CE 3DQ5 Project Report

Group 6 - Tue: Xinyu Chen(400221680); Ruiyan Guo(400256752)

1. Introduction

CE 3DQ5 is a project based course that focuses on digital design. The concept of hardware design was introduced and was applied throughout the labs.This project is a conclusion of what was learned in the past separately. It integrates the Top FSM, UART transmission, VGA controller and the accessing of SRAM / DP-RAM. It provides a clear interference relationship of these digital design components by implementing them through the clocked state transitions.

2. Design Structure

This project includes 5 modules in total, which are top level FSM, Milestone2, Milestone1, DP-RAM and VGA controller. The top level FSM module uses the SRAM_address, SRAM_we_n and SRAM_write_data to control the SRAM, since the SRAM are used by both Milestone 1 module and Milestone 2 module. The Milestone 2 module is used to generate and store the YUV data into the corresponding locations in SRAM, which will be treated as the input of the Milestone 1 module. When implementing the Milestone 2 module, the DP-RAM module is used as a temporary storage place to store the matrix data for further multiplication purposes and their intermediate multiplication results. After the YUV data is derived and writtened properly, the designed Milestone 1 module converts it into RGB data which will then be saved into its corresponding SRAM locations. At last, the VGA controller module will interface with the SRAM module to read and display the RGB pixels on the monitor.

3. Implementation Details

3.1 Milestone 1

3.1.1 Hardware Structure and FSM

The milestone 1 contains two processes, compute the value of Y / U' / V' and calculate the RGB pixel values. Since the pixels are processed in pairs, an odd and an even pixel pair will be processed at the same time. Y values are stored in pairs in the register Y_value. U' and V' are distinguished by even and odd, they are saved separately in U_even_value, U_odd_value, V_even_value, V_odd_value registers. The RGB values are also saved separately with even and odd, in R_odd, G_odd, B_odd, R_even, G_even, B_even registers. When calculating U' and V' values, two sets of shift registers are used to store U and V values respectively. Each shift register set contains 6 registers, which values are updated once for each pixel pair calculation. For the calculation part, two multipliers are used in total.

For the finite state machines (FSMs), 25 Lead In states, 9 Common Case and 39 Lead Out states are used. The lead in states deal with the very first pixel pair in each row, the common cases deal with the pixel pairs from 3 ~ 157, the lead out state deal with pixel pairs from 158 ~ 160.

3.1.2 Efficiency and Timing

During the 9 clock cycles for the common case, the multipliers are used 8 times, the resulting efficiency can be calculated as 8/9 = 88.9%. The whole simulation takes approximately 33536us to complete, which are 1676800 clock cycles in total.

3.1.3 Design Thought

For milestone 1, the very basic concept that was followed throughout the design process is what needs to be prepared, what needs to be implemented and what should be prepared ahead next to maintain this consistency. The milestone was separated into three general states, Lead In, Common Case and Lead out. For the common case, for each iteration, the idea is to write the results from the previous loop, compute the multiplication results from the previous read and do the read for the next iteration. This structure helps make the design more compact, since there is no conflict between the multiplication part and the read/write from SRAM. The first 3cc write the previous multiplication results, the mid 3cc assign the reading address, the last 3cc store the reading values, at the same time, the multiplier maintains working from 1 ~ 8cc.

3.2 Milestone 2

3.2.1 Hardware Structure and FSM

The purpose of this milestone is to inverse discrete cosine transform data in. All states in this module are divided into 4 stages which are fetching S'(FS'), calculating T(CT), calculating S(CS) and writing S(WS). The 2 main tasks include address generation and matrix multiplication. In the states related to FS' and WS, transferring data between RAMS and SRAMs depends on the address based on different col/row indexes and counters. For the 2 calculation states, the matrix read from the previous stages is used to multiply by C or C transpose whose data have been stored in a dual-port-RAM. FS' and CS' can be completed in one group of common cases, and the other 2 stages are allowed to be implemented in another group of common cases, which can operate the whole common case in two stages. The lead in includes fetching the data of the first block and calculating the first 64 Ts, and the lead out includes calculating the values of S in the last block and writing them into SRAM.

For the common cases, in stage FS', the 16-bit data (S') keeps being fetched from SRAM and written into a dual-port-RAM as two in one address until the top 32 addresses of the RAM are occupied. Then the S's and Cs are extracted from 2 RAMs separately in the

stage CT in order to calculate T with 3 multipliers. Aftering using adders to sum all products of S's and C, a shift register is needed to implement the "/256" which is equivalent to shift right 3 bits. Every time a T is generated, it is stored into the third dual-port-RAM. The two stages can be done in the same common cases and both take advantage of 64 CCs.

As soon as all T values are stored, CS is ready to start in the next state. Using 3 multipliers to multiply C transpose and T, and then add them up with adders. Since the result of the summary before has to be divided by 65536, a shift register is also needed here to transfer the data 16 bit right. After all calculations and shifts are done, all S are transferred into the RAM which stores S' before. And they are stored as one in each position in the bottom 64 addresses of the RAM. While all of the 32-bit Ss are obtained, the CS stage is finished and the M2_state goes to the WS stage. In this stage, S is fetched from the RAM and written into the SRAM.

3.2.2 Efficiency

Since the limited number of multipliers and dual-port-RAMS allowed to be used are both 3. In order to observe the maximum utilization, three multipliers are used in 2 clocks and two multipliers are used in the next clock cycle. The total number of clock cycles with multiples used is 6 in each common case for calculating T and S, the utilization is 6/7 equals to 85.7% which satisfies the requirement for this milestone. It takes almost 55779375 ns to complete the test with the cat.sram_d1 when simulating in Modelsim. It goes through 2920840 clock cycles. CC_FS'_CS and CC_CT_WS run 64 times when they operate one block and they work for 2400 blocks.

