Task 2 Split 4 - VHDL Code for the Homework Task 2.

Logic Systems And Processors (České Vysoké Učení Technické v Praze)

```vhdl

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

entity DisplayLogicCurve4parts is
port(
      xyoffset : in  std_logic_vector(10 DOWNTO 0); -- position of parts
      yrow, xcolumn : in unsigned(9 downto 0); -- row and  column number of VGA video
      VGA_CLK : in std_logic;
      VGA_VS : in  std_logic; -- synchronization of position values
      VGA_R, VGA_G, VGA_B: out std_logic_vector(9 downto 0)--  color information
    );
end;

architecture behavioral of DisplayLogicCurve4parts is
-- Intensity of 10bit color in percent
constant C100 : std_logic_vector(9 downto 0) := (others=>'1');
constant C75 : std_logic_vector(9 downto 0) := (9=>'1', 8=>'0', others=>'1');
constant C50 : std_logic_vector(9 downto 0) := (9=>'0', others=>'1');
constant C25 : std_logic_vector(9 downto 0) := (9=>'0', 8=>'0', others=>'1');
constant C0 : std_logic_vector(9 downto 0) := (others=>'0');
constant G588 : std_logic_vector(9 downto 0) := "1001001100";
constant B884 : std_logic_vector(9 downto 0) := "1101110100";
constant R40 : std_logic_vector(9 downto 0) := "0010100000";
constant B22 : std_logic_vector(9 downto 0) := "0001011000";
constant G111 : std_logic_vector(9 downto 0) := "0110111100";
constant R255 : std_logic_vector(9 downto 0) := "1111111111";

-- records are VHDL equivalents of structures
type RGB_type is
   record
      R : std_logic_vector(9 downto 0);
      G : std_logic_vector(9 downto 0);
      B : std_logic_vector(9 downto 0);
   end record;
-- Used colors - we defined them by the way allowing good overlapping
constant BLUE : RGB_type := (C0,C0,C50);
constant GREEN : RGB_type := (C0,C50,C0);
constant RED : RGB_type := (C50,C0,C0);
constant YELLOW : RGB_type := (C75,C75,C0);
constant BLACK : RGB_type := (C0,C0,C0);
constant DARK_BLUE: RGB_type := (R40,B22,G111);
constant NAVY_BLUE: RGB_type := (C0,G588,B884);
constant WHITE: RGB_type := (R255,R255,R255);

constant YSIZE : integer := 240; -- height of flag
constant XSIZE : integer := 320; -- width of flag
constant XY_MAXSIZE : integer := XSIZE;

subtype xyintd is integer range 0 to XY_MAXSIZE;

constant SORGX : integer := 16;
constant SORGY : integer := 15;
constant SROW : integer := 128;
constant SROWCOUNT : integer := 56;


--defintion of array
subtype xyinteger is integer range -1024 to 1024;
type xyPosition_type is array (0 to 7) of xyinteger;
signal xyPosition : xyPosition_type;

component Shark is
   PORT
   (
      address     : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
      clock    : IN STD_LOGIC   := '1';
      q     : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
   );
END COMPONENT;

COMPONENT Shark2Ports IS
   PORT
   (
      address_a      : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
      address_b      : IN STD_LOGIC_VECTOR (12 DOWNTO 0);
      clock    : IN STD_LOGIC   := '1';
      q_a      : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
      q_b      : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
   );
END COMPONENT;

COMPONENT BlackCurve is
PORT(
   xcolumn : in unsigned(9 downto 0);
   q  : out unsigned(6 downto 0)
);
END COMPONENT;


signal shark_address_s  : STD_LOGIC_VECTOR(12 DOWNTO 0);
signal shark_q_s : STD_LOGIC_VECTOR (1 DOWNTO 0);
signal shark_addressY_s  : STD_LOGIC_VECTOR(12 DOWNTO 0);
signal shark_qY_s : STD_LOGIC_VECTOR (1 DOWNTO 0);

-- array of BlackCurve
signal blueCurve_addressB_s, blueCurve_addressY_s : unsigned(9 downto 0);   -- was 9 before
signal blueCurveB_q_s, blueCurveY_q_s    : unsigned(6 downto 0);


begin

Shark2Ports_inst : Shark2Ports
   PORT MAP(
       clock   => VGA_CLK,
       address_a => shark_address_s,
       address_b => shark_addressY_s,
       q_a   => shark_q_s,
       q_b    => shark_qY_s);

   blueCB_inst : BlackCurve
      PORT MAP(xcolumn => blueCurve_addressB_s,
               q       => blueCurveB_q_s);

   blueCY_inst : BlackCurve
      PORT MAP(xcolumn => blueCurve_addressY_s,
               q       => blueCurveY_q_s);

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

    LSPflag : process(xcolumn, yrow, xyPosition, blueCurveB_q_s, blueCurveY_q_s,shark_q_s)
    variable RGB : RGB_type;
    variable x0d, y0d, x1d, y1d, x2d, y2d, x3d, y3d : xyintd;
   variable x0d_outside, y0d_outside, x1d_outside, y1d_outside: boolean;
   variable x2d_outside, y2d_outside, x3d_outside, y3d_outside: boolean;
   variable shark01, shark01Y, shark02, shark02Y, shark03, shark03Y, isNavyBlueInFlag, isWhiteInFlag, isDarkBlueInFlag :boolean;

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

