3DQ5 Digital Systems

Group 19
Project Report

Hardware Image Decompressor

Authors:
Luke Pritchard

Student Info
pritchlj@mcmaster.ca
001311288

Robert Ballantyne

ballanra@mcmaster.ca
001412337

November 27, 2017

Introduction

At the surface level, the project is about creating a hardware implementation of an image decompressor using Verilog. This project represents the culmination of a semester's worth of experiential learning. We draw on the knowledge from all the previous labs and lectures to interface the project in a hierarchical structure in the form of three individual project milestones.

Design Structure

In the current state of our project, we have only completed the first milestone. So the modules that we created ourselves are limited to the module needed for milestone one that interfaces with the top level FSM file. We reused the following modules from Lab 5 Experiment 4: `PB_Controller, VGA_SRAM_interface, UART_SRAM_interface, SRAM_Controller,` and our own, `Milestone1`. Each of these was instantiated as a unit in the top level FSM. We can hypothesize that the proceeding milestones would also both need their own units that would incorporate dual port RAMs along with any other signals required.

We encountered great difficulty in creating the interface for milestone 1 so that we could connect it to the top level FSM. This came from a lack of understanding of the naming conventions and syntax with regards to the port declaration and what is passed to a module and defining the specific input and outputs for the top level file, and the corresponding milestone 1.

```
milestone1 M1_unit(

  .CLOCK_50_I(CLOCK_50_I),
  .Resetn(resetn),
  .SRAM_ADDRESS_m1(SRAM_ADDRESS_m1),
  .SRAM_we_n_m1(SRAM_we_n_m1),
  .SRAM_write_data_m1(SRAM_write_data_m1),
  .M1_START(M1_START),
  .M1_END(M1_END),
  .SRAM_read_data(SRAM_read_data)
);
```

Listing 1: Milestone 1 unit defined

There was little that needed to be added for milestone 1 as the work was focused entirely on SRAM manipulation so the only completely new signals that were needed were: `M1_START, M1_END, SRAM_read_data_m1, SRAM_address_m1, SRAM_we_n_m1, SRAM_write_data_m1` and the milestone would then proceed. The states that were contained in milestone 1 were placed and referenced in the header file `define_state.h`. Thus the top level FSM can now interface with the milestone when the top level sends the start signal and stops when it receives the end signal.

An alternative approach that was considered but not used in the final outcome was to simply include all of the code used in Milestone 1 in the top level file. While this would have removed the need for communication between files, at least in regards to the top-level

files and any Milestone code, this choice was ultimately revoked, as it would have made debugging far more challenging if future Milestones were to be added, as errors would be more difficult to locate.

Implementation Details

Milestone 1

As noted above the top level FSM passes control to the Milestone 1 FSM when the start signal is sent to the module. The FSM for Milestone 1 has 28 states, 12 for the common case where 4 pixels are processed. The lead-in is 10 states and the lead out is 5. We decided on 12 cycles for our common case as we chose to calculate and write the new Red, Green, and Blue pixel values in pairs of pixels. Each pair of pixels would require 6 states to effectively calculate, however, 12 states were used as the total common case. This was done to remove the need for excessive flag registers, which would have made debugging far more difficult and tedious.

In the edge case where clipping can occur, such as when the RGB values are calculated to be above or below what we consider the zero or maximum threshold, the register that holds the output from the multipliers for computed RGB values is then sent to a combinational block to clip the unneeded values, since calculations are performed in 32 bit registers. This is done by using muxes that select the MSB of an RGB value and assigns a new register to 0 for a negative value and another mux to check the preceding MSB-1 to MSB-7 bits if they are high which would mean the value is above the range for pixels so the new register is assigned FF. Otherwise, the new register is assigned the value of the pixel register passed to this block. The purpose of creating a new register is to avoid latching the pixel register in the case where no clipping occurs.

Our utilization of the multipliers in our common case out of the 12 clock cycles we had, we use 3 multipliers in only 2 states which are: `S_COMMONCASE_4` and `S_COMMONCASE_9` we made this decision as we struggled to find how to write the RGB data without using some sort of flags to figure out whether we were writing R, G, or B in the transitions from the 6 cycle common case to the next. In our current implementation of 12 cycles we only have one flag `firstrun` that is active low for when the first common case run is occurring after the lead in transition. This helped minimize the overhead complexity of our design. The milestone ran for a total of 4.68 milliseconds. The utilization of our multipliers is 95.8% based on using 4 multipliers in 10 out of 12 clock cycles and using 3 out of 4 multipliers in 2 out of 12 clock cycles.

