CpE 487 Digital Design Lab

Lab 3: VGA Bouncing Ball I

1. Introduction

In this lab, we will program the FPGA on the Nexys2 board to display a “bouncing ball” on a 640 x 480 VGA monitor connected to the VGA interface on the board. This will requires us to generate the required sync and video signals consistent with the VGA standard.

2. VGA Display

Video Graphics Array (VGA) is a standard that was originally developed for driving CRT displays from a PC. The original image format was 640x480 RGB color. Over the years, VGA has been extended to accommodate much higher resolution displays. In the lab, we will limit ourselves to the 640x480 format.

The VGA protocol was designed to drive a cathode ray tube (CRT) display in which an electron beam is raster scanned across the screen as shown in Figure 1. Each video frame, the beam is scanned across the screen 480 times to create 480 lines of displayable information. We divide, in turn, each horizontal line into 640 pixels of displayable information. Each pixel is also defined by a red, green and blue intensity which defines the brightness and the color of that pixel. The display runs at a frame rate of 60 complete frames per second. This is fast enough for your eye to see a continuous (rather than a flickering) image.

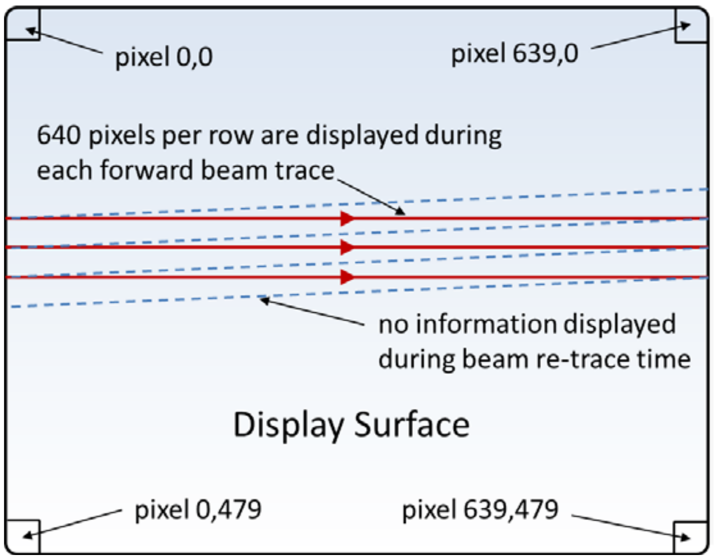

pixel 0,0
pixel 639,0
640 pixels per row are displayed during each forward beam trace
no information displayed during beam re-trace time
Display Surface
pixel 0,479
pixel 639,479

Figure 1 VGA Raster Scan Format

The PC (or FPGA in our case) generates horizontal and vertical synchronization signals to control the raster scanning of the display. A horizontal (HSYNC) pulse triggers the horizontal scanning of the next line. A VSYNC pulse brings the beam back up to the top of the display (row 0) to begin a new frame. In addition to the sync pulses, the controller must supply red, green and blue video signals that describe the intensity of the current pixel. These are analog signals that range between 0V and 0.7V. These are generated on the Nexys2 board using an 8-bit video signal and a simple resistor based D/A converter as shown in Figure 2. The resistor values are chosen to work in conjunction with the 75Ω termination resistance of the VGA display. The 8-bit video signal includes 3-bits of red and green intensity and 2-bits of blue intensity (your eye is less sensitive to small changes in blue levels).

