COMP ENG 3DQ5:

Final Project

Due: November 27th, 2017

Dr. Nicola Nicolici

Ryan Ganeshan
1322407

Moshiur Howlader
1316948

Introduction:

For this project, an image decompression algorithm (.mic 11) was implemented in hardware. This project consisted of three milestones, of which Milestone 1 was fully completed, and Milestone 2 was partially completed. Milestone 3 was not attempted. Milestone 1 required us to implement a 6-tap filter, and a Colour Space Conversion scheme. We were successfully able to grasp all the concepts related to Milestone 1. The verilog code passed both testbenches and was also put on the board successfully. This was confirmed by the CE 3DQ5 TAs. Milestone 2 requires the implementation of inverse discrete cosine transform (IDCT). We were able to complete the skeleton of the fetch stage and the skeleton of the compute T stage. These skeletons had the logic correct for the operations needed in the respective stages. When we tried to implement the full layout of the compute T stage, we experienced bugs. We reverted our progress due to time constraints. We will outline these Milestone 2 details in this report. We had some critical issues with our Milestone 1 completion timeline, which cut into our allocated time to work on Milestone 2. Thus, we did not complete Milestone 2 as planned.

Design Structure:

Below is the circuit that we have implemented in this project. Milestone 1 and 2 work separately.

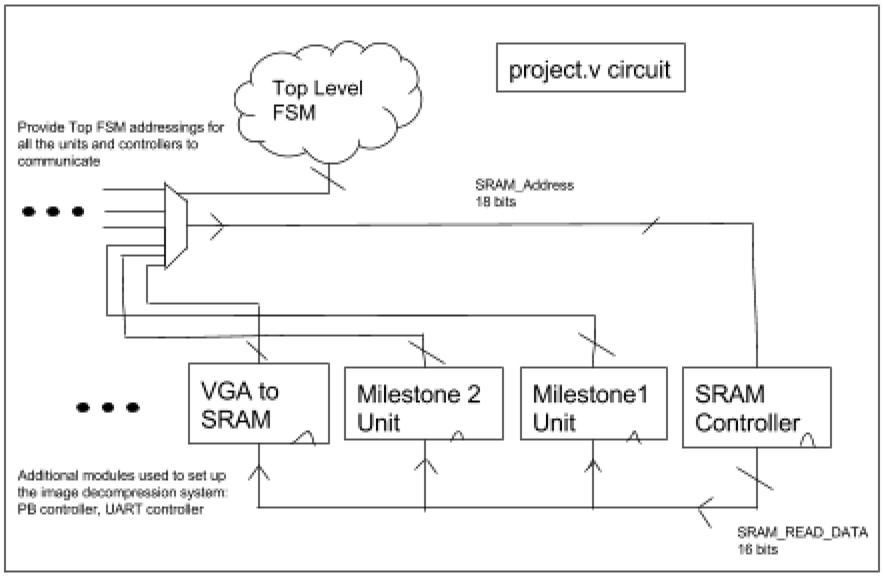

project.v circuit
Top Level FSM
Provide Top FSM addressings for all the units and controllers to communicate
SRAM_Address 18 bits
VGA to SRAM
Milestone 2 Unit
Milestone1 Unit
SRAM Controller
Additional modules used to set up the image decompression system: PB controller, UART controller
SRAM_READ_DATA 16 bits

Figure 1: Project Top Level Control Overview

The main change implemented in the top level FSM was the combinational logic to move from one top level state to another. The M1_start and M1_finish signals communicate with the M1 unit for it to work together with the top level. The same is true with the M2_start, and M2_finish. There were no additional custom modules added. The diagram above show most modules used. PB_controller, VGA_controller, and UART Controller was also used.

