Oblig1:

FIRST.VHD

```
--This is a comment,which end here
--this is a new comment
library IEEE; --makes the library IEEE visible
use IEEE.std_logic_1164.all; --makes use of the package std_logic_1164
use IEEE.numeric_std.all;    --makes use of the package numeric_std

--The entity defines all the signals in and out (inout) of the circuit
--we want to describe
--Please note: VHDL is NOT case sensitive
entity FIRST is
  port
  (
  --Signalname : <mode> <data_type>; mode: in, out or inout
    UP          : in  std_logic;
    CLK         : in  std_logic; -- Clock from switch CLK1/INP1
    RESET       : in  std_logic; -- Global Asynchronous Reset
    LOAD        : in  std_logic; -- Synchronous Reset
    INP         : in  std_logic_vector(3 downto 0); -- Start Value
    COUNT       : out std_logic_vector(3 downto 0); -- Counting value
    MAX_COUNT  : out std_logic;  -- Max counting value
    MIN_COUNT  : out std_logic
  );
end FIRST;

--The architeture describes the functionality of the entity it belongs to.
--An entity can have many architetures, whereas an architecture can only belong to one
--entity.
--The architecture below describes a 4-bits up-counter, with an asynchronous reset
--and a synchronous reset (LOAD). When the counter reaches it maximum value the signal
--MAX_COUNT goes active

--architecture <architectureName> of <Belonging entity> is
architecture MY_FIRST_ARCH of FIRST is

  --Area for declarations internal to the architecture
  --for example internal signals

  signal COUNT_I : unsigned(3 downto 0);
  --We will use the signed and/or unsigned data types for arithmetric operations
  --Examples: Arithmetric operators: +,-,* and / Comparations:>,>=,>,<, <=, = and /=
begin
```

```
--Here starts the description

COUNTER: --This is a label which will be visible inside the simulator
process (RESET,CLK) --sensitivity list, include all input signals to the process
                    --which matters for the functionality of the process
begin
  --Due to the inherent priority in an if-statement the RESET-signal will have
  --priority above the CLK if both statement evaluates to TRUE.
  --That is exactly how an asynchronous reset should function.
  if(RESET  = '1') then
    COUNT_I <= "0000";
    --More general statement, which s independant of number of bits
    --COUNT_I <= (others => '0');
  elsif rising_edge(CLK) then --rising_edge(falling_edge) are functions defined in
                              --the std_logic_1164 package
  --elsif (CLK'event and CLK = '1') then
    -- Synchronous reset
    if LOAD = '1' then
      COUNT_I <= unsigned(INP); --Type casting from std_logic_vector to unsigned
    elsif UP = '1' then
      COUNT_I <= COUNT_I + 1;
    else
      COUNT_I <= COUNT_I - 1;
    end if;
  end if;
end process COUNTER;

--Concurrent signal assignment (CSA)
COUNT <= std_logic_vector(COUNT_I);--Type casting from unsigned to std_logic_vector

--Concurrent signal assignment (CSA)
--if UP = '0' then
  MAX_COUNT <= '1' when COUNT_I = "1111" and UP = '1' else '0';

--else
  MIN_COUNT <= '1' when COUNT_I = "0000" and UP = '0' else '0';

--end if;
--This is 100% equivalent to the process below:
--Generally a process is easier to read (and understand) if the code is indented
--properly, but for small statements like the one above it is ok to use a CSA.

--process (COUNT_I)
```

```
  --begin
  --  if COUNT_I = "1111" then
  --      MAX_COUNT <= '1';
  --  else
  --      MAX_COUNT <= '0';
  --  end if;
  --end process;

end MY_FIRST_ARCH;
```

TB_FIRST.VHD

```
         library IEEE;
use IEEE.Std_Logic_1164.all;

entity TEST_FIRST is
  -- The entity for a testbench is normally empty
end TEST_FIRST;

architecture TESTBENCH of TEST_FIRST is

  -- Component declarations
  Component FIRST
    port
    (
      CLK        : in  std_logic; -- Clock from switch CLK1/INP1
      RESET      : in  std_logic; -- Global Asynchronous Reset
      LOAD       : in  std_logic; -- Synchronous Reset
      INP        : in  std_logic_vector(3 downto 0); -- Start Value
      COUNT      : out std_logic_vector(3 downto 0); -- Counting value
      MAX_COUNT  : out std_logic  -- Max counting value
    );
  end Component;

  --testbench internal signals
  --which should be used to connect with the component first
  --input to UUT should be given initial values
  signal  MCLK       : std_logic := '0'; --:= initial value
  signal  RESET      : std_logic := '0';
  signal  LOAD       : std_logic := '0';
  signal  INP        : std_logic_vector(3 downto 0) := "0000";
  signal  COUNT      : std_logic_vector(3 downto 0);
  signal  MAX_COUNT  : std_logic;
  constant Half_Period : time := 10 ns;  --50Mhz klokkefrekvens

begin

  --Instantiates "Unit Under Test", UUT
  UUT : FIRST
  port map
  (
  --<formal name> => <actual name>
    CLK        =>  MCLK,
    RESET      =>  RESET,
    LOAD       =>  LOAD,
    INP        =>  INP,
    COUNT      =>  COUNT,
    MAX_COUNT  =>  MAX_COUNT
  );

  -- Defines the clock
  MCLK <= not MCLK after Half_Period;

  -- The input stimuli to UUT
  STIMULI :
  process
  --a process with an empty sensitivity list should include wait statements
  begin
    RESET <= '1', '0' after 100 ns;
    INP <= "1010" after Half_Period*6;
    wait for 2*Half_Period*10;
    LOAD <= '1', '0' after 2*Half_Period;
    wait;
  end process;

end TESTBENCH;
```

TB_EF.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_ef is
--  Port ( );
end tb_ef;

architecture Behavioral of tb_ef is
```

```
component ef is
--  Port ( );
    port
    (
  --Signalname : <mode> <data_type>; mode: in, out or inout
    SW      : in  std_logic_vector(1 downto 0);
    --SW2      : in  std_logic; -- Clock from switch CLK1/INP1
    LD      : out std_logic_vector(3 downto 0) -- Global Asynchronous Reset

    );
end component;

  signal swit   : std_logic_vector(1 downto 0);

begin


  test: ef port map (swit, open);

  switch: process
  begin
    swit <= "00";
    wait for 100 ns;
    swit <= "01";
    wait for 100 ns;
            swit <= "10";
    wait for 100 ns;
    swit <= "11";
    wait for 100 ns;
  end process switch;



end Behavioral;
```

EF.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ef is
--  Port ( );
```

```
    port
    (
  --Signalname : <mode> <data_type>; mode: in, out or inout
    SW      : in  std_logic_vector(1 downto 0);
    --SW2      : in  std_logic; -- Clock from switch CLK1/INP1
    LD      : out std_logic_vector(3 downto 0) -- Global Asynchronous Reset

    );
end ef;

architecture Behavioral of ef is

begin

    EXAMPLE:
    process (SW)

    begin
     case SW is
        when "00" =>
            LD <= "1110";
        when "01" =>
            LD <= "1101";
        when "10" =>
            LD <= "1011";
        when others =>
            LD <= "0111";
      end case;

     end process EXAMPLE;
end Behavioral;
```

DFF.VHD

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

entity dff is
  port (
    -- System Clock and Reset
    rst_n     : in  std_logic;   -- Reset
    mclk      : in  std_logic;   -- Clock
    -- Shifted data in and out
    din       : in  std_logic;   -- Data in
    dout      : out std_logic    -- Data out
  );
end dff;

architecture rtl of dff is

begin
  P_DFF : process(rst_n, mclk)
  begin
    if rst_n='0' then
      dout <= '0';
    elsif rising_edge(mclk) then
      dout <= din;
    end if;
  end process P_DFF;
end rtl;
```

TB_SHIFT.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity tb_shift is
--  Port ( );
end tb_shift;

architecture Behavioral of tb_shift is

  Component shift8
    port
    (
        rst_n     : in  std_logic;   -- Reset
        mclk      : in  std_logic;   -- Clock
        -- Shifted data in and out
        din       : in  std_logic;   -- Data in
        dout      : out std_logic    -- Data out
    );
  end Component;
  Component shift32
      port
      (
        rst_n     : in  std_logic;   -- Reset
        mclk      : in  std_logic;   -- Clock
        -- Shifted data in and out
        din       : in  std_logic;   -- Data in
        dout      : out std_logic    -- Data out
      );
    end Component;
    Component shiftn
      port
      (
        rst_n     : in  std_logic;   -- Reset
        mclk      : in  std_logic;   -- Clock
        -- Shifted data in and out
        din       : in  std_logic;   -- Data in
        dout      : out std_logic    -- Data out
      );
    end Component;

  signal rst_n   : std_logic;
  signal mclk    : std_logic;
  signal din     : std_logic;
  signal dout    : std_logic;

  signal restulat : std_logic_vector(2 downto 0);

begin

  shift_8: shift8 port map (rst_n, mclk, din, restulat(0));
  shift_32: shift32 port map (rst_n, mclk, din, restulat(1));

  shift_n: shiftn
          generic map (N => 64)
          port map (rst_n, mclk, din, restulat(2));

  P_CLK_0: process
  begin
    mclk <= '0';
    wait for 50 ns;
    mclk <= '1';
    wait for 50 ns;
  end process P_CLK_0;

  rst_n  <= '0','1' after 100 ns;
  din <= '1', '0' after 150ns,'1' after 300ns;

  dout <= (restulat(0) and restulat(1) and restulat(2));
end Behavioral;
```

SHIFTN.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity shiftn is
  generic (N: integer := 64);
  port (
    rst_n     : in  std_logic;   -- Reset
    mclk      : in  std_logic;   -- Clock
    din       : in  std_logic;
    dout      : out std_logic
    );
end shiftn;

architecture Behavioral of shiftn is


Component dff
    port
    (
        rst_n     : in  std_logic;   -- Reset
        mclk      : in  std_logic;   -- Clock
        -- Shifted data in and out
        din       : in  std_logic;   -- Data in
        dout      : out std_logic    -- Data out
    );
  end Component;

  signal SEL: std_logic_vector(N-1 downto 0);

begin

  F: for I in 0 to N-1 generate

    FIRSTBIT: if I=0 generate
      G1: dff port map (rst_n, mclk, din, SEL(0));
    end generate FIRSTBIT;
    MID: if I>0 generate
          G2: dff port map (rst_n, mclk, SEL(I-1), SEL(I));
    end generate MID;


  end generate F;


  dout <= SEL(N-1);

end Behavioral;
```

SHIFT32.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity shift32 is
  port (
    rst_n     : in  std_logic;   -- Reset
    mclk      : in  std_logic;   -- Clock
    din       : in  std_logic;
    dout      : out std_logic
    );
end shift32;

architecture Behavioral of shift32 is


  Component shift8
    port
    (
        rst_n     : in  std_logic;   -- Reset
        mclk      : in  std_logic;   -- Clock
        -- Shifted data in and out
        din       : in  std_logic;   -- Data in
        dout      : out std_logic    -- Data out
    );
  end Component;


  --signal  RST      : std_logic := '0'; --:= initial value
  --signal  CLK      : std_logic := '0';
  --signal  DATAIN   : std_logic := '0';
  --signal  DATAOUT  : std_logic;


  signal SEL: std_logic_vector(2 downto 0);
    --signal SEL: std_logic_vector(n-1 downto 0);


begin


    G1: shift8 port map (rst_n, mclk, din, SEL(0));
    G2: shift8 port map (rst_n, mclk, SEL(0), SEL(1));
    G3: shift8 port map (rst_n, mclk, SEL(1), SEL(2));
    G4: shift8 port map (rst_n, mclk, SEL(2), dout);

   -- G3: dff port map (rst => rst_n, clk => mclk, datain => SEL(0), dataout => SEL(1));


   -- G1: Df port map (rst => rst_n, clk => mclk, datain => din, dataout => SEL(0));

 --dout <= SEL(1);
 --B1: Df port map (SEL(0), din);
 --G2: Df port map (dout, SEL(1));
 --G3: Df port map (dout, SEL(2));


end Behavioral;
```

SHIFT8.VHD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity shift8 is
  port (
    rst_n     : in  std_logic;   -- Reset
    mclk      : in  std_logic;   -- Clock
    din       : in  std_logic;
    dout      : out std_logic
    );
```

```
end shift8;

architecture Behavioral of shift8 is


  Component dff
    port
    (
        rst_n     : in  std_logic;   -- Reset
        mclk      : in  std_logic;   -- Clock
        -- Shifted data in and out
        din       : in  std_logic;   -- Data in
        dout      : out std_logic    -- Data out
    );
  end Component;


  --signal  RST      : std_logic := '0'; --:= initial value
  --signal  CLK      : std_logic := '0';
  --signal  DATAIN   : std_logic := '0';
  --signal  DATAOUT  : std_logic;


  signal SEL: std_logic_vector(6 downto 0);
    --signal SEL: std_logic_vector(n-1 downto 0);


begin


    G1: dff port map (rst_n, mclk, din, SEL(0));
    G2: dff port map (rst_n, mclk, SEL(0), SEL(1));
    G3: dff port map (rst_n, mclk, SEL(1), SEL(2));
    G4: dff port map (rst_n, mclk, SEL(2), SEL(3));
    G5: dff port map (rst_n, mclk, SEL(3), SEL(4));
    G6: dff port map (rst_n, mclk, SEL(4), SEL(5));
    G7: dff port map (rst_n, mclk, SEL(5), SEL(6));
    G8: dff port map (rst_n, mclk, SEL(6), dout);

   -- G3: dff port map (rst => rst_n, clk => mclk, datain => SEL(0), dataout => SEL(1));


   -- G1: Df port map (rst => rst_n, clk => mclk, datain => din, dataout => SEL(0));
```

```
 --dout <= SEL(1);
 --B1: Df port map (SEL(0), din);
 --G2: Df port map (dout, SEL(1));
 --G3: Df port map (dout, SEL(2));


end Behavioral;
```

Oblig2:

SEG7CTRL.VHD

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

use work.subprog_pck.all;

entity seg7ctrl is
port
(
mclk         : in std_logic; --100MHz, positive flank
reset        : in std_logic; --Asynchronous reset, activeh
d0           : in std_logic_vector(3 downto 0); -- first display?
d1           : in std_logic_vector(3 downto 0); -- second display?
d2           : in std_logic_vector(3 downto 0); -- third display?
d3           : in std_logic_vector(3 downto 0); -- fourth display?
dec          : in std_logic_vector(3 downto 0); -- pumktum
abcdefgdec_n : out std_logic_vector(7 downto 0); -- what to be displayed inverted
a_n          : out std_logic_vector(3 downto 0) -- select display
);
end entity seg7ctrl;

architecture disp of seg7ctrl is

begin

        Process (mclk, reset)
        variable counter : std_logic_vector(1 downto 0) := "00";
        begin
```

```
	if reset = '1' then
		-- reset all
		counter := "00";
		abcdefgdec_n <= "00000000";
		a_n <= "0000";


	elsif rising_edge(mclk) then
		if (counter = "11") then
		counter := "00";
		else
		counter := std_logic_vector(unsigned(counter) + 1);
		end if;

		case counter is
			when "00" =>
				a_n <= "0001";
				abcdefgdec_n <= hex2seg7(d0, dec(0));
			when "01" =>
				a_n <= "0010";
				abcdefgdec_n <= hex2seg7(d1, dec(1));
			when "10" =>
				a_n <= "0011";
				abcdefgdec_n <= hex2seg7(d2, dec(2));
			when others =>
				a_n <= "0100";
				abcdefgdec_n <= hex2seg7(d3, dec(3));
		end case;

	end if;
	end process;
end;
```

REG_CTRL.VHD

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

use work.subprog_pck.all;

entity regctrl is
```

```
port (
mclk        : in std_logic; --100MHz, positive flank
reset       : in std_logic; --Asynchronous reset, activeh
SW                          : in std_logic_vector(7 downto 0);
BTNR              : in std_logic;
BTNL              : in std_logic
);
end regctrl;

architecture thing of regctrl is

component seg7ctrl is
port
(
mclk        : in std_logic; --100MHz, positive flank
reset       : in std_logic; --Asynchronous reset, activeh
d0          : in std_logic_vector(3 downto 0); -- first display?
d1          : in std_logic_vector(3 downto 0); -- second display?
d2          : in std_logic_vector(3 downto 0); -- third display?
d3          : in std_logic_vector(3 downto 0); -- fourth display?
dec         : in std_logic_vector(3 downto 0); -- pumktum
abcdefgdec_n : out std_logic_vector(7 downto 0); -- what to be displayed inverted
a_n         : out std_logic_vector(3 downto 0) -- select display
);
end component seg7ctrl;

signal d0          : std_logic_vector(3 downto 0); -- first display?
signal d1          : std_logic_vector(3 downto 0); -- second display?
signal d2          : std_logic_vector(3 downto 0); -- third display?
signal d3          : std_logic_vector(3 downto 0); -- fourth display?
signal dec          : std_logic_vector(3 downto 0); -- pumktum
signal abcdefgdec_n : std_logic_vector(7 downto 0); -- what to be displayed inverted
signal a_n          : std_logic_vector(3 downto 0);


begin



  process (reset, mclk, BTNR) is
  begin
  if (reset ='1') or (BTNR = '1') then
          d0 <= "0000";
          d1 <= "0000";
          d2 <= "0000";
          d3 <= "0000";

  end if;

  if BTNL = '1' and rising_edge(mclk) then
          case SW(7 downto 6) is
                  when "00" =>
          d0 <= SW(3 downto 0);
                  when "01" =>
          d1 <= SW(3 downto 0);
                  when "10" =>
          d2 <= SW(3 downto 0);
                  when others =>
          d3 <= SW(3 downto 0);

          end case;
  end if;

  dec <= "1111";
  end process;

  SEG: seg7ctrl port map (mclk, reset, d0, d1, d2, d3, dec, abcdefgdec_n, a_n);

end;
```

PARGENE.VHD

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

use work.subprog_pck.all;

entity pargen is
  port (
    rst_n       : in  std_logic;
    mclk        : in  std_logic;
    indata1     : in  std_logic_vector(15 downto 0);
    indata2     : in  unsigned(15 downto 0);
    par        : out std_logic);
```

```
end pargen;

  --signal parity1, parity2 : out std_logic;

architecture rtl1 of pargen is

  signal B   : std_logic;
  signal A    : std_logic;

  begin

  process (rst_n, mclk) is

  begin
    if (rst_n = '0') then
      par <= '0';
    elsif rising_edge(mclk) then
     foc(indata1, B);
            foc(indata2, A);

            par <=  B xor A;
    end if;
  end process;
end rtl1;
```

PARGEN.VHD

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

use work.subprog_pck.all;

entity pargen is
  port (
    rst_n        : in  std_logic;
    mclk         : in  std_logic;
    indata1      : in  std_logic_vector(15 downto 0);
    indata2      : in  unsigned(15 downto 0);
    par          : out std_logic);
end pargen;
```

```
architecture rtl1 of pargen is

begin
  process (rst_n, mclk) is

  begin
    if (rst_n = '0') then
      --parity1 := '0';
      --parity2 := '0';
      par <= '0';
    elsif rising_edge(mclk) then

                        par <= func(indata1) xor func(indata2);

    end if;
  end process;
end rtl1;
```

TB_SEG7CTRL

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

use work.subprog_pck.all;

entity tb_seg7ctrl is

end tb_seg7ctrl;

architecture TestBench of tb_seg7ctrl is

component seg7ctrl is
port
(
mclk         : in std_logic; --100MHz, positive flank
reset        : in std_logic; --Asynchronous reset, activeh
d0           : in std_logic_vector(3 downto 0); -- first display?
d1           : in std_logic_vector(3 downto 0); -- second display?
d2           : in std_logic_vector(3 downto 0); -- third display?
d3           : in std_logic_vector(3 downto 0); -- fourth display?
dec          : in std_logic_vector(3 downto 0); -- pumktum
abcdefgdec_n : out std_logic_vector(7 downto 0); -- what to be displayed inverted
a_n          : out std_logic_vector(3 downto 0) -- select display
);
end component seg7ctrl;

signal mclk         : std_logic; --100MHz, positive flank
signal reset        : std_logic; --Asynchronous reset, activeh
signal d0           : std_logic_vector(3 downto 0); -- first display?
signal d1           : std_logic_vector(3 downto 0); -- second display?
signal d2           : std_logic_vector(3 downto 0); -- third display?
signal d3           : std_logic_vector(3 downto 0); -- fourth display?
signal dec          : std_logic_vector(3 downto 0); -- pumktum
signal abcdefgdec_n : std_logic_vector(7 downto 0); -- what to be displayed inverted
signal a_n          : std_logic_vector(3 downto 0);


begin

  P_CLK_0: process
  begin
    mclk <= '0';
    wait for 5 ns;
    mclk <= '1';
    wait for 5 ns;
  end process P_CLK_0;

  reset <= '1', '0' after 10 ns;
  d0 <= "0011";
  d1 <= "0001";
  d2 <= "1011";
  d3 <= "0111";

  dec <= "1111";

  G1: seg7ctrl port map (mclk, reset, d0, d1, d2, d3, dec, abcdefgdec_n, a_n);

end;
```

TB_PARGEN.VHD

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

entity tb_pargen is
```

```
  -- empty;
end tb_pargen;

architecture beh1 of tb_pargen is

  component pargen is
    port (rst_n   : in  std_logic;
          mclk    : in  std_logic;
          indata1 : in  std_logic_vector(15 downto 0);
          indata2 : in  unsigned(15 downto 0);
          par     : out std_logic);
  end component pargen;

  signal rst_n   : std_logic;
  signal mclk    : std_logic;
  signal indata1 : std_logic_vector(15 downto 0);
  signal indata2 : unsigned(15 downto 0);
  signal par     : std_logic;

begin

  UUT: entity work.pargen(rtl1)
    port map (rst_n   => rst_n,
              mclk    => mclk,
              indata1 => indata1,
              indata2 => indata2,
              par     => par);

  P_CLK_0: process
  begin
    mclk <= '0';
    wait for 50 ns;
    mclk <= '1';
    wait for 50 ns;
  end process P_CLK_0;

  rst_n  <= '0', '1' after 100 ns;
  indata1 <= x"0001",
             x"0003" after 500 ns,
             x"0004" after 900 ns;
  indata2 <= x"0000",
             x"0001" after 100 ns,
             x"0003" after 200 ns,
             x"0007" after 300 ns,
             x"000F" after 400 ns,
             x"0000" after 500 ns,
             x"0001" after 600 ns,
             x"0003" after 700 ns,
             x"0007" after 800 ns,
             x"000F" after 900 ns;

end beh1;
```

TB_HEX2SEG7.VHD

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

use work.subprog_pck.all;


entity tb_hex2seg7 is

end tb_hex2seg7;



architecture TestBench of tb_hex2seg7 is

        signal output               : std_logic_vector(7 downto 0);

        signal SEL                              : std_logic;
        signal indata       : std_logic_vector(3 downto 0);

        component seg7model is
                port
                (
                a_n           : in  std_logic_vector(3 downto 0);
                abcdefgdec_n  : in  std_logic_vector(7 downto 0);
                disp3         : out std_logic_vector(3 downto 0);
                disp2         : out std_logic_vector(3 downto 0);
                disp1         : out std_logic_vector(3 downto 0);
                disp0         : out std_logic_vector(3 downto 0)
                );
  end component seg7model;
```

```
                    signal a_n           : std_logic_vector(3 downto 0);
                    --signal abcdefgdec_n  : std_logic_vector(7 downto 0);
                    signal disp3         : std_logic_vector(3 downto 0);
                    signal disp2         : std_logic_vector(3 downto 0);
                    signal disp1         : std_logic_vector(3 downto 0);
                    signal disp0         : std_logic_vector(3 downto 0);

begin

          --variable indata       :  std_logic_vector(3 downto 0) := "0001";
          --variable SEL                      : std_logic := '0';
          --variable output                   : std_logic_vector(7 downto 0);




          --kalle hex2seg7(SIGNAL, SEL), returnerer char og send det inn i seg7model_beh

 SEL <= '0';

  indata <= "0000",
           "0001" after 100 ns,
           "0011" after 200 ns,
           "0111" after 300 ns,
           "1111" after 400 ns,
           "0000" after 500 ns,
           "0001" after 600 ns,
           "0011" after 700 ns,
           "0111" after 800 ns,
           "1111" after 900 ns;

          output <= hex2seg7(indata, SEL);
          a_n <= "0000"; --hvilken skjerm

          G1: seg7model port map (a_n, output, disp3, disp2, disp1, disp0);

end;
```

SUBPROG_PCK

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

```
package subprog_pck is
                    function func (indata1 : in std_logic_vector(15 downto 0))
                    return std_logic;
                    function func (indata2 : in  unsigned(15 downto 0))
                    return std_logic;

                    procedure foc (signal indata1 : in std_logic_vector; signal par : out std_logic);
                    procedure foc (signal indata2 : in unsigned; signal par : out std_logic);

                    function hex2seg7 (signal STATE : in std_logic_vector(3 downto 0); signal SEL :
in std_logic)
                    return std_logic_vector;
end subprog_pck;

package body subprog_pck is

          function func (indata1 : in std_logic_vector(15 downto 0))
          return std_logic is variable parity1 : std_logic;
                    begin
                    parity1 := '0';
                    for i in indata1'range loop
                              if indata1(i) = '1' then
                              parity1 := not parity1;
                              end if;
                    end loop;
                              return parity1;
                    end;


          function func (indata2 : in  unsigned(15 downto 0))
          return std_logic is variable parity2 : std_logic;
          begin
          parity2 := '0';
     for j in indata2'range loop
      parity2 := parity2 xor indata2(j);
     end loop;
          return parity2;
          end;


          function hex2seg7 (signal STATE : in std_logic_vector(3 downto 0); signal SEL : in
std_logic)
          return std_logic_vector is variable char : std_logic_vector(7 downto 0);
          begin
            case STATE is
                    when "0000" => char := "00000011";
                    when "0001" => char := "10011111";
                    when "0010" => char := "00100101";
                    when "0011" => char := "00001101";
                    when "0100" => char := "10011001";
                    when "0101" => char := "01001001";
                    when "0110" => char := "01000001";
                    when "0111" => char := "00011111";
                    when "1000" => char := "00000001";
                    when "1001" => char := "00011001";
                    when "1010" => char := "00010001";
                    when "1011" => char := "11000001";
                    when "1100" => char := "01100011";
                    when "1101" => char := "10000101";
                    when "1110" => char := "01100001";
                    when "1111" => char := "01110001";

                    when others => char := "00000000";

            end case;

            if (SEL = '1' ) then
                    char(0) := '0';

            end if;

            return char;
          end;


          procedure foc (signal indata1 : in std_logic_vector; signal par : out std_logic) is
          variable parity1 : std_logic := '0';

          begin
          parity1 := '0';
     for i in indata1'range loop
       if indata1(i) = '1' then
         parity1 := not parity1;
       end if;
     end loop;
            par <= parity1;
```

```
  end procedure foc;

            procedure foc (signal indata2 : in unsigned; signal par : out std_logic) is
            variable parity2 : std_logic;
            begin
      parity2 := '0';
      for j in indata2'range loop
       parity2 := parity2 xor indata2(j);
      end loop;
              par <= parity2;
  end procedure foc;


end subprog_pck;
```

STOPWATCH.VHD

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

use work.subprog_pck.all;

entity stopwatch is
port (
mclk         : in std_logic; --100MHz, positive flank
reset        : in std_logic; --Asynchronous reset, activeh
SW                            : in std_logic_vector(7 downto 0);
BTNR              : in std_logic;
BTNL              : in std_logic;
BTNC              : in std_logic
);
end stopwatch;

architecture thing of stopwatch is

component seg7ctrl is
port
(
mclk         : in std_logic; --100MHz, positive flank
reset        : in std_logic; --Asynchronous reset, activeh
d0          : in std_logic_vector(3 downto 0); -- first display?
d1          : in std_logic_vector(3 downto 0); -- second display?
d2          : in std_logic_vector(3 downto 0); -- third display?
d3          : in std_logic_vector(3 downto 0); -- fourth display?
dec         : in std_logic_vector(3 downto 0); -- pumktum
abcdefgdec_n : out std_logic_vector(7 downto 0); -- what to be displayed inverted
a_n         : out std_logic_vector(3 downto 0) -- select display
);
end component seg7ctrl;

signal d0          : std_logic_vector(3 downto 0); -- first display?
signal d1          : std_logic_vector(3 downto 0); -- second display?
signal d2          : std_logic_vector(3 downto 0); -- third display?
signal d3          : std_logic_vector(3 downto 0); -- fourth display?
signal dec         : std_logic_vector(3 downto 0); -- pumktum
signal abcdefgdec_n : std_logic_vector(7 downto 0); -- what to be displayed inverted
signal a_n         : std_logic_vector(3 downto 0);



 begin

  process (reset, mclk, BTNR) is
  variable counter : std_logic_vector(26 downto 0) := "000000000000000000000000000";
  variable running : std_logic := '0';
  begin
  if (reset ='1') or (BTNR = '1') then
        d0 <= "0000";
        d1 <= "0000";
        d2 <= "0000";
        d3 <= "0000";
        running := '0';

  end if;

