```
-- Message module
-- This module will get the signals from dice component and display the
-- constant message and some changeable message.
library IEEE;
use  IEEE.STD_LOGIC_1164.all;
use  IEEE.STD_LOGIC_ARITH.all;
use  IEEE.STD_LOGIC_UNSIGNED.all;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

entity message is
        PORT(
        clock                           : in STD_LOGIC;
        vclock                          : in STD_LOGIC;
        reset                           : in STD_LOGIC;
        piece1_wins, piece2_wins        : in boolean;
        piece1, piece2                  : in boolean;
        step_size                       : in STD_LOGIC_VECTOR(2 downto 0);
        vga_red, vga_green, vga_blue    : out BOOLEAN;
        countX,countY                   : in STD_LOGIC_VECTOR(9 downto 0)
        );
END message;

architecture dice_message of message is

-- Video Display Signals
-- Signals for Video ROM Memory for Pixel Data
signal rom_address: std_logic_vector(8 Downto 0);
signal rom_data: std_logic_vector(7 Downto 0);
signal rom_mux_output: std_logic;
--------------------------------------------------------------------------------
signal col_address, row_address, con_row, con_col: std_logic_vector(5 Downto 0);
signal pixel_col_count, pixel_row_count: std_logic_vector(5 Downto 0);
--------------------------------------------------------------------------------
signal  col_32_address, row_32_address :std_logic_vector(4 Downto 0);
signal  pixel_32_col_count,pixel_32_row_count: std_logic_vector(6 downto 0);
--------------------------------------------------------------------------------

signal format_address: std_logic_vector(7 downto 0);
signal format_data: std_logic_vector(5 downto 0);
--------------------------------------------------------------------------------

--signal format1_address: std_logic_vector(5 downto 0);
--signal format1_data: std_logic_vector(5 downto 0);

signal final_message: std_logic_vector(1 downto 0);
signal message_relative_position: std_logic_vector(3 downto 0);

signal title_dice,color   : BOOLEAN;
signal Rev_video : std_logic;
signal fresh_clock : std_logic;
signal fresh_counter : std_logic_vector(5 downto 0);


signal red_data,green_data, blue_data: std_logic;
signal in_screen, in_screen_vertical, in_screen_horizontal: boolean;
signal color_index : Integer;

begin

 --Small 8 by 8 Character Genrator ROM for Video Display
 tiny_char_gen_rom: lpm_rom
      GENERIC MAP ( lpm_widthad => 9,
        lpm_numwords => "512",
```

```
        lpm_outdata => "UNREGISTERED",
        lpm_address_control => "UNREGISTERED",
-- Reads in mif file for character generator data
         lpm_file => "char_set.mif",
         lpm_width => 8)
      PORT MAP ( address => rom_address, q => rom_data);
-- Character Format ROM for Video Display
-- Displays constant format character data
-- on left side of Display area
format_rom: lpm_rom
      GENERIC MAP ( lpm_widthad => 8,
        lpm_numwords => "160",
        lpm_outdata => "UNREGISTERED",
        lpm_address_control => "UNREGISTERED",
-- Reads in mif file for data display format
        lpm_file => "dice_con.mif",
         lpm_width => 6)
      PORT MAP ( address => format_address, q => format_data);

-- local R,G,B data
output: process(clock)
BEGIN
IF clock'EVENT and clock = '1' THEN

if color then
red_data   <= rom_mux_output and rev_video and (con_row(0) or con_row(3));
Green_data <= rom_mux_output and rev_video and con_row(1);
Blue_data  <= rom_mux_output and rev_video and con_row(2);
else
red_data   <= rom_mux_output and rev_video;
Green_data <= rom_mux_output and rev_video;
Blue_data  <= rom_mux_output and rev_video;
end if;

if title_dice then
-- Display 8 by 8 font with 32 by 32 pixel array
-- Step through rows for each character pattern

rom_address(2 Downto 0) <= pixel_row_count(4 Downto 2);
-- Mux to pick off correct rom data bit from 8-bit word
-- for on screen character generation
rom_mux_output <= rom_data ( (CONV_INTEGER(NOT pixel_col_count(4 downto 2))));

else

-- Display 8 by 8 font with 16 by 16 pixel array
-- Step through rows for each character pattern

