COE3DQ5 – Project Report

Wednesday Group 25

Sartaj Aujla, Yuvraj Bal

aujlas8@mcmaster.ca, baly@mcmaster.ca

Date: November 29th, 2021

Introduction

The objective of this project is to implement a custom image decompressor in hardware. Compressed data for a 320*240 pixel image will be delivered to the SRAM using the UART interface and the decompressed image is read by the VGA controller. The decompression takes place in 3 steps. Milestone 3 involves lossless decoding and dequantization of the compressed bitstream. Once the dequantized data is ready we move to milestone 2 which involves inverse signal transform and transforming the image into the frequency domain. The output of milestone 2 is the downsampled image. Milestone 1 takes the downsampled image as the input and performs interpolation to get the upsampled data. After this, we perform colourspace conversion to obtain the decompressed image

Design Structure

This project required several modules, some of them were given to us during labs and others were created according to the design requirements. In milestones 1 and 2 we used the SRAM_controller, UART_SRAM_interface, UART_receive_controller, VGA_SRAM_interface and VGA_controller, these modules were given in lab 5. The SRAM_controller interface generates the protocol signals to communicate with external SRAM. The UART_SRAM_interface monitors data from UART and writes into SRAM. The VGA_SRAM_interface generates the address for reading the SRAM and displaying the image on the monitor. Hsync and Vsync signals are generated using the VGA_controller module. These modules are the backbone of the design. The top-level module is called the project, it connects all the different modules together, controlling when certain modules are active. We also created additional modules named m1 and m2 to implement the design in a modular way. Module m1 performs interpolation on the downsampled values to obtain the upsampled values. Then it performs colourspace conversion which writes the output RGB values to the SRAM. The VGA_SRAM_interface reads from the SRAM and displays on the VGA monitor. Module m2 performs the inverse signal transform and writes the downsampled values into the SRAM. The finite state machine moves to milestone 2 once the image has been transmitted, and it sets the m2 start flag to 1. Once, milestone 2 will be completed, the m2_end flag will be 1 and the FSM will move to milestone 1. After, milestone 1 is completed the m1_end flag will be 1 and then it moves back to the IDLE state.

Implementation Details

Milestone 1: In this milestone, we perform interpolation and colourspace conversion. The design comprises 8 lead-in states, 16 common states and 4 lead-out states.

Lead In: The design enters the lead_in_0 state when the m1_start flag is asserted. We have used the lead-in cases to read the downsampled YUV values before we start performing multiplications for interpolation and colourspace conversion. The first five lead-in states are used to move to the addresses where U0-U3, V0-V3 and Y0Y1 are stored in the SRAM. There is a 2 cycle delay when reading values from the SRAM so once these values are read we use 2 shift registers U_shift and V_shift which are 48 bits wide to store these downsampled values. A 48-bit register is used because it can store 6 values which are 8 bits each. This solves our purpose of storing the 6 downsampled U and V values used for interpolation, particularly for the odd values. The following lead in states is used to buffer the UV values which are then shifted in the shift registers to be used for multiplication. In the last lead in the state, we perform multiplication to obtain the upsampled values and store the partial products in 2 different registers for U and V named U_odd_accum and V_odd_accum which are 32 bits each. Then we move to the first common case where the multiplications for upsampling and colorspace conversion are completed.

common case state								
SRAM_address	U0,U1	V0,V1	U2,U3	V2,V3	Y0, Y1			
SRAM_read_data			U0,U1	V0,V1	U2, U3	V2,V3	Y0, Y1	
U buf				U0,U1		U2, U3		
V buf					V0,V1		V2,V3	
Y buf								Y0, Y1
SRAM_write_data								
SRAM_we_n	1	1	1	1	1	1	1	1
operations	lead in 0							cc till col 23
Multiplier1								21(U0+U3)
Multiplier2								21(V0+V3)

Common Case: The first 2 common cases compute the upsampled value for U and V and the next 5 common case states compute the RGB value for 2 pixels using the upsampled values calculated before. In the first set of common cases, we read YUV values needed for the next interpolation and CSC calculation and buffer the values in U_even, U_odd 16 bit registers(similarly for V). The RGB values for the next 2 pixels are computed in the common case states 7-14, where the first 3 states carry out upsampling and state 10-14 perform CSC. In common case 11, we increment y address value by 1, except the case when the col_counter reaches 80, which is the end of the row.

