

Aufgabe 1: Schaltwerksanalyse

(40 Punkte)

Betrachten Sie die nachfolgende in VHDL beschriebene Architecture eines Schaltwerks.

```
architecture Behavioral of Automat is

type state_type is (S0, S1, S2, S3);
signal state : state_type := S0;
signal next_state : state_type := S0;

signal x : STD_LOGIC_VECTOR(1 downto 0);
signal y : STD_LOGIC_VECTOR(1 downto 0);

begin

x <= x1 & x0;

process_State_Logic : process (state, x)
begin
  case state is
    when S0 =>
      case x is
        when "00" => next_state <= S0;
        when "01" => next_state <= S1;
        when "10" => next_state <= S2;
        when "11" => next_state <= S3;
      end case;
    when S1 =>
      case x is
        when "00" => next_state <= S1;
        when "01" => next_state <= S2;
        when "10" => next_state <= S3;
        when "11" => next_state <= S0;
      end case;
    when S2 =>
      case x is
        when "00" => next_state <= S2;
        when "01" => next_state <= S3;
        when "10" => next_state <= S0;
        when "11" => next_state <= S1;
      end case;
    when S3 =>
      case x is
        when "00" => next_state <= S3;
        when "01" => next_state <= S0;
        when "10" => next_state <= S1;
        when "11" => next_state <= S2;
      end case;
  end case;
end process process_State_Logic;
```

```
process_State_Register : process (reset, clock, next_state)
begin
  if (reset = '1') then
    state <= S0;
  elsif rising_edge(clock) then
    state <= next_state;
  end if;
end process process_State_Register;

process_State_Output : process (state)
begin
  case state is
    when S0 => y <="00";
    when S1 => y <="01";
    when S2 => y <="10";
    when S3 => y <="11";
  end case;
end process process_State_Output;

y1 <= y(1);
y0 <= y(0);

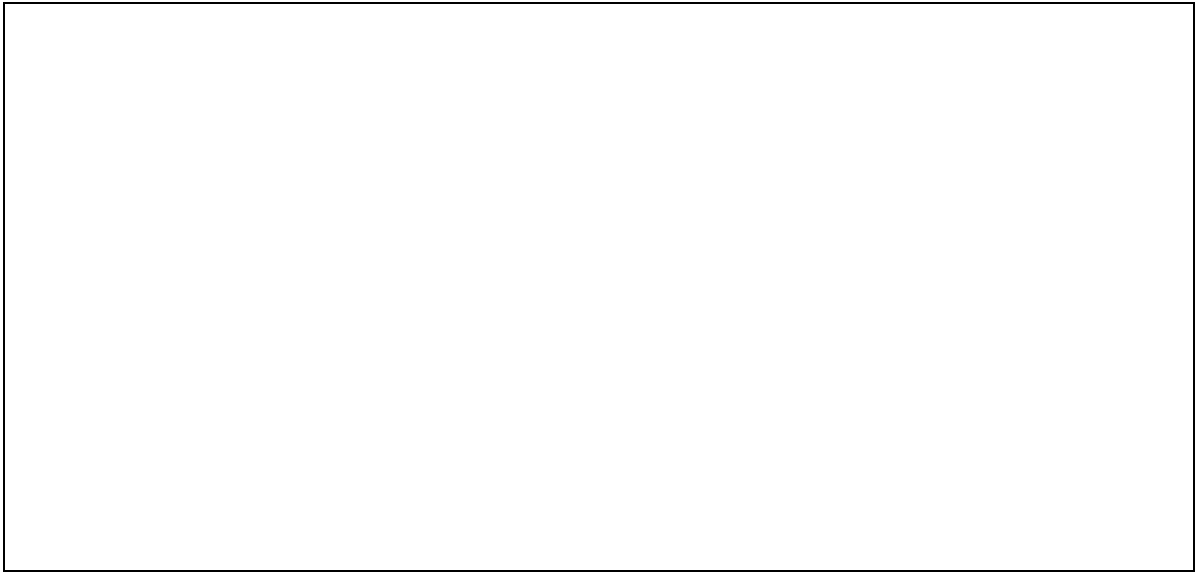
end Behavioral;
```

- a) Um welchen Automatentyp handelt es sich? Bitte begründen Sie Ihre Antwort kurz.


- b) Wie viele Flipflops werden für die Realisierung dieses Schaltwerks benötigt? Begründen Sie Ihre Antwort kurz.

- c) Welche Ausgabe erzeugt die taktsequentielle Eingabe von 00 – 01 – 11 – 10?

- d) Vervollständigen Sie die VHDL-Beschreibung des Schaltwerks, indem Sie die Entity angeben.



- e) Zeichnen Sie den Zustandsübergangsgraphen des Schaltwerks.



- f) Geben Sie die Ausgabefunktionen $y_{1,DMF}^n$ und $y_{0,KMF}^n$ des Schaltwerks an. Als Zustandskodierung ist die Binärdarstellung der Zustandsnummern zu verwenden ($S_0 = 00$, $S_1 = 01$, $S_2 = 10$, $S_3 = 11$).

Betrachten Sie ab hier für den Rest der Aufgabe 1 das nachfolgende in Form des Zustandsgraphen in Abbildung 1 gegebene Schaltwerk. Als Zustandskodierung ist die Binärdarstellung der Zustandsnummern zu verwenden.