rom_address(2 Downto 0) <= pixel_row_count(3 Downto 1);
-- Mux to pick off correct rom data bit from 8-bit word
-- for on screen character generation
rom_mux_output <= rom_data ( (CONV_INTEGER(NOT pixel_col_count(3 downto 1))));

end if;

-- decide the display area
in_screen  <= in_screen_horizontal and in_screen_vertical;

-- The RGB signal pins to the VGA monitor
vga_Red    <= (red_data ='1') and in_screen;
vga_Green  <= (green_data ='1') and in_screen;
vga_Blue   <= (blue_data ='1') and in_screen;
END IF;
END process output;
```

```
VIDEO_DISPLAY: Process(clock)
Begin
--
-- Generate Video on Screen Signals for Pixel Data
-- Generate row and col address for 16 by 16 font
--

IF clock'EVENT and clock = '1' THEN
        col_address <= countX(9 downto 4);
        row_address <= countY(9 downto 4);
        pixel_col_count <= countX(3 downto 0);
        pixel_row_count <= countY(3 downto 0);

        col_32_address <= countX(9 downto 5);
        row_32_address <= countY(9 downto 5);
        pixel_32_col_count <= countX(4 downto 0);
        pixel_32_row_count <= countY(4 downto 0);

-- if col_address is within 0--39 then, the pixel should be displayed
        IF col_address >= "000000" and col_address <= "100111" THEN
                in_screen_horizontal <= true;
        ELSE
                in_screen_horizontal <= false;
        END IF;
-- if row_address is within 0--29 then, the pixel should be displayed
        IF row_address >= "000000" and row_address <= "011101" THEN
                in_screen_vertical <= true;
        ELSE
                in_screen_vertical <= false;
        END IF;

END IF;

end process VIDEO_DISPLAY;

-- Address for Constant Character Data ROM
format_address(3 Downto 0) <= con_col(3 Downto 0);
format_address(7 Downto 4) <= con_row(3 Downto 0);

VIDEO_DISPLAY_DATA: process(clock)
begin

IF clock'EVENT and clock = '1' THEN
-- Reverse Video for Title at top of screen
        If (row_address >= "00010") and (row_address <= "00100")Then
                rev_video <= '1';
        ELSE
                rev_video <= '0';
        End if;
        IF col_32_address >= "00101" and col_32_address <= "01110" and
                row_32_address = "00001" THEN
                color <= false;
                title_dice <= true;                     -- display the title
                con_col <= col_32_address - "00101";
                con_row <= row_32_address - "00001";
                rom_address(8 downto 3) <= format_data;
        ELSIF col_address >="001100" and col_address <="011011" and
                row_address = "00100" THEN
                                con_col <= col_address - "001100";
                                con_row <= row_address - "00011" +
                                           message_relative_position;
                                rom_address(8 downto 3) <= format_data;
                                title_dice <=false;
```

```
                                color <= true;
                ELSE
                title_dice <=false;
                color <= false;
                rom_address(8 downto 3) <= "100000";
        END IF;
END IF;
end process VIDEO_DISPLAY_DATA;
-- following process decide what final message will be displayed in the
-- given area based the DICE RACE result;

-- in here, I want to use vsync =16ms as the basic fresh clock, every 64*16=1s
-- it will generate a fresh clock;

fresh_clock_gen: process(vclock)
begin
IF vclock'EVENT and vclock = '1' then
        IF fresh_counter = "111111" THEN
                fresh_counter<= "000000";
                fresh_clock <=  '1';
        else
        fresh_counter <= fresh_counter + '1';
        fresh_clock <='0';
        end if;
end if;
end process ;

message_position_gen: process(fresh_clock)
begin
IF reset = '0' THEN
        message_relative_position <= "0000";
ELSIF fresh_clock'EVENT and fresh_clock = '1' then

        IF message_relative_position > "110" THEN
                message_relative_position <= "0000";
        else
                message_relative_position <= message_relative_position +'1';
        end if;
end if;
end process message_position_gen;
end dice_message;
```