4. Blatt	Fachgebiet Architektur eingebetteter Systeme **Rechnerorganisation Praktikum**	AES Embedded Systems Architecture WS2017/18

Ausgabe: 27. November 2017

Abgaben

Theorie	entfällt
Praxis	03. Dezember 2017
Rücksprache	04./05. Dezember 2017

Ab diesem Aufgabenblatt werden die Vorgaben nicht mehr bei ISIS hochgeladen. Verwenden Sie die Vorgaben, welche Sie im Gitlab-Repository finden. Aktualisieren Sie zum Erhalten der neuen Vorgaben ihr Repository durch die Ausführung des Kommandos `git pull vorgaben master` in dem Ordner, in dem sich ihr Repository befindet.

Aufgabe 1: Sign-Extension (2 Punkte)

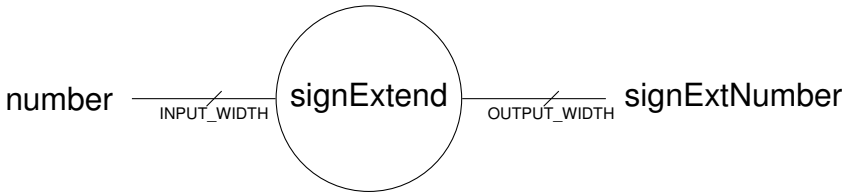

Abbildung 1: Entity signExtend

Name	Typ	in / out	Beschreibung
INPUT_WIDTH	generic	integer	Breite des Eingangs
OUTPUT_WIDTH	generic	integer	Breite des Ausgangs
number	in	signed(INPUT_WIDTH-1 downto 0)	zu erweiternde Zahl
signExtNumber	out	signed(OUTPUT_WIDTH-1 downto 0)	erweiterte Zahl

Das Ziel dieses Praktikums ist die Implementierung eines MIPS-kompatiblen Prozessors.
Diese MIPS-CPU verarbeitet Befehle, welche zum Teil Zahlen beinhalten.
Da die CPU eine 32-Bit-Architektur ist, sind Daten (und auch Befehle) 32 Bit breit.
Somit sind Zahlen, welche aus den Befehlen extrahiert werden, kleiner (16 Bit) als die zur Weiterverarbeitung benötigten 32 Bit.
Insofern müssen wir diese Zahlen um 16 Bit "strecken", d.h. nach vorne hin erweitern.
Bedenken Sie, dass alle Zahlendarstellungen in der MIPS-CPU Zweierkomplement-Darstellungen sind!

1. Implementieren Sie zur gegebenen `entity signExtend` die `architecture behavioral` in der Datei `signExtend.vhd`. Verwenden Sie dazu keine Funktionen aus der Library `numeric_std`.

2. Testen und verifizieren Sie Ihr Design mithilfe der vorgegebenen Testbench, indem Sie im Aufgabenordner das Kommando `make clean all` ausführen.

Aufgabe 2: Links-Shifter (2 Punkte)

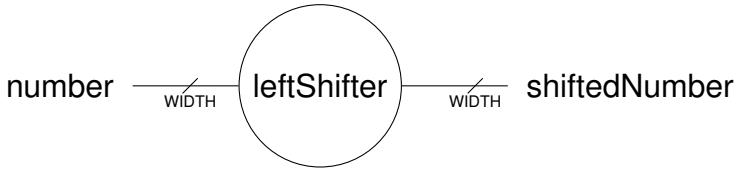

Abbildung 2: Entity leftShifter

Name	Typ	in / out	Beschreibung
WIDTH	generic	integer	Breite des Eingangs
SHIFT_AMOUNT	generic	integer	Weite des Shifts (Verschiebung in Bit)
number	in	std_logic_vector(WIDTH-1 downto 0)	zu shiftende Zahl
shiftedNumber	out	std_logic_vector(WIDTH-1 downto 0)	geshiftete Zahl

Da die MIPS-CPU eine 32-Bit-Architektur ist, werden aus dem Speicher immer 4 Byte geladen, um ein 32-Bit-Wort zu bilden. Somit werden die zwei niederwertigsten Bits einer Byte-Adresse nicht benötigt, da wir immer 4 Byte laden. Aus Optimierungsgründen werden diese Bits in manchen Situationen weggelassen. Wir müssen nun eine Schaltung entwerfen, welche diese Wort-Adresse mithilfe einer Schiebeoperation nach links zurück in eine Byte-Adresse umwandelt.

1. Implementieren Sie zur gegebenen `entity leftShifter` die `architecture behavioral` in der vorgegebenen Datei `leftShifter.vhd`. Verwenden Sie dazu keine Funktionen aus der Library `numeric_std`.
2. Testen und verifizieren Sie Ihr Design mithilfe der vorgegebenen Testbench, indem Sie im Aufgabenordner das Kommando `make clean all` ausführen.

