DHBW
Duale Hochschule
Baden-Württemberg

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:

- 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	**Summe**
Punktzahl:	9	25	8	7	49
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) Warum wird in VHDL häufig von paralleler Ausführung gesprochen? Nenne zwei Gründe (2)

   Solution: Physikalische Abbildung von Logik, Signalzuweisung unabhängig von Reihenfolge

   (b) Was sind „Architecture“ und „Entity“ in VHDL, und wie arbeiten diese beiden zusammen? (3)

   Solution: Die Entity beschreibt Ein- und Ausgänge eines VHDL-Moduls, die Architecture beschreibt die Funktion. Eine Architecture wird immer einer Entity zugeordnet.

   (c) Wie kann die Flexibilität einer Entity durch Generics erhöht werden? Nenne ein Beispiel. (2)

   Solution: Durch Generics können Konstanten an die Entity weitergegeben werden. Diese können zum Beispiel variable Vektorbreiten oder Frequenzen erzeugen.

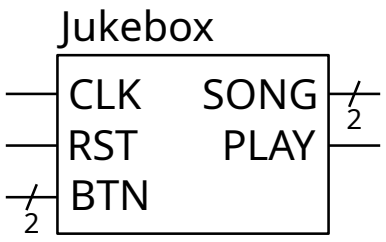
   (d) Welche Informationen enthält eine Constraints-Datei? Während welchem Prozesschritt bei der Bitfile-Generierung wird die Constraints-Datei gelesen? (2)

   Solution: Die Constraints-Datei enthält den Pin-Namen des Footprints. Die Constraints werden während der Implementierung gelesen und verarbeitet.

VHDL Programmieren

2. Die folgende Abbildung zeigt die Entity einer Komponente `JUKEBOX`. Der darauf folgende Code zeigt die Entity einer Komponente `MP3_PLAYER`. Diese Komponenten werden hintereinander geschaltet.

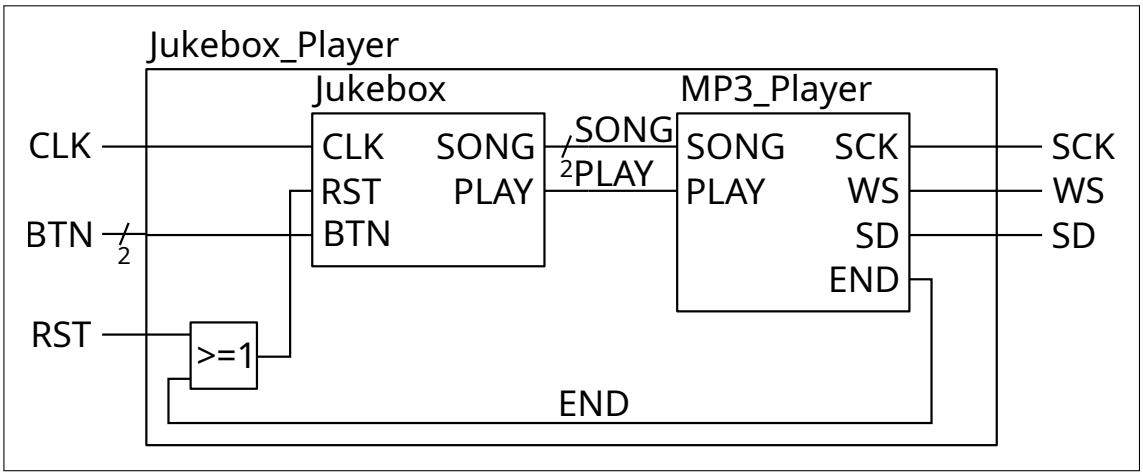
   - Über die beiden Buttons (BTN) kann jeweils ein Lied ausgewählt werden.
   - Die Auswahl wird dann durch die SONG-Ausgänge an den MP3-Player weitergegeben.
   - Das PLAY-Signal gibt an, ob der MP3-Player etwas abspielen soll (1) oder nicht (0).
   - Die Ausgabe des MP3-Player ist ein I2S-Bus, welcher aus den Signalen SCK, WS und SD besteht und an einen externen DAC weitergegeben wird.
   - Ist das abgespielte Lied zu Ende, gibt der MP3-Player einen High-Puls (1) auf dem END-Signal aus. Die Jukebox muss dann aufhören Musik abzuspielen.
   - CLK und RST sind externe Eingänge.
   - Die Clock hat eine Frequenz von 10 Hz.




```
entity MP3_PLAYER is
    Port (  SONG : in  STD_LOGIC_VECTOR (1 downto 0);
            PLAY : in  STD_LOGIC;
            SCK  : out STD_LOGIC;
            WS   : out STD_LOGIC;
            SD   : out STD_LOGIC;
            END  : out STD_LOGIC );
end MP3_PLAYER;
```

