

City University of Hong Kong
Department of Electronic Engineering

EE 2000 – Lab Manual 5

Logic design of a simple VGA controller

Course Leader: Dr. Ray C.C. CHEUNG

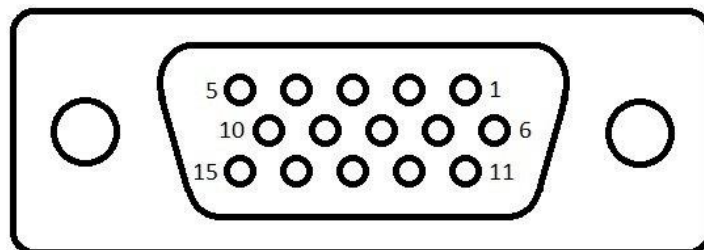
Objectives:

- Get familiar with the state machine design in VHDL.
- Learn how to use the Video Graphics Array (VGA) interface.
- Learn how to generate Hsync and Vsync signals.
- Use VHDL to implement a simple VGA controller to display a fixed pattern.

In this lab, you need to program two counters for generating the Hsync and Vsync signals with a correct counter maximum number as shown in page 12 and 13, and also one FSM with simple push button. There are 4 different parts and 3 checkpoints as shown in Page 13, 18 and 21.

Experiment:

Video Graphics Array (VGA) is a standard interface for controlling analog monitors. It is used for high-definition video, including resolutions of 1080p and higher. VGA connections use a 15 pins connector called a DB15. Function of each pin is listed as table below.



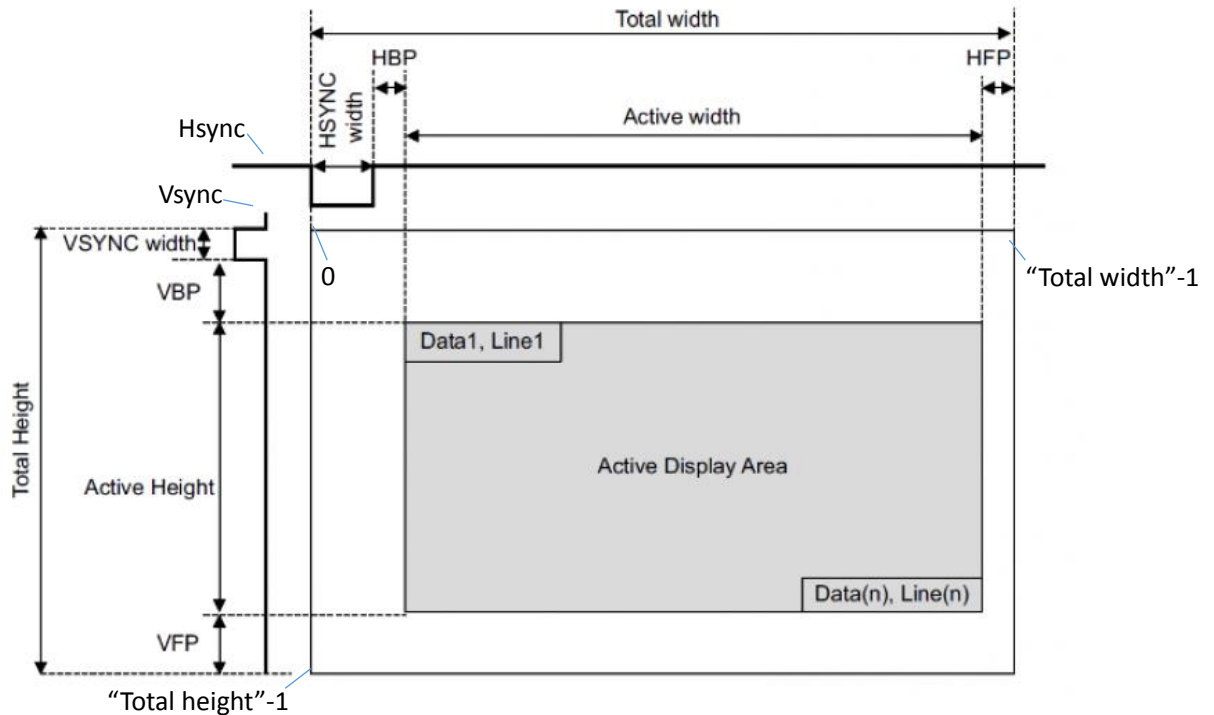
Pin 1	RED	Red video
Pin 2	GREEN	Green video
Pin 3	BLUE	Blue video
Pin 4	ID2/RES	formerly Monitor ID bit 2, reserved since E-DDC
Pin 5	GND	Ground (HSync)
Pin 6	RED_RTN	Red return

Pin 7	GREEN_RTN	Green return
Pin 8	BLUE_RTN	Blue return
Pin 9	KEY/PWR	formerly key, now +5V DC, powers EDID EEPROM chip on some monitors
Pin 10	GND	Ground (VSync)
Pin 11	ID0/RES	formerly Monitor ID bit 0, reserved since E-DDC
Pin 12	ID1/SDA	formerly Monitor ID bit 1, I ² C data since DDC2
Pin 13	HSync	Horizontal sync
Pin 14	VSycn	Vertical sync
Pin 15	ID3/SCL	formerly Monitor ID bit 3, I ² C clock since DDC2

Keep in mind that, a monitor always displays a row, pixel by pixel from left to right. After a row is completed, it will move to the left-most of next row. From top to bottom, after all rows are displayed, it moves back to the upper left corner and continues for the next frame.

