Example 39

A VGA Controller

In this example we will show how to implement a VGA controller for a standard VGA monitor in VHDL.

Prerequisite knowledge:
Example 33 – Counters

39.1 Background

A VGA controller is a component that controls five signals basic to video display. These signals are the horizontal sync $HS$, the vertical sync $VS$, and the three color signals $R$, $G$, and $B$ which output the level of red, green, and blue, respectively. A VGA monitor works on the basis of emitting energy in the red, green, and blue spectrum proportional to the voltage on the corresponding $R$, $G$, or $B$ signal input to the screen. Each colored dot on the screen is called a pixel (for picture element). How the monitor accomplishes this emission of color depends on the technology such as a cathode ray tube monitor or a liquid crystal display. In either case, the screen will display pixels starting from the top left corner of the screen moving towards the right and transitions, line-by-line in a direction moving towards the bottom of the screen. A horizontal sync pulse synchronizes each new line. Once it reaches the bottom of the screen, a vertical sync pulse causes it to start over again from the top left corner. The video controller must continuously output $R$, $G$, and $B$ levels while synchronizing the horizontal lines and timing the vertical retrace to refresh the screen. The $R$, $G$, and $B$ inputs to a monitor are analog. However, since we will be using an FPGA to control them, we will use digital outputs of $0v$ and $3.3v$, essentially turning them off or on. A standard monitor operates at $60$ Hz. That is, the screen is refreshed $60$ times per second, which is equal to once every $1/60^{th}$ of a second. Therefore, given a target resolution supported by the monitor, timing for each pixel, horizontal line, and vertical retrace can be derived.

39.2 Timing

In this example, we will design a VGA controller for a standard screen displaying a resolution of $640 \times 480$, a common, lower resolution for a VGA monitor. A $25$ MHz clock will be used to drive the controller and as we will later discover, it is sufficient to accomplish the target resolution. We will refer to this as the pixel clock. First, we must derive the timing for the horizontal sync signal, $HS$. This signal consists of four regions, the sync pulse ($SP$), back porch ($BP$), horizontal video ($HV$), and front porch ($FP$). The sync pulse signals the beginning of a new line, this is accomplished by bringing $HS$ low.
The signal is then brought high for the back porch where pixels are not yet written to the screen at the left. After the back porch, the *HS* signal remains high during the horizontal video period where pixels are written to the screen proceeding from left to right. Finally, the *HS* signal also remains high during the front porch where no pixels are written to the screen at the right. Fig. 39.1 shows the horizontal sync regions and timing.

![Figure 39.1 Horizontal Timing](image)

Pixel clock = 25 MHz  
Pixel time = 0.04 µs  
Horizontal video = 640 pixels x 0.04 µs = 25.60 µs  
Back porch, BP = 16 pixels x 0.04 µs = 0.64 µs  
Front porch, FP = 16 pixels x 0.04 µs = 0.64 µs  
Sync pulse, SP = 128 pixels x 0.04 µs = 5.12 µs  
Horizontal Scan Lines = SP + BP + HV + FP  
= 128 + 16 + 640 + 16  
= 800 pixels x 0.04 µs = 32 µs  
1/60 Hz = 16.67 ms / 32 µs = 521 horizontal scan lines per frame

At a 25 MHz pixel clock, each pixel will take $\frac{1}{25 \times 10^6} = 0.04 \mu$s. Next, we should consider the horizontal video period since we desire 640 pixels per line for a 640x480 resolution. Six hundred forty pixels require $640 \times 0.04 \mu$s = 25.60 µs to display across one line. According to specification, the length of the sync pulse, *SP*, should be approximately one-fifth of the horizontal video time. Therefore, $SP = \frac{25.60 \mu s}{5} = 5.12 \mu$s. Using our 25 MHz clock, this means that *SP* requires 5.12µs / 0.04µs = 128 clock ticks or pixels. Also according to specification, the back and front porches should each be approximately one fortieth of the time required for the horizontal video. Therefore, the porches require $\frac{25.60 \mu s}{40} = 0.64 \mu$s each. Using a 25 MHz pixel clock, each porch requires 0.64 µs / 0.04 µs = 16 clock ticks or pixels.

