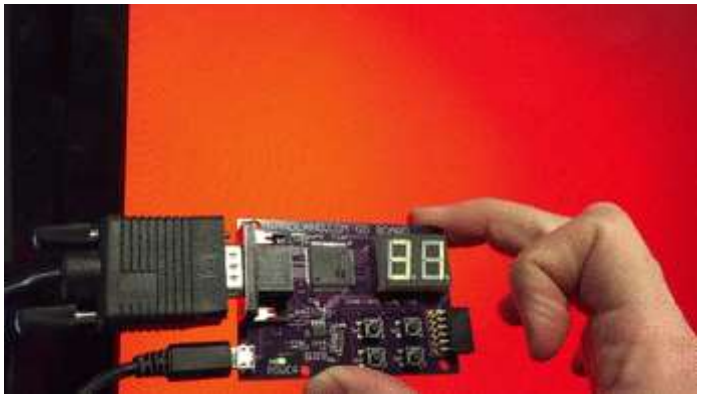
Go Board – VGA Introduction

Learn how VGA works, display test patterns to your VGA monitor

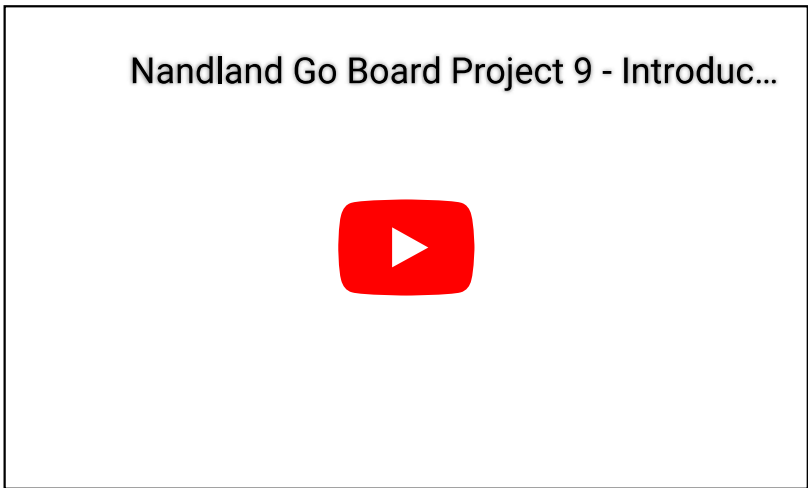
VGA stands for Video Graphics Array and is a very common display interface. VGA was first introduced in 1987 and it is still widely supported today. Nearly 30 years for VGA's prominence, very impressive! This project will get you acquainted with VGA and show you how the Go Board is able to drive your desktop monitor directly.

This is the goal outcome for this project:



Go Board drives VGA Display

Should you prefer to follow along with a YouTube video, well you can do that too! Please subscribe to my channel.



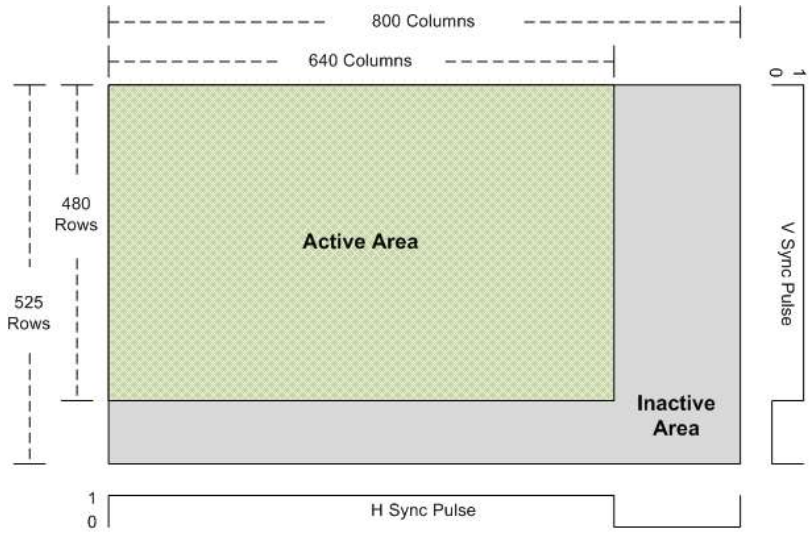
What is VGA? How Does it Work?

