ENGINEERING

Project Report

COE3DQ5 Project

COMPENG 3DQ5 – Digital Systems Design

Lab Section L02

Wed - Group 51

Olukemi Odujinrin (odujinro, 400317700)

Taiwo Rabiu (rabiut, 400331895)

Submitted: November 28, 2022

Introduction

The main objective of the COE3DQ5 project is to make students familiar with work on a larger design scale that also includes the hardware implementation of several types of digital signal processing algorithms. Students will gain experience in digital system design by implementing the custom McMaster Image Compression revision 16 (.mic16) image compression specification in hardware. In addition, the hardware design and implementation must meet the 50 MHz clock constraint, some latency constraints while using hardware resources wisely.

This project involves hardware implementation of an image decompressor. Compressed data for a 320 x 240 pixel image will be delivered to the Altera DE2 board via the universal asynchronous receiver/transmitter (UART) interface from a personal computer (PC) and stored in the external static random access memory (SRAM). The image decoding circuitry will read the compressed data, recover the image using a custom digital circuit designed by you and store it back to the SRAM, from where the video graphics array (VGA) controller will read it and display it to the monitor.

Design Structure

Reused Modules	Custom Modules
SRAM_controller	Project
VGA_SRAM_interface	Milestone1 (Completed)
UART_SRAM_interface,	Milestone2 (Implementation Completed)
PB_controller	

SRAM_controller, VGA_SRAM_interface, UART_SRAM_interface and PB_controller are from previous labs used to process the information that Milestone 1 and 2 both manipulate.

For Milestone 1, we create a module (separate sv files) that is called by the main top module (project.sv) to control which the module runs and when. Milestone1 is a module used for interpolation and color conversion of the RGB pixels and is expected to run after the completion of Milestone 2.

For Milestone 2, we create a similar module and is called by the project module. The purpose on this custom module is to implement IDCT (Inverse Discrete Cosine Transform).

For the Project module, it is used as the main controller of all the modules listed above, to enable and disable clocks, the UART, the timer and the modules. It is the "control panel" and we modified the project modules to include our custom modules.

Implementation Details

Project Progress

Week	Task
Week 1	• Read & understand project outline • State Table for Milestone 1
Week 2	• Complete State Table for Milestone 1 • Implement Milestone 1
Week 3	• Verify & Complete Milestone1
Week 4	• State Table for Milestone 2
Week 5	• Implement & Verify Milestone 2

Team Contribution

Olukemi Odujinrin	Taiwo Rabiu
Debugged M1 State Table	Began and generated M1 State Table
Implemented second half of M1 code	Debugged M1 testbench and VGA controller
Verified M1 (Testing)	Implemented first half of M1 code
Generated M2 State Table	Verified M1 (Testing)
Implemented M2 code	Resolved Implementation Errors M2
Verified M2 (Testing)	Verified M2 (Testing)

FSM & State Table

To create our design, we created state tables to map out the follow of the states, registers, and the data that we manipulated. Attached to our GitHub repo located (in doc folder) are the state table we created to design and implement M1 & M2. Theses tables helped visualize the timing and examine what happens at each clock cycle. As designers, it helps us decide when and where we will obtain data, manipulate it, and store it.

For Milestone 1, there were 3 main states: Lead In, Common Case and Lead Out. These 3 mega states played different roles and in total approximately 70 states were used to complete Milestone 1. Understanding that the Lead In and Lead Out are only executed once, the Common Case executes the most times for Milestone 1, accumulating the most clock cycles. We did our best to design a system that used a minimal number of states by performing operations (reading, writing, addition, subtraction, multiplication, storing) synchronously where possible. In general, to complete this design, there 4 reads of YUV (Even& Odd) and 6 writes of RGB (Even & Odd). This all took 16 – 18 clock cycles, indicating the common case.

For Milestone 2, there were 4 mega states: Fetch S', Calculate T, Calculate S and Write S. A common case is obtained for CS and WS here FS' and CT are executed as the same time, respectively. In general, to complete this design. T and S were calculated in 4 clock cycles (1 matrix multiplication). FS' occurred in 64 clock cycles (plus lead in and out) to fetch all 64 S' values of the 8x8 matrix.

