Chapter 11

SRAM Video Frame Buffer

The purpose of this laboratory is to combine your video display controller with your SRAM memory controller to create a Video frame buffer. This video frame buffer will allow you to display images from the Micron memory onto the VGA display.

Exercises

Exercise 1 – Background

The development board that we are using in the lab contains one Micron 128Mb CellularRAM. We will use this memory to store several images for the 640x480 VGA display. A memory used to store a complete image of video data is often called a frame buffer. You will design a circuit that performs this frame buffer function. To store a full frame of data, you will need to store the color information of each pixel in the 640x480 display (307,200 pixels). You will need to allocate 8 bits of memory for each pixel to support 256 colors for each pixel. Each word of the Micron CellularRAM is 16 bits and thus can store two image pixels. Each 16-bit word in the memory will store the display information for 2 pixels as shown in the table below. Note that the pixel to display at the even pixel_x location (or pixel 0) is stored in the lower byte of the word and the pixel to display at the odd pixel_x location (or pixel 1) is stored in the upper byte of the word.

Pixel 1								Pixel 0							
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R2	R1	R0	G2	G1	G0	B1	B0	R2	R1	R0	G2	G1	G0	B1	B0

With two pixels stored in each word, you will need at least 640x480/2 or 153,600 words to store a complete image. To simplify the addressing of the memory, a complete row of the image (640 pixels) will be stored in 512 consecutive memory locations. The first 320 memory locations will hold the actual pixel data corresponding to columns 0 through 639 (only 320 memory locations are needed to store a complete row since each word can hold two pixels). Locations 320-511 of this consecutive block of memory do not hold any image data (this memory is wasted). Because 512 consecutive memory locations reserved for a single line, the lower 9 bits of the memory address are used to indicate the image column.

The information for each row of the image will be stored consecutively in memory. Memory locations 0-511 (0x0-0x1ff) will store the first row, memory locations 512-1023 (0x200-0x3ff) will store the second row, and so on. To simplify the memory addressing, 512 rows of memory will be allocated for the 480 lines of the 640x480 image. The first 480 rows of memory will correspond to actual image information and the next 32 rows will not be used. Because 512 rows of memory are allocated, the next 9 address bits can be used to indicate the image row.

While allocating enough memory for an image that is 1024x512 is wasteful in memory, this memory layout allows you to address the memory on even powers of two boundaries and will significantly simplify your frame buffer circuit. In total, this memory layout will require 512x512 words or 262,144 words to store a complete image.

Because this memory has 8M 16-bit words (2^{23}) and each image requires 262,144 words (2^{18}), you can store 32 different images in this memory. The first image is located at address 0x0, the second image is located at address

0x40_000, the third image is located at address 0x80_000, and so on. The top five bits of the memory address can be used to select the image you want to display.

The coordinates of two adjacent pixels on the 640x480 display will correspond to a specific memory address within the Micron memory. The pixel_x and pixel_y address of pixels on the VGA display controller correspond to specific memory addresses. To create the frame buffer, it is essential for you to understand this translation. The top 9 bits of the 10-bit pixel_x value will be used as the lower 9 bits of the memory address. The least significant bit of pixel_x will be used to select which byte of the 16 bit word to use. The 9 bit pixel_y value will be used to for bits [17:9] of the memory address. The top five memory addresses are determined by the image you wish to display. The table below summarizes the addressing this memory.

Address[22:18]	Address[17:9]	Address[8:0]
Image Number	Image Row	Image Column

The following example demonstrates this memory mapping. Consider the pixel locations (pixel_x = 136, pixel_y = 312) and (pixel_x = 137, pixel_y = 312) located in image 0 of the memory. These pixels are next to each other on the display so they will both be stored in the same word of memory. Pixel (136, 312) will be stored in the lower byte of the word (bits [7:0])and pixel (137, 312) will be stored in the upper byte of the word (bits [15:8]). The lower 9 bits of the memory address are determined by the top 9 bits of the pixel_x value. Since pixel_x = 136 or 0x088, the top 9 bits of this address are 0x44 (0x88 » 1) or binary 0_0100_0100. Since pixel_y = 312 or 0x138, the next 9 bits of the memory address are set to 0x138 or 1_0011_1000. Assuming we are using image 0 (the top 5 bits are zero), the address of these two pixels will be 0x0 « 18 | 0x138 « 9 | 0x44 = 0x27044 (0_0000 & 1_0011_1000 & 0_0100_0100 = 000_0010_0111_0000_0100_0100).

To test your understanding of the image memory layout, answer the following questions on Learning Suite:

Question: When organizing the CellularRAM memory as described in the laboratory write-up, how many words are not displayed or "wasted"?

