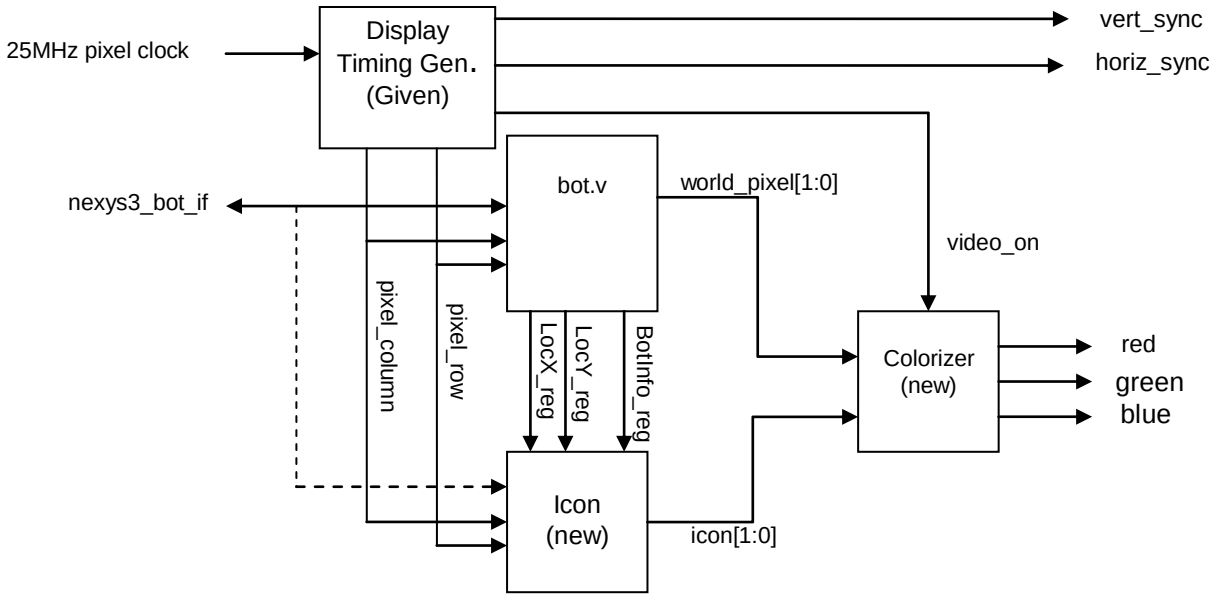
RojoBot World Video Controller
(Last Updated 14-October-2014)

One of the primary outcomes for Project 2 is to give you experience designing and implementing video. For this part of the project you will implement an animated and colorful graphical display for your RojoBot world that displays an image of the Rojobot virtual world with the Bot moving around on it. Many project teams have built on what they learned implementing the video controller to make their final project "visually interesting" (one of the requirements of the final project). It is important to note that even though your output will be displayed on a VGA-compatible display, the controller you produce for this project is not a VGA graphics controller such as you would see on a PC or laptop.

The video controller for this project overlays an icon of your Bot (be creative here) onto a background generated from the BotSim world map. The icon moves through the RojoBot world based on the current{X,Y} coordinates of the Bot. These coordinates are returned by BotSim in the LocX and LocY registers as described in the *BotSim Functional Specification*. The orientation (which way the Bot is pointing) is returned in the BotInfo register.

Display Subsystem Architecture



A high-level block diagram for the display subsystem is shown above. The subsystem combines two bit maps to produce the image on the display. bot.v includes logic that produces a pixel stream (`world_pixel[1:0]`) representing an image of the RojobBot world. The 128 x 128 world is displayed as a 512 x 512 image on the VGA monitor: thus your implementation should represent each world location as a 4 x 4 pixel block on the screen. The two-bit world_pixel output indicates "ground", "obstruction", "black line", or "reserved" for each location on the screen.

