5 Appendix

```
end top;

architecture Behavioral of top is

component Lab_ADC
   Port ( reset_in          : in STD_LOGIC;
          clock_in          : in STD_LOGIC;
          spi_clk_out       : out STD_LOGIC;
          chip_sel_out      : out STD_LOGIC;
          spi_data_in       : in STD_LOGIC;
          addr_in           : in STD_LOGIC_VECTOR(13 downto 0);
          data_out          : out STD_LOGIC_VECTOR(11 downto 0);
          start_in          : in STD_LOGIC;
        ready_out           : out STD_LOGIC);
end component;

component Lab_WINDOW
   Port    ( reset_in       : in STD_LOGIC;
             clock_in       : in STD_LOGIC;
             frame_addr_in  : in STD_LOGIC_VECTOR(13 downto 0);
             adc_addr_out   : out STD_LOGIC_VECTOR(13 downto 0);
             adc_data_in    : in STD_LOGIC_VECTOR(11 downto 0);
             mem_addr_in    : in STD_LOGIC_VECTOR(8 downto 0);
             mem_data_out   : out STD_LOGIC_VECTOR(19 downto 0);
             start_in       : in STD_LOGIC;
             ready_out      : out STD_LOGIC);
end component;
```

```
component Lab_FFT
   Port    ( reset_in      : in STD_LOGIC;
                             clock_in      : in STD_LOGIC;
             addr_out      : out STD_LOGIC_VECTOR(8 downto 0);
             data_in       : in STD_LOGIC_VECTOR(19 downto 0);
             addr_in       : in STD_LOGIC_VECTOR(7 downto 0);
             data_out      : out STD_LOGIC_VECTOR(31 downto 0);
                             start_in      : in STD_LOGIC;
             ready_out     : out STD_LOGIC);
end component;

component Lab_MEL
   Port    ( reset_in      : in STD_LOGIC;
                             clock_in      : in STD_LOGIC;
             fft_addr_out : out STD_LOGIC_VECTOR(7 downto 0);
             fft_data_in : in STD_LOGIC_VECTOR(31 downto 0);
             mem_addr_in : in STD_LOGIC_VECTOR(3 downto 0);
             mem_data_out : out STD_LOGIC_VECTOR(47 downto 0);
                             start_in      : in STD_LOGIC;
             ready_out     : out STD_LOGIC);
end component;

component Lab_DCT
   Port    ( reset_in      : in STD_LOGIC;
                             clock_in      : in STD_LOGIC;
             mel_addr_out : out STD_LOGIC_VECTOR(3 downto 0);
```

```
        mel_data_in : in STD_LOGIC_VECTOR(47 downto 0);
        mem_addr_in : in STD_LOGIC_VECTOR(3 downto 0);
        mem_data_out : out STD_LOGIC_VECTOR(67 downto 0);
                        start_in      : in STD_LOGIC;
        ready_out     : out STD_LOGIC);
end component;

component Lab_CMP
   Port    ( reset_in      : in STD_LOGIC;
                        clock_in      : in STD_LOGIC;
        dct_addr_out : out STD_LOGIC_VECTOR(3 downto 0);
        dct_data_in : in STD_LOGIC_VECTOR(67 downto 0);
        record_in     : in STD_LOGIC;
        number_in     : in STD_LOGIC_VECTOR(3 downto 0);
        led_out       : out STD_LOGIC_VECTOR(6 downto 0);
                        start_in      : in STD_LOGIC;
        ready_out     : out STD_LOGIC);
end component;

component Lab_CTRL
   Port    ( reset_in          : in STD_LOGIC;
        clock_in              : in STD_LOGIC;
        start_adc_out     : out STD_LOGIC;
        ready_adc_in      : in STD_LOGIC;
        frame_addr_out    : out STD_LOGIC_VECTOR(13 downto
        0); start_window_out : out STD_LOGIC;
        ready_window_in : in STD_LOGIC;
```

```
            start_fft_out     : out STD_LOGIC;
            ready_fft_in      : in STD_LOGIC;
            start_mel_out     : out STD_LOGIC;
            ready_mel_in      : in STD_LOGIC;
            start_dct_out     : out STD_LOGIC;
            ready_dct_in      : in STD_LOGIC;
            start_comp_out    : out STD_LOGIC;
            ready_comp_in     : in STD_LOGIC;
            start_debug_out : out STD_LOGIC;
            ready_debug_in    : in STD_LOGIC;
            start_in          : in STD_LOGIC;
            ready_out         : out STD_LOGIC);
end component;

signal mem_data_adc         : std_logic_vector(11 downto 0);
signal mem_addr_adc         : std_logic_vector(13 downto 0);
signal mem_data_win         : std_logic_vector(19 downto 0);
signal mem_addr_win         : std_logic_vector(8 downto 0);
signal mem_data_fft         : std_logic_vector(31 downto 0);
signal mem_addr_fft         : std_logic_vector(7 downto 0);
signal mem_data_mel         : std_logic_vector(47 downto 0);
signal mem_addr_mel         : std_logic_vector(3 downto 0);
signal mem_data_dct         : std_logic_vector(67 downto 0);
signal mem_addr_dct         : std_logic_vector(3 downto 0);

signal Frame_addr           : std_logic_vector(13 downto 0);
signal start_adc        : std_logic;
```

```
signal ready_adc        : std_logic;
signal start_win        : std_logic;
signal ready_win        : std_logic;
signal start_fft        : std_logic;
signal ready_fft        : std_logic;
signal start_mel        : std_logic;
signal ready_mel        : std_logic;
signal start_dct        : std_logic;
signal ready_dct        : std_logic;
signal start_cmp        : std_logic;
signal ready_cmp        : std_logic;

begin

        led_slc    <= "1110";

        Component1 : Lab_ADC
                PORT MAP      (           reset_in       => rst,
                              clock_in       => clk,
                              spi_clk_out    => spi_clk,
                              chip_sel_out => cs_n,
                              spi_data_in    => spi_data_in,
                              addr_in        => mem_addr_adc,
                              data_out       => mem_data_adc,
                              start_in       => start_adc,
                              ready_out      => ready_adc);
```

```
ready_adc_out      <= ready_adc;


Component2: Lab_WINDOW
   PORT MAP      (    reset_in       => rst,
                clock_in       => clk,
                frame_addr_in => Frame_addr,
                adc_addr_out => mem_addr_adc,
                adc_data_in    => mem_data_adc,
                mem_addr_in  =>  mem_addr_win,
                mem_data_out  =>  mem_data_win,
                start_in => start_win,
                ready_out => ready_win);

ready_win_out      <= ready_win;

Component3: Lab_FFT
   PORT MAP      (    reset_in      => rst,
                                clock_in      => clk,
                addr_out      => mem_addr_win,
                data_in       => mem_data_win,
                addr_in       => mem_addr_fft,
                data_out      => mem_data_fft,
                                start_in      => start_fft,
                ready_out     => ready_fft);

ready_fft_out      <= ready_fft;
```

```
Component4: Lab_MEL
   PORT MAP      ( reset_in      => rst,
                                    clock_in      => clk,
                fft_addr_out => mem_addr_fft,
                fft_data_in => mem_data_fft,
                mem_addr_in => mem_addr_mel,
                mem_data_out => mem_data_mel,
                                    start_in      => start_mel,
                ready_out     => ready_mel);

ready_mel_out      <= ready_mel;

 Component5: Lab_DCT
   PORT MAP      ( reset_in      => rst,
                                    clock_in      => clk,
                mel_addr_out => mem_addr_mel,
                mel_data_in => mem_data_mel,
                mem_addr_in => mem_addr_dct,
                mem_data_out => mem_data_dct,
                                    start_in      => start_dct,
                ready_out     => ready_dct);

ready_dct_out       <= ready_dct;

Component6: Lab_CMP
  PORT MAP      ( reset_in      => rst,
                                  clock_in      => clk,
```

```
                dct_addr_out => mem_addr_dct,
                dct_data_in    => mem_data_dct,
                record_in      => rec_in,
                number_in      => num_in,
                led_out        => led_out,
                                start_in       => start_cmp,
                ready_out      => ready_cmp);

ready_cmp_out       <= ready_cmp;


Component7: Lab_CTRL
PORT MAP          ( reset_in            => rst,
                clock_in            => clk,
                start_adc_out       => start_adc,
                ready_adc_in        => ready_adc,
                frame_addr_out      => Frame_addr,
                start_window_out    => start_win,
                ready_window_in     => ready_win,
                start_fft_out       => start_fft,
                ready_fft_in        => ready_fft,
                start_mel_out       => start_mel,
                ready_mel_in        => ready_mel,
                start_dct_out       => start_dct,
                ready_dct_in        => ready_dct,
                start_comp_out      => start_cmp,
                ready_comp_in       => ready_cmp,
```

```
                start_debug_out     => open,
                ready_debug_in      => '0',
                start_in            => start,
                ready_out           => ready_ctrl);

end Behavioral;




entity Lab_DEBUG is
    Generic ( addr_size : integer);
    Port ( reset_in                 : in STD_LOGIC;
           clock_in     : in STD_LOGIC;
                        mem_addr_out : out STD_LOGIC_VECTOR (addr_size-1 downto 0);
                        mem_data_in : in STD_LOGIC_VECTOR (31 downto 0);
                        txd_out      : out STD_LOGIC;
           start_in     : in STD_LOGIC;
           ready_out    : out STD_LOGIC);
end Lab_DEBUG;

architecture Lab_DEBUG_beh of Lab_DEBUG is

signal send                             : std_logic;
signal ready                    : std_logic;
signal data                             : std_logic_vector(7 downto 0);
signal mem_data                 : std_logic_vector(31 downto 0);
```

```
signal mem_addr                 : std_logic_vector(addr_size-1 downto 0);
signal mem_addr_lmt     : std_logic_vector(addr_size-1 downto 0);


type state_type is (initial, header1, header1w, header2, header2w, header3, header3w, header4, header4w,
read1, read2, read3, read4, addr, last,
                                                send1, send1w, send2, send2w, send3, send3w, send4, send4w,
header5, header5w, header6, header6w, header7, header7w, header8, header8w);
signal DBG_state          : state_type := initial;


component UART_TX_CTRL
   Port ( SEND     : in  STD_LOGIC;
          DATA     : in STD_LOGIC_VECTOR (7 downto 0);
          CLK      : in  STD_LOGIC;
          READY    : out STD_LOGIC;
          UART_TX : out  STD_LOGIC);
end component;


begin


   mem_addr_lmt    <= (others=>'1');


          process (clock_in) begin
                  if (rising_edge(clock_in)) then
                          if (reset_in = '1') then
                                  DBG_state             <= initial;
                                  ready_out       <= '0';
                          else
             case DBG_state is
```

```
            when initial                    => if (start_in = '1') then
                                               DBG_state   <= header1;
                                               ready_out   <= '0';
                                            else
                                               DBG_state   <= initial;
                                               ready_out   <= '1';
                                            end if;
                                                                                                    --
mem_addr      <= "11111111111111";

mem_addr      <= mem_addr_lmt;


            when header1                    => if (ready = '1') then
                                               data        <= X"55";
                                               send        <= '1';
                                               DBG_state   <= header1w;
                                            else
                                               send        <= '0';
                                               DBG_state   <= header1;
                                            end if;


            when header1w                   => send         <= '0';
                                            DBG_state      <= header2;


            when header2                    => if (ready = '1') then
```

```
                    data                <= X"AA";
                    send                <= '1';
                    DBG_state   <= header2w;
                else
                    send                    <= '0';
                    DBG_state   <= header2;
                end if;

when header2w                           => send                 <= '0';
                DBG_state       <= header3;


when header3                            => if (ready = '1') then
                    data                    <= X"CC";
                    send                    <= '1';
                    DBG_state   <= header3w;
                else
                    send                    <= '0';
                    DBG_state   <= header3;
                end if;

when header3w                           => send                 <= '0';
                DBG_state       <= header4;


when header4                            => if (ready = '1') then
                    data                    <= X"03";
                    send                    <= '1';
```

```
                    DBG_state   <= header4w;
                else
                    send        <= '0';
                    DBG_state   <= header4;
                end if;

        when header4w                   => send         <= '0';
                DBG_state       <= read1;

        when read1                      => mem_addr     <= mem_addr +
1;
DBG_state       <= read2;

        when read2                      => DBG_state    <= read3;

                when read3                      => DBG_state
        <= read4;

        when read4                      => mem_data     <=
mem_data_in;
                DBG_state       <= send1;

        when send1                      => if (ready = '1') then
                    data        <= mem_data(31 downto 24);
                    send        <= '1';
                    DBG_state   <= send1w;
                else
```

```
                send                    <= '0';
                DBG_state   <= send1;
            end if;

                    when send1w                         => send
<= '0';
            DBG_state       <= send2;

    when send2                                  => if (ready = '1') then
                data                    <= mem_data(23 downto 16);
                send                    <= '1';
                DBG_state   <= send2w;
            else
                send                    <= '0';
                DBG_state   <= send2;
            end if;

                    when send2w                         => send
<= '0';
            DBG_state       <= send3;


    when send3                                  => if (ready = '1') then
                data                    <= mem_data(15 downto 8);
                send                    <= '1';
                DBG_state   <= send3w;
            else
                send                    <= '0';
```

```
                    DBG_state    <= send3;
                end if;

                        when send3w                     => send
<= '0';
                DBG_state        <= send4;

    when send4                                          => if (ready = '1') then
                    data                <= mem_data(7 downto 0);
                    send                <= '1';
                    DBG_state    <= send4w;
                else
                    send                <= '0';
                    DBG_state    <= send4;
                end if;

                        when send4w                     => send
<= '0';
                DBG_state        <= addr;

    when addr                           => --if (mem_addr = "00111111111111") then
                 if (mem_addr = mem_addr_lmt) then
                    DBG_state    <= header5;
                else
                    DBG_state    <= read1;
```

```
                    end if;


when header5                        => if (ready = '1') then
                        data                <= X"AA";
                        send                <= '1';
                        DBG_state   <= header5w;
                    else
                        send                <= '0';
                        DBG_state   <= header5;
                    end if;

when header5w                       => send                 <= '0';
                    DBG_state       <= header6;


when header6                        => if (ready = '1') then
                        data                <= X"55";
                        send                <= '1';
                        DBG_state   <= header6w;
                    else
                        send                <= '0';
                        DBG_state   <= header6;
                    end if;

when header6w                       => send                 <= '0';
```

```
                    DBG_state       <= header7;

when header7                        => if (ready = '1') then
                      data          <= X"03";
                      send          <= '1';
                      DBG_state     <= header7w;
                    else
                      send          <= '0';
                      DBG_state     <= header7;
                    end if;

when header7w                       => send                 <= '0';
                    DBG_state       <= header8;

when header8                        => if (ready = '1') then
                      data          <= X"CC";
                      send          <= '1';
                      DBG_state     <= header8w;
                    else
                      send          <= '0';
                      DBG_state     <= header8;
                    end if;

                        when header8w                       => send
  <= '0';
```

```
                    DBG_state       <= last;

        when last               => if (ready = '1') then
                    DBG_state       <= initial;
                else
                    DBG_state       <= last;
                end if;

        when others                         => DBG_state        <= initial;

      end case;
    end if;
          end if;
     end process;

     mem_addr_out   <= mem_addr;

     Component1 : UART_TX_CTRL
          PORT MAP (  SEND          => send,
                      DATA          => data,
                      CLK           => clock_in,
                      READY         => ready,
                      UART_TX       => txd_out);

end Lab_DEBUG_beh;
```

```
entity Lab_ADC is
   Port    ( reset_in           : in STD_LOGIC;
                                clock_in     : in STD_LOGIC;
                                spi_clk_out : out STD_LOGIC;
                                chip_sel_out : out STD_LOGIC;
                                spi_data_in : in STD_LOGIC;
                                addr_in              : in STD_LOGIC_VECTOR(13 downto 0);
                                data_out             : out STD_LOGIC_VECTOR(11 downto 0);
                                start_in     : in STD_LOGIC;
          ready_out    : out STD_LOGIC);
end Lab_ADC;

architecture Lab_ADC_beh of Lab_ADC is

component Memory1_adc
   Port ( clka     : in STD_LOGIC;
          addra    : in STD_LOGIC_VECTOR(13 downto 0);
          dina     : in STD_LOGIC_VECTOR(11 downto 0);
          wea      : in STD_LOGIC_VECTOR(0 downto 0);
          clkb     : in STD_LOGIC;
          addrb    : in STD_LOGIC_VECTOR(13 downto 0);
          doutb    : out STD_LOGIC_VECTOR(11 downto 0));
end component;
```

```
signal clk_125                  : std_logic:='0';
signal clk_cnt          : std_logic_vector(2 downto 0):="000";
signal counter1                 : std_logic_vector(7 downto 0);
signal counter2            : std_logic_vector(7 downto 0);
signal counter3                 : std_logic_vector(18 downto 0);
signal data_cnt         : std_logic_vector(3 downto 0);
signal adc_rd_rdy               : std_logic:='0';
signal adc_rd_rdy_d1    : std_logic:='0';
signal ADC_data                 : std_logic_vector(11 downto 0):=(others=>'0');
signal ADC_data_sum             : std_logic_vector(16 downto 0):=(others=>'0');

signal mem_addr_a               : std_logic_vector(13 downto 0):=(others=>'0');
signal mem_data_a               : std_logic_vector(11 downto 0):=(others=>'0');
signal mem_we_a                 : std_logic_vector(0 downto
0):=(others=>'0');

type state_type1 is (initial, waitt, cs_low, cs_high,
cs_last); signal ADC_state1     : state_type1 := initial;

type state_type2 is (zero2, zero1, zero0, get_data, sum_st, write_st, waitt);
signal ADC_state2 : state_type2 := zero2;

begin

   spi_clk_out     <= clk_125;

        process (clock_in) begin
                if (rising_edge(clock_in)) then
```

```
                clk_cnt      <= clk_cnt + 1;


                -- Generate 12.5 MHz clock from 100 MHz clock
                if (clk_cnt = "000") then
                        clk_125             <= '1';
                elsif (clk_cnt = "100") then
                        clk_125             <= '0';
                end if;
        end if;
end process;


process (clock_in) begin
        if (rising_edge(clock_in)) then
                if (reset_in = '1') then
                        ADC_state1          <= initial;
                else
  case ADC_state1 is

    when initial                            => if (start_in = '1') then
                           ADC_state1   <= waitt;
                           ready_out    <= '0';
                        else
                           ADC_state1   <= initial;
                           ready_out    <= '1';
                        end if;
                        chip_sel_out    <= '1';
                        adc_rd_rdy      <= '0';
```

```
                counter1        <= (others=>'0');
                counter2        <= (others=>'0');
                counter3        <= (others=>'0');

        when waitt              => if (clk_cnt = "011") then
                ADC_state1 <= cs_low;
                adc_rd_rdy <= '1';
                chip_sel_out <= '0';    -- goes low 10 ns before falling edge of
spi_clk
                else
                ADC_state1 <= waitt;
                chip_sel_out <= '1';
                end if;

        when cs_low             => if (counter1 = "01111101") then    -- approximately 128 clk
low
                ADC_state1    <= cs_high;
                chip_sel_out <= '1';
                adc_rd_rdy    <= '0';
                counter1        <= (others=>'0');
                else
                ADC_state1    <= cs_low;
                chip_sel_out <= '0';
                counter1        <= counter1 + 1;
                end if;

        when cs_high            => if (counter2 = "01001000") then    -- approximately 72 clk
high
```

```
ADC_state1    <= cs_last;
                                    counter2          <= (others=>'0');
                                else
                                    ADC_state1    <= cs_high;
                                    counter2          <= counter2 + 1;
                                end if;
                                chip_sel_out <= '1';


                                        when cs_last                                    => if (counter3 =
"11111111111111111111") then      -- 16384 * 32
                                    ADC_state1    <= initial;
                                    counter3          <= (others=>'0');
                                    chip_sel_out <= '1';
                                else
                                    ADC_state1    <= cs_low;
                                    counter3          <= counter3 + 1;
                                    chip_sel_out <= '0';
                                    adc_rd_rdy    <= '1';
                                end if;


                when others                                     => ADC_state1             <= initial;


                end case;
            end if;
                        end if;
            end process;
```

```
process (clk_125) begin
        if (rising_edge(clk_125)) then
                if (reset_in = '1') then
                        ADC_state2          <= zero2;
                else
        case ADC_state2 is

            when zero2                                          => if (adc_rd_rdy_d1 = '0' and
adc_rd_rdy = '1') then        -- rising edge of adc_read_rdy
                                ADC_state2   <= zero1;
                            else
                                ADC_state2   <= zero2;
                            end if;

            when zero1              => ADC_state2        <= zero0;            -- here is zero1

            when zero0              => ADC_state2        <= get_data;            -- here is zero0
                            ADC_data    <= (others=>'0');
                            data_cnt    <= (others=>'0');

            when get_data             => if (data_cnt = "1011") then
                                ADC_state2   <= sum_st;
                                data_cnt    <=
                                (others=>'0');
                            else
                                ADC_state2   <= get_data;
                                data_cnt        <= data_cnt + 1;
```

```
                    end if;
                    ADC_data(0)             <= spi_data_in;
                    ADC_data(11 downto 1)   <= ADC_data(10 downto 0);

        when sum_st             => if (counter3(4 downto 0) = "00000") then
                      ADC_data_sum          <= "00000" & ADC_data;
                    else
                      ADC_data_sum          <= ADC_data_sum + ADC_data;
                    end if;

                                                                                    if
(counter3 = "0000000000000000000") then
                      mem_addr_a    <= (others=>'1');
                                                                                    end if;
                    ADC_state2 <= write_st;

        when write_st           => if (counter3(4 downto 0) = "11111") then
                      mem_data_a  <=  ADC_data_sum(16  downto  5);
                      mem_we_a <= (others=>'1');

mem_addr_a          <= mem_addr_a + 1;
                    end if;
                    ADC_state2 <= waitt;

        when waitt                  => if (adc_rd_rdy = '0') then
                      ADC_state2 <= zero2;
                    else
                      ADC_state2 <= waitt;
```

```
                                end if;
                                mem_we_a     <= (others=>'0');

          when others                                   => ADC_state2          <= zero2;

       end case;

       adc_rd_rdy_d1       <= adc_rd_rdy;
    end if;
            end if;
     end process;

            Component1 : Memory1_adc
            PORT MAP      (          clka     => clk_125,
                                     addra => mem_addr_a,
                                     dina     => mem_data_a,
                                     wea      => mem_we_a,
                                     clkb     => clock_in,
                                     addrb    => addr_in,
                                     doutb    => data_out);

end Lab_ADC_beh;




entity Lab_WINDOW is
```

```
    Port    ( reset_in      : in STD_LOGIC;
              clock_in      : in STD_LOGIC;
              frame_addr_in : in STD_LOGIC_VECTOR(13 downto 0);
              adc_addr_out : out STD_LOGIC_VECTOR(13 downto 0);
              adc_data_in : in STD_LOGIC_VECTOR(11 downto 0);
              mem_addr_in : in STD_LOGIC_VECTOR(8 downto 0);
              mem_data_out : out STD_LOGIC_VECTOR(19 downto 0);
              start_in      : in STD_LOGIC;
              ready_out     : out STD_LOGIC);
end Lab_WINDOW;

architecture Lab_WINDOW_beh of Lab_WINDOW is

component Memory1_win
    Port ( clka     : in STD_LOGIC;
           addra    : in STD_LOGIC_VECTOR(8 downto 0);
           dina     : in STD_LOGIC_VECTOR(19 downto
           0); wea          : in STD_LOGIC_VECTOR(0
           downto 0); clkb: in STD_LOGIC;
           addrb    : in STD_LOGIC_VECTOR(8 downto 0);
           doutb    : out STD_LOGIC_VECTOR(19 downto 0));
end component;

component Multiplier1_win
    Port ( CLK      : in STD_LOGIC;
           A        : in STD_LOGIC_VECTOR(11 DOWNTO 0);
           B        : in STD_LOGIC_VECTOR(7 DOWNTO 0);
```

```
        P       : out STD_LOGIC_VECTOR(19 DOWNTO 0));
end component;

signal adc_addr                 : std_logic_vector(8 downto 0);
signal coe_addr                 : std_logic_vector(8 downto 0);
signal coefficient              : std_logic_vector(7 downto 0);

signal WIN_data                 : std_logic_vector(19 downto
0):=(others=>'0'); signal mem_addr_a  : std_logic_vector(8 downto
0):=(others=>'0'); signal mem_we_a      : std_logic_vector(0 downto
0):=(others=>'0');

type state_type1 is (initial, read_adc1, read_adc2, wr_win);
signal WIN_state1     : state_type1 := initial;

begin

  adc_addr_out          <= ("00000" & adc_addr) + frame_addr_in;

        process (clock_in) begin
                if (rising_edge(clock_in)) then
                        if (reset_in = '1') then
                                WIN_state1          <= initial;
                        else
            case WIN_state1 is

              when initial                      => if (start_in = '1') then
                                    WIN_state1 <= read_adc1;
```

```
adc_addr                    <= "000000001";
                                ready_out    <= '0';
                            else
                                WIN_state1 <= initial;

adc_addr                    <= "000000000";
                                ready_out    <= '1';
                            end if;

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

            when read_adc1          => adc_addr                     <= "000000010";

WIN_state1      <= read_adc2;

            when read_adc2          => adc_addr                     <= "000000011";
                            mem_we_a        <= (others=>'1');

WIN_state1      <= wr_win;


                                    when wr_win                             => if (adc_addr
= "000000001") then

        WIN_state1      <= initial;
                                                                                    else

        WIN_state1      <= wr_win;
                                                                                    end if;
```

```
adc_addr                    <= adc_addr + 1;

            when others                                     => WIN_state1           <= initial;

          end case;
        end if;
                end if;
        end process;

        mem_addr_a    <= adc_addr - "000000011";
        Component1 : Memory1_win
        PORT MAP    (         clka    => clock_in,
                              addra => mem_addr_a,
                              dina    => WIN_data,
                              wea     => mem_we_a,
                              clkb    => clock_in,
                              addrb   => mem_addr_in,
                              doutb   => mem_data_out);

    Component2: Multiplier1_win
    PORT MAP    ( CLK     => clock_in,
                  A       => adc_data_in,
                  B       => coefficient,
                  P       => WIN_data);

        coe_addr          <= adc_addr - "000000010";
```

```
with coe_addr select

        coefficient         <= X"00" when "000000000",

            X"00" when "000000001",

            X"00" when "000000010",

            X"00" when "000000011",

            X"00" when "000000100",

            X"00" when "000000101",

            X"00" when "000000110",

            X"00" when "000000111",

            X"00" when "000001000",

            X"00" when "000001001",

            X"00" when "000001010",

            X"01" when "000001011",

            X"01" when "000001100",

            X"01" when "000001101",

            X"01" when "000001110",

            X"01" when "000001111",

            X"01" when "000010000",

            X"01" when "000010001",

            X"02" when "000010010",

            X"02" when "000010011",

            X"02" when "000010100",

            X"02" when "000010101",

            X"02" when "000010110",

            X"03" when "000010111",

            X"03" when "000011000",
```

