Embedded Systems Design – Advanced Course

Simple Image Analysis on a BMP file

Group 1:
Adis Bjelosevic
Mehmet Bozkurt
Orhan Mekic

Introduction

This project was carried out as a part of the course Embedded System Design-Advanced Course. The goal of the project was to create a BMP decoder and perform some simple image analysis operations on the displayed image, using the Virtex-II V2MB1000 Development Board as platform. This document describes the system functionality and how the different parts of the system were implemented. The appendix contains a complete source code listing.

System design

Overview

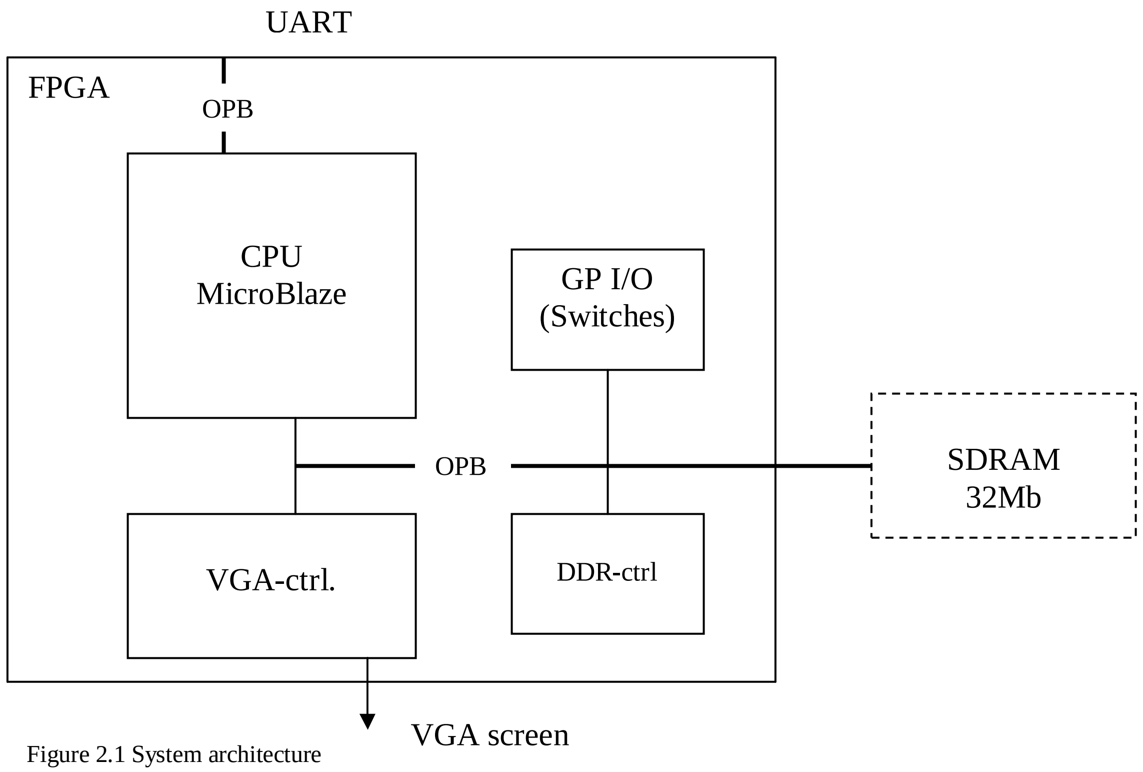


Figure 2.1 System architecture

Above is an image of the system architecture. The heart of the system is the MicroBlaze CPU (100 MHz) which combined with our software performs the main tasks of the system.
One module built inside the FPGA is a 64 kbit block RAM (referred as BRAM in this document), which is mainly used as video RAM (VRAM) and program memory. The system basically reads a bitmap image on the UART and displays it on a VGA-monitor, after it has stored the image in SDRAM. Switches are used to select the mode of operation for the system. The modes are edge detection along the two coordinate axes.

