Digitaltechnik
Wintersemester 2022/2023
13. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

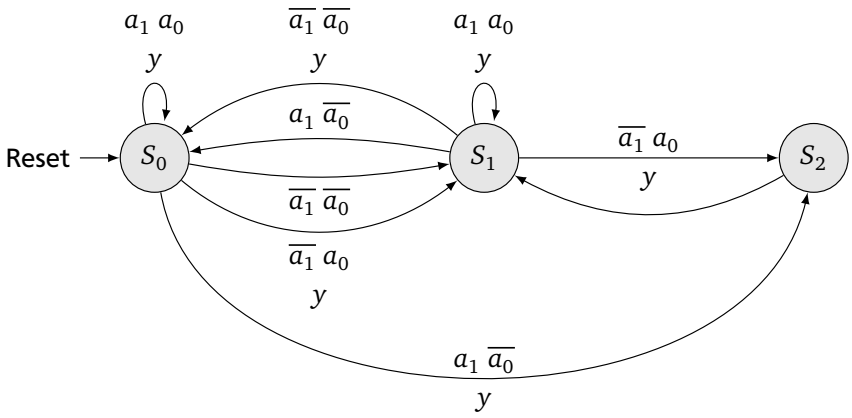
M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW06

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen. Die mit **"Zusatzaufgabe"** gekennzeichneten Aufgaben sind zur zusätzlichen Vertiefung für interessierte Studierende gedacht und daher nicht im Zeitumfang von 90 Minuten einkalkuliert.

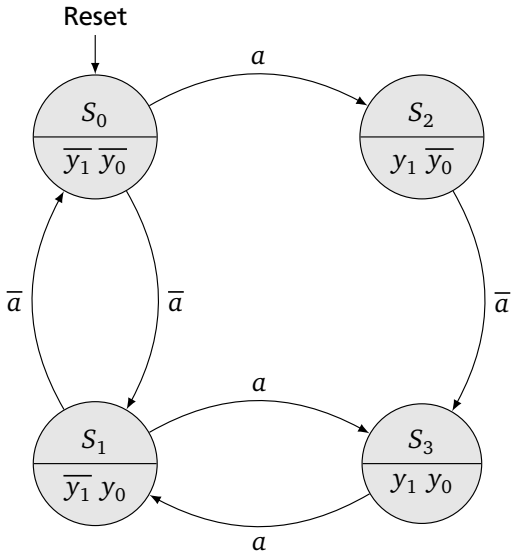
Übung 13.1 Robuste Endliche Automaten [20 min]

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.

a)

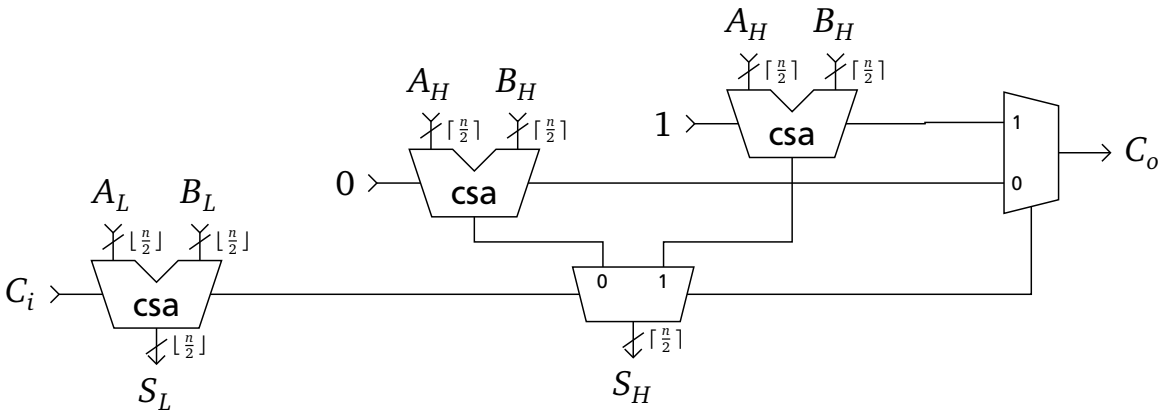


b)



Übung 13.2 Conditional Sum Adder (CSA) [25 min]

Ein Nachteil des Ripple-Carry-Adders ist dessen lineare Übertragskette vom LSB bis zum MSB, wodurch der kritische Pfad linear mit der Bitbreite wächst. Ein n-Bit CSA bricht diese Übertragskette auf, indem für die oberen $\lceil \frac{n}{2} \rceil$ Eingabebits sowohl die einfache Summe ($A_H + B_H$), als auch dessen Inkrement ($A_H + B_H + 1$) *gleichzeitig* berechnet werden. Sobald der Übertrag des unteren Halbworts ($A_L + B_L + C_i$) verfügbar ist, muss nur noch das korrekte Ergebnis (Summe und Übertrag) aus den beiden Berechnungen für das obere Halbwort ausgewählt werden.



Übung 13.2.1 Rekursive Implementierung

Implementieren Sie den CSA in SystemVerilog als rekursives Modul mit Übertragsein- und ausgang:

arith/adder/csa.sv

```
2  module csa #(parameter     WIDTH=4)
3              (input logic  [WIDTH-1:0] A, B, input  logic CI,
4               output logic [WIDTH-1:0] S,    output logic CO);
```

Ein 1 bit CSA entspricht einem Volladdierer. Beachten Sie, dass `WIDTH` nicht immer ohne Rest durch zwei teilbar ist. Verwenden Sie die Module für Halb- und den Volladdierer (aus Übung 10), die in Moodle unter `SystemVerilog/src/arith/adder` zur Verfügung stehen. Die Verzögerungszeit der Multiplexer soll 4 ns betragen.

Übung 13.2.2 Modul-Kapselung

Verpacken Sie den CSA in ein Modul mit der folgender allgemeiner Addierer-Schnittstelle:

```
21 module add #(parameter    WIDTH=4)
22             (input logic [WIDTH-1:0] A, B, output logic [WIDTH:0] S);
```

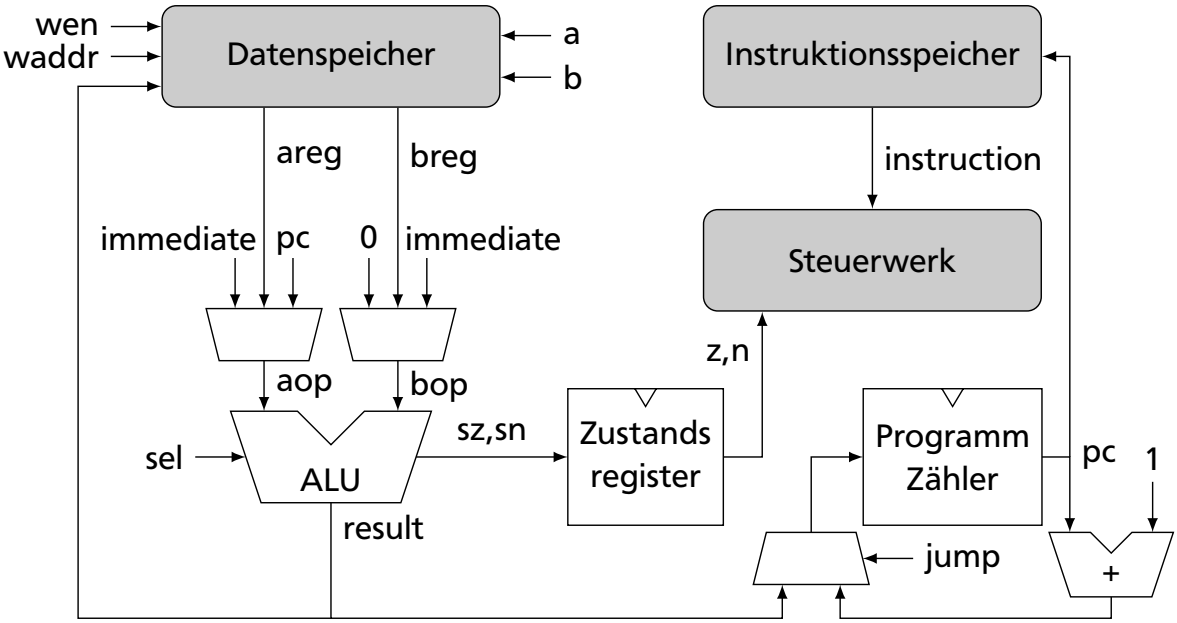
Übung 13.2.3 Verifikation

Schreiben Sie eine Testbench, die den CSA mit einer per **`localparam`** konfigurierbaren Bitbreite erschöpfend funktional validiert. Bestimmen Sie dabei auch die maximale Verzögerungszeit des CSA und ergänzen Sie dazu folgende Tabelle:

`WIDTH`	2	4	6	8	10
$t_{pd,CSA}$					

Übung 13.3 Modellprozessor [40 min]

In dieser Aufgabe wird ein einfacher Prozessor in SystemVerilog beschrieben und ein arithmetischer Algorithmus auf Basis des realisierten Instruktions-Satzes implementiert. Folgende Grafik zeigt die Architektur des Prozessors:



Einige für den Prozessor benötigte Quelldateien stehen in Moodle unter `SystemVerilog/src/processor` zur Verfügung.

Übung 13.3.1 Instruktionspeicher

Der Instruktionsspeicher benötigt lediglich einen asynchronen Leseport. Seine Initialisierung mit den Instruktionen des auszuführenden Programms erfolgt später in der Testbench des Prozessors. Implementieren Sie den Instruktionsspeicher mit folgender Schnittstelle:

processor/imem.sv

```
module imem #(parameter     WIDTH = 8,                  // Bitbreite der Instruktionen
              parameter     DEPTH = 16)                 // Anzahl der Instruktionen
             (input  logic [$clog2(DEPTH)-1:0] ADDR,   // Leseadresse
              output logic [WIDTH-1:0] D);              // Lesedaten
```

Übung 13.3.2 Datenspeicher (Register)

Der Datenspeicher wird auch als Register-Satz bezeichnet und benötigt neben zwei asynchronen Leseports einen synchronen Schreibport. Dieser Speicher hat keinen Reset-Eingang und wird bei Bedarf durch das Ausführen bestimmter Instruktionen initialisiert. Implementieren Sie den Datenspeicher mit folgender generischer Schnittstelle:

processor/dmem.sv

```
module dmem
  #(parameter    WIDTH = 8,                                  // Bitbreite der Register
    parameter    DEPTH = 16)                                 // Anzahl der Register
   (input  logic CLK,                                        // Takt
    input  logic [$clog2(DEPTH)-1:0] AADDR,BADDR,WADDR,      // Schreib/Lese Adressen
    input  logic [WIDTH-1:0] WDATA,                          // Schreibdaten
    input  logic WEN,                                        // Schreibzugriff aktivieren
    output logic [WIDTH-1:0] ADATA, BDATA);                  // Lesedaten
```

Übung 13.3.3 Arithmetisch-Logische Einheit (ALU)

Die ALU soll folgende Operationen umsetzen:

SEL	0	1	2	3	4	5	6	7	8	9	10
R	A+B	A-B	A&B	A|B	A^B	A<<B	A>>B	A<<<B	A>>>B	&A	|A

Dafür können die entsprechenden SystemVerilog Operatoren verwendet werden. Für alle anderen (ungenutzten) Werte des Selektionssignals (`SEL`) soll das Ergebnis der Addition ausgegeben werden. Neben dem Operationsergebnis sollen zwei Statusausgänge `Z` und `N` anzeigen, ob das Ergebnis 0 bzw. negativ ist. Implementieren Sie die kombinatorische ALU mit folgender generischer Schnittstelle:

processor/alu.sv

```
module alu #(parameter     WIDTH = 8)       // Bitbreite der Ein-/Ausgänge
            (input  logic [WIDTH-1:0] A,B, // Operanden
             input  logic [3:0] SEL,       // Auswahlsignal
             output logic [WIDTH-1:0] R,   // Ergebnis
             output logic Z,N);            // Statussignale
```

Übung 13.3.4 Steuerwerk und Gesamtmodell

ALU, Instruktions- und Datenspeicher müssen im Modul des Prozessors instanziiert und mit dem Steuerwerk verknüpft werden. Die Bitbreiten der Daten und Adressleitungen werden durch den Instruktionssatz bestimmt und in `processor/isa.svh` als Präprozessor-Makros (beginnend mit backtick: `` ` ``) definiert. Folgender Teil des Moduls ist bereits vorgegeben:

processor/core_stub.sv

```
`include "isa.svh"

module core (input logic CLK, RESET);

  localparam ZERO = `DATA_WIDTH'd0;

  logic signed [  `DATA_WIDTH-1:0] areg,breg,aop,bop,result,immediate;
  logic        [ `DADDR_WIDTH-1:0] a,b,r,waddr;
  logic        [ `INSTR_WIDTH-1:0] instruction;
  logic        [ `IADDR_WIDTH-1:0] pc;
  logic        [`OPCODE_WIDTH-1:0] opcode;
  logic        [              3:0] sel;
  logic                            wen,z,n,sz,sn,jump;

  // Datenspeicher (Register)
  dmem #(`DATA_WIDTH, `DATA_DEPTH) i_dmem
   (.CLK(CLK),   .WEN(wen),
    .AADDR(a),   .BADDR(b),   .WADDR(waddr),
    .ADATA(areg),.BDATA(breg),.WDATA(result));

  // Instruktionsspeicher
  imem #(`INSTR_WIDTH, `INSTR_DEPTH) i_imem (pc, instruction);

  // Arithmetisch-Logische Einheit
  alu #(`DATA_WIDTH) i_alu (aop,bop,sel,result,sz,sn);

  // Steuerwerk hier einfügen

endmodule
```

Das Steuerwerk soll als kombinatorische Logik im Modul des Prozessors realisiert werden. Es erzeugt aus der aktuellen Instruktion die Signale zum Ansteuern aller anderen Komponenten und realisiert so den Instruktionssatz des Prozessors:

Befehl	kodierte Instruktion	Registeränderung	nächster Programmzähler
ADD(r,a,b)	{4'b0000,7'bx,r,a,b}	R[r] = R[a] + R[b]	pc+1
SUB(r,a,b)	{4'b0001,7'bx,r,a,b}	R[r] = R[a] - R[b]	pc+1
AND(r,a,b)	{4'b0010,7'bx,r,a,b}	R[r] = R[a] & R[b]	pc+1
OR(r,a,b)	{4'b0011,7'bx,r,a,b}	R[r] = R[a] | R[b]	pc+1
XOR(r,a,b)	{4'b0100,7'bx,r,a,b}	R[r] = R[a] ^ R[b]	pc+1
SHL(r,a,b)	{4'b0101,7'bx,r,a,b}	R[r] = R[a]<< R[b]	pc+1
SHR(r,a,b)	{4'b0110,7'bx,r,a,b}	R[r] = R[a]>> R[b]	pc+1
ASHL(r,a,b)	{4'b0111,7'bx,r,a,b}	R[r] = R[a]<<<R[b]	pc+1
ASHR(r,a,b)	{4'b1000,7'bx,r,a,b}	R[r] = R[a]>>>R[b]	pc+1
ARED(r,a,b)	{4'b1001,7'bx,r,a,b}	R[r] = & R[a]	pc+1
ORED(r,a,b)	{4'b1010,7'bx,r,a,b}	R[r] = | R[a]	pc+1
MOV(r,a)	{4'b1011,7'bx,r,a,0}	R[r] = R[a]	pc+1
LDI(immediate)	{4'b1100,immediate}	R[0] = immediate	pc+1
JMP(immediate)	{4'b1101,immediate}		pc+ immediate
JN(immediate)	{4'b1110,immediate}		pc+(n ? immediate : 1)
JZ(immediate)	{4'b1111,immediate}		pc+(z ? immediate : 1)

Dabei sind a,b und r Registeradressen der Breite `DADDR_WIDTH und die immediate Einträge sind vorzeichenbehaftete Konstanten der Breite `DATA_WIDTH. n und z sind die Statussignale der ALU für die unmittelbar zuvor ausgeführten Instruktion. Sie müssen in einem Statusregister gepuffert werden, um in Abhängigkeit vom Ergebnis einer Berechnung einen Sprung im Programmfluss auszuführen. Wie im Schaltbild des Prozessors angedeutet, sollte das Sprungziel (pc+immediate) durch die ALU berechnet werden.
Die Befehle MOV ("move", für das Kopieren von Registern) und LDI ("load immediate", für das Laden von Konstanten) führen eigentlich keine Berechnung aus, lassen sich als Addition mit Null aber auch über die ALU realisieren.
Ergänzen Sie das Prozessormodul um Steuerwerk, Statusregister und Programmzähler.

