COMPENG 3DQ5 Project

Amhar Rishan - rishanm - 400460667
Victoria Black - blackv3- 400444380

1. Introduction

This project involved implementing the McMaster Image Compression Revision 18 (.mic18) decompression standard in hardware. The aim was to decode compressed image data for a 320x240 image, recover it, and display it on a VGA monitor using the Altera DE2-115 FPGA board. The compressed data was delivered via UART and stored in SRAM, while our hardware design performed the decoding, leveraging finite state machines, arithmetic processing, and resource-efficient logic to reconstruct the image.

2. Implementation Details

2.1 Upsampling and Colour Space Conversion (Milestone 1)

Milestone 1 tackled the initial stages of the decompression process: converting YUV data back to RGB and upsampling the U and V chroma components to restore the full color information.

SRAM_address	Y50Y51	V28V29	U28U29				
SRAM_read_data			Y50Y51	V28V29	U28U29		
SRAM_write_data				write Ro and Be	write Go and Bo		write re and ge
SRAM_we_n	1	1	1	0	0	1	0
R	Ro					Re	
G				Go		Ge	
B	Be			Bo			
Y				Y50Y51			
V							
U							
U_even						U25	
U[(j+5)/2]						U28	
U[(j-5)/2]						U23	
U[(j+3)/2]						U27	
U[(j-3)/2]						U24	
U[(j+1)/2]						U21	
U[(j-1)/2]						U25	
V_even					V25		
V[(j+5)/2]					V28		
V[(j-5)/2]					V23		
V[(j+3)/2]					V27		
V[(j-3)/2]					V24		
V[(j+1)/2]					V21		
V[(j-1)/2]					V25		
M1		V`1 = V1*21	V`3 = V3*159	U`2 = U2*-52		O1 = Y_odd` * 76284	O2 = Uo` * -25624
M2		V`2 = V2*-52	U`1 = U1*21	U`3 = U3*159		O3 = Vo` * 104595	O5 = Uo` * 132251
M3	E1 = Ye` * 76284	E2 = Ue` * -25624	E3 = Ve` * 104595	E4 = Ve` * -53281	E5 = Ue` * 132251	O4 = Vo` * -53281	
A1	Go = O1 + O2 + O4				Vo` = Vo - 128	Uo` = Uo - 128	Ro = (O1 + O3) >> 16
A2	Ue` = Ue - 128	Ve` = Ve - 128	V`1 = V`1 + 128	U`1 = U`1 + 128	Re = (E1 + E3) >> 16		Ye` = Ye - 16
A3	Bo = O1 + O5				Ge = (E1 + E2 + E4) >> 16		Yo` = Yo - 16
A4	V1 = V[(j+5)/2] + V[(j-5)/2]	V3 = V[(j+1)/2] + V[(j-1)/2]	U2 = U[(j+3)/2] + U[(j-3)/2]	Vo = (V`1 + V`2 + V`3) >> 8	Uo = (U`1 + U`2 + U`3) >> 8		
A5	V2 = V((i+3)/2] + V(i-3)/2]	U1 = U((i+5)/2] + U(i-5)/2]	U3 = U((i+1)/2] + U(i-1)/2]			Be = (E1 + E5) >> 16	
M1_buff		M3					M1
M3_buff				M3			M3
A2_buff			A2				
A3_buff		A2					
Y_odd	A3						
Go_buff		A1					
Bo_buff		A3					

Register Usage (For space constraints, these are the main ones):

Module (Instance)	Register Group	Size	Purpose
Milestone_1 (M1_unit)	y_counter, u_counter, v_counter	18 bits	Tracks number of values read
Milestone_1 (M1_unit)	data_counter, write_counter row_counter	18 bits	- Tracks read/write address offset - Tracks the rows are completed
Milestone_1 (M1_unit)	y_reg, u_reg, v_reg	16 bits	Used to store each of the respective values
Milestone_1 (M1_unit)	u[5][1:0], u[3][1:0], u[1][1:0], v[5][1:0], v[3][1:0], v[1][1:0]	16 bits	Used to hold the consecutive values of the odd U/V values. (U[(J-5)/2] = u[5][1] .etc)
Milestone_1 (M1_unit)	m1op1, m2op2, mult1 m2op1, m2op2, mult2 m3op1, m3op2, mult3	32 bits	Multipliers and their respective operands

Milestone_1 (M1_unit)	mult1_long, mult2_long, mult3_long	64 bits	Used to store the results of each of the multipliers
Milestone_1 (M1_unit)	add1, add2, add3, add4, add5	32 bits	Used as accumulators to store values for interpolation and colour space conversion
Milestone_1 (M1_unit)	Re, Ge, Be, Ro, Go, Bo	32 bits	Used to store the even/odd RGB values
Milestone_1 (M1_unit)	Re_clip, Ge_clip, Be_clip, Ro_clip, Go_clip, Bo_clip	8 bits	Used to store the clipped values of the RGB values

