Digitaltechnik
Wintersemester 2017/2018
11. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

Andreas Engel, Raad Bahmani **KW03**

Die Präsenzübungen werden in Kleingruppen während der wöchentlichen Übungsstunde bearbeitet. Bei Fragen hilft Ihnen Ihr Tutor gerne weiter. Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 11.1 Zeitverhalten sequentieller Beschreibungen [15 min]

Ermitteln Sie das Schaltverhalten der nachfolgenden Signale für die ersten 100 ns. Bedenken Sie dabei, dass bei einfachen **always** Blöcken (im Gegensatz zu **always_comb**) die Signalinitialisierung nicht als Signaländerung interpretiert wird.

seq/timing.sv

```
1  `timescale 1 ns / 10 ps
2  module timing;
3    localparam x = 2;
4
5    logic [2:0] a = 0;
6    always begin if (!a[0]) #10; else #(3+x); a <= a+1; end
7
8    logic b, c, d, e, f, g, h, i, j;
9    assign b = ^a;
10   always              begin             c   = b;     d   = c; @(negedge a[0]); end
11   always              begin             e   = b; #a; f   = e; @(posedge a[0]); end
12   always @(negedge b) begin     g <= c; h <= g;                               end
13   always @(f|d)       begin #2; i   = e; j <= i;                              end
14 endmodule
```

Übung 11.2 Strukturierung von sequentiellen Beschreibungen [15 min]

Wandeln Sie folgende kontrollflusslastige Beschreibung eines sequentiellen 4 bit Multiplizierers in eine äquivalente Beschreibung um, welche dessen Umsetzung als Register-Transfer-Logik besser erkennen lässt. Verfolgen Sie dafür folgende Grundregeln:

- nur ein Signal pro **`always_ff`** Block (beschreibt ein Register)
- kombinatorische Logik *vollständig* als nebenläufige Zuweisungen (beschreibt die Transfer-Logik)

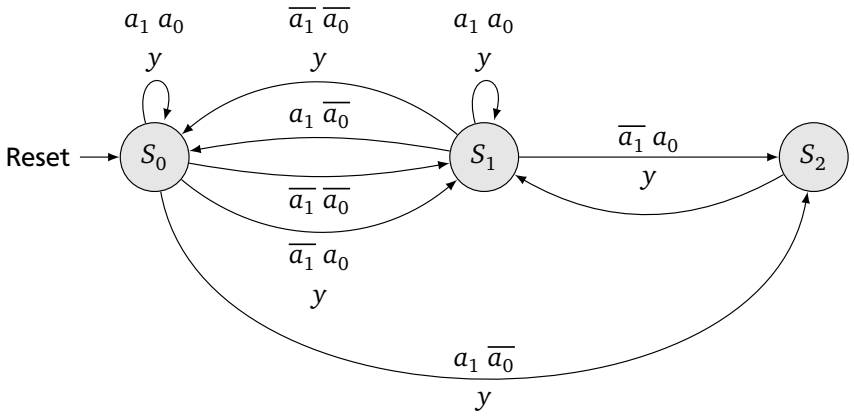
arith/mul/sequential.sv

```
module mul (input  logic CLK, RST, START, input  logic [3:0] A, B,
            output logic DONE,            output logic [7:0] Y);

  logic [2:0] n;
  logic [3:0] b;
  logic [7:0] a, p;

  always_ff @(posedge CLK) begin
    if (RST) begin
      {n, a, b, p, DONE, Y} <= 0;
    end else if (START) begin
      p <= 0; a <= A; b <= B; n <= 4; DONE <= 0;
    end else if (n > 1) begin
      if (b[0]) p <= p + a;
      a <= a << 1; b <= b >> 1; n <= n-1;
    end else if (n == 1) begin
      Y <= b[0] ? p + a : p; n <= 0; DONE <= 1;
    end else begin
      {DONE, Y} <= 0;
    end
  end
endmodule
```

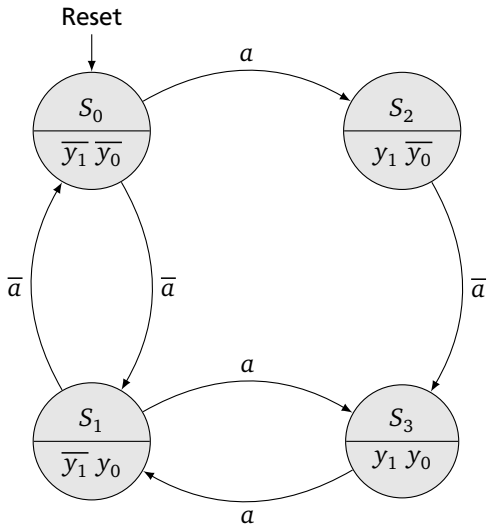
Übung 11.3 Robuste Endliche Automaten [20 min]

Implementieren Sie folgende endliche Automaten in SystemVerilog. Wenn eines der Eingangsbits `1'bz` oder `1'bx` ist, soll der Automat in den Startzustand wechseln und dabei kein Ausgangsbit auf 1 setzen. Verwenden Sie den === Operator zum Vergleich zwischen Ausdrücken vierwertiger Logik, da ein Vergleich mit `s == 1'bx` für alle Werte von `s` immer `1'bx` ergibt und dieses als logisch falsch interpretiert wird.

a)



b)



Übung 11.4 RCA-basierter Zähler [30 min]

Übung 11.4.1 Generischer Ripple-Carry Adder (RCA)

Implementieren Sie den RCA aus Übung 10.5.3 für eine generische Bitbreite mit dem Parameter `WIDTH`. Halb- und Volladdierer sollen aus den Übungen 10.5.1 und 10.5.2 übernommen werden.

Übung 11.4.2 Zähler

Verwenden Sie den generischen RCA zur Implementierung eines 10 bit Zählers mit folgender Schnittstelle:

arith/counter.sv

```
1 `timescale 1 ns / 10 ps
2 module counter(input  logic       CLK, // Taktsignal
3                                   RST, // synchrones Reset, high-active
4                                   INC, // Zähler erhöhen,   high-active
5                output logic [9:0] VAL);// aktueller Zählerwert
```

Das Zählerregister soll eine Ausgabeverzögerung von $t_{ccq} = t_{pcq} = 2\,\text{ns}$ haben.

Übung 11.4.3 Testbench

Implementieren Sie eine selbstüberprüfende Testbench für den Zähler aus Übung 11.4.2. Diese soll einen 20 MHz Takt an den Zähler anlegen und das funktionale Verhalten des Zählers überprüfen.