System Synthesis and Modeling (CSCE 3953)

Final Project (Step 2) Report

Zack Fravel

11/9/16

zpfravel@uark.edu

**Report**

Step 2 of the project had us design, simulate, and synthesize our register components as well as the program counter for our RISC microprocessor. looked at the configuration of our system and determined to break this up into four separate modules. The memory address register (MAR) checks if an enable signal is active, if so it latches the data from the bus into its register and continuously outputs its contents to the memory. The memory data register (MDR) is a little more complicated in that it requires two registers internally, one for reading and one for writing to/from memory. So, I have three enable signals, one for reading, writing, and outputting to the bus. When the read signal is high, the register latches in the data it is receiving from memory. When the output enable signal is active, it sends this data through a tri state to the bus. Whenever write enable is active, the other internal register latches the data from the bus and continuously outputs its data to the memory. Finally, the general purpose register (GPR) is a simple read/write register. I have an in-enable signal to allow the register to latch data from the bust, and an out-enable signal to allow the tri state buffer to send the register's contents across the bus. All of these registers also check for signal reset and set their contents to 16'h0000.

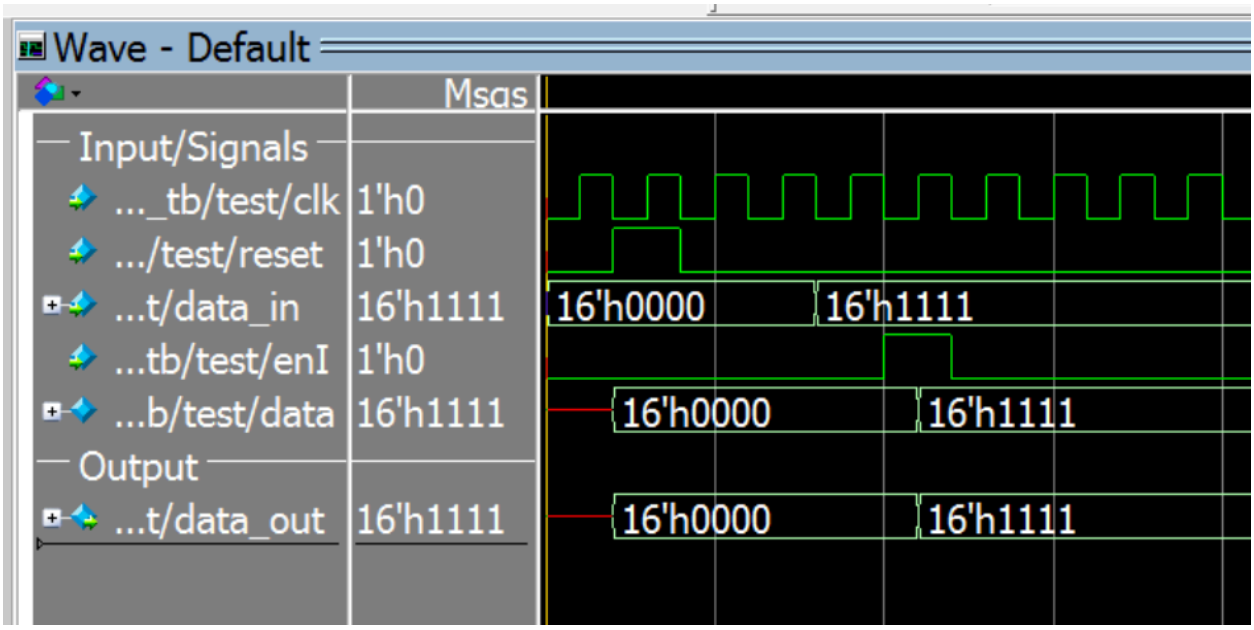
Last but not least there is the program counter. The program counter is very simple, on reset it sets its internal register to 16'h0000. The counter continuously checks for an increment signal that is generated from the FSM's in the decoding process, upon increment == 1 the counter increments its value by one. It's output is also gated by a tri state buffer. All designs were synthesized successfully, I have included screenshots of delay.

