Einführung in

Computer Microsystems

Sommersemester 2010

Wolfgang Heenes

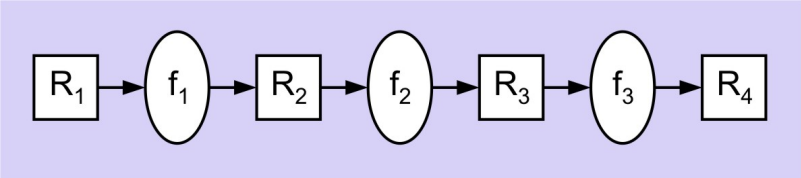
TECHNISCHE UNIVERSITÄT DARMSTADT

4. Aufgabenblatt

12.05.2010

Aufgabe 1: Register-Transfer-Logik

Gegeben ist folgende Pipeline in Register-Transfer-Logik:



Die kombinatorische Logik zwischen den Registern soll folgende Funktionen realisieren.

- f_1: verdoppeln
- f_2: plus 5
- f_3: quadrieren

Die Pipeline berechnet also $R_4 = (2R_1 + 5)^2$.

a) Beschreiben Sie die Pipeline in Verilog HDL. Die Funktionen sollen in Verilog HDL als `function` realisiert werden.

Ein Beispiel[1] für eine Funktion in Verilog HDL ist im Folgenden angegeben:

```
module arithmetic_unit (result, operand_1, operand_2);
  output[3:0] result;
  input [3:0] operand_1, operand_2;

  assign result = largest_operand (operand_1, operand_2);

  function [3:0] largest_operand;
  input [3:0] operand_1, operand_2;
   largest_operand = (operand_1 >= operand_2) ? operand_1 : operand_2;
  endfunction
endmodule
```

Die Funktion *largest_operand* bestimmt den größeren Operanden.

Die Register R1, R2, R3 und R4 sind jeweils 8-Bit breit. Überlaufen der Register kann vernachlässigt werden.

Das Register R1 soll mit einem Wert geladen werden können. Wenn ein Steuersignal `ldreg1` gesetzt ist, soll über den Port `in` von außen ein Wert in das Register R1 geladen werden.

Das Ergebnis der Berechnung soll über den Port `out` nach außen geführt werden.

Weisen Sie die korrekte Funktionsweise der Pipeline durch Simulation nach.

[1] vgl. auch Ciletti, Michael D.: Advanced Digital Design with the Verilog HDL. Prentice Hall, 2003. Seite 190

b) Die Pipeline aus Aufgabenteil a) soll um einen asynchronen Reset-Eingang `areset` erweitert werden, mit dem die Pipeline zurückgesetzt werden kann. Wird `areset` gesetzt, soll die Berechnung sofort gestoppt werden und solange keine neue Berechnung begonnen werden, bis `areset` wieder den Wert 0 annimmt.

c) Da die Pipeline einige Takte benötigt um das Ergebnis zu berechnen, liegen am Ausgang zwischenzeitlich falsche Werte an. Das Module aus Aufgabenteil b) soll deshalb um ein Signal `valid` erweitert werden. `valid` soll genau dann den Wert 1 haben, wenn der Wert an `out` ein korrektes Ergebnis eines geladenen Wertes ist.

Aufgabe 2: Zeitbehaftete Simulation

Betrachtet wird der 4-Bit Ripple-Carry Addierer aus dem 2. Aufgabenblatt, Aufgabe 4. Die technische Realisierung, z. B. auf einem FPGA, führt dazu, dass Signale sich nicht sofort ändern, da Leitungen und Lock-Up-Tabellen Verzögerungen hervorrufen. Diese Verzögerungen sollen im Folgenden an dem 4-Bit Ripple-Carry Addierer gezeigt werden. Als FPGA ist das **Spartan3E**-Device (vgl. 2. Aufgabenblatt, Aufgabe 2) zu verwenden.

```
module FourBitAdder(A, B, Sum);
    input [3:0] A;
    input [3:0] B;
    output [4:0] Sum;

    wire [2:0] carry;

    FullAdder Bit0 (.A(A[0]), .B(B[0]), .CarryIn(1'b0),
                    .Sum(Sum[0]), .CarryOut(carry[0]));

    FullAdder Bit1 (.A(A[1]), .B(B[1]), .CarryIn(carry[0]),
                    .Sum(Sum[1]), .CarryOut(carry[1]));

    FullAdder Bit2 (.A(A[2]), .B(B[2]), .CarryIn(carry[1]),
                    .Sum(Sum[2]), .CarryOut(carry[2]));

    FullAdder Bit3 (.A(A[3]), .B(B[3]), .CarryIn(carry[2]),
                    .Sum(Sum[3]), .CarryOut(Sum[4]));
endmodule
```

Neben der bekannten Verhaltenssimulation kann man mit der Entwicklungsumgebung auch zeitbehaftete Simulationen durchführen. Dazu wird in dem linken oberen Fenster unter *Sources for* der Punkt **Post-Route Simulation** ausgewählt. Unter *Processes* kann nun durch Auswahl von **Simulate Post-Place & Route Model** die zeitbehaftete Simulation gestartet werden.

a) Vergleichen Sie die Ergebnisse der Verhaltenssimulation mit den Ergebnissen der zeitbehafteten Simulation.

b) Wie lange dauert es, bis eine Änderung am Eingang eine Änderung am Ausgang bewirkt?