Part 1: Background study the timing diagram of the VGA interface

The VGA ports use two synchronization signals, one called Hsync (horizontal sync) to specify the time taken to scan a line, and the other called Vsync (vertical sync) to specify the time taken to scan all lines within a frame. Properly setting the Hsync and Vsync signals will be our main focus. The VGA timing specification diagram is shown below.



Based on the resolution of your monitor, we can search the corresponding timing value. In this lab, the monitor used in our demonstration station is with a resolution of 1280*1024 at a refresh rate of 60Hz. We can get the following table.

Table 1: The useful parameters are extracted as follows:

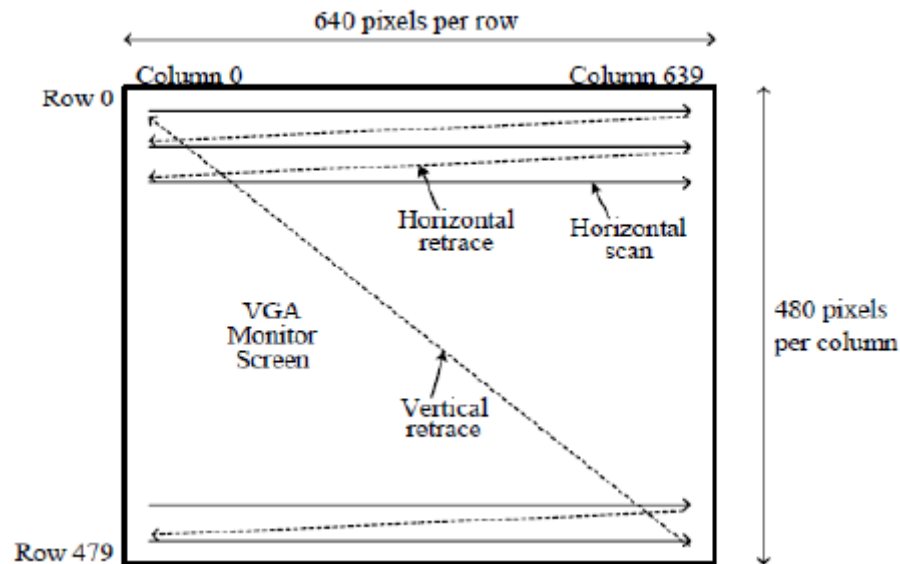
Pixel clock	108MHz
HSYNC width	112
HBP	248
HFP	48
Total width	1688
VSYNC width	3
VBP	38
VFP	1
Total height	1066

The pixel clock frequency defines the frequency of our VGA circuit design should be run at. In other words, all the vga control signal should synchronous with pixel clock. Sync pulse refers to HSYNC width/VSYNC width. Horizontal back porch/vertical back porch refers to HBP/VBP. Horizontal front porch/vertical front porch refers to HFP/VFP. Horizontal/vertical refers to total width/ total height. Based on the timing value, we can design a VGA controller circuit to generate Hsync and Vsync signal, which are shown in the previous timing diagram.

Part 2: Implement a vga_signal_generator module

i. Generate Hsync and Vsync

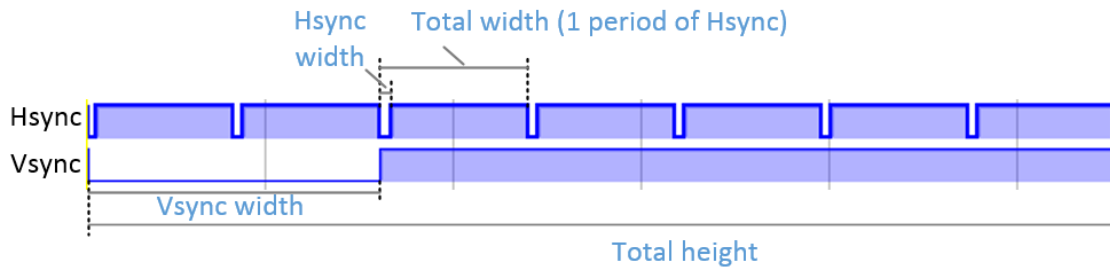
For VGA interface, the monitor scans the screen pixel by pixel, from left to right, from top to bottom as the figure below. After the whole screen is scanned, it goes back to the start point and continue to scan the screen to display next frame.



We divide the horizontal space of a screen, i.e. a line, into “Total width” pixels, starting at pixel 0 and ending at pixel Total width-1. For each clock cycle, we scan 1 pixel in a line. For example, if Hsync width=2, total width=10, Hsync first keeps low for the first 2 clock cycles, then high for the next $(10 - 2 = 8)$ clock cycles, as shown in the figure below. “total width” clock cycles is a period of Hsync. The length of one period of Hsync is equal to the time taken in scanning a line. When the monitor begins to scan a new line, Hsync should be repeated.

Likewise, we divide the vertical space into “Total height” horizontal lines, with the scanning through of all rows counting as one refresh of the screen, starting at line 0 and ending at line Total height -1. As shown in the figure below, if Vsync width=2, total height=7, Vsync first keeps low for the first two periods of Hsync, then high for the next $(7 - 2 = 5)$ periods of Hsync. One period of Vsync equals to the time taken in scanning all the lines within a frame. When the monitor start a new frame, Vsync should be repeated.

As shown in the figure below, Hsync demonstrating the monitor scan 10 pixels in a line, while Vsync demonstrate the monitor scan 7 line in a frame.



Hints:

One of the solutions to generate such signal is that, we can set up two counters, a horizontal counter, `x_cnt`, to count pixels scanned in each line and a vertical counter, `y_cnt`, to count lines scanned within a frame. The horizontal counter needs to reset itself when it reaches the end of a line (1688-1 in this case). The vertical counter should increase 1 only when the horizontal counter hits the end of a line (1688-1 in this case). Similarly, vertical counter needs to reset itself when it reaches the end of the frame (1066-1 in this case). Keep in mind that the two counters are starting at 0.