Übung 13.3.5 Assembler-Programm – **Zusatzaufgabe**

Um die Funktionalität der Prozessor-Implementierung zu überprüfen, muss ein konkretes Programm in den Instruktionsspeicher geladen werden, dessen Abarbeitung dann beobachtet werden kann. Dazu wird folgende Testbench zur Verfügung gestellt:

processor/tb.sv

```
1  `default_nettype none
2  `timescale 1 ns / 10 ps
3  `include "isa.svh"
4
5  `define PROGRAM "simple.asm"
6
7  module tb;
8
9    // Prozessor takten
10   logic   clk=0, reset=1;
11   always  #0.5           clk   <= ~clk;
12   initial @(posedge clk) reset <= 0;
13   core uut (clk, reset);
14
15   // simuierte Signale (Speicher müssen explizit hinzugefügt werden)
16   initial begin
17     $dumpfile("tb.vcd");
18     $dumpvars;
19     for (int i=0; i<`INSTR_DEPTH; i++) $dumpvars(1, uut.i_imem.m[i]);
20     for (int i=0; i<`DATA_DEPTH;  i++) $dumpvars(1, uut.i_dmem.m[i]);
21   end
22
23   // Programm in Instruktionsspeicher laden
24   `include "asm.svh"
25   initial begin
26     clear_instructions;
```

```
27      `include   `PROGRAM
28      $readmemb({`PROGRAM,".bin"}, uut.i_imem.m);
29    end
30
31    // Simulation bei Endlosschleife abbrechen
32    always @(posedge clk) if (uut.opcode == `JMP && uut.immediate == 0) begin
33      $display("FINISHED tb");
34      $finish;
35    end
36  endmodule
```

Dabei wird das zu ladende Programm in Zeile 5 spezifiziert, welches neben SystemVerilog Kommentaren ausschließlich die oben angegebenen Assembler Befehle verwenden darf. Ein einfaches Beispiel für ein solches Assembler-Programm sieht wie folgt aus:

processor/simple.asm

```
1  /*PC*/
2  /* 0*/ LDI(1);       // R[0] = 1
3  /* 1*/ MOV(1,0);     // R[1] = R[0] = 1
4  /* 2*/ LDI(2);       // R[0] = 2
5  /* 3*/ MOV(2,0);     // R[2] = R[0] = 2
6  /* 4*/ ADD(3,1,2);   // R[3] = R[1] + R[2] = 3
7  /* 5*/ JMP(0);       // Endlosschleife
```

Die erste Kommentarspalte gibt dabei die Adresse des Befehls im Instruktionsspeicher an. Dies ist hilfreich bei der Verwendung von Sprüngen, da hier (im Gegensatz zu vollwertigen Assembler-Programmen) keine Sprungmarken verwendet werden können. Stattdessen muss der relative Abstand zum Sprungziel als `immediate` des Sprungbefehls angegeben werden. Daher realisiert der unbedingte Sprung um Null Instruktionen (`JMP(0)`) eine Endlosschleife. Diese Endlosschleife wird zum Abbruch der Simulation verwendet und sollte daher der letzte Befehl eines jeden Programms sein.
Die Testbench nimmt an, dass die lokalen Arrays im Instruktions- und Datenspeicher mit `m` bezeichnet werden (Zeile 19, 20, 28). Passen Sie Ihre Implementierungen entsprechend an, da sonst auch GTKWave nach der Simulation nicht die richtigen Signale anzeigt.
Zum Starten der Simulation genügt der Aufruf der Testbench, das Assembler-Programm muss also nicht als Teil der Quelldateien spezifiziert werden.
Realisieren Sie eine sequentielle Multiplikation von zwei vorzeichenlosen 8 bit Operanden. In Java würde dieser Algorithmus wie folgt implementiert:

processor/mul.java

```
1  int a = 42;
2  int b = 37;
3  int p = 0;
4  for (int n=8; n!=0; n--) {
5    if (b & 1 == 1) p += a;
6    a = a << 1;
7    b = b >> 1;
8  }
```

Dabei werden in den ersten beiden Zeilen die miteinander zu multiplizierenden Operanden `a` und `b` spezifiziert. Nach Abbruch der Schleife enthält `p` das Produkt `a * b`. Setzen Sie diesen Algorithmus mit den Assembler-Befehlen des Modellprozessors um. Dabei sollen die Variable `a`, `b` und `p` in den Registern 1, 2 und 3 abgelegt werden. Evaluieren Sie Ihre Implementierung für verschiedene Operanden.

Digitaltechnik
Wintersemester 2022/2023
12. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW05

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 12.1 2 bit Addierer [20 min]

In dieser Aufgabe implementieren Sie einen 2 bit Addierer auf verschiedene Arten.
Bei einem 2 bit Addierer handelt es sich um eine kombinatorische Schaltung mit folgender Schnittstelle:

- Inputs:
  - 2 bit breite Zahlen A und B ($A := a_1 a_0$, $B := b_1 b_0$)
- Outputs:
  - 2 bit breite Summe S von A und B ($S := s_1 s_0$)
  - Übertrag C

a) Implementieren Sie den 2 bit Addierer mit Basisgattern:
   1. Stellen Sie die Wahrheitstabelle für den 2 bit Addierer auf.
   2. Nutzen Sie Karnaugh Diagramme, um die minimierte DNF für s_0, s_1 und C zu ermitteln.
   3. Implementieren Sie die Schaltung als zweistufige Logik.
   4. Modellieren Sie die zweistufige Logik als Verhaltensbeschreibung in SystemVerilog. Verwenden Sie für Basisgatter die entsprechenden Operatoren (&, |, ~).

b) Implementieren Sie den 2 bit Addierer mit Hilfe von Halb- und Volladdierern.
   1. 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 und C aufgeteilt. Implementieren Sie einen Halbaddierer als Verhaltensbeschreibung in SystemVerilog.
   2. Ein Volladdierer ist eine kombinatorische Schaltung zur Addition von drei 1 bit Eingängen (A, B und C_{in}). Das 2 bit Ergebnis wird auf die beiden Signale S und C_{out} aufgeteilt. Implementieren Sie einen Volladdierer in SystemVerilog. Verwenden Sie dafür zwei Halbaddierer.
   3. Implementieren Sie den 2 bit Addierer als Strukturbeschreibung in SystemVerilog. Verwenden Sie dafür einen Halb- und einen Volladdierer.

c) Implementieren Sie den 2 bit Addierer als Verhaltensbeschreibung in SystemVerilog.

Übung 12.2 Pipelining – Timing-Bedingungen [20 min]

Folgender SystemVerilog Code beschreibt die kombinatorische Schaltung $Y = A + (\overline{A \oplus D})\ C + \overline{B}$ zwischen zwei Registern im Modul `base`, sowie die dazugehörige funktionale Verifikation in der Testbench `base_tb`:

seq/pipeline/gates.sv

```
`timescale 1 ns / 10 ps
module or_gate  #(parameter W=2) (input logic [W-1:0] A, output logic Y);
  assign #(W)   Y = |A;
endmodule

module and_gate #(parameter W=2) (input logic [W-1:0] A, output logic Y);
  assign #(W)   Y = &A;
endmodule

module xor_gate #(parameter W=2) (input logic [W-1:0] A, output logic Y);
  assign #(W+1) Y = ^A;
endmodule

module inv_gate                  (input logic         A, output logic Y);
  assign #(1)   Y = ~A;
endmodule
```

seq/pipeline/register.sv

```
`timescale 1 ns / 10 ps
module register #(parameter W      = 1,
                  parameter tsetup = 0.9,
                  parameter thold  = 0.5,
                  parameter tcq    = 0.1)
                 (input logic CLK, input logic [W-1:0] D, output logic [W-1:0] Q);

  logic [W-1:0] t;
  always_ff @(posedge CLK) begin
            t <= D;
    #(tcq) Q <= t;
  end
endmodule
```

seq/pipeline/base.sv

```
module base (input logic CLK, A, B, C, D, output logic Y);
  logic a,b,c,d,n1,n2,n3,n4,y;
  register #(4) rin (CLK, {A,B,C,D}, {a,b,c,d});
  xor_gate      g1  ({a,d},     n1);
  inv_gate      g2  ( n1,       n2);
  and_gate      g3  ({n2,c},    n3);
  inv_gate      g4  ( b,        n4);
  or_gate  #(3) g5  ({a,n3,n4}, y );
  register      rout(CLK, y,    Y );
endmodule
```

seq/pipeline/base_tb.sv

```
`timescale 1 ns / 10 ps
module base_tb;

  logic a,b,c,d,y,clk = 0;
  always #4.8 clk = ~clk;

  base uut (clk,a,b,c,d,y);
```

```
8
9   localparam L = 2;
10  logic [L-1:0] e;
11
12  always @(posedge clk) begin
13    e <= {e[L-2:0], a | ~(a^d)&c | ~b};
14  end
15
16  initial begin
17    $dumpfile("base_tb.vcd");
18    $timeformat(-9, 2, " ns", 10);
19    $dumpvars;
20
21    for (int i=0; i<16+L; i++) begin
22      #1 {a,b,c,d} <= i;
23      if (y!==e[L-1]) $display("%t: expected %0d but got %0d",$realtime,e[L-1],y);
24      @(posedge clk);
25    end
26
27    $display("FINISHED base_tb");
28    $finish;
29  end
30
31 endmodule
```

Übung 12.2.1 Timing-Analyse

Die Parameter der Registerimplementierung (`tsetup`, `thold` und `tcq`) beschreiben die Setup-, Hold- und Verzögerungszeiten (mit $t_{\text{pcq}} = t_{\text{ccq}}$) in Nanosekunden. Mit welcher Frequenz kann das `base` Modul theoretisch maximal getaktet werden ohne die Timing-Bedingungen zu verletzen? Welche Latenz hat das Modul?

Übung 12.2.2 Testbench-Analyse

Mit welcher Frequenz wird die Schaltung in der Testbench getaktet? Warum entdeckt die Testbench keine funktionalen Fehler, obwohl die theoretischen Timing-Bedingungen der Register im `base` Modul verletzt werden?

Übung 12.2.3 Überprüfen von Setup- und Hold-Bedingung

Erweitern Sie die Registerimplementierung so, dass die Timing-Bedingungen t_{setup} und t_{hold} automatisch überprüft werden. Dazu bietet es sich an, die Zeitpunkte der Änderungen am Dateneingang und der steigenden Taktflanken in entsprechenden **`always`** Blöcken zu kontrollieren. Bei einer Verletzung der Bedingungen soll eine entsprechende Meldung ausgegeben werden.

Übung 12.2.4 Zusätzliche Pipeline-Stufen

Modifizieren Sie das `base` Modul durch Einführen zusätzlicher Pipeline-Stufen so, dass es mit Taktperiodendauer 4,2 ns betrieben werden kann, was einer Taktfrequenz von 238 MHz entspricht. Die verwendeten Logikgatter sollen dabei nicht verändert werden. Modifizieren Sie auch die Testbench so, dass das schnellere Modul korrekt getestet wird. Wie wirkt sich diese Modifikation auf die Latenz der Schaltung aus?

Übung 12.3 Strukturierung sequentieller 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* mittels nebenläufiger Zuweisungen realisieren (beschreibt die Transfer-Logik)

arith/mul/sequential.sv

```
module mul4x4 (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 12.4 Barrel-Shifter [15 min]

In der Vorlesung (VL 07) wurden sogenannte Barrel-Shifter behandelt. Wir betrachten ergänzend die "Rotate Left" Variante (umlaufender Linksshift). Anders als die "Arithmetic" Variante aus der Vorlesung lässt der rotierende Shifter nach links "raus geschobene" Bits nicht fallen, sondern fügt diese am anderen Ende des zu verschiebenden Wortes wieder ein.

a) Erstellen Sie eine Verhaltensbeschreibung in SystemVerilog. Der Parameter *SIZE* in der gegebenen Schnittstelle bestimmt die Anzahl an Steuersignalen. (Hinweis: Die minimale Lösung besteht aus einer Zeile.)

comb/barrel/Barrel_Funct.sv

```
module functional #(parameter SIZE=2)
                   (input  logic [SIZE-1:0]      S,
                    input  logic [(2**SIZE)-1:0] I,
                    output logic [(2**SIZE)-1:0] O);
```

b) Erstellen Sie eine Strukturbeschreibung in SystemVerilog basierend auf Multiplexern. Anstelle von Multiplexermodulen soll der ternäre Operator zur Instantiierung verwendet werden.

comb/barrel/Barrel_Struct.sv

```
module structural #(parameter SIZE=2)
                   (input  logic [SIZE-1:0]      S,
                    input  logic [(2**SIZE)-1:0] I,
                    output logic [(2**SIZE)-1:0] O);
```

c) Erstellen Sie eine Testbench (muss nicht selbstprüfend sein), die beide Implementierungen für einen von Ihnen gewählten Parameter *SIZE* mittels eines aussagekräftigen Bitmusters für alle möglichen Schiebewerte *S* testet.

Digitaltechnik
Wintersemester 2022/2023
11. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW04

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 11.1 Pipelining – Register Transfer Logik [20 min]

Folgende Grafik zeigt eine Pipeline zur Berechnung der Funktion $(x^2+5)\cdot 2-8$:

Mittels eines Eingangsignals **set** wird signalisiert, dass mit einer steigenden Taktflanke ein neuer Wert in das erste Register der Pipeline geladen werden soll. Der in R5 gespeicherte Wert entspricht dem Ergebnis der Funktion.

a) Setzen Sie die Pipeline in SystemVerilog um. Erweitern Sie dafür den unten gegebenen Quelltext für das Pipeline-Modul und entwerfen Sie zusätzliche Module zum Berechnen der Funktion jeder Pipeline-Stufe.

Das Verwenden der arithmetischen Operatoren von SystemVerilog ist in den funktionalen Zusatzmodulen erlaubt. Die Setup/Hold-Zeiten der Register sowie Überläufe können vernachlässigt werden.

seq/pipeline/pipeline.sv

