DHBW
Duale Hochschule
Baden-Württemberg

Programmentwurf

für

Entwurf digitaler Systeme

TEL21
05.12.2023
Torben Mehner

Matrikelnummer: _____

Erlaubte Hilfsmittel:

- Ausgeteilte Formelsammlung
- Nicht-programmierbarer Taschenrechner

Wichtige Hinweise zur Durchführung der Klausur:

- Tragen Sie Ihre Matrikelnummer in das Deckblatt ein.
- Wenn Sie die Heftung lösen, müssen Sie jedes Blatt mit Ihrer Matrikelnummer kennzeichnen.
- Verwenden Sie 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	6	**Summe**
Punktzahl:	8	5	6	12	7	6	44
Davon erreicht:							

Note: _____

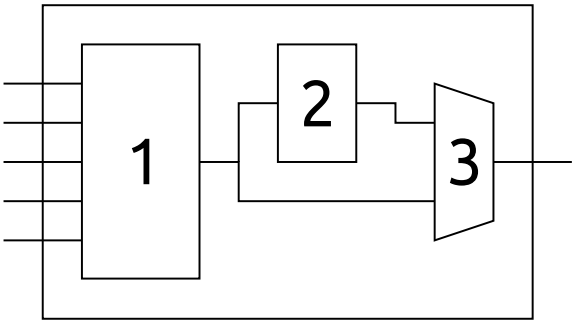
Unterschrift
Korrektor: _____

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) Beschrifte die mit Zahlen gekennzeichneten Bestandteile eines Configurable Logic Blocks (CLB). Erkläre kurz die Funktion jedes Bestandteils. (3)



Solution: 1) Look Up Table ($\frac{1}{2}$): bildet Logikfunktion ab ($\frac{1}{2}$)
2) Flip-Flop ($\frac{1}{2}$) für synchrone Vorgänge ($\frac{1}{2}$)
3) Multiplexer ($\frac{1}{2}$) wählt synchrones/asynchrones Signal ($\frac{1}{2}$)

   (b) Innerhalb eines FPGAs befinden sich neben den CLBs, welche die Logik abbilden auch noch Input-Output-Blocks (IOB). Beschreibe den Aufbau und die Funktionsweise dieser Verbindungen. (2)

Solution: MOSFET und SRAM-Zelle ergibt programmierbare Schalter (1), die beliebige Verbindungen herstellen können (1).
Aufgrund schwierig gestellter Frage auch Erklärung von IO-Blocks möglich...

   (c) Nenne einen Vorteil von FPGAs gegenüber dem Standardzellen Design. (1)

Solution: FPGAs sind von der Stange erhältlich und sind rekonfigurierbar und eignen sich daher fürs Prototyping (je 1)

   (d) Beim Standardzellendesign gibt es verschiedene Optimierungsvarianten jeder Zelle. Nenne zwei. (2)

Solution: Low-Power, High Speed, Low Footprint (je 1)

2. Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. Für ein richtiges Kreuz gibt es +1 Punkte. Ein falsches Kreuz bringt -0.5 Punkte. Wird bei einer Aussage nichts angekreuzt, gibt es keinen Abzug. (5)

   (a) Der Ergebnisvektor einer Addition in VHDL hat die Länge 15, wenn ein Summand eine Länge von 7 und der Andere eine Länge von 8 hat.
   ◯ Wahr √ **Falsch**

   (b) Bei einem Moore-Automat ist der Ausgang nur vom aktuellen Zustand abhängig.
   √ **Wahr** ◯ Falsch

   (c) Auf dem Y-Diagramm gibt es die Verhaltens, Struktur und Geometrieebenen.
   √ **Wahr** ◯ Falsch

   (d) Bei einem Digitalen Signalprozessor handelt es sich um einen SIMD-Prozessor.
   √ **Wahr** ◯ Falsch

   (e) Mit `TYPE state_type IS (one, two, three);` erstellt man einen Dateityp in VHDL.
   √ **Wahr** ◯ Falsch

3. Mithilfe einer Ampelanlage wird ein Fußgängerüberweg zwischen einem Bahnhof und der Fußgängerzone geregelt. Diese Ampelanlage soll mithilfe einer State-Machine umgesetzt werden.

