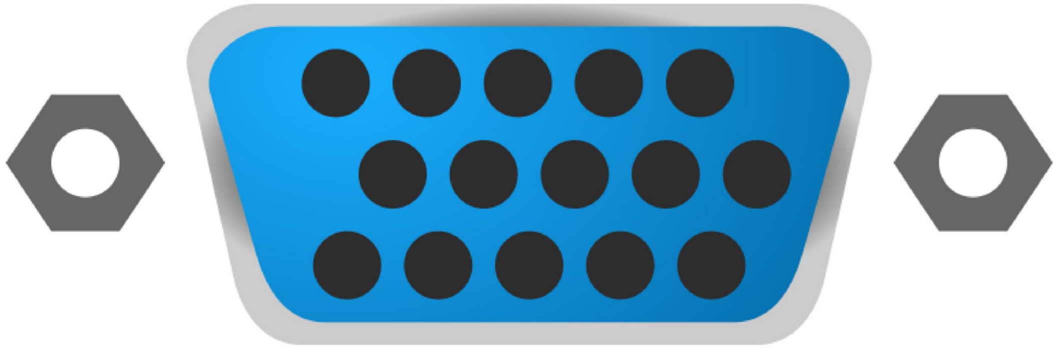
Embedded Thoughts

Driving a VGA Monitor Using an FPGA



Learning how to directly drive a VGA monitor with an FPGA opens up a window for many potential projects: video games, image processing, a terminal window for a custom processor, and many more. To get started, we will need to learn how to drive the necessary signals to display things on a VGA monitor. We will then use a test circuit to display some colors on the screen. In future posts I will detail how to design pixel generating circuits for displaying custom graphics and animations. More on that soon!

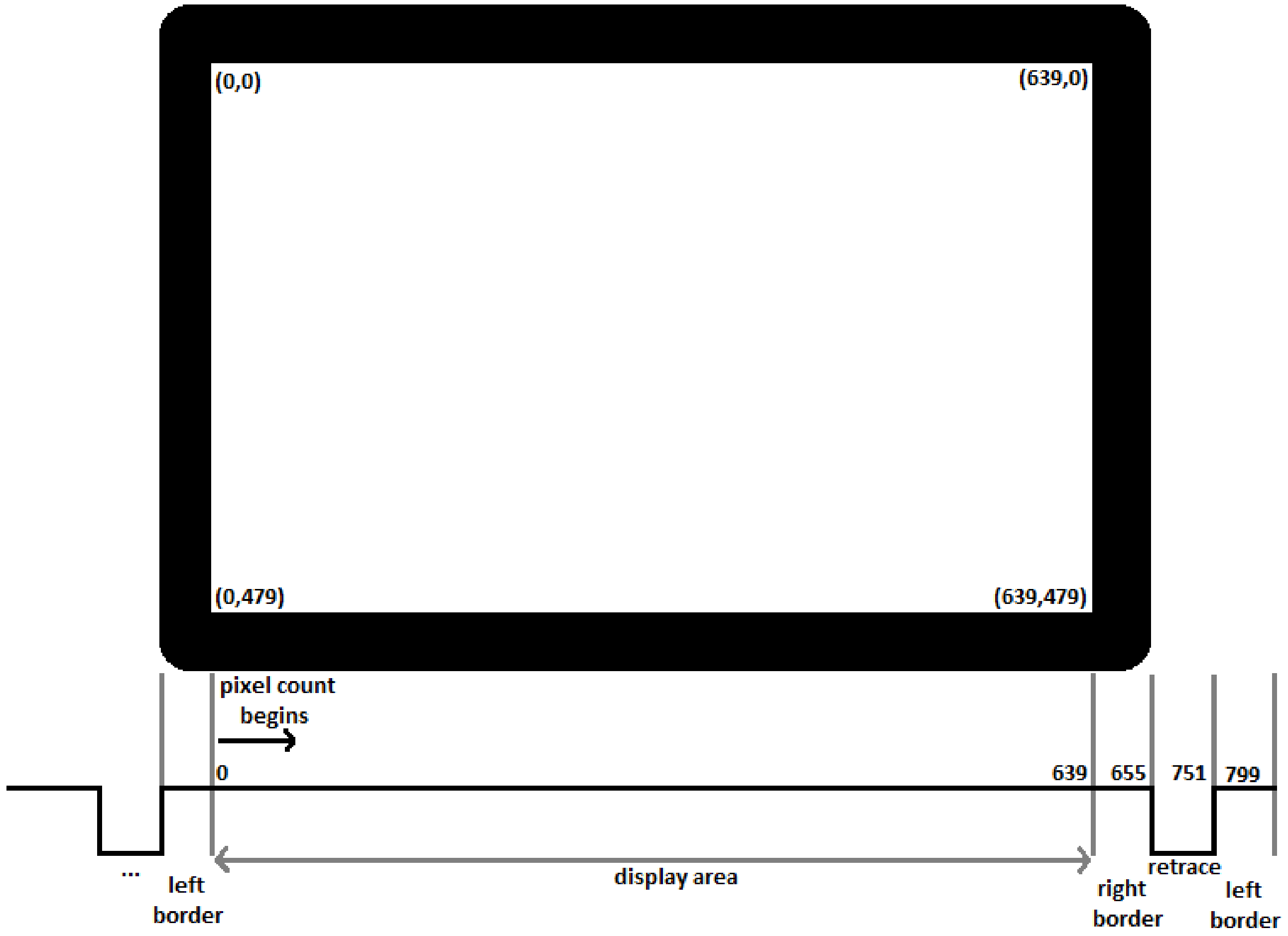
Computer monitors used to be bulky cathode ray tube (CRT) devices. VGA technology was developed with driving the physical CRT in mind, so knowing how that device works can be instructive in understanding why VGA signals are driven the way they are. That being said, computer monitors nowadays are LCD monitors without a CRT, yet the VGA interfaces for these monitors still use the same signals to display images on their screens. Instead of going into the details of CRTs in this post, we will instead inspect the necessary signals for a VGA monitor and their timing diagrams, and implement a synchronization circuit in Verilog HDL.

