

ECE241 - Digital Systems

University of Toronto

Lab 7: Complex State Machines and Video Graphics Array (VGA) Display

1 Introduction

The purpose of this laboratory is to further expand your understanding of finite state machines (FSMs) and to learn how one can use a “Video Graphics Array” (VGA) Adapter to create graphics and animation on a computer screen. You will create FSMs to interact with a VGA Adapter that has been implemented for you to control and generate images and animations on the screen.

2 Preparation

To prepare for this laboratory, first read the VGA Adapter documentation provided at:

http://www.eecg.utoronto.ca/~jayar/ece241_07F/vga

Be sure to read the four sections: “How do monitors work?”, “How does the VGA Adapter work?”, “VGA Adapter’s Interface” and “Changing Adapter Parameters”

Also, download the lab starter kit located at:

http://www.eecg.utoronto.ca/~jayar/ece241_07F/Lab7_starterkit.zip

Once you have read the VGA Adapter documentation proceed to complete parts I and II described below. Your preparation, to be shown to your TA at the beginning of the lab, must consist of the following:

1. The code used to implement each of the circuits described in parts I and II. The code should be **PRINTED** and placed in your lab book **before** you arrive in the lab.
2. Simulation of the FSM and the datapath for the circuit in part II. First simulate the FSM and the datapath separately, then put them together. The simulation must clearly indicate the operation of each circuit. The simulation must be **PRINTED** and **COMMENTED** before the lab begins. The comments must explain why the simulation is correct.

You need not include part III in your preparation. This part is a bonus to this lab and deals with the creation of animation using the VGA Adapter. If your project involves creating graphics using the VGA Adapter, you are encouraged to attempt this part to better understand some of the challenges you will face in your project.

3 In the lab

In the lab you will have to implement and test circuits described in the sections below. To simplify some of the steps a starter kit has been provided on the course website. The starter kit is a ZIP archive containing a Quartus II projects for each part of the lab. Unzip the archive into a work directory called *lab7*.

3.1 Part Ia

To familiarize yourself with the VGA Adapter module you will perform a simple exercise to display a custom image on the screen. To do this you will need to use the bmp2mif converter provided in the starter kit. It is a program that converts a bitmap image into a stream of bits that can be programmed into the memory on an FPGA.

To be able to display a picture you must first draw a picture using a graphics editing tool that can save files in a BMP format. The Microsoft Windows "Paint" program is one such tool. The image we will draw will cover the entire screen and thus has to be created correctly to be displayed properly. Perform the following steps to draw an image:

1. Start the Paint program, typically available from the Start Menu: that is Start->Programs->Accessories->Paint
2. Select the menu item Image->Attributes. In the dialog box set "width" equal to 160 and "height" equal to 120, as this is the resolution of the monitor that the adapter uses. Select "pixels" as the unit. Select "colours" as well.
3. Draw a picture of your own design; use simple colours like Red, Green and Blue.
4. Save the file, using File->Save As and save it as a 24-colour bitmap image.bmp.
5. Run the bmp2mif.exe converter program to convert your BMP file to a Memory Initialization File (MIF) we will use next. (To do that, start up a DOS command shell on windows using Start->Run and type "cmd" into the Open: box that pops up. This will create a window you can type commands into. Change into a folder that contains both bmp2mif.exe and the file.bmp you wish to convert. Then type "bmp2mif file.bmp." This will produce two .mif files - image.mono.mif and image.mcolour.mif.)

Recall from the description of the VGA Adapter that it uses memory to store the current state of each pixel on the screen. Usually, this memory is initialized to 0 at first and hence you only see a black background. However, we can change the initial state of the VGA Adapter memory, causing it to display an image. You will use the image you created earlier with paint as that background.

Perform the following steps to change the initial image displayed by the Adapter:

1. The project for this part is provided in the starter kit. Open the project named *background* in the *part1* subdirectory to begin your work.
2. The BMP2MIF converter created a file called image.colour.mif, where your background image is stored. Copy this file to your working directory and change its name to display.mif. **Note:** The choice of the file name is not accidental. If you look carefully at the implementation of the VGA Adapter you will see that it has a parameter called BACKGROUND_IMAGE. This parameter is set to "display.mif" by default and signifies that the Quartus II software should use display.mif file to initialize the memory for the VGA Adapter. Note also that a display can only have one display file - it is programmed into the memory when the FPGA is configured, and that only happens once per circuit.

3. Assign pins to your project and compile it.
4. Program the circuit described in the preparation onto the Altera DE2 board.

When you program the DE2 board and connect a monitor to its graphics port, you should be able to see the image you have drawn.

