Rechnerarchitektur
Sommersemester 25

ertis embedded real-time systems ulm university institute

Laborprojekt – Versuch 2

In modernen Prozessoren sind Registerdateien (Registerfiles) essenzielle Bausteine, die eine schnelle und effiziente Speicherung von temporären Daten während der Ausführung von Programmen ermöglichen. Die Architektur eines solchen Registerfiles hängt stark von der zugrunde liegenden Speichertechnologie und den spezifischen Anforderungen des Prozessors ab. In diesem Versuch werden wir schrittweise ein Registerfile entwickeln, das in einem RISC-V-Prozessor verwendet werden kann. Der Versuch beginnt mit der Implementierung eines einfachen linearen Speichers, der später zu einem multiportierten Speicher und schließlich zu einem vollständigen Registerfile für die RISC-V-Architektur ausgebaut wird.

Ziele

- **Konzeptionierung eines linearen Speichers:** Zu Beginn des Versuchs wird ein einfacher linearer Speicher (RAM) implementiert. Dieser Speicher stellt die Grundlage für alle weiteren Entwicklungen dar und dient als erster Schritt, um ein tieferes Verständnis für die Funktionsweise von Speicherkomponenten in einem Prozessor zu erlangen. Ziel ist es, den linearen RAM mit einer generischen Adresseingabe und Datenbreite zu entwickeln, der es ermöglicht, Daten zu speichern und darauf zuzugreifen.

- **Entwicklung eines Registerfiles für RISC-V:** Nachdem der lineare Speicher erfolgreich entwickelt wurde, erweitern Sie den RAM zu einem multiportierten Speicher in Form des spezialisierten Registerfile erweitert, das den spezifischen Anforderungen der RISC-V-Architektur entspricht. Das Registerfile wird so konzipiert, dass es nur eine begrenzte Anzahl von Registern (in der Regel 32) enthält und die RISC-V-Spezifikation für Registeradressen und deren Zugriff einhält. Besonderes Augenmerk liegt auf der korrekten Handhabung des Zero-Registers, das immer den Wert null zurückgeben muss und nicht überschreibbar ist. Diese Implementierung bildet das Herzstück des RISC-V-Prozessors, da die Register für die Zwischenspeicherung von Daten während der Programmausführung benötigt werden.

Vorbereitung

Registerfile

Eignen Sie sich die Spezifikation des Registerfiles der RISC-V-Spezifikation (Handbuch) so an, dass Sie die grundlegenden Eigenschaften und Funktionen des Registerfiles wiedergeben können. Ein Registerfile ist eine Sammlung von Registern, die zum Zwischenspeichern von Werten verwendet wird, die während der Ausführung von Instruktionen benötigt werden. Für die Umsetzung eines RISC-V-kompatiblen Registerfiles müssen Sie insbesondere die Adressierung und den Datentransfer zwischen den Registern verstehen. Die Spezifikationen für die RISC-V Architektur und die Details des Registerfiles finden Sie im offiziellen RISC-V Handbuch, insbesondere im Kapitel über die Register und deren Zugriff (*RISC-V User-Level ISA Specification*, Version 20191213, https://riscv.org/specifications/). Erarbeiten Sie ein Konzept zur Umsetzung des Registerfiles, das flexibel bezüglich der Wortbreite (*word_width*) und der Anzahl der Register (*reg_amount*) ist. Achten Sie darauf, dass die Umsetzung alle gängigen RISC-V Adressierungen und Zugriffsmechanismen berücksichtigt.

GHDL-Standards

Lesen Sie sich in die GHDL-Optionen ein, insbesondere, wie Sie die verschiedenen VHDL-Standards und Optionen für die Simulation einstellen können. GHDL ist ein leistungsstarker VHDL-Compiler und Simulator, der auch die Nutzung von VHDL 2008 ermöglicht. Die Einstellung auf VHDL 2008 ist erforderlich, da zukünftige Testbenches und Komponenten nur noch in dieser Version der Sprache geschrieben werden. Weitere Informationen zu GHDL und den verfügbaren Optionen finden Sie unter:

- GHDL-Dokumentation: https://ghdl.readthedocs.io/en/latest/
- VHDL 2008-Standard:
  IEEE Std 1076-2008, https://ieeexplore.ieee.org/document/4341515

Ab sofort werden alle Komponenten und Testbenches nur noch mit VHDL 2008 verwendet. Dies stellt sicher, dass fortschrittliche VHDL-Konstrukte wie *assertions*, *generate blocks*, und verbesserte Typen verwendet werden können, die die Codequalität und Simulationseffizienz erhöhen.

