Hochschule Karlsruhe – Technik und Wirtschaft
Fakultät für Informatik und Wirtschaftsinformatik
Prof. Dr. A. Ditzinger / Dipl.-Inform. (FH) O. Gniot / Dipl.-Ing. J. Krastel

D i g i t a l l a b o r

Versuch: MACH Programmierung

Ziel: Im heutigen Versuch gehen wir zur nächsten Technologiestufe, den CPLDs weiter. Durch Verwendung eines MACH Bausteines können wir gleichzeitig auf die In-System-Programmierung zurückgreifen. Sie brauchen den Baustein also nicht mehr in einem speziellen Gerät zu programmieren, sondern über den JTAG Anschluss direkt auf der Platine.

Sie sollten heute die Beschreibung von sequentiellen Schaltungen mit VHDL und die Simulation dieser Schaltungen kennen lernen. Natürlich sollen Sie auch Ihre Kenntnisse über das Erstellen hierarchischer Designs und den Umgang mit einer Testbench vertiefen.

Zu jeder Aufgabe ist in der Dokumentation ein Screen Shot des Simulationsresultates gefordert, den Sie mit dem „Snipping Tool“ einfach aufzeichnen können.

Das Ziel des heutigen Versuchs ist eine Ampelsteuerung, die aus vier Komponenten gemäß Bild 1 aufgebaut ist. Die einzelnen Komponenten werden nun Schritt für Schritt entworfen. Bleiben Sie für alle Aufgaben im gleichen Projekt und verwenden Sie durchgängig den Datentyp **„std_logic“**.

Für die Durchführung des Versuchs brauchen Sie ein ispMach-Board und ein I/O-Board.

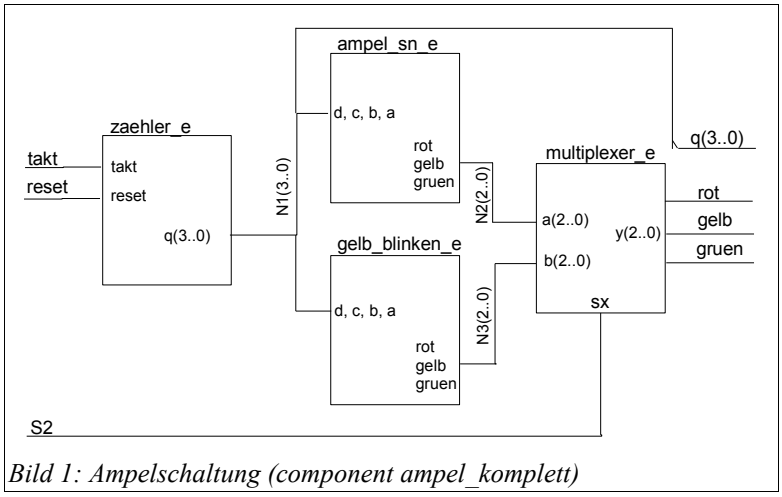


Bild 1: Ampelschaltung (component ampel_komplett)

Aufgabe 1

Erstellen Sie einen 4-Bit Binärzähler mit asynchronem Reset in VHDL. Die Entity muss den Namen "zaehler_e" haben. Benutzen Sie folgende Port Namen:

```
takt, reset : in  std_logic;
q           : out std_logic_vector(3 downto 0)
```

Simulieren Sie das Modell durch Vorgabe der Stimuli per Simulator. Was passiert, wenn Sie Reset einfach auf '0' setzen und den Takt loslaufen lassen? Ist das ein Fehler? Wie lässt sich diese Situation vermeiden?

Aufgabe 2

Zur Simulation der Aufgabe 1 erstellen Sie sich eine Testbench mit zwei Prozessen:

```
tb_res: process -- Prozess für Reset und ggf. weitere Signale
      begin
        reset <= .... ; -- ab hier folgen Ihre Zuweisungen
        wait for 10 ns; -- mit wait for ... getrennt.
        reset <= .... ; usw.
        wait ; -- Schluss
    end process;

tb_takt: process -- zur Takterzeugung, Periode 100ns
         begin
           takt <= '0';        -- initialisiere
           loop
             wait for 50 ns;   -- einen halben Takt warten
             takt <= not takt; -- takt kippen
           end loop;
       end process;
```

Simulieren Sie Ihr Design mit dieser Testbench.

Programmieren des CPLD-Bausteins:

Der CPLD-Baustein auf dem ispMach-Board hat einen eingebauten Oszillator, der noch konfiguriert werden muss. Dazu dient die Datei "HARD_A2.vhd" (siehe Anhang A). Sie finden die Datei im LAT unter `W:\IWI-I\DTL_Vorlage`. Kopieren Sie sich die Datei in ihr Projektverzeichnis.

