Technische Universität Darmstadt
FG Eingebettete Systeme und
ihre Anwendungen (ESA)

Prof. Dr. Andreas Koch
Holger Lange

06.06.2007

Einführung in Computer Microsystems
4. Aufgabenblatt
Sommersemester 2007

Aufgabe 1: 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
```

Lösung:

Y	Kombinatorische Logik, da vollständig wegen Zuweisung `Y = 0;` (nicht lokal). Eingang A[3] ist unbenutzt.
SUM0	Kombinatorische Logik, da vollständig (nicht lokal)
SUM1	unvollständig, nicht lokal, ergibt Latch
SUM2, SUM3	Flip-Flops, da flankengesteuert und nicht lokal
I	Nicht lokal, aber vollständig, temporäre Variable deren Wert nirgends benutzt wird, entfällt
C0	Vollständig wegen `C0 = 0;` (nicht lokal), wird zu Verdrahtung (= komb. Logik)
C1	Vollständig (triviales `if (C1 == 1) C1 = 0;` wird zu `C1 = 0;`. C1 ist dann auch lokal, weil C1 am Ende des `always`-Blocks immer 0 ist und es wegen seiner räumlichen Lokalität nirgendwo anders geschrieben wird, also auch am Anfang einer neuen Iteration des Blocks immer 0 ist), wird zu Verdrahtung (= komb. Logik)
C2	Wird zu Verdrahtung da lokal und vollständig, Flankensteuerung ist in diesem Fall irrelevant, da der letzte Wert von C2 nirgends benutzt wird (anderenfalls würde für C2 sowohl Verdrahtung als auch ein Flip-Flop erzeugt)
C3	Vollständig (triviales `if (C3 == 1) C3 = 0;` wird zu `C3 = 0;`. C3 ist dann auch lokal, weil C3 am Ende des `always`-Blocks immer 0 ist und es wegen seiner räumlichen Lokalität nirgendwo anders geschrieben wird, also auch am Anfang einer neuen Iteration des Blocks immer 0 ist), wird zu Verdrahtung (= komb. Logik); Flankensteuerung ist in diesem Fall irrelevant, da der letzte Wert von C3 nirgends benutzt wird (anderenfalls würde für C3 sowohl Verdrahtung als auch ein Flip-Flop erzeugt)

Es sei darauf hingewiesen, dass der Verilog-Code in der Aufgabenstellung *keinen* guten Modellierungsstil darstellt. Er dient vielmehr als abschreckendes Beispiel, wie ein auf den ersten Blick relativ harmlos und einfach wirkender, aber *schlecht* entworfener Verilog-Code zu völlig unübersichtlichen Syntheseergebnissen und damit kaum vorauszusagender Hardware-Implementierung mit potenziellen Fehlfunktionen führt. Daher der Rat, nicht unnötig von den in der Vorlesung vorgeschlagenen Modellierungsweisen für kombinatorische und synchrone Logik abzuweichen.

Aufgabe 2: BCD nach Binär Konverter

Implementieren Sie den folgenden Pseudo-Code für einen BCD nach Binär Konverter als Verhaltensbeschreibung in Verilog. Die Länge der BCD-Zahl sei 24 Bit. Verwenden Sie nur rein kombinatorische Logik und achten Sie darauf, dass bei der Synthese keine Latches enstehen. Testen Sie die Funktion ihres Moduls mit ISE. Bestimmen Sie durch Post-Layout Simulation die maximale Taktfrequenz Ihres Moduls nach der Synthese.

Pseudo-Code für BCD nach Binär Konverter:

1. **`bcd`** ← **`bcd_data_input`**
2. **`bin`** ← 0 (gleiche Bitbreite wie **`bcd`**)
3. Für **`count`** ← 1 bis Bitbreite von **`bcd`** iteriere:
   - 3.1. **`{bcd, bin}`** ← **`{bcd, bin} >>`** 1
   - 3.2. Für jede 4-Bit Folge (3…0, 7…4, …) in **`bcd`** iteriere
     Wenn die 4-Bit Folge größer 7, dann ziehe 3 von dieser Folge ab
4. **`bin`** enthält die konvertierte Zahl

Lösung:

1. Möglichkeit mit inneren **`for`**-Schleifen für **`k`** und bitweiser Zuweisung:

```
`timescale 1ns / 1ps
`define LENGTH 24
module bcd_to_bin(
  input [`LENGTH-1:0] BCD,
  output [`LENGTH-1:0] BIN
);

  reg [2*`LENGTH-1:0] bcd_concat_bin;
  reg [3:0] temp;
  integer i, j, k;

  assign BIN = bcd_concat_bin[`LENGTH-1:0];

  always @(BCD) begin
    bcd_concat_bin = {BCD, `LENGTH'b0};
    // Schieben und subtrahieren
    for (i = 0; i < `LENGTH; i = i + 1) begin
      bcd_concat_bin = bcd_concat_bin >> 1;
      // 4-Bit Folgen
      for (j = 0; j < `LENGTH/4; j = j + 1) begin
        // Extrahiere 4 Einzelbits
        for (k = 0; k < 4; k = k + 1)
          temp[k] = bcd_concat_bin[`LENGTH + j*4 + k];
        if (temp[3] == 1) // gröSSer als 7 (MSB = 1)
          temp = temp - 3;
        // Und wieder bitweise zurück
        for (k = 0; k < 4; k = k + 1)
          bcd_concat_bin[`LENGTH + j*4 + k] = temp[k];
```

```
      end
    end
  end
endmodule
```

2. Möglichkeit mit Verilog-2001 Operator (`[position -: length]` bzw. `[position +: length]`) für variablen Teilabgriff (part select). Die inneren `for`-Schleifen für `k` können dann entfallen, `k` dient nur noch als temporäre Hilfsvariable:

```
`timescale 1ns / 1ps
`define LENGTH 24
module bcd_to_bin(BCD, BIN);
  input [23:0] BCD;
  output [23:0] BIN;

  reg [2*`LENGTH-1:0] bcd_concat_bin;
  integer i, j, k;

  assign BIN = bcd_concat_bin[`LENGTH-1:0];

  always @(BCD) begin
    bcd_concat_bin = {BCD, `LENGTH'b0};
    // Schieben und subtrahieren
    for (i = 0; i < `LENGTH; i = i + 1) begin
      bcd_concat_bin = bcd_concat_bin >> 1;
      // 4-Bit Folgen
      for (j = 0; j < `LENGTH/4; j = j + 1) begin
        k = `LENGTH + j*4 + 3;
        // 4 Bit breite Teilabgriffe (part selects)
        if (bcd_concat_bin[k] == 1) // gröSSer als 7 (MSB = 1)
          bcd_concat_bin[k -: 4] = bcd_concat_bin[k -: 4] - 3;
      end
    end
  end
endmodule
```

Die maximale Taktfrequenz liegt bei ca. 77 MHz (13 ns Taktperiode).