  -- start = BTNL, STOp = BTNC
  if BTNL = '1' then
        running := '1';
  elsif BTNC = '1' then
        running := '0';
  end if;


  if rising_edge(mclk) and running = '1' then
        -- counter for ea second
        if counter = "101111101011110000100000000" then
```

```
                counter := "000000000000000000000000000";

        if d0 = "1111" then
                d0 <= "0000";
                if d1 = "1111" then
                        d1 <= "0000";
                        if d2 <= "1111" then
                                d2 <= "0000";
                                if d3 <= "1111" then
                                        d3 <= "0000";
                                else
                                        d3 <= std_logic_vector(unsigned(d1) +1);
                                end if;
                        else
                                d2 <= std_logic_vector(unsigned(d1) +1);
                        end if;
                else
                        d1 <= std_logic_vector(unsigned(d1) +1);
                end if;
        else
                d0 <= std_logic_vector(unsigned(d0) +1);
        end if;



        else
                counter := std_logic_vector(unsigned(counter) + 1);
        end if;

  end if;

  dec <= "1111";
  end process;

  SEG: seg7ctrl port map (mclk, reset, d0, d1, d2, d3, dec, abcdefgdec_n, a_n);

end;
```

SEG7MODEL_ENT

```
-- Dette er entity for modell av sjusegmentdisplayene.  De er modellert ved at man
-- f�r vist ASCII-verdien av tallet/bokstaven som vises p� segmentene
-- Dersom man merker DISP0,..3 i waveform vieweren og velger radix ascii
-- F�r man vist tall/bokstav som vist p� sjusegmentene.
```

```
library IEEE;
use IEEE.std_logic_1164.all;

entity seg7model is
  port
  (
    a_n          : in  std_logic_vector(3 downto 0);
    abcdefgdec_n  : in  std_logic_vector(7 downto 0);
    disp3        : out std_logic_vector(3 downto 0);
    disp2        : out std_logic_vector(3 downto 0);
    disp1        : out std_logic_vector(3 downto 0);
    disp0        : out std_logic_vector(3 downto 0)
  );
end seg7model;
```

SEG7MODEL_BEH

```
-- Dette er modell for sjusegmentdisplayene.  De er modellert ved at man
-- f�r vist ASCII-verdien av tallet/bokstaven som vises p� segmentene
-- Dersom man merker disp0,..3 i waveform vieweren og velger radix ascii
-- F�r man vist tall/bokstav som vist p� sjusegmentene.

library IEEE;
use IEEE.std_logic_1164.all;

architecture beh of seg7model is
  signal char : std_logic_vector(3 downto 0);
begin
  display :
  process(a_n,char)
  begin
    --Default verdier
    --Benytter man default verdier kan man sl�yfe
    --else i if setninger uten � f� laget en latch
    --En annen fordel er at koden kan bli enklere.
    --Benytter 'Z'(h�y impendans) for � vise at et display er slukket
    disp0 <= "ZZZZ";
    disp1 <= "ZZZZ";
    disp2 <= "ZZZZ";
    disp3 <= "ZZZZ";
    if a_n(3) = '0' then
      disp3 <= char;
    end if;
    if a_n(2) = '0' then
```

```
      disp2 <= char;
    end if;
    if a_n(1) = '0' then
      disp1 <= char;
    end if;
    if a_n(0) = '0' then
      disp0 <= char;
    end if;
  end process dispLAY;

--De to metodene nedenfor er helt ekvivalente beskrivelser
--Legg merke til alternativ koding nederst dersom man bare vil vise tallverdier
--Kan v�re fint � benytte dersom man lager digitalklokke

--  ENCODE:
--  process (abcdefg_n)
--  begin
--    case abcdefg_n(7 downto 1) is
--      when "0000001" => char <= X"30"; --0
--      when "1001111" => char <= X"31"; --1
--      when "0010010" => char <= X"32"; --2
--      when "0000110" => char <= X"33"; --3
--      when "1001100" => char <= X"34"; --4
--      when "0100100" => char <= X"35"; --5
--      when "0100000" => char <= X"36"; --6
--      when "0001111" => char <= X"37"; --7
--      when "0000000" => char <= X"38"; --8
--      when "0001100" => char <= X"39"; --9
--      when "0001000" => char <= X"41"; --A
--      when "1100000" => char <= X"42"; --B
--      when "0110001" => char <= X"43"; --C
--      when "1000010" => char <= X"44"; --D
--      when "0110000" => char <= X"45"; --E
--      when "0111000" => char <= X"46"; --F
--      when "0000100" => char <= X"67"; --G
--      when "1101000" => char <= X"68"; --H
--      when "0000111" => char <= X"49"; --I
--      when "1000011" => char <= X"4A"; --J
--      when "1110001" => char <= X"4C"; --L
--      when "1101010" => char <= X"6E"; --n
--      when "1100010" => char <= X"6F"; --o
--      when "0011000" => char <= X"50"; --P
--      when "1111010" => char <= X"72"; --r
--      when "1110000" => char <= X"74"; --t
--      when "1100011" => char <= X"75"; --u
--      when "1000100" => char <= X"59"; --Y
--      when others    => char <= "XXXXXXXX";
--    end case;
--  end process encode;

--  with abcdefgdec_n(7 downto 1) select
--      char <= X"30" when "0000001", --0
--              X"31" when "1001111", --1
--              X"32" when "0010010", --2
--              X"33" when "0000110", --3
--              X"34" when "1001100", --4
--              X"35" when "0100100", --5
--              X"36" when "0100000", --6
--              X"37" when "0001111", --7
--              X"38" when "0000000", --8
--              X"39" when "0001100", --9
--              X"41" when "0001000", --A
--              X"42" when "1100000", --B
--              X"43" when "0110001", --C
--              X"44" when "1000010", --D
--              X"45" when "0110000", --E
--              X"46" when "0111000", --F
--              X"67" when "0000100", --G
--              X"68" when "1101000", --H
--              X"49" when "0000111", --I
--              X"4A" when "1000011", --J
--              X"4C" when "1110001", --L
--              X"6E" when "1101010", --n
--              X"6F" when "1100010", --o
--              X"50" when "0011000", --P
--              X"72" when "1111010", --r
--              X"74" when "1110000", --t
--              X"75" when "1100011", --u
--              X"59" when "1000100", --Y
--              "XXXXXXXX" when others;

-- Eventuelt kan det v�re hensiktsmessig � vise bare hexadesimale tall 0-F

   with abcdefgdec_n(7 downto 1) select
     char <= X"0" when "0000001", --0
             X"1" when "1001111", --1
```

```
            X"2" when "0010010", --2
            X"3" when "0000110", --3
            X"4" when "1001100", --4
            X"5" when "0100100", --5
            X"6" when "0100000", --6
            X"7" when "0001111", --7
            X"8" when "0000000", --8
            X"9" when "0000100", --9
            X"A" when "0001000", --A
            X"B" when "1100000", --B
            X"C" when "0110001", --C
            X"D" when "1000010", --D
            X"E" when "0110000", --E
            X"F" when "0111000", --F
            "XXXX" when others;

end architecture beh;
```

Oblig 3:

POS_SEG7_CTRL.VHD

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

architecture pos_seg7_ctrl of pos_seg7_ctrl is

 component pos_ctrl is
   port (
        rst       : in  std_logic;          -- Reset
   rst_div   : in  std_logic;          -- Reset
   mclk      : in  std_logic;          -- Clock
   mclk_div  : in  std_logic;          -- Clock to p_reg
   sync_rst  : in  std_logic;          -- Synchronous reset
   sp        : in  signed(7 downto 0);  -- Setpoint (wanted position)
   a         : in  std_logic;          -- From position sensor
   b         : in  std_logic;          -- From position sensor
   pos       : out signed(7 downto 0);  -- Measured Position
   force_cw  : in  std_logic;          -- Force motor clock wise motion
   force_ccw : in  std_logic;  -- Force motor counter clock wise motion
```

```
    motor_cw  : out std_logic;          -- Motor clock wise motion
    motor_ccw : out std_logic           -- Motor counter clock wise motion
          );
  end component pos_ctrl;


component seg7ctrl is
  port
  (
  mclk        : in std_logic; --100MHz, positive flank
          reset       : in std_logic; --Asynchronous reset, activeh
          d0          : in std_logic_vector(3 downto 0); -- first display?
          d1          : in std_logic_vector(3 downto 0); -- second display?
          d2          : in std_logic_vector(3 downto 0); -- third display?
          d3          : in std_logic_vector(3 downto 0); -- fourth display?
          dec         : in std_logic_vector(3 downto 0); -- pumktum
          abcdefgdec_n : out std_logic_vector(7 downto 0); -- what to be displayed inverted
          a_n         : out std_logic_vector(3 downto 0) -- select display
  );
end component seg7ctrl;


component cru is
port (
    arst        : in  std_logic;
    refclk : in std_logic;
    rst_div : out std_logic;
    rst   : out std_logic;
    mclk_div  : out std_logic;
    mclk  : out std_logic
);
end component cru;

          signal posi : signed(7 downto 0);
          signal rst         : std_logic;
          signal mclk      : std_logic;
          signal mclk_div           : std_logic;
          signal rst_div   : std_logic;
          signal sp1 : std_logic_vector(7 downto 0);


begin
```

```
sp1 <= '0' & sp(6 downto 0);

  O2: entity work.cru
      port map (
      arst          => arst,
      refclk => refclk,
      rst_div => rst_div,
      rst   => rst,
      mclk_div  => mclk_div,
      mclk  => mclk
      );

O: entity work.pos_ctrl
    port map (sync_rst => sync_rst,
                                rst_div => rst_div,
                                mclk_div => mclk_div,
                                sp => signed(sp1),
                                force_cw => force_cw,
                                force_ccw => force_ccw,
                                motor_cw => motor_cw,
                                motor_ccw => motor_ccw,
            mclk => mclk,
            a => a,
            b => b,
                                rst => rst,
            pos => posi);

O1: entity work.seg7ctrl
    port map (
            abcdefgdec_n => abcdefgdec_n,
            mclk => mclk,
            reset => rst,
            d0 => std_logic_vector(posi(3 downto 0)),
            d1 => std_logic_vector(posi(7 downto 4)),
            d2 => sp1(3 downto 0),
            d3 => sp1(7 downto 4),
            dec => "0000",
            a_n => a_n);



      --force_ccw <= BTNL;
      --force_cw <= BTNR;
```

```
   --sync_rst <= BTNC;


end architecture;
```

POS_SEG7_CTRL_ENT.VHD

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


entity pos_seg7_ctrl is
  port (
    -- System Clock and Reset
    arst        : in  std_logic;      -- Reset
    sync_rst    : in  std_logic;      -- Synchronous reset
    refclk      : in  std_logic;      -- Clock
    sp          : in  std_logic_vector(7 downto 0);  -- Set Point
    a           : in  std_logic;      -- From position sensor
    b           : in  std_logic;      -- From position sensor
    force_cw    : in  std_logic;      -- Force motor clock wise motion
    force_ccw   : in  std_logic;                    -- Force motor counter clock wise motion
    motor_cw    : out std_logic;      -- Motor clock wise motion
    motor_ccw   : out std_logic;      -- Motor counter clock wise motion
    -- Interface to seven segments
    abcdefgdec_n : out std_logic_vector(7 downto 0);
    a_n         : out std_logic_vector(3 downto 0)
    );
end pos_seg7_ctrl;
```

RSTSYNCH_ENT.VHD

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

entity rstsynch is
  port (
    arst        : in  std_logic; -- Asynch. reset
          mclk        : in  std_logic; -- Master clock
          mclk_div    : in  std_logic; -- Master clock div. by 128
          rst         : out  std_logic; --Synch. reset master clock
          rst_div     : out  std_logic --Synch. reset div. by 128
    );
end rstsynch;
```

RSTSYNCH_RTL.VHD

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

architecture dav of rstsynch is
  signal rst_s1, rst_s2 : std_logic;
  signal rst_div_s1, rst_div_s2 : std_logic;

 begin

   P_RST_0: process(arst, mclk)
   begin
    if arst = '1' then
                rst_s1 <= '1';
                          rst_s2 <= '1';
            elsif rising_edge(mclk) then
        rst_s1 <= '0';
                            rst_s2 <= rst_s1;
            end if;
   end process P_RST_0;

   P_RST_1: process (arst, mclk_div)
   begin
    if arst = '1' then
                 rst_div_s1 <= '1';
                 rst_div_s2 <= '1';
            elsif rising_edge(mclk_div) then
     rst_div_s1 <= '0';
     rst_div_s2 <= rst_div_s1;

    end if;
   end process P_RST_1;

   rst <= rst_s2;
   rst_div <= rst_div_s2;

end architecture dav;
```

SEG7CTRL.VHD

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

use work.subprog_pck.all;

entity seg7ctrl is
port
(
mclk         : in std_logic; --100MHz, positive flank
reset        : in std_logic; --Asynchronous reset, activeh
d0           : in std_logic_vector(3 downto 0); -- first display?
d1           : in std_logic_vector(3 downto 0); -- second display?
d2           : in std_logic_vector(3 downto 0); -- third display?
d3           : in std_logic_vector(3 downto 0); -- fourth display?
dec          : in std_logic_vector(3 downto 0); -- pumktum
abcdefgdec_n : out std_logic_vector(7 downto 0); -- what to be displayed inverted
a_n          : out std_logic_vector(3 downto 0) -- select display
);
end entity seg7ctrl;

architecture disp of seg7ctrl is

begin

        Process (mclk, reset)
        variable counter : unsigned(15 downto 0) := (others => '0');
        begin

        if reset = '1' then
                -- reset all
                counter := (others => '0');
                abcdefgdec_n <= "00000000";
                a_n <= "0000";


        elsif rising_edge(mclk) then
                counter := counter + 1;

                case counter(15 downto 14) is
                        when "00" =>
                                a_n <= "1110";
                                abcdefgdec_n <= hex2seg7(d0, dec(0));
                        when "01" =>
                                a_n <= "1101";
                                abcdefgdec_n <= hex2seg7(d1, dec(1));
                        when "10" =>
                                a_n <= "1011";
                                abcdefgdec_n <= hex2seg7(d2, dec(2));
                        when others =>
                                a_n <= "0111";
                                abcdefgdec_n <= hex2seg7(d3, dec(3));
                end case;

        end if;
        end process;
end;
```

SEG7MODEL_BEH.VHD

```
-- Dette er modell for sjusegmentdisplayene.  De er modellert ved at man
-- f�r vist ASCII-verdien av tallet/bokstaven som vises p� segmentene
-- Dersom man merker disp0,..3 i waveform vieweren og velger radix ascii
-- F�r man vist tall/bokstav som vist p� sjusegmentene.

library IEEE;
use IEEE.std_logic_1164.all;

architecture beh of seg7model is
  signal char : std_logic_vector(3 downto 0);
begin
  display :
  process(a_n,char)
  begin
    --Default verdier
    --Benytter man default verdier kan man sl�yfe
    --else i if setninger uten � f� laget en latch
    --En annen fordel er at koden kan bli enklere.
    --Benytter 'Z'(h�y impendans) for � vise at et display er slukket
    disp0 <= "ZZZZ";
    disp1 <= "ZZZZ";
    disp2 <= "ZZZZ";
    disp3 <= "ZZZZ";
    if a_n(3) = '0' then
      disp3 <= char;
    end if;
    if a_n(2) = '0' then
```

```
      disp2 <= char;
    end if;
    if a_n(1) = '0' then
      disp1 <= char;
    end if;
    if a_n(0) = '0' then
      disp0 <= char;
    end if;
  end process dispLAY;

--De to metodene nedenfor er helt ekvivalente beskrivelser
--Legg merke til alternativ koding nederst dersom man bare vil vise tallverdier
--Kan v�re fint � benytte dersom man lager digitalklokke

--  ENCODE:
--  process (abcdefg_n)
--  begin
--    case abcdefg_n(7 downto 1) is
--      when "0000001" => char <= X"30"; --0
--      when "1001111" => char <= X"31"; --1
--      when "0010010" => char <= X"32"; --2
--      when "0000110" => char <= X"33"; --3
--      when "1001100" => char <= X"34"; --4
--      when "0100100" => char <= X"35"; --5
--      when "0100000" => char <= X"36"; --6
--      when "0001111" => char <= X"37"; --7
--      when "0000000" => char <= X"38"; --8
--      when "0001100" => char <= X"39"; --9
--      when "0001000" => char <= X"41"; --A
--      when "1100000" => char <= X"42"; --B
--      when "0110001" => char <= X"43"; --C
--      when "1000010" => char <= X"44"; --D
--      when "0110000" => char <= X"45"; --E
--      when "0111000" => char <= X"46"; --F
--      when "0000100" => char <= X"67"; --G
--      when "1101000" => char <= X"68"; --H
--      when "0000111" => char <= X"49"; --I
--      when "1000011" => char <= X"4A"; --J
--      when "1110001" => char <= X"4C"; --L
--      when "1101010" => char <= X"6E"; --n
--      when "1100010" => char <= X"6F"; --o
--      when "0011000" => char <= X"50"; --P
--      when "1111010" => char <= X"72"; --r
--      when "1110000" => char <= X"74"; --t
--      when "1100011" => char <= X"75"; --u
--      when "1000100" => char <= X"59"; --Y
--      when others    => char <= "XXXXXXXX";
--    end case;
--  end process encode;

--  with abcdefgdec_n(7 downto 1) select
--      char <= X"30" when "0000001", --0
--              X"31" when "1001111", --1
--              X"32" when "0010010", --2
--              X"33" when "0000110", --3
--              X"34" when "1001100", --4
--              X"35" when "0100100", --5
--              X"36" when "0100000", --6
--              X"37" when "0001111", --7
--              X"38" when "0000000", --8
--              X"39" when "0001100", --9
--              X"41" when "0001000", --A
--              X"42" when "1100000", --B
--              X"43" when "0110001", --C
--              X"44" when "1000010", --D
--              X"45" when "0110000", --E
--              X"46" when "0111000", --F
--              X"67" when "0000100", --G
--              X"68" when "1101000", --H
--              X"49" when "0000111", --I
--              X"4A" when "1000011", --J
--              X"4C" when "1110001", --L
--              X"6E" when "1101010", --n
--              X"6F" when "1100010", --o
--              X"50" when "0011000", --P
--              X"72" when "1111010", --r
--              X"74" when "1110000", --t
--              X"75" when "1100011", --u
--              X"59" when "1000100", --Y
--              "XXXXXXXX" when others;

-- Eventuelt kan det v�re hensiktsmessig � vise bare hexadesimale tall 0-F

   with abcdefgdec_n(7 downto 1) select
      char <= X"0" when "0000001", --0
              X"1" when "1001111", --1
```

```
        X"2" when "0010010", --2
        X"3" when "0000110", --3
        X"4" when "1001100", --4
        X"5" when "0100100", --5
        X"6" when "0100000", --6
        X"7" when "0001111", --7
        X"8" when "0000000", --8
        X"9" when "0000100", --9
        X"A" when "0001000", --A
        X"B" when "1100000", --B
        X"C" when "0110001", --C
        X"D" when "1000010", --D
        X"E" when "0110000", --E
        X"F" when "0111000", --F
        "XXXX" when others;

end architecture beh;
```

SEG7MODEL_ENT.VHD

```
-- Dette er entity for modell av sjusegmentdisplayene.  De er modellert ved at man
-- f�r vist ASCII-verdien av tallet/bokstaven som vises p� segmentene
-- Dersom man merker DISP0,..3 i waveform vieweren og velger radix ascii
-- F�r man vist tall/bokstav som vist p� sjusegmentene.

library IEEE;
use IEEE.std_logic_1164.all;

entity seg7model is
  port
  (
    a_n           : in  std_logic_vector(3 downto 0);
    abcdefgdec_n  : in  std_logic_vector(7 downto 0);
    disp3         : out std_logic_vector(3 downto 0);
    disp2         : out std_logic_vector(3 downto 0);
    disp1         : out std_logic_vector(3 downto 0);
    disp0         : out std_logic_vector(3 downto 0)
  );
end seg7model;
```

SUBPROG_PCK.VHD

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

```
package subprog_pck is
                    function func (indata1 : in std_logic_vector(15 downto 0))
                    return std_logic;
                    function func (indata2 : in  unsigned(15 downto 0))
                    return std_logic;

                    procedure foc (signal indata1 : in std_logic_vector; signal par : out std_logic);
                    procedure foc (signal indata2 : in unsigned; signal par : out std_logic);

                    function hex2seg7 (signal STATE : in std_logic_vector(3 downto 0); signal SEL :
in std_logic)
                    return std_logic_vector;
end subprog_pck;

package body subprog_pck is

          function func (indata1 : in std_logic_vector(15 downto 0))
          return std_logic is variable parity1 : std_logic;
                    begin
                    parity1 := '0';
                    for i in indata1'range loop
                              if indata1(i) = '1' then
                              parity1 := not parity1;
                              end if;
                    end loop;
                              return parity1;
                    end;


          function func (indata2 : in  unsigned(15 downto 0))
          return std_logic is variable parity2 : std_logic;
          begin
          parity2 := '0';
     for j in indata2'range loop
      parity2 := parity2 xor indata2(j);
     end loop;
          return parity2;
          end;


          function hex2seg7 (signal STATE : in std_logic_vector(3 downto 0); signal SEL : in
std_logic)
```

```
      return std_logic_vector is variable char : std_logic_vector(7 downto 0);
      begin
        case STATE is
                when "0000" => char := "00000011";
                when "0001" => char := "10011111";
                when "0010" => char := "00100101";
                when "0011" => char := "00001101";
                when "0100" => char := "10011001";
                when "0101" => char := "01001001";
                when "0110" => char := "01000001";
                when "0111" => char := "00011111";
                when "1000" => char := "00000001";
                when "1001" => char := "00011001";
                when "1010" => char := "00010001";
                when "1011" => char := "11000001";
                when "1100" => char := "01100011";
                when "1101" => char := "10000101";
                when "1110" => char := "01100001";
                when "1111" => char := "01110001";

                when others => char := "00000000";

        end case;

        if (SEL = '1' ) then
                char(0) := '0';

        end if;

        return char;
      end;


      procedure foc (signal indata1 : in std_logic_vector; signal par : out std_logic) is
      variable parity1 : std_logic := '0';

      begin
      parity1 := '0';
for i in indata1'range loop
  if indata1(i) = '1' then
    parity1 := not parity1;
  end if;
end loop;
```

```
            par <= parity1;
  end procedure foc;

          procedure foc (signal indata2 : in unsigned; signal par : out std_logic) is
          variable parity2 : std_logic;
          begin
     parity2 := '0';
     for j in indata2'range loop
      parity2 := parity2 xor indata2(j);
     end loop;
            par <= parity2;
  end procedure foc;


end subprog_pck;
```

TB_CTRL.VHD

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


entity tb is

end tb;

architecture tb_ctrl of tb is

 component p_ctrl is
   port (
           clk : in std_logic;
           rst : in std_logic;
           sp : in signed(7 downto 0);
           pos : in signed(7 downto 0);
           motor_cw  : out std_logic;          --Motor Clock Wise direction
           motor_ccw : out std_logic
           );
  end component p_ctrl;


          signal clk : std_logic;
          signal rst : std_logic;
          signal pos : signed(7 downto 0);
```

```
            signal sp : signed(7 downto 0);
            signal motor_ccw : std_logic;
            signal motor_cw : std_logic;

begin

Q: entity work.p_ctrl
    port map (clk => clk,
                                motor_cw => motor_cw,
                                motor_ccw => motor_ccw,
                                rst => rst,
                                sp => sp,
            pos => pos);


            P_CLK_0: process
  begin
    clk <= '0';
    wait for 50 ns;
    clk <= '1';
    wait for 50 ns;
  end process P_CLK_0;


  rst <= '1', '0' after 100 ns;

            pos <= "00000000",
            "00001100" after 200 us,
            "00110000" after 300 us,
            "00000000" after 400 us,
            "00000000" after 500 us;

            sp <= "00001000",
            "00000000" after 200 us,
            "00000011" after 300 us,
            "00001000" after 400 us,
            "00000000" after 500 us;



end architecture tb_ctrl;
```

TB_FULL_POS_CTRL

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

entity tb is

end tb;

architecture tb_full of tb is

 component pos_ctrl is
   port (
         rst       : in  std_logic;          -- Reset
   rst_div   : in  std_logic;          -- Reset
   mclk      : in  std_logic;          -- Clock
   mclk_div  : in  std_logic;          -- Clock to p_reg
   sync_rst  : in  std_logic;          -- Synchronous reset
   sp        : in  signed(7 downto 0);  -- Setpoint (wanted position)
   a         : in  std_logic;          -- From position sensor
   b         : in  std_logic;          -- From position sensor
   pos       : out signed(7 downto 0);  -- Measured Position
   force_cw  : in  std_logic;          -- Force motor clock wise motion
   force_ccw : in  std_logic;  -- Force motor counter clock wise motion
   motor_cw  : out std_logic;          -- Motor clock wise motion
   motor_ccw : out std_logic           -- Motor counter clock wise motion
         );
  end component pos_ctrl;

         signal rst       :  std_logic;          -- Reset
   signal rst_div   :  std_logic;          -- Reset
   signal mclk      :  std_logic;          -- Clock
   signal mclk_div  :  std_logic;          -- Clock to p_reg
   signal sync_rst  :  std_logic;          -- Synchronous reset
   signal sp        :  signed(7 downto 0);  -- Setpoint (wanted position)
   signal a         :  std_logic;          -- From position sensor
   signal b         :  std_logic;          -- From position sensor
   signal pos       :  signed(7 downto 0);  -- Measured Position
   signal force_cw  :  std_logic;          -- Force motor clock wise motion
   signal force_ccw :  std_logic;  -- Force motor counter clock wise motion
   signal motor_cw  :  std_logic;          -- Motor clock wise motion
   signal motor_ccw :  std_logic;           -- Motor counter clock wise motion

component motor is
```

```
  port (
    motor_cw  : in  std_logic;
    motor_ccw : in  std_logic;
    a         : out std_logic;
    b         : out std_logic
    );
end component motor;

 --signal sp        :  signed(7 downto 0); -- Setpoint (wanted position)

            begin

            O: entity work.pos_ctrl
    port map (sync_rst => sync_rst,
                                                rst_div => rst_div,
                                                mclk_div => mclk_div,
                                                sp => sp,
                                                force_cw => force_cw,
                                                force_ccw => force_ccw,
                                                motor_cw => motor_cw,
                                                motor_ccw => motor_ccw,
            mclk => mclk,
            a => a,
            b => b,
                                    rst => rst,
            pos => pos);

            O2: entity work.motor
            port map (
                        motor_cw => motor_cw,
                        motor_ccw => motor_ccw,
                        a => a,
                        b => b
            );


P_CLK_0: process
  begin
    mclk <= '0';
            mclk_div <= '0';
    wait for 50 ns;
    mclk <= '1';
            mclk_div <= '1';
    wait for 50 ns;
  end process P_CLK_0;

  rst <= '1', '0' after 100 ns;
  rst_div <= '1', '0' after 100 ns;
  sync_rst <= '1', '0' after 100 ns;


            sp <= "00001000",
            "00000000" after 200 us,
            "00000011" after 300 us,
            "00001000" after 400 us,
            "00000000" after 500 us;
            force_cw <= '0';
            force_ccw <= '0';


end architecture tb_full;
```