Today VGA is slowly being replaced by other displays such as DVI, HDMI, and DisplayPort to name a few. The reason that the Go Board uses VGA as opposed to something like HDMI is that VGA is easier to deal with. The HDMI specification is very detailed, so interfacing directly from an FPGA to an HDMI display is more of a challenge than a VGA display. It's best to learn the concepts about displays first using VGA.

The main difference between VGA and modern display interfaces is analog vs. digital. Analog means that the

color driven to the pixel is determined by a voltage level. Digital means that the color driven to the pixel is determined by a binary code.

In addition to the analog voltage level, there's some control signals, usually referred to as syncs that tell the monitor when to drive each pixel. These are Horizontal Sync and Vertical Sync. Let's look at a single image frame.

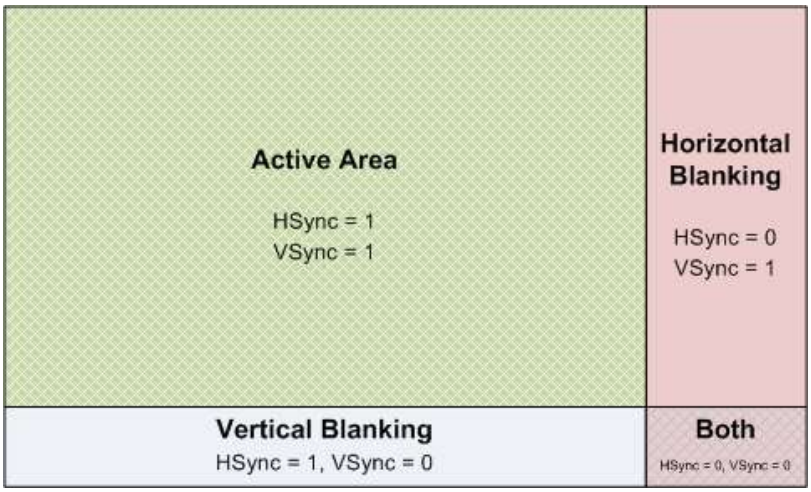


VGA Active vs. Inactive Area with 25 MHz Clock

In the image above there is the green rectangle area which is the Active Area. This is the part of that frame that will actually be displayed on the monitor. When the image is displayed on the monitor, the pixels are actually displayed from left to right, top to bottom. Therefore, the top-left corner is the first pixel displayed and the bottom-

right corner is the last pixel displayed. Because the pixels get displayed so quickly, the human eye is unable to see the it, and it looks like the entire image is drawn at once.

The two control signals **H Sync** and **V Sync** are also shown in the image above. HSync is high when you are in the active area of the frame horizontally. VSync is high when you are in the active area of the frame vertically. Therefore, if HSync *and* VSync are both high, then we are in the active area of the frame. If just HSync is high, then we are in the bottom inactive area of the frame. This is called the vertical blanking space. If just VSync is high, then we are in the horizontal blanking area of the frame. This is perhaps shown best in the image below:



Active Area, Horizontal Blanking, Vertical Blanking

For this project, we will be displaying video data at 60 Hz, meaning 60 frames per second. This project will be using an active area of 640 by 480, meaning 640 columns and 480 rows in the active area. There are a total of 794 columns and 525 rows in the total frame. This is a total of 794*525=416,850 pixels. Remember that the Go Board uses a 25 MHz clock. So if you draw one pixel per clock cycle, how much time does it take to draw an entire frame? 416850 / 25000000 = 0.01667 seconds. Remember that we are displaying frames at 60 Hz, and 1/60Hz = 0.01667 seconds per frame, so the math works! The total number of pixels in the frame allows us to draw one frame every 16.67 milliseconds. **The main clock on**

the Go Board was specifically chosen to works well when driving a VGA display.

