Praktikum zur Vorlesung

Prozessorarchitektur

SS 2018

Übungsblatt 3

Entwurf einer ALU mit Registerbank

Abgabefrist: Mittwoch 02.05.2018, 10:00 Uhr

1.1. Einführung

In dieser Übung werden Sie einen einfachen ALU-Block und eine Registerbank entwerfen. Ihre Aufgabe ist es, VHDL Quellen und eine Simulationsdatei für die Zielmodule zu erstellen.

1.2. Design einer Registerbank (Register File)

Implementieren Sie eine Registerbank bestehend aus den acht 8-Bit Registern ***r0*** bis ***r7***, sowie einem Schreibport und zwei Leseports mit den in Abbildung 1.1 dargestellten Schnittstellen.

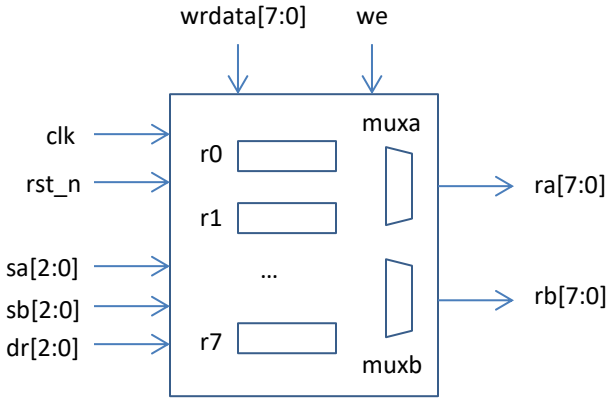

Abbildung 1.1. Registerbank.

Die Schnittstellen haben folgende Bedeutung:

- ***clk*** 1-Bit Taktsignal
- ***rst_n*** 1-Bit asynchrones Reset, low aktiv
- ***ra[7:0]*** 8-Bit Datenausgang (Leseport A)
- ***rb[7:0]*** 8-Bit Datenausgang (Leseport B)
- ***wrdata[7:0]*** 8-Bit Dateneingang (Schreibport)
- ***sa[2:0]*** Auswahl des Registers, dessen Wert am Leseport A ausgegeben wird
- ***sb[2:0]*** Auswahl des Registers, dessen Wert am Leseport B ausgegeben wird
- ***dr[2:0]*** Auswahl des Zielregisters, das mit dem Datum vom Schreibport beschrieben wird
- ***we*** 1-Bit Write-Enable-Signal, high-aktiv

Wenn das Reset-Signal aktiv ist, werden alle Register auf den Wert 0 gesetzt. Das Lesen von Daten aus den Registern erfolgt asynchron zum Takt. Das bedeutet, dass eine Änderung an ***sa*** bzw. ***sb*** sofort dazu führt, dass der Wert des ausgewählten Registers am zugehörigen Leseport anliegt. Das Schreiben der Daten in die Register erfolgt synchron mit der steigenden Taktflanke und nur dann, wenn das ***wr***-Signal aktiv ist. Das Steuersignal ***dr*** legt das zu schreibende Zielregister fest.

1. Erstellen Sie einen VHDL-Entwurf für die Registerbank mit dem Namen ***rf.vhd*** und eine Testbench ***rf_tb.vhd***. Erstellen Sie dafür ein neues Projekt (***lab3***) in Vivado.

2. Die Testbench-Simulation beginnt mit der Initialisierung der Register ***r0*** bis ***r7*** mit den folgenden Werten:

   r0 = x”F0”,
   r1 = x”F1”,
   …
   r7 = x”F7”

   Danach finden die folgenden Aktionen statt, wobei in jedem Taktzyklus eine Aktion ausgeführt wird. Zuerst wird ***sa*** auf ***0*** gesetzt. Dann erhält ***sb*** fortlaufend die Werte von ***1*** bis ***7***. Dann wird ***sb*** auf ***7*** gesetzt und danach erhält ***sa*** fortlaufend die Werte von ***1*** bis ***7***.

1.3. Design einer ALU

Implementieren Sie einen wie in Abbildung 1.2 illustriert einfachen kombinatorischen ALU-Block der acht arithmetische bzw. logische Operationen auf den 8-Bit Eingangsvektoren ***a*** und ***b*** ausführen kann. Das Ergebnis der Operation wird am 8-Bit Ausgang ***y*** ausgegeben. Die auszuführende ALU-Operation wird durch das Eingangssignal ***alu_op*** ausgewählt.

Operation	Opcode (alu_op)
y = a + b	000
y = a - b	001
y = a + 1	010
y = a – 1	011
y = a and b (bitweises und)	100
y = a or b (bitweises oder)	101
y = a xor b (bitweises xor)	110
y = not a	111

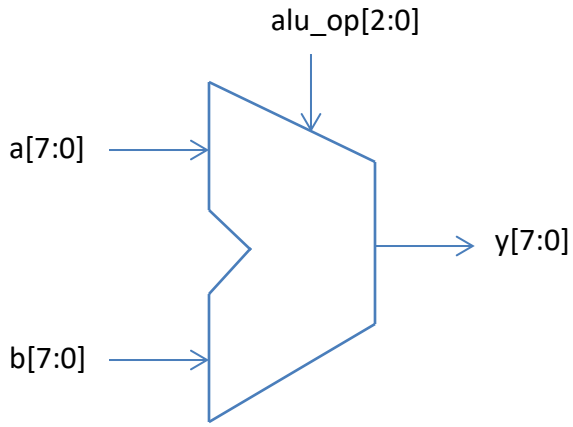

Abbildung 1.2. Einfacher ALU-Block.

1. Erstellen Sie eine VHDL-Datei mit dem Namen ***alu.vhd*** in Vivado. Bearbeiten Sie die Datei um die ALU-Funktionalität laut Spezifikation zu implementieren. Führen Sie die RTL-Analyse auf dem Modul durch.

2. Erstellen Sie eine Testbench mit dem Namen ***alu_tb.vhd*** mit folgendem Verhalten:

   - Definieren Sie die Taktquelle mit der Frequenz von 1 MHz.
   - Initialisieren Sie ***a*** mit x”F0” und ***b*** mit x”0F”.
   - Prüfen Sie alle ALU-Operationen (von 0 bis 7) mit den eingestellten Eingabewerten, wobei jede Operation nach einem Takt beendet ist.

- Nun setzen Sie ***alu_op*** auf x"00" und ändern den Wert von ***b*** auf x"F0". Testen Sie erneut alle ALU-Operationen, ohne die Eingabewerte zu ändern.
- Beenden Sie die Simulation.