```
1  module pipeline (input logic clock, set, input logic [7:0] in,
2                   output logic [7:0] out);
3
4    // Register zum puffern der Werte zwischen den Pipeline-Stufen
5    logic [7:0] register1, register2, register3, register4, register5;
6
7    // Ausgänge der einzelnen Rechenoperationen
8    logic [7:0] out1, out2 , out3 , out4;
```

b) Erstellen Sie eine Testbench, um die Funktion der Pipeline anhand von einigen Werten per Simulation zu testen.

c) Nehmen Sie nun an, dass die t_{pcq}- sowie t_{setup}-Zeit der Register bei 0,5 ns liegt und die kombinatorischen Schaltungen für die Rechnungen zwischen den Registern kritische Pfade mit folgenden Verzögerungen haben: x^2: 0,6 ns, +5: 0,8 ns, ×2: 0,5 ns, −8: 1 ns. Mit welcher Taktfrequenz lässt sich die Pipeline maximal betreiben? Welche Frequenz wäre möglich, wenn man auf Register 2 und 4 verzichtet?

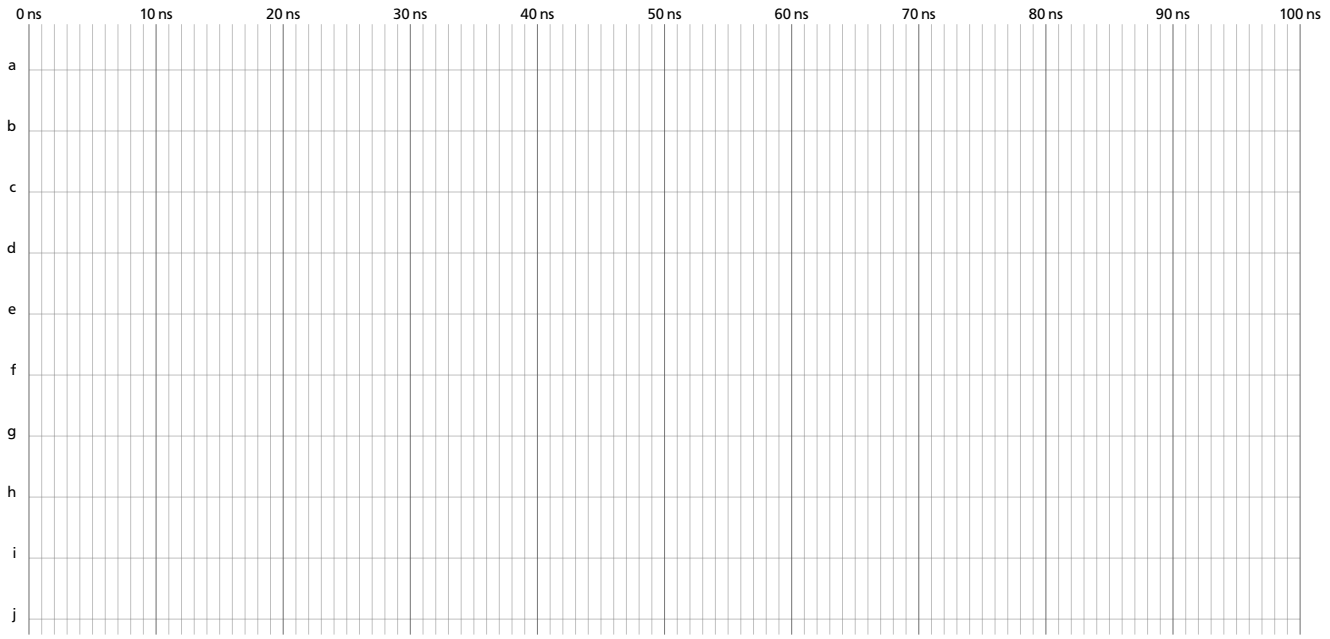
d) Welcher Vor- und welcher Nachteil ergibt sich hauptsächlich aus der Verwendung von vielen Pipeline-Stufen?

Übung 11.2 Zeitverhalten sequentieller Beschreibungen [15 min]

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

seq/timing.sv

```
`timescale 1 ns / 10 ps
module timing;
  localparam x = 2;

  logic [2:0] a = 0;
  always begin if (!a[0]) #10; else #(3+x); a <= a+1; end

  logic b, c, d, e, f, g, h, i, j;
  assign b = ^a;
  always               begin             c  = b;     d  = c; @(negedge a[0]); end
  always               begin             e  = b; #a; f  = e; @(posedge a[0]); end
  always @(negedge b) begin     g <= c; h <= g;                              end
  always @(f|d)       begin #2; i  = e; j <= i;                              end
endmodule
```



Übung 11.3 Parametrisierte Moduldefinition [15 min]

In dieser Aufgabe soll ein XOR Gatter mit einer variablen Anzahl an Eingängen realisiert werden.

a) Implementieren Sie ein funktionales Modul (Verhaltensbeschreibung) zu folgender Schnittstelle:

comb/xor/Parameter_Xor_Funct.sv

```
module xor_functional #(parameter SIZE = 2)
                       (input  logic [SIZE-1 : 0] I,
                        output logic              O);
```

b) Realisieren Sie ein äquivalentes strukturelles Modul (Strukturbeschreibung) zu nachfolgender Schnittstelle. Nutzen Sie dafür eine **for** Schleife um eine variable Anzahl an Zuweisungen zu generieren.

comb/xor/Parameter_Xor_Struct.sv

```
1 module xor_structural #(parameter SIZE = 2)
2                         (input  logic [SIZE-1 : 0] I,
3                          output logic              O);
```

c) Entwickeln Sie eine selbsttestende Testbench für beide Module mit Größe 4.

Digitaltechnik
Wintersemester 2022/2023
10. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW03

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
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`
2. `A >>> D >> C`
3. `D >> C > A << B`
4. `A + B >> C << D`
5. `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.
2. Namen dürfen mit Ziffern anfangen.
3. Anzahl von Leerzeichen sind irrelevant.

c) Geben Sie die Bedeutung der dargestellten Operationen an.

1. `A << 2`
2. `~& B`

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.

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

c) Wozu dienen beide Beschreibungsarten?

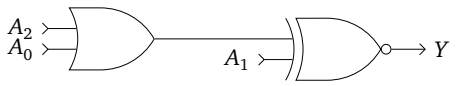
Übung 10.3 Synthese [20 min]

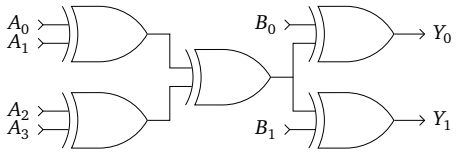
a) Zeichnen Sie die von folgenden Modulen (`m1` und `m2`) beschriebenen kombinatorischen Schaltungen.

```
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.





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.

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.

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.

Übung 10.4.4 Operator-Erweiterung

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

Digitaltechnik
Wintersemester 2022/2023
9. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW02

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 9.1 Realisierung endlicher Automaten EX10-C-1 EX10-C-2 EX10-C-3 [25 min]

Gegeben ist eine Zustandsübergangs- und Ausgangstabelle eines endlichen Automaten:

Zustand		Eingänge		Nächster Zustand	
S_1	S_0	B	A	S_1'	S_0'
0	0	*	0	0	0
0	0	*	1	0	1
0	1	0	*	0	1
0	1	1	*	1	0
1	0	*	0	1	0
1	0	0	*	1	0
1	0	1	1	0	0

Zustand		Ausgang
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

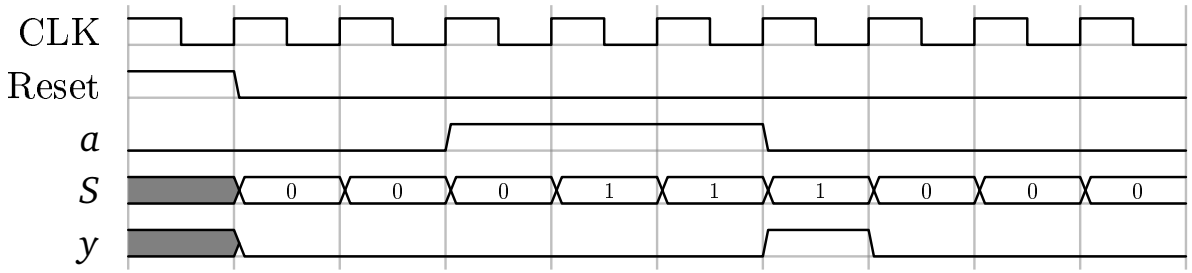
a) Welche Art von Automat wird durch die Tabellen beschrieben? Wie sind die Zustände kodiert?

b) Zeichnen Sie das zugehörige FSM Diagramm.

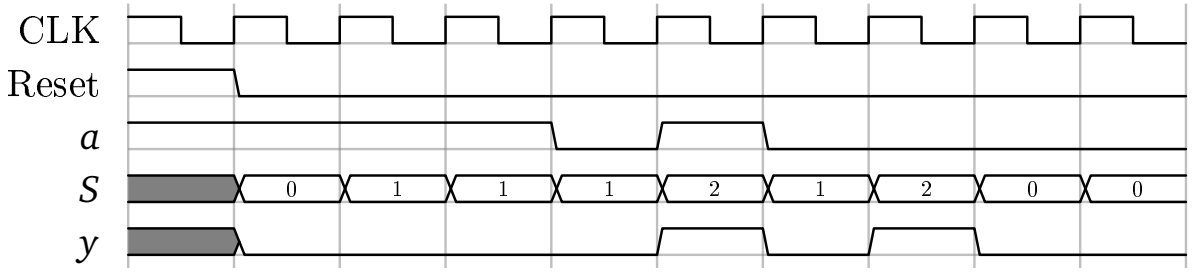
c) Setzen Sie den Automaten als synchrone sequentielle Schaltung um. Nutzen Sie dafür kombinatorische Logik und D-Flip-Flops für die Zustände. Es bietet sich an, die Zustandsübergangsfunktionen zuerst zu minimieren.

Übung 9.2 Automat aus Timing-Diagramm [10 min]

a) Zeichnen Sie aus dem gegebenen Timing-Diagramm das zugehörige FSM-Diagramm eines Mealy-Automaten.



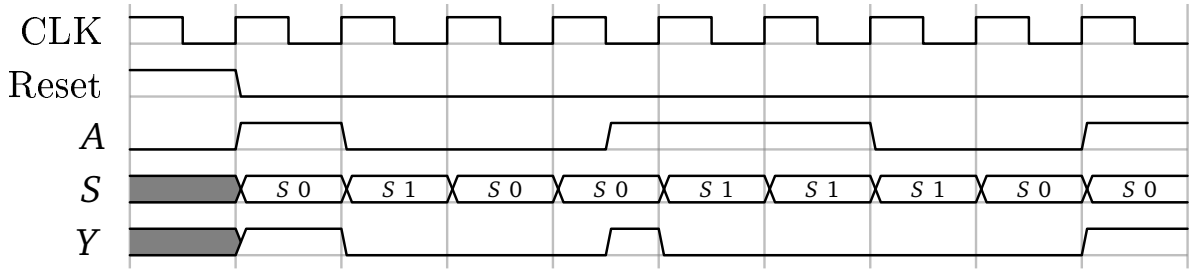
b) Zeichnen Sie aus dem gegebenen Timing-Diagramm das zugehörige FSM-Diagramm eines Moore-Automaten.



Übung 9.3 Mealy vs. Moore [15 min]

Übung 9.3.1 Analyse eines Timing-Diagramms

Das folgende Timing-Diagramm zeigt das Verhalten eines Zustandsautomaten unbekannten Typs mit dem Eingang A, einem Ausgang Y und dem Zustand S:

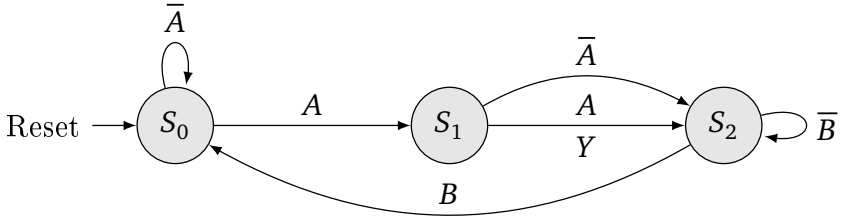


a) Handelt es sich um einen Mealy- oder Moore-Automaten? Woran haben sie dies erkannt?

b) Beschreiben sie die Unterschiede zwischen beiden Automatentypen.

Übung 9.3.2 Automat umwandeln EX10-C-4

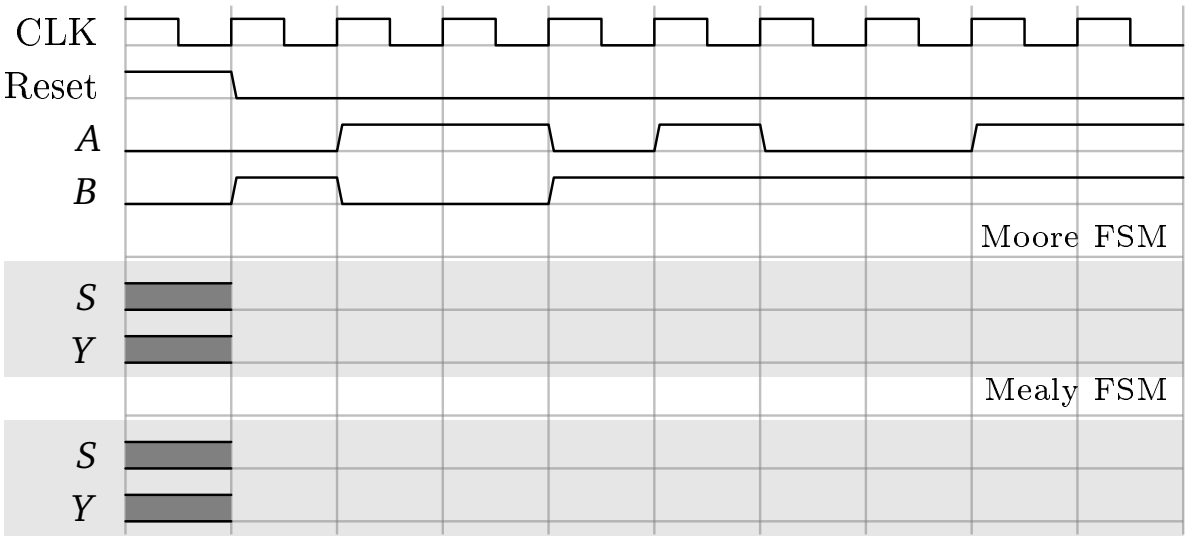
Gegeben ist folgender endlicher Automat mit den Eingängen A und B, sowie einem Ausgang Y:



a) Handelt es sich um einen Moore- oder einen Mealy-Automat?

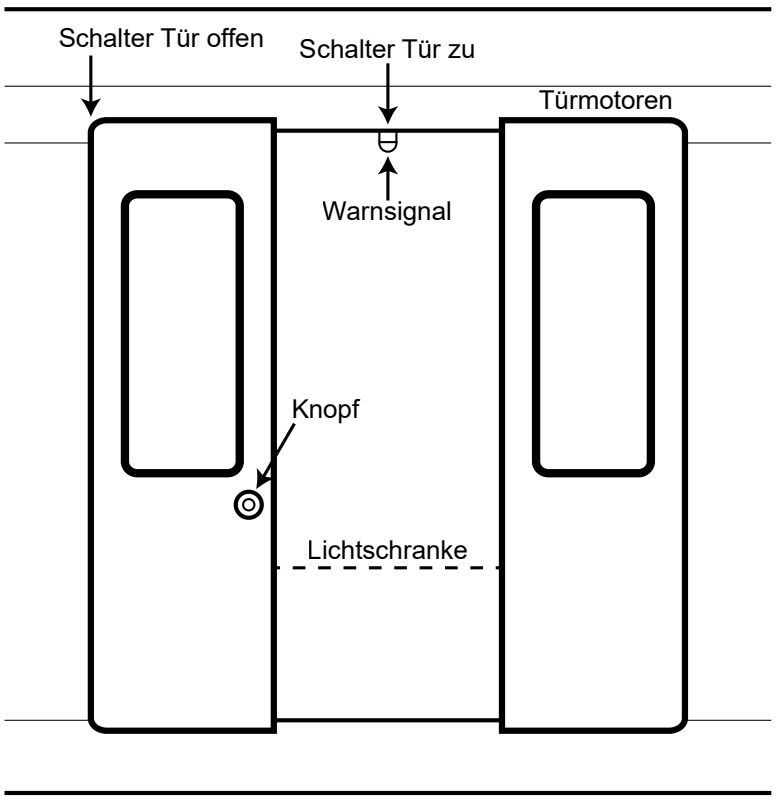
b) Entwerfen Sie einen äquivalenten Automaten des anderen Typs.

c) Vergleichen Sie das zeitliche Verhalten der beiden Automaten im folgenden Timing-Diagramm:



Übung 9.4 Entwurf endlicher Automaten am Beispiel der Steuerung einer Zugtür [25 min]

Die folgende Abbildung zeigt eine übliche Zugtür:



Die Steuerung der Zugtür soll folgendermaßen umgesetzt werden:

Im Startzustand ist die Tür gesperrt. Hält der Zug, so kann der Zugführer die Tür freigeben, indem er ein Freigabesignal **(F)** sendet. Ist die Tür freigegeben, aber noch geschlossen, so soll die grüne Beleuchtung des Knopfs an der Tür **(B)** aufleuchten. Ein Drücken des Knopfes **(K)** durch einen Passagier leitet dann das Öffnen der Tür ein. Um die Tür zu öffnen, muss an den Motoren ein entsprechendes Signal **(M1)** anliegen, bis ein Schalter **(T1)** signalisiert, dass die Tür vollständig offen ist. *Zur Vereinfachung können Sie annehmen, dass die Tür nicht automatisch wieder schließt und eine Freigabe der Tür nur einmal pro Halt stattfindet.*

Ist die Tür freigegeben, kann der Zugführer sie jederzeit wieder sperren, indem er ein Sperrsignal **(S)** sendet. Ist die Tür noch geschlossen, so lässt sich diese nun nicht mehr öffnen. Das Schließen der Tür geschieht ähnlich wie das Öffnen durch das Anlegen eines Signals **(M2)** an die Türmotoren, bis ein Schalter **(T2)** signalisiert, dass die Tür zu ist. Ist die Tür gerade noch dabei sich zu öffnen, so wird sie erst nach dem vollständigen Öffnen geschlossen.