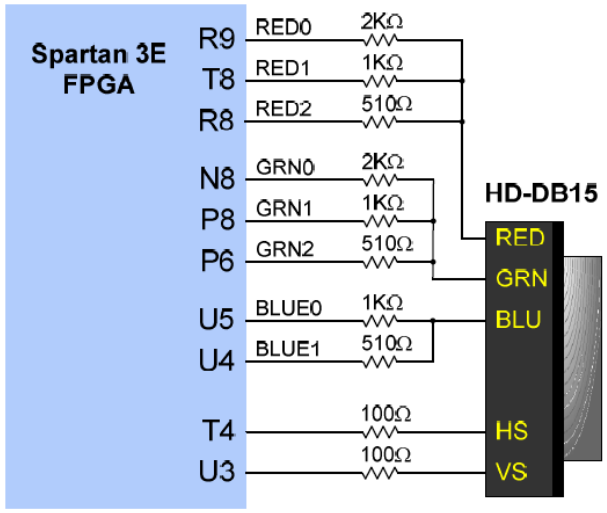
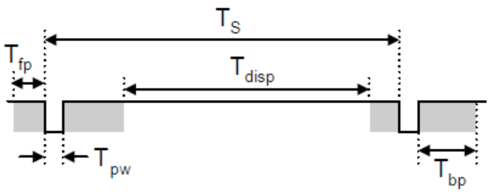


Figure 2 Nexys2 VGA Interface

System timings for 640x480 60 Hz. operation are shown in Figure 3. The waveform describes both the vertical and horizontal sync signals. Note that the horizontal (line) period contains display time for the 640 pixels T_{disp}, the HSYNC pulse T_{pw} and two blanking periods T_{fp} and T_{bp} which allow time for the beam retrace. Similarly the vertical (frame) period contains time for the 480 lines, the VSYNC pulse and extra time for the vertical retrace. The time periods shown in terms of *Clks* assume a 25 MHz clock.



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 3 VGA System Timings

3. Hardware Setup

Connect the VGA display to the VGA port on the Nexys2 board as shown in Figure 4.

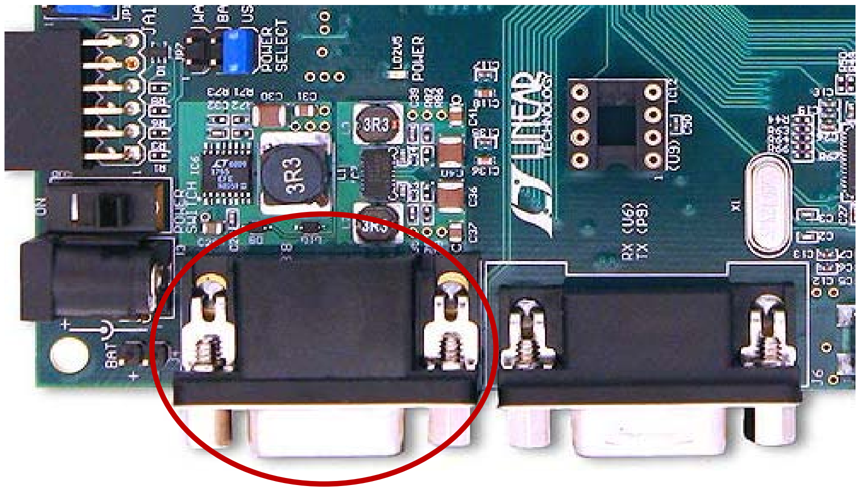


Figure 4   VGA Port

4. Configuring FPGA

4.1 Create a New Project

Use the Xilinx ISE software to create a new project named *VGAball* using the same *project settings* as in Labs 1 and 2.

4.2 Add Source for "vga_sync"

Create a new VHDL source module called *vga_sync*. This module will be used to generate the horizontal and vertical sync waveforms and also the pixel row and column addressing. Enter the following code into the *vga_sync.vhd* edit window:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vga_sync is
   Port ( clock_25MHz : in  STD_LOGIC;
          red : in  STD_LOGIC;
          green : in  STD_LOGIC;
          blue : in  STD_LOGIC;
          red_out : out  STD_LOGIC;
          green_out : out  STD_LOGIC;
          blue_out : out  STD_LOGIC;
          hsync : out  STD_LOGIC;
          vsync : out  STD_LOGIC;
          pixel_row : out  STD_LOGIC_VECTOR (9 downto 0);
          pixel_col : out  STD_LOGIC_VECTOR (9 downto 0));
end vga_sync;

architecture Behavioral of vga_sync is
signal h_cnt, v_cnt: STD_LOGIC_VECTOR (9 DOWNTO 0);
begin
sync_pr:          process
variable video_on: STD_LOGIC;
          begin
                    wait until rising_edge(clock_25MHz);

