Digitaltechnik
Wintersemester 2017/2018
10. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

Andreas Engel, Raad Bahmani **LÖSUNGSVORSCHLAG** **KW02**

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 10.1 Ausführungsreihenfolge [5 min]

Klammern Sie folgende Ausdrücke entsprechend der Reihenfolge der Evaluation von Teilausdrücken. Bei gleicher Präzedenz werden Operatoren von links nach rechts ausgewertet.

a) `A != B & C` ⇒ `(A != B) & C`

b) `A >>> B >> C` ⇒ `(A >>> B) >> C`

c) `A >> B > C <<< D` ⇒ `(A >> B) > (C <<< D)`

d) `A >> B + C << D` ⇒ `(A >> (B + C)) << D`

e) `A && & B & C && D` ⇒ `(A && ((&B) & C)) && D`

Übung 10.2 Verhaltens- und Strukturbeschreibung [5 min]

a) Welches der folgenden Module verwendet eine Verhaltensbeschreibung, welches eine Strukturbeschreibung?

```
1 module m1 (input logic  A, B, C, output logic Y);
2   logic n1, n2;
3   or2  i1 (A, B, n1);
4   inv  i2 (.A(n1), .Y(n2));
5   xor2 i3 (n2, C, Y);
6 endmodule
```

Strukturbeschreibung:
Einbinden und Verdrahtung von Submodulen

```
1 module m2 (input logic A, B, C, D, output logic Z);
2   assign Z = A & ~B | ~D & C;
3 endmodule
```

Verhaltensbeschreibung:
Ausgang als kombinatorische Verschaltung der Eingänge

```
1 module m3 (input logic A, B, C, D, output logic Y);
2   logic  n1, n2, n3;
3   assign n1 =  A |  B;
4   assign n2 =  B &  C;
5   assign n3 = n2 ~^ D;
6   assign Y  = n1 ~| n3;
7 endmodule
```

Verhaltensbeschreibung:
Ausgang als kombinatorische Verschaltung der Eingänge

```
1 module m4 (input logic A, B, C, D, output logic Y);
2   nand2 i (.A(A | B), .B(((B & C) ~^ D)), .Y(Y));
3 endmodule
```

Struktur- und Verhaltensbeschreibung:
Beide Konzepte sind frei kombinierbar.

b) Warum dienen beide Beschreibungsarten der Abstraktion und damit der Beherrschung der Schaltungskomplexität?

Die Strukturbeschreibung führt zu einer Modularhierarchie, bei der komplexere Schaltungen aus einfacheren Schaltungen zusammengesetzt werden können. Außerdem kann die Regularität von Schaltungen durch das mehrfache Einbinden desselben Submoduls ausgenutzt werden.

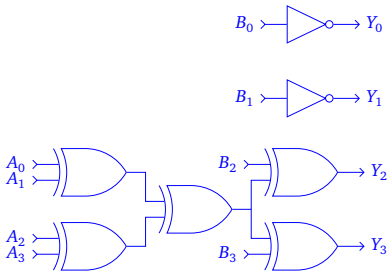
Ohne die abstrakte Verhaltensbeschreibung müsste jede strukturelle Modulhierarchie die Basisgatter der Zieltechnologie auf der untersten Ebene verwenden. Die automatische Abbildung der abstrakten (also Zieltechnologie-unabhängigen) Verhaltensbeschreibung auf eine konkrete Zieltechnologie erhöht also die Wiederverwendbarkeit bzw. Portierbarkeit von Hardwarebeschreibungen.

Übung 10.3 Synthese [20 min]

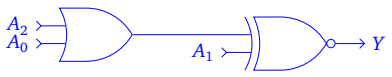
Zeichnen Sie die von folgenden Modulen (`m1` bis `m4`) beschriebenen kombinatorischen Schaltungen.

```
module m1 (input  logic A, B,
           output logic Y);

  assign Y = A | B ? A : B;

endmodule
```

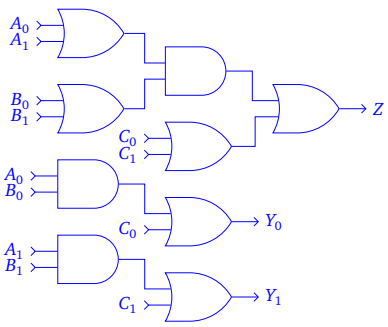

```
module m2 (input  logic [3:0] A, B,
           output logic [3:0] Y);

  assign Y = {{2{^A}},2'h3} ^ B;

endmodule
```




```
module hlp(input  logic A, B,
           output logic Y);
  assign Y = | {A,B,B,A};
endmodule

module m3 (input  logic [3:0] A,
           output logic       Y);

  logic [1:0] s;
  logic [3:0] n;
  hlp h1(n[3], n[1], s[1]);
  hlp h2(.Y(s[0]), .B(n[0]), .A(s[1]));
  assign Y = s[0] ~^ n[2];
  assign n = A << 1;

endmodule
```




```
module m4 (input  logic [1:0] A,B,C,
           output logic [1:0] Y,
           output logic       Z);

  assign Y = A &  B |  C;
  assign Z = A && B || C;

endmodule
```



Übung 10.4 Arithmetisch Logische Einheit (ALU) [20 min]

Eine ALU ist eine (kombinatorische oder sequentielle) Schaltung, welche ein Ergebnis aus mehreren Operanden berechnet. Die auszuführende Operation kann dabei über einen Selektionssignal (operation code) ausgewählt werden. Die ALU bildet damit das Herzstück der meisten Rechnerarchitekturen (siehe Vorlesung Rechnerorganisation).

