3DQ5 Project Report
Milestone I & Milestone II Synthesis and Analysis - Group 19
John Lui 400010430 luij1
Sahil Patel 400076790 pates22
November 26th, 2018

1. Introduction

The goal of this project is to create a custom digital hardware implementation of an image decompressor on the Altera Cyclone II FPGA. The decompression process is split into 3 sections: Colorspace Conversion and Interpolation (Milestone 1) focuses on turning brightness and tints into RGB values for the final image. Dequantization and Inverse Signal Transform (Milestone 2), worries about obtaining the brightness and tint values from the lossless decoding of the bitstream data of the image and converts it to YUV pre IDCT values. Finally Lossless Decoding (Milestone 3) takes the, meaning we tackle the decompression algorithm in reverse order. Data will be stored and transmitted using the universal asynchronous receiver/transmitter (UART) interface and the stored data will be located in the static and dynamic RAM modules. And finally the decompressed image circuitry will store the image in the SRAM where the VGA interface will operate to display the image onto the monitor.

2. Design Structure

The design is connected through a top level finite state machine which handles all the individual units and modules associated with the project. This top level file (named “project.v”) is a child file from the parent file provided in experiment 4a from laboratory 5. This is because we wanted the universal asynchronous receiver/transmitter (UART) and the VGA interface modules already predefined and set up in the experiment. This foundation helped in developing the milestones outright instead of having to setup nitty gritty stuff that did not correspond with the project content. The following modules were found in the main project to level file as they would be accessed by other units at different times:

SRAM - The static random access memory was important to place in the top level file as it was utilized in nearly every other unit. The UART and VGA components used it when we received them from experiment files. Our Milestone 1 and 2 needed to manipulate many values found in the SRAM as well as read them.

DRAM - Although milestone 2 is the only unit that utilized this module, we still decided to place it in the top level file as it is good practice. In the future if we decide to implement milestone 3 or other implementations that would require the DRAM this modular approach would allow us to do so easily. There is also the point that having the DRAM separate from the M2 module allows for easier debugging as all DRAM bugs get ironed out into a different file.

Milestone 1/2 - These modules conduct their respective stage of decompression. They are mutually exclusive and are controlled by the project top level fsm making it easy to manipulate and order them. Had we placed the entire decompression fsm into the project document, debugging would become very difficult, but placing them in a modular format like this allows for an easier more omnichannel approach to debugging.

Other minor modules included the push button unit and the seven segment display unit. These were small insignificant units that came with the experiment file.

3. Implementation Details Milestone 1

Milestone 1 was first initiated by creating a table that outlined the multipliers' operands at a given state/clock tick. We experimented a lot and were very tempted to implement a design which involved 2 multipliers instead of 3 (saving resources) as well as gave a **100%** utilization (figure 1), however we decided not to finalize as the design would be slow and perhaps not allowed. After brainstorming furthur we settled on the design shown in figure 2 as it made the most sense to calculate RGB even, then RGB odd in that order. We easily met the 80% utilization requirement as we covered 16/18 multipliers per cycle meaning we achieved just below 89%.

M1	e	e	e	u	v	v	o	o
M2	e	e	u	u	v	o	o	o

figure 1

	1	2	3	4	5	6
M1	e	e	u	v	o	o
M2	e	e	u	v	o	o
M3	e		u	v	o	
SRAM call	W: BeRo	W: GoBo	R: Y	R: U/V	W: ReGe	
					R = read, W = write	

figure 2

To implement the design we coded a circuit very similar to that found in figure 20 in the project document. Since the U' and V' odd values were calculated using a array of U and V values, due to the symmetric and function nature, we have shift register which keep track of 6 U and V values. The it is necessary we use shift register because preceding odd U' and V' values utilize 5 of the 6 indices used in the previous one.

We next allocated buffers and accumulators that would be necessary as read values and even write values would spill over. RGB accumulators were made thus that over stage 1 and 2, and 5 and 6, the RGB value would retain its value until all the elements of the matrix were multiplied and added. Buffers for other minor reasons such as one for B odd for when we write Bo Re and etc. were planned ahead of time and made to be used in the circuit.