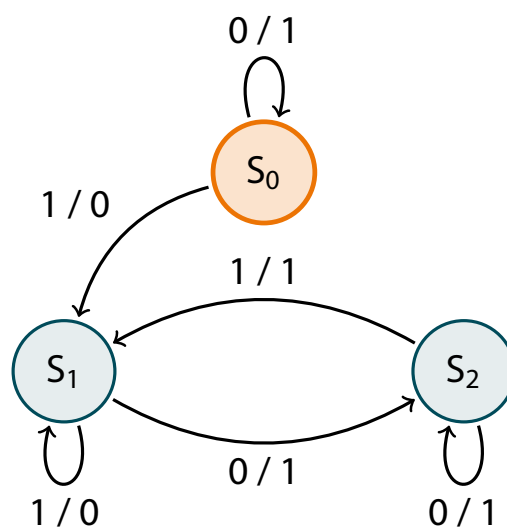


Abbildung 1: Zustandsgraph mit Startzustand S_0

- g) Um welchen Automatentyp handelt es sich? Bitte begründen Sie Ihre Antwort kurz.

- h) Vervollständigen Sie die nachfolgende Übergangstabelle für das Schaltwerk aus Abbildung 1.

$(Z_1 Z_0)^n$	Folgezustand $(Z_1 Z_0)^{n+1}$		Ausgabe y	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
00				
01				
10				
11				

- i) Die Zustandsübergangsfunktion $Z_{1, \text{KKN}}^{n+1}$ des Schaltwerks aus Abbildung 1 sei gegeben als

$$Z_{1, \text{KKN}}^{n+1} = (Z_1 + Z_0 + x)^n (\bar{Z}_1 + Z_0 + \bar{x})^n (Z_1 + Z_0 + \bar{x})^n (Z_1 + \bar{Z}_0 + \bar{x})^n$$

Bestimmen Sie $Z_{1, \text{KMF}}^{n+1}$, indem Sie eine Minimierung mit Hilfe des nachfolgenden KV-Diagramms (1 Diagramm als Ersatz!) durchführen und anschließend $Z_{1, \text{KMF}}^{n+1}$ explizit angeben. Berücksichtigen Sie die sich aus h) ergebende „don't care“-Terme. Markieren Sie jeden zusammengefassten Term im KV-Diagramm, indem Sie ihn einkreisen.

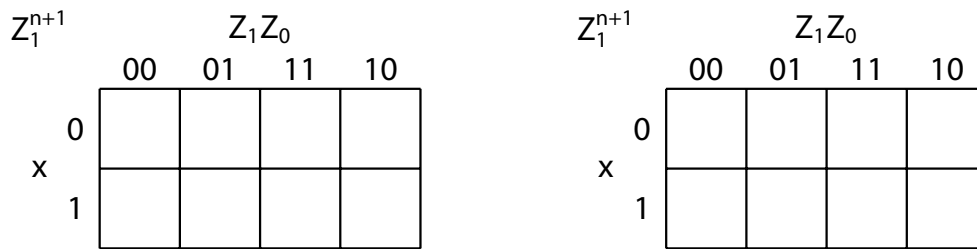


Abbildung 2: KV-Diagramme für die Minimierung von Z_1^{n+1} . Das rechte KV-Diagramm ist Ersatz!

- j) Die Zustandsübergangsfunktion $Z_{0, \text{DKN}}^{n+1}$ des Schaltwerks aus Abbildung 1 sei gegeben als

$$Z_{0, \text{DKN}}^{n+1} = (\bar{Z}_1 \bar{Z}_0 x + \bar{Z}_1 Z_0 x + Z_1 \bar{Z}_0 x)^n$$

Bestimmen Sie $Z_{0, \text{DMF}}^{n+1}$ auf Basis von $Z_{0, \text{DKN}}^{n+1}$ **sowie den sich aus der Übergangstabelle aus h) ergebende „don't care“-Termen** mithilfe des Verfahrens von Quine und McCluskey (QMC) und der nachfolgenden Tabelle. Nutzen Sie die in der VL gegebene Kodierung (nicht negierte Variable – 0, Negierte Variable – 1). Markieren Sie die don't care Terme indem Sie diese einklammern. Kennzeichnen Sie alle Primimplikanten mit einem Stern.

Klasse	Nr.	Minterme	Verschmolzene Terme (Nr.)	Neue Terme	Verschmolzene Terme (Nr.)	Neue Terme
K0	0	0000	0,2	00-0

Abbildung 3: Beispielhaft ausgefüllte erste Zeile einer QMC-Tabelle ohne Zusammenhang mit diesem Schaltwerk

Klasse	Nr.	Minterme	Verschmolzene Terme (Nr.)	Neue Terme	Verschmolzene Terme (Nr.)	Neue Terme

Warum ist eine Primimplikantentafel in diesem Beispiel überflüssig? Begründen Sie kurz und geben Sie $Z_{0,\text{DMF}}^{n+1}$ explizit an.

k) Verwenden Sie für das Schaltwerk aus Abbildung 1 die Zustandsübergangsfunktionen