3.2.3 Implementation Comparison and Bugs Fixed

As the way mentioned to calculate T below, we tried to store all data including S', S, C transpose and T into 2 dual-port-RAMS because the total storage they need is smaller than 256*16-bit. However, it is not convenient to operate a few types of data in the same clock cycle. For example, in the CC_FS'_CS stage, we use both ports of RAM2(storing T) and RAM1(storing C transpose) to obtain two values of T and C transpose in one clock cycle. And we also need to store S into one RAM, which needs the write_enable of this RAM to be high. It is impossible to write any data into the RAM when the both ports of this RAM are reading other data. In conclusion, three dual-port-RAMs are necessary if we want the implementation completed in the shortest time. So we keep the 3 RAMs for storage in our implementation.

The most challenging bug is the incorrect assigning of the writing address for U/V segment. The written data always mismatches when the address reaches the first block of the second in U(SRAM_address: 39040). It confuses us because we misunderstood the row offset of the writing U/V segment is 160 which is 80 actually. And the result we

calculated by hand was very close to the result generated by code because the S' keeps the same values for a few addresses, so we kept thinking the error happened in the fourth row of the first block which was likely to be caused by the calculation or clipping. By the end, we found the correct row number and fixed this bug.

3.2.4 Special Idea

For the CT calculation in this milestone, S' and C need to be multiplied with each other. Since S' is the matrix in front of C, therefore the result is obtained based on multiplying the rows of S' and the columns of C. However the way to store matrices in dual-port RAMS is row by row. The critical point is that the columns of C are the rows of C transpose, which is the property of matrix. It is easy to calculate T by multiplying the rows of S' and C transpose because the addresses for them are the same in different RAMs. The advantage of this implementation is the same value can be assigned for the addresses of both S' and C transpose, it reduces the number of states and CCs necessary, which saves the space occupied for hardware. And C matrix will not be used in this module because the values in C transpose are input to the RAM by hand.

3.2.5 Milestone 2 Working Problem

For the test of milestone 2, our code passed all tests of the panda and cat and achieved "NO MISMATCH". However, for the motorcycle test, it failed in the one specific block in specific rows. The values calculated by hand and generated by code are a little bit different from the expected result. It seems to be caused by the rounding error in the hex calculation by the system.

3.3 Register Table and Critical Path

3.3.1 Register Table

Since we use a large number of registers like buffers or counters to store data temporarily and control the state changing separately. The critical registers are selected to clarify their functionalities in the code.

Module (instance)	Register name	Bits	Description
Milestone 1	U_buffer V_buffer	8	Used to store the U/V data which is not used in this common case. The stored data will be U_plus5 in the next common case

Milestone 1	counter_Y counter_U counter_V counter_RGB;	18	Used to record the address respect to the offset of each type of data, and used as the flag to control the state changing
Milestone 2	ram_counter	9	The register has 9 bits, every 3 bits can be used as the index of different matrices when operating matrix multiplications
Milestone 2	write_s_prime_ready write_S_end write_S_prime_ready Stop sign_a	1	These registers are used as the flag to indicate the different operations to do or the next states to go.
Milestone 2	col_block_counter col_block_counter_w row_block_counter, row_block_counter_w	9	Record the index of col/row or the number of blocks. Also control the states changing by comparing with the fixed values

3.3.2 Critical Path

From node: milestone2:M2_unit|op3[0]. To node: milestone2:M2_unit|T[31]. Times: 15.369us.

Register op3 is used as the factor to calculate T and S. Since the size of T is 32 bits which is one of the largest sizes in the milestone, the op has 16 bits. The difference between the bits of T[31] and op3[0] is 48 bits. The op3 goes through the multiplier, adder and shift register to connect to T. And the register of T never connects with the register of op3 directly. That's why they have the critical path.

3.4 Timeline

Week	Xinyu Chen	Ruiyan Guo

1	Reading the project description and discussion the contents	Read project materials and get the picture of how these milestone work
2	Listening to the lectures and creating the state table. Going to office hours asking state table problems.	Start initiating the state table, keep modifying after consulting TAs and Professor.
3	Correcting the state table and starting coding Milestone 1.	Finalize the state table version. Initiate coding for M1
4	Listening to lectures about Milestone 2. Coding and debugging Milestone 2.	Continue coding on M1 and discuss some of the M2 contents, thoughts. Start debugging M1 with help from office hours.
5	Debugging and Verifying Milestone 2. Changing some errors about the address generation, finally make it pass the cat and panda testbench	Continuing debugging M1. Fully verified M1 and participated in some of M2 debugging.

4. Conclusion

The project provides the amazing chance to take advantage of the knowledge learned from the course. It is very challenging and we didn't finish all of the milestones. However, it helps us study more about verilog which is completely different from the software language we learned before. For example, We learned the significance of the state table to implement common cases when we start coding.

During the past few weeks, we went to office hours regularly to ask professor or TAs questions. In the second week, we asked the professor about the state table question and got help to correct the state table. The clear state table saved us a lot of time and made the state pattern organized better in Milestone 1. And we asked the TA Karim who taught us to add different wave.do files into the sim folder, which helped a lot when we were debugging.

5. Reference

Nicola Nicolici 2021, Mcmaster University, accessed 11-26 November 2021, <https://avenue.cllmcmaster.ca/d2l/le/content/414166/Home>