Übung 10.4.1 Modul-Schnittstelle

Beschreiben Sie die Modul-Schnittstelle einer ALU mit zwei 32 bit Eingängen (`A` und `B`), einem 3 bit Selektionssignal (`OPC`), und einem 32 bit Ergebnis (`R`) mit SystemVerilog.

```
module alu (input logic [31:0] A, B, input logic [2:0] OPC, output logic [31:0] R);
endmodule
```

Übung 10.4.2 Operator-Implementierung

Die ALU soll eine Addition von `A` und `B` und einen arithmetischen Rechtsshift von `A` um `B` Stellen durchführen können. Realisieren Sie diese Operationen als SystemVerilog Module.

alu/operators.sv

```
module add32 (input logic [31:0] A, B, output logic [31:0] R);
  assign R = A + B;
endmodule

module sra32 (input logic [31:0] A, B, output logic [31:0] R);
  assign R = A >>> B;
endmodule
```

Übung 10.4.3 Operator-Auswahl

Implementieren Sie die ALU in SystemVerilog basierend auf den bisher beschriebenen Modulen. Für `OPC == 0` soll die Addition und für `OPC == 1` der arithmetische Rechtsshift ausgegeben werden. Für alle anderen Werte des Selektionssignals soll die ALU den Wert 0 ausgeben.

alu/base.sv

```
module alu (input logic [31:0] A, B, input logic [2:0] OPC, output logic [31:0] R);

  logic [31:0] R0, R1;

  add32 op0 (A, B, R0);
  sra32 op1 (A, B, R1);

  assign R = (OPC == 3'd0) ? R0 :
             (OPC == 3'd1) ? R1 :
             32'd0;
endmodule
```

Übung 10.4.4 Operator-Erweiterung

Erweitern Sie die ALU um eine weitere Operation. Für `OPC == 2` soll `A + B + 1` berechnet werden.

alu/extended.sv

```
module alu (input logic [31:0] A, B, input logic [2:0] OPC, output logic [31:0] R);

  logic [31:0] R0, R1, R2;
  add32 op0 (A,  B,     R0);
  sra32 op1 (A,  B,     R1);
  add32 op2 (R0, 32'd1, R2);
```

```
7
8     assign R = (OPC == 3'd0) ? R0 :
9                (OPC == 3'd1) ? R1 :
10               (OPC == 3'd2) ? R2 :
11               32'd0;
12  endmodule
```

Übung 10.5 Ripple-Carry Adder (RCA) [30 min]

Ein n bit RCA ist eine kombinatorische Schaltung zur Addition von zwei n bit breiten Eingängen (`A` und `B`). Das Ergebnis (`S`) der Addition ist $n+1$ bit breit. Der RCA folgt dem Prinzip der schriftlichen Addition. Beginnend bei der am wenigsten signifikanten Stelle werden die beiden Eingangsbits und das von der vorherigen Stelle übertragene carry Bit addiert. Neben der Summe für die aktuelle Stelle entsteht dabei auch der Übertrag für die nächste Stelle.
Im Folgenden soll das Zeitverhalten eines RCA mit SystemVerilog beschrieben und analysiert werden. Verwenden Sie bei allen Verhaltensbeschreibungen dafür folgende Verzögerungszeiten: $t_{\text{pd,XOR}} = 3\,\text{ns}$, $t_{\text{pd,AND}} = 2\,\text{ns}$ und $t_{\text{pd,OR}} = 1\,\text{ns}$.

Übung 10.5.1 Halbaddierer

Ein Halbaddierer ist eine kombinatorische Schaltung zur Addition von zwei 1 bit Eingängen (`A` und `B`). Das 2 bit Ergebnis wird auf die beiden Signale `S` (LSB) und `CO` (MSB) aufgeteilt. Implementieren Sie einen Halbaddierer in SystemVerilog.

rca/half_adder.sv

```
1  `timescale 1 ns / 10 ps                                    // A B | CO S
2  module half_adder (input logic A, B, output logic S, CO); // 0 0 |  0 0
3    assign #3 S  = A ^ B;                                    // 0 1 |  0 1
4    assign #2 CO = A & B;                                    // 1 0 |  0 1
5  endmodule                                                  // 1 1 |  1 0
```