Keep in mind that different FPGA development boards have different color depth capabilities. I will focus here on using the Basys 2 which has 8-bit color and the Basys 3 which has 12-bit color. While I am covering the Basys 2 here, in the future I will focus on and use the more capable Basys 3 for my VGA projects.

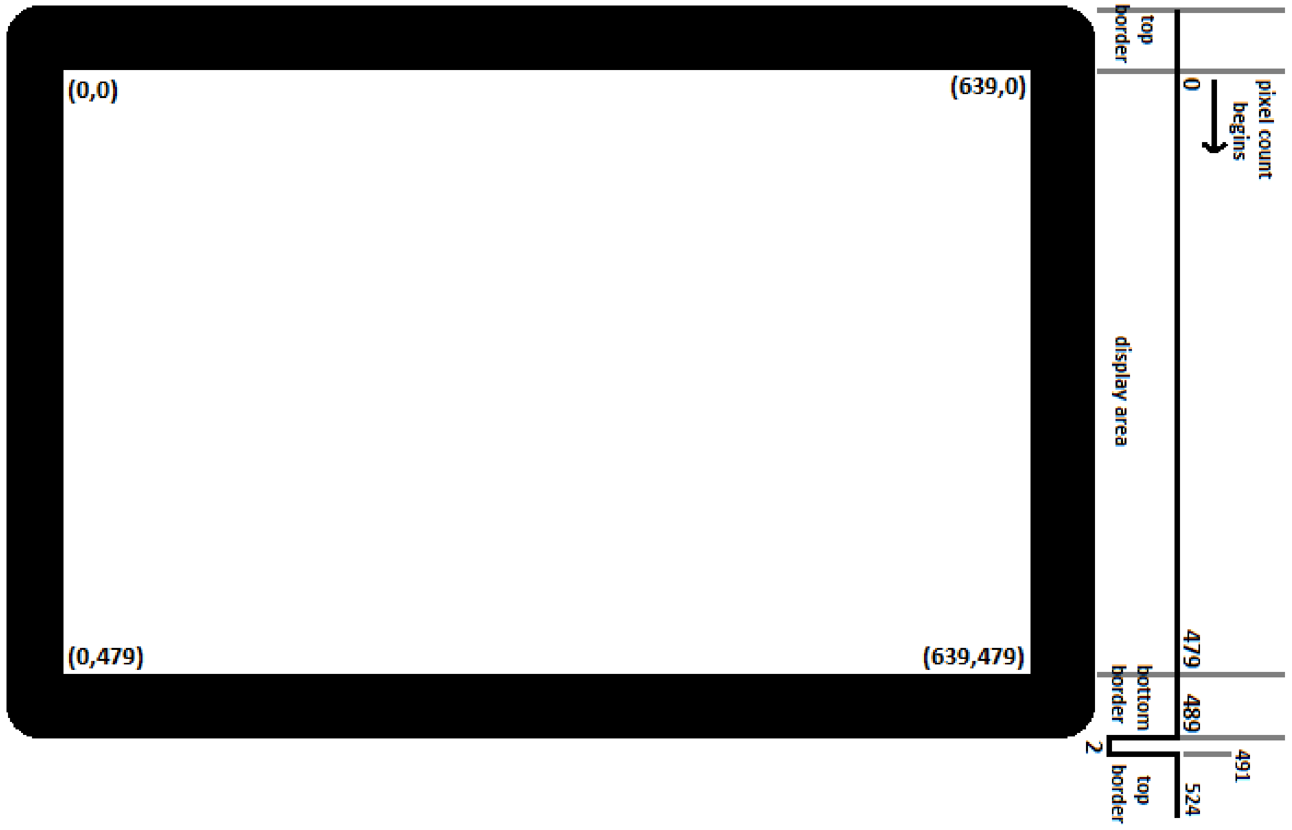
Each pixel on the monitor has a red, green, and blue color component. The red, green, and blue color signals from the VGA port to the monitor are analog and are generated by resistor ladder DACs on the FPGA board that receive input from dedicated FPGA IO lines. The Basys 2 board used eight IO lines to drive the color signals, and therefore uses an 8-bit color scheme, with 3 bits determining red and green color intensities, and 2 bits determining the blue color intensity. The Basys 3 has 12 IO lines to drive color signals, with 4 bits driving each respective RGB color intensity.

The VGA ports on both boards use two synchronization signals, one called hsync (horizontal sync) to specify the time taken to scan through a row of pixels, and the other called vsync (vertical sync) to specify the time taken to scan through an entire screen of pixel rows. Properly driving the hysnc and vsync signals will be our main focus. Once the VGA synchronization circuit is designed and tested, we can in the future abstract away the details and just use it as a ready made module for our VGA projects.

