```
---------------------------------Author: Xiaofei Dong--------------------
---------------------------------Course: ee552---------------------------
---------------------------------Instructor: Dr. Elliott-----------------
---------------------------------Group: CDMA based communication system--

-- This is an incomplete adder. For n-bit input, regular adders have
-- 2^n possible inputs of addends, here only 2^(n-1) possibilities.
-- The n-bit incomplete adder in implemented with (n-1)-bit adder and
--- multiplexer.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library work;
use work.CDMA_pkg.all;

entity adderinc is

  generic (
    width : positive);

  port (
    adden1 : in  std_logic_vector(width-1 downto 0);
    adden2 : in  std_logic_vector(width-1 downto 0);
    sum    : out std_logic_vector(width downto 0));

end adderinc;

architecture behavioural of adderinc is
  signal lowersum : std_logic_vector(width -1 downto 0);
  signal zeros : std_logic_vector(width-1 downto 0);
  signal sel : std_logic_vector(1 downto 0);

begin  -- behavioural

  zeros <= (others =>'0');
  sel <= adden1(width-1) & adden2(width-1);

    selects: process
    begin  -- process
      case sel is
        when "00" => sum <= '0' & lowersum ;
        when "01" => sum <= "01" & adden1(width-2 downto 0);
        when "10" => sum <= "01" & adden2(width-2 downto 0);
        when "11" => sum <= '1' & zeros;
        when others => null;
      end case;
    end process;

      lowersum <= adden1(width-2 downto 0) + adden2(width-2 downto 0);

end behavioural;

-- Author: Xiaofei Dong
-- Course: ee552
```

```
-- Instructor: Dr. Elliot
-- Group:  CDMA based communication System
-- Synchronizer.vhd
-- the binary pattern correlator with four stages of pipelines;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library work;
use work.CDMA_pkg.all;

entity synchronizer is

  generic (
    code_width : positive := 32);

-- sample at 4 times the transmitted bps

  port (
    reset, clock             : in  std_logic;
    device_en                : in  std_logic;
    Rx                       : in  std_logic;
    frame_bit                : out std_logic;
    bit_ready                : out std_logic);
end synchronizer;

architecture mixed of synchronizer is
constant adden2 : std_logic_vector(code_width-1 downto 0) :=  x"0fff00f0";

  signal bit_in, flag : std_logic;
  signal ones,zeros : std_logic ;
  signal adden1 :std_logic_vector(code_width-1 downto 0);
  signal bit_ready_internal : std_logic;
  signal distance : std_logic_vector(5 downto 0);
  signal temp0 : std_logic;
  signal d_delayed : std_logic_vector(2 downto 0);
  signal weight_vec : std_logic_vector(code_width-1 downto 0);

begin  -- mixed

  ones <= '1';
  zeros <= '0';

-- s2preg is an N-bit serial_to_parallel shift register;
  serial2parallel: s2preg generic map (
    WIDTH     => code_width)
    port map (
      clock     => clock,
      reset     => reset,
      enable    => device_en,          -- shift in enable
      bitin     => Rx,
      frameout  => adden1);

      -- correlating two sequences
      modulo2: process
begin  -- process
```

```
      weight_vec <= adden1 xnor adden2;
      end process;

  -- four stages of pipelines;
  -- pipelinemeter is the summing network that contains the pipelined
  -- incomeplete adders;

  measure : pipelinedmeter port map (
    reset         => reset,
    clock         => clock,
    weight_vector => weight_vec,
    distance      => distance);

-- decision unit that decides whether frame has been synchronized;

  decision: process
  begin  -- process
    if reset = '1' then
      flag <= '0';
    elsif distance = X"20" then
      flag <= '1';
    else
      flag <= '0';
    end if;
    bit_ready_internal <= flag;
  end process;

-- the handshaking signal indicating the frame has been synchronized is
-- synchronized with the system clock;
  latchout : dff_en port map (
    clock  => clock,
    reset  => reset,
    enable => ones,
    d      => bit_ready_internal,
    q      => bit_ready);

-- because four pipeline stages are used, the output frame bit is the
-- fourth bit of the serial_to_parallel register;

frame_bit <= adden1(3);

end behavioural;



-- pipelined signal distance finder.  four pipeline registers are
-- inserted after every incomplete adder bank;

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
library work;
use work.CDMA_pkg.all;

entity pipelinedmeter is
```

```
  generic (
    width : positive := 32);

  port (
    reset         : in  std_logic;
    clock         : in  std_logic;
    weight_vector : in  std_logic_vector(width -1 downto 0);
    distance      : out std_logic_vector(5 downto 0));

end pipelinedmeter;

architecture mixed of pipelinedmeter is

  signal ones,zeros : std_logic;
  signal x : std_logic_vector(113 downto 0);
  signal y : std_logic_vector(87 downto 0);

