COEDQ5 Project Report

Group 27

Valentina Salguero – 400017555

Veneta Grigorova – 400031820

salguerv@mcmaster.ca

grigorov@mcmaster.ca

November 27th, 2017

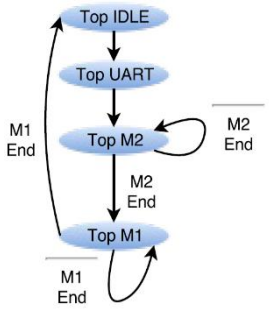
INTRODUCTION

The aim of this project was to design and implement the custom McMaster Image Compression specification in hardware. For the purposes of this project the image is already compressed with the ".mic11" specification. This meant the requirements of this project were only related to decompression. The decompression steps were divided into 3 milestones. The output of milestone 3 will be the input of milestone 2. Milestone 2 is where the Inverse Signal Transform (IDCT) occurs. In this milestone and 8x8 block of image data is retrieved from the memory. The IDCT is performed on those fetched values and written back to the YUV memory locations. This milestone ends when all the Y values, and the downsampled U and V values have been calculated and stored to memory and output. The output of this milestone will be the input to the milestone 1. Due to horizontal downsampling, only even U and V chroma values contain information. Milestone 1 performs interpolation of the U and V planes (Equation 10) and colourspace conversion (Equation 11). Odd U/V columns are calculated using a FIR filter (Equation 10). Since there is no vertical downsampling for U/V, the rows will be processed individually. As well, the Y plane contains the values for pixel brightness, which human eyes are sensitive to, and will be processed individually. As sections of the U and V plane are restored, colourspace conversion of YUV values to RGB values is initiated. This milestone ends when all the RGB values have been calculated and inputted into the memory and output. The output of this milestone is the final decompressed image.

DESIGN STRUCTURE

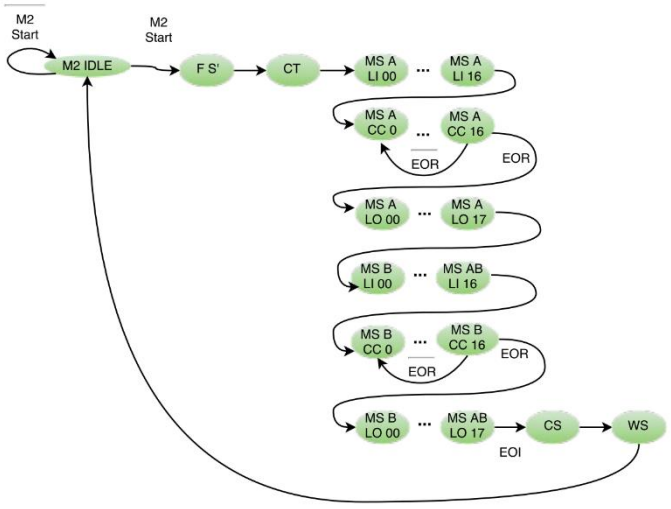
This project consists of 3 main modules: The top-level FSM, milestone 1, and milestone 2. The reused modules are UART_SRAM_interface.v, VGA_SRAM_interface.v, Clock_100_PLL.v, SRAM_Controller.v, VGA_Controller.v, PB_Controller.v, and UART_Recieve_Controller.v, dual_port_RAM.v. These modules are taken from Lab 5 experiment 4a where a similar setup was used. Thus, there is no need to reimplement them. The custom modules are milestone1.v and milestone2.v.

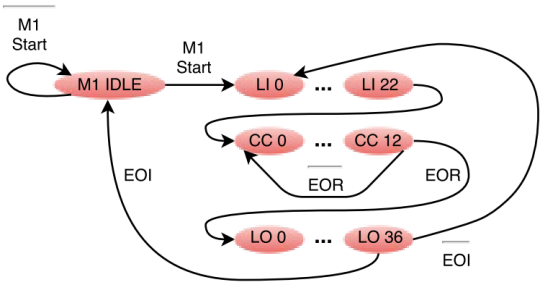
The original experiment4a.v is now titled project.v and acts as the top-level FSM. A new state was incorporated into the top FSM for each milestone that was implemented. Also incorporated were end signals that control when to transition from milestone 2 to milestone 1 to top-level idle. Furthermore, access was assigned to the SRAM for UART and VGA at the appropriate time for each milestone that requires their use. This is done by wiring together the SRAM input signals (address, write data, and write enable) to the output signals of the particular module that has control at that time.



The FSMs for each respective milestone are activated by start signals from the top-level FSM in project.v. The termination of the lower-level FSMs is controlled by end-of-image signals based on row and column counters. Furthermore, in milestone 2, the dual-port_RAM.v module is instantiated three times, in order to create three DPRAMs for storing S', T, and S values during IDCT computation.