Now we will begin to talk about resolution and pixels. Keep in mind that the **display area** on the screen using VGA has a resolution of 640 by 480 pixels. The origin of the display area is (0,0) and is located in the upper left corner. The x dimension increases from left to right, while y increases from top to bottom. We will run our VGA circuit at a 25 MHz pixel rate, meaning that 25 million pixels will be scanned each second. The monitors usually include small black borders surrounding the display area.



We will divide each horizontal scan row into 800 pixels, starting at pixel 0 and ending at pixel 799. We will scan through each horizontal row of pixels at 25 MHz. The pixel count begins at the start of the display area, and runs through all 640 display pixels, followed by a 16 pixel right border. Next is a 96 “pixel” retrace, which is a delay artifact of CRT monitors that we must include, and finally a 48 pixel left border.



We will divide the vertical space into 525 horizontal rows, with the scanning through of all rows counting as one refresh of the screen.

The vertical pixel count starts in the display area and ends at pixel 479, continues into a 10 pixel bottom border, then a 2 pixel retrace, and finally ends after a 33 pixel top border.

The pixels in the border and retrace areas should be black, so we will generate a video_on signal that is asserted when the current pixel is in the horizontal and vertical display region. This signal will turn on our RGB color signals for the display pixels only.

To avoid noticeable flickering from the screen, a goal of a 60 Hz refresh rate is generally acceptable. If the VGA circuit is clocked at 25 MHz, then each pixel scan will have a period of 1/25 Mhz = 40 ns. If we multiply that by 800 pixels per row, and 525 rows per screen, we get around 1/60 seconds per screen refresh, which meets our goal.