X"03" when "000011001",

X"03" when "000011010",

X"03" when "000011011",

X"04" when "000011100",

X"04" when "000011101",

X"04" when "000011110",

X"05" when "000011111",

X"05" when "000100000",

X"05" when "000100001",

X"06" when "000100010",

X"06" when "000100011",

X"06" when "000100100",

X"07" when "000100101",

X"07" when "000100110",

X"07" when "000100111",

X"08" when "000101000",

X"08" when "000101001",

X"08" when "000101010",

X"09" when "000101011",

X"09" when "000101100",

X"0A" when "000101101",

X"0A" when "000101110",

X"0A" when "000101111",

X"0B" when "000110000",

X"0B" when "000110001",

X"0C" when "000110010",

X"0C" when "000110011",

X"0D" when "000110100",

X"0D" when "000110101",

X"0E" when "000110110",

X"0E" when "000110111",

X"0F" when "000111000",

X"0F" when "000111001",

X"10" when "000111010",

X"10" when "000111011",

X"11" when "000111100",

X"11" when "000111101",

X"12" when "000111110",

X"12" when "000111111",

X"13" when "001000000",

X"13" when "001000001",

X"14" when "001000010",

X"15" when "001000011",

X"15" when "001000100",

X"16" when "001000101",

X"16" when "001000110",

X"17" when "001000111",

X"17" when "001001000",

X"18" when "001001001",

X"19" when "001001010",

X"19" when "001001011",

X"1A" when "001001100",

X"1B" when "001001101",

X"1B" when "001001110",

X"1C" when "001001111",

X"1D" when "001010000",

X"1D" when "001010001",

X"1E" when "001010010",

X"1F" when "001010011",

X"1F" when "001010100",

X"20" when "001010101",

X"21" when "001010110",

X"21" when "001010111",

X"22" when "001011000",

X"23" when "001011001",

X"23" when "001011010",

X"24" when "001011011",

X"25" when "001011100",

X"25" when "001011101",

X"26" when "001011110",

X"27" when "001011111",

X"28" when "001100000",

X"28" when "001100001",

X"29" when "001100010",

X"2A" when "001100011",

X"2B" when "001100100",

X"2B" when "001100101",

X"2C" when "001100110",

X"2D" when "001100111",

X"2E" when "001101000",

X"2E" when "001101001",

```
X"2F" when "001101010",
X"30" when "001101011",
X"31" when "001101100",
X"31" when "001101101",
X"32" when "001101110",
X"33" when "001101111",
X"34" when "001110000",
X"34" when "001110001",
X"35" when "001110010",
X"36" when "001110011",
X"37" when "001110100",
X"38" when "001110101",
X"38" when "001110110",
X"39" when "001110111",
X"3A" when "001111000",
X"3B" when "001111001",
X"3B" when "001111010",
X"3C" when "001111011",
X"3D" when "001111100",
X"3E" when "001111101",
X"3F" when "001111110",
X"3F" when "001111111",
X"40" when "010000000",
X"41" when "010000001",
X"42" when "010000010",
X"43" when "010000011",
X"43" when "010000100",
```

