Bert Zhao
EE 371

Abstract: In this lab, we work with memory as we implement it on the DE1-SoC. More specifically, we work with a 32x4 RAM. This means that the memory module has a 5-bit address width, enough to store 32 words. The width of each word is also 4 bits. The overall goal of this lab is to learn how to read and write data to this memory module.

Introduction: The main purpose of this lab is to introduce the concepts behind memory. We implement a 32x4 RAM in multiple ways on the FPGA to learn the properties of memory such as the read and write processes. This lab first explores Quartus' own implementation of memory using a M10K memory blocks. We then export this design to the DE1-SoC so that we can physically manipulate the write address and data. We then repeat this process with our own memory implementation with a two dimensional array. In the final part of the lab, we explore a two port RAM module so that we can have independent read and write addresses.

Materials:

- **Altera Terasic DE1-SoC Board with power and USB cables**
- **PC with Altera Quartus Prime Lite installed**

Procedure: Task 1

The purpose of the first task of this lab is to implement a 32x4 RAM using Quartus' own memory solution. We use a M10K memory block to create a single port memory module. We create this first RAM module by following the given directions in the lab specifications. We then explore the functionality of this memory through software only by creating a top level file to initialize this memory and then simulate its behavior using ModelSim.

Task 2

In the second task of this lab, we continue from where we left off from task one. We create a different top level file, this time enabling us to export the design to the DE1-SoC. For the inputs, we use SW3-0 for the input data, SW9 for the write enable signal, and SW8-4 for the write address. For the outputs, we use HEX5-4 for the write address, HEX2 for the write data, and HEX0 for the read data. The major difference we have with this module's implementation is that KEY0 is our clock signal.

Task 3

In the third task of the lab, we implement the 32x4 RAM module from scratch instead of using the preconstructed on board memory block. This implementation is done with a 32x4 array. We again export this design to the DE1-SoC using the same inputs and outputs as task 2.

Task 4

In the final part of this lab, we return to Quartus' own implementation of memory. This time we create a dual port memory module, so that we can have separate write and read addresses for our 32x4 RAM. We also initialize the stored memory to values of our own choice using a .mif file. To read and write to our memory module, we first export the design to the DE1-SoC. The inputs are SW8-4 for the write address, SW 3-0 for the write data, SW9 for the write enable signal, the 50MHz clock, and KEY0 for reset to the read address. The outputs are HEX5-4 for the write address, HEX3-2 for the read address, HEX1 for the write data, and HEX0 for the read data. Since we did not have more switches, the read address was done using a circular up counter.

Results and Analysis:

Tasks 1 and 3 were tested using ModelSim. Tasks 2 and 4 were not due to having such similar functionalities as the previous tasks. The flow summaries for each task was also captured. The most important statistic of note is the total block memory bits. In each case, there were 128 block memory bits, one for each bit of data for each address. This lab also required the use of KEY0 for a clock input rather than the standard 50MHz or divided clock. The use of a manual clock makes it much easier to observe the behavior of the read and write processes per clock cycle. This is especially important because the data output occurs one clock cycle after the same data is input. It would be almost impossible to notice this behavior on a faster clock. The simulation waveforms and flow summaries are included below. Task 4 also includes a SignalTap analysis to verify the original contents of the .nif file.

ModelSim Screenshots:

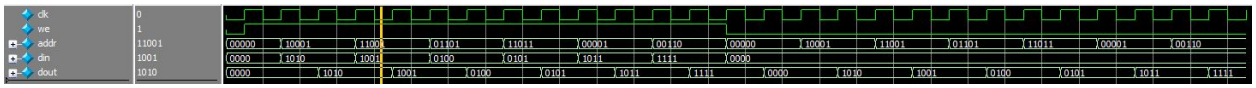

Figure 1: Task 1 ram32x4 ModelSim screenshot

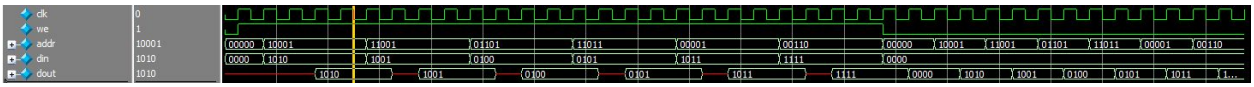

Figure 2: Task 3 ram32x4 ModelSim screenshot

Flow Summaries:

Flow Summary

Flow Status	Successful - Fri Oct 12 17:14:23 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	ram
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	15
Total virtual pins	0
Total block memory bits	128
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Figure 3: Task 1 Flow Summary