Finally, calculating the total number of clock pulses for a line we have $SP + BP + HV + FP = 128+16+640+16 = 800$ pixel clock pulses at 0.04 µs per pulse yields 32 µs for the entire line. Considering the line by counting pixels, we have 128 invisible pixels for the sync pulse where *HS* is low. The horizontal video region, where pixels are visible, starts at 128+16=144 after the sync pulse and invisible back porch and continues for 640 pixels stopping at 144+640=784. At this point, there is no visible video again at the front porch for 16 pixels until 784+16 = 800 where the sequence starts over again. Given that one entire screen, or frame, must be written in $\frac{1}{60}$th of a second or 16.67 ms
and each line requires 32 µs, it is possible to write \( 16.67 \text{ ms} / 32 \mu\text{s} = 521 \) horizontal lines per screen. This is consistent with our target resolution of 640x480 which requires 480 lines per screen. Since 521 is slightly greater than 480, it is perfect for synchronizing the vertical sync signal since it also has a vertical sync pulse, back porch, vertical video, and front porch region.

Of the four regions that make up the vertical sync signal, the sync pulse \((SP)\), back porch \((BP)\), vertical video \((VV)\), and front porch \((FP)\), we already know that the vertical video region must be 480 lines. Fig. 39.2 shows the vertical sync signal and timing.

The vertical video region requires 480 lines at 32 µs per line = 15.360 ms. According to specification, the vertical sync pulse should be approximately \(1/240^{th}\) of the vertical video timing. Therefore, \(15.360 \text{ ms} / 240 = 0.064 \text{ ms}\) for the vertical SP. At 32 µs per line, this requires 0.064 ms / 32 µs = 2 lines. Finally, splitting the remaining 39 lines between the back porch and the front porch using a 75%, 25% split as per specification, the back porch will be 29 lines and the front porch will be 10 lines. Finally, the 521 lines are split into a vertical \(SP + BP + VV + FP\)

\[
= 2 + 29 + 480 + 10
= 521 \text{ lines x 32 µs} = 16.672 \text{ ms}
\]

1/60 Hz = 16.67 ms

The vertical video region requires 480 lines at 32 µs per line = 15.360 ms. According to specification, the vertical sync pulse should be approximately \(1/240^{th}\) of the vertical video timing. Therefore, \(15.360 \text{ ms} / 240 = 0.064 \text{ ms}\) for the vertical SP. At 32 µs per line, this requires 0.064 ms / 32 µs = 2 lines. Finally, splitting the remaining 39 lines between the back porch and the front porch using a 75%, 25% split as per specification, the back porch will be 29 lines and the front porch will be 10 lines. Finally, the 521 lines are split into a vertical \(SP + BP + VV + FP\)

\[
= 2 + 29 + 480 + 10
= 521 \text{ lines x 32 µs} = 16.672 \text{ ms}
\]

1/60 Hz = 16.67 ms
Listings 39.1 through 39.4 show the VHDL implementation of this VGA controller showing red and green stripes across the screen. Listing 39.1 shows the entity and constants for the architecture. Listing 39.2 shows the clock divider and horizontal sync counter. Listing 39.3 shows the vertical sync counter. Listing 39.4 contains the visible video signal and signals for red, green, and blue.

**Listing 39.1  vga_stripes.vhd (entity and constants for the architecture)**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vga_stripes is
  Port ( clk, clr : in std_logic;
         hsync, vsync : out std_logic;
         red, green, blue : out std_logic);
end vga_stripes;

architecture vga_stripes of vga_stripes is

constant hpixels: std_logic_vector(9 downto 0) := "1100100000";
  --Value of pixels in a horizontal line = 800
constant vlines: std_logic_vector(9 downto 0) := "1000001001";
  --Number of horizontal lines in the display = 521
constant hbp: std_logic_vector(9 downto 0) := "0010010000";
  --Horizontal back porch = 144 (128+16)
constant hfp: std_logic_vector(9 downto 0) := "1100010000";
  --Horizontal front porch = 784 (128+16+640)
constant vbp: std_logic_vector(9 downto 0) := "0000011111";
  --Vertical back porch = 31 (2+29)
constant vfp: std_logic_vector(9 downto 0) := "0111111111";
  --Vertical front porch = 511 (2+29+480)
signal hc, vc: std_logic_vector(9 downto 0);
  --These are the Horizontal and Vertical counters
signal vidon : std_logic;
  --Tells whether or not it’s ok to display data
signal vsenable: std_logic;
  --Enable for the Vertical counter
```

As input, the entity defines a 25 MHz clock, `clk`, and a clear signal, `clr`. The controller outputs `hsync`, `vsync`, `red`, `green`, and `blue`. For convenience, we define constants for the horizontal and vertical regions including the horizontal and vertical sync pulses, back porches, video regions, and front porches. We will create a horizontal counter that counts pixels and a vertical counter to count lines. The constants will define the regions according to these counter values.

Listing 39.2 also shows a process for the horizontal sync counter. This counter must count up to 800 decimal and therefore must be 10 bits wide. When the counter
equals $hpixels$, it has reached 800 and must start over. At that time, it also must signal
the vertical line counter to count a line by bringing $vсенable$ high for one clock cycle. Finally, the $h-sync$ signal is low during the sync pulse and back porch, that is, when $hc$ is
less than 128. Otherwise, $h-sync$ is high. At this point, we are only setting the $h-sync$
signal. We will consider another signal, $vidon$, to differentiate the visible video ranges
from the porches.

Listing 39.2 vga_stripes.vhd (continued, horizontal sync)