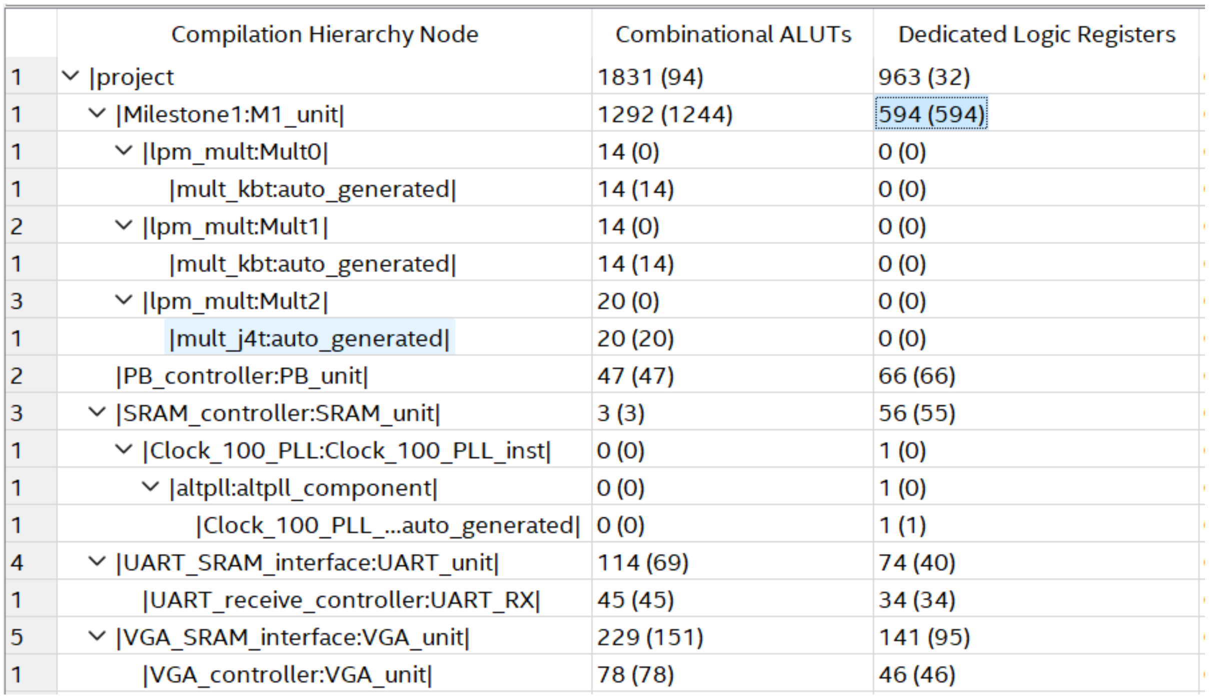
Registers, Multiplier, Counter, Adders & Comparator Usage

For Milestone 1, 3 multipliers were allowed and therefore we decided use 1 multiplier for the multiplication of U’, 1 multiplier for the multiplication V’ (interpolation) and the last multiplier for RGB (color conversion). For Milestone 2, we were limited to 2 multipliers. They were used to implement the matrix multiplication for S’*C and Ct*T.

We believe this is the best design for the limited multipliers we were given and allowed us to use less states and resources.

We used registers along side adders, counters, and comparators. For simplicity and a smooth debugging, we did create more registers for every element that needed to be kept tracked of. For example, u1, u2, u3, u4, u5, u6 and u7 were separate registers. Although it adds to the register list, it made debugging easier and since they were used throughout the design, the trade-offs are justified.

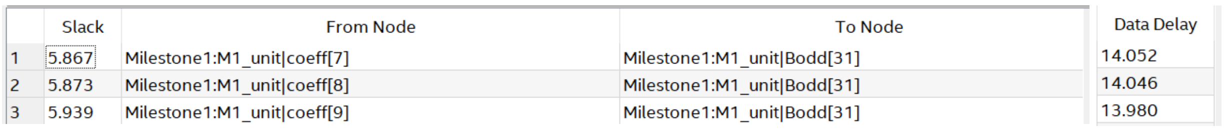
Below are the registers used in Milestone 1. In total, 594 registers were used for Milestone 1. Note that upon completion of Milestone 2 and 3 the same analysis would be applied.



