Basic VGA Controller Design Example

Author: Devon Andrade

Date: 8/18/2015

Revision: 1.0

Theory of Operation

This design example shows you how to utilize the LCD on the Max 10 Nios II Embedded Evaluation Kit. The end result of this example is that the LCD is driven with varying colors depending on the order the buttons on the board are pressed in (each button press corresponds to a different intensity in the three color groups). Below, you will find descriptions for how the LCD works as well as how each of the modules in this design was constructed.

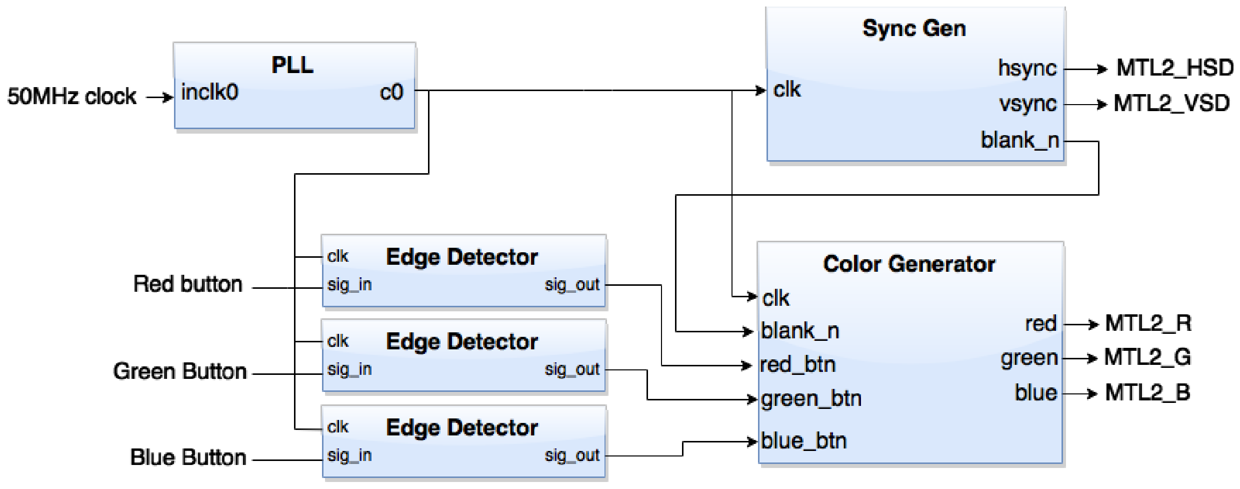

Figure 1: Block diagram of the entire design

Video Graphics Array (VGA) Interface

Before you can understand how the sync generator and color generator are designed, you need to understand the ubiquitous Video Graphics Array (VGA) interface. The LCD driver on this board (the chip that takes in video signals and translates them into what the LCD needs) accepts standard VGA signals (the same signals that would be used to drive a monitor) for sending pixel data as well as an extra signal for enabling and disabling the backlight (MTL2_BL_ON_n). The VGA interface consists of three 8-bit signals for each of the color values (red, green, and blue) along with two synchronization signals (which tell the monitor when to update the screen). The synchronization signals require exact timing to make sure the display is drawing the correct pixels at the correct time (in essence, determining the resolution of the display).

Figure 2 below shows the timing parameters for the common 640x480 resolution. Although this diagram doesn't show it, there's a certain "pixel clock" which is the frequency at which video data and the synchronization pulses are clocked out at. For the 640x480 resolution, the pixel clock would have a frequency of 25.175MHz (this is a part of the VESA standard, although LCD datasheets should give you this number as well). This would be the frequency you would run your VGA controller at within your hardware. Most monitors don't require you to send out the actual pixel clock with the data—they'll assume the pixel clock you're using based on the timing of your synchronization pulses and the resolution they're set to. With that said, low-level LCD driver or video codec chips seem to be the exception and you will need to output a pixel clock alongside your color data on the NEEK board (MTL2_DCLK).

Let’s start off by explaining the timing required for the horizontal synchronization pulse. Each line of pixels (640 pixels in this case) is ended with a horizontal synchronization pulse (active low in this case). There is a certain period of time before and after the synchronization pulse that is called the “blanking” period where the screen doesn’t display any video data. This was used in older CRT monitors to give the electron beams time to move back to the beginning of the next line and has been carried over into modern electronics for backwards compatibility. The blanking time in front of the sync pulse is called the “front porch” while the blanking time after the pulse is called the “back porch.” During the blanking period it is recommended to drive all of the color signals to zero (aka, black) to make sure nothing is displayed.

