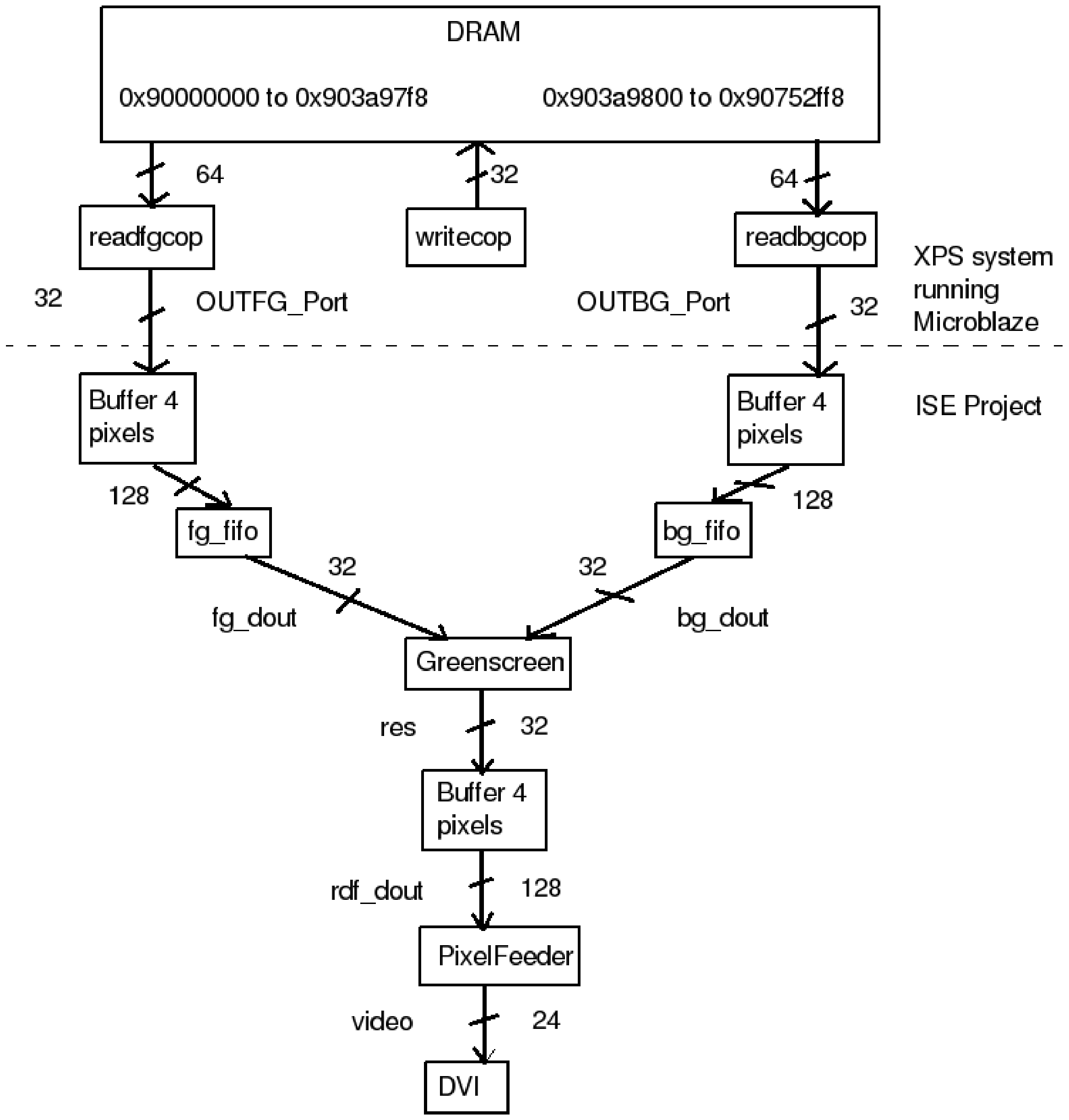
CS150 Final Project Report
Greenscreen
Spring 2013

Amy Shieh
Abirami Thevanayagam
Github team 4

1. Project Functional Description and Design Requirements

We tried to implement a green screen accelerator in real time. We ended up implementing a green screen accelerator for static images we stored in DRAM. We created an XPS system that implements Microblaze coprocessors to read and write from DRAM. Then, we created external ports for those coprocessors so that we could read from DRAM in our ISE project. From the ISE project, we then implemented our green screen algorithm in Verilog and tried to send our data to the DVI core, which would output our processed image to the monitor.