Now, it would be nice if all we had to do was drive HSync and VSync as shown in the image above. That would make the most sense. But unfortunately things aren't that simple. Remember that originally VGA monitors were Cathode Ray Tubes (CRTs). Due to the way CRT displays work, we have to deal with the **Front Porch** and **Back Porch**. The front porch and back porch allow the active area to be shifted around your VGA monitor. You can think of them as modifications to your HSync and VSync pulses. For the Go Board, I show the times for each the front and back porch that work well. (Note that these values are modified slightly from here).

VGA Timings using 25 MHz Clock		
Line/Frame Part	Horizontal Pixels	Vertical Lines
Whole Area	800	525
Visible Area	640	480
Front Porch	18	10
Sync Pulse	92	2
Back Porch	50	33

The timings above seem to work best for me. If you get better results with different values, please let everyone know in the comments below.

How Do Colors Work? Converting Digital Outputs to Analog

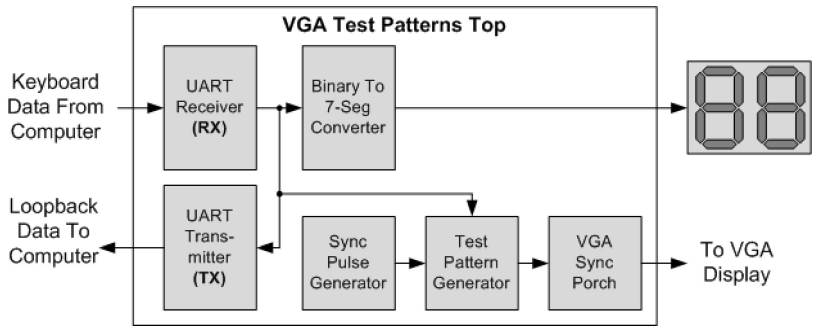
The Go Board does not have an analog output. Remember that VGA requires an analog voltage that represents the color value driven on the Red, Green, and Blue inputs. An

analog voltage of 0.7 Volts represents full on for that particular color. So for example if you tie Red to 0.7 V, Blue to 0.0 V and Green to 0.0 V, then the monitor will display a purely red image (assuming you're driving the Syncs correctly as well). The way the Go Board converts its digital outputs to an analog voltage is by a resistor divider. If you're really interested in how this works, check out the schematic.

What you really need to know is that on the Go Board, each color (R, G, B) has 3-bits associated to it. This gives us a total of 512 possible colors. If you want to set the image to a pure red image, set the 3-bit RGB Outputs to 111, 000, and 000 respectively, for full red intensity. On the Go Board, bit 2 is the MSb, bit 0 is the LSb, so half Red intensity would be 100 for example.

Project Description

Create a project that outputs test patterns to the VGA port on the Go Board. The test patterns should be selectable by the computer keyboard. Use the UART and Binary-To-7-Segment modules from the previous project.



Go Board – VGA Project Block Diagram

Look at the block diagram above. Now things are starting to get really interesting! We have a lot of modules that are all working together. The projects are building on each other to become more complicated and exercise more functionality on the FPGA. The code to drive the VGA display is linked in the EDA Playground environment

below. The testbench only exercises the Sync Pulse Generator and the Test Pattern Generator. I included the necessary code for the VGA Front and Back Porch modifications in the EDA Playground for you. Once you have a handle on the VGA code, take a look at the top level module below to see how everything gets wired up. The results of the programmed Go Board are shown in the GIF below. I also included a short description of each of the new VGA modules in the block diagram.

Sync Pulse Generator:

The purpose of this module is to generate the sync pulses for the active and the inactive area using a 25 MHz clock. You should have some input parameters (in Verilog) or generics (in VHDL) that set the active and total number of rows and columns. This way, if the frame size changes, only the input values need to be updated. This module should simply generate free-running HSync and VSync that downstream modules can use to know where they are in the frame.

Test Pattern Generator:

The test pattern generator should be looking at where you are in your image based on the HSync and VSync inputs from the Sync Generator. Based on your location, you can keep a Column Counter and Row Counter. When you reach certain parts of the image, you can drive the pixel values differently, generating different colors. The actual test pattern that gets selected should be driven by some input to the module.

VGA Sync Porch:

This module modifies the sync pulses to add in the front and back porches required to properly drive the VGA display. The output of this module should the actual VGA signals that go right to the VGA pins.