Aus Sicherheitsgründen soll die Tür ein Warnsignal während des Schließens ausgeben **(W)**. Außerdem ist eine Lichtschranke **(L)** installiert, die solange eine 1 ausgibt, wie sie nicht unterbrochen wird. Die Tür darf nur geschlossen werden, wenn die Lichtschranke nicht unterbrochen wird, ebenso muss ein Unterbrechen der Lichtschranke während eines Schließvorgangs zum sofortigen Öffnen der Tür führen.
Ist die Tür gesperrt und geschlossen, soll dem Zugführer ein entsprechendes Signal gesendet werden **(Z)**.

a) Entwerfen Sie das FSM-Diagramm eines Moore-Automaten, der die Türsteuerung umsetzt.

b) Geben Sie die Zustandsübergangs- und die Ausgabetabelle an. Sie müssen die Zustände nicht kodieren.

Übung 9.5 One-Hot-Kodierung [5 min]

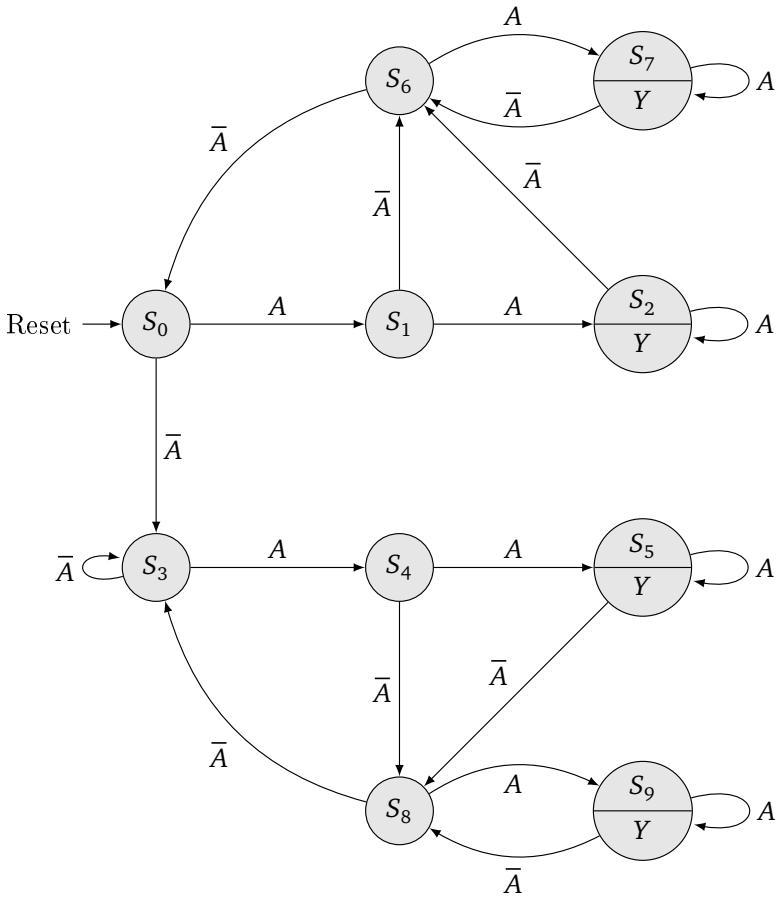
Neben der Kodierung als Binärzahl wird der Zustand von Automaten auch oft als One-Hot-Kodierung umgesetzt.

a) Wie viele Bits (und damit Speicherelemente) benötigt man für die Kodierung der Zustände eines Automaten mit 8 Zuständen bei einer Kodierung als Binärzahl? Wie viele bei einer One-Hot-Kodierung?

b) Welchen großen Nachteil und welchen großen Vorteil hat die One-Hot-Methode gegenüber einer Kodierung als Binärzahl?

Übung 9.6 Reduzierung der Anzahl der Zustände – Zusatzaufgabe

Der folgende Automat besitzt deutlich mehr Zustände als für seine Funktion unbedingt notwendig wären. Geben sie einen Moore-Automaten an, der dieselbe Funktion mit möglichst wenigen Zuständen umsetzt.



Digitaltechnik
Wintersemester 2022/2023
8. Übung

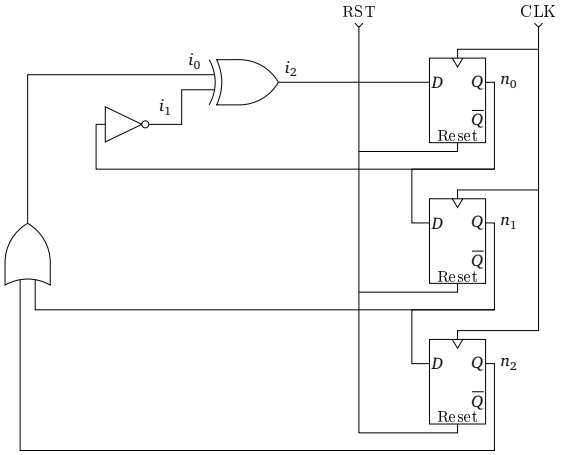
TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW50

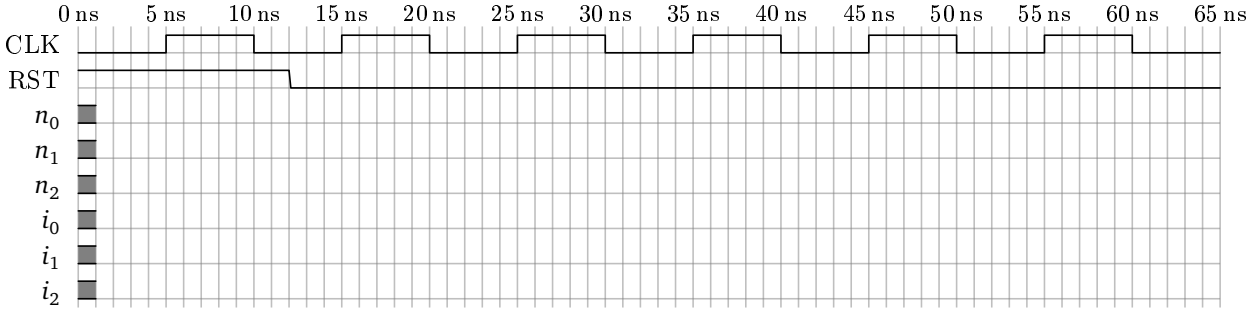
Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen. Die mit **"Zusatzaufgabe"** gekennzeichneten Aufgaben sind zur zusätzlichen Vertiefung für interessierte Studierende gedacht und daher nicht im Zeitumfang von 90 Minuten einkalkuliert.

Übung 8.1 Flip-Flops – Wiederholung/Vertiefung [15 min]

Gegeben ist folgende Schaltung mit synchron zurücksetzbaren D-Flip-Flops und die Verzögerungszeiten $t_{\text{pd,NOT}} = t_{\text{cd,NOT}} = 1\,\text{ns}, t_{\text{pd,OR}} = t_{\text{cd,OR}} = 3\,\text{ns}, t_{\text{pd,XOR}} = t_{\text{cd,XOR}} = 3\,\text{ns}, t_{\text{pd},D_{FF}} = t_{\text{cd},D_{FF}} = 2\,\text{ns}$.



a) Vervollständigen Sie das folgende Timing-Diagramm. Markieren Sie dabei auftretende Störimpulse (Glitches):



b) Was tut die Schaltung?

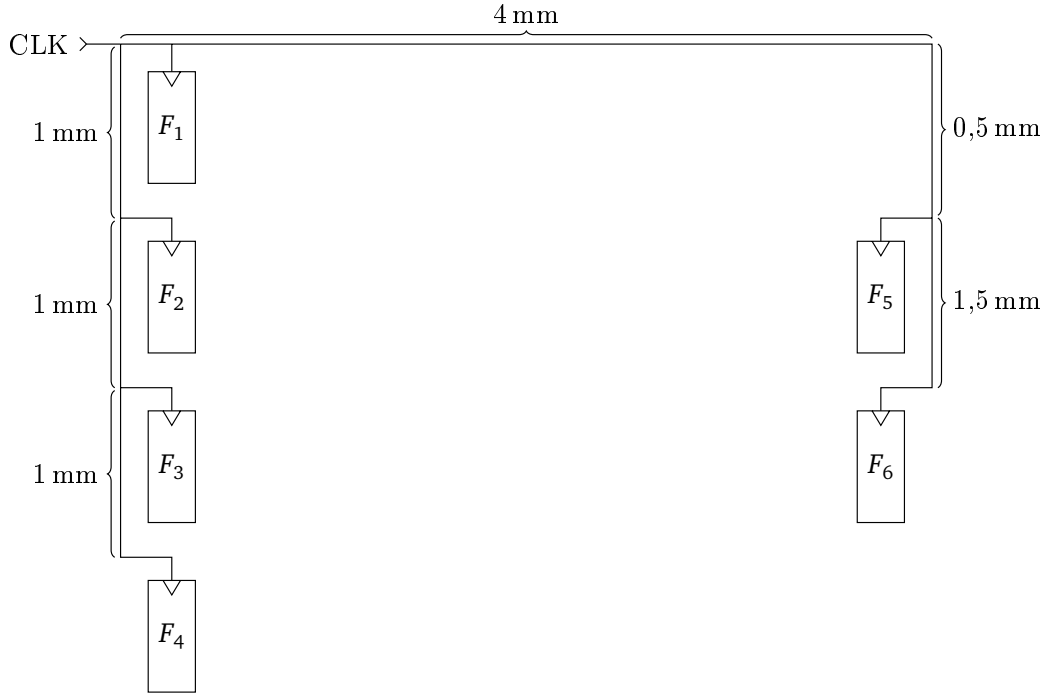
c) Müssen die gefundenen Störimpulse verhindert werden, damit die Korrektheit der Schaltung sichergestellt ist?

d) Wäre das Schaltverhalten identisch, wenn man die D-Flip-Flops durch D-Latches ersetzt? Begründen Sie Ihre Antwort.

Übung 8.2 Timing in synchronen sequentiellen Schaltungen [15 min]

a) Setzen Sie zunächst die folgenden Funktionen in der gegebenen Vorlage um. Die D-Flip-Flops F_1, F_2, F_3, F_4 geben dabei A, B, C, D aus; Y ist die Eingabe für F_5 und Z für F_6. Es stehen hierfür neben AND, OR und NOT auch NAND und NOR Gatter zur Verfügung. Verwenden Sie so wenig Gatter wie möglich.

$Y = \overline{BC + A} \quad Z = \overline{(BC + A) + D}$



b) Berechnen Sie, mit welcher Taktrate Ihre Schaltung maximal betrieben werden kann. Benutzen Sie dazu die folgenden Timing-Charakteristiken. Beachten Sie Leitungsverzögerung zunächst nicht.

1. $t_{cd,NOT} = 10\,ps$
2. $t_{pd,NOT} = 20\,ps$
3. $t_{cd,AND/OR} = 15\,ps$
4. $t_{pd,AND/OR} = 25\,ps$
5. $t_{cd,NAND/NOR} = 25\,ps$
6. $t_{pd,NAND/NOR} = 40\,ps$
7. $t_{ccq} = 20\,ps$
8. $t_{pcq} = 40\,ps$
9. $t_{setup} = 50\,ps$
10. $t_{hold} = 50\,ps$

c) Prüfen Sie, ob die Hold-Bedingung aller D-Flip-Flops erfüllt wird. Erweitern Sie gegebenenfalls Ihre Schaltung, um diese zu garantieren. Benutzen Sie dafür gewöhnliche Buffer, welche hier die selben Timing-Charakteristiken wie NOT Gatter besitzen.

d) Mit welcher maximalen Taktrate kann die Schaltung nun betrieben werden?

e) Beschreiben Sie, was man unter einer sogenannten Taktverschiebung (Clock-Skew) versteht und zu welchen Problemen diese eventuell führen kann. Kann der Clock-Skew als konstanter Wert angenommen werden?

f) Überlegen Sie, wie der Clock-Skew eine Takterhöhung ermöglichen kann.

g) Berechnen Sie die maximale Taktfrequenz unter Berücksichtigung des Clock-Skews. Gehen Sie von einer Signalausbreitungsgeschwindigkeit von $2 \cdot 10^8$ m/s aus.

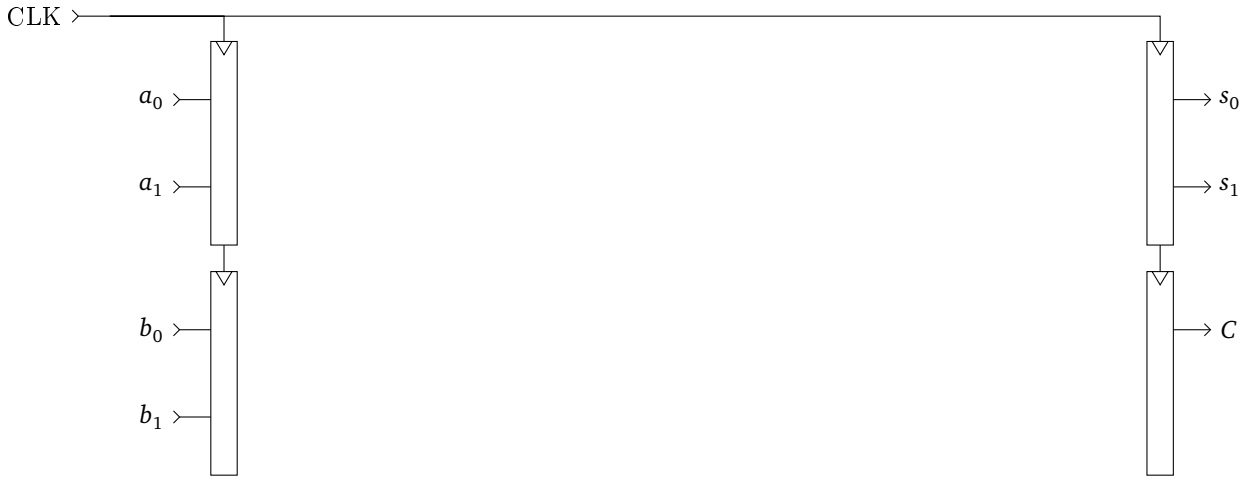
Übung 8.3 Parallelität [5 min]

a) Beschreiben Sie zunächst kurz den Unterschied zwischen zeitlicher und räumlicher Parallelität. Nennen Sie auch Vor- und Nachteile des jeweiligen Ansatzes

b) Wo begegnen Ihnen die Konzepte von zeitlicher und räumlicher Parallelität im Alltag? Nennen Sie Beispiele.

Übung 8.4 Beschleunigung mittels Pipelining [15 min]

a) Konstruieren Sie eine synchrone sequentielle Schaltung, die zwei 2 bit breite Zahlen A und B ($A := a_1a_0$, $B := b_1b_0$) addiert. Die Ausgabe besteht aus der Summe S ($S := s_1s_0$) und dem Übertrag C. Ergänzen Sie dafür folgende Vorlage mit möglichst wenigen XOR, AND und OR Gattern. Orientieren Sie sich am Vorgehen bei der schriftlichen Addition zur Herleitung der Formeln für s_0, s_1 und C.



b) Bestimmen Sie den Durchsatz und die minimale Latenz der Schaltung. Gehen Sie dabei von folgenden Timing-Charakteristiken aus:

$t_{cd,NOT} = 10\,ps$	$t_{cd,AND} = 20\,ps$	$t_{cd,OR} = 20\,ps$	$t_{cd,XOR} = 30\,ps$
$t_{pd,NOT} = 20\,ps$	$t_{pd,AND} = 40\,ps$	$t_{pd,OR} = 40\,ps$	$t_{pd,XOR} = 50\,ps$
$t_{ccq} = 5\,ps$	$t_{pcq} = 10\,ps$	$t_{setup} = 20\,ps$	$t_{hold} = 10\,ps$

c) Erweitern Sie die Schaltung mit Hinblick auf zeitliche Parallelität. Fügen Sie dazu zwei Pipeline-Stufen ein.

d) Bestimmen Sie den Durchsatz und die minimale Latenz der Schaltung mit Pipeline-Stufen.

e) Identifizieren Sie ein Problem das sich negativ auf die Latenz von Addierern für breitere Eingaben auswirkt.