Flow Summary

Flow Status	Successful - Tue Oct 16 01:52:45 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	67
Total virtual pins	0
Total block memory bits	128
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Figure 4: Task 2 Flow Summary

Flow Summary

Flow Status	Successful - Tue Oct 16 01:25:47 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	ram32x4
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	0
Total pins	15
Total virtual pins	0
Total block memory bits	128
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Figure 5: Task 3 Flow Summary

Flow Summary

Flow Status	Successful - Tue Oct 16 01:53:34 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	30
Total pins	67
Total virtual pins	0
Total block memory bits	128
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Figure 6: Task 4 Flow Summary

SignalTap Screenshot:

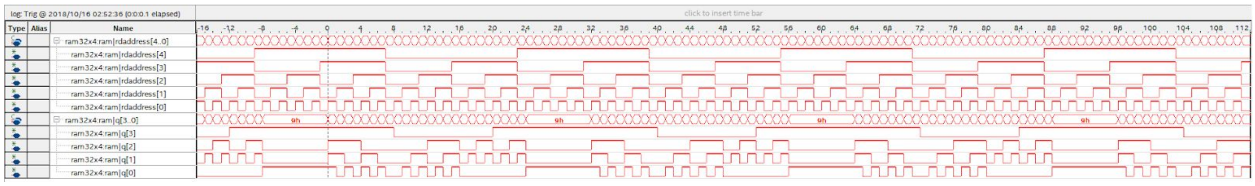

Figure 7: Task 4 SignalTap screenshot

Conclusion:

Overall, this lab introduces the concepts of memory previously covered in lecture. This lab provides some hands on experience with RAM in particular. We are given the option to write whatever data we choose on any address of our 32x4 RAM. The other major type of memory is ROM, and we spend more time with that in homework 2. From start to finish, this lab took approximately 6 hours to complete.

Appendix:

```
module ram(clk, addr, din, we, dout);
    input logic clk;
    input logic [4:0] addr;
    input logic we;
    input logic [3:0] din;
    output logic [3:0] dout;

    ram32x4 ram(.address(addr), .clock(clk), .data(din), .wren(we), .q(dout));
endmodule

`timescale 1 ps / 1 ps

module ram_tb();
    logic clk, we;
    logic [4:0] addr;
    logic [3:0] din, dout;

    // Set up the clock.
    parameter CLOCK_PERIOD = 100;
    initial begin
        clk <= 0;
        forever #(CLOCK_PERIOD/2) clk <= ~clk;
    end

    ram dut(clk, addr, din, we, dout);

    initial begin
        we <= 0; addr <= 0; din <= 0; @(posedge clk);
        we <= 1; @(posedge clk);
        addr <= 17; din <= 10; @(posedge clk); @(posedge clk);
        addr <= 25; din <= 9; @(posedge clk); @(posedge clk);
        addr <= 13; din <= 4; @(posedge clk); @(posedge clk);
        addr <= 27; din <= 5; @(posedge clk); @(posedge clk);
        addr <= 1; din <= 11; @(posedge clk); @(posedge clk);
        addr <= 6; din <= 15; @(posedge clk); @(posedge clk);
        we <= 0; addr <= 0; din <= 0; @(posedge clk); @(posedge clk);
        addr <= 17; @(posedge clk); @(posedge clk);
        addr <= 25; @(posedge clk); @(posedge clk);
        addr <= 13; @(posedge clk); @(posedge clk);
        addr <= 27; @(posedge clk); @(posedge clk);
        addr <= 1; @(posedge clk); @(posedge clk);
        addr <= 6; @(posedge clk); @(posedge clk);
        $stop;
    end
endmodule
```

Code 1: Task 1 ram.sv

```
module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
   input logic CLOCK_50; // 50MHz clock.
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;
   input logic [3:0] KEY; // True when not pressed, False when pressed
   input logic [9:0] SW;

   logic [4:0] addr;
   logic [3:0] din;
   assign addr = SW[8:4];
   assign din = SW[3:0];
   assign wren = SW[9];
   assign clk = ~KEY[0];
   logic [3:0] dout;

   ram32x4 ram(.address(addr), .clock(clk), .data(din), .wren, .q(dout));

   // Hex displays for the address
   display addr_0(.in({addr[3:0]}), .hex(HEX4));
   display addr_1(.in({3'b000, addr[4]}), .hex(HEX5));

   // Hex displays for the data in
   display din_disp(.in(din), .hex(HEX2));

   // Hex displays for the read
   display dout_disp(.in(dout), .hex(HEX0));
endmodule
```

