COE3DQ5 – Project Report

Hemant Arora, Warun Gumarawell

Group 32

November 25, 2024

1 Introduction

The objective of this project is to design and implement a hardware-based image decompressor compliant with the McMaster Image Compression revision 18 (.mic18) specification. The decompressor processes a compresses bitstream representing a 320x240 resolution image, delivered via UARTs interface to an Altera DE2-115 FPGA board. The design includes decoding the compresses data, performing dequantization, applying the inverse discrete cosine transform (IDCT), and executing colour space conversion (CSC), to reconstruct the image. The processed image is stored in SRAM and displayed via a VGA controller. This project focuses on optimizing real-time processing and addressing resource constraints through efficient hardware design and the use of fixed-point arithmetic.

2 Implementation Details

2.1 Upsampling and Colour Space Conversion (Milestone 1)

	CC_1	CC_2	CC_3	CC_4	CC_5	CC_6
multiplier 1	C odd	C even	C even	C even	C odd	C even (n + 2)
multiplier 2	U	U	U	C even	C odd	C odd
multiplier 3	V	V	V		C odd	

Figure 1: Common case for Upsampling and colour space conversion

Register name	Bits	Description
even_pixel_counter	18	Tracks the current even pixel index to determine the appropriate SRAM locations for fetching Y, U, and V values.
col_counter	8	Tracks the number of even pixels in the current row. Incrementing only for even-numbered pixels, then resets at the end of the row.
write_address_counter	18	Tracks the current address in SRAM where the RGB data will be written.
SRAM_Y SRAM_U [2] SRAM_V [2]	16	These registers store the values read from SRAM, with each holding two 8-bit values. The dual U and V registers are used to support both color space conversion for even pixels and upsampling to generate values for odd pixels.
SRAM_address SRAM_write_data SRAM_we_n	18 16 1	These are used for interfacing with SRAM, handling address selection, data writing, and write enable control.
FIR_U [6] FIR_V [6]	8	These registers store six consecutive values needed for upsampling for both U and V.
U_prime V_prime	32	These registers hold the intermediate and final upscaled values for the odd pixel during the MAC operation for both U and V.
Y_CSC[2]	32	This register stores the constant value calculated during the YUV to RGB color space conversion
Red [2] Green [2] Blue [2]	32	These registers hold the partial and final RGB values, with two registers for each colour to store the values for both even and odd pixels.
op1 [2] op2 [2] op3 [2]	32	These registers hold the two operands for each of three the multipliers, with each register
toggle	1	This register is used to switch between reading the UV values for the even pixel color space conversion or the odd pixel upsampling.
done	1	Indicates the completion of processing all pixels in the image.
state	5	Tracks the current state of the finite state machine controlling interpolation, colour space conversion, and SRAM operations.

Figure 2: Register Table summarizing the purpose and usage of all registers in Milestone 1

In the common case, we perform 16 multiplications in 6 clock cycles, with a maximum of 3 multiplications per clock cycle, giving a utilization of 16/18 = 88.89%. With 76,800 pixels, we process one odd and one even pixel, entering the common case 38,400 times. Each row has 9 lead-in and 3 lead-out states, requiring 233,280 clocks plus 1 for the done state. The total number of multiplications needed is 38,400 * 16, and the potential multiplications in 233,280 clocks is 233,280 * 3. Performing (38,400 * 16) / (233,280 * 3) yields a utilization of 87.79%