A framework is provided for you as below. All the variables are pre-defined for you. Notice: we can refer to table 1 in page 5 for all the vga parameters, like “total width/height”, “Hsync/Vsync width”.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;

entity vga_signal_generator is
port( pixel_clk      :in std_logic;
rst                :in std_logic;
      vga_r,vga_g,vga_b :out std_logic_vector(3 downto 0);
      vga_hs,vga_vs     :out std_logic;
--button input, change select display pattern
      turn_mode       :in std_logic;
--connect with LD0-LD3
      mode             :out std_logic_vector(3 downto 0)
);
end vga_signal_generator;

architecture Behavioral of vga_signal_generator is
signal x_cnt: integer range 0 to 4095;
signal y_cnt: integer range 0 to 4095;
signal hsync_r:std_logic;
```

```

signal vsync_r:std_logic;
signal hs_de:std_logic;
signal vs_de:std_logic;
signal key_counter: integer range 0 to 2000000;
signal btnDetection: std_logic;
signal dis_mode:unsigned(3 downto 0);
signal vga_r_reg,vga_g_reg,vga_b_reg:std_logic_vector(3 downto 0);//horizontal
counter
begin
--generate horizontal counter signal x_cnt
horizontal_cnt: process(pixel_clk,rst)
begin
    if(rst='1')then
        x_cnt<=0;
    elsif rising_edge(pixel_clk)then
        --Write your code here
        --Hints:

        --(check x_cnt reach total width or not
        --if yes, reset it to 0
        --if no, increase x_cnt)
    end if;
end process;

-- generate horizontal sync signal hsync_r
hsync_gen: process(pixel_clk,rst)
begin
    if(rst='1')then
        hsync_r<='1';
    elsif rising_edge(pixel_clk)then
        --Write your code here
        --Hints:

        --(check x_cnt value,
        --if it is between 0 and Hsync width, hsync_r set to 0
        --if not, hsync_r set to 1)
    end if;
end process;

```

```

--generate vertical counter signal y_cnt
vertical_cnt: process(pixel_clk,rst)
begin
if(rst='1')then
    y_cnt<=0;
elseif rising_edge(pixel_clk)then
    --Write your code here
    --Hints:

    --(check y_cnt
    --if y_cnt reach total height, reset it to 0
    --if not, check x_cnt value,
    --if x_cnt reach total width, y_cnt increase 1)
end if;
end process;

--generate vertical sync signal vsync_r
Vsync_gen: process(pixel_clk,rst)
begin
if(rst='1')then
    vsync_r<='1';
elseif rising_edge(pixel_clk)then
    --Write your code here
    --Hints:

    --(check y_cnt value,
    --if it is between 0 and Vsync width, vsync_r set to 0
    --if not, vsync_r set to 1)
end if;
end process;

```

ii. Simulate your semi-completed design

It is a good habit to do simulation when you finish some small. Because at this moment, the design is small and it is easy to debug your code. Since we want to observe the internal signal, we should add this signals to the waveform window manually.

Add the testbench vga_disp_tb.vhd to your project. To accelerate the simulation process, as we did in Lab 4, we should modify the value TIME_10MS in “vga_signal_generator.vhd” to 10. Then, run behavioral simulation.

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

entity vga_disp_tb is
end vga_disp_tb;

architecture Behavioral of vga_disp_tb is
component vga_control_circuit is
port(
    clk100M: in std_logic;
    rst: in std_logic;
    key: in std_logic;
    vga_r,vga_g,vga_b: out std_logic_vector(3 downto 0);
    vga_hs,vga_vs: out std_logic;
    led: out std_logic_vector(3 downto 0)
);
end component;
signal clk100M: std_logic;
signal rst: std_logic;
signal key: std_logic;
signal vga_r,vga_g,vga_b: std_logic_vector(3 downto 0);
signal vga_hs,vga_vs: std_logic;
signal led: std_logic_vector(3 downto 0);
--signal i: integer range 0 to 10;

constant clk_period: time := 10ns;
begin
    uut: vga_control_circuit
    port map(clk100M,rst,key,vga_r,vga_g,vga_b,vga_hs,vga_vs,led);

    clk_gen: process
    begin
        clk100M <= '0';
        wait for clk_period/2;
        clk100M <= '1';
```

