Michael Chan, Siddhanta Shrestha

ECE 332: Lab Report 4

**Introduction**

This lab was a culmination of everything we have learned in this class. It is a combination of lab 2 and 3 where the image captured was first underwent image processing (by turning the image black and white) then was compressed using Run Length Encoding (RLE). Because this compression process was lossless, the image looked exactly like the original. Only difference being the reduction of picture size.

**Detailed Procedure**

Most of the procedure was given to us in the lab overview thus the process was to simply follow it. We already had the RLE component done using Lab 3, and the Image processing part done using a part of Lab 2. We simply had to add the 8 PIOs into QSYS to make sure that the two systems could properly interact with each other. We did this by modifying the connections so that the system knew what communicated with what. Then we needed to compile the design which was the longest part of the lab. After that we needed to simply implement the PIOs using code so that they did what we wanted them to do in this lab. First we needed to capture the image on the screen when a button was pressed. This code was previously written in Lab 2. Then the image was converted to black and white with a second button press, again written in lab 2. We then had to capture 8 bits of data at a time from the original image and save that data onto the FIFO buffer. This was then sent over to the RLE which then outputted a 32 bit encoded segment, of which we only needed 24 bits of. This was then saved onto a FIFO buffer which then was outputted onto the screen when a button was pressed.

**Hardware Changes**

There were some major hardware changes for this lab done in QSYS so that the DE1-SoC board can interact with the RLE state machine we made in lab 3. The first thing we added was a ***RESULT\_READY\_PIO*** which allows the FIFO buffer to communicate with the ARM core. This parallel I/O lets the ARM core know that the buffer is full with encoded data segments to be displayed. The next hardware addition was the ***RLE\_FLUSH\_PIO*** which essentially empties the RLE at the end of bitstream once the FIFO is no longer sending any more data to the RLE. ***RLE\_RESET\_PIO*** is what is used when the ARM core wants to initialize the RLE encoder. The reset and flush PIOs are what ensures that the RLE unit is ready for the next round. ***IDATA\_PIO[23:0]*** is where the encoded bit is stored post RLE and is then transferred to the FIFO buffer to then be sent back to the ARM core. The ***ODATA\_PIO[7:0]*** is the opposite where it receives the bits from the FIFO buffer which is sent to the RLE encoder. Both of these PIOs are necessary since the original image captures 8 bits at a time while the RLE outputs 32 bits. Thus it is necessary to have these PIOs so that the correct amount of bits are transferred from one location to another. ***FIFO\_IN\_FULL\_PIO*** is what is used by the FIFO buffer to indicate to the ARM core when it is unable to receive any more bits until the buffer is emptied. This ensures that no bits are lost in the process since the ARM core waits to send more data. ***FIFO\_IN\_WRITE\_REQ\_PIO*** is used by the ARM core to request to write bits into the FIFO buffer. If the buffer is full then the previous PIO will indicate that to the core. This is essentially the same with the ***FIFO\_OUT\_READ\_REQ\_PIO*** which is used to see if the core can start reading the FIFO buffer. These two PIOs are necessary for the core to ensure that the buffer is ready to be written to/read from by the ARM core. The same necessity as the full PIO, making sure that no data will be lost during the transportation process. We updated the DE1\_SoC\_With\_D5M.v to instantiate our verilog modules. These were the RLE encoders and the FIFO send/receive buffers. This is important because the PIO connections we added earlier are used as the inputs and outputs. We also connected our modules to each other by adding wires assignments.

**Software Changes**

With lab4 being cumulative, the software changes that we implemented were improvements of the prior lab’s algorithm. While we already figured out how to convert a colored image into black and white, the next step was to convert that black and white image to 1 bit pixels in preparation for transfer to the RLE. These one bit pixels also needed to be stored in 8 bit arrays. This was accomplished by figuring out how many 8 bits were needed in an image.

240 x 320 = 76800 bits / 8 = 9600

Using the calculation above, we determined that a character buffer of size 9600 would be sufficient to store our image in an 8 bit array. To send data through to the RLE, we had to use the alt\_write\_byte(), alt\_read\_byte(), and alt\_read\_word() functions. These were necessary for the software to communicate with FPGA RLE (Hardware). We also didn’t not write a compression algorithm in C because the point of this lab was to use software to communicate with the hardware to perform that function. We wrote our software functions based on the block diagram. The ARM core was used to send and received data that we requested to operate the RLE

Another change we needed to make was to store the data received from the RLE (32bits) on to the SDRAM. This was accomplished by using the alt\_write\_byte(), alt\_read\_byte(), and alt\_read\_word() again to store the information from the IDATA\_PIO on to the SDRAM at the address of 0xC0000000. Finally, we had to write an algorithm to decompress the data from 32 bits back to our black and white image to be shown on the VGA port along with the compression ratio. As mentioned earlier, the compression ratio was different depending on how many bits were used. The uncompressed image is known to be 76800bits, while the compressed image depends on how many alternating black and white pixels. A screen captured image of only black pixels will have a compression ratio of 76800/32 = 2400 or 76800/24 = 3200. The 2400 ratio is the actual value, while 3200 value is due to us only using 24 bits from the output of the RLE.

**Problems Encountered and Solutions**

We had many issues with getting the ARM core to communicate with the RLE. The first was not being able to get the correct bit readings on the RESULTS\_READY\_PIO. This was due to the program using the incorrect .sof file. We needed to download the “time\_limited.sof” file in order to use the updated Qsys configuration.

Another issue we encountered was with the image received from the RLE. The image was shifted 8 bits to the right on the entire image. We found this to be from improper order of commands when writing to the RLE. Instead of the sequence below, we wrote to the FIFO\_IN\_WRITE\_REQ\_PIO first causing the first 8 bits to be unknown values entered into the RLE. Writing to the ODATA\_PIO\_BASE first ensures that there are known values for the RLE to accept.

| alt\_write\_byte(**ALT\_FPGA\_BRIDGE\_LWH2F\_OFST** + ODATA\_PIO\_BASE, bitarr[y]); alt\_write\_byte(**ALT\_FPGA\_BRIDGE\_LWH2F\_OFST** + FIFO\_IN\_WRITE\_REQ\_PIO\_BASE, 1); alt\_write\_byte(**ALT\_FPGA\_BRIDGE\_LWH2F\_OFST** + FIFO\_IN\_WRITE\_REQ\_PIO\_BASE, 0); |
| --- |

**Results and Conclusions**

The resulting image after compression was no different than the original because we were doing lossless compression which means that the image quality is preserved. This is essential when trying to decrease the file size of the image without losing any image quality. This lab was really interesting because it was a great way to tie all the labs together to show how everything is working.