Importieren sie die Datei in ihr Projekt. Wenn Sie sich an die Namens-Vorgaben gehalten haben, müsste ihr Projekt wie in Bild 2 dargestellt aussehen. Wichtig ist, dass HARD_A2e das Top Modul ist.

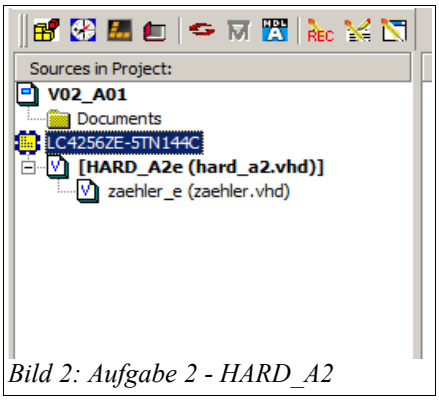


Bild 2: Aufgabe 2 - HARD_A2

Weisen Sie mit dem Constraint Editor die Pins zu. Zähler-ausgang => LEDs D4 .. D1, reset => Schalter S1, Test-Eingang T1 => Taster 1. (Hinweis: Taster und Schalter sind als Pull UP zu konfigurieren).

Die zugehörigen Pins können der Bedienungsanleitung „Arbeiten mit dem ispMACH 4000ZE Breakout Board“ entnommen werden.

Nach dem erfolgreichen Compilieren programmieren Sie nun die Hardware.

Nach dem erfolgreichen Test löschen Sie HARD_A2e wieder aus dem Projekt.

Aufgabe 3

Wie Sie in TI 1 gesehen haben, kann es ganz schön mühsam sein, eine rein kombinatorische Schaltung mit KV-Diagrammen zu entwerfen. Hier machen wir es besser: Entwerfen Sie ein VHDL Modell für einen rein kombinatorischen Ampel Steuerungsblock, der die in Tabelle 1 gezeigte Funktionstabelle mit 16 Ampelphasen implementiert.

Simulieren Sie den Block mit einer Testbench, die die 16 Eingangswerte mit Hilfe einer FOR-Schleife erzeugt.

Eingänge des Schaltnetzes				Ausgänge des Schaltnetzes			
D	C	B	A	GRÜN	GELB	ROT	
0	0	0	0	1	0	0	GRÜN-Phase
0	0	0	1	1	0	0	
0	0	1	0	1	0	0	
0	0	1	1	1	0	0	
0	1	0	0	0	1	0	GELB-Phase
0	1	0	1	0	1	0	
0	1	1	0	0	0	1	ROT-Phase
0	1	1	1	0	0	1	
1	0	0	0	0	0	1	
1	0	0	1	0	0	1	
1	0	1	0	0	0	1	
1	0	1	1	0	0	1	
1	1	0	0	0	0	1	
1	1	0	1	0	0	1	
1	1	1	0	0	1	1	ROT-GELB-Phase
1	1	1	1	0	1	1	

Tabelle 1: Funktionstabelle des Schaltnetzes

Aufgabe 4

Fügen Sie den Ampel Steuerungsblock aus der vorigen Aufgabe in ein Strukturmodell ein, das den Zähler und das Ampel Schaltnetz instanziert. Der Name der Entity sollte "zaehler_ampel_e" heißen (siehe Bild 3). Neben den Ports aus Aufgabe 1 kommen folgende Ports hinzu:

rot, gelb, gruen : out std_logic

Simulation mit der Testbench aus Aufgabe 2.

Zum Programmieren der Hardware kopieren Sie die Datei "HARD_A4.vhd" (siehe Anhang B) in ihr Projektverzeichnis und importieren sie in ihr Projekt.

Wenn Sie sich an obige Vorgaben gehalten haben, müsste ihr Projekt wie in Bild 4 dargestellt aussehen.

Weisen Sie die benötigten Pins zu. Es müssen alle Signals aus der Liste einem entsprechenden Pin zugewiesen werden.

Testen Sie Ihre Schaltung.

Nach dem erfolgreichen Test löschen Sie „HARD_A4e“ wieder aus ihrem Projekt.

