ELE306 FALL2004 LAB3

VGA and PS/2 -- Reusing existing VHDL modules

In this lab, we will learn about the use of existing VHDL modules in new designs. Two modules in particular will help interface the keyboard via PS/2 port and the VGA monitor.

Obtain the Existing VHDL Modules

The four existing VHDL modules to be used here are: sevenseg.vhd (from lab2), keyboard.vhd, vga_sync.vhd and square.vhd. The last three files are at the ELE306 ftp.

- At ftp://ftp.ele.uri.edu/outgoing/jcl/306 obtain keyboard.vhd, vga_sync.vhd and square.vhd. You may use web browser or any ftp program on this anonymous ftp.
- keyboard.vhd: is the interface circuit to PS/2 port that receives keyboard input.
- vga_sync.vhd: is the circuit that generates timing signal and display signal for the VGD display. Our lab is equipped with dual-input LCD monitors. Simply switch to the second VGA input source with the middle button (with minus sign) next to the power switch.
- square.vhd: is the circuit that produce display signal at the right time, in collaboration with vga_sync.vhd, to show an image of a square on the monitor.
- You may save these files anywhere now and import then into your lab3 projects.
- There are two ways to incorporate "external" files into a new project:
    - At the second page of the "New Project Wizard", add the external files. Remember that these files need not be in the project directory.
    - Go to "Project → Add/Remove Files in Project" after a new project has created to add external files or remove the unnecessary files.

Exercise #1: (Creating a design with the keyboard as an input)

Report for Exercise #1: Draw the flowchart of keyboard.vhd

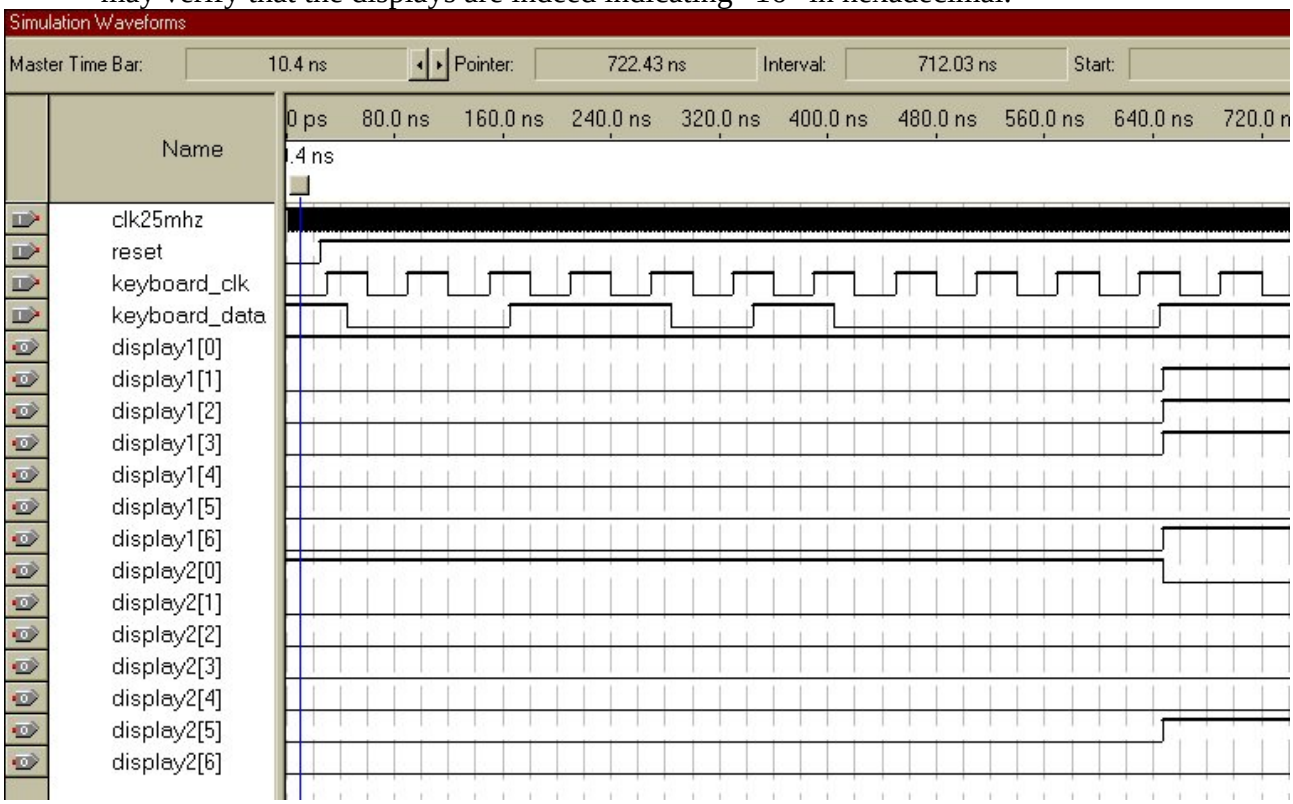
- Create a project called lab3a. This simple circuit receives keyboard input and display the received code on the 7segment display.
- Add sevenseg.vhd and keyboard.vhd to this new project. The list of included files will be shown on the left window pane of Quartus II. Click on the "Files" tab at the bottom. You should see the two included files under "Device Design Files."
- Create a VHDL file called lab3a.vhd with the following contents.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY lab3a IS
    PORT(   clk25MHz : IN std_logic;
            reset : IN std_logic;
            keyboard_clk, keyboard_data : IN std_logic;
            display1, display2 : OUT std_logic_vector(6 DOWNTO 0)
            );
```

