DHBW
Duale Hochschule
Baden-Württemberg

Programmentwurf

für

Entwurf digitaler Systeme

TEL20
13.12.2022
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:	6	5	8	15	9	7	50
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) Digitale Logik wird in General Purpose Processors (GPPs), anwenderkonfigurierbare ICs und Application Specific Integrated Circuits (ASICs) unterteilt. In welche Gruppe sind FPGAs einzuordnen? (1)

   Solution: Anwenderkonfigurierbare ICs

   (b) Erkläre, warum sich die Verwendung von ASICs erst bei mittleren bis hohen Stückzahlen lohnen und nenne einen Vorteil gegenüber den anderen Gruppen. (2)

   Solution: Der initiale Designaufwand für ASICs ist sehr hoch (1). Die daraus entstehenden ICs sind besser auf die Anwendung angepasst (1).

   (c) Erkläre, was bei der Synthese geschieht und wie dies mit dem Y-Diagramm zusammen hängt. (2)

   Solution: Bei der Synthese wird aus dem VHDL-Code eine Netzliste auf Register-Transfer-Level (1). Dies entspricht einer Konversion von der Verhaltens- in die Strukturebene (1).

   (d) Erkläre die Funktion der Constraint-Datei (ucf-Datei). (1)

   Solution: Die ucf-Datei verlinkt die Ein- und Ausgänge der Entity mit den physikalischen IOs des FPGA.

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

   (a) Kombinatorische Logik benötigt immer einen Reset-Eingang, um sie in einen definierten Zustand zu versetzen.
   ◯ Wahr √ **Falsch**

   (b) Ein Signal wird in VHDL am Ende eines Prozesses auf den ersten, innerhalb des Prozess zugewiesenen Wert gesetzt.
   ◯ Wahr √ **Falsch**

   (c) Die Entity eines VHDL-Moduls nennt die vorhandenen Ein- und Ausgänge.
   √ **Wahr** ◯ Falsch

   (d) Die Entity einer VHDL-Testbench hat die selben Ein- und Ausgänge wie das zu testende Modul
   ◯ Wahr √ **Falsch**

   (e) Wenn die Zustände eines Zustandsautomats abhängig von Eingängen in andere Zustände übergehen spricht man von einem Mealy-Automat.
   ◯ Wahr √ **Falsch**

3. Analog zur Multiplikation ist auch eine Division auf dem FPGA in Anlehnung an die schriftliche Division möglich.

5	1	3	:	3	=	1	7	1
3								
2	1							
2	1							
	0	3						
		3						
		0						

(a) Berechne nun im Binärsystem 101101 : 11 (3)

Solution:

1	0	1	1	0	1	:	1	1	=	0	1	1	1	1
	0													
1	0	1												
	1	1												
	1	0	1											
		1	1											
		1	0	0										
			1	1										
				1	1									
				1	1									
					0									

(0,5) pro Schritt, (0,5) fürs Ergebnis

(b) Erkläre, warum eine Division verglichen mit einer Subtraktion lange für die Durchführung benötigt. (2)

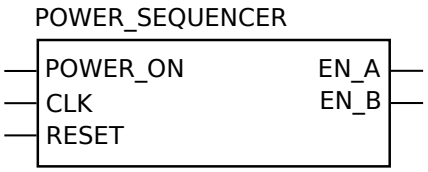
Solution: Verglichen mit einer Subtraktion werden bei einer Division deutlich mehr Gatter durchlaufen (1), da in jedem Schritt der Division eine Subtraktion vorhanden ist (1).

(c) Eine Division wie in Teilaufgabe (a) dauert 400 ns. Diese Division ist allerdings Teil einer Verarbeitungskette, bei der alle 250 ns Daten ankommen. Nenne eine Technik, die es erlaubt die Division mit dieser Datenrate durchzuführen und erkläre kurz, wie dies umgesetzt wird. (3)

Solution: Durch Pipelining kann die Division schneller Daten aufnehmen (1). Dabei wird die Division auf mehrere Takte aufgeteilt (1).

VHDL Programmieren

4. Für ein System mit verschiedenen Spannungsversorgungen soll ein FPGA als Power-Sequencer eingesetzt werden. Jede Spannungsversorgung hat einen Enable-Eingang (EN_A, EN_B), der bei einem '1'-Pegel die Spannungsversorgung anschaltet und bei einem '0'-Pegel abschaltet. Der FPGA kann die Enable-Signale ansteuern.



