**Protokollierung zu Versuch 7**

**Gruppe 12**

Yazan Al-Kabbani (1648827)

Osama Hammoud (1536285)

Ahmet Taner Kahraman (1660926)

**Protokollierung Aufgabe 1**

actor.vhd

*----------------------------------------------------------------------------*

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

library work;

use work.AshaTypes.ALL;

entity actor is

    Port (

        Clock : in std\_logic; *--! Taktsignal*

        Reset : in std\_logic; *--! Resetsignal*

        Switches : in std\_logic\_vector(3 downto 0); *--! Die acht Schalter*

        ButtonsIn : in std\_logic\_vector(3 downto 0); *--! Die vier Taster*

        SensorVibe : in std\_logic; *--! Eingang: Virbationssensor*

        SensorDoor : in std\_logic; *--! Eingang: Tuersensor*

        ADCRegister : in ADCRegisterType; *--! Datenregister aller ADC-Werte*

        LEDsOut : out std\_logic\_vector(5 downto 0); *--! Die acht LEDs*

        SevenSegmentValue : out std\_logic\_vector (15 downto 0);*--! treibt die 7-Segment-Anzeigen*

        PWM1FanInsideValue : out std\_logic\_vector(7 downto 0); *--! Signalquellwert Luefter innen*

        PWM2FanOutsideValue : out std\_logic\_vector(7 downto 0); *--! Signalquellwert Luefter aussen*

        PWM3LightValue : out std\_logic\_vector(7 downto 0); *--! Signalquellwert Licht*

        PWM4PeltierValue : out std\_logic\_vector(7 downto 0); *--! Signalquellwert Peltier*

        PeltierDirection : out std\_logic; *--! Signalquellwert Peltier Richtung*

*----- Werte von Bluetooth*

        LEDsBT : in std\_logic\_vector(5 downto 0); *--! Die acht LEDs*

        SevenSegmentValueBT : in std\_logic\_vector (15 downto 0); *--! 7SegmentEingang von BT*

        PWM1FanInsideValueBT : in std\_logic\_vector(7 downto 0); *--! Signalquellwert Luefter innen, von Bt*

        PWM2FanOutsideValueBT : in std\_logic\_vector(7 downto 0); *--! Signalquellwert Luefter aussen, von Bt*

        PWM3LightValueBT : in std\_logic\_vector(7 downto 0); *--! Signalquellwert Licht, von Bt*

        PWM4PeltierValueBT : in std\_logic\_vector(7 downto 0); *--! Signalquellwert Peltier, von Bt*

        PeltierDirectionBT : in std\_logic; *--! Signalquellwert Peltier Richtung, von Bt*

*----- Werte von Regelung*

        PWM1FanInsideValueControl : in std\_logic\_vector(7 downto 0); *--! Signalquellwert Luefter innen, von Regelung*

        PWM2FanOutsideValueControl : in std\_logic\_vector(7 downto 0); *--! Signalquellwert Luefter aussen, von Regelung*

        PWM4PeltierValueControl : in std\_logic\_vector(7 downto 0); *--! Signalquellwert Peltier, von Regelung*

        PeltierDirectionControl : in std\_logic; *--! Signalquellwert Peltier Richtung, von Regelung*

        ControlLightDiffOut : in unsigned(12 downto 0); *--! Aktuelle Regeldifferenz Licht*

        ControlTempDiffOut : in unsigned(12 downto 0) *--! Aktuelle Regeldifferenz Temperatur*

    );

end actor;

architecture Behavioral of actor is

*-- Zustandsautomat für Modus Auswahl*

type state\_typeM is (Asha1,Asha2,Asha3,

                    SensorRead1,SensorRead2,SensorRead3,

                    ManualActor1,ManualActor2,ManualActor3,

                    AutoActor1,AutoActor2,AutoActor3,

                    Bluetooth1,Bluetooth2,Bluetooth3); *--type of state machine(M for Modus).*

signal current\_m,next\_m:state\_typeM;    *--current and next state declaration.*

*-- Zustandsautomat für Sensor Zustaende.*

type state\_typeS is (Init, Init2, Light, Light2, TempIn, TempIn2, TempOut, TempOut2, Vibe, Vibe2, Door, Door2 ); *--type of state machine(S for Sensor).*

signal current\_s,next\_s: state\_typeS; *--current and next state declaration.*

begin

*-- FSM Prozess zur Realisierung der Speicherelemente - Abhängig vom Takt den nächsten Zustand setzen*