```
END lab3a;

ARCHITECTURE mylab3a OF lab3a IS
    COMPONENT keyboard
            PORT(   keyboard_clk, keyboard_data, clock_25Mhz ,
                    reset, read : IN STD_LOGIC;
                    scan_code : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                    scan_ready : OUT STD_LOGIC
                );
    END COMPONENT;

    COMPONENT sevenseg
            PORT(   ain : IN std_logic_vector(3 DOWNTO 0);
                    aout : OUT std_logic_vector( 6 DOWNTO 0)
                );
    END COMPONENT;

--define internal signals
SIGNAL scan_code : std_logic_vector(7 DOWNTO 0); --keyboard
SIGNAL scan_ready : std_logic; --keyboard

BEGIN
    keyin: keyboard PORT MAP(keyboard_clk, keyboard_data, clk25MHz,
                reset => NOT reset, read => '0', scan_code => scan_code, scan_ready => scan_ready);
    disp1: sevenseg PORT MAP(scan_code(7 DOWNTO 4), display1 );
    disp2: sevenseg PORT MAP(scan_code(3 DOWNTO 0), display2 );
END mylab3a;
```

- We learned in lab2 that “named association” or “positional association” can be used in “Port Map”. Here you see an example of how these two usages can be mixed together. The restriction is that positional association must go first. Once you start with named association, you will have to use it all the way to the end!
- The above VHDL code simply put together three existing modules (one keyboard module and two 7-segment modules) to form a new design. If successful the two 7-segment displays on the UP-2 board will show the 8-bit code being transmitted from the keyboard when a key is hit. For details of these “scan codes for PS/2 keyboard” can be found in Chapter 10 of “Rapid Prototyping of Digital Systems.”
- For now, we will not delve into the workings of the PS/2 standard, but data is serially transmitted over the keyboard_data line and the keyboard module recombines the 11bit packets into an 8bit character that is held in scan_code. When the process of receiving the 11bits is done, the scan_ready line will be set to signify a character is ready to be read. The packet consists of a start bit ‘0’, the 8 character bits, an odd parity bit, and a stop bit ‘1’. To send a code with value of 16 (hexadecimal or “00010110” in binary), the following series of bits would be sent “00110100001”, or as seen in Figure 10.1 in Chapter 10 of ““Rapid Prototyping of Digital Systems.” **The code itself was sent in low to high order!!!**