X"44" when "010000101",

X"45" when "010000110",

X"46" when "010000111",

X"46" when "010001000",

X"47" when "010001001",

X"48" when "010001010",

X"49" when "010001011",

X"4A" when "010001100",

X"4A" when "010001101",

X"4B" when "010001110",

X"4C" when "010001111",

X"4D" when "010010000",

X"4D" when "010010001",

X"4E" when "010010010",

X"4F" when "010010011",

X"50" when "010010100",

X"51" when "010010101",

X"51" when "010010110",

X"52" when "010010111",

X"53" when "010011000",

X"54" when "010011001",

X"54" when "010011010",

X"55" when "010011011",

X"56" when "010011100",

X"57" when "010011101",

X"57" when "010011110",

X"58" when "010011111",

X"59" when "010100000",

X"59" when "010100001",

X"5A" when "010100010",

X"5B" when "010100011",

X"5C" when "010100100",

X"5C" when "010100101",

X"5D" when "010100110",

X"5E" when "010100111",

X"5E" when "010101000",

X"5F" when "010101001",

X"60" when "010101010",

X"60" when "010101011",

X"61" when "010101100",

X"62" when "010101101",

X"62" when "010101110",

X"63" when "010101111",

X"64" when "010110000",

X"64" when "010110001",

X"65" when "010110010",

X"66" when "010110011",

X"66" when "010110100",

X"67" when "010110101",

X"68" when "010110110",

X"68" when "010110111",

X"69" when "010111000",

X"69" when "010111001",

X"6A" when "010111010",

X"6B" when "010111011",

X"6B" when "010111100",

X"6C" when "010111101",

X"6C" when "010111110",

X"6D" when "010111111",

X"6D" when "011000000",

X"6E" when "011000001",

X"6F" when "011000010",

X"6F" when "011000011",

X"70" when "011000100",

X"70" when "011000101",

X"71" when "011000110",

X"71" when "011000111",

X"72" when "011001000",

X"72" when "011001001",

X"73" when "011001010",

X"73" when "011001011",

X"74" when "011001100",

X"74" when "011001101",

X"75" when "011001110",

X"75" when "011001111",

X"75" when "011010000",

X"76" when "011010001",

X"76" when "011010010",

X"77" when "011010011",

X"77" when "011010100",

X"77" when "011010101",

X"78" when "011010110",

X"78" when "011010111",

X"79" when "011011000",

X"79" when "011011001",

X"79" when "011011010",

X"7A" when "011011011",

X"7A" when "011011100",

X"7A" when "011011101",

X"7B" when "011011110",

X"7B" when "011011111",

X"7B" when "011100000",

X"7C" when "011100001",

X"7C" when "011100010",

X"7C" when "011100011",

X"7C" when "011100100",

X"7D" when "011100101",

X"7D" when "011100110",

X"7D" when "011100111",

X"7D" when "011101000",

X"7E" when "011101001",

X"7E" when "011101010",

X"7E" when "011101011",

X"7E" when "011101100",

X"7E" when "011101101",

X"7F" when "011101110",

X"7F" when "011101111",

X"7F" when "011110000",

X"7F" when "011110001",

X"7F" when "011110010",

X"7F" when "011110011",

X"7F" when "011110100",

X"7F" when "011110101",

X"80" when "011110110",

X"80" when "011110111",

X"80" when "011111000",

X"80" when "011111001",

X"80" when "011111010",

X"80" when "011111011",

X"80" when "011111100",

X"80" when "011111101",

X"80" when "011111110",

X"80" when "011111111",

X"80" when "100000000",

X"80" when "100000001",

X"80" when "100000010",

X"80" when "100000011",

X"80" when "100000100",

X"80" when "100000101",

X"80" when "100000110",

X"80" when "100000111",

X"80" when "100001000",

X"80" when "100001001",

X"7F" when "100001010",

X"7F" when "100001011",

X"7F" when "100001100",

X"7F" when "100001101",

X"7F" when "100001110",

X"7F" when "100001111",

X"7F" when "100010000",

X"7F" when "100010001",

X"7E" when "100010010",

X"7E" when "100010011",

X"7E" when "100010100",

X"7E" when "100010101",

X"7E" when "100010110",

X"7D" when "100010111",

X"7D" when "100011000",

X"7D" when "100011001",

X"7D" when "100011010",

X"7C" when "100011011",

X"7C" when "100011100",

X"7C" when "100011101",

X"7C" when "100011110",

X"7B" when "100011111",

X"7B" when "100100000",

X"7B" when "100100001",

X"7A" when "100100010",

X"7A" when "100100011",

X"7A" when "100100100",

X"79" when "100100101",

X"79" when "100100110",

X"79" when "100100111",

X"78" when "100101000",

X"78" when "100101001",

X"77" when "100101010",

X"77" when "100101011",

X"77" when "100101100",

X"76" when "100101101",

X"76" when "100101110",

X"75" when "100101111",

X"75" when "100110000",

X"75" when "100110001",

X"74" when "100110010",

X"74" when "100110011",

X"73" when "100110100",

X"73" when "100110101",

X"72" when "100110110",

X"72" when "100110111",

X"71" when "100111000",

X"71" when "100111001",

X"70" when "100111010",

X"70" when "100111011",

X"6F" when "100111100",

X"6F" when "100111101",

X"6E" when "100111110",

X"6D" when "100111111",

X"6D" when "101000000",

X"6C" when "101000001",

X"6C" when "101000010",

X"6B" when "101000011",

X"6B" when "101000100",

X"6A" when "101000101",

X"69" when "101000110",

X"69" when "101000111",

X"68" when "101001000",

X"68" when "101001001",

X"67" when "101001010",

X"66" when "101001011",

X"66" when "101001100",

X"65" when "101001101",

X"64" when "101001110",

X"64" when "101001111",

X"63" when "101010000",

X"62" when "101010001",

X"62" when "101010010",

X"61" when "101010011",

X"60" when "101010100",

X"60" when "101010101",

X"5F" when "101010110",

X"5E" when "101010111",

X"5E" when "101011000",

X"5D" when "101011001",

X"5C" when "101011010",

X"5C" when "101011011",

X"5B" when "101011100",

X"5A" when "101011101",

X"59" when "101011110",

X"59" when "101011111",

X"58" when "101100000",

X"57" when "101100001",

X"57" when "101100010",

X"56" when "101100011",

X"55" when "101100100",

X"54" when "101100101",

X"54" when "101100110",

X"53" when "101100111",

X"52" when "101101000",

X"51" when "101101001",

X"51" when "101101010",

X"50" when "101101011",

X"4F" when "101101100",

X"4E" when "101101101",

X"4D" when "101101110",

X"4D" when "101101111",

X"4C" when "101110000",

X"4B" when "101110001",

X"4A" when "101110010",

X"4A" when "101110011",

X"49" when "101110100",

X"48" when "101110101",

X"47" when "101110110",

X"46" when "101110111",

X"46" when "101111000",

X"45" when "101111001",

X"44" when "101111010",

X"43" when "101111011",

X"43" when "101111100",

X"42" when "101111101",

X"41" when "101111110",

X"40" when "101111111",

X"3F" when "110000000",

X"3F" when "110000001",

X"3E" when "110000010",

X"3D" when "110000011",

X"3C" when "110000100",

X"3B" when "110000101",

X"3B" when "110000110",

X"3A" when "110000111",

X"39" when "110001000",

X"38" when "110001001",

X"38" when "110001010",

X"37" when "110001011",

X"36" when "110001100",

X"35" when "110001101",

X"34" when "110001110",

X"34" when "110001111",

X"33" when "110010000",

X"32" when "110010001",

X"31" when "110010010",

X"31" when "110010011",

X"30" when "110010100",

X"2F" when "110010101",

X"2E" when "110010110",

X"2E" when "110010111",

X"2D" when "110011000",

X"2C" when "110011001",

X"2B" when "110011010",

X"2B" when "110011011",

X"2A" when "110011100",

X"29" when "110011101",

X"28" when "110011110",

X"28" when "110011111",

X"27" when "110100000",

X"26" when "110100001",

X"25" when "110100010",

X"25" when "110100011",

X"24" when "110100100",

X"23" when "110100101",

X"23" when "110100110",

X"22" when "110100111",

X"21" when "110101000",

X"21" when "110101001",

X"20" when "110101010",

X"1F" when "110101011",

X"1F" when "110101100",

X"1E" when "110101101",

X"1D" when "110101110",

X"1D" when "110101111",

X"1C" when "110110000",

X"1B" when "110110001",

X"1B" when "110110010",

X"1A" when "110110011",

X"19" when "110110100",

X"19" when "110110101",

X"18" when "110110110",

X"17" when "110110111",

X"17" when "110111000",

X"16" when "110111001",

X"16" when "110111010",

X"15" when "110111011",

X"15" when "110111100",

X"14" when "110111101",

X"13" when "110111110",

X"13" when "110111111",

X"12" when "111000000",

X"12" when "111000001",

X"11" when "111000010",

X"11" when "111000011",

X"10" when "111000100",

X"10" when "111000101",

X"0F" when "111000110",

X"0F" when "111000111",

X"0E" when "111001000",

X"0E" when "111001001",

X"0D" when "111001010",

X"0D" when "111001011",

X"0C" when "111001100",

X"0C" when "111001101",

X"0B" when "111001110",

X"0B" when "111001111",

X"0A" when "111010000",

X"0A" when "111010001",

X"0A" when "111010010",

X"09" when "111010011",

X"09" when "111010100",

X"08" when "111010101",

X"08" when "111010110",

X"08" when "111010111",

X"07" when "111011000",

X"07" when "111011001",

X"07" when "111011010",

X"06" when "111011011",

X"06" when "111011100",

X"06" when "111011101",

X"05" when "111011110",

X"05" when "111011111",

X"05" when "111100000",

X"04" when "111100001",

X"04" when "111100010",

X"04" when "111100011",

X"03" when "111100100",

X"03" when "111100101",

X"03" when "111100110",

X"03" when "111100111",

X"03" when "111101000",

X"02" when "111101001",

X"02" when "111101010",

X"02" when "111101011",

X"02" when "111101100",

X"02" when "111101101",

X"01" when "111101110",

X"01" when "111101111",

X"01" when "111110000",

X"01" when "111110001",

X"01" when "111110010",

X"01" when "111110011",

X"01" when "111110100",

X"00" when "111110101",

X"00" when "111110110",

X"00" when "111110111",

X"00" when "111111000",

X"00" when "111111001",

X"00" when "111111010",

X"00" when "111111011",

X"00" when "111111100",

X"00" when "111111101",

X"00" when "111111110",

```
                    X"00" when "111111111",
                    X"00" when others;

end Lab_WINDOW_beh;



entity Lab_CTRL is
   Port    ( reset_in          : in STD_LOGIC;
             clock_in          : in STD_LOGIC;
             start_adc_out     : out STD_LOGIC;
             ready_adc_in      : in STD_LOGIC;
             frame_addr_out    : out STD_LOGIC_VECTOR(13 downto 0);
             start_window_out : out STD_LOGIC;
             ready_window_in : in STD_LOGIC;
             start_fft_out     : out STD_LOGIC;
             ready_fft_in      : in STD_LOGIC;
             start_mel_out     : out STD_LOGIC;
             ready_mel_in      : in STD_LOGIC;
             start_dct_out     : out STD_LOGIC;
             ready_dct_in      : in STD_LOGIC;
             start_comp_out    : out STD_LOGIC;
             ready_comp_in     : in STD_LOGIC;
             start_debug_out : out STD_LOGIC;
             ready_debug_in    : in STD_LOGIC;
             start_in          : in STD_LOGIC;
             ready_out         : out STD_LOGIC);
```

```
end Lab_CTRL;

architecture Lab_CTRL_beh of Lab_CTRL is

signal frame_addr                : std_logic_vector(13 downto 0);
signal frame_addr_inc : std_logic_vector(13 downto 0);

type state_type1 is (initial, start_adc, wait_adc1, wait_adc2, start_window, wait_window1,
wait_window2, start_fft, wait_fft1, wait_fft2, start_mel,
        wait_mel1, wait_mel2, start_dct, wait_dct1, wait_dct2, start_comp, wait_comp1,
wait_comp2, start_debug, wait_debug1, wait_debug2);
signal CTRL_state1     : state_type1 := initial;

begin

   frame_addr_inc      <= "00000100000000";        -- shift window by 256
   frame_addr_out      <= frame_addr;

        process (clock_in) begin
            if (rising_edge(clock_in)) then
                if (reset_in = '1') then
                    CTRL_state1         <= initial;
                else
          case CTRL_state1 is

            when initial                        => if (start_in = '1') then
                                CTRL_state1 <= start_adc;
                                ready_out    <= '0';
```

```
                                    else
                                       CTRL_state1 <= initial;
                                       ready_out    <= '1';
                                    end if;
                                    start_adc_out    <= '0';
                                    start_window_out <= '0';
                                    start_fft_out    <= '0';
                                    start_mel_out    <= '0';
                                    start_dct_out    <= '0';
                                    start_comp_out   <= '0';
                                    start_debug_out  <= '0';
                                    frame_addr       <= "00000000000000" - frame_addr_inc;


            when start_adc             => start_adc_out    <= '1';

CTRL_state1      <= wait_adc1;

                                          when wait_adc1             => start_adc_out    <= '0';
                                                                        CTRL_state1      <= wait_adc2;

            when wait_adc2             => if (ready_adc_in = '1') then
                                       CTRL_state1      <= start_window;
                                                                                                  else

CTRL_state1      <= wait_adc2;
                                                                                                  end if;
```

```
                    when start_window                    => start_window_out <=
'1';
                                                         frame_addr      <= frame_addr +
frame_addr_inc;
                                                         CTRL_state1     <= wait_window1;

            when wait_window1        => start_window_out   <= '0';
                                 CTRL_state1        <= wait_window2;

            when wait_window2        => if (ready_window_in = '1') then
                                    CTRL_state1     <= start_fft;
                                                                                    else
CTRL_state1     <= wait_window2;
                                                                                    end if;

            when start_fft           => start_fft_out   <= '1';
CTRL_state1     <= wait_fft1;

            when wait_fft1           => start_fft_out   <= '0';
                                 CTRL_state1     <= wait_fft2;

            when wait_fft2           => if (ready_fft_in = '1') then
                                    CTRL_state1     <= start_mel;
                                                                                    else
```

```
CTRL_state1     <= wait_fft2;
                                                                    end if;

        when start_mel          => start_mel_out    <= '1';

CTRL_state1     <= wait_mel1;

        when wait_mel1          => start_mel_out    <= '0';
                                CTRL_state1     <= wait_mel2;

        when wait_mel2          => if (ready_mel_in = '1') then
                                   CTRL_state1     <= start_dct;
                                                                    else

CTRL_state1     <= wait_mel2;
                                                                    end if;

        when start_dct          => start_dct_out    <= '1';

CTRL_state1     <= wait_dct1;

        when wait_dct1          => start_dct_out    <= '0';
                                CTRL_state1     <= wait_dct2;

        when wait_dct2          => if (ready_dct_in = '1') then
                                   CTRL_state1     <= start_comp;
                                                                    else
```

```
CTRL_state1     <= wait_dct2;
                                                                                end if;

            when start_comp          => start_comp_out <= '1';

CTRL_state1     <= wait_comp1;

            when wait_comp1          => start_comp_out <= '0';
                                    CTRL_state1     <= wait_comp2;

            when wait_comp2          => if (ready_comp_in = '1') then
                                    if (frame_addr = "11111000000000") then     -- 2^14 - 512
                                       CTRL_state1     <= initial;
                                    else
                                       CTRL_state1     <= start_window;
                                    end if;
                                                                                else

CTRL_state1     <= wait_comp2;
                                                                                end if;

--              when start_debug         => start_debug_out <= '1';
--
CTRL_state1     <= wait_debug1;
--
--              when wait_debug1         => start_debug_out  <= '0';
--                                  CTRL_state1     <= wait_debug2;
```

```
--
--                when wait_debug2          => if (ready_debug_in = '1') then
--                                            if (frame_addr = "11111000000000") then      -- 2^14 - 512
--                                               CTRL_state1      <= initial;
--                                            else
--                                               CTRL_state1      <= start_window;
--                                            end if;
--                                                                                                                  else
--
CTRL_state1      <= wait_debug2;
--                                                                                                                  end if;



                when others                                    => CTRL_state1                  <= initial;


            end case;
          end if;
                    end if;
          end process;