Latency Analysis: 3 addresses are being written every 7 clock cycles. The total number of RGB addresses that need to be written to is 115200. Therefore, we would need $\frac{115200\ x\ 7}{3} = 268800$ clock cycles on average to accomplish it. The multiplier usage is $\frac{16}{21} = 76.2\%$

Task Allocation (Amhar's Perspective)

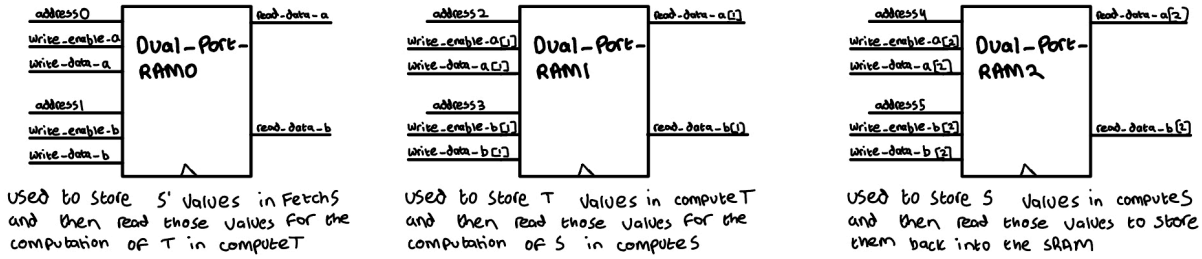
- Amhar: Developed the state tables for Milestone 1, handled variable initialization and manipulation, designed the LeadIn and CommonCase states, troubleshooted the code.
- Victoria: Designed the inner states for LO.

Task Allocation (Victoria's Perspective)

- Amhar: Developed the state tables for Milestone 1, handled variable initialization and manipulation, designed the LeadIn and CommonCase states, troubleshooted the code.
- Victoria: Designed the inner states for LO, wrote the internal running mathematics for the commonCase states following the state table. Asked TA questions in lab for both partners and resolved quartus errors.

2.2 Inverse Discrete Cosine Transform (Milestone 2)

Milestone 2 focused on recovering spatial-domain data from quantized frequency-domain coefficients. Using the IDCT, we reconstructed the downsampled YUV data into its original resolution



DP_S'_address_a	0	1	2	3	
DP_S'_data_a					
DP_S'_q_a		S'02 S'03	S'04 S'05	S'06 S'07	S'00 S'01
DP_S'_address_b					
DP_S'_data_b					
DP_S'_q_b					
DP_T_address_a		0	3		
DP_T_data_a		T0(add1)	T2(add3)		
DP_T_q_a					
DP_T_address_b		1			
DP_T_data_b		T1(add2)			
DP_T_q_b					

	CC0_CT	CC1_CT	CC2_CT	CC3_CT	CC4_CT	CC5_CT	CC6_CT	CC7_CT
M1	S'00*C03	S01 * C14	S02 * C23	S03 * C34	S04 * C43	S05 * C54	S06 * C63	S07 * C74
M2	S00 * C04	S00 * C05	S02 * C24	S02 * C25	S04 * C44	S04 * C45	S06 * C64	S06 * C65
M3	S01 * C13	S01 * C15	S03 * C33	S03 * C35	S05 * C53	S05 * C55	S07 * C73	S07 * C75
A1		M1+M3		A1+ M1+M3		A1+ M1+M3		A1+ M1+M3
A2	A2 + M1	M2	A2 + M1	A2 + M2	A2 + M1	A2 + M1	A2 + M1	A2 + M2
A3	A3 + M2 + M3		M2 + M3		A3 + M2 + M3		A3 + M2 + M3	

T Multiplication

DP_S_address_a	0	16					
DP_S_data_a	WRITE S0 - A1	WRITE S16 - A3					
DP_S_q_a							
DP_S_address_b	8						
DP_S_data_b	WRITE S8 - A2						
DP_S_q_b							
DP_T_address_a	16		32		48		0
DP_T_data_a							
DP_T_q_a	T20		T40		T60		T00
DP_T_address_b	24		40		56		8
DP_T_data_b							
DP_T_q_b	T30		T50		T70		T10

M1	C103 * T00	C114 * T10	C123 * T20	C134 * T30	C143 * T40	C154 * T50	C163 * T60	C174 * T70
M2	C104 * T00	C105 * T00	C124 * T20	C125 * T20	C144 * T40	C145 * T40	C164 * T60	C165 * T60
M3	C113 * T10	C115 * T10	C133 * T30	C135 * T30	C153 * T50	C155 * T50	C173 * T70	C175 * T70
A1		M1+M3		A1+ M1+M3		A1+ M1+M3		A1+ M1+M3
A2	A2 + M1	M2	A2 + M1	A2 + M2	A2 + M1	A2 + M2	A2 + M1	A2 + M2
A3	A3 + M2 + M3		M2 + M3		A3 + M2 + M3		A3 + M2 + M3	