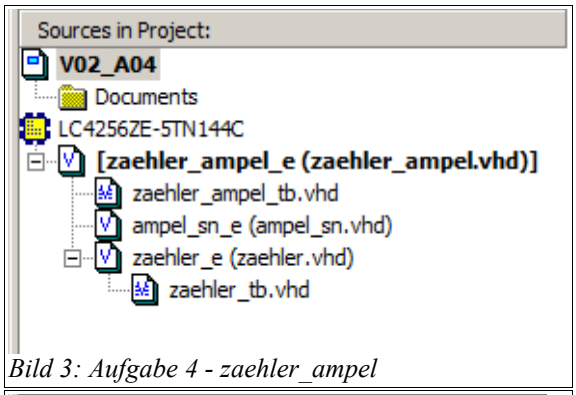


Bild 3: Aufgabe 4 - zaehler_ampel

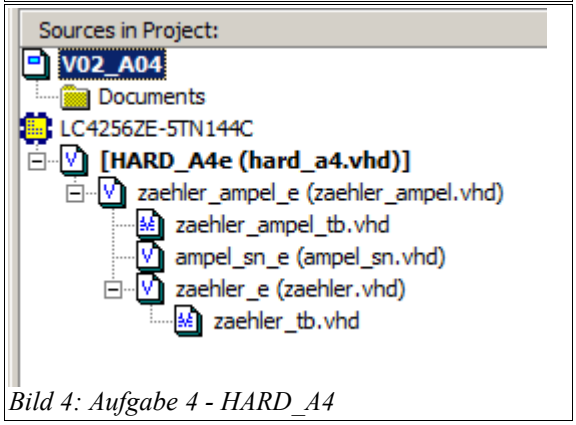


Bild 4: Aufgabe 4 - HARD_A4

Aufgabe 5

Entwerfen Sie einen zusätzlichen Ampel Steuerungsblock als getrenntes VHDL-Modul. Er liefert ebenfalls die Signale "rot / gelb / gruen" und zwar so, dass die gelbe LED zwei Takte leuchtet und zwei Takte aus ist. Die anderen sind immer aus.

Simulation mit der Testbench aus Aufgabe 3.

Aufgabe 6

Entwerfen Sie ein sequentielles VHDL Modell eines 2:1 Multiplexers für zwei je drei Bit breite std_logic Vektoren A und B, die abhängig von einem Signal Select auf den drei Bit breiten Ausgangsvektor Y durchgeschaltet werden.

Test per Simulation.

Aufgabe 7

Als letztes fügen Sie alle Komponenten so wie in Bild 1 gezeigt zusammen. Die Entity muss ampel_komplett_e heißen. Neben den Ports aus Aufgabe 4 kommt noch der Port:

S2 : in std_logic

zur Umschaltung der Betriebsarten hinzu. Simulieren Sie mit Hilfe der um S2 erweiterten Testbench aus Aufgabe 2.

Zum Programmieren der Hardware kopieren Sie sich die Datei "HARD_A7.vhd" (siehe Anhang C) in ihr Projektverzeichnis und importieren sie in ihr Projekt.

Wenn Sie sich an obige Vorgaben gehalten haben, müsste ihr Projekt wie in Bild 5 dargestellt aussehen.

Weisen Sie die benötigten Pins zu. Verwenden Sie Schalter S2 als Betriebsarten Umschalter.

Testen Sie Ihre Schaltung.

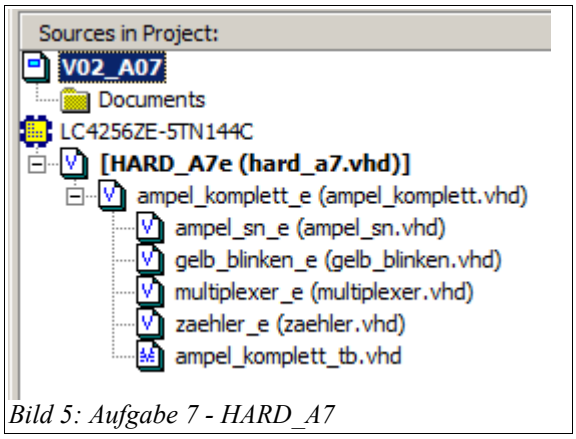
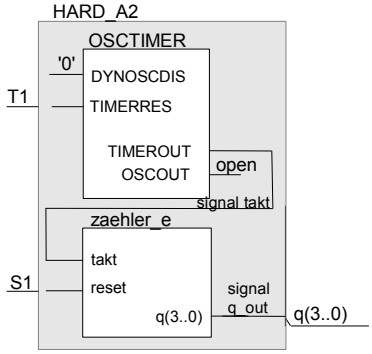