end Lab_CTRL_beh;




entity Lab_FFT is
   Port    ( reset_in      : in STD_LOGIC;
                             clock_in      : in STD_LOGIC;
```

```
            addr_out      : out STD_LOGIC_VECTOR(8 downto 0);
            data_in       : in STD_LOGIC_VECTOR(19 downto 0);
            addr_in       : in STD_LOGIC_VECTOR(7 downto 0);
            data_out      : out STD_LOGIC_VECTOR(31 downto 0);
                                start_in      : in STD_LOGIC;
            ready_out     : out STD_LOGIC);
end Lab_FFT;

architecture Lab_FFT_beh of Lab_FFT is

component FFT1
   Port ( aclk                      : IN STD_LOGIC;
          aresetn                   : IN STD_LOGIC;
          s_axis_config_tdata       : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          s_axis_config_tvalid      : IN STD_LOGIC;
          s_axis_config_tready      : OUT STD_LOGIC;
          s_axis_data_tdata         : IN STD_LOGIC_VECTOR(47 DOWNTO 0);
          s_axis_data_tvalid        : IN STD_LOGIC;
          s_axis_data_tready        : OUT STD_LOGIC;
          s_axis_data_tlast         : IN STD_LOGIC;
          m_axis_data_tdata         : OUT STD_LOGIC_VECTOR(63 DOWNTO 0);
          m_axis_data_tvalid        : OUT STD_LOGIC;
          m_axis_data_tlast         : OUT STD_LOGIC;
          event_frame_started       : OUT STD_LOGIC;
          event_tlast_unexpected    : OUT STD_LOGIC;
          event_tlast_missing       : OUT STD_LOGIC;
          event_data_in_channel_halt : OUT STD_LOGIC);
```

```
end component;

component Multiplier1_fft
   Port ( CLK      : in STD_LOGIC;
          A        : in STD_LOGIC_VECTOR(31 DOWNTO 0);
          B        : in STD_LOGIC_VECTOR(31 DOWNTO 0); P
                   : out STD_LOGIC_VECTOR(63 DOWNTO 0));
end component;

component Memory1_fft
   Port ( clka     : in STD_LOGIC;
          addra    : in STD_LOGIC_VECTOR(7 downto 0);
          dina     : in STD_LOGIC_VECTOR(31 downto
          0); wea             : in STD_LOGIC_VECTOR(0
          downto 0); clkb: in STD_LOGIC;
          addrb    : in STD_LOGIC_VECTOR(7 downto 0);
          doutb    : out STD_LOGIC_VECTOR(31 downto 0));
end component;

-- FFT signals
signal fft_conf_data          : std_logic_vector(7 downto
0); signal fft_conf_valid            : std_logic:='0';
signal fft_conf_ready               : std_logic;
signal fft_data_in                  : std_logic_vector(47 downto 0);
signal fft_valid_in                 : std_logic;
signal fft_ready_out          : std_logic;
signal fft_last_in                  : std_logic;
```

```
signal fft_data_out                    : std_logic_vector(63 downto 0);
signal fft_valid_out          : std_logic;
signal fft_last_out                    : std_logic;
signal fft_frame_started : std_logic;
signal fft_last_unexpected          : std_logic;
signal fft_last_missing : std_logic;
signal fft_halt          : std_logic;

-- Lab_WIN signals
signal win_addr                        : std_logic_vector(8 downto 0);
signal win_data                        : std_logic_vector(19 downto 0);
type memory_win is array (0 to 511) of std_logic_vector(19 downto
0); signal win_data_arr : memory_win :=(others=>X"00000");
signal win_arr_ind1                            : integer range 0 to 511 := 0;
signal win_arr_ind2                            : integer range 0 to 511 := 0;

-- FFT memory signals
signal mem_addr_a                      : std_logic_vector(7 downto 0):=(others=>'0');
signal mem_data_a                      : std_logic_vector(31 downto 0):=(others=>'0');
signal mem_we_a                           : std_logic_vector(0 downto
0):=(others=>'0');

-- Multiplier signals
signal mult1_inp              : std_logic_vector(31 downto 0);
signal mult2_inp                       : std_logic_vector(31 downto 0);
signal mult1_outp                      : std_logic_vector(63 downto 0);
signal mult2_outp                      : std_logic_vector(63 downto 0);
signal result_sum                      : std_logic_vector(63 downto 0);
```

```
type state_type1 is (initial, read_win1, read_win2, read_win3, read_win4, write_conf1, write_conf2,
write_fft1, write_fft2, write_fft3, read_fft1, read_fft2, read_fft3);

signal FFT_state1     : state_type1 := initial;


begin


   addr_out        <= win_addr;

   win_data        <= data_in;

         process (clock_in) begin

                  if (rising_edge(clock_in)) then

                           if (reset_in = '1') then

                                    FFT_state1                    <= initial;

                           else

            case FFT_state1 is


               when initial                              => if (start_in = '1') then

         FFT_state1       <= read_win1;

         ready_out      <= '0';

                                    else

         FFT_state1       <= initial;

         ready_out      <= '1';

                                    end if;

fft_conf_valid              <= '0';
```

```
                                fft_valid_in                    <= '0';
                                fft_last_in             <= '0';

                -- Read window data and write it into an internal array
                when read_win1                  => win_addr             <= (others=>'0');
win_arr_ind1                    <= 0;
FFT_state1              <= read_win2;

                                                when read_win2              => win_addr                 <= win_addr +
1;
FFT_state1          <= read_win3;

                                                when read_win3              => win_addr             <= win_addr + 1;
FFT_state1          <= read_win4;

                when read_win4              => if (win_addr = "000000001") then
FFT_state1          <= write_conf1;
                                        else
                                            FFT_state1 <= read_win4;
            win_addr            <= win_addr + 1;
                                        end if;
win_data_arr(win_arr_ind1)          <= win_data;
win_arr_ind1                                    <= win_arr_ind1 + 1;
```

```
                                        -- Write configuration data to the FFT IP Core
            when write_conf1        => fft_conf_data            <= "00101001";
fft_conf_valid              <= '1';
FFT_state1      <= write_conf2;

                                        when write_conf2            => if (fft_conf_ready = '1') then
fft_conf_valid   <= '0';
FFT_state1          <= write_fft1;
                                                                                                    else
        FFT_state1      <= write_conf2;
                                                                                                    end if;

                                        -- Run FFT IP Core with the data inside the array
            when write_fft1         => if (fft_ready_out = '1') then
win_arr_ind2                <= 0;
fft_valid_in      <= '1';
                                    FFT_state1      <= write_fft2;
                                                                                                    else
                                    FFT_state1      <= write_fft1;
                                                                                                    end if;
```

```
                when write_fft2          => if (fft_ready_out = '1') then
                                                                                                    if
(win_arr_ind2 = 510) then
fft_last_in    <= '1';
            FFT_state1     <= write_fft3;
else
            fft_last_in    <= '0';

            FFT_state1     <= write_fft2;
end if;
win_arr_ind2         <= win_arr_ind2 + 1;
                                                                                                    else
fft_last_in       <= '0';
FFT_state1        <= write_fft2;
                                                                                                    end if;

                                        when write_fft3          => fft_valid_in        <= '0';
fft_last_in         <= '0';
win_arr_ind2                        <= 0;
FFT_state1                    <= read_fft1;
```

```
                                        -- Calculate the FFT squared sum and write into the memory
            when read_fft1          => if (fft_valid_out = '1') then

mem_we_a            <= (others=>'1');

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

FFT_state1    <= read_fft2;
                                                                                                    else

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

mem_addr_a    <= (others=>'0');
                                FFT_state1    <= read_fft1;
                                                                                                    end if;


            when read_fft2        => if (fft_valid_out = '1') then
                                    if (mem_addr_a = "11111110") then
                                        mem_we_a          <= (others=>'1');
                                        mem_addr_a        <= mem_addr_a + 1;
                                        FFT_state1    <= read_fft3;
                                    else

        mem_we_a            <= (others=>'1');

        mem_addr_a    <= mem_addr_a + 1;

        FFT_state1    <= read_fft2;

end if;
```

```
            else

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

mem_addr_a       <= mem_addr_a;

FFT_state1     <= read_fft2;

            end if;


                                                when read_fft3              => mem_we_a               <=
(others=>'0');

            FFT_state1                  <= initial;


                when others                                          => FFT_state1 <= initial;


              end case;
            end if;
                        end if;
             end process;


            fft_data_in                          <= X"0000000" & win_data_arr(win_arr_ind2);
    Component1: FFT1
    PORT MAP      ( aclk                    => clock_in,
                  aresetn                 => '1',
                  s_axis_config_tdata       => fft_conf_data,
                  s_axis_config_tvalid      => fft_conf_valid,
                  s_axis_config_tready      => fft_conf_ready,
```

```
            s_axis_data_tdata           => fft_data_in,
            s_axis_data_tvalid          => fft_valid_in,
            s_axis_data_tready          => fft_ready_out,
            s_axis_data_tlast           => fft_last_in,
            m_axis_data_tdata           => fft_data_out,
            m_axis_data_tvalid          => fft_valid_out,
            m_axis_data_tlast           => fft_last_out,
            event_frame_started         => fft_frame_started,
            event_tlast_unexpected      => fft_last_unexpected,
            event_tlast_missing         => fft_last_missing,
            event_data_in_channel_halt => fft_halt);

mult1_inp <= fft_data_out(31 downto 0);
Component2: Multiplier1_fft
PORT MAP    ( CLK      => clock_in,
          A       => mult1_inp,
          B       => mult1_inp,
          P       => mult1_outp);

mult2_inp <= fft_data_out(63 downto 32);
Component3: Multiplier1_fft
PORT MAP    ( CLK      => clock_in,
          A       => mult2_inp,
          B       => mult2_inp,
          P       => mult2_outp);

      result_sum    <= mult1_outp + mult2_outp;
```

```
        mem_data_a      <= result_sum(63 downto 32);

        Component4: Memory1_fft
        PORT MAP    (       clka    => clock_in,
                            addra => mem_addr_a,
                            dina    => mem_data_a,
                            wea     => mem_we_a,
                            clkb    => clock_in,
                            addrb   => addr_in,
                            doutb   => data_out);

end Lab_FFT_beh;



entity Lab_MEL is
   Port   ( reset_in     : in STD_LOGIC;
                         clock_in     : in STD_LOGIC;
        fft_addr_out : out STD_LOGIC_VECTOR(7 downto 0);
        fft_data_in : in STD_LOGIC_VECTOR(31 downto 0);
        mem_addr_in : in STD_LOGIC_VECTOR(3 downto 0);
        mem_data_out : out STD_LOGIC_VECTOR(47 downto 0);
                         start_in     : in STD_LOGIC;
        ready_out    : out STD_LOGIC);
end Lab_MEL;

architecture Lab_MEL_beh of Lab_MEL is
```

```
component Multiplier1_mel
   Port ( CLK      : in STD_LOGIC;
          A        : in STD_LOGIC_VECTOR(31 DOWNTO 0);
          B        : in STD_LOGIC_VECTOR(7 DOWNTO 0);
          P        : out STD_LOGIC_VECTOR(39 DOWNTO 0));
end component;

component Memory1_mel
   Port ( clka     : in STD_LOGIC;
          addra    : in STD_LOGIC_VECTOR(3 downto 0);
          dina     : in STD_LOGIC_VECTOR(47 downto
          0); wea             : in STD_LOGIC_VECTOR(0
          downto 0); clkb: in STD_LOGIC;
          addrb    : in STD_LOGIC_VECTOR(3 downto 0);
          doutb    : out STD_LOGIC_VECTOR(47 downto 0));
end component;

signal fft_addr                 : std_logic_vector(7 downto 0);
signal fft_data                 : std_logic_vector(31 downto 0);

-- MEL memory signals
signal mem_addr_a                 : std_logic_vector(3 downto 0):=(others=>'0');
signal mem_data_a                 : std_logic_vector(47 downto 0):=(others=>'0');
signal mem_we_a                     : std_logic_vector(0 downto
0):=(others=>'0');
```

```
type state_type1 is (initial, build_filt, mult_wt1, mult_wt2, mult_st1, mult_st2, mult_st3, mult_st4,
mult_st5, mult_st6, mult_st7, mult_st8, mult_st9, mult_st10, mult_st11, mult_st12,

                                        mult_st13, mult_st14, mult_st15, write_st1, write_st2, write_st3,
write_st4, write_st5, write_st6, write_st7, write_st8, write_st9, write_st10, write_st11, write_st12,
write_st13);

signal MEL_state1     : state_type1 := initial;


type mel_coeff is array (0 to 255) of std_logic_vector(7 downto 0);

signal mel_filt1 : mel_coeff;

signal mel_filt2 : mel_coeff;

signal mel_filt3 : mel_coeff;

signal mel_filt4 : mel_coeff;

signal mel_filt5 : mel_coeff;

signal mel_filt6 : mel_coeff;

signal mel_filt7 : mel_coeff;

signal mel_filt8 : mel_coeff;

signal mel_filt9 : mel_coeff;

signal mel_filt10 : mel_coeff;

signal mel_filt11 : mel_coeff;

signal mel_filt12 : mel_coeff;

signal mel_filt13 : mel_coeff;

signal ind                    : integer range 0 to 255 := 0;


constant filter1_start : integer := 2;          constant filter1_top : integer := 6;          constant filter1_stop :
integer := 12;

constant filter2_start : integer := 6;          constant filter2_top : integer := 12;          constant filter2_stop
: integer := 19;

constant filter3_start : integer := 12;          constant filter3_top : integer := 19;          constant filter3_stop
: integer := 27;
```

```
constant filter4_start : integer := 19;          constant filter4_top : integer := 27;          constant filter4_stop
: integer := 36;

constant filter5_start : integer := 27;          constant filter5_top : integer := 36;          constant filter5_stop
: integer := 47;

constant filter6_start : integer := 36;          constant filter6_top : integer := 47;          constant filter6_stop
: integer := 61;

constant filter7_start : integer := 47;          constant filter7_top : integer := 61;          constant filter7_stop
: integer := 76;

constant filter8_start : integer := 61;          constant filter8_top : integer := 76;          constant filter8_stop
: integer :=  95;

constant filter9_start : integer := 76;          constant filter9_top : integer := 95;          constant filter9_stop
: integer :=  117;

constant filter10_start : integer := 95;          constant filter10_top : integer := 117;          constant
filter10_stop : integer := 143;

constant filter11_start : integer := 117;          constant filter11_top : integer := 143;          constant
filter11_stop : integer := 174;

constant filter12_start : integer := 143;          constant filter12_top : integer := 174;          constant
filter12_stop : integer := 211;

constant filter13_start : integer := 174;          constant filter13_top : integer := 211;          constant
filter13_stop : integer := 255;


constant filter1_inc : std_logic_vector(7 downto 0):= "00111111";                    constant filter1_dec :
std_logic_vector(7 downto 0):= "00101010";

constant filter2_inc : std_logic_vector(7 downto 0):= "00101010";       constant filter2_dec :
std_logic_vector(7 downto 0):= "00100100";

constant filter3_inc : std_logic_vector(7 downto 0):= "00100100";       constant filter3_dec :
std_logic_vector(7 downto 0):= "00011111";

constant filter4_inc : std_logic_vector(7 downto 0):= "00011111";       constant filter4_dec :
std_logic_vector(7 downto 0):= "00011011";

constant filter5_inc : std_logic_vector(7 downto 0):= "00011100";       constant filter5_dec :
std_logic_vector(7 downto 0):= "00010110";

constant filter6_inc : std_logic_vector(7 downto 0):= "00010111";       constant filter6_dec :
std_logic_vector(7 downto 0):= "00010010";
```

```
constant filter7_inc : std_logic_vector(7 downto 0):= "00010010";       constant filter7_dec :
std_logic_vector(7 downto 0):= "00010000";

constant filter8_inc : std_logic_vector(7 downto 0):= "00010001";       constant filter8_dec :
std_logic_vector(7 downto 0):= "00001101";

constant filter9_inc : std_logic_vector(7 downto 0):= "00001101";       constant filter9_dec :