S Multiplication

Register Usage:

Module (Instance)	Register Group	Size	Purpose
Milestone_2 (M2_unit)	col_block, row_block	6 bits	Tracks the index of the respective block
FetchS (FS_unit)	fetch_counter row_counter	6 bits	- Used to traverse each column in a block - Used to change from block to block
FetchS (FS_unit)	fetch_buff	6 bits	Buffers the value read from the SRAM
computeT (CT_unit)	row_counter	8 bits	Used to traverse through the rows in a block
computeS (CS_unit)	col_counter	5 bits	Used to go through the columns in a block
computeT, computeS	cc_counter	4 bits	A tracker used to count the common case's
computeT, computeS	m1, m1op1, m1_c m2, m2op1, m2_c m3, m3op1, m3_c	32 bits	Multipliers and their respective operands
computeT, computeS	mult1_long, mult2_long, mult3_long	64 bits	Used to store the results of each of the multipliers
writeS(WS_unit)	write_counter	4 bits	A counter that tracks the number of writes
writeS(WS_unit)	write_counter	4 bits	A counter that tracks the number of writes
writeS(WS_unit)	S0, S1	32 bits	Intermediate results for processing data in writeS.
writeS(WS_unit)	S0_clip, S1_clip	8 bits	Clipped versions of S0 and S1 for 8-bit pixel data in writeS.
writeS(WS_unit)	sram_row_count	13 bits	Tracks the row of SRAM being written
For all	address0, address1, address2, address3, address4, address5	7 bits	Address ports for the DPRAMs
For all	write_data_a[2:0], write_data_b[2:0] read_data_a[2:0], read_data_b[2:0] write_enable_a[2:0], write_enable_b[2:0]	32 bits 1 bit	- Data that is written to DPRAM - Data that is read from DPRAM - Write enable for DPRAM

Latency Analysis: Each 8x8 block is processed in approximately 500 cycles using three parallel multipliersm, ensuring real-time performan

Task Allocation

Amhar POV

- Amhar: Developed the state tables for Milestone 2. Wrote the fetchS, computeT and computeS modules, handling data fetch and intermediate computation stages, Did the majority of troubleshooting. Also developed the main Milestone_2 module.
- Victoria: Designed and implemented the writeS module.

Victoria POV

- Amhar: Developed the state tables for Milestone 2. Wrote the fetchS and computeS modules, handling data fetch and intermediate computation stages, Did the majority of troubleshooting.
- Victoria: Designed and implemented the writeS module, in addition to designing the computeT module. Also designed and wrote the main Milestone_2 module. Aided in troubleshooting.

2.4 Resource Usage and Critical Path

Resource Usage: The design used approximately 90% of the register (1339 registers), which is efficient compared to initial experiments. Optimization increased multiplier and RAM usage. As to reduce the number of elements required to complete the milestones. Comparing with Lab5 experiment 4, we used 369 registers and it makes sense as to why there are so many more registers being used. This can be explained due to the fact that we have several more modules.

Critical Path: The most significant slowdown was the multiplication stage during Milestone 1, when calculating the Ro values. It was found to be 13.803 ns, this can be explained as we take the value of multiplier 1 and previous multipliers and accumulate it to our Ro register after shifting it by 16 bits.

3. Weekly Activity and Progress

Week 1	Reviewed project specification, revised labs, Started the state table for Milestone 1
Week 2	Completed state table for Milestone 1, started coding Milestone 1
Week 3	Completed coding for Milestone 1 and started debugging.
Week 4	Completed debugging of Milestone 1, started state table for Milestone 2, started coding Milestone 2
Week 5	Focused on testing of milestone 2, documentation for submission.

Discussions were made with the professor, in-lab TAs as well as our friends (other group members) and others that were present during the lab period. These interactions were made to gain insights on how certain aspects of the project could be tackled.

4. Conclusion

Reflecting on this project, we're proud of the progress we made. Milestone 1 taught us the importance of efficient resource allocation and robust debugging, while Milestone 2 highlighted the challenges of implementing computationally intensive operations in hardware. Although there's room for improvement—the skills we developed will serve us well in future digital system designs.

Our **submission** is a completed Milestone 1 and a partially complete Milestone 2. For Milestone 1 the Github commit can be found on the repository dated 25/11/2024 with a message "FINAL M1 SUBMISSION" and for Milestone 2 it can be found dated 25/11/2024 with a message "Milestone 2 submission"

5. References

COE3DQ5 Course Notes
McMaster Image Compression (.mic18) Specification
Quartus Timing and Resource Utilization Tools