TB_POS.VHD

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

entity tb is

end tb;

architecture tb_pos of tb is

 component pos_meas_beh is
    port (a : in std_logic;
             b : in std_logic;
             sync_rst : in std_logic;
             clk : in std_logic;
             rst : in std_logic;
             pos : out signed(7 downto 0));
  end component pos_meas_beh;

            signal a : std_logic;
            signal b : std_logic;
            signal sync_rst : std_logic;
            signal clk : std_logic;
            signal rst : std_logic;
            signal pos : signed(7 downto 0);

begin

Q: entity work.pos_meas
    port map (sync_rst => sync_rst,
            clk => clk,
            a => a,
            b => b,
                                    rst => rst,
            pos => pos);

P_CLK_0: process
  begin
    clk <= '0';
    wait for 50 ns;
    clk <= '1';
    wait for 50 ns;
  end process P_CLK_0;

  P_a_0: process
  begin
    a <= '0';
    wait for 20 ns;
            b <= '0';
            wait for 20 ns;
    a <= '1';
    wait for 20 ns;
            b <= '1';
            wait for 20 ns;
  end process P_a_0;

  rst <= '0', '1' after 100 ns;
  sync_rst <= '0', '1' after 100 ns;



 end architecture;
```

TB_POS_SEG7.VHD

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

entity tb is

end tb;

architecture tb_pos_seg7 of tb is

 component pos_seg7_ctrl_ent is
  port (
    -- System Clock and Reset
    arst         : in  std_logic;       -- Reset
    sync_rst     : in  std_logic;       -- Synchronous reset
    refclk       : in  std_logic;       -- Clock
    sp           : in  std_logic_vector(7 downto 0);  -- Set Point
    a            : in  std_logic;       -- From position sensor
    b            : in  std_logic;       -- From position sensor
    force_cw     : in  std_logic;       -- Force motor clock wise motion
    force_ccw    : in  std_logic;                          -- Force motor counter clock wise motion
    motor_cw     : out std_logic;       -- Motor clock wise motion
    motor_ccw    : out std_logic;       -- Motor counter clock wise motion
    -- Interface to seven segments
    abcdefgdec_n : out std_logic_vector(7 downto 0);
    a_n          : out std_logic_vector(3 downto 0)
    );
        end component;

component motor is
  port (
    motor_cw  : in  std_logic;
    motor_ccw : in  std_logic;
    a         : out std_logic;
    b         : out std_logic
    );
end component motor;


component seg7model is
  port
  (
    a_n           : in  std_logic_vector(3 downto 0);
    abcdefgdec_n  : in  std_logic_vector(7 downto 0);
    disp3         : out std_logic_vector(3 downto 0);
    disp2         : out std_logic_vector(3 downto 0);
    disp1         : out std_logic_vector(3 downto 0);
    disp0         : out std_logic_vector(3 downto 0)
  );
end component seg7model;

    signal refclk      :  std_logic;        -- Clock
    signal sync_rst  :  std_logic;          -- Synchronous reset
    signal sp2        :  signed(7 downto 0); -- Setpoint (wanted position)
    signal a          :  std_logic;          -- From position sensor
    signal b          :  std_logic;          -- From position sensor
    signal force_cw  :  std_logic;          -- Force motor clock wise motion
    signal force_ccw :  std_logic;                      -- Force motor counter clock wise motion
        signal arst : std_logic;
        signal abcdefgdec_n : std_logic_vector(7 downto 0);
        signal a_n          : std_logic_vector(3 downto 0);
        signal motor_ccw : std_logic;
        signal motor_cw : std_logic;

        begin

        O1: entity work.seg7model
        port map (
                a_n => a_n,
                abcdefgdec_n => abcdefgdec_n,
                disp0 => open,
                disp1 => open,
                disp2 => open,
                disp3 => open
        );
        O2: entity work.motor
        port map (
                motor_cw => motor_cw,
                motor_ccw => motor_ccw,
                a => a,
                b => b
        );


        O: entity work.pos_seg7_ctrl
    port map (sync_rst => sync_rst,
                        arst => arst,
                        refclk => refclk,
                        sp => std_logic_vector(sp2),
                        force_cw => force_cw,
                        force_ccw => force_ccw,
                        motor_cw => motor_cw,
                        motor_ccw => motor_ccw,
          --mclk => mclk,
          a => a,
          b => b,
                        --rst => rst,
                        a_n => a_n,
                        abcdefgdec_n => abcdefgdec_n
          );

          force_ccw <= '0';
          force_cw <= '0';
          arst <= '1', '0' after 100 ns;
          sync_rst <= '1', '0' after 100 ns;

          P_CLK_0: process
  begin
    refclk <= '0';
          --mclk_div <= '0';
    wait for 50 ns;
    refclk <= '1';
          --mclk_div <= '1';
    wait for 50 ns;
  end process P_CLK_0;



    sp2 <= "00001000",
          "00000000" after 1000 us,
          "00000011" after 2000 us,
          "00001000" after 3000 us,
          "00100000" after 4000 us,
          "00110011" after 5000 us;



end architecture tb_pos_seg7;
```

CLKDIV_ENT.VHD

```
library ieee;
```

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

entity clkdiv is
 port (
          rst      : in std_logic; -- Restet
              mclk     : in std_logic; -- Master clock
              mclk_div : out std_logic-- Master clock div. by 128
     );

end clkdiv;
```

CLKDIV_RTL.VHD

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

architecture dav of clkdiv is
  signal mclk_cnt : unsigned(6 downto 0);
 begin

   P_CLKDIV: process(rst, mclk)
   begin
     if rst = '1' then
               mclk_cnt <= (others => '0');
            elsif rising_edge(mclk) then
      mclk_cnt <= mclk_cnt + 1;
    end if;
   end process P_CLKDIV;

 mclk_div <= std_logic(mclk_cnt(6));

end dav;
```

CRU_ENT

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

architecture dav of clkdiv is
  signal mclk_cnt : unsigned(6 downto 0);
 begin
```

```
  P_CLKDIV: process(rst, mclk)
  begin
    if rst = '1' then
            mclk_cnt <= (others => '0');
         elsif rising_edge(mclk) then
     mclk_cnt <= mclk_cnt + 1;
   end if;
  end process P_CLKDIV;

 mclk_div <= std_logic(mclk_cnt(6));

end dav;
```

CRU_RTL.VHD

```
library IEEE;
use IEEE.std_logic_1164.all;
library unisim;
use unisim.all;

architecture beh of cru is
  component bufg
   port(
            i : in std_logic;
            o : out std_logic
          );
  end component;

  component rstsynch is
   port (
            arst        : in std_logic;  --Asynch. restet
            mclk        : in std_logic;  -- Master clock
            mclk_div    : in std_logic;  -- Master clock div. by 128
            rst         : out std_logic; -- Sync. reset div. by 128
            rst_div     : out std_logic  -- Sync. reset div. by 128
         );

  end component rstsynch;

  component clkdiv is
   port (
            rst         : in std_logic;  -- Reset
            mclk        : in std_logic;  -- Master clock
            mclk_div    : out std_logic  -- Master clock div. by 128
         );
  end component clkdiv;

  signal rst_i, rst_local, rst_div_local, rst_div_i : std_logic;
  signal mclk_i, mclk_div_local, mclk_div_i : std_logic;

begin

   bufg_0: bufg
           port map (
            i => refclk,
            o => mclk_i
            );
         rstsynch_0: rstsynch
    port map (
            arst     => arst,         --[in] Asynch. reset
                    mclk     => mclk_i,       --[in] Master clock
                    mclk_div => mclk_div_i,   --[in] Master clock div. by 128
                    rst      => rst_local,    --[out] Synch. reset master clock
                    rst_div  => rst_div_local --[out] Synch. reset mclk div. by 128
                  );

         bufg_1: bufg
            port map (
            i => rst_local,
                    o => rst_i
            );
         bufg_2: bufg
            port map (
            i => rst_div_local,
                    o => rst_div_i
            );

         clkdiv_0: clkdiv
            port map (
                  rst      => rst_i,         --[in] Reset
                    mclk     => mclk_i,         --[in] Master clock
                    mclk_div => mclk_div_local --[out] Master clock div. by 128
            );
         bufg_3: bufg
            port map (
            i => mclk_div_local,
                        o => mclk_div_i
              );
             rst       <= rst_i;
    rst_div   <= rst_div_i;
    mclk      <= mclk_i;
    mclk_div  <= mclk_div_i;


end architecture beh;
```

MOTOR_BEH.VHD

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

architecture motor_beh of motor is

begin

  motor_moving : process
  begin

    a <= '0';
    b <= '0';
    loop
      if motor_cw = '1' and motor_ccw = '0' then
        a <= '0';
        wait for phase90;
        b <= '1';
        wait for phase90;
        a <= '1';
        wait for phase90;
        b <= '0';
      elsif motor_ccw = '1' and motor_cw = '0' then
        a <= '1';
        wait for phase90;
        b <= '1';
        wait for phase90;
        a <= '0';
        wait for phase90;
        b <= '0';
      end if;
      wait for phase90;
    end loop;
  end process;

end architecture motor_beh;
```

MOTOR_ENT

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

entity motor is
  generic (
    phase90 : time := 50 us
    );
  port (
    motor_cw  : in  std_logic;
    motor_ccw : in  std_logic;
    a         : out std_logic;
    b         : out std_logic
    );
end motor;
```

P_CTRL

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


        --rst       : in  std_logic;           -- Reset
    --clk       : in  std_logic;           -- Clock
    --sp        : in  signed(7 downto 0);  -- Set Point
    --pos       : in  signed(7 downto 0);  -- Measured position
    --motor_cw  : out std_logic;           --Motor Clock Wise direction
    --motor_ccw : out std_logic            --Motor Counter Clock Wise direction

architecture pos_ctrl of p_ctrl is
        TYPE State_type IS (idle_st, sampel_st, motor_st);
        Signal State : State_type;


begin

-- read pos from pos_meas_beh
-- make motor go to pos (SP) setpoint
```

```
        process (clk, rst, sp, pos) is
        variable err : signed(7 downto 0);
        begin
        if rst = '1' then
                motor_cw <= '0';
                motor_ccw <= '0';
                State <= idle_st;
        elsif rising_edge(clk) then

        case state is
                when idle_st =>
                        motor_ccw <= '0';
                        motor_cw <= '0';
                        State <= sampel_st;
                when sampel_st =>
                        err := sp - pos;
                        State <= motor_st;
                when motor_st =>
                        if err > 0 then
                                motor_cw <= '1';
                                motor_ccw <= '0';
                                State <= sampel_st;
                        elsif err < 0 then
                                motor_cw <= '0';
                                motor_ccw <= '1';
                                State <= sampel_st;
                        else
                                State <= idle_st;
                        end if;
                end case;
        end if;

        end process;



end architecture pos_ctrl;
```

P_CTRL_ENT

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

```
entity p_ctrl is
  port (
    -- System Clock and Reset
    rst       : in  std_logic;           -- Reset
    clk       : in  std_logic;           -- Clock
    sp        : in  signed(7 downto 0);  -- Set Point
    pos       : in  signed(7 downto 0);  -- Measured position
    motor_cw  : out std_logic;           --Motor Clock Wise direction
    motor_ccw : out std_logic            --Motor Counter Clock Wise direction
    );
end p_ctrl;
```

POS_CTRL

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

architecture pos_ctrl of pos_ctrl is

  signal cw : std_logic;
  signal ccw : std_logic;

 component p_ctrl is
    port (
         clk : in std_logic;
         rst : in std_logic;
         sp : in signed(7 downto 0);
         pos : in signed(7 downto 0);
         cw  : out std_logic;           --Motor Clock Wise direction
         ccw : out std_logic
         );
  end component p_ctrl;

  component pos_meas_beh is
    port (a : in std_logic;
         b : in std_logic;
         sync_rst : in std_logic;
         clk : in std_logic;
         rst : in std_logic;
         pos : out signed(7 downto 0));
  end component pos_meas_beh;
```

```
          signal sp1 : signed(7 downto 0 );
          signal postemp : signed(7 downto 0);

begin

--mask away sp bit
sp1 <= '0' & sp(6 downto 0);


          Q: entity work.pos_meas
   port map (sync_rst => sync_rst,
          clk => mclk,
          a => a,
          b => b,
                              rst => rst,
          pos => postemp);

          Q1: entity work.p_ctrl
   port map (clk => mclk_div,   -- implement its own clock
                              motor_cw => cw,
                              motor_ccw => ccw,
                              rst => rst_div,
                              sp => sp1,
          pos => postemp);


          process (mclk, rst)
          begin
          -- forced running of motor.
          if force_cw = '1' then
                    if force_ccw = '1' then
                              motor_cw <= cw; -- from p_ctrl
                              motor_ccw <= ccw;
                    else
                              motor_cw <= '1';
                              motor_ccw <= '0';
                    end if;
          else
                    if force_ccw = '0' then
                              motor_cw <= cw; -- from p_ctrl
                              motor_ccw <= ccw;
                    else
                              motor_cw <= '0';
                              motor_ccw <= '1';
                        end if;
                  end if;
                  pos <= postemp;
                  end process;


end architecture pos_ctrl;
```

POS_CTRL_ENT

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

entity pos_ctrl is
  port (
    -- System Clock and Reset
    rst       : in  std_logic;          -- Reset
    rst_div   : in  std_logic;          -- Reset
    mclk      : in  std_logic;          -- Clock
    mclk_div  : in  std_logic;          -- Clock to p_reg
    sync_rst  : in  std_logic;          -- Synchronous reset
    sp        : in  signed(7 downto 0);  -- Setpoint (wanted position)
    a         : in  std_logic;          -- From position sensor
    b         : in  std_logic;          -- From position sensor
    pos       : out signed(7 downto 0);  -- Measured Position
    force_cw  : in  std_logic;          -- Force motor clock wise motion
    force_ccw : in  std_logic;  -- Force motor counter clock wise motion
    motor_cw  : out std_logic;          -- Motor clock wise motion
    motor_ccw : out std_logic           -- Motor counter clock wise motion
    );
end pos_ctrl;
```

POS_MEAS_BEH

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


architecture pos_beh of pos_meas is

TYPE State_type IS (start_up_st, wait_a1_st, wait_a0_st, up_down_st, count_up_st,
count_down_st);
```

```
        Signal State : State_type;

begin

        process(a, b, sync_rst, clk)
                variable posi : signed(7 downto 0) := "00000000";
                variable counter : unsigned(1 downto 0) := "00";
        begin

        if rst = '1' then
                posi := "00000000";
                State <= start_up_st;

        elsif rising_edge(clk) then
        if sync_rst = '1' then
                posi := "00000000";
                State <= start_up_st;
        end if;

        case state is
                when start_up_st =>
                        if a = '1' then
                                State <= wait_a0_st;
                        else
                                State <= wait_a1_st;
                        end if;

                when wait_a0_st =>
                        if a='1' then
                                State <= wait_a0_st;
                        else
                                State <= up_down_st;
                        end if;
                when wait_a1_st =>
                        if a='1' then
                                State <= wait_a0_st;
                        else
                                State <= wait_a1_st;
                        end if;
                when up_down_st =>
                        if b='1' then
                                State <= count_down_st;
                        else
                        State <= count_up_st;
                    end if;
                when count_down_st =>
                            if posi > 0 then
                                posi := posi - 1;
                            end if;
                    State <= wait_a1_st;
                when count_up_st =>

                        if posi < 127 then
                                posi := posi + 1;
                            end if;
                    State <= wait_a1_st;


                end case;
            end if;
            pos <= posi;
            end process;

end architecture pos_beh;
```

POS_MEAS_ENT

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity pos_meas is
  port (
    -- System Clock and Reset
    rst      : in  std_logic;          -- Reset
    clk      : in  std_logic;          -- Clock
    sync_rst : in  std_logic;          -- Sync reset
    a        : in  std_logic;          -- From position sensor
    b        : in  std_logic;          -- From position sensor
    pos      : out signed(7 downto 0)   -- Measured position
    );
end pos_meas;
```

Oblig 4:

TASK1

COMPUTE_PIPE_RTL

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

architecture rtl of compute_pipelined is
  signal addresult_i  : unsigned(17 downto 0) := (others => '0');
        signal e_reg : std_logic_vector(15 downto 0) := (others => '0');
        signal dvalid_r : std_logic := '0';

begin

 process (rst, clk) is
  variable multresult_i : unsigned(33 downto 0) := (others => '0');

 begin
  if rst = '1' then
   result <= (others => '0');
   max    <= '0';
   rvalid <= '0';
         addresult_i <= (others => '0');
         e_reg <= (others => '0');
         dvalid_r <= '0';
         --multresult_i := (others => '0');
  elsif rising_edge(clk) then
        --rvalid <= '0'; -- prøver å sync
   if (dvalid = '1' or dvalid_r = '1') then
          --multresult_i := addresult_i * unsigned(e);
    addresult_i <= (unsigned("00" & a) + unsigned("00" & b)) +
                (unsigned("00" & c) + unsigned("00" & d));
                e_reg <= e;

    multresult_i := addresult_i * unsigned(e_reg);
    if (multresult_i(33 downto 32) = "00") then
     result <= std_logic_vector(multresult_i(31 downto 0));
     max    <= '0';
    else
                 result <= (others => '1');
     max    <= '1';
```

```
        end if;
      else
            result <= (others => '0');
        max    <= '0';
      end if;
          dvalid_r <= dvalid;
          rvalid <= dvalid_r;


    end if;
  end process;

end architecture rtl;
```

COMPUTE_ENT

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

entity compute is
  port
    (rst    : in  std_logic;
     clk    : in  std_logic;
     a      : in  std_logic_vector(15 downto 0);
     b      : in  std_logic_vector(15 downto 0);
     c      : in  std_logic_vector(15 downto 0);
     d      : in  std_logic_vector(15 downto 0);
     e      : in  std_logic_vector(15 downto 0);
     dvalid : in  std_logic;
     result : out std_logic_vector(31 downto 0);
     max    : out std_logic;
     rvalid : out std_logic);
end entity compute;
```

TB_COMPUTE_PIPELINED

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

entity tb_compute_pipelined is
```

```
  -- empty;
end tb_compute_pipelined;

architecture beh of tb_compute_pipelined is

  component compute_pipelined
    port (
      rst    : in  std_logic;
      clk    : in  std_logic;
      a      : in  std_logic_vector(15 downto 0);
      b      : in  std_logic_vector(15 downto 0);
      c      : in  std_logic_vector(15 downto 0);
      d      : in  std_logic_vector(15 downto 0);
      e      : in  std_logic_vector(15 downto 0);
      dvalid : in  std_logic;
      result : out std_logic_vector(31 downto 0);
      max    : out std_logic;
      rvalid : out std_logic);
  end component;

  signal rst    : std_logic;
  signal clk    : std_logic:= '0';
  signal a      : std_logic_vector(15 downto 0);
  signal b      : std_logic_vector(15 downto 0);
  signal c      : std_logic_vector(15 downto 0);
  signal d      : std_logic_vector(15 downto 0);
  signal e      : std_logic_vector(15 downto 0);
  signal dvalid : std_logic;
  signal result : std_logic_vector(31 downto 0);
  signal max    : std_logic;
  signal rvalid : std_logic;

begin

  compute_pipelined_0: entity work.compute_pipelined
    port map (
      rst    => rst,
      clk    => clk,
      a      => a,
      b      => b,
      c      => c,
      d      => d,
      e      => e,
      dvalid => dvalid,
      result => result,
      max    => max,
      rvalid => rvalid);

  -- Clock and reset generation
  clk <= not clk after 10 ns ;
  rst <= '1', '0' after 20 ns;

  P_TEST: process
  begin

    a <= (others => '0');
    b <= (others => '0');
    c <= (others => '0');
    d <= (others => '0');
    e <= (others => '0');
    dvalid <= '0';

    wait until falling_edge(rst);
    wait for 40 ns;

    wait until falling_edge(clk);

    a <= x"0001";
    b <= x"0002";
    c <= x"0003";
    d <= x"0004";
    e <= x"0005";
    dvalid <= '1';

    wait on clk until rvalid='1';
    assert (result=x"00000032" and max='0')
      report "Result not equal 32 hex and max not equal 0" severity FAILURE;

    a <= (others => '0');
    b <= (others => '0');
    c <= (others => '0');
    d <= (others => '0');
    e <= (others => '0');
    dvalid <= '0';

    wait on clk until rvalid='0';
```

```
assert (result=x"00000000" and max='0')
  report "Result not equal zero and max not equal 0" severity FAILURE;

a <= x"0105";
b <= x"0206";
c <= x"0307";
d <= x"0408";
e <= x"0509";
dvalid <= '1'; -- endre til 1

wait for 20 ns; --10ns clk updates

a <= x"FFFF";
b <= x"0000";
c <= x"0001";
d <= x"0000";
e <= x"0001";
dvalid <= '1';

wait on clk until rvalid='1';
assert (result=x"0032DCEA" and max='0')
  report "Result not equal 0032DCEA hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"FFFF";
c <= x"FFFF";
d <= x"FFFF";
e <= x"0001";
dvalid <= '1';

wait for 20 ns;
assert (result=x"00010000" and max='0')
  report "Result not equal 00010000 hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"0000";
c <= x"0001";
d <= x"0000";
e <= x"FFFE";
dvalid <= '1';

wait for 20 ns;
assert (result=x"0003FFFC" and max='0')
```

```
  report "Result not equal 0003FFFC hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"0000";
c <= x"0001";
d <= x"0000";
e <= x"FFFF";
dvalid <= '1';

wait for 20 ns;
assert (result=x"FFFE0000" and max='0')
  report "Result not equal FFFE0000 hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"0000";
c <= x"FFFF";
d <= x"0000";
e <= x"FFFF";
dvalid <= '1';

wait for 20 ns;
assert (result=x"FFFF0000" and max='0')
  report "Result not equal FFFF0000 hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"0000";
c <= x"FFFF";
d <= x"0000";
e <= x"7FFF";
dvalid <= '1';

wait for 20 ns;
assert (result=x"FFFFFFFF" and max='1')
  report "Result not equal FFFFFFFF hex and max not equal 1" severity FAILURE;

a <= (others => '0');
b <= (others => '0');
c <= (others => '0');
d <= (others => '0');
e <= (others => '0');
dvalid <= '0';

wait for 20 ns;
```

```
    assert (result=x"FFFD0002" and max='0')
      report "Result not equal FFFD0002 hex and max not equal 0" severity FAILURE;

    wait on clk until rvalid='0';
    assert (result=x"00000000" and max='0')
      report "Result not equal zero and max not equal 0" severity FAILURE;

    assert (FALSE)
      report "Simulation complete." severity NOTE;

    wait;

  end process;

end beh;
```

TB_COMPUTE.VHD

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

entity tb_compute is
  -- empty;
end tb_compute;

architecture beh of tb_compute is

  component compute
    port (
      rst    : in  std_logic;
      clk    : in  std_logic;
      a      : in  std_logic_vector(15 downto 0);
      b      : in  std_logic_vector(15 downto 0);
      c      : in  std_logic_vector(15 downto 0);
      d      : in  std_logic_vector(15 downto 0);
      e      : in  std_logic_vector(15 downto 0);
      dvalid : in  std_logic;
      result : out std_logic_vector(31 downto 0);
      max    : out std_logic;
      rvalid : out std_logic);
  end component;

  signal rst    : std_logic;
  signal clk    : std_logic:= '0';
  signal a      : std_logic_vector(15 downto 0);
  signal b      : std_logic_vector(15 downto 0);
  signal c      : std_logic_vector(15 downto 0);
  signal d      : std_logic_vector(15 downto 0);
  signal e      : std_logic_vector(15 downto 0);
  signal dvalid : std_logic;
  signal result : std_logic_vector(31 downto 0);
  signal max    : std_logic;
  signal rvalid : std_logic;

begin

  compute_0: compute
    port map (
      rst    => rst,
      clk    => clk,
      a      => a,
      b      => b,
      c      => c,
      d      => d,
      e      => e,
      dvalid => dvalid,
      result => result,
      max    => max,
      rvalid => rvalid);

  -- Clock and reset generation
  clk <= not clk after 10 ns ;
  rst <= '1', '0' after 20 ns;

  P_TEST: process
  begin

    a <= (others => '0');
    b <= (others => '0');
    c <= (others => '0');
    d <= (others => '0');
    e <= (others => '0');
    dvalid <= '0';
```

```
wait until falling_edge(rst);
wait for 40 ns;

wait until falling_edge(clk);

a <= x"0001";
b <= x"0002";
c <= x"0003";
d <= x"0004";
e <= x"0005";
dvalid <= '1';

wait on clk until rvalid='1';
assert (result=x"00000032" and max='0')
  report "Result not equal 32 hex and max not equal 0" severity FAILURE;

a <= (others => '0');
b <= (others => '0');
c <= (others => '0');
d <= (others => '0');
e <= (others => '0');
dvalid <= '0';

wait on clk until rvalid='0';
assert (result=x"00000000" and max='0')
  report "Result not equal zero and max not equal 0" severity FAILURE;

a <= x"0105";
b <= x"0206";
c <= x"0307";
d <= x"0408";
e <= x"0509";
dvalid <= '1';

wait on clk until rvalid='1';
assert (result=x"0032DCEA" and max='0')
  report "Result not equal 0032DCEA hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"0000";
c <= x"0001";
d <= x"0000";
```

```
e <= x"0001";
dvalid <= '1';

wait for 20 ns;
assert (result=x"00010000" and max='0')
  report "Result not equal 00010000 hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"FFFF";
c <= x"FFFF";
d <= x"FFFF";
e <= x"0001";
dvalid <= '1';

wait for 20 ns;
assert (result=x"0003FFFC" and max='0')
  report "Result not equal 0003FFFC hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"0000";
c <= x"0001";
d <= x"0000";
e <= x"FFFE";
dvalid <= '1';

wait for 20 ns;
assert (result=x"FFFE0000" and max='0')
  report "Result not equal FFFE0000 hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"0000";
c <= x"0001";
d <= x"0000";
e <= x"FFFF";
dvalid <= '1';

wait for 20 ns;
assert (result=x"FFFF0000" and max='0')
  report "Result not equal FFFF0000 hex and max not equal 0" severity FAILURE;

a <= x"FFFF";
b <= x"0000";
c <= x"FFFF";
```

```
    d <= x"0000";
    e <= x"FFFF";
    dvalid <= '1';

    wait for 20 ns;
    assert (result=x"FFFFFFFF" and max='1')
      report "Result not equal FFFFFFFF hex and max not equal 1" severity FAILURE;

    a <= x"FFFF";
    b <= x"0000";
    c <= x"FFFF";
    d <= x"0000";
    e <= x"7FFF";
    dvalid <= '1';

    wait for 20 ns;
    assert (result=x"FFFD0002" and max='0')
      report "Result not equal FFFD0002 hex and max not equal 0" severity FAILURE;

    a <= (others => '0');
    b <= (others => '0');
    c <= (others => '0');
    d <= (others => '0');
    e <= (others => '0');
    dvalid <= '0';

    wait on clk until rvalid='0';
    assert (result=x"00000000" and max='0')
      report "Result not equal zero and max not equal 0" severity FAILURE;

    assert (FALSE)
      report "Simulation complete." severity NOTE;

    wait;

  end process;

end beh;
```

COMPUTE_RTL

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

```
architecture rtl of compute is
begin

  process (rst, clk) is
    variable addresult_i  : unsigned(17 downto 0);
    variable multresult_i : unsigned(33 downto 0);
  begin
    if rst = '1' then
      result <= (others => '0');
      max    <= '0';
      rvalid <= '0';
    elsif rising_edge(clk) then
      if (dvalid = '1') then
        addresult_i := (unsigned("00" & a) + unsigned("00" & b)) +
                       (unsigned("00" & c) + unsigned("00" & d));
        multresult_i := addresult_i * unsigned(e);
        if (multresult_i(33 downto 32) = "00") then
          result <= std_logic_vector(multresult_i(31 downto 0));
          max    <= '0';
        else
          result <= (others => '1');
          max    <= '1';
        end if;
      else
        result <= (others => '0');
        max    <= '0';
      end if;
      rvalid <= dvalid;
    end if;
  end process;

end architecture rtl;
```

