COE3DQ5 - Hardware Implementation of an Image Decompressor

Amr Elhossan, Group 31, 400394120

November 27, 2023

1 Introduction

This project involves the practical application of digital system design, focusing on implementing the McMaster Image Compression revision 17 (.mic17) specification in hardware.

The objective is to decompress a 320 x 240 pixel image compressed in the mic17 standard. The custom-designed hardware circuitry will decode the compressed image data, and reconstruct it into a 3-byte RGB format.

2 Implementation Details

2.1 Upsampling and Colourspace Conversion

•

State	0	1	2	3	4	5	6	7	8	9	10	11
M1		E	E	E	E	E		E	E	E	E	E
M2		O	O	O	O	O		O	O	O	O	O
M3	V	V	V	V	V	V	V	V	V	V	V	V
M4	U	U	U	U	U	U	U	U	U	U	U	U

Table 1: M1 multiplier division of labour

•

Register Name	**Bits**	**Description**
2 x FIR_buf	6x8	Shift registers used to shift U and V values for interpolation
6 x CSC_accumulator	32	Accumulator registers used to accumulate partial products of CSC
2 x interp_accumulator	32	Accumulator registers used to accumulate partial products of upsampling
i	9	Counter register used to keep track of current row
j	9	Counter register used to keep track of current column
UCSC_sram_data	2x16	Buffer register U and V data read from the SRAM
UCSC_sram_data_buf	2x8	Buffer register for U and V data from the UCSC_sram_data reg
2 x upsampled_U/V	8	Buffer registers stores the interpolated result from the interp_accumulator
2 x cache	32	Buffer registers holds first common partial product from CSC
2 x coefficient_select_U/V	2	Control registers select the coefficient operands used for interpolation
coefficient_select_RGB	3	Control register selects the coefficient operands used in CSC
op_select_RGB	2	Control register selects between YUV used in CSC
2 x U/V_buf	8	Buffer register holds UV data for use in CSC
4 x SRAM_address_cont	18	Counter registers hold SRAM address for YUV and RGB address offsets

Table 2: M1 register characterization table

- In M1, the common case produces 4 pixels per 12 cycles which corresponds to 3 cycles/pixel. The common case produces a total of 308 pixels which is 924 cycles. The common case repeats 240 times meaning it contributes 221760 cycles.

  The lead-out is 33 cycles and the lead-in is 23 cycles for the first row and 12 cycles for the proceeding rows meaning the lead-out contributes 7920 cycles and the lead-in contributes 2891 cycles. The total number of clock cycles M1 takes to finish is 232571.

2.2 Inverse Discrete Cosine Transform

•

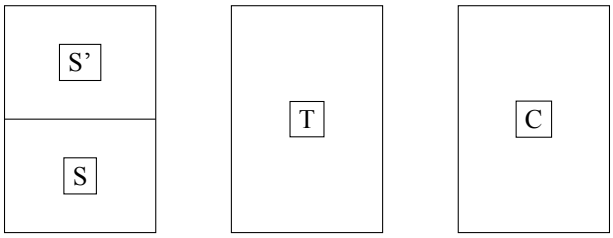

Figure 1: Signal transform memory layout

$$T_{00} = \boxed{S'_{00}C_{00} + S'_{01}C_{10} + S'_{02}C_{20} + S'_{03}C_{30}} + \boxed{S'_{04}C_{40} + S'_{05}C_{50} + S'_{06}C_{60} + S'_{07}C_{70}}$$

$$T_{01} = \boxed{S'_{00}C_{01} + S'_{01}C_{11} + S'_{02}C_{21} + S'_{03}C_{31}} + \boxed{S'_{04}C_{41} + S'_{05}C_{51} + S'_{06}C_{61} + S'_{07}C_{71}}$$

$$S_{00} = \boxed{C_{00}T_{00} + C_{10}T_{10} + C_{20}T_{20} + C_{30}T_{30}} + \boxed{C_{40}T_{40} + C_{50}T_{50} + C_{60}T_{60} + C_{70}T_{70}}$$

$$S_{10} = \boxed{C_{01}T_{00} + C_{11}T_{10} + C_{21}T_{20} + C_{31}T_{30}} + \boxed{C_{41}T_{40} + C_{51}T_{50} + C_{61}T_{60} + C_{71}T_{70}}$$