Sync to Count:

Although this module is not shown in the block diagram above, I did add it to the EDA Playground environment. The purpose of this module is to simply output the Column and Row Index of the current pixel that we are on. This is useful information to know both in the Test Pattern Generator and in the VGA Sync Porch modules, so I chose to write the module once and instantiate it in two places. Reuse FTW.

VGA Test Pattern Code in VHDL

VGA Test Pattern Code in Verilog

And now the top level code that instantiates everything:

VHDL Code – VGA_Test_Patterns_Top.vhd:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity VGA_Test_Patterns_Top is
5    port (
6      -- Main Clock (25 MHz)
7      i_Clk         : in std_logic;
8
9      -- UART Data
10     i_UART_RX : in  std_logic;
11     o_UART_TX : out std_logic;
```

```
12
13      -- Segment1 is upper digit, Segmen
14      o_Segment1_A : out std_logic;
15      o_Segment1_B : out std_logic;
16      o_Segment1_C : out std_logic;
17      o_Segment1_D : out std_logic;
18      o_Segment1_E : out std_logic;
19      o_Segment1_F : out std_logic;
20      o_Segment1_G : out std_logic;
21
22      o_Segment2_A : out std_logic;
23      o_Segment2_B : out std_logic;
24      o_Segment2_C : out std_logic;
25      o_Segment2_D : out std_logic;
26      o_Segment2_E : out std_logic;
27      o_Segment2_F : out std_logic;
28      o_Segment2_G : out std_logic;
29
30      -- VGA
31      o_VGA_HSync : out std_logic;
32      o_VGA_VSync : out std_logic;
33      o_VGA_Red_0 : out std_logic;
34      o_VGA_Red_1 : out std_logic;
35      o_VGA_Red_2 : out std_logic;
36      o_VGA_Grn_0 : out std_logic;
37      o_VGA_Grn_1 : out std_logic;
38      o_VGA_Grn_2 : out std_logic;
39      o_VGA_Blu_0 : out std_logic;
40      o_VGA_Blu_1 : out std_logic;
41      o_VGA_Blu_2 : out std_logic
42      );
43  end entity VGA_Test_Patterns_Top;
44
45  architecture RTL of VGA_Test_Patterns_
46
47    signal w_RX_DV     : std_logic;
48    signal w_RX_Byte   : std_logic_vectc
49    signal w_TX_Active : std_logic;
50    signal w_TX_Serial : std_logic;
51
52    signal w_Segment1_A, w_Segment2_A :
53    signal w_Segment1_B, w_Segment2_B :
54    signal w_Segment1_C, w_Segment2_C :
55    signal w_Segment1_D, w_Segment2_D :
56    signal w_Segment1_E, w_Segment2_E :
57    signal w_Segment1_F, w_Segment2_F :
58    signal w_Segment1_G, w_Segment2_G :
59
60    -- VGA Constants to set Frame Size
61    constant c_VIDEO_WIDTH : integer :=
62    constant c_TOTAL_COLS  : integer :=
```

```
63    constant c_TOTAL_ROWS  : integer :=
64    constant c_ACTIVE_COLS : integer :=
65    constant c_ACTIVE_ROWS : integer :=
66
67    signal r_TP_Index        : std_logic
68
69    -- Common VGA Signals
70    signal w_HSync_VGA       : std_logic
71    signal w_VSync_VGA       : std_logic
72    signal w_HSync_Porch     : std_logic
73    signal w_VSync_Porch     : std_logic
74    signal w_Red_Video_Porch : std_logic
75    signal w_Grn_Video_Porch : std_logic
76    signal w_Blu_Video_Porch : std_logic
77
78    -- VGA Test Pattern Signals
79    signal w_HSync_TP     : std_logic;
80    signal w_VSync_TP     : std_logic;
81    signal w_Red_Video_TP : std_logic_ve
82    signal w_Grn_Video_TP : std_logic_ve
83    signal w_Blu_Video_TP : std_logic_ve
84
85  begin
86
87    UART_RX_Inst : entity work.UART_RX
88      generic map (
89        g_CLKS_PER_BIT => 217)
90      port map (
91        i_Clk       => i_Clk,
92        i_RX_Serial => i_UART_RX,
93        o_RX_DV     => w_RX_DV,
94        o_RX_Byte   => w_RX_Byte);
95
96
97    -- Creates a simple loopback to test
98    UART_TX_Inst : entity work.UART_TX
99      generic map (
100       g_CLKS_PER_BIT => 217)
101     port map (
102       i_Clk       => i_Clk,
103       i_TX_DV     => w_RX_DV,
104       i_TX_Byte   => w_RX_Byte,
105       o_TX_Active => w_TX_Active,
106       o_TX_Serial => w_TX_Serial,
107       o_TX_Done   => open
108       );
109
110   -- Drive UART line high when transmi
111   o_UART_TX <= w_TX_Serial when w_TX_A
112       i_Binary_Num => w_RX_Byte(7 dowr
113       o_Segment_A  => w_Segment1_A,
```