Übung 8.5 Metastabilität [20 min]

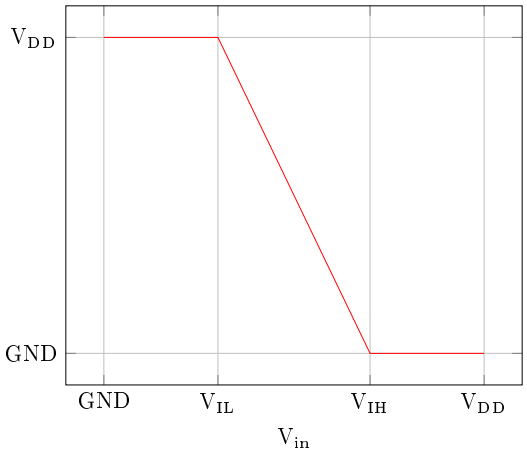
In der Vorlesung haben Sie bereits das SR-Latch basierend auf NOR-Gattern kennengelernt. Unter Umständen kann es hier zu Metastabilität kommen. Dies soll nachfolgend weiter untersucht werden.

a) Erklären Sie kurz, um welches Phänomen es sich bei Metastabilität handelt und unter welchen Umständen dieses auftreten kann.

b) Geben Sie die Wahrheitstabelle sowie eine Gatterschaltung für ein SR-Latch basierend auf NOR-Gattern an.

c) Wenn S und R inaktiv sind, verhält sich das SR-Latch wie eine bistabile Grundschaltung. Das Schaltverhalten kann in diesem Fall daher stark vereinfacht durch folgende Transferfunktion für Inverter beschrieben werden, welche üblicherweise für die Konstruktion einer solchen bistabilen Schaltung verwendet werden.

$$V_{\text{out}} = f(V_{\text{in}}) = \begin{cases} V_{\text{DD}} & \text{für } V_{\text{in}} \leq V_{\text{IL}} \\ (V_{\text{IH}} - V_{\text{in}}) \cdot \frac{V_{\text{DD}}}{V_{\text{IH}} - V_{\text{IL}}} & \text{für } V_{\text{IL}} < V_{\text{in}} < V_{\text{IH}} \\ \text{GND} & \text{für } V_{\text{in}} \geq V_{\text{IH}} \end{cases}$$



Nehmen Sie an, dass S und R aktiv sind. Was passiert, wenn S und R exakt zeitgleich inaktiv werden?

d) Sei $V_{DD} = 5\,V$, $V_{IL} = 1\,V$ und $V_{IH} = 2\,V$. Berechnen Sie V_Q und $V_{\overline{Q}}$ der entsprechenden Ausgänge Q und $\overline{Q}$ für den *metastabilen* Zustand der bistabilen Grundschaltung bzw. des SR-Latches.

Digitaltechnik
Wintersemester 2022/2023
7. Übung

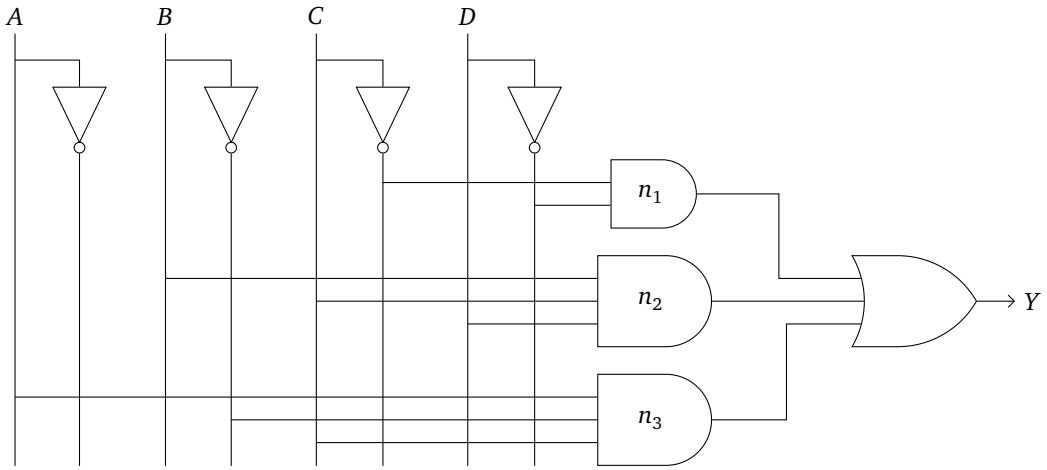
TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW49

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 7.1 Störimpulse – Wiederholung [20 min]

Gegeben sei die folgende Schaltung



Gehen Sie davon aus, dass jedes Gatter eine Verzögerungszeit von 5 ns hat.

Übung 7.1.1 Störimpulse erkennen

Identifizieren Sie die kritischen Eingangstransitionen (mit nur einer geänderten Variable) im Karnaugh Diagramm bei denen Störimpulse auftreten können.
Verifizieren Sie anhand von Timing-Diagrammen, ob bei diesen Transitionen tatsächlich Störimpulse auftreten.

Übung 7.1.2 Störimpulse beheben

Geben Sie eine funktional äquivalente Schaltung ohne Störimpulse an. Versuchen Sie dies mit möglichst wenigen Modifikationen der bestehenden Schaltung zu erreichen.

Übung 7.2 Addierer / Subtrahierer [25 min]

Übung 7.2.1 Ripple-Carry Addierer / Subtrahierer

Entwerfen Sie die Schaltung eines Ripple-Carry Addierers / Subtrahierers. Die beiden 4-bit breiten Zahlen A und B werden addiert, wenn am Eingang V eine 0 anliegt und subtrahiert, wenn am Eingang V eine 1 anliegt. Verwenden Sie dafür Volladdierer und die in der Vorlesung vorgestellten Gatter.

Übung 7.2.2 Überlauf- und Unterlauferkennung

Erweitern Sie Ihre Schaltung um ein Ausgangsbit F, welches angibt, ob bei der Addition / Subtraktion ein Überlauf / Unterlauf aufgetreten ist.

Übung 7.2.3 Analyse des Zeitverhaltens

Berechnen Sie die Ausbreitsungsverzögerung t_{pd} und Kontaminationsverzögerung t_{cd} Ihres 4 Bit breiten Addierers / Subtrahierers mit Überlauf- und Unterlauferkennung. Gehen Sie dabei davon aus, dass die verwendeten Volladdierer wie in der Vorlesung vorgestellt umgesetzt wurden. Benutzen Sie dafür die folgenden Zeitcharakteristiken:

a) $t_{cd,NOT} = 10\,ps$
b) $t_{pd,NOT} = 20\,ps$
c) $t_{cd,AND} = 20\,ps$
d) $t_{pd,AND} = 40\,ps$
e) $t_{cd,OR} = 20\,ps$
f) $t_{pd,OR} = 40\,ps$
g) $t_{cd,XOR} = 30\,ps$
h) $t_{pd,XOR} = 50\,ps$

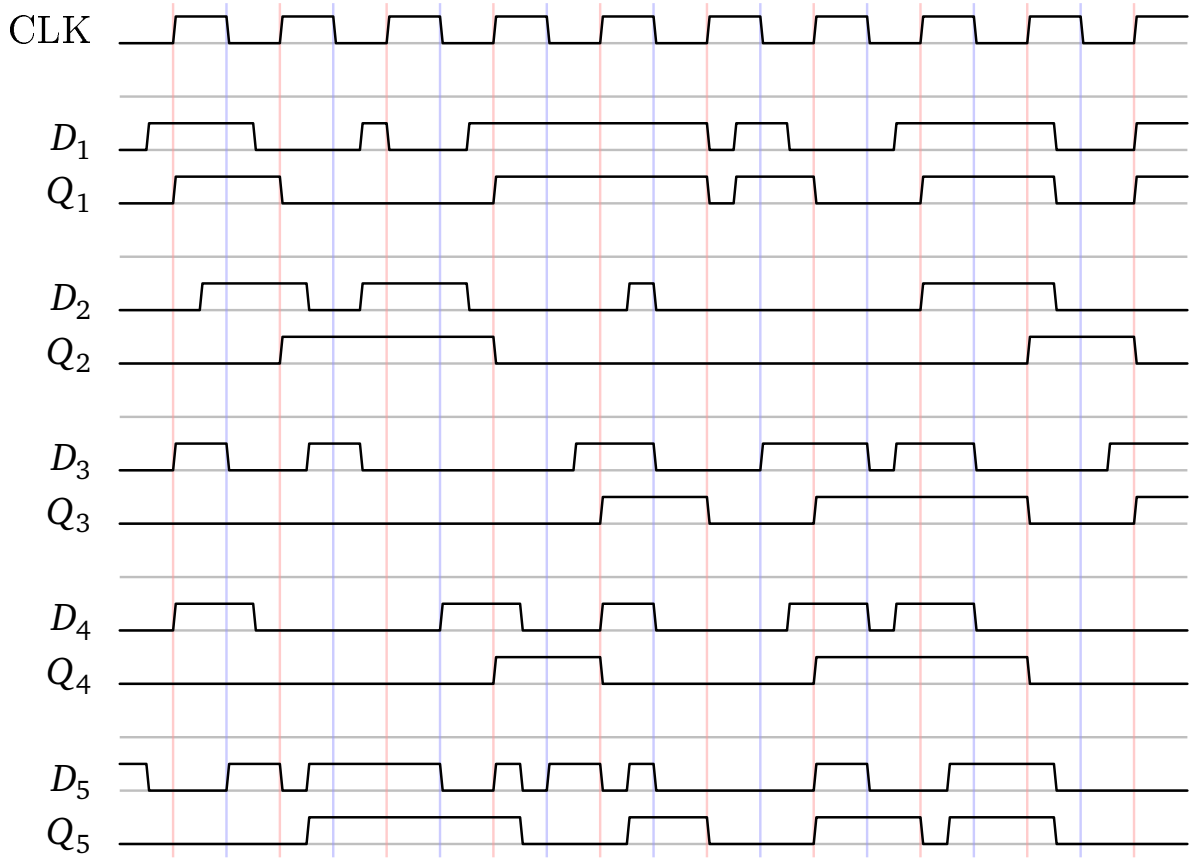
Übung 7.3 Timing-Diagramm: Flip-Flops vs Latches

EX8-2-1 EX8-2-2 EX8-2-3 EX8-2-4 EX8-2-5 EX8-2-6

[15 min]

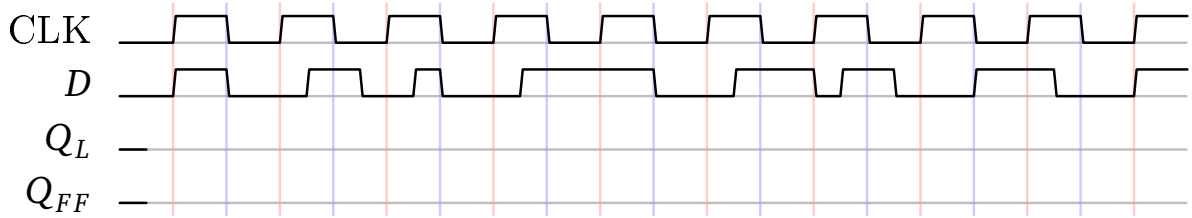
Übung 7.3.1 Am Schaltverhalten erkennen

Das folgende Timing-Diagramm beschreibt das Schaltverhalten von fünf Speicherelementen. Gegeben ist der Takteingang CLK, der Dateneingang D_i und der Datenausgang Q_i. Geben Sie an, welche Speicherelemente Flip-Flops und welche Latches sind. Markieren Sie Ihre Antwort im Diagramm und begründen Sie diese.



Übung 7.3.2 Schaltverhalten reproduzieren

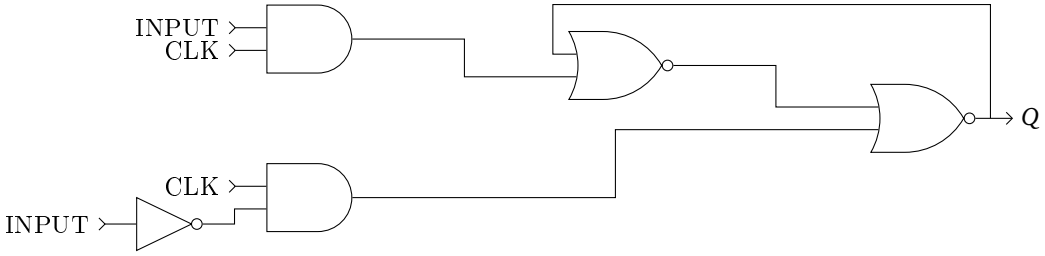
Ergänzen Sie im folgenden Timing-Diagramm das Schaltverhalten eines D-Latches mit Ausgang Q_L und eines D-Flip-Flops mit Ausgang Q_{FF}. Beide Speicherelemente werden vom gleichen Takt- (CLK) und Datensignal (D) gesteuert.



Übung 7.4 Flip-Flops und Latches [30 min]

Übung 7.4.1 An Logikgatterschaltung erkennen

Folgende Logikgatterschaltung beschreibt eines der in der Vorlesung vorgestellten Speicherelemente. Um welches Speicherelement handelt es sich und wie funktioniert dieses?



Übung 7.4.2 Toggle-Flip-Flops

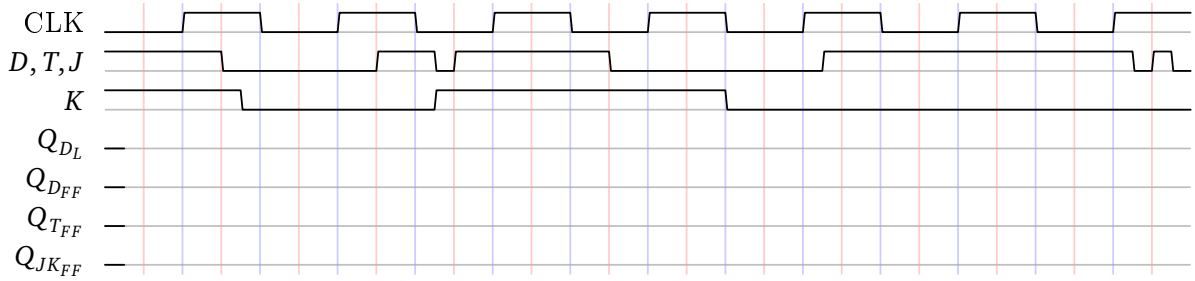
Entwerfen Sie ein T-Flip-Flop. Der Zustand Q wechselt bei jeder steigenden Taktflanke genau dann, wenn am Eingang T eine 1 anliegt und behält ansonsten seinen Wert – es gibt kein Datensignal D. Verwenden Sie zur Realisierung ein D-Flip-Flop und primitive Gatter (AND, OR, Inverter).

Übung 7.4.3 JK-Flip-Flops

Entwerfen Sie ein JK-Flip-Flop. Am Ausgang Q liegt nach steigender Taktflanke eine 1 an, wenn am Eingang J eine 1 und am Eingang K eine 0 anliegt. Sind die Eingänge $J = 0$ und $K = 1$, so liegt nach der steigenden Taktflanke eine 0 an Q an. Für $J = K = 0$ hält das JK-Flip-Flop seinen vorherigen Zustand. Im Fall $J = K = 1$ wechselt der Zustand von Q zu $\overline{Q}$. Verwenden Sie zur Realisierung ein D-Flip-Flop und primitive Gatter (AND, OR, Inverter).

Übung 7.4.4 Schaltverhalten reproduzieren

Ergänzen Sie im folgenden Timing-Diagramm das Schaltverhalten eines D-Latches mit Ausgang Q_{D_L}, eines D-Flip-Flops mit Ausgang $Q_{D_{FF}}$, eines T-Flip-Flops mit Ausgang $Q_{T_{FF}}$ und eines JK-Flip-Flops mit Ausgang $Q_{JK_{FF}}$.