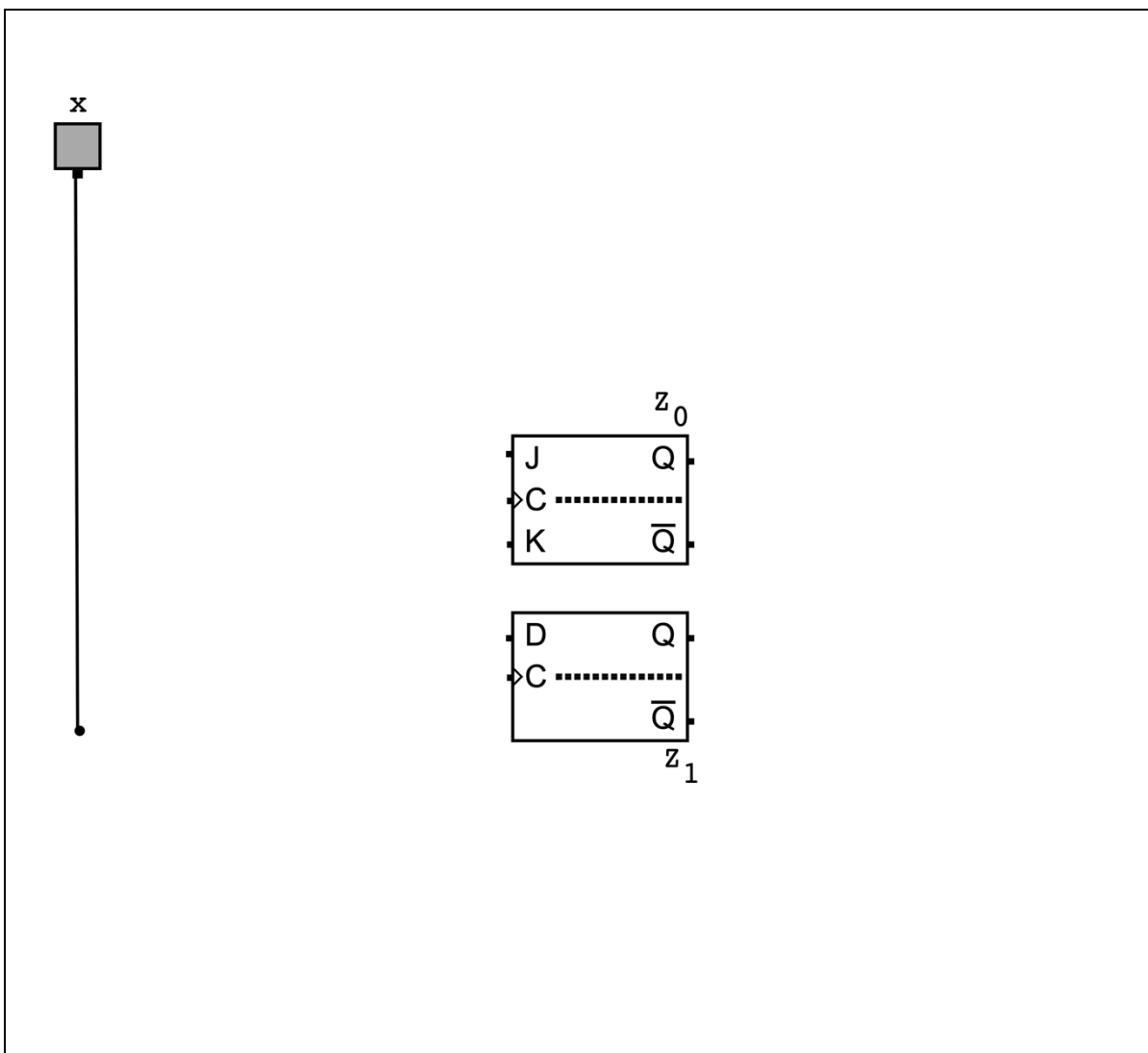
$$\begin{aligned} Z_{0,\text{KMF}}^{n+1} &= x \\ Z_{1,\text{DMF}}^{n+1} &= Z_0^n \bar{x} + Z_1^n x \end{aligned}$$

und bestimmen Sie die Ansteuergleichung für ein D-Flipflop unter der Bedingung, dass es in einer Realisierung das Zustandsbit Z_1 repräsentieren soll.

