COE3DQ5 – Project Report

Group 15 - Tuesday

Yiming Chen, 400230266, Ruiyi Deng, 400240387

cheny466@mcmaster.ca, dengr6@mcmaster.ca

Nov 25, 2021

Introduction:

The objective of this project is to gain experience in digital system design by creating a hardware implementation of an image decompressor. The hardware image compression specification we are using for this project is the. mic15. The hardware implementation was done by taking a 320 x 240-pixel image and running it through the Altera DE2 board through the UART interface, from a personal computer and then storing it in the SRAM. The compressed data is read by the image decoding circuitry devised by us with the specifications in the project manual, and then store it back into the SRAM. From there, the VGA controller will read and display the image.

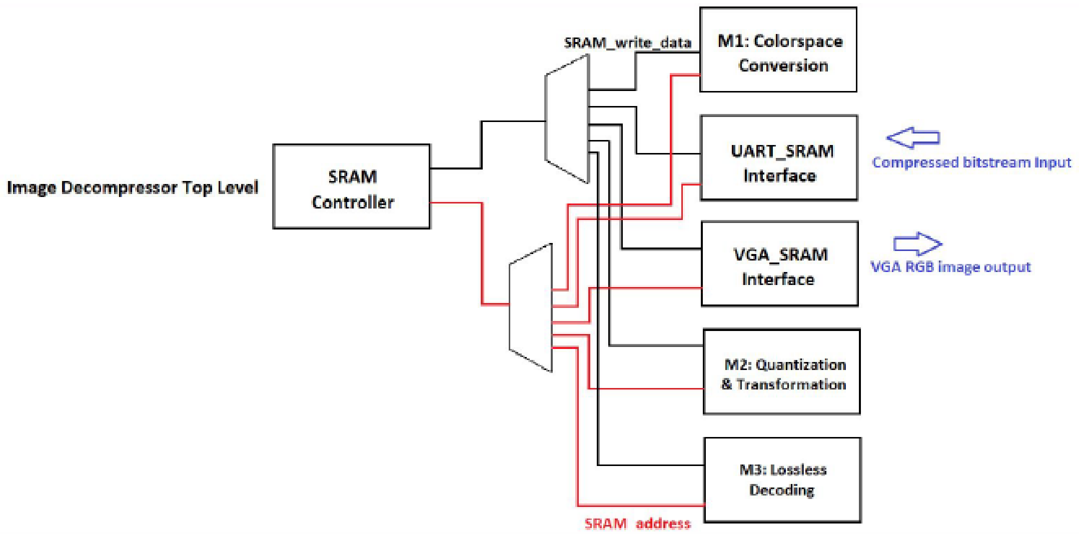


Figure 1. top-level block diagram design

Design Structure:

The design is composed of a top-level file named "project.sv" which connects all the units together including SRAM controller, UART SRAM interface, VGA SRAM interface and milestone 1,2,3 units. The top-level block diagram is shown in figure 1.

In the beginning, compressed image data is sent through the UART protocol, where the UART SRAM interface is used to transfer data into the SRAM. For all binary information is stored as a bitstream file, a lossless decoding is created in milestone 3 functioning as a filter for reading headers, obtaining data, and putting them in the correct place within the SRAM. Then, the quantization module (milestone 2) is created to convert the original image data in the time domain to YUV data in the frequency domain. DPRAM units is used as an intermediate storage place for matrix computation and sending back for computations. To export and display the compressed image, a YUV-to-RGB color-space conversion unit (milestone 1) is implemented. It holds chroma and luma sampling data and then computes following the requirements given. At the end of the process, the VGA unit will extract the computed RGB from the SRAM and generate the final .ppm file. During the verification process, tb_project.sv file is used to generate the .ppm file.

Project Implementation:

Milestone 1:

The milestone 1 unit consists of a general FSM and it can be described as 3 different stages, lead-in, common case, and the lead-out part. The strategy to use these three stages follows the utilization of the multipliers, as shown in the table below.

E	**E**	**E**	**V**	**U**	**O**	**O**	**O**
E	**E**	**V**	**V**	**U**	**U**	**O**	**O**

Since the U’ and V’ odd values were calculated using the array of U and V values, we can decrease the times of multiplication dur to the symmetric characteristic. We set 6 U and 6 V shift registers to keep track of the values used for both odd U’ and V’ values and the RGB values multiplications. In the milestone 1 unit, the assigned registers for shift registers are u[5:0] and v[5:0]. Because of the symmetry of the system, what is done to V is also the same to U, so the next part will focus on V. For the registers defined, v_buf is used to data read from the SRAM for the last shift registers. V_reg has two uses, first it will function as the storage space for the even multiplication like v’0 = v0 and v’2 = v1. After the even multiplication, it will play a role as the accumulator for the odd v’ calculations. This saves a lot of memory due to the full utilization of the v_reg register. Besides, there is a register named data_counter, it is used to monitor the u and v address and fetch the data from the SRAM.