52(U0+U2)	159(U0+U1)	76284*Y0	25624*U'0	132251*U'0	76284*Y1	53281*V'1	21(U0+U4)	52(U0+U3)	159(U1+U2)	76284*Y2	25624*U'2	132251*U'2	76284*Y3	53281*V'3	21(U0+U5)	
52(V0+V2)	159(V0+V1)	104595*V'0	53281*V'0	25624*U'1	104595*V'1	132251*U'1	21(V0+V4)	52(V0+V3)	159(V1+V2)	104595*V'2	53281*V'2	25624*U'3	104595*V'3	132251*U'3	21(V0+V5)	
			76284*Y0			76284*Y1										
					25624*U'1											
			R0		R1						R2		R3			
			G0			G1					G2			G3		
				B0		B1						B2		B3		

Upsampling: It is performed using two multipliers and the data to be upsampled is taken from the shift register. U_shift and V_shift are 48-bit registers used to store the 6 downsampled values which are 8 bits

each. Upsampling is done in the common case wherein each state once the multiplication has been done, the result is stored in the U_odd_accum, 32-bit register. This register carries the partial sum till the final value is computed where we shift 8 bits to the right to obtain the desired upsampled value which is further used for colourspace conversion

Colorspace Conversion: This step requires 10 multiplication for a pair of pixels. The multiplication is done in the common case and it is carried out in 5 clock cycles. The red value is stored in the RGB_red, 32-bit register(similar for blue and green). We use our common case to calculate the RGB value for 2 pairs of pixels, so the next CSC cannot be stored in the same registers because we haven't written the previous pair RGB values. For this, we had to use another set of RGB_red_buf, a 32-bit register(similar for blue and green) to store the RGB values after doing CSC. The last common case writes the RGB values for pixel 318,319 and then moves to the lead out state. If the last row(240) is computed we move from common case 15 to the M1_IDLE state.

Lead-out: Lead-out state starts when the col_counter value reaches 80, in the first lead-out state we set col_counter to 0 again and move to the next three lead-out cases which are used to add a delay of 3 clock cycles. From the last lead-out case, we move to the lead in the state(next row).

Design Approaches and Verification

The general design verification approach was to use the v0 testbench to test the flow of inputs to computations and how outputs are written back to the SRAM. Initially, we encountered an error in which the write location for every RGB value written to the SRAM was shifted by 1 from where it should have been actually stored in the memory. The issue was resolved by analyzing when the address was written, and whether the write enables are high or low using the ModelSim simulation viewer. We also use the HxD hex viewer to verify whether the correct address was being used enabling us to understand the appropriate address sections and offsets. We also faced an issue where some RGB red, green and blue values were being overwritten in consequent clock cycles before they could be appropriately written to the SRAM. This issue was resolved by creating buffer registers to temporarily store some of the values before they were written to the SRAM. Lastly, we had a small problem where some of the values in the accumulator were incorrect. Errors like these were made easier to debug by using an Excel sheet to calculate the correct values at different locations. The state table was referenced at all steps through the design verification process to ensure that the design was implemented into code properly and to see whether the simulation viewer matched the expected results.

Requirements Validation

Runtime: For one row, the lead-in is 8 clock cycles, the common case is 16 clock cycles * 80 iterations which are equal to 1280 clock cycles. There are 4 lead-out states. In total for one row, there are a total of 1292 clock cycles. Thus the total runtime for 1 row is 1292 clock cycles * 20 ns which is 0.258 ms. For 240 rows there are 1292 clock cycles * 240 rows which is equal to 310080 clock cycles. Thus, the total runtime for milestone 1 is 310080 clock cycles * 20 ns which is 6.202 ms.

