COMP ENG 3DQ5 - PROJECT REPORT

Group 49
Ebrahim Simmons - simmoe1 - 400200042
Bilal Yusuf - yusufb1 - 400185626
Monday, November 29th, 2021

1. Introduction

Within this project, we implement an image decoding circuitry that decompresses and recovers a 320x240 pixel image compressed using .mic15 image compression specification. The image is delivered to the Altera DE2-115 board via the UART interface and is stored in the external SRAM. Afterwards, using the implemented hardware it is decoded. The decompressed image is then stored back in the SRAM, where the VGA controller reads, and displays said image.

2. Design Structure

Our design structure was divided into different modules for each separate milestone. The majority of the modules are not different from what we were given. The two exceptions being the UART SRAM interface module and the top level module. We changed the UART_SRAM interface to remove two states that parsed the PPM file's header. In addition, our top level module (project) was changed in order to add our milestones.

There were some modules that we reused:

- **project**
    - This is the top level module in which we combine all the sections of our project. It contains a finite state machine that figures out which module is interacting with the SRAM.
- **VGA_SRAM_interface**
    - This module is needed to display the recovered image from the SRAM onto a monitor using VGA. This is controlled through the finite state machine in the top level module
- **UART_SRAM_interface**
    - This module is required to send the compressed data to the SRAM. Like the previous module, this is controlled through the finite state machine in the top level module

- **SRAM_controller**
    - This module is required to interact with the SRAM, controlled by the top level module
- **dual_port_RAM**
    - There are 3 instances of dual_port_RAM that stores the data for computations that are done in Milestone 2. This is received through the state machine done in milestone 2.

3. Implementation Details

Milestone 1

Within this milestone, we have an always_ff block that contains our state machine. Most of our module is contained within this always_ff block.

We use six 32 bit registers for the multiplier operands (oper1, oper2, oper3, oper4) and the final results between them (multiplier1, multiplier2). We use 64 bit registers for the multiplication of the 2 multipliers. We used shift register structures to hold Y, U, and V values that were read from the SRAM. Twelve 8 bit registers were used for both U and V. In addition, two 16 bit registers were used as a buffer for U and V shift registers.

In order to implement accumulators we have two 32-bit registers for both the colourspace conversion and interpolation. We used six 32 bit registers to hold the values as entering colourspace conversion, the even and odd values of Y, U, and V. We also used six 32 bit registers to hold the final RGB values after the completion of the colourspace conversion. These were clipped and held in three 16 bit registers, for writing the rgb values.

For the SRAM addressing, we used three counters, for U/V reading, Y, reading, and RGB writing. The first two are 16 bits, while the last is 18 bits. The 16 bit counters were used for counting the spot in a row, and handles row border cases. We created two flags, one which determined when to read the new U/V values (denoted by **uvREAD**) and the other to sync shift registers in our common case (denoted by **sync**).

Using our constraint of 2 multipliers, we used both multipliers for 8 out of 9 clock cycles giving us a utilization rate of **88.9%**, higher than the requirement of 85%.

Our lead in states make sure that the calculation does not start until the previous step was completed before. Most of the time in the milestone is in the common case where reading, computation, and writing take place for 9 clock cycles. A link to our state table,

that shows our thought process throughout the milestone can be found at: https://docs.google.com/spreadsheets/d/1GEI-BlTxYQsEb-Y74IHul-EpEZ-QgMS9v6pJ1GXGr-g/edit?usp=sharing

Our final implementation of this milestone is the first structure we attempted. We could not find any flaws with the way we went about this, so we never attempted to try it in a different way. Our troubleshooting for this milestone was very difficult. Meeting with Karim and Trevor helped us understand how to troubleshoot effectively. We mainly troubleshooted by finding the pixel with the error in ModelSim and performing calculations to see where we went wrong. A lot of the problems were caused by syntax errors and errors with our common cases.

In this project there are 76800 pixels, processed in 38400 common state iterations. Through our simulation, we see that the milestone 1 state lasts for **6912.6us**. We know that there are 76800 pixels that are processed in 38400 iterations. This means that our milestone 1 lasts for (1 leadout state) + (27 lead in states) + (38400*9 common cases) + (3 clock cycles in top level module). This nets us a result of **345631** clock cycles, which is **6912.6us**.

Milestone 2

In this milestone, we have an always_ff block that handles our lead in states.

We use nine bit registers for the multiplier operands (oper1, oper2, oper3, oper4, oper5, oper6) and the final results between them (multiplier1, multiplier2, multiplier3). We use 64 bit registers for the multiplication of the 3 multipliers.We have a flag introduced that could be used to determine which region the addresses are in.

We were not able to finish milestone 2, therefore we did not attempt to troubleshoot as we did in milestone 1.

Milestone 3

We were unable to begin this milestone.

Contributions & Timeline

In the beginning of the project, we failed to organize our time effectively. We fell behind, not having a significant amount of progress until late in the project phase. Additionally,

we should have utilized the TAs help earlier, instead of trying to figure out issues solely by ourselves.

Both group members worked on the project together. All implementation and brainstorming was done together during our weekly meetings. When writing code, we would both work on separate machines, and screen share, as we progressed throughout the project.

Week #	Group Member	Contribution
Week 1	Ebrahim Simmons	Read the project document
	Bilal Yusuf	
Week 2	Ebrahim Simmons	Continued to read the project document. Began state table brainstorming.
	Bilal Yusuf	
Week 3	Ebrahim Simmons	Worked on and finalizing the state table
	Bilal Yusuf	
Week 4	Ebrahim Simmons	Began milestone 1. Troubleshooted milestone 1.
	Bilal Yusuf	
Week 5	Ebrahim Simmons	Finished Milestone 1. Began milestone 2. Completed part of milestone 2. Wrote the project report.
	Bilal Yusuf	

4. Conclusion

After finishing this project, we have both gained invaluable knowledge and experience that will aid us in our coming years in other university courses and job environments. The most important takeaway for us was how important it is to have a thought process setup and a plan laid out before any coding can begin. Having a state table helped us greatly with our common cases for milestone 1, knowing how many common cases we should have and what each common case should accomplish helped us code efficiently. Furthermore, when it came to troubleshooting it was a different experience than other courses. In other projects or labs, troubleshooting is relatively straightforward as the problem can be seen right away. Using ModelSim and hex viewers to find the problems by looking at each value and using those values to understand what is wrong in your code was an important skill to learn. In the real-world environment, when debugging the bug will not always be outlined for you. Knowing how to use different software and

programs to debug code is an important skill, even if the debug process is a lengthy one. The working code for milestone 1 was pushed at 4:35pm on Nov 29th (Commit: 0ec9c46aaeb541a349f5f72698d2dcb6ddecdba6). The unfinished code for milestone 2 was pushed at 10:12pm on Nov. 29th.

5. References

Throughout this project the only resources we consulted were the project documentation, lecture notes, and the professors and TAs.