```
begin
  --Counter for the horizontal sync signal
  process (clk, clr)
  begin
    if clr = '1' then
      hc <= "00000000000";
    elsif (clk'event and clk = '1') then
      if hc = hpixels-1 then
        --The counter has reached the end of pixel count
        hc <= "00000000000";
        --reset the counter
        vsenable <= '1';
        --Enable the vertical counter to increment
      else
        hc <= hc + 1;
        --Increment the horizontal counter
        vsenable <= '0';
        --Leave the vsenable off
      end if;
    end if;
  end process;

  hsync <= '0' when hc < 128 else '1';
  --Horizontal Sync Pulse is low when hc is 0 – 127
```

Listing 39.3 shows a similar process for the vertical sync counter. Every time the
$vсенable$ signal goes high, the vertical counter increments to count a line. When the
counter has reached $vlines$, it must reset to start counting lines for a new frame. Finally, $v-sync$ is set as low if $vc$ is less than 2 for the vertical sync pulse and high otherwise.

Now that we have implemented a counter for the horizontal sync and a counter for
the vertical sync signals and have defined $h-sync$ and $v-sync$ accordingly, we must
differentiate between regions with visible video from the porches. In Listing 39.4, we set
the $vidon$ signal to high when the horizontal counter is between the horizontal back porch
and front porch and the vertical counter is between the vertical back porch and front
porch. Otherwise, $vidon$ is low where there is no visible video.
--Counter for the vertical sync signal
process(clk, clr)
begin
    if clr = '1' then
        vc <= "0000000000";
    elsif(clk'event and clk = '1' and vsenable='1') then
        --Increment when enabled
        if vc = vlines-1 then
            --Reset when the number of lines is reached
            vc <= "0000000000";
        else
            vc <= vc + 1; --Increment vertical counter
        end if;
    end if;
end process;

vsync <= '0' when vc < 2 else '1';
--Vertical Sync Pulse is low when vc is 0 or 1

vidon <= '1' when (((hc < hfp) and (hc > hbp))
    and (vc < vfp) and (vc > vbp)) else '0';
--Enable video out when within the porches
process(vidon, hc)
begin
    red <= '0';
    green <= '0';
    blue <= '0';
    if vidon = '1' then
        red <= vc(3);
        green <= not vc(3);
    end if;
end process;
end vga_stripes;

Finally, a process for driving the red, green, and blue signals is given. All signals are defaulted to low. In the visible video regions, when vidon is high, red is high when vc(3) is high and green is the opposite. This way, every eight lines the colors will alternate between red and green showing a band of red eight lines high and a band of green eight lines high.

Figure 39.3 shows a simulation for 17.0 ms which is slightly longer than the time required to display one frame. Notice the vsync signal pulse in the beginning and at the 16.67 ms time indicating the start of a new frame. Also note the back and front porches
shown when $vidon$ is low. Fig. 39.4 shows a simulation for only the first 36 visible lines which alternate between red and green every eight lines.

Figure 39.3 A simulation of the VGA controller for one frame

Figure 39.4 A simulation of the VGA controller for the first 36 visible lines
39.3 Using the VGA Port on the Spartan 3 Board

Listings 39.5 and 39.6 give the top-level design for implementing the VGA controller on the Spartan 3 board.

**Listing 39.5 vga_top.vhd**

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity vga_top is
  port(
    mclk : in STD_LOGIC;
    btn : in STD_LOGIC_VECTOR(3 downto 0);
    hsync : out STD_LOGIC;
    vsync : out STD_LOGIC;
    red : out STD_LOGIC;
    green : out STD_LOGIC;
    blue : out STD_LOGIC
  );
end vga_top;

architecture vga_top of vga_top is

  component vga_stripes
    port(
      clk : in std_logic;
      clr : in std_logic;
      hsync : out std_logic;
      vsync : out std_logic;
      red : out std_logic;
      green : out std_logic;
      blue : out std_logic);
  end component;

  signal clr, clk: std_logic;
  signal clkdiv: std_logic_vector(23 downto 0);

begin

  -- Divide the master clock (50Mhz) down to a lower frequency.
  -- clock divider
  process(mclk, clr)
  begin
    if clr = '1' then
      clkdiv <= X"000000";
    elsif mclk'event and mclk = '1' then
      clkdiv <= clkdiv + 1;
    end if;
  end process;

  clk <= clkdiv(0);  -- 25 MHz
  clr <= btn(3);
```