We knew that over the course of 2 cycles, we would need to call U and V once, and Y twice as the shift register would hold U and V values for both odd and even calculation. We played around with multiple variations and settled on one where we would read a Y value in stage 3 and either U or V in stage 4. All the values arrived and got updated on time to calculate RGB values over and over again.

After constructing the common case for the U cycle, we copied the states and pasted them with minor changes to make the common case for the V cycle. The last state of each of the cycles would lead to the beginning of the other effectively making the loop we need. We considered utilizing a flag that would flip every cycle to control muxes over every difference between the common cases however it proved to be very tedious thus we just stuck to more states.

To lead into the common case, we recognize that all we needed to do was load the shift registers with values while not operating on them, so we did just that. We called two U and V addresses (only two because the 0 index value takes up multiple places in the register to cover for negative indexes), a Y value and that lead perfectly into our common case. For the lead out, we knew that the final call would be U 158,159 thus the lead out would be very similar to the V cycle repeating over and over again. When the cycle repeats, it would append index 159 for both registers instead of calling new values to cover for indexes out of range.

Finally since the operands were concrete per cycle we mage made a huge mux (similar to the one in figure 20 of the project document), which would select which operands the 3 multipliers would have. This was done in combinational logic allowing multiplier values to be available very quickly.

4. Implementation Details Milestone 2

The first step in developing the milestone 2 system is to start developing the state table, this is initiated by first mapping out what values are needed at which time. For example we know that the very first step is to fetch S' values from the external SRAM. This is the first part of the Lead in state, the next step of the lead in is to transition to the first Compute T state. This transition is needed because there will be a writing latency in terms of the values being registered in the DRAM, and also we will be able to take advantage of this buffer state to pre read values of S prime to perform T calculations. Within the T calculation, use a compute T counter to know how many calculations it needs to be done, and within the state we also have logic to control when to write computed values into the DRAM.

More specifically our memory maps for the DRAMs are described as such, DRAM_0 contains S', and S values. The S' values are mapped as a 32 bit value which will take 1 full address. The advantage to this is when the DRAM is later referenced the addressing can be simply incremented by one, and no further manipulation will be needed. The S values will be clipped after initial entry the DRAM, therefore there the S values will be 16 bits long, and will take up the lower half of the DRAM. DRAM_1 contains just the C values needed for the system, and is stored alone because C values require the most bandwidth amongst all variables used. Therefore it is imperative that this value does not need to be read/written with any other variables involved with this system. Lastly, the T values are stored in DRAM_2 alone due to the flexibility in reading and written as well as the DRAM_2 being available to use for the convenience of this milestone.

Some of the more important design decisions include how each state is tracked, and transitioned. For example, in our design we focused on heavily using counters and comparators in order to know how many S values to fetch we used a FetchS Counter which counts to up to 64. This allows us to keep track of how many S' values have been read. This piece of hardware can be described as a mux which is driven by a comparator which compares the current value held in the Fetch_S Register. The Mux is set to a reset/zero if the driving signal is a 1 and increments by one if the driving signal is a 0. The present value of the register is then sent to the FSM for controlling different events. This sample circuit is just an example of how comparators, muxes and registers have been used in within our custom designs. This is useful because some counters (like the SRAM_Address Counter) cannot be used in certain states, and this allows us to track different states with an intuitive debugging tool as we know at a specific interval an event is supposed happen. Therefore the extensive use of counters and flags allow us to control the transitions precisely.

Another design decision we considered was splitting up T values between two DRAMs, this was an intriguing solution to the S computation as this allowed us to pull 4 T values given a clock cycle interval. However this implementation was not feasible due to the need to also pull 4 C Constant Values from another DRAM. This approach heavily relied on availability of bandwidth in order to provide the variables needed to compute certain values. This is a huge problem for this milestone as the DRAM allow us to buffer large quantities of values that will be needed due to the nature of the Matrix Calculations involved with the Inverse Cosine Transform Function.

The last major design decision/choice we made for this milestone is the way we implement the flow of the states. For example, from the recommended outline of layering the continuous sequence of Fetch S, Compute T, Compute S, and Write S. This can be implemented in many ways, the way we decided to implement this was to create a Lead In Phase, Two Mega States (which switch during the majority of the milestone), and the Lead Out phase which implements the structure shown in class. The following image will illustrate this milestone structure.