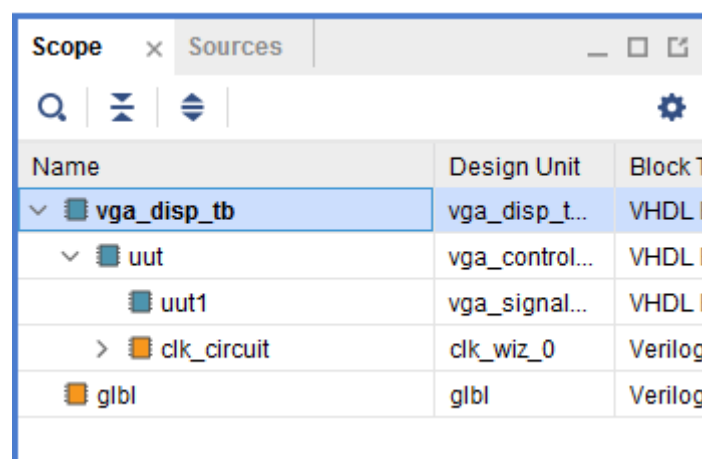
```

        wait for clk_period/2;
end process;

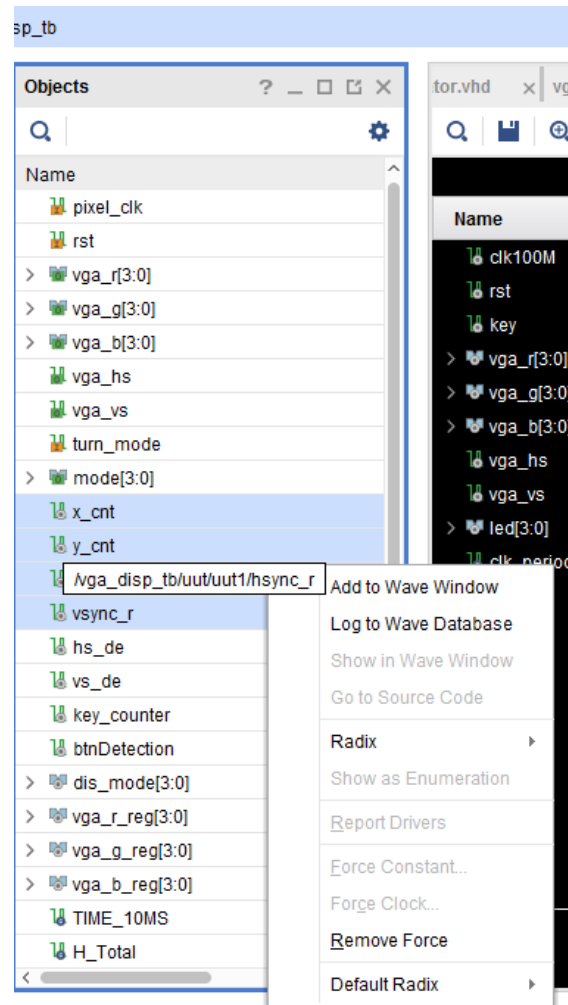
stim_proc: process
begin
    rst<='0';
    key<='0';
    wait for 1100ns;
    rst<='1';
    wait for 30ns;
    rst<='0';
    wait for 450us;
    for i in 0 to 5 loop
        wait for 34000ns;
        key<='1';
        wait for 10ns;
        key<='0';
    end loop;
    wait;
end process;
end Behavioral;

```

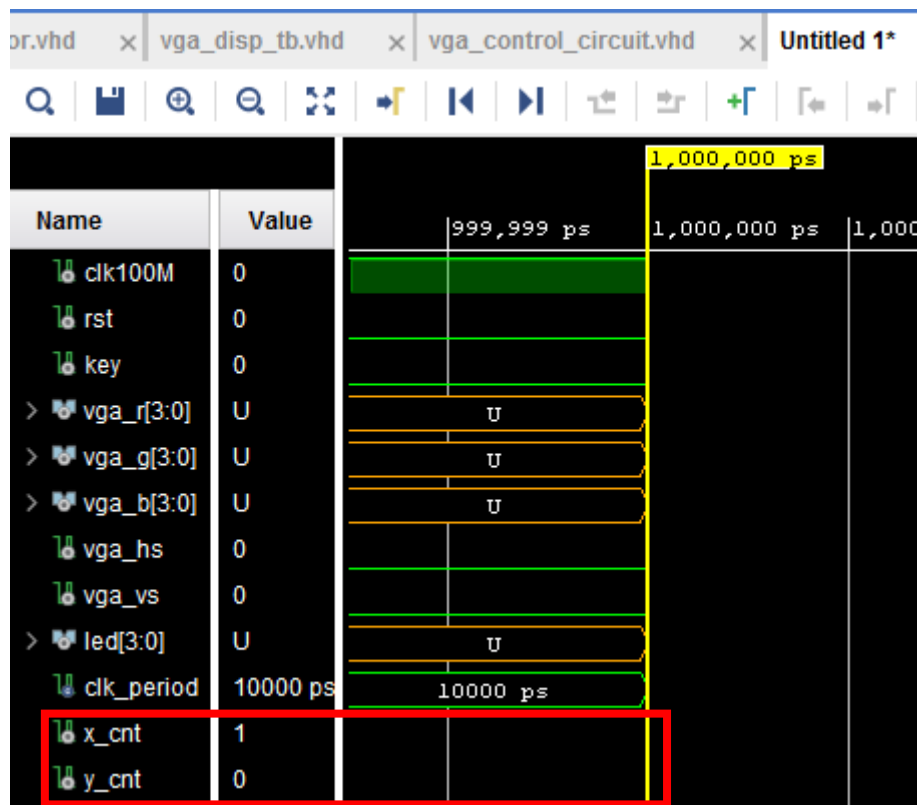
Run behavioral simulation. Then locate to the scope panel, expand uut, choose uut1 as below. Uut1 is the design we edit before.