Question: Which bits of a frame buffer word store the "Green" pixel values for the "odd" pixel (i.e., the pixel that is in an odd column)?

Question: Which memory address stores the pixel information for location (x, y) = (590, 157) in image #14 (i.e., the image using the 5-bit image value of 01110)?

Question: What image number, row, and column are associated with memory address 0x1C5D2C?

Exercise 2 – Video Frame Buffer

The primary design exercise is to create a video frame buffer circuit. A video frame buffer circuit works in conjunction with the VGA display controller and the memory controller to continuously read the frame memory and generate the pixel data to display.

One of the challenges of this assignment is that your memory controller must keep reading the memory at rate that is fast enough to provide pixel data to the VGA display. Each memory read of this memory will provide enough data for two pixels. The VGA controller you developed will require two clock cycles for each pixel or four clock cycles for two pixels. This means that you MUST be able to read a 16-bit word (i.e., two pixels of data) from this memory every four clock cycles.

Review your memory controller and make sure you can perform a read in four clock cycles. It is possible to complete a read in four clock cycles (i.e. in 80 ns) using the memories on the Nexys2 board. Also, you need to be able to back to back memory reads every four clock cycles. This means that you need to perform one memory read after another memory read without going through an IDLE state. You may need to modify your memory controller to support back to back read operations (back to back memory operations were not required in the memory controller lab). The ASM diagram below demonstrates how you can modify your memory controller state machine to support back to back memory read operations without going through the IDLE state.

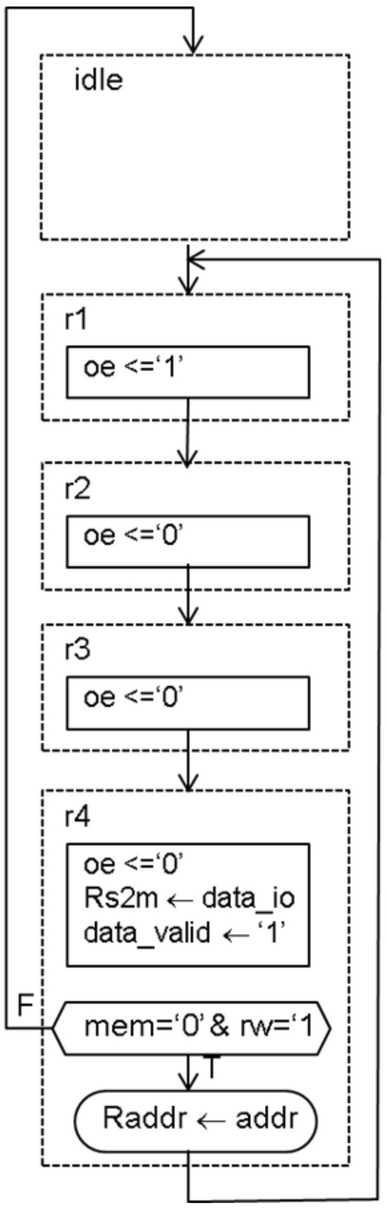

Figure 11.1: Frame buffer state machine.

Another challenge you must face when completing this lab is initiating the read operations soon enough for the data. Because it will take four clock cycles to perform a read, you need to initiate the read operation at least four clock cycles before the pixel data is needed. Further, you will need to initiate the reading of a new line before the line starts (sometime in the blanking period). Review the timing diagram below. This timing diagram demonstrates the reading of the memory, the pixel_x/pixel_y signals, and the pixel out.