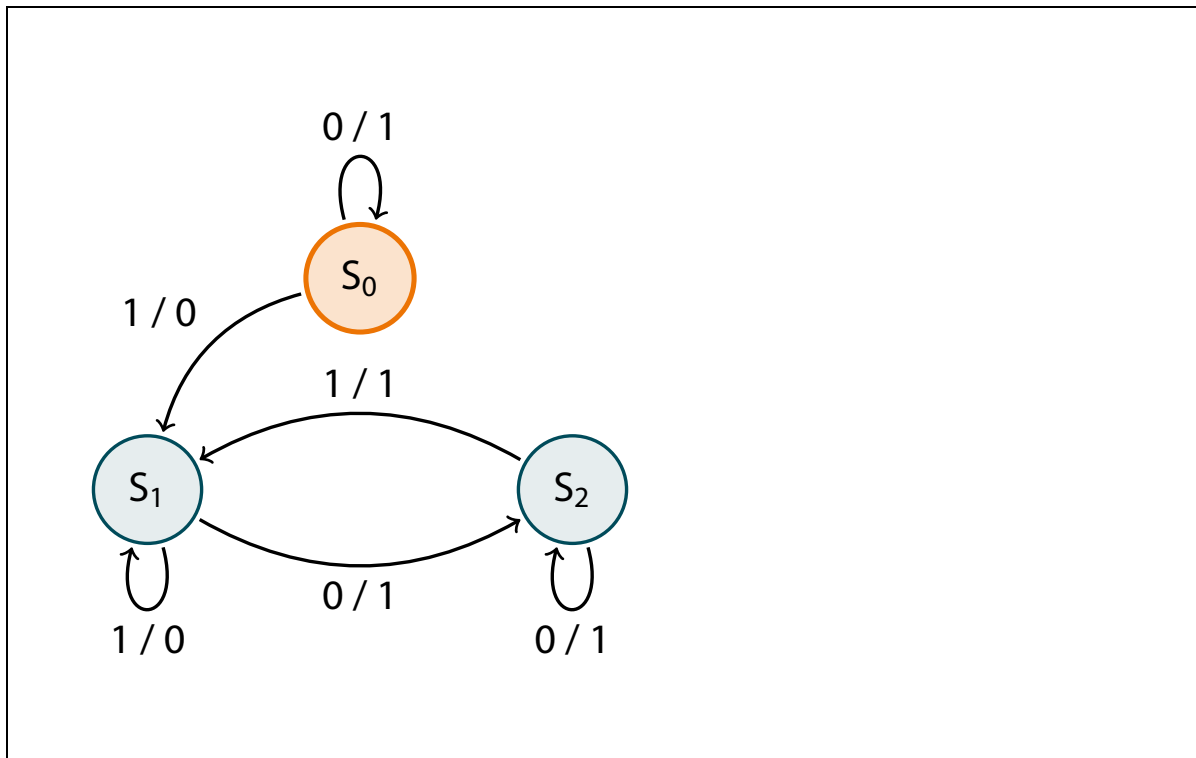
Was ist der Nachteil, wenn man $Z_{1,\text{DMF}}^{n+1}$ statt $Z_{1,\text{KMF}}^{n+1}$ als Ansteuergleichung für das D-Flipflop im Rahmen der Realisierung des Schaltwerks nutzt?

Bestimmen Sie auf Basis von $Z_{0,KMF}^{n+1}$ die Ansteuergleichung für ein JK-Flipflop unter der Bedingung, dass es in einer Realisierung das Zustandsbit Z_0 repräsentieren soll. Gibt es etwas bei der Verwendung Ihrer Ansteuergleichung zu berücksichtigen?

Realisieren Sie das Schaltwerk als LogicCircuits-Schaltung unter der Verwendung der eben bestimmten Ansteuergleichungen mithilfe eines taktflankengesteuerten D-Flipflops und eines taktflankengesteuerten JK-Flipflops und $y^n = \bar{x} + Z_1$.



- l) Ändern Sie das Schaltwerk soweit ab, dass es nicht mehr nur partiell definiert ist und im Anschluss keine „don't care“-Terme mehr enthält. Sie dürfen keine Zustände entfernen! Kanten dürfen verändert werden.



- m) Welche der nachfolgenden Aussagen sind für Ihren abgeänderten Automaten aus l) im Vergleich zur vorherigen Version wahr bzw. falsch? Bitte kreuzen Sie entsprechend an. Falsche Kreuze führen zu Punktabzügen, allerdings kann es keine negativen Punkte für diese Teilaufgabe geben.

wahr	falsch	Aussage
<input type="checkbox"/>	<input type="checkbox"/>	Die Anzahl der Zustände ändert sich nicht.
<input type="checkbox"/>	<input type="checkbox"/>	Es werden mehr Zustandsbits benötigt.
<input type="checkbox"/>	<input type="checkbox"/>	Die Ansteuergleichungen aller Flipflops bleiben gleich.
<input type="checkbox"/>	<input type="checkbox"/>	Eine Realisierung als Moore-Automat ist nicht mehr möglich.
<input type="checkbox"/>	<input type="checkbox"/>	Die Ausgabe des Automaten bei gleicher Eingabe ändert sich.
<input type="checkbox"/>	<input type="checkbox"/>	Für die Realisierung werden auf jeden Fall mehr Flipflops benötigt als bei der gegebenen Variante.
<input type="checkbox"/>	<input type="checkbox"/>	Es können keine JK-Flipflops mehr für die Realisierung genutzt werden.

$$\Sigma_{A1} = \underline{\hspace{2cm}} / 40 \text{ Punkte}$$

Aufgabe 2: Einfache CPU

(30 Punkte)

