CODE VDHL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package sample is
    constant NUMBER_OF_BITS : integer := 8;
    constant NUMBER_OF_SAMPLES : integer := 127;
    constant COUNTER_WIDTH : integer := NUMBER_OF_BITS + 2;
    type sampleRegsType is array(0 to NUMBER_OF_SAMPLES - 1) of
unsigned(NUMBER_OF_BITS -1 downto 0);
--     constant defaultSinus : sampleRegsType := ("0000",--  0,
--                                                "0010",--  6  1.568
--
             "0011",-- 12  3.119
--
             "0101",-- 18  4.635
--
             "0110",-- 24  6.1
--
             "0111",-- 30  7.5
--
             "1001",-- 36  8.817
--
             "1010",-- 42  10.0
--
             "1011",-- 48  11.15
--
             "1100",-- 54  12.135
--
             "1101",-- 60  12.99
--
             "1110",-- 66  13.7
--
             "1110",-- 72  14.27
--
             "1111",-- 78  14.67
--
             "1111",-- 84  14.91
--
             "1111");-- 90 15
    constant defaultSinus : sampleRegsType := ("00000000",--  0,
                                               "00000011",-- 3,
                                               "00000110",-- 6,
                                               "00001001",-- 9,
                                               "00001101",--13,
                                               "00010000",--16,
                                               "00010001",--19,
                                               "00010100",--22,
                                               "00010101",--25,
                                               "00011100",--28,
                                               "00011111",--31,
                                               "00100010",--34,
                                               "00100110",--38,
                                               "00101001",--41,
                                               "00101100",--44,
                                               "00101111",--47,
                                               "00110010",--50,
```

```
"00110101",--53,
"00111000",--56,
"00111011",--59,
"00111110",--62,
"01000001",--65,
"01000100",--68,
"01000111",--71,
"01001010",--74,
"01001101",--77,
"01010000",--80,
"01010011",--83,
"01010110",--86,
"01011001",--89,
"01011100",--92,
"01011111",--95,
"01100010",--98,
"01100101",--101,
"01101000",--104,
"01101011",--107,
"01101101",--109,
"01110000",--112,
"01110011",--115,
"01110110",--118,
"01111001",--121,
"01111011",--123,
"01111110",--126,
"10000001",--129,
"10000100",--132,
"10000110",--134,
"10001011",--137,
"10001100",--140,
"10001110",--142,
"10010001",--145,
"10010011",--147,
"10010110",--150,
"10011000",--152,
"10011011",--155,
"10011101",--157,
"10100000",--160,
"10100010",--162,
"10100101",--165,
"10100111",--167,
"10101001",--169,
"10101100",--172,
"10101110",--174,
"10110000",--176,
"10110011",--179,
"10110101",--181,
"10110111",--183,
"10111001",--185,
"10111011",--187,
"10111110",--190,
"11000000",--192,
"11000010",--194,
"11000100",--196,
"11000110",--198,
"11001000",--200,
"11001010",--202,
"11001011",--203,
"11001101",--205,
```

```
                                                    "11001111",--207,
                                                    "11010001",--209,
                                                    "11010011",--211,
                                                    "11010101",--213,
                                                    "11010110",--214,
                                                    "11011000",--216,
                                                    "11011010",--218,
                                                    "11011011",--219,
                                                    "11011101",--221,
                                                    "11011110",--222,
                                                    "11100000",--224,
                                                    "11100001",--225,
                                                    "11100011",--227,
                                                    "11100100",--228,
                                                    "11100110",--230,
                                                    "11100111",--231,
                                                    "11101000",--232,
                                                    "11101010",--234,
                                                    "11101011",--235,
                                                    "11101100",--236,
                                                    "11101101",--237,
                                                    "11101110",--238,
                                                    "11101111",--239,
                                                    "11110001",--241,
                                                    "11110010",--242,
                                                    "11110011",--243,
                                                    "11110011",--243,
                                                    "11110100",--244,
                                                    "11110101",--245,
                                                    "11110110",--246,
                                                    "11110111",--247,
                                                    "11111000",--248,
                                                    "11111000",--248,
                                                    "11111001",--249,
                                                    "11111010",--250,
                                                    "11111010",--250,
                                                    "11111011",--251,
                                                    "11111011",--251,
                                                    "11111100",--252,
                                                    "11111100",--252,
                                                    "11111101",--253,
                                                    "11111101",--253,
                                                    "11111110",--254,
                                                    "11111110",--254,
                                                    "11111110",--254,
                                                    "11111110",--254,
                                                    "11111111",--255,
                                                    "11111111",--255,
                                                    "11111111",--255,
                                                    "11111111");--255,
end package;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.sample.all;
```