For the RGB multiplication part, we have set registers R,G, and B to be the accumulator during the multiplication which in the table is the E and O part. The multiplication 1 consists of op1 and coeff1 and its result is stored in the multi1 register. All these registers are set to be signed for the negative sign calculations. The process is similar for the multiplier 2. Also, the clipping algorithm is implemented in line 546-548. This algorithm firstly judges the sign of the RGB data and then does the >>> 16 operation which represents 1/65536.

Apart from that, there is a special binary signal flag which is used in the common case stage. Because we only need to read u/v for one time every 16 clock cycles and the common case is set to 8 stages. This will help to avoid reading repeatably.

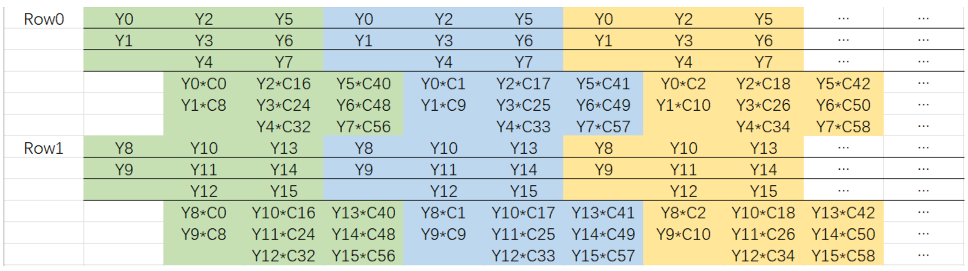
To analyzing the efficiency of the multipliers. For the common cases which occupy the most of the usage of the 2 multipliers. The efficiency can be expressed as (8*2)/(9*2) = 88.88% and compared with the common cases the lead in and lead out part can be negalected. Notes that the condition milestone 1 moves to the lead out stage is at line 440 (y_counter - row_counter >= 18'd160). Different from u/v, y registers will read for new values every 8 clock cycles, and for each row we just need to read 160 y values. When entering the lead out stage, it means the FSM is ready to continue with the next row.

For the debugging part, we used the HxD application to verify each address. In the wave.do file, we monitor M1_state, y_counter, u, u_buf and u_reg as well as the RGB multiplication part R and R_out. Coeff1 and op1 is also used when checking whether the multiplication have errors in the bit calculations. Moreover, the flag is used to check when to read the new u/v data. At first, because of the wrong implementation of the multiplications, we have countless mismatches. When spending about 15 hours, we finally reached the no mismatch results and successfully generate the motorcycle.ppm file after modifying the VGA_address set in the project.sv file.

Milestone 2:

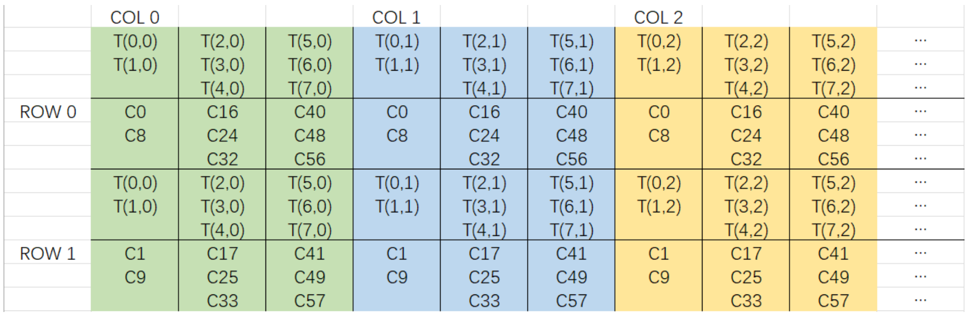
In the milestone 2, FSM computes the Pre-IDCT to Post-IDCT value (YUV) through 4 stages: Fs, Ct, Cs, and Ws. In state Fs, the Pre-IDCT value S' of a block is transported from SRAM into a DPRAM every other cycle in order to make that the DPRAM has 2 values per address. In state Ct and Cs, the FSM controls address buses and decide where to write value S', T and C. Matrix multiplication of $T = (S' * C) / 256$ is performed in Ct and $S = C^T * T$ is computed in Cs in the same time. During Ct, T is stored 1 value per address (32-bits) in DPRAM1 for further calculation in order to use the parallel R/W function in DPRAM rationally. Two values were stored in S per address (every 8-bits) into DPRAM0 as SRAM will read and write S into itself in state Ws. We set the state Cs (nth block) and Fs (n+1st block) to be executed in the same time to meet the efficiency requirement in the document, and for state Ct (n+1th block) and Ws (nth block) are executed in the same time just like Cs and Fs mentioned above. In this case, the lead-in is the Fs & Ct for the first Y block and Cs & Ws are computed by the last V block's lead-out.