std_logic_vector(7 downto 0):= "00001011";

constant filter10_inc : std_logic_vector(7 downto 0):= "00001011";      constant filter10_dec :
std_logic_vector(7 downto 0):= "00001001";

constant filter11_inc : std_logic_vector(7 downto 0):= "00001001";      constant filter11_dec :
std_logic_vector(7 downto 0):= "00000111";

constant filter12_inc : std_logic_vector(7 downto 0):= "00001000";      constant filter12_dec :
std_logic_vector(7 downto 0):= "00000110";

constant filter13_inc : std_logic_vector(7 downto 0):= "00000110";      constant filter13_dec :
std_logic_vector(7 downto 0):= "00000101";


signal fft_inp                  : std_logic_vector(31 downto 0);

signal filt_inp                 : std_logic_vector(7 downto 0);

signal mult_out                 : std_logic_vector(39 downto 0);

signal filt1_sum : std_logic_vector(47 downto 0);

signal   filt2_sum  :  std_logic_vector(47  downto  0);

signal   filt3_sum  :  std_logic_vector(47  downto  0);

signal   filt4_sum  :  std_logic_vector(47  downto  0);

signal   filt5_sum  :  std_logic_vector(47  downto  0);

signal   filt6_sum  :  std_logic_vector(47  downto  0);

signal   filt7_sum  :  std_logic_vector(47  downto  0);

signal   filt8_sum  :  std_logic_vector(47  downto  0);

signal filt9_sum       : std_logic_vector(47 downto 0);

signal filt10_sum               : std_logic_vector(47 downto 0);

signal filt11_sum               : std_logic_vector(47 downto 0);

signal filt12_sum               : std_logic_vector(47 downto 0);
```

```
signal filt13_sum                : std_logic_vector(47 downto 0);

begin

   fft_addr_out       <= fft_addr;
   fft_data           <= fft_data_in;

          process (clock_in) begin
                  if (rising_edge(clock_in)) then
                          if (reset_in = '1') then
                                  MEL_state1                  <= initial;
                          else
            case MEL_state1 is

              when initial                          => if (start_in = '1') then

          MEL_state1      <= build_filt;

          ready_out       <= '0';
                                  else

          MEL_state1      <= initial;

          ready_out       <= '1';
                                  end if;
                                  ind                                  <= 0;

mem_we_a          <= (others=>'0');
```

```
filt1_sum               <= (others=>'0');

filt2_sum               <= (others=>'0');

filt3_sum               <= (others=>'0');

filt4_sum               <= (others=>'0');

filt5_sum               <= (others=>'0');

filt6_sum               <= (others=>'0');

filt7_sum               <= (others=>'0');

filt8_sum               <= (others=>'0');

filt9_sum               <= (others=>'0');

filt10_sum              <= (others=>'0');

filt11_sum              <= (others=>'0');

filt12_sum              <= (others=>'0');

filt13_sum              <= (others=>'0');


                                when build_filt                 => if (ind = 255) then

        ind                     <= 0;

        fft_addr        <= "00000000";
```

```
            MEL_state1      <= mult_wt1;
                                                                                    else
            ind                                 <= ind + 1;
            MEL_state1      <= build_filt;
                                                                                    end if;

                                                                                    if (ind
<= filter1_start) then
            mel_filt1(ind)    <= "00000000";
                                                                                    elsif
(ind <= filter1_top) then
            mel_filt1(ind)    <= mel_filt1(ind-1) + filter1_inc;
                                                                                    elsif
(ind < filter1_stop) then
            mel_filt1(ind)    <= mel_filt1(ind-1) - filter1_dec;
                                                                                    else
            mel_filt1(ind)    <= "00000000";
                                                                                    end if;

                                                                                    if (ind
<= filter2_start) then
            mel_filt2(ind)    <= "00000000";
                                                                                    elsif
(ind <= filter2_top) then
```

```
            mel_filt2(ind)    <= mel_filt2(ind-1) + filter2_inc;
                                                                    elsif
(ind < filter2_stop) then
            mel_filt2(ind)    <= mel_filt2(ind-1) - filter2_dec;
                                                                    else
            mel_filt2(ind)    <= "00000000";
                                                                    end if;

                                                                    if (ind
<= filter3_start) then
            mel_filt3(ind)    <= "00000000";
                                                                    elsif
(ind <= filter3_top) then
            mel_filt3(ind)    <= mel_filt3(ind-1) + filter3_inc;
                                                                    elsif
(ind < filter3_stop) then
            mel_filt3(ind)    <= mel_filt3(ind-1) - filter3_dec;
                                                                    else
            mel_filt3(ind)    <= "00000000";
                                                                    end if;

                                                                    if (ind
<= filter4_start) then
            mel_filt4(ind)    <= "00000000";
```

```
                                                                    elsif
(ind <= filter4_top) then

            mel_filt4(ind)    <= mel_filt4(ind-1) + filter4_inc;

                                                                    elsif
(ind < filter4_stop) then

            mel_filt4(ind)    <= mel_filt4(ind-1) - filter4_dec;

                                                                    else

            mel_filt4(ind)    <= "00000000";

                                                                    end if;


                                                                    if (ind
<= filter5_start) then

            mel_filt5(ind)    <= "00000000";

                                                                    elsif
(ind <= filter5_top) then

            mel_filt5(ind)    <= mel_filt5(ind-1) + filter5_inc;

                                                                    elsif
(ind < filter5_stop) then

            mel_filt5(ind)    <= mel_filt5(ind-1) - filter5_dec;

                                                                    else

            mel_filt5(ind)    <= "00000000";

                                                                    end if;


                                                                    if (ind
<= filter6_start) then
```

```
            mel_filt6(ind)    <= "00000000";
                                                                    elsif
(ind <= filter6_top) then
            mel_filt6(ind)    <= mel_filt6(ind-1) + filter6_inc;
                                                                    elsif
(ind < filter6_stop) then
            mel_filt6(ind)    <= mel_filt6(ind-1) - filter6_dec;
                                                                    else
            mel_filt6(ind)    <= "00000000";
                                                                    end if;


                                                                    if (ind
<= filter7_start) then
            mel_filt7(ind)    <= "00000000";
                                                                    elsif
(ind <= filter7_top) then
            mel_filt7(ind)    <= mel_filt7(ind-1) + filter7_inc;
                                                                    elsif
(ind < filter7_stop) then
            mel_filt7(ind)    <= mel_filt7(ind-1) - filter7_dec;
                                                                    else
            mel_filt7(ind)    <= "00000000";
                                                                    end if;
```

```
                                                                    if (ind
<= filter8_start) then

            mel_filt8(ind)    <= "00000000";

                                                                    elsif
(ind <= filter8_top) then

            mel_filt8(ind)    <= mel_filt8(ind-1) + filter8_inc;

                                                                    elsif
(ind < filter8_stop) then

            mel_filt8(ind)    <= mel_filt8(ind-1) - filter8_dec;

                                                                    else

            mel_filt8(ind)    <= "00000000";

                                                                    end if;


                                                                    if (ind
<= filter9_start) then

            mel_filt9(ind)    <= "00000000";

                                                                    elsif
(ind <= filter9_top) then

            mel_filt9(ind)    <= mel_filt9(ind-1) + filter9_inc;

                                                                    elsif
(ind < filter9_stop) then

            mel_filt9(ind)    <= mel_filt9(ind-1) - filter9_dec;

                                                                    else

            mel_filt9(ind)    <= "00000000";

                                                                    end if;
```

```
                                                                    if (ind
<= filter10_start) then

            mel_filt10(ind) <= "00000000";

                                                                    elsif
(ind <= filter10_top) then

            mel_filt10(ind) <= mel_filt10(ind-1) + filter10_inc;

                                                                    elsif
(ind < filter10_stop) then

            mel_filt10(ind) <= mel_filt10(ind-1) - filter10_dec;

                                                                    else

            mel_filt10(ind) <= "00000000";

                                                                    end if;


                                                                    if (ind
<= filter11_start) then

            mel_filt11(ind) <= "00000000";

                                                                    elsif
(ind <= filter11_top) then

            mel_filt11(ind) <= mel_filt11(ind-1) + filter11_inc;

                                                                    elsif
(ind < filter11_stop) then

            mel_filt11(ind) <= mel_filt11(ind-1) - filter11_dec;

                                                                    else

            mel_filt11(ind) <= "00000000";
```

```
                                                                    end if;

                                                                    if (ind
<= filter12_start) then

            mel_filt12(ind) <= "00000000";

                                                                    elsif
(ind <= filter12_top) then

            mel_filt12(ind) <= mel_filt12(ind-1) + filter12_inc;

                                                                    elsif
(ind < filter12_stop) then

            mel_filt12(ind) <= mel_filt12(ind-1) - filter12_dec;

                                                                    else

            mel_filt12(ind) <= "00000000";

                                                                    end if;



                                                                    if (ind
<= filter13_start) then

            mel_filt13(ind) <= "00000000";

                                                                    elsif
(ind <= filter13_top) then

            mel_filt13(ind) <= mel_filt13(ind-1) + filter13_inc;

                                                                    elsif
(ind < filter13_stop) then

            mel_filt13(ind) <= mel_filt13(ind-1) - filter13_dec;

                                                                    else
```

```
        mel_filt13(ind) <= "00000000";
                                                                                end if;


                                        when mult_wt1                   => MEL_state1
        <= mult_wt2;

                                        when mult_wt2                   => MEL_state1
        <= mult_st1;

                                        when mult_st1                   => fft_inp
        <= fft_data;

filt_inp            <= mel_filt1(ind);

MEL_state1      <= mult_st2;

                                        when mult_st2                   => filt_inp
        <= mel_filt2(ind);

MEL_state1      <= mult_st3;

                                        when mult_st3                   => filt_inp
        <= mel_filt3(ind);

filt1_sum    <= filt1_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st4;

                                        when mult_st4                   => filt_inp
        <= mel_filt4(ind);
```

```
filt2_sum    <= filt2_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st5;


                                        when mult_st5                   => filt_inp
        <= mel_filt5(ind);

filt3_sum    <= filt3_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st6;


                                        when mult_st6                   => filt_inp
        <= mel_filt6(ind);

filt4_sum    <= filt4_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st7;


                                        when mult_st7                   => filt_inp
        <= mel_filt7(ind);

filt5_sum    <= filt5_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st8;


                                        when mult_st8                   => filt_inp
        <= mel_filt8(ind);

filt6_sum    <= filt6_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st9;
```

```
                                        when mult_st9                   => filt_inp
        <= mel_filt9(ind);

filt7_sum    <= filt7_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st10;


                                        when mult_st10                  => filt_inp
        <= mel_filt10(ind);

filt8_sum    <= filt8_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st11;


                                        when mult_st11                  => filt_inp
        <= mel_filt11(ind);

filt9_sum    <= filt9_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st12;


                                        when mult_st12                  => filt_inp
        <= mel_filt12(ind);

filt10_sum    <= filt10_sum + ("00000000" & mult_out);

MEL_state1      <= mult_st13;


                                        when mult_st13                  => filt_inp
        <= mel_filt13(ind);

filt11_sum    <= filt11_sum + ("00000000" & mult_out);
```

```
MEL_state1      <= mult_st14;

                                        when mult_st14                  => filt12_sum   <=
filt12_sum + ("00000000" &
mult_out);

MEL_state1      <= mult_st15;

                                        when mult_st15                  => filt13_sum   <=
filt13_sum + ("00000000" & mult_out);

                                                                                        if
(fft_addr = "11111111") then

        fft_addr        <= "00000000";

        ind                             <= 0;

        MEL_state1      <= write_st1;

                                                                                        else

        fft_addr        <= fft_addr + 1;

        ind                             <= ind + 1;

        MEL_state1      <= mult_wt1;

                                                                                        end if;


                                        when write_st1                  => mem_addr_a     <=
"0000";
                        mem_data_a      <= filt1_sum;
                        mem_we_a        <=
                        (others=>'1');
```

```
                    MEL_state1      <= write_st2;

                        when write_st2                  => mem_addr_a     <=
"0001";
                    mem_data_a      <= filt2_sum;
                    mem_we_a        <=
                    (others=>'1'); MEL_state1
                                <= write_st3;

                        when write_st3                  => mem_addr_a     <=
"0010";
                    mem_data_a      <= filt3_sum;
                    mem_we_a        <=
                    (others=>'1'); MEL_state1
                                <= write_st4;

                        when write_st4                  => mem_addr_a     <=
"0011";
                    mem_data_a      <= filt4_sum;
                    mem_we_a        <=
                    (others=>'1'); MEL_state1
                                <= write_st5;

                        when write_st5                  => mem_addr_a     <=
"0100";
                    mem_data_a      <= filt5_sum;
                    mem_we_a        <=
                    (others=>'1'); MEL_state1
                                <= write_st6;

                        when write_st6                  => mem_addr_a     <=
```