	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers
1	|project	1831 (94)	963 (32)
1	|Milestone1:M1_unit|	1292 (1244)	594 (594)
1	|lpm_mult:Mult0|	14 (0)	0 (0)
1	|mult_kbt:auto_generated|	14 (14)	0 (0)
2	|lpm_mult:Mult1|	14 (0)	0 (0)
1	|mult_kbt:auto_generated|	14 (14)	0 (0)
3	|lpm_mult:Mult2|	20 (0)	0 (0)
1	|mult_j4t:auto_generated|	20 (20)	0 (0)
2	|PB_controller:PB_unit|	47 (47)	66 (66)
3	|SRAM_controller:SRAM_unit|	3 (3)	56 (55)
1	|Clock_100_PLL:Clock_100_PLL_inst|	0 (0)	1 (0)
1	|altpll:altpll_component|	0 (0)	1 (0)
1	|Clock_100_PLL_...auto_generated|	0 (0)	1 (1)
4	|UART_SRAM_interface:UART_unit|	114 (69)	74 (40)
1	|UART_receive_controller:UART_RX|	45 (45)	34 (34)
5	|VGA_SRAM_interface:VGA_unit|	229 (151)	141 (95)
1	|VGA_controller:VGA_unit|	78 (78)	46 (46)

Timing Analysis

The Timing Analysis below is based on Milestone 1. It is assumed the same analysis would be applied in the same manner for Milestones 2 & 3 upon completion.

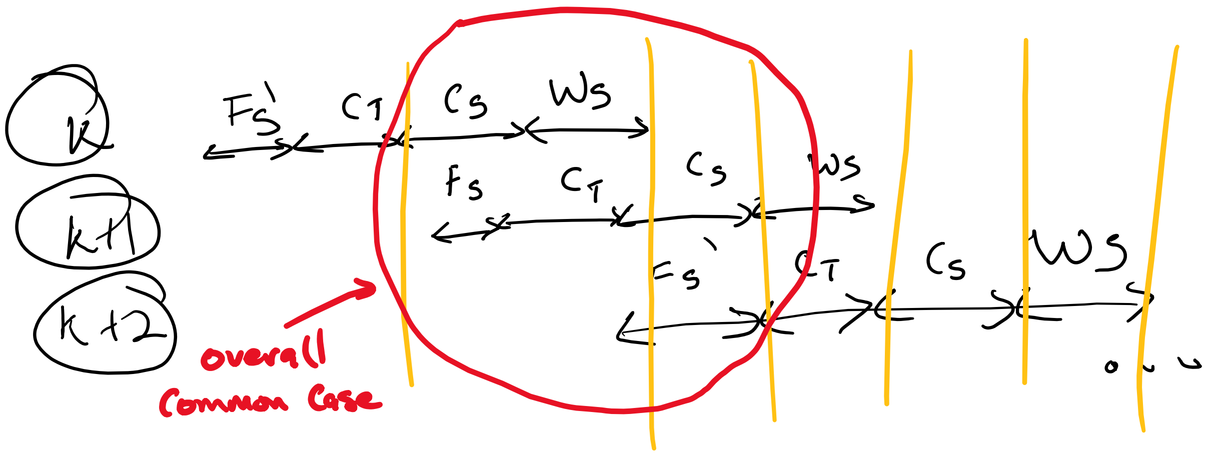


	Slack	From Node	To Node	Data Delay
1	5.867	Milestone1:M1_unit|coeff[7]	Milestone1:M1_unit|Bodd[31]	14.052
2	5.873	Milestone1:M1_unit|coeff[8]	Milestone1:M1_unit|Bodd[31]	14.046
3	5.939	Milestone1:M1_unit|coeff[9]	Milestone1:M1_unit|Bodd[31]	13.980

Above show the critical path which is the longest path or worst case timing path that our design takes to execute. Below are the top 3 critical delays in our design. The top slack is 5.867 with the highest data delay of 14.052. This tells us that the longest path (1) is created from the source node Milestone1:M1_unit|coeff[7] to Milestone1:M1_unit|Bodd[31], the destination node, making it the critical path of our design.

