DHBW
Duale Hochschule
Baden-Württemberg

Nachprüfung Programmentwurf
für
Entwurf digitaler Systeme

TEL22
27.11.2024
Torben Mehner

Matrikelnummer: _____

Erlaubte Hilfsmittel:

- Ausgeteilte Formelsammlung
- Nicht-programmierbarer Taschenrechner

Wichtige Hinweise zur Durchführung der Klausur:

- Trage die Matrikelnummer in das Deckblatt ein.
- Wenn die Heftung gelöst wird, muss jedes Blatt mit der Matrikelnummer gekennzeichnet werden.
- Sollte der Platz für eine Antwort nicht reichen, ist die Rückseite zu verwenden.
- Verwende keinen Rotstift.
- Ergebnisse werden nur gewertet, wenn der Lösungsweg ersichtlich ist!
- Bei Täuschungsversuch wird die gesamte Klausur mit der Note 5,0 bewertet.

Aufgabe Nr.:	1	2	3	4	5	**Summe**
Punktzahl:	11	10	13	6	5	45
Davon erreicht:						

Note: _____

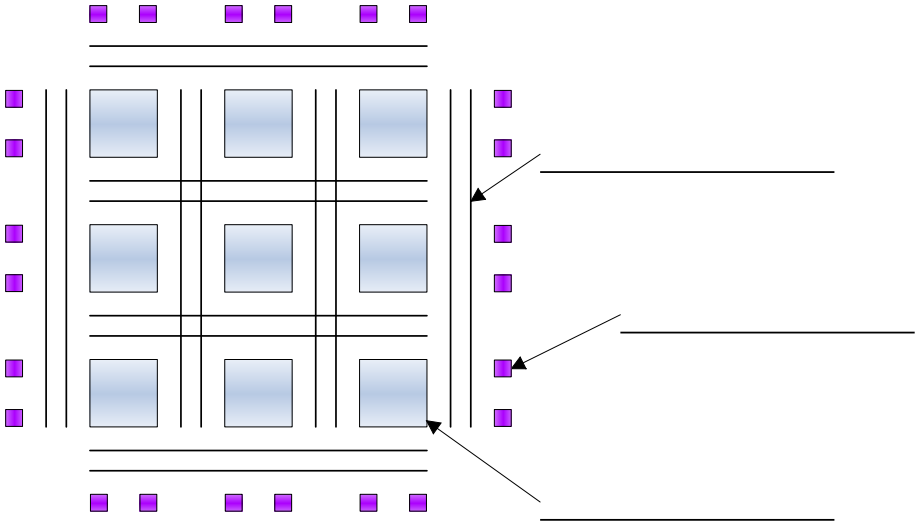
Unterschrift
Korrektor: _____

Viel Erfolg!

Verständnis und Wissen

1. Die folgenden Fragen befassen sich mit dem in der Vorlesung übermittelten Wissen und dessen Verständnis. Eine Antwort in Stichpunkten genügt.

   (a) Benennen Sie die gekennzeichneten Stellen im Aufbau eines FPGAs und erklären Sie knapp deren Aufgaben. (3)



Solution: Interconnections ($\frac{1}{2}$): verbindet CLBs ($\frac{1}{2}$)
Input-/Output-Blocks ($\frac{1}{2}$): Schaltet Ein- und liest Ausgänge ($\frac{1}{2}$)
Configurable Logic Blocks ($\frac{1}{2}$): Bildet Logikfunktionen ab ($\frac{1}{2}$)

   (b) Führen Sie eine schriftliche, binäre Multiplikation der Binärzahlen **0b111001** und **0b101010** durch. (2)

Solution:

```
                    111001 * 101010
                    ---------------
                            000000
                           1110010
                          00000000
                         111001000
                        0000000000
                       11101000000
                    ---------------
                      100101011010
```

   (c) Erklären Sie kurz den Vorteil von einem Multiplizierer mit Pipelining im Vergleich zu einem Multiplizierer ohne Pipelining. Beziehen Sie sich dabei auf den Taktschlupf. (1)

Solution: Der Takt kann nur so schnell wie die langsamste Einheit sein. Eine Multiplikation dauert lange. Pipelining verteilt eine Multiplikation über viele Takte, so kann der Takt beschleunigt werden. Dadurch wird der Taktschlupf verkleinert.