Implementation Details:

Milestone 1:

Intro

For this milestone, we were tasked with implementing the color space conversion and interpolation stages of decompression using the .mic11 standard. This milestone takes post-IDCT data and first interpolates values to produce double the U and V sampled that initially exist after IDCT transformation. Our logic for performing color space conversion, interpolation, and then writing post color space conversion (CSC) values to the SRAM is as follows. We would first gather U and V values necessary for interpolation as defined by the project document (U/V$_{[(i-5)/2]}$.....U/V$_{[(i+5)/2]}$). We would then perform interpolation on these values using the 6-tap filter outlined in the project manual. We would compute partial products, then accumulate these partial products in an accumulator register. We had to use the partial product technique due to the limitation of having four multipliers. The color space conversion calculations were done in the same fashion. We used the CSC formula outlined in the lab manual and exploited the matrix zeroes to simplify it. We stored repeatedly used values in buffers so that we would not need to waste our multiplier when values are again needed later in the calculation. The R, G, and B formulas that we used are outlined to the left.

$$R = \frac{76281(Y'-16)+104595(V'-128)}{65536}$$

$$G=\frac{76281(Y'-16)-25624(U'-128)-53281(V'-128)}{65536}$$

$$B= \frac{76281(Y'-16)+132251(U'-128)}{65536}$$

Coding Approach Taken and Hardware Relation:

We followed a very strict coding style. In order to keep the amount of coding low, we focused on reusing states to perform all functions necessary in our milestone. The most optimized design given our knowledge was implemented. We had a small lead in that computed two sets of interpolations and one set of colour-space conversion. Thus, the colour-space conversion was done for the first pixel in the row and this value was written when we get into our common case. We tried to match the end of our lead out to transition easily into our common case. This was, our common case could handle the write from our lead in and also handle the writes necessary for the RGB values produced by the last iteration of the common case. We produce 4 sets of RGB pixel data for every iteration of common case. Thus, every common case writes 4 sets of RGB values following the arrangement outlined on the left, to the SRAM. In the Lead in, we read an appropriate set of U and V values, as well as Y values to feed data into the common case. Then in the common case we perform 2 Y reads and 1 U and V read each, to shift in new data into our interpolation (U's and V's) and color space conversion (Y's) calculations. Since we perform two sets of interpolations in our common case, we shift in U and V values into the U and V value shift registers twice

R_XG_X
B_XR_{X+1}
$G_{X+1}B_{X+1}$

$R_{X+2}G_{X+2}$
$B_{X+2}R_{X+3}$
$G_{X+3}B_{X+3}$

per common case iteration. This ensures that the correct U and V values are used as operands for calculation. We used a flag-based approach to implement our lead out states. Flags based off of our pixel counter was used to stop reading as well as to stop shifting in new U and V values. Instead of shifting in new U and V values, the last U and V (159) was shifted in repeatedly. The implementation of this is seen in Figure 4 below.

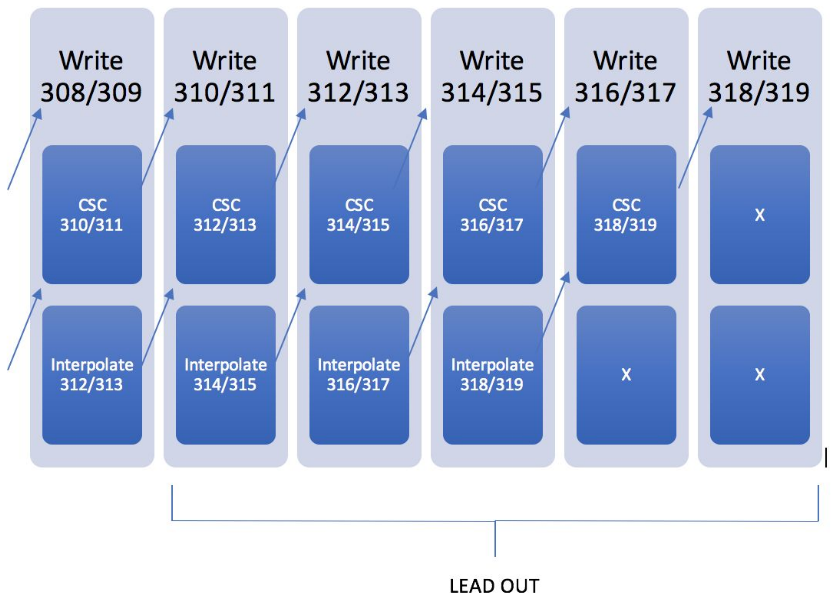


Figure 4: Milestone 1 Logic and Lead Out Implementation

$$240*(\mathit{Num\ of\ CC\ Lead\_In} + \mathit{Common\ Case\ Iteration} + \mathit{Lead\_Out})*\text{ClockCycle\_Duration}$$