Code 2: Task 2 DE1_SoC.sv

```
module ram32x4(addr, clk, din, we, dout);
   input logic clk, we;
   input logic [4:0] addr;
   input logic [3:0] din;
   output logic [3:0] dout;

   // Signal declaration
   logic [3:0] memory_array [0:31];
   logic [3:0] data_reg;

   always_ff @(posedge clk) begin
      // Write
      if (we)
         memory_array[addr] <= din;
      // Read
      data_reg <= memory_array[addr];
   end

   // Output
   assign dout = data_reg;
endmodule
module ram32x4_tb();
   logic clk, we;
   logic [4:0] addr;
   logic [3:0] din, dout;

   // Set up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   ram32x4 dut(addr, clk, din, we, dout);
```

```
    initial begin
        we <= 0; addr <= 0; din <= 0; @(posedge clk);
        we <= 1; @(posedge clk);
        addr <= 17; din <= 10; @(posedge clk); @(posedge clk); @(posedge clk); @(posedge clk);
        addr <= 25; din <= 9; @(posedge clk); @(posedge clk); @(posedge clk); @(posedge clk);
        addr <= 13; din <= 4; @(posedge clk); @(posedge clk); @(posedge clk); @(posedge clk);
        addr <= 27; din <= 5; @(posedge clk); @(posedge clk); @(posedge clk); @(posedge clk);
        addr <= 1; din <= 11; @(posedge clk); @(posedge clk); @(posedge clk); @(posedge clk);
        addr <= 6; din <= 15; @(posedge clk); @(posedge clk); @(posedge clk); @(posedge clk);
        we <= 0; addr <= 0; din <= 0; @(posedge clk); @(posedge clk);
        addr <= 17; @(posedge clk); @(posedge clk);
        addr <= 25; @(posedge clk); @(posedge clk);
        addr <= 13; @(posedge clk); @(posedge clk);
        addr <= 27; @(posedge clk); @(posedge clk);
        addr <= 1; @(posedge clk); @(posedge clk);
        addr <= 6; @(posedge clk); @(posedge clk);
        $stop;
    end
endmodule
```

Code 3: Task 3 ram32x4.sv

```
module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW);
    input logic CLOCK_50; // 50MHz clock.
    output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
    output logic [9:0] LEDR;
    input logic [3:0] KEY; // True when not pressed, False when pressed
    input logic [9:0] SW;

    // Input/output assignments
    logic [4:0] addr_w, addr_r;
    logic [3:0] din;
    assign addr_w = SW[8:4];
    assign din = SW[3:0];
    assign wren = SW[9];
    assign reset = ~KEY[0];
    logic [3:0] dout;

    // 32x4 RAM module
    ram32x4 ram(.clock(CLOCK_50), .data(din), .rdaddress(addr_r), .wraddress(addr_w), .wren, .q(dout));

    // Circular up counter
    counter raddr(.clk(CLOCK_50), .reset, .out(addr_r));

    // Hex displays for the write address
    display addr_w0(.in({addr_w[3:0]}), .hex(HEX4));
    display addr_w1(.in({3'b000, addr_w[4]}), .hex(HEX5));

    // Hex displays for the read address
    display addr_r0(.in({addr_r[3:0]}), .hex(HEX2));
    display addr_r1(.in({3'b000, addr_r[4]}), .hex(HEX3));

    // Hex displays for the data in
    display din_disp(.in(din), .hex(HEX1));

    // Hex displays for the read
    display dout_disp(.in(dout), .hex(HEX0));
endmodule
```

Code 4: Task 4 DE1_SoC.sv

```
module counter(clk, reset, out);
   input logic clk, reset;
   output logic [4:0] out;

   always_ff @(posedge clk) begin
      if (reset) begin
         out <= 0;
      end else begin
         if (out < 32) begin
            out <= out + 1;
         end else begin // restart counter
            out <= 0;
         end
      end
   end
endmodule
```

Code 5: Task 4 counter.sv

```
module display(in, hex);
   input logic [3:0] in;
   output logic [6:0] hex;

   always @(in)
      case (in)
         0: hex = 7'b1000000; // 0
         1: hex = 7'b1111001; // 1
         2: hex = 7'b0100100; // 2
         3: hex = 7'b0110000; // 3
         4: hex = 7'b0011001; // 4
         5: hex = 7'b0010010; // 5
         6: hex = 7'b0000010; // 6
         7: hex = 7'b1111000; // 7
         8: hex = 7'b0000000; // 8
         9: hex = 7'b0010000; // 9
         10: hex = 7'b0001000;    // A
         11: hex = 7'b0000011;    // b
         12: hex = 7'b1000110;    // C
         13: hex = 7'b0100001;    // d
         14: hex = 7'b0000110;    // E
         15: hex = 7'b0001110;    // F
      endcase
endmodule
```

Code 6: display.sv