COMP ENG 3DQ5 Project
Joshua Umansky (400234265), George Wahba (400371904)
Group 73
November 27, 2023

Introduction

Through the completion of the project, the objective was to gain an experience with digital system design by implementing a custom image compression. This image compression is specifically completed in hardware for a 320x240 image through the FPGA Altera DE2 board which is delivered by a UART interface with a computer, and then stored in an external SRAM. The goal of the project is to read the compressed data in the SRAM and display the uncompressed image through the VGA controller onto a monitor.

Upsampling and Colour Space Conversion

When organising the calculations for interpolation and colour space conversion, the choice was made to calculate pairs of pairs of pixels, therefore generating four pixels per common case. This was completed in eleven states per common case, and followed the structure shown in Figure 1.1. Using this methodology, the calculations were split according to Figure 1.2, making use of all four given multipliers on any cycle; the calculations for the individual pixels follow the convention shown.

Common Case n-1	Common Case n	Common Case n+1
Read k	Read k+1	Read k+2
Compute k-1	Compute k	Compute k+1
Write k-2	Write k-1	Write k

Figure 1.1

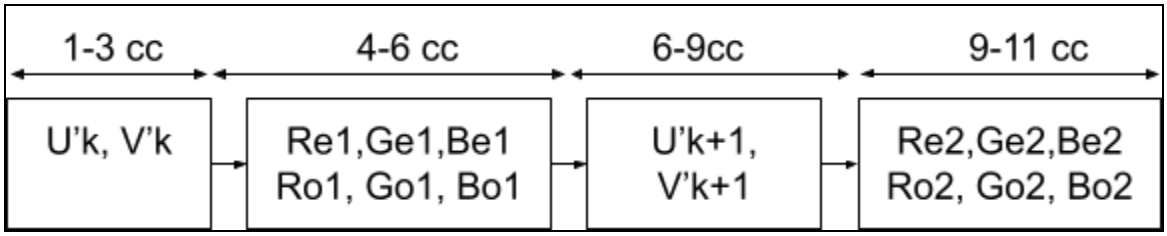

Figure 1.2

Milestone 1 - Register Description

Register Name	Bits	Description
U/V Shift Registers	12x8	Used in a shift register structure to store the consecutive values of U and V from the SRAM
Y, U, V, RGB_address	4x18	Store the individual addresses of the locations in the SRAM for the different variables
U, V, Y, RGB_offset	3x8, 18	Store the offset values for interfacing through the SRAM locations, and drive MUX logic
Ubuffer(1,2) Vbuffer(1,2)	4x8	Store the values of U, V that are read, then used in the next common case

Yeven(1,2),Yodd(1,2), Uneven(1,2), Veven(1,2)	8x32	Store the values of Y, U, V used in the RGB calculations
Acc_u, Acc_v	2x32	Accumulators for U' and V', used in the RGB calculations
Multi_op, Multi_a	8x32, 4x64	Multiplier registers used in the assign statements for multi1-multi4
Re(1,2), Ge(1,2), Be(1,2), Ro(1,2), Go(1,2), Bo(1,2)	12x32	Store the calculated values of RGB, which are scaled and written to the RGB locations in SRAM

Latency Analysis

Our design implemented the multipliers in a manner that maximised the usage of the multipliers on any given clock cycle. Therefore, we were able to fully utilise the multipliers on any given cycle that data was available to be calculated. This resulted in nine states in any given row that did not utilise the multipliers. The latency analysis can be modelled as below;

Row n: 16 (Lead-In States) + 79*11 (Common Case) + 6 (Lead-Out) = 891 clock cycles

Entire Image: 891 * 240 = 213,840 clock cycles

Therefore, as can be seen, the entire module takes 213,840 clock cycles for the process to complete, and this then results in the following multiplier usage;

Row n: 10 (Lead-In States) + 11*79 (Common Case) + 1 (Lead-Out) = 880 clock cycles

Entire Image: 880 * 240 = 211,200 clock cycles

Using these calculations, the resultant multiplier usage is 98.76%.

Inverse Discrete Cosine Transform

When designing the conceptual ERAM layout, we decided on having the setup as shown in Figure 2.1. This results in the ability to obtain four different T values per clock cycle, hence allowing the calculation of four individual S values. We followed this approach to reduce code complexity, and have an end result that was modular between the Compute T, and Compute S portions of the calculations. Our C matrix is stored in a series of LUT's and is accessed using two control variables (row, column) that are controlled by the FPGA.