```
      function IsShark(x,y,mode:xyintd) return boolean is
      begin
         case mode is
            when 1 => return x>=SORGX and x<SORGX+SROW
               and y>=SORGY and y<SORGY+SROWCOUNT;
            when 2 => return x>=(SORGX + 150) and x<(SORGX + 150)+SROW
               and y>=(SORGY) and y<SORGY+SROWCOUNT;
            when 3 => return x>=(SORGX + 90) and x<(SORGX + 90)+SROW
                  and y>=(SORGY + 160) and y<(SORGY + 160)+SROWCOUNT;
          when others => return false;
         end case;
      end function;

------------------------------------------------------------------------
      function InCurve(x,y:xyintd;yoffset:unsigned) return boolean is
      begin
         return ((y > to_integer(yoffset) + 117) and (y < to_integer(yoffset)+ 135))
               or ((y > to_integer(yoffset) + 86) and (y < to_integer(yoffset)+ 104));
      end function;

      function InRectangle(x,y:xyintd) return boolean is
      begin
         return (y >= 81) and (y <= 161);
      end function;

------------------------------------------------------------------------
      function CalculatePictureAddress(x,y,pic:xyintd;yoffset:STD_LOGIC_VECTOR) return std_logic_vector is
      variable address:std_logic_vector(12 DOWNTO 0);
      begin
         case pic is
            when 1 => address := std_logic_vector(to_unsigned((y-SORGY)*SROW + (SORGX+SROW-x),yoffset'LENGTH));
            when 2 => address := std_logic_vector(to_unsigned((y-SORGY)*SROW + (SORGX+SROW + 150 - x),yoffset'LENGTH));
            when others => address := std_logic_vector(to_unsigned((y-(SORGY+160))*SROW + (x-(SORGX+90)),yoffset'LENGTH));
            end case;
         return address;
      end function;
------------------------------------------------------------------------
      function CalculateColour(address:STD_LOGIC_VECTOR (1 DOWNTO 0)) return RGB_type is
      variable colour:RGB_type := YElLOW;  -- Yellow Sharks :)
      begin
        if address = "00" then
          colour:= NAVY_BLUE;
        end if;
         return colour;
      end function;


------------------------------------------------------------------------------------------------------------------------------------------------
      procedure diff_limit(
                xy: in unsigned(9 downto 0);
                offset: in xyinteger;
                limit: in natural;
                z: out xyintd;
                outside:out boolean
                ) is
      variable diff:integer;
      begin

           diff:=to_integer(xy)-limit/2-offset;
         if diff>=limit or diff<0 then z:=limit; outside:=true;
         else z:=xyintd(diff); outside:=fal
         end if;
      end procedure diff_limit;

      function "or"(color1, color2 : RGB_type) return RGB_type is
         variable result : RGB_type;
      begin
         if color2/=WHITE then
           result.R := color1.R or color2.R;
           result.G := color1.G or color2.G;
           result.B := color1.B or color2.B;
         else --white color is not in our flag, but this code can be inspiration for you
           result.R := color1.R xor color2.R;
           result.G := color1.G xor color2.G;
           result.B := color1.B xor color2.B;
         end if;
         return result;
      end function;

    begin

      diff_limit(xcolumn, xyPosition(0), XSIZE, x0d, x0d_outside);
      diff_limit(xcolumn, xyPosition(1), XSIZE, x1d, x1d_outside);
      diff_limit(xcolumn, xyPosition(2), XSIZE, x2d, x2d_outside);
      diff_limit(xcolumn, xyPosition(3), XSIZE, x3d, x3d_outside);    -- out paramaters can be omitted

