ECEN3763 – Small Frame Buffers
Week 7

Due Friday, March 4th, 11:59pm – 35 points

Summary:

For Project Week 7, you will enhance your video controller pixel generator, by adding a larger video memory to your system. The goal of this lab is to display images on your monitor using the frame buffer as the image source.

Using the M10K memory blocks on the CycloneV device, you will create a large frame buffer. The Cyclone 5 device does not contain enough memory for a full 720p 24 bit color image.

You can select one of two options for this project. One option is to create a full 24 bit color resolution display, which limits the image resolution to 320 x 180, or ¼ resolution. The second option is to display your image using 640 x 360 resolution, but using 12 bit color resolution.

For both options, find a .jpg or .bmp picture that you want to display. Using a program that will allow you to modify the image resolution (Paint works on a Windows computer, Gimp on a Linux computer), reduce the image resolution to the required size. Most images are already 24 bit color, so all it is not necessary to modify the color depth.

Lab Instructions:

1. Start with one of your **720p** video designs.

2. Modify your design as needed. You will modify your pixel generator to include a frame buffer, and will display the image from the frame buffer.

3. Find an image to display (you may test with the provided images if desired), and convert your image to a .mif file using the script provided in this document.

4. To implement the full color resolution option, you will display each image pixel as a 4x4 array of pixels on the monitor. The resulting image will be highly pixelated, but colorful. If you select the 12 bit color resolution option, you will display each image pixel as a 2x2 array of pixels on the monitor. The image will be smoother but less colorful.

Submission and Grading:

Your submission will be a single .zip (or .7z or .tar) file uploaded to Canvas for Project/Lab Week 7. The submitted file should contain the following:

a) Submit a .qar file for your design. The image displayed is your choice, but cannot be the images provided with this lab, and cannot be an image submitted by another student in this class. Make sure your archive builds correctly.

b) You may choose to include a brief text file describing issues with your final design (if any), or anything else you want me to know about your submission.

c) This project is worth 35 points.

Part 1: Create your new frame buffer

In a 1280 x 720 display (720p), there are 921,600 unique pixel locations. The DE10_Standard board supports 24 bits (or 3 bytes) of color data per pixel, requiring 2.77 Mbytes of memory for a full pixel and color resolution display. This is more memory that is available in the CycloneV FPGA, so we cannot create a full pixel resolution, full color resolution frame buffer using the embedded memory blocks.

The CycloneV device on the DE10-Standard board contains 553 M10K memory blocks, or approximately 550Kbytes of memory. The 4x4 pixel design will use 172,800 Kbytes (about 31 percent of the memory). The 2x2 pixel design (with 4 bits per color) will use twice as much memory, or about 62%.

For the full color resolution option, create a frame buffer that displays each memory location 16 times, in a 4x4 square. Maintain the color resolution at 24 bits per pixel (8 bits each RGB). The resulting frame buffer size will be 320 x 180 x 3 bytes/pixel, or 57,600 words of 24 bit wide memory.

For the half color resolution option, create a frame buffer that displays each memory location 4 times, in a 2x2 square. Reduce the color resolution to 12 bits per pixel (4 bits each RGB). The resulting frame buffer size will be 640 x 360 x 1.5 bytes/pixel, or 345,600 words of 12 bit wide memory.

Expect your Quartus compile times to be longer than for earlier projects. **Simulation will be your best friend here, avoiding unnecessary Quartus compiles.**

1) Start with an existing 720p project.

2) You will need a memory initialization file (.mif) for this project. We will construct our .mif files as part of this lab, but when you generate your memory block, you can use an empty file.

3) With your project open, add a ROM from the IP catalog with the following characteristics:

a. 1 port ROM
b. 57600 deep x 24 bits wide, or 345600 deep x 12 bits wide
c. single clock
d. do not register the output port
e. include the memory content file
f. Allow In-System Memory Content Editor

Note that when you browse for the .mif file in the ROM dialog, by default the selector is looking for a .hex file. Just change the dropdown for the extension to .mif to locate your file. Why the Quartus GUI people didn't default to look for both is one of life's great mysteries.

4) Create a new version of the pixel generator module, and instantiate the memory.