```
114         o_Segment_B  => w_Segment1_B,
115         o_Segment_C  => w_Segment1_C,
116         o_Segment_D  => w_Segment1_D,
117         o_Segment_E  => w_Segment1_E,
118         o_Segment_F  => w_Segment1_F,
119         o_Segment_G  => w_Segment1_G
120         );
121 
122     o_Segment1_A <= not w_Segment1_A;
123     o_Segment1_B <= not w_Segment1_B;
124     o_Segment1_C <= not w_Segment1_C;
125     o_Segment1_D <= not w_Segment1_D;
126     o_Segment1_E <= not w_Segment1_E;
127     o_Segment1_F <= not w_Segment1_F;
128     o_Segment1_G <= not w_Segment1_G; --
129         i_Binary_Num => w_RX_Byte(3 dowr
130         o_Segment_A  => w_Segment2_A,
131         o_Segment_B  => w_Segment2_B,
132         o_Segment_C  => w_Segment2_C,
133         o_Segment_D  => w_Segment2_D,
134         o_Segment_E  => w_Segment2_E,
135         o_Segment_F  => w_Segment2_F,
136         o_Segment_G  => w_Segment2_G
137         );
138 
139     o_Segment2_A <= not w_Segment2_A;
140     o_Segment2_B <= not w_Segment2_B;
141     o_Segment2_C <= not w_Segment2_C;
142     o_Segment2_D <= not w_Segment2_D;
143     o_Segment2_E <= not w_Segment2_E;
144     o_Segment2_F <= not w_Segment2_F;
145     o_Segment2_G <= not w_Segment2_G;
146 
147     -------------------------------------
148     -- VGA Test Patterns
149     -------------------------------------
150     -- Purpose: Register test pattern fr
151     -- Only least significant 4 bits are
152     p_TP_Index : process (i_Clk)
153     begin
154       if rising_edge(i_Clk) then
155         if w_RX_DV = '1' then
156           r_TP_Index <= w_RX_Byte(3 dowr
157         g_TOTAL_ROWS  => c_TOTAL_ROWS,
158         g_ACTIVE_COLS => c_ACTIVE_COLS,
159         g_ACTIVE_ROWS => c_ACTIVE_ROWS
160         )
161       port map (
162         i_Clk       => i_Clk,
163         o_HSync     => w_HSync_VGA,
164         o_VSync     => w_VSync_VGA,
```

```
      o_Col_Count => open,
      o_Row_Count => open
      );

  Test_Pattern_Gen_inst : entity work.
    generic map (
      g_Video_Width => c_VIDEO_WIDTH,
      g_TOTAL_COLS  => c_TOTAL_COLS,
      g_TOTAL_ROWS  => c_TOTAL_ROWS,
      g_ACTIVE_COLS => c_ACTIVE_COLS,
      g_ACTIVE_ROWS => c_ACTIVE_ROWS
      )
    port map (
      i_Clk       => i_Clk,
      i_Pattern   => r_TP_Index,
      i_HSync     => w_HSync_VGA,
      i_VSync     => w_VSync_VGA,
      --
      o_HSync     => w_HSync_TP,
      o_VSync     => w_VSync_TP,
      o_Red_Video => w_Red_Video_TP,
      o_Blu_Video => w_Blu_Video_TP,
      o_Grn_Video => w_Grn_Video_TP
      );