(a) Mit welcher Art von Zustandsautomat lässt sich der Sequencer realisieren? (1)

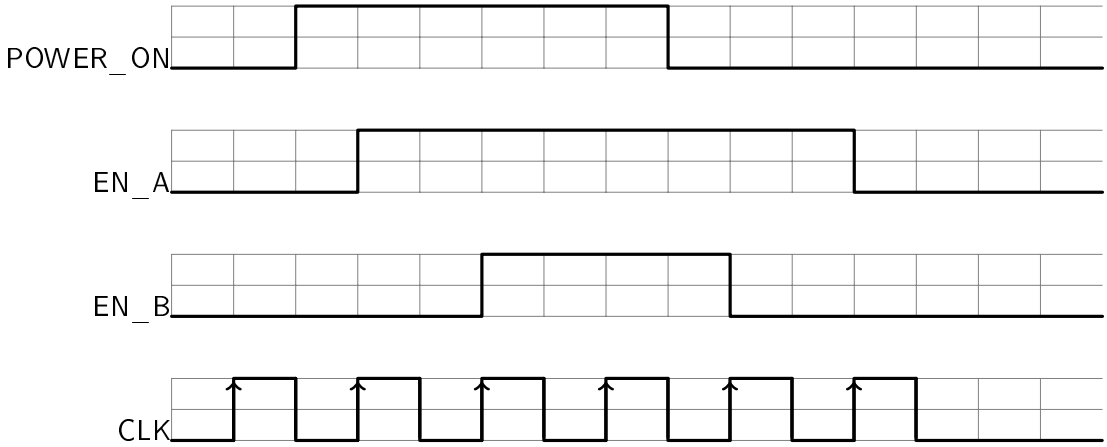
Solution: Full-Custom-Design aufgrund der hohen Stückzahlen

(b) Schreibe eine Entity für das VHDL-Modul `POWER_SEQUENCER`. (3)

Solution: Entity Syntax richtig (1) Eingänge richtig (1) Ausgang richtig (1)

```
entity POWER_SEQUENCER is
    Port (  POWER_ON : in  STD_LOGIC;
            CLK : in  STD_LOGIC;
            RESET : in  STD_LOGIC;
            EN_A : out  STD_LOGIC;
            EN_B : out  STD_LOGIC);
end POWER_SEQUENCER;
```

Geht der Eingang POWER_ON auf '1', wird beim nächsten Takt das Supply A angeschaltet und beim darauf folgenden das Supply B. Geht der Eingang POWER_ON auf '0', wird zuerst Supply B abgeschaltet, dann Supply A. Bei einem Reset (RESET='1') sollen beide Versorgungen unmittelbar und gleichzeitig abgeschaltet werden. Siehe dazu das Timing-Diagramm.



(c) Schreibe die Architecture des Zustandsautomats für POWER_SEQUENCER. Beachte dabei, dass dieser Zustandsautomat simuliert werden soll. (11)

Solution: Architecture richtig (1) State Type richtig (1) Signal richtig (1) Prozesse angelegt (2) Reset-Block richtig (1) CLK-Abfrage richtig (1) State Transition richtig (2) Output function richtig (2)

```
architecture Behavioral of sequencer is
    type state_type is (s_off, s_turn_on, s_on, s_turn_off);
    signal state : state_type := s_off;
begin

    state_trans: process(CLK, RESET)
    begin
        if( RESET = '1' ) then
            state <= s_off;
        elsif( rising_edge(CLK) ) then
            case state is
                when s_off =>
                    if( POWER_ON = '1' ) then
                        state <= s_turn_on;
                    end if;
                when s_turn_on =>
                    if( POWER_ON = '1' ) then
                        state <= s_turn_on;
                    else
                        state <= s_off;
                    end if;
                when s_on =>
                    if( POWER_ON = '0' ) then
                        state <= s_turn_off;
                    end if;
                when s_turn_off =>
                    state <= s_off;
                when others =>
                    state <= s_off;
            end case;
        end if;
    end process state_trans;

    out_proc : process( state )
    begin
        case state is
            when s_off =>
                EN_A <= '0';
                EN_B <= '0';
            when s_turn_on =>
                EN_A <= '1';
                EN_B <= '0';
```

```
            when s_on =>
                EN_A <= '1';
                EN_B <= '1';
            when s_turn_off =>
                EN_A <= '1';
                EN_B <= '0';
            when others =>
                EN_A <= '0';
                EN_B <= '0';
        end case;
    end process out_proc;

end Behavioral;
```