```
entity counter_generator is
    port(clock : in std_ulogic;
            reset : in std_ulogic;
                 increment : in unsigned(COUNTER_WIDTH downto 0);
                 offset : in unsigned(COUNTER_WIDTH downto 0);
                 pause : in unsigned(COUNTER_WIDTH downto 0);
                 interval_out : out std_ulogic_vector(1 downto 0);

                 counter_out : out unsigned(COUNTER_WIDTH downto 0));
end counter_generator;

architecture Behavioral of counter_generator is
    signal counter : unsigned(COUNTER_WIDTH downto 0);
    signal counter_pause : unsigned(COUNTER_WIDTH downto 0);
    signal interval : std_ulogic_vector(1 downto 0);
begin
    sinPWM_process : process(clock, reset)
      begin
          if (reset = '1') then
                counter_pause <= to_unsigned(0, COUNTER_WIDTH + 1); --
geaendert von COUNTER_WIDT nach COUNTER_WIDTH + 1 14.01.13
                if (offset < to_unsigned(NUMBER_OF_SAMPLES - 1,
COUNTER_WIDTH)) then
                           counter <= offset;
                           interval <= "00";
                      else
                     if (offset <= 2 * to_unsigned(NUMBER_OF_SAMPLES - 1,
COUNTER_WIDTH)) then
                                counter <= offset -
to_unsigned(NUMBER_OF_SAMPLES - 1, COUNTER_WIDTH);
                                interval <= "01";
                             else
                          if (offset <= 3 * to_unsigned(NUMBER_OF_SAMPLES
- 1, COUNTER_WIDTH)) then
                                      counter <= offset - 2 *
(NUMBER_OF_SAMPLES - 1);
                                --     counter <= offset - 2 *
to_unsigned(NUMBER_OF_SAMPLES - 1, COUNTER_WIDTH);
                                      -- geaendert am 14.01.13
                                   interval <= "10";
                                else
                                      counter <= offset - 3 *
(NUMBER_OF_SAMPLES - 1);
                                --    counter <= offset - 3 *
to_unsigned(NUMBER_OF_SAMPLES - 1, COUNTER_WIDTH);
                                      -- geaendert am 14.01.13
                                   interval <= "11";
                                    end if;
                             end if;
                      end if;
             else
            if (rising_edge(clock)) then
                           if (counter < (to_unsigned(NUMBER_OF_SAMPLES -
1, COUNTER_WIDTH) - increment)) then
                               if ((counter_pause < pause) or (pause =
to_unsigned(0, COUNTER_WIDTH))) then
                                    counter <= counter + increment + 1;
                                           counter_pause <=
counter_pause + 1;
```

```
                                    else
                                        counter_pause <= to_unsigned(0,
COUNTER_WIDTH + 1); -- geaendert von COUNTER_WIDT nach COUNTER_WIDTH + 1
14.01.13
                                    end if;
                                else
                                    counter <= counter + increment + 1 -
to_unsigned(NUMBER_OF_SAMPLES - 1, COUNTER_WIDTH);
                                    interval <=
std_ulogic_vector(unsigned(interval) + 1);
                                end if;
                            end if;
                end if;
        end process;
        counter_out <= counter;
        interval_out <= interval;
end Behavioral;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.sample.all;

entity sin_PWM_generator is
    port(clock : in std_ulogic;
             reset : in std_ulogic;
                  interval : in std_ulogic;
                  counter_in : in unsigned(COUNTER_WIDTH downto 0);
                  counter_out : out unsigned(COUNTER_WIDTH downto 0));
end sin_PWM_generator;