"0101";

mem_data_a <= filt6_sum;

```
                mem_we_a        <= (others=>'1');
                MEL_state1      <= write_st7;

                when write_st7                  => mem_addr_a      <=
"0110";
                mem_data_a      <= filt7_sum;
                mem_we_a        <=
(others=>'1'); MEL_state1
                                <= write_st8;

                when write_st8                  => mem_addr_a      <=
"0111";
                mem_data_a      <= filt8_sum;
                mem_we_a        <=
(others=>'1'); MEL_state1
                                <= write_st9;

                when write_st9                  => mem_addr_a      <=
"1000";
                mem_data_a      <= filt9_sum;
                mem_we_a        <=
(others=>'1'); MEL_state1
                                <= write_st10;

                when write_st10                         => mem_addr_a
<= "1001";
                mem_data_a      <= filt10_sum;
                mem_we_a        <=
(others=>'1'); MEL_state1
                                <= write_st11;
```

```
when write_st11 => mem_addr_a <= "1010";
```

```
                                    mem_data_a      <= filt11_sum;
                                    mem_we_a        <=
                                    (others=>'1'); MEL_state1
                                                    <= write_st12;

                                            when write_st12                         => mem_addr_a
<= "1011";
                                    mem_data_a      <= filt12_sum;
                                    mem_we_a        <=
                                    (others=>'1'); MEL_state1
                                                    <= write_st13;

                                            when write_st13                         => mem_addr_a
<= "1100";
                                    mem_data_a      <= filt13_sum;
                                    mem_we_a        <=
                                    (others=>'1'); MEL_state1
                                                    <= initial;



          when others                               => MEL_state1           <= initial;


        end case;
      end if;
              end if;
     end process;

  Component1: Multiplier1_mel
  PORT MAP    ( CLK      => clock_in,
```

```
A       => fft_inp,
```

```
        B       => filt_inp,
        P       => mult_out);


    Component2 : Memory1_mel
    PORT MAP    (       clka    => clock_in,
                        addra => mem_addr_a,
                        dina    => mem_data_a,
                        wea     => mem_we_a,
                        clkb    => clock_in,
                        addrb   => mem_addr_in,
                        doutb   => mem_data_out);

end Lab_MEL_beh;


entity Lab_DCT is
  Port  ( reset_in      : in STD_LOGIC;
                        clock_in      : in STD_LOGIC;
        mel_addr_out : out STD_LOGIC_VECTOR(3 downto 0);
        mel_data_in : in STD_LOGIC_VECTOR(47 downto 0);
        mem_addr_in : in STD_LOGIC_VECTOR(3 downto 0);
        mem_data_out : out STD_LOGIC_VECTOR(67 downto 0);
                        start_in      : in STD_LOGIC;
        ready_out     : out STD_LOGIC);
end Lab_DCT;
```

```
architecture Lab_DCT_beh of LAB_dct is

component Multiplier1_dct
   Port ( CLK      : in STD_LOGIC;
          A        : in STD_LOGIC_VECTOR(47 DOWNTO 0);
          B        : in STD_LOGIC_VECTOR(15 DOWNTO 0); P
                   : out STD_LOGIC_VECTOR(63 DOWNTO 0));
end component;

component Memory1_dct
   Port ( clka     : in STD_LOGIC;
          addra    : in STD_LOGIC_VECTOR(3 downto 0);
          dina     : in STD_LOGIC_VECTOR(67 downto 0);
          wea      : in STD_LOGIC_VECTOR(0 downto 0);
          clkb     : in STD_LOGIC;
          addrb    : in STD_LOGIC_VECTOR(3 downto 0);
          doutb    : out STD_LOGIC_VECTOR(67 downto 0));
end component;

signal mel_addr                  : std_logic_vector(3 downto 0);
signal mel_data                  : std_logic_vector(47 downto 0);

-- MEL memory signals
signal mem_addr_a                  : std_logic_vector(3 downto 0):=(others=>'0');
signal mem_data_a                  : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_we_a                      : std_logic_vector(0 downto
0):=(others=>'0');
```

```
type state_type1 is (initial, mult_wt1, mult_wt2, mult_st1, mult_st2, mult_st3, mult_st4, mult_st5,
mult_st6, mult_st7, mult_st8, mult_st9, mult_st10, mult_st11, mult_st12,

                                        mult_st13, mult_st14, mult_st15, write_st, check_st);

signal DCT_state1     : state_type1 := initial;


type dct_coeff is array (0 to 12) of std_logic_vector(15 downto 0);

signal dct_filt1 : dct_coeff := (X"7FFF", X"7FFF", X"7FFF", X"7FFF", X"7FFF", X"7FFF", X"7FFF",
X"7FFF", X"7FFF", X"7FFF", X"7FFF", X"7FFF", X"7FFF");

signal dct_filt2 : dct_coeff := (X"7F10", X"77AE", X"6957", X"54E1", X"3B7C", X"1EA2", X"0000",
X"E15E", X"C484", X"AB1F", X"96A9", X"8852", X"80F0");

signal dct_filt3 : dct_coeff := (X"7C47", X"5FCE", X"2D63", X"F092", X"B74A", X"8EAA", X"8001",
X"8EAA", X"B74A", X"F092", X"2D63", X"5FCE", X"7C47");

signal dct_filt4 : dct_coeff := (X"77AE", X"3B7C", X"E15E", X"96A9", X"80F0", X"AB1F", X"0000",
X"54E1", X"7F10", X"6957", X"1EA2", X"C484", X"8852");

signal dct_filt5 : dct_coeff := (X"7156", X"0F6E", X"A032", X"83B9", X"D29D", X"48B6", X"7FFF",
X"48B6", X"D29D", X"83B9", X"A032", X"0F6E", X"7156");

signal dct_filt6 : dct_coeff := (X"6957", X"E15E", X"80F0", X"C484", X"54E1", X"77AE", X"0000",
X"8852", X"AB1F", X"3B7C", X"7F10", X"1EA2", X"96A9");

signal dct_filt7 : dct_coeff := (X"5FCE", X"B74A", X"8EAA", X"2D63", X"7C47", X"F092", X"8001",
X"F092", X"7C47", X"2D63", X"8EAA", X"B74A", X"5FCE");

signal dct_filt8 : dct_coeff := (X"54E1", X"96A9", X"C484", X"77AE", X"1EA2", X"80F0", X"0000",
X"7F10", X"E15E", X"8852", X"3B7C", X"6957", X"AB1F");

signal dct_filt9 : dct_coeff := (X"48B6", X"83B9", X"0F6E", X"7156", X"A032", X"D29D", X"7FFF",
X"D29D", X"A032", X"7156", X"0F6E", X"83B9", X"48B6");

signal dct_filt10 : dct_coeff := (X"3B7C", X"80F0", X"54E1", X"1EA2", X"8852", X"6957", X"0000",
X"96A9", X"77AE", X"E15E", X"AB1F", X"7F10", X"C484");

signal dct_filt11 : dct_coeff := (X"2D63", X"8EAA", X"7C47", X"B74A", X"F092", X"5FCE", X"8001",
X"5FCE", X"F092", X"B74A", X"7C47", X"8EAA", X"2D63");

signal dct_filt12 : dct_coeff := (X"1EA2", X"AB1F", X"77AE", X"80F0", X"6957", X"C484", X"0000",
X"3B7C", X"96A9", X"7F10", X"8852", X"54E1", X"E15E");

signal dct_filt13 : dct_coeff := (X"0F6E", X"D29D", X"48B6", X"A032", X"7156", X"83B9", X"7FFF",
X"83B9", X"7156", X"A032", X"48B6", X"D29D", X"0F6E");

signal dct_filt                 : std_logic_vector(15 downto 0);
```

```
signal filter_order              : std_logic_vector(3 downto 0):="0000";
signal ind                       : integer range 0 to 12 := 0;

signal dct_inp                   : std_logic_vector(47 downto 0);
signal filt_inp                  : std_logic_vector(15 downto 0);
signal mult_out                  : std_logic_vector(63 downto 0);
signal mult_out_ext              : std_logic_vector(67 downto 0);
signal filt_sum             : std_logic_vector(67 downto 0);

begin

   mel_addr_out       <= mel_addr;
   mel_data                <= mel_data_in;

   dct_filt <= dct_filt1(ind) when filter_order = "0000" else
               dct_filt2(ind) when filter_order = "0001" else
               dct_filt3(ind) when filter_order = "0010" else
               dct_filt4(ind) when filter_order = "0011" else
               dct_filt5(ind) when filter_order = "0100" else
               dct_filt6(ind) when filter_order = "0101" else
               dct_filt7(ind) when filter_order = "0110" else
               dct_filt8(ind) when filter_order = "0111" else
               dct_filt9(ind) when filter_order = "1000" else
               dct_filt10(ind) when filter_order = "1001" else
               dct_filt11(ind) when filter_order = "1010" else
               dct_filt12(ind) when filter_order = "1011" else
               dct_filt13(ind);
```

```
mult_out_ext        <= mult_out(63) & mult_out(63) & mult_out(63) & mult_out(63) & mult_out;


        process (clock_in) begin
                if (rising_edge(clock_in)) then
                        if (reset_in = '1') then
                                DCT_state1                      <= initial;
                        else
          case DCT_state1 is

            when initial                                => if (start_in = '1') then
        DCT_state1      <= mult_wt1;
        ready_out       <= '0';
                                else
        DCT_state1      <= initial;
        ready_out       <= '1';
                                end if;
                                ind                                     <= 0;
mel_addr                        <= "0000";
filter_order                    <= "0000";
mem_we_a        <= (others=>'0');
filt_sum                        <= (others=>'0');
```

```
                                        when mult_wt1                   => DCT_state1
        <= mult_wt2;
                                                        mel_addr        <= mel_addr + 1;


                                        when mult_wt2                   => DCT_state1
        <= mult_st1;
                                                        mel_addr        <= mel_addr + 1;


                                        when mult_st1                   => dct_inp
        <= mel_data;

filt_inp        <= dct_filt;

mel_addr                <= mel_addr + 1;

                                                                                        ind
                <= ind + 1;

DCT_state1      <= mult_st2;


                                        when mult_st2                   => dct_inp
        <= mel_data;

filt_inp        <= dct_filt;

mel_addr                <= mel_addr + 1;

                                                                                        ind
                <= ind + 1;

DCT_state1      <= mult_st3;
```

```
                                        when mult_st3                   => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;

mel_addr                    <= mel_addr + 1;

                                                                                ind
                    <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st4;


                                        when mult_st4                   => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;

mel_addr                    <= mel_addr + 1;

                                                                                ind
                    <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st5;


                                        when mult_st5                   => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;

mel_addr                    <= mel_addr + 1;
```

```
                                                                    ind
                <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st6;


                                        when mult_st6                   => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;

mel_addr                <= mel_addr + 1;
                                                                    ind
                <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st7;


                                        when mult_st7                   => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;

mel_addr                <= mel_addr + 1;
                                                                    ind
                <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st8;
```

```
                                        when mult_st8                   => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;

mel_addr                    <= mel_addr + 1;

                                                                                ind
                    <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st9;


                                        when mult_st9                   => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;

mel_addr                    <= mel_addr + 1;

                                                                                ind
                    <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st10;


                                        when mult_st10                  => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;

mel_addr                    <= mel_addr + 1;
```

```
                                                                                ind
                <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st11;


                                        when mult_st11                  => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;
                                                                                ind
                <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st12;


                                        when mult_st12                  => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;
                                                                                ind
                <= ind + 1;

filt_sum    <= filt_sum + mult_out_ext;

DCT_state1      <= mult_st13;


                                        when mult_st13                  => dct_inp
        <= mel_data;

filt_inp            <= dct_filt;
```

```
filt_sum     <= filt_sum + mult_out_ext;

DCT_state1       <= mult_st14;


                                        when mult_st14                          => filt_sum     <=
filt_sum + mult_out_ext;

DCT_state1       <= mult_st15;


                                        when mult_st15                          => filt_sum     <=
filt_sum + mult_out_ext;
                                                                DCT_state1 <= write_st;


                                        when write_st                           => mem_addr_a      <=
filter_order;
                                mem_data_a      <= filt_sum;
                                mem_we_a        <=
                                (others=>'1'); DCT_state1
                                                <= check_st;


            when check_st                               => if (filter_order = "1100") then
        -- 12

        filter_order        <= "0000";

        DCT_state1      <= initial;
                                                                                                else

        filter_order        <= filter_order + 1;

        DCT_state1      <= mult_wt1;
```

```
                                                                                    end if;

                                                                                    ind
                        <= 0;

mel_addr                        <= "0000";

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

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



          when others                                   => DCT_state1                 <= initial;

        end case;
      end if;
            end if;
      end process;

  Component1: Multiplier1_dct
  PORT MAP    ( CLK     => clock_in,
          A       => dct_inp,
          B       => filt_inp,
          P       => mult_out);


      Component2 : Memory1_dct
      PORT MAP     (          clka     => clock_in,
                              addra => mem_addr_a,
```

```
                                    dina    => mem_data_a,
                                    wea     => mem_we_a,
                                    clkb    => clock_in,
                                    addrb   => mem_addr_in,
                                    doutb   => mem_data_out);

end Lab_DCT_beh;



entity Lab_CMP is
   Port    ( reset_in      : in STD_LOGIC;
                             clock_in      : in STD_LOGIC;
             dct_addr_out : out STD_LOGIC_VECTOR(3 downto 0);
             dct_data_in : in STD_LOGIC_VECTOR(67 downto 0);
             record_in     : in STD_LOGIC;
             number_in     : in STD_LOGIC_VECTOR(3 downto 0);
             led_out       : out STD_LOGIC_VECTOR(6 downto 0);
                             start_in      : in STD_LOGIC;
             ready_out     : out STD_LOGIC);
end Lab_CMP;

architecture Lab_CMP_beh of Lab_CMP is

component Memory1_cmp
   Port ( clka     : in STD_LOGIC;
          addra    : in STD_LOGIC_VECTOR(9 downto 0);
```

```
        dina    : in STD_LOGIC_VECTOR(67 downto 0);
        wea     : in STD_LOGIC_VECTOR(0 downto 0);
        clkb    : in STD_LOGIC;
        addrb   : in STD_LOGIC_VECTOR(9 downto 0);
        doutb   : out STD_LOGIC_VECTOR(67 downto 0));
end component;

type state_type1 is (initial, mem_wt1, mem_wt2, wrt_st1, wrt_st2, wrt_st3, wrt_st4, wrt_st5,
wrt_st6); signal CMP_state1    : state_type1 := initial;

type state_type2 is (initial, mem_wt1, mem_wt2, mem_wt3, mem_wt4, read_st1, read_st2, read_st3,
read_st4, read_st5, read_st6);
signal CMP_state2     : state_type2 := initial;

signal rec_data             : std_logic;
signal rec_number           : std_logic_vector(3 downto 0);

signal dct_addr                     : std_logic_vector(3 downto 0);
signal dct_data                     : std_logic_vector(67 downto 0);

signal mem_addr_a           : std_logic_vector(9 downto 0);
signal mem_addr_b           : std_logic_vector(9 downto 0);
signal mem_data_a                   : std_logic_vector(67 downto 0):=(others=>'0');
--------------------------------------------------------------------------------
signal mem_we_a                         : std_logic_vector(0 downto
0):=(others=>'0'); signal mem_we0_a: std_logic_vector(0 downto
0):=(others=>'0'); signal mem_we1_a: std_logic_vector(0 downto
0):=(others=>'0');
```

```
signal mem_we2_a                : std_logic_vector(0 downto 0):=(others=>'0');
signal mem_we3_a                : std_logic_vector(0 downto 0):=(others=>'0');
signal mem_we4_a                : std_logic_vector(0 downto 0):=(others=>'0');
signal mem_we5_a                : std_logic_vector(0 downto 0):=(others=>'0');
signal mem_we6_a                : std_logic_vector(0 downto 0):=(others=>'0');
signal mem_we7_a                : std_logic_vector(0 downto 0):=(others=>'0');
signal mem_we8_a                : std_logic_vector(0 downto 0):=(others=>'0');
signal mem_we9_a                : std_logic_vector(0 downto 0):=(others=>'0');
signal mem_weT_a                : std_logic_vector(0 downto 0):=(others=>'0');
--------------------------------------------------------------------------------
signal mem_data0_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data1_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data2_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data3_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data4_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data5_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data6_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data7_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data8_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_data9_b              : std_logic_vector(67 downto 0):=(others=>'0');
signal mem_dataT_b              : std_logic_vector(67 downto 0):=(others=>'0');
--------------------------------------------------------------------------------
signal diff0                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff1                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff2                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff3                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff4                : std_logic_vector(67 downto 0):=(others=>'0');
```

```
signal diff5                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff6                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff7                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff8                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff9                : std_logic_vector(67 downto 0):=(others=>'0');
--------------------------------------------------------------------------------
signal diff_abs0                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs1                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs2                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs3                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs4                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs5                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs6                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs7                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs8                : std_logic_vector(67 downto 0):=(others=>'0');
signal diff_abs9                : std_logic_vector(67 downto 0):=(others=>'0');
--------------------------------------------------------------------------------
signal diff_sum0                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum1                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum2                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum3                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum4                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum5                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum6                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum7                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum8                : std_logic_vector(77 downto 0):=(others=>'0');
signal diff_sum9                : std_logic_vector(77 downto 0):=(others=>'0');
```

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

signal diff_sum_01                  : std_logic_vector(77 downto 0):=(others=>'0');

signal diff_sum_23                  : std_logic_vector(77 downto 0):=(others=>'0');

signal diff_sum_45                  : std_logic_vector(77 downto 0):=(others=>'0');

signal diff_sum_67                  : std_logic_vector(77 downto 0):=(others=>'0');

signal diff_sum_89                  : std_logic_vector(77 downto 0):=(others=>'0');

signal diff_sum_03                  : std_logic_vector(77 downto 0):=(others=>'0');

signal diff_sum_47                  : std_logic_vector(77 downto 0):=(others=>'0');

signal diff_sum_07                  : std_logic_vector(77 downto 0):=(others=>'0');

signal diff_sum_09                  : std_logic_vector(77 downto 0):=(others=>'0');

---------------------------------------------------------------------------------

signal det_num_01                   : std_logic_vector(3 downto 0):=(others=>'0');

signal det_num_23                   : std_logic_vector(3 downto 0):=(others=>'0');

signal det_num_45                   : std_logic_vector(3 downto 0):=(others=>'0');

signal det_num_67                   : std_logic_vector(3 downto 0):=(others=>'0');

signal det_num_89                   : std_logic_vector(3 downto 0):=(others=>'0');

signal det_num_03                   : std_logic_vector(3 downto 0):=(others=>'0');

signal det_num_47                   : std_logic_vector(3 downto 0):=(others=>'0');

signal det_num_07                   : std_logic_vector(3 downto 0):=(others=>'0');

signal det_num_09                   : std_logic_vector(3 downto 0):=(others=>'0');


signal rec_led              : std_logic_vector(6 DOWNTO 0):= (others=>'0');

signal det_led              : std_logic_vector(6 DOWNTO 0):= (others=>'0');


signal start_in2          : std_logic:='0';

signal ready_out2                   : std_logic:='0';
```

```
begin

   dct_addr_out        <= dct_addr;
   dct_data            <= dct_data_in;
   led_out             <= rec_led when rec_data = '1'
                      else det_led when rec_data = '0';


         process (clock_in) begin
                  if (rising_edge(clock_in)) then
                           if (reset_in = '1') then
                                    CMP_state1                    <= initial;
                           else
           case CMP_state1 is

              when initial                              => if (start_in = '1') then
         CMP_state1       <= mem_wt1;
         ready_out        <= '0';
         rec_data         <= record_in;
         rec_number       <= number_in;
                                    else
         CMP_state1       <= initial;
         ready_out        <= '1';
                                    end if;
```

```
dct_addr                    <= (others=>'0');

mem_addr_a      <= (others=>'1');

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

start_in2       <= '0';



                        when mem_wt1                    => CMP_state1
        <= mem_wt2;
                                dct_addr        <= dct_addr + 1;


                        when mem_wt2                    => CMP_state1
        <= wrt_st1;
                                dct_addr        <= dct_addr + 1;


                        when wrt_st1                    => if (dct_addr =
"1110") then
                    CMP_state1      <= wrt_st2;
                else
                    dct_addr        <= dct_addr + 1;
                    CMP_state1      <= wrt_st1;
                end if;
                mem_addr_a      <= mem_addr_a + 1;
                mem_data_a      <= dct_data;
                mem_we_a        <= (others=>'1');
```

```
                    when wrt_st2                    => mem_we_a
<= (others=>'0');
                    if (mem_addr_a = "1100110010") then      -- 818
                      CMP_state1     <= wrt_st4;
                    else
                      CMP_state1     <= wrt_st3;
                    end if;

          when wrt_st3              => if (start_in = '1')
                      then CMP_state1          <=
                      mem_wt1; ready_out
                                <= '0';
                    else
                      CMP_state1     <= wrt_st3;
                      ready_out      <= '1';
                    end if;
                    dct_addr       <= (others=>'0');

          when wrt_st4              => if (rec_data = '1') then
                      CMP_state1     <= initial;
                    else
                      CMP_state1     <= wrt_st5;
                      start_in2      <= '1';
                    end if;

          when wrt_st5              => start_in2      <= '0';
                    CMP_state1      <= wrt_st6;
```

