TOTAL 23.5/36

M1 14.5
M2 9

GOOD PROGRESS BUT STILL LOTS OF DEBUGGING FOR COMPUTER SYSTEMS

COE3DQ5 – Project Report

James Cameron & Matthew Rakic
Group 81 Thursday
November 24, 2025

THX TO YOU AFTER THE GAME ON JAN8

SEE YOU BOTH IN 3DY4

Introduction

The overall goal of the COMPENG 3DQ5 project was to design, integrate, and verify the code required to decompress images encoded in the custom McMaster Image Compression (.mic19) format. Using SystemVerilog on the DE2-115 FPGA board, the project aimed to build the decompression pipeline that converts a .mic19 bitstream back into a viewable PPM image. This required implementing the five major decoding stages, lossless decoding, requantization, inverse DCT, chroma upsampling, and colour-space conversion, organized across three milestones. The objective was to deepen the understanding of digital systems while learning new methods through the design process.

Implementation Details

Upsampling and Colour Space Conversion (Milestone 1)

STATE	C1	C2	C3	C4	C5	C6	C7	C8
Multiplier 0	V'8	E2 (Ubuff x -12845)	O1 (Vbuff x 52298)	U'0	U'4	U'8	V'0	V'4
Multiplier 1	V'9 (Vodd is accumulated	E3 (Vbuff x -26640)	O2 (Ubuff x -12835)	U'1	U'5	U'9 (Uodd is accumulated)	V'1	V'5
Multiplier 2	E0 (Yeven x 38142)	E4 (Ubuff x 66093)	O3 (Vbuff x -26640)	U'2	U'6	unused	V'2	V'6
Multiplier 3	E1 (Vbuff x 52298)	O0 (Yodd x 38142)	O4 (Ubuff x 66093)	U'3	U'7	unused	V'3	V'7

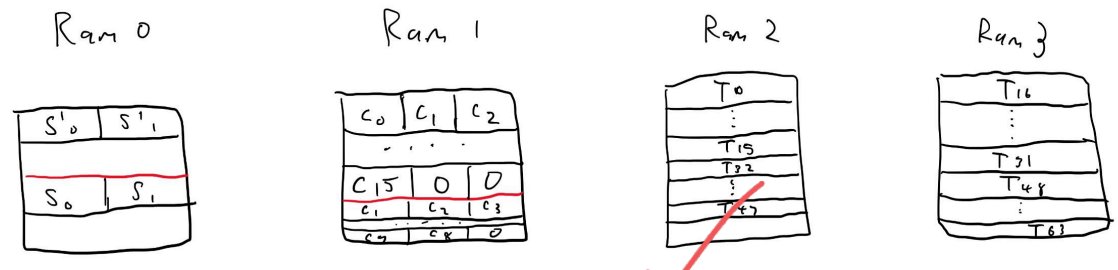
The interpolation calculations follow the sequence shown above. In the common case, the partial products for U′odd are generated in states C4–C6, while the partial products for V′odd occur in C7–C8 and finish at the start of C1. This ensures the required U′ and V′ values are available on time for colour-space conversion. The RGB conversion is structured similarly: even RGB values are computed in C1–C2, and odd RGB values in C2–C3, allowing the design to write one RGB pixel pair to SRAM every cycle. By organizing the interpolation and colour-space conversion this way, the shifted U′ and V′ registers are always ready at the start of each cycle. The design can compute RGB values for the previous pixel while beginning the next U′odd and V′odd accumulations, improving multiplier utilization during both the lead-in and lead-out stages.