Design Alternatives

We understood that there are many ways to implement the hardware of an image decompressor. Milestone 1 came with a lot of freedom in terms of implementation and as a team we choose the design that we did as it met with the constraints, and it was the easiest to implement. The other design alternatives that can be seen in the multiple state table versions we came up with, were all valid. With enough time, they too could have also been implemented, however, some were too long in clock cycles or used more registers. Overall, our final Milestone 1 design served as the best alternative design. Milestone 2 had more constraints therefore, the design alternatives are limited but with that, we took the same approach for designing Milestone 2. Although incomplete the approach we took for milestone 2 was to first fetch S' for Y, calculate T, then calculate S and write S. This in total would have taken 70+ states. After Y is done the program would return to the mega sates (highlighted in red) for U and V with a slight change in starting address and bound case implementation. The alternative was to create another state machine (with the same approach) after the completion of the first finite state machine to do U and V. This is also valid as it produces the same outcome but in order to maintain high utilization, we decided with the first implementation as it will reduce the total number of clock cycles used.

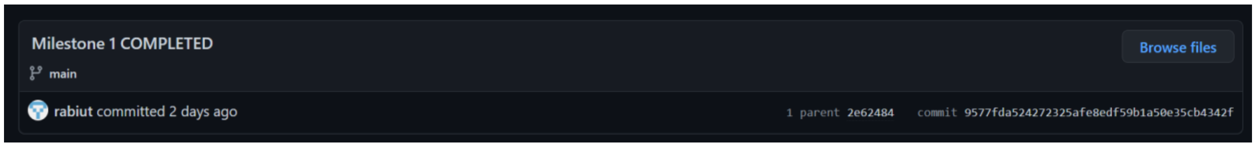


Verification & Debugging

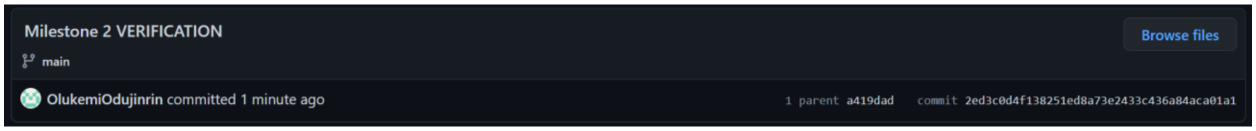
The verification stage required a lot of patience, we spent a good amount of time verifying Milestone 1 and successfully completed it. For Milestone 2, we design, mapped (state table) and implemented the code. We began verification but unfortunately, did not complete it. To fix bugs for M1, we cross checked the outputs at each mega state and ensured key states, resulted in expected output on the Modelsim waveform. To see in detail the behaviour of our design, we edited the test bench to tell us more information about each state. Displaying information like the M1_state, Reven, Geven, Bodd and more, we can see were the error lied, go back to the state in the code and verify our edits. This made it easier to spot the bugs and complete the milestone. Overall, debugging is a long rigorous task, but Modelsim features (for example wave.do) made it possible to identify problems we may have missed, fix our state table based on the feedback and change our code.

Conclusion

Milestone 1 Status – **Date:** Nov 25th, 2022 **Commit Message:** “Milestone 1 COMPLETED”


Milestone 1 COMPLETED
main
rabiut committed 2 days ago
1 parent 2e62484 commit 9577fda524272325afe8edf59b1a50e35cb4342f

Milestone 2 Status – **Date:** Nov 28th, 2022 **Commit Message:** “Milestone 2 VERIFICATION”


Milestone 2 VERIFICATION
main
OlukemiOdujinrin committed 1 minute ago
1 parent a419dad commit 2ed3c0d4f138251ed8a73e2433c436a84aca01a1

Milestone 3 Status – No Implementation Available

Our learning experience has been mostly positive. We worked well as a team and each of us helped one another understand and execute the project. We did not attempt to conquer and divide and although we did not complete the entire project we stand by that decision because we both benefit from understanding the project. This will help us during our cross examination as we took the opportunity to only move forward once both group members understood the task at hand.

We utilized the TAs and classmate assistance wherever appropriate, and we focused on asking conceptual questions such as “How many clock cycles does it take for the data in the SRAM to show up after reading?” rather than direct, how to implement XYZ questions. The TAs and our classmates helped fill holes in our knowledge which helped us get as far as we did.

References

1. COE3DQ5 Labs 1 – 5 (GitHub)
2. COE3DQ5 Avenue lectures, notes & Lab videos
3. Project Specification Document – 3dq5 2022 project description
4. TAs: Mushifiqur Rahman, Graham Power, Mihail Georgiev and Amin Eljirby