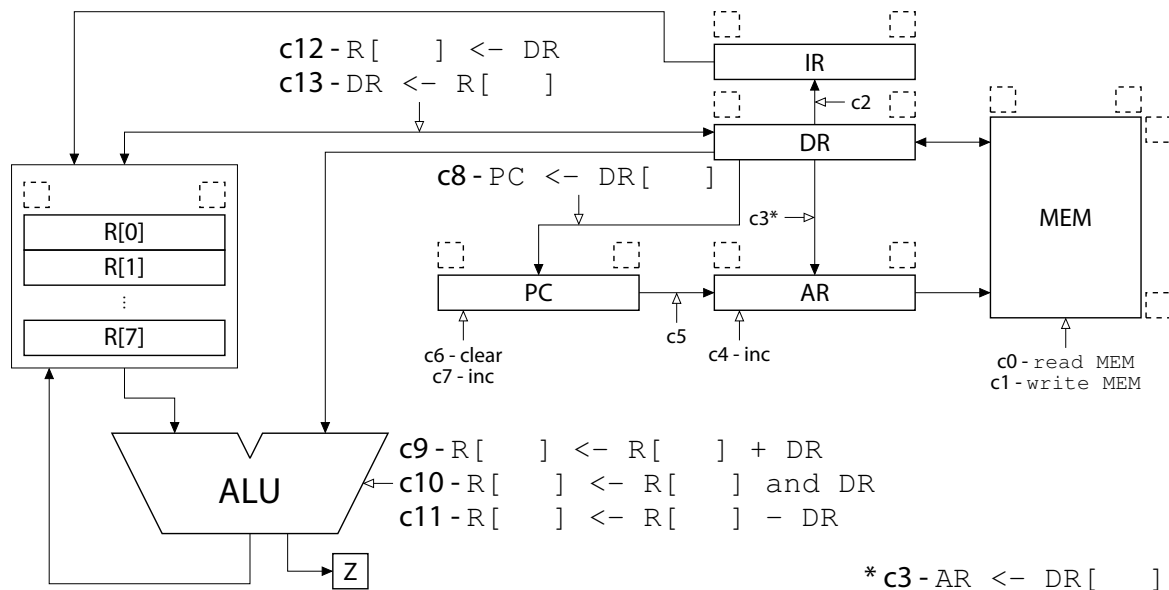


Abbildung 4: Operationswerk

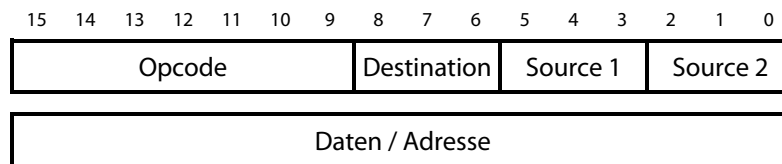


Abbildung 5: Befehlsformat

Gegeben sei das in Abbildung 4 dargestellte Operationswerk (OW) mit dem Speicher MEM, der bis zu 512 Werte aufnehmen kann. Das Befehlsformat der CPU ist in Abbildung 5 dargestellt. Sollte es sich bei dem Speicherinhalt nicht um einen Befehl handeln, so werden die kompletten 16 Bit in Abhängigkeit von der Befehlsausführung als Daten oder Adresse interpretiert und müssen von der ALU verarbeitbar sein. Die CPU stellt ein Flag zur Verfügung. Das Z-Flag gibt an, ob das Ergebnis der letzten Operation eine Null ($Z=1$) war.

Das Register des Register-Files mit dem Index 0 ($R[0]$) ist dauerhaft 0 und kann nicht überschrieben werden. Wird dennoch versucht, einen Wert in $R[0]$ zu schreiben, so bleibt dieses unverändert (kann also als „Papierkorb“ genutzt werden). Das ist seitens der Hardware realisiert und muss im RT-Code nicht berücksichtigt werden.

Die Indizierung der Register des Register-Files erfolgt direkt über das IR-Register. Die Deklaration und Verwendung des Register-Files in RT-Easy ist exemplarisch im nachfolgenden Codeausschnitt dargestellt.

```
# Deklaration eines Register-File mit vier Registern a 8 Bit
declare register array R(7:0)[4]
```

```
# Addition des Inhalts von R[3] auf den Inhalt von dem Register R mit dem
# Der 2 Bit Index steht in diesem Fall in INDEXREGISTER(3:2)
R[INDEXREGISTER(3:2)] <- R[INDEXREGISTER(3:2)] + R[3];
```

- a) Ergänzen Sie im OW aus Abbildung 4 die fehlenden Breitenangaben der einzelnen Register in den gestrichelten Kästchen. Jedes Register soll nur so breit sein, wie es notwendig ist. Sollten Sie einige Werte nicht direkt berechnen können genügt die Angabe als Potenz von 2 (beipiealsweise 2^2 statt 4). Ergänzen Sie zudem die fehlenden Indizes für die Zugriffe auf das Register-File. Nutzen Sie dabei D für Destination, S1 für Source1 und S2 für Source2. Beachten Sie, dass die Befehle aus Aufgabe b) ausführbar sein müssen.
- b) Das gegebene Operationswerk soll um ein mikroprogrammiertes Steuerwerk zu einer mikroprogrammierten CPU ergänzt werden, die über den nachfolgenden Befehlssatz verfügt.

Opcode	Befehl	Beschreibung
0	LOAD D X	Lädt den sich unter Adresse X im Speicher befindenden Wert in das durch den Destination-Teil des Befehls indizierte Register des Register-Files (R[D]).
1	STORE S2 X	Speichert den Inhalt von R[S2] unter der Adresse X im Speicher.
2	AND D S1 S2	Verundet den Wert R[S1] mit dem von R[S2] und speichert das Ergebnis in R[D].
3	JMP X	Setzt den Programmablauf an Adresse X fort.
4	ADDX D S1 X	Addiert den sich unter Adresse X im Speicher befindenden Wert auf den von R[S1] und speichert das Ergebnis in R[D].
5	CPBE S1 S2 X	Vergleicht den Wert von R[S1] mit dem von R[S2] und setzt den Programmablauf an Adresse X fort, falls die Werte gleich sind.

Beachten Sie hierbei, dass Befehle mit Adressen als Parameter Zweiwortbefehle sind.

Deklarieren Sie die benötigten Register, den Speicher und das Register-File und ergänzen Sie im nachfolgenden RT-Programm die Indizierungen sowie die zu den Registertransferoperationen korrespondierenden Kontrollsignale (rechts neben dem Code - wo keine Linie ist, soll auch kein Signal angegeben werden).

Implementieren Sie zudem die Befehle **ADDX** sowie **CPBE** unter Einhaltung des **Moore-Timings** und geben Sie die entsprechenden Kontrollsignale an.