Lead In Phase	Lead In Phase	Mega State 1	Mega State 2	Lead Out Phase	Lead Out Phase
Fetch S	Compute T	Compute S	Write S		
		Fetch S	Compute T	Compute S	Write S

It is critical that there are transition states in between that facilitates the pre loading of values throughout the operation of this system. We decided to pursue this implementation as it separates the system into six distinct states that performs different tasks. This helps to keep our ideas/thoughts organized throughout the development process. When compared to other implementations of similar designs there is no inherent advantage/disadvantage to this design. However, one argument can be made that the number of transition states between every state above produces a lower utilization rate. For example a sample calculation for the utilization rate of out system is below:

153, 600 / 157, 304 = .97645 = 97.645 *% Theoretical Multiplication Utilization Rates*

The 128 clock cycle block is representing the 30 by 40 blocks of continuous multiplication blocks, and the utilization rate for this block is very dominant throughout the system computations. This affords us the ability to create many transition states which allow us to read values or write values in a timely manner.

This milestone was very challenging as it presented itself as a problem of system bandwidth, as oppose to milestone 1. Where milestone 1 is a problem of timing, and managing when specific data points are pulled, computed and written. The amount of care needed to manage the three DRAM's was very challenging, and tested our understanding, creativity and problem solving.

Milestone Timeline (weeks):

1. Understanding Project Documents
2. Milestone 1 Early Design
3. Milestone 1 Debug
4. Milestone 1 Debug / Milestone 2 Early Design
5. Milestone 2 Debug

Partner Contributions

Milestone 1:
State Table/Early Design: 60% Sahil, 40% John
Code Implementation: 60% Sahil, 40% John
Debug/Implementation: 60% Sahil, 40% John

Milestone 2:
State Table/Early Design: 60% John, 40% Sahil
Code Implementation: 70% John, 30% Sahil
Debug/Implementation: 50% Sahil, 50% John

5. Conclusion

This project really tested our understanding and creativity within designing digital circuits/hardware implementation for a large comprehensive system. One of the things we struggled with throughout the project was keeping our thoughts and ideas clear over long stretches of time, as well as between partners. It really showed the importance to be able to effectively communicate ideas throughout team members in order to develop solutions in an efficient timeline. Likewise another big takeaway from this project is the importance of initial documentation. Many issues faced during the debugging stage, could have been mitigated by proper documentation and organization in the early stages of design. Nevertheless classroom collaboration was very prevalent throughout the design process. For example, discussions held about potential design implementations and resource management was done with Shaun Feere, Rey Pastolero, and Pierre Tadrus. These discussions opened up many interesting pathways to take the solution, and greatly diversified our thinking throughout the design process. Likewise TA's (Alex, Pooyan and Trevor)s during lab sessions were used when faced with confusing bugs/errors. This is greatly helpful as Quartus Errors are not often descriptive/accurate and a TA explanation of such errors were extremely helpful throughout the synthesis and debugging stages. Overall the project presented a great opportunity to test our understanding of digital design and afforded us the opportunity to learn through mistakes to become better hardware developers.

References

Individuals:

- **Peers**: Rey Pastolero, Shaun Feere, Pierre Tadrus
    - Simple quick easy to solve error questions
- **Teaching Assistants**: Alex Lao, Pooyan Mehrvazy, Trevor Pogue
    - Design verification
    - Proof of design
    - Finite state diagram confirmation
- **Professor**: Dr. Nicolichi
    - Invaluable insight and perspective on project
    - Providing us with a picture of our group for uploading

Websites:

- https://www.binaryhexconverter.com/hex-to-decimal-converter
- https://stackoverflow.com/
- https://www.google.ca/drive/ (Sheets)
- http://www.ece.mcmaster.ca/~nicola/3dq5/2018/labs_project.html

Texts:

- COE3DQ5 Project Description 2018 Hardware Implementation of an Image Decompressor

Software:

- http://www.ece.mcmaster.ca/~nicola/3dq5/2018/coe3dq5_lab5_2018.zip
- http://www.ece.mcmaster.ca/~nicola/3dq5/2018/dual_port_RAM0.v