Übung zur Vorlesung Technische Grundlagen der Informatik

Prof. Dr. Andreas Koch
Thorsten Wink

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 09/10

Übungsblatt 7 - Lösungsvorschlag

Die folgenden Aufgaben sollen in der Hardwarebeschreibungssprache Verilog bearbeitet werden. Zur Simulation können Sie XILINX ISE verwenden. Es ist als WebPack-Edition frei verfügbar und auch auf den Poolrechnern der RBG installiert. Dort kann es einfach mit dem Befehl *ise* gestartet werden. Ein Tutorial zur Installation und Benutzung finden Sie auf unserer Webseite.

Aufgabe 7.1 Zähler in Verilog

Wir betrachten noch einmal den Zähler aus der letzten Hausaufgabe.

```
module counter(
  input clk,
  output reg[3:0] count
  );

  initial count = 0;      //nur für die Simulation notwendig

  always @(posedge clk)   //getakteter Prozess
    count <= count + 1;   //um 1 erhöhen
endmodule
```

a) Der Zähler soll um einen Eingang für ein *enable*-Signal erweitert werden. Es wird nur gezählt, wenn *enable* 1 ist.

```
module counter(
  input clk,
  output reg[3:0] count,
  input enable
  );

  initial count = 0;

  always @(posedge clk)
    if (enable)                 //nur wenn enable=1 ist hochzählen
      count <= count + 1;
endmodule
```

b) Der Zähler soll mit einem synchronen Reset (*sreset*) erweitert werden, so dass mit steigende Taktflanke der Zähler auf 0 zurückgesetzt wird wenn *sreset*=1.

```
module counter(
  input clk,
  output reg[3:0] count,
  input enable,
  input sreset
  );
```

```
  //initial count = 0;              //nun unnötig, da mit dem Reset-Signal zurückgesetzt wird

  always @(posedge clk)
    if (sreset)                     //wenn sreset gesetzt ist, count auf 0 setzen
      count <= 0;
    else if (enable)                //sonst wie bisher
      count <= count + 1;
endmodule
```

c) Der Zähler soll mit einem asynchronen Reset (*areset*) erweitert werden, so dass unabhängig vom Takt der Zähler auf 0 zurückgesetzt wird.

```
module counter(
  input clk,
  output reg[3:0] count,
  input enable,
  input sreset,
  input areset
  );

  always @(posedge clk or posedge areset)       //asynchroner Reset, muss in die sensitivity-list
                                                //aufgenommen werden
    if (areset)                                 //asynchroner Reset
      count <=0;
    else if (sreset)                            //synchroner Reset
      count <= 0;
    else if(enable)                             //wie bisher
      count <= count + 1;
endmodule
```

d) Über eine Leitung *set* und einen 4-Bit-Dateneingang *value* soll der Zähler synchron auf den Wert von *value* gesetzt wird, sobald *set* 1 ist.

```
module counter(
  input clk,
  output reg[3:0] count,
  input enable,
  input sreset,
  input areset,
  input set,
  input [3:0] value
  );

  always @(posedge clk or posedge areset)
    if (areset)
      count <=0;
    else if (sreset)
      count <= 0;
    else if (set)                       //set-Vorgang
      count <= value;                   //value-Wert übernehmen
    else if (enable)
      count <= count + 1;
endmodule
```

e) Der Zähler soll nur bis zu einem Wert *max* zählen, der über einen zu definierenden Parameter gesetzt werden kann. Ist kein Parameter beim Modulaufruf angegeben, soll wie bisher ohne einen Schwellwert gezählt werden.

```
module counter(
  input clk,
  output reg[3:0] count,
  input enable,
  input sreset,
  input areset,
  input set,
  input [3:0] value
  );
  parameter max = 15;                       //Standardwert 15, maximaler Wert bei 4 Bit

  always @(posedge clk or posedge areset)
    if (areset)
      count <=0;
    else if (sreset)
      count <= 0;
    else if (set)
      count <= value;
    else if (enable)
      if (count == max)                     //Schwellenwert erreicht
        count <= 0;
      else
        count <= count + 1;
endmodule
```

f) Schreiben Sie einen Testrahmen für die letzte Teilaufgabe, so dass $max = 5$. Zu Beginn sollen alle Eingangssignal auf 0 liegen. Nach 7 ns soll ein synchroner Reset erfolgen, danach soll der Startwert 3 gesetzt werden und der Zähler gestartet werden. Geben Sie ein Timing-Diagramm an, bei dem die Werte für *clk* und *count* zu sehen sind.