                    -- Generate Horizontal Timing Signals for Video Signal
                    -- h_cnt counts pixels across line (800 total = 640 active + extras for sync and blanking)
                    -- Active picture for 0 <= h_cnt <= 639
                    -- Hsync for 659 <= h_cnt <= 755
                    if h_cnt >= 799 then
                              h_cnt <= "0000000000"; else
                              h_cnt <= h_cnt+1;
                    end if;
                    if (h_cnt >= 659) and (h_cnt <= 755) then
                              hsync <= '0'; else
                              hsync <= '1';
                    end if;

                    -- Generate Vertical Timing Signals for Video Signal
                    -- v_cnt counts lines down screen (525 total = 480 active + extras for sync and blanking)
                    -- Active picture for 0 <= v_cnt <= 479
                    -- Vsync for 493 <= h_cnt <= 494
                    if (v_cnt >= 524) and (h_cnt = 699) then
                              v_cnt <= "0000000000";
                    elsif h_cnt = 699 then
                              v_cnt <= v_cnt+1;
                    end if;
                    if (v_cnt >= 493) and (v_cnt <= 494) then
                              vsync <= '0'; else
                              vsync <= '1';
                    end if;

                    -- Generate Video Signals and Pixel Address
                    if (h_cnt <= 639) and (v_cnt <= 479) then
                              video_on := '1'; else
                              video_on := '0';
                    end if;
                    pixel_col <= h_cnt;
                    pixel_row <= v_cnt;

                    -- Register video to clock edge and suppress video during blanking and sync periods
                    red_out <= red and video_on;
```

```
                green_out <= green and video_on;
                blue_out <= blue and video_on;
        end process;
end Behavioral;
```

Expand the **Synthesize** command in the *Process* window and run **Check Syntax** to verify that you have entered the code correctly.

This module uses a 25MHz clock to drive horizontal and vertical counters *h_cnt* and *v_cnt* respectively. These counters are then used to generate the various timing signals. *vsync* and *hsync* are the vertical and horizontal sync waveforms that will go directly to the VGA display. *pixel_col* and *pixel_row* are the column and row address of the current pixel being displayed. This module also takes as input the current red, blue and video data and gates it with a signal called *video_on*. This ensures that no video is sent to the display during the sync and blanking periods. Note that red, green and blue video are each represented as 1-bit (on-off) quantities. This is sufficient resolution for our application.

4.3 Add Source for "ball"

Create a new VHDL source module called *ball*. This module will be used to generate the red, green and blue video that will paint the ball on to the VGA display at its current position. Enter the following code into the *ball.vhd* edit window:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ball is
   Port ( v_sync : in  STD_LOGIC;
          pixel_row : in  STD_LOGIC_VECTOR(9 downto 0);
          pixel_col : in  STD_LOGIC_VECTOR(9 downto 0);
          red : out  STD_LOGIC;
          green : out  STD_LOGIC;
          blue : out  STD_LOGIC);
end ball;

architecture Behavioral of ball is

constant size: integer:=8;
signal ball_on: STD_LOGIC;          -- indicates whether ball is over current pixel position

-- current ball position - intitialized to center of screen
signal ball_x: STD_LOGIC_VECTOR(9 downto 0):= CONV_STD_LOGIC_VECTOR(320,10);
signal ball_y: STD_LOGIC_VECTOR(9 downto 0):= CONV_STD_LOGIC_VECTOR(240,10);
```

```
-- current ball motion - initialized to +4 pixels/frame
signal ball_y_motion: STD_LOGIC_VECTOR(9 downto 0):= "0000000100";

begin
red <= '1';                          -- color setup for red ball on white background
green <= not ball_on;
blue <= not ball_on;

-- process to draw ball current pixel address is covered by ball position
bdraw: process (ball_x, ball_y, pixel_row, pixel_col) is
        begin
                if (pixel_col >= ball_x - size) and
                        (pixel_col <= ball_x + size) and
                        (pixel_row >= ball_y - size) and
                        (pixel_row <= ball_y + size) then
                        ball_on <= '1'; else
                        ball_on <= '0';
                end if;
        end process;

-- process to move ball once every frame (i.e. once every vsync pulse)
mball: process
        begin
                wait until rising_edge(v_sync);
                -- allow for bounce off top or bottom of screen
                if ball_y + size >= 480 then
                        ball_y_motion <= "1111111100";     --  -4 pixels
                elsif ball_y <= size then
                        ball_y_motion <= "0000000100";     --  +4 pixels
                end if;
                ball_y <= ball_y + ball_y_motion; -- compute next ball position
        end process;
end Behavioral;
```