COMPUTE_PIPELINED_ENT

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

entity compute_pipelined is
  port
    (rst    : in  std_logic;
     clk    : in  std_logic;
     a      : in  std_logic_vector(15 downto 0);
     b      : in  std_logic_vector(15 downto 0);
     c      : in  std_logic_vector(15 downto 0);
     d      : in  std_logic_vector(15 downto 0);
     e      : in  std_logic_vector(15 downto 0);
     dvalid : in  std_logic;
     result : out std_logic_vector(31 downto 0);
     max    : out std_logic;
     rvalid : out std_logic);
end entity;
```

TASK2

RAM_LAB4_RTL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity ram_lab4 is
  Port (
    clka : in STD_LOGIC;
    rsta : in STD_LOGIC;
    ena : in STD_LOGIC;
    wea : in STD_LOGIC_VECTOR ( 3 downto 0 );
    addra : in STD_LOGIC_VECTOR ( 9 downto 0 );
    dina : in STD_LOGIC_VECTOR ( 31 downto 0 );
    douta : out STD_LOGIC_VECTOR ( 31 downto 0 )
  );

end ram_lab4;

architecture rtl of ram_lab4 is
begin

  memory: process (rsta, clka) is
    type memory_array is array (0 to 1023) of std_logic_vector(31 downto 0);
    variable lab4_memory : memory_array;
  begin
    if rsta='1' then
      lab4_memory := (others => x"DEADBEEF");
      douta <= (others => '0');
    elsif rising_edge(clka) then
      if ena='1' then
        if wea /= "0000" then
          if wea(0)='1' then
            lab4_memory(to_integer(unsigned(addra)))(7 downto 0) := dina(7 downto 0);
          end if;
          if wea(1)='1' then
            lab4_memory(to_integer(unsigned(addra)))(15 downto 8) := dina(15 downto 8);
          end if;
          if wea(2)='1' then
            lab4_memory(to_integer(unsigned(addra)))(23 downto 16) := dina(23 downto 16);
          end if;
          if wea(3)='1' then
            lab4_memory(to_integer(unsigned(addra)))(31 downto 24) := dina(31 downto 24);
          end if;
        else
          douta <= lab4_memory(to_integer(unsigned(addra)))(31 downto 0);
        end if;
      end if;
    end if;
  end process memory;
end;
```

LAB4_TB_BDY

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

use std.textio.all;

package body lab4_tb_pck is

  constant T_100M           : time := 10 ns;        -- 100MHz clock period
  constant T_OFFSET_100M    : time := 0 ns;         -- time of positive edge clk i.r.t begining of
cycle
  constant T_HIGH_100M      : time := 0.5 * T_100M; -- duration of clock being high

  constant T_125M           : time := 8 ns;         -- 125MHz clock period
  constant T_OFFSET_125M    : time := 4 ns;         -- time of positive edge clk i.r.t begining of
cycle
  constant T_HIGH_125M      : time := 0.5 * T_125M; -- duration of clock being high
```

```
-- wait n T cycles
procedure Tcycle(n : natural) is
begin
  wait for n * T_100M;  -- wait n cycles
end Tcycle;

-- perform chip reset
procedure Reset (
  signal rst_n : out std_logic;
  file log : text;
  constant cycle : in integer) is
begin
  writef(log, cycle,"Reset Performed");
  rst_n <= '0';
  Tcycle(10); -- wait 10 cycles
  rst_n <= '1';
end Reset;


procedure AXI4LiteWrite (
  constant tsize           : in  transaction_size;
  constant addr            : in  std_logic_vector(31 downto 0);
  constant data            : in  std_logic_vector;

  -- AXI4Lite clock
  signal  s00_axi_aclk      : in  std_logic;

  -- Write Address Channel
  signal  s00_axi_awaddr    : out std_logic_vector(31 downto 0);
  signal  s00_axi_awprot    : out std_logic_vector(2 downto 0);
  signal  s00_axi_awvalid   : out std_logic;
  signal  s00_axi_awready   : in  std_logic;

  -- Write Data Channel
  signal  s00_axi_wdata     : out std_logic_vector(31 downto 0);
  signal  s00_axi_wstrb     : out std_logic_vector(3 downto 0);
  signal  s00_axi_wvalid    : out std_logic;
  signal  s00_axi_wready    : in  std_logic;

  -- Write Respons Channel
  signal  s00_axi_bresp     : in  std_logic_vector(1 downto 0);
  signal  s00_axi_bvalid    : in  std_logic;
  signal  s00_axi_bready    : out std_logic;

  file     log              :       text;
  constant cycle            : in    natural;
  variable error_found      : out   boolean;
  signal   error_no         : inout natural) is
    variable data_slv : std_logic_vector(31 downto 0);
    variable s00_axi_bvalid_eq1 : boolean;
begin
  error_found:= FALSE;
  s00_axi_bvalid_eq1 := FALSE;

  wait until rising_edge(s00_axi_aclk);

  if tsize=BYTE and data'length=8 then
    data_slv(7 downto 0)   := data;
    data_slv(15 downto 8)  := data;
    data_slv(23 downto 16) := data;
    data_slv(31 downto 24) := data;
  elsif tsize=HALFWORD and data'length=16 then
    data_slv(15 downto 0)  := data;
    data_slv(31 downto 16) := data;
  elsif tsize=SINGLE and data'length=32 then
    data_slv := data;
  else
    error_no <= error_no + 1;
    error_found:= TRUE;
    writef(log, cycle,"AXI4Lite write access aborted: Illegal data length");
    return;
  end if;

  if (tsize=BYTE and addr(1 downto 0)="00") then
    s00_axi_wstrb<= force "0001";
    writef(log, cycle,"AXI4Lite byte 0 write to AXI4PIFB              " & " @ " &
         lv2strx(addr,32) & " <= " & lv2strx(data_slv(7 downto 0),8));
  elsif (tsize=BYTE and addr(1 downto 0)="01") then
    s00_axi_wstrb<= force "0010";
    writef(log, cycle,"AXI4Lite byte 1 write to AXI4PIFB              " & " @ " &
         lv2strx(addr,32) & " <= " & lv2strx(data_slv(15 downto 8),8));
  elsif (tsize=BYTE and addr(1 downto 0)="10") then
    s00_axi_wstrb<= force "0100";
    writef(log, cycle,"AXI4Lite byte 2 write to AXI4PIFB              " & " @ " &
         lv2strx(addr,32) & " <= " & lv2strx(data_slv(23 downto 16),8));
```

```
elsif (tsize=BYTE and addr(1 downto 0)="11") then
  s00_axi_wstrb<= force "1000";
  writef(log, cycle,"AXI4Lite byte 3 write to AXI4PIFB               " & " @ " &
      lv2strx(addr,32) & " <= " & lv2strx(data_slv(31 downto 24),8));
elsif (tsize=HALFWORD and addr(1 downto 0)="00") then
  s00_axi_wstrb <= force "0011";
  writef(log, cycle,"AXI4Lite halfword lower bytes write to AXI4PIFB " & " @ " &
      lv2strx(addr,32) & " <= " & lv2strx(data_slv(15 downto 0),16));
elsif (tsize=HALFWORD and addr(1 downto 0)="10") then
  s00_axi_wstrb<= force "1100";
  writef(log, cycle,"AXI4Lite halfword upper bytes write to AXI4PIFB " & " @ " &
      lv2strx(addr,32) & " <= " & lv2strx(data_slv(31 downto 16),16));
elsif (tsize=SINGLE and addr(1 downto 0)="00") then
  s00_axi_wstrb<= force "1111";
  writef(log, cycle,"AXI4Lite single word write to AXI4PIFB          " & " @ " &
      lv2strx(addr,32) & " <= " & lv2strx(data_slv,32));
else
  error_no <= error_no + 1;
  error_found:= TRUE;
  writef(log, cycle,"AXI4Lite write access aborted: Unaligned register address.");
  return;
end if;

s00_axi_awaddr <= force addr;
s00_axi_awprot <= force "000";
s00_axi_awvalid<= force '1';
s00_axi_wdata  <= force data_slv;
s00_axi_wvalid <= force '1';
s00_axi_bready <= force '1';

wait until rising_edge(s00_axi_awready) for 2 us;

if s00_axi_wready='1' then
  if s00_axi_bvalid='1' then
    s00_axi_bvalid_eq1:= TRUE;
  end if;
  wait until rising_edge(s00_axi_aclk);
  s00_axi_awaddr <= force (others => '0');
  s00_axi_awprot <= force "000";
  s00_axi_awvalid<= force '0';
  s00_axi_wdata  <= force (others => '0');
  s00_axi_wvalid <= force '0';
else
```

```
    wait until rising_edge(s00_axi_wready) for 2 us;
    if s00_axi_bvalid='1' then
      s00_axi_bvalid_eq1:= TRUE;
    end if;
    s00_axi_awaddr <= force (others => '0');
    s00_axi_awprot <= force "000";
    s00_axi_awvalid<= force '0';
    wait until rising_edge(s00_axi_aclk);
    s00_axi_wdata  <= force (others => '0');
    s00_axi_wvalid <= force '0';
  end if;

  if s00_axi_bvalid='0' and not s00_axi_bvalid_eq1 then
    wait until rising_edge(s00_axi_bvalid) for 2 us;
  end if;

  wait until falling_edge(s00_axi_aclk);

  if (s00_axi_bvalid/='1' and (not s00_axi_bvalid_eq1)) or s00_axi_bresp/="00" then
    error_no <= error_no + 1;
    error_found:= TRUE;
    writef(log, cycle,"AXI4Lite write access aborted: Write access timeout.");
  end if;

  wait until rising_edge(s00_axi_aclk);
  s00_axi_bready<= force '0';

  wait until rising_edge(s00_axi_aclk);

end ;


procedure AXI4LiteCheck (
  constant tsize           : in  transaction_size;
  constant addr             : in  std_logic_vector(31 downto 0);
  variable data            : in  std_logic_vector;

  -- AXI4Lite clock
  signal  s00_axi_aclk      : in  std_logic;

  -- Read Address Channel
  signal  s00_axi_araddr    : out std_logic_vector(31 downto 0);
  signal  s00_axi_arprot    : out std_logic_vector(2 downto 0);
  signal  s00_axi_arvalid   : out std_logic;
  signal  s00_axi_arready   : in  std_logic;

  -- Read Data Channel
  signal  s00_axi_rdata     : in  std_logic_vector(31 downto 0);
  signal  s00_axi_rresp     : in  std_logic_vector(1 downto 0);
  signal  s00_axi_rvalid    : in  std_logic;
  signal  s00_axi_rready    : out std_logic;

  file     log             :       text;
  constant cycle           : in    natural;
  variable error_found      : out   boolean;
  signal   error_no         : inout natural) is
    variable rdata          : std_logic_vector(data'length-1 downto 0); -- data read
    variable error_found_i  : boolean;
    variable error_no_i     : natural;

begin

  if ((tsize=BYTE and data'length/=8) or
      (tsize=HALFWORD and data'length/=16) or
      (tsize=SINGLE and data'length/=32)) then
    error_no <= error_no + 1;
    error_found:= TRUE;
    writef(log, cycle,"AXI4Lite check access aborted: Illegal data length");
    return;
  end if;

  AXI4LiteRead(tsize           => tsize,
               addr            => addr,
               data            => rdata,
               s00_axi_aclk     => s00_axi_aclk,
               s00_axi_araddr   => s00_axi_araddr,
               s00_axi_arprot   => s00_axi_arprot,
               s00_axi_arvalid  => s00_axi_arvalid,
               s00_axi_arready  => s00_axi_arready,
               s00_axi_rdata    => s00_axi_rdata,
               s00_axi_rresp    => s00_axi_rresp,
               s00_axi_rvalid   => s00_axi_rvalid,
               s00_axi_rready   => s00_axi_rready,
               log             => log,
               cycle           => cycle,
               error_found      => error_found_i,
               error_no        => error_no_i);

  if not error_found_i then
    if (tsize=BYTE and addr(1 downto 0)="00" and rdata /= data) then
      writef(log, cycle,"  ERROR! byte 0 data read: " & lv2strx(rdata,8) &
               ", expected : " & lv2strx(data,8));
      error_no    <= error_no + 1;
      error_found := TRUE;
    elsif (tsize=BYTE and addr(1 downto 0)="01" and rdata /= data) then
      writef(log, cycle,"  ERROR! byte 1 data read: " & lv2strx(rdata,8) &
           ", expected : " & lv2strx(data,8));
      error_no    <= error_no + 1;
      error_found := TRUE;
    elsif (tsize=BYTE and addr(1 downto 0)="10" and rdata /= data) then
      writef(log, cycle,"  ERROR! byte 2 data read: " & lv2strx(rdata,8) &
           ", expected : " & lv2strx(data,8));
      error_no    <= error_no + 1;
      error_found := TRUE;
    elsif (tsize=BYTE and addr(1 downto 0)="11" and rdata /= data) then
      writef(log, cycle,"  ERROR! byte 3 data read: " & lv2strx(rdata,8) &
           ", expected : " & lv2strx(data,8));
      error_no    <= error_no + 1;
      error_found := TRUE;
    elsif (tsize=HALFWORD and addr(1 downto 0)="00" and rdata /= data) then
      writef(log, cycle,"  ERROR! Halfword lower bytes data read: " & lv2strx(rdata,16) &
               ", expected : " & lv2strx(data,16));
      error_no    <= error_no + 1;
      error_found := TRUE;
    elsif (tsize=HALFWORD and addr(1 downto 0)="10" and rdata /= data) then
      writef(log, cycle,"  ERROR! Halfword upper bytes data read: " & lv2strx(rdata,16) &
               ", expected : " & lv2strx(data,16));
      error_no    <= error_no + 1;
      error_found := TRUE;
    elsif (tsize=SINGLE and addr(1 downto 0)="00" and rdata /= data) then
      writef(log, cycle,"  ERROR! Word data read: " & lv2strx(rdata,32) &
               ", expected : " & lv2strx(data,32));
      error_no    <= error_no + 1;
      error_found := TRUE;
    end if;
  else
      error_no    <= error_no + 1;
      error_found := TRUE;
  end if;
  end procedure AXI4LiteCheck;


  -- AXI4Lite Data Read
  procedure AXI4LiteRead (
    constant tsize            : in  transaction_size;
    constant addr             : in  std_logic_vector(31 downto 0);
    variable data             : out std_logic_vector;

    -- AXI4Lite clock
    signal  s00_axi_aclk      : in  std_logic;

    -- Read Address Channel
    signal  s00_axi_araddr    : out std_logic_vector(31 downto 0);
    signal  s00_axi_arprot    : out std_logic_vector(2 downto 0);
    signal  s00_axi_arvalid   : out std_logic;
    signal  s00_axi_arready   : in  std_logic;

    -- Read Data Channel
    signal  s00_axi_rdata     : in  std_logic_vector(31 downto 0);
    signal  s00_axi_rresp     : in  std_logic_vector(1 downto 0);
    signal  s00_axi_rvalid    : in  std_logic;
    signal  s00_axi_rready    : out std_logic;

    file     log              :       text;
    constant cycle            : in    natural;
    variable error_found      : out   boolean;
    variable error_no         : inout natural) is
  begin
    error_found:= FALSE;

    if ((tsize=BYTE and data'length/=8) or
        (tsize=HALFWORD and data'length/=16) or
        (tsize=SINGLE and data'length/=32)) then
      error_no := error_no + 1;
      error_found:= TRUE;
      writef(log, cycle,"AXI4Lite read access aborted: Illegal data length");
      return;
    end if;

    wait until rising_edge(s00_axi_aclk);
```

```
s00_axi_arvalid <= force '1';
s00_axi_araddr  <= force addr;
s00_axi_arprot  <= force "000";
s00_axi_rready  <= force '1';

wait until rising_edge(s00_axi_arready) for 2 us;

wait until rising_edge(s00_axi_aclk);
s00_axi_arvalid <= force '0';
s00_axi_araddr  <= force (others => '0');

if s00_axi_rvalid='0' then
 wait until rising_edge(s00_axi_rvalid) for 2 us;
end if;

wait until falling_edge(s00_axi_aclk);

if s00_axi_rvalid/='1' or s00_axi_rresp/="00" then
  data := x"00";
  error_no := error_no + 1;
  error_found:= TRUE;
  writef(log, cycle,"AXI4Lite read access aborted: Read access timeout.");
else

  if (tsize=BYTE and addr(1 downto 0)="00") then
    data := s00_axi_rdata(7 downto 0);
    writef(log, cycle,"AXI4Lite byte 0 read from AXI4PIFB             " & " @ " &
         lv2strx(addr,32) & " <= " & lv2strx(s00_axi_rdata(7 downto 0),8));
  elsif (tsize=BYTE and addr(1 downto 0)="01") then
    data := s00_axi_rdata(15 downto 8);
    writef(log, cycle,"AXI4Lite byte 1 read from AXI4PIFB             " & " @ " &
         lv2strx(addr,32) & " <= " & lv2strx(s00_axi_rdata(15 downto 8),8));
  elsif (tsize=BYTE and addr(1 downto 0)="10") then
    data := s00_axi_rdata(23 downto 16);
    writef(log, cycle,"AXI4Lite byte 2 read from AXI4PIFB             " & " @ " &
         lv2strx(addr,32) & " <= " & lv2strx(s00_axi_rdata(23 downto 16),8));
  elsif (tsize=BYTE and addr(1 downto 0)="11") then
    data := s00_axi_rdata(31 downto 24);
    writef(log, cycle,"AXI4Lite byte 3 read from AXI4PIFB             " & " @ " &
         lv2strx(addr,32) & " <= " & lv2strx(s00_axi_rdata(31 downto 24),8));
  elsif (tsize=HALFWORD and addr(1 downto 0)="00") then
    data := s00_axi_rdata(15 downto 0);
    writef(log, cycle,"AXI4Lite halfword lower bytes read from AXI4PIFB" & " @ " &
             lv2strx(addr,32) & " <= " & lv2strx(s00_axi_rdata(15 downto 0),16));
      elsif (tsize=HALFWORD and addr(1 downto 0)="10") then
        data := s00_axi_rdata(31 downto 16);
        writef(log, cycle,"AXI4Lite halfword upper bytes read from AXI4PIFB" & " @ " &
             lv2strx(addr,32) & " <= " & lv2strx(s00_axi_rdata(31 downto 16),16));
      elsif (tsize=SINGLE and addr(1 downto 0)="00") then
        data := s00_axi_rdata(31 downto 0);
        writef(log, cycle,"AXI4Lite single word read from AXI4PIFB         " & " @ " &
             lv2strx(addr,32) & " <= " & lv2strx(s00_axi_rdata,32));
      else
        error_no := error_no + 1;
        error_found:= TRUE;
        writef(log, cycle,"Illegal AXI4Lite read access: Unaligned register address.");
      end if;
    end if;

    wait until rising_edge(s00_axi_aclk);
    s00_axi_rready<= force '0';

    wait until rising_edge(s00_axi_aclk);

  end ;

end package body lab4_tb_pck;
```

BASE_PCK

```
use std.textio.all;

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

--library novalib;
--use novalib.nova_pck.all;

library std_developerskit;
use std_developerskit.std_iopak.all;

package base_pck is

  constant ADDRESS_LENGTH : natural:= 32;
constant DATA_LENGTH    : natural:= 32;

type sl_ptr is access std_logic;
type slv_ptr is access std_logic_vector;

function hex (data : std_logic_vector; n : integer) return std_logic_vector;
function lv2str (data : std_logic_vector) return string;
function lv2strx (data : std_logic_vector; n : integer) return string;
function strip (data : string) return string;
procedure writef (file log : text; constant cycle : in integer; msg : in string);
procedure writef (file log : text; constant cycle : in integer;
            msg : in string; constant error_no : in integer);
procedure writef (file log : text;
            msg : in string; constant error_no : in integer);
procedure writef (file log : text; msg : in string);
procedure clk_gen (signal clk : out std_logic;
                  signal cycle_no : inout natural;
                  signal run : in std_logic;
                  constant period : in time;
                  constant high   : in time;
                  constant offset : in time);
procedure clk_gen_changing (signal clk : out std_logic;
                  signal cycle_no : inout natural;
                  signal run : in std_logic;
                  signal period : in  time;
                  signal high   : in  time;
                  constant offset : in  time);
procedure clk_gen_np (signal clk_p : out std_logic;
                       signal clk_n : out std_logic;
                    signal cycle_no : inout natural;
                    signal run : in std_logic;
                    constant period : in  time;
                    constant high   : in  time;
                    constant offset : in  time);
procedure str_gen (signal str_n : out std_logic;
                  signal rst_n : in std_logic;
                  signal clk   : in std_logic;
                  signal run   : in std_logic;
                  constant str_time : in natural;
                  constant period_length : in natural);
procedure str_gen_n (signal str_n : out std_logic;
                   signal rst_n : in std_logic;
                   signal clk   : in std_logic;
```

```
                          signal run   : in std_logic;
                          constant str_time : in natural;
                          constant period_length : in natural);
  procedure str_gen_np (signal str_p : out std_logic;
           signal str_n : out std_logic;
                          signal rst_n : in std_logic;
                          signal clk   : in std_logic;
                          signal run   : in std_logic;
                          constant str_time : in natural;
                          constant period_length : in natural);
  procedure random_gen (signal rst_n       : in std_logic;
                            signal clk         : in std_logic;
                            signal run_random  : in std_logic;
                            signal random_data : out std_logic;
                            signal run         : in std_logic);
  procedure random_gen (constant WIDTH : in natural range 2 to 32;
       signal seed        : in std_logic_vector;
       signal loopmode    : in boolean;
       signal rst_n       : in std_logic;
                            signal clk         : in std_logic;
                            signal run_random  : in std_logic;
                            signal random_data : out std_logic;
                            signal run         : in std_logic);
  function power(size : natural) return natural;

----------------------------------------------
-- This procedure removes leading spaces
----------------------------------------------
  procedure rm_space (l : inout line);

---------------------------------------------------------
--  This procedure read a word from line and returns a
--    string containing the word until first ' ', NBSP or HT
--  The actual length of the word is returned and
--    the string word is converted to lower.
---------------------------------------------------------
  procedure readword ( l        : inout line;
                               l_width  : out natural;
                               word     : out string);

-------------------------------------------------------------
-- This procedure returns the length of the line; max 40
-------------------------------------------------------------
```

```
  procedure string_length(l : inout line; value : out natural);

-------------------------------------------------------------
-- This procedure removes char up to the specified character
-------------------------------------------------------------
  procedure find_char(l : inout line;
                      constant char : character);

end package base_pck;
```

BASE_BDY

```
use std.textio.all;

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

library std_developerskit;
use std_developerskit.std_iopak.all;

package body base_pck is

  -- convert hex of type x"AF" to shorter vectors
  function hex (data : std_logic_vector;
                n : integer)
    return std_logic_vector is
    variable ptr : slv_ptr;
    variable local_data : std_logic_vector(data'range) := data;
  begin
    assert (n <= local_data'length)
      report "Initialization vector too short"
      severity ERROR;

    ptr := new std_logic_vector'(local_data((local_data'length - n) to (local_data'length - 1)));
    return ptr.all;
  end hex;

  -- Rounds value up to neareast length multiple of 4; i.e. 4*round_up_val>=n
```

```
function round_up4 (n : integer)
  return integer is
  variable round_up_val : integer:= n/4;
begin
  if (n rem 4) /= 0 then
    round_up_val := round_up_val+1;
  end if;
  return round_up_val;
end round_up4;

-- convert std_logic_vector to string

function lv2str (data : std_logic_vector)
  return string is
  variable local_data : std_logic_vector(data'range) := data;
  variable str : string(local_data'length downto 1);
begin
  for i in 1 to local_data'length loop
    case local_data(i-1) is
      when 'U' => str(i) := 'U';
      when 'X' => str(i) := 'X';
      when '0' => str(i) := '0';
      when '1' => str(i) := '1';
      when 'Z' => str(i) := 'Z';
      when 'W' => str(i) := 'W';
      when 'L' => str(i) := 'L';
      when 'H' => str(i) := 'H';
      when '-' => str(i) := '-';
      when others => str(i) := '?';
    end case;
  end loop;
  return str;
end lv2str;

-- convert std_logic_vector to string in hex format
function lv2strx (data : std_logic_vector;
                  n : integer)
  return string is
  variable m : integer:= round_up4(n);
  variable local_data : std_logic_vector(data'range) := data;
  variable str    : string(m downto 1);
  variable vector : std_logic_vector((m*4)-1 downto 0);
  variable nibble : std_logic_vector(3 downto 0);
```

```
begin
  vector := (others => '0'); --clear data
  vector(local_data'length-1 downto 0) := local_data;

  for i in 1 to m loop
    nibble := vector(i*4-1 downto (i-1)*4);

    case nibble is
      when "0000" => str(i) := '0';
      when "0001" => str(i) := '1';
      when "0010" => str(i) := '2';
      when "0011" => str(i) := '3';
      when "0100" => str(i) := '4';
      when "0101" => str(i) := '5';
      when "0110" => str(i) := '6';
      when "0111" => str(i) := '7';
      when "1000" => str(i) := '8';
      when "1001" => str(i) := '9';
      when "1010" => str(i) := 'A';
      when "1011" => str(i) := 'B';
      when "1100" => str(i) := 'C';
      when "1101" => str(i) := 'D';
      when "1110" => str(i) := 'E';
      when "1111" => str(i) := 'F';
      when others => str(i) := 'X';
    end case;
  end loop;
  return str;
end lv2strx;

-- remove blank tail characters
function strip (data : string)
  return string is
  variable index : natural:= 1;
begin
  for i in data'length downto 1 loop
    if data(i)/=' ' then
      index:= i;
      exit;
    end if;
  end loop;
  return data(1 to index);
end strip;
```

```
-- Write message to logfile
procedure writef (file log : text;
                 constant cycle : in integer;
                 msg : in string) is
  variable BufLine : line;
begin
  write(BufLine,cycle,right,5);
  write(Bufline,string'(" "));
  write(BufLine,msg); -- write message into buffer
  writeline(log,Bufline); -- exit buffer to standard output
end writef;

procedure writef (file log : text;
                 constant cycle : in integer;
                 msg : in string;
                        constant error_no : in integer) is
  variable BufLine : line;
begin
  write(BufLine,cycle,right,5);
  write(Bufline,string'(" "));
  write(BufLine,msg); -- write message into buffer
  write(BufLine,error_no,right,5);
  writeline(log,Bufline); -- exit buffer to standard output
end writef;

 procedure writef (file log : text;
                 msg : in string;
                        constant error_no : in integer) is
  variable BufLine : line;
begin
  write(BufLine,msg); -- write message into buffer
  write(BufLine,error_no,right,5);
  writeline(log,Bufline); -- exit buffer to standard output
end writef;

-- Write message to logfile
procedure writef (file log : text;
                 msg : in string) is
  variable BufLine : line;
begin
  write(BufLine,msg); -- write message into buffer
  writeline(log,Bufline); -- exit buffer to standard output
```

```
end writef;

procedure clk_gen (signal clk : out std_logic;
                          signal cycle_no : inout natural;
                          signal run : in std_logic;
                          constant period : in  time;
                          constant high   : in  time;
                          constant offset : in  time ) is
  constant low : time:= period - offset - high;
begin
  loop
    cycle_no<= cycle_no + 1;
    wait for offset;
    clk <= '1';
    wait for high;
    clk <= '0';
    wait for low;
    if (run = '0') then
      wait;
    end if;
  end loop;
end procedure clk_gen;

procedure clk_gen_changing (signal clk : out std_logic;
                          signal cycle_no : inout natural;
                          signal run : in std_logic;
                          signal period : in  time;
                          signal high   : in  time;
                          constant offset : in  time ) is
begin
  loop
    cycle_no<= cycle_no + 1;
    wait for offset;
    clk <= '1';
    wait for high;
    clk <= '0';
    wait for period-offset-high;
    if (run = '0') then
      wait;
    end if;
  end loop;
end procedure clk_gen_changing;
```