*--> In Versuch 6 zu implementieren!-*

FSM\_seq: process (Clock,Reset)

begin

    if(Reset='1') then *--Beim Reset Signal*

        current\_m<=Asha1; *--den Zustand fÃ¼r den Modus zurÃ¼cksetzen auf Asha1*

        current\_s<=Init; *--und den Zustand fÃ¼r den gewÃ¤hlten Sensor auf Init setzen (relevant fÃ¼r Modus 1)*

    end if;

    if(rising\_edge(Clock)) then *--Bei jedem Clock Signal*

        current\_m<=next\_m; *--den Zustand fÃ¼r den Modus aktuallisieren*

        current\_s<=next\_s;

    end if; *--und den Zustand fÃ¼r den gewÃ¤hlten Sensor aktuallisieren (relevant fÃ¼r Modus 1)*

end process FSM\_seq;

*-- FSM Prozess (kombinatorisch) zur Realisierung der Modul Zustände aus den Typen per Switch Case: state\_typeM*

*-- Setzt sich aus aktuellem Zustand und folgendem Zustand zusammen: current\_m,next\_m*

*--> In Versuch 6-10 zu implementieren*

FSM\_modul:process(current\_m, ButtonsIn(0),ButtonsIn(1))

begin

    next\_m <= current\_m;

    case current\_m is

        when Asha2 =>

            if (ButtonsIn(0)='1') then

                next\_m <= Asha3;

            elsif (ButtonsIn(1)='1') then

                next\_m <= Asha1;

            end if;

        when SensorRead2 =>

            if (ButtonsIn(0)='1') then

            next\_m <= SensorRead3;

            elsif (ButtonsIn(1)='1') then

            next\_m <= SensorRead1;

            end if;

        when ManualActor2 =>

            if (ButtonsIn(0)='1') then

                next\_m <= ManualActor3;

            elsif (ButtonsIn(1)='1') then

                next\_m <= ManualActor1;

            end if;

        when AutoActor2 =>

            if (ButtonsIn(0)='1') then

                next\_m <= AutoActor3;

            elsif (ButtonsIn(1)='1') then

                next\_m <= AutoActor1;

            end if;

        when Bluetooth2 =>

            if (ButtonsIn(0)='1') then

                next\_m <= Bluetooth3;

            elsif (ButtonsIn(1)='1') then

                next\_m <= Bluetooth1;

            end if;

        when Asha1 =>

            if (ButtonsIn(1)='0') then

                next\_m <= Bluetooth2;

            end if;

        when Asha3 =>

            if (ButtonsIn(0)='0') then

                next\_m <= SensorRead2;

            end if;

        when SensorRead1 =>

            if (ButtonsIn(1)='0') then

                next\_m <= Asha2;

            end if;

        when SensorRead3 =>

            if (ButtonsIn(0)='0') then

                next\_m <= ManualActor2;

            end if;

        when ManualActor1 =>

            if (ButtonsIn(1)='0') then

                next\_m <= SensorRead2;

            end if;

        when ManualActor3 =>

            if (ButtonsIn(0)='0') then

                next\_m <= AutoActor2;

            end if;

        when AutoActor1 =>

            if (ButtonsIn(1)='0') then

                next\_m <= ManualActor2;

            end if;

        when AutoActor3 =>

            if (ButtonsIn(0)='0') then

                next\_m <= Bluetooth2;

            end if;

        when Bluetooth1=>

            if (ButtonsIn(1)='0') then

                next\_m <= AutoActor2;

            end if;

        when Bluetooth3 =>

            if (ButtonsIn(0)='0') then

                next\_m <= Asha2;

            end if;

    end case;

end process;

*-- FSM Prozess (kombinatorisch) zur Realisierung der Ausgangs- und Übergangsfunktionen*

*-- Hinweis: 12 Bit ADC-Sensorwert für Lichtsensor: ADCRegister(3),*

*-- 12 Bit ADC-Sensorwert für Temp. (außen): ADCRegister(1),*

*-- 12 Bit ADC-Sensorwert für Temp. (innen): ADCRegister(0),*

*--> In Versuch 6-10 zu implementieren!-*

FSM\_comb:process (current\_s,current\_m, ButtonsIn(2) , ADCRegister, SensorVibe, SensorDoor)

begin

*-- to avoid latches always set current state (Versuch 6)*

    next\_s <= current\_s;

*-- Modus 0: "ASHA" Auf 7 Segment Anzeige*

    case current\_m is

        when Asha1|Asha2|Asha3 => *--ASHA state*

            LEDsOut<= b"111111";