In practice, the compressed image files are accessed in the FSM from the UART through serial communication. However, during simulation it is assumed that the compressed image files will already be loaded to UART and files are sent directly to the top-level FSM to be decompressed. The bypassing of the UART is achieved by transitioning from top-level idle state directly to milestone 1. For this to work, the files must be manually included as done in the beginning of the code.





IMPLEMENTATION DETAILS

MILESTONE I

The tasks in Milestone 1 were to perform FIR Interpolation and Colour Space Conversion (CSC).To interpolate the U/V value for an odd column pixel we needed to know the values in the previous three even columns and the next three even columns. This is implemented as a circular shift register with six registers. After every six cycles of rotation the next even value is inserted by multiplexing the input to the first register.

Rather than using a multiplexer to select which even value to process, the value from the last register is fed to a combinatorial multiplier circuit where it is multiplied by a corresponding FIR coefficient at every rotation step. The coefficients are fed to the multiplier one at a time through a multiplexer.

The results from the multiplication are added/subtracted to/from an accumulator register, which is initialized to value 128 at the beginning of each 6-step cycle, by multiplexing its input. After all six steps are completed the accumulated value is divided to 65535 by shifting it to the right 16 times.

Since multipliers take up many gates, it was important to ensure that only one multiplier for the FIR module is implemented, but not use use different ones for each multiplication. This was achieved by explicitly multiplexing the operands fed to the two inputs.

Two of these FIR modules are used – one for interpolating the U values and one for interpolating the V values.

The Colour Space Conversion requires multiplication of matrices. Again, only one multiplier is used and the operands are multiplexed to the inputs. Adjusted Y, U, and V data is multiplexed to one of the inputs while five coefficients are multiplexed to the other input. The result from the multiplication is selectively accumulated into three accumulator registers, which are initialized to zero at the beginning of each 5-step cycle. After all five steps are completed the accumulated values are divided to 256 by shifting them to the right 8 times, and are latched into one of three buffers – for R data, for G data, or for B data.

Two of these CSC modules are used – one for the even columns and one for the odd columns.

To maintain the flow of Y, U even, and V even data into the FIR and CSC modules and R, G, and B data from the CSC modules to the SRAM, FIFO buffers with three registers in each are used.

PROCESSING PERIOD

Given that the U and V values for odd pixels are interpolated using data from 3 even pixels before and 3 even pixels after, it makes sense to process the data 4 pixels at a time. Writing 4 pixels takes 6 clock periods, as each pixel consists of 3 values and the values are stored two by two, e.g. R0G0, B0R1, G1B1, R2G2, B2R3, G3B3.

NUMBER OF MULTIPLICATIONS

For every pixel, 5 multiplications are needed to perform CSC conversion. For every odd pixel, 2x6 multiplications are needed to perform FIR interpolation.

Thus, if we process 4 pixels at a time, total of $4 \times 5 + 2 \times 2 \times 6 = 44$ multiplications are needed.

MULTIPLIER UTILIZATION

Since four multipliers are used, the average utilization is

$$\frac{number\ of\ multiplications}{4 \times number\ of\ clock\ periods\ per\ processing\ period} = \frac{44}{4 \times 13} = 85\%$$

READING FROM AND WRITING TO SRAM

Since 4 pixels are processed during each processing period, the data for the same number of pixels should be read from and written to SRAM. Reading data for 4 consecutive pixels requires 4 reads due to missing data for the odd pixels. For example, reading Y0Y1, Y2Y3, U0U2, V0V2 provides all the data for pixels 0, 1, 2, and 3. Writing data for 4 consecutive pixels requires 6 writes: R0G0, B0R1, G1B1, R2G2, B2R3, and G3B3. Thus, during one processing period of max 13 clock cycles, 10 SRAM locations are to be accessed, leaving 3 clock cycles to output the addresses for the next period (considering the latency of the SRAM.

DEBUGGING

While working on Milestone 1, we encountered many data errors. It was important to trace the data along the entire path, from SRAM, through the FIR and CSC modules and back to SRAM. We created an Excel spreadsheet with formulas, which converts given SRAM address to corresponding row number un column number, so we could easily find out which pixel data (and for which colour) is incorrect. Later, we implemented the formulas into the test bench, so it was printing row and column instead of an address.

Since we needed to increment and keep track of 4 different addresses (Y, U, V, and RGB) it was important to know whether a correct address was output at the right time. Again, we used a spreadsheet to generate the start and end addresses for every row of data.

We also modified the test bench to ignore the clipping errors until we figured out the rest of the issues. Since clipping occurs ~2400 times in the motorcycle frame, this reduced significantly the number of errors to sift through.

MILESTONE II

Because of the difficulties we encountered with addresses in Milestone 1, we took different approach in Milestone 2. All addresses are generated by combinatorial circuits, which are fed the row and the column numbers.

The ICT conversion in Milestone 2 requires multiplication of 8x8 matrices. We implemented this again by using 4 multipliers, and multiplexing needed operands to their inputs.