System functionality

Hardware & Peripherals

VGA-controller

Our VGA-controller is based on one supplied to us by our tutors. Originally it was a monochrome 640x480 resolution controller. This controller occupied 640*480=307200 bits on BRAM and provided just two colors. In order for us to create an acceptable solution we had to increase the number of colors used. We decided to go with 16 colors, with a color lookup table, which require 4 bits for each pixel. Because of BRAM limitations and our wish to keep the entire screen content in BRAM we changed the resolution to 320x240. The VGA-controller still occupies 307200 bits in BRAM because 320*240*4=307200 bits.

Using CoreGen we generated a VRAM-block netlist to use with our solution. This reduced the place-and-route time significantly. We also changed the data width from one bit to four bits in our new VRAM core.

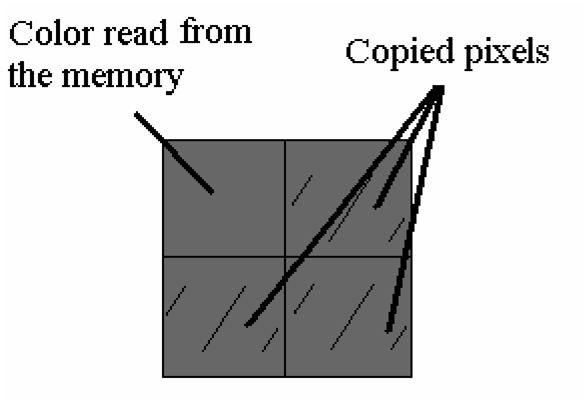
The VGA-controller continuously reads the content of the VRAM and displays it on the screen. What ever is written to VRAM is shown and as long as the content doesn’t change the image shown on the screen does not change. Values between 0 and 15 are stored in the VRAM. These values represent indices to a color lookup table. Values from the lookup table are directly put on the VGA-connectors. Each connector requires 3 bits, 3 for red, 3 for green and 3 for blue. Below is the color lookup table used in our VGA-controller.

```
constant COL_LUT: colorlut_type := (
     "000 000 000", -- 0 black
     "001 000 000", -- 1 dark red
     "000 001 000", -- 2 dark green
     "000 000 001", -- 3 dark blue
     "001 001 000", -- 4
     "001 000 001", -- 5
     "000 001 001", -- 6
     "001 001 001", -- 7
     "011 011 011", -- 8 grey
     "011 000 000", -- 9 red
     "000 011 000", -- 10 green
     "000 000 011", -- 11 blue
     "011 011 000", -- 12
     "011 000 011", -- 13
     "000 011 011", -- 14
     "111 111 111" -- 15 white
);
```

Here is an example for the red color lookup (most significant bits are used) where *do* is a signal (0-15):

```
VGA_red   <= "000" when pixel_en = '0' else
COL_LUT(CONV_INTEGER(do))(0 to 2);
```

Because of the screen size being 640x480 we had to read every pixel, in VRAM, three times to get the desired resolution of 320x240.



DDR-controller

The DDR-controller was provided to us by our tutors. It is needed because the onboard 32 mega bytes SDRAM do not support the 100 MHz clock. This design uses the existing opb_ddr IP and a custom clock module to generate the required clock signals. It produces a 66 MHz clock signals from the 100 MHz signals.

UARTLite

The UART handles communication to and from the board and is connected to the OPB-bus. It communicates at 19200 kbit/s and is set as the standard-in and standard-out. In our design the UART is used for receiving the image file and printing text to the user.

GPIO

The GPIO is connected to the OPB and provides access to the switches available on the board. Its bandwidth is set to 8 bits and the platform generates primitives for reading the switches.

The software

In our solution the software part contains the main logic. The program performs three main tasks before the image is displayed.

First the program receives a bitmap file over the UART; the UART is set as standard-in and standard-out. If the file is not a bitmap the program displays an error message to the user and exits. If the file is a bitmap the program decodes the image and stores it in SDRAM. The SDRAM is accessed through a pointer. The original file is always kept in SDRAM. Because of the bitmap (bmp) file format we implemented one method for reading a whole word at a time in addition to a method that just reads one byte at a time.