- Perform a functional simulation to verify the functionality. You will have to provide the keyboard response (just gave a number for scan_code and activate scan_ready at the appropriate time). The following sample waveform shown the simulation condition that emulates that found in Figure 10.1 mentioned above.

- Note that clk25mhz was set with cycle time= 1ns and keyboard_clk with cycle time = 30ns. The keyboard_data was set by selecting the appropriate region and assign logic values. You may verify that the displays are indeed indicating “16” in hexadecimal.



- Assign the pins given in the following table.

Pin Name	Pin Location
clk25MHz	Pin_91
display1[0]	Pin_13
display1[1]	Pin_12
display1[2]	Pin_11
display1[3]	Pin_9
display1[4]	Pin_8
display1[5]	Pin_7
display1[6]	Pin_6
display2[0]	Pin_24
display2[1]	Pin_23
display2[2]	Pin_21
display2[3]	Pin_20
display2[4]	Pin_19
display2[5]	Pin_18
display2[6]	Pin_17
keyboard_clk	Pin_30
keyboard_data	Pin_31
reset	Pin_41

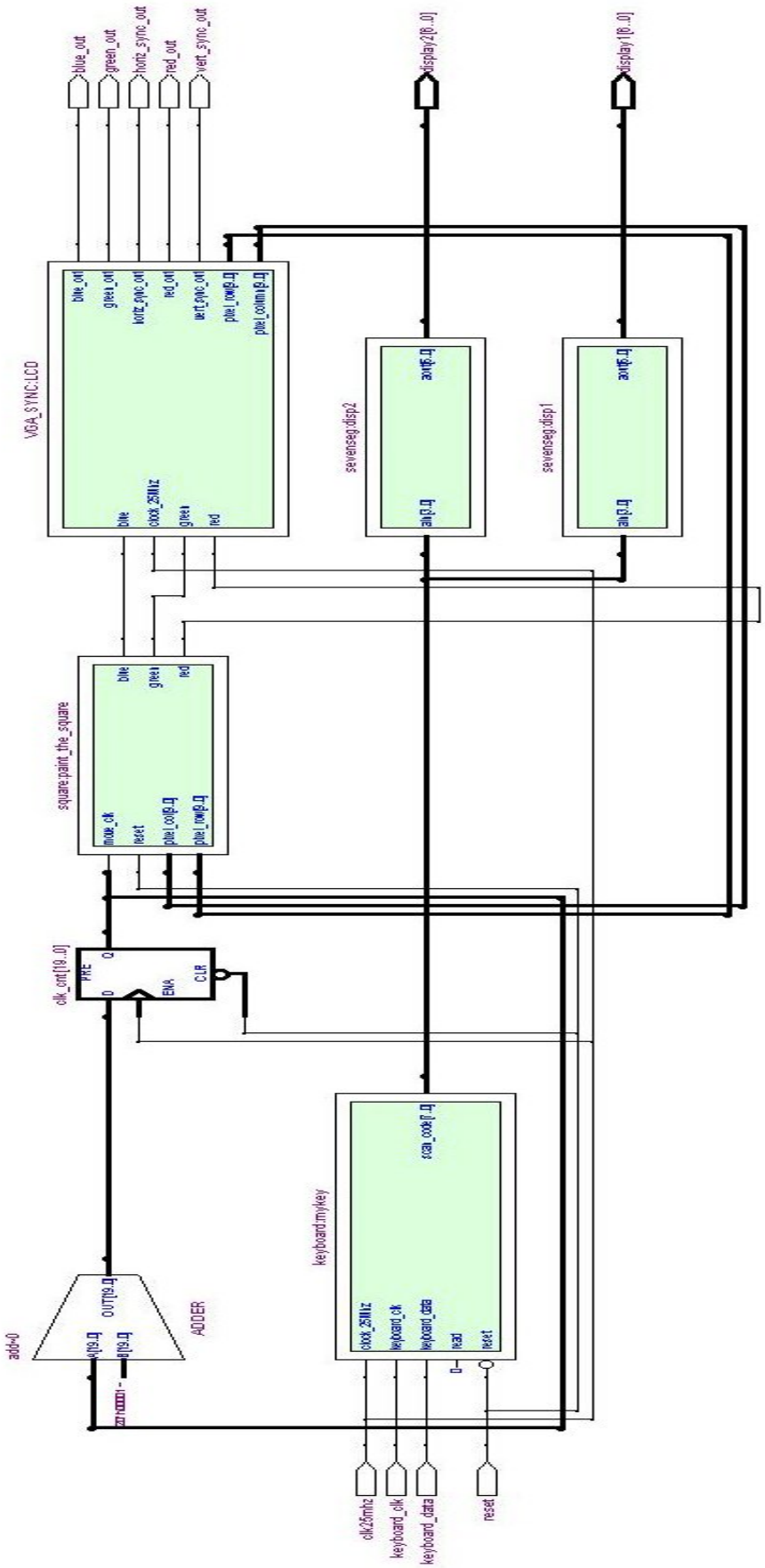
- Note that PIN_91 is the designated input for the global clock signal of FPGA.

- From Table 10.3 of the above mention book, available at lab, a 1-byte "make code" and a 2-bytes "break code" are associated with each key (Figure 10.3 show the key numbers). When a key is pressed, the "make code" is sent. If you hold the key, this code will remain at the display (it was latched). When the key is finally release, the 2 bytes "break code" are sent from the keyboard. For a single key stroke (without shift, alt or num-lock) the first byte of the break code is "F0" follow by the same code as in the make code. You can barely see "F0" flash by because it will be replaced quickly with the second byte in the code.
- Since this is an existing module, we will skip the timing simulation and go straight to programming the FPGA, and then demonstrate the working results to the TA.

Exercise #2: (Using a VGA monitor)

Report for Exercise #2: Draw the flowcharts of vga_sync.vhd and square.vhd

- Create a new project call lab3b and import external files: sevenseg.vhd, keyboard.vhd, vga_sync.vhd and square.vhd.
- The RTL schematic of this circuit is shown on the next page.