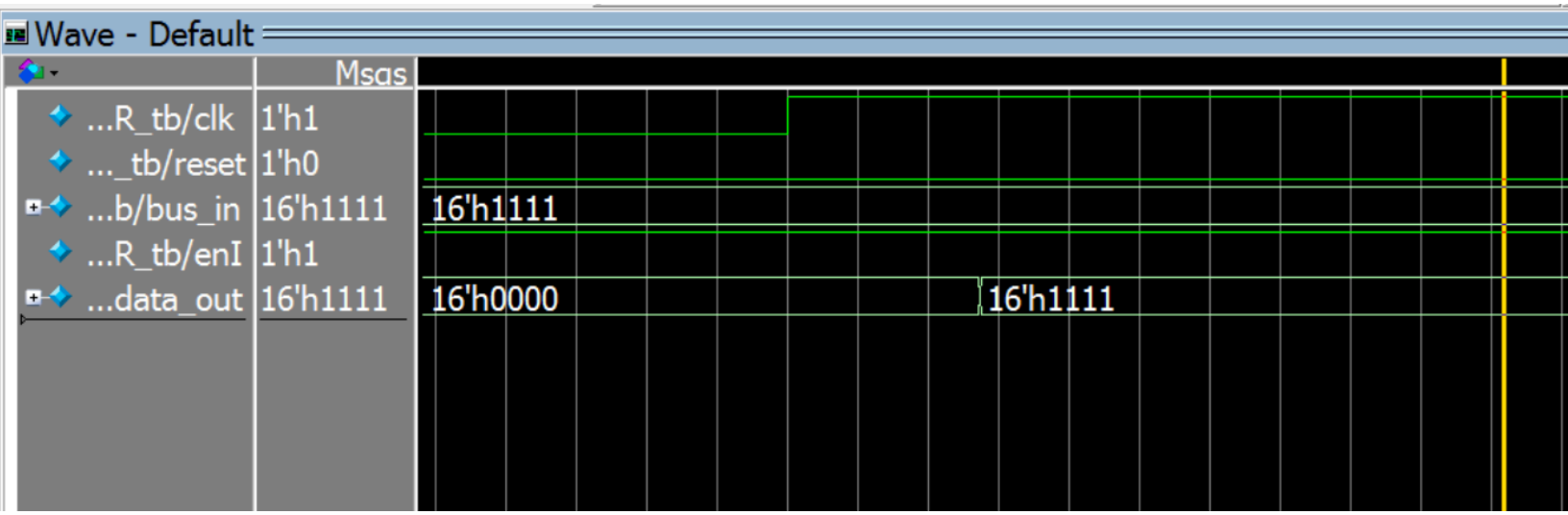
**Source Code/Screenshots/Testbench**

**MAR**

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 2)

module MAR(clk, reset, data_in, enI, data_out);

// Input/Output/Register Declaration
input clk; input reset; input[15:0] data_in;
input enI; output[15:0] data_out;
reg[15:0] data;

// Architecture

always@(posedge clk or posedge reset)
begin
        if (reset == 1)
           begin
                data <= 16'h0000;
           end
        else
           if (enI == 1)
                data <= data_in;
end

        assign data_out = data;

endmodule
```

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 2)

`timescale 1ns/1ns
module MAR_tb;

// Inputs
        reg clk; reg reset; reg[15:0] bus_in; reg enI;
// Ouputs
        wire data_out;
// Declare module
        MAR test(clk, reset, bus_in, enI, data_out);
// Testbench

        always #10 clk = ~clk;

        initial
        begin
                clk = 0; reset = 0; bus_in = 16'h0000; enI = 0;
                #20; reset = 1;
                #20; reset = 0;
                #40; bus_in = 16'h1111;
                #20; enI = 1;
                #20; enI = 0;
        end

endmodule
```