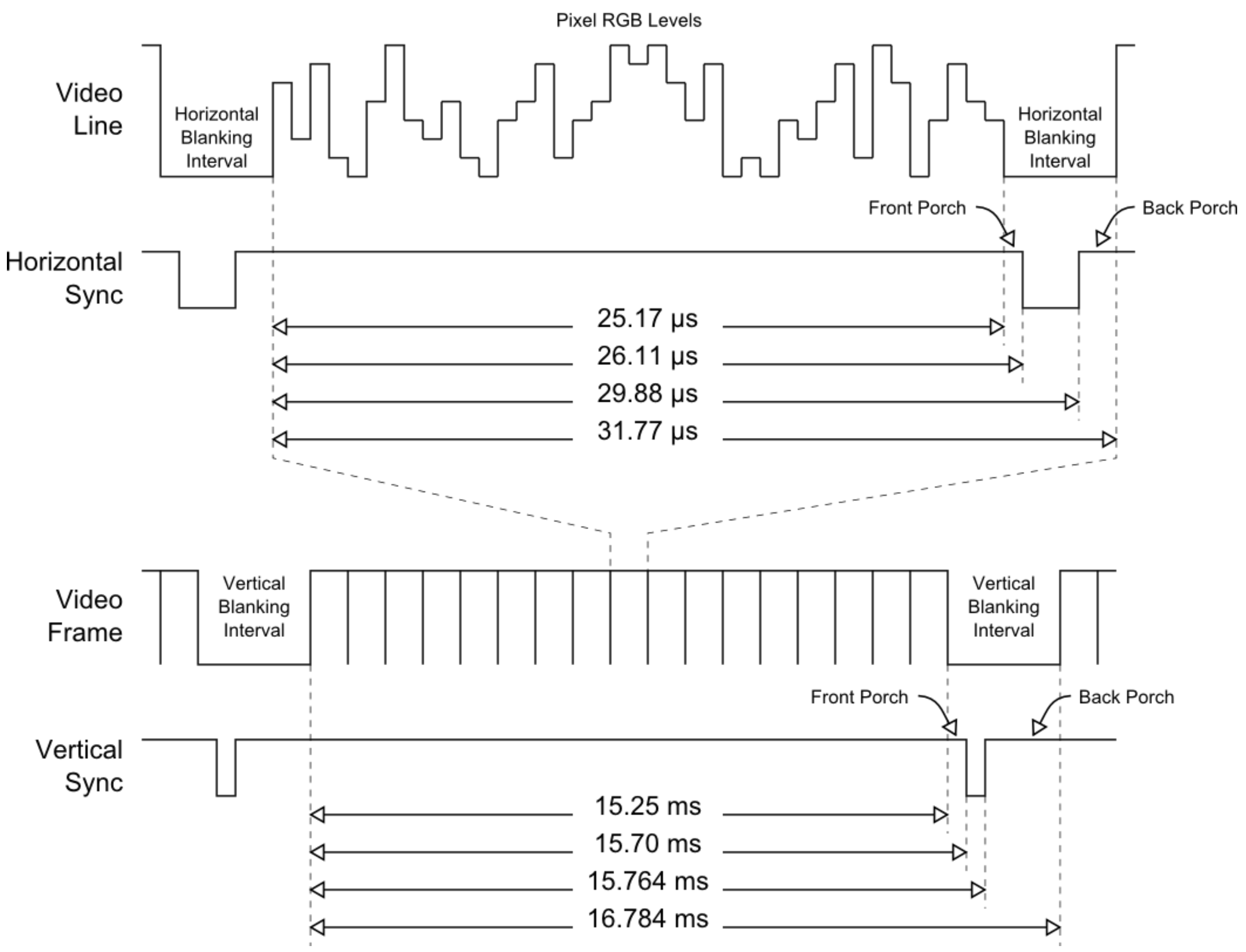

Figure 2: Timing parameters for the 640x480 resolution

As for actual video data, each pixel clock cycle after the blanking period a new pixel will be clocked “into” the screen with the screen displaying pixel 0 in the top left corner and moving to the right to fill the line. Once the line is complete, the blanking period begins and the video data is driven low and the cycle repeats itself. In essence, for each line of data, the following is occurring (all of the timing parameters below are standard values you can find in the VESA standard online):