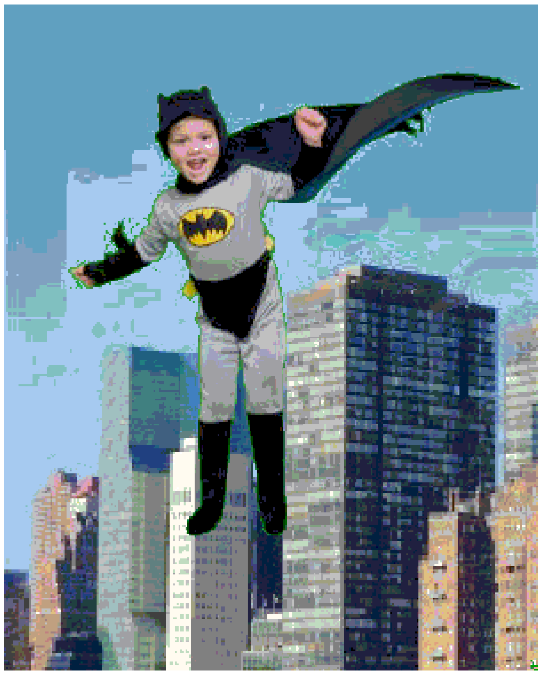
2. High level organization



3. Detailed Description of Sub-pieces

Software implementation of algorithm:

We implemented our greenscreen algorithm in Matlab to do some fine tuning on the tolerances used in the algorithm. We tested different algorithms, as well as different image formats such as RGB and HSL. We finally chose RGB although HSL was simpler because the DVI core takes images in RGB format. At first, we wanted to implement an algorithm that would produce a very high quality image, using different tolerances for the edges between the foreground and background. However, to make our computation simpler, we finally decided on an algorithm based on computing the Euclidean distance between each pixel value of the foreground and that of our target color (green). Based on the tolerance and the computed Euclidean distance, we used a multiplexer to decide which pixel would get incorporated into the final image. To prep our images for processing, we used Matlab to get the pixel value arrays in hex.



Microblaze coprocessors:

Read coprocessors: readfgcop, readbgcop

We have two read coprocessors that read from two different address spaces of DRAM. readfgcop reads from 0x90000000 to 0x903a97f8, which is the foreground image for our green screen algorithm. readbgcop reads from 0x903a9800 to 0x90752ff8, which is the foreground image for our green screen algorithm. Each image is 800x600 pixels, and we are using RGB format to store the information for each pixel, so each pixel takes 3 bytes to store. We have 8 byte aligned memory, so each 8 byte aligned address in memory can store a pixel.

Write coprocessor: writecop

We have one coprocessor that writes to DRAM. First, we start with a text file containing all the pixel values of an image in hex. Then, we send that data over serial UART and then have a C program running in Microblaze to intercept that data. The C program buffers up 6 characters coming in, converting each character into hex, then stores that pixel value into DRAM.

ISE project:

We added some external ports to our XPS project so we could access the data that the read coprocessors were getting from DRAM. Then we added FIFOs to buffer up that data to make sure that the foreground and background images would be synchronized when being processed. The data coming out of the FIFO was sent into the greenscreen module, and the data coming out of the greenscreen module was sent into PixelFeeder, which sent it to DVI.

4. Status and Results:

Getting data into DRAM posed a larger challenge than we expected. At first, the DRAM controller seemed to have a hard time processing the large amount of data we were sending. For that reason, we got rid of line breaks in our data, thus reducing the load by ⅓, since had had a line break after each nibble.

Halfway through the project, we decided that we were going to implement static image processing as opposed to real time video image processing. This decision was based on the complexity of the VGA core, as well as it not being a core part of our project. This enabled us to spend more time working with DRAM as well as the DVI core, which were crucial parts of our process.

For a big part of our project, our XPS project was corrupted and did not give us expected results when importing into ISE. This was a big setback as it was extremely difficult to debug and we were finally forced to rebuild our system, losing a lot of time. However, we overcame this obstacle and our XPS system is now functional.

All of our modules (DRAM, green screen module, DVI) are working individually. We can read and write to DRAM sending data over serial UART, and we can access this data in our ISE project. Our green screen module also works as intended. DVI is working not only in the MIPS skeleton framework we started out using but also in the ISE project when given a constant color.

Our entire project does not run as intended due to timing constraints that we encountered. We are not processing our data fast enough for the 50MHz clock speed for the DVI core. The bottleneck here is that we are only reading two words at a time from DRAM. We attempted to fix this issue by reading eight words; however we ran into errors and ran out of time to debug it Although we are not getting the correct output from the DVI core, we have used Chipscope to confirm that the flow of data is correct up until after the greenscreen module. Logically, our flow of data is correct for the entire system.

<u>5 Division of Labor:</u>

When working on the software model, we tested different algorithms individually. However, when implementing hardware, we found that it was best to work together as to be completely in sync on the design decisions being made. Also, this allowed for easier debugging as well allowing both of us to learn about the different tools we used.

6. Conclusions:

We learned that the Xilinx tools have a very large learning curve and can be quite unintuitive to use. As compile time and synthesize times were quite large, we learned to carefully check our code as we wrote it. We also learned a lot of interfacing, as well as FSL and NPI protocols, from reading datasheets. As we struggled with a lot of timing issues, we also learned quite a bit about timing in our design.

If we could go back in time, we wish we could have explored the different options we had for the interfacing our different modules. For example, if we had done our whole project in XPS rather than importing it into ISE we could have saved a lot of time and debugging. If we had known about the timing sensitivity of the DVI core, we would have explored different options for DVI, such as sending our processed data back over UART and then displaying it in Matlab. We would also have tried to compare notes with our classmates earlier on, as we could have all saved ourselves some time.

7. Statistics:

LUTs: 5858

Timing: 62.5MHz

Hours spent on project: 220