  VGA_Sync_Porch_Inst : entity work.VG
    generic map (
      g_Video_Width => c_VIDEO_WIDTH,
      g_TOTAL_COLS  => c_TOTAL_COLS,
      g_TOTAL_ROWS  => c_TOTAL_ROWS,
      g_ACTIVE_COLS => c_ACTIVE_COLS,
      g_ACTIVE_ROWS => c_ACTIVE_ROWS
      )
    port map (
      i_Clk       => i_Clk,
      i_HSync     => w_HSync_VGA,
      i_VSync     => w_VSync_VGA,
      i_Red_Video => w_Red_Video_TP,
      i_Grn_Video => w_Blu_Video_TP,
      i_Blu_Video => w_Grn_Video_TP,
      --
      o_HSync     => w_HSync_Porch,
      o_VSync     => w_VSync_Porch,
      o_Red_Video => w_Red_Video_Porch
      o_Grn_Video => w_Blu_Video_Porch
      o_Blu_Video => w_Grn_Video_Porch
      );

  o_VGA_HSync <= w_HSync_Porch;
  o_VGA_VSync <= w_VSync_Porch;

```

```
    o_VGA_Red_0 <= w_Red_Video_Porch(0);
    o_VGA_Red_1 <= w_Red_Video_Porch(1);
    o_VGA_Red_2 <= w_Red_Video_Porch(2);

    o_VGA_Grn_0 <= w_Grn_Video_Porch(0);
    o_VGA_Grn_1 <= w_Grn_Video_Porch(1);
    o_VGA_Grn_2 <= w_Grn_Video_Porch(2);

    o_VGA_Blu_0 <= w_Blu_Video_Porch(0);
    o_VGA_Blu_1 <= w_Blu_Video_Porch(1);
    o_VGA_Blu_2 <= w_Blu_Video_Porch(2);

  end architecture RTL;
```

Verilog Code – VGA_Test_Patterns_Top.v:

```
module VGA_Test_Patterns_Top
  (input  i_Clk,       // Main Clock
   input  i_UART_RX,   // UART RX Data
   output o_UART_TX,   // UART TX Data
   // Segment1 is upper digit, Segment
   output o_Segment1_A,
   output o_Segment1_B,
   output o_Segment1_C,
   output o_Segment1_D,
   output o_Segment1_E,
   output o_Segment1_F,
   output o_Segment1_G,
   //
   output o_Segment2_A,
   output o_Segment2_B,
   output o_Segment2_C,
   output o_Segment2_D,
   output o_Segment2_E,
   output o_Segment2_F,
   output o_Segment2_G,