1. Pixel data is shifted out over 640 clock cycles (each clock cycle determining the color for a different pixel in the line, starting on the left and moving right).
2. The blanking period begins (the front porch) and video data is driven low for 16 clock cycles (16 * (1/25.175MHz) = 0.63us).
3. The horizontal synchronization pulse is driven low for 96 cycles (96 * (1/25.175MHz) = 3.81us). This is still part of the blanking period, so we continue to drive the video data low (aka, black).
4. The horizontal synchronization pulse goes back to being asserted, and we wait 48 clock cycles (48 * (1/25.175MHz) = 1.91us) for the back porch before we repeat the process and start producing pixels for the next line (moving down the screen as it goes).

The process above repeats for 480 times to display an entire frame of video data. Just as there is a blanking period after displaying an entire line of video data, there is also a blanking period after displaying an entire frame. This was used by CRT monitors to move the electron beams back to the top left corner of the screen. And just as with the horizontal timing, the blanking period consists of a front porch, synchronization pulse, and back porch where each of these parameters is measured in “lines” (how many horizontal lines it takes to fulfill that part of the blanking period). And since this is a blanking period, the video data should be driven low. In the sync generator Verilog module, there is code that ORs the two blanking signals together to make a single active-low “blank_n” signal. This signal is checked within the color generator module to determine whether to drive the color signals low, or to display color data. The process to display an entire frame of information is shown below:

1. Follow the process for displaying a single line of data above and repeat it 480 times.
2. The vertical blanking period begins (front porch) and the video data is driven low for 10 lines (each one of these lines is exactly the same as the lines produced before, except the video data is always low).
3. The vertical synchronization pulse is driven low for two lines.
4. The vertical synchronization pulse is re-asserted and we wait 33 lines for the back porch before we start repeating the process to display the next frame of data.

In essence, each line of video data actually takes up 800 clock cycles (640 for the visible area + 160 for the blanking period) and each entire frame of video data takes 525 lines (480 for the visible area + 45 for the blanking period). When updating the video data to be displayed onto the screen (for instance, if you’re storing the video data in memory), it is highly recommended to update only during the vertical blanking period (when the screen isn’t displaying anything). This ensures that you don’t try to update the screen while your VGA controller is halfway through drawing the screen. This can cause an effect called screen tearing which results in the screen showing half of the new video data, and half of the old video data. Many computer games have an option in their graphics settings called “Vsync” which tells the game to wait until the vertical blanking period before updating the screen (this reduces frame rate, but eliminates screen tearing).

LCD Timing Parameters

The NEEK board’s LCD follows the same VGA interface described above albeit with different timing parameters than those described in the previous section. There are standard resolutions that most monitors conform to, and you can visit tinyvga.com to discover the exact timings (pixel clock frequency, and horizontal/vertical timings) for most standard resolutions. Unfortunately, the NEEK's LCD doesn't use a standard resolution; it instead opts for a non-standard 800x480 resolution. Below are the timing parameters required to display data on the NEEK's LCD:

Pixel Clock Frequency: 33.33MHz (33MHz will suffice)

Table 1: LCD timing parameters for the NEEK board

Horizontal Scanline Part	Pixels		Vertical Frame part	Lines
Visible area	800		Visible area	480
Front Porch	210		Front Porch	22
Sync Pulse	30		Sync Pulse	13
Back Porch	16		Back Porch	10
Whole line	1056		Whole frame	525

Most LCD screens will take timing parameters that are "close enough" and still manage to display everything correctly. With that said, it is highly recommended that you use the exact values above to get the crispest display.

Sync Generator Module

This module is required to output the horizontal and vertical synchronizations pulses at the correct time. It also outputs a single active-low "blank_n" signal that gets pulled low whenever the screen is displaying either the horizontal or vertical blanking periods (this is to tell the color generator to pull the color data low and display black). This module is parameterized to the point where it's generic enough to work with any resolution. All you need to do is override the timing parameters when you instantiate the module, and make sure to pass in the correct pixel clock. In this design, we are using one of the Max 10's PLLs to take the 50MHz input clock and divide it down to 33MHz.

