Übung zur Vorlesung Einführung in Computer Microsystems

Prof. Dr. A. Koch
Thorsten Wink

TECHNISCHE UNIVERSITÄT DARMSTADT

Sommersemester 11
Übungsblatt 4

Aufgabe 4.1 Synthese 1

Beschreiben Sie den Ablauf der Synthese.

Aufgabe 4.2 Synthese 2

Warum sind die Syntheseziele minimaler Flächenverbrauch und maximale Taktfrequenz (in der Regel) nicht gleichzeitig opimierbar?

Aufgabe 4.3 potenzielle Register

Geben Sie für alle der folgenden `reg`-Variablen an, ob sie bei der Synthese in Latches, Flip-Flops oder kombinatorische Logik übersetzt werden, oder ob sie entfallen. Begründen Sie ihre Antworten mit den Kriterien für potenzielle Register aus der Vorlesung.

```
module potential_regs(A, S1, S2, Y, SUM0, SUM1, SUM2, SUM3);
  input [3:0] A;
  input [7:0] S1, S2;
  output reg Y;
  output reg [7:0] SUM0, SUM1, SUM2, SUM3;

  reg [31:0] I;
  reg C0, C1 = 0, C2, C3 = 0;

  always @(*) begin
    C0 = 0;
    for (I = 0; I < 8; I = I + 1) begin
      {C0, SUM0[I]} = S1[I] + S2[I] + C0;
      if (A)
        {C1, SUM1[I]} = S1[I] + S2[I] + C1;
    end
    if (C1 == 1) C1 = 0;
  end

  always @(posedge C0) begin
    C2 = 0;
    for (I = 0; I < 8; I = I + 1) begin
      {C2, SUM2[I]} = S1[I] + S2[I] + C2;
      if (A)
        {C3, SUM3[I]} = S1[I] + S2[I] + C3;
    end
    if (C3 == 1) C3 = 0;
  end

  always @(A) begin
    Y = 0;
    if (A[0] == 1) Y = 1;
    else if (A[1] == 1) Y = 1;
    else if (A[2] == 1) Y = 1;
  end
endmodule
```

Übung zur Vorlesung Einführung in Computer Microsystems

Aufgabe 4.4 Fehlersuche

Gegeben ist folgendes Verilog-Modul, das einige Fehler enthält:

```
module faulty(A, B, Y, Z);
    input A;
    input [1:0] B;
    output reg Y;
    output [1:0] Z;

    reg A;

    always @(A)
      Z = A & B[0];

    always @(B)
      Z = B[1] & Y;

    always @(A)
      Y = B | A;

endmodule
```

Finden und beheben Sie die Fehler in der Verilog-Verhaltensbeschreibung. Vergleichen Sie dazu auch die Funktion des Moduls bei der Verhaltenssimulation mit der Post-Layout Simulation und den RTL-Schematics. Machen Sie Verbesserungsvorschläge und begründen Sie diese.

Wieviele Register (Flip-Flops) werden bei der Übersetzung des Moduls in Hardware erzeugt?

Aufgabe 4.5 generate

Was bewirkt der folgende Code?

```
module generate_example(

  imput  wire        read,write,
  input  wire [31:0] data_in,
  input  wire  [3:0] address,
  output wire [31:0] data_out
);
genvar i;

generate
  for (i=0; i < 4; i=i+1) begin : MEM
      memory U (read, write,
                data_in[(i*8)+7:(i*8)],
                address,data_out[(i*8)+7:(i*8)]);
  end
endgenerate
endmodule
```

Diese Hausaufgaben müssen bis 17.6.11, 18:00 über das Moodle-System abgegeben werden.

Hausaufgabe 4.1 Pipeline (10 Punkte)

Hier nochmal die Pipeline-Aufgabe aus der Klausur.

$result = (2 * A + B) * 13 - C$

a) Zeichnen Sie ein Übersichtsplan der Pipeline. Hieraus soll ersichtlich sein, welche Operationen in welchem Takt durchgeführt werden. Zeichnen Sie hierzu alle Register, Funktionen und Module ein.

b) Beschreiben Sie das Modul in Verilog. Dabei gelten folgende Bedingungen:

- Alle Datenleitungen sollen 32 Bit breit sein
- Die Multiplikation kann durch ein bereitgestelltes Modul mit der folgenden Schnittstelle durchgeführt werden:

```
module mult(
  input  wire        clk,          //Takt
  input  wire        reset,        //synchroner Reset
  input  wire [31:0] A,            //Eingang A
  input  wire [31:0] B,            //   "    B
  output wire [31:0] result        //Ergebnis
);
endmodule
```

  Dieses Modul benötigt 2 Taktzyklen, bis ein gültiges Ergebnis anliegt.
- Die Addition kann durch den + Operator durchgeführt werden. Sie benötigt 1 Takt. Diese Addition kann nur 2 Eingänge verarbeiten.
- Die Subtraktion kann durch den - Operator durchgeführt werden. Sie benötigt 1 Takt.
- Der $\star$ Operator darf NICHT verwendet werden.
- Achten Sie auf eine effiziente Implementierung

Der Code muss synthetisierbar sein. Kommentieren Sie Ihren Code.

```
module top(
  input  wire        clk,          //Takt
  input  wire        reset,        //synchroner Reset
  input  wire [31:0] A,            //Eingang A
  input  wire [31:0] B,            //   "    B
  input  wire [31:0] C,            //   "    C
  input  wire        input_valid,  //=1: es liegen gültige Eingänge an

  output wire [31:0] result,       //Ergebnis
  output wire        result_ready  //=1: das Ergebnis ist gültig
);

//Hier den eigenen Code einfügen

endmodule
```

Hier die Implementierung von `mult` (Diese Modul darf nicht verändert werden):

```
module mult(
  input  wire        clk,          //Takt
  input  wire        reset,        //synchroner Reset
  input  wire [31:0] A,            //Eingang A
  input  wire [31:0] B,            //   "    B
  output wire [31:0] result        //Ergebnis
);
reg [31:0] r1, r2;

always@(posedge clk) begin
  if(reset)begin
    r1 <= 0;
    r2 <= 0;
  end
  else begin
    r1 <= A * B;
    r2 <= r1;
  end
end

assign result = r2;

endmodule
```

c) Schreiben Sie eine Testbench für die Pipeline. Legen Sie dazu mindestens zwei verschiedene Eingabedatenpaare (ungleich 0)an und überprüfen Sie, ob das Ergebnis korrekt ist. Schreiben Sie das erwartete Ergebnis an der entsprechenden Stelle als Kommentar in den Code.

```
module tb()

  // Inputs
  reg clk;
  reg reset;
  reg [31:0] A;
  reg [31:0] B;
  reg [31:0] C;
  reg input_valid;

  // Outputs
  wire [31:0] result;
  wire result_ready;

  // Instantiate the Unit Under Test (UUT)
  top uut (
    .clk(clk),
    .reset(reset),
    .A(A),
    .B(B),
    .C(C),
    .input_valid(input_valid),
    .result(result),
    .result_ready(result_ready)
  );

  //Stimulus

endmodule
```