COMPENG 3DQ5 – Final Project Report

Group 34 – Syed Ali Raza, Raul Singh Sidhu

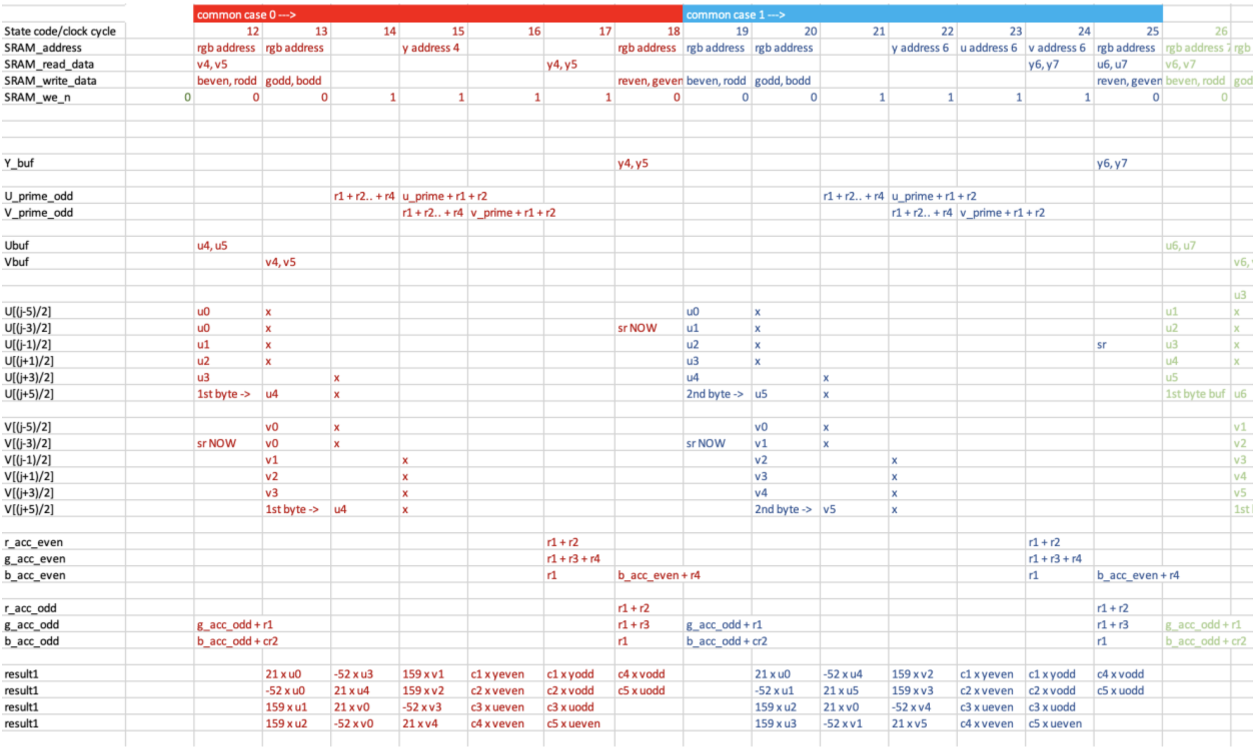
Monday, November 27th 2023

Introduction

This report details the development of a hardware-based image decompression system focusing on lossless decoding, dequantization, signal transformation (IDCT), interpolation, and colorspace conversion within strict resource and efficiency constraints. The project required implementation across three key milestones: decoding compressed bitstreams, executing inverse discrete cosine transformations on downsampled images, and performing upsampling with colorspace conversion to yield final RGB outputs.

Implementation Details

Upsampling and Colour Space Conversion (Milestone 1)