IMPORTANT: Make sure you understand how this initialization process works: the memory (in the case of the VGA adapter, this memory is the framebuffer) is initialized with the contents of the MIF file (which stands for "Memory Initialization File") *only* when the FPGA is programmed. The MIF file is just the stream of raw data, and is not specific to the VGA Adapter - it can be used to initialize any kind of memory. The memory initialized this way can be changed – by modifying individual pixels on the screen as discussed in the VGA Adapter documentation. Note that as soon as you draw a pixel using the VGA Adapter, the contents of this memory will be altered. Thus, if you used a background image as we have shown above, the background image will be permanently altered. Resetting the VGA Adapter will NOT restore the "background image."

3.2 Part Ib

In this part you are asked to design a very simple circuit using the VGA adapter. The circuit has to perform the following functions:

1. Accept the X and Y coordinate inputs and the color input from the switches on the DE2 board.
2. Set the given color of the pixel at the given coordinates when push-button is pressed.

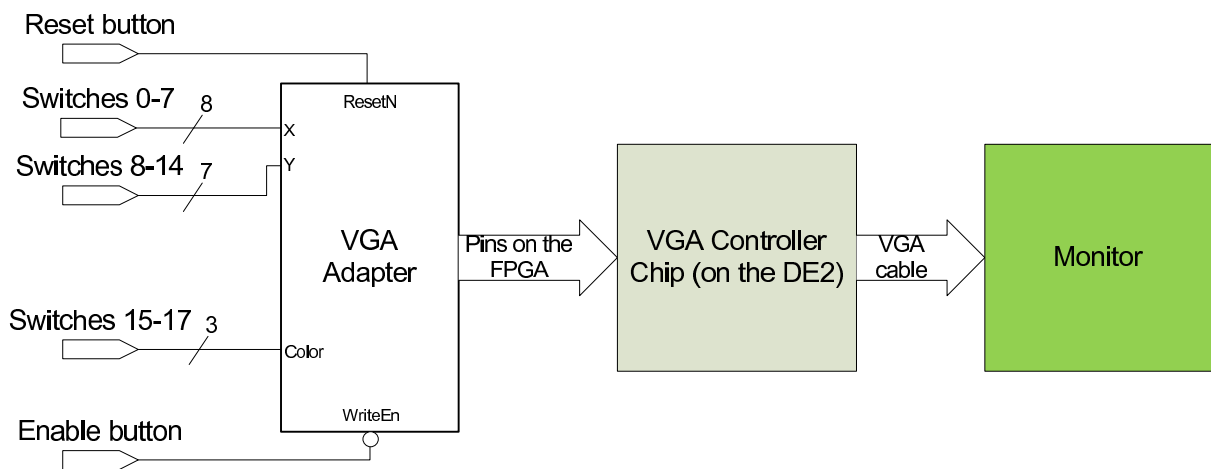


Figure 1: Full schematic of the circuit.

The schematic of the circuit is given in Figure 1. You should reuse the project file from *Part Ia*.

After completing this circuit you will be able to manually (and tediously) draw any picture on the screen. Once again, observe that any image which you loaded in *Part Ia* is gradually lost as you change the colors of the pixels. This is because the VGA adapter has only one framebuffer (which is basically a block of memory) to store the value of each pixel. This buffer was initialized with your image, but as you change the colors of the pixels, the old values are overwritten and lost.

3.3 Part II

In this part you will learn how to draw pixels on the screen after your circuit begins running. To demonstrate this, you will design a simple “Etch-a-Sketch” drawing system (see: http://en.wikipedia.org/wiki/Etch_A_Sketch if you don’t know what that is).

This Etch-a-Sketch is a circuit that moves a cursor around the screen, either up, down, left, or right, and draws a new pixel with every move the cursor makes. The system has 6 inputs, described below. The output is facilitated by the VGA Adapter and appears on the screen. The inputs are:

1. Resetn - an active low input to reset the system. Reset should cause the cursor to go to position (0,0) the top-left corner of the screen.
2. CLOCK_50 - the clock input to drive the finite state machine of the system as well as the VGA adapter.
3. Four switches labelled left, right, up, down - to indicate the direction to move the cursor in. Each time an input, for example left is set to 1, the drawing system show move the cursor left by exactly 1 pixel. (It should wait until the Left signal returns to 0 before moving the cursor again).

The high-level design of the circuit for the etch-a-sketch system is given in Figure 2. It contains 3 major blocks:

1. The VGA adapter responsible for the drawing of pixels on the screen. See the website above for a description of how the adapter works.
2. The datapath that controls the position of the cursor, providing the VGA adapter with the (X,Y) (i.e. column, row) location where a pixel should be drawn
3. A finite state machine that receives the input from a user and directs the datapath to change the position of the cursor accordingly, by adding and/or subtracting from the (X,Y) position of the cursor.