(a) Zeichne ein Blockschaltbild, der Top-Level-Entity `JUKEBOX_PLAYER`, die Jukebox und MP3-Player enthält. Zeichne auch die interne Verdrahtung und beschrifte die Signale, wo nötig. Für das `END`-Signal gibt es keinen eigenen Eingang in `JUKEBOX`, finde hierfür eine passende Lösung. *Hinweis: Die Box hier könnte etwas klein sein, nutze gerne die Rückseite.* (5)

> **Solution:** Top-Entity (1), Benennung ($\frac{1}{2}$), Jukebox abzeichnen ($\frac{1}{2}$), MP3-Player Entity (1), SONG/PLAY Signal inklusive Benennung (je $\frac{1}{2}$), Externe Signale (2), jeder Fehler -$\frac{1}{2}$. End an Reset ($\frac{1}{2}$), mit Oder-Gatter ($\frac{1}{2}$)



(b) Schreibe die Architecture zur Top-Level-Entity `JUKEBOX_PLAYER`. Binde dabei die Komponenten `MP3_PLAYER` und `JUKEBOX` ein. *Hinweis:* Nutze für die Instantiierung der Komponenten die folgende Kurzschreibweise und ergänze was noch fehlt: (6)

```
comp_0 : entity work.component
```

Solution: Architecture (1), Signale (2), Instantiierung (2), Reset-Signal und Oder-Gatter (1)

```
architecture Structural of JUKEBOX_PLAYER is
    signal SONG : STD_LOGIC_VECTOR(1 downto 0);
    signal PLAY    : STD_LOGIC;
    signal END     : STD_LOGIC;
    signal SIG_RST : STD_LOGIC;
begin
    jukebox_0 : entity work.JUKEBOX
    port map(
        CLK => CLK,
        RST => SIG_RST,
        BTN => BTN,
        SONG => SONG,
        PLAY => PLAY );

    player_0 : entity work.MP3_PLAYER
    port map(
        SONG => SONG,
        PLAY => PLAY,
        SCK => SCK,
        WS => WS,
        SD => SD,
        END => END );
```

```
    SIG_RST <= RST or END;
end Structural;
```