architecture Behavioral of sin_PWM_generator is
begin
    sinPWM_process : process(clock, reset)
      begin
          if (reset = '1') then
                        counter_out <= to_unsigned(0, COUNTER_WIDTH + 1);
-- geaendert von COUNTER_WIDT nach COUNTER_WIDTH + 1 14.01.13
              else
             if (rising_edge(clock)) then
                            if (interval = '0') then
                                counter_out <= counter_in;
                            else
                              counter_out <=
TO_UNSIGNED(NUMBER_OF_SAMPLES - 1, COUNTER_WIDTH) - counter_in;
                  end if;
                        end if;
              end if;
        end process;
end Behavioral;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use ieee.std_logic_unsigned.all;
use work.sample.all;

entity PWM_generator is
    port(clock : in std_ulogic;
            reset : in std_ulogic;
                 sync : in std_ulogic;
               direction : in std_ulogic;
                 enable : in std_ulogic;
                 halfBridgeMode : std_ulogic;
                 halfPeriod : std_ulogic;
         finalValue : in unsigned(NUMBER_OF_BITS - 1 downto 0);
                 PWM_signal : out std_ulogic;
                 PWM_signal_alt : out std_ulogic);
end PWM_generator;

architecture Behavioral of PWM_generator is
    signal PWM_cycle_counter : unsigned(COUNTER_WIDTH - 1 downto 0);
      signal finalValue_old : unsigned(NUMBER_OF_BITS - 1 downto 0);
      signal direction_old : std_ulogic;
      signal halfPeriod_old : std_ulogic;
      signal pwm_intern : std_ulogic;
begin
    PWM_process : process(clock, reset)
      begin
          if (reset = '1') then
                PWM_cycle_counter <= to_unsigned(1, COUNTER_WIDTH);
                     PWM_signal <= '0';
                     PWM_signal_alt <= '0';
                     finalValue_old <= x"00";
                     direction_old <= '0';
                     halfPeriod_old <= '0';
            else
                if (rising_edge(clock)) then
                          if (direction_old = '0') then
                                if (sync = '0') then
                                  if (PWM_cycle_counter <
finalValue_old) then
                                                PWM_cycle_counter <=
PWM_cycle_counter + 1;
                                          PWM_signal <= enable and not
halfPeriod_old;
                                          PWM_signal_alt <= enable and
halfPeriod_old;
--                              PWM_signal <= '1';
--                              PWM_Signal_alt <= '1';
                                           else
--                              PWM_signal <= '0';
--                              PWM_Signal_alt <= '0';
                                                PWM_signal <= enable and
(halfPeriod_old and not halfBridgeMode);
                                                PWM_signal_alt <= enable and
(not halfPeriod_old and not halfBridgeMode);
                                          end if;
                                   else
                                       if (finalValue = 0) then
--                              PWM_signal <= '0';
```

```
--                              PWM_Signal_alt <= '0';
                                                PWM_signal <= enable and
(halfPeriod and not halfBridgeMode);
                                                PWM_signal_alt <= enable and
(not halfPeriod and not halfBridgeMode);
                                            else
--                              PWM_signal <= '1';
--                              PWM_Signal_alt <= '1';
                                                  PWM_signal <= enable and
not halfPeriod;
                                                  PWM_signal_alt <= enable
and halfPeriod;
                                            end if;
                                            finalValue_old <= finalValue;
                                            direction_old <= direction;
                                                 halfPeriod_old <=
halfPeriod;
                                        PWM_cycle_counter <= to_unsigned(1,
COUNTER_WIDTH);
                                    end if;
                                else
                                    if (sync = '0') then
                                     if (PWM_cycle_counter < 255 -
finalValue_old) then
                                                PWM_signal <= enable and
(halfPeriod_old and not halfBridgeMode);
                                                PWM_signal_alt <= enable and
(not halfPeriod_old and not halfBridgeMode);
--                              PWM_signal <= '0';
--                              PWM_Signal_alt <= '0';
                                                PWM_cycle_counter <=
PWM_cycle_counter + 1;
                                             else
                                            PWM_signal <= enable and not
halfPeriod_old;
                                            PWM_signal_alt <= enable and
halfPeriod_old;
--                              PWM_signal <= '1';
--                              PWM_Signal_alt <= '1';
                                             end if;
                                   else
                                       if (finalValue = 255) then
--                              PWM_signal <= '1';
--                              PWM_Signal_alt <= '1';
                                                  PWM_signal <= enable and
not halfPeriod;
                                                  PWM_signal_alt <= enable
and halfPeriod;
                                            else
--                              PWM_signal <= '0';
--                              PWM_Signal_alt <= '0';
                                                PWM_signal <= enable and
(halfPeriod and not halfBridgeMode);
                                                PWM_signal_alt <= enable and
(not halfPeriod and not halfBridgeMode);
                                            end if;
                                            finalValue_old <= finalValue;
                                            direction_old <= direction;
                                                 halfPeriod_old <=
halfPeriod;
```