Multiplier Utilization: During lead-in states, there is 12.5% multiplier utilization. In the common case, the multiplier utilization is 100% and in lead out, it is approximately 92.3%. There are a total of 1920 lead-in case clock cycles (8 * 240), 307200 common case clock cycles (16*80*240) and 960 lead-out case clock cycles (4*240). In total there are 310080 clock cycles. Therefore, 12.5%(1920 lead-in clock cycles) +100%(307200 common case clock cycles)+ 92.3%(960 clock cycles))/(310080 clock cycles) = 99.43% Multiplier Utilization. Therefore, the multiplier utilization is approximately 99.43% for milestone 1 which is greater than 85% thus meeting multiplier utilization requirements.

Register Usage for Milestone 1 (Module m1)

Register Name	Bits	Description
logic [15:0] Y_address, U_address, V_address; logic [17:0] RGB_address; (18 bit)	16/ 18	Address Register used for holding address of Y/U/V values for reading and RGB values for writing
logic [15:0] Y_even, Y_odd, U_even, V_even; logic [7:0] U_odd, V_odd; (8 bit)	16/ 8	Registers used to store Y/U/V values read from SRAM before transferring to shift register
logic [47:0] U_shift, V_shift;	48	Shift registers used to store U/V values for upsampling and colour space conversion
logic signed [31:0] U_odd_accum, U_odd_accum2; logic signed [31:0] V_odd_accum, V_odd_accum2;	16	Registers used to accumulate sum of upsampling products for next colorspace conversion
logic signed [31:0] RGB_red, RGB_red_buf; logic signed [31:0] RGB_green_buf; logic signed [31:0] RGB_blue, RGB_blue_buf;	18	Registers used to RGB, reg, green and blue values for colorspace conversion, values stored are pre-clipped.
logic [31:0] Multi_op_1_1, Multi_op_1_2, Multi_result1, Multi_op_2_1, Multi_op_2_2, Multi_result2; logic signed [31:0] Multi_result_long1; logic signed [31:0] Multi_result_long2;	18	Registers used to store multiplier operands and products
logic [8:0] col_counter;	8	Data counter used to track the column number in a row that has been processed

Critical Path and Timing Analyzer

After inspecting the critical path in the Timing Analyzer it is determined that the critical paths are the paths that go from changing the state bit to finishing the next state elements. Specifically, the path between state bit 0 and the RGB registers is used to store the RGB pixels and U/V accumulators. This can be explained because this path contains several combination logic like flip flops, OR gates, multiplexer selectors due to multiplication for Upsampling/CSC and 2 adders before the data flow to the RGB pixel register. This was verified by both checking the state table to see the flow of data to that result and the data arrival worst-case path in the Timing Analyzer Menu. This enabled us to view the location of the exact logic element/register through which the data is being passed when travelling through the path by viewing the same node on the RTL viewer.

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	2.196	m1:m1_unit|state.state_bit_0	m1:m1_unit|RGB_green[15]	clk_50	clk_50	20.000	-0.081	17.721
2	2.213	m1:m1_unit|state.state_bit_0	m1:m1_unit|U_odd_accum[22]	clk_50	clk_50	20.000	-0.085	17.700
3	2.219	m1:m1_unit|state.state_bit_2	m1:m1_unit|RGB_green[15]	clk_50	clk_50	20.000	-0.081	17.698

Milestone 2: In this milestone, we compute the YUV data from the YUV Pre IDCT block. This is computed using 4 major states which consist of Fetch S', Compute T, Compute S and Write S. Once the values from the SRAM are read, we store them in DRAM 1 in a way so that we fetch 8*8 blocks of data. Compute T involves reading from the DRAM and multiplication with a constant matrix and we write the data in DRAM 2. Then this data is read to compute S which we store in the second half of DRAM 1 and this data is written into the SRAM YUV block as input to milestone 1.
The write and read addresses differ because when we read from SRAM we need to store them in a way that 8*8 blocks of data can be ready for multiplication.