Simulation

5. Die Komponente POWER_SEQUENCER soll mittels der Testbench POWER_SEQUENCER_TB getestet werden.

Listing 1: Architecture der Testbench

```
architecture arch_0 of POWER_SEQUENCER_TB is

    component POWER_SEQUENCER is
    port(   POWER_ON, CLK, RESET: in STD_LOGIC;
            EN_A, EN_B: out STD_LOGIC );
    end component;

    signal POWER_ON, CLK, RESET, EN_A, EN_B : STD_LOGIC := '0';

begin
    uut : POWER_SEQUENCER
    port map( POWER_ON, CLK, RESET, EN_A, EN_B );

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

    stimulus : process
    begin
        -- Hier Stimuli einfuegen
    end process stimulus;

    en_a_assertion : process
    begin
        -- Hier Asserts einfuegen
    end process assertion;
end arch_0;
```

(a) Entwirf den Inhalt des Stimulus-Prozess, um die normale Funktion des Sequencers zu testen. (Normale Funktion: Nachdem POWER_ON auf '1' geht sind beide Spannungen aktiv, bevor POWER_ON wieder auf '0' geht). (3)

Solution: Wait Statements richtig (1) Zeiten richtig (1) Power_On richtig (1)

```
wait for 10 ns;

POWER_ON <= '1';

wait for 30 ns; -- mindestens 20 ns
```

```
POWER_ON <= '0';

wait;
```

(b) Entwirf den Inhalt des zugehörigen Assertion-Prozess für das Signal EN_A. Gib bei einem Fehler einen Error mit einer aussagekräftigen Nachricht aus. Die Syntax von assert lautet: (6)

```
assert condition
    report string
    severity severity_level ;
```

Solution: Zeiten richtig (2,5) Assert richtig (1) Pegel, Beschriftung richtig (2,5)

```
wait for 5 ns; -- vor anschalten

assert EN_A = '0'
    report "Enable A is high, even if power is off"
    severity error;

wait for 15 ns;

assert EN_A = '1'
    report "Enable A is low, even if power is on"
    severity error;

wait for 10 ns;

assert EN_A = '1'
    report "Enable A is low, even if power is on"
    severity error;

wait for 20 ns;

assert EN_A = '1'
    report "Enable A is low too early"
    severity error;

wait for 10 ns;

assert EN_A = '0'
    report "Enable A is not turning off"
    severity error;
```

Fehlersuche

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

   - Line 14. parse error, unexpected IDENTIFIER
   - Line 15. Undefined symbol 'fulladder'.
   - Line 27. Type of Q is incompatible with type of Q.
   - Line 28. Type of COUT is incompatible with type of CARRY.

   (a) Korrigiere die 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  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity four_bit_adder is
5      Port (  A : in  STD_LOGIC_VECTOR (1 downto 0);
6              B : in  STD_LOGIC_VECTOR (1 downto 0);
7              Q : out  STD_LOGIC_VECTOR (1 downto 0);
8              CARRY : out  STD_LOGIC_VECTOR);
9  end four_bit_adder;
10
11 architecture Behavioral of four_bit_adder is
12     signal c1 : STD_LOGIC := '0';
13
14 fa0: fulladder
15     port map(
16         A => A(0),
17         B => B(0),
18         CIN => '0',
19         Q => Q(0),
20         COUT => c1 );
21
22 fa1: entity work.fulladder
23     port map(
24         A => A(1),
25         B => B(1),
26         CIN => c1,
27         Q => Q,
28         COUT => CARRY );
29 end Behavioral;
```

Tabelle 1: Verbesserungen eintragen

Fehler	Zeile	Korrektur
1	8	CARRY : out STD_LOGIC);
2	13	begin
3	14	fa0: entity work.fulladder;
4	27	Q => Q(1)

Der Volladdierer wird benutzt um einen Taktgenerator zu bauen. Der Ausgang Q wird als Takt-Ausgang verwendet. Allerdings wird der Prozess, der auf einer steigende Flanke von Q ausgelöst wird, manchmal zu oft ausgeführt, es ist als kämen steigende Flanken aus dem nichts.

(b) Wie nennt man das Phänomen, dass unerwartete Signalzustände durch Umschalten auftreten? (1)

Solution: Hazards

(c) Wie kann man dieses Problem lösen? (2)

Solution: Durch die Einführung von NEXT- und REG-Signalen, wobei REG an die Clock weitergeleitet wird und in NEXT das aktuelle Ergebnis gespeichert wird (1). Bei einer rising edge einer CLK wird dann das NEXT in das REG Signal gespeichert (1).