Register Name	Bits	Description
counter	14 bits	Tracks the current pixel index as the module processes Y, U, and V sample locations across the frame.
U_regs[0..9] V_regs[0..9]	10 bits x 16 registers	Ten-element sliding window buffer storing consecutive U, V samples. Used to provide the 10-tap filter input required for both even and odd U, V interpolation calculations.
Y_even, Y_odd	8 bits x 2 registers	Stores the Y (luma) value corresponding to the even/odd pixel of each pixel pair being processed.
U_buff, V_buff	8 bits x 2 registers	Buffers the U4 & V4 used during the even-pixel chroma interpolation and subsequent colour computation.
U_acc, V_acc	32 bits x 2 registers	Accumulates the weighted sum of the 10-tap U filter outputs used to generate the interpolated odd-pixel U′ & V’ value.
R,G,B_even R,G,B_odd	32 bits x 6 registers	Holds the computed R, G, B components for the even & odd pixels before clipping and packing into SRAM output.
line_threshold	14 bits	Stores the final counter value for each processed line, used to detect when the module should transition to the lead-out sequence at the end of a row.
lof	1 bit	“Lead-Out Flag” indicates that the module has reached the final values of a line and should enter the lead-out handling states.
locounter	3 bits	Counts the number of lead-out cycles executed to ensure proper 3-pixel finalization at the end of each line.

TOTAL	480	Optimized to 471 during synthesis, verified in the Fitter file.

The latency analysis shows that Milestone One requires a total of 111,603 clock cycles, consisting of 1 cycle in IDLE, 10 Lead-in cycles, 8 cycles for each of the 95 Common cases, 5 cycles for each Lead-out, and 2 DONE cycles. This results in 775 cycles per line, and with 144 lines, the total is 111,600 cycles, plus 3 overhead cycles from the IDLE and DONE states. To determine multiplier utilization, each row contains 95 Common cases where 30 of 32 multipliers are used, one Lead-out consisting of 5 cycles, with full usage in 3 out of 5 cycles (9 of 15 multipliers), and one Lead-in of 10 cycles with 18 of 40 multipliers used. Combining these contributions gives a per-row efficiency of (30×95+9+18)/(32×95+15+40)=92.9%. Applying this percentage to the total runtime yields 103,739 effective multiplier-active cycles, demonstrating that the design achieves 92.9% overall multiplier utilization during Milestone One.

Inverse Discrete Cosine Transform (Milestone 2)



CT1	CT2	CT1	CT2	CT1	CT2	CS1	CS2	CS1	CS2	CS1	CS2
ram0a[31:16]	ram0a[15:0]	ram0a[31:16]	ram0a[15:0]	ram0a[31:16]	ram0a[15:0]	ram1a[26:18]	ram1a[26:18]	ram1a[26:18]	ram1a[26:18]	ram1a[26:18]	ram1a[26:18]
ram1a[26:18]	ram1a[26:18]	ram1a[26:18]	ram1a[26:18]	ram1a[26:18]	ram1a[26:18]	ram2a	ram3a	ram2a	ram3a	ram2a	ram3a
ram0a[15:0]	ram0b[31:16]	ram0a[15:0]	ram0b[31:16]	ram0a[15:0]	0	ram1a[17:9]	ram1a[17:9]	ram1a[17:9]	ram1a[17:9]	ram1a[17:9]	0
ram1a[17:9]	ram1a[17:9]	ram1a[17:9]	ram1a[17:9]	ram1a[17:9]	0	ram3a	ram2a	ram3a	ram2a	ram3a	0
ram0b[31:16]	ram0b[15:0]	ram0b[31:16]	ram0b[15:0]	ram0b[31:16]	0	ram1a[8:0]	ram1a[8:0]	ram1a[8:0]	ram1a[8:0]	ram1a[8:0]	0
ram1a[8:0]	ram1a[8:0]	ram1a[8:0]	ram1a[8:0]	ram1a[8:0]	0	ram2b	ram3b	ram2b	ram3b	ram2b	0