•

Register Name	**Bits**	**Description**
accumulator	32	Accumulator register used to sum partial products from matrix multiplication
8 x S_matrix	8	Buffer registers used to hold S elements to fill RAM words with 2 S elements
4 x S_prime_data	32	Buffer registers used to hold row of S'
4 x coeff_data	32	Buffer registers used to hold a column of C
8 x M_data	32	Buffer registers used to hold a column of T
S_prime_data_buf	32	Buffer register holds an element of S' to setup next next row
2 x M_data_buf	32	Buffer registers used to hold elements of T to setup next column
2 x M_address_a/b	6	Counter registers used to address the T matrix
2 x S_prime_address_a/b	5	Counter registers used to address the S' matrix
2 x coeff_address_a/b	4/5	Counter registers used to address the C matrix
write_s	1	Control register set to write S to SRAM
s_prime_full	1	Control register set to indicate if S array is full
op_select	1	Control register selects which operands are fed to the multiplier
row_identifier	1	Control register indicates which row is being processed
start_buf	1	Buffer register used to detect the rising edge of the start signal
matrix_select	1	Control register selects which matrix is being processed
element_cont	6	Counter register counts how many elements of S have been written to SRAM
block_cont_i_w	7	Counter register keeps track of the written row blocks
block_cont_j_w	6	Counter register keeps track of the written column blocks
segment_indicator_w	1	Control register indicates which segment is being written Y or UV
fill_s_prime_buf	1	Buffer register used to detect the rising edge of the fill_s_prime signal
write_s_buf	1	Buffer register used to detect the rising edge of the write_s signal

Table 3: M2 register characterization table

- In M2, the common mega-state is 269 cycles long. This common case runs 2398 times which contributes 645062 cycles. The lead in has $F_{S'}$ and C_T routines which are 68 and 134 cycles respectively. The lead out has C_S and W_S routines which are 135 and 36 cycles respectively. This means the total number of cycles M2 requires to finish completely is 645435.

2.3 Lossless Decoding and Dequantization

-

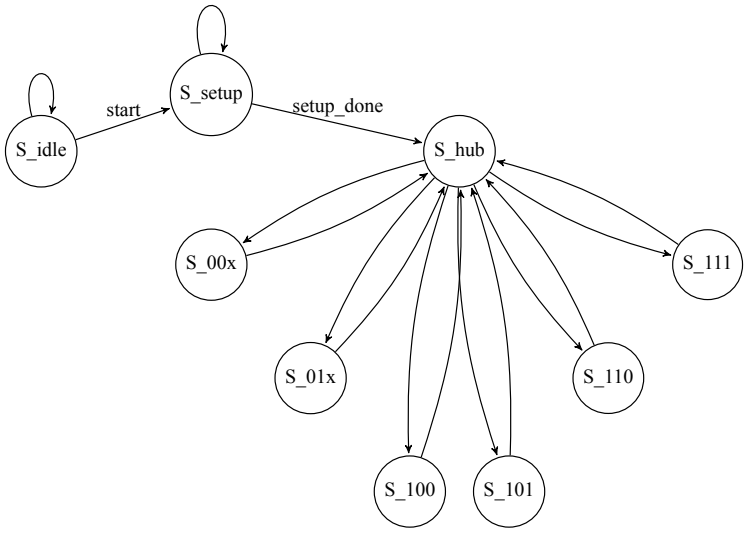

Figure 2: Abstracted state machine design of lossless decoding

-

Register Name	Bits	Description
SR	32	Buffer register used to hold mic17 encoded data
2 x SRAM_data_buf	16	Buffer registers used to hold mic17 encoded data read from the SRAM
element_cont	6	Counter register used to keep track of current element
index_ptr	5	Register used to keep track of current index in SR
data	16	Register holds quantized data from lossless decoding
count	3	Register holds number of times specific data will be written to RAM
k	3	Counter register keeps track of how many times specfic data was written to RAM
load_buf	1	Buffer register used to detect rising edge of load signal

Table 4: M3 register characterization table

- In the worst-case scenario, the decode_controller has to decode an 8x8 block of 9-bit signed values. In this case, the control flow would bounce between the hub and 00x state for each 9-bit signed value. This means that to decode 64 elements, 64 round trips between the hub and 00x have to occur which is 128 cycles. This is possible due to my implementation hiding the SRAM access latency.