- Create lab3b.vhd as follows. To avoid re-typing duplicated parts from lab3a, you may open lab3a.vhd, while in the editor and from lab3a directory, and copy and paste the entire code to lab3b.vhd. NOTE: This maneuver will change your working directory to lab3b. Remember to change it back to lab3b before saving.
- A description of how the vga_sync.vhd file and VGA monitor functions can be found in Chapter 9 of the book *Rapid Prototyping of Digital Systems*. From the flowchart you've resevered engineered how vag_sync.vhd generates those timing signal for the VGA video. Here are some vital numbers:
    - The VGA display is 640 pixels in a row and 480 pixels in a column (or simply 640X480), where pixel=picture element.
    - The Vertical sync signal occur every 16.6ms; this means a refresh rate of 60Hz. When vertical sync occurs, it means the beginning of a new frame. Since we want 480 rows, horizontal sync should be generated every 31.77μs. This can be seen from Figures 9.2 to 9.4.
    - To display anything on the screen, you control the red, green and blue (RGB) pixel data. In this case, we can only set RGB range from (000: black) to (111: white), or 8 different colors. As shown on Figure 9.4, all 640 pixels on one row are spread out in 25.17μs, or 39.328ns for every pixel. The on-board clock generator provides 25.175MHz (39.722ns cycle time), while not 100% accurate it is close enough.
    - Use the flowchart of vga_sync.vhd to formulate how the timing signals are generated.
- The square.vhd file draws a square on the screen that moves towards the bottom right. It keeps tracks of the timing signals and output the pixel data at the pre-defined time. HINT: imagine a continuous scanning of row by row and then repeat the entire frame again; you may map time to the space on the screen! Use the flowchart of square.vhd to see how the square on screen is generated.

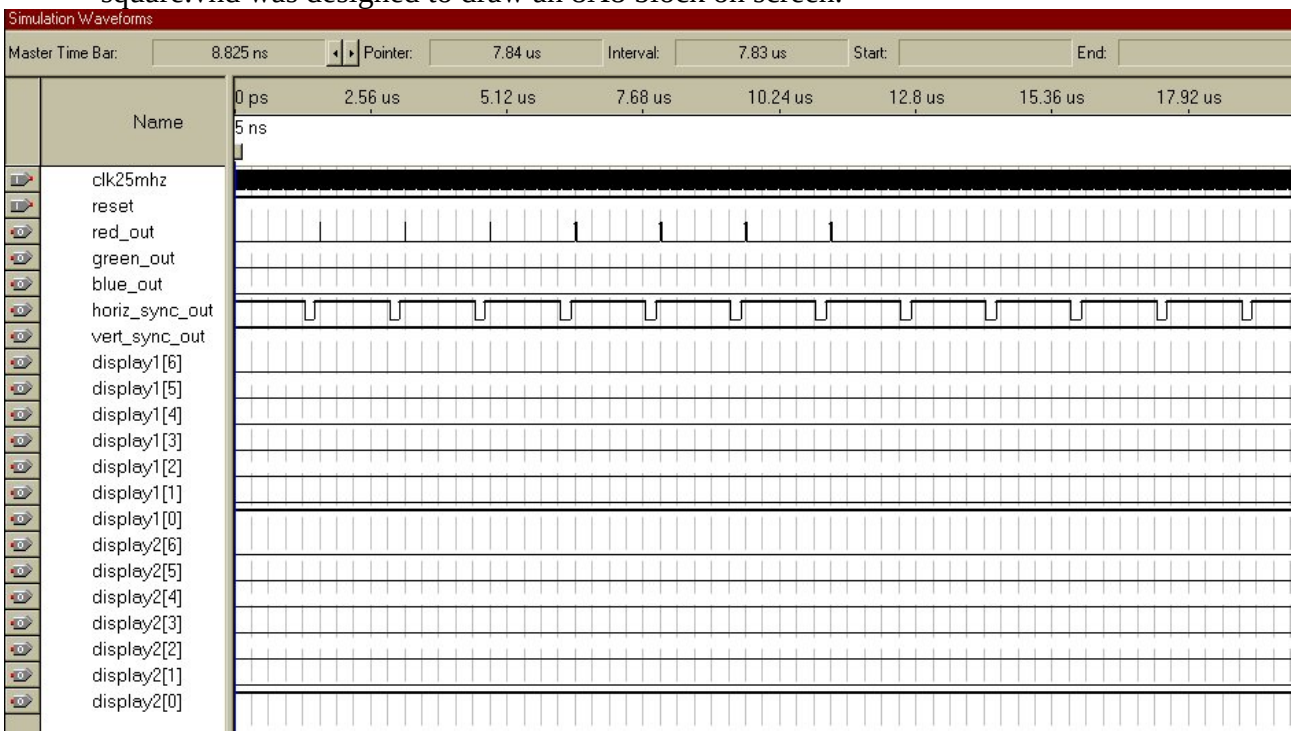

blue_out
green_out
horiz_sync_out
red_out
vert_sync_out
display2[6..0]
display1[6..0]
VGA_SYNC:LCD
sevenseg:disp2
sevenseg:disp1
square:paint_the_square
clk_cnt[19..0]
keyboard:mykey
ADDER
add=0
clk25mhz
keyboard_clk
keyboard_data
reset

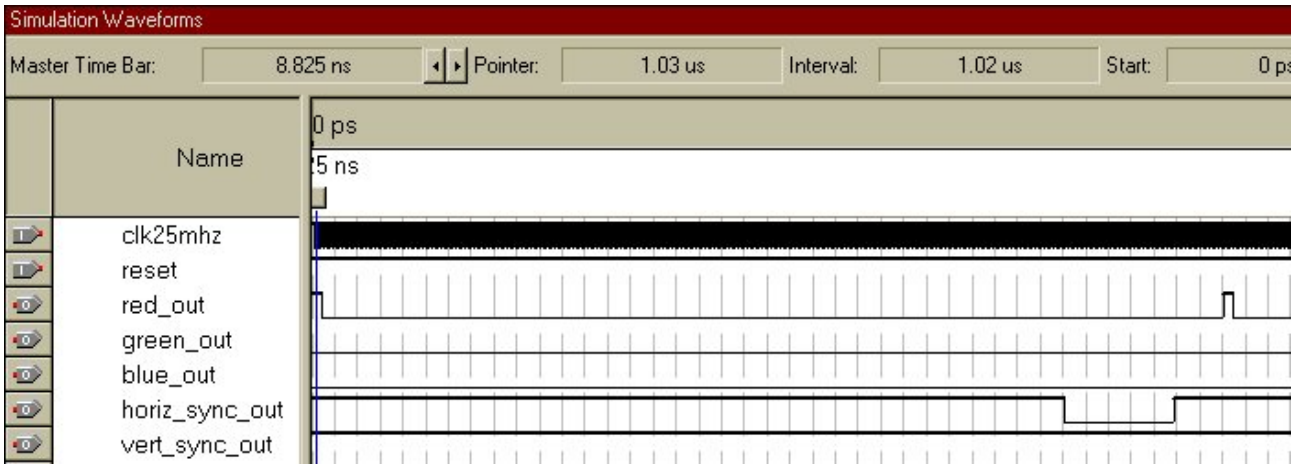
p

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_unsigned.all;

ENTITY lab3b IS
	PORT(  clk25MHz : IN std_logic;
		reset : IN std_logic;
		red_out : OUT std_logic;
		green_out : OUT std_logic;
		blue_out : OUT std_logic;
		horiz_sync_out : OUT std_logic;
		vert_sync_out : OUT std_logic;