Zusatzdateien

Erarbeiten Sie sich die Inhalte der Ihnen zur Verfügung gestellten Zusatzdateien so, dass Sie diese anhand des Codes erklären können.

Aufgaben

Aufgabe 1: Linearer Speicher (Single-Port-RAM)

Der einfache lineare RAM (Random Access Memory) mit einer generischen Größe und einer 32-Bit-Datenbreite soll über ein Schreibsignal (`pi_we` für Write Enable) und eine Adresse (`pi_addr`) angesteuert werden. Die Datenbreite ist mit `word_width` und die Adressbreite mit `adr_width` zu definieren. Neben einem Takt- und Reset-Eingang (`pi_clk`,`pi_rst`) der RAM über einen Dateneingang und Datenausgang verfügen, welche bei einer positiven Taktflanke beschrieben und gelesen werden Der RAM kann bei einem positiven Taktzyklus beschrieben und gelesen werden. Setzen Sie bei der Aktivierung des Resets den Speicherinhalt auf Null zurücksetzt. Für die Abbildung des Speichers sollen sie in diesem Versuch sogenannte *types* nutzen:

```
1     type memory is array (0 to 3) of std_logic_vector(7 downto 0);
2     signal regs : memory := (others => (others => '0'));  -- Array
```

Das Code-Beispiel zeigt die Definition des *types* memory, welches aus 4 Datenwörtern mit 8 Bit Datenbreite besteht und als Signal genutzt wird.

- Erstellen Sie den Unterordner *<Projektordner>/Komponenten/RAM*.
- Erstellen Sie die Entity Single_Port_RAM in der Datei *Single_Port_RAM.vhdl* und setzen Sie das beschriebene Verhalten um.
- Erstellen Sie zum Testen Ihrer Umsetzung die Testbench-Entity *Single_Port_RAM_tb* in der Datei *Single_Port_RAM_tb.vhdl* im Unterordner *<Projektordner>/Testbenches/RAM*, welche den Single-Port-RAM für eine Datenbreite von 16-Bit testet.
- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Skripts/RAM*, mit dem Sie die Simulation der Register inklusive der Analyse und Elaboration automatisch und erfolgreich ausführen können.

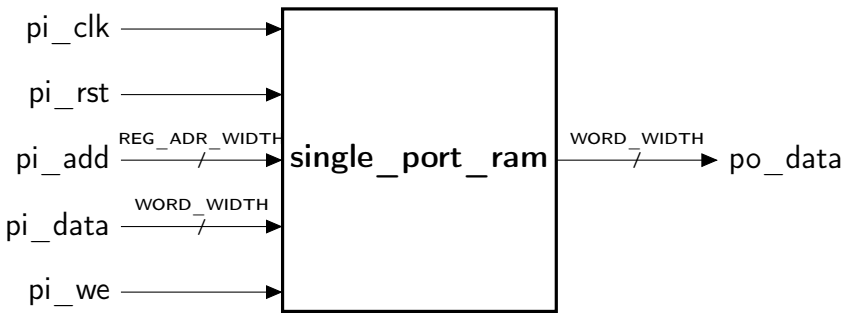

Abbildung 1: Single-Port-RAM

Aufgabe 2: Registerfile mit generischer Datenbreite

Ein Registerfile ist eine wichtige Komponente in Prozessoren wie dem RISC-V, die zur Zwischenspeicherung von Daten verwendet wird. In einem RISC-V-Prozessor besteht das Registerfile aus einer Sammlung von Registern, die in der Regel eine Datenbreite von 32 Bits haben und die mit einer 5 Bits breiten Adresse angesteuert werden. Diese Register werden verwendet, um temporäre Daten während der Ausführung von Instruktionen zu speichern.