Let’s take a look at the code for the Basys 2 Verilog HDL implementation:

```
module vga_sync

        (
                input wire clk, reset,
                output wire hsync, vsync, video_on, p_tick,
                output wire [9:0] x, y
        );

        // constant declarations for VGA sync parameters
        localparam H_DISPLAY       = 640; // horizontal display area
        localparam H_L_BORDER      =  48; // horizontal left border
        localparam H_R_BORDER      =  16; // horizontal right border
        localparam H_RETRACE       =  96; // horizontal retrace
        localparam H_MAX           = H_DISPLAY + H_L_BORDER + H_R_BORDER + H_RETRACE - 1;
        localparam START_H_RETRACE = H_DISPLAY + H_R_BORDER;
        localparam END_H_RETRACE   = H_DISPLAY + H_R_BORDER + H_RETRACE - 1;

        localparam V_DISPLAY       = 480; // vertical display area
        localparam V_T_BORDER      =  10; // vertical top border
        localparam V_B_BORDER      =  33; // vertical bottom border
        localparam V_RETRACE       =   2; // vertical retrace
        localparam V_MAX           = V_DISPLAY + V_T_BORDER + V_B_BORDER + V_RETRACE - 1;
        localparam START_V_RETRACE = V_DISPLAY + V_B_BORDER;
        localparam END_V_RETRACE   = V_DISPLAY + V_B_BORDER + V_RETRACE - 1;

        // mod-2 counter to generate 25 MHz pixel tick
        reg pixel_reg;
        wire pixel_next, pixel_tick;

        always @(posedge clk)
                pixel_reg <= pixel_next;

        assign pixel_next = ~pixel_reg; // next state is complement of current
assign pixel_tick = (pixel_reg == 0); // assert tick half of the time

// registers to keep track of current pixel location
reg [9:0] h_count_reg, h_count_next, v_count_reg, v_count_next;

// register to keep track of vsync and hsync signal states
reg vsync_reg, hsync_reg;
wire vsync_next, hsync_next;

// infer registers
always @(posedge clk, posedge reset)
        if(reset)
            begin
            v_count_reg <= 0;
            h_count_reg <= 0;
            vsync_reg   <= 0;
            hsync_reg   <= 0;
            end
        else
            begin
            v_count_reg <= v_count_next;
            h_count_reg <= h_count_next;
            vsync_reg   <= vsync_next;
            hsync_reg   <= hsync_next;
            end

// next-state logic of horizontal vertical sync counters
always @*
        begin
        h_count_next = pixel_tick ?
                       h_count_reg == H_MAX ? 0 : h_count_reg + 1
                       : h_count_reg;

        v_count_next = pixel_tick && h_count_reg == H_MAX ?
                       (v_count_reg == V_MAX ? 0 : v_count_reg + 1)
                       : v_count_reg;
```

```
                end

        // hsync and vsync are active low signals
        // hsync signal asserted during horizontal retrace
        assign hsync_next = h_count_reg >= START_H_RETRACE
                            && h_count_reg <= END_H_RETRACE;

        // vsync signal asserted during vertical retrace
        assign vsync_next = v_count_reg >= START_V_RETRACE
                            && v_count_reg <= END_V_RETRACE;

        // video only on when pixels are in both horizontal and vertical display region
        assign video_on = (h_count_reg < H_DISPLAY)
                           && (v_count_reg < V_DISPLAY);

        // output signals
        assign hsync  = hsync_reg;
        assign vsync  = vsync_reg;
        assign x      = h_count_reg;
        assign y      = v_count_reg;
        assign p_tick = pixel_tick;
endmodule
```

The outputs used for driving the VGA monitor are hsync and vsync. To let exterior graphics generation circuits know when to output color signals through the VGA port for the display area, the video_on signal is output. The current pixel location on the screen, x and y, are output to let the external graphics circuits know where the current scan pixel is, as well as p_tick which is output to signal when the pixel location has changed.

We then have constant declarations for the different pixel area widths on the screen, as well as some combinations of these values, such as the max horizontal and vertical pixel values, as well as the pixel values for the start and end of the retrace sections.