		keyboard_clk, keyboard_data : IN std_logic;
		display1, display2 : OUT std_logic_vector(6 DOWNTO 0)
		);
END lab3b;

ARCHITECTURE mylab3b OF lab3b IS

	--Define the components
	COMPONENT keyboard
		PORT(   keyboard_clk, keyboard_data, clock_25Mhz,
			reset, read : IN STD_LOGIC;
			scan_code : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
			scan_ready : OUT STD_LOGIC
			);
	END COMPONENT;

	COMPONENT sevenseg
		PORT(   ain : IN std_logic_vector(3 DOWNTO 0);
			aout : OUT std_logic_vector( 6 DOWNTO 0)
			);
	END COMPONENT;

	COMPONENT vga_sync
		PORT(clock_25Mhz, red, green, blue : IN STD_LOGIC;
			 red_out, green_out, blue_out,
			 horiz_sync_out, vert_sync_out : OUT STD_LOGIC;
			 pixel_row, pixel_column  : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)
			);
	END COMPONENT;

	COMPONENT square
	PORT(   move_clk : IN std_logic;
			reset : IN std_logic;
			pixel_col : IN std_logic_vector( 9 DOWNTO 0);
			pixel_row : IN std_logic_vector( 9 DOWNTO 0);
			red : OUT std_logic;
			green : OUT std_logic;
			blue : OUT std_logic
			);
	END COMPONENT;

	--define signals
	SIGNAL red, green, blue : std_logic; --vga display
	SIGNAL pixel_row, pixel_column : std_logic_vector(9 DOWNTO 0); --vga display
```

```
        SIGNAL clk_cnt : std_logic_vector( 19 DOWNTO 0);  --step down counter for move_clk
        SIGNAL scan_code : std_logic_vector( 7 DOWNTO 0); --keyboard
        SIGNAL scan_ready : std_logic; --keyboard
BEGIN
        LCD: vga_sync PORT MAP( clk25MHz, red, green, blue, red_out, green_out, blue_out,
                        horiz_sync_out, vert_sync_out, pixel_row, pixel_column);
        paint_the_square: square PORT MAP(clk_cnt(19), reset, pixel_column, pixel_row,
                        red, green, blue );
        keyin: keyboard PORT MAP(keyboard_clk, keyboard_data, clk25MHz,
                reset => NOT reset, read => '0', scan_code => scan_code, scan_ready => scan_ready);
        disp1: sevenseg PORT MAP(scan_code(7 DOWNTO 4), display1 );
        disp2: sevenseg PORT MAP(scan_code(3 DOWNTO 0), display2 );

        PROCESS(clk25MHz, reset )
        begin
                IF (reset='0') THEN
                        clk_cnt <= "00000000000000000000";
                ELSIF( clk25MHz'event AND clk25MHz = '1') THEN
                        clk_cnt <= clk_cnt + 1;
                END IF;
        END PROCESS;
END mylab3b;
```