2.4 Resource Usage and Critical Path

- The compilation report in Quartus reports a register and logic element count of 2124 and 3964 respectively. The boilerplate design had 369 registers and 616 logic elements which is a delta of 1755 registers and 3348 logic elements.

- The main and, it must be stated, a very large area where the design implementation can be improved is the IDCT_controller. I get a whole element of T in two clock cycles by multiplying and accumulating four S' elements and four coefficients each cycle.

  For some reason, I decided to buffer an entire row of S' and buffer an entire column of C when I only ever need four per cycle. To refactor and reduce my resource usage I would remove the two extra C and S' buffers and just read from RAM every cycle. However, due to my memory layout in order to remove the two extra S' buffers, I would have to rethink the way I am calculating T since the C_T co-routine can only control one port of the S/S' DPRAM. As a result, instead of getting one element of T every two clock cycles, I would have to get 2 elements of T every four clock cycles which only needs two elements of S' every cycle. This refactoring removes in total three 32-bit registers. Furthermore, since I am now using a memory access pattern, I could encode this into a universal addressing counter instead of having separate addressing counters for each matrix. Not only does this reduce my register count but it reduces my logic element count since I wouldn't have multiple addressing counters $\rightarrow$ multiple ripple carry adders.

  The exact same thought process could and should be extended to the C_S co-routine. I could also potentially do away with buffering the output of the ports altogether if the critical path does not produce any timing problems. I initially buffered the output since I was not sure if the RAM had any propagation latency.

  A further optimization on C_S could be to just write S either directly to the SRAM, or directly to the DPRAM without buffering it to the next common case initially done to fit two S elements in one word. In hindsight, after completing and integrating M3, I don't necessarily need an explicit W_S co-routine since my implementation falls exactly within 128 cycles in the worst case. Also, if I decided not to pack two elements of S in one word, my W_S co-routine would increase to 64 cycles from 32 cycles which would still be able to integrate nicely with my 128 cycles worst-case implementation of M3

- According to the timing analyzer in Quartus, the critical path starts at op_select and propagates to accumulator in the IDCT_controller. This makes perfect sense since the path consists of a 32-bit multiplier and potentially up to three 32-bit ripple carry adders. I was fully expecting either this or the path from index_ptr to RAM_address in my M3 to be my critical path.

3 Weekly Activity and Progress

Week	Progress
Week 1	Began reading the project document, started and finished M1 state table
Week 2	Began and finished coding M1, debugging M1
Week 3	Began and finished M2 state table, began coding M2 while considering M3 integration
Week 4	Debugging M2, began M3 thought process and code
Week 5	Finished M3 and integration

Table 5: Weekly activity and progress across the project duration

During the Lab and office hour sessions, Graham and Aaron gave me many insights into how I might approach certain design implementations. One anecdote is the access wire and index in my M3 that Graham suggested. This elegantly allows the lossless decoder's shifting window to wrap around the shift register.

4 Conclusion

Implementing the mic17 compression standard in hardware has exposed me to the subtle intricacies of digital hardware design. Having an implied clock at which all registers update is quite different compared to typical sequential programming. Over the past five weeks, I have learned how concepts taught throughout the semester apply to a real-world application.

In fact, working and mostly struggling through the project phase has really developed my appreciation for the computer engineers who have come before me. I could not even imagine the subtle bugs and errors that would arise when trying to implement image and video compression to variable image and video sizes. Yet, I am able to compress any oddly sized image with just the click of a button.

5 References

[1] Jason Thong, Adam Kinsman, and Nicola Nicolici, "COE3DQ5 Project Description 2023 Hardware Implementation of an Image Decompressor," McMaster University, Ontario, Canada, Tech. Report. 6 Sep. 2023.