   (a) Welcher Zustandsautomat passt am Besten zu einer Ampelschaltung? Begründe deine Antwort kurz. (2)

   Solution: Moore Automat (1)

   (b) Nachfolgend befindet sich der State-Conversion-Prozess eines Zustandsautomats für die Ampelschaltung. Bei `s_rot` stehen die Autos und die Fußgänger können laufen. Die Ampel soll nun so umgebaut werden, dass im Normalfall die Fußgänger laufen können (`s_rot`). Wenn ein Auto eine Induktionsschleife auslöst, was durch eine logische '1' auf dem Signal `IND_IN` dargestellt wird, soll die Ampel die Phasen durchlaufen und dann wieder im Zustand `s_rot` auf ein Fahrzeug warten. (2)

   Schreibe den State-Conversion-Prozess neu, sodass diese Funktion umgesetzt wird.

```
state_conv: process( slow_clk, RST )
begin
    if( RST = '1' ) then
        state <= s_rot;
    elsif( rising_edge( slow_clk ) ) then
        case state is
            when s_rot => state <= s_rotgelb;
            when s_rotgelb => state <= s_gruen;
            when s_gruen => state <= s_gelb;
            when s_gelb => state <= s_rot;
            when others => state <= s_rot;
        end case;
    end if;
end process state_conv;
```

Solution: Abschreiben (½), If-Syntax (½), Logik richtig (1)

```
state_conv: process( slow_clk, RST )
begin
    if( RST = '1' ) then
        state <= s_rot;
    elsif( rising_edge( slow_clk ) ) then
        case state is
            when s_rot =>
                if IND_IN = '1' then
                    state <= s_rotgelb;
                else
                    state <= s_rot;
                end if;
            when s_rotgelb => state <= s_gruen;
            when s_gruen => state <= s_gelb;
            when s_gelb => state <= s_rot;
            when others => state <= s_rot;
        end case;
    end if;
end process state_conv;
```

(c) Ändert sich durch die Änderung die Art des Zustandsautomats? Begründe deine Entscheidung. (2)

Solution: Nein (1). Die Ausgabe is nach wie vor nur vom Zustand abhängig (1).

VHDL Programmieren

4. Ein Hardware-Block soll ein PWM-Signal erzeugen. Das Ausgangssignal ist zu Beginn eines Zyklus high, bis ein Zähler den Eingangswert `t_1` erreicht. Dann schaltet das Ausgangssignal auf low. Ein Zyklus soll genau so lang sein, dass ein der Eingangswert `t_1` die Ein-Zeit in 1-Prozent-Schritten angibt (0-100%).

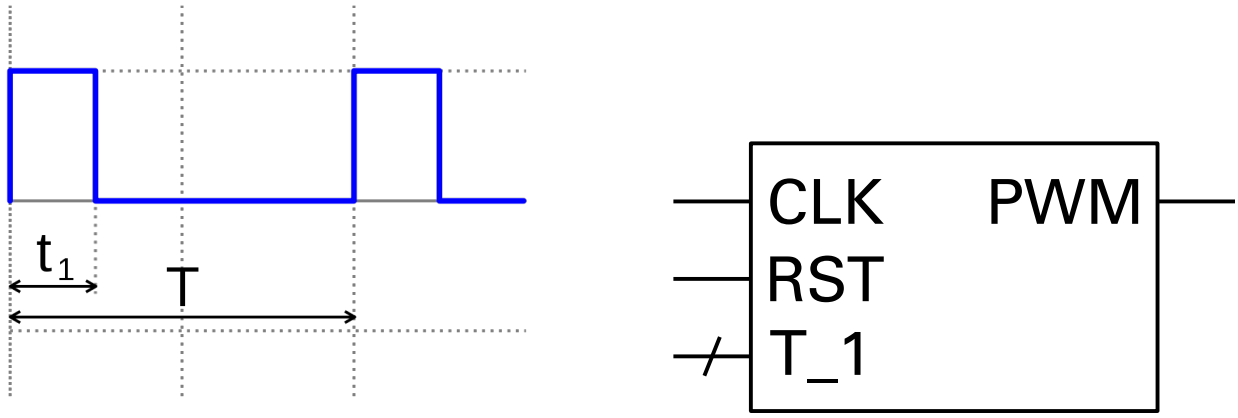


Abbildung 1: PWM-Signal(links) PWM Entity (rechts)