When the program reads one word it has to perform some additional bit-shifting because the most significant byte is read first. We also had to implement a method for skipping bytes because all bytes in the bitmap were not important for our purposes. Because of size restrictions we never use functions like `printf` or `getchar` instead we use `xil_printf` and UART functions provided to us by the Xilinx Platform Studio for communication to and from the UART.

Upon receiving the image it is stored upside down in SDRAM. This is due to the way the image is stored in the bmp file. Thus, the program has to reorder the bytes stored in order to display the image correctly on the screen. This is performed in the same memory location as where the image is already stored. Thus the flipping operation does not increase the memory usage in SDRAM.

After the image has been rotated the program enters a loop. In every iteration, the program reads the switches on the board and depending on mode (switch value) it performs one of three different operations. In the first mode the program simply displays the original image. In the second mode the program does edged detection along the x-axis and displays the image. Finally, in the third mode the program does edge detection along the y-axis and displays the image.

Image content is displayed when data is written to the VRAM. The VRAM is accessed through a pointer in the program. The method `memcpy` is used because it instantly copies bytes from any memory location, to VRAM. No delays are noticed and there is never any "painting" seen on the screen, e.g. pixels are not drawn line by line, instead the image is displayed instantly.

The edge detection we use in our solution is based on the differences between individual pixels along one of the two axes. The processed pixels are stored directly after the original image in SDRAM.

Conclusion & Discussion

The following section discusses our projects development and change during its course. This section does not provide a discussion on the implementation itself, because it's pretty straight forward and is easily understood by reading the previous sections in this report and viewing the appendix.

This project can hardly be called a success if one looks at the initial project proposal and project description. The initial proposal was to create a JPEG-decoder in hard/software. We were not able to fulfil our goals due to a number of reasons. First of all when we selected our project we were not aware of the difficulties it would present, and this was because we did not have enough knowledge regarding the technology we wanted to implement and we did not have enough knowledge about the platform. We thought that JPEG-decoding software would be easy to find and fit on the platform, which turned out to be a really challenging task. Other factors that affected our project were all the time we spent on trying to get the most basic parts of the system to work. A few examples of such problems were; the decoder source not fitting in the memory, a VGA-controller not synthesizing for our platform, the file system module provided in the platform did not work as expected despite the fact that we followed the instructions in the documentation. If the FPGA we used got to warm it produced unexpected results. These are just a few examples and although described here very shortly they consumed many working hours. As the weeks went on and we ran in to problems in each step forward, we realised that we had to change our goals and go with a different solution for us not to miss the project deadline. This would of course not be acceptable in a real life situation but instead of quitting and doing nothing we decided to at least try to do something presentable. This change did not mean that all our problems were solved but the ones that we now were faced with were easier to overcome.

All in all we succeeded in creating an application that works and that uses a number of hardware cores together with software. The system performs quite well and no delays are really experienced except when initially transmitting the image to the board due to communication latency. Image analysis and the action of displaying the image are very fast.

Experiences gained from this project are quite a few. We learned that hardware/software co design can be quite challenging and puts higher requirements on work method, problem solving and debugging. From the beginning one needs to have a greater knowledge of the system you intend to implement and the platform it going to run on. Often we ran into unexpected problems and things that ought to work did not. For example, when we added the GPIO for the switch support nothing worked. But after removed our changes and redid them it worked. This taught us not to give up on an idea to quickly. This project also taught us not to put to much faith into development tools and their output.

