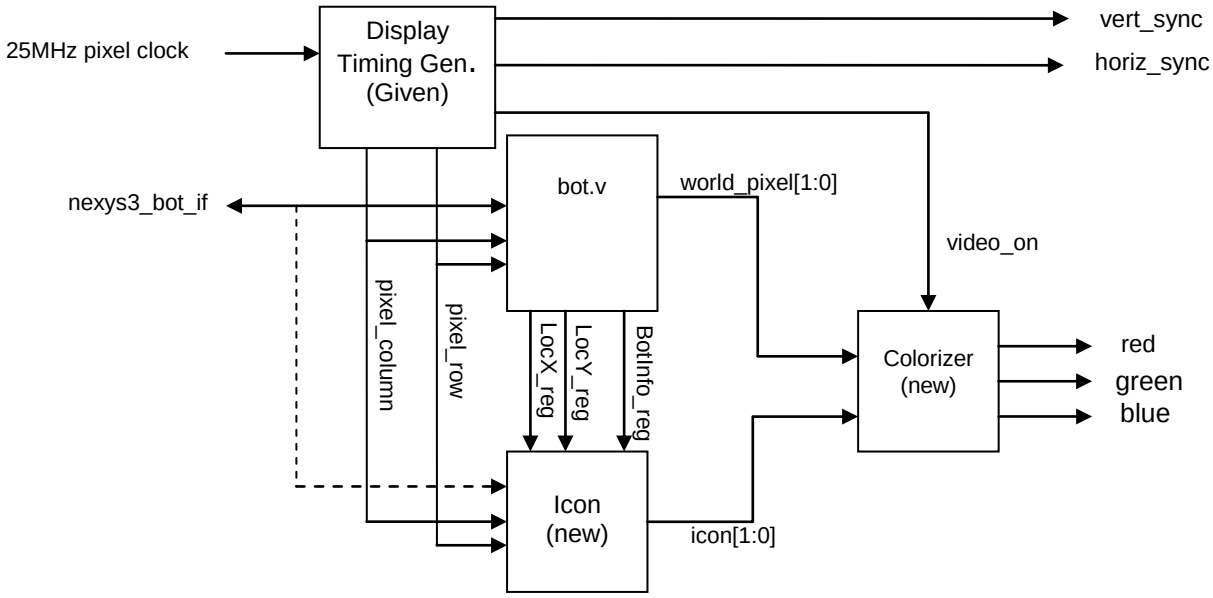
RojoBot World Video Controller

One of the primary outcomes for Project 1 is to give you experience designing and implementing video. For Project 1 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 this 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 monitor, 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 2.0 Functional Specification*. The orientation (which way the Bot is pointing) is returned in the BotInfo register.

Video Graphics Subsystem Architecture



A high-level block diagram for the video graphics subsystem is shown above. The subsystem combines two bit maps to produce the image on the monitor. 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 8-bit, 256-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 screen. Refer to pages 15 – 17 of the *Nexys3™ Board Reference Manual* for a description to VGA signal timing.

The VGA image has 480 rows with 640 pixels (columns) in each row. Each pixel consists of three colors: Red, Green, and Blue. The Nexys3 supports 256 colors (3 color bits per pixel for red and green, two color bits for blue). 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 VGA image displays only 480 lines of the map image, the last 32 lines will not be visible. The maps provided for Project 1 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, the Xilinx Core Generator (in the *ISE Design Tools*/*Tools* menu) may be used to create the ROM. 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 module compares the values in these registers with the current row/column raster 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, 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. 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:

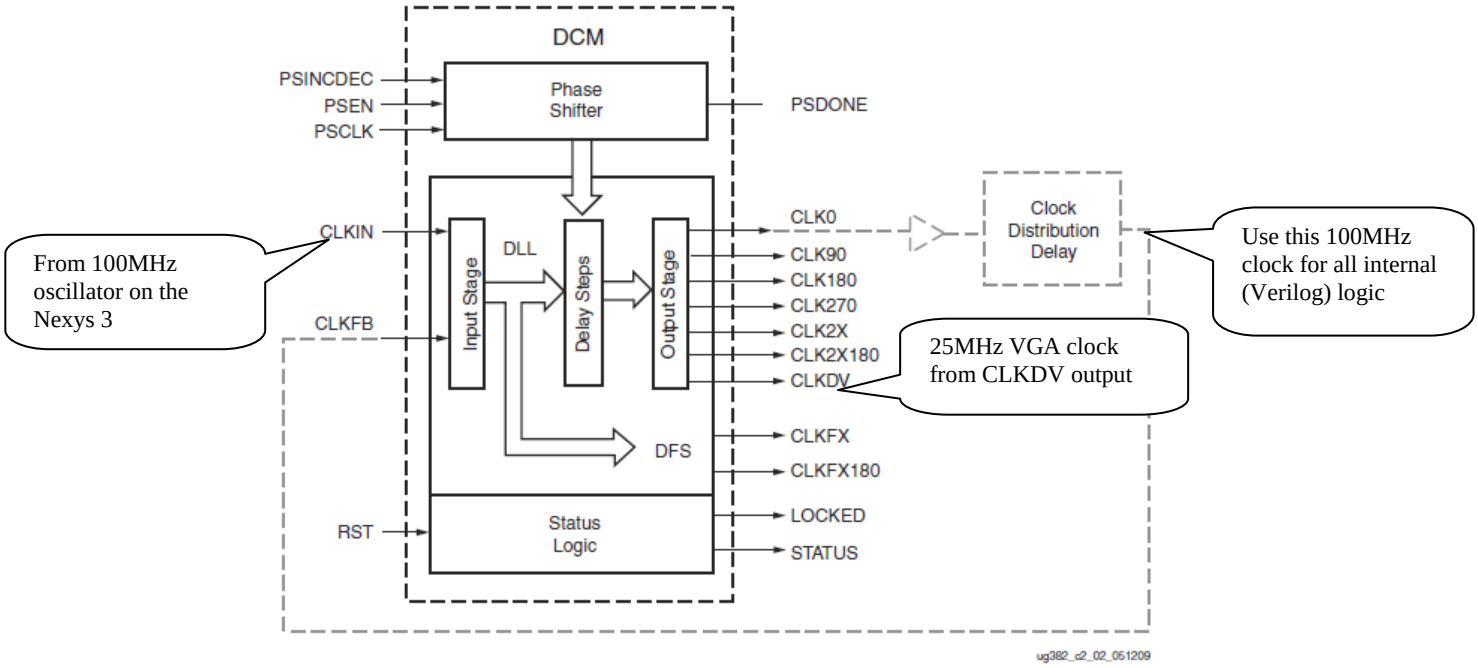
World[1:0]	Icon[1:0]	Color
00	00	Background
01	00	Black Line
10	00	Obstruction
11	00	Reserved
x	01	Icon color 1
x	10	Icon color 2
x	11	Icon color 3

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.

Digital Clock Manager (DCM)

The 640 x 480 VGA timing is based on a 25MHz pixel clock. Use a DCM to generate the 25MHz clock from the 100MHz oscillator input. A Verilog template is provided for you to use to instantiate a DCM in your HDL design. You should use the 25Mhz clock for the graphics modules only, leaving the rest of the logic running at 100MHz. When using this clocking method connect the internal 100MHz clock signal to the DCM output as shown below through a BUFG (global clock buffer). Otherwise, you may experience excessively long place & route run times, and excessive clock skew. The figure is copied from page 63 of the *Spartan-6 FPGA Clocking Resources Guide (UG382 v1.7)*.



You may also use the Xilinx Core Generator to create the 25MHz clock using the *Clocking Wizard*.

Given

The Project 1 release package includes several files to help you complete your VGA 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 *Project 1 List of Files*.

Project Tasks

- Design and implement the colorizer module.
- Design and implement the icon module.
- Integrate the DCM, display timing generator, icon, and colorizer modules into your system.
- Add VGA signal output ports to `nexys3fpga.v` and `nexys3fpga.ucf` (if necessary)
- Check/fix your Project 1 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.
- Check that your black line following algorithm correctly handles a right and left turning track by demonstrates the your Bot traverses *world_map_lr* correctly. *world_map_lr* is the map that you will use in your demo.

References

[1] *Digilent Nexys 3 Board Reference Manual*, Digilent Inc
[2] *Xilinx ISE Software Manuals* (ISE Design Tools→ Documentation →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] *Project 1 List of Files* (Revision 2.1), by Roy Kravitz

<finis>