5) You will need to generate an address to the memory based on your horizontal and vertical counters. To create the 4x4 pixel, you will not use the lowest order two bits of either address counter. To create the 2x2 pixel, you will not use the low order bit of either address counter.

6) You can test your design during development by creating a simple .mif file, or by using the In-System Memory Content Editor as before.

Part 2: Creating the .mif file for your image

I am certain you could do some very impressive and extensive image manipulation in Matlab, but for this class we will use Matlab to break out the RGB values of an image, and construct the resulting .mif file. The resizing (downsizing) of any image results in a loss of clarity. Use of more sophisticated image processing tools would help.

1. Start with an image you want to display. Using your photo tool of choice (Paint is a simple tool in Windows), resize the image to be either 320 or 640 pixels wide.

2. Process the image in Matlab to generate your .mif file. The Matlab script is available in Appendix A of this document.

Note: The Matlab script as written will work for the 4x4 pixel approach. You will need to make minor modifications to the script to work for the 2x2 version of your project.

I will provide a .mif files to assist with your project testing, but for your submission create an image of your choosing.

Part 3: Updating just the .mif file contents (Information only)

If all you want to do is use different .mif file contents in your design, but not change the hardware, there are several ways to do this. The most time consuming is to rerun an entire place and route, and this is not necessary. The name of the .mif file you are using is linked to the ROM memory block you created, so it is easier to keep the name of the .mif file the same.

The .mif file contents do not have any effect on the synthesized logic or the placement of the logic. After the Quartus fitter has run, the Quartus assembler step reads the .mif file contents and merges it with the .sof file. Each embedded memory bit has a corresponding value in the configuration file, and the .mif file contents are simply added to the .sof file.

Methods to use modified .mif file contents

1. In Quartus, run Processing > Update Memory Initialization File, then rerun just the programming file generation step, Processing > Start > Run Assembler.

2. If you want to create a script instead of using the GUI, run these two commands in the Quartus Tcl console:

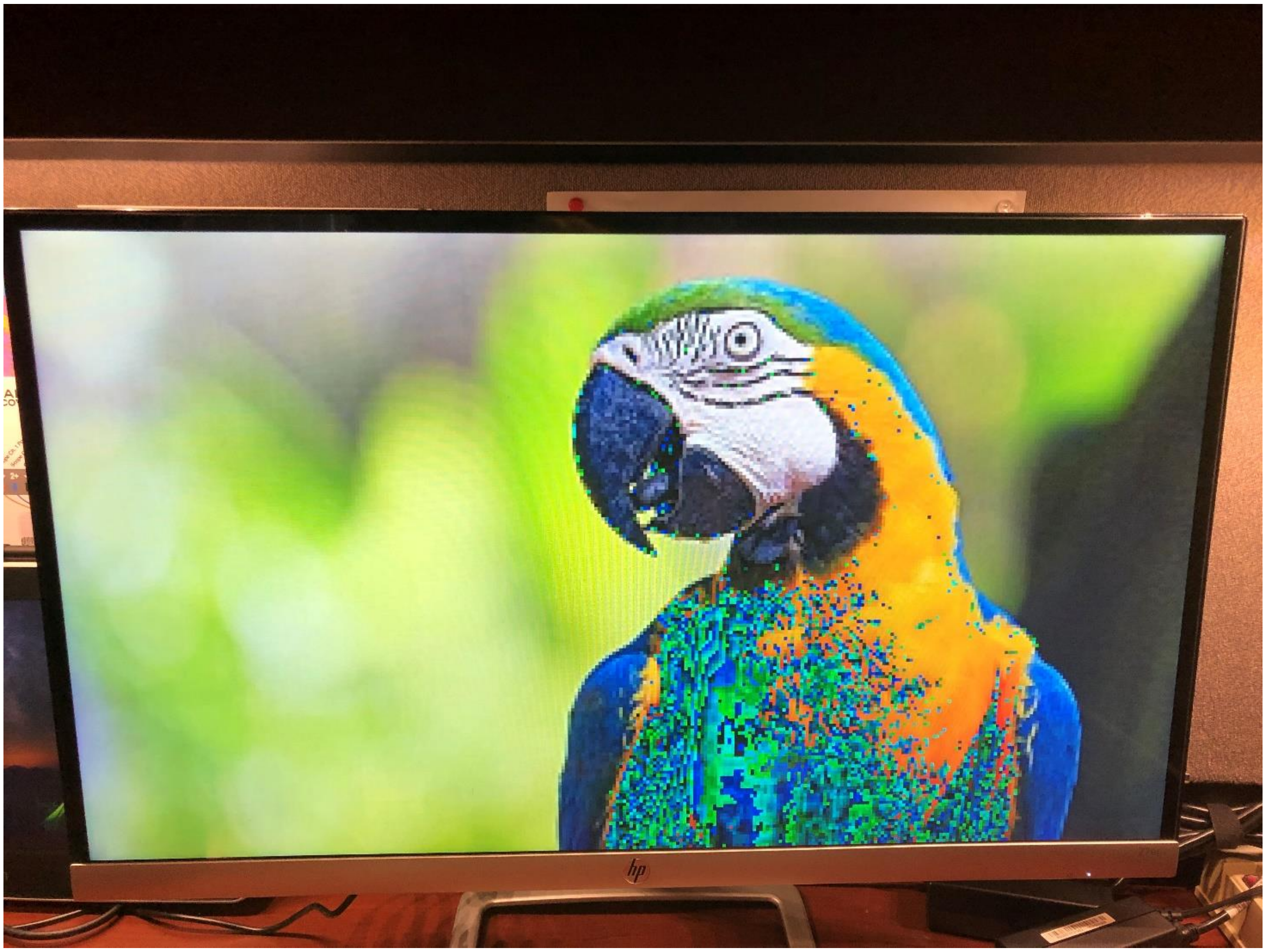
    a. quartus_cdb –update_mif <project name>
    b. quartus_asm <project name>