Appendix

C Source

```
#include "stdio.h"
#include "stdlib.h"
#include "xparameters.h"
#include "xgpio_l.h"
#include "xio.h"
#include "xuartlite_l.h"
#include "xbasic_types.h"

typedef unsigned char  byte;
typedef unsigned short word;
typedef unsigned long  dword; #define NORMAL 0

#define DER_X  1
#define DER_Y  2

void swap(byte *a, byte *b){
  byte tmp = *a;
  *a = *b;
  *b = tmp;
}

void delay(int c)
{
      int b= 10;
      int a=2;
      int i;
      for(i=0;i < c;++i) b+=a;
}

byte readOneByte(void)
{
      return (byte)XUartLite_RecvByte(STDIN_BASEADDRESS);
}

int getWord()
{
      int w = 0;
      int tmp = 0;
      w = readOneByte();
      tmp = readOneByte();
      tmp = tmp << 8;
      w += tmp;
      return w;
}

 void skipBytes(int num)
{
      int i=0;
      for(;i<num;++i) readOneByte();
}

void edgeDetect_x(byte* img, byte* vga,int width, int height, int n)
{
      int i;
      int j;
      int start = n;
      int diff;
      for(i=0;i<height;i++)
            for( j=0;j<width;j++) {
                  diff = img[j+320*i] - img[j+1+320*i];
                  img[n++]=diff;
            }
      memcpy(vga, &img[start], n-start);
}

void edgeDetect_y(byte* img, byte* vga,int width, int height, int n)
{
      int i;
      int j;
      int start = n;
      int diff;
      for(i=0;i<height-1;i++)
            for( j=0;j<width;j++) {
```

```
                        diff = img[j+320*i] - img[j+320*(i+1)];
                        img[n++]=diff;
                  }
            memcpy(vga, &img[start], n-start);
}

 int _wait(loop_count)
int loop_count;
{
      int sum, data;
      sum = 0;
      for (data = 0; data < loop_count; data++) {
            sum = sum + data;
      }
}

int main()
{
      byte t;
      int i=0;
      int j=0;
      int loop = 1;
      int sleepCount = 0;
      byte b;
      byte* data;
      int width  = 0;
      int height = 0;
      int num_colors = 0;
      int pixOffset =0 ;
      int n =0;
      int header_size;
      int pixel_size;
      int file_size;
      unsigned char* vga_ctrl = (unsigned
char*)XPAR_MYVGACTRL_BASEADDR;
      unsigned char* sdram = (unsigned char*)XPAR_DDRCTRL_BASEADDR;
      XGpio_mSetDataDirection(XPAR_SWITCHES_BASEADDR, 0xFF);

      xil_printf("Running\n\r");
      if( (char)readOneByte() !='B' ||  (char)readOneByte() !='M')  //
0 1
      {
            xil_printf("Not a bitmap file.\n\r");
            exit(1);
      }

      file_size = getWord(); // 2 3

      skipBytes(6); // 4 5 6 7 8 9
      pixOffset = getWord(); // 10 11
      skipBytes(2);     // 12 13
      header_size = getWord(); // 14 15
      skipBytes(2);     // 16 17
      width = getWord(); //18 19
      skipBytes(2);     // 20 21
      height= getWord(); // 22 23
      skipBytes(4);      // 24 25 26 27
      num_colors = readOneByte();  // 28
      skipBytes(5);  //29 30 31 32 33
      pixel_size = getWord(); // 34 35
      skipBytes(50);    xil_printf("file size: %d\n\r", file_size);
      xil_printf("offset: %d\n\r",  pixOffset);
      xil_printf("header: %d\n\r", header_size);
      xil_printf("pixel data size: %d\n\r", pixel_size);
      for(i=0;i<height;i++) {
            for( j=0;j<width;j++) {
                  b = readOneByte();
                  sdram[n++] = b >> 4;
                  j++;
                  sdram[n++] = b;
            }
      }

      xil_printf("Displaying...\n\r");
      for(i=0;i<height/2;++i)
            for( j=0;j<width;++j)
                  swap( &sdram[j+i*320], &sdram[j+320*(height-1-i)]);

      while(1){
            t = (XGpio_mGetDataReg(XPAR_SWITCHES_BASEADDR));
            t = (t >> 6);
            if( t == NORMAL ){
                  memcpy( vga_ctrl, sdram, n);
                  xil_printf("Normal display, mode %d\n\r", t);
            }
            if( t == DER_X ){
                  edgeDetect_x(sdram, vga_ctrl,width, height,n);

                  xil_printf("Edge detect x, mode %d\n\r", t);
            }
            if( t == DER_Y ){
                  edgeDetect_y(sdram, vga_ctrl,width, height,n);

                  xil_printf("Edge detect y, mode %d\n\r", t);
            }
            readOneByte();
      }
}
```

