COE3DQ5 – Lab # 5 Report

Group # 33
Fahad Mahmood – 001414984 – mahmof4@mcmaster.ca
Wei Che Kao – 001328256 – kaow@mcmaster.ca

October 26, 2018

Exercise 1:

In the first section, we create a state to start writing data and check for even or odd location before going to write cycle. We then go to write cycle and do the same thing for 3FFFF times. And then we will jump to two delay cycles and decrement the BIST address twice to get ready for the read cycle since it's in the decreasing order. In read cycle we check for mismatch until BIST address decrement to 0 then we will go to delay 3 and 4. We will still check mismatch at the two delay cycles. Before we go to the second write data cycle, delay 4 cycle will start writing data for PAT3 and PAT2, also check for even odd and we will set the BIST address to 3FFFF. The implementation for section 2 is pretty much the same as section 1. The only difference is we start write in decreasing order and read in increasing order. We verified this using ModelSim.

Exercise 2:

Firstly, we created states in the define_state.h, which are same as the states in the lab. Then, we assigned parameter R segment, even G segment, odd G segment and B segment according to the SRAM memory map. We implemented the incrementation of SRAM_addresses in the states in the always comb block provided in the source code depending on the color according to the SRAM memory map. SRAM_address_o for even and odd G incremented every other cycle whereas, for R and B, it incremented in every cycle. In the sequential block, in the delay states we stored R0, R1, G0 and B1. We did not have to buffer G1 or B0 because we were not fetching G1 at the moment and we were directly fetching B0 from the SRAM. To implement looping fetching logic, we used four fetching states. In first state we implemented same logic as delay states to store B1 and then we fetched RGB, in second state we had to check if SRAM_address was even or odd to buffer G accordingly, in third state we buffered R and fetched RGB, in last state we incremented SRAM address and checked SRAM_address for even or odd to buffer G accordingly. These fetching states kept looping until we reached last pixel position where we stopped. Lastly, we used ModelSim and displayed our code and implementation on monitor to verify.