            SevenSegmentValue <= x"FFFF";

*-- Versuch 6*

*-- Modus 1: "Sensorwerte Auslesen"*

*-- Durchschalten der Sensoren per BTN2*

*-- Ausgabe des ausgewalten Sensors ueber SiebenSegmentAnzeige*

*-- when state ... TODO*

        when SensorRead1|SensorRead2|SensorRead3 => *--Modus 1: Sensoren auslesen*

            LEDsOut(5 downto 4)<= "00";

*-- Wenn Button(2) gedr�ckt wird, wird in den �bergangszustand gewechselt*

            if(ButtonsIn(2)='1') then

                case current\_s is

                    when Init =>   *-- Init*

                        next\_s <= Init2;

                        LEDsOut(3 downto 0)<= "0000"; *-- LEDs anpassen*

                    when Init2 =>   *-- Init*

                        next\_s <= Init2;

                        LEDsOut(3 downto 0)<= "0000"; *-- LEDs anpassen*

                    when Light =>   *-- Lichtssensor*

                        next\_s <= Light2;

                        LEDsOut(3 downto 0)<= "1000"; *-- LEDs anpassen*

                    when Light2 =>  *-- Lichtssensor*

                        next\_s <= Light2;

                        LEDsOut(3 downto 0)<= "1000"; *-- LEDs anpassen*

                    when TempIn =>  *-- Temperatursensor innen*

                        next\_s <= TempIn2;

                        LEDsOut(3 downto 0)<= "0100"; *-- LEDs anpassen*

                    when TempIn2 => *-- Temperatursensor innen*

                        next\_s <= TempIn2;

                        LEDsOut(3 downto 0)<= "0100"; *-- LEDs anpassen*

                    when TempOut => *-- Temperatursensor au�en*

                        next\_s <= TempOut2;

                        LEDsOut(3 downto 0)<= "0010"; *-- LEDs anpassen*

                    when TempOut2 => *-- Temperatursensor au�en*

                        next\_s <= TempOut2;

                        LEDsOut(3 downto 0)<= "0010"; *-- LEDs anpassen*

                    when Vibe =>   *-- Vibrationssensor*

                        next\_s <= Vibe2;

                        LEDsOut(3 downto 0)<= "0001"; *-- LEDs anpassen*

                    when Vibe2 =>  *-- Vibrationssensor*

                        next\_s <= Vibe2;

                        LEDsOut(3 downto 0)<= "0001"; *-- LEDs anpassen*

                    when Door =>   *-- Doorssensor*

                        next\_s <= Door2;

                        LEDsOut(3 downto 0)<= "0000"; *-- LEDs anpassen*

                    when Door2 => *-- Doorssensor*

                        next\_s <= Door2;

                        LEDsOut(3 downto 0)<= "0000"; *-- LEDs anpassen*

                end case;

            end if;

*-- Wenn Button(2) im �bergangszustand losgelassen wird, wird in den n�chsten Zustand gewechselt*

            if(ButtonsIn(2)='0') then

                case current\_s is

                    when Init =>

                        next\_s <= Init;

                        SevenSegmentValue <= X"ffff";

                        LEDsOut(3 downto 0) <= b"1010";

                    when Init2 =>

                        next\_s <= Light;

                    when Light =>

                        next\_s <= Light;

                        SevenSegmentValue <= X"0" & ADCRegister(3);

                        LEDsOut(3 downto 0) <= b"1000";

                    when Light2 =>

                        next\_s <= TempIn;

                    when TempIn =>

                        next\_s <= TempIn;

                        SevenSegmentValue <= X"0" & ADCRegister(0);

                        LEDsOut(3 downto 0) <= b"0100";

                    when TempIn2 =>

                        next\_s <= TempOut;

                    when TempOut =>

                        next\_s <= TempOut;

                        SevenSegmentValue <= X"0" & ADCRegister(1);

                        LEDsOut(3 downto 0) <= b"0010";

                    when TempOut2 =>

                        next\_s <= Vibe;

                    when Vibe =>

                        next\_s <= Vibe;

                        SevenSegmentValue <= b"000000000000000" & SensorVibe;

                        LEDsOut(3 downto 0) <= b"0001";

                    when Vibe2 =>

                        next\_s <= Door;

                    when Door =>

                        next\_s <= Door;

                        SevenSegmentValue <= b"000000000000000" & SensorDoor;