- For the functional simulation, we will mainly test the generation of timing signals. Since the entire screen will take 16.6ms to display, we will need to "compress" the simulation time a little. First, let clk25MHz=1ns; this effectively speed up the circuit by 40 times. This can only be done in functional simulation. Note that the simulation time in the following screenshot has been extended to 20µs. This is done by change the "Edit → End Time…" setting while performing simulation. We see 8 pulses are shown for the red_out; the square.vhd was designed to draw an 8X8 block on screen.





- The above screen shot is a zoom-in view of the simulation. You may calculate the duration of each red_out to confirm that it actually represent 8 pixels on each row.

- Assign the following pins.

Pin Name	Pin Location
blue_out	Pin_238
clk25MHz	Pin_91
display1[0]	Pin_13
display1[1]	Pin_12
display1[2]	Pin_11
display1[3]	Pin_9
display1[4]	Pin_8
display1[5]	Pin_7
display1[6]	Pin_6
display2[0]	Pin_24
display2[1]	Pin_23
display2[2]	Pin_21
display2[3]	Pin_20
display2[4]	Pin_19
display2[5]	Pin_18
display2[6]	Pin_17
green_out	Pin_237
Horiz_sync_out	Pin_240
keyboard_clk	Pin_30
keyboard_data	Pin_31
red_out	Pin_236
Reset	Pin_41
vert_sync_out	Pin_239

- Program the UP-2 board after compilation and then demonstrate the working results to the TA or the instructor.

Assignments

1. Modify exercises #2 to produce a bouncing square that bounds off the boundaries of the screen. Remember that the screen is at the resolution of 640X480.

2. Extend from the above circuit of assignment 1, upon receiving a numeric key input (0-7 or 1-8 depends on your own definition), the color of the square changes accordingly. In the exercise#2, only red-out is used. When using all three RGB outputs, you should have 8 colors (including black or the background color!) available.

3. Extend from the above circuit of assignment 2 so the square will freeze whenever you type in the command “freeze” from the keyboard. The animation will continue when you type in “live”.