Rechnerarchitektur – Praktische Übungen

Übung 2: Architektur und Leistungsbewertung

Aufgabe 1.1: PCI Express (3 Punkte)

a) Vervollständigen Sie untenstehende Tabelle entsprechend dem bei PCI Express verwendeten Schichtenmodell:

Schicht	Aufgaben	Name der Übertragungseinheit

b) Was verbirgt sich hinter dem Begriff „split transactions“ und warum werden diese bei PCI Express verwendet?

Aufgabe 1.2: Von-Neumann-Architektur und Harvard-Architektur (5 Punkte)

a) Wie verhält sich die Aussage „Programme sind auch Daten!“ zur Von-Neumann-Architektur? Wie zur Harvard-Architektur?

b) Der Von-Neumann-Rechner arbeitet nach dem SISD Prinzip. Wie ist diese Arbeitsweise charakterisiert?

c) Stellen sie beide Modelle hinsichtlich ihrer Vor- und Nachteile gegenüber.

d) Ist es möglich, die Vorteile der Harvard Architektur auch für Interpreter und virtuelle Maschinen zu nutzen? Wenn ja, wie, wenn nein, warum nicht?

Aufgabe 1.3: Auswirkung von Architekturänderungen (6 Punkte, Klausur SS 2012)

Auf einem System, welches nur ein Programm ausführt, benötigen Gleitkommabefehle durchschnittlich 12 CPI. Für alle anderen Befehle wird ein Durchschnitt von 1,5 CPI angenommen. Das Programm besteht zu einem Drittel aus Gleitkommabefehlen.

a) Durch Einbau eines Gleitkomma-Koprozessors verringert sich die Ausführungszeit der Gleitkommabefehle auf 6 CPI. Berechnen Sie die hierdurch erzielte Gesamtbeschleunigung des Programms.

b) Wäre es günstiger, statt des Einbaus des Gleitkomma-Prozessors den Takt des Systems um den Faktor 1,5 zu erhöhen? Begründen Sie Ihre Antwort.

Aufgabe 1.4 (praktisch): Untersuchung des Wishbone-Bus des OR1200 OpenRISC Prozessors

Der sogenannte „Wishbone-Bus“ ist ein als Open-Source offengelegter Bus zur Verbindung von Teilen eines Chip-Designs. Diese Teile können beispielsweise unterschiedliche IP-Cores sein. („IP“ steht für Intellectual Property, d.h. ein IP-Core ist ein geistiges Eigentum eines Herstellers, der dieses wiederum herausgeben bzw. Lizenzen zur Nutzung verkaufen kann). Im (Hardware-)Open Source Bereich ist dieser Bus weitverbreitet, auch bei dem von uns im Praktikum genutzten SoC findet er Verwendung.

Unter anderem dienen beim OpenRISC 1200 Wishbone Verbindungen zur Anbindung des Speichers an die CPU. Hierbei wird aus Gründen, die wir erst im Kapitel „Speicher“ der Vorlesung genauer beleuchten werden, das Wishbone-Interface zum Speicher (bzw. den Speichern) nicht direkt an die CPU geführt sondern es wird jeweils noch ein Zwischenspeicher (Cache) für Programme (Instruction Cache, ICache) und Daten (Data Cache, DCache) dazwischengeschaltet. Diese binden dann jeweils über den Wishbone-Bus (WB I bzw. WB D) die Speicher an, siehe auch Abbildung 1.

Die Wishbone Spezifikation finden Sie unter http://cdn.opencores.org/downloads/wbspec_b3.pdf