The Basys 2 board usually runs at 50 MHz, so we implement a mod-2 counter to divide the clock speed down to 25 MHz, and assign pixel_tick to be asserted at 25 MHz.

The v_count, and h_count registers keep track of the vertical and horizontal pixel values, while the v_sync and h_sync registers keep track of the vsync and hsync output signal states. The next state logic for the h_count register increments the value when the 25MHz pixel_tick signal is asserted, until it reaches the max horizontal pixel value where it is reset to 0. The next state logic for the v_count register increments the value when pixel_tick is asserted and when the h_count register is at the max value, that is when a pixel row is finished and the next row down should be scanned through. The v_count register is also reset to 0 after it reaches its maximum value.

The hsync and vsync signals for the VGA are active low, so their corresponding state registers are asserted during the retrace periods.

Next let's consider a test circuit that we can use on the Basys 2 that uses the eight onboard switches to output different colors on the VGA monitor.

```
module vga_test
        (
                input wire clk, reset,
                input wire [7:0] sw,
                output wire hsync, vsync,
                output wire [7:0] rgb
        );

        // register for Basys 2 8-bit RGB DAC
        reg [7:0] rgb_reg;

        // video status output from vga_sync to tell when to route out rgb signal to DAC
        wire video_on;

        // instantiate vga_sync
        vga_sync vga_sync_unit (.clk(clk), .reset(reset), .hsync(hsync), .vsync(vsync),
                                 .video_on(video_on), .p_tick(), .x(), .y());

        // rgb buffer
        always @(posedge clk, posedge reset)
        if (reset)
           rgb_reg <= 0;
        else
           rgb_reg <= sw;

        // output
        assign rgb = (video_on) ? rgb_reg : 8'b0;
endmodule
```

When the video_on signal is asserted from the vga_sync_unit, we route the states of the input switches to rgb_reg, which is tied to the rgb color signal outputs.

```
# clock pins for Basys2 Board
NET "clk" LOC = "B8"; # Bank = 0, Signal name = MCLK

NET "reset" LOC = "A7";  # Bank = 1, Signal name = BTN3

# Pin assignment for SWs
NET "sw[7]" LOC = "N3";  # Bank = 2, Signal name = SW7
NET "sw[6]" LOC = "E2";  # Bank = 3, Signal name = SW6
NET "sw[5]" LOC = "F3";  # Bank = 3, Signal name = SW5
NET "sw[4]" LOC = "G3";  # Bank = 3, Signal name = SW4
NET "sw[3]" LOC = "B4";  # Bank = 3, Signal name = SW3
NET "sw[2]" LOC = "K3";  # Bank = 3, Signal name = SW2
NET "sw[1]" LOC = "L3";  # Bank = 3, Signal name = SW1
NET "sw[0]" LOC = "P11";  # Bank = 2, Signal name = SW0

### Pin assignment for VGA
NET "hsync"   LOC = "J14"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = HSYNC
NET "vsync"   LOC = "K13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = VSYNC

NET "rgb[7]"   LOC = "F13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED2
NET "rgb[6]"   LOC = "D13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED1
NET "rgb[5]"   LOC = "C14"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = RED0
NET "rgb[4]" LOC = "G14"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN2
NET "rgb[3]" LOC = "G13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN1
NET "rgb[2]" LOC = "F14"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = GRN0
NET "rgb[1]"  LOC = "J13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = BLU2
NET "rgb[0]"  LOC = "H13"  | DRIVE = 2  | PULLUP ; # Bank = 1, Signal name = BLU1
```

The UCF used in ISE is shown above. For the 8 switches from left to right, the rgb signals are as follows: r2, r1, r0, g2, g1, g0, b1, b0. The three red and green signals are inputs to 3-bit DACs, and specify a range of 8 color intensities for red and green. Likewise, the 2 blue bits specify a range of 4 blue color intensities.

In order to make the above HDL work for the Basys 3 board, we must consider that it instead uses 12 bits to specify color, and also has a 100 MHz crystal oscillator on board. We will then need to widen the rgb color register and output port, as well as change the mod-2 counter to a mod-4 counter to divide down 100 MHz to 25 MHz for the pixel_tick signal.