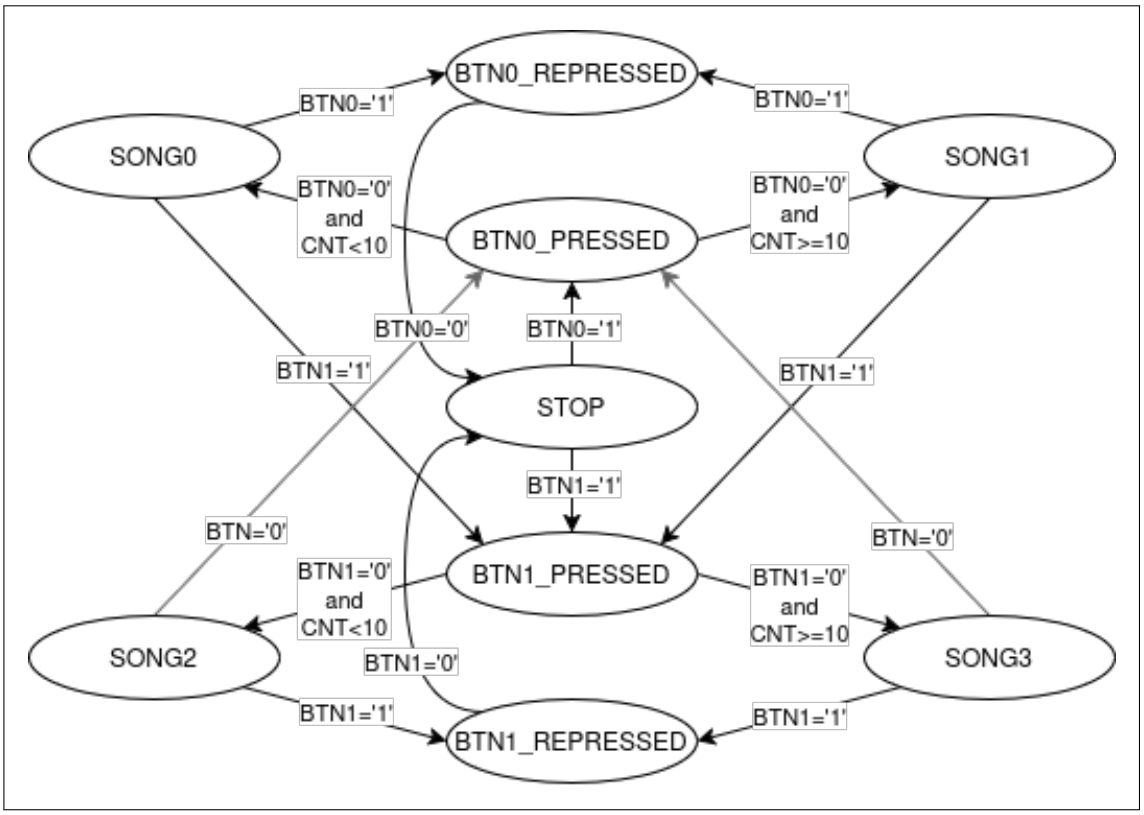
(c) Ein Hersteller von Grußkarten plant mit diesem Chip mehrere Millionen Karten pro Jahr herzustellen. Welche Realisierungsvariante wäre hier zu bevorzugen? Warum? Nenne einen weiteren Vorteil, den der Hersteller von Grußkarten durch diese Realisierungsvariante hat. (3)

Solution: Full-Custom Design, Hohe Stückzahlen, Optimierung auf Platzbedarf/Energieverbrauch

(d) JUKEBOX soll als State Machine umgesetzt werden. Dabei sind die in der Tabelle gezeigten Aktionen und Reaktionen auszuführen. Zeichne ein Diagramm der verschiedenen Zustände sowie deren Übergänge. Ergänze gegebenenfalls hierfür notwendige Signale. (7)

Aktion	Reaktion
Wenn kein Lied gespielt wird:	
BTN0 kurz gedrückt und losgelassen (<1 s)	Spielt Lied Nummer 0
BTN0 lange gedrückt und losgelassen (≥ 1 s)	Spielt Lied Nummer 1
BTN1 kurz gedrückt und losgelassen (<1 s)	Spielt Lied Nummer 2
BTN1 lange gedrückt und losgelassen (≥ 1 s)	Spielt Lied Nummer 3
Wenn gerade ein Lied gespielt wird:	
Gleicher BTN gedrückt	Stoppt Lied
Anderer BTN gedrückt	Startet anderes Lied

Solution: Zustände (STANDBY, BTN0_PRESSED, BTN1_PRESSED, PLAYING_0, PLAYING_1, PLAYING_2, PLAYING_3) ($3\frac{1}{2}$) Signal für Zähler (1) Signalabfrage richtig ($\frac{1}{2}$) Zustandsübergänge (Reset, 4x BTN to Playing, Stop, 2x Start anderes Lied) (2)



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

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

Solution: Case (1) und alle Zustände aus d) abgedeckt (1) SONG richtig (1) PLAY richtig (1)

```
case state is
    when s1 =>
        SONG <= x1;
        PLAY <= '1';
    when s2 =>
        SONG <= x1;
        PLAY <= '1';
    when s3 =>
        SONG <= not x1;
        PLAY <= '1';
    when s4 =>
        SONG <= not x1;
        PLAY <= '1';
end case;
end process out_fn;
```

Simulation

3. Der folgenden Code zeigt die Entity eines asynchronen 4-Bit-Addierers.

Listing 1: 4-Bit-Addierer

```
entity four_bit_adder is
    Port(
        A : in  UNSIGNED(3 downto 0);
        B : in  UNSIGNED(3 downto 0);
        Q : out UNSIGNED(3 downto 0);
        C : out STD_LOGIC );
end four_bit_adder;
```