The icon module stores a small (16 x 16) image of the RojoBot. The position of the Bot icon on the screen is determined by the values of the `LocX_reg` and `LocY_reg` outputs from the Bot. The icon

module produces a two-bit output that indicates "transparent" or one of 3 opaque colors. The colorizer module combines these two pixel steams into an 12-bit, 4096-color output. The RojoBot icon will appear in the foreground in front of the world background.

Display Timing Generator

The Display Timing Generator (DTG) generates the video raster signals Vertical Sync (`vert_sync`); Horizontal Sync (`horiz_sync`); `video_on`, which indicates the viewable region of the video screen; and `pixel_row` and `pixel_column`, which indicate the current vertical and horizontal pixel position on the display. Refer to pages 13 – 17 of the *Nexys4™ Board Reference Manual* for a description of the VGA signal timing.

The display image has 480 rows with 640 pixels (columns) in each row. Each pixel consists of three colors: Red, Green, and Blue. The Nexys4 supports 4096 colors (4 color bits per pixel for red, green, and blue for a total of 12-bit color). The pixel information is delivered to the monitor serially by row. The end of each row is indicated by a *horizontal sync pulse* of a particular length. At the end of all 480 rows, a longer, *vertical sync pulse* occurs. At the end of a horizontal sync pulse, the beam goes back to pixel 0 of the next row. At the end of a vertical pulse, the beam goes to pixel 0 in row 0. The VGA signals redraw the entire screen 60 times per second.

Because the image contains only 480 lines of the map image, the last 32 lines will not be visible. The maps provided for the project stay away from last 32 lines.

Icon Module

The icon module stores a 16 x 16 x 2 bitmap image of the Bot in read-only memory. This may be synthesized from HDL. Alternately, you can use the IP generator in Vivado or the ISE Core Generator. Each pixel in the image is stored as two bits in the ROM. If the bits are '00' it means 'transparent' – the background color (world) will show through. Values 01, 10, and 11 means display one of the 3 colors of the RojoBot image. Thus the icon need not be rectangular, and can have windows in it. If a 16 x 16 pixel icon appears too small for your satisfaction, then scale the image up for improved appearance. Show each pixel as a 2x2 or 4x4 block on the screen.

Two 10-bit inputs determine the vertical and horizontal position of the top-left corner of the icon. The icon module compares the values in these registers with the current row/column display pixel position generated by the DTG to determine whether or not to display the icon at the current raster position. The X and Y position input values are compared with the DTG row/column values. Thus, with the proper scaling between Bot's location (in Loc_X and Loc_Y), the icon can be positioned anywhere on the screen with a resolution of one pixel, vertically or horizontally. The RojoBot icon moves around its world as the Bot module changes the values of its position outputs. The colorizer determines the colors of the RojoBot icon.

The icon module should also show the RojoBot's orientation. The `BotInfo_reg` output provides eight headings: 0, 45, 90, 135, 180, 225, 270, and 315 degree encoded into 3 bits [0 (0) – 315 (7)]. Design the icon logic to change the orientation of the icon image in response to the `BotInfo_reg` input. An efficient implementation might use two bitmaps, one for 0, 90, 180, 270 degrees, and another for 45, 135, 225, 315 degrees, but you are free to use 8 icons (one per orientation) if you prefer. The Verilog code can be written to flip the image vertically and horizontally by changing the way the icon ROM row and column pixels are addressed.

Colorizer Module

The Colorizer module maps the two pixel streams from the Bot and Icon modules to screen color such that the RojoBot icon appears in the foreground while the world image forms the background. Its function is defined in the following table:

World[1:0]	Icon[1:0]	Color
00	00	Background
01	00	Black Line
10	00	Obstruction
11	00	Reserved
x	01	Icon color 1
x	10	Icon color 2
x	11	Icon color 3

The colors are 8-bit values of your choice – three bits each representing red, green and two-bits representing blue.

Make sure that the colorizer output is 000 (black) during the blanking interval, when the `video_on` signal from the DTG is zero.