Below is the modified HDL for the Basys 3, along with the XDC configuration file. Here (https://www.youtube.com/watch?v=6_GxkslqbcU) is a video that I found helpful for getting started using the Basys 3 with Vivado.

```
module vga_sync

        (
                input wire clk, reset,
                output wire hsync, vsync, video_on, p_tick,
                output wire [9:0] x, y
        );

        // constant declarations for VGA sync parameters
        localparam H_DISPLAY       = 640; // horizontal display area
        localparam H_L_BORDER      =  48; // horizontal left border
        localparam H_R_BORDER      =  16; // horizontal right border
        localparam H_RETRACE       =  96; // horizontal retrace
        localparam H_MAX           = H_DISPLAY + H_L_BORDER + H_R_BORDER + H_RETRACE - 1;
        localparam START_H_RETRACE = H_DISPLAY + H_R_BORDER;
        localparam END_H_RETRACE   = H_DISPLAY + H_R_BORDER + H_RETRACE - 1;

        localparam V_DISPLAY       = 480; // vertical display area
        localparam V_T_BORDER      =  10; // vertical top border
        localparam V_B_BORDER      =  33; // vertical bottom border
        localparam V_RETRACE       =   2; // vertical retrace
        localparam V_MAX           = V_DISPLAY + V_T_BORDER + V_B_BORDER + V_RETRACE - 1;
        localparam START_V_RETRACE = V_DISPLAY + V_B_BORDER;
        localparam END_V_RETRACE   = V_DISPLAY + V_B_BORDER + V_RETRACE - 1;

        // mod-4 counter to generate 25 MHz pixel tick
        reg [1:0] pixel_reg;
        wire [1:0] pixel_next;
        wire pixel_tick;

        always @(posedge clk, posedge reset)
                if(reset)
                  pixel_reg <= 0;
                else
                pixel_reg <= pixel_next;

        assign pixel_next = pixel_reg + 1; // increment pixel_reg

        assign pixel_tick = (pixel_reg == 0); // assert tick 1/4 of the time

        // registers to keep track of current pixel location
        reg [9:0] h_count_reg, h_count_next, v_count_reg, v_count_next;

        // register to keep track of vsync and hsync signal states
        reg vsync_reg, hsync_reg;
        wire vsync_next, hsync_next;

        // infer registers
        always @(posedge clk, posedge reset)
                if(reset)
                    begin
                    v_count_reg <= 0;
                    h_count_reg <= 0;
                    vsync_reg   <= 0;
                    hsync_reg   <= 0;
                    end
                else
                    begin
                    v_count_reg <= v_count_next;
                    h_count_reg <= h_count_next;
                    vsync_reg   <= vsync_next;
                    hsync_reg   <= hsync_next;
                    end

        // next-state logic of horizontal vertical sync counters
        always @*
                begin
                h_count_next = pixel_tick ?
                               h_count_reg == H_MAX ? 0 : h_count_reg + 1
                               : h_count_reg;
                v_count_next = pixel_tick && h_count_reg == H_MAX ?
                               (v_count_reg == V_MAX ? 0 : v_count_reg + 1)
                               : v_count_reg;
                end

        // hsync and vsync are active low signals
        // hsync signal asserted during horizontal retrace
        assign hsync_next = h_count_reg >= START_H_RETRACE
                            && h_count_reg <= END_H_RETRACE;

        // vsync signal asserted during vertical retrace
        assign vsync_next = v_count_reg >= START_V_RETRACE
                            && v_count_reg <= END_V_RETRACE;

        // video only on when pixels are in both horizontal and vertical display region
        assign video_on = (h_count_reg < H_DISPLAY)
                          && (v_count_reg < V_DISPLAY);

        // output signals
        assign hsync  = hsync_reg;
        assign vsync  = vsync_reg;
        assign x      = h_count_reg;
        assign y      = v_count_reg;
        assign p_tick = pixel_tick;
endmodule
```

```
module vga_test
	(
		input wire clk, reset,
		input wire [11:0] sw,
		output wire hsync, vsync,
		output wire [11:0] rgb
	);

	// register for Basys 2 8-bit RGB DAC
	reg [11:0] rgb_reg;

	// video status output from vga_sync to tell when to route out rgb signal to DAC
	wire video_on;

	// instantiate vga_sync
	vga_sync vga_sync_unit (.clk(clk), .reset(reset), .hsync(hsync), .vsync(vsync),
	                        .video_on(video_on), .p_tick(), .x(), .y());

	// rgb buffer
	always @(posedge clk, posedge reset)
	if (reset)
	    rgb_reg <= 0;
	else
	    rgb_reg <= sw;

	// output
	assign rgb = (video_on) ? rgb_reg : 12'b0;
endmodule
```

```
# Clock signal
set_property PACKAGE_PIN W5 [get_ports clk]
        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

# Switches
set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]

#VGA Connector
set_property PACKAGE_PIN G19 [get_ports {rgb[8]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[8]}]
set_property PACKAGE_PIN H19 [get_ports {rgb[9]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[9]}]
set_property PACKAGE_PIN J19 [get_ports {rgb[10]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[10]}]
set_property PACKAGE_PIN N19 [get_ports {rgb[11]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[11]}]
set_property PACKAGE_PIN N18 [get_ports {rgb[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[0]}]
set_property PACKAGE_PIN L18 [get_ports {rgb[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[1]}]
set_property PACKAGE_PIN K18 [get_ports {rgb[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[2]}]
set_property PACKAGE_PIN J18 [get_ports {rgb[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[3]}]
set_property PACKAGE_PIN J17 [get_ports {rgb[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[4]}]
set_property PACKAGE_PIN H17 [get_ports {rgb[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[5]}]
set_property PACKAGE_PIN G17 [get_ports {rgb[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[6]}]
set_property PACKAGE_PIN D17 [get_ports {rgb[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {rgb[7]}]
set_property PACKAGE_PIN P19 [get_ports hsync]
        set_property IOSTANDARD LVCMOS33 [get_ports hsync]
set_property PACKAGE_PIN R19 [get_ports vsync]
        set_property IOSTANDARD LVCMOS33 [get_ports vsync]

set_property PACKAGE_PIN U18 [get_ports reset]
        set_property IOSTANDARD LVCMOS33 [get_ports reset]
```