Highlight the *ball* module in the *Hierarchy* window, expand the **Design Utilities** command in the *Process* window and run **Check Syntax** to verify that you have entered the code correctly

This module maintains signals *ball_x* and *ball_y* which represent the current position of the ball on the screen. These are initialized to (320,240) to start the ball in the center of the screen. The module also maintains a signal *ball_y_motion* that represents the number of pixels that the ball should move in one frame period. This is initialized to +4 pixels/frame. The module generates one-bit red, green and blue video signals which are normally all set to ‘1’. This produces a white screen background. When the signal *ball_on* is set, the green and blue signals go to ‘0’ which makes those pixels red.

The module takes as input the current pixel row and column address which is generated by the *vga_sync* module. Whenever the ball position is within 8 pixels of the current pixel address (in both x and y directions), the process *bdraw* sets the signal *ball_on*. This paints a red ball around the current pixel address.

A second process *mball* (activated by the *vsync* signal) updates the ball position once every frame. When the ball reaches the top of the screen, it changes the ball motion to -4 pixels per frame. When it reaches the bottom of the screen it changes the ball motion to +4 pixels per frame.

4.4 Add Source for "vga_top"

Create a new VHDL source module called *vga_top*. This module will connect the *vga_sync* and *ball* modules together and connect the appropriate signals to the Nexys2 VGA port. Enter the following code into the *ball.vhd* edit window:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vga_top is
   Port ( clk_50MHz : in  STD_LOGIC;
          vga_red : out  STD_LOGIC_VECTOR (2 downto 0);
          vga_green : out  STD_LOGIC_VECTOR (2 downto 0);
          vga_blue : out  STD_LOGIC_VECTOR (1 downto 0);
          vga_hsync : out  STD_LOGIC;
          vga_vsync : out  STD_LOGIC);
end vga_top;

architecture Behavioral of vga_top is

signal ck_25: STD_LOGIC;

-- internal signals to connect modules
signal S_red, S_green, S_blue: STD_LOGIC;
signal S_vsync: STD_LOGIC;
signal S_pixel_row, S_pixel_col: STD_LOGIC_VECTOR (9 downto 0);

component ball is
   Port ( v_sync : in  STD_LOGIC;
          pixel_row : in  STD_LOGIC_VECTOR(9 downto 0);
          pixel_col : in  STD_LOGIC_VECTOR(9 downto 0);
          red : out  STD_LOGIC;
          green : out  STD_LOGIC;
          blue : out  STD_LOGIC);
end component;
```