2.2 Inverse Discrete Cosine Transform (Milestone 2)

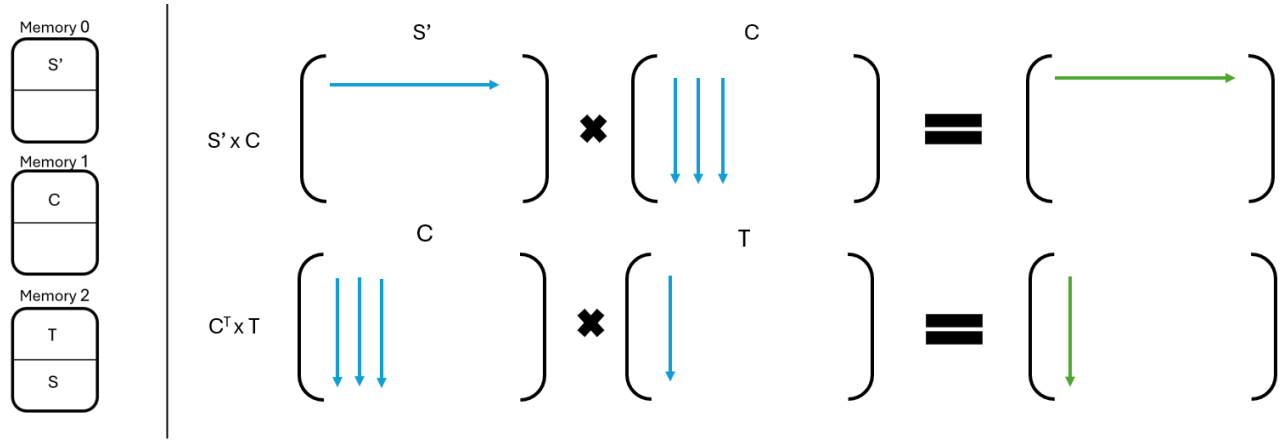

Figure 3: Design Approach for Milestone 2

Register name	Bits	Description
state	3	Tracks the current state of the finite state machine responsible for managing accesses to the SRAM and embedded RAM ports.
index_count index_count_delay1	7	This register tracks the current address read from the embedded RAM during data transfers to SRAM. The delayed version accounts for the 1-clock cycle latency of the embedded RAM, ensuring SRAM writes to the correct address, offset by one cycle.
col_block_write row_block_write	6 5	Track the row and column blocks that determine the specific locations in SRAM being written to during memory operations.
block_state_write	2	Tacks whether the current write operation in SRAM is in the Y, U, or V memory locations.
SRAM_data_write_buffer_reg	8	This register temporarily stores one value read from the embedded RAM, as each SRAM location requires two values.
column_count row_count	8	These registers hold the current row and column indices of the two matrices being multiplied.
state_count state_count_dealy	2	These registers track row/column progression in matrix multiplication. With an 8x8 matrix and three multipliers, 3 cycles are needed per row to accumulate all 8 values. The delay register accounts for the 1-cycle read latency of the embedded RAM, ensuring correct data processing.
lead_out_ena lead_out_ena_delay lead_out_ena_delay2	1	These signals indicate when matrix multiplication is complete. The original signal tracks when the row and column indices reach their maximum, while the delays account for the 1-cycle latency of the embedded RAM and ensure enough time to write the final two values.
mac_reg[3]	32	Stores the intermediate values computed during the matrix multiplication process
result_reg[2]	32	Buffers two MAC values, providing the necessary time to store them properly
write_address_count_reg	7	Tracks the location in memory where the results of the matrix multiplication should be written.
done	1	Indicates the completion of IDCT.
m3_ena	1	Enables or disables the milestone 3, which is responsible for decoding values from SRAM.

Figure 4: Register Table summarizing the purpose and usage of all registers in Milestone 2

To compute 8 values of the resultant matrix, it takes 24 clock cycles, broken down into 16 cycles with 3 multiplications and 8 cycles with 2 multiplications. At the start of the entire matrix calculation, there is an additional 1 clock cycle where no multiplication occurs while waiting for data from the embedded RAM. At the end, 2 clock cycles are required to write the final data back to the embedded RAM. Since you need to calculate 64 values in total, total clock cycles for one matrix multiplication is 8 * 24 + 3 = 195 clocks For 2,400 8x8 blocks, each requiring 2 matrix multiplications, you have 488 lead-in clocks (explained in M3) and 64 lead-out clocks to write the last matrix to SRAM. Thus the total number of clocks is 2400 * (195 + 195) + 552 = 936552. The overall utilization is (2,400 * (171 + 171)) /(2,400 * (195 + 195) 552) = 87.64%.