```
                                        PWM_cycle_counter <= to_unsigned(1,
COUNTER_WIDTH);
                                    end if;
                                end if;
                            end if;
                end if;
        end process;
end Behavioral;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity clock_divider is
    port(clock : in std_ulogic;
            reset : in std_ulogic;
                 divider : in unsigned(15 downto 0);
                 clock_out : out std_ulogic);
end clock_divider;

architecture Behavioral of clock_divider is
    signal divider_counter : unsigned(15 downto 0);
begin
    CLK_process : process(clock, reset)
       begin
           if (reset = '1') then
                  divider_counter <= x"0000";
                       clock_out <= '0';
              else
                  if (divider > 0) then
                      if (rising_edge(clock)) then
                                    if (divider_counter < divider) then
                                        divider_counter <=
divider_counter + 1;
                                             clock_out <= '0';
                                    else
                                        divider_counter <= x"0000";
                                             clock_out <= '1';
                                    end if;
                            end if;
                        else
                            clock_out <= clock;
                        end if;
              end if;
        end process;
end Behavioral;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
--use ieee.std_logic_unsigned.all;
use work.global.all;
use work.sample.all;

entity Modulator is
   port(clock : in std_ulogic;
           reset : in std_ulogic;
           scl : in std_ulogic;
           sda : inout std_ulogic;
              phaseLFullbridge : out std_ulogic_vector(2 downto 0);
              phaseLHalfbridge : out std_ulogic_vector(2 downto 0);
              indicator : out std_ulogic_vector(5 downto 0);
              sync_out : out std_ulogic;
              testbit2 : out std_ulogic);
--        index2 : out std_ulogic_vector(10 downto 0);
--        index1 : out std_ulogic_vector(10 downto 0));
end Modulator;

architecture Behavioral of Modulator is
    -- declarations for the I2C interface
    component I2C is
        port(clock : in std_ulogic;
                reset : in std_ulogic;
                scl : in std_ulogic;
                sda : inout std_ulogic;
                     readRegister : out std_ulogic;
                     writeRegister : out std_ulogic;
                     registerIn : in std_ulogic_vector(15 downto 0);
                     registerOut : out std_ulogic_vector(15 downto 0);
                        address : out std_ulogic_vector(7 downto 0));
      end component;
      signal appRegs : appRegsType;
      signal readRegister : std_ulogic;
      signal writeRegister : std_ulogic;
      signal registerIn : std_ulogic_vector(15 downto 0);
      signal registerOut : std_ulogic_vector(15 downto 0);
      signal address : std_ulogic_vector(7 downto 0);
      signal dummy : std_ulogic;

      -- declarations for everything else
    component counter_generator is
        port(clock : in std_ulogic;
                reset : in std_ulogic;
                     increment : in unsigned(COUNTER_WIDTH downto 0);
                     offset : in unsigned(COUNTER_WIDTH downto 0);
                     pause : in unsigned(COUNTER_WIDTH downto 0);
                     interval_out : out std_ulogic_vector(1 downto 0);
                   counter_out : out unsigned(COUNTER_WIDTH downto 0));
    end component;
    component clock_divider is
        port(clock : in std_ulogic;
                reset : in std_ulogic;
                     divider : in unsigned(15 downto 0);
                     clock_out : out std_ulogic);
    end component;
    component sin_PWM_generator is
        port(clock : in std_ulogic;
```