```
component vga_sync is
          Port (  clock_25MHz : in  STD_LOGIC;
         red : in  STD_LOGIC;
         green : in  STD_LOGIC;
         blue : in  STD_LOGIC;
         red_out : out  STD_LOGIC;
         green_out : out  STD_LOGIC;
         blue_out : out  STD_LOGIC;
         hsync : out  STD_LOGIC;
         vsync : out  STD_LOGIC;
         pixel_row : out  STD_LOGIC_VECTOR (9 downto 0);
         pixel_col : out  STD_LOGIC_VECTOR (9 downto 0));
end component;

begin

--  Process to generate 25 MHz clock from 50 MHz system clock
ckp:      process
          begin
                    wait until rising_edge(clk_50MHz);
                    ck_25 <= not ck_25;
          end process;

          -- vga_driver only drives MSB of red, green & blue
          -- so set other bits to zero
          vga_red(1 downto 0) <= "00";
          vga_green(1 downto 0) <= "00";
          vga_blue(0) <= '0';

add_ball: ball port map(                  --instantiate ball component
                    v_sync => S_vsync,
                    pixel_row => S_pixel_row,
                    pixel_col => S_pixel_col,
                    red => S_red,
                    green=> S_green,
                    blue => S_blue);

vga_driver: vga_sync port map(            --instantiate vga_sync component
                    clock_25MHz => ck_25,
                    red => S_red,
                    green => S_green,
                    blue => S_blue,
                    red_out => vga_red(2),
                    green_out => vga_green(2),
                    blue_out => vga_blue(1),
                    pixel_row => S_pixel_row,
                    pixel_col => S_pixel_col,
                    hsync => vga_hsync,
                    vsync => S_vsync);

          vga_vsync <= S_vsync;                    --connect output vsync

end Behavioral;
```

Right click on the module *vga_top* in the *Hierarchy* window and select **Set as Top Module**.

4.5 Synthesis and Implementation

Highlight the *vga_top* module in the *Hierarchy* window and execute the **Synthesis** command in the *Process* window.

Add an Implementation Constraint source file *vga_top.ucf* and enter the following data into the edit window:

```
NET     "clk_50MHz"   LOC = B8;

NET     "vga_hsync"     LOC = T4;
NET     "vga_vsync"     LOC = U3;

NET     "vga_red[0]"    LOC = R9;
NET     "vga_red[1]"    LOC = T8;
NET     "vga_red[2]"     LOC = R8;
NET     "vga_green[0]"  LOC = N8;
NET     "vga_green[1]"  LOC = P8;
NET     "vga_green[2]"   LOC = P6;
NET     "vga_blue[0]"    LOC = U5;
NET     "vga_blue[1]"    LOC = U4;

NET "ck_25" TNM_NET = ck_25_net;
TIMESPEC TS_ck_25 = PERIOD "ck_25_net" 40 ns HIGH 50%;
```

You will notice that in addition to specifying I/O pin numbers, the *vga_top.ucf* also specifies a timing constraint. As part of the module *vga_top*, we generate a 25 MHz clock named *ck_25*. This signal is used to clock many flip-flops in the module *vga_sync*. The high degree of loading on this signal raises the possibility that signal delays and clock skew may conspire to cause some of these flip-flops to not satisfy required setup and hold times. The *TNM_NET* command identifies all the registers clocked by *ck_25*. The timing constraint *TS_ck_25* tells the synthesis program that these circuits need to run with a clock period of 40 ns and a 50% duty cycle. The synthesis software will use this information to ensure that all setup and hold times are met on this clock network at 25 MHz.

Now highlight the *vga_top* module in the Hierarchy window and run **Implement Design** followed by **Generate Programming File**.

4.6 Download and Run

Use the *Adept* software to download your configuration file *vga_top.bit* and check out the result.

4.7 Now let's make some changes …

Modify the VHDL code in the module ball to achieve the following:

1. Change the size and/or color of the ball and run your code.

2. Change the square ball to a round ball

3. Introduce a new signal *ball_x_motion* to allow the ball to move horizontally as well as vertically and add code so that it will also bounce of the left and right side walls.