Listing 39.6 vga_top.vhd (continued)

```vhdl
U1 : vga_stripes
    port map(
        clk => clk,
        clr => clr,
        hsync => hsync,
        vsync => vsync,
        red => red,
        green => green,
        blue => blue
    );
end vga_top;
```

Note that we simply wire up the `vga_stripes` component to a 25 MHz clock and connect the outputs to the VGA pinouts shown in Listing 38.7. These pinouts need to be included in the .ucf file for the project.

Listing 39.7 VGA pinouts for .ucf file

```text
NET "mclk" LOC = "T9" ;
NET "red" LOC = "R12" ;
NET "green" LOC = "T12" ;
NET "blue" LOC = "R11" ;
NET "hsync" LOC = "R9" ;
NET "vsync" LOC = "T10" ;
```
39.4 Other Resolutions

There are many other standard resolutions supported by most modern screens such as 800x600 and 1024x768. In order to achieve a higher resolution, a higher frequency pixel clock must be chosen. For example, to operate the VGA controller to output a resolution of 800x600, a 25 MHz clock would not be fast enough. Recall that with a 25 MHz clock and 640 visible pixels across each line, we only had 521 vertical lines. These lines had to be split between the vertical sync pulse, the vertical back and front porches, and the visible video. Obviously, if we expanded the pixels across to 800 visible pixels, the number of vertical lines would significantly decrease to much fewer than 521, while we desire 600 visible vertical lines. For any desired resolution, the pixel clock frequency must be calculated such that the following are true:

<table>
<thead>
<tr>
<th>Given: a target HV and VV pixel resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal constraints</td>
</tr>
<tr>
<td>HSP = 1/5 * HV</td>
</tr>
<tr>
<td>(horizontal sync pulse pixels)</td>
</tr>
<tr>
<td>HBP = 1/40 * HV</td>
</tr>
<tr>
<td>(horizontal back porch pixels)</td>
</tr>
<tr>
<td>HFP = 1/40 * HV</td>
</tr>
<tr>
<td>(horizontal front porch pixels)</td>
</tr>
<tr>
<td>Vertical constraints</td>
</tr>
<tr>
<td>VSP = 1/240 * VV</td>
</tr>
<tr>
<td>(vertical sync pulse lines)</td>
</tr>
<tr>
<td>VFP+VBP = 0.08125 * VV</td>
</tr>
<tr>
<td>(vertical back porch lines)</td>
</tr>
<tr>
<td>VBP = 0.75 * (VFP+VBP)</td>
</tr>
<tr>
<td>(vertical front porch lines)</td>
</tr>
</tbody>
</table>

where HSP+HBP+HV+HFP = HP_total pixels and VSP+VBP+VV+VFP = VP_total lines. Recall that VP_total must occur in 1/60th of a second. Then,

\[
\frac{16.67\text{ms}}{\text{VP}_{\text{total}}} = T_{\text{line}} \text{ms}
\]

and \[\frac{T_{\text{line}}}{\text{HP}_{\text{total}}} = T_{\text{pixel}} \text{ms}\]

Using these to solve for the clock frequency required for a pixel clock to achieve 1024x768 resolution, we obtain:

\[
\begin{align*}
\text{HV} &= 1024 \text{ (given)} \\
\text{HSP} &= 1/5 \times 1024 \approx 205 \\
\text{HBP} &= 1/40 \times 1024 \approx 26 \\
\text{HFP} &= 1/40 \times 1024 \approx 26 \\
\text{Therefore, HP}_{\text{total}} &= 205+26+1024+26 = 1281 \text{ pixels} \\
\text{VV} &= 768 \text{ (given)} \\
\text{VSP} &= 1/240 \times 768 \approx 3 \\
\text{VFP+VBP} &= 0.08125 \times 768 \approx 62 \\
\text{VBP} &= 0.75 \times 62 \approx 46 \\
\text{VFP} &= 0.25 \times 62 \approx 16 \\
\text{Therefore, VP}_{\text{total}} &= 3+46+768+16 = 833 \text{ lines}
\end{align*}
\]

Finally, 16.67ms/833 lines = \(T_{\text{line}} = 20\mu s\), and thus \(20\mu s/1281 \text{ pixels} = T_{\text{pixel}} = 0.0156\mu s\) for a pixel clock of \(1/0.0156\mu s \approx 64 \text{ MHz}\) clock. We have also calculated the constants for our VHDL VGA controller.