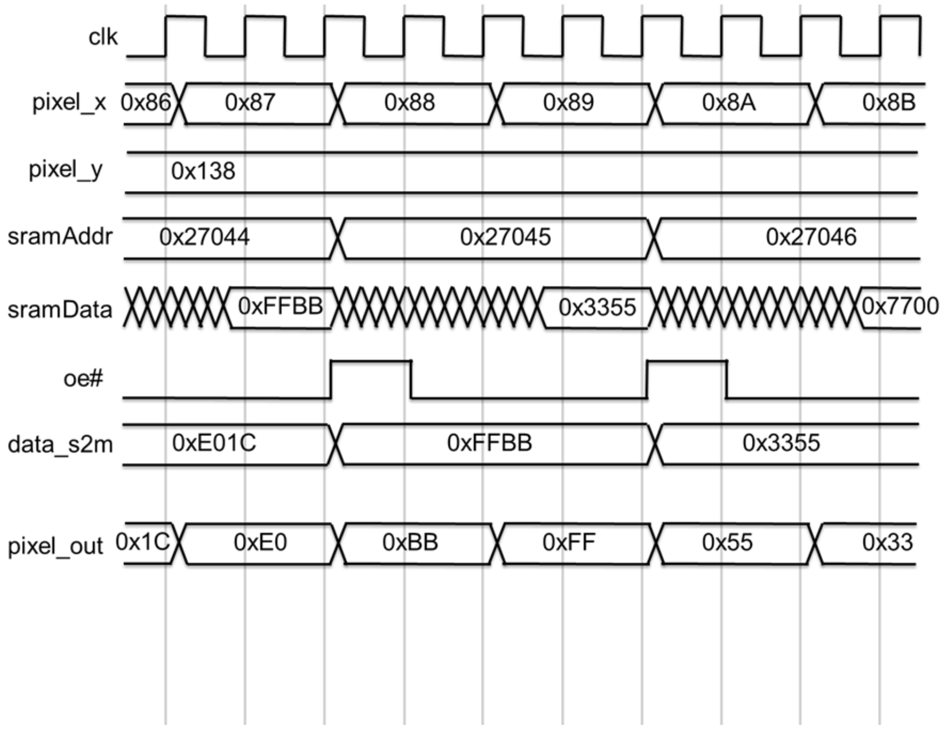

Figure 11.2: Frame buffer waveform.

Unlike previous labs, you will not receive as much tutorial or background material on how to build this circuit. Your frame buffer circuit should instance your VGA controller and your modified SRAM memory controller. The primary function of your circuit is to read the SRAM memory, store the results internally in the circuit, and pass the appropriate pixel values to the "pixel_out" in your VGA controller. Your circuit will use the five switches to determine which of the 32 images to display.

You will need to create a sequencer or state machine to control the memory address and the control signals in the memory controller. Feel free to work with a partner on the design and coding of this circuit. You may split of the coding between you and share your code. However, each student must turn in the code for the entire lab and complete a lab write-up as if he or she is working alone.

Question: Summarize any changes you had to make to your SRAM memory controller.

Upload: Copy and paste your frame buffer VHDL file.

Exercise 3 – Testbench

A testbench will be provided for you to help you simulate your frame buffer circuit. This testbench will connect your top-level frame buffer circuit to the Micron memory. Unlike previous testbenches, this testbench will NOT evaluate your circuit output and determine whether your circuit is working correctly or not. This testbench is given to you to help you see the operation or your frame buffer circuit in conjunction with the memory. You will need to carefully review the function of your circuit and convince yourself that the circuit is working properly.

To facilitate the use of this testbench, name your circuit "framebuffer.vhd" and provide the following top-level ports:

Port Name	Direction	Width	Purpose
clk	Input	1	50 MHz clock
btn0	Input	1	Push button 0 (connects to your internal asynchronous reset)
sw	Input	5	5 right-most switches (sw4, sw3, sw2, sw1, and sw0). The switches will determine which image you will display.
Hsync	Output	1	VGA horizontal sync
Vsync	Output	1	VGA horizontal sync
vgaRed	Output	3	VGA red pixel bits
vgaBlue	Output	2	VGA blue pixel bits
vgaGreen	Output	3	VGA green pixel bits
MemAdr	Output	23	The address pins to the memory device
MemOE	Output	1	Output enable pin to memory device
MemWR	Output	1	Write enable pin to memory device
RamCS	Output	1	Memory chip select
RamLB	Output	1	Memory ‘lower byte’ control signal
RamUB	Output	1	Memory ‘upper byte’ control signal
RamCLK	Output	1	Memory CLK signal
RamADV	Output	1	Memory ADV signal
RamCRE	Output	1	Memory CRE signal
MemDB	Inout	16	Data pins to memory

The memory model used in this testbench has been modified from the original Micron memory model. This memory model has been modified to help you debug your frame buffer circuit. Since you will not need to write memory in this laboratory assignment, this memory model will not properly simulate memory writes (i.e., if you write to a location in memory and then go back and read it, you will not read what you wrote).

For memory reads, this memory model has been modified to return the following values:

- The memory model will return ‘X’ if you read from a location of memory that does not correspond to an image pixel. For example, if you read from memory location 0x190 or decimal 400, you will receive all ‘X’s. This location corresponds to a column of memory that is not visible. This is done to help you see when you are reading from a memory location that should not be displayed on the screen
- For valid image memory locations, the memory model has been modified to return a valid value (i.e., not ‘x’s). The memory model has been modified to return a value based on the address. The bits of the 16 bit word you read are determined as follows:
  - Data[15:12] – address[21:18] (bottom four bits of “image number” portion of address)
  - Data[11:8] – address[2:0] & ‘1’ (bottom three bits of “column number” portion of address concatenated with ‘1’)
  - Data[7:4] – address[12:9] (bottom four bits of “row number” portion of address)
  - Data[3:0] – address[2:0] & ‘0’ (bottom three bits of “column number” portion of address concatenated with ‘0’)

This pre-populated memory was created to help you make sure that the value you read and display at a given pixel is correct.