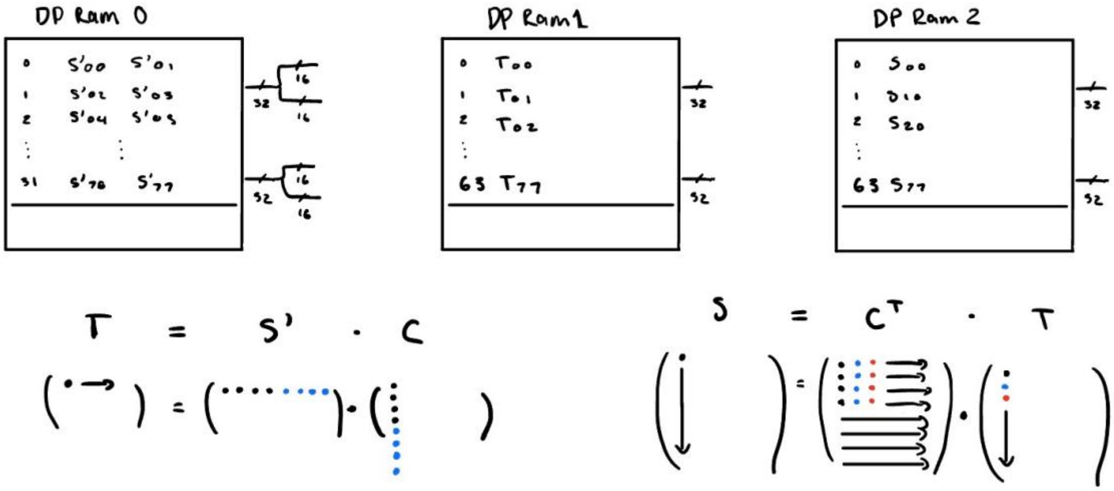
		common case 0 --->							common case 1 --->								
State code/clock cycle		12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	
SRAM_address		rgb address	rgb address		y address 4			rgb address	rgb address	rgb address		y address 6	u address 6	v address 6	rgb address	rgb address 7	rgb
SRAM_read_data		v4, v5					y4, y5							y6, y7	u6, u7	v6, v7	
SRAM_write_data		beven, rodd	godd, bodd					reven, geven	beven, rodd	godd, bodd					reven, geven	beven, rodd	god
SRAM_we_n	0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	
Y_buf								y4, y5							y6, y7		
U_prime_odd				r1 + r2.. + r4	u_prime + r1 + r2						r1 + r2.. + r4	u_prime + r1 + r2					
V_prime_odd					r1 + r2.. + r4	v_prime + r1 + r2						r1 + r2.. + r4	v_prime + r1 + r2				
Ubuf		u4, u5														u6, u7	
Vbuf			v4, v5														v6,
																	u3
U[(j-5)/2]		u0	x						u0	x						u1	x
U[(j-3)/2]		u0	x					sr NOW	u1	x						u2	x
U[(j-1)/2]		u1	x						u2	x					sr	u3	x
U[(j+1)/2]		u2	x						u3	x						u4	x
U[(j+3)/2]		u3		x					u4		x					u5	
U[(j+5)/2]		1st byte ->	u4	x					2nd byte ->	u5	x					1st byte buf	u6
V[(j-5)/2]			v0	x						v0	x						v1
V[(j-3)/2]		sr NOW	v0	x					sr NOW	v1	x						v2
V[(j-1)/2]			v1		x					v2		x					v3
V[(j+1)/2]			v2		x					v3		x					v4
V[(j+3)/2]			v3		x					v4		x					v5
V[(j+5)/2]			1st byte ->	u4	x					2nd byte ->	v5	x					1st
r_acc_even							r1 + r2							r1 + r2			
g_acc_even							r1 + r3 + r4							r1 + r3 + r4			
b_acc_even							r1	b_acc_even + r4						r1	b_acc_even + r4		
r_acc_odd								r1 + r2							r1 + r2		
g_acc_odd		g_acc_odd + r1						r1 + r3	g_acc_odd + r1						r1 + r3	g_acc_odd + r1	
b_acc_odd		b_acc_odd + cr2						r1	b_acc_odd + cr2						r1	b_acc_odd + cr2	
result1			21 x u0	-52 x u3	159 x v1	c1 x yeven	c1 x yodd	c4 x vodd		21 x u0	-52 x u4	159 x v2	c1 x yeven	c1 x yodd	c4 x vodd		
result1			-52 x u0	21 x u4	159 x v2	c2 x veven	c2 x vodd	c5 x uodd		-52 x u1	21 x u5	159 x v3	c2 x veven	c2 x vodd	c5 x uodd		
result1			159 x u1	21 x v0	-52 x v3	c3 x ueven	c3 x uodd			159 x u2	21 x v0	-52 x v4	c3 x ueven	c3 x uodd			
result1			159 x u2	-52 x v0	21 x v4	c4 x veven	c5 x ueven			159 x u3	-52 x v1	21 x v5	c4 x veven	c5 x ueven			

Module (Instance)	Register Name	Bits	Description
Milestone1	u_minus_five, u_minus_three, u_minus_one, u_plus_one, u_plus_three, u_plus_five, v_minus_five, v_minus_three, v_minus_one, v_plus_one, v_plus_three, v_plus_five	8	These are registers used within the shift register structure. They hold U and V data required for the interpolation of odd pixels. There two sets of 6 registers, for U and V respectively. u_minus_one and v_minus_one has the added functionality of containing the upsampled even pixel data.
Milestone1	u_prime_odd, v_prime_odd	32	These are two accumulating registers used for the summation of multiplications. They hold upsampled odd pixel data.
Milestone1	y_buf, u_buf, v_buf	16	These registers hold two bytes of Y, U, or V data when it is read from the SRAM.