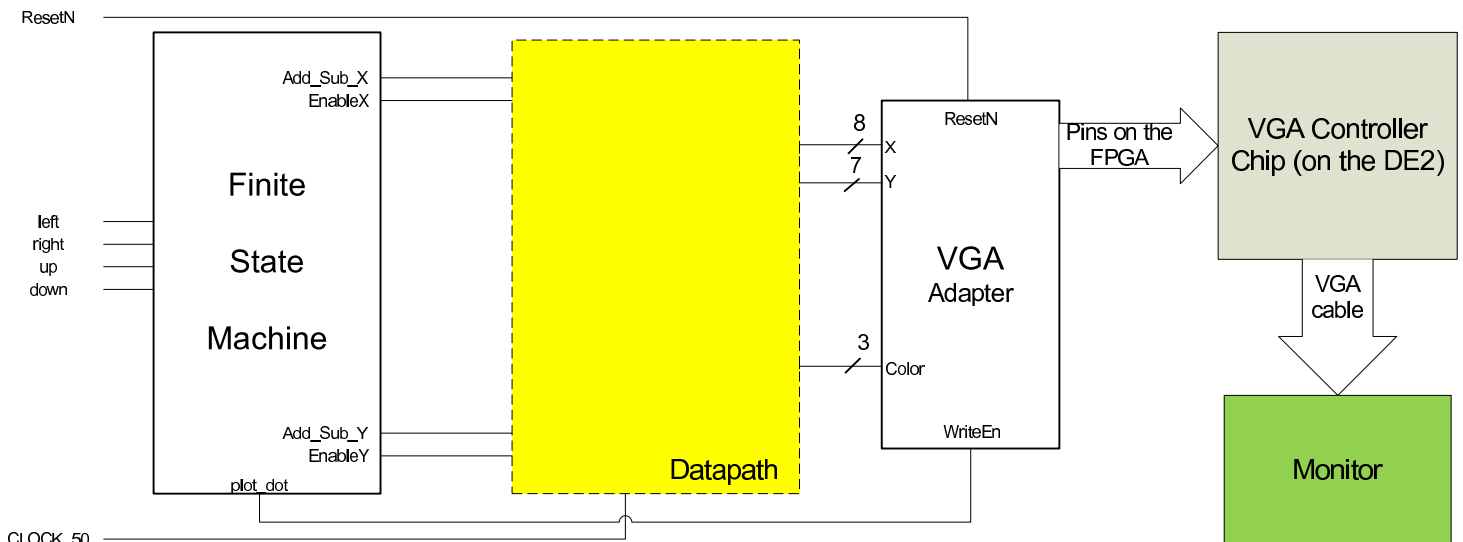


Figure 2: Design Overview - State Machine, Datapath and VGA Adapter. Although not shown, ResetN signal should be connected to all the registers in the circuit (including FSM state register).

You may use the circuit from *Part Ib* as a base for your design.

3.4 Part III (Bonus)

Now that we covered drawing pixels on the screen we can proceed to create a simple animation. We will create a circuit that takes a small image (16x16 pixels) and moves it around the screen. As the image moves our circuit will have to make it seem as though the image is seamlessly moving around the screen.

To move the image round the screen you will have to create a circuit that takes as input the (X,Y) coordinate of where the top left corner of the image is to be drawn. It will then read the image from memory (LPM_RAM_DQ module) and draw it one pixel at a time at the specified location. To do so you will need to create a state machine that performs the following function:

1. Set Counter_X and Counter_Y to 0.
2. While Counter_X is less than 16, load a pixel from memory containing `sprite.mif`. Then draw that pixel at location (X+Counter_X,Y+Counter_Y) on the screen. Increment Counter_X.
3. If Counter_Y is less than 15, then increment Counter_Y and set Counter_X to 0. Go to step 2.
4. Stop when Counter_Y reaches 16.

The above procedure can be used to draw an image at any location (X,Y). To move the image, you will have to first erase it from the location it is currently at and then draw it again at an alternate location.

Before we get to animating our sprite, lets first draw it on the screen. Do so by completing the following steps:

1. The project for this part is provided in the starter kit. Open the project named *part3* in the *part3* subdirectory to begin your work.
2. Create a 16x16 bitmap image that is to move around the screen. Make sure to set the image width and height to 16 pixel.
3. Use the `bmp2mif.exe` converter to convert the image into an MIF file. Call it `sprite.mif`.
4. In your design instantiate a memory using an LPM_RAM_DQ and use `sprite.mif` as its memory initialization file.
5. Create a circuit to draw an image at a specified location on the screen as discussed above.
6. Compile the circuit and download it onto the DE2 board. When your circuits starts you should be able to see the image you have drawn somewhere on the screen.

Now that we have a picture drawn on the screen, we need to move it around. There is a simple way to do that when the screen background is black. First, we draw an image at location (X,Y). Then to move the image we simply draw a black box on top of the image and redraw the image somewhere else.

1. Extend the above circuit to be capable of erasing an image from a given location (fill the given space with black pixels).
2. Create a circuit to change the location of the top left corner of where the image is to be drawn, once every 60th of a second.
3. Put the circuits together to see if your image moves around the screen. Compile the circuit and program it onto the FPGA to see if it works.
4. If the circuit works, think of a way to get the image to bounce of the sides of the screen as it moves about. Implement the enhancement and show the circuit to your Teaching Assistant.