Digitaltechnik
Wintersemester 2021/2022
10. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

Prof. Dr.-Ing. Thomas Schneider, M.Sc. Daniel Günther, M.Sc. Amos Treiber

LÖSUNGSVORSCHLAG **KW03**

Bitte bearbeiten Sie die Übungsblätter bereits im Voraus, sodass Sie Ihre Lösungen zusammen mit Ihren Kommilitonen und Tutoren während der wöchentlichen Übungsstunde diskutieren können.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 10.1 Verilog Operatoren [5 min]

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

1. `A & B != C` ⇒ `A & (B != C)`
2. `A >>> D >> C` ⇒ `(A >>> D) >> C`
3. `D >> C > A << B` ⇒ `(D >> C) > (A << B)`
4. `A + B >> C << D` ⇒ `((A + B) >> C) << D`
5. `A && & C & D && B` ⇒ `(A && ((&C) & D)) && B`

b) Im Folgenden sind verschiedene Aussagen zu SystemVerilog gegeben. Geben Sie ein Beispiel an, falls die jeweilige Aussage stimmt. Korrigieren Sie die Aussage andernfalls.

1. Groß-/Kleinschreibung wird ignoriert. Falsch. Groß- und Kleinschreibung ist relevant.
2. Namen dürfen mit Ziffern anfangen. Falsch. Ein Name muss mit einem Buchstaben beginnen.
3. Anzahl von Leerzeichen sind irrelevant. Stimmt. `assign Y = A + B ;` und `assign Y=A+B;` sind äquivalent.

c) Geben Sie die Bedeutung der dargestellten Operationen an.

1. `A << 2` Logischer Shift von `A` um 2 Stellen nach links (entspricht $A \cdot 2^2$).
2. `~& B` Unäre Reduktion mit NAND auf alle Bits von `B` (entspricht also $\overline{B_1\ B_2\ \ldots\ B_n}$).

Übung 10.2 Verhaltens- und Strukturbeschreibung [15 min]

Realisieren Sie die nachfolgende Funktion in SystemVerilog: $Y = A\,\overline{B} + \overline{D}\,C$

a) Nutzen Sie die Verhaltensbeschreibung zur Darstellung der Funktion.

Aufgabe2/Verhaltensbeschreibung.sv

```
1 module aufgabe_a (input logic A, B, C, D
2                   output logic Y);
3
4   assign Y = A & ~B | ~D & C;
5
6 endmodule
```

b) Nutzen Sie die Strukturbeschreibung zur Darstellung der Funktion. Erstellen Sie dafür zunächst geeignete Module für die Basisoperationen.

Aufgabe2/Strukturbeschreibung.sv

```
module not_gate (input  A,
                 output Y);

  assign Y = ~A;

endmodule


module or_gate (input logic A, B,
                output logic Y);

  assign Y = A | B;

endmodule


module and_gate (input logic A, B,
                 output logic Y);

  assign Y = A & B;

endmodule


module aufgabe_b (input logic A, B, C, D
                  output logic Y);

  logic nB, nD, t1, t2;         //temporäre Variablen für Zwischenergebnisse

  not_gate not1 (.A(B), .Y(nB));
  not_gate not2 (.A(D), .Y(nD));
  and_gate and1 (.A(A), .B(nB), .Y(t1));
  and_gate and2 (.A(nD), .B(C), .Y(t2));
  or_gate or1 (.A(t1), .B(t2), .Y(Y));

endmodule
```

c) Wozu dienen beide Beschreibungsarten?

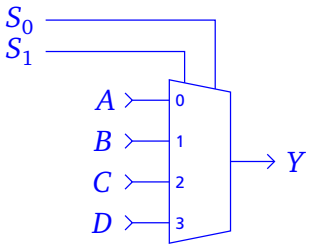
Die Strukturbeschreibung führt zu einer Modularhierarchie, bei der komplexere Schaltungen aus einfacheren Schaltungen zusammengesetzt werden können. Außerdem kann die Regularität von Schaltungen durch das mehrfache Einbinden desselben Submoduls ausgenutzt werden.

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

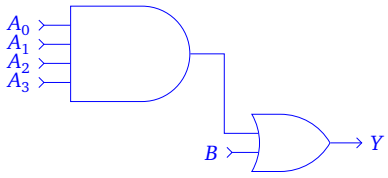
Übung 10.3 Synthese [20 min]

a) Zeichnen Sie die von folgenden Modulen (`m1` und `m2`) beschriebenen kombinatorischen Schaltungen.

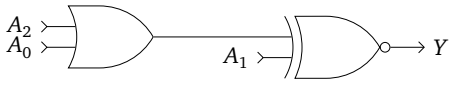
```
module m1 (input logic A, B, C, D,
           input logic [1:0] S,
           output logic Y);

  assign Y = S[1] ? (S[0] ? D : C)
                  : (S[0] ? B : A);

endmodule
```



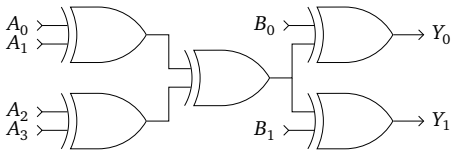

```
module m2 (input logic [3:0] A,
           input logic B,
           output logic Y);

  assign Y = &A | B;

endmodule
```



b) Erstellen Sie aus den gegebenen Schaltungen SystemVerilog-Module.




```
module m3 (input logic [2:0] A,
           output logic Y);

  assign Y = (A[2] | A[0]) ~^ A[1];

endmodule
```




```
module m4 (input  logic [3:0] A,
           input logic [1:0] B,
           output logic [1:0] Y);

  assign Y = {2{^A}} ^ B;

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 ein 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.

processor/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.

processor/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.

processor/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);

  assign R = (OPC == 3'd0) ? R0 :
             (OPC == 3'd1) ? R1 :
             (OPC == 3'd2) ? R2 :
             32'd0;
endmodule
```