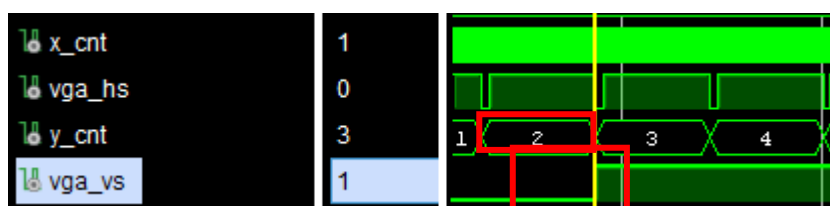
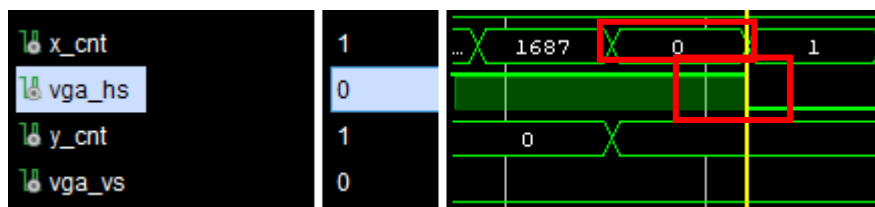
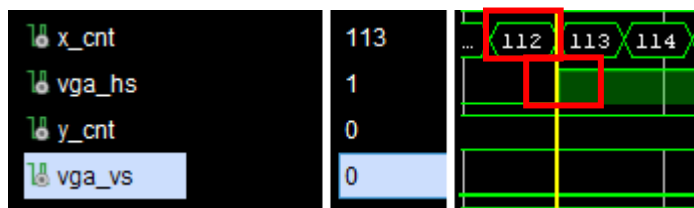
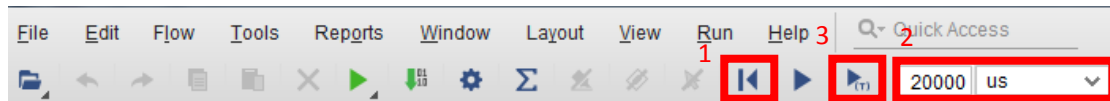
On the left of scope panel, we can see objects panel. Choose signal x_cnt, y_cnt, hsync_r, vsync_r, then right click, choose Add to wave window.

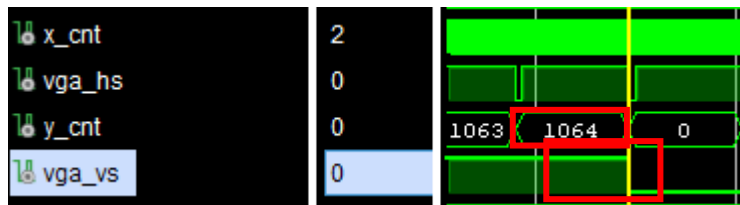


In the wave window, you can find we have 4 new signals but no waveform for them.



Click restart on the toolbar (or ctrl+shift+F5), then, change the simulation time to 13000us, then click run for 13000us (or shift+F2)



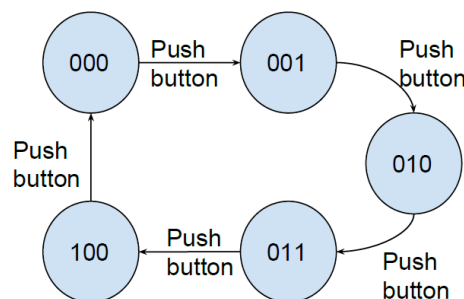


Checkpoint 1: Zoom (ctrl+mouse wheel) the waveform to proper level and locate to the according place. Check the 4 points as shown above and demonstrate your results to the tutor.

iii. Model a FSM for the controller

Each pixel on the monitor has a red, green, and blue color components. The red, green, and blue color signals from the VGA port to the monitor are analog and are generated by resistor ladder DACs on the FPGA board that receive input from dedicated FPGA IO lines. The ZedBoard uses 12 IO lines to derive the color signal, and therefore uses an 12-bit color scheme, with 4 bits determining red, green and blue color intensity respectively.

In this experiment, we will implement a similar FSM as what we have done in the Lab 4, which can change the display pattern when the button is clicked. In total, we pre-define 5 patterns to be display on the monitor. 3 4-bit registers are defined to cache the red, green and blue color intensity, i.e. vga_r_reg, vga_g_reg and vga_b_reg. Different pre-defined display pattern require different setting of the 3 registers. A register “dis_mode” is defined to store the current display pattern number (i.e. it functions similar to the register “state” in Lab 4). When the button is hit, dis_mode increase 1. The diagram of the FSM is as follows, the state is same as dis_mode value. We also use a signal “rst” to reset the FSM.



For the 5 pre-defined display pattern:

dis_mode	pattern
000	Black screen: vga_r_reg=vga_g_reg=vga_b_reg="0000"

001	White screen: vga_r_reg=vga_g_reg=vga_b_reg="1111"
010	Red screen: vga_r_reg="1111", vga_g_reg=vga_b_reg="0000"
011	Green screen: vga_g_reg="1111", vga_r_reg=vga_b_reg="0000"
100	Blue screen: vga_b_reg="1111", vga_r_reg=vga_g_reg="0000"

```

key_detect: process(pixel_clk)
begin
    if rising_edge(pixel_clk)then
        if(turn_mode='1')then
            key_counter<=0;
        elsif (turn_mode='0') and (key_counter<=TIME_10MS)then
            key_counter<=key_counter+1;
        end if;
    end if;
end process;

button_assign: process(key_counter)
begin
    if(key_counter=TIME_10MS)then
        btnDetection<='1';
    else
        btnDetection<='0';
    end if;
end process;

--once detect the button pressed, register dis_mode increase 1
mode<=std_logic_vector(dis_mode);
display_pattern_reg: process(pixel_clk,rst)
begin
    if(rst='1')then
        dis_mode<="0000";
    elsif rising_edge(pixel_clk)then
        --Write your code here

    end if;