   // VGA
   output o_VGA_HSync,
   output o_VGA_VSync,
   output o_VGA_Red_0,
   output o_VGA_Red_1,
   output o_VGA_Red_2,
   output o_VGA_Grn_0,
   output o_VGA_Grn_1,
```

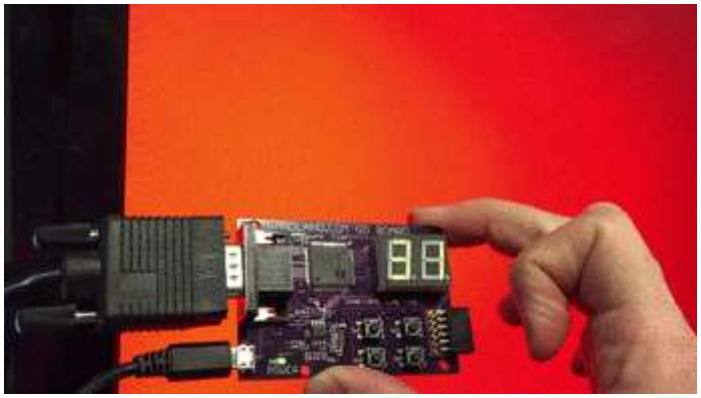
```
30      output o_VGA_Grn_2,
31      output o_VGA_Blu_0,
32      output o_VGA_Blu_1,
33      output o_VGA_Blu_2
34      );
35
36    // VGA Constants to set Frame Size
37    parameter c_VIDEO_WIDTH = 3;
38    parameter c_TOTAL_COLS  = 800;
39    parameter c_TOTAL_ROWS  = 525;
40    parameter c_ACTIVE_COLS = 640;
41    parameter c_ACTIVE_ROWS = 480;
42
43    wire w_RX_DV;
44    wire [7:0] w_RX_Byte;
45    wire w_TX_Active, w_TX_Serial;
46
47    wire w_Segment1_A, w_Segment2_A;
48    wire w_Segment1_B, w_Segment2_B;
49    wire w_Segment1_C, w_Segment2_C;
50    wire w_Segment1_D, w_Segment2_D;
51    wire w_Segment1_E, w_Segment2_E;
52    wire w_Segment1_F, w_Segment2_F;
53    wire w_Segment1_G, w_Segment2_G;
54
55    reg [3:0] r_TP_Index = 0;
56
57    // Common VGA Signals
58    wire [c_VIDEO_WIDTH-1:0] w_Red_Videc
59    wire [c_VIDEO_WIDTH-1:0] w_Grn_Videc
60    wire [c_VIDEO_WIDTH-1:0] w_Blu_Videc
61
62    // 25,000,000 / 115,200 = 217
63    UART_RX #(.CLKS_PER_BIT(217)) UART_R
64    (.i_Clock(i_Clk),
65     .i_RX_Serial(i_UART_RX),
66     .o_RX_DV(w_RX_DV),
67     .o_RX_Byte(w_RX_Byte));
68
69    UART_TX #(.CLKS_PER_BIT(217)) UART_T
70    (.i_Clock(i_Clk),
71     .i_TX_DV(w_RX_DV),      // Pass RX
72     .i_TX_Byte(w_RX_Byte),  // Pass RX
73     .o_TX_Active(w_TX_Active),
74     .o_TX_Serial(w_TX_Serial),
75     .o_TX_Done());
76
77    // Drive UART line high when transmi
78    assign o_UART_TX = w_TX_Active ? w_T
79
80
```

```
81    // Binary to 7-Segment Converter for
82    Binary_To_7Segment SevenSeg1_Inst
83    (.i_Clk(i_Clk),
84     .i_Binary_Num(w_RX_Byte[7:4]),
85     .o_Segment_A(w_Segment1_A),
86     .o_Segment_B(w_Segment1_B),
87     .o_Segment_C(w_Segment1_C),
88     .o_Segment_D(w_Segment1_D),
89     .o_Segment_E(w_Segment1_E),
90     .o_Segment_F(w_Segment1_F),
91     .o_Segment_G(w_Segment1_G));
92
93    assign o_Segment1_A = ~w_Segment1_A;
94    assign o_Segment1_B = ~w_Segment1_B;
95    assign o_Segment1_C = ~w_Segment1_C;
96    assign o_Segment1_D = ~w_Segment1_D;
97    assign o_Segment1_E = ~w_Segment1_E;
98    assign o_Segment1_F = ~w_Segment1_F;
99    assign o_Segment1_G = ~w_Segment1_G;
100
101
102   // Binary to 7-Segment Converter for
103   Binary_To_7Segment SevenSeg2_Inst
104   (.i_Clk(i_Clk),
105    .i_Binary_Num(w_RX_Byte[3:0]),
106    .o_Segment_A(w_Segment2_A),
107    .o_Segment_B(w_Segment2_B),
108    .o_Segment_C(w_Segment2_C),
109    .o_Segment_D(w_Segment2_D),
110    .o_Segment_E(w_Segment2_E),
111    .o_Segment_F(w_Segment2_F),
112    .o_Segment_G(w_Segment2_G));
113
114   assign o_Segment2_A = ~w_Segment2_A;
115   assign o_Segment2_B = ~w_Segment2_B;
116   assign o_Segment2_C = ~w_Segment2_C;
117   assign o_Segment2_D = ~w_Segment2_D;
118   assign o_Segment2_E = ~w_Segment2_E;
119   assign o_Segment2_F = ~w_Segment2_F;
120   assign o_Segment2_G = ~w_Segment2_G;
121
122   ////////////////////////////////
123   // VGA Test Patterns
124   ////////////////////////////////
125   // Purpose: Register test pattern fr
126   // Only least significant 4 bits are
127   always @(posedge i_Clk)
128   begin
129     if (w_RX_DV == 1'b1)
130       r_TP_Index <= w_RX_Byte[3:0];
131   end
```