Übung 10.5.2 Volladdierer

Ein Volladdierer ist eine kombinatorische Schaltung zur Addition von drei 1 bit Eingängen (`A`, `B` und `CI`). Das 2 bit Ergebnis wird auf die beiden Signale `S` (LSB) und `CO` (MSB) aufgeteilt. Implementieren Sie einen Volladdierer in SystemVerilog. Verwenden Sie dafür zwei Halbaddierer-Instanzen.

rca/full_adder.sv

```
1  `timescale 1 ns / 10 ps
2  module full_adder (input logic A, B, CI, output logic S, CO);
3    logic s0, c0, c1;
4    half_adder ha0 (A,  B,  s0, c0);
5    half_adder ha1 (s0, CI, S,  c1);
6    assign #1 CO = c0 | c1;
7  endmodule
```

Übung 10.5.3 Carry-Chain

Implementieren Sie einen 3 bit RCA mit Hilfe von drei Volladdierern in SystemVerilog.

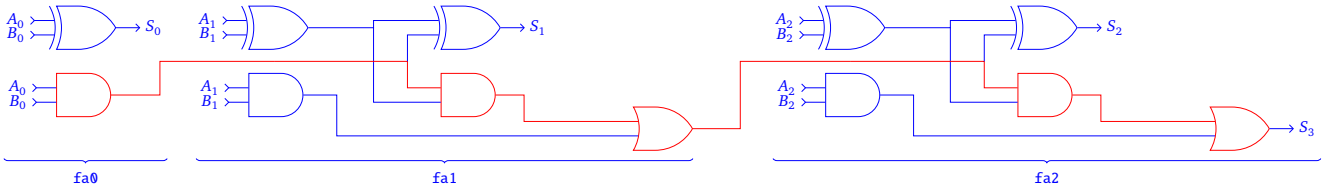
rca/add3.sv

```
1  `timescale 1 ns / 10 ps
2  module add3 (input logic [2:0] A, B, output logic [3:0] S);
3    logic [1:0] c;
4    full_adder fa0 (A[0], B[0], 1'b0, S[0], c[0]);
5    full_adder fa1 (A[1], B[1], c[0], S[1], c[1]);
6    full_adder fa2 (A[2], B[2], c[1], S[2], S[3]);
7  endmodule
```

Übung 10.5.4 Timing-Analyse

Zeichnen Sie die kombinatorische Schaltung des 3 bit RCA. Verwenden Sie dafür möglichst wenige Basisgatter (XOR, AND und OR). Markieren Sie den kritischen Pfad der Schaltung. Mit welcher Taktrate kann die Schaltung maximal betrieben werden, wenn Operanden und Ergebnis in Registern mit folgenden Charakteristiken gespeichert werden: $t_{\text{ccq}} = 100\,\text{ps}$, $t_{\text{pcq}} = 500\,\text{ps}$, $t_{\text{setup}} = 1{,}5\,\text{ns}$, $t_{\text{hold}} = 2\,\text{ns}$? Mit welcher Frequenz könnte ein entsprechender 32 bit RCA betrieben werden?

Jede Volladdierer-Stufe besteht aus zwei XOR, zwei AND und einem OR Gatter. Für den ersten Volladdierer (`fa0`) ist `CI == 0`. Dadurch entfällt in dieser Stufe der zweite Halbaddierer (`fa0.ha1`) und das OR Gatter:



Der (rot markierte) kritische Pfad der kombinatorischen Schaltung verläuft entlang der Carry-Chain und besteht aus zwei OR und drei AND Gattern, was einer Verzögerung von 8 ns entspricht. Zusammen mit t_{pcq} und t_{setup} der angrenzenden Register muss die Taktperiode damit mindestens 10 ns lang sein. Die Taktrate ist daher auf 100 MHz begrenzt.
Für jede weitere Volladdierer-Stufe verlängert sich der kritische Pfad um 3 ns. Ein 32 bit RCA könnte daher mit $\frac{1}{97\,\text{ns}} = 10{,}3\,\text{MHz}$ betrieben werden.

Übung 10.5.5 Simulation

Simulieren Sie Ihren RCA mit der in Moodle bereitgestellten Testbench (`rca/add3_tb.sv`). Warum ist die simulierte Verzögerung zwischen Ein- und Ausgang des RCA manchmal kleiner oder größer als in Übung 10.5.4 berechnet?

Bei 280 ns wechselt (A, B) von $(1, 5)$ nach $(1, 6)$. Der Ausgang ändert sich 6 ns später. Diese Verzögerung ist kürzer als die in Übung 10.5.4 berechneten 8 ns, weil nicht die komplette Carry-Chain umschalten muss (das MSB bleibt unverändert). Der kritische Pfad entspricht immer einer worst-case Analyse.
Bei 300 ns wechselt (A, B) von $(1, 6)$ nach $(1, 7)$. Der Ausgang nimmt erst nach 9 ns das korrekte Ergebnis an. Diese Verzögerung ist länger als die in Übung 10.5.4 berechneten 8 ns. Der Simulator hat also das OR Gatter des ersten Volladdierers nicht entfernt. Für eine genauere Timing-Analyse müsste daher die synthetisierte Basisgatterschaltung simuliert werden.