(d) Nennen Sie für 5 Zustände der STD_LOGIC das in VHDL verwendete Kurzzeichen und den Namen. (Beispiel: '-': Don't care, die Nennung des Beispiels gibt keinen Punkt) (5)

Solution: 0: Low, 1: High, L: Weak Low, H: Weak High, Z: High Impedance, X: Strong Conflict, W: Weak Conflict, U: Undefined (each $\frac{1}{2}$)

VHDL Programmieren

2. Der folgende Code zeigt den Frequenzteiler aus der Vorlesung.

```
entity counter is
    Generic (
        FREQ_IN : INTEGER := 50000000;
        FREQ_OUT : INTEGER := 5 );
    Port (
        CLK_IN : in  STD_LOGIC;
        RST : in  STD_LOGIC;
        CLK_OUT : out  STD_LOGIC);
end counter;

architecture Behavioral of counter is
    constant MAX : UNSIGNED(31 downto 0) :=
        to_unsigned((FREQ_IN/FREQ_OUT/2)-1, 32);
    signal sig_CNT : UNSIGNED(31 downto 0) := (others=>'0');
    signal sig_CLK : STD_LOGIC := '0';
begin

    cnt_proc: process( CLK_IN, RST )
    begin
        if( RST = '1' ) then
            sig_CNT <= (others => '0');
            sig_CLK <= '0';
        elsif( rising_edge( CLK_IN ) ) then
            if( sig_CNT = MAX ) then
                sig_CNT <= (others => '0');
                sig_CLK <= not sig_CLK;
            else
                sig_CNT <= sig_CNT + 1;
            end if;
        end if;
    end process cnt_proc;

    CLK_OUT <= sig_CLK;
end Behavioral;
```

(a) Der Frequenzteiler soll nun mit einer variablen Ausgangsfrequenz versehen werden. Die eingestellte Frequenz soll immer nur zum Ende einer Ausgangsperiode gesetzt werden. Welche Änderungen müssen am Code vorgenommen werden? Worin liegt hierbei ein Problem bei der Synthese? (4)

Solution: FREQ_OUT von Generic zu Port (1), Einführung von MAX_NEXT als Signal statt Konstante, Speichern von MAX_NEXT in MAX_REG und Verwendung von MAX_REG als Höchstwert (2), das Problem liegt in der Berechnung des Höchstwerts, was

eine Division erfordert, die nicht synthetisierbar ist (1).

(b) Schreibe die Architektur des Frequenzteilers mit variabler Ausgangsfrequenz. Ignoriere dabei das Problem bei der Synthese aus Aufgabenstellung a) und nimm an, es wäre synthetisierbar. (6)

```
entity counter is
    Generic (
        FREQ_IN : INTEGER := 50000000;
    Port (
        CLK_IN : in  STD_LOGIC;
        FREQ_OUT : in UNSIGNED(31 downto 0);
        RST : in  STD_LOGIC;
        CLK_OUT : out  STD_LOGIC);
end counter;

architecture Behavioral of counter is
```