```
procedure clk_gen_np (signal clk_p : out std_logic;
                          signal clk_n : out std_logic;
                       signal cycle_no : inout natural;
                       signal run : in std_logic;
                       constant period : in  time;
                       constant high   : in  time;
                       constant offset : in  time ) is
  constant low : time:= period - offset - high;
begin
  loop
    cycle_no<= cycle_no + 1;
    wait for offset;
    clk_p <= '1';
    clk_n <= '0';
    wait for high;
    clk_p <= '0';
    clk_n <= '1';
    wait for low;
    if (run = '0') then
      wait;
    end if;
  end loop;
end procedure clk_gen_np;


procedure str_gen (signal str_n : out std_logic;
                     signal rst_n : in std_logic;
                     signal clk   : in std_logic;
                     signal run   : in std_logic;
                     constant str_time : in natural;
                     constant period_length : in natural) is

  constant str_time_i : unsigned(power(period_length)-1 downto 0):=
                    to_unsigned(str_time, power(period_length));
  variable clk_cnt: unsigned(power(period_length)-1 downto 0);

begin
  loop
    wait until rising_edge(clk);
    if rst_n='0' then
          str_n <= '0';
          clk_cnt := (others => '0');
    else
      if clk_cnt = str_time_i then
            str_n <= '1';
      else
        str_n <= '0';
      end if;
          clk_cnt := clk_cnt+1;
    end if;
    if (run = '0') then
      wait;
    end if;
  end loop;
end procedure str_gen;


procedure str_gen_n (signal str_n : out std_logic;
                              signal rst_n : in std_logic;
                              signal clk   : in std_logic;
                              signal run   : in std_logic;
                              constant str_time : in natural;
                              constant period_length : in natural) is

  constant str_time_i : unsigned(power(period_length)-1 downto 0):=
                         to_unsigned(str_time, power(period_length));
  variable clk_cnt: unsigned(power(period_length)-1 downto 0);

begin
  loop
    wait until rising_edge(clk);
    if rst_n='0' then
          str_n <= '1';
          clk_cnt := (others => '0');
    else
      if clk_cnt = str_time_i then
            str_n <= '0';
      else
        str_n <= '1';
      end if;
          clk_cnt := clk_cnt+1;
    end if;
    if (run = '0') then
      wait;
    end if;
  end loop;
end procedure str_gen_n;

procedure str_gen_np (signal str_p : out std_logic;
           signal str_n : out std_logic;
                              signal rst_n : in std_logic;
                              signal clk   : in std_logic;
                              signal run   : in std_logic;
                              constant str_time : in natural;
                              constant period_length : in natural) is

  constant str_time_i : unsigned(power(period_length)-1 downto 0):=
                       to_unsigned(str_time, power(period_length));
  variable clk_cnt: unsigned(power(period_length)-1 downto 0);

begin
  loop
    wait until rising_edge(clk);
    if rst_n='0' then
      str_p <= '0';
      str_n <= '1';
              clk_cnt := (others => '0');
    else
      if clk_cnt = str_time_i then
        str_p <= '1';
                str_n <= '0';
      else
        str_p <= '0';
        str_n <= '1';
      end if;
              clk_cnt := clk_cnt+1;
    end if;
    if (run = '0') then
      wait;
    end if;
  end loop;
end procedure str_gen_np;


procedure random_gen (signal rst_n       : in std_logic;
                                 signal clk         : in std_logic;
                                 signal run_random  : in std_logic;
                                 signal random_data : out std_logic;
                              signal run        : in std_logic ) is

type taps_array_type is array (2 to 32) of std_logic_vector(31 downto 0);

constant TAPS_ARRAY : taps_array_type :=
  (2  => (0|1        => '1', others => '0'),
   3  => (0|2        => '1', others => '0'),
   4  => (0|3        => '1', others => '0'),
   5  => (1|4        => '1', others => '0'),
   6  => (0|5        => '1', others => '0'),
   7  => (0|6        => '1', others => '0'),
   8  => (1|2|3|7    => '1', others => '0'),
   9  => (3|8        => '1', others => '0'),
   10 => (2|9        => '1', others => '0'),
   11 => (1|10       => '1', others => '0'),
   12 => (0|3|5|11   => '1', others => '0'),
   13 => (0|2|3|12   => '1', others => '0'),
   14 => (0|2|4|13   => '1', others => '0'),
   15 => (0|14       => '1', others => '0'),
   16 => (1|2|4|15   => '1', others => '0'),
   17 => (2|16       => '1', others => '0'),
   18 => (6|17       => '1', others => '0'),
   19 => (0|1|14|18  => '1', others => '0'),
   20 => (2|19       => '1', others => '0'),
   21 => (1|20       => '1', others => '0'),
   22 => (0|21       => '1', others => '0'),
   23 => (4|22       => '1', others => '0'),
   24 => (0|2|3|23   => '1', others => '0'),
   25 => (2|24       => '1', others => '0'),
   26 => (0|1|15|25  => '1', others => '0'),
   27 => (0|1|14|26  => '1', others => '0'),
   28 => (2|27       => '1', others => '0'),
   29 => (1|28       => '1', others => '0'),
   30 => (0|3|5|29   => '1', others => '0'),
   31 => (2|30       => '1', others => '0'),
   32 => (1|5|6|31   => '1', others => '0'));

  constant WIDTH : natural := 32;
  constant SEED  : natural := 21;

  variable bits0_nminus2_zero : std_logic;
  variable feedback           : std_logic;
  variable taps : std_logic_vector(WIDTH - 1 downto 0);
  variable lfsr_reg : std_logic_vector(WIDTH - 1 downto 0);
  variable lfsr_reg_temp : std_logic_vector(WIDTH - 1 downto 0);

begin

  taps := TAPS_ARRAY(WIDTH)(WIDTH - 1 downto 0);

  loop

    if run='0' then
         wait;
    elsif rst_n = '0' then
      random_data<= '0';
      lfsr_reg := std_logic_vector(to_unsigned(SEED, WIDTH));
      lfsr_reg_temp := (others => '0');
    elsif falling_edge(clk) then
      if run_random = '1' then
        bits0_nminus2_zero := '0';
        for n in 0 to WIDTH - 2 loop
          bits0_nminus2_zero := bits0_nminus2_zero or lfsr_reg(n);
        end loop;
        feedback := lfsr_reg(WIDTH - 1) xor (not bits0_nminus2_zero);
        for n in 1 to WIDTH - 1 loop
          if (taps(n - 1) = '1') then
            lfsr_reg_temp(n) := lfsr_reg(n - 1) xor feedback;
          else
            lfsr_reg_temp(n) := lfsr_reg(n - 1);
          end if;
        end loop;
        random_data <= lfsr_reg(0);
        lfsr_reg_temp(0) := feedback;
        lfsr_reg := lfsr_reg_temp;
     end if;
    end if;

    wait until falling_edge(clk);

 end loop;

end procedure random_gen;

procedure random_gen (constant WIDTH : in natural range 2 to 32;
       signal seed         : in std_logic_vector;
       signal loopmode     : in boolean;
       signal rst_n        : in std_logic;
                             signal clk          : in std_logic;
                             signal run_random  : in std_logic;
                             signal random_data : out std_logic;
                             signal run          : in std_logic) is

type taps_array_type is array (2 to 32) of std_logic_vector(31 downto 0);

constant TAPS_ARRAY : taps_array_type :=
  (2  => (0|1       => '1', others => '0'),
   3  => (0|2       => '1', others => '0'),
   4  => (0|3       => '1', others => '0'),
   5  => (1|4       => '1', others => '0'),
   6  => (0|5       => '1', others => '0'),
   7  => (0|6       => '1', others => '0'),
   8  => (1|2|3|7   => '1', others => '0'),
   9  => (3|8       => '1', others => '0'),
   10 => (2|9       => '1', others => '0'),
   11 => (1|10      => '1', others => '0'),
   12 => (0|3|5|11  => '1', others => '0'),
   13 => (0|2|3|12  => '1', others => '0'),
   14 => (0|2|4|13  => '1', others => '0'),
   15 => (0|14      => '1', others => '0'),
   16 => (1|2|4|15  => '1', others => '0'),
   17 => (2|16      => '1', others => '0'),
   18 => (6|17      => '1', others => '0'),
   19 => (0|1|14|18 => '1', others => '0'),
   20 => (2|19      => '1', others => '0'),
   21 => (1|20      => '1', others => '0'),
   22 => (0|21      => '1', others => '0'),
   23 => (4|22      => '1', others => '0'),
   24 => (0|2|3|23  => '1', others => '0'),
   25 => (2|24      => '1', others => '0'),
   26 => (0|1|15|25 => '1', others => '0'),
   27 => (0|1|14|26 => '1', others => '0'),
   28 => (2|27      => '1', others => '0'),
   29 => (1|28      => '1', others => '0'),
   30 => (0|3|5|29  => '1', others => '0'),
   31 => (2|30      => '1', others => '0'),
   32 => (1|5|6|31  => '1', others => '0'));
  variable bits0_nminus2_zero : std_logic;
  variable feedback           : std_logic;

  variable taps : std_logic_vector(WIDTH - 1 downto 0);
  variable lfsr_reg : std_logic_vector(WIDTH - 1 downto 0);
  variable lfsr_reg_temp : std_logic_vector(WIDTH - 1 downto 0);

begin

  taps := TAPS_ARRAY(WIDTH)(WIDTH-1 downto 0);

  loop

    if run='0' then
        exit;
    elsif rst_n = '0' then
      random_data<= '0';
      lfsr_reg := SEED;
      lfsr_reg_temp := (others => '0');
    elsif falling_edge(clk) then
      if run_random = '1' then
        if loopmode then
          lfsr_reg := lfsr_reg(WIDTH-2 downto 0) & lfsr_reg(WIDTH-1);
          random_data <= lfsr_reg(0);
        else
          bits0_nminus2_zero := '0';
          for n in 0 to WIDTH - 2 loop
            bits0_nminus2_zero := bits0_nminus2_zero or lfsr_reg(n);
          end loop;
          feedback := lfsr_reg(WIDTH - 1) xor (not bits0_nminus2_zero);
          for n in 1 to WIDTH - 1 loop
            if (taps(n - 1) = '1') then
              lfsr_reg_temp(n) := lfsr_reg(n - 1) xor feedback;
            else
              lfsr_reg_temp(n) := lfsr_reg(n - 1);
            end if;
          end loop;
          random_data <= lfsr_reg(0);
          lfsr_reg_temp(0) := feedback;
          lfsr_reg := lfsr_reg_temp;
        end if;
      end if;
    end if;
```

```
      wait until falling_edge(clk);

    end loop;

    wait;

  end procedure random_gen;

  function power(size : natural) return natural is
    variable x : natural;
    variable remainder : natural;
    variable cnt : natural;
  begin
    x := Size;
    cnt := 0;

    for i in size downto 0 loop
      remainder:= x rem 2;
      x := (x / 2) + remainder;
      cnt := cnt + 1;
      if x <= 1 then
        return cnt;
      end if;
    end loop;
  end power;

----------------------------------------------------------------
-- This procedure removes leading spaces
----------------------------------------------------------------
  procedure rm_space(l : inout line) is
    variable l_tmp : line:= l;
    variable skipchar : character;
    variable index : natural;
  begin
    index:= l'low;
    while l'length>0 and index<=l'high loop
       if l_tmp(index)=' ' or
          l(index)=character'val(160) or
          l(index)=HT then
         index:= index+1;
       else
         exit;
      end if;
    end loop;
    if l'length=0 or index>l'high then
      l:= null;
    else
      l:= new string'(l_tmp(index to l_tmp'high));
    end if;
    deallocate(l_tmp);
  end procedure rm_space;

-----------------------------------------------------------------
--  This procedure read a word from line and returns
--    a string containing the word until first ' ', NBSP, HT or +
--  The actual length of the word is returned and
--    the string word is converted to lower.
-----------------------------------------------------------------
  procedure readword(l        : inout line;
                     l_width  : out natural;
                     word     : out string) is
    variable word_tmp    : string(word'range); -- range given by word
    variable char_tmp    : character;
    variable l_width_tmp : natural;

  begin
    word_tmp := (others => ' ');
    l_width_tmp:= 0;
    rm_space(l);
    if (l /= NULL) then
      l_width_tmp:= word_tmp'left;
      while l'length>0 and
            not (l(l'left)=' ' or
                 l(l'left)=character'val(160) or -- Non Blank SPace
                 l(l'left)=HT or
                 l(l'left)='+') loop
        read(l,word_tmp(l_width_tmp));
        l_width_tmp:= l_width_tmp+1;
      end loop;
    end if;
    l_width := l_width_tmp-1;
    word := to_lower(word_tmp);  -- std_iopak

  end procedure readword;
```

```
---------------------------------------------------------------
-- This procedure returns the length of the line; max 40
---------------------------------------------------------------
  procedure string_length(l : inout line; value : out natural) is
  begin
    if l'high<40 then
      value:= l'high;
    else
      value:= 40;
    end if;
  end procedure string_length;

---------------------------------------------------------------
-- This procedure removes char up to the specified character
---------------------------------------------------------------
  procedure find_char(l : inout line;
                      constant char : character) is
    variable l_tmp : line:= l;
    variable skipchar : character;
    variable index : natural;
  begin
    index:= l'low;
    while l'length>0 and index<=l'high loop
      if l_tmp(index)=char then
        index:= index+1;
        exit;
      end if;
      index:= index+1;
     end loop;
    if l'length=0 or index>l'high then
      l:= null;
    else
      l:= new string'(l_tmp(index to l_tmp'high));
    end if;
    deallocate(l_tmp);
  end procedure find_char;

---------------------------------------------------------------
-- This procedure removes char up to the specified character
--   and also returns the character before searched character
---------------------------------------------------------------
  procedure find_char(l : inout line;
                      constant char : character;
                      lastchar : out character) is
    variable l_tmp : line:= l;
    variable skipchar : character;
    variable index : natural;
  begin
    index:= l'low;
    lastchar:= ' ';
    while l'length>0 and index<=l'high loop
      if l_tmp(index)=char then
        lastchar:= l_tmp(index-1);
        index:= index+1;
        exit;
      end if;
      index:= index+1;
     end loop;
    if l'length=0 or index>l'high then
      l:= null;
    else
      l:= new string'(l_tmp(index to l_tmp'high));
    end if;
    deallocate(l_tmp);
  end procedure find_char;

end package body base_pck;
```

TB_LAB4_ENT

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

library work;
use work.lab4_pck.all;

library work;
use work.all;
use work.lab4_tb_pck.all;

entity tb_lab4 is

  -- empty signal list
end tb_lab4;
```

TB_LAB4_BEH

```
use std.textio.all;
use std.env.all; -- Defines finish(0) function etc.

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

library std_developerskit;
use std_developerskit.std_iopak.all;

library modelsim_lib;
use modelsim_lib.util.all;

library work;
use work.all;
use work.base_pck.all;
use work.lab4_tb_pck.all;


library work;
use work.lab4_pck.all;

architecture beh of tb_lab4 is

  -- Logfile Declarations
  file tb_lab4_log : text open write_mode is "tb_lab4.log";

  component motor is
    generic (
      phase90 : time);
    port (
      run       : in  std_logic;
      motor_cw  : in  std_logic;
      motor_ccw : in  std_logic;
      a         : out std_logic;
      b         : out std_logic);
  end component motor;

  component lab4_top
    port (
      arst              : in    std_logic;
      sync_rst          : in    std_logic;
      mclk              : in    std_logic;
      a                 : in    std_logic;
      b                 : in    std_logic;
      force_cw          : in    std_logic;
      force_ccw         : in    std_logic;
      motor_cw          : out   std_logic;
      motor_ccw         : out   std_logic;
      a_n               : out   std_logic_vector(3 downto 0);
      abcdefgdec_n      : out   std_logic_vector(7 downto 0);
      sw                : in    std_logic_vector(7 downto 0);
      DDR_addr          : inout STD_LOGIC_VECTOR (14 downto 0);
      DDR_ba            : inout STD_LOGIC_VECTOR (2 downto 0);
      DDR_cas_n         : inout STD_LOGIC;
      DDR_ck_n          : inout STD_LOGIC;
      DDR_ck_p          : inout STD_LOGIC;
      DDR_cke           : inout STD_LOGIC;
      DDR_cs_n          : inout STD_LOGIC;
      DDR_dm            : inout STD_LOGIC_VECTOR (3 downto 0);
      DDR_dq            : inout STD_LOGIC_VECTOR (31 downto 0);
      DDR_dqs_n         : inout STD_LOGIC_VECTOR (3 downto 0);
      DDR_dqs_p         : inout STD_LOGIC_VECTOR (3 downto 0);
      DDR_odt           : inout STD_LOGIC;
      DDR_ras_n         : inout STD_LOGIC;
      DDR_reset_n       : inout STD_LOGIC;
      DDR_we_n          : inout STD_LOGIC;
      FIXED_IO_ddr_vrn  : inout STD_LOGIC;
      FIXED_IO_ddr_vrp  : inout STD_LOGIC;
      FIXED_IO_mio      : inout STD_LOGIC_VECTOR (53 downto 0);
      FIXED_IO_ps_clk   : inout STD_LOGIC;
      FIXED_IO_ps_porb  : inout STD_LOGIC;
      FIXED_IO_ps_srstb : inout STD_LOGIC);
  end component;

  signal mclk               : std_logic;         -- Master testbench clock TBD MHz
  signal clk_125m           : std_logic;         -- AXI4 clock 125 MHz
  signal s_axi_aclk         : std_logic;         -- AXI4 clock
  signal clk_125m_cycle_no : natural   := 0;   -- cycle count 125MHz.
  signal cycle_no           : natural   := 0;   -- cycle count 40 MHz.
  signal error_no           : natural   := 0;   -- Error count

  signal run                : std_logic := '1'; -- Setting to '0' will terminate simulation
```

```
  signal arst              : std_logic;
  signal sync_rst          : std_logic;
  signal a                 : std_logic;
  signal b                 : std_logic;
  signal force_cw          : std_logic;
  signal force_ccw         : std_logic;
  signal motor_cw          : std_logic;
  signal motor_ccw         : std_logic;
  signal a_n               : std_logic_vector(3 downto 0);
  signal abcdefgdec_n      : std_logic_vector(7 downto 0);
  signal sw                : std_logic_vector(7 downto 0);
  signal DDR_addr          : STD_LOGIC_VECTOR (14 downto 0);
  signal DDR_ba            : STD_LOGIC_VECTOR (2 downto 0);
  signal DDR_cas_n         : STD_LOGIC;
  signal DDR_ck_n          : STD_LOGIC;
  signal DDR_ck_p          : STD_LOGIC;
  signal DDR_cke           : STD_LOGIC;
  signal DDR_cs_n          : STD_LOGIC;
  signal DDR_dm            : STD_LOGIC_VECTOR (3 downto 0);
  signal DDR_dq            : STD_LOGIC_VECTOR (31 downto 0);
  signal DDR_dqs_n         : STD_LOGIC_VECTOR (3 downto 0);
  signal DDR_dqs_p         : STD_LOGIC_VECTOR (3 downto 0);
  signal DDR_odt           : STD_LOGIC;
  signal DDR_ras_n         : STD_LOGIC;
  signal DDR_reset_n       : STD_LOGIC;
  signal DDR_we_n          : STD_LOGIC;
  signal FIXED_IO_ddr_vrn  : STD_LOGIC;
  signal FIXED_IO_ddr_vrp  : STD_LOGIC;
  signal FIXED_IO_mio      : STD_LOGIC_VECTOR (53 downto 0);
  signal FIXED_IO_ps_clk   : STD_LOGIC;
  signal FIXED_IO_ps_porb  : STD_LOGIC;
  signal FIXED_IO_ps_srstb : STD_LOGIC;

begin

  motor_1: motor
    generic map (
      phase90 => 50 us)
    port map (
      run       => run,
      motor_cw  => motor_cw,
      motor_ccw => motor_ccw,
    a          => a,
    b          => b);

DUT: lab4_top
  port map (
    arst              => arst,
    sync_rst          => sync_rst,
    mclk              => mclk,
    a                 => a,
    b                 => b,
    force_cw          => force_cw,
    force_ccw         => force_ccw,
    motor_cw          => motor_cw,
    motor_ccw         => motor_ccw,
    a_n               => a_n,
    abcdefgdec_n      => abcdefgdec_n,
    sw                => sw,
    DDR_addr          => DDR_addr,
    DDR_ba            => DDR_ba,
    DDR_cas_n         => DDR_cas_n,
    DDR_ck_n          => DDR_ck_n,
    DDR_ck_p          => DDR_ck_p,
    DDR_cke           => DDR_cke,
    DDR_cs_n          => DDR_cs_n,
    DDR_dm            => DDR_dm,
    DDR_dq            => DDR_dq,
    DDR_dqs_n         => DDR_dqs_n,
    DDR_dqs_p         => DDR_dqs_p,
    DDR_odt           => DDR_odt,
    DDR_ras_n         => DDR_ras_n,
    DDR_reset_n       => DDR_reset_n,
    DDR_we_n          => DDR_we_n,
    FIXED_IO_ddr_vrn  => FIXED_IO_ddr_vrn,
    FIXED_IO_ddr_vrp  => FIXED_IO_ddr_vrp,
    FIXED_IO_mio      => FIXED_IO_mio,
    FIXED_IO_ps_clk   => FIXED_IO_ps_clk,
    FIXED_IO_ps_porb  => FIXED_IO_ps_porb,
    FIXED_IO_ps_srstb => FIXED_IO_ps_srstb);


-- Core design clock
CLK_GEN_100M: clk_gen(
  clk      => mclk,
    cycle_no => cycle_no,
    run      => run,
    period   => T_100M,
    high     => T_HIGH_100M,
    offset   => T_OFFSET_100M
  );

  -- AXI4 clock
  CLK_GEN_125M: clk_gen(
    clk      => clk_125m,
    cycle_no => clk_125m_cycle_no,
    run      => run,
    period   => T_125M,
    high     => T_HIGH_125M,
    offset   => T_OFFSET_125M
  );

  -- Clock assignments to core
  s_axi_aclk <= clk_125m;

  TB_STIM: -- Testbench stimuli process
  process


    alias ARESET          is <<signal .tb_lab4.DUT.lab4processor_0.ARESET          :
std_logic_vector(0 downto 0)>>;
    alias s_axi_araddr    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_araddr :
std_logic_vector(31 downto 0)>>;
    alias s_axi_arprot    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_arprot :
std_logic_vector(2 downto 0)>>;
    alias s_axi_arready   is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_arready:
std_logic>>;
    alias s_axi_arvalid   is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_arvalid:
std_logic>>;
    alias s_axi_awaddr    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_awaddr :
std_logic_vector(31 downto 0)>>;
    alias s_axi_awprot    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_awprot :
std_logic_vector(2 downto 0)>>;
    alias s_axi_awready   is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_awready:
std_logic>>;
    alias s_axi_awvalid   is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_awvalid:
std_logic>>;
```

```
   alias s_axi_bready    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_bready :
std_logic>>;
   alias s_axi_bresp     is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_bresp  :
std_logic_vector(1 downto 0)>>;
   alias s_axi_bvalid    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_bvalid : std_logic>>;
   alias s_axi_rdata     is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_rdata  :
std_logic_vector(31 downto 0)>>;
   alias s_axi_rready    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_rready :
std_logic>>;
   alias s_axi_rresp     is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_rresp  :
std_logic_vector(1 downto 0)>>;
   alias s_axi_rvalid    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_rvalid : std_logic>>;
   alias s_axi_wdata     is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_wdata  :
std_logic_vector(31 downto 0)>>;
   alias s_axi_wready    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_wready :
std_logic>>;
   alias s_axi_wstrb     is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_wstrb  :
std_logic_vector(3 downto 0)>>;
   alias s_axi_wvalid    is <<signal .tb_lab4.DUT.lab4processor_0.M00_AXI_wvalid :
std_logic>>;

   -- setting axi_clk_run signal to '0' will terminate simulation
   alias axi_clk_run    is <<signal .tb_lab4.DUT.lab4processor_0.run : std_logic>>;

   -- Zynq uP Write
   procedure MW (
     constant tsize    : in transaction_size;
     constant addr     : in std_logic_vector(ADDRESS_LENGTH-1 downto 0);
--      constant data     : in std_logic_vector(DATA_LENGTH-1 downto 0)) is
     constant data     : in std_logic_vector) is
       variable error_found : boolean;
   begin

     AXI4LiteWrite(tsize           => tsize,
                   addr            => addr,
                   data            => data,
                   s00_axi_aclk     => s_axi_aclk,
                   s00_axi_awaddr   => s_axi_awaddr,
                   s00_axi_awprot   => s_axi_awprot,
                   s00_axi_awvalid  => s_axi_awvalid,
                   s00_axi_awready  => s_axi_awready,
                   s00_axi_wdata    => s_axi_wdata,
                   s00_axi_wstrb    => s_axi_wstrb,
                   s00_axi_wvalid   => s_axi_wvalid,
                   s00_axi_wready   => s_axi_wready,
                   s00_axi_bresp    => s_axi_bresp,
                   s00_axi_bvalid   => s_axi_bvalid,
                   s00_axi_bready   => s_axi_bready,
                   log              => tb_lab4_log,
                   cycle            => cycle_no,
                   error_found      => error_found,
                   error_no         => error_no);

     wait until rising_edge(mclk);

   end procedure MW;

   -- Zynq uP Read
   procedure MR (
     constant tsize    : in  transaction_size;
     constant addr     : in  std_logic_vector(ADDRESS_LENGTH-1 downto 0);
     variable data     : out std_logic_vector) is
       variable error_found : boolean;
       variable error_no_i  : natural;
   begin

     AXI4LiteRead(tsize            => tsize,
                addr             => addr,
                data             => data,
                s00_axi_aclk     => s_axi_aclk,
                s00_axi_araddr   => s_axi_araddr,
                s00_axi_arprot   => s_axi_arprot,
                s00_axi_arvalid  => s_axi_arvalid,
                s00_axi_arready  => s_axi_arready,
                s00_axi_rdata    => s_axi_rdata,
                s00_axi_rresp    => s_axi_rresp,
                s00_axi_rvalid   => s_axi_rvalid,
                s00_axi_rready   => s_axi_rready,
                log              => tb_lab4_log,
                cycle            => cycle_no,
                error_found      => error_found,
                error_no         => error_no_i);

     if error_found then
       error_no <= error_no + error_no_i;
     end if;
```

```
   wait until rising_edge(mclk);

end procedure MR;

--  Zynq uP Check
procedure MC (
  constant tsize    : in  transaction_size;
  constant addr     : in  std_logic_vector(ADDRESS_LENGTH-1 downto 0);
  constant data     : in  std_logic_vector) is
    variable data_i      : std_logic_vector(data'length-1 downto 0);
    variable error_found : boolean;
begin

  data_i := data;

  AXI4LiteCheck(tsize            => tsize,
              addr             => addr,
              data             => data_i,
              s00_axi_aclk     => s_axi_aclk,
              s00_axi_araddr   => s_axi_araddr,
              s00_axi_arprot   => s_axi_arprot,
              s00_axi_arvalid  => s_axi_arvalid,
              s00_axi_arready  => s_axi_arready,
              s00_axi_rdata    => s_axi_rdata,
              s00_axi_rresp    => s_axi_rresp,
              s00_axi_rvalid   => s_axi_rvalid,
              s00_axi_rready   => s_axi_rready,
              log              => tb_lab4_log,
              cycle            => cycle_no,
              error_found      => error_found,
              error_no         => error_no);

  wait until rising_edge(mclk);
end procedure MC;

variable bvalue   : std_logic_vector(7 downto 0);
variable hvalue   : std_logic_vector(15 downto 0);
variable wvalue   : std_logic_vector(31 downto 0);
variable ramaddr  : std_logic_vector(31 downto 0);
variable wdata    : std_logic_vector(31 downto 0);

begin
```

```
-- Initializing signals
-- NOTE: Remove a and b signal init when using motor model.
-- a            <= '0';
-- b            <= '0';
------------------------------------------------------------

force_cw      <= '0';
force_ccw     <= '0';
sw(7)         <= '0';       -- Select switches (i.e. sw(6:0) as setpoint initially
sw(6 downto 0) <= "1011101"; -- Switch value selected

s_axi_araddr  <= (others => '0');
s_axi_arprot  <= (others => '0');
s_axi_arvalid <= '0';
s_axi_awaddr  <= (others => '0');
s_axi_awprot  <= (others => '0');
s_axi_awvalid <= '0';
s_axi_bready  <= '0';
s_axi_rready  <= '0';
s_axi_wdata   <= (others => '0');
s_axi_wstrb   <= (others => '0');
s_axi_wvalid  <= '0';

-- Setting reset signals for 10 cycles
ARESET(0) <= '0';
arst      <= '1';
sync_rst  <= '1';

Tcycle(10);

ARESET(0) <= '1';
arst      <= '0';
sync_rst  <= '0';

Tcycle(10);

-- Reset complete; perform simulation
-- Perform 32-bit r/w access of the LAB4REG_RWTEST register
wdata := x"12345678";
MW(SINGLE, LAB4REG_RWTEST, wdata);
MR(SINGLE, LAB4REG_RWTEST, wvalue);
writef(tb_lab4_log, cycle_no, "Read value in testbench : " & lv2strx(wvalue,32));
```

```
    MC(SINGLE, LAB4REG_RWTEST, x"12345678");

    wdata := x"ABCDEF98";
    MW(SINGLE, LAB4REG_RWTEST, wdata);
    MR(SINGLE, LAB4REG_RWTEST, wvalue);
    writef(tb_lab4_log, cycle_no, "Read value in testbench : " & lv2strx(wvalue,32));
    MC(SINGLE, LAB4REG_RWTEST, x"ABCDEF98");

        -- Switch to processor register LAB4REG_SETPOINT as setpoint
    sw(7) <= '1';
        MW(BYTE, LAB4REG_SETPOINT, "01001010");

       --testing test registers


    MW(SINGLE, LAB4REG_32, x"FFFFFFFF");
    MR(SINGLE, LAB4REG_32, wvalue);
    writef(tb_lab4_log, cycle_no, "Read test_value in testbench : " & lv2strx(wvalue,32));
    MC(SINGLE, LAB4REG_32, x"FFFFFFFF");



    MW(SINGLE, LAB4REG_16,  x"00005111");
    MR(SINGLE, LAB4REG_16, wvalue);
    writef(tb_lab4_log, cycle_no, "Read test_value in testbench : " & lv2strx(wvalue,32));
    MC(SINGLE, LAB4REG_16, x"00005111");

    -- <Add register accesses here >


        -- Example RAM accesses when comments removed:

     MC(SINGLE, LAB4RAM_BASE_ADDRESS, x"DEADBEEF");  -- Reset value in RAM
simulation model

    MW(SINGLE, LAB4RAM_BASE_ADDRESS, x"12345678");
    ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 4);
    MW(SINGLE, ramaddr, x"87654321");
    ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 8);
    MW(SINGLE, ramaddr, x"ABCDEF98");

    MC(SINGLE, LAB4RAM_BASE_ADDRESS, x"12345678");
    ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 4);
```

```
MC(SINGLE, ramaddr, x"87654321");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 8);
MC(SINGLE, ramaddr, x"ABCDEF98");


	MW(HALFWORD, LAB4RAM_BASE_ADDRESS, x"1234");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 2);
MW(HALFWORD, ramaddr, x"5678");

MC(HALFWORD, LAB4RAM_BASE_ADDRESS, x"1234");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 2);
MC(HALFWORD, ramaddr, x"5678");

MC(SINGLE, LAB4RAM_BASE_ADDRESS, x"56781234");


MW(BYTE, LAB4RAM_BASE_ADDRESS, x"01");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 1);
MW(BYTE, ramaddr, x"02");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 2);
MW(BYTE, ramaddr, x"03");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 3);
MW(BYTE, ramaddr, x"04");

MC(BYTE, LAB4RAM_BASE_ADDRESS, x"01");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 1);
MC(BYTE, ramaddr, x"02");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 2);
MC(BYTE, ramaddr, x"03");
ramaddr:= std_logic_vector(unsigned(LAB4RAM_BASE_ADDRESS) + 3);
MC(BYTE, ramaddr, x"04");

MC(SINGLE, LAB4RAM_BASE_ADDRESS, x"04030201");


Tcycle(100000); -- Run to let the position signals from motor model change ...

run <= '0';
axi_clk_run <= '0';

-- Write error result
if (error_no > 0) then
```

```
      writef(tb_lab4_log, cycle_no,"  ERROR: Simulation failed! Number of errors: " &
to_string(error_no));
      assert (false) report "ERROR: Simulation failed! Number of errors: " & to_string(error_no)
        severity note;
    else
      writef(tb_lab4_log, cycle_no, "  NOTE: Testbench simulation successful!");
      assert (false)
        report "Testbench simulation successful!"
        severity note;
    end if;

    wait; -- Terminate the process

  end process;

end architecture beh;
```