As for the DPRAM planning, DPRAM0 stores two 16-bits S' together from address 0 ~ 63 and two 8-bits S in Y0Y1 format from 64 ~ 127, and they follow the same order in SRAM. For example, S'0S'1 at address 0, S'2S'3 at address 1 and Y0Y1 at address 64, Y2Y3 at address 65 etc. Since 32-bits are necessary to compute T under the requirement of 85% utilization, each T is stored as matrix row by row in the DPRAM1, by doing this, two Ts can be read parallelly to compute S in 3 cycles. For example, T(0,0) at address 0 and T(0,1) at address 1 until hits next row (T(1,0) at address 8).



Row0	Y0	Y2	Y5	Y0	Y2	Y5	Y0	Y2	Y5	···	···
	Y1	Y3	Y6	Y1	Y3	Y6	Y1	Y3	Y6	···	···
		Y4	Y7		Y4	Y7		Y4	Y7	···	···
		Y0*C0	Y2*C16	Y5*C40	Y0*C1	Y2*C17	Y5*C41	Y0*C2	Y2*C18	Y5*C42	···
		Y1*C8	Y3*C24	Y6*C48	Y1*C9	Y3*C25	Y6*C49	Y1*C10	Y3*C26	Y6*C50	···
			Y4*C32	Y7*C56		Y4*C33	Y7*C57		Y4*C34	Y7*C58	···
Row1	Y8	Y10	Y13	Y8	Y10	Y13	Y8	Y10	Y13	···	···
	Y9	Y11	Y14	Y9	Y11	Y14	Y9	Y11	Y14	···	···
		Y12	Y15		Y12	Y15		Y12	Y15	···	···
		Y8*C0	Y10*C16	Y13*C40	Y8*C1	Y10*C17	Y13*C41	Y8*C2	Y10*C18	Y13*C42	···
		Y9*C8	Y11*C24	Y14*C48	Y9*C9	Y11*C25	Y14*C49	Y9*C10	Y11*C26	Y14*C50	···
			Y12*C32	Y15*C56		Y12*C33	Y15*C57		Y12*C34	Y15*C58	···