VGA-contoller (VHDL)

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
----------------------------------------------------------------------
--------
-- entity
----------------------------------------------------------------------
--------
```

```
entity OPB_VGACtrl is
  generic
  (
    C_BASEADDR         : std_logic_vector(0 to 31) := X"FFFFFFFF";
    C_HIGHADDR         : std_logic_vector(0 to 31) := X"00000000"
  );
  port
  (
    --Required OPB bus ports, do not add to or delete
    OPB_ABus     : in  std_logic_vector(0 to 31);
    OPB_BE       : in  std_logic_vector(0 to 3);
    OPB_Clk      : in  std_logic;
    OPB_DBus     : in  std_logic_vector(0 to 31);
    OPB_RNW      : in  std_logic;
    OPB_Rst      : in  std_logic;
    OPB_select   : in  std_logic;
    OPB_seqAddr  : in  std_logic;
    Sln_DBus     : out std_logic_vector(0 to 31);
    Sln_errAck   : out std_logic;
    Sln_retry    : out std_logic;
    Sln_toutSup  : out std_logic;
    Sln_xferAck  : out std_logic;
    -- VGA stuff
    VGA_hsync     : out std_logic;
    VGA_vsync     : out std_logic;
    VGA_red    : out std_logic_vector(0 to 2);
    VGA_green     : out std_logic_vector(0 to 2);
    VGA_blue      : out std_logic_vector(0 to 2)

 );
end entity OPB_VGACtrl; --USER-- change entity name

-----------------------------------------------------------------------
--------
-- architecture
-----------------------------------------------------------------------
--------

architecture imp of OPB_VGACtrl is --USER-- change entity name
  -- the vga clk should be 25.175 MHz, but if we divide OPB_Clk by 4 we
get
  -- 25 MHz. This will do.
  --   One line
  --   8 pixels front porch
  --  96 pixels horizontal sync
  --  40 pixels back porch
  --   8 pixels left border
  -- 640 pixels video
  --   8 pixels right border
  -- ---
  -- 800 pixels total per line

  -- One field(frame)
  --   2 lines front porch
  --   2 lines vertical sync
  --  25 lines back porch
  --   8 lines top border
  -- 480 lines video
  --   8 lines bottom border
  -- ---
  -- 525 lines total per field
-----------------------------------------------------------------------
--------
-- timing Constants
-----------------------------------------------------------------------
--------

-- Horizontal timing constants
constant H_FRONTPORCH:   integer := 8;
constant H_SYNC:         integer := 96;
constant H_BACKPORCH:    integer := 40;
constant H_LBORDER:      integer := 8;
constant H_PIXEL:        integer := 640;
constant H_RBORDER:      integer := 8;
constant H_PERIOD:       integer := 800;
-- Vertical timing constants
constant V_FRONTPORCH:   integer := 2;
constant V_SYNC:         integer := 2;
constant V_BACKPORCH:    integer := 25;
constant V_TBORDER:      integer := 8;
constant V_PIXEL:        integer := 480;
constant V_BBORDER:      integer := 8;
constant V_PERIOD:       integer := 525;
-- Horizontal timing positions
constant H_FRONTPORCH_START:   integer := 0;
--   0
constant H_SYNC_START:         integer := H_FRONTPORCH_START +
H_FRONTPORCH;   --   8
constant H_BACKPORCH_START:    integer := H_SYNC_START + H_SYNC;
-- 104
constant H_LBORDER_START:      integer := H_BACKPORCH_START +
H_BACKPORCH;     -- 144
constant H_PIXEL_START:        integer := H_LBORDER_START + H_LBORDER;
-- 152
constant H_RBORDER_START:      integer := H_PIXEL_START + H_PIXEL;
-- 792
-- Vertical timing positions
constant V_FRONTPORCH_START:   integer := 0;
--   0
constant V_SYNC_START:         integer := V_FRONTPORCH_START +
V_FRONTPORCH;     --   2
constant V_BACKPORCH_START:    integer := V_SYNC_START + V_SYNC;
--   4
constant V_TBORDER_START:      integer := V_BACKPORCH_START +
V_BACKPORCH;       --  29
constant V_PIXEL_START:        integer := V_TBORDER_START + V_TBORDER;
--  37
constant V_BBORDER_START:      integer := V_PIXEL_START + V_PIXEL;
-- 517

