite .....

Ein Pipeline-Register ist ein wesentliches Element in einem Prozessor mit Pipelining. Pipelining ermöglicht eine effizientere Befehlsverarbeitung, indem die Ausführung eines Befehls in mehrere Stufen unterteilt wird. Jede Stufe bearbeitet dabei einen Teil des Befehls.

Das Pipeline-Register mit einer generischen Datenbreite `registerWidth` dient als Zwischenspeicher für die Daten während ihres Transfers zwischen den verschiedenen Verarbeitungsstufen der Pipeline. Es stellt sicher, dass die Daten konsistent an die nächste Stufe weitergegeben werden, wodurch die Synchronisierung innerhalb der Pipeline gewährleistet wird.

Das Verhalten des Registers wird durch einen synchronen Prozess beschrieben, der auf die steigende Flanke des Taktsignals `pi_clk` und das Rücksetzsignal `pi_rst` reagiert:

- Bei einem aktiven Rücksetzsignal wird das Register auf einen definierten Anfangswert zurückgesetzt.
- Andernfalls wird das Eingangssignal `pi_data` in das Register übernommen.
- Das Ausgangssignal `po_data` entspricht stets dem aktuellen Registerwert.

Aufgaben

- Erstellen Sie den Unterordner *<Projektordner>/Komponenten/Register*.
- Implementieren Sie die Entity `PipelineRegister` in der Datei *PipelineRegister.vhdl* gemäß der oben beschriebenen Spezifikation.
- Erstellen Sie eine Testbench `PipelineRegister_tb` in der Datei *PipelineRegister_tb.vhdl* im Ordner *<Projektordner>/Testbenches/Register*, die das Register für die Datenbreiten $5, 6, 8, 16, 32$ mit einer festgelegten Auswahl von Binärwerten testet.
- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Skripts/Register*, welches die Simulation der Testbench inklusive Analyse und Elaboration automatisch und fehlerfrei ausführt.
- Laden Sie die erstellten VHDL-Dateien, die Testbench und alle zugehörigen Dateien als ZIP-Archiv mit dem Namen: *Name_Vorname_Versuch1_Aufgabe2.zip* in Moodle hoch.

Abnahme Die Abnahme erfolgt nach folgenden Kriterien:

- Korrekte Implementierung und Funktionalität des Pipeline-Registers.
- Einhaltung der Programmierkonventionen (*CodingConventions.vhdl*).
- Nutzung der vordefinierten Konstanten aus *Constant_Packages.vhdl*.
- Erfolgreiche Simulation mit einer Testbench, die verschiedene Datenbreiten testet.
- Strukturierte und vollständige Bereitstellung der Dateien im ZIP-Archiv.

Aufgabe 3: 2-zu-1-Multiplexer mit generischer Datenbreite .....

Ein 2-zu-1-Multiplexer ist ein grundlegendes digitales Schaltungselement, das zwischen zwei Eingangssignalen umschaltet und eines davon als Ausgangssignal ausgibt. Die Auswahl des Eingangssignals erfolgt über ein Steuersignal.

Das Verhalten des Multiplexers kann durch die folgende logische Funktion beschrieben werden:

- Ist das Steuersignal `pi_sel` auf '0' gesetzt, wird das erste Eingangssignal `pi_first` mit der generischen Breite `dataWidth` auf den Ausgang `po_res` geschaltet.
- Andernfalls wird das zweite Eingangssignal `pi_second` auf den Ausgang weitergeleitet.

Aufgaben

- Erstellen Sie den Unterordner *<Projektordner>/Komponenten/Multiplexer*.
- Implementieren Sie die Entity `Multiplexer` in der Datei *gen_mux.vhdl* gemäß der beschriebenen Spezifikation.
- Entwickeln Sie eine Testbench `gen_mux_tb` in der Datei *gen_mux_tb.vhdl* im Unterordner *<Projektordner>/Testbenches/Multiplexer*, die den Multiplexer für die Datenbreiten $5, 6, 8, 16, 32$ mit festgelegten Binärwerten an den Eingängen testet.
- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Testbenches/Multiplexer*, das die Simulation der Multiplexer-Testbench inklusive Analyse und Elaboration automatisch und fehlerfrei ausführt.
- Laden Sie die erstellte VHDL-Implementierung, die Testbench und alle zugehörigen Dateien als ZIP-Archiv mit dem Namen: *Name_Vorname_Versuch1_Aufgabe3.zip* in Moodle hoch.

Abnahmekriterien Die Abnahme erfolgt auf Basis folgender Punkte:

- Korrekte und lauffähige Implementierung des Multiplexers.
- Einhaltung der Programmierkonventionen gemäß *CodingConventions.vhdl*.
- Nutzung der vordefinierten Konstanten aus *Constant_Packages.vhdl*.
- Erfolgreiche Simulation mit einer Testbench, die verschiedene Datenbreiten testet.
- Strukturierte und vollständige Bereitstellung der Dateien im ZIP-Archiv.