(a) Schreibe die Entity des PWM-Generators, wie in der Abbildung rechts gezeigt. Beachte, dass `T_1` ein Vektor ist. Lege die Länge des Vektors kleinstmöglich für die Aufgabe fest. (4)

Solution: Entity (1), Ports (je ½), Länge (1)

```
entity pwm_generator is
port(
    clk : IN STD_LOGIC;
    rst : IN STD_LOGIC;
    t_1 : IN UNSIGNED(6 downto 0);
    pwm : OUT STD_LOGIC );
end pwm_generator;
```

(b) Was müsste geändert werden, um die PWM anstatt in 1-Prozent-Schritten in Tausendstel-Schritten einstellen zu können? (1)

Solution: Der Eingangsvektor müsste 10 bit lang sein (½) und die Architektur bis 1000 Zählen (½).

(c) Schreiben Sie die Architektur des PWM-Generators (7)

Solution: Architektur(1), Signal(1), Prozess(1), Reset(1), Rising Edge(1), Zählen(1), Ausgabe(1)

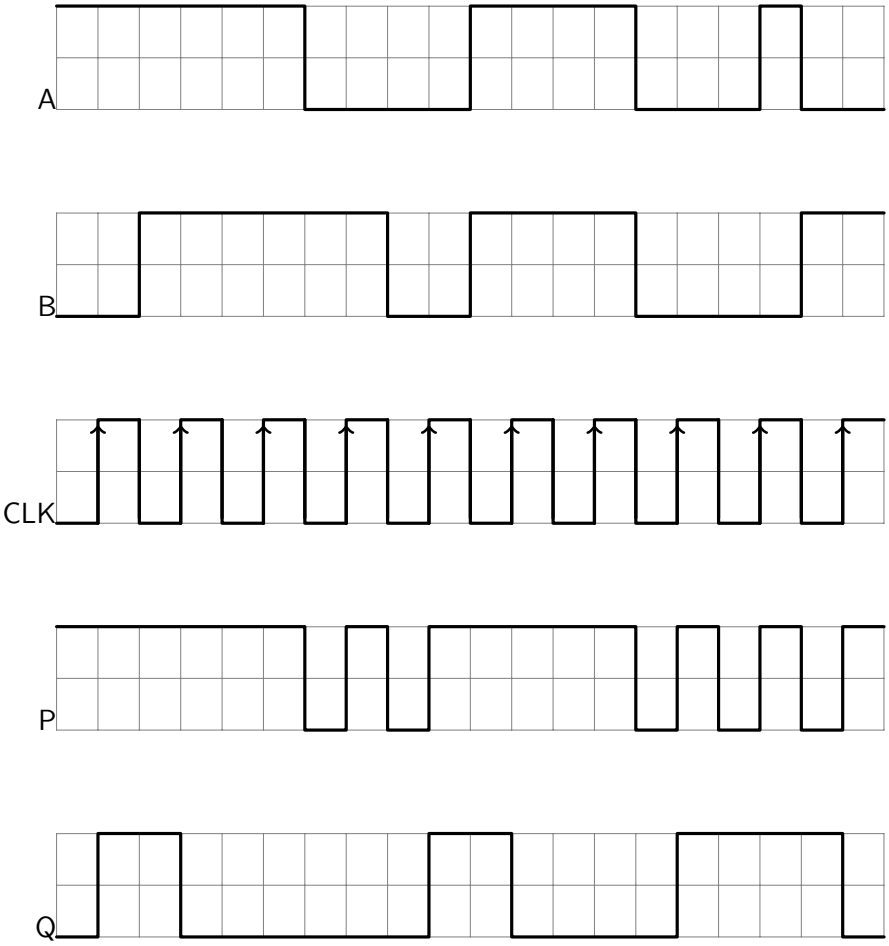
```
architecture Behavioral of pwm_generator is
    signal t : unsigned (7 downto 0);
begin

    count : process (clk)
    begin
        if rising_edge(clk) then
            if( t < 100 ) then
                t <= t+1;
            else
                t <= to_unsigned(0, 8);
            end if;
        end if;
    end process;

    pwm <=  '1' when t < t_1 else
            '0';
end Behavioral;
```

Simulation

5. Die folgenden Schaubilder zeigen die Stimuli einer Testbench für die UUT. Ein Clock-Zyklus (High und Low) dauert 100ns. Listing 1 zeigt die Testbench, Listing 2 die Archtiektur der Unit-Under-Test. Die Signallaufzeiten sind idealisiert nicht vorhanden.