Hinweis: Das Setzen des Flags ist im nachfolgenden Quellcode nicht implementiert. Dennoch muss es bei Bedarf von Ihnen genutzt werden.

```
# Deklarationen
```

```
INIT:  PC <- 0; _____  
FETCH: AR <- PC, PC <- PC + 1; _____  
      read MEM; _____  
      IR <- DR | switch IR(      ) { _____  
      case 0: goto LOAD  
      case 1: goto STORE  
      case 2: goto AND  
      case 3: goto JMP  
      case 4: goto ADDX  
      case 5: goto CPBE  
      default: goto FETCH };  
LOAD:  AR <- PC, PC <- PC + 1; _____  
      read MEM; _____  
      AR <- DR(      ); _____  
      read MEM; _____  
      R[      ] <- DR | goto FETCH; _____  
STORE: ... # Muss hier nicht weiter betrachtet werden  
AND:   DR <- R[      ]; _____  
      R[      ] <- R[      ] and DR | goto FETCH; _____  
JMP:   AR <- PC; _____  
      read MEM; _____  
      PC <- DR(      ) | goto FETCH; _____  
ADDX:  # Muss von Ihnen realisiert werden
```

CPBE: # Muss von Ihnen realisiert werden

- c) Erstellen Sie ein horizontal mikroprogrammiertes Steuerwerk, welches das durch den RT-Code in b) spezifizierte Verhalten inklusive **Ihres CPBE** Befehls realisiert. Füllen Sie hierfür die Tabelle auf der nachfolgenden Seite aus. Leere Felder werden hierbei als 0 interpretiert. Sie müssen also nur die 1en eintragen. Ergänzen Sie zudem den für das Mikroprogramm eventuell abgeänderten RT-Code des CPBE Befehls. Es stehen Ihnen **ausschließlich** die nachfolgenden Condition Select Signale zur Verfügung:

Condition Select	Funktion
00	Nicht springen
01	Springe zu der dekodierten Opcode-Adresse (Mapping-ROM)
10	Springe, falls Z = 0
11	Springe unbedingt

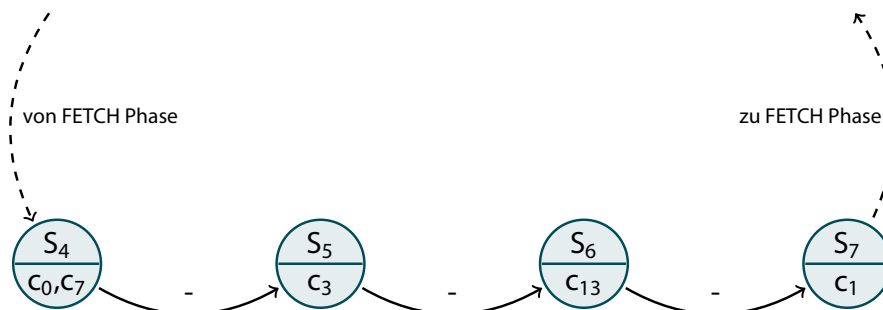
- d) Geben Sie das Mapping ROM an.

[illegible]

- e) Nennen Sie einen Vor- und einen Nachteil der Realisierung des Steuerwerks als horizontales Mikroprogramm im Vergleich zu einem vertikalen Mikroprogramm.

- f) Stimmt es, dass die Realisierung als Mikroprogramm mit dem mehrfachen Befehlsformat genauso viele Zeilen hat, wie die Realisierung im vertikalen Mikrobefehlsformat? Begründen Sie Ihre Antwort.

- g) Betrachten Sie den gegebenen Automaten, der die Ausführungsphase eines Befehls der CPU dieser Aufgabe beschreibt. Was macht dieser Befehl? Geben Sie die Antwort in einem Satz!



$$\Sigma_{A2} = \underline{\hspace{2cm}} / 30 \text{ Punkte}$$

Aufgabe 3: RISC-V und EduCore-V Tiny (24 Punkte)