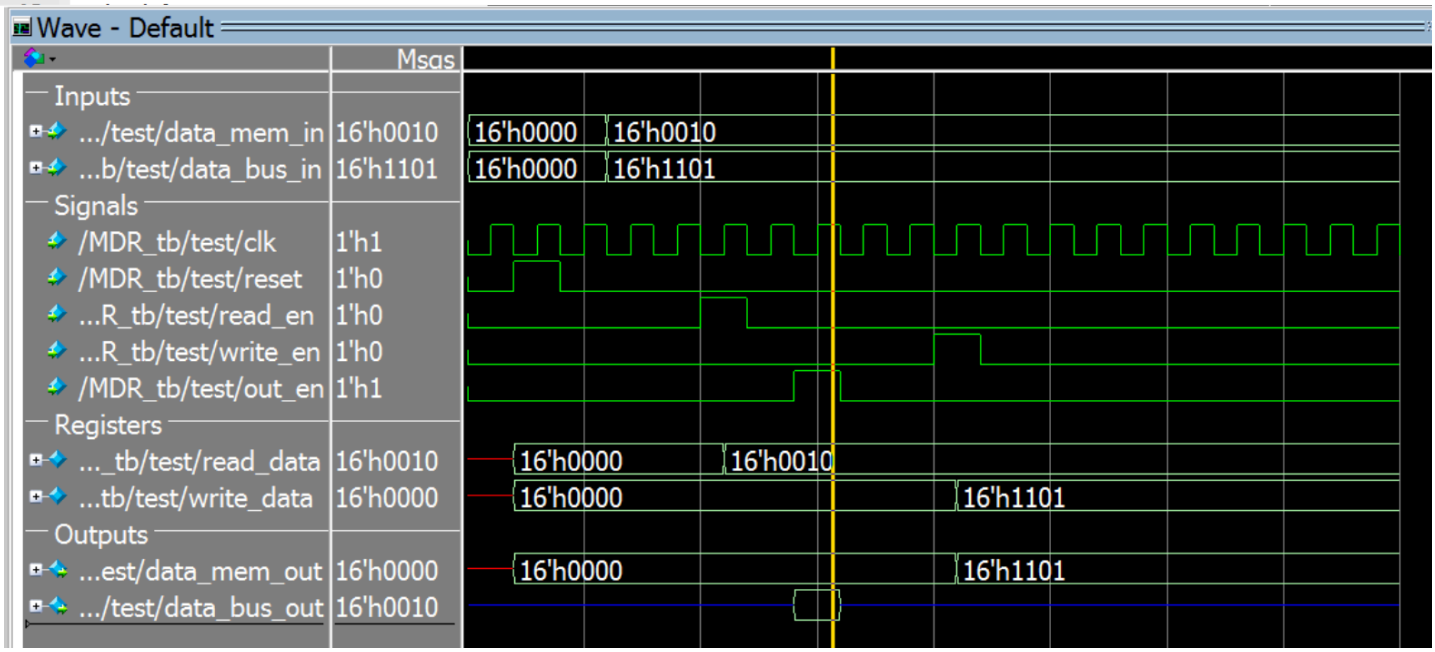


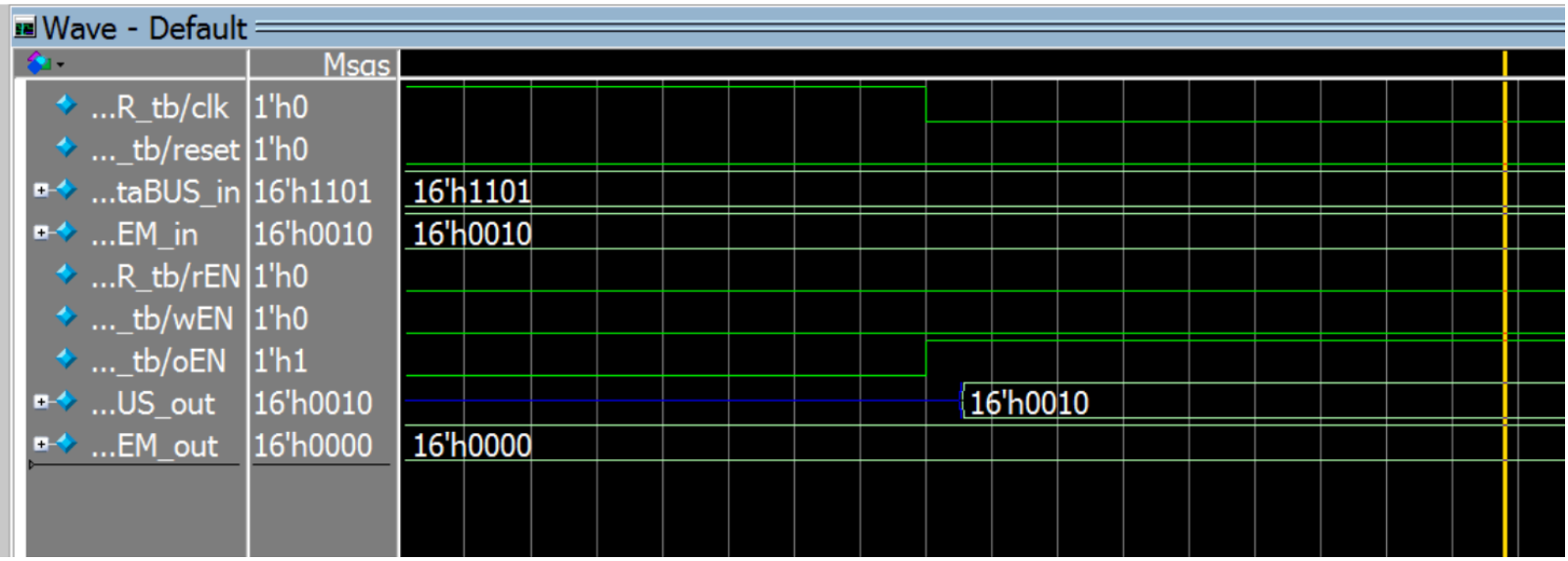

**MDR**

sers/Meryl/Documents/System Synthesis and Modeling/Final Project/Step 2/MDR.v (/MDR_tb/test) - Default

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 2)

module MDR(clk, reset, data_bus_in, data_bus_out, data_mem_in, data_mem_out, read_en, write_en, out_en);

// Input/Output/Register Declaration
input clk; input reset; input[15:0] data_mem_in; input[15:0] data_bus_in; input read_en; input write_en; input out_en
output[15:0] data_mem_out; output tri[15:0] data_bus_out;

reg[15:0] read_data;
reg[15:0] write_data;

// Architecture
always@(posedge clk or posedge reset)
begin
        if (reset == 1)
           begin
                read_data  <= 16'h0000;
                write_data <= 16'h0000;
           end
        else
                if(read_en == 1)
                        read_data <= data_mem_in;
                else if(write_en == 1)
                        write_data <= data_bus_in;
end

        assign data_mem_out = write_data;
        assign data_bus_out = (out_en) ? read_data:16'hzzzz;

endmodule
```

```
// Zack Fravel
// System Synthesis and Modeling
// Final Project (Step 2)

`timescale 1ns/1ns
module MDR_tb;

// Inputs
	reg clk; reg reset; reg[15:0] dataBUS_in; reg[15:0] dataMEM_in; reg rEN; reg wEN; reg oEN;
// Ouputs
	wire dataBUS_out; wire dataMEM_out;
// Declare module
	MDR test(clk, reset, dataBUS_in, dataBUS_out, dataMEM_in, dataMEM_out, rEN, wEN, oEN);
// Testbench

	always #10 clk = ~clk;

	initial
	begin
		clk = 0; reset = 0; dataBUS_in = 16'h0000;
		dataMEM_in = 16'h0000; rEN = 0; wEN = 0; oEN = 0;