```

```

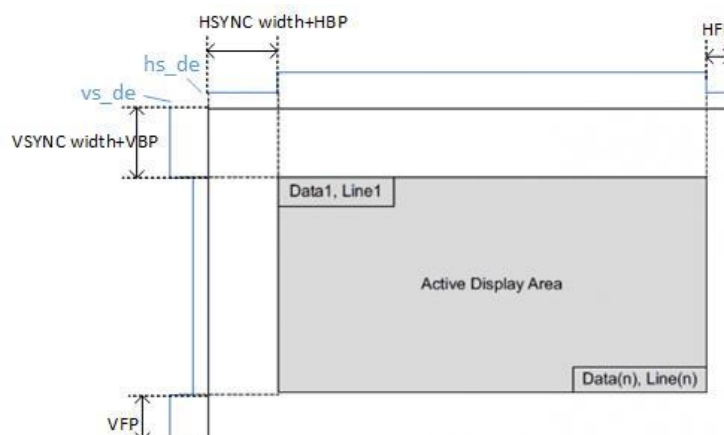
end process;

--based on the value of dis_mode, change vga_r_reg/ vga_g_reg/ vga_b_reg
accordingly
rgb_setting: process(pixel_clk,rst)
begin
    if(rst='1')then
        vga_r_reg<="0000";
        vga_g_reg<="0000";
        vga_b_reg<="0000";
    elsif rising_edge(pixel_clk)then
        --Write your code here

    end if;
end process;

```

Similar to the Lab4, the last step is to assign correct value to the output signal. Since only pixels falls into active display area will be visible, we need a horizontal enable signal (hs_de) and a vertical enable signal (vs_de) goes high when the scan comes into the active display area. In this lab, hs_de keeps low at first, and when x_cnt hits 359 (i.e. HSYNC width+HBP-1), hs_de is changed to be high. When x_cnt hits 1639 (i.e. Total width-HFP-1), hs_de goes low. Vs_de keeps low at first, and when y_cnt hits 40 (i.e. HSYNC width+HBP-1), vs_de goes high. When y_cnt hits 1064 (i.e. Total height-VFP-1), vs_de goes low. Registers vga_r_reg/vga_g_reg/vga_b_reg are assigned to vga_r/vga_g/vga_b only when hs_de and vs_de are both high, i.e. the scan comes into the active display area.



Keep in mind that we use two counters to trace the scan. Horizontal counter value trace the horizontal position the monitor scans. Likewise, vertical counter value trace

the vertical position the monitor scans.

Notice: we can refer to table 1 in page 5 for all the vga parameters, like “HBP”, “HFP”, “VBP”, “VFP” “Hsync/Vsync width”, “Total width/height”.

```
--generate horizontal enable signal hs_de
horizontal_enable: process(pixel_clk,rst)
begin
    if(rst='1')then
        hs_de<='0';
    elsif rising_edge(pixel_clk)then
        --Write your code here
        --Hints:
        --(check x_cnt,
        --if it is between Hsync width+HBP and Total width- HFP, hs_de is set to 1
        --otherwise, hs_de is set to 0)
        end if;
    end process;

--generate vertical enable signal vs_de
vertical_enable: process(pixel_clk,rst)
begin
    if(rst='1')then
        vs_de<='0';
    elsif rising_edge(pixel_clk)then
        --Write your code here
        --Hints:
        --(check y_cnt,
        --if it is between Vsync width+VBP and Total height- VFP, vs_de is set to 1
        --otherwise, vs_de is set to 0)
        end if;
    end process;

--assign correct value to output signals
vga_hs<=hsync_r;
vga_vs<=vsync_r;
vga_rgb_assignment:process(hs_de,vs_de, vga_r_reg, vga_g_reg, vga_b_reg)
begin
    case (hs_de and vs_de) is
        when '1'    =>vga_r<=vga_r_reg;
```

```

        vga_g<=vga_g_reg;
        vga_b<=vga_b_reg;
    when others=>vga_r<="000";
        vga_g<="000";
        vga_b<="000";

    end case;

end process;

end Behavioral;
```

At this stage, we have completed the `vga_data_gen` module.

Finally, we need to configure a clock module to generate the pixel clock which is used to synchronous all the vga control signal. This can be done by using clocking wizard in IP Catalog. We have already done the setting and instantiate it in the top module vga control circuit by using the statements below.

```
--instantiate a clock generator here
```

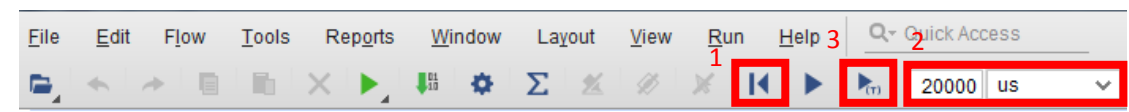
clk_circuit: clk_wiz_0

```
port map(pixel_clk,'0',clk100M);
```

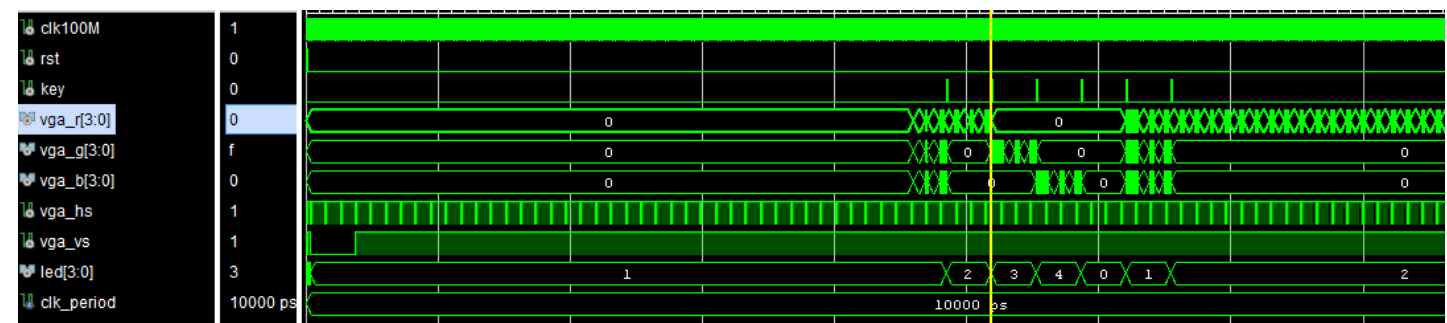
Part 3: Simulation

Use the same testbench, run behavioral simulation. Following the instruction in Part 2 simulation step, add internal signal `x_cnt` in `vga_signal_generator` module to the wave window manually.