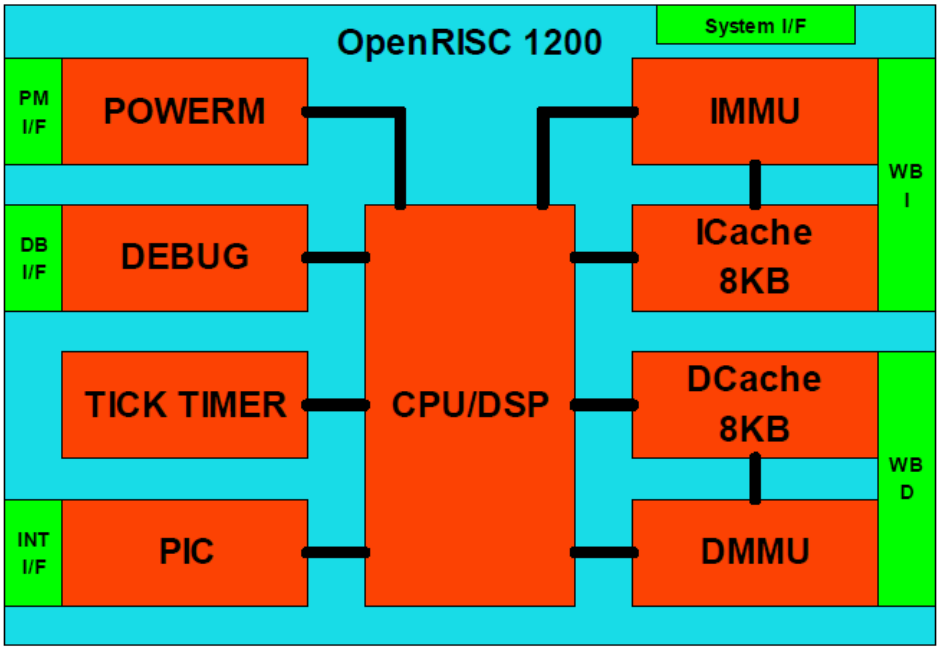

Abbildung 1: Wishbone (WB) Interfaces in der OpenRISC 1200 Architektur (Quelle: D. Lampret, J. Baxter: OpenRISC 1200 IP Core Specification v0.11)

Frage 1: Handelt es sich bei dieser Mikroarchitektur um eine von Neumann oder um eine Harvard-Architektur? Begründen Sie Ihre Antwort!

Teil 1: Verilog-Simulation und Untersuchung der Signale auf dem Wishbone-Bus

Wie das Gesamtsystem auch, so ist der Wishbone-Bus des OpenRISC in der Hardwarebeschreibungssprache Verilog beschrieben. Aus dieser Beschreibung kann dann mit Hilfe entsprechender Tools eine passende Beschreibung der elektrischen Schaltung erzeugt werden, unter anderem zur Herstellung des Chips, z.B. eines anwendungsspezifischen integrierten Schaltkreises (Application Specific Integrated Circuit, ASIC) oder eines programmierbaren Logikbausteins (Field Programmable Gate Array, FPGA). Im Praktikum verwenden wir hier das Tool Quartus II des Herstellers Altera, um den FPGA auf dem Experimentalmodul zu programmieren.

Im Folgenden sollen die auf den Wishbone-Bussen für Instruktionen und Daten (WB I und WB D) bei der Ausführung eines C-Programmes auf dem OpenRISC Prozessor vorhandenen Signale untersucht werden. Hierzu nutzen wir folgende Teile, die auf Ihrem virtuellen Laborrechner bereits installiert sind:

1. Beschreibung des OpenRISC 1200 Prozessors sowie des Wishbone Bus in Form von **Verilog Quellcode** (Hardwarebeschreibungssprache)
2. **Verilog Simulator Icarus**, welcher eine taktgenaue Simulation des OpenRISC Prozessors und des Wishbone-Busses ermöglicht
3. Darstellungsprogramm zur Visualisierung der unterschiedlichen Signale des Busses: **gtkwave**
4. **GNU Toolchain** bestehend aus Compiler (gcc), Debugger (gdb), etc. die uns die Übersetzung von Quellcode in C/C++ und Assembler zu Binärcode für den Simulator und das Experimentalboard ermöglicht

Um die Signale auf dem Wishbone Bus zu untersuchen, werden wir im ersten Schritt ein einfaches, vorgegebenes C-Programm auf dem simulierten OpenRISC 1200 Prozessor ausführen und die dabei auftretenden Signale in eine Datei schreiben. Gehen Sie dabei in folgenden Schritten vor:

- Starten Sie Ihren virtuellen Laborrechner und melden Sie sich unter der Kennung „openrisc“ an.
- Wechseln Sie in das Verzeichnis `~/soc-design/orpsocv2/sw/tests/or1200/sim` und sehen Sie sich dort den Quelltext des C-Programmes an, welches wir ausführen wollen. Diesen finden Sie in der Datei `or1200-cbasic.c` Zur Anzeige können Sie beispielsweise das Textprogramm `scite` verwenden, welches auf den virtuellen Maschinen vorinstalliert ist.
- Wechseln Sie in das Verzeichnis `~/soc-design/orpsocv2/sim/run` und führen Sie dort den Test durch Eingabe von

  ```
  make rtl-test TEST=or1200-cbasic VCD=1
  ```

  aus. (Die Bezeichnung „rtl“ steht für „register transfer level“, d.h. eine Verilog-Beschreibung des Designs auf dieser Ebene, vergl. erste Vorlesung. Das Flag VCD=1 bewirkt die Generierung einer Ausgabedatei, welche die Betrachtung der einzelnen Signale - unter anderem auf dem Wishbone Bus - erlaubt.)
- Jetzt sollen die Signale auf dem Wishbone Bus, welcher die Instruktionen bereitstellt, angezeigt werden. Dieses erreichen Sie durch Eingabe von

```
gtkwave ~/soc-design/orpsocv2/sim/out/or1200-cbasic.vcd ~/signals/wishbone_instruction_bus.sav
```

  Dieser Befehl lädt die im vorherigen Schritt erzeugte Ausgabedatei mit der Endung .vcd und wählt danach genau die Signale zur Darstellung aus, die in der Datei wishbone_instruction_bus.sav aufgelistet sind.