3. Use the In-System Memory Content Editor
    a. Edit > Import Data From File
    b. Note that this approach is only temporary, and does not update the configuration file, but is a useful approach for testing. You could also Tcl script this and update your display with different images at a desired interval.

If you do want to change the .mif file that is associated with the ROM block, you can do this by (carefully) modifying the .v file that is generated with the ROM IP. In the .v file, look for a line containing **altsyncram_component.init_file** and modify the filename. I do not know how this might affect simulation of the ROM component, so do this at your own risk.

Expected Results

I found a .jpg image of a parrot, and decided to use that. First the image was read into Microsoft Paint and the resolution was reduced to 320 pixels wide. The vertical dimension was left at whatever value resulted by maintaining the aspect ratio. This image was then processed in Matlab to create the .mif file, which was then displayed in 720p using the 4x4 pixels from this project. You can see the obvious pixelation, but the image is easily recognizable.



Appendix A:

This Matlab script is not very sophisticated, no doubt your could write a better version (if you do, please pass it along to me).

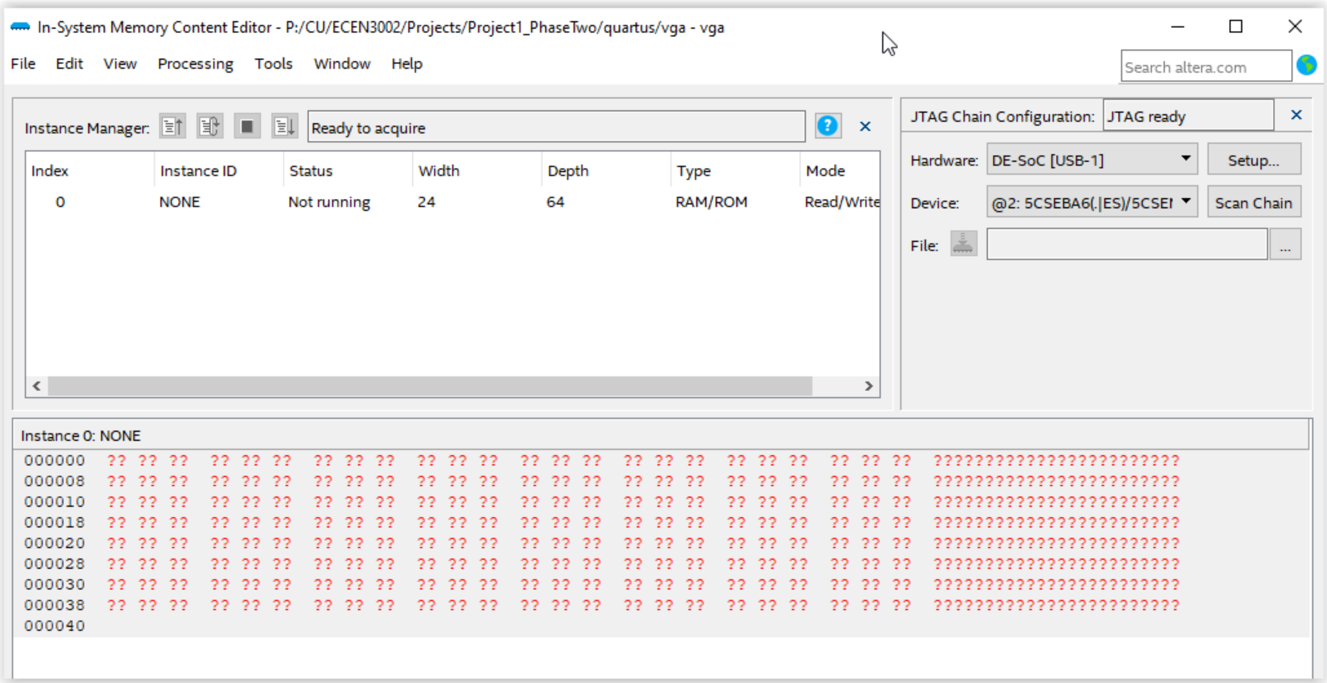
The script reads in an image based on the `depth` and `wordlen` values, and separates out the red, green, and blue color values. The script then constructs a .mif file with the appropriate headers, and writes the RGB data into the .mif file.

```
% Assumes picture is 320 pixels wide, 8 bits each RGB
% Framebuffer is 57600 24 bit words
pic = imread('<your_file_here.jpg_or_bmp');
depth = 57600;
wordlen = 24;