```
                reset : in std_ulogic;
                     interval : in std_ulogic;
                     counter_in : in unsigned(COUNTER_WIDTH downto 0);
                     counter_out : out unsigned(COUNTER_WIDTH downto 0));
    end component;
    component PWM_generator is
        port(clock : in std_ulogic;
                reset : in std_ulogic;
                       sync : in std_ulogic;
                       direction : in std_ulogic;
                       enable : in std_ulogic;
                       halfBridgeMode : std_ulogic;
                     halfPeriod : std_ulogic;
              finalValue : in unsigned(NUMBER_OF_BITS - 1 downto 0);
                     PWM_signal : out std_ulogic;
                       PWM_signal_alt : out std_ulogic);
    end component;
       signal clock_out : std_ulogic;
       signal sync : std_ulogic;
       signal PWMsign : std_ulogic;
       signal sinIndex_L1 : unsigned(COUNTER_WIDTH downto 0);
       signal PWM_out_L1 : std_ulogic;
       signal PWM_out_L1_alt : std_ulogic;
       signal sinIndex_L2 : unsigned(COUNTER_WIDTH downto 0);
       signal PWM_out_L2 : std_ulogic;
       signal PWM_out_L2_alt : std_ulogic;
       signal sinIndex_L3 : unsigned(COUNTER_WIDTH downto 0);
       signal PWM_out_L3 : std_ulogic;
       signal PWM_out_L3_alt : std_ulogic;
       signal counterL1 : unsigned(COUNTER_WIDTH downto 0);
       signal intervalL1 : std_ulogic_vector(1 downto 0);
       signal counterL2 : unsigned(COUNTER_WIDTH downto 0);
       signal intervalL2 : std_ulogic_vector(1 downto 0);
       signal counterL3 : unsigned(COUNTER_WIDTH downto 0);
       signal intervalL3 : std_ulogic_vector(1 downto 0);
       signal reset1 : std_ulogic;
       signal phaseL1 : std_ulogic;
       signal phaseL2 : std_ulogic;
       signal phaseL3 : std_ulogic;
       signal PWM_width_L1 : unsigned(NUMBER_OF_BITS - 1 downto 0);
       signal PWM_width_L2 : unsigned(NUMBER_OF_BITS - 1 downto 0);
       signal PWM_width_L3 : unsigned(NUMBER_OF_BITS - 1 downto 0);
begin
    -- I2C interface proecesses
    ModulI2C : I2C port map(clock, reset, scl, sda, readRegister,
writeRegister, registerIn, registerOut, address);
    TransferProcess : process(clock, reset, readRegister, writeRegister,
registerIn, registerOut, address)
    begin
        if (reset = '1') then
              appRegs(0) <= x"0000";
                     -- register 1 bit 2,1,0 enable phases 3,2,1
                appRegs(1) <= "0000011100000111";
                     -- register 1 bit 5,4,3 enable halfbridge mode
phases 3,2,1
                     -- register 2 increment
                     appRegs(2) <= x"0029";
                     -- register 3 pause
                appRegs(3) <= x"0000";
                     -- register 4 multiplier factor
```

```
                        appRegs(4) <= x"0100";
                        -- register 5 clock divider
                        appRegs(5) <= x"0001";
                   appRegs(6) <= "0000000101010000";
                -- register 6 phase shift of phase 1
                        appRegs(7) <= x"0100";
                        -- register 7 multipiler factor phase 1
                   appRegs(8) <= "0000000010101000";
                        appRegs(9) <= x"0100";
                        appRegs(10) <= "0000000000000000";
                        appRegs(11) <= x"0100";
                        for I in FIRST_REGISTER to LAST_REGISTER loop
                        --    appRegs(I) <= x"0000";
                        end loop;
           else
                     if (rising_edge(clock) and (writeRegister = '1')) then
                           appRegs(to_integer(unsigned(address))) <=
registerOut;
                        end if;
                     if (rising_edge(clock) and (readRegister = '0')) then
                              registerIn <=
appRegs(to_integer(unsigned(appRegs(0))));
                        end if;
                end if;
          end process;