      diff_limit(yrow, xyPosition(4), YSIZE, y0d, y0d_outside);
      diff_limit(yrow, xyPosition(5), YSIZE, y1d, y1d_outside);
      diff_limit(yrow, xyPosition(6), YSIZE, y2d, y2d_outside);
      diff_limit(yrow, xyPosition(7), YSIZE, y3d, y3d_outside);      -- out paramaters can be omitted


      isNavyBlueInFlag  := not x1d_outside and  not y1d_outside;
      isWhiteInFlag := not x0d_outside and  not y0d_outside;
      isDarkBlueInFlag := not x2d_outside and  not y2d_outside;
      -- Drawing Pictures and Cut Out
      shark01:= IsShark(x3d,y3d,1); -- drawing picture
      shark01Y:= IsShark(x1d,y1d,1); -- cut out of picture
      shark02:= isShark(x3d,y3d,2); -- drawing picture
      shark02Y:= isShark(x1d,y1d,2); -- cut out of picture
      shark03:= isShark(x3d,y3d,3); -- drawing picture
      shark03Y:= isShark(x1d,y1d,3); -- cut out of picture


      RGB := BLACK;  -- we initilize RGB color to have always a value


      if isWhiteInFlag and not InCurve(x0d,y0d,blueCurveY_q_s) and InRectangle(x0d,y0d)
      then RGB := RGB or WHITE;
      end if;


      if isNavyBlueInFlag and (shark_qY_s = "01")
                     and (not InRectangle(x1d,y1d) and not InCurve(x1d,y1d,blueCurveB_q_s))
            then RGB := RGB or NAVY_BLUE;
      end if;


      if isDarkBlueInFlag and InCurve(x2d,y2d,blueCurveB_q_s)
        then RGB := RGB or RED;
      end if;

      if (shark01 and shark_q_s /= "01") or (shark02 and shark_q_s /= "01")
          or (shark03 and shark_q_s /= "01")
      then
         RGB := RGB or CalculateColour(shark_q_s);
      end if;

    if isDarkBlueInFlag then
      blueCurve_addressB_s   <= to_unsigned(x2d, blueCurve_addressB_s'LENGTH);

    else
      blueCurve_addressB_s  <= (others=>'0');
    end if;

    if isWhiteInFlag then
      blueCurve_addressY_s   <= to_unsigned(x0d, blueCurve_addressY_s'LENGTH);

    else
      blueCurve_addressY_s  <= (others=>'0');
    end if;

      if shark01 then shark_address_s  <= CalculatePictureAddress(x3d,y3d,1, shark_address_s);
      elsif shark02 then shark_address_s  <= CalculatePictureAddress(x3d,y3d,2, shark_address_s);
      elsif shark03 then shark_address_s  <= CalculatePictureAddress(x3d,y3d,3, shark_address_s);
      else shark_address_s  <=(others=>'0');
      end if;

      if shark01Y then  shark_addressY_s  <= CalculatePictureAddress(x1d,y1d,1, shark_addressY_s);
      elsif shark02Y then shark_addressY_s  <= CalculatePictureAddress(x1d,y1d,2, shark_addressY_s);
      elsif shark03Y then shark_addressY_s  <= CalculatePictureAddress(x1d,y1d,3, shark_addressY_s);
      else shark_addressY_s  <=(others=>'0');
      end if;


      VGA_R<=RGB.R; VGA_G<=RGB.G; VGA_B<=RGB.B;
--------------------------------------------------------------------------------------------------------------------
   end process;

   vs_copy : process(VGA_VS)
   variable dify, minusdify, difx, minusdifx : xyinteger;
   begin
      if rising_edge(VGA_VS) then -- Copy data to assure their stability in picture
         dify := to_integer(signed(xyoffset)); minusdify:=-dify;

         difx := dify+dify/4; minusdifx :=-difx;
         xyPosition(0) <= minusdifx; xyPosition(4) <= minusdify;  -- move green part to the left upper corner
         xyPosition(1) <= minusdifx; xyPosition(5) <= dify;  -- move yellow part to the left bottom corner
         xyPosition(2) <= difx; xyPosition(6) <= dify;  -- move blue part to the right bottom corner
         xyPosition(3) <= difx; xyPosition(7) <= minusdify;  -- move 1/0 picture to the right upper corner
      end if;
   end process;

end architecture behavioral;
```