(a) Programmiere die vollständige Testbench für den 4-Bit-Addierer (Entity und Architecture). Überprüfe darin das Ergebnis dere Addition mit einem assert-Statement eines Schweregrads, der die Simulation nicht unterbricht. (8)

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

Leere Entity(1), Architecture($\frac{1}{2}$), Signale ($1\frac{1}{2}$), Instantiierung (1), Stim-Prozess (Proc, Loop, Logik, Wait) (2), Assert-Prozess (Variable, Result, Assert, Severity) (2)

Solution:

```
entity four_bit_adder_tb is
end four_bit_adder_tb;

architecture Testbench of four_bit_adder_tb is
    signal A, B : UNSIGNED(3 downto 0) := to_unsigned(0, 4);
    signal Q : UNSIGNED(4 downto 0);
begin
    uut : entity work.four_bit_adder
        port map(
            A => A,
            B => B,
            Q => Q(3 downto 0),
            C => Q(4)
        );

    -- Stimulus process
    stim_proc: process
    begin
```

```
        for I in 0 to 15 loop
            for J in 0 to 15 loop
                A <= to_unsigned(I,4);
                B <= to_unsigned(J,4);

                wait for 50 ns;
            end loop;
        end loop;

        wait;
    end process;

    check_proc: process(q)
        variable result : unsigned(4 downto 0);
    begin

        result := A+B;

        assert q=result
            report "unexpected value. i = " & integer'image(to_integer(u
            severity warning;
    end process;

end Testbench;
```

Fehlersuche

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

```
Line 7: Syntax error near ")".
Line 21. Type of CLK_OUT is incompatible with type of 0.
Line 27. Parameter CLK_OUT of mode out can not be associated with a formal parameter
of mode in.
```

   (a) Bestimme die Fehler. Trage dazu die Zeile, in welcher der Fehler ist und die gesamte, korrigierte Zeile oder eine Erklärung was zu tun ist um den Fehler zu beheben in die Tabelle ein. Libraries sind richtig eingebunden. (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 taktteiler is
2      Generic (
3              FREQ_IN : integer := 50E6;
4              FREQ_OUT : integer := 2 );
5      Port (  CLK_IN : in  STD_LOGIC;
6              RST : in  STD_LOGIC;
7              CLK_OUT : out  STD_LOGIC;);
8  end taktteiler;
9
10 architecture Behavioral of taktteiler is
11     signal CNT : integer range 0 to FREQ_IN/(2*FREQ_OUT)-1
12         := 0;
13     constant MAX : integer range 0 to FREQ_IN/(2*FREQ_OUT)-1
14         := FREQ_IN/(2*FREQ_OUT)-1;
15 begin
16
17     proc_cnt : process(CLK_IN, RST)
18     begin
19         if( RST = '1' ) then
20             CNT <= 0;
21             CLK_OUT <= 0;
22         elsif( CLK_IN = '1' ) then
23             if CNT < MAX then
24                 CNT <= CNT+1;
25             else
26                 CNT <= 0;
27                 CLK_OUT <= not CLK_OUT;
28             end if;
29         end if;
30     end process proc_cnt;
31
32 end Behavioral;
```

Tabelle 1: Verbesserungen eintragen

Fehler	Zeile	Korrektur
1(1)	7	CLK_OUT : out STD_LOGIC);
2(1)	21	CLK_OUT <= '0';
3(2)	27	Einführen eines Signals als Zwischenspeicher für CLK_OUT, da aus OUT nicht zurück gelesen werden kann.

(b) Nachdem die Syntax-Fehler korrigiert sind, lässt sich der Code auf der Hardware ausführen. Doch leuchtet die mit OUT_CLK verbundene LED nur schwach auf, ein Blinken ist nicht zu erkennen. (3)

Worin liegt der Fehler? Erkläre, warum dieser vorliegt und wie man diesen korrigiert.

Solution: Es gibt keine Abfrage nach einer Taktflanke. (1)
Stattdessen wird immer hochgezählt, solange CLK = 1 ist. (1)
Einführen einer rising_edge-Abfrage in Zeile 22 (1)