Since each of the multipliers is processing one row by one column during one processing cycle, an entire row (column) of the result is obtained if all 4 multipliers are used twice during a processing period.

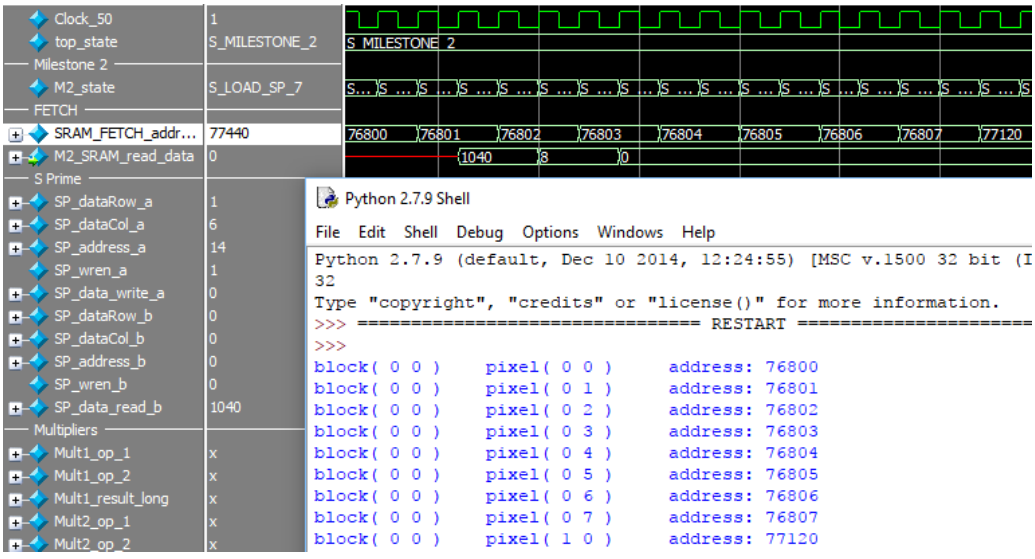
Thus, our common case consists of 16 clock cycles.

The constants are hardcoded into the FSM for ease and to save memory in the DPRAMs.

Despite considerable efforts, milestone 2 was not completed. The addressing of the SRAMs and DPRAMs is performed correctly. However, the final values written to the SRAM are not correct.

DEBUGGING

One of the most significant challenges posed by milestone two is the addressing of various memories. Due to the different arrangements of data storage in the DPRAMs and SRAM, the separate formulas for were implemented into hardware to generate read and write addresses for each memory. The result of each circuit was verified using a program in Python which computes all the types of addresses correctly. Its output was used to compare the addresses in ModelSim wherever there was a mismatch.



DISTRIBUTION AND PROGRESSION OF TASKS

WEEK	VENETA	VALENTINA
WEEK 1	▪ Read project document	
WEEK 2	▪ Made state table for Milestone 1	▪ Read project document
WEEK 3	▪ Coded Milestone 1	▪ Coded Milestone 1 ▪ Set up ModelSim for Milestone 1
WEEK 4	▪ Finished coding for Milestone 1 ▪ Debugged Milestone 1	▪ Debugged Milestone 1
WEEK 5	▪ Made state table for Milestone 2 ▪ Coded Milestone 2 ▪ Debugged Milestone 2 ▪ Wrote report	▪ Coded part of Milestone 2 ▪ Debugged Milestone 2 ▪ Wrote Report

CONCLUSION

In implementing this project, the group realized there was a steep learning curve. The insights gained extended beyond image decompression, to debugging, time management, and communication skills. The milestones gave a challenging introduction to proper debugging skills as well as re-enforcing the importance of planning using state tables. The state tables made it easier to compare what was expected to what was actually happening in the code. It became clear that although many bugs were caused by human error, timing played a big role in the bugs as well. If an address was not read at the proper time, or the wrong location was written to, the code would crumble. As well, it became clear that just because an error was detected in a certain state, the cause of the problem could have happened at any point prior to that point. Furthermore, understanding how the modules interacted played a big role in transitioning from simulation to testing on the board. These bugs and their solutions, had been experience in some way during the labs and lectures. As such, the project was an ideal way to combine the knowledge from this course.

Overall, the project was long and challenging. It was near impossible for 2 people to finish the project in 2.5 weeks. Without the help of our peers, our final code would not work in the same way. Taher, and David gave us a mini tutorial on how and why to change parameters within the code to transition from simulation to the board. As well, Suhaib and Faris suggest a work around for our slightly off calculations in milestone 1 as they had previously experienced the same issue. The last weekend was an overall group effort in helping each other with final coding, debugging and filling in each other's knowledge gaps. In the end, there was more information and experience retained from the project, than from the labs and lectures. We feel the experience of being tasked with a simplified model of a real-world project has given a great base to continue developing our knowledge for upper-year projects and future careers in the field.