Lab 4 Report - Exercise 2

Frank Shan
Mark Naguib

For session 1 of exercise 2, we used the same code that we made in experiment 4 with a few simple changes. In experiment 4, we went all the way up to 256K or 3FFFF in hex, in this exercise however, we changed the hex number 3FFFF to 1FFFF to represent 128K instead to indicate that we are only going up to 128k. To account for whether the address is increasing or decreasing, a forward variable is created. Since the address needs to be in increasing order in session 1, forward would be 1'b1, indicating that forward is true. At the end of the read state, BIST_we_n, the write enable needs to be set to 1'b0, to disable the enable status. At the end of state 4, instead of going to idle state like in experiment 4, we instead set the address to 3FFFF/256K, forward to be 1'b0, and go back to state write state to start session 2.

States for session 1: S_IDLE -> S_WRITE_CYCLE -> S_DELY_1 -> S_DELY_2 -> S_READ_CYCLE -> S_DELY_3 -> S_DELY_4 -> S_WRITE_CYCLE

For the second session of this exercise, we had to read and write the second half of the external SRAM where we had to start from 256k-1 address to the 128k address. We first created a forward variable where it decides the direction of looking at the address. We added conditional statements to check the correct direction. We had also changed the assign statement of the: assign BIST_expected_data[15:0] <= BIST_address[15:0] - 16'd2; Here it had to be changed to -16d'1 for the forward direction and +16d'1 for the reverse direction. These new statements were added to the delay 2, write, and read cycle. In the write cycle this was added: This checks to see if the forward direction is finished and whether it should switch to the second session where it sets the address to 256k-1 position and starts back at delay 1 to go through the whole process again. THe next snippet of code finds out if the second session has reached the 128k-1 position so that the code can be sent to the read cycle.

```
if(BIST_address == 18'h20000) begin
   BIST_we_n<=1'b1;
   BIST_address <= 18'h3FFFF;
   BIST_state <= S_DELAY_1;
end
```

The read cycle code is almost identical to the ones above except the data is checked for any mismatches and checks to see if it can move onto delay 3. Finally in delay 4:This code checks if it should move onto session 2 or to finish the code completely.

```
if(BIST_address == 18'h1FFFF) begin
   BIST_we_n<=1'b1;
   BIST_address <= 18'd0;
   BIST_state <= S_DELAY_1;
end
if(forward == 1'b1) begin
   forward<=1'b0;
   BIST_address <=18'h3FFFF;
   BIST_state <= S_WRITE_CYCLE;
end
else if(forward == 1'b0) begin
   BIST_state <= S_IDLE;
   BIST_finish <= 1'b1;
end
```

States for session 2: S_WRITE_CYCLE -> S_DELAY_1 -> S_DELAY_2 -> S_READ_CYCLE -> S_DELY_3 -> S_DELY_4 -> S_IDLE

States for the whole exercise: S_IDLE -> S_WRITE_CYCLE -> S_DELAY_1 -> S_DELAY_2 -> S_READ_CYCLE -> S_DELY_3 -> S_DELY_4 -> S_WRITE_CYCLE -> S_DELAY_1 -> S_DELAY_2 -> S_READ_CYCLE -> S_DELY_3 -> S_DELY_4 -> S_IDLE