```
132
133     // Generates Sync Pulses to run VGA
134     VGA_Sync_Pulses #(.TOTAL_COLS(c_TOTA
135                       .TOTAL_ROWS(c_TOTA
136                       .ACTIVE_COLS(c_ACT
137                       .ACTIVE_ROWS(c_ACT
138     (.i_Clk(i_Clk),
139      .o_HSync(w_HSync_Start),
140      .o_VSync(w_VSync_Start),
141      .o_Col_Count(),
142      .o_Row_Count()
143     );
144
145
146     // Drives Red/Grn/Blue video - Test
147     Test_Pattern_Gen  #(.VIDEO_WIDTH(c_V
148                         .TOTAL_COLS(c_TC
149                         .TOTAL_ROWS(c_TC
150                         .ACTIVE_COLS(c_A
151                         .ACTIVE_ROWS(c_A
152     Test_Pattern_Gen_Inst
153      (.i_Clk(i_Clk),
154       .i_Pattern(r_TP_Index),
155       .i_HSync(w_HSync_Start),
156       .i_VSync(w_VSync_Start),
157       .o_HSync(w_HSync_TP),
158       .o_VSync(w_VSync_TP),
159       .o_Red_Video(w_Red_Video_TP),
160       .o_Grn_Video(w_Grn_Video_TP),
161       .o_Blu_Video(w_Blu_Video_TP));
162
163     VGA_Sync_Porch  #(.VIDEO_WIDTH(c_VID
164                       .TOTAL_COLS(c_TOTA
165                       .TOTAL_ROWS(c_TOTA
166                       .ACTIVE_COLS(c_ACT
167                       .ACTIVE_ROWS(c_ACT
168     VGA_Sync_Porch_Inst
169      (.i_Clk(i_Clk),
170       .i_HSync(w_HSync_TP),
171       .i_VSync(w_VSync_TP),
172       .i_Red_Video(w_Red_Video_TP),
173       .i_Grn_Video(w_Grn_Video_TP),
174       .i_Blu_Video(w_Blu_Video_TP),
175       .o_HSync(w_HSync_Porch),
176       .o_VSync(w_VSync_Porch),
177       .o_Red_Video(w_Red_Video_Porch),
178       .o_Grn_Video(w_Grn_Video_Porch),
179       .o_Blu_Video(w_Blu_Video_Porch));
180
181     assign o_VGA_HSync = w_HSync_Porch;
182     assign o_VGA_VSync = w_VSync_Porch;
```

```
183
184     assign o_VGA_Red_0 = w_Red_Video_Por
185     assign o_VGA_Red_1 = w_Red_Video_Por
186     assign o_VGA_Red_2 = w_Red_Video_Por
187
188     assign o_VGA_Grn_0 = w_Grn_Video_Por
189     assign o_VGA_Grn_1 = w_Grn_Video_Por
190     assign o_VGA_Grn_2 = w_Grn_Video_Por
191
192     assign o_VGA_Blu_0 = w_Blu_Video_Por
193     assign o_VGA_Blu_1 = w_Blu_Video_Por
194     assign o_VGA_Blu_2 = w_Blu_Video_Por
195
196 endmodule
```

Now after building the code in iCEcube2, we should take a quick look at the synthesis results. How many LUTs and Registers (Flip-Flops) does your design use? Let's take a look at the programmed Go Board in action.



Keyboard Presses Change Test Pattern on VGA Display

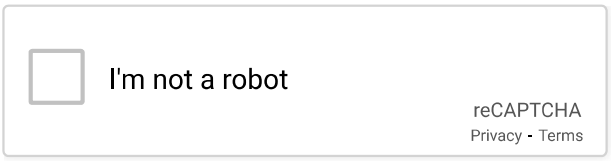
And Finally, PONG on the Go Board

Leave A Comment

Comment...

Name (required)

Email (required)



Post Comment