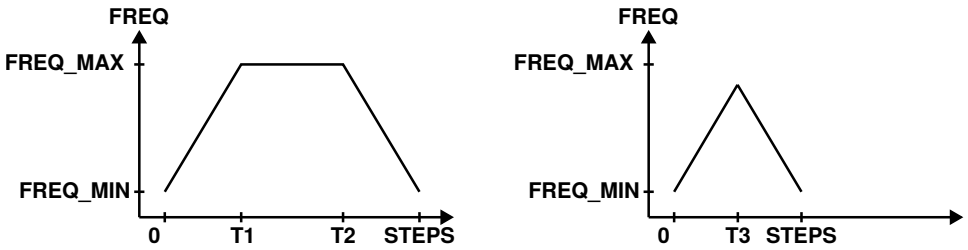
Solution: Signale: constant zu signal (1), andere Signale abschreiben ($\frac{1}{2}$), 0Hz verhindern (1), MAX setzen (1) - am Ende einer Periode (1), process Syntax abschreiben ($\frac{1}{2}$), zählen richtig abschreiben (1)

```
    signal MAX_NEXT, MAX_REG : UNSIGNED(31 downto 0) :=
        to_unsigned(1, 32);
    signal sig_CNT : UNSIGNED(31 downto 0) := (others=>'0');
    signal sig_CLK : STD_LOGIC := '0';
begin

    max_proc: process( FREQ_OUT )
    begin
        if( FREQ_OUT = 0 ) then
            MAX_NEXT <= to_unsigned((FREQ_IN/2)-1, 32);
        else
            MAX_NEXT <= to_unsigned((FREQ_IN/FREQ_OUT/2)-1, 32);
        end if;
    end process max_proc;

    cnt_proc: process( CLK_IN, RST )
    begin
        if( RST = '1' ) then
            sig_CNT <= (others => '0');
            sig_CLK <= '0';
            MAX_REG <= MAX_NEXT;
        elsif( rising_edge( CLK_IN ) ) then
            if( sig_CNT = MAX_REG ) then
                sig_CNT <= (others => '0');
```

```
                sig_CLK <= not sig_CLK;
                MAX_REG <= MAX_NEXT;
            else
                sig_CNT <= sig_CNT + 1;
            end if;
        end if;
    end process cnt_proc;

    out_proc: process( sig_CLK )
    begin
        CLK_OUT <= sig_CLK;
    end process out_proc;
```

```
end Behavioral;
```

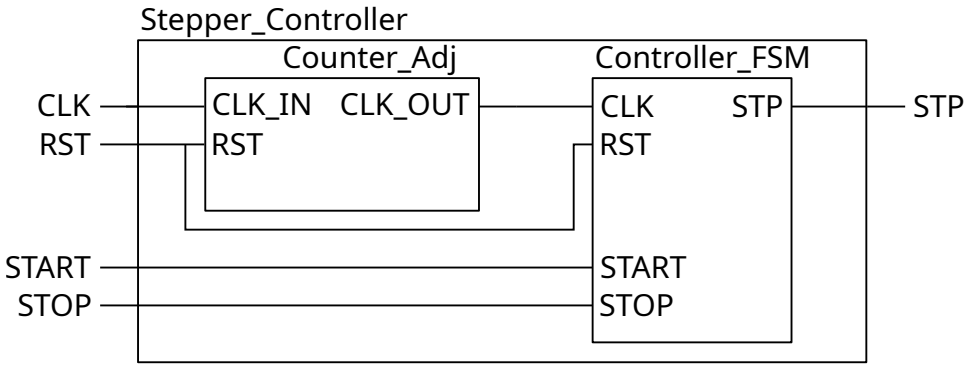
3. Es soll ein Controller für Schrittmotoren erstellt werden. Dieser erhält eine Anzahl an Schritten, die der Motor fahren soll durch den Eingang STEPS, sowie ein Start-Signal. Geht das Start-Signal auf '1', gibt der Controller sich beschleunigende Taktpulse aus, beginnend mit der Geschwindigkeit FREQ_MIN, bis eine Geschwindigkeit FREQ_MAX erreicht ist. Dann hält er diese Geschwindigkeit bis zur Entschleunigung, bei der die Geschwindigkeit von FREQ_MAX bis auf FREQ_MIN reduziert wird.