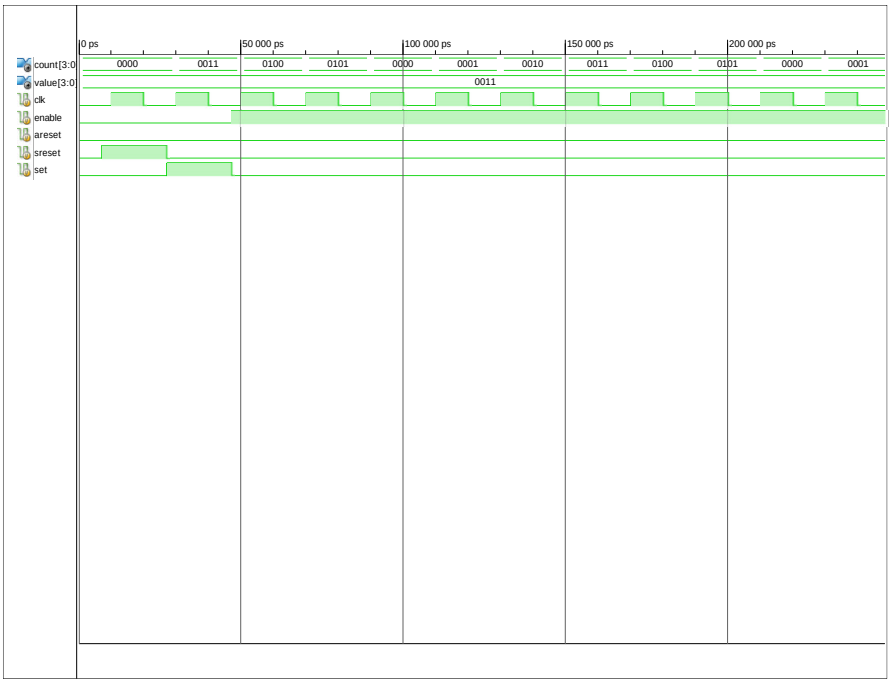
```
module countertest();

  reg clk;
  wire [3:0] count;
  reg enable,areset,sreset,set;
  wire [3:0] value = 4'b0011;

  //Instanziierung eines Counter mit den gewuenschten Eingaengen
  counter #(5) mycounter(
                         .clk(clk),
                         .count(count),
                         .enable(enable),
                         .areset(areset),
                         .sreset(sreset),
                         .set(set),
                         .value(value)
                         );
  //Taktgenerierung
  initial clk = 0;
  always
    #5 clk = ~clk;

  //Stimulus
  initial begin
    enable = 0;
    areset = 0;
    sreset = 0;
    set = 0;
    #7;
```

```
        sreset = 1;
        #20;
        sreset = 0;
        set = 1;
        #20;
        set = 0;
        enable = 1;
      end
    endmodule
```



Aufgabe 7.2 Paritätsbit

Schreiben Sie ein Verilog-Modul, das zu einem übergebenen Bitstring von n Bits ein Paritätsbit hinten anhängt, welches 1 ist, wenn die Anzahl der Einsen im Bitstring ungerade ist, und das 0 ist, wenn die Anzahl der Einsen im Bitstring gerade ist. Der so entstandene neue Bitstring soll der Ausgang des Moduls sein. Wird kein Parameter angegeben, so soll die Bitbreite des Ausgangs 9 Bit betragen.

```
module parity
  #(parameter n = 8)  //Parameterdeklaration im Modulkopf, damit er schon in der Port-Liste
                      //verwendet werden kann
   (
     input [n-1:0] in,
     output[n:0]   out
    );

  wire parity;
  assign parity = ^in;         //Reduktion mit xor
  assign out = {in, parity};  //Ausgang = Eingang mit angehaengtem Paritaetsbit
endmodule
```

Aufgabe 7.3 Fragen

a) Wie können Werte an Wire-Variablen zugewiesen werden? Geben Sie ein Beispiel an. Können Wires Werte speichern?

`wire s; assign s=1;`
`wire s = a & b;`
Wires können Werte nur transportieren, nicht speichern. Sie werden zur Verbindung von Modulen und Gattern verwendet.

b) Wie können Zuweisungen an Signale verzögert werden?
Verzögerungen können z. B. mit `assign #15 wire1 = x & y` angegeben werden.

c) Wie unterscheiden sich `initial` und `always`?
Ein `Initial` Statement wird nur einmalig ausgeführt. `Always` Statements werden in einer Endlosschleife ausgeführt.

Hausaufgabe 7.1 Multiplexer in Verilog

Beschreiben Sie einen 8:1 Multiplexer in Verilog. Zur Auswahl aus den Eingängen IN$x, x \in \{0,...,7\}$ soll der Steuereingang S[2:0] dienen. Die Bitbreite der Eingänge und des Ausgangs sollen parametrisierbar sein, falls keine Bitbreite angegeben wird soll der Standardwert 4 verwendet werden.

```
module mux8
  #(parameter width=4)   //Bitbreite
  (
  input [width-1:0] IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7,
  input [2:0]       S,
  output[width-1:0] OUT
  );

  always@(*) begin //wird immer ausgewertet, kombinatorische Logik
    case(S)
      0: OUT = IN0;
      1: OUT = IN1;
      2: OUT = IN2;
      3: OUT = IN3;
      4: OUT = IN4;
      5: OUT = IN5;
      6: OUT = IN0;
      7: OUT = IN7;
    endcase
  end
endmodule
```

Eine weitere Lösung ist mittels `assign`-Statements möglich.

Hausaufgabe 7.2 Multiplexer in Verilog (2)

Schreiben Sie ein strukturelles Verilog-Modul, welches die Funktion $Y = A\bar{B} + \bar{B}\bar{C} + \bar{A}BC$ realisiert. Verwenden Sie dazu nur den Multiplexer aus der vorherigen Aufgabe.

```
module muxfunction(
  input A, B, C,
  output Y
  );

  mux8 #(.width(1)) mymux(    //Instanz des Multiplexers, 1Bit breit
   .IN0(1),                   //Werte aus Wahrheitstabelle der Funktion
   .IN1(0),
   .IN2(0),
   .IN3(1),
   .IN4(1),
   .IN5(1),
   .IN6(0),
```

```
    .IN7(0),
    .OUT(Y),
    .S({A,B,C})
    );
```

Plagiarismus

Der Fachbereich Informatik misst der Einhaltung der Grundregeln der wissenschaftlichen Ethik großen Wert bei. Zu diesen gehört auch die strikte Verfolgung von Plagiarismus. Weitere Infos unter www.informatik.tu-darmstadt.de/plagiarism