Aufgabe 3: 32-Bit-Multiplizierer (6 Punkte)

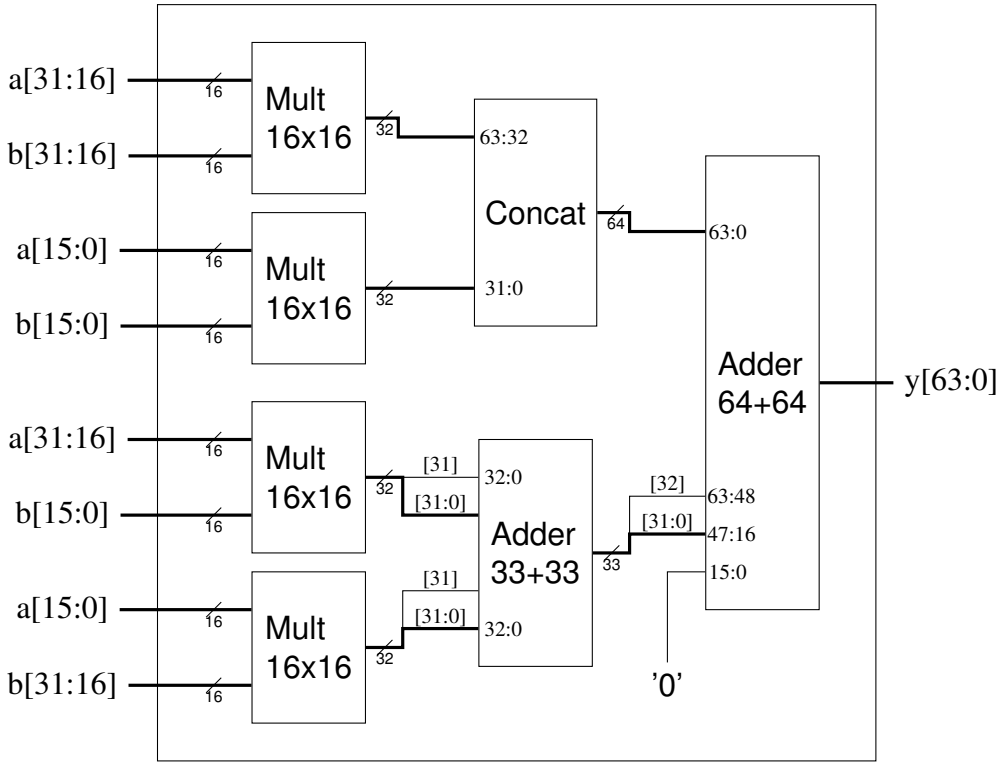

Abbildung 3: RTL 32 Bit-Multiplizierer

Für unsere 32 Bit-Architektur wird ein Multiplizierer benötigt, welcher zwei 32 Bit Operanden multipliziert. Das Ergebnis ist 64 Bit breit.
Zur Realisierung des Multiplizierers stehen bereits vorgegebene Multiplizierer mit 16 Bit Operanden zur Verfügung, welche wie in 3 gezeigt verschaltet werden.
Zur Realisierung der Funktionalität werden 4 16x16 Multiplizierer benötigt. Die Multiplizierer berechnen das Produkt der unteren Hälfte der beiden Eingangsbits, der oberen Hälften der beiden Eingangsbits und der Kombination von oberen und unteren Eingangsbits.
Das Ergebnis der Multiplikation der beiden oberen und unteren Eingangsbits können konkateniert werden, sodass aus den beiden 32 Bit Vektoren ein 64 Bit-Vektor entsteht. Die 32 Bit des Ergebnis der Multiplikation der oberen Hälften werden der oberen Hälfte des konkatenierten Vektors zugewiesen. Die unteren Bits des konkatenierten Vektors entsprechen dem Ergebnis der Multiplikation der unteren Hälften.
Die kombinierten Produkte werden miteinander addiert, wobei das oberste Bit der Produkte dupliziert werden muss, sodass zwei 33 Bit Zahlen addiert werden. Das 33 Bit Ergebnis der Addition wird verwendet um einen zweiten 64 Bit Vektor zu erzeugen, dessen Bits 47 bis 16 aus den unteren 32 Bit des Additionsergebnisses bestehen. Die verbleibenden unteren 16 Bits werden auf den Wert `'0'` gesetzt und alle Bits im Bereich 63 bis 48 werden auf das höchstwertigste Bit des Additonsergebnis gesetzt (Bit 32). Abschließend werden die beiden 64 Bit Vektoren addiert um das Endergebnis zu erhalten.
Beachten Sie bei der Implementierung die Kombinationen von vorzeichenlosen (`unsigned`) und vorzeichenbehafteten (`signed`) Zahlen. Die untere Hälfte der Eingabe (Bits 0 bis 15) können immer als vorzeichenlos betrachtet werden, während die vorderen Bits (16 bis 31) als vorzeichenbehaftet betrachtet werden müssen. Die Multiplikation einer vorzeichenlosen und vorzeichenbehafteten Zahl ergibt immer eine vorzeichenbehaftete Zahl. Für ihre Implementierung finden Sie bereits vorgegebene 16 Bit Multiplizierer, welche zwei vorzeichenlose, zwei vorzeichenbehaftete und ein vorzeichenbe-

hafteten und ein vorzeichenlosen Operanden multiplizieren.

1. Implementieren Sie die in 3 gezeigte Funktionalität in der `architecture structural` in der Datei `mult32x32.vhd`.
2. Testen und verifizieren Sie Ihr Design mithilfe der vorgegebenen Testbench, indem Sie im Aufgabenverzeichnis das Kommando `make clean all` aufrufen.

Literatur

[1] Mentor Graphics Corporation. *ModelSim SE Reference Manual*, 6.4a edition.

[2] Mentor Graphics Corporation. *ModelSim SE Tutorial*, 6.4a edition.

[3] Mentor Graphics Corporation. *ModelSim SE User's Manual*, 6.4a edition.