Übung 7.5 2-Bit Zähler EX9-4-1 EX9-4-2 [15 min]

Entwerfen Sie einen Zähler, der eine 2-Bit Zahl B_1B_0 mit jedem Takt um 1 erhöht. Wird die höchste darstellbare Zahl erreicht, fängt der Zähler erneut bei 0 an. Verwenden Sie dafür 2 D-Flip-Flops und primitive Logikgatter. Beachten Sie, dass CLK ausschließlich mit den dafür vorgesehenen Inputs der Flip-Flops verbunden werden soll.
Hinweis: Für die minimalste Lösung wird neben den Flip-Flops nur 1 weiteres Gatter benötigt.

Digitaltechnik
Wintersemester 2022/2023
6. Übung

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann

KW48

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 6.1 Logikminimierung mit Karnaugh Diagrammen – Wiederholung EX4-4 [10 min]

Erstellen Sie für folgende Funktionen (in verkürzter Minterm/Maxterm-Schreibweise mit "Don't Cares") jeweils ein Karnaugh Diagramm. Markieren Sie die Primimplikanten und geben Sie einen minimalen boole'schen Ausdruck für die Funktion an.

a) $Y : (A, B, C, D) \mapsto m_2 + d_4 + m_5 + d_7 + m_{10} + d_{12} + d_{13}$

b) $Y : (A, B, C, D) \mapsto m_2 + d_3 + d_6 + d_7 + m_8 + d_9 + d_{10} + d_{11} + d_{12} + m_{13} + d_{14} + m_{15}$

Übung 6.2 Logikminimierung mit Espresso [15 min]

Übung 6.2.1 Eingabe

Erstellen Sie eine Espresso-Repräsentation für die Funktion $Y : (A, B, C, D) \mapsto d_0 + d_2 + m_3 + d_5 + m_9 + d_{13} + m_{14} + m_{15}$.

Übung 6.2.2 Ausgabe

Minimieren Sie Y mit Espresso. Wenden Sie dafür sowohl die Heuristik als auch das exakte Minimierungsverfahren an. Geben Sie den jeweils ermittelten boole'schen Ausdruck für Y an.

Übung 6.2.3 Qualität der Heuristik

Minimieren Sie nun die im Moodle verfügbare boole'sche Funktion (`U5.2.3.esp`). Vergleichen Sie die Laufzeit und das Ergebnis (Anzahl der resultierenden Implikanten) von Heuristik und exaktem Verfahren.

Übung 6.3 Vierwertige Logik [10 min]

Erstellen Sie die Wahrheitswertetabelle für $Y = A \oplus B$ in vierwertiger Logik ($A, B, Y \in \{X, 0, 1, Z\}$) unter Zuhilfenahme bekannter Resolutionstabellen.

Übung 6.4 Zeitverhalten kombinatorischer Schaltungen

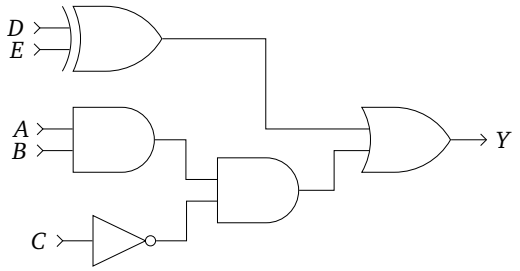
In dieser Aufgabe werden ausschließlich die folgendermaßen spezifizierten Gatter verwendet:

Gatter	AND	OR	NOT	XOR	AND3
t_{pd}	2 ns	2 ns	1 ns	3 ns	3 ns
t_{cd}	2 ns	2 ns	1 ns	2 ns	2 ns

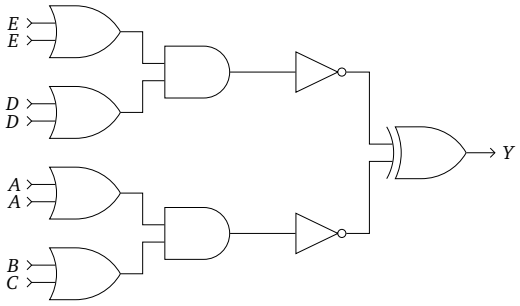
Übung 6.4.1 Kürzester und längster Pfad [10 min]

Berechnen Sie t_{pd} und t_{cd} für die folgenden Schaltungen.

a)

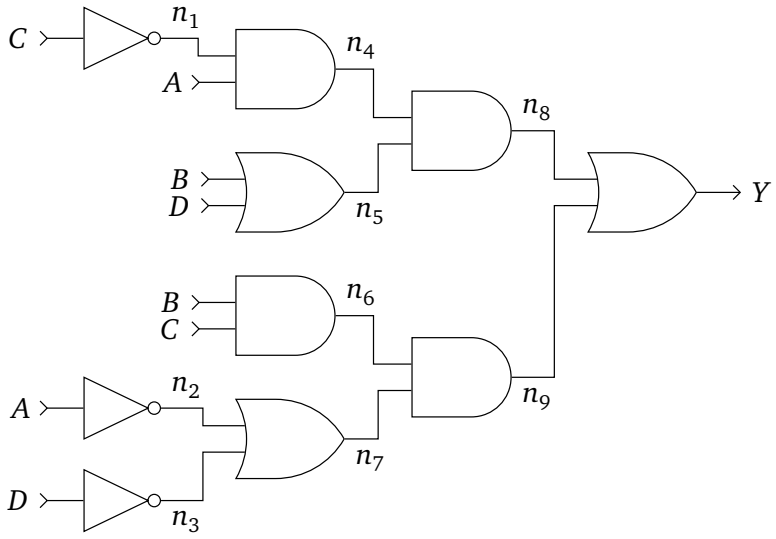


b)

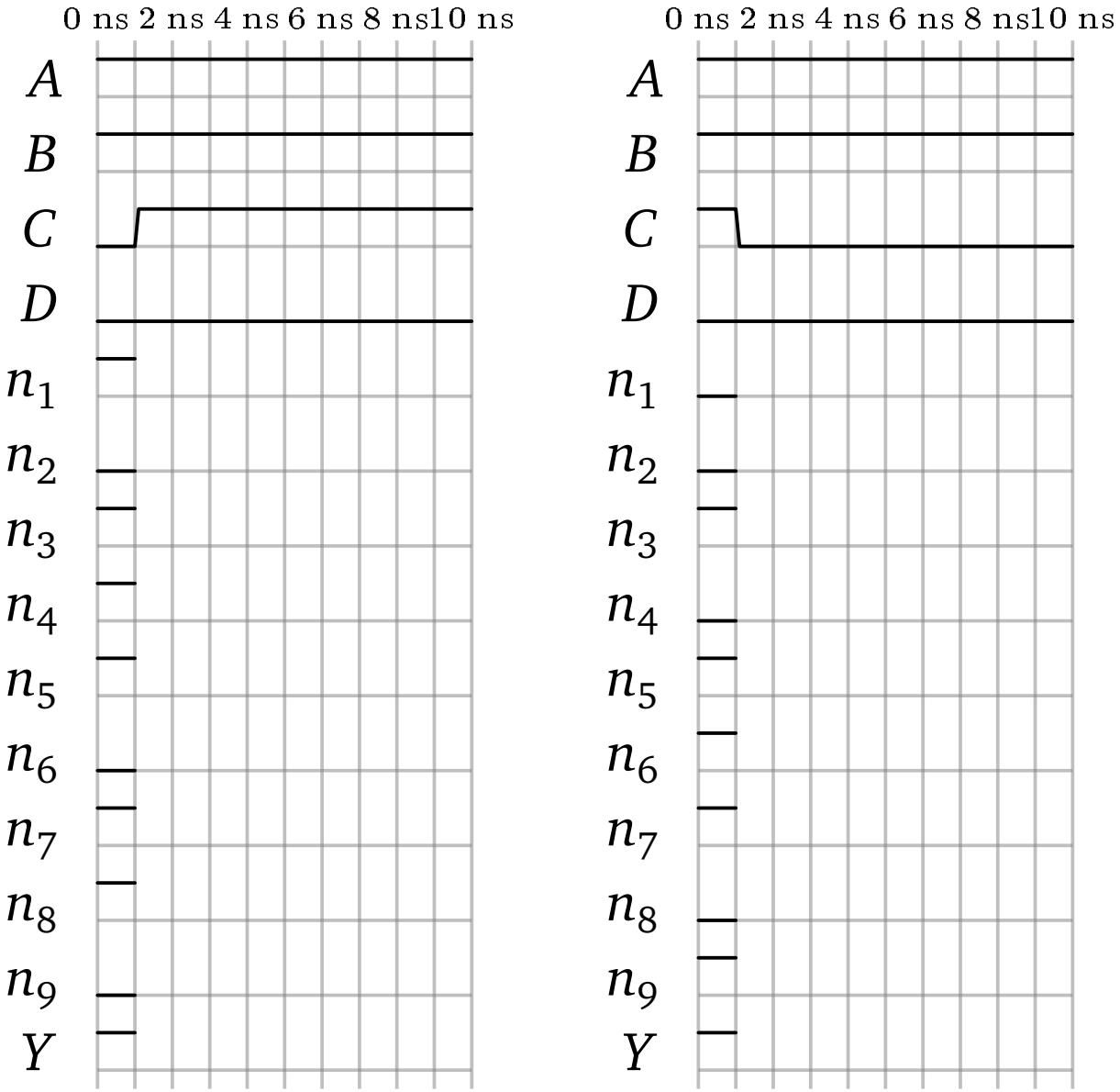


Übung 6.4.2 Störimpulse [30 min]

a) Tragen Sie die durch folgende Schaltung repräsentierte Funktion in ein Karnaugh Diagramm ein und markieren Sie alle Stellen, an denen Störimpulse auftreten *können*.



b) Ergänzen Sie den Zeitverlauf aller Knoten des Schaltnetzes in folgenden Diagrammen. Verwenden Sie dazu die zu Beginn dieser Aufgabe spezifizierten Gatterverzögerungszeiten. Treten Störimpulse auf?



$ABCD$: 1100 → 1110

$ABCD$: 1110 → 1100

c) Geben Sie nun einen funktional äquivalenten Ausdruck an, der keine Störimpulse enthält.

Digitaltechnik
Wintersemester 2022/2023
5. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW47

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen. Die mit **"Zusatzaufgabe"** gekennzeichneten Aufgaben sind zur zusätzlichen Vertiefung für interessierte Studierende gedacht und daher nicht im Zeitumfang von 90 Minuten einkalkuliert.

Übung 5.1 Normalformen und Theoreme der boole'schen Algebra – Wiederholung [15 min]

Die Funktion f bildet die vier Eingangsvariablen $A, B, C, D \in \mathbb{B}$ auf eine Ausgabevariable $Y \in \mathbb{B}$ ab. Die Ausgabe ist genau dann 1, wenn $A = 0$ gilt oder folgendes zutrifft:

Wenn $A = 1$ gilt, dann müssen mindestens zwei oder keine der verbleibenden Eingänge gleich 1 sein.

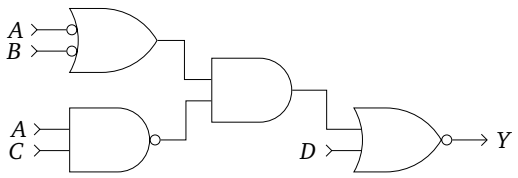
Erstellen Sie zunächst eine Wahrheitstabelle für die Funktion f. Geben Sie anschließend die konjunktive Normalform (KNF) der Funktion an. Formen Sie dann die KNF mit Hilfe der Rechenregeln der bool'schen Algebra in die äquivalente minimale Summe der Produkte (von Literalen) um. Geben Sie dabei für jeden Umformungsschritt das verwendete Axiom bzw. Theorem an. Realisieren Sie zum Abschluss die Summe als eine Gatterschaltung.

Übung 5.2 Bubble Pushing [15 min]

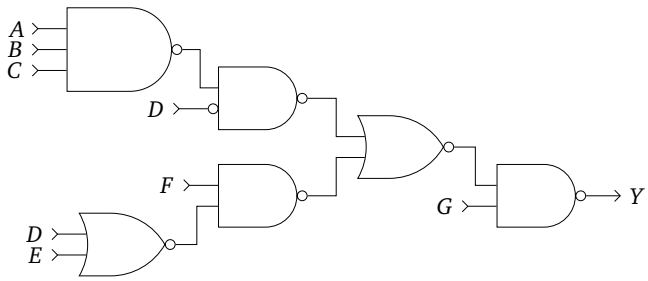
Verschieben Sie die Invertierungsblasen in den folgenden Schaltungen so weit wie möglich in die jeweils angegebene Richtung. Geben Sie die Funktion der umgeformten Schaltung zusätzlich als boole'schen Ausdruck an.

Übung 5.2.1 Vom Ausgang zu den Eingängen

a)

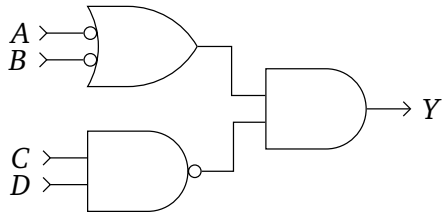


b)

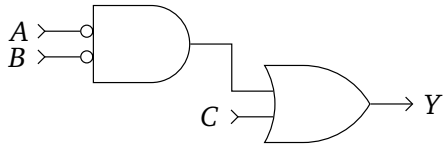


Übung 5.2.2 Von den Eingängen zum Ausgang

a)

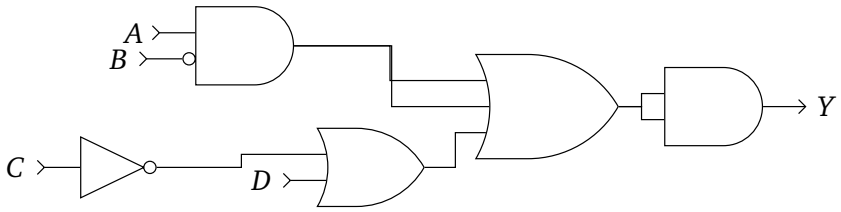


b)



Übung 5.2.3 Invertierung mittles Bubble Pushing

Geben Sie zunächst einen bool'schen Ausdruck an, welcher die folgende Gatterschaltung repräsentiert. Ermitteln Sie anschließend das Komplement dieser Funktion, indem Sie Bubble Pushing auf die Schaltung anwenden.



Übung 5.3 Logikrealisierung am Beispiel einer Majoritätsschaltung [20 min]

Übung 5.3.1 Zweistufige Logik EX2-2-9 EX2-2-10

Die Ausgabe der Funktion Y mit Inputs A, B, C, und D ist genau dann 1, wenn die Mehrheit ihrer Eingänge den Wert 1 haben. Geben Sie die DNF der Funktion f an und realisieren Sie diese mit zweistufiger Logik.

Übung 5.3.2 Multiplexer und Decoder EX7-1-2 EX7-1-3 EX7-1-6

a) Welchen Teil der zweistufigen Schaltung können Sie durch einen Decoder ersetzen? Setzen Sie die Schaltung mithilfe eines Decoders um.

b) Anstelle zahlreicher einzeln verdrahteter Gatter, werden in Rechnersystemen häufig Multiplexer (bzw. Look-Up-Tabellen) verwendet. Setzen Sie die Schaltung nun auch mithilfe eines Multiplexers um.

Übung 5.4 Karnaugh Diagramme EX4-2-1 EX4-2-2 EX4-2-3 EX4-2-4 [10 min]

Übung 5.4.1 Grafisch unterstützte Logikminimierung EX4-2-5 EX4-2-6 EX4-2-7

Minimieren Sie die folgenden Funktionen mit Hilfe von Karnaugh Diagrammen.

a) $Y = A\,\overline{C} \;+\; C\,A\,B \;+\; \overline{C}\,(\overline{B}\,\overline{A} + B)$

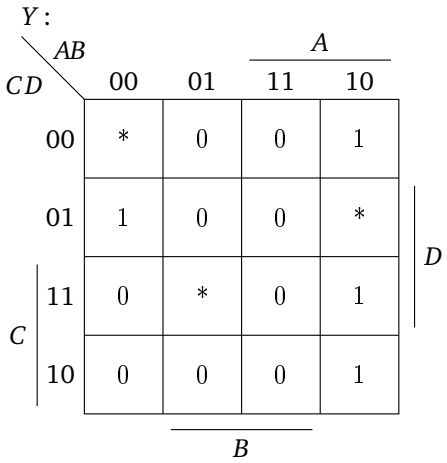
b) $Y = A\,\overline{D}\,\overline{B} \;+\; D(B\,A + \overline{B}\,\overline{A}) \;+\; \overline{A}\,\overline{B}\,(C\,\overline{D} \;+\; \overline{C}\,\overline{D})$

c) $Y = \; C(B\overline{D} + A(BD + \overline{D}\,\overline{B}) + D\,\overline{A}\,\overline{B}) + B\,\overline{A}\,\overline{D}\,\overline{C}$

Verwenden Sie "Don't Cares" spezifiziert durch die Funktion $X = A\,B\,\overline{D}\,\overline{C} \;+\; C(\overline{B}(\overline{A}\,\overline{D} \;+\; AD) \;+\; B\,D\,\overline{A})$.