For example, if you read from memory location 0x27044 (corresponding to pixel (136,312) and (137,312) as described earlier in the laboratory write-up), the memory will return: 0x0988. This value was obtained as follows:

- Data[15:12] = “0000”. This address location corresponds to “image number”. Since we are addressing image 0, the bottom four bits of the image number are “0000”
- Data[11:8] = “1001”. This corresponds to the least three significant bits of the address “100” concatenated with the value ‘1’
- Data[7:4] = “1000”. This corresponds to address[12:9] (bottom four bits of “row number” portion of address)
- Data[3:0] = “1000”. This corresponds to address[2:0] concatenated with ‘0’

When simulating your frame buffer with this memory, you should display a color value of 0x88 for pixel (136,312) and the color value 0x09 for pixel (137,312).

Simulate at least one full frame to demonstrate the proper operation of the memory reading and the VGA control signals. Make sure you are displaying the correct value at the start of the line, end of the line, start of a frame, and end of a frame. When you are convinced your circuit is working properly, proceed to the download phase.

Exercise 4 – Download

After you have successfully verified your design in VHDL, you are ready to generate a bitfile for your top-level frame buffer. To complete the generate bitstream process:

- Make sure you provide a UCF entry for each input/output/inout of the design
- Make sure each pin is "Mapped" (see the place and route report)
- Review all of your synthesis warnings and make sure you understand and accept each warning

After generating a valid bitstream, you are ready to download the bitstream on your board. Before downloading, however, it is helpful to initialize the memory with some known pattern. To help you verify your frame buffer circuit, I have created a circuit that can be used to initialize the memory. Download the memoryInit.bit bitfile and use this bitfile to initialize the contents of the memory. The instructions for initializing your memory are as follows:

- BTN0: When this button is pressed, the 8-bit value on the switches will be written to ALL memory locations (each memory location will be loaded with the 16-bit value sw & sw). This button can be used to initialize the memory to a specific value. If you run your frame buffer after pressing this button, you should see a single color on the screen
- BTN1: When this button is pressed, the value on the switches will be written to all valid image locations for a single image. It will NOT write the value on the switches to locations in memory that do not correspond to the displayable region of the frame buffer. Only one image will be written and the image that is written is determined by the "Image register" (see instructions for BTN3)
- BTN2: When this button is pressed, a checkerboard pattern will be written to the memory in all valid image locations for one image. The color value of one checker color is determined by the switches and the other checker color is the inverse of the value of the switches. Like BTN1, the image that is being written is determined by the "Image register"
- BTN3: When this button is pressed, the least 5 significant bits of the switches will be written into the "Image" register. The image register is used to determine which image is being written when either BTN1 or BTN2 is pressed

The following example demonstrates how you can use this bitfile to initialize your memory:

1. Set the switches to "11100000" (all red). Press BTN0 to initialize the entire memory with the color red
2. Keep the lower 5 switches set to "00000" and press BTN3. This will set the internal Image register to 0 (this will be displayed on the seven segment display)
3. Set the switches to "00011100" to specify the color "Green". A green block should show up on the display. Press BTN1 to write the Green color to every pixel in image 0
4. Set the lower 5 switches to "00001" and press BTN3. This will set the internal image register to 1
5. Set the switches to "000000011" to specify the color "Blue". Press BTN2 to write a Blue/Yellow pattern into image 1

At this point, you have loaded a full green image in image #0, a checkerboard image in image 1, and red images in all of the other 30 images. You should download your frame buffer circuit and see if these images are properly displayed on the VGA screen.

Personal Exploration

There is no personal exploration required for this laboratory assignment.

Pass Off

Unlike previous labs, this is not an “all or nothing” pass off. This is a difficult lab and partial credit will be given to students who are able to get the lab partially working. You can decide whether you want to complete the lab in full or get partial credit. The following scale will be used to pass off your lab with partial credit:

- **100%** – The frame buffer circuit works perfectly and the “checkerboard” image appears properly on the screen
- **90%** – The frame buffer circuit works mostly correct but there is one small error (a corner case—for example, the last column is not displayed correctly, or there is a column out of place)
- **80%** – The frame buffer circuit works mostly correct but there is more than one noticeable error in the display
- **70%** – The frame buffer circuit sort-of works. There is something being displayed on the screen that is related to the memory but there are several obvious problems with the display

Demonstrate a working design to the TA by demonstrating the ability to read from SRAM and display it on the screen.