The lead-in and lead-out states focus on setting up and performing the unique calculations for the beginning and end of rows of pixels respectively. We made the effort to use as few states as possible, as to ensure that the simulation runs as quickly as possible.

Debugging

For the errors that we encountered in our debugging process. The initial problem, which has been mentioned previously was the interfacing of the milestone 1 with the top level

FSM. The hierarchical structure proved difficult for us to establish as we were not aware of the logic needed in staying consistent with the port naming conventions across files this caused us to consider putting more of these values into I-O at the top of the milestone 1 and Top level FSM. This caused a back and forth between two errors where the nettype was already defined and is defined in multiple places. It said defined in multiple places due to our attempt to define a Top level module in the M1 Verilog file as we thought they both needed an interface between the two when in actuality if we defined the M1 unit in the top level file and have IO within the M1 Verilog file was able to get it working. Now we were able to begin the debugging process for the constructed milestone 1. Next, we simulated our code in Modelsim and noticed immediate errors in the computations after only a few initial writes. The first real bug encountered was within the reading and shifting of the interpolation registers where we forgot to shift one of the registers from the 4th to the 3rd. So the pixel value didn't propagate into the next register which was a simple fix to an error that we overlooked.

The greatest difficulty that we had was an error that appeared after a few hundred writes, and then the error propagated for a total of 6 clock cycles. This burst of errors occurred every few hundred writes with the same offset between the error burst of 468. The 6 errors repeating led us to hypothesize that the problem lay in our interpolation registers as a wrong value was read in and then propagates an error for 6 clock cycles. The difficulty lay in determining how we were reading the values from the SRAM into our interpolation registers. We believe that may have been due to the edge cases when we transition in between rows and are values are off usually on the odd pixel and usually only by a value of 2 from the expected true value to be written. In ModelSim when we increased the mismatch number in the test bench it appeared that this was our only error left. We had to revamp our source code as we forgot to deal with leading out of each individual row where we originally treated the SRAM as a block, this caused errors in the interpolation of the ending row pixels. The `firstrun` flag needed to be changed from only being run once to running every single line. This is the large bug that we overlooked and once implemented our code ran without mismatches.

Another major bug that occurred was in the lead out states, proceeding to a new pixel row. Problems occurred with the addressing to the SRAM, as reading was occurring too quickly, and as a result, reading the incorrect locations. The solution to this required a number of fixes, mainly involving resetting registers to their default state, such as the firstrun register, and ensuring that the U,V, and Y addresses were not incrementing in the event of the ending of a pixel row.

Conclusion

With this project, we were able to partially construct the image decompressor. We learned generally the work that comes with building and maintaining a large computer hardware project. We built an intuition in learning to debug hardware-specific errors and how to divide and conquer aspects of the project within each milestone. We were assisted by Alex the TA who inputted code so that the Top level only simulated the milestone 1 code and not the UART, the code that he wrote was only a few lines and not fundamental to what the project is. A fellow colleague, Suhaib Chowdhury, explained the subtleties of clipping

and that the MSB indicates a negative to be rounded to be clipped to 0 and if any of the following bits were high but not signed clip it to FF. The nature of our collaboration was entirely conceptual and we implemented our code independently.

Week 1	We familiarized ourselves with all of the milestones and read the project requirements.
Week 2	Constructed our state table for milestone 1 and verified with a TA to explain our thinking and approach for the project. Robert began to construct the state table and Luke suggested to use 6 common case states.
Week 3	Wrote the initial code for milestone 1 and made adjustments to the state table. Robert made the suggestion to redo the state table using 12 clock cycles and Luke changed the common case. Robert wrote the skeleton code for milestone 1 in the common case and Luke wrote lead in and lead out.
Week 4	Struggled to interface the top level FSM with milestone 1. Luke was building the module interface and naming the ports, Robert tried to connect the two modules in different manners and reconfiguring the IO. Both efforts did not make much progress.
Week 5	Received assistance and interfaced milestone 1 then proceeded to debug the milestone. Robert debugged the largest bug and Luke debugged smaller bugs. Luke wrote most of the report while in with discussion Robert.