                        LEDsOut(3 downto 0) <= b"0000";

                    when Door2 =>

                        next\_s <= Init;

                end case;

            end if;

*-- Versuch 7*

*-- Modus 2: Manuelle Aktorsteuerung*

*-- nur erlauben, wenn keine Regelung aktiv ist!*

*-- when ... TODO*

        when ManualActor1|ManualActor2|ManualActor3 =>

            LEDsOut(5 downto 4)<= "01";

*-- LEDsOut(3 downto 0)<= "0101";*

            if (Switches(0) = '1') then

                PWM1FanInsideValue <= b"11111111" *-- Innenl�fter auf 100%*

            else

                PWM1FanInsideValue <= b"00000000" *-- Innenl�fter aus*

            end if;

            if (Switches(1) = '1') then

                PWM2FanOutsideValue <= b"11111111"; *--Au�enl�fter auf 100%*

            else

                PWM2FanOutsideValue <= b"00000000"; *--Au�enl�fter aus*

            end if;

            if (Switches(2) = '1') then

                PWM3LightValue <= b"11111111"; *--Licht auf 100%*

            else

                PWM3LightValue <= b"00000000"; *--Licht aus*

            end if;

            if (Switches(3) = '1') then

                PWM4PeltierValue <= b"11111111"; *--Peltier auf 100%*

                PeltierDirection <= '1'; *--Peltier auf Heizen*

            else

                PWM4PeltierValue <= b"00000000"; *--Peltier aus*

                PeltierDirection <= '1'; *--Peltier auf Heizen*

            end if;

        end if;

*-- Versuch 9*

*-- Modus 3: geregelte Aktorsteuerung*

*-- when ... TODO*

        when AutoActor1|AutoActor2|AutoActor3 =>

            LEDsOut(5 downto 4)<= "10";

            LEDsOut(3 downto 0)<= "1010";

*-- Versuch 10*

*-- Modus 4: Steuerung ueber Smartphone-App*

*-- when ... TODO*

        when Bluetooth1|Bluetooth2|Bluetooth3 =>

            LEDsOut(5 downto 4)<= "11";

            LEDsOut(3 downto 0)<= "0101";

 when others =>

*-- DEFAULT Werte setzen TODO*

 end case;

end process;

end Behavioral;

asha-vibe.vhd

*----------------------------------------------------------------------------*

*--! Standardbibliothek benutzen*

library IEEE;

*--! Logikelemente verwenden*

use IEEE.STD\_LOGIC\_1164.ALL;

*--! Numerisches Rechnen ermoeglichen*

use IEEE.NUMERIC\_STD.ALL;

*--! @brief ASHA-Modul - Signale des Vibrationssensors aufbereiten*

*--! @details Dieses Modul Empfaengt das Signal des Vibrationssensors im Haus*

*--! und wertet es aus. Ueber das Signal SensorVibeHouseOn kann dieses Modul*

*--! dem Hauptmodul signalisieren, wenn es das Haus auschalten soll.*

entity AshaVibe is

  Port (

    Clock : in std\_logic;         *--! Taktsignal*

    Reset : in std\_logic;         *--! Resetsignal*

    SensorVibe : in std\_logic;      *--! Vibrationssensorsignal direkt vom Haus*

    SensorVibeHouseOn : out std\_logic); *--! Haus an/aus-Signal des Vibe-Moduls*

end AshaVibe;

architecture Behavioral of AshaVibe is

  signal SensorVibeOld : std\_logic;   *--! Speicherregister fuer alten Sensorzustand*

begin

*--folgende Zeile l�schen, wenn Vibe\_Detect implementiert ist!*

*-- SensorVibeHouseOn <= '1';*

*--! Realisierung des Vibrationsdetektors, Versuch 7*

  Vibe\_Detect:Process (Clock) *-- Vibrationsdetektor*

  begin

    if rising\_edge(Clock) then

*-- TODO*

*-- Bei einer Erschütterung wird der Ausgang so gesetzt, dass das Haus ausgeschalt wird*

      if (SensorVibe = '0') then

        SensorVibeHouseOn <= '0';

*-- Beim Reset wird der Ausgang zurückgesetzt*

      elsif (Reset = '1') then

        SensorVibeHouseOn <= '1';

    end if;

  end Process Vibe\_Detect; *-- Vibrationsdetektor*

end Behavioral;

asha-pwm.vhd

*----------------------------------------------------------------------------*

*--! Standardbibliothek benutzen*

library IEEE;