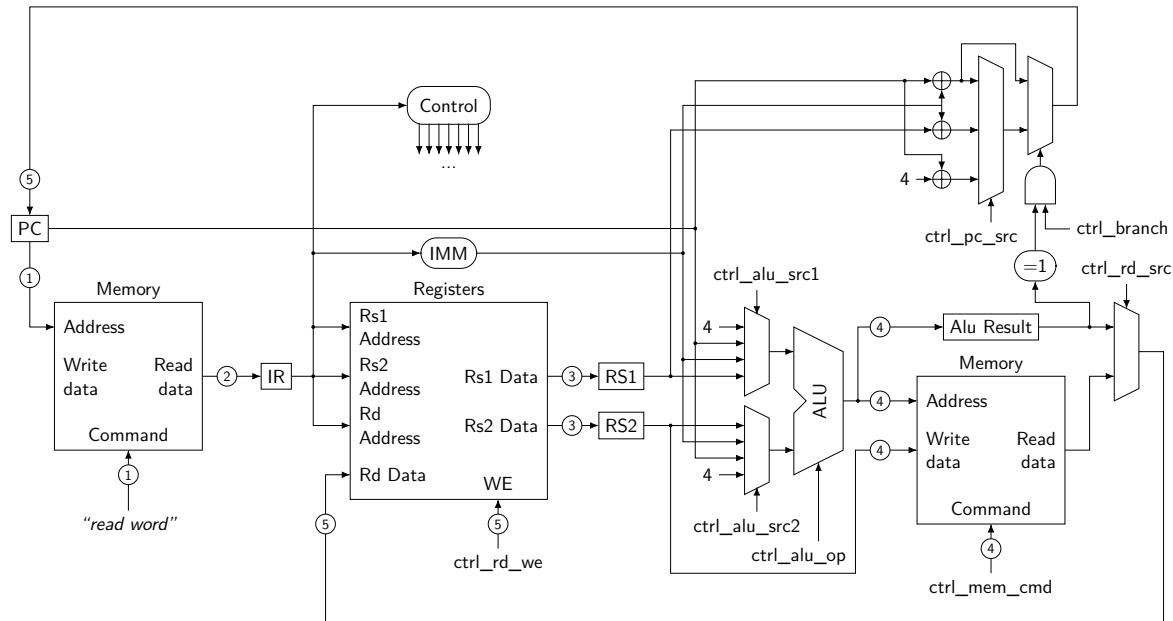


Abbildung 6: Schematische Darstellung des EduCore-V Tiny

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
func7							rs2				rs1				func3			rd				opcode									
0	0	0	1	0	0	0											1	1	0						0	1	1	0	0	1	1

Abbildung 7: Befehlsformat des dot Befehls

ctrl_alu_src1	REG, IMM, PC, 4
ctrl_alu_src2	REG, IMM, PC, 4
ctrl_mem_cmd	WRITE_B, WRITE_H, WRITE_W, READ_B, READ_H, READ_W, READ_BU, READ_HU, NOP
ctrl_alu_op	NOP, OP1, OP2, ADD, SUB, OR, AND, XOR, SLL, SRL, SRA, EQ, NEQ, LT, GE, LTU, GEU, DOT
ctrl_rd_src	ALU_RESULT, MEM_DATA
ctrl_rd_we	false, true
ctrl_pc_src	PC_NEXT, PC_IMM, RS1_IMM
ctrl_branch	false, true

Abbildung 8: Kontrollsignale ergänzt um dot

In dieser Aufgabe sollen Sie den RV32I Basisbefehlssatz des in Abbildung 6 schematisch dargestellten EduCore-V Tiny um das Skalarprodukt (dot product) zweier Vektoren erweitern. Der Befehl `dot` soll die Berechnung des Skalarproduktes zweier vierelementiger Vektoren $u \in \mathbb{N}^4$ und $v \in \mathbb{N}^4$ in der Art vornehmen, dass

$$s = u \cdot v = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{pmatrix} \cdot \begin{pmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{pmatrix} = \sum_{i=0}^3 u_i v_i$$

wobei für die einzelnen Elemente der Vektoren $0 \leq u_j, v_i \leq 255 : 0 \leq j, i \leq 3$ gilt.

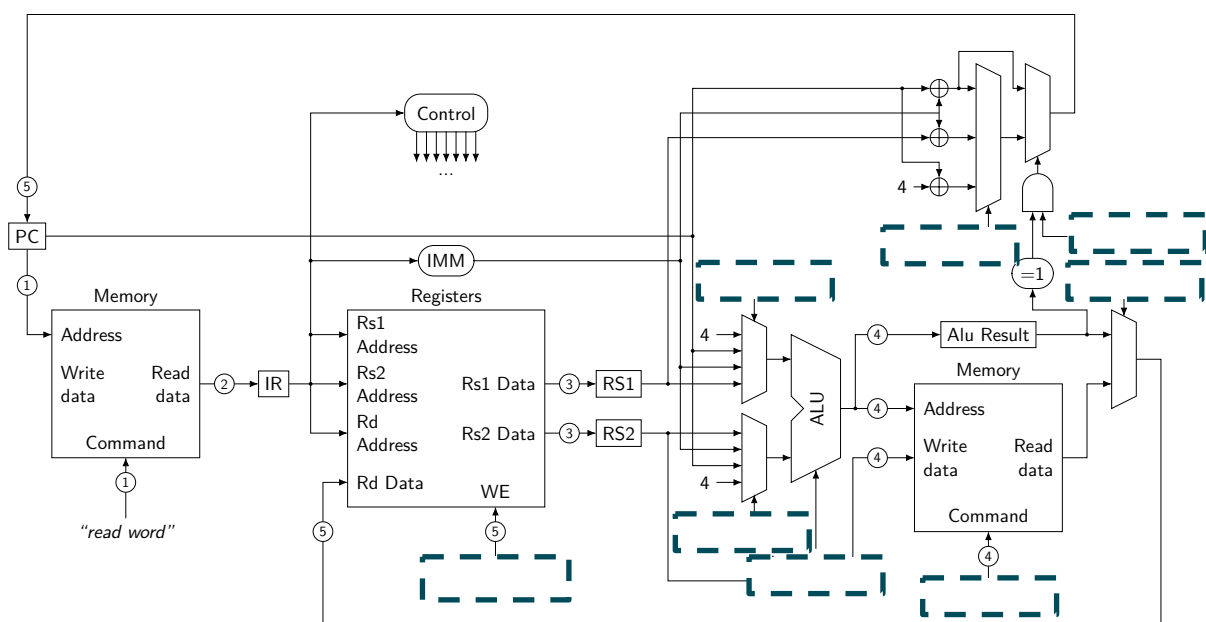
Ein einzelner Vektor wird als 32 Bit Wert dargestellt. Gemäß Abbildung 7 soll der Befehl `dot` zu den Register-Register Befehlen gehören und den Opcode 0110011 erhalten. Ferner wird der func3 Teil auf 6 und der func7 teil auf 8 gesetzt.