LAB4_TB_PCK

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

library work;
use work.base_pck.all;

package lab4_tb_pck is

  type transaction_size is (BYTE, HALFWORD, SINGLE, DOUBLE);

  -- clk timing
  constant T_100M         : time;
  constant T_OFFSET_100M  : time;
  constant T_HIGH_100M    : time;

  -- AXI4lite clk timing
  constant T_125M          : time;
  constant T_OFFSET_125M   : time;
  constant T_HIGH_125M     : time;

  procedure Tcycle (n : natural);
procedure Reset (
  signal   rst_n : out std_logic;
  file     log   :     text;
  constant cycle : in  integer
);

procedure AXI4LiteWrite (
  constant tsize           : in  transaction_size;
  constant addr            : in  std_logic_vector(31 downto 0);
  constant data            : in  std_logic_vector;

  -- AXI4Lite clock
  signal  s00_axi_aclk      : in  std_logic;

  -- Write Address Channel
  signal  s00_axi_awaddr    : out std_logic_vector(31 downto 0);
  signal  s00_axi_awprot    : out std_logic_vector(2 downto 0);
  signal  s00_axi_awvalid   : out std_logic;
  signal  s00_axi_awready   : in  std_logic;

  -- Write Data Channel
  signal  s00_axi_wdata     : out std_logic_vector(31 downto 0);
  signal  s00_axi_wstrb     : out std_logic_vector(3 downto 0);
  signal  s00_axi_wvalid    : out std_logic;
  signal  s00_axi_wready    : in  std_logic;

  -- Write Respons Channel
  signal  s00_axi_bresp     : in  std_logic_vector(1 downto 0);
  signal  s00_axi_bvalid    : in  std_logic;
  signal  s00_axi_bready    : out std_logic;

  file     log              :      text;
  constant cycle            : in    natural;
  variable error_found      : out   boolean;
  signal   error_no         : inout natural
);

procedure AXI4LiteRead (
  constant tsize           : in  transaction_size;
  constant addr            : in  std_logic_vector(31 downto 0);
  variable data            : out std_logic_vector;

  -- AXI4Lite clock
  signal  s00_axi_aclk      : in  std_logic;

  -- Read Address Channel
  signal  s00_axi_araddr    : out std_logic_vector(31 downto 0);
  signal  s00_axi_arprot    : out std_logic_vector(2 downto 0);
  signal  s00_axi_arvalid   : out std_logic;
  signal  s00_axi_arready   : in  std_logic;

  -- Read Data Channel
  signal  s00_axi_rdata     : in  std_logic_vector(31 downto 0);
  signal  s00_axi_rresp     : in  std_logic_vector(1 downto 0);
  signal  s00_axi_rvalid    : in  std_logic;
  signal  s00_axi_rready    : out std_logic;

  file     log             :       text;
  constant cycle           : in    natural;
  variable error_found      : out   boolean;
  variable error_no         : inout natural
);

procedure AXI4LiteCheck (
  constant tsize           : in  transaction_size;
  constant addr            : in  std_logic_vector(31 downto 0);
  variable data            : in std_logic_vector;

  -- AXI4Lite clock
  signal  s00_axi_aclk      : in  std_logic;

  -- Read Address Channel
  signal  s00_axi_araddr    : out std_logic_vector(31 downto 0);
  signal  s00_axi_arprot    : out std_logic_vector(2 downto 0);
  signal  s00_axi_arvalid   : out std_logic;
  signal  s00_axi_arready   : in  std_logic;

  -- Read Data Channel
  signal  s00_axi_rdata     : in  std_logic_vector(31 downto 0);
  signal  s00_axi_rresp     : in  std_logic_vector(1 downto 0);
  signal  s00_axi_rvalid    : in  std_logic;
  signal  s00_axi_rready    : out std_logic;

  file     log             :       text;
  constant cycle           : in    natural;
  variable error_found      : out   boolean;
```

```
    signal   error_no         : inout natural
  );

end package lab4_tb_pck;
```

LAB4PROCESSOR_ENT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity lab4processor is

  port (
    ACLK              : out   STD_LOGIC;
    ARESET            : out   STD_LOGIC_VECTOR(0 downto 0);
    DDR_addr          : inout STD_LOGIC_VECTOR (14 downto 0);
    DDR_ba            : inout STD_LOGIC_VECTOR (2 downto 0);
    DDR_cas_n         : inout STD_LOGIC;
    DDR_ck_n          : inout STD_LOGIC;
    DDR_ck_p          : inout STD_LOGIC;
    DDR_cke           : inout STD_LOGIC;
    DDR_cs_n          : inout STD_LOGIC;
    DDR_dm            : inout STD_LOGIC_VECTOR (3 downto 0);
    DDR_dq            : inout STD_LOGIC_VECTOR (31 downto 0);
    DDR_dqs_n         : inout STD_LOGIC_VECTOR (3 downto 0);
    DDR_dqs_p         : inout STD_LOGIC_VECTOR (3 downto 0);
    DDR_odt           : inout STD_LOGIC;
    DDR_ras_n         : inout STD_LOGIC;
    DDR_reset_n       : inout STD_LOGIC;
    DDR_we_n          : inout STD_LOGIC;
    FIXED_IO_ddr_vrn  : inout STD_LOGIC;
    FIXED_IO_ddr_vrp  : inout STD_LOGIC;
    FIXED_IO_mio      : inout STD_LOGIC_VECTOR (53 downto 0);
    FIXED_IO_ps_clk   : inout STD_LOGIC;
    FIXED_IO_ps_porb  : inout STD_LOGIC;
    FIXED_IO_ps_srstb : inout STD_LOGIC;
    M00_AXI_araddr    : out   STD_LOGIC_VECTOR (31 downto 0);
    M00_AXI_arprot    : out   STD_LOGIC_VECTOR (2 downto 0);
    M00_AXI_arready   : in    STD_LOGIC;
    M00_AXI_arvalid   : out   STD_LOGIC;
    M00_AXI_awaddr    : out   STD_LOGIC_VECTOR (31 downto 0);
    M00_AXI_awprot    : out   STD_LOGIC_VECTOR (2 downto 0);
    M00_AXI_awready   : in    STD_LOGIC;
    M00_AXI_awvalid   : out   STD_LOGIC;
    M00_AXI_bready    : out   STD_LOGIC;
    M00_AXI_bresp     : in    STD_LOGIC_VECTOR (1 downto 0);
    M00_AXI_bvalid    : in    STD_LOGIC;
    M00_AXI_rdata     : in    STD_LOGIC_VECTOR (31 downto 0);
    M00_AXI_rready    : out   STD_LOGIC;
    M00_AXI_rresp     : in    STD_LOGIC_VECTOR (1 downto 0);
    M00_AXI_rvalid    : in    STD_LOGIC;
    M00_AXI_wdata     : out   STD_LOGIC_VECTOR (31 downto 0);
    M00_AXI_wready    : in    STD_LOGIC;
    M00_AXI_wstrb     : out   STD_LOGIC_VECTOR (3 downto 0);
    M00_AXI_wvalid    : out   STD_LOGIC);

end lab4processor;
```

LAB4PROCESSOR_DMY

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

library work;
use work.all;
use work.base_pck.all;
use work.lab4_tb_pck.all;

architecture dmy of lab4processor is
  signal clk_125m          : std_logic;        -- AXI4 clock 125 MHz
  signal cycle_no          : natural   := 0;   -- cycle count 40 MHz.
  signal run               : std_logic := '1'; -- Run forever
begin

    ACLK              <= clk_125m;
    DDR_addr          <= (others => '0');
    DDR_ba            <= (others => '0');
    DDR_cas_n         <= '1';
    DDR_ck_n          <= '1';
    DDR_ck_p          <= '0';
    DDR_cke           <= '0';
    DDR_cs_n          <= '1';
    DDR_dm            <= (others => '0');
    DDR_dq            <= (others => '0');
    DDR_dqs_n         <= (others => '1');
    DDR_dqs_p         <= (others => '0');
    DDR_odt           <= '0';
    DDR_ras_n         <= '1';
    DDR_reset_n       <= '1';
    DDR_we_n          <= '1';
    FIXED_IO_ddr_vrn  <= '1';
    FIXED_IO_ddr_vrp  <= '0';
    FIXED_IO_mio      <= (others => '0');
    FIXED_IO_ps_clk   <= '0';
    FIXED_IO_ps_porb  <= '0';
    FIXED_IO_ps_srstb <= '0';

    -- AXI4 clock
    CLK_GEN_125M: clk_gen(
      clk      => clk_125m,
      cycle_no => cycle_no,
      run      => run,
      period   => T_125M,
      high     => T_HIGH_125M,
      offset   => T_OFFSET_125M
    );

end dmy;
```

LAB4_TOP_STR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.lab4_pck.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;
```

```
architecture str of lab4_top is

-- START ADDING COMPONENT

-- COMPONENT PS + AXI INTERCONNECT
   component design_1_wrapper is
    port (
      ACLK              : out   STD_LOGIC;
      ARESET            : out   STD_LOGIC_VECTOR (0 to 0);
      DDR_addr          : inout STD_LOGIC_VECTOR (14 downto 0);
      DDR_ba            : inout STD_LOGIC_VECTOR (2 downto 0);
      DDR_cas_n         : inout STD_LOGIC;
      DDR_ck_n          : inout STD_LOGIC;
      DDR_ck_p          : inout STD_LOGIC;
      DDR_cke           : inout STD_LOGIC;
      DDR_cs_n          : inout STD_LOGIC;
      DDR_dm            : inout STD_LOGIC_VECTOR (3 downto 0);
      DDR_dq            : inout STD_LOGIC_VECTOR (31 downto 0);
      DDR_dqs_n         : inout STD_LOGIC_VECTOR (3 downto 0);
      DDR_dqs_p         : inout STD_LOGIC_VECTOR (3 downto 0);
      DDR_odt           : inout STD_LOGIC;
      DDR_ras_n         : inout STD_LOGIC;
      DDR_reset_n       : inout STD_LOGIC;
      DDR_we_n          : inout STD_LOGIC;
      FIXED_IO_ddr_vrn  : inout STD_LOGIC;
      FIXED_IO_ddr_vrp  : inout STD_LOGIC;
      FIXED_IO_mio      : inout STD_LOGIC_VECTOR (53 downto 0);
      FIXED_IO_ps_clk   : inout STD_LOGIC;
      FIXED_IO_ps_porb  : inout STD_LOGIC;
      FIXED_IO_ps_srstb : inout STD_LOGIC;
      M00_AXI_araddr    : out   STD_LOGIC_VECTOR (31 downto 0);
      M00_AXI_arprot    : out   STD_LOGIC_VECTOR (2 downto 0);
      M00_AXI_arready   : in    STD_LOGIC;
      M00_AXI_arvalid   : out   STD_LOGIC;
      M00_AXI_awaddr    : out   STD_LOGIC_VECTOR (31 downto 0);
      M00_AXI_awprot    : out   STD_LOGIC_VECTOR (2 downto 0);
      M00_AXI_awready   : in    STD_LOGIC;
      M00_AXI_awvalid   : out   STD_LOGIC;
      M00_AXI_bready    : out   STD_LOGIC;
      M00_AXI_bresp     : in    STD_LOGIC_VECTOR (1 downto 0);
      M00_AXI_bvalid    : in    STD_LOGIC;
      M00_AXI_rdata     : in    STD_LOGIC_VECTOR (31 downto 0);
      M00_AXI_rready    : out   STD_LOGIC;
      M00_AXI_rresp     : in    STD_LOGIC_VECTOR (1 downto 0);
      M00_AXI_rvalid    : in    STD_LOGIC;
      M00_AXI_wdata     : out   STD_LOGIC_VECTOR (31 downto 0);
      M00_AXI_wready    : in    STD_LOGIC;
      M00_AXI_wstrb     : out   STD_LOGIC_VECTOR (3 downto 0);
      M00_AXI_wvalid    : out   STD_LOGIC);
    end component lab4processor;



-- COMPONENT AXI4PIFB (AXI INTERCONNECT TO PIF)
component axi4pifb
    generic (
      PIF_DATA_LENGTH    : integer;
      PIF_ADDR_LENGTH    : integer;
      C_S_AXI_DATA_WIDTH : integer;
      C_S_AXI_ADDR_WIDTH : integer);
    port (
      s_axi_aclk        : in  std_logic;
      s_axi_aresetn     : in  std_logic;
      s_axi_awaddr      : in  std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
      s_axi_awprot      : in  std_logic_vector(2 downto 0);
      s_axi_awvalid     : in  std_logic;
      s_axi_awready     : out std_logic;
      s_axi_wdata       : in  std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
      s_axi_wstrb       : in  std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
      s_axi_wvalid      : in  std_logic;
      s_axi_wready      : out std_logic;
      s_axi_bresp       : out std_logic_vector(1 downto 0);
      s_axi_bvalid      : out std_logic;
      s_axi_bready      : in  std_logic;
      s_axi_araddr      : in  std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
      s_axi_arprot      : in  std_logic_vector(2 downto 0);
      s_axi_arvalid     : in  std_logic;
      s_axi_arready     : out std_logic;
      s_axi_rdata       : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
      s_axi_rresp       : out std_logic_vector(1 downto 0);
      s_axi_rvalid      : out std_logic;
      s_axi_rready      : in  std_logic;
      pif_clk           : out std_logic;
      pif_rst           : out std_logic;
      pif_regcs         : out std_logic_vector(31 downto 0);
      pif_memcs         : out std_logic_vector(31 downto 0);
```

```
      pif_addr          : out std_logic_vector(PIF_ADDR_LENGTH-1 downto 0);
      pif_wdata         : out std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
      pif_re            : out std_logic_vector(0 downto 0);
      pif_we            : out std_logic_vector(0 downto 0);
      pif_be            : out std_logic_vector((PIF_DATA_LENGTH/8)-1 downto 0);
      rdata_lab4reg2pif : in  std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
      mdata_lab4ram2pif : in std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
      ack_lab4reg2pif   : in  std_logic);
  end component;



-- COMPONENT REGISTER
component lab4_reg is
   generic (
      -- Width of PIF data bus
      PIF_DATA_LENGTH     : integer := PIF_DATA_LENGTH;
      -- Width of PIF address bus
      PIF_ADDR_LENGTH     : integer := PIF_ADDRESS_LENGTH
   );
   port (
      -- Add ports here:
        -- TBD.
        setpoint       : out std_logic_vector(7 downto 0);
      -- Add ports ends

      -- Do not modify the ports beyond this line
      -- Clock Signal
      pif_clk          : in std_logic;
      -- Reset Signal. This signal is active HIGH
      pif_rst          : in std_logic;
      -- Register chip select
      pif_regcs        : in std_logic;
      -- Write address
      pif_addr         : in std_logic_vector(PIF_ADDR_LENGTH-1 downto 0);
      -- Write data
      pif_wdata        : in std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
      -- Read enable strobe
      pif_re           : in std_logic_vector(0 downto 0);
      -- Write enable strobe
      pif_we           : in std_logic_vector(0 downto 0);
      -- Write strobes. This signal indicates which byte lanes hold
      --   valid data. There is one write strobe bit for each eight
```

```
        --   bits of the write data bus.
        pif_be              : in std_logic_vector((PIF_DATA_LENGTH/8)-1 downto 0);
        -- Read data
        rdata_2pif              : out std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
        -- Register read and write access acknowledge
        ack_2pif        : out std_logic
    );
end component;

-- COMPONENT MOTOR CONTROLLER
component pos_seg7_ctrl is
    port (
        -- System Clock and Reset
        arst        : in  std_logic;      -- Reset
        sync_rst     : in  std_logic;      -- Synchronous reset
        refclk      : in  std_logic;      -- Clock
        sp          : in  std_logic_vector(7 downto 0);  -- Set Point
        a           : in  std_logic;      -- From position sensor
        b           : in  std_logic;      -- From position sensor
        force_cw     : in  std_logic;      -- Force motor clock wise motion
        force_ccw    : in  std_logic;  -- Force motor counter clock wise motion
        motor_cw     : out std_logic;      -- Motor clock wise motion
        motor_ccw    : out std_logic;      -- Motor counter clock wise motion
        -- Interface to seven segments
        abcdefgdec_n : out std_logic_vector(7 downto 0);
        a_n         : out std_logic_vector(3 downto 0)
    );
end component;

component ram_lab4
  port (
    clka  : in  STD_LOGIC;
    rsta  : in  STD_LOGIC;
    ena   : in  STD_LOGIC;
    wea   : in  STD_LOGIC_VECTOR (3 downto 0);
    addra : in  STD_LOGIC_VECTOR (9 downto 0);
    dina  : in  STD_LOGIC_VECTOR (31 downto 0);
    douta : out STD_LOGIC_VECTOR (31 downto 0));
end component;

-- SIGNALS

signal aclk : std_logic;
signal pifb_axi_aresetn  : std_logic_vector(0 downto 0);
signal pifb_axi_awaddr   : std_logic_vector(31 downto 0);
signal pifb_axi_awprot   : std_logic_vector(2 downto 0);
signal pifb_axi_awvalid  : std_logic;
signal pifb_axi_awready  : std_logic;
signal pifb_axi_wdata    : std_logic_vector(31 downto 0);
signal pifb_axi_wstrb    : std_logic_vector(3 downto 0);
signal pifb_axi_wvalid   : std_logic;
signal pifb_axi_wready   : std_logic;
signal pifb_axi_bresp    : std_logic_vector(1 downto 0);
signal pifb_axi_bvalid   : std_logic;
signal pifb_axi_bready   : std_logic;
signal pifb_axi_araddr   : std_logic_vector(31 downto 0);
signal pifb_axi_arprot   : std_logic_vector(2 downto 0);
signal pifb_axi_arvalid  : std_logic;
signal pifb_axi_arready  : std_logic;
signal pifb_axi_rdata    : std_logic_vector(31 downto 0);
signal pifb_axi_rresp    : std_logic_vector(1 downto 0);
signal pifb_axi_rvalid   : std_logic;
signal pifb_axi_rready   : std_logic;

signal rdata_lab4reg2pif : std_logic_vector(31 downto 0);
signal mdata_lab4ram2pif : std_logic_vector(31 downto 0);
signal ack_lab4reg2pif   : std_logic;

signal pif_memcs         : std_logic_vector(31 downto 0);
signal pif_regcs         : std_logic_vector(31 downto 0);
signal pif_addr          : std_logic_vector(31 downto 0);
signal pif_wdata         : std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
signal pif_re            : std_logic_vector(0 downto 0);
signal pif_we            : std_logic_vector(0 downto 0);
signal pif_be            : std_logic_vector((PIF_DATA_LENGTH/8)-1 downto 0);
signal pif_clk           : std_logic;
signal pif_rst           : std_logic;

signal rst_mclk_s1       : std_logic;
signal rst_mclk_s2       : std_logic;

signal setpoint          : std_logic_vector(7 downto 0);
signal sw_setpoint       : std_logic_vector(7 downto 0);
signal ps_setpoint       : std_logic_vector(7 downto 0);
signal setpoint_select   : std_logic;
```

```
-- END SIGNALS

begin

  -- PORT MAP PS + AXI INTERCONNECT

lab4processor_0: design_1_wrapper
    port map (
      ACLK              => aclk,
      ARESET            => pifb_axi_aresetn,
      DDR_addr          => DDR_addr,
      DDR_ba            => DDR_ba,
      DDR_cas_n         => DDR_cas_n,
      DDR_ck_n          => DDR_ck_n,
      DDR_ck_p          => DDR_ck_p,
      DDR_cke           => DDR_cke,
      DDR_cs_n          => DDR_cs_n,
      DDR_dm            => DDR_dm,
      DDR_dq            => DDR_dq,
      DDR_dqs_n         => DDR_dqs_n,
      DDR_dqs_p         => DDR_dqs_p,
      DDR_odt           => DDR_odt,
      DDR_ras_n         => DDR_ras_n,
      DDR_reset_n       => DDR_reset_n,
      DDR_we_n          => DDR_we_n,
      FIXED_IO_ddr_vrn  => FIXED_IO_ddr_vrn,
      FIXED_IO_ddr_vrp  => FIXED_IO_ddr_vrp,
      FIXED_IO_mio      => FIXED_IO_mio,
      FIXED_IO_ps_clk   => FIXED_IO_ps_clk,
      FIXED_IO_ps_porb  => FIXED_IO_ps_porb,
      FIXED_IO_ps_srstb => FIXED_IO_ps_srstb,
      M00_AXI_araddr    => pifb_axi_araddr,
      M00_AXI_arprot    => pifb_axi_arprot,
      M00_AXI_arready   => pifb_axi_arready,
      M00_AXI_arvalid   => pifb_axi_arvalid,
      M00_AXI_awaddr    => pifb_axi_awaddr,
      M00_AXI_awprot    => pifb_axi_awprot,
      M00_AXI_awready   => pifb_axi_awready,
      M00_AXI_awvalid   => pifb_axi_awvalid,
      M00_AXI_bready    => pifb_axi_bready,
      M00_AXI_bresp     => pifb_axi_bresp,
      M00_AXI_bvalid    => pifb_axi_bvalid,
      M00_AXI_rdata     => pifb_axi_rdata,
      M00_AXI_rready    => pifb_axi_rready,
      M00_AXI_rresp     => pifb_axi_rresp,
      M00_AXI_rvalid    => pifb_axi_rvalid,
      M00_AXI_wdata     => pifb_axi_wdata,
      M00_AXI_wready    => pifb_axi_wready,
      M00_AXI_wstrb     => pifb_axi_wstrb,
      M00_AXI_wvalid    => pifb_axi_wvalid
    );

  -- PORT MAP AXI4PIFB
  axi4pifb_0: axi4pifb
    generic map (
      PIF_DATA_LENGTH    => PIF_DATA_LENGTH,
      PIF_ADDR_LENGTH    => PIF_ADDRESS_LENGTH,
      C_S_AXI_DATA_WIDTH => PIF_DATA_LENGTH,
      C_S_AXI_ADDR_WIDTH => PIF_ADDRESS_LENGTH)
    port map (
      s_axi_aclk        => aclk,
      s_axi_aresetn     => pifb_axi_aresetn(0),
      s_axi_awaddr      => pifb_axi_awaddr,
      s_axi_awprot      => pifb_axi_awprot,
      s_axi_awvalid     => pifb_axi_awvalid,
      s_axi_awready     => pifb_axi_awready,
      s_axi_wdata       => pifb_axi_wdata,
      s_axi_wstrb       => pifb_axi_wstrb,
      s_axi_wvalid      => pifb_axi_wvalid,
      s_axi_wready      => pifb_axi_wready,
      s_axi_bresp       => pifb_axi_bresp,
      s_axi_bvalid      => pifb_axi_bvalid,
      s_axi_bready      => pifb_axi_bready,
      s_axi_araddr      => pifb_axi_araddr,
      s_axi_arprot      => pifb_axi_arprot,
      s_axi_arvalid     => pifb_axi_arvalid,
      s_axi_arready     => pifb_axi_arready,
      s_axi_rdata       => pifb_axi_rdata,
      s_axi_rresp       => pifb_axi_rresp,
      s_axi_rvalid      => pifb_axi_rvalid,
      s_axi_rready      => pifb_axi_rready,
      pif_clk           => pif_clk,
      pif_rst           => pif_rst,
      pif_regcs         => pif_regcs,
      pif_memcs         => pif_memcs,
      pif_addr          => pif_addr,
      pif_wdata         => pif_wdata,
      pif_re            => pif_re,
      pif_we            => pif_we,
      pif_be            => pif_be,
      rdata_lab4reg2pif => rdata_lab4reg2pif,
      mdata_lab4ram2pif => mdata_lab4ram2pif,
      ack_lab4reg2pif   => ack_lab4reg2pif
    );


  -- A better solution may be to move the CRU module from the
  --   pos_seg7_ctrl module up to this level and use the reset
  --   signal from the CRU module, but then the pos_seq7_ctrl
  --   module will not be reused without changes ......
  P_LAB4_REG_RST_SYNCH :
  process(arst, mclk)
  begin
    if arst = '1' then
      rst_mclk_s1 <= '1';
      rst_mclk_s2 <= '1';
    elsif rising_edge(mclk) then
      rst_mclk_s1 <= '0';
      rst_mclk_s2 <= rst_mclk_s1;
    end if;
  end process P_LAB4_REG_RST_SYNCH;


  lab4_reg_0: lab4_reg
    generic map (
        PIF_DATA_LENGTH => PIF_DATA_LENGTH,
        PIF_ADDR_LENGTH => PIF_ADDRESS_LENGTH)
    port map (
        setpoint    => ps_setpoint,
        pif_clk     => mclk,
        pif_rst     => rst_mclk_s2,
        pif_regcs   => pif_regcs(0),
        pif_addr    => pif_addr,
        pif_wdata   => pif_wdata,
        pif_re      => pif_re,
        pif_we      => pif_we,
        pif_be      => pif_be,
        rdata_2pif  => rdata_lab4reg2pif,
        ack_2pif    => ack_lab4reg2pif
    );

   --Instantiation of ram module
  ram_lab4_0: ram_lab4
    port map (
      clka  => pif_clk,
      rsta  => pif_rst,
      ena   => pif_memcs(0),
      wea   => pif_be,
      addra => pif_addr(11 downto 2), -- Word adressing; removing (2 lsb) byte address bits.
      dina  => pif_wdata,
      douta => mdata_lab4ram2pif);

  pos_seq7_ctrl_0: pos_seg7_ctrl
    port map (
        arst => arst,
        sync_rst => sync_rst,
        refclk => mclk,
        sp => setpoint,
        a => a,
        b => b,
        force_cw => force_cw,
        force_ccw => force_ccw,
        motor_cw => motor_cw,
        motor_ccw => motor_ccw,
        abcdefgdec_n => abcdefgdec_n,
        a_n => a_n
    );

  -- Concurrent statements

  -- Select processor (i.e PS) setpoint if external switch bit 7
  --   is equal '1' (i.e. (sw(7)='1') else use external switch
  --   bit 6 downto 0 with unused bit 7 set to '0'.
  setpoint_select <= sw(7);
  sw_setpoint <= '0' & sw(6 downto 0);
  setpoint <= ps_setpoint when setpoint_select='1' else sw_setpoint;

end str;
```