Milestone1	y_counter, u_counter, v_counter, rgb_counter	18	These 3 registers are counters used to stagger our SRAM address to ensure our reads and writes are operating as they're designed to.
Milestone1	pixel_counter, row_counter	18	These 2 registers are counters created to loop from the lead in, to the lead out when an entire row of RGB pixel data is generated.
Milestone1	r_acc_even, g_acc_even, b_acc_even, r_acc_odd, g_acc_odd, b_acc_odd;	32	These registers are accumulators that handle the colourspace conversion using the upsampled data and coefficient matrix.
Milestone1	r_clip_even, g_clip_even, b_clip_even, r_clip_odd, g_clip_odd, b_clip_odd;	8	These registers hold our clipped accumulator data. This data is written back to the SRAM in pairs, 2 bytes of data per location.
Milestone1	op1, op2, op3, op4, op5, op6, op7, op8, result1, result2, result3, result4	32	These registers are used for handling our multiplications. They operate on a 32-bit signed format.
Milestone1	result_long1, result_long2, result_long3, result_long4	64	These registers hold the results for our multiplications before being clipped.
Milestone1	case_flag	1	This is a flag created to iterate between our common cases. It is used to determine when to read U and V data, and which byte of U and V buf is inserted to the shift register structure.
Milestone1	lead_out	1	This is a flag used to condense our leadout into our common case. We stop reading U and V data and only the second byte of the buffer is used for the shift register structure.

Milestone 1 requires a total of 270482 clock cycles. Each clock cycle lasts 20 ns, meaning Milestone 1 requires 5.41 ms to run. When it comes to multiplier utilization within our common case, we're doing 22 multiplications within 7 clock cycles to compute 2 pixels. The maximum number of multiplications that can be done within 7 clock cycles is 28 multiplications. 22/28 = 78.6% utilization within our common case. 4 pixels are computed within the lead in & lead out. This means the common case computes 316 of them. 2 per common case implies that is repeats 158 times. (22 + 22*158 + 0)/(44 + 28*158 + 8) = 78.1% utilization, proving our design falls within constraints.

Inverse Discrete Cosine Transform (Milestone 2)



Module (Instance)	Register Name	Bits	Description
Milestone2	row_index, col_index	3	These registers are counters used to navigate rows and columns of a single 8x8 block of data.
Milestone2	row_block, col_block	7	These registers are counters used to navigate which block the IDCT is being applied to.
Milestone2	row_address, col_address, col_address_ws	9	These registers are used to determine our SRAM address to ensure we read & write as designed. They represent the row and column for a specific block we're working in.
Milestone2	fs_counter	7	This register counts how many times S' is written.
Milestone2	fs_buf	16	This register is a buffer for S'. It enables us to pack two S' values within one location of DP RAM 0.
Milestone2	t_acc	32	This register is the accumulator for T.
Milestone2	t_counter	7	This register counts how many times T is written.
Milestone2	c_column_count	3	This register is used to select which column of the C matrix will be used as operands.
Milestone2	s0, s1, s2, s3	32	These registers are accumulators that hold the results for our S computations before being clipped.
Milestone2	s0_clip, s1_clip, s2_clip, s3_clip	8	These registers hold our clipped accumulator data. This data is written back to the SRAM in pairs, 2 bytes of data per location.
Milestone2	t_offset	3	This is a counter used to ensure that we iterate through our T matrix as designed.
Milestone2	s_counter	8	This register counts how many times S is written.
Milestone2	s_buf	16	This register is a buffer for S. It enables us to pack two S values within one location of the SRAM.
Milestone2	s_flag, fs_flag	1	s_flag is used to select which row values of the Ct matrix will be used as operands. fs_flag is used to determine when to stop fetching S' within our mega state.
Milestone2	s_offset	3	This register is a counter used to stagger which addresses need to be read from DPRAM 3 to ensure the SRAM is being filled as designed.
Milestone2	c0, c1, c2, c3	6	These are used to select values of the C matrix.
Milestone2	C0, C1, C2, C3	32	These are used to hold the C matrix data.
Milestone2	op0, op1, op2, op3, result1, result2, result3, result4	32	These registers are used for handling our multiplications. They operate on a 32-bit signed format.
Milestone2	result_long1, result_long2, result_long3, result_long4	64	These registers hold the results for our multiplications before being clipped.