Das Verhalten des Registerfiles besteht hauptsächlich aus dem Lesen und Schreiben von Daten. In einem Takt-Zyklus kann der Prozessor Daten aus den zwei Registern lesen, um sie für eine Operation zu verwenden, und in einem Register das Ergebnis einer Operation in ein anderes Register zu schreiben. Dies ermöglicht es dem Prozessor, Daten zwischen verschiedenen Teilen eines Programms zu übertragen und Zwischenergebnisse zu speichern. Das Registerfile soll über eine generische Datenbreite *word_width* (Standardwert: *WORD_WIDTH*), Adressbreite *adr_width* (Standardwert: *REG_ADR_WIDTH*) und Anzahl von Registern *reg_amount* (Standardwert: *2**REG_ADR_WIDTH*) verfügen. Das Zero-Register darf nicht überschrieben werden und soll immer 0 zurückgeben.

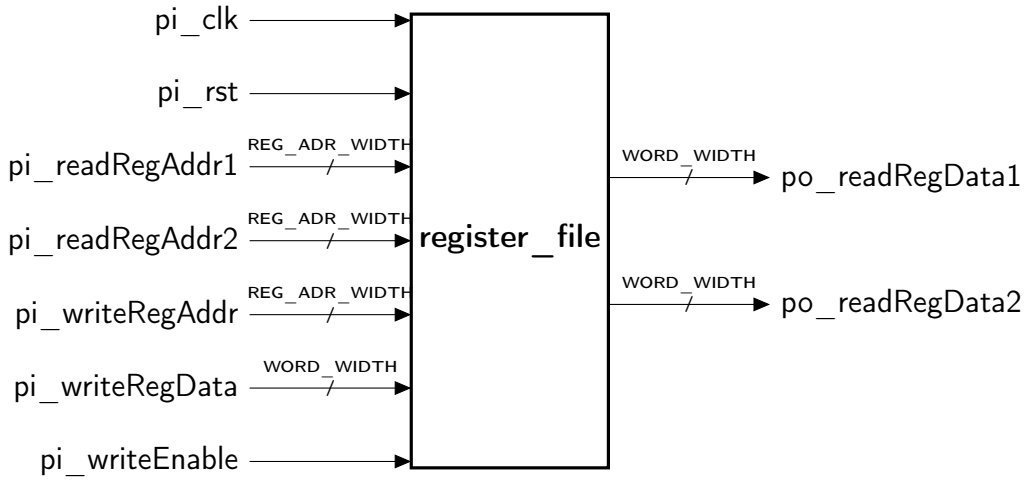

Abbildung 2: register_file

Unter der angepassten Verwendung des *types: reg_array* kann das Registerfile bei einem Reset auf einen festen Standardwert gesetzt werden (in der Regel alles auf null). Bei einer steigenden Flanke des Taktsignals werden die beiden Registeradressen (*pi_readRegAddr1*, *pi_readRegAddr2*) gelesen und auf *pi_readRegData1* und *pi_readRegData2* ausgegeben. Wenn bei der steigenden Flanke zusätzlich das *pi_writeEnable*-Signal auf $'1'$ gesetzt ist, werden die Daten des Dateneingang *pi_writeRegData* auf die Adresse *pi_writeRegAddr* geschrieben.

- Erstellen Sie den Unterordner *<Projektordner>/Komponenten/Registerfile*.

- Erstellen Sie die Entity register_file (basierend auf dem Multi-Port-RAM)in der Datei *register_file.vhdl* und setzen Sie das beschriebene Verhalten um.
- Testen Sie Ihre Umsetzung mit der Testbench-Entity *register_file_tb* in der Datei *register_file_tb.vhdl* im Unterordner *<Projektordner>/Testbenches/Registerfile*, welche das register_file für die Datenbreite $16, 32$ testet.
- Erstellen Sie ein Bash-Skript im Ordner *<Projektordner>/Skripts/Registerfile*, mit dem Sie die Simulation der Register-File-Testbench inklusive der Analyse und Elaboration automatisch und erfolgreich ausführen können.

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