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 50ns;
        CLK <= not(CLK);
    end process clocking;

    stimulus_a : process
    begin
        -- hier Aufgabenteil a)
    end process stimulus_a;

    stimulus_b : process
    begin
        -- hier steht Stimulus B
    end process stimulus_b;
end arch_0;
```

Listing 2: Architecture der Unit-under-test

```
architecture arch_uut of SOME_ENTITY is
    P <= A OR CLK;

    process( CLK )
    begin
        if rising_edge(CLK) then
            Q <= P XOR B;
        end if;
    end process;
end arch_uut;
```

(a) Programmieren Sie den Stimuli-Prozess für A. Sie müssen den umgebenden Prozess nicht nocheinmal abschreiben. Ersetzten Sie einfach das Kommentar "- - hier Aufgabenteil a)" (3)

Solution:

```
A <= '1';
wait for 300ns;
A <= '0';
wait for 200ns;
A <= '1';
wait for 200ns;
A <= '0';
wait for 150ns;
A <= '1';
wait for 50ns;
A <= '0';
wait;
```

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

Fehlersuche

6. Beim Programmieren des Hardware-Multiplizierers meldet die Entwicklungsumgebung folgende Syntax-Fehler:

   - Line 5: Syntax error near end.
   - Line 11: Syntax error near state_conv.
   - Line 27: Syntax error near '.
   - Line 28: Syntax error near '.
   - Line 29: Syntax error near '.
   - Line 30: Syntax error near '.
   - Line 32. End Identifier state_conv does not match declaration, out_fn.

   (a) Korrigiere die vier Fehler. Trage dazu die Zeile, in welcher der Fehler ist und die gesamte, korrigierte Zeile in die Tabelle ein. (Librarys verursachen keine Fehler) (4)

   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 ampel is
2     Port ( CLK : in  STD_LOGIC;
3            RST : in  STD_LOGIC;
4            LED : out  STD_LOGIC_VECTOR(2 downto 0);
5 end ampel;
6
7 architecture Behavioral of ampel is
```

```
 8      TYPE state_type IS ( s_rot, s_rotgelb, s_gruen, s_gelb );
 9      signal state : state_type := s_rot;
10
11      state_conv: process( slow_clk, RST )
12      begin
13          if( rising_edge( CLK ) ) then
14              case state is
15                  when s_rot => state <= s_rotgelb;
16                  when s_rotgelb => state <= s_gruen;
17                  when s_gruen => state <= s_gelb;
18                  when s_gelb => state <= s_rot;
19                  when others => state <= s_rot;
20              end case;
21          end if;
22      end process state_conv;
23
24      out_fn: process( state )
25      begin
26          case state is
27              when s_rot => LED <= '100';
28              when s_rotgelb => LED <= '110';
29              when s_gruen => LED <= '001';
30              when s_gelb => LED <= '010';
31          end case;
32      end process state_conv;
33
34  end Behavioral;
```

Tabelle 1: Verbesserungen eintragen

Fehler	Zeile	Korrektur
1	4	`LED : out STD_LOGIC_VECTOR(2 downto 0) );`
2	10	`begin`
3	27	`when s_rot => LED <= '100';`
4	32	`end process state_conv;`
Bonus	11	`state_conv: process( CLK, RST )`

(b) Die Umsetzung der oben gezeigte Ampelschaltung ist nicht vollständig und hat dadurch einen großen Nachteil. Was wurde hier vergessen? Welche Auswirkungen hat es? (2)

Solution: Der Reset wurde nicht implementiert (1). Dadurch kann die Schaltung nicht in einen definierten Ausgangszustand versetzt werden (1).