Compute T: Lead in involves reading data from DRAM 1 and using 3 multipliers we calculate T00 in 3 cycles after the read data is obtained. Data for computing next T is read at the same time multiplication is performed

DRAM_ADDRESS1	0	1	2	3	0	1	2
DATA__OUT0(even)		Y0	Y2	Y4	Y6	Y0	Y2
DATA__OUT1(odd)		Y1	Y3	Y5	Y7	Y1	Y3
Multiplier 1			Y0*C0	Y3*C24	Y6*C48	Y0*C1	Y2*C17
Multiplier 2			Y1*C8	Y4*C32	Y7*C56	Y1*C9	Y3*C25
Multiplier 3			Y2*C16	Y5*C40		Y6*C49	Y7*C57
Ct_ready			0	0	1	0	0
Ct					T00		

Common Case: For common case, we do not read Y6 and Y7, instead we store them in a buffer to reduce states and thus increase multiplier utilization

0	1	2
Y4	Y0	Y2
Y5	Y1	Y3
Y4*C33	Y0*C2	Y2*C18
Y5*C41	Y1*C10	Y3*C26
	Y6*C50	Y7*C58
1	0	0
T01		

Project Timeline (Equal Contribution)

Week	Both	Sartaj Aujla (50%)	Yuvraj Bal (50%)
1	Read project and milestone requirements. Meeting on Wed Oct 27 for 2 hours	Read project and milestone requirements, created state table document with basic design elements in milestones	Read project and milestone requirements understanding inputs and outputs

2	Started Milestone 1 State Table Design with focus on Upsampling	Milestone 1 State Table created, Lead in States	Milestone 1 State Table created
3	Continued Milestone 1 State Table with focus on Color Space Conversion	Milestone 1 Common Case Design and Fix Lead-in states	Milestone 1 Common Case and Lead-out states
4	Finished Milestone 1 State Table (Nov 17))and verified with TA. Wrote milestone 1 Code (Nov 17-21) and began design Verification (Nov 21)	Focused more on milestone 1 coding, (Lead-in and Common Case) and designing the code structure. Debugging the code eliminating compile errors	Focused more on debugging, design verification and additional design components initially overlooked with focus on the lead out and common case states
5	Finished Milestone 1 Design Verification (Nov 21 - 26). Milestone 2 State Table (Nov 21-28) and Finished Project Report (Nov 27-28)	Equal focus on milestone 1 debugging, started Milestone 2 code and Project Report	Equal focus on milestone 1 debugging, started Milestone 2 state table and Project Report

Conclusion

We were able to have partial development of the hardware implementation of an image decompressor. Milestone 1 was successfully completed which includes upsampling and colorspace conversion. Milestone 2, the inverse discrete cosine transform was only partially completed and milestone 3, was not attempted. It was learned that designing complex systems required thoughtful consideration, planning and design decisions. Specifically focusing on the design process and using tools like state tables to further break down into simpler system components. Furthermore, we learned how to perform design verification and debug design-related errors. Lastly, we further developed interpersonal skills including managing time and collaborating effectively. Ultimately enabling us to understand pivotal aspects of digital design and allowing us to connect all the material covered in this course.

Milestone 1 Commit: “Milestone 1 Fixed with Debug Files” (Nov 29 2021, 10:25 PM EST)

Link:https://github.com/3dq5-2021/project-group25-wednesday/commit/08cf46f838cf3828d75ee2adad8c3fe34368d71b (Latest Commit is also the same)

References

Dr. Nicola Nicolici: Reference Material and Lectures

Teaching Assistants:

- Thaejaesh Sooriyakumaran: FSM discussion
- Karim Mahmoud: Design verification
- Trevor Pogue: Design verification

Reference Material: COE3DQ5 Project Description 2021: Hardware Implementation of an Image Decompressor