------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component vram
      port (
      addra: IN std_logic_VECTOR(16 downto 0);
      addrb: IN std_logic_VECTOR(16 downto 0);
      clka: IN std_logic;
      clkb: IN std_logic;
      dinb: IN std_logic_VECTOR(3 downto 0);
      douta: OUT std_logic_VECTOR(3 downto 0);
      web: IN std_logic);
end component;

-- XST black box declaration
attribute box_type : string;
attribute box_type of vram: component is "black_box";
----------------------------------------------------------------------
type colorlut_type is array (0 to 15) of std_logic_vector(0 to 8);

constant COL_LUT: colorlut_type := (
      "000000000", -- 0 black
      "001000000", -- 1 dark red
      "000001000", -- 2 dark green
      "000000001", -- 3 dark blue
        "001001000", -- 4
      "001000001", -- 5
      "000001001", -- 6
      "001001001", -- 7
      "011011011", -- 8 grey
      "011000000", -- 9 red
      "000011000", -- 10 green
      "000000011", -- 11 blue
      "011011000", -- 12
      "011000011", -- 13
      "000011011", -- 14
      "111111111" -- 15 white
);

  signal cnt : std_logic_vector(0 to 1);  -- divide OPB_Clk by 4
  -- out signals from the flip-floops
  signal x_reg : std_logic_vector(0 to 9);  -- x position, 0..799
  signal y_reg : std_logic_vector(0 to 8);  -- y position, 0..524
  -- in signals to the flip-floops
  signal x_next : std_logic_vector(0 to 9);  -- x position, 0..799
  signal y_next : std_logic_vector(0 to 8);  -- y position, 0..524
  -- pixel_clk is high the opb_clk befor a new pixel is moved to the
vga output
  -- it may not be used as a clock (would introduce logic in the clk
path, i.e.
  -- bad design!)
  -- pixel_clk works as an enable for all registers
  signal pixel_clk : std_logic;
  signal h_pixel_en : std_logic;    -- 152 <= x < 792
  signal v_pixel_en : std_logic;    --  37 <= y < 517
  signal pixel_en : std_logic;
  signal pixel_en_reg : std_logic;

  signal we : std_logic;
  signal di : std_logic_VECTOR(3 downto 0);
  signal do : std_logic_VECTOR(3 downto 0);
  signal r_addr : std_logic_vector(16 downto 0);
  signal w_addr : std_logic_vector(16 downto 0);
  signal cs : std_logic;

  signal hold: std_logic;
  signal new_row: std_logic;
begin
------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
VGA_memory : vram
            port map (
                  addra => r_addr,
                  addrb => w_addr,
                  clka => OPB_clk,
                  clkb => OPB_clk,
                  dinb => di,
                  douta => do,
                  web => we);
-- INST_TAG_END ------ End INSTANTIATION Template ------------

  pixel_clk <= '1' when cnt = "00" else '0';
  h_pixel_en <= '1' when x_next >= H_PIXEL_START and x_next <
H_RBORDER_START else '0';
  v_pixel_en <= '1' when y_next >= V_PIXEL_START and y_next <
V_BBORDER_START else '0';
  pixel_en <= h_pixel_en and v_pixel_en;


  -- update pixel_en, cnt, x and y registers
  process (OPB_Clk, OPB_Rst)
  begin  -- process
    if OPB_Rst = '1' then
      x_reg <= (others => '0');
      y_reg <= (others => '0');
      cnt <= (others => '0');
      --hold <= '0';
      --new_row <= '0';
    elsif OPB_Clk'event and OPB_Clk = '1' then
            cnt <= cnt + 1;
      if pixel_clk = '1' then
        pixel_en_reg <= pixel_en;
        x_reg <= x_next;
        y_reg <= y_next;
      end if;
    end if;
  end process;