```
        when wrt_st6                    => if (ready_out2 = '1') then
                                              CMP_state1      <= initial;
                                           else
                                              CMP_state1      <= wrt_st6;
                                           end if;

        when others                                         => CMP_state1                   <= initial;

      end case;
    end if;
          end if;
   end process;

mem_weT_a <= mem_we_a when rec_data = '0' else (others=>'0');
mem_we0_a <= mem_we_a when (rec_data = '1' and rec_number = "0000") else (others=>'0');
mem_we1_a <= mem_we_a when (rec_data = '1' and rec_number = "0001") else (others=>'0');
mem_we2_a <= mem_we_a when (rec_data = '1' and rec_number = "0010") else (others=>'0');
mem_we3_a <= mem_we_a when (rec_data = '1' and rec_number = "0011") else (others=>'0');
mem_we4_a <= mem_we_a when (rec_data = '1' and rec_number = "0100") else (others=>'0');
mem_we5_a <= mem_we_a when (rec_data = '1' and rec_number = "0101") else (others=>'0');
mem_we6_a <= mem_we_a when (rec_data = '1' and rec_number = "0110") else (others=>'0');
mem_we7_a <= mem_we_a when (rec_data = '1' and rec_number = "0111") else (others=>'0');
mem_we8_a <= mem_we_a when (rec_data = '1' and rec_number = "1000") else (others=>'0');
mem_we9_a <= mem_we_a when (rec_data = '1' and rec_number = "1001") else (others=>'0');
```

```
rec_led         <= "0000001" when rec_number = "0000" else
                   "1001111" when rec_number = "0001" else
                   "0010010" when rec_number = "0010" else
                   "0000110" when rec_number = "0011" else
                   "1001100" when rec_number = "0100" else
                   "0100100" when rec_number = "0101" else
                   "0100000" when rec_number = "0110" else
                   "0001111" when rec_number = "0111" else
                   "0000000" when rec_number = "1000" else
                   "0000100" when rec_number = "1001";


        process (clock_in) begin
                if (rising_edge(clock_in)) then
                        if (reset_in = '1') then
                                CMP_state2                      <= initial;
                        else
          case CMP_state2 is

            when initial                                => if (start_in2 = '1') then

        CMP_state2      <= mem_wt1;

        ready_out2      <= '0';
                                    else

        CMP_state2      <= initial;
```

```
            ready_out2      <= '1';
                                    end if;
mem_addr_b      <= (others=>'0');
diff_sum0       <= (others=>'0');
diff_sum1       <= (others=>'0');
diff_sum2       <= (others=>'0');
diff_sum3       <= (others=>'0');
diff_sum4       <= (others=>'0');
diff_sum5       <= (others=>'0');
diff_sum6       <= (others=>'0');
diff_sum7       <= (others=>'0');
diff_sum8       <= (others=>'0');
diff_sum9       <= (others=>'0');

                                    when mem_wt1                    => CMP_state2
            <= mem_wt2;
                                                mem_addr_b <= mem_addr_b + 1;

                                    when mem_wt2                    => CMP_state2
            <= mem_wt3;
                                                mem_addr_b<= mem_addr_b + 1;
```

```
                    when mem_wt3                    => CMP_state2
<= mem_wt4;
                                                    mem_addr_b<= mem_addr_b + 1;

                    when mem_wt4                    => CMP_state2
<= read_st1;
                                                    mem_addr_b<= mem_addr_b + 1;


                    when read_st1                   => if (mem_addr_b =
"1100110110") then      -- 822
                        mem_addr_b      <=
                        (others=>'0'); CMP_state2   <=
                        read_st2;
                    else
                        mem_addr_b      <= mem_addr_b + 1;
                        CMP_state2      <= read_st1;
                    end if;
                    diff_sum0 <= diff_sum0 + ("0000000000" & diff_abs0);
                    diff_sum1 <= diff_sum1 + ("0000000000" & diff_abs1);
                    diff_sum2 <= diff_sum2 + ("0000000000" & diff_abs2);
                    diff_sum3 <= diff_sum3 + ("0000000000" & diff_abs3);
                    diff_sum4 <= diff_sum4 + ("0000000000" & diff_abs4);
                    diff_sum5 <= diff_sum5 + ("0000000000" & diff_abs5);
                    diff_sum6 <= diff_sum6 + ("0000000000" & diff_abs6);
                    diff_sum7 <= diff_sum7 + ("0000000000" & diff_abs7);
                    diff_sum8 <= diff_sum8 + ("0000000000" & diff_abs8);
                    diff_sum9 <= diff_sum9 + ("0000000000" & diff_abs9);
```

```
                    when read_st2                    => if (diff_sum0 <
diff_sum1) then
                        det_num_01      <= "0000";
                        diff_sum_01     <= diff_sum0;
                    else
                        det_num_01      <= "0001";
                        diff_sum_01     <= diff_sum1;
                    end if;

                    if (diff_sum2 < diff_sum3) then
                        det_num_23      <= "0010";
                        diff_sum_23     <= diff_sum2;
                    else
                        det_num_23      <= "0011";
                        diff_sum_23     <= diff_sum3;
                    end if;

                    if (diff_sum4 < diff_sum5) then
                        det_num_45      <= "0100";
                        diff_sum_45     <= diff_sum4;
                    else
                        det_num_45      <= "0101";
                        diff_sum_45     <= diff_sum5;
                    end if;

                    if (diff_sum6 < diff_sum7) then
```

```
        det_num_67      <= "0110";
        diff_sum_67     <= diff_sum6;
    else
        det_num_67      <= "0111";
        diff_sum_67     <= diff_sum7;
    end if;

    if (diff_sum8 < diff_sum9) then
        det_num_89      <= "1000";
        diff_sum_89     <= diff_sum8;
    else
        det_num_89      <= "1001";
        diff_sum_89     <= diff_sum9;
    end if;
    CMP_state2          <= read_st3;

            when read_st3                   => if (diff_sum_01 <
diff_sum_23) then
        det_num_03      <= det_num_01;
        diff_sum_03     <= diff_sum_01;
    else
        det_num_03      <= det_num_23;
        diff_sum_03     <= diff_sum_23;
    end if;

    if (diff_sum_45 < diff_sum_67) then
        det_num_47      <= det_num_45;
```

```
                diff_sum_47    <= diff_sum_45;
            else
                det_num_47     <= det_num_67;
                diff_sum_47    <= diff_sum_67;
            end if;
            CMP_state2         <= read_st4;

                when read_st4                   => if (diff_sum_03 <
diff_sum_47) then
                det_num_07     <= det_num_03;
                diff_sum_07    <= diff_sum_03;
            else
                det_num_07     <= det_num_47;
                diff_sum_07    <= diff_sum_47;
            end if;
            CMP_state2         <= read_st5;

                when read_st5                   => if (diff_sum_89 <
diff_sum_07) then
                det_num_09     <= det_num_89;
                diff_sum_09    <= diff_sum_89;
            else
                det_num_09     <= det_num_07;
                diff_sum_09    <= diff_sum_07;
            end if;
            CMP_state2         <= read_st6;

        when read_st6                   =>      case det_num_09 is
```

```
                when "0000"    => det_led     <= "0000001";
                when "0001"    => det_led     <= "1001111";
                when "0010"    => det_led     <= "0010010";
                when "0011"    => det_led     <= "0000110";
                when "0100"    => det_led     <= "1001100";
                when "0101"    => det_led     <= "0100100";
                when "0110"    => det_led     <= "0100000";
                when "0111"    => det_led     <= "0001111";
                when "1000"    => det_led     <= "0000000";
                when "1001"    => det_led     <= "0000100";
                when others      => det_led       <= "0000001";

            end case;

        CMP_state2     <= initial;

    when others                         => CMP_state2                   <= initial;

  end case;
end if;
        end if;
 end process;
```

```
 process (clock_in) begin
          if (rising_edge(clock_in)) then
diff0     <= mem_data0_b - mem_dataT_b;
diff1     <= mem_data1_b - mem_dataT_b;
diff2     <= mem_data2_b - mem_dataT_b;
diff3     <= mem_data3_b - mem_dataT_b;
diff4     <= mem_data4_b - mem_dataT_b;
diff5     <= mem_data5_b - mem_dataT_b;
diff6     <= mem_data6_b - mem_dataT_b;
diff7     <= mem_data7_b - mem_dataT_b;
diff8     <= mem_data8_b - mem_dataT_b;
diff9     <= mem_data9_b - mem_dataT_b;
          end if;
 end process;

 process (clock_in) begin
          if (rising_edge(clock_in)) then
if (diff0(67) = '1') then     diff_abs0  <= not(diff0)+1;   else diff_abs0  <= diff0;  end if;
if (diff1(67) = '1') then     diff_abs1  <= not(diff1)+1;   else diff_abs1  <= diff1;  end if;
if (diff2(67) = '1') then     diff_abs2  <= not(diff2)+1;   else diff_abs2  <= diff2;  end if;
if (diff3(67) = '1') then     diff_abs3  <= not(diff3)+1;   else diff_abs3  <= diff3;  end if;
if (diff4(67) = '1') then     diff_abs4  <= not(diff4)+1;   else diff_abs4  <= diff4;  end if;
if (diff5(67) = '1') then     diff_abs5  <= not(diff5)+1;   else diff_abs5  <= diff5;  end if;
if (diff6(67) = '1') then     diff_abs6  <= not(diff6)+1;   else diff_abs6  <= diff6;  end if;
if (diff7(67) = '1') then     diff_abs7  <= not(diff7)+1;   else diff_abs7  <= diff7;  end if;
if (diff8(67) = '1') then     diff_abs8  <= not(diff8)+1;   else diff_abs8  <= diff8;  end if;
if (diff9(67) = '1') then     diff_abs9  <= not(diff9)+1;   else diff_abs9  <= diff9;  end if;
```

```
        end if;
end process;

Component0 : Memory1_cmp
PORT MAP    (       clka    => clock_in,
                    addra => mem_addr_a,
                    dina    => mem_data_a,
                    wea     => mem_we0_a,
                    clkb    => clock_in,
                    addrb   => mem_addr_b,
                    doutb   => mem_data0_b);

Component1 : Memory1_cmp
PORT MAP    (       clka    => clock_in,
                    addra => mem_addr_a,
                    dina    => mem_data_a,
                    wea     => mem_we1_a,
                    clkb    => clock_in,
                    addrb   => mem_addr_b,
                    doutb   => mem_data1_b);

Component2 : Memory1_cmp
PORT MAP    (       clka    => clock_in,
                    addra => mem_addr_a,
                    dina    => mem_data_a,
                    wea     => mem_we2_a,
                    clkb    => clock_in,
```

```
                    addrb    => mem_addr_b,
                    doutb    => mem_data2_b);

Component3 : Memory1_cmp
PORT MAP     (      clka     => clock_in,
                    addra => mem_addr_a,
                    dina     => mem_data_a,
                    wea      => mem_we3_a,
                    clkb     => clock_in,
                    addrb    => mem_addr_b,
                    doutb    => mem_data3_b);

Component4 : Memory1_cmp
PORT MAP     (      clka     => clock_in,
                    addra => mem_addr_a,
                    dina     => mem_data_a,
                    wea      => mem_we4_a,
                    clkb     => clock_in,
                    addrb    => mem_addr_b,
                    doutb    => mem_data4_b);

Component5 : Memory1_cmp
PORT MAP     (      clka     => clock_in,
                    addra => mem_addr_a,
                    dina     => mem_data_a,
                    wea      => mem_we5_a,
                    clkb     => clock_in,
```

```
                    addrb    => mem_addr_b,
                    doutb    => mem_data5_b);

Component6 : Memory1_cmp
PORT MAP     (      clka     => clock_in,
                    addra => mem_addr_a,
                    dina     => mem_data_a,
                    wea      => mem_we6_a,
                    clkb     => clock_in,
                    addrb    => mem_addr_b,
                    doutb    => mem_data6_b);

Component7 : Memory1_cmp
PORT MAP     (      clka     => clock_in,
                    addra => mem_addr_a,
                    dina     => mem_data_a,
                    wea      => mem_we7_a,
                    clkb     => clock_in,
                    addrb    => mem_addr_b,
                    doutb    => mem_data7_b);

Component8 : Memory1_cmp
PORT MAP     (      clka     => clock_in,
                    addra => mem_addr_a,
                    dina     => mem_data_a,
                    wea      => mem_we8_a,
                    clkb     => clock_in,
```

```
                                addrb    => mem_addr_b,
                                doutb    => mem_data8_b);

        Component9 : Memory1_cmp
        PORT MAP     (          clka     => clock_in,
                                addra => mem_addr_a,
                                dina     => mem_data_a,
                                wea      => mem_we9_a,
                                clkb     => clock_in,
                                addrb    => mem_addr_b,
                                doutb    => mem_data9_b);

        ComponentT : Memory1_cmp
        PORT MAP     (          clka     => clock_in,
                                addra => mem_addr_a,
                                dina     => mem_data_a,
                                wea      => mem_weT_a,
                                clkb     => clock_in,
                                addrb    => mem_addr_b,
                                doutb    => mem_dataT_b);



end Lab_CMP_beh;


entity test is
```

```
end test;

architecture Test_Beh of test is

component Lab_ADC
   Port    ( reset_in          : in STD_LOGIC;
                               clock_in     : in STD_LOGIC;
                               spi_clk_out : out STD_LOGIC;
                               chip_sel_out : out STD_LOGIC;
                               spi_data_in : in STD_LOGIC;
                               addr_in             : in STD_LOGIC_VECTOR(13 downto 0);
                               data_out            : out STD_LOGIC_VECTOR(11 downto 0);
                               start_in     : in STD_LOGIC;
          ready_out    : out STD_LOGIC);
end component;

component Lab_WINDOW
   Port    ( reset_in      : in STD_LOGIC;
          clock_in          : in STD_LOGIC;
          frame_addr_in : in STD_LOGIC_VECTOR(13 downto 0);
          adc_addr_out : out STD_LOGIC_VECTOR(13 downto 0);
          adc_data_in : in STD_LOGIC_VECTOR(11 downto 0);
          mem_addr_in : in STD_LOGIC_VECTOR(8 downto 0);
          mem_data_out : out STD_LOGIC_VECTOR(19 downto 0);
          start_in      : in STD_LOGIC;
          ready_out     : out STD_LOGIC);
end component;
```

```
component Lab_CTRL
   Port    ( reset_in          : in STD_LOGIC;
             clock_in              : in STD_LOGIC;
             start_adc_out     : out STD_LOGIC;
             ready_adc_in      : in STD_LOGIC;
             frame_addr_out    : out STD_LOGIC_VECTOR(13 downto
             0); start_window_out : out STD_LOGIC;
             ready_window_in : in STD_LOGIC;
             start_fft_out     : out STD_LOGIC;
             ready_fft_in      : in STD_LOGIC;
             start_mel_out     : out STD_LOGIC;
             ready_mel_in      : in STD_LOGIC;
             start_dct_out     : out STD_LOGIC;
             ready_dct_in      : in STD_LOGIC;
             start_comp_out    : out STD_LOGIC;
             ready_comp_in     : in STD_LOGIC;
             start_debug_out : out STD_LOGIC;
             ready_debug_in    : in STD_LOGIC;
             start_in          : in STD_LOGIC;
             ready_out         : out STD_LOGIC);
end component;

signal rst                          : STD_LOGIC;
signal clk                          : STD_LOGIC:='0';
signal Mem_addr          : STD_LOGIC_VECTOR (13 downto 0):=(others=>'0');
signal Mem_data          : STD_LOGIC_VECTOR (11 downto 0);
```

```
signal  Frame_addr  :  STD_LOGIC_VECTOR  (13  downto  0);
signal start :  STD_LOGIC;
signal ready      :  STD_LOGIC;
signal start_adc            :  STD_LOGIC;
signal ready_adc            :  STD_LOGIC;
signal start_win            :  STD_LOGIC;
signal ready_win            :  STD_LOGIC;
signal spi_data_arr : STD_LOGIC_VECTOR(11 downto 0);
signal spi_arr_cpy : STD_LOGIC_VECTOR(11 downto 0);
signal spi_data : STD_LOGIC;
signal   spi_clk   :   STD_LOGIC;
signal cs_n : STD_LOGIC;
signal data_cnt : STD_LOGIC_VECTOR(5 downto 0):=(others=>'1');
signal data_cnt2            : STD_LOGIC_VECTOR(3 downto 0):=(others=>'0');

type state_type1 is (state0, state1, state2, state3,
state4); signal ADC_state1      : state_type1 := state0;

begin

        DUT1 : Lab_ADC
        PORT MAP     (          reset_in          => rst,
                    clock_in                  => clk,
                    spi_clk_out    => spi_clk,
                    chip_sel_out => cs_n,
                    spi_data_in    => spi_data,
                    addr_in                   => Mem_addr,
```