begin  -- mixed

  ones <='1';
  zeros<='0';


  -- the bits are xor 'ed in pairs;
  first_stage: for i in 0 to width/2 -1 generate
    constant index0 : positive := 2*i;
    constant index1 : positive := 2*i+1;
    begin
      first_stage: half_adder port map (
        a => x(index0),
        b => x(index1),
        c => y(index1),
        s => y(index0));
    end generate first_stage;

        reg : latchreg generic map (
          width => 32)
          port map (
            clock        => clock,
            reset        => reset,
            enable       => ones,
            parallel_in  => y(31 downto 0),
            parallel_out => x(63 downto 32));

  -- every two bits are operated like they are summed;
  sums: for i in 1 to 4 generate
    begin
    con1: if i = 1 generate
      constant stage_x_start : positive := 32;
      constant stage_y_start : positive := 32;
      constant adder_size : positive := 2;
      constant adder_queue : positive := 8;
      begin
        adderqueue1: for j in 0 to adder_queue-1 generate
        constant blocks : positive := adder_size+1;
        begin
        adderi : adderinc generic map (
```

```
        width => adder_size)
        port map (
          adden1 => x(stage_x_start+(2*j+1)*adder_size-1 downto stage_x_start+
2*j*adder_size),
          adden2 => x(stage_x_start+2*(j+1)*adder_size-1 downto stage_x_start +
(2*j+1)*adder_size),
          sum    => y(stage_y_start+(j+1)*blocks-1 downto
stage_y_start+j*blocks));
      end generate adderqueue1;
        reg : latchreg generic map (
          width => 24)
          port map (
            clock        => clock,
            reset        => reset,
            enable       => ones,
            parallel_in  => y(55 downto 32),
            parallel_out => x(87 downto 64));
    end generate con1;

    con2: if i=2 generate
      constant stage_x_start : positive := 64;
      constant stage_y_start : positive := 56;
      constant adder_size : positive := 3;
      constant adder_queue : positive := 4;
      begin
       adderqueue2: for j in 0 to adder_queue-1 generate
        constant blocks : positive := adder_size+1;
        begin
        adderi : adderinc generic map (
        width => adder_size)
        port map (
          adden1 => x(stage_x_start+(2*j+1)*adder_size-1 downto stage_x_start+
2*j*adder_size),
          adden2 => x(stage_x_start+2*(j+1)*adder_size-1 downto stage_x_start +
(2*j+1)*adder_size),
          sum    => y(stage_y_start+(j+1)*blocks-1 downto
stage_y_start+j*blocks));
      end generate adderqueue2;
        -- insert pipeline regsiter;
        reg : latchreg generic map (
          width => 16)
          port map (
            clock        => clock,
            reset        => reset,
            enable       => ones,
            parallel_in  => y(71 downto 56),
            parallel_out => x(103 downto 88));

    end generate con2;

    con3: if i=3 generate
      constant stage_x_start : positive := 88;
      constant stage_y_start : positive := 72;
      constant adder_size : positive := 4;
      constant adder_queue : positive := 2;
      begin
      adderqueue3: for j in 0 to adder_queue-1 generate
```

```
        constant blocks : positive := adder_size+1;
        begin
        adderi : adderinc generic map (
        width => adder_size)
        port map (
          adden1 => x(stage_x_start+(2*j+1)*adder_size-1 downto stage_x_start+
2*j*adder_size),
          adden2 => x(stage_x_start+2*(j+1)*adder_size-1 downto stage_x_start +
(2*j+1)*adder_size),
          sum    => y(stage_y_start+(j+1)*blocks-1 downto
stage_y_start+j*blocks));
      end generate adderqueue3;

      --insert pipeline register
      reg : latchreg generic map (
        width => 10)
        port map (
          clock        => clock,
          reset        => reset,
          enable       => ones,
          parallel_in  => y(81 downto 72),
          parallel_out => x(113 downto 104));
    end generate con3;

    con4: if i=4 generate
      constant stage_x_start : positive := 104;
      constant stage_y_start : positive := 82;
      constant adder_size : positive := 5;
      constant adder_queue : positive := 1;
      begin
      adderqueue4: for j in 0 to adder_queue-1 generate
        constant blocks : positive := adder_size+1;
        begin
        adderi : adderinc generic map (
        width => adder_size)
        port map (
          adden1 => x(stage_x_start+(2*j+1)*adder_size-1 downto stage_x_start+
2*j*adder_size),
          adden2 => x(stage_x_start+2*(j+1)*adder_size-1 downto stage_x_start +
(2*j+1)*adder_size),
          sum    => y(stage_y_start+(j+1)*blocks-1 downto
stage_y_start+j*blocks));
      end generate adderqueue4;
    end generate con4;

  end generate sums;

    x(31 downto 0) <= weight_vector;
    distance <= y(87 downto 82);

end mixed;
```