          -- other processes
          reset1 <= reset or appRegs(4)(0);
          ModulCLK : clock_divider port map(clock, reset,
unsigned(appRegs(5)), clock_out);
          ModulSYNC : clock_divider port map(clock_out, reset,
to_unsigned((to_integer(defaultSinus(NUMBER_OF_SAMPLES - 1)) - 1), 16),
sync);
          ModulCNT_L1 : counter_generator port map(sync, reset1,
unsigned(appRegs(2)(COUNTER_WIDTH downto 0)),
unsigned(appRegs(6)(COUNTER_WIDTH downto 0)),
unsigned(appRegs(3)(COUNTER_WIDTH downto 0)), intervalL1, counterL1);
          ModulCNT_L1 : counter_generator port map(sync, reset1,
unsigned(appRegs(2)(COUNTER_WIDTH downto 0)), "00101010000",
unsigned(appRegs(3)(COUNTER_WIDTH downto 0)), intervalL1, counterL1);
--        ModulSIN_L1 : sin_PWM_generator port map(sync, reset1,
intervalL1(0), counterL1, sinIndex_L1);
        PWM_width_L1 <= resize(unsigned(appRegs(7)) *
defaultSinus(to_integer(sinIndex_L1)), NUMBER_OF_BITS * 2)(NUMBER_OF_BITS
* 2 -1  downto NUMBER_OF_BITS) when reset = '0' else x"00";
          ModulPWM_L1 : PWM_generator port map (clock_out, reset1, sync,
intervalL1(0), appRegs(1)(0), appRegs(1)(8), intervalL1(1), PWM_width_L1,
PWM_out_L1, PWM_out_L1_alt);
          ModulCNT_L2 : counter_generator port map(sync, reset1,
unsigned(appRegs(2)(COUNTER_WIDTH downto 0)),
unsigned(appRegs(8)(COUNTER_WIDTH downto 0)),
unsigned(appRegs(3)(COUNTER_WIDTH downto 0)), intervalL2, counterL2);
          ModulCNT_L2 : counter_generator port map(sync, reset1,
unsigned(appRegs(2)(COUNTER_WIDTH downto 0)), "00010101000",
unsigned(appRegs(3)(COUNTER_WIDTH downto 0)), intervalL2, counterL2);
--        ModulSIN_L2 : sin_PWM_generator port map(sync, reset1,
intervalL2(0), counterL2, sinIndex_L2);
        PWM_width_L2 <= resize(unsigned(appRegs(9)) *
defaultSinus(to_integer(sinIndex_L2)), NUMBER_OF_BITS * 2)(NUMBER_OF_BITS
* 2 -1  downto NUMBER_OF_BITS) when reset = '0' else x"00";
```

```
       ModulPWM_L2 : PWM_generator port map (clock_out, reset1, sync,
intervalL2(0), appRegs(1)(1), appRegs(1)(9), intervalL2(1), PWM_width_L2,
PWM_out_L2, PWM_out_L2_alt);
       ModulCNT_L3 : counter_generator port map(sync, reset1,
unsigned(appRegs(2)(COUNTER_WIDTH downto 0)),
unsigned(appRegs(10)(COUNTER_WIDTH downto 0)),
unsigned(appRegs(3)(COUNTER_WIDTH downto 0)), intervalL3, counterL3);
       ModulCNT_L3 : counter_generator port map(sync, reset1,
unsigned(appRegs(2)(COUNTER_WIDTH downto 0)), "00000000000",
unsigned(appRegs(3)(COUNTER_WIDTH downto 0)), intervalL3, counterL3);
--     ModulSIN_L3 : sin_PWM_generator port map(sync, reset1,
intervalL3(0), counterL3, sinIndex_L3);
    PWM_width_L3 <= resize(unsigned(appRegs(11)) *
defaultSinus(to_integer(sinIndex_L3)), NUMBER_OF_BITS * 2)(NUMBER_OF_BITS
* 2 -1  downto NUMBER_OF_BITS) when reset = '0' else x"00";
       ModulPWM_L3 : PWM_generator port map (clock_out, reset1, sync,
intervalL3(0), appRegs(1)(2), appRegs(1)(10), intervalL3(1),
PWM_width_L3, PWM_out_L3, PWM_out_L3_alt);