LAB4_TOP_ENT

```
--------------------------------------------------------------------------------
```

```
-- Company:
-- Engineer: Olivier FAVIERES
--
-- Create Date: 07/04/2017 12:04:51 PM
-- Module Name: lab4_top - Structural
-- Tool Versions:
-- Description:
--
-- Revision 0.02 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

library work;
use work.lab4_pck.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 lab4_top is
  port (
        arst : in std_logic;
        sync_rst : in std_logic;
        mclk : in std_logic;
        a : in std_logic;
        b : in std_logic;
        force_cw : in std_logic;
        force_ccw : in std_logic;
        motor_cw : out std_logic;
        motor_ccw : out std_logic;
        a_n : out std_logic_vector(3 downto 0)    ;
        abcdefgdec_n : out std_logic_vector(7 downto 0);
        sw : in std_logic_vector(7 downto 0);
        DDR_addr : inout STD_LOGIC_VECTOR ( 14 downto 0 );
        DDR_ba : inout STD_LOGIC_VECTOR ( 2 downto 0 );
        DDR_cas_n : inout STD_LOGIC;
        DDR_ck_n : inout STD_LOGIC;
        DDR_ck_p : inout STD_LOGIC;
        DDR_cke : inout STD_LOGIC;
        DDR_cs_n : inout STD_LOGIC;
        DDR_dm : inout STD_LOGIC_VECTOR ( 3 downto 0 );
        DDR_dq : inout STD_LOGIC_VECTOR ( 31 downto 0 );
        DDR_dqs_n : inout STD_LOGIC_VECTOR ( 3 downto 0 );
        DDR_dqs_p : inout STD_LOGIC_VECTOR ( 3 downto 0 );
        DDR_odt : inout STD_LOGIC;
        DDR_ras_n : inout STD_LOGIC;
        DDR_reset_n : inout STD_LOGIC;
        DDR_we_n : inout STD_LOGIC;
        FIXED_IO_ddr_vrn : inout STD_LOGIC;
        FIXED_IO_ddr_vrp : inout STD_LOGIC;
        FIXED_IO_mio : inout STD_LOGIC_VECTOR ( 53 downto 0 );
        FIXED_IO_ps_clk : inout STD_LOGIC;
        FIXED_IO_ps_porb : inout STD_LOGIC;
        FIXED_IO_ps_srstb : inout STD_LOGIC);
end lab4_top;
```

LAB4_PCK.VHD

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

package lab4_pck is

  -- ---------------------------------------------------------------------------
  --          GENERAL
  -- ---------------------------------------------------------------------------

  constant PIF_ADDRESS_LENGTH              : natural :=  32;
  constant PIF_DATA_LENGTH                 : natural :=  32;

  constant LAB4REG_REV_STATUS_V            : std_logic_vector(7 downto 0) := x"50";  -- RO
U8,8
```

```
  constant LAB4REG_REV_LETTER_1_V           : std_logic_vector(7 downto 0) := x"31";  -- RO
U8,8
  constant LAB4REG_REV_LETTER_2_V           : std_logic_vector(7 downto 0) := x"31";  -- RO
U8,8

  -- ------------------------------------------------------------------------
  --          RAM BASE ADDRESSES
  -- ------------------------------------------------------------------------

  constant LAB4RAM_BASE_ADDRESS            : std_logic_vector(31 downto 0) :=
x"40400000";  -- RW U8,8



  -- ------------------------------------------------------------------------
  --          REGISTER ADDRESSES
  -- ------------------------------------------------------------------------

  constant LAB4REG_BASE_ADDRESS             : std_logic_vector(31 downto 0) :=
x"40000000";  --MODULE ADDRESS SPACE START

  -- LAB4REG register module addresses
  constant LAB4REG_REV_STATUS               : std_logic_vector(31 downto 0) := x"40000000";
-- RO U8,8
  constant LAB4REG_REV_LETTER_1             : std_logic_vector(31 downto 0) := x"40000004";
-- RO U8,8
  constant LAB4REG_REV_LETTER_2             : std_logic_vector(31 downto 0) := x"40000008";
-- RO U8,8
  constant LAB4REG_RWTEST                   : std_logic_vector(31 downto 0) := x"4000000C";  --
RW U32,32

  constant LAB4REG_SETPOINT                 : std_logic_vector(31 downto 0) := x"40000010";  --
RW U8,8

  constant LAB4REG_32                                           : std_logic_vector(31 downto
0) := x"40000014";  -- test 32
  constant LAB4REG_16                                           : std_logic_vector(31 downto
0) := x"40000018";  -- test 16

end lab4_pck;
```

CRU_RTL.VHD

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
library unisim;
use unisim.all;

architecture beh of cru is
   component bufg
    port(
              i : in std_logic;
              o : out std_logic
            );
   end component;

   component rstsynch is
    port (
              arst        : in std_logic;  --Asynch. restet
              mclk        : in std_logic;  -- Master clock
              mclk_div    : in std_logic;  -- Master clock div. by 128
              rst         : out std_logic; -- Sync. reset div. by 128
              rst_div     : out std_logic  -- Sync. reset div. by 128
           );

   end component rstsynch;

   component clkdiv is
    port (
              rst         : in std_logic;  -- Reset
              mclk        : in std_logic;  -- Master clock
              mclk_div    : out std_logic  -- Master clock div. by 128

           );
   end component clkdiv;

   signal rst_i, rst_local, rst_div_local, rst_div_i : std_logic;
   signal mclk_i, mclk_div_local, mclk_div_i : std_logic;

  begin

    bufg_0: bufg
             port map (
              i => refclk,
              o => mclk_i
              );
           rstsynch_0: rstsynch
    port map (
          arst     => arst,         --[in] Asynch. reset
                mclk     => mclk_i,       --[in] Master clock
                mclk_div => mclk_div_i,   --[in] Master clock div. by 128
                rst      => rst_local,    --[out] Synch. reset master clock
                rst_div  => rst_div_local --[out] Synch. reset mclk div. by 128
               );

       bufg_1: bufg
         port map (
          i => rst_local,
                o => rst_i
         );
       bufg_2: bufg
         port map (
          i => rst_div_local,
                o => rst_div_i
         );

       clkdiv_0: clkdiv
         port map (
              rst      => rst_i,         --[in] Reset
                mclk     => mclk_i,         --[in] Master clock
                mclk_div => mclk_div_local --[out] Master clock div. by 128
         );
       bufg_3: bufg
         port map (
          i => mclk_div_local,
                o => mclk_div_i
         );
         rst       <= rst_i;
   rst_div   <= rst_div_i;
   mclk      <= mclk_i;
   mclk_div  <= mclk_div_i;


end architecture beh;
```

CRU_ENT

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

```
entity cru is
  port (
    arst         : in  std_logic; -- Asynch. reset
              refclk       : in  std_logic; --Reference clock
              rst          : out  std_logic; --Synchronized arst_ for mclk
              rst_div      : out  std_logic; --Synchronized arst_ for mclk_div
              mclk         : out  std_logic; -- Master clock
              mclk_div     : out  std_logic -- Master clock div. by 128

    );
end cru;
```

CLKDIV_RTL

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

architecture dav of clkdiv is
  signal mclk_cnt : unsigned(6 downto 0);
 begin

   P_CLKDIV: process(rst, mclk)
   begin
     if rst = '1' then
              mclk_cnt <= (others => '0');
            elsif rising_edge(mclk) then
      mclk_cnt <= mclk_cnt + 1;
    end if;
   end process P_CLKDIV;

 mclk_div <= std_logic(mclk_cnt(6));

end dav;
```

CLKDIV_ENT

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

entity clkdiv is
```

```
 port (
         rst      : in std_logic; -- Restet
             mclk     : in std_logic; -- Master clock
             mclk_div : out std_logic-- Master clock div. by 128
    );

end clkdiv;
```

AXI4PIFB_RTL

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

library work;
use work.lab4_pck.all;

architecture rtl of axi4pifb is

  type fsm_state_type is (IDLE,
                         WR_ACCESS, WR_WAIT, WR_ACCESS_DONE, WR_COMPLETE,
                         RD_ACCESS, RD_DATA_WAIT, RD_ACCESS_DONE, RD_COMPLETE);
  signal fsm_state : fsm_state_type;

  -- AXI4LITE signals
  signal axi_awaddr      : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  signal axi_awready      : std_logic;
  signal axi_wready      : std_logic;
  signal axi_wdata       : std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal axi_wstrb      : std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
  signal axi_bresp       : std_logic_vector(1 downto 0);
  signal axi_bvalid      : std_logic;
  signal axi_araddr      : std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
  signal axi_arready     : std_logic;
  signal axi_rdata       : std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
  signal axi_rresp       : std_logic_vector(1 downto 0);
  signal axi_rvalid      : std_logic;

  -- Register read and write access acknowledge synch signals
  signal ack_2pif_s1, ack_2pif_s2 : std_logic;
  signal ack_2pif_d1, ack_2pif_str : std_logic;
  -- Reset synch signals
  signal pif_rst_s1, pif_rst_s2 : std_logic;

  -- Internal signals
  signal pif_regcs_i    : std_logic_vector(31 downto 0);
  signal pif_memcs_str  : std_logic_vector(31 downto 0);
  signal pif_addr_i     : std_logic_vector(PIF_ADDR_LENGTH-1 downto 0);
  signal pif_re_i       : std_logic_vector(0 downto 0);
  signal pif_we_i       : std_logic_vector(0 downto 0);
  signal ack_2pif       : std_logic;
  signal mem_ack_rd     : std_logic;
  signal mem_ack_wr     : std_logic;
  signal mem_ack_d1     : std_logic;
  signal mem_ack_str    : std_logic;
  signal mem_ack_str_d1 : std_logic;

begin

  P_RST_SYNCH_AXI_ACLK : process( s_axi_aclk, s_axi_aresetn )
  begin
    if s_axi_aresetn='0' then
      pif_rst_s1 <= '1';
      pif_rst_s2 <= '1';
    elsif rising_edge(s_axi_aclk) then
      pif_rst_s1 <= '0';
      pif_rst_s2 <= pif_rst_s1;
    end if;
  end process P_RST_SYNCH_AXI_ACLK;


  -- Synchronize the register chip select signal
  P_SYNCH_REGCS: process ( s_axi_aclk, s_axi_aresetn )
  begin
    if s_axi_aresetn='0' then
      ack_2pif_s1  <= '0';
      ack_2pif_s2  <= '0';
      ack_2pif_d1  <= '0';
      ack_2pif_str <= '0';
    elsif rising_edge(s_axi_aclk) then
      ack_2pif_s1  <= ack_2pif;
      ack_2pif_s2  <= ack_2pif_s1;
      ack_2pif_d1  <= ack_2pif_s2;
      ack_2pif_str <= (ack_2pif_s2 and (not ack_2pif_d1)) or mem_ack_str_d1;
  end if;
end process;


P_MEM_ACK_STR: process ( s_axi_aclk, s_axi_aresetn )
begin
  if s_axi_aresetn='0' then
    mem_ack_d1     <= '0';
    mem_ack_str    <= '0';
    mem_ack_str_d1 <= '0';
  elsif rising_edge(s_axi_aclk) then
    mem_ack_d1     <= mem_ack_rd or mem_ack_wr;
    mem_ack_str    <= (mem_ack_rd or mem_ack_wr) and not mem_ack_d1;
    mem_ack_str_d1 <= mem_ack_str;
  end if;
end process;


P_AXI4LITE_BRIDGE_FSM:
process ( s_axi_aclk, s_axi_aresetn ) is
  variable pif_memcs_i  : std_logic_vector(31 downto 0);
  variable pif_addr_sel : std_logic_vector(4 downto 0);
begin
  if ( s_axi_aresetn = '0') then
    axi_awready <= '0';
    axi_awaddr  <= (others => '0');
    axi_wready  <= '0';
    axi_wdata   <= (others => '0');
    axi_wstrb   <= (others => '0');
    axi_arready <= '0';
    axi_araddr  <= (others => '1');
    axi_bvalid  <= '0';
    axi_bresp   <= "00";
    axi_rvalid  <= '0';
    axi_rresp   <= "00";
    axi_rdata   <= (others => '0');

    pif_regcs_i   <= (others => '0');
    pif_memcs_i   := (others => '0');
    pif_memcs_str <= (others => '0');
    pif_addr_i    <= (others => '0');
    pif_wdata     <= (others => '0');
    -- Single bit, but vector type due to Xilinx generated RAM has vector type we signal.
    pif_we_i      <= (others => '0');
    -- Write strobes. This signal indicates which byte lanes hold
    --   valid data. There is one write strobe bit for each eight
    --   bits of the write data bus.
    pif_be            <= (others => '0');
    mem_ack_wr    <= '0';
    pif_re_i      <= (others => '0');
    mem_ack_rd    <= '0';

    fsm_state <= IDLE;

  elsif rising_edge( s_axi_aclk ) then

    -- Default values
    axi_awready <= '0';
    axi_wready  <= '0';
    axi_arready <= '0';

    case fsm_state is

      when IDLE  =>
        if (s_axi_awvalid = '1' and s_axi_wvalid = '1' and ack_2pif_s2='0') then
          axi_awready <= '1';
          axi_awaddr  <= s_axi_awaddr;
          axi_wready  <= '1';
          axi_wdata   <= s_axi_wdata;
          axi_wstrb   <= s_axi_wstrb;
          fsm_state   <= WR_ACCESS;
        elsif (s_axi_arvalid = '1' and ack_2pif_s2='0') then
          axi_arready <= '1';
          axi_araddr  <= s_axi_araddr;
          fsm_state   <= RD_ACCESS;
        end if;


      when  WR_ACCESS=>
        pif_addr_i   <= axi_awaddr(PIF_ADDR_LENGTH-1 downto 0);
        pif_wdata    <= axi_wdata(PIF_DATA_LENGTH-1 downto 0);
        pif_we_i     <= (others => '1');
        pif_be       <= axi_wstrb;
        pif_regcs_i  <= (others => '0');
        pif_memcs_i  := (others => '0');
        mem_ack_wr   <= '0';
    if axi_awaddr(25 downto 21)="00000" then
      pif_addr_sel:= axi_awaddr(20 downto 16);

      case pif_addr_sel is
        when LAB4REG_BASE_ADDRESS(20 downto 16) => pif_regcs_i(0) <= '1';
        when others =>
          mem_ack_wr <= '1';
          pif_regcs_i <= (others => '0');
      end case;
    else
      mem_ack_wr <= '1';
      pif_addr_sel:= axi_awaddr(25 downto 21);
      case pif_addr_sel is
        when LAB4RAM_BASE_ADDRESS(25 downto 21) => pif_memcs_i(0) := '1';
        when others =>  pif_memcs_i := (others => '0');
      end case;
    end if;
    fsm_state<= WR_WAIT;


  when WR_WAIT =>
    if (ack_2pif_str='1') then
      pif_regcs_i  <= (others => '0');
      pif_memcs_i  := (others => '0');
      pif_addr_i   <= (others => '0');
      pif_we_i     <= (others => '0');
      pif_be       <= (others => '0');
      mem_ack_wr   <= '0';
      axi_bvalid   <= '1';
      axi_bresp    <= "00";
      fsm_state    <= WR_ACCESS_DONE;
    end if;


  when WR_ACCESS_DONE =>
    if (s_axi_bready = '1') then
      axi_bvalid <= '0';
      fsm_state  <= WR_COMPLETE;
    end if;


  when WR_COMPLETE =>
    if (s_axi_arvalid = '1' and ack_2pif_s2='0') then
```

```
    axi_arready <= '1';
    axi_araddr  <= s_axi_araddr;
    fsm_state   <= RD_ACCESS;
  else
    fsm_state   <= IDLE;
  end if;


when RD_ACCESS =>
  pif_addr_i   <= axi_araddr(PIF_ADDR_LENGTH-1 downto 0);
  pif_re_i     <= (others => '1');
  pif_regcs_i  <= (others => '0');
  pif_memcs_i  := (others => '0');
  mem_ack_rd   <= '0';
  if axi_araddr(25 downto 21)="00000" then
    pif_addr_sel:= axi_araddr(20 downto 16);
    case pif_addr_sel is
      when LAB4REG_BASE_ADDRESS(20 downto 16) => pif_regcs_i(0) <= '1';
      when others =>
        mem_ack_rd    <= '1';
        pif_regcs_i  <= (others => '0');
    end case;
  else
    mem_ack_rd    <= '1';
    pif_addr_sel:= axi_araddr(25 downto 21);
    case pif_addr_sel is
      when LAB4RAM_BASE_ADDRESS(25 downto 21) => pif_memcs_i(0) := '1';
      when others =>  pif_memcs_i := (others => '0');
    end case;
  end if;
  fsm_state<= RD_DATA_WAIT;


when RD_DATA_WAIT =>
  if (ack_2pif_str='1') then
    pif_regcs_i  <= (others => '0');
    pif_memcs_i  := (others => '0');
    pif_addr_i   <= (others => '0');
    pif_re_i     <= (others => '0');
    mem_ack_rd   <= '0';
    if axi_araddr(25 downto 21)="00000" then
      pif_addr_sel:= axi_araddr(20 downto 16);
      axi_rdata <= rdata_lab4reg2pif;
```

```
              case pif_addr_sel is
                when LAB4REG_BASE_ADDRESS(20 downto 16) => axi_rdata <=
rdata_lab4reg2pif;
                when others =>  axi_rdata <= (others => '0');
              end case;
            else
              pif_addr_sel:= axi_araddr(25 downto 21);
              case pif_addr_sel is
                when LAB4RAM_BASE_ADDRESS(25 downto 21) => axi_rdata <=
mdata_lab4ram2pif;
                when others  => axi_rdata <= (others => '0');
              end case;
            end if;
            axi_rvalid <= '1';
            axi_rresp  <= "00"; -- 'OK' response
            fsm_state  <= RD_ACCESS_DONE;
          end if;


        when RD_ACCESS_DONE =>
          if (s_axi_rready = '1') then
               -- Read data is accepted by the master
            axi_rvalid <= '0';
            axi_rdata  <= (others => '0');
            fsm_state  <= RD_COMPLETE;
          end if;


        when RD_COMPLETE =>
          if (s_axi_awvalid = '1' and s_axi_wvalid = '1' and ack_2pif_s2='0') then
            axi_awready <= '1';
            axi_awaddr  <= s_axi_awaddr;
            axi_wready  <= '1';
            axi_wdata   <= s_axi_wdata;
            axi_wstrb   <= s_axi_wstrb;
            fsm_state   <= WR_ACCESS;
          else
            fsm_state   <= IDLE;
          end if;

      end case;

      -- Memory chip select is a strobe due FIFO read/write access
      for i in 0 to 31 loop
        pif_memcs_str(i) <= pif_memcs_i(i) and mem_ack_str;
      end loop;

    end if;

  end process P_AXI4LITE_BRIDGE_FSM;

  -- Concurrent statements

  -- Read and write input acknowledge for register modules
  ack_2pif  <=  ack_lab4reg2pif;

  -- Concurrent I/O connections assignments
  s_axi_awready         <= axi_awready;
  s_axi_wready          <= axi_wready;
  s_axi_bresp  <= axi_bresp;
  s_axi_bvalid  <= axi_bvalid;
  s_axi_arready         <= axi_arready;
  s_axi_rdata   <= axi_rdata;
  s_axi_rresp   <= axi_rresp;
  s_axi_rvalid  <= axi_rvalid;


  -- Concurrent statements
  pif_clk   <= s_axi_aclk;
  pif_rst   <= pif_rst_s2;
  pif_regcs <= pif_regcs_i;
  pif_memcs <= pif_memcs_str;
  pif_addr  <= pif_addr_i;
  pif_re    <= pif_re_i;
  pif_we    <= pif_we_i;

end rtl;
```

AXI4PIFB_ENT

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

library work;
```

```
use work.lab4_pck.all;

entity axi4pifb is

 generic (
   -- Width of PIF data bus
   PIF_DATA_LENGTH         : integer          := PIF_DATA_LENGTH;
   -- Width of PIF address bus
   PIF_ADDR_LENGTH         : integer          := PIF_ADDRESS_LENGTH;
   -- Width of S_AXI data bus
   C_S_AXI_DATA_WIDTH   : integer          := 32;
   -- Width of S_AXI address bus
   C_S_AXI_ADDR_WIDTH   : integer          := 32
 );

 port (

   -- AXI4LITE interface
   -- Global Clock Signal
   s_axi_aclk           : in std_logic;
   -- Global Reset Signal. This Signal is Active LOW
   s_axi_aresetn        : in std_logic;
   -- Write address (issued by master, acceped by Slave)
   s_axi_awaddr          : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
   -- Write channel Protection type. This signal indicates the
   -- privilege and security level of the transaction, and whether
   -- the transaction is a data access or an instruction access.
   s_axi_awprot         : in std_logic_vector(2 downto 0);
   -- Write address valid. This signal indicates that the master signaling
   -- valid write address and control information.
   s_axi_awvalid        : in std_logic;
   -- Write address ready. This signal indicates that the slave is ready
   -- to accept an address and associated control signals.
   s_axi_awready        : out std_logic;
   -- Write data (issued by master, acceped by Slave)
   s_axi_wdata                   : in std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
   -- Write strobes. This signal indicates which byte lanes hold
   -- valid data. There is one write strobe bit for each eight
   -- bits of the write data bus.
   s_axi_wstrb         : in std_logic_vector((C_S_AXI_DATA_WIDTH/8)-1 downto 0);
   -- Write valid. This signal indicates that valid write
   -- data and strobes are available.
   s_axi_wvalid           : in std_logic;
```

```
-- Write ready. This signal indicates that the slave
-- can accept the write data.
s_axi_wready          : out std_logic;
-- Write response. This signal indicates the status
-- of the write transaction.
s_axi_bresp          : out std_logic_vector(1 downto 0);
-- Write response valid. This signal indicates that the channel
-- is signaling a valid write response.
s_axi_bvalid          : out std_logic;
-- Response ready. This signal indicates that the master
-- can accept a write response.
s_axi_bready          : in std_logic;
-- Read address (issued by master, acceped by Slave)
s_axi_araddr          : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
-- Protection type. This signal indicates the privilege
-- and security level of the transaction, and whether the
-- transaction is a data access or an instruction access.
s_axi_arprot          : in std_logic_vector(2 downto 0);
-- Read address valid. This signal indicates that the channel
-- is signaling valid read address and control information.
s_axi_arvalid          : in std_logic;
-- Read address ready. This signal indicates that the slave is
-- ready to accept an address and associated control signals.
s_axi_arready          : out std_logic;
-- Read data (issued by slave)
s_axi_rdata          : out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
-- Read response. This signal indicates the status of the
-- read transfer.
s_axi_rresp          : out std_logic_vector(1 downto 0);
-- Read valid. This signal indicates that the channel is
-- signaling the required read data.
s_axi_rvalid          : out std_logic;
-- Read ready. This signal indicates that the master can
-- accept the read data and response information.
s_axi_rready          : in std_logic;

-- Register and memory processor interface (PIF)
-- Clock and reset signals
-- Clock, equal s_axi_aclk input
pif_clk          : out std_logic;
-- Reset signal, active HIGH and equal to inverted s_axi_aresetn
pif_rst          : out std_logic;
-- Register chip select
```

```
    pif_regcs            : out std_logic_vector(31 downto 0);
    -- Memory chip select
    pif_memcs            : out std_logic_vector(31 downto 0);
    -- Write address
    pif_addr             : out std_logic_vector(PIF_ADDR_LENGTH-1 downto 0);
    -- Write data
    pif_wdata            : out std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
    -- Read enable strobe
    pif_re               : out std_logic_vector(0 downto 0);
    -- Write enable strobe
    pif_we               : out std_logic_vector(0 downto 0);
    -- Write strobes. This signal indicates which byte lanes hold
    --   valid data. There is one write strobe bit for each eight
    --   bits of the write data bus.
    pif_be               : out std_logic_vector((PIF_DATA_LENGTH/8)-1 downto 0);
    -- Read data
    rdata_lab4reg2pif    : in std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
    mdata_lab4ram2pif    : in std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
    -- Register read and write access acknowledge
    ack_lab4reg2pif      : in std_logic
  );

end axi4pifb;
```

SUBPROG_PCK

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

package subprog_pck is
            function func (indata1 : in std_logic_vector(15 downto 0))
            return std_logic;
            function func (indata2 : in  unsigned(15 downto 0))
            return std_logic;

            procedure foc (signal indata1 : in std_logic_vector; signal par : out std_logic);
            procedure foc (signal indata2 : in unsigned; signal par : out std_logic);

            function hex2seg7 (signal STATE : in std_logic_vector(3 downto 0); signal SEL :
in std_logic)
            return std_logic_vector;
end subprog_pck;
```

```
package body subprog_pck is

        function func (indata1 : in std_logic_vector(15 downto 0))
        return std_logic is variable parity1 : std_logic;
                begin
                parity1 := '0';
                for i in indata1'range loop
                        if indata1(i) = '1' then
                        parity1 := not parity1;
                        end if;
                end loop;
                        return parity1;
                end;


        function func (indata2 : in  unsigned(15 downto 0))
        return std_logic is variable parity2 : std_logic;
        begin
        parity2 := '0';
    for j in indata2'range loop
     parity2 := parity2 xor indata2(j);
    end loop;
        return parity2;
        end;


        function hex2seg7 (signal STATE : in std_logic_vector(3 downto 0); signal SEL : in
std_logic)
        return std_logic_vector is variable char : std_logic_vector(7 downto 0);
        begin
         case STATE is
                when "0000" => char := "00000011";
                when "0001" => char := "10011111";
                when "0010" => char := "00100101";
                when "0011" => char := "00001101";
                when "0100" => char := "10011001";
                when "0101" => char := "01001001";
                when "0110" => char := "01000001";
                when "0111" => char := "00011111";
                when "1000" => char := "00000001";
                when "1001" => char := "00011001";
                when "1010" => char := "00010001";
                when "1011" => char := "11000001";
                when "1100" => char := "01100011";
                when "1101" => char := "10000101";
                when "1110" => char := "01100001";
                when "1111" => char := "01110001";

                when others => char := "00000000";

           end case;

           if (SEL = '1' ) then
                char(0) := '0';

           end if;

           return char;
          end;


          procedure foc (signal indata1 : in std_logic_vector; signal par : out std_logic) is
          variable parity1 : std_logic := '0';

          begin
          parity1 := '0';
    for i in indata1'range loop
      if indata1(i) = '1' then
        parity1 := not parity1;
      end if;
    end loop;
           par <= parity1;
end procedure foc;

          procedure foc (signal indata2 : in unsigned; signal par : out std_logic) is
          variable parity2 : std_logic;
          begin
    parity2 := '0';
    for j in indata2'range loop
     parity2 := parity2 xor indata2(j);
    end loop;
           par <= parity2;
end procedure foc;
end subprog_pck;
```

SEG7MODEL_ENT