Equation 3: Formulas for Calculating Milestone 1 Runtime

Using equation 3, we obtain: 240*(16+12*80+0)*20ns = 4.685 ms , where 20 ns is the clock cycle duration. The 80 is because of the 80 iteration required for processing row of 320 pixel (one iteration of common case processes 4 pixels). This value does indeed match with our milestone 1 testbench_v2 simulation result of 4,685,190 ns. In our implementation, we have a Lead_In = 16 cc, Common Case Iteration = 12 cc, and Lead Out = 0 cc. Lead out is coded to be absorbed into our common case naturally (With flags to handle the non-shifting of the U159 and V159 values). To verify whether we have met our multiplier utilization constraint of 85% minimum usage for the 4 multipliers, we do a sanity check. In Lead_In states, we use have 4*16 multiplier usage, and Common Case we have 4*12 multiplier usage. Therefore based on our code our multiplier usage is(32+44*80)) /(4*16 + 4*12*80) = 90.98% multiplier utilization.

Alternative Approaches, Verification Strategy, and Bug Experiences, Root Cause Analysis:

We have gained very valuable knowledge and skills by completing this milestone. Through the implementation roadblocks, bugs, and troubleshooting it took to fix issues, we feel that we are much better digital circuit designers now, compared to when we started this project.

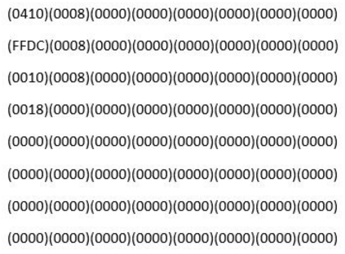
We hit a major bug that delayed our milestone two start time by 2 days. We decided to do a lead out with flags, which would use our common case to shift in U and V values following a different scheme. We opted to do this because it seemed like a cleaner implementation and would not require much coding. The alternate approach was to hard code the lead out states. This requires hard coding approximately forty different states.This seemed inefficient to us given that we had the necessary logic available in our common case. After implementing the flags, we noticed that the U's and V's were shifting incorrectly. Because of the circular nature of our lead out, it was a little difficult to track down the root cause of this error. Eventually, we realized that we were losing U/V 158 and it was not getting put into the buffer. This was caused by the wrong counter conditions. We had incorrect logic to assert the flag than what we initially thought. Although both members attempted to fix this subtle, but major bug, majority of the root cause analysis was done by Ryan. Due to time constraint, Moshiur was coding the hard lead out, in the possible event that we were unable to find the bug in our original code.

The other bugs we experienced were not terribly complicated in nature. We had addressing problems at first which we fixed by moving around code amongst our states to allow clock cycle delays and buffer register reads to line up correctly. We had several typos as well which would cause severe, but easily fixable errors. All our bugs were fixed by creating signals in modelsim that were relevant to the error we were experiencing. We organized the signals in an intuitive way and would then sweep through the code and compare results with hand calculations. Once we noticed a deviation, we looked at the nth state at which the incorrect signal is written and then we go into our verilog code to look at the n-1th state (non-blocking).

Milestone 2:

Intro

We correctly implemented the first 8-by-8 block processing for Fetch Prime S, and compute T. We were able fetch the first pre-IDCT block from the SRAM:



```
(0410)(0008)(0000)(0000)(0000)(0000)(0000)(0000)
(FFDC)(0008)(0000)(0000)(0000)(0000)(0000)(0000)
(0010)(0008)(0000)(0000)(0000)(0000)(0000)(0000)
(0018)(0000)(0000)(0000)(0000)(0000)(0000)(0000)
(0000)(0000)(0000)(0000)(0000)(0000)(0000)(0000)
(0000)(0000)(0000)(0000)(0000)(0000)(0000)(0000)
(0000)(0000)(0000)(0000)(0000)(0000)(0000)(0000)
(0000)(0000)(0000)(0000)(0000)(0000)(0000)(0000)
```

We were able to compute a set of T values for one 8x8 block: Which, after sign clipping was: 5945, 5935, 5918, 5894, -141, -151, -169, -192, 153, 143, 126, 102, 135, 135, 135, and 135. This can be found in the waveform.