The design of the module itself is relatively simple. Using the passed in parameters (that have a 1:1 relationship with the parameters in table 1), the sync generator calculates at what time to display the sync pulses. To know what it's currently displaying, the sync gen uses a pair of counters: one representing the current x coordinate (which pixel in the line we're currently on) and one representing the current y coordinate (which line we're currently on). When the horizontal counter reaches the end of the line, it resets back to zero and increments the vertical counter. Once the vertical counter reaches the end of the frame, it resets back to zero and the process starts over.

Color Generator Module

This module is responsible for determining which color will be shown on the screen. Depending on which button is pressed, the color generator will cycle through different intensities for each color. There is one button for each color group: red, green, and blue. Each press of one of these buttons will increase the intensity for that color on the screen until after 4 presses, in which it will roll back to the lowest intensity. Press KEY0, KEY1, and KEY2 to change the red, green, and blue colors respectively.

The color generator module also handles displaying black during the blanking periods. It contains a few assign statements that will either display the current color values or black depending on the "blank_n" signal coming from the sync generator. This ensures that nothing will be displayed during any of the blanking periods.

Edge Detectors

Since the buttons act as asynchronous inputs into an otherwise synchronous system, good practice dictates that we synchronize these signals before passing them along to the color generator. As well as synchronizing the button inputs, this module also detects the falling edge of the input (the buttons are active-low) and sends out a one clock cycle pulse. This prevents our color generator from accidentally sampling the button inputs multiple times for however long they are high.

To perform the synchronization a standard two flip-flop approach is used as shown below. To detect the falling edge, an AND gate is used between the outputs of the flip-flops. If the output of the first flip-flop is low and the output of the second one is high, then the signal has just recently fallen low and we need to send out a one-clock cycle pulse to inform the color generator. We could get small glitches thanks to metastability issues, but considering the low speeds this design is running at, the chances of glitches occurring are minimal to insignificant. Adding more synchronization stages to this before performing the edge detection would be one way to improve the design (this is left as an exercise for the reader).

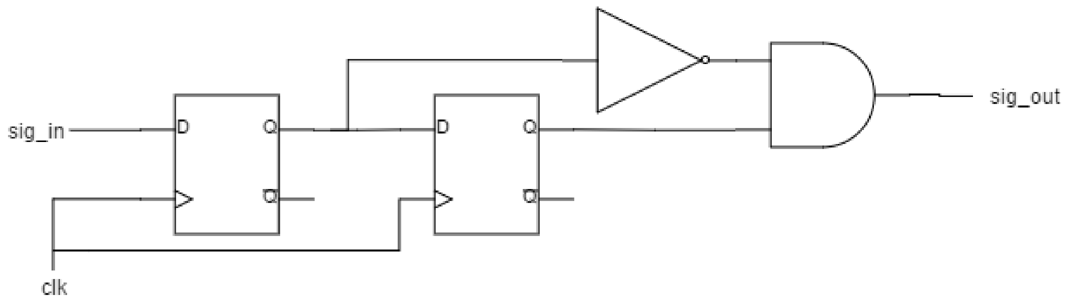

Figure 3: Block diagram of edge detector circuit

How to Compile the Hardware

Follow the steps on the Design Store web page to extract and install the Basic_VGA_Controller platform file. The following steps describe how to setup a project in the Quartus II software in order to program the MAX10 FPGA device with the design example.

1. Launch Quartus II and open the top.qpf project file using File->Open Project.
2. Compile the project by clicking on Processing->Start Compilation.
3. Once the project is done compiling, launch the Quartus II Programmer from the tools menu (Tools->Programmer).
4. Plug in your development kit and make sure "USB-Blaster II" or "NEEK10" is shown next to the "Hardware Setup…" button. If it doesn't, click on that button and select the correct programmer.

5. Once the programmer is selected, click on “Auto-Detect” in the programmer’s sidebar. If a dialog box appears asking for which device is on the JTAG chain, select “10M50DA”.
6. Double click in the File list where it says “<none>” and select the output_files/top.sof file.
7. Check the “Program/Configure” box then click “Start” to program the device.
8. After programming, press KEY0, KEY1, and KEY2 to change the red, green, and blue colors respectively.