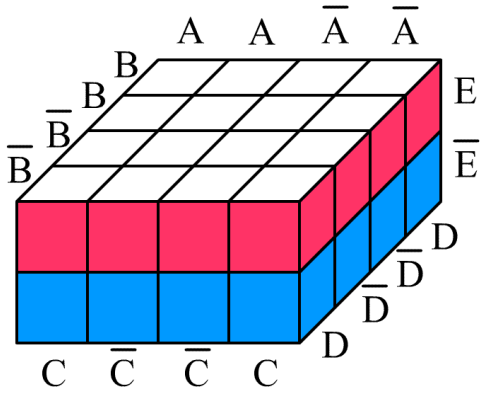
Übung 5.4.2 Extraktion einer KNF aus einem Karnaugh Diagramm

In der Vorlesung haben Sie bereits gelernt, wie Sie aus Karnaugh Diagrammen eine minimierte disjunktive Normalform extrahieren können. Überlegen Sie nun, wie Sie aus folgendem Karnaugh Diagramm eine entsprechend minimale konjunktive Normalform extrahieren könnten. Klären Sie auch die Rolle von Don't Cares in diesem Fall.



Übung 5.5 Karnaugh Diagramm für 5 Variablen – Zusatzaufgabe

Bisher haben wir gelernt, wie wir mittels Karnaugh Diagrammen bool'sche Gleichungen mit bis zu vier Variablen minimieren können. Tatsächlich bieten Karnaugh Diagramme auch die Möglichkeit, Ausdrücke mit mehr als vier Variablen zu minimieren. Exemplarisch wollen wir dies nun für einen bool'schen Ausdruck mit 5 Variablen durchführen. Hierzu müssen wir das Karnaugh Diagramm um eine weitere Schicht erweitern. Zwangsweise führt dies zu einer dreidimensionalen Struktur. Diese könnte etwa folgendermaßen aussehen:



Quelle: `https://de.wikibooks.org/wiki/Karnaugh-Veitch-Diagramm`

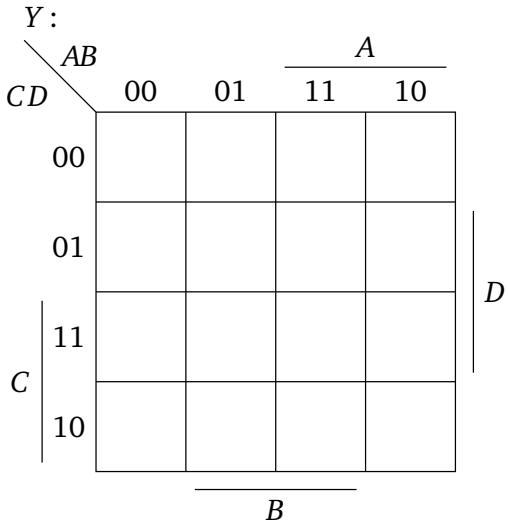
Wir versuchen in diesem Block nun Quader mit 2^k Einträgen zu finden, die nur 1 und * enthalten. Die Quader müssen, wie auch die Rechtecke bei Karnaugh Diagrammen, hierbei so groß wie möglich sein. Unser Ziel ist es, alle Einsen mit so wenigen Quadern wie möglich zu überdecken.

Minimieren Sie mit diesem Verfahren die folgende Gleichung. Nutzen sie dafür die gegebenen Karnaugh Diagramme.

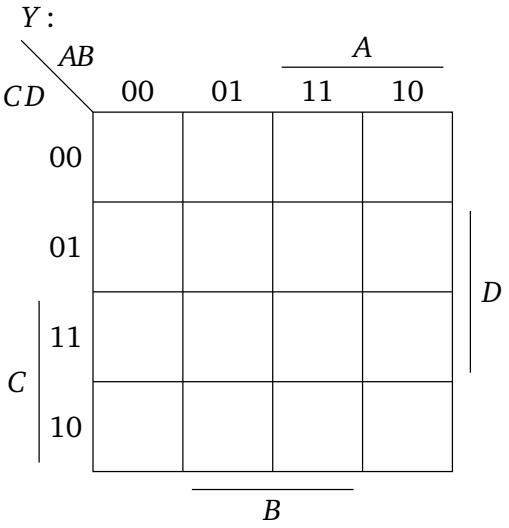
$$Y = \overline{A}\,\overline{B}\,C\,\overline{D} + A\,D\,E\,\overline{B} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{D}\,E + A\,\overline{C}\,D\,E + B\,D\,\overline{E}$$

Verwenden Sie auch folgende Don't Cares:

$$X = \overline{A}\,\overline{B}\,\overline{C}\,D\,\overline{E} + A\,B\,D\,C\,E + \overline{A}\,B\,D\,E$$



Für E



Für $\overline{E}$

Digitaltechnik
Wintersemester 2022/2023
4. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW46

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 4.1 Logikgatter und Boole'sche Algebra EX3-1-1 [10 min]

a) Zeichnen Sie eine Logikgatterschaltung, die aus den Signalen $A, B \in \mathbb{B}$ das Ergebnis $F = \overline{A\,B}$ berechnet, und ausschließlich aus NOR-Gattern mit je zwei Eingängen besteht.

b) Zeichnen Sie eine Logikgatterschaltung, die aus den Signalen $A, B, C \in \mathbb{B}$ das Ergebnis $F = A \oplus B \oplus C$ berechnet, und ausschließlich aus NOR-Gattern mit je zwei Eingängen besteht.

Übung 4.2 Transmissiongatter [10 min]

Jede kombinatorische Schaltung lässt sich als Schaltnetz aus Transmissionsgattern darstellen. Analog zu CMOS-Schaltungen müssen dabei immer zwei komplementäre Pfade realisiert werden:

- der 1-Pfad von V_{DD} zum Ausgang
- der 0-Pfad von GND zum Ausgang

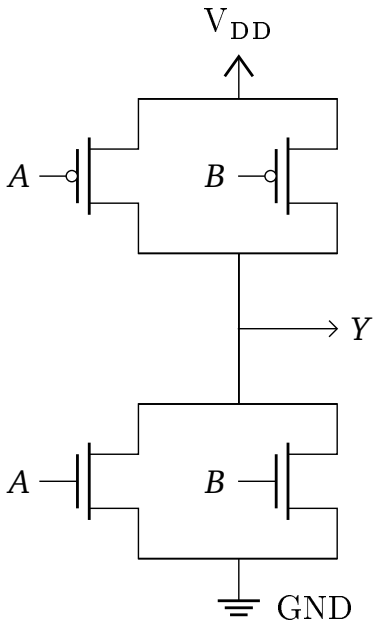
Bei jeder Kombination der Eingänge muss **genau einer** der beiden Pfade durchschalten, um den Ausgang auf eine logische 1 (V_{DD}) oder eine logische 0 (GND) zu ziehen. Im 1-Pfad entspricht eine Reihenschaltung der logischen Und-Verknüpfung der sequentiellen Teilschaltungen, während eine Parallelschaltung die logische Oder-Verknüpfung realisiert. Im 0-Pfad ist dies genau umgekehrt (komplementär). Die beiden Steuereingänge eines Transmissionsgatters (EN und $\overline{EN}$) müssen jeweils mit der positiven und negierten Form ein und desselben Eingangs beschaltet werden.
Realisieren Sie nun die folgenden Funktionen mit Transmissionsgattern:

a) $Y = (A\,B) + C$

b) $Y = (C + \overline{B} + \overline{E})\,(\overline{D}\,A)$

Übung 4.3 Fehlerhafte Schaltung [5 min]

Folgende Schaltung ist gegeben. Chips, die entsprechend diesem Enwurf hergestellt sind, brennen im Betrieb aus.



a) Geben Sie die Input Variationen an, für welche die Chips ausbrennen.

b) Gibt es Input Kombinationen, für die ein valider Output zustande kommt? Falls ja, geben Sie diese an.

c) Alle Eingänge werden jetzt nur mit Input A verbunden. Entsteht dadurch ein funktionsfähiger Chip?

Übung 4.4 Komplementbildung [15 min]

Vereinfachen Sie das Komplement der folgenden Ausdrücke mit Hilfe der Rechenregeln der boole'schen Algebra. Geben Sie für jeden Umformungsschritt das verwendete Axiom bzw. Theorem an.

a) $F = A + B(C\ D)$

b) $F = \overline{C} + (\overline{A}\ D + \overline{B}) + D\ (\overline{B\ \overline{B}} \oplus C)$

c) $F = A\ B + (\overline{C} + D\ \overline{A})\overline{B} + C\ \overline{D}$

Übung 4.5 Boole'sche Algebra EX3-1-2 EX3-1-3 EX3-1-4 EX3-1-5 [15 min]

Vereinfachen Sie die folgenden Ausdrücke mit Hilfe der Rechenregeln der boole'schen Algebra. Geben Sie für jeden Umformungsschritt das verwendete Axiom bzw. Theorem an.

a) $F = \overline{(\overline{A} + D)\ \overline{(B + \overline{C})}\ \overline{(\overline{C} + D)}}$

b) $F = \overline{A\ \overline{B}\ \overline{C + D}\ C\ A}$

c) $F = \overline{A}\ \overline{B}\ C + A\ \overline{B}\ \overline{C} + A\ \overline{B}\ C + A\ B\ \overline{C} + A\ B\ C$

Übung 4.6 Entwurf realer Schaltungen [15 min]

Übung 4.6.1 Lichtsteuerung für ein Kraftfahrzeug

Entwerfen Sie die digitale Schaltung einer einfachen Lichtsteuerung für ein Kraftfahrzeug. Die Schaltung hat die Eingänge L (links blinken), R (rechts blinken), W (Warnblinker) und E_I (Blinkimpuls) sowie die Ausgänge E_L (linker Blinker) und E_R (rechter Blinker). Folgende Spezifikationen soll die Schaltung erfüllen:

- Wenn der Eingang L bzw. R gesetzt ist, soll der linke bzw. rechte Blinker blinken.
- Wenn der Eingang W gesetzt ist, sollen beide Blinker blinken.
- Die Blinkimpulse liegen am Eingang E_I an, d.h. E_I ist oszillierend auf 0 und 1 gesetzt.
- Wenn der Ausgang E_L bzw. E_R gesetzt ist, leuchtet der linke bzw. rechte Blinker.

Übung 4.6.2 Redundantes Überwachungssystem

Die Deutsche Bahn benutzt während einer Zugfahrt ein redundantes Fehlersystem, welches eine Notbremsung im Falle eines betrieblichen Fehlers automatisch auslöst. Dafür simulieren drei Computer den weiteren Fahrtverlauf und geben ein Signal C_i ($1 \leq i \leq 3$) aus, welches bei gesetzter "0" einen Fehler signalisiert und ansonsten "1" ausgibt. Entwerfen Sie eine Schaltung, die die Eingänge C_1, C_2, C_3 und E_I (Blinkimpuls) erhält und die Ausgänge E_G (grüne LED), E_R (rote LED) und N (Notbremse) nach folgenden Spezifikationen setzt:

- die grüne LED (E_G) soll leuchten (logische "1"), wenn keiner der drei Computer einen Fehler ausgibt.
- die rote LED (E_R) soll leuchten (logische "1"), wenn genau einer der drei Computer einen Fehler ausgibt.
- die rote LED (E_R) soll blinken, wenn mindestens zwei der drei Computer einen Fehler ausgeben. Als Blinkimpuls soll der Eingang E_I verwendet werden.
- Die Notbremse (N) wird ausgelöst (logische "1"), wenn mindestens zwei der drei Computer einen Fehler ausgeben.

Digitaltechnik
Wintersemester 2022/2023
3. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW45

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 3.1 Zahlendarstellung, Arithmetik, und Bitbreitenerweiterung [5 min]

Führen Sie die folgenden Berechnungen im Zweierkomplement-System durch. Bilden Sie dafür das Zweierkomplement der Zahlen und erweitern Sie die Bitbreite, falls notwendig. Wandeln Sie das Ergebnis ins Dezimalformat und ins 12 bit Hexadezimalformat um.

a) $12_{10} - 16_{10}$

b) $7_{10} - 1_{10}$

Übung 3.2 Reelle Zahlen (Fortsetzung von Übung 2.5) [6 min]

Letzte Woche haben wir Dezimalzahlen in ihre Binärdarstellung und in 16-bit Festkommazahlen umgerechnet. In dieser Woche betrachten wir binäre Gleitkommazahlen nach IEEE 754 half precision Standard (1-bit Vorzeichen s, 5-bit Exponent e, 10-bit Mantisse f, bias $= 15$).[1]
Erinnerung zu Gleitkommazahlen: Darstellung $(-1)^s \cdot 1.f \cdot 2^{e-\text{bias}}$

Übung 3.2.1 Dezimal zu Binär umrechnen

Stellen Sie folgende Dezimalzahlen als IEEE 754 Gleitkommazahlen dar. Das Ergebnis der Umrechnung in das Binärsystem aus der letzten Woche ist hier bereits gegeben. Wenn Sie runden müssen, runden Sie auf falls die erste wegfallende Binärstelle 1 ist, sonst runden Sie ab.
In der letzten Woche haben wir bereits gesehen, dass bei der Darstellung als Festkommazahl bei der ersten Zahl 2 Nachkommastellen verloren gehen, während die zweite Zahl sogar auf 0 gerundet wird. Was beobachten Sie im Vergleich dazu bei den Gleitkommazahlen?

a) $13.603515625_{10} = 1101.100110101_2$

b) $-0.0029296875_{10} = -0.0000000011_2$

Übung 3.2.2 Binär zu Dezimal umrechnen

Rechnen Sie die Gleitkommazahl 1101001010000000_2 in eine Dezimalzahl um.

Übung 3.3 Fragen zu Festkommazahlen und Gleitkommazahlen [4 min]

Beurteilen Sie die Korrektheit der folgenden Aussagen.

a) Die Zweierkomplement-Darstellung wird auch in IEEE 754 Gleitkommazahlen verwendet.

b) Die Addition von positiven Festkommazahlen ist ähnlich einfach wie die von Zahlen in Zweierkomplement-Darstellung.

c) Die Addition von positiven Gleitkommazahlen ist ähnlich einfach wie die von Zahlen in Zweierkomplement-Darstellung.

d) Eine IEEE 754 single precision Gleitkommazahl kann 0.4 exakt darstellen.

[1] Wir betrachten hier half precision, damit wir nicht von Hand mit sehr langen Zahlen rechnen müssen. In der Realität werden von Computern meist single und double precision (32 und 64 bit) genutzt.

Übung 3.4 Logikgatter-Schaltungen [2 min]

Implementieren Sie die folgenden Funktionen mit Logikgattern:

a) $F = ((A\,B) \oplus C) + \overline{C\,D}$

b) $F = (A \oplus \overline{C}) + (\overline{A + B} \oplus (C\ D))$

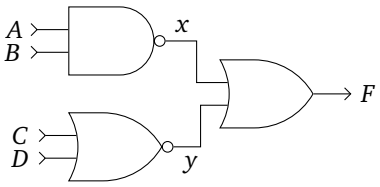
Übung 3.5 Logikgatter-Substitution NAND1 NAND2 NAND3 NAND4 NAND5 NAND6 NAND7 NAND8 [3 min]

a) Zeichnen Sie eine Logikgatterschaltung, die ein Signal $A \in \mathbb{B}$ invertiert, und ausschließlich aus NOR-Gattern besteht.

b) Realisieren Sie eine AND Schaltung und nutzen Sie dafür ausschließlich NOR-Gatter.