2.3 Lossless Decoding and Dequantization (Milestone 3)

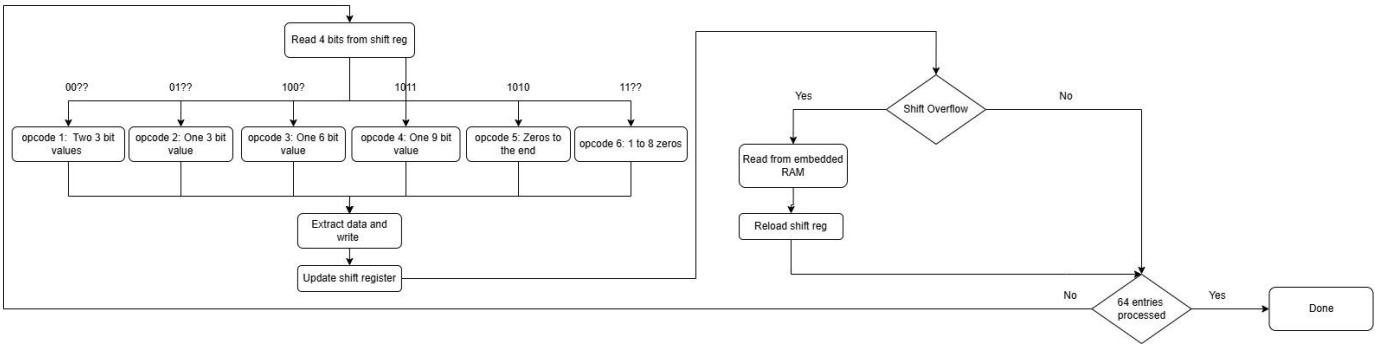

Figure 5: Flowchart describing the approach for Milestone 3

Register name	Bits	Description
SRAM_address	18	Tracks the SRAM address for filling the embedded RAM buffer.
write_en_decode	1	Enables writing to the embedded memory that stores the decoded matrix, i.e. input for M2.
state	3	Tracks the current state of the finite state machine responsible for managing the decoding.
index_count index_count_delay	6	Tracks the current pixel being decoded. A delayed version offsets the count by one cycle to allow for applying the quantization and determining the diagonal memory address
count	6	Used for codes with multiple writes.
data	16	Holds the decoded data after sign extension, but before quantization.
write_en write_en_dealy1 write_en_dealy2	1	Enables writing to the embedded RAM buffer from SRAM, delays account for the 2-clock delay in SRAM, ensuring proper timing for writing the data to the buffer.
write_prt write_prt_delay1 write_prt_delay2	8	Holds the address for writing to the embedded RAM buffer from SRAM, delays introduced to account for the 2-clock delay in SRAM, ensuring proper timing for writing the address to the buffer.
write_data	16	Holds the data to be written to the buffer form SRAM
shift_reg memory_data	32 16	Holds the string of bits being decoded, with one register storing the current string and another holding the next 16 bits to be shifted in as the current string is read.
shift_count	5	Counts how many bits have been shifted out of the register.
op_code	3	Stores the current operation being performed.

Figure 6: Register Table summarizing the purpose and usage of all registers in Milestone 3

The decoding process takes up to 3 clock cycles per value, with one state dedicated to accessing SRAM, resulting in a maximum of 192 clock cycles to decode 64 values with 9-bit encoded data. However, the shift register accumulates data over several cycles, reducing the frequency of SRAM access. This results in a maximum observed clock cycle count of 180 with 9-bit encoded data. The lead-in process involves filling the buffer with 255 words from SRAM and writing to embedded memory, taking 256 clock cycles. Decoding the first matrix requires up to 192 clock cycles. Since M2 and M3 are highly dependent on each other, both start and finish at the same time. M2 begins after the first matrix is decoded.

2.4 Resource Usage and Critical Path

The resource usage of our project, as reported by Quartus, shows a total of 3,840 LUTs used out of 114,480 available, which is about 3% of the total available logic elements. In comparison, Lab 5 Experiment 4 had a LUT usage of only 616 (<1%). The significant difference in LUT usage makes sense as our project involves more calculations and logic complexity compared to the simpler Lab 5 experiment which only included UART, SRAM, and VGA interfaces. When accounting for the resources used by the interfaces in both projects, the code itself (excluding the interfaces) uses 3,840 - 616 = 3,224 LUTs, reflecting the added complexity and functionality of our current project.