- Machen Sie sich ein wenig mit der Bedienung und der Nutzung von gtkwave vertraut, indem Sie einzelne Signale ansehen, die Zeitachse verschieben, etc.

Aufgabe: Klassifizierung der Wishbone Signale

- Lesen Sie die Bedeutung der einzelnen Signale des Wishbone-Busses in der Wishbone Spezifikation nach (URL siehe Anfangsteil der Aufgabenbeschreibung). Ordnen Sie dann jedes Signal einem der folgenden Busteile zu:

  Datenbus: _____

  Adressbus: _____

  Steuerbus: _____

- Wie breit ist der Adressbus insgesamt? Wie breit ist der Datenbus insgesamt?

  _____

- Betrachten Sie das Taktsignal des Wishbone Busses. Lesen Sie die Dauer einer Taktperiode ab und berechnen Sie daraus die Taktfrequenz des Busses.

  _____

Aufgabe: Anzeige des Wishbone Busses zur Übertragung von Daten zum Datencache

- Bisher haben wir nur den Wishbone-Bus zur Übertragung von Instruktionen zur CPU untersucht. Erweitern Sie die Anzeige nun so, dass Sie auch den Wishbone-Bus zur Übertragung von Daten zwischen CPU und Speicher sehen (WB D).
- Machen Sie einen Screenshot von einem Bereich, welcher das Lesen von Daten über den Datenbus zeigt.

Aufgabe: Betrachtung des Verilog-Quellcodes zum Arbiter des Wishbone Bus „WB I“

- Als Design-Software für den FPGA ist auf Ihrer virtuellen Maschine das Tool Altera Quartus vorinstalliert. Starten Sie dieses, indem Sie auf der Konsole „quartus“ eingeben und öffnen Sie dann das Gesamtprojekt mit dem Namen `ORDB2A.qpf` Dieses sollte - sofern Sie das Tool zum ersten Mal nutzen - bereits direkt nach dem Start in der Mitte des Arbeitsfensters zur Auswahl stehen.
- Wählen Sie im „Project Navigator“ unter dem Reiter „Hierarchy“ die Einheit „arbiter_ibus:arbiter_ibus0“ aus. Betrachten Sie den Verilog Quellcode. (Es wird nicht vorausgesetzt, dass Sie Verilog bereits können bzw. komplett verstehen - eine Verilog-Einführung folgt in einer der nächsten Praktika. An dieser Stelle sollen Sie noch nichts modifizieren, nur einen Eindruck von Verilog bekommen.)
- Wozu könnte diese Einheit dienen? Wer kommuniziert auf dem Wishbone Bus „WB I“ und wer tritt als Master bzw. Slave auf?

Teil 2: Eigenes C-Programm und dessen Auswirkungen auf die Signale auf den Wishbone-Bus

Als nächstes soll ein kurzes eigenes C-Programm entwickelt und auf der Verilog Simulation des OpenRISC 1200 ausgeführt werden.

- Entwickeln Sie ein C-Programm, welches bei seiner Ausführung den Wert 0x42 auf dem Wishbone Bus „WB D“ als Daten überträgt. Führen Sie das Programm aus und stellen Sie die Signale des Busses mit Hilfe von gtkwave dar. Dokumentieren Sie den erfolgreichen Ablauf Ihres Programmes mit Hilfe eines Screenshots.

  Hinweise:
  Um die Erstellung und den Test Ihres Programmes zu vereinfachen, können Sie die aus dem ersten Aufgabenteil bekannte Datei „or1200-cbasic.c“ als Vorlage nutzen. Kommentieren Sie den dort enthaltenen Testcode aus und fügen Sie Ihren eigenen Code ein. Danach können Sie - wie aus Teil 1 bekannt - Ihr Programm übersetzen, auf dem Simulator ausführen und die Signale untersuchen.

Für die Abnahme benötigen Sie:

- Antwort und Begründung zu Frage 1
- jeweils ein Screenshot der Signale auf den Wishbone Bussen für Instruktionen und Daten
- Berechnung der Taktfrequenz des Busses
- Klassifizierung der Signale (Zugehörigkeit zu Daten-, Adress- und Steuerbus)
- Eigenes C-Programm und Screenshot von zugehörigen Signalen