		#20; reset = 1; #20; reset = 0;

		#20; dataBUS_in = 16'h1101; dataMEM_in = 16'h0010;

		#40; rEN = 1; #20; rEN = 0;

		#20; oEN = 1; #20; oEN = 0;

		#40; wEN = 1; #20; wEN = 0;

	end
```

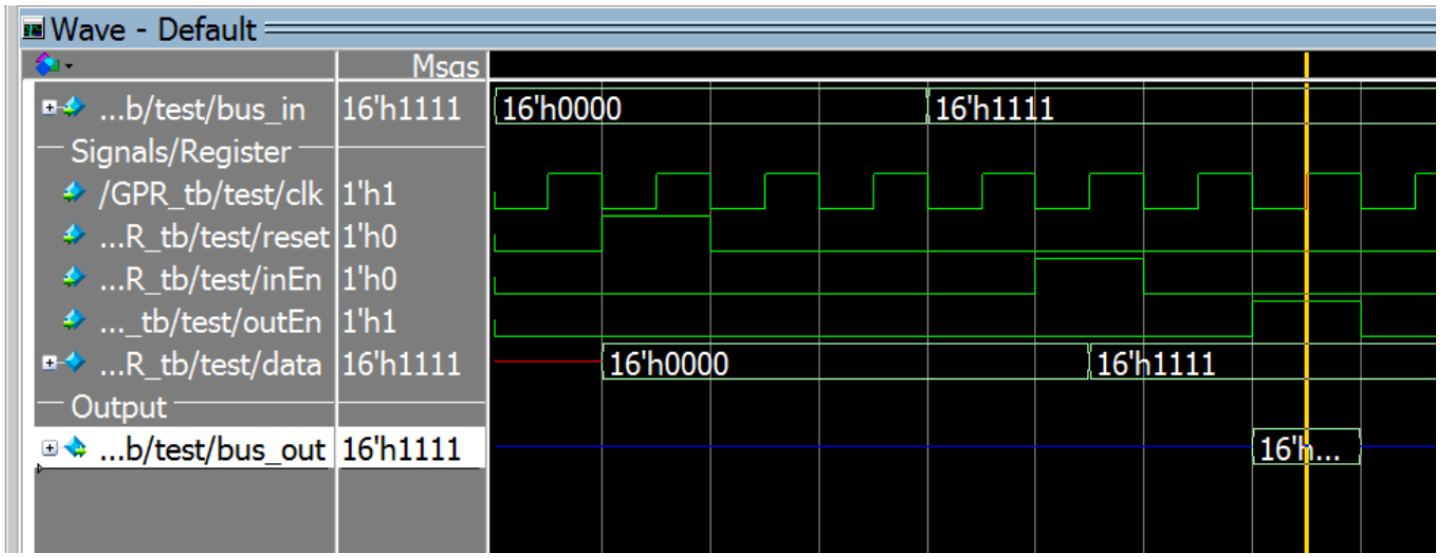


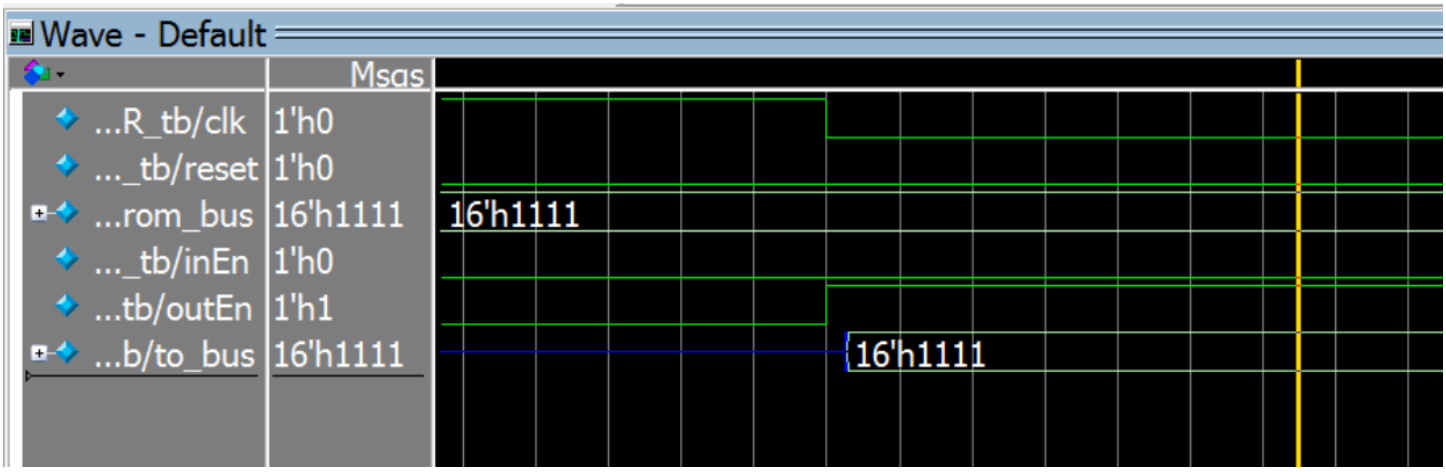



**GPR**