By default, the simulation will stop at 1000ns. In this experiment, we need to observe much longer time. So we need to restart the simulation, i.e. click “1”, and type the simulation time 20000 us, i.e. alter box “2” value, and then re-run it, i.e. click “3”.



If your design is correct, signal “led” will be switched from 0-4. Once the pulse of signal “key” occurs, signal “led” will increase 1 as below.



Checkpoint 2: Show your simulation waveform to your demonstrator and verify your design.

Part 4: Program the design to ZedBoard

- i. Modify the value TIME_10MS in “vga_signal_generator.vhd” to 1000000.
- ii. Add the constraint file to your project.

In this experiment, we will use the GCLK on board as the clock input. Button BTNU(T18) is used as “rst” input. Button BTNC(P16) is used as “key” input. When we click BTNC(P16), it will change the display pattern on monitor. LD0-LD3 are used to display the value of “disp_mode” in vga_signal_generator module. Vga_vs, vga_hs, vga_r, vga_g, vga_b are connected to the corresponding pin of VGA interface on-board.

Normally, in the constraint file, once we assign a pin to the I/O port of our design, we need to set its IOSTANDARD, like:

```
set_property PACKAGE_PIN Y9 [get_ports {clk100M}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk100M}]
```

constraint file:

```
# Clk(GCLK on board)
set_property PACKAGE_PIN Y9 [get_ports {clk100M}]
set_property IOSTANDARD LVCMOS33 [get_ports {clk100M}]

# Rst
set_property PACKAGE_PIN T18 [get_ports {rst}]
set_property IOSTANDARD LVCMOS33 [get_ports {rst}]

# key
set_property PACKAGE_PIN P16 [get_ports {key}]
set_property IOSTANDARD LVCMOS33 [get_ports {key}]

# Led0
set_property PACKAGE_PIN T22 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]

# Led1
set_property PACKAGE_PIN T21 [get_ports {led[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
```

```

# Led2
set_property PACKAGE_PIN U22 [get_ports {led[2]]}
set_property IOSTANDARD LVCMOS33 [get_ports {led[2]]}

# Led3
set_property PACKAGE_PIN U21 [get_ports {led[3]]}
set_property IOSTANDARD LVCMOS33 [get_ports {led[3]]}

set_property PACKAGE_PIN Y19 [get_ports {vga_vs}]
set_property IOSTANDARD LVCMOS33 [get_ports {vga_vs}]

set_property PACKAGE_PIN AA19 [get_ports {vga_hs}]
set_property IOSTANDARD LVCMOS33 [get_ports {vga_hs}]

set_property PACKAGE_PIN V20 [get_ports {vga_r[0]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_r[0]]}
set_property PACKAGE_PIN U20 [get_ports {vga_r[1]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_r[1]]}
set_property PACKAGE_PIN V19 [get_ports {vga_r[2]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_r[2]]}
set_property PACKAGE_PIN V18 [get_ports {vga_r[3]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_r[3]]}

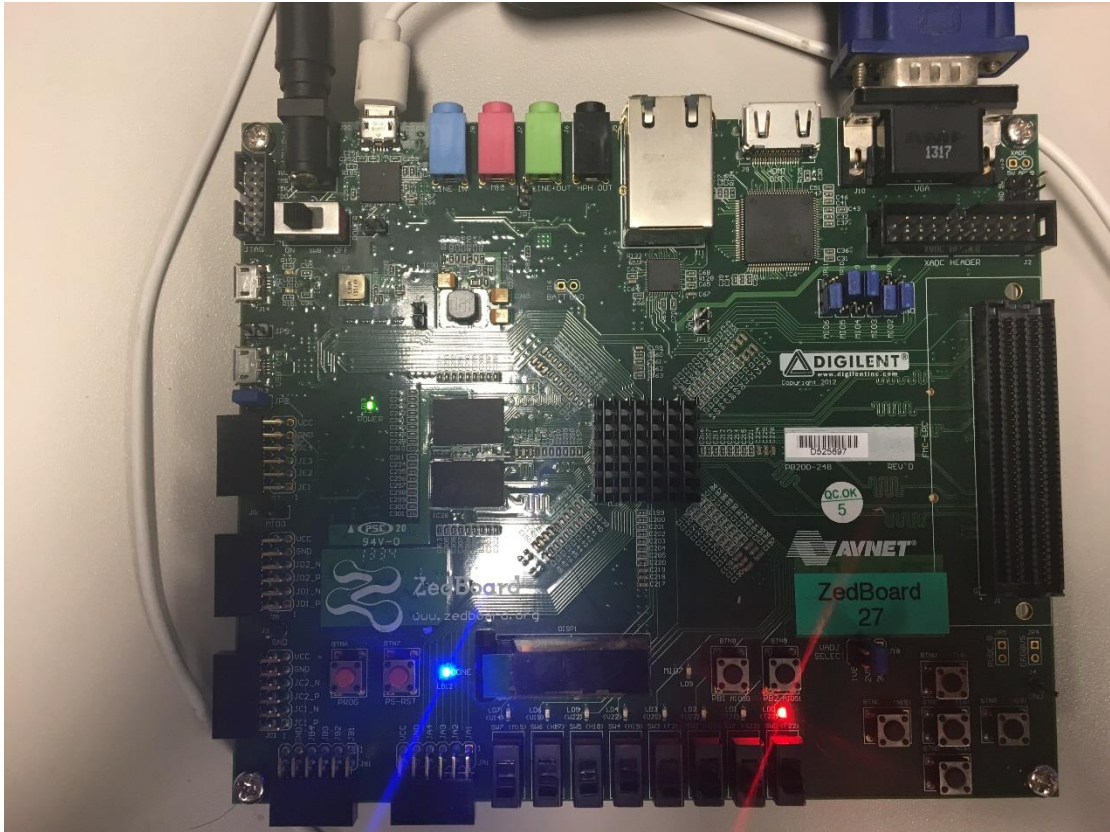
set_property PACKAGE_PIN AB22 [get_ports {vga_g[0]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_g[0]]}
set_property PACKAGE_PIN AA22 [get_ports {vga_g[1]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_g[1]]}
set_property PACKAGE_PIN AB21 [get_ports {vga_g[2]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_g[2]]}
set_property PACKAGE_PIN AA21 [get_ports {vga_g[3]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_g[3]]}

set_property PACKAGE_PIN Y21 [get_ports {vga_b[0]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_b[0]]}
set_property PACKAGE_PIN Y20 [get_ports {vga_b[1]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_b[1]]}
set_property PACKAGE_PIN AB20 [get_ports {vga_b[2]]}
set_property IOSTANDARD LVCMOS33 [get_ports {vga_b[2]]}