After doing the matrix multiplication for (1/256)*(S’xC), we have verified that these 16 T values were correct. These T values corresponds to the top left corner values of the T matrix for the 8x8 block of index (0,0).

Coding Approach Taken and Hardware Relation:

Due to time constraints, we have opted to try to have an iteration of S_Fetch_S_Prime and Compute_T (called S_ MS_B in our code). Similar approach to coding style was taken from M1, where we figured out what the minimum repeatable common case worth of cycles were needed to do fetching and computation of S’ (16 bit signed) and T (32 bit signed) respectively. Given the amount of work done in M2, we currently have used two DP-RAM, one for storing S’ and the other for T. We have implemented our cosine coefficient multiplier operands with muxes. And the multiplication pattern used to compute T matrix was by accumulating through the S’ rows, and across the columns of the Cosine matrix. Since we had 4 multipliers, we chose to accumulate left 4 columns of C matrix, accumulating term by term in 8 clock cycles, we collect the 8 partial T terms to accumulate for storing back of the final T values to. The counting circuit based off “SAMPLE_COUNTER” is a direct hardware translation from the circuit created by Dr. Nicolici in lecture. A similar circuit would have been used to write back the sampled to the SRAM, had we progressed to implementing the “Write S” stage.

Alternative Approaches, Verification Strategy, and Bug Experiences, Root Cause Analysis:

The process we have used to debug some of the issues with this preliminary designs were checking the address manually for where the S’ values should be stored in the hex editor to figure out whether the S’ fetching was done correctly for the first pre-IDCT Y 8x8 block. After checking SRAM locations 76800 to 76807, and every offset to that by 320*n, we verified that our S’ fetching was correct. One significant issue we had that we fixed was the sign extension of T values. We noticed that for most positive values, our final T value was correct. For negative numbers, the numbers were completely different. We realized that something with sign extension was wrong, so we sign extended by extending the msb of the number to fill the rest of the leading msb bits.

Another interesting issue was the incorrect writing pattern of the T values. We noticed that at the beginning of when T values were finally computed, the T values were not correct. We tracked this issue going through our S_MS_B states to check whether our conditions to loop back were correct, and noticed that at the beginning of the S_MS_B iteration, there were no values to write. Therefore, we decided to add this flag: T_Accum_Write_Flag to correctly perform DPRAM writing without adding extra delay states.

Schedule	**Description**	**Contributions:**
Week 1 (Oct. 23 to 27)	Reading through Project Outline	**Ryan** and **Moshiur:** Gain general knowledge, set up project files
Week 2 (Oct. 30 to Nov. 3)	Milestone 1 Design Milestone 1 State Table	**Ryan:** Design, Revise state table **Moshiur:** State table, design
Week 3 (Nov. 6 to 10)	Milestone 1 State Table Milestone 1 Coding	**Ryan:** Common state, Coding and debugging **Moshiur:** Common state, Coding and debugging
Week 4 (Nov. 13 to 17)	Milestone 1 Coding Milestone 1 Debugging	**Ryan:** Common state/ Lead out coding/ debugging **Moshiur:** Common state/ Hard-Lead out coding/ debugging
Week 5 (Nov. 20 to 27, where Nov. 23 to 26 was allocated for Milestone 2)	1) Milestone 1 Debugging 2)Milestone 1 Hardware Implementation 3)Milestone 2 State Table 4)Milestone 2 Design 5)Milestone 2 Coding 6)Milestone 2 Debugging 7)Project Report	**Ryan:** Milestone 2 Coding, debugging, and milestone 2 design. **Moshiur:** focused on milestone 1 hardware testing onto the board, and milestone 2 coding, Milestone 1 and 2 top level FSM. Otherwise roughly equal contribution.

Conclusion:

This project gave us firsthand experience of what it feels like to be an engineer. Problem solving, debugging unpredictable/persistent issues, brainstorming with colleagues to understand tough concepts together, designing unique solutions, iterating and failing until the design finally works. Spending hours to debug one subtle issue and figuring out the issue is what makes an engineer an engineer. This skill that we have refined from this project will be invaluable in our future careers, and we hope to improve this skill as we grow as an engineer.