Übung 3.6 Wahrheitstabellen EX2-2-1 EX2-2-2 EX2-2-3 EX2-2-4 [2 min]

Stellen Sie die Wahrheitstabelle für die folgende Schaltung auf. Geben Sie dabei auch die Zwischenwerte x und y an.



A	B	C	D	x	y	F

Übung 3.7 XOR3 [5 min]

a) *XOR3* ist wie folgt definiert. Geben Sie die Wahrheitstabelle an, die diese Funktion charakterisiert.

$$XOR3 = (A \oplus B) \oplus C$$

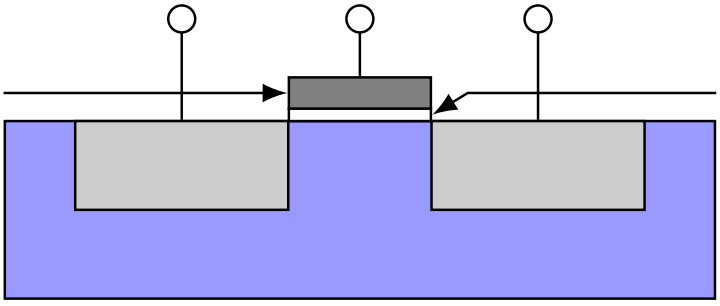
b) Zeigen Sie, dass die XOR Funktion die folgenden Eigenschaften besitzt:

1. Kommutativität

2. Assoziativität

Übung 3.8 Transistor Aufbau [2 min]

Die folgende Abbildung zeigt den Querschnitt eines Transistors. Um welchen MOSFET-Typ handelt es sich? Beschriften Sie alle wichtigen Elemente und geben Sie auch den Dotierungstyp der verschiedenen Halbleiterbereiche an. Beschreiben Sie die abstrakte Funktionsweise dieses Transistors.

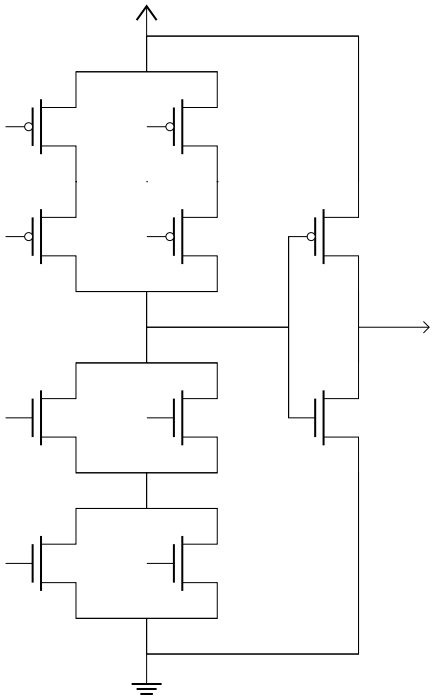


Übung 3.9 CMOS Logik [15 min]

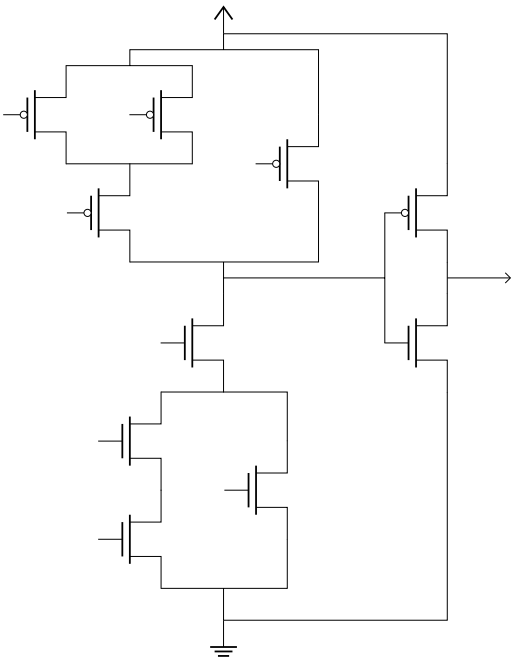
Übung 3.9.1 Schaltung beschriften

Beschriften Sie die folgenden CMOS-Schaltungen so, dass diese die angegebenen Funktionen realisieren. Ihnen stehen die Eingänge sowohl in positiver als auch negierter Form zur Verfügung. Beschriften Sie auch die Versorgungsspannungsleitungen und den Ausgang.

a) $Y = (A + D)\,(B + C)$

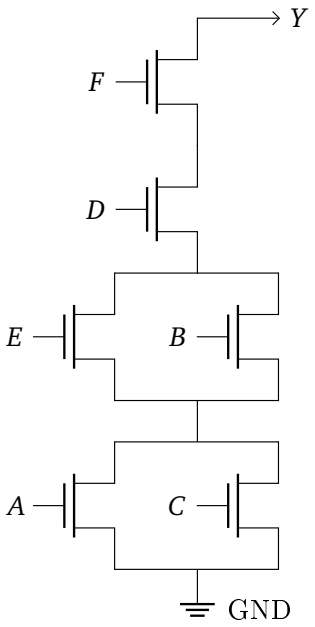


b) $Y = ((D\ B) + C)\,A$



Übung 3.9.2 Schaltung ergänzen

In der folgenden Aufgabe ist das pull-down Netz der Schaltung bereits gegeben. Vervollständigen Sie die Schaltung durch das Ergänzen des pull-up Netzes und geben Sie die implementierte Funktion an.



Übung 3.9.3 Schaltung entwerfen

Realisieren Sie die folgende Funktion als CMOS Schaltung. Ihnen stehen die Eingänge sowohl in positiver als auch negierter Form zur Verfügung.
$Y = ((\overline{A}\,\overline{C}) + (\overline{B}(\overline{D} + \overline{E})))$

Digitaltechnik
Wintersemester 2022/2023
2. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW44

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 2.1 Wertebereich binärer Zahlendarstellungen [15 min]

Der Wertebereich einer Funktion $f : A \to B$ auf der Eingabemenge A wird durch die Menge $f(A) = \{f(a) : a \in A\} \subseteq B$ charakterisiert. Diese Schreibweise wird in den folgenden beiden Teilaufgabe verwendet, um den Wertebereich der verschiedenen binären Zahlendarstellungen zu beschreiben. Die Angabe einer konkreten Binärdarstellung ist für beide Teilaufgaben nicht notwendig.

Übung 2.1.1 Minimale und maximale Werte

Tragen Sie in folgender Tabelle die minimal und maximal darstellbare Dezimalzahl ein.

	Vorzeichenlos: $u_{2,k}(\mathbb{B}^k)$		Vorzeichen und Betrag: $vb_{2,k}(\mathbb{B}^k)$		Zweierkomplement: $s_k(\mathbb{B}^k)$	
k	min	max	min	max	min	max
6						
8						
13						

Übung 2.1.2 Notwendige Bitbreite

Tragen Sie in folgender Tabelle die minimal notwendige Bitbreite k zur binären Darstellung der Dezimalzahlen n ein. Die beiden schwarz hinterlegten Felder bleiben frei.

Dezimal n	Vorzeichenlos $\min k \in \mathbb{N} : n \in u_{2,k}(\mathbb{B}^k)$	Vorzeichen und Betrag $\min k \in \mathbb{N} : n \in vb_{2,k}(\mathbb{B}^k)$	Zweierkomplement $\min k \in \mathbb{N} : n \in s_k(\mathbb{B}^k)$
301_{10}			
3_{10}			
512_{10}			
15_{10}			
-25_{10}			
-256_{10}			

Übung 2.2 Konvertierung zwischen Zahlendarstellungen EX5-10-1 EX5-10-3 [15 min]

Vervollständigen Sie die folgenden Tabellen. Nutzen Sie für die Konvertierung von der Dezimaldarstellung in die Binärdarstellung jeweils beide in der Vorlesung vorgestellten Konvertierungsverfahren. Geben Sie alle Ziffernfolgen jeweils mit der minimal möglichen Länge an.

Übung 2.2.1 Vorzeichenlose Zahlendarstellungen $u_{b,k}$

Dezimal	Binär	Hexadezimal	Octal
234_{10}			
	$1001\ 0100_2$		
		$1FC_{16}$	
125_{10}			

Übung 2.2.2 Zweierkomplement Darstellung s_k EX5-5

In der Vorlesung wurde die hexadezimale Darstellung für vorzeichenlose Zahlen behandelt. Die binäre Ziffernfolge der Darstellung mit Vorzeichen und Betrag und der Zweierkomplementdarstellung kann aber ebenfalls hexadezimal dargestellt werden. Ist die Bitbreite nicht festgelegt, muss man dabei eine “sign extension” auf eine durch 4 teilbare Bitbreite durchführen.
Zum Umrechnen von Dezimal- zu Zweierkomplement-Zahlen verwenden Sie die Methode “maximale Zweierpotenzen abziehen”.

Dezimal	Binär	Hexadezimal
15_{10}		
	$101\ 0011_2$	
		$1FC_{16}$
-148_{10}		
	$010\ 0001_2$	
		$C5_{16}$

Übung 2.3 Addition von Zweierkomplement-Zahlen EX5-6 [3 min]

Addieren Sie die folgenden Zweierkomplement-Zahlen. Geben Sie Summe und Summanden auch dezimal an. Tritt ein Überlauf auf?

	0	1	0	0	1	0	0	0
+	1	1	1	0	1	0	0	1

	0	1	1	1	1	1	0	1
+	0	0	0	1	1	0	0	1

Übung 2.4 Subtraktion von Zweierkomplement-Zahlen [6 min]

Wandeln Sie die folgenden Dezimalzahlen in 1 Byte breite Zweierkomplement-Zahlen um. Subtrahieren Sie die Binärdarstellungen voneinander, indem Sie den Minuend mit dem negierten Subtrahend addieren. Wandeln Sie das Ergebnis wieder ins Dezimalformat um. Hat die Subtraktion einen Überlauf verursacht?

a) $45_{10} - 70_{10}$

b) $-76_{10} - 57_{10}$

Übung 2.5 Reelle Zahlen [15 min]

In dieser Aufgabe betrachten wir binäre 16-bit Festkommazahlen (Vorzeichen und Betrag) mit 7 Nachkommastellen ($\ell = 7$), also 1-bit Vorzeichen, 8-bit Vorkommateil, 7-bit Nachkommateil.

Übung 2.5.1 Dezimal zu Binär umrechnen

Rechnen Sie folgende Dezimalzahlen zuerst in das Binärsystem um und geben Sie diese dann als binäre Festkommazahlen an. Wenn Sie runden müssen, runden Sie auf falls die erste wegfallende Binärstelle 1 ist, sonst runden Sie ab. Was beobachten Sie?

a) 13.603515625

b) −0.0029296875

Übung 2.5.2 Binär zu Dezimal umrechnen

Rechnen Sie die Festkommazahl 0000101101101000_2 in eine Dezimalzahl um.

Übung 2.6 Fragen zu Zweierkomplement-Zahlen [4 min]

Beurteilen Sie die Korrektheit der folgenden Aussagen.

a) Gegeben eine positive Zahl in Zweierkomplement-Darstellung (s_k), so ist ihre Darstellung als vorzeichenlose Binärzahl ($u_{2,k}$) identisch.

b) Gegeben eine positive Zahl als vorzeichenlose Binärzahl ($u_{2,k}$), so ist ihre Zweierkomplement-Darstellung (s_k) identisch.

c) Gegeben eine positive Zahl in Zweierkomplement-Darstellung (s_k), so ist ihre Darstellung mit Vorzeichen und Betrag ($vb_{2,k}$) identisch.

d) Im Gegensatz zur Vorzeichen und Betrag-Darstellung ($vb_{2,k}$) kann bei Zweierkomplementzahlen (s_k) das MSB nicht als Vorzeichen betrachtet werden.

e) In Zweierkomplement-Darstellung (s_k) wird die Zahl -1_{10} unabhängig von der Bitbereite immer durch eine Bitfolge dargestellt, in der alle Bits gesetzt sind.

Digitaltechnik
Wintersemester 2022/2023
1. Übung

TECHNISCHE UNIVERSITÄT DARMSTADT

M.Sc. Daniel Günther, M.Sc. Andreas Brüggemann KW43

Die Übungsblätter werden in den wöchentlichen Übungsstunden bearbeitet und diskutiert.
Mit der angegebenen Bearbeitungszeit für die einzelnen Aufgaben können Sie Ihren Leistungsstand besser einschätzen.

Übung 1.1 Informationsmengen [4 min]

Übung 1.1.1

Wie viele verschiedene Zustände lassen sich mit den folgenden Informationsmengen darstellen?

a) 13 bit

b) 1 Byte

c) 4 Nibble

Übung 1.1.2

Stellen Sie eine allgemeine Formel $f(x)$ auf, welche als Eingabe die Anzahl der Bits x erhält und die Anzahl der damit darstellbaren Zustände ausgibt. Geben Sie auch eine Formel an, welche die gleiche Ausgabe hat, aber die Anzahl der Nibble als Eingabe erhält.

Übung 1.1.3

a) Angenommen Sie wollen 65 537 Stunden eindeutig codieren. Wie viele Bits werden benötigt?

b) Angenommen Sie wollen 65 536 Studenten eindeutig codieren. Wie viele Bytes werden benötigt?

c) Angenommen Sie wollen alle rationalen Zahlen in $[0, 5]$ eindeutig codieren. Wie viele Bits werden benötigt?

Übung 1.2 Zählerüberlauf [5 min]

Nehmen Sie für diese Aufgabe an, dass ein Jahr 365,25 Tage hat.

Übung 1.2.1 Ventilator

Ein Ventilator dreht sich mit einer Rotationsgeschwindigkeit von 5 °/ms. Jedes mal wenn sich der Ventilator einmal komplett gedreht hat, erhöht dies einen 32 bit Zähler. Wie lange dauert es, bis der Zähler überläuft? Geben Sie das Ergebnis gerundet in Jahren an.

Übung 1.2.2 Schrittzähler

Geben Sie an, aus wie vielen Bits das Register eines Schrittzählers (in dem die Anzahl der gelaufenen Schritte gespeichert wird) mindestens bestehen muss, um alle Schritte ohne Überlauf zu zählen.
Das Ergebnis kann variieren, je nach Annahme wie lange der Nutzer lebt und wie viele Schritte pro Tag gelaufen werden.

Übung 1.3 Einheitenvorsätze [4 min]

Vervollständigen Sie die folgenden Umrechnungen.

a) $100\,\text{MiByte} = ?\ \text{Nibble}$

b) $50\,\text{GHz} = ?\ \text{Hz}$

c) $32\,\text{Gibit} - 32\,\text{MiByte} = ?\ \text{bit}$

Übung 1.4 Zweierpotenzen [6 min]

a) Stellen Sie die folgenden Zahlen als Summen von Zweierpotenzen dar.

   1. 5_{10}
   2. 42_{10}
   3. 791_{10}

b) Ergänzen Sie die Summen aus der vorherigen Teilaufgabe so, dass jede Summe alle Zweierpotenzen beinhaltet, die kleiner sind als die bereits in der Summe enthaltene höchste Zweierpotenz. Damit das Ergebnis der Summe gleich bleibt, verwenden Sie den Koeffizienten 0 für alle neu in die Summe aufgenommenen Zweierpotenzen.

c) Verwenden Sie die Summen aus der vorherigen Teilaufgabe, um jeder der folgenden Zahlen ihre binäre Darstellung zuzuweisen.

   1. 5_{10}
   2. 42_{10}
   3. 791_{10}

Übung 1.5 Oktalsystem [3 min]

In dieser Aufgabe betrachten wir dreistellige Zahlen im Oktalsystem ($b = 8$).

a) Was ist 270_8 im Dezimalsystem?

b) Was ist die größte darstellbare Zahl (oktal und umgerechnet in das Dezimalsystem)?

c) Was ist die kleinste darstellbare Zahl?

d) Wie viele Zahlen sind darstellbar?

e) Was ist die niedrigstwertige Stelle (LSD) von 345_8?

Übung 1.6 Zusatzaufgaben

a) Erklären Sie kurz die Relation von Abstraktion und Schichtenmodell.

b) Beschreiben Sie Hierarchie und Modularität und wie diese in Verbindung zueinander stehen.

c) Erklären Sie warum das Binärsystem eine digitale Disziplin ist.