Rechnerarchitektur
Sommersemester 24

ertis embedded real-time systems ulm university institute

Laborprojekt – Versuch 6

Dieser Versuch beinhaltet zwei Teilbereiche: Im ersten Teil des Versuchs werden Sie die Struktur und Abhängigkeit vom Takt der einzelnen Phasen optimieren und auf die Notwendigkeit eines zweiten verzögerten Takts sowie das Triggern auf fallende Taktflanken verzichten. Des Weiteren trennen wir die Umsetzung von der Testbench selbst. Der zweite Teil des Versuchs beschäftigt sich mit der Umsetzung von bedingten Sprüngen. Für die Umsetzung wird die zur Verfügung gestellte Umsetzung so erweitert, dass über einen zusätzlichen Addierer die Sprungadresse bestimmt und die Entscheidung, ob gesprungen wird, über die ALU, ein zu ergänzendes Zero-Flag und eine zusätzliche Schaltung ausgewertet wird. Zudem erfordert ein durchgeführter bedingter Sprung, dass bereits geladene Befehle gelöscht werden. Dies werden Sie in diesem Versuch mithilfe von Multiplexern umsetzen.

Abschließend können Sie sich optional der Herausforderung einer Single-Cycle-Architektur stellen.

Ziele des 6. Versuchs

- Optimierung und Anpassung der aktuellen Architekturen (inkl. ALU mit Zero-Flag)
- Konzeptionierung, Umsetzung und Validierung der bedingten Sprungbefehle
- Konzeptionierung, Umsetzung und Validierung der Stalls
- Optional: Konzeptionierung, Umsetzung und Validierung einer Single-Cycle-Architektur

Vorbereitung

Spezifikation RISC-V (R/I/U-Befehle)

Eignen Sie sich den Ablauf, die Phasen und die Ansteuerung für die Ausführung von bedingten Sprungbefehlen basierend auf der Vorlesung und der Referenz (RV32I Base Integer Instruction Set, Version 2.0) an. Eignen Sie sich ein Grundverständnis so an, dass Sie die Befehlscodierung und die Ansteuerung mithilfe der Referenzkarten für jede Phase wiedergeben und erklären

können. Erarbeiten Sie sich ein Konzept zur Umsetzung der RISC-V-Sprungbefehle und klären Sie, was die Problematik bei bedingten Sprüngen ist.

GHDL-Standards

Lesen Sie sich in die GHDL-Optionen ein, insbesondere wie Sie Standards der Sprache einstellen. Ab sofort werden alle Komponenten und Testbenches nur noch mit VHDL 2008 genutzt.

Aufgaben

Aufgabe 1: Anpassung der Architektur

Die aktuelle Architektur ist zwar funktional, aber nicht optimal bezüglich ihres zeitlichen Verhaltens. Im Folgenden werden die genutzten Entitäten überarbeitet und mithilfe einer zur Verfügung gestellten Architektur der bisherigen Umsetzung und einer angepassten Testbench überprüft. Sichern Sie ggf. Ihre bisherigen Umsetzungen oder erweitern Sie die jeweilige Architektur, sodass Sie Ihre alten Umsetzungen weiter nutzen können.

- Ändern Sie die Architekturen der generischen Register (*gen_register*, *ControlWordRegister*) so um, dass diese nun auf die steigende Taktflanke sensitiv sind.
- Ändern Sie das Verhalten Ihres Decoders so, dass dieser nicht auf die Taktflanke, sondern nur auf die Änderung des Befehlswortes sensitiv reagiert.
- Passen Sie das Verhalten Ihrer ALU nun so an, dass diese ohne Prozess funktioniert. Die Auswahl und Ausgabe könnte auch hier über einen Ansatz mit *when ... select* erfolgen.
- Erweitern Sie Ihre ALU um einen zusätzlichen Ausgang *po_zero*, der zusätzlich zum Ergebnis der gewählten Operation angibt, ob das Ergebnis 0 ist (*po_zero='1'*).
- Testen Sie Ihre Anpassung und Erweiterung mit der zur Verfügung gestellten Testbench *my_alu_tb.vhdl* vollständig.

  ***Anmerkung**:* Sollten Sie Probleme haben, prüfen Sie Ihre Umsetzung der ALU, insbesondere der Vergleichsbefehle für Werte des Typs *unsigned*.
- Testen Sie Ihre kompletten Anpassungen und Erweiterungen abschließend mit der zur Verfügung gestellten Entity *riu_only_RISC_V* und der Testbench *riu_only_RISC_V_tb.vhdl* vollständig.

- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Testbenches/RISCV*, mit dem Sie die Simulation der RISC-V und die angepasste ALU inklusive der Analyse und Elaboration automatisch und erfolgreich ausführen können.

Aufgabe 2: Bedingte Sprünge

Die Umsetzung der bedingten Sprungbefehle wird in diesem Versuch über einen zusätzlichen Multiplexer realisiert, der zwischen dem Sprungziel und dem PC auswählt, sowie einen zusätzlichen Volladdierer, der das Sprungziel mithilfe des vorgehaltenen PC und dem B-Immediate berechnet, und eine zusätzliche Schaltung, die bestimmt, ob der Sprung ausgeführt wird. Betrachtet man die Sprungbefehle, die über das Steuersignal *IS_BRANCH* definiert werden, kann man erkennen, dass es sich bei *BEQ* und *BNE* um konträre Aussagen handelt. Diese können über ein zusätzliches Signal (*CMP_RESULT*) negiert werden. Damit kann eine Funktion/Schaltung umgesetzt werden, die angibt, ob gesprungen wird:

Ein Beispiel: Unter der Annahme, es handelt sich um *BEQ* und *po_zero* ist *true*, dann könnte mit *CMP_RESULT* = *false* gesprungen werden über die Funktion:

is_branch AND (po_zero XOR cmp_result)

Für *BNE* müsste dementsprechend *CMP_RESULT* mit *true* belegt werden.

- Erstellen Sie eine KOPIE der Datei *riu_only_RISC_V.vhdl* und benennen Sie diese in *riub_only_RISC_V.vhdl* um.

- Erweitern Sie Ihren Decoder um die bedingten Sprünge und setzen Sie Ihr Konzept nachvollziehbar um. Nutzen Sie *CMP_RESULT*, *IS_BRANCH* und das vorhandene Ergebnis der ALU.

- Erweitern Sie die EX-Phase des Systems um einen Volladdierer, welcher die Sprungadresse aus dem B-Immediate und dem PC berechnet und erweitern Sie das System zusätzlich um ein Pipelineregister für das Ergebnis zur Verwendung in der MEM-Phase.

- Erweitern Sie die EX-Phase um eine Schaltung, die aus *CMP_RESULT*, *IS_BRANCH* und dem LSB des Ergebnisses der ALU bestimmt, ob ein Sprung durchgeführt wird, und halten Sie das Ergebnis (im Folgenden als *B_SEL* benannt) bis zur MEM-Phase vor.

- Erweitern Sie Ihr System um einen Multiplexer vor dem PC-Register, mit *B_SEL* als Selektor aus der MEM-Phase, dem ursprünglichen Eingang des PC als ersten und dem Sprungziel aus der MEM-Phase als zweiten Dateneingang.

- Erweitern Sie ggf. noch den OP-Code der B-Befehle in Ihrem Entscheider nach dem Sign-Extender.

- Testen Sie abschließend Ihr Testsystem mit der Testbench *riub_only_RISC_V_1_tb.vhdl* erfolgreich.
- Passen Sie ggf. das Bash-Skript im Ordner *<Projektordner>/Testbenches/RISCV* an und führen Sie dieses erfolgreich aus.

Aufgabe 3: Flushing

Wenn ein bedingter Sprung ausgeführt wird, muss an diesem Punkt sichergestellt werden, dass alle anderen spekulativ ausgeführten Befehle gelöscht werden. Dazu können vor den Pipelineregistern Multiplexer geschaltet werden, die das System mit Nullen sozusagen spülen oder über den bereits vorhandenen Reset umgesetzt werden.

- Setzen Sie Ihr Konzept für das Flushing um und testen Sie Ihre Umsetzung mit der Testbench *riub_only_RISC_V_tb.vhdl* erfolgreich.
- Passen Sie ggf. das Bash-Skript im Ordner *<Projektordner>/Testbenches/RISCV* an und führen Sie dieses erfolgreich aus.

Aufgabe 4: Abgabe

Laden Sie die erstellte Ordnerstruktur mit den dazugehörigen Dateien als Zip-Datei unter dem Namen

Name_Vorname_Versuch6.zip in Moodle hoch.