Bild 5: Aufgabe 7 - HARD_A7

Anhang A: Datei "HARD_A2.vhd"

```
library ieee;
library MACH;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use MACH.components.all;

entity HARD_A2e is
  port (S1      : in  std_logic; -- Schalter S1 => reset
        T1      : in  std_logic; -- Taster   T1 => Test-Eingang
        q       : out std_logic_vector(3 downto 0)); -- Zaehlerausgang
end;

architecture HARD_A2a of HARD_A2e is
  signal takt   : std_logic;
  signal q_out  : std_logic_vector(3 downto 0);

component OSCTIMER
  generic (TIMER_DIV : string);
  port (DYNOSCDIS    : in  std_logic;
        TIMERRES     : in  std_logic;
        OSCOUT       : out std_logic;
        TIMEROUT     : out std_logic);
end component;

component zaehler_e is
  port (takt, reset : in  std_logic;
        q           : out std_logic_vector(3 downto 0));
end component;

begin
  i1: OSCTIMER
  generic map (TIMER_DIV => "1048576") -- Teilungsfaktor - es sind nur 3 Werte
                                       -- zulässig: 128, 1024 und 1048576
  port    map (DYNOSCDIS => '0',
               TIMERRES  => not T1,    -- Taster T1 zum Test
               OSCOUT    => open,
               TIMEROUT  => takt);     -- auf signal takt

  i2 : zaehler_e  port map (takt=>takt, reset=>S1, q=>q_out);

  q <= not q_out; -- aktuellen Zaehlerstand den LEDs invertiert zuweisen
                  -- da LEDs leuchten, wenn eine 0 anliegt


end HARD_A2a;
```



Anhang B: Datei "HARD_A4.vhd"

```
library ieee;
library MACH;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use MACH.components.all;

entity HARD_A4e is
  port (S1               : in  std_logic; -- Schalter S1 => reset
        T1               : in  std_logic; -- Taster T1 => Test-Eingang
        rot, gelb, gruen : out std_logic;
        q                : out std_logic_vector(3 downto 0)); -- Zaehlerausgang
end;

architecture HARD_A4a of HARD_A4e is
  signal takt   : std_logic;
  signal q_out  : std_logic_vector(3 downto 0);

component OSCTIMER
  generic (TIMER_DIV : string);
  port    (DYNOSCDIS : in  std_logic;
           TIMERRES  : in  std_logic;
           OSCOUT    : out std_logic;
           TIMEROUT  : out std_logic);
end component;

component zaehler_ampel_e is
  port (takt              : in  std_logic;
        reset             : in  std_logic;
        q                 : out std_logic_vector(3 downto 0);
        rot, gelb, gruen  : out std_logic);
end component;

begin
  i1: OSCTIMER
  generic map (TIMER_DIV => "1048576") -- Teilungsfaktor
  port    map (DYNOSCDIS => '0',
               TIMERRES  => not T1,    -- Taster T1 zum Test
               OSCOUT    => open,
               TIMEROUT  => takt);     -- auf signal takt

 i2: zaehler_ampel_e port map (takt => takt, reset => S1,
                               rot => rot, gelb => gelb, gruen => gruen,
                               q => q_out);

  q <= not q_out; -- aktuellen Zaehlerstand den LEDs invertiert zuweisen
                  -- da LEDs leuchten, wenn eine 0 anliegt

end HARD_A4a;
```

Anhang C: Datei "HARD_A7.vhd"

```
library ieee;
library MACH;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use MACH.components.all;

entity HARD_A7e is
  port (S1               : in  std_logic; -- S1 => reset
        S2               : in  std_logic; -- S2 => Sx
        T1               : in  std_logic; -- Test-Eingang
        rot, gelb, gruen : out std_logic;
        q                : out std_logic_vector(3 downto 0)); - Zaehlerausgang
end;

architecture HARD_A7a of HARD_A7e is
  signal takt   : std_logic;
  signal q_out  : std_logic_vector(3 downto 0);

component OSCTIMER
  generic (TIMER_DIV : string);
  port (DYNOSCDIS    : in  std_logic;
        TIMERRES     : in  std_logic;
        OSCOUT       : out std_logic;
        TIMEROUT     : out std_logic);
end component;

component ampel_komplett_e is
  port (takt         : in  std_logic;
        reset        : in  std_logic;
        S2           : in  std_logic; -- Umschalter, Ampel oder gelb blinken
        q            : out std_logic_vector(3 downto 0);
        rot, gelb, gruen : out std_logic);
end component;

begin
  i1: OSCTIMER
  generic map (TIMER_DIV => "1048576")
  port    map (DYNOSCDIS => '0',
               TIMERRES  => not T1,
               OSCOUT    => open,
               TIMEROUT  => takt);

 i2: ampel_komplett_e port map (takt=>takt, reset=>S1, S2=>S2,
                                q=>q_out,
                                rot=>rot, gelb=>gelb, gruen=>gruen);
q <= not q_out; -- aktuellen Zaehlerstand den LEDs invertiert zuweisen
                -- da LEDs leuchten, wenn eine 0 anliegt

end HARD_A7a;
```