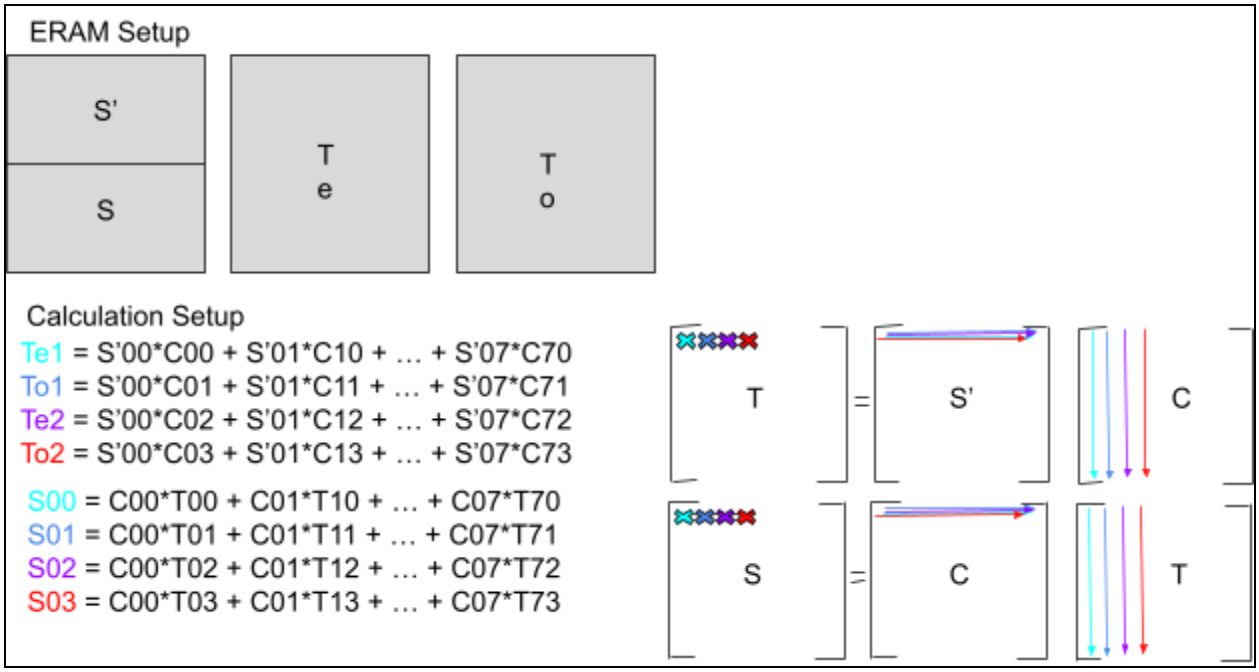

Figure 2.1

Milestone 2 - Register Description

Register Name	Bits	Description
FetchS_Address, FetchS_Offset, RAMS_Offest	2x18, 1x8	Stores the address of S', stores the offset value of S' going across the row, stores the offset of S' with respect to the RAM since we store 2 S' in the same "block"
RAM_S_Address1, RAM_S_Address2, RAM_TRAM_E_Address1, RAM_TRAM_E_Address2, RAM_TRAM_O_Address1, RAM_TRAM_O_Address2	6x7	The address of pairs of calculated S values, the addresses of pairs of even and odd calculated T values
Read_TRAM_E1, Read_TRAM_E2, Read_TRAM_O1, Read_TRAM_O2, Read_S_1, Read_S_2	6x32	The values of the calculated T values in both the even and odd external RAMs which are going to be read, and the 2 S values from the RAM that are read
Write_TRAM_E1, Write_TRAM_E2, Write_TRAM_O1, Write_TRAM_O2, Write_S_1, Write_S_2	6x32	The values of the calculated T values in both the even and odd external RAMs which are going to be written to the corresponding address, and the 2 S values from the RAM that are going to be written to the corresponding address
Data_counter, Column_counter	2x8	Data counter counts the amount of data points that we have pulled from the SRAM, and the column counter allows us to know when to jump to the next row in the SRAM, for example, going from position 0,7 to 320,0
Sprime_buffer, S_buffer	2x17	These contain the buffered value for both S prime and the calculated S value
Acc_t1, Acc_t2, Acc_t3, Acc_t4	4x32	Four actuators which are used to store the value of the calculated T and S values
Multi_op, Multi_a	8x32, 4x64	Multiplier registers used in the assign statements for multi1-multi4
C_Matrix	1x6	This is the value that contains where the iteration is within the transposed matrix from 0-63

Resource Usage and Critical Path

When compiling the Milestone 1 version of our design, the logic elements required by Quartus is 3486. When relating this to the base of our project (Lab 5 Experiment 4) which has a logic element usage of 616, we have quite a large increase in logic element usage, but still falls well below the maximum logic elements of the FPGA (114480).

As we progressed through the project, in order to reduce complexity in our code, we attempted to reduce the amount of registers that were used as much as possible, an example was originally two intermediate registers were used for U' and V', which were eliminated as they were not needed, and all mathematical operations could be completed using bit shifted versions of the accumulator registers (acc_u, acc_v).

Through inspecting the critical path in Quartus, the path between the multiplier operator, and acc_v is shown to have the highest delay (17.370ns). This is a result of the data having to be passed through the multiplier circuitry, and then having to be added to the accumulator (acc_v) through an addition circuit, eventually being stored into the accumulator register.

Weekly Activity and Progress

Week	Project Progress	Contributions
1	Read the overview of the project	Both group members completed reviewing the project document
2	Began the creation of Milestone 1 state table	Josh completed this task*
3	Finalised the Milestone 1 state table, began the programming of Milestone1.sv	Josh finalised the state table Both group members completed the programming of Milestone 1.
4	Began and finalised Milestone 1 debugging, Milestone 2 state table started	Both group members worked together on the debugging of Milestone 1, and the planning Milestone 2
5	Milestone 2 programming and debugging	Josh completed the fetch S' states George completed the Compute T, and Compute S states Both group members worked to debug Milestone 2

**Collaboration Note: Original conceptual design of an 11 state common case was completed with Milan Joaquin through the use of a whiteboard in the lab*

Conclusion

Through the development process of Milestone 1 and 2, a deeper understanding of SRAM and ERAM addressing and data storage was obtained, as well as the process of image compression and decompression. While a deep understanding of Verilog programming was obtained, this project was an imperative learning opportunity in terms of the engineering design process in regards to project planning and time management.

Milestone 1 Completion Version (Name, Date): "Working Milestone 1 (Nov 23)", November 23, 2023