```
-- Dette er entity for modell av sjusegmentdisplayene.  De er modellert ved at man
-- f�r vist ASCII-verdien av tallet/bokstaven som vises p� segmentene
-- Dersom man merker DISP0,..3 i waveform vieweren og velger radix ascii
-- F�r man vist tall/bokstav som vist p� sjusegmentene.

library IEEE;
use IEEE.std_logic_1164.all;

entity seg7model is
  port
  (
    a_n           : in  std_logic_vector(3 downto 0);
    abcdefgdec_n  : in  std_logic_vector(7 downto 0);
    disp3         : out std_logic_vector(3 downto 0);
    disp2         : out std_logic_vector(3 downto 0);
    disp1         : out std_logic_vector(3 downto 0);
    disp0         : out std_logic_vector(3 downto 0)
  );
end seg7model;
```

SEG7MODEL_BEH

```
-- Dette er modell for sjusegmentdisplayene.  De er modellert ved at man
-- f�r vist ASCII-verdien av tallet/bokstaven som vises p� segmentene
-- Dersom man merker disp0,..3 i waveform vieweren og velger radix ascii
-- F�r man vist tall/bokstav som vist p� sjusegmentene.

library IEEE;
use IEEE.std_logic_1164.all;

architecture beh of seg7model is
  signal char : std_logic_vector(3 downto 0);
begin
  display :
  process(a_n,char)
  begin
    --Default verdier
    --Benytter man default verdier kan man sl�yfe
    --else i if setninger uten � f� laget en latch
    --En annen fordel er at koden kan bli enklere.
    --Benytter 'Z'(h�y impendans) for � vise at et display er slukket
    disp0 <= "ZZZZ";
    disp1 <= "ZZZZ";
    disp2 <= "ZZZZ";
    disp3 <= "ZZZZ";
    if a_n(3) = '0' then
      disp3 <= char;
    end if;
    if a_n(2) = '0' then
      disp2 <= char;
    end if;
    if a_n(1) = '0' then
      disp1 <= char;
    end if;
    if a_n(0) = '0' then
      disp0 <= char;
    end if;
  end process dispLAY;

--De to metodene nedenfor er helt ekvivalente beskrivelser
--Legg merke til alternativ koding nederst dersom man bare vil vise tallverdier
--Kan v�re fint � benytte dersom man lager digitalklokke

--  ENCODE:
--  process (abcdefg_n)
--  begin
--    case abcdefg_n(7 downto 1) is
--      when "0000001" => char <= X"30"; --0
--      when "1001111" => char <= X"31"; --1
--      when "0010010" => char <= X"32"; --2
--      when "0000110" => char <= X"33"; --3
--      when "1001100" => char <= X"34"; --4
--      when "0100100" => char <= X"35"; --5
--      when "0100000" => char <= X"36"; --6
--      when "0001111" => char <= X"37"; --7
--      when "0000000" => char <= X"38"; --8
--      when "0001100" => char <= X"39"; --9
--      when "0001000" => char <= X"41"; --A
--      when "1100000" => char <= X"42"; --B
--      when "0110001" => char <= X"43"; --C
--      when "1000010" => char <= X"44"; --D
--      when "0110000" => char <= X"45"; --E
--      when "0111000" => char <= X"46"; --F
--      when "0000100" => char <= X"67"; --G
--      when "1101000" => char <= X"68"; --H
--      when "0000111" => char <= X"49"; --I
--      when "1000011" => char <= X"4A"; --J
--      when "1110001" => char <= X"4C"; --L
--      when "1101010" => char <= X"6E"; --n
--      when "1100010" => char <= X"6F"; --o
--      when "0011000" => char <= X"50"; --P
--      when "1111010" => char <= X"72"; --r
--      when "1110000" => char <= X"74"; --t
--      when "1100011" => char <= X"75"; --u
--      when "1000100" => char <= X"59"; --Y
--      when others    => char <= "XXXXXXXX";
--    end case;
--  end process encode;

--  with abcdefgdec_n(7 downto 1) select
--      char <= X"30" when "0000001", --0
--              X"31" when "1001111", --1
--              X"32" when "0010010", --2
--              X"33" when "0000110", --3
--              X"34" when "1001100", --4
--              X"35" when "0100100", --5
--              X"36" when "0100000", --6
--              X"37" when "0001111", --7
--              X"38" when "0000000", --8
--              X"39" when "0001100", --9
--              X"41" when "0001000", --A
--              X"42" when "1100000", --B
--              X"43" when "0110001", --C
--              X"44" when "1000010", --D
--              X"45" when "0110000", --E
--              X"46" when "0111000", --F
--              X"67" when "0000100", --G
--              X"68" when "1101000", --H
--              X"49" when "0000111", --I
--              X"4A" when "1000011", --J
--              X"4C" when "1110001", --L
--              X"6E" when "1101010", --n
--              X"6F" when "1100010", --o
--              X"50" when "0011000", --P
--              X"72" when "1111010", --r
--             X"74" when "1110000", --t
--             X"75" when "1100011", --u
--             X"59" when "1000100", --Y
--             "XXXXXXXX" when others;

-- Eventuelt kan det v�re hensiktsmessig � vise bare hexadesimale tall 0-F

  with abcdefgdec_n(7 downto 1) select
     char <= X"0" when "0000001", --0
            X"1" when "1001111", --1
            X"2" when "0010010", --2
            X"3" when "0000110", --3
            X"4" when "1001100", --4
            X"5" when "0100100", --5
            X"6" when "0100000", --6
            X"7" when "0001111", --7
            X"8" when "0000000", --8
            X"9" when "0000100", --9
            X"A" when "0001000", --A
            X"B" when "1100000", --B
            X"C" when "0110001", --C
            X"D" when "1000010", --D
            X"E" when "0110000", --E
            X"F" when "0111000", --F
            "XXXX" when others;

end architecture beh;
```

SEG7CTRL

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

use work.subprog_pck.all;

entity seg7ctrl is
port
(
mclk        : in std_logic; --100MHz, positive flank
reset       : in std_logic; --Asynchronous reset, activeh
d0          : in std_logic_vector(3 downto 0); -- first display?
d1          : in std_logic_vector(3 downto 0); -- second display?
d2          : in std_logic_vector(3 downto 0); -- third display?
d3          : in std_logic_vector(3 downto 0); -- fourth display?
dec         : in std_logic_vector(3 downto 0); -- pumktum
abcdefgdec_n : out std_logic_vector(7 downto 0); -- what to be displayed inverted
a_n         : out std_logic_vector(3 downto 0) -- select display
);
end entity seg7ctrl;

architecture disp of seg7ctrl is

begin

        Process (mclk, reset)
        variable counter : unsigned(15 downto 0) := (others => '0');
        begin

        if reset = '1' then
                -- reset all
                counter := (others => '0');
                abcdefgdec_n <= "00000000";
                a_n <= "0000";


        elsif rising_edge(mclk) then
                counter := counter + 1;

                case counter(15 downto 14) is
                        when "00" =>
                                a_n <= "1110";
                                abcdefgdec_n <= hex2seg7(d0, dec(0));
                        when "01" =>
                                a_n <= "1101";
                                abcdefgdec_n <= hex2seg7(d1, dec(1));
                        when "10" =>
                                a_n <= "1011";
                                abcdefgdec_n <= hex2seg7(d2, dec(2));
                        when others =>
                                a_n <= "0111";
                                abcdefgdec_n <= hex2seg7(d3, dec(3));
                end case;

        end if;
        end process;
```

```
end;
```

RSTSYNCH_RTL

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

architecture dav of rstsynch is
  signal rst_s1, rst_s2 : std_logic;
  signal rst_div_s1, rst_div_s2 : std_logic;

 begin

   P_RST_0: process(arst, mclk)
   begin
    if arst = '1' then
                rst_s1 <= '1';
                          rst_s2 <= '1';
            elsif rising_edge(mclk) then
        rst_s1 <= '0';
                            rst_s2 <= rst_s1;
            end if;
   end process P_RST_0;

   P_RST_1: process (arst, mclk_div)
   begin
    if arst = '1' then
                 rst_div_s1 <= '1';
                 rst_div_s2 <= '1';
            elsif rising_edge(mclk_div) then
     rst_div_s1 <= '0';
     rst_div_s2 <= rst_div_s1;

    end if;
   end process P_RST_1;

   rst <= rst_s2;
   rst_div <= rst_div_s2;

end architecture dav;
```

RSTSYNCH_ENT

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

entity rstsynch is
  port (
    arst        : in  std_logic; -- Asynch. reset
            mclk        : in  std_logic; -- Master clock
            mclk_div     : in  std_logic; -- Master clock div. by 128
            rst          : out  std_logic; --Synch. reset master clock
            rst_div      : out  std_logic --Synch. reset div. by 128

    );
end rstsynch;
```

POS_SEG7_CTRL

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


entity pos_seg7_ctrl is
  port (
    -- System Clock and Reset
    arst         : in  std_logic;       -- Reset
    sync_rst     : in  std_logic;       -- Synchronous reset
    refclk       : in  std_logic;       -- Clock
    sp           : in  std_logic_vector(7 downto 0);  -- Set Point
    a            : in  std_logic;       -- From position sensor
    b            : in  std_logic;       -- From position sensor
    force_cw     : in  std_logic;       -- Force motor clock wise motion
    force_ccw    : in  std_logic;                         -- Force motor counter clock wise motion
    motor_cw     : out std_logic;       -- Motor clock wise motion
    motor_ccw    : out std_logic;       -- Motor counter clock wise motion
    -- Interface to seven segments
    abcdefgdec_n : out std_logic_vector(7 downto 0);
    a_n          : out std_logic_vector(3 downto 0)
    );
end pos_seg7_ctrl;
```

POS_SEG7_CTRL

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

architecture pos_seg7_ctrl of pos_seg7_ctrl is

 component pos_ctrl is
   port (
        rst      : in  std_logic;         -- Reset
   rst_div   : in  std_logic;         -- Reset
   mclk      : in  std_logic;         -- Clock
   mclk_div  : in  std_logic;         -- Clock to p_reg
   sync_rst  : in  std_logic;         -- Synchronous reset
   sp        : in  signed(7 downto 0);  -- Setpoint (wanted position)
   a         : in  std_logic;         -- From position sensor
   b         : in  std_logic;         -- From position sensor
   pos       : out signed(7 downto 0);  -- Measured Position
   force_cw  : in  std_logic;         -- Force motor clock wise motion
   force_ccw : in  std_logic;  -- Force motor counter clock wise motion
   motor_cw  : out std_logic;         -- Motor clock wise motion
   motor_ccw : out std_logic          -- Motor counter clock wise motion
        );
  end component pos_ctrl;


component seg7ctrl is
  port
  (
  mclk        : in std_logic; --100MHz, positive flank
        reset       : in std_logic; --Asynchronous reset, activeh
        d0          : in std_logic_vector(3 downto 0); -- first display?
        d1          : in std_logic_vector(3 downto 0); -- second display?
        d2          : in std_logic_vector(3 downto 0); -- third display?
        d3          : in std_logic_vector(3 downto 0); -- fourth display?
        dec         : in std_logic_vector(3 downto 0); -- pumktum
        abcdefgdec_n : out std_logic_vector(7 downto 0); -- what to be displayed inverted
        a_n         : out std_logic_vector(3 downto 0) -- select display
  );
end component seg7ctrl;


component cru is
port (
    arst         : in  std_logic;
    refclk : in std_logic;
    rst_div : out std_logic;
    rst   : out std_logic;
    mclk_div  : out std_logic;
    mclk  : out std_logic
);
end component cru;

            signal posi : signed(7 downto 0);
            signal rst         : std_logic;
            signal mclk      : std_logic;
            signal mclk_div           : std_logic;
            signal rst_div   : std_logic;
            signal sp1 : std_logic_vector(7 downto 0);


begin

sp1 <= '0' & sp(6 downto 0);

  O2: entity work.cru
      port map (
      arst          => arst,
      refclk => refclk,
      rst_div => rst_div,
      rst   => rst,
      mclk_div  => mclk_div,
      mclk  => mclk
      );

O: entity work.pos_ctrl
    port map (sync_rst => sync_rst,
                                rst_div => rst_div,
                                mclk_div => mclk_div,
                                sp => signed(sp1),
                                force_cw => force_cw,
                                force_ccw => force_ccw,
                                motor_cw => motor_cw,
                                motor_ccw => motor_ccw,
            mclk => mclk,
            a => a,
            b => b,
                                rst => rst,
            pos => posi);

O1: entity work.seg7ctrl
    port map (
            abcdefgdec_n => abcdefgdec_n,
            mclk => mclk,
            reset => rst,
            d0 => std_logic_vector(posi(3 downto 0)),
            d1 => std_logic_vector(posi(7 downto 4)),
            d2 => sp1(3 downto 0),
            d3 => sp1(7 downto 4),
            dec => "0000",
            a_n => a_n);




      --force_ccw <= BTNL;
      --force_cw <= BTNR;
      --sync_rst <= BTNC;




end architecture;
```

POS_MEAS_ENT

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity pos_meas is
  port (
    -- System Clock and Reset
    rst      : in  std_logic;           -- Reset
    clk      : in  std_logic;           -- Clock
    sync_rst : in  std_logic;           -- Sync reset
    a        : in  std_logic;           -- From position sensor
    b        : in  std_logic;           -- From position sensor
    pos      : out signed(7 downto 0)   -- Measured position
    );
end pos_meas;
```

POS_MEAS_BEH

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


architecture pos_beh of pos_meas is

TYPE State_type IS (start_up_st, wait_a1_st, wait_a0_st, up_down_st, count_up_st,
count_down_st);
        Signal State : State_type;

begin

        process(a, b, sync_rst, clk)
                variable posi : signed(7 downto 0) := "00000000";
                variable counter : unsigned(1 downto 0) := "00";
        begin

        if rst = '1' then
                posi := "00000000";
                State <= start_up_st;

        elsif rising_edge(clk) then
        if sync_rst = '1' then
                posi := "00000000";
                State <= start_up_st;
        end if;

        case state is
                when start_up_st =>
                        if a = '1' then
                                State <= wait_a0_st;
                        else
                                State <= wait_a1_st;
                        end if;

                when wait_a0_st =>
                        if a='1' then
                                State <= wait_a0_st;
                        else
                State <= up_down_st;
            end if;
        when wait_a1_st =>
            if a='1' then
                State <= wait_a0_st;
            else
                State <= wait_a1_st;
            end if;
        when up_down_st =>
            if b='1' then
                State <= count_down_st;
            else
                State <= count_up_st;
            end if;
        when count_down_st =>
                    if posi > 0 then
                        posi := posi - 1;
                    end if;
            State <= wait_a1_st;
        when count_up_st =>

                if posi < 127 then
                        posi := posi + 1;
                    end if;
            State <= wait_a1_st;


        end case;
    end if;
    pos <= posi;
    end process;

end architecture pos_beh;
```

POS_CTRL_ENT

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

entity pos_ctrl is
  port (
    -- System Clock and Reset
    rst       : in  std_logic;          -- Reset
    rst_div   : in  std_logic;          -- Reset
    mclk      : in  std_logic;          -- Clock
    mclk_div  : in  std_logic;          -- Clock to p_reg
    sync_rst  : in  std_logic;          -- Synchronous reset
    sp        : in  signed(7 downto 0);  -- Setpoint (wanted position)
    a         : in  std_logic;          -- From position sensor
    b         : in  std_logic;          -- From position sensor
    pos       : out signed(7 downto 0);  -- Measured Position
    force_cw  : in  std_logic;          -- Force motor clock wise motion
    force_ccw : in  std_logic;  -- Force motor counter clock wise motion
    motor_cw  : out std_logic;          -- Motor clock wise motion
    motor_ccw : out std_logic           -- Motor counter clock wise motion
    );
end pos_ctrl;
```

POS_CTRL

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

architecture pos_ctrl of pos_ctrl is

  signal cw : std_logic;
  signal ccw : std_logic;

 component p_ctrl is
    port (
         clk : in std_logic;
         rst : in std_logic;
         sp : in signed(7 downto 0);
         pos : in signed(7 downto 0);
         cw  : out std_logic;           --Motor Clock Wise direction
         ccw : out std_logic
         );
  end component p_ctrl;

  component pos_meas_beh is
    port (a : in std_logic;
         b : in std_logic;
         sync_rst : in std_logic;
         clk : in std_logic;
```

```
            rst : in std_logic;
            pos : out signed(7 downto 0));
  end component pos_meas_beh;

           signal sp1 : signed(7 downto 0 );
           signal postemp : signed(7 downto 0);

begin

--mask away sp bit
sp1 <= '0' & sp(6 downto 0);


           Q: entity work.pos_meas
    port map (sync_rst => sync_rst,
           clk => mclk,
           a => a,
           b => b,
                            rst => rst,
           pos => postemp);

           Q1: entity work.p_ctrl
    port map (clk => mclk_div,   -- implement its own clock
                            motor_cw => cw,
                            motor_ccw => ccw,
                            rst => rst_div,
                            sp => sp1,
           pos => postemp);


           process (mclk, rst)
           begin
           -- forced running of motor.
           if force_cw = '1' then
                     if force_ccw = '1' then
                            motor_cw <= cw; -- from p_ctrl
                            motor_ccw <= ccw;
                     else
                            motor_cw <= '1';
                            motor_ccw <= '0';
                     end if;
           else
                     if force_ccw = '0' then
```

```
                        motor_cw <= cw; -- from p_ctrl
                        motor_ccw <= ccw;
                else
                        motor_cw <= '0';
                        motor_ccw <= '1';
                end if;
        end if;
        pos <= postemp;
        end process;


end architecture pos_ctrl;
```

P_CTRL_ENT

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

entity p_ctrl is
  port (
    -- System Clock and Reset
    rst       : in  std_logic;            -- Reset
    clk       : in  std_logic;            -- Clock
    sp        : in  signed(7 downto 0);  -- Set Point
    pos       : in  signed(7 downto 0);  -- Measured position
    motor_cw  : out std_logic;            --Motor Clock Wise direction
    motor_ccw : out std_logic             --Motor Counter Clock Wise direction
    );
end p_ctrl;
```

P_CTRL

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



        --rst       : in  std_logic;            -- Reset
    --clk       : in  std_logic;            -- Clock
    --sp        : in  signed(7 downto 0);  -- Set Point
    --pos       : in  signed(7 downto 0);  -- Measured position
    --motor_cw  : out std_logic;            --Motor Clock Wise direction
```

```
   --motor_ccw : out std_logic            --Motor Counter Clock Wise direction

architecture pos_ctrl of p_ctrl is
        TYPE State_type IS (idle_st, sampel_st, motor_st);
        Signal State : State_type;


begin

-- read pos from pos_meas_beh
-- make motor go to pos (SP) setpoint
        process (clk, rst, sp, pos) is
        variable err : signed(7 downto 0);
        begin
        if rst = '1' then
                motor_cw <= '0';
                motor_ccw <= '0';
                State <= idle_st;
        elsif rising_edge(clk) then

        case state is
                when idle_st =>
                        motor_ccw <= '0';
                        motor_cw <= '0';
                        State <= sampel_st;
                when sampel_st =>
                        err := sp - pos;
                        State <= motor_st;
                when motor_st =>
                        if err > 0 then
                                motor_cw <= '1';
                                motor_ccw <= '0';
                                State <= sampel_st;
                        elsif err < 0 then
                                motor_cw <= '0';
                                motor_ccw <= '1';
                                State <= sampel_st;
                        else
                                State <= idle_st;
                        end if;
                end case;
        end if;
```

```
            end process;


end architecture pos_ctrl;
```

MOTOR_ENT

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

entity motor is
  generic (
    phase90 : time := 50 us
    );
  port (
    run       : in  std_logic;
    motor_cw  : in  std_logic;
    motor_ccw : in  std_logic;
    a         : out std_logic;
    b         : out std_logic
    );
end motor;
```

MOTOR_BEH

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

architecture motor_beh of motor is

begin

  motor_moving : process
  begin

    a <= '0';
    b <= '0';

    while run='1' loop
      if motor_cw = '1' and motor_ccw = '0' then
        a <= '0';
        wait for phase90;
        b <= '1';
        wait for phase90;
        a <= '1';
        wait for phase90;
        b <= '0';
      elsif motor_ccw = '1' and motor_cw = '0' then
        a <= '1';
        wait for phase90;
        b <= '1';
        wait for phase90;
        a <= '0';
        wait for phase90;
        b <= '0';
      end if;
      wait for phase90;
    end loop;

    wait;

  end process;

end architecture motor_beh;
```

LAB4REG_RTL

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

library work;
use work.lab4_pck.all;


architecture rtl of lab4_reg is

  -- Internal registers
  signal pif_regcs_s1, pif_regcs_s2 : std_logic;
  signal reg_data_out  : std_logic_vector(PIF_DATA_LENGTH-1 downto 0);

  -- Register read and write access acknowledge
  signal regrdack_2pif  : std_logic;
  signal regwrack_2pif  : std_logic;
```

```
 -- Internal registers
 signal rwtest_i       : std_logic_vector(31 downto 0);
 signal setpoint_i     : std_logic_vector(7 downto 0);

 signal test_32                    : std_logic_vector(31 downto 0);
 signal test_16                    : std_logic_vector(15 downto 0);

begin

 -- Synchronize the register chip select signal
 P_SYNCH_REGCS: process (pif_rst, pif_clk)
 begin
   if pif_rst='1' then
     pif_regcs_s1 <= '0';
     pif_regcs_s2 <= '0';
   elsif rising_edge(pif_clk) then
     pif_regcs_s1 <= pif_regcs;
     pif_regcs_s2 <= pif_regcs_s1;
   end if;
 end process;

 -- Memory mapped register write logic
 P_WRITE: process (pif_rst, pif_clk)
 begin
     if pif_rst = '1' then

         regwrack_2pif  <= '0';
         rwtest_i       <= (others => '0');
         setpoint_i     <= (others => '0');
                        test_32 <= (others => '0');
                        test_16 <= (others => '0');

     elsif rising_edge(pif_clk) then

       -- Default values
       -- TBD.

       if (pif_regcs_s2='1' and pif_we(0) = '1') then

         -- Register write acknowledge
         regwrack_2pif <= '1';
```

```
if pif_be(0)='1' then
  if pif_addr(15 downto 2) = LAB4REG_RWTEST(15 downto 2) then
    rwtest_i(7 downto 0) <=  pif_wdata(7 downto 0);
                         end if;
                        if pif_addr(15 downto 2) = LAB4REG_32(15 downto 2) then
                                    test_32 (7 downto 0) <=  pif_wdata(7 downto 0);
                                     end if;
                        if pif_addr(15 downto 2) = LAB4REG_16(15 downto 2) then
                                    test_16 (7 downto 0) <=  pif_wdata(7 downto 0);

  end if;
  if pif_addr(15 downto 2) = LAB4REG_SETPOINT(15 downto 2) then
    setpoint_i(7 downto 0) <=  pif_wdata(7 downto 0);
  end if;
end if;

if pif_be(1)='1' then
  if pif_addr(15 downto 2) = LAB4REG_RWTEST(15 downto 2) then
                                    rwtest_i(15 downto 8) <=  pif_wdata(15 downto 8);
                                     end if;
                        if pif_addr(15 downto 2) = LAB4REG_32(15 downto 2) then
                                    test_32 (15 downto 8) <=  pif_wdata(15 downto 8);
                         end if;
                        if pif_addr(15 downto 2) = LAB4REG_16(15 downto 2) then
                                    test_16 (15 downto 8) <=  pif_wdata(15 downto 8);
  end if;
end if;

if pif_be(2)='1' then
  if pif_addr(15 downto 2) = LAB4REG_RWTEST(15 downto 2) then
    rwtest_i(23 downto 16) <=  pif_wdata(23 downto 16);
                         end if;
                        if pif_addr(15 downto 2) = LAB4REG_32(15 downto 2) then
                                    test_32 (23 downto 16) <=  pif_wdata(23 downto 16);
  end if;
end if;

if pif_be(3)='1' then
  if pif_addr(15 downto 2) = LAB4REG_RWTEST(15 downto 2) then
    rwtest_i(31 downto 24) <=  pif_wdata(31 downto 24);
                         end if;
                        if pif_addr(15 downto 2) = LAB4REG_32(15 downto 2) then
                                    test_32 (31 downto 24) <=  pif_wdata(31 downto 24);
```

```
          end if;
        end if;

      elsif (pif_regcs_s2='0') then
        regwrack_2pif <= '0';
      end if;
    end if;


end process;


-- Combinational memory mapped register read logic
P_READ: process (pif_regcs_s2, pif_re, pif_addr,
                 rwtest_i, setpoint_i, test_32, test_16)
  begin
    -- Address decoding for reading registers
    if ( pif_regcs_s2='1' and pif_re(0)='1' ) then

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

      if pif_addr(15 downto 2) = LAB4REG_RWTEST(15 downto 2) then
        reg_data_out <= rwtest_i;
      end if;

                  if pif_addr(15 downto 2) = LAB4REG_32(15 downto 2) then
        reg_data_out <= test_32;
      end if;
                   if pif_addr(15 downto 2) = LAB4REG_16(15 downto 2) then
        reg_data_out(15 downto 0) <= test_16;
      end if;

      if pif_addr(15 downto 2) = LAB4REG_SETPOINT(15 downto 2) then
        reg_data_out(7 downto 0) <= setpoint_i;
      end if;

    else
      reg_data_out  <= (others => '0');
    end if;
end process P_READ;


-- Read register output
```

```
  P_READ_OUT: process( pif_rst, pif_clk ) is
  begin
    if ( pif_rst = '1' ) then
      rdata_2pif  <= (others => '0');
      regrdack_2pif <= '0';
    elsif (rising_edge (pif_clk)) then
      if (pif_regcs_s2 = '1' and pif_re(0) = '1') then
        -- Register read data
        rdata_2pif <= reg_data_out;
        -- Register read acknowledge
        regrdack_2pif <= '1';
      elsif (pif_regcs_s2 = '0') then
        rdata_2pif  <= (others => '0');
        regrdack_2pif <= '0';
      end if;
    end if;
  end process P_READ_OUT;


  -- Concurrent statements

  ack_2pif <= regrdack_2pif or regwrack_2pif;

  setpoint <= setpoint_i;

end rtl;
```

LAB4REG_ENT

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

library work;
use work.lab4_pck.all;

entity lab4_reg is
  generic (
    -- Width of PIF data bus
    PIF_DATA_LENGTH         : integer := PIF_DATA_LENGTH;
    -- Width of PIF address bus
    PIF_ADDR_LENGTH         : integer := PIF_ADDRESS_LENGTH
    );

  port (
    -- Add ports here:
    setpoint        : out std_logic_vector(7 downto 0);
    -- Add ports ends

    -- Do not modify the ports beyond this line
    -- Clock Signal
    pif_clk         : in std_logic;
    -- Reset Signal. This signal is active HIGH
    pif_rst         : in std_logic;
    -- Register chip select
    pif_regcs       : in std_logic;
    -- Write address
    pif_addr        : in std_logic_vector(PIF_ADDR_LENGTH-1 downto 0);
    -- Write data
    pif_wdata       : in std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
    -- Read enable strobe
    pif_re          : in std_logic_vector(0 downto 0);
    -- Write enable strobe
    pif_we          : in std_logic_vector(0 downto 0);
    -- Write strobes. This signal indicates which byte lanes hold
    --   valid data. There is one write strobe bit for each eight
    --   bits of the write data bus.
    pif_be          : in std_logic_vector((PIF_DATA_LENGTH/8)-1 downto 0);
    -- Read data
    rdata_2pif      : out std_logic_vector(PIF_DATA_LENGTH-1 downto 0);
    -- Register read and write access acknowledge
    ack_2pif        : out std_logic

  );

end lab4_reg;
```