red   = pic(:,:,1);
red   = red';
green = pic(:,:,2);
green = green';
blue  = pic(:,:,3);
blue = blue';

% Write header info
fid = fopen('<your_name_here.mif', 'w');
fprintf(fid, 'DEPTH=%d;\n', depth);
fprintf(fid, 'WIDTH=%d;\n', wordlen);
fprintf(fid, 'ADDRESS_RADIX = DEC;\n');
fprintf(fid, 'DATA_RADIX = HEX;\n');
fprintf(fid, 'CONTENT\n');
fprintf(fid, 'BEGIN\n\n');

% Write RGB data for 320 pixel image
%for i = 0 : depth - 1
%    fprintf(fid, '%d\t:\t%x%x%x;\n', i, red(i+1), green(i+1),
blue(i+1));
%end

% Write RGB data for 640 pixel image
%for i = 0 : depth - 1
%    fprintf(fid, '%d\t:\t%x%x%x;\n', i, ...
        bitshift(red(i+1),-4), bitshift(green(i+1),-4),
bitshift(blue(i+1),-4));
%end
```

```
fprintf(fid, 'END;\n');
fclose(fid);
```

Appendix B: In-System Memory Content Editor (Same as Homework 7)

1. Open the memory content editor (Tools > In-System Memory Content Editor. In the JTAG Chain Configuration area, select the cable and Device @2. Your display should look like this:



2. Highlight the line in the white display area, then read the memory contents. This can be done by click on the leftmost icon with the up arrow, Processing > Read Data From In-System Memory, or by pressing F5. Your memory contents should now be shown.

3. In the hex display region, enter new values for addresses 0 - 2, then update the memory by either pressing the icon with the down arrow or by pressing F7. Experiment with changing other values that result in the color bars changing color or intensity.

The In-System Memory Content Editor can be used to read or write memory in a design. One limitation of this approach is that any memory you use must be created as a single port memory, as the editor uses the second port to access the memory.