Figure 2. Ct matrix multiplication

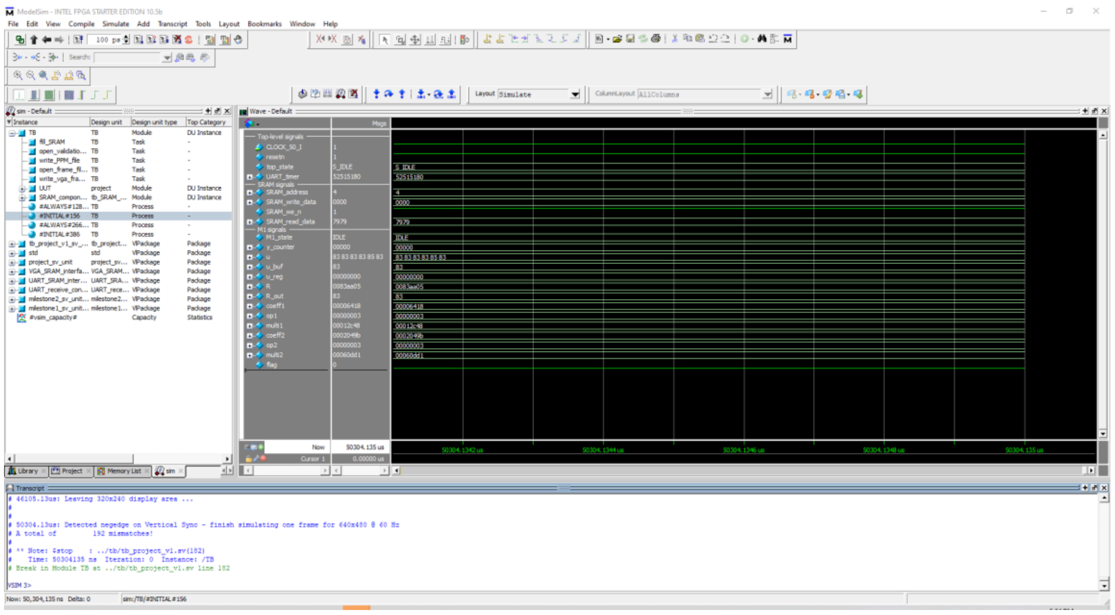


	COL 0			COL 1			COL 2			
	T(0,0)	T(2,0)	T(5,0)	T(0,1)	T(2,1)	T(5,1)	T(0,2)	T(2,2)	T(5,2)	...
	T(1,0)	T(3,0)	T(6,0)	T(1,1)	T(3,1)	T(6,1)	T(1,2)	T(3,2)	T(6,2)	...
		T(4,0)	T(7,0)		T(4,1)	T(7,1)		T(4,2)	T(7,2)	...
ROW 0	C0	C16	C40	C0	C16	C40	C0	C16	C40	...
	C8	C24	C48	C8	C24	C48	C8	C24	C48	...
		C32	C56		C32	C56		C32	C56	...
	T(0,0)	T(2,0)	T(5,0)	T(0,1)	T(2,1)	T(5,1)	T(0,2)	T(2,2)	T(5,2)	...
	T(1,0)	T(3,0)	T(6,0)	T(1,1)	T(3,1)	T(6,1)	T(1,2)	T(3,2)	T(6,2)	...
		T(4,0)	T(7,0)		T(4,1)	T(7,1)		T(4,2)	T(7,2)	...
ROW 1	C1	C17	C41	C1	C17	C41	C1	C17	C41	...
	C9	C25	C49	C9	C25	C49	C9	C25	C49	...
		C33	C57		C33	C57		C33	C57	...

Figure 3: Cs matrix multiplication

For computation, we are allowed to use exactly three multipliers in 1 clock cycle, so we need 8 multiplication × 3 clock cycles to get ca1 value. C is carried out using a mux selected by coeff 1,2,3. In state Ct, we use the FSM to read the first row of S' 8 times and multiply each number with 7 rows from the C matrix. The detail is explained below in figure 2.

C^T coefficients become the new row multiplies 8 columns as the order in matrix multiplication matters, and in this case, they are Ts. The detail is explained below in Figure 3.



```
# 46105.13us: Leaving 320x240 display area ...
#
#
# 50304.13us: Detected negedge on Vertical Sync - finish simulating one frame for 640x480 @ 60 Hz
# A total of          192 mismatches!
#
# ** Note: $stop    : ../tb/tb_project_v1.sv(182)
#    Time: 50304135 ns  Iteration: 0  Instance: /TB
# Break in Module TB at ../tb/tb_project_v1.sv line 182

VSIM 3>
```

Now: 50,304,135 ns Delta: 0 sim:/TB/#INITIAL#156

Figure 4. Simulation result of the milestone 2.

We decide to fetch S' into DPRAM and write S into SRAM every other CC to let the multiplication be synchronized with the data read from or written into SRAM. When two blocks are processed together in common case, the utilization is 100% as Fs takes 64 times but only writes 32 times, Ct and Cs take 192 CCs, Ws takes 32 times for an entire block, we use 3 multipliers together to compute Ct and Cs. The lead-in Fs and lead-out Ws is ignored because compared to the computation part it can be neglected. Below is the estimated utilization when we take the lead-in and lead-out state in CC into consideration:

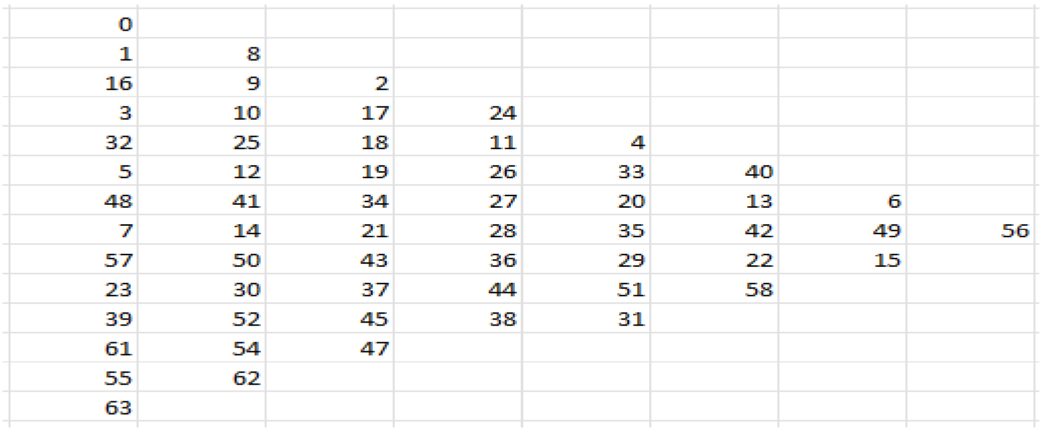
$$Cs \ \& \ Fs = (192*8/9)/(192+6) = 86.2\%$$
$$Ct \ \& \ Ws = (192*8/9)/(192+6) = 86.2$$
$$Fs \text{ and } Ws = 0\%$$

During the debug and verification stage, we found that there are always 192 mismatches in the result as we can see in figure 4.

It causes the block of first lead-in to be entirely black instead of the correct color and pattern. After debugging process, we are able to determine that the problem is from the MULTI part of our code. But we didn't figure out how to solve the problem and get it fully passed in the testbench, so the milestone 2 is not fully passed but with some mismatches.

Milestone 3:

Milestone 3 generally contains three parts: zig-zag W/R, header detection and shift. For zig-zag W/R we intend to implement the order in this manner as shown below.



0							
1	8						
16	9	2					
3	10	17	24				
32	25	18	11	4			
5	12	19	26	33	40		
48	41	34	27	20	13	6	
7	14	21	28	35	42	49	56
57	50	43	36	29	22	15	
23	30	37	44	51	58		
39	52	45	38	31			
61	54	47					
55	62						
63							

Figure 5. zig-zag order

Project Timetable:

Week	Project Progress	Member Contribution
1	Read through the project description document and get a big picture of image decompressor.	Read the document and prepare for the state tables.
2	Start building, modifying, and completing the state table for milestone Verify the correctness of the state table for milestone 1 module with professor.	Yiming Chen: Build the state table and determine the mode to use as well as the framework for which clock cycle to implement lead-in, common case, and the lead-out stage. Ruiyi Deng: Modify some errors in the state table and fill in the address part and multiplication which helps to code.

3	Continue to work with milestone 1. Debug milestone 1 module at the compilation stage.	Yiming Chen: Start coding with the milestone 1. Implement the registers and the multiplication part. Do analysis and complete the lead in and lead out part. Ruiyi Deng: Implement the preliminary state table with the codes. Build the framework for the common case.
4	Continue to debug for the milestone 1. Fix mismatches through testbench files. Start to build the milestone 2 state table. Start to code for milestone 2.	Yiming Chen: Start to build the verification environment and start to debug. At the first time after running the .tb file there are a lot of mismatches. Ruiyi Deng: Start to build state table for milestone 2 and update the table according to the lectures. Start to code for milestone 2.
5	Continue to code for milestone 2. Modify the top-level files project.sv to accommodate for the milestone 2. Try to fix mismatches and start writing the report.	Yiming Chen: Finish debugging with milestone 1, after reaching no mismatches, help code for the milestone 2 for Fs and Ws part and do the verification for milestone2. Ruiyi Deng: Continue to code for milestone2 and focus on the Cs and Ct multiplications. Build Cs_Fs and Ct_Ws stages and modify the errors.

Conclusion:

In conclusion, this project gave us an unforgettable learning experience, it was meaningful but also exhausted. During the five weeks, we had a better understanding of how to implement a scaled project in the future and how a digital system is designed. We learned how to teamwork, communication and manage our time. To us, the difficulty of the project is hard and quite time consuming. Without what we learned above; we would not finish milestone 2 unit in the last week. Also, we are not able to meet in person as we live in different cities, so we have used some online cooperative editing software like Microsoft Teams to work together on the same document at the same time, check the correctness of the part wrote by another teammate and fix the wrong places.

Other notes: About the git push part

During the project period, complete milestone 1 is reached with no mismatches. The commit for this is 2fd9d2a made on Nov 23. For milestone 2 there remains a problem that the first block of the .ppm file is black and the number of mismatches is 192. We infer this comes from the lead in part (64+64+64)(Y+U+V), however due to the deadline we cannot fix it in time. To check milestone 2 please refer to the last commit. For milestone 3 some basic ideas of designing are provided in the report. Also, we do not have enough time to build it and verify these ideas.