In Milestone 1, the use of three separate counters — even_pixel_counter, write_address_counter, and col_counter — was unnecessary. By storing only the row and column counts, we can dynamically calculate the pixel count, as done in Milestone 2, removing the need for additional registers. Additionally, the write address calculation can be optimized. Instead of using a separate counter, we can directly compute the write address. Since each even pixel occupies 3 memory locations (including its odd counterpart), the address can be derived by left-shifting the even pixel count by 1 (equivalent to multiplying by 2) and then adding the original value. This operation effectively calculates the result of multiplying by 3, simplifying the logic and reducing resource usage. Furthermore, one of the calculations for the even

color space conversion is done in the common cases of the pixel before it, which seems a bit unorganized. To improve clarity and efficiency, we would rearrange the multiplications so that everything is done in one common case

In Milestone 2, the third dual-port memory is used with one port for writing temporary matrix values and the other for reading the final matrix values (S) to write to SRAM. Since every 8 clock cycles we need to write up to 3 temporary values, we currently write the first one directly from the MAC register and buffer the other two. Similarly, each SRAM location requires two values, so we buffer one. Instead, we can manage memory access more efficiently at a lower level. In the first clock cycle, both ports will write two temporary values directly from the MAC registers, while the third value is buffered and written in the second clock cycle. In the third, fourth, fifth, and sixth cycles, we can read from both ports and write the values directly to SRAM, eliminating the need for additional buffering, saving registers.

The critical path in the Timing Analyzer flows from the embedded RAM storing the matrices to the MAC register holding the partial products in Milestone 2. This path has a setup slack of 4.104 ns. The next slowest paths are similar. These paths pass through at least one multiplier and adder which are typically the slowest components therefore they dominate the critical path. Additionally, they may pass through multiplexers for operand selection and memory access control. This is expected, as in this design, unlike Milestone 1 where operands were registered before being passed to the multipliers, operands are used unregistered in each cycle. This design choice introduces more delay, which increases the timing path and contributes to the critical path.

3 Weekly Activity and Progress

Week	Project Progress	Contribution
1	Reviewed project specifications and analyzed the .mic18 compression flow, revising lab materials accordingly.	**Hemant:** Focused on understanding the project concept and overall flow. **Warun:** Focused on understanding the project concept and overall flow.
2	Designed the state table for M1 and began implementing Verilog for CSC and upsampling.	**Hemant:** Drafted the preliminary state table and contributed to selecting the final version. Began working on common-case Verilog code for M1 **Warun:** Collaborated on state table finalization and began working on Verilog code for lead-in and lead-out cases.
3	Completed the Verilog implementation for M1, validated the results, and then designed the state table for M2.	**Hemant:** Debugged M1 module and performed simulations. Worked on state table for M2 and discussed finalizing on design. **Warun:** Debugged M1 module and verified implementation on board. Worked on state table for M2 and discussed finalizing on design.
4	Finished the Verilog implementation for M2, tested and validated the outcomes.	**Hemant:** Implemented the core logic for M2, thoroughly debugging and verifying its functionality through simulations and hardware testing. **Warun:** Contributed to the implementation of M2 logic, collaborated on debugging efforts, and ensured successful verification through simulations and hardware testing.
5	Implemented and tested M3, then finalized the project documentation.	**Hemant**: Implemented the decoding logic and FSM for M3, conducting thorough testing and verification through simulations. Contributed to the project report by completing calculations and drafting key sections. **Warun**: Implemented the decoding logic and FSM for M3, conducting thorough testing and verification on board. Contributed to the project report by creating diagrams and drafting key sections. **Bradley:** Received guidance from Bradley on modifying the testbench to ensure it wrote to the correct memory address for M3.

Figure 7: Table describing weekly project progress and contributions

4 Conclusion

In conclusion, this project served as a comprehensive application of concepts learned in the COE3DQ5 course, including finite state machines, memory interfacing, and signal processing. By completing the milestones, we gained hands-on experience in designing and implementing components like color space conversion, IDCT, and lossless decoding, reinforcing our understanding of hardware design. This project also highlighted the importance of debugging, iterative development, and resource management in creating efficient systems. Overall, it was a challenging yet rewarding experience that enhanced our technical skills and problem-solving abilities, preparing us for future real-world hardware design projects.