Register Name	Bits	Description
address_a[3:0] address_b[3:0]	8 bits x 8 registers	Used to address a location in each of the 4 embedded RAMs
read_data_a32[3:0] read_data_b32[3:0]	32 bits x 8 registers	Used to store the value read from the x embedded RAM from ports a and b of each of the 4 embedded RAMs
write_data_a[3:0] write_data_b	32 bits x 5 registers	Used to store a value to be written into the embedded RAM memory, data_b only has 1, as we only write to port b for 1 RAM
write_enable_a[3:0] write_enable_b	1 bit x 5 registers	Used to enable the write of each of the embedded RAMs, enable_b only has 1, as we only write to port b on 1 RAM
Ri, Ci, Rb, Cb	4 bits x 4 registers	Used for indexing through S' values
ct_acc, cs_acc	32 bits x 2 registers	Used for accumulating the multiplier results for the compute T and compute S matrix multiplications
ct_acc_buf, cs_acc_buf	32 bits x 2 registers	Used for storing the accumulated multiplier results for computing T and S for writing, while the accumulators are reset
prev_value	32 bits	Used to store the previous accumulator value so it can be appended to the current one to push 2 S values to 1 memory slot
sprime_even_buf	16 bits	Used as a buffer for storing the even and odd S' values together
ct_i, ct_j, cs_i, cs_j	4 bits x 4 registers	Used for indexing through the matrices of compute T and S
s_address	8 bits	Used for tracking the address to write the S values into RAM0

Resource Usage and Critical Path

In Lab 5 Experiment 4, our design used 616 logic elements, including 369 top-level registers. After Milestone 1, this increased to 2693 logic cells and 836 dedicated logic registers. The increase is primarily due to the four multipliers, which require many 32-bit operands, results, and 64-bit intermediate values. Since the top level still contains the same 369 registers as in Lab 5, all additional registers come from Milestone 1 and reflect the added arithmetic, accumulation, and data-processing hardware.

After completing Milestone one, one memory waste became apparent. Because even and odd RGB values are written in different stages, we used separate registers for each. With a different write schedule, a single register per colour could have been reused.

The timing analysis reports an Fmax of 54.19 MHz, confirming safe operation at our 50 MHz clock. The critical path runs from the M1 state register through a large combinational multiplexer, into the 32×32 multipliers, and finally into the U and V accumulators. This is expected, as the multipliers and wide arithmetic form the longest delay in the design.

Weekly Activity and Progress

Week	Project Progress
Week 1 Oct 20 - Oct 26	- Spent the majority of time fixing our lab 5 code as it did not work properly, and we wanted a good baseline to begin the project on. Finished lab 5 code on Oct. 23. - Once finished, we began reading the project document to understand what would be required throughout the project.
Week 2 Oct 27 - Nov 2	- Spent a week developing our state table and mapping out how each of our states would work - Most of our time was spent planning and reconfiguring our states, as they did not work as we had planned for them to work originally
Week 3 Nov 3 - Nov 9	- Began development of milestone 1 - Focused on implementing the lead-in of the code first
Week 4 Nov 10 - Nov 16	- Completed initial full code integration of milestone 1 - Had many timing and addressing issues that needed to be fixed, as the always ff addressing and timing did not work as we had planned originally
Week 5 Nov 17- Nov 24	- Discovered an issue and fixed the implementation of Milestone 1 on Nov. 18. - Immediately began planning and development of milestone 2 - Implemented and verified FS and CS states, and implemented CT and WS states, but did not have time to verify.

Contributions

During week 1, we both spent time in the lab fixing the Lab 5 code to have a functioning baseline for the project. We then spent time individually reading the project document to understand the project requirements.

During week 2, we spent time equally collaborating on the state table development done on Google Sheets, and troubleshooting was done by both members as required.

During week 3, we began development of Milestone 1, which was mostly done from home. We finished the lead-in and most of the common case together. James debugged the common case while Matthew designed the logic for the lead-out case.

During week 4, Milestone 1 was finalized and tested in the lab together. We then collaborated on planning for Milestone 2, creating a general state table and compiling ideas and notes from class.

During week 5, for milestone 2, James set up the M2 module files and implemented FS and WS states, while Matthew implemented CS and CT states. Together, we worked on implementing them into the mega states.

Conclusion

Throughout this project, we learned to approach hardware design in a more systematic and structured way. Debugging off-by-one errors, timing issues, and state-machine behaviour showed us the importance of understanding what happens on every clock cycle. We also gained experience managing registers, organizing multi-step calculations, and writing memory access patterns. Working through these challenges improved our ability to read waveforms, trace errors, and make changes to our design when needed.

Most recent completed milestone: Nov 24th “M1” d705347