  -- produce x_next and y_next
  process (x_reg, y_reg)
    variable x : std_logic_vector(0 to 9);  -- x position, 0..799
    variable y : std_logic_vector(0 to 8);  -- y position, 0..524
  begin
    x := x_reg;
    y := y_reg;
    x := x + 1;
    if x = H_PERIOD then
      x := "0000000000";
      y := y + 1;
      if y = V_PERIOD then
```

```
        y := "000000000";
      end if;
    end if;
    x_next <= x;
    y_next <= y;
  end process;
-----------------------------------------------------------------------
------ sync signals ---
  -- hsync
  process (OPB_Clk)
  begin  -- process
    if OPB_Clk'event and OPB_Clk = '1' then
    -- if OPB_Rst = '1' then
     --     vga_hsync <= '0';
    -- else
      if x_next >= H_SYNC_START and x_next < H_BACKPORCH_START then
        vga_hsync <= '0';
      else
        vga_hsync <= '1';
      end if;
    -- end if;
    end if;
  end process;

  -- vsync
  process (OPB_Clk)
  begin  -- process
    if OPB_Clk'event and OPB_Clk = '1' then
    -- if OPB_Rst = '1' then
      --vga_vsync <= '0';
    -- else
       if y_next >= V_SYNC_START and y_next < V_BACKPORCH_START then
        vga_vsync <= '0';
      else
        vga_vsync <= '1';
      end if;
    --end if;
    end if;
  end process;

-----------------------------------------------------------------------
- pixel output ---
VGA_red   <= "000" when pixel_en = '0' else COL_LUT(CONV_INTEGER(do))(0
to 2);
VGA_green <= "000" when pixel_en = '0' else COL_LUT(CONV_INTEGER(do))(3
to 5);
VGA_blue  <= "000" when pixel_en = '0' else COL_LUT(CONV_INTEGER(do))(6
to 8);


  process (OPB_Clk)
  begin  -- process
    if OPB_Clk'event and OPB_Clk = '1' then  -- rising clock edge
      if OPB_Rst = '1' then
        hold <= '0';
        r_addr <= (others => '0');
        new_row <= '0';
      else
            if pixel_clk = '1' then
                  if (x_next = "0000000000" and y_next(8) = '1') then
                        r_addr <= r_addr - "101000000";
                  end if;
                        if  pixel_en ='1' then
                              if hold = '1' then
                              r_addr <= r_addr + 1;
                              hold <= '0';
                        else
                              hold <= '1';
                        end if;
                        --if x_next = H_RBORDER_START - 1 then
                              --if new_row = '0' then
                                    --r_addr <= r_addr - "101000000";
                                    --new_row <= '1';
                                    --hold <= '0';
                              --else
                                    --new_row <= '0';
                              --end if;
                        --end if;

                  elsif v_pixel_en = '0' then
                        r_addr <= (others => '0');
                        --new_row <= '0';
                        --hold <= '0';
                        end if;
                  end if;
      end if;
    end if;
  end process;

-----------------------------------------------------------------------
---- OPB slave ---
cs <= '1' when OPB_ABus >= C_BASEADDR and OPB_ABus <= C_HIGHADDR and
OPB_select = '1' else '0';
we <= '1' when cs = '1' and OPB_RNW = '0' else '0';
di <= OPB_DBus(28 to 31);
w_addr <= OPB_ABus(15 to 31);

  Sln_DBus     <= (others => '0');
  Sln_errAck   <= '0';
  Sln_retry    <= '0';
  Sln_toutSup  <= '0';

  process (OPB_Clk)
  begin  -- process
    if OPB_Clk'event and OPB_Clk = '1' then  -- rising clock edge
          if we = '1' then
               Sln_xferAck <= '1';
       else
               Sln_xferAck <= '0';
             end if;
       end if;
  end process;
end architecture imp;
```