    phaseLFullbridge(0) <= PWM_out_L1;
       phaseLFullbridge(1) <= PWM_out_L2;
       phaseLFullbridge(2) <= PWM_out_L3;
       phaseLHalfbridge(0) <= PWM_out_L1_alt;
       phaseLHalfbridge(1) <= PWM_out_L2_alt;
       phaseLHalfbridge(2) <= PWM_out_L3_alt;
       indicator(0) <= PWM_out_L1;
       indicator(1) <= PWM_out_L2;
       indicator(2) <= PWM_out_L3;
       indicator(3) <= PWM_out_L1_alt;
       indicator(4) <= PWM_out_L2_alt;
       indicator(5) <= PWM_out_L3_alt;

       testbit2 <= '1' when reset = '1' else '0';

    sync_out <= sync;
--     index1 <= STD_ULOGIC_VECTOR(sinIndex_L1);
--     index2 <= "000" & STD_ULOGIC_VECTOR(PWM_width_L1);
--     index2 <= STD_ULOGIC_VECTOR(counterL1);
end Behavioral;
```

TESTBENCH

```
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;
USE ieee.std_logic_unsigned.ALL;
USE ieee.math_real.ALL;


ENTITY I2C_modulator_tb IS
END I2C_modulator_tb;

ARCHITECTURE behavior OF I2C_modulator_tb IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT Modulator
    PORT(
                 clock : IN  std_logic;
         reset : IN  std_logic;
         scl : IN  std_logic;
         sda : INOUT  std_logic;
              phaseLFullbridge : OUT std_ulogic_vector(2 downto 0);
              phaseLHalfbridge : OUT std_ulogic_vector(2 downto 0);
              indicator : out std_ulogic_vector(5 downto 0);
              sync_out : out std_logic;
         testbit2 : OUT  std_logic
--         index2 : OUT  std_logic_vector(10 downto 0);
--         index1 : OUT  std_logic_vector(10 downto 0)


           );
    END COMPONENT;


   --Inputs
   signal clock : std_logic := '0';
   signal reset : std_logic := '0';
   signal scl : std_logic := '0';

      --BiDirs
   signal sda : std_logic;

      --Outputs
      signal sync_out : std_logic;
   signal testbit2 : std_logic;
   signal phaseLFullbridge : std_ulogic_vector(2 downto 0);
   signal phaseLHalfbridge : std_ulogic_vector(2 downto 0);
      signal indicator : std_ulogic_vector(5 downto 0);

--   signal index2 : std_logic_vector(10 downto 0);
--   signal index1 : std_logic_vector(10 downto 0);

   -- Clock period definitions
   constant clock_period : time := 10 ns;

BEGIN
```

```
      -- Instantiate the Unit Under Test (UUT)
   uut: Modulator PORT MAP (
          clock => clock,
          reset => reset,
          scl => scl,
          sda => sda,
                 phaseLFullbridge => phaseLFullbridge,
                 phaseLHalfbridge => phaseLHalfbridge,
          testbit2 => testbit2,
--               index2 => index2,
--               index1 => index1,
          sync_out => sync_out,
                 indicator => indicator
        );

   -- Clock process definitions
   clock_process :process
   begin
           clock <= '0';
           wait for clock_period/2;
           clock <= '1';
           wait for clock_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
           reset <= '1';
           sda <= 'H';
           scl <= '0';
           wait for 100ns;
           reset <= '0';

--         sda <= '1';
wait for clock_period * 1000;
      wait for 100 ns;

      wait for clock_period*10;

   -- insert stimulus here

      wait;
   end process;




END;
```