*--! Logikelemente verwenden*

use IEEE.STD\_LOGIC\_1164.ALL;

*--! Numerisches Rechnen ermoeglichen*

use IEEE.NUMERIC\_STD.ALL;

*--! @brief ASHA-Modul - PWM-Signale erzeugen*

*--! @details  Dieses Modul erzeugt die PWM-Signale fuer PWM-Aktoren*

entity AshaPWM is

  Port (

    Clock : in std\_logic;                   *--! Taktsignal*

    Reset : in std\_logic;                   *--! Resetsignal*

    EnPWMClock : in std\_logic;                *--! Enable-Signal fuer die PWM-Abarbeitung*

    PWM1FanInsideValue : in std\_logic\_vector(7 downto 0);   *--! Signalquellwert Luefter innen*

    PWM2FanOutsideValue : in std\_logic\_vector(7 downto 0);  *--! Signalquellwert Luefter aussen*

    PWM3LightValue : in std\_logic\_vector(7 downto 0);     *--! Signalquellwert Licht*

    PWM4PeltierValue : in std\_logic\_vector(7 downto 0);   *--! Signalquellwert Peltier*

    PWM1FanInsideSignal : out std\_logic;          *--! PWM-Aktorsignal Luefter innen*

    PWM2FanOutsideSignal : out std\_logic;           *--! PWM-Aktorsignal Luefter aussen*

    PWM3LightSignal : out std\_logic;            *--! PWM-Aktorsignal Licht*

    PWM4PeltierSignal : out std\_logic);           *--! PWM-Aktorsignal Peltier*

end AshaPWM;

architecture Behavioral of AshaPWM is

signal PWMCounter : unsigned(7 downto 0):= (others => '0');

begin

*-- Die nachfolgenden Zeilen m�ssen nach der Implementierung von*

*-- PWM\_Gen wieder entfernt werden! TODO*

*-- PWM1FanInsideSignal<='1';*

*-- PWM2FanOutsideSignal<='1';*

*-- PWM3LightSignal<='1';*

*-- PWM4PeltierSignal<='1';*

*-- PWMCounter<=(others=>'0');*

*--! PWM Generierung -> Versuch 7*

*-- Hinweis: Die Aktoren sind low-active!*

  PWM\_Count:Process (EnPWMClock, Reset)

    begin

      if (Reset = '1') then

        PWMCounter <= (others => '0');

      elsif rising\_edge(EnPWMClock) then

        if (PWMCounter = 255) then

          PWMCounter <= (others => '0');

        else

          PWMCounter <= PWMCounter + 1;

        end if;

      end if;

    end Process;

  PWM\_Gen:Process (Clock)

  begin

*-- TODO*

*-- Den Innenlüfter einschalten, wenn Counter < als gewünschter PWM-Wert und dieser dabei > 0*

    if (PWMCounter <= unsigned(PWM1FanInsideValue) and unsigned(PWM1FanInsideValue) > 0) then

      PWM1FanInsideSignal<='0';

    else

      PWM1FanInsideSignal<='1';

    end if;

*-- Den Außenlüfter einschalten, wenn Counter < als gewünschter PWM-Wert und dieser dabei > 0*

    if (PWMCounter <= unsigned(PWM2FanOutsideValue) and unsigned(PWM2FanOutsideValue) > 0) then

      PWM2FanOutsideValue<='0';

    else

      PWM2FanOutsideValue<='1';

    end if;

*-- Das Licht einschalten, wenn Counter < als gewünschter PWM-Wert und dieser dabei > 0*

    if (PWMCounter <= unsigned(PWM3LightValue) and unsigned(PWM3LightValue) > 0) then

      PWM3LightValue<='0';

    else

      PWM3LightValue<='1';

    end if;

*-- Den Peltier einschalten, wenn Counter < als gewünschter PWM-Wert und dieser dabei > 0*

    if (PWMCounter <= unsigned(PWM4PeltierValue) and unsigned(PWM4PeltierValue) > 0) then

      PWM4PeltierValue<='0';

    else

      PWM4PeltierValue<='1';

    end if;

  end Process PWM\_Gen;

end Behavioral;

AshaPWMTestbench.vhd

*----------------------------------------------------------------------------*

*-- Company:*

*-- Engineer:*

*--*

*-- Create Date: 12.06.2023 15:38:01*

*-- Design Name:*

*-- Module Name: AshaPWMTestbench - Behavioral*

*-- Project Name:*

*-- Target Devices:*

*-- Tool Versions:*

*-- Description:*

*--*

*-- Dependencies:*

*--*

*-- Revision:*

*-- Revision 0.01 - File Created*

*-- Additional Comments:*

*--*

*----------------------------------------------------------------------------------*