Above is a video demonstrating some colors on my VGA monitor using the Basys 3. Remember that the Basys 2 has 8-bit color, meaning that there are 2^8 = 256 possible colors that can be displayed, while the Basys 3 has 12-bit color, with 2^{12} = 4096 possible colors. You don't have to use all of the color bits that a given FPGA board supplies you, and if you wish you can hold some bits constant and reduce the memory requirements for an image at the cost of reduced color depth.

In my next post we will consider how to store image data in an FPGA's Block RAM, and design a pixel generation circuit that will move a video game sprite around on the screen :).

To see a complete FPGA video game project using VGA and block RAM to store sprites, click here (https://embeddedthoughts.com/2016/12/09/yoshis-nightmare-fpga-based-video-game/).

July 29, 2016December 30, 2016 Embedded Thoughts 8 bit color Artix 7 Basys 2 Basys 3 FPGA hsync Spartan 3 synchronization Verilog VGA VGA Monitor vsync Xilinx

5 thoughts on “Driving a VGA Monitor Using an FPGA”

1. **ramesh** says:
   February 13, 2018 at 12:18 am
   hello sir,
   when i implemented this module on nexys 4 DDR artix7 FPGA, on the monitor screen “connection not supported” message displayed.
   kindly please help to resolve this issue

   ↳ Reply
2. Pingback: [Перевод] Решение FizzBuzz на FPGA с генерацией видео – CHEPA website
3. **faiz** says:
   May 27, 2019 at 9:13 pm
   image displaying plz

   ↳ Reply
4. Pingback: A simple universal 800×600 VGA signal generation circuit – nerdhut
5. **Himangshu Choudhury** says:
   June 26, 2020 at 8:46 am
   The STARTV_RETRACE and END_V RETRACE are wrong please rectify the limit should be 490-491. whereas it is 513-514

   ↳ Reply