MIFs, Sprites and BMP2MIF

Introduction

In Lab 7 Part II you were provided a magic file called *black.mif*. It was used to initialize the frame buffer with an all-black image. A MIF (Memory Initialization File) is an Altera memory format used to initialize memories on the FPGA. The file records each address in the memory along with the data at each address.

This supplement provides more details about how the MIF file is used and how to create MIF files with your own custom images using the bmp2mif tool. The bmp2mif tool is a program that converts a bitmap image into a stream of bits that can be programmed into the memory on an FPGA.

The tool and source files can be downloaded at `http://www.eecg.toronto.edu/~jayar/ece241_08F/vga/vga-download.html`.

A fancier version was contributed by a student in ECE241F 2015. Look for *Img2Mif* on Piazza.

How does a MIF file work?

In Lab 7, the *black.mif* file is used to initialize the contents of the frame buffer memory with all zeroes. If you have not done so already, you can look at the contents of the MIF file since it is a text file, which is essentially a list of addresses and the data value at each address. Since *black.mif* initializes the memory to all zeroes, the display is all black.

When you generate a memory module in Quartus you have the option of specifying a file to initialize the memory. In Lab 7 Part I, you probably did not see this because we told you to hit *Finish* before you got to see that option. The next time you need to generate a RAM module, just keep hitting *Next* and you will eventually see the window to specify the MIF file to use.

If you specify a MIF file, then at the time you generate a bitstream Quartus will read the MIF file and add the necessary bits into the bitstream to load that data into the memory when the FPGA is programmed. Note that this means that the initialization of the memory using the MIF file only happens once when the FPGA is programmed. There is no function that will reload data from the MIF file into the memory after the FPGA is configured.

If you look at the `fill.v` skeleton code that was provided for Lab 7, you will see a `defparam` statement that specifies *black.mif*. This is overriding a default MIF file specified by the parameter `BACKGROUND_IMAGE` in the `vga_adapter.v` file. You can load a different initial image by specifying a different file instead of *black.mif*.

Re-loading the MIF data

Consider the following common scenario. You have a game with a background image that stays constant. However, you have an image that moves across the background, say the image of a car. In general graphics terminology, the car is called a *sprite*, i.e., a small image that you are moving around the screen. Like in Lab 7 Part III, you want to draw the car and then move it. First you have to erase the car by redrawing the background under the car and then draw the car in the new position. Since the background is no longer simply black, you have to find the

right pixel values to erase the car and restore the backgound. Given that you cannot simply reload the MIF file into the framebuffer, you will have to store the background data in another memory first, call it the *Background* memory. To create this *Background* memory, figure out the size that you need and generate it as you did in Lab 7 Part I. Make sure to get to the step of specifying the MIF file for initialization of this memory. You can now initialize both the frame buffer and the *Background* memory with your background image MIF file when the FPGA is configured.

To erase the car, copy the appropriate pixels from the *Background* memory into the frame buffer when you want to erase the car, i.e., you should not need to redraw the whole screen. Keep track of where the car is located and only redraw those pixels.

If you want to restore a pixel at coordinate (X,Y), you will need to find that pixel in the *Background* memory. See the section below on converting the coordinate to a memory address.

The image for the car, i.e., the sprite for the car, can also be stored in a memory the same way as the *Background* memory. Whenever you want to draw the car, read the sprite memory and copy the image to the desired position in the frame buffer.

Creating the image and generating the MIF file are described next.

Creating a Bitmap

To be able to display a picture you must first draw a picture using a graphics editing tool that can save files in BMP format. The Microsoft Windows "Paint" program is one such tool. You will need to set the size of your bitmap. For example, if you wanted an image that would cover the entire 160×120 screen, you would need to set the BMP size to 160×120. Or, if you wanted to display a smaller 10×20 graphic for a sprite image in your game, you would also need to set the BMP size accordingly. The settings for the pixel height and width can be found in Images–>Attributes, making sure to also select 'colours.' After drawing your picture, save the file as a 24-bit colour bitmap. Although the image is saved as 24-bit pixels, remember that the default pixel size in the VGA adapter is three bits.

Creating a MIF using bmp2mif

We will use the bmp2mif.exe converter program to convert your BMP file to a MIF. To do this, 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.colour.mif.

Note that you can also use Quartus to create or manually edit MIF files as well.

Converting (X,Y) Pixel Coordinates to a Memory Address

The VGA adapter takes (X,Y) coordinates as inputs to specify where on the screen to put a pixel. The pixel is stored in the frame buffer, which is just a memory module. The (X,Y) coordinate must be converted to an address for the memory module to specify the location to store the pixel value. This conversion is done by the `vga_address_translator` module. To understand how the conversion is done, you need to know the order of the pixel data in memory. Assume that you have a screen of 160×120 pixels. The first row of pixels are stored in the first 160 addresses of the memory starting at address 0. The next row of pixels are stored in the next 160 addresses starting at address 160, etc. You can see how this conversion is done in the `vga_address_translator` module.

Using the MIF file in your design

To use the MIF file in your design, create a memory using the method described in Lab 7. You will need to create a memory with 3-bit words if you are using colours and 1-bit words if you are using mono. The depth of the memory will be the number of pixels in your image. When creating the memory using Quartus, click next until you are prompted to select your MIF file as your initialization file.