MCCM (The Series 7 clock manager)

The 640 x 480 VGA timing is based on a 25MHz pixel clock. Use the Clocking Wizard in the Vivado's IP Integrator (Flow Manager/IP Catalog/FPGA Features and Design/Clocking Wizard) to generate a 25MHz clock from the 100MHz oscillator input. It is fairly straightforward to navigate though the tabs in the Clocking Wizard to create one or more clock outputs. Use the 25Mhz clock for the VGA controller modules, leaving the rest of the logic running at some other frequency. In my design I used the Clocking Wizard to generate a 75MHz clock for the system logic and a 25MHz clock to drive the VGA logic, but you are welcome to experiment. The Clocking Wizard, coupled with the Series 7 MCCM and PLL circuitry provides synchronized clocks at a variety of frequencies both faster and slower than the 100MHz oscillator input. The clock routing and MCCM and PLL clock generators provide are described in 7-Series FPGAs Clocking Resources (UG472)

The Clocking wizard will generate a number of files including VHDL and Verilog instantiation templates. Like your other Verilog modules, the MMCM created by the Clocking Wizard must be instantiated in your top level module. Navigate to the directory containing the Clocking Wizard output and find the `.veo` file. (`<project directory>/sources_1/ip/clk_wiz_0/clk_wiz_0.veo` in my project). Open the file with a text editor and cut/paste the instantiation code into your top level module. You may safely tie the `reset` input to 1'b0. In my project the instantiation looks like this:

```
 // instantiate the clock generator
wire   VGA_clk;
clk_wiz_0 CLKGEN
(
   // Clock in ports
    .clk_in1(clk),          // input clock is 100 MHz
    // Clock out ports
    .sysclk(sysclk),        // sysclk is 75 MHz
    .VGA_clk(VGA_clk),      // VGA_clk is 25MHz
    // Status and control signals
    .reset(1'b0),       // input reset
```

```
    .locked(locked)         // output locked
);
```

As you can see, I have connected the MMCM created by the Clocking Wizard to my clock signals. As with the world maps, Vivado will treat the MMCM as a black box and includes it in the synthesized result.

Given

The Project 2 release package includes several files to help you complete your display system design. Use these in addition to the Verilog you've already implemented for the project to complete the assignment. A description of the files is provided in *ECE 540 Project 2 List of Files*.

Project Tasks

- Design and implement the colorizer module.
- Design and implement the icon module.
- Generate and integrate the 25MHz clock using the IP Integrator Clocking Wizard.
- Integrate the display timing generator, icon, and colorizer modules into your system.
- Add VGA signal output ports to `nexys4fpga.v` and include the `nexys4fpga_withvideo.xdc` in your Vivado project.
- Check/fix your Project 2 application program to make sure it successfully traverses *world_map_loop*. This world map is more difficult to traverse than world_map but still consists of right turns.

NOTE: You will need to replace the world_map.ngc file with one of the other maps. The easiest way to do this is to delete the current world_map.ngc file from your project (right click on the file in the Hierarchy pane and Remove File from Project. Than Add the new world_map.ngc to the project in the same way you added all of the other files. You may leave world_map.v alone; it does not change.

- Check that your black line following algorithm correctly handles a right and left turns. *world_map_lr* is the map that you will use in your demo. You will need to replace the loop virtual world with the new one. The new virtual world should be named world_map like the others.

References

[1] *Digilent Nexys4 Board Reference Manual,* Digilent Inc.
[2] *Xilinx Vivado Software Manuals*
[3] *BotSim 2.0 Functional Specification (Revision 2.1),* by Roy Kravitz
[4] *BotSim 2.0 Theory of Operation (Revision 2.1),* by Roy Kravitz
[5] *ECE 540 Project 2 List of Files* (Revision 2.1), by Roy Kravitz
[6] *7-Series FPGAs Clocking Resources* (UG472)

<finis>