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

*-- Uncomment the following library declaration if using*

*-- arithmetic functions with Signed or Unsigned values*

*--use IEEE.NUMERIC\_STD.ALL;*

*-- Uncomment the following library declaration if instantiating*

*-- any Xilinx leaf cells in this code.*

*--library UNISIM;*

*--use UNISIM.VComponents.all;*

entity AshaPWMTestbench is

*--  Port ( );*

end AshaPWMTestbench;

architecture Behavioral of AshaPWMTestbench is

*-- Komponentbeschreibung*

COMPONENT AshaPWM

PORT(   Clock : IN std\_logic;

        Reset : IN std\_logic;

        EnPWMClock : IN std\_logic;

        PWM1FanInsideValue : IN std\_logic\_vector(7 downto 0);

        PWM2FanOutsideValue : IN std\_logic\_vector(7 downto 0);

        PWM3LightValue : IN std\_logic\_vector(7 downto 0);

        PWM4PeltierValue : IN std\_logic\_vector(7 downto 0);

        PWM1FanInsideSignal : OUT std\_logic;

        PWM2FanOutsideSignal : OUT std\_logic;

        PWM3LightSignal : OUT std\_logic;

        PWM4PeltierSignal : OUT std\_logic );

END COMPONENT;

*-- Eingabesignale*

signal Clock : std\_logic := '0';

signal Reset : std\_logic := '0';

signal EnPWMClock : std\_logic := '0';

signal PWM1FanInsideValue : std\_logic\_vector(7 downto 0) := (others => '0');

signal PWM2FanOutsideValue : std\_logic\_vector(7 downto 0) := (others => '0');

signal PWM3LightValue : std\_logic\_vector(7 downto 0) := (others => '0');

signal PWM4PeltierValue : std\_logic\_vector(7 downto 0) := (others => '0');

*-- Ausgabesignale*

signal PWM1FanInsideSignal : std\_logic;

signal PWM2FanOutsideSignal : std\_logic;

signal PWM3LightSignal : std\_logic;

signal PWM4PeltierSignal : std\_logic;

*-- Clock-Period*

constant Clock\_period : time := 4 ns;

constant EnPWMClock\_period : time := 4 ns;

begin

uut: AshaPWM

PORT MAP (

    Clock => Clock,

    Reset => Reset,

    EnPWMClock => EnPWMClock,

    PWM1FanInsideValue => PWM1FanInsideValue,

    PWM2FanOutsideValue => PWM2FanOutsideValue,

    PWM3LightValue => PWM3LightValue,

    PWM4PeltierValue => PWM4PeltierValue,

    PWM1FanInsideSignal => PWM1FanInsideSignal,

    PWM2FanOutsideSignal => PWM2FanOutsideSignal,

    PWM3LightSignal => PWM3LightSignal,

    PWM4PeltierSignal => PWM4PeltierSignal );

*-- Clock-Signale aktualisieren*

Clock\_Update:process

begin

    Clock <= '0';

    wait for Clock\_period/2;

    Clock <= '1';

    wait for Clock\_period/2;

end process;

EnPWMClock\_process :process

begin

    EnPWMClock <= '0';

    wait for EnPWMClock\_period/2;

    EnPWMClock <= '1';

    wait for EnPWMClock\_period/2;

end process;

stim\_proc: process

begin

*--Werte f�r Aktoren setzen*

    PWM1FanInsideValue <= b"01000000"; *-- 25% an*

    PWM2FanOutsideValue <= b"00000000"; *-- immer aus*

    PWM3LightValue <= b"10000000"; *-- 50% an*

    PWM4PeltierValue <= b"11111111"; *-- immer an*

    wait for 1024 ns;

*--Werte wechseln*

    PWM1FanInsideValue <= b"11111111"; *-- 50% an*

    PWM2FanOutsideValue <= b"01000000"; *-- immer an*

    PWM3LightValue <= b"10000000"; *-- 25% an*

    PWM4PeltierValue <= b"00000000"; *-- immer aus*

    wait;

end process;

response\_control: process

begin

    wait for 100 ns;

end process;

end Behavioral;

**Vorbereitung 7 - die korrigierten Notizen**

**Aufgabe 2**

xxxx

**Aufgabe 4**

xxxx