```
        data_out            => Mem_data,
        start_in            => start_adc,
        ready_out           => ready_adc);

DUT2: Lab_WINDOW
PORT MAP    ( reset_in      => rst,
        clock_in    => clk,
        frame_addr_in => Frame_addr,
        adc_addr_out   => Mem_addr,
        adc_data_in    => Mem_data,
        mem_addr_in    => "000000000",
        mem_data_out   =>  open,
        start_in       => start_win,
        ready_out      => ready_win);

DUT3: Lab_CTRL
PORT MAP    ( reset_in          => rst,
        clock_in        => clk,
        start_adc_out      => start_adc,
        ready_adc_in       => ready_adc,
        frame_addr_out     => Frame_addr,
        start_window_out   => start_win,
        ready_window_in    => ready_win,
        start_fft_out      => open,
        ready_fft_in       => '0',
        start_mel_out      => open,
        ready_mel_in       => '0',
```

```
                start_dct_out       => open,
                ready_dct_in        => '0',
                start_comp_out      => open,
                ready_comp_in       => '0',
                start_debug_out     => open,
                ready_debug_in      => '1',         -- assign ready_debug = 1 to continue windowing
                start_in            => start,
                ready_out           => ready);


-- generate 100 MHz clock
clk <= not clk after 5 ns;


process (cs_n) begin
            if (falling_edge(cs_n)) then
        data_cnt    <= data_cnt + 1;
            end if;
end process;


process (clk) begin
            if (rising_edge(clk)) then
      if (data_cnt < "010000") then
                                spi_data_arr      <= X"111";
              elsif (data_cnt < "100000") then
                                spi_data_arr      <= X"333";
              elsif (data_cnt < "110000") then
                                spi_data_arr      <= X"555";
              else
```

```
                    spi_data_arr      <= X"777";
            end if;
          end if;
end process;

process (spi_clk) begin
          if (falling_edge(spi_clk)) then
                    if (rst = '1') then
                              ADC_state1          <= state0;
                    else
          case ADC_state1 is

            when state0                    => if (cs_n = '0') then
                                      ADC_state1    <= state1;
                                   else
                                      ADC_state1    <= state0;
                                   end if;
                                   spi_data       <= '0';

            when state1                    => ADC_state1     <= state2;

            when state2                    => ADC_state1     <= state3;
                                   spi_arr_cpy    <= spi_data_arr;

            when state3                    => if (data_cnt2 = "1011") then
                                       ADC_state1       <= state4;
                                       data_cnt2 <= (others=>'0');
```

```
                    else
                        ADC_state1      <= state3;
                        data_cnt2       <= data_cnt2 + 1;
                    end if;
                    spi_data            <= spi_arr_cpy(11);
                    spi_arr_cpy(11 downto 1) <= spi_arr_cpy(10 downto 0);

            when state4                 => if (cs_n = '1') then
                    ADC_state1 <= state0;
                    else
                    ADC_state1 <= state4;
                    end if;

            when others                 => ADC_state1 <= state0;

          end case;
        end if;
    end if;
end process;

process
    begin
      rst     <= '1';
      start   <= '0';
      wait for 100 us;

      rst     <= '0';
```

```
        start    <= '0';
        wait for 100 us;

        rst      <= '0';
        start    <= '1';
        wait for 100 us;

        rst      <= '0';
        start    <= '0';
        wait for 1050000 us;
                wait;

end process;

end Test_Beh;



entity test is

end test;

architecture Test_Beh of test is

component Lab_FFT
    Port    ( reset_in      : in STD_LOGIC;
                            clock_in      : in STD_LOGIC;
```

```
        addr_out      : out STD_LOGIC_VECTOR(8 downto 0);
        data_in       : in STD_LOGIC_VECTOR(19 downto 0);
        addr_in       : in STD_LOGIC_VECTOR(7 downto 0);
        data_out      : out STD_LOGIC_VECTOR(31 downto 0);
                start_in      : in STD_LOGIC;
        ready_out     : out STD_LOGIC);
end component;

signal rst                  : STD_LOGIC;
signal clk            : STD_LOGIC:='0';
signal clk_16               : STD_LOGIC:='0';
signal win_addr             : STD_LOGIC_VECTOR (8 downto 0);
signal win_data             : STD_LOGIC_VECTOR (19 downto 0);
signal win_data_d1          : STD_LOGIC_VECTOR (19 downto 0);

type memory_win is array (0 to 511) of std_logic_vector(19 downto 0);
signal win_data_arr : memory_win :=(others=>X"00000");
signal win_arr_ind          : INTEGER range 0 to 511 := 0;

type memory_fft is array (0 to 255) of std_logic_vector(31 downto 0);
signal fft_data_arr : memory_fft :=(others=>X"00000000");
signal fft_arr_ind          : INTEGER range 0 to 255 := 0;

signal RAM_addr             : STD_LOGIC_VECTOR (7 downto 0):=(others=>'0');
signal RAM_data             : STD_LOGIC_VECTOR (31 downto 0);
signal start          : STD_LOGIC;
signal ready           : STD_LOGIC;
```

```
signal start_sampling : STD_LOGIC:='0';
signal data_enable              : STD_LOGIC:='0';
signal read_start             :  STD_LOGIC:='0';

type state_type1 is (initial, addr1, addr2, addr3);
signal FFT_state1          : state_type1 := initial;

begin

        DUT1: Lab_FFT
                PORT MAP     (          reset_in       => rst,
                        clock_in           => clk,
                                                addr_out              => win_addr,
                                                data_in               => win_data_d1,
                                                addr_in               => RAM_addr,
                                                data_out              => RAM_data,
                                                start_in            => start,
                                                ready_out       => ready);


   -- generate 100 MHz clock
   clk    <= not clk after 5 ns;

   -- generate 16 KHz clock
   clk_16          <= not clk_16 after 31250 ns;

   txt_read: process(clk_16)
      file readFile : TEXT open READ_MODE is "C:\Users\ASUS\Desktop\lab_fft1\input.txt";
```

```
        variable myLine : LINE;
        variable val : integer;
        variable ind : integer range 0 to 511 := 0;
        begin
          if rising_edge(clk_16) then
            if start_sampling = '1'
            then
              readline(readFile, myLine);
              read(MyLine,val);
              if (ind /= 512) then
                win_data_arr(ind) <= std_logic_vector(to_signed(val,20)); -- assumed here 20-bit DATA
width.
                ind := ind+1;
              end if;
            end if;
          end if;
    end process;

    -- simulate Simple Dual Port RAM
    win_arr_ind <= to_integer(unsigned(win_addr));
    process (clk) begin
      if (rising_edge(clk)) then
        win_data        <= win_data_arr(win_arr_ind);
        win_data_d1     <= win_data;
      end if;
    end process;

    process(clk)begin
```

```
if rising_edge(clk) then
   case FFT_state1 is

        when initial                    => if (read_start = '1') then
                                           FFT_state1 <= addr1;
                                           RAM_addr   <= (others=>'0');
                                        else
                                           FFT_state1 <= initial;
                                        end if;
                                        fft_arr_ind   <= 0;

        when addr1                      => RAM_addr      <= RAM_addr + 1;
                                        FFT_state1    <= addr2;

        when addr2                      => RAM_addr      <= RAM_addr + 1;
                                        FFT_state1    <= addr3;

        when addr3          => if (RAM_addr = "00000001") then
                                           FFT_state1    <= initial;
                                        else
                                           FFT_state1    <= addr3;
                                        end if;
                                        RAM_addr            <= RAM_addr + 1;
                                        fft_arr_ind         <= fft_arr_ind + 1;
                                        fft_data_arr(fft_arr_ind) <= RAM_data;

        when others                     => FFT_state1          <= initial;
```

```
        end case;
    end if;
end process;

txt_write:process(clk_16)
    file writeFile : TEXT open WRITE_MODE is "C:\Users\ASUS\Desktop\lab_fft1\output.txt";
    variable sample : LINE;
    variable ind : integer range 0 to 255 := 0;
    begin
        if rising_edge(clk_16) then
            if data_enable='1' then
                if (ind /= 256) then
                    write(Sample, to_integer(signed(fft_data_arr(ind))));
                    writeline(writeFile, sample);
                    ind := ind+1;
                end if;
            end if;
        end if;
end process;

process
    begin
        rst                 <= '1';
        start           <= '0';
        start_sampling <= '0';
        read_start                  <= '0';
```

```
wait for 100 us;

rst                 <= '0';
wait for 100 us;

-- read "input.txt" file and write data in window array
start_sampling <= '1';
wait for 32 ms;

start_sampling <= '0';
wait for 100 us;

-- run Lab_FFT with the data inside the window array
start           <= '1';
wait for 10 ns;

start           <= '0';
wait for 100 us;

-- read internal memory of Lab_FFT and write FFT data in FFT array
read_start                  <= '1';
wait for 10 ns;

read_start                  <= '0';
wait for 100 us;

-- read FFT array and write FFT data in "output.txt" file
```

```
        data_enable     <= '1';
        wait for 16 ms;

        data_enable     <= '0';
        wait;

   end process;

end Test_Beh;



entity test3 is

end test3;

architecture Behavioral of test3 is

component Lab_DCT
   Port    ( reset_in      : in STD_LOGIC;
                           clock_in      : in STD_LOGIC;
           mel_addr_out : out STD_LOGIC_VECTOR(3 downto 0);
           mel_data_in : in STD_LOGIC_VECTOR(47 downto 0);
           mem_addr_in : in STD_LOGIC_VECTOR(3 downto 0);
           mem_data_out : out STD_LOGIC_VECTOR(67 downto 0);
                           start_in      : in STD_LOGIC;
           ready_out     : out STD_LOGIC);
```

```
end component;

signal rst                : STD_LOGIC;
signal clk                : STD_LOGIC:='0';
signal RAM_addr           : STD_LOGIC_VECTOR (3 downto 0):=(others=>'0');
signal RAM_data           : STD_LOGIC_VECTOR (67 downto 0);
signal MEL_addr           : STD_LOGIC_VECTOR (3 downto 0);
signal MEL_data           : STD_LOGIC_VECTOR (47 downto 0);
signal MEL_data_d1        : STD_LOGIC_VECTOR (47 downto 0);
signal start        :  STD_LOGIC;
signal ready        : STD_LOGIC;
signal ind                          : INTEGER range 0 to 12 := 0;

type memory_mel_data is array (0 to 12) of std_logic_vector(47 downto 0);
signal mem_mel : memory_mel_data := (X"800000000000", X"080000000000", X"008000000000",
X"000800000000", X"000080000000", X"000008000000",
                                                                    X"000000800000",
X"000000080000", X"000000008000", X"000000000800", X"000000000080", X"000000000008",
X"000000000000");

begin

        DUT1: Lab_DCT
   PORT MAP      ( reset_in      => rst,
                                        clock_in       => clk,
                                        mel_addr_out => MEL_addr,
                                        mel_data_in    => MEL_data,
                                        mem_addr_in    => RAM_addr,
```

```
                                        mem_data_out => RAM_data,
                                        start_in     => start,
                                        ready_out    => ready);

-- generate 100 MHz clock
clk <= not clk after 5 ns;

ind      <= to_integer(unsigned(MEL_addr));

process (clk) begin
        if (rising_edge(clk)) then
            MEL_data_d1    <= mem_mel(ind);
            MEL_data       <= MEL_data_d1;
        end if;
end process;

--MEL_data <= X"800000000000";

process
   begin
     rst        <= '1';
     start      <= '0';
     wait for 100 ns;

     rst        <= '0';
     start      <= '0';
     wait for 100 ns;
```

```
        rst         <= '0';
        start       <= '1';
        wait for 100 ns;

        rst         <= '0';
        start       <= '0';
        wait until ready = '1';

        -- read back the DCT memory
        for I in 0 to 12 loop
            RAM_addr    <= RAM_addr + 1;
            wait for 10 ns;
        end loop;
              wait;

end process;

end Behavioral;



entity test is

end test;

architecture Behavioral of test is
```

```
component Lab_MEL
   Port    ( reset_in      : in STD_LOGIC;
                           clock_in      : in STD_LOGIC;
             fft_addr_out : out STD_LOGIC_VECTOR(7 downto 0);
             fft_data_in : in STD_LOGIC_VECTOR(31 downto 0);
             mem_addr_in : in STD_LOGIC_VECTOR(3 downto 0);
             mem_data_out : out STD_LOGIC_VECTOR(47 downto 0);
                           start_in      : in STD_LOGIC;
             ready_out     : out STD_LOGIC);
end component;

signal rst                  : STD_LOGIC;
signal clk                  : STD_LOGIC:='0';
signal RAM_addr          : STD_LOGIC_VECTOR (3 downto 0):=(others=>'0');
signal RAM_data          : STD_LOGIC_VECTOR (47 downto 0);
signal FFT_data          : STD_LOGIC_VECTOR (31 downto 0);
signal start       :  STD_LOGIC;
signal ready       : STD_LOGIC;

begin

          DUT1 : LAB_mel
          PORT MAP      (           reset_in => rst,
                                          clock_in       => clk,

                                    fft_addr_out => open,
                                    fft_data_in    => FFT_data,
```

```
                mem_addr_in    => RAM_addr,
               mem_data_out => RAM_data,
               start_in       => start,
               ready_out      => ready);

-- generate 100 MHz clock
clk <= not clk after 5 ns;

process
   begin
      rst         <= '1';
      start       <= '0';
      wait for 100 ns;

      rst         <= '0';
      start       <= '0';
      wait for 100 ns;

      rst         <= '0';
      start       <= '1';
      FFT_data    <= X"80000000";
      wait for 100 ns;

      rst         <= '0';
      start       <= '0';
      wait until ready = '1';
```

```
    -- read back the MEL memory
    for I in 0 to 12 loop
        RAM_addr    <= RAM_addr + 1;
        wait for 10 ns;
    end loop;
          wait;

end process;

end Behavioral;




entity test is

end test;

architecture Test_Beh of test is

component Lab_DEBUG
   Port ( reset_in              : in STD_LOGIC;
        clock_in    : in STD_LOGIC;
                    mem_addr_out : out STD_LOGIC_VECTOR (13 downto
                    0); mem_data_in : in STD_LOGIC_VECTOR (31 downto
                    0); txd_out : out STD_LOGIC;
        start_in    : in STD_LOGIC;
```

```
        ready_out    : out STD_LOGIC);
end component;

signal clk      : STD_LOGIC:='0';
signal rst      : STD_LOGIC;
signal start    : STD_LOGIC;
signal ready    :  STD_LOGIC;
signal mem_addr : STD_LOGIC_VECTOR (13 downto 0);
signal mem_data : STD_LOGIC_VECTOR (31 downto 0);
signal mem_data_d1 : STD_LOGIC_VECTOR (31 downto 0);
signal uart_txd : STD_LOGIC;

begin

        Component2 : Lab_DEBUG
        PORT MAP ( reset_in => rst,
                                clock_in       => clk,
                                mem_addr_out => mem_addr,
                                mem_data_in    => mem_data_d1,
                                txd_out        => uart_txd,
                                start_in       => start,
                                ready_out      => ready);

-- generate 100 MHz clock
clk <= not clk after 5 ns;

process
```

```
    begin
      rst                   <= '1';
      start         <= '0';
      wait for 100 ns;

      rst                   <= '0';
      start         <= '0';
      wait for 100 ns;

      rst                   <= '0';
      start         <= '1';
      wait for 100 ns;

      rst                   <= '0';
      start         <= '0';
            wait;
end process;

process (clk) begin
          if (rising_edge(clk)) then
      if (mem_addr(1 downto 0) = "00") then
                            mem_data        <= X"11111111";
            elsif (mem_addr(1 downto 0) = "01") then
                            mem_data        <= X"33333333";
            elsif (mem_addr(1 downto 0) = "10") then
                            mem_data        <= X"55555555";
            else
```

```
                    mem_data        <= X"77777777";
            end if;
            mem_data_d1 <= mem_data;
          end if;
end process;


end Test_Beh;
```