Unfortunately, Milestone 2 is not entirely completed which makes it difficult to provide a latency analysis. However, we can break down the FSM in terms of clock cycles. The Fs, Ct, (Cs, Fs), (Ws, Ct), Cs, Ws, states last 64, 128, 128, 128, 128, 32 clock cycles respectively. This leaves our multiplier utilization to be 2400(128 + 128) / (2400(128 + 128 + 6) + 64 + 32) = 98.1%. This accounts for the 5-clock cycle stretch for accommodating our final reads and writes in between the mega states.

Resource Usage and Critical Path

The project's resource utilization, as indicated by the Quartus reports, demonstrates a substantial increase in the usage of logic elements from the initial lab 5 experiment 4s etup to the current implementation. Specifically, the starting point utilized 616 logic cells, compared to the 4006 logic cells in the project's current state (after adding milestones 1 and 2). This increase can be attributed to the added complexity and functionality required for the IDCT computation, and upsampling/colorspace conversion. The dedicated logic registers also saw an increase from the initial 369 to 1591 in the current build, making space for the additional data handling and processing needs of the expanded project. Despite this growth in resource demand, the project has been engineered within the hardware constraints, effectively managing the computational requirements.

During the development process, it has become evident that certain aspects of the design could benefit from optimization. Notably, in the transition to Milestone 2, the implementation required combining existing states into "megastates". Due to time constraints, we adopted a strategy that had minimal regard for resource efficiency. This resulted in a design with a higher usage of multiplexers and logic resources. Upon reflection, implementing the logic for these "megastates" from the ground up could have minimized resource consumption significantly. Given more time, we wish we could've taken a more cohesive approach to the FSM design. Such improvements would lead to a more resource-efficient solution without compromising the functionality or performance of the system.

The critical path analysis of Milestone 1, which is from node ***op1*** to our ***v_prime_odd*** accumulator involves the upsampling of chrominance values U' and V'. This is due to the nature of this path having to go through multiple adders and then back to an accumulator. The positive slack in the worst-case paths indicates that the design meets the timing requirements with the current configuration. These paths are associated with the multiplier operations in the image processing algorithm. These multiplier operations are crucial and may require optimization for improved performance or to prevent challenges if the design's complexity increases or if operating conditions change.

Weekly Activity and Progress

Week # (Date)	Project Progress	Individual Contributions
Week 1 (Oct 23rd)	Read the project document and began the state table for Milestone 1.	Project document was read individually, and state table was collaborated on.
Week 2 (Oct 30th)	Completed the state table and began the coding process of Milestone 1.	Both the state table and coding were collaborated on.
Week 3 (Nov 6th)	Milestone 1 was debugged and completed.	Completed milestone 1 together.
Week 4 (Nov 13th)	Milestone 2 was understood, and state tables were created. Coding began.	Understanding and creation of Milestone 2 state tables were done together.
Week 5 (Nov 20th)	Continued coding Milestone 2.	Coded Milestone 2 together.

All work was done together in the lab room. Both parties contributed equally.

Conclusion

In conclusion, we successfully implemented the interpolation and colourspace conversion aspects of an efficient image decompression system based in hardware, meeting the resource constraints and multiplier utilization targets. The system capably, performs inverse transforms on one block of downsampled image data, and executes upsampling with colorspace conversion, resulting in accurate RGB output. This project demonstrates the design challenges posed by hardware-based solutions in digital image processing.

Finished Milestone 1 was committed to GitHub on November 15th (finish panda cat).
A partial Milestone 2 was committed to GitHub on November 27th (Its over). This milestone successfully applies the IDCT to a single 8 x 8 block and writes it back to the SRAM using mega states with appropriate addressing logic.

References

[1] Nicolici, N., Thong, J., Kinsman, A. 2023. "COMP ENG 3DQ5 Project Description 2023: Hardware Implementation of an Image Decompressor" In: 3DQ5. Pages 1-30.