- a) Könnten der func7 und func3 auch anders gewählt werden? Begründen Sie Ihre Antwort kurz.

- b) Realisieren Sie die Berechnung des Skalarproduktes unter der ausschließlichen Verwendung des RV32I Basisbefehlssatzes sowie der Pseudoinstruktionen als RISC-V Assembler Programm. **Zudem darf der in der praktischen Übung realisierte `mul` Befehl verwendet werden.** Laden Sie in die beiden Register `x11` und `x12` die beiden Vektoren $(11,22,33,44)^T$ sowie $(55,66,77,88)^T$ und führen Sie die Berechnung des Skalarproduktes durch, wobei das Ergebnis nach der Berechnung in Register `x10` abgelegt werden soll.



- c) Geben Sie die vom Kontrollsignalgenerator für die Realisierung des `dot` Befehls zu erzeugenden Kontrollsignale an, indem Sie **sämtliche** Signalwerte in der folgenden schematischen Darstellung des EduCore-V Tiny ergänzen (gestrichelte Kästchen ausfüllen). Sie finden die vorhandenen Signale in Abbildung 8.



- d) Ergänzen Sie nun den nachfolgenden Codeausschnitt der ALU um den `dot` Befehl. Nutzen Sie für die case-Abfrage die Konstante `CTRL_ALU_OP_DOT`.

```
architecture rtl of m_alu is
    constant CONST_ZERO : std_logic_vector(31 downto 0) := (others => '0');
    constant CONST_ONE  : std_logic_vector(31 downto 0) := (0 => '1', others => '0');

begin

    process(reg_op1, reg_op2, operation)
    begin
        case (operation) is
            -- ...
            when CTRL_ALU_OP_ADD
                => result <= std_logic_vector(unsigned(reg_op1) + unsigned(reg_op2));

            when CTRL_ALU_OP_SUB
                => result <= std_logic_vector(unsigned(reg_op1) - unsigned(reg_op2));

            when CTRL_ALU_OP_MERGE
                => result <= reg_op1(31 downto 16) & reg_op2(15 downto 0) ;

            when CTRL_ALU_OP_OR    => result <= reg_op1 or reg_op2;
            when CTRL_ALU_OP_AND  => result <= reg_op1 and reg_op2;
            when CTRL_ALU_OP_XOR  => result <= reg_op1 xor reg_op2;
            -- LOESUNG --

            -- ...
            when others            => result <= CONST_ZERO;
        end case;
    end process;

end architecture rtl;
```

- e) Realisieren Sie die Berechnung des Skalarproduktes unter Verwendung des von Ihnen erweiterten RV32I Basisbefehlssatzes sowie Pseudoinstruktionen als RISC-V Assembler Programm. Laden Sie in die beiden Register `x11` und `x12` die beiden Vektoren $(11,22,33,44)^T$ sowie $(55,66,77,88)^T$ und führen Sie die Berechnung des Skalarproduktes durch, wobei das Ergebnis nach der Berechnung in Register `x10` abgelegt werden soll.

Nutzen Sie explizit den Aufruf Ihres eben realisierten Befehls `dot x10, x11, x12`. Bedenken Sie, dass der Compiler Ihren Befehl nicht kennt.

Assemblercode

- f) Geben Sie den resultierenden Speicherinhalt (nachfolgende Tabelle) des EduCore-V Tiny direkt nach dem Uploaden Ihres Programms aus d) an. Obwohl das gesamte Programm im Speicher liegt, müssen Sie nur die Zeilen von `dot` angeben. Bedenken Sie beim Angeben des Speichers, dass jede Speicherstelle ein Byte aufnehmen kann und als Byte-Reihenfolge im Speicher Little Endian genutzt wird. Sie können die Inhalte in der Binär- oder Hexadezimaldarstellung angeben.

Adr.	Speicherinhalt
00	
01	
02	
03	
04	
05	
06	
07	

Adr.	Speicherinhalt
08	
09	
0A	
0B	
0C	
0D	
0E	
0F	

Adr.	Speicherinhalt
10	
11	
12	
13	
14	
15	
16	
17	

$\Sigma_{A3} = \underline{\hspace{2cm}} / 24 \text{ Punkte}$

Aufgabe 4: Allgemeine Fragen

(6 Punkte)

- a) Handelt es sich bei einem JK-Flipflop stets auch um ein MS-Flipflop? Begründen Sie Ihre Antwort kurz.

- b) Welchen Vorteil bietet ein JK-Flipflop gegenüber einem RS-Flipflop?

- c) Kann auf einem FPGA jede Schaltung realisiert werden? Begründen Sie Ihre Antwort kurz.

- d) Nennen Sie 2 Vorteile einer Minimierten Schaltung bezüglich einer nicht minimierten Schaltung. Und ist es sinnvoll, stets die Minimalform einer Schaltfunktion zu verwenden? Begründen Sie Ihre Antwort kurz.

- e) Kann die Geschwindigkeit des an einem Schaltwerk anliegenden Takts beliebig erhöht werden? Begründen Sie Ihre Antwort kurz.

- f) Ist ein 8 Bit Addierer stets langsamer als ein 4 Bit Addierer? Begründen Sie Ihre Antwort kurz.

$\Sigma_{A4} = \underline{\hspace{2cm}} / 6 \text{ Punkte}$