Dabei gilt es, zwei Fälle zu unterscheiden:

- Die maximale Geschwindigkeit wird erreicht $STEPS \geq 2*(FREQ_{MAX} - FREQ_{MIN})$ (linke Abbildung)
  - $T_1 = FREQ_{MAX} - FREQ_{MIN}$
  - $T_2 = STEPS - (FREQ_{MAX} - FREQ_{MIN})$
- Die maximale Geschwindigkeit wird nicht erreicht $STEPS < 2*(FREQ_{MAX} - FREQ_{MIN})$ (rechte Abbildung)
  - $T_3 = \frac{STEPS}{2}$

Der Controller soll mit einer State-Machine umgesetzt werden. Diese ist in der folgenden Abbildung gezeigt. Der Zustand soll synchron mit der Ausgangsfrequenz des Flexiblen Frequenzteilers weitergeschaltet werden. Die State-Machine soll die Frequenz des Teilers steuern. Abhängig vom Zustand soll diese Clock dann an den Schrittmotor-Treiber (STP) weitergeleitet werden.



(a) Um welche Art von State-Machine handelt es sich dabei? Begründe deine Antwort. (2)

Solution: Mealy-Automat, da der STP-Ausgang abhängig vom Zustand und dem CLK-Eingang ist.

(b) Zeichne ein Zustandsdiagramm, das die Zustände zeigt, sowie die Bedingungen der Übergänge der verschiedenen Zustände ineinander. Vermerke bitte auch den Reset-Zustand. (7)

Solution: Zustände (STOP, ACCELERATE, RUN, DECELLERATE) (2) Reset Zustand (1) Bedingungen (1,5) Übergänge (2,5)

(c) Programmiere den Output-Prozess des Zustandsautomats. Beginne mit: (4)

```
out_fn: process (state)
begin
```

Solution: Case (1) und alle Zustände aus d) abgedeckt (1) STOP richtig (1) Andere Zustände (1)

```
case state is
    when STOP =>
        STP <= '0';
    when ACCELERATE =>
        STP <= CLK;
    when RUN =>
        STP <= CLK;
    when DECELLERATE =>
        STP <= CLK;
end case;
end process out_fn;
```

```
end process out_fn;
```

Simulation

4. Die Komponente `SOME_ENTITY` soll mittels der Testbench `testbench` getestet werden.

Listing 1: Die Testbench

```
entity testbench is
end testbench;

architecture arch_0 of testbench is

component SOME_ENTITY is
port(   A, B, CLK: in STD_LOGIC;
P, Q: out STD_LOGIC );
end component;

signal CLK, A, B, P, Q : STD_LOGIC := '0';
```

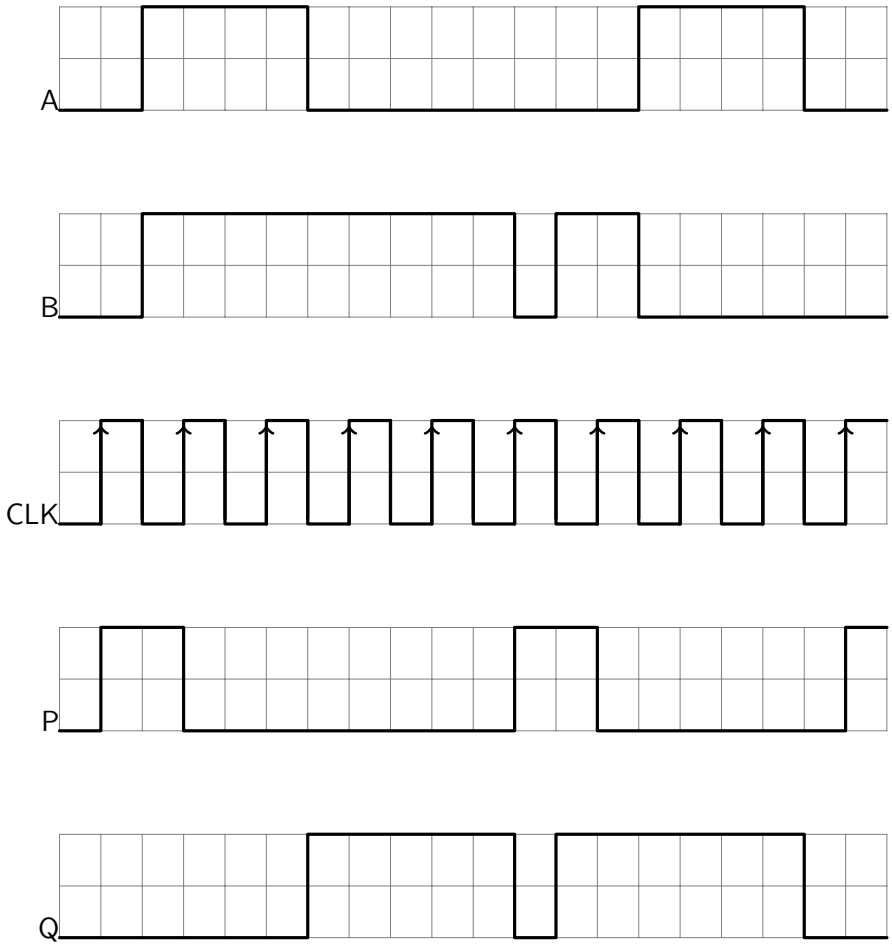
```
begin
uut : SOME_ENTITY
port map( A=>A, B=>B, CLK=>CLK, P=>P, Q=>Q );

clocking : process
begin
wait for 5ns;
CLK <= not(CLK);
end process clocking;

stimulus : process
begin
wait for 10 ns;
A <= not(A);
B <= not(B);
wait for 20 ns;
A <= '0';
wait for 20 ns;
B <= '1';
wait for 5 ns;
B <= '0';
wait for 5 ns;
B <= '1';
end process stimulus;
end arch_0;
```