```

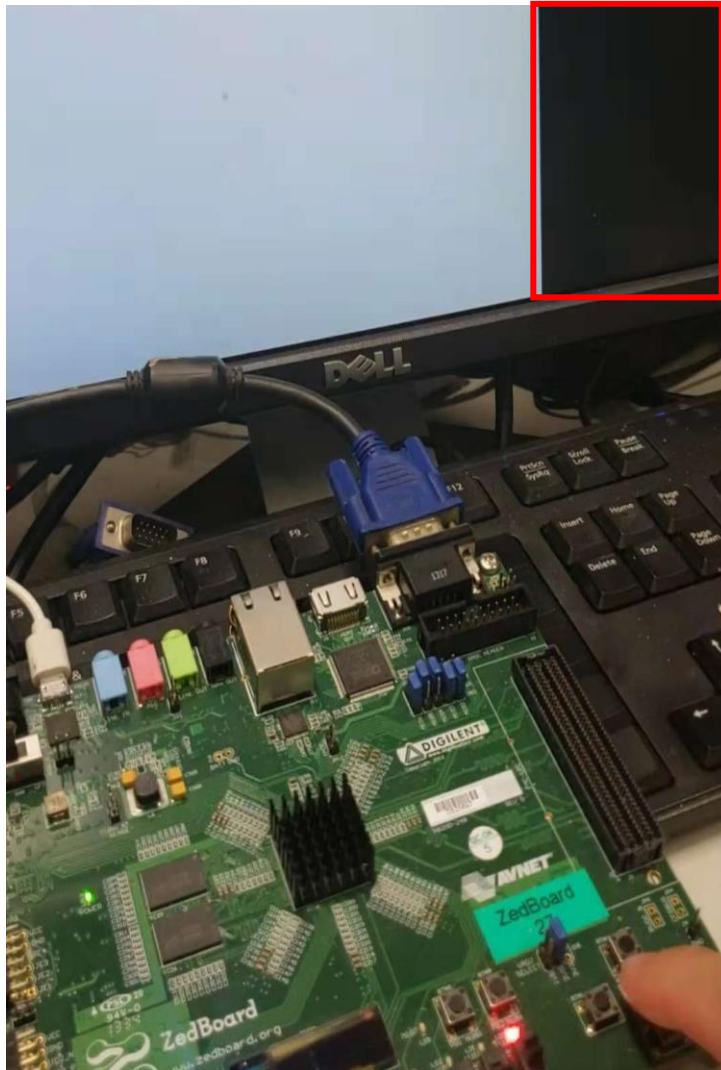
```
set_property PACKAGE_PIN    AB19  [get_ports {vga_b[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {vga_b[3]}]
```

- iii. **Run synthesis, Run implementation, Generate bitstream** as Lab 1.
- iv. Connect the ZedBoard to PC. Then, we **program** the bitstream file to the ZedBoard.
- v. Connect VGA interface on-board with your monitor. After that, you can verify the design by checking the output of leds and the monitor. When press BTNU, all LED should be dim. The monitor should display black screen. Then press BTNC, LD0 illuminate and monitor change to white screen. Following the pre-defined display pattern table in Part 2 and verify your design.



Checkpoint 3: Go to the designated place in the lab to demo your final logic design.

Bonus point: Modify the design so that it is adapted to a monitor with a new resolution of 800*600 and refresh rate at 60Hz. Because of resolution mismatch, you will find that there is a black space on the right part of the screen. Think about why.



(Hints:

1. You should go to the website <http://tinyvga.com/vga-timing/800x600@60Hz> to check the vga timing value and pixel clock frequency for the monitor of a resolution of 800*600 at 60Hz.
2. Modify 6 process blocks: horizontal_cnt, hsync_gen, vertical_cnt, vsync_gen, horizontal_enable, vertical_enable.
3. Change the setting of the IP we used to generate pixel clock, i.e. change the output frequency of IP “clk_wiz_0” required by a specific monitor.)