Listing 2: Architecture der Unit-under-test

```
architecture arch_uut of SOME_ENTITY is
begin
process( clk )
begin
if rising_edge(clk) then
P <= A XOR B;
end if;
end process;

Q <= A AND B;
end arch_uut;
```

(a) Zeichnen Sie den Verlauf von `A` und `B` in das unten stehende Diagramm ein. (2)

(b) Zeichnen Sie nun den Verlauf der Ausgänge `P` und `Q` in das unten stehende Diagramm ein. (4)



Fehlersuche

5. Beim Programmieren des Hardware-Multiplizierers meldet die Entwicklungsumgebung Syntax-Fehler in Zeile 5, 8, 11, 24 und 32.

   (a) Korrigiere die Fehler. Trage dazu die Zeile, in welcher der Fehler ist und die gesamte, korrigierte Zeile in die Tabelle ein. (Libraries wurden eingebunden und verursachen keine Fehler) (5)

   Besteht der Fehler darin, dass eine gesamte Zeile fehlt, kann diese durch das Nennen der vorherigen Zeile mit einem "+"˙ eingefügt werden (Beispiel: 1+: das hier fehlt zwischen Zeile 1 und 2).

```
 1 entity hw_mul is
 2 Generic(
 3         N : integer := 4;
 4         M : integer := 4 )
 5 Port (  f1 : in  STD_LOGIC_VECTOR (N-1 downto 0);
 6         f2 : in  STD_LOGIC_VECTOR (M-1 downto 0);
 7         clk : in STD_LOGIC;
 8         q : out  STD_LOGIC (M+N-1 downto 0));
 9 end hw_mul;
10
11 architecture Behavioral is
12     signal sum : unsigned( M+N-1 downto 0);
13 begin
14
15     process( clk )
16         variable temp : unsigned( M+N-1 downto 0);
17     begin
18         if( risind_edge(clk) then
19             sum <= to_unsigned(0, M+N);
20
21             for I in 0 to N-1 loop
22                 temp := (others => '0');
23                 if( f1(I) = 1) then
24                     temp(I+M-1 downto I) := unsigned(f2);
25                 end if;
26
27                 sum <= sum + temp;
28
29             end loop;
30
31             q <= sum;
32         end if;
33
34     end process;
35
36 end Behavioral;
```

Tabelle 1: Verbesserungen eintragen

Fehler	Zeile	Korrektur
1	4	M : integer := 4);
2	8	q : out STD_LOGIC_VECTOR(N+M-1 downto 0));
3	11	architecture Behavioral of hw_mul is
4	23	if(f1(I) = '1') then
5	31	q <= std_logic_vector(sum);