```
1  // Zack Fravel
2  // System Synthesis and Modeling
3  // Final Project (Step 2)
4
5  module GPR(clk, reset, bus_in, inEn, outEn, bus_out);
6
7  // Input/Output/Register Declaration
8  input clk; input reset; input[15:0] bus_in; input inEn; input outEn;
9  output tri[15:0] bus_out;
10
11 reg[15:0] data;
12
13 // Architecture
14 always@(posedge clk or posedge reset)
15 begin
16         if (reset == 1)
17                 data  <= 16'h0000;
18         else
19                 if(inEn == 1)
20                         data <= bus_in;
21 end
22
23         assign bus_out = (outEn) ? data:16'hzzzz;
24
25 endmodule
```

```
1  // Zack Fravel
2  // System Synthesis and Modeling
3  // Final Project (Step 2)
4
5  `timescale 1ns/1ns
6  module GPR_tb;
7
8  // Inputs
9          reg clk; reg reset; reg[15:0] from_bus; reg inEn; reg outEn;
10 // Ouputs
11         wire to_bus;
12 // Declare module
13         GPR test(clk, reset, from_bus, inEn, outEn, to_bus);
14 // Testbench
15
16         always #10 clk = ~clk;
17
18         initial
19         begin
20                 clk = 0; reset = 0; from_bus = 16'h0000; inEn = 0; outEn = 0;
21
22                 #20; reset = 1; #20; reset = 0;
23
24                 #40; from_bus = 16'h1111; #20; inEn = 1; #20 inEn = 0;
25
26                 #20; outEn = 1; #20; outEn = 0;
27
28         end
29
30 endmodule
```




**Program Counter**

```
1  // Zack Fravel
2  // System Synthesis and Modeling
3  // Final Project (Step 2)
4
5  module programCounter(clk, reset, incriment, oTriEn, PC_out);
6
7  // Input/Output/Register Declaration
8  input clk; input reset; input incriment;
9  input oTriEn;  output tri[15:0] PC_out;
10 reg[15:0] counter;
11
12 // Architecture
13
14 always@(posedge clk or posedge reset)
15 begin
16
17         if(reset == 1)
18                 counter <= 16'h0000;
19         else
20            if(incriment == 1)
21                 counter <= counter + 1;
22
23 end
24
25         assign PC_out = (oTriEn) ? counter:16'hzzzz;
26
27 endmodule
```

```
1  // Zack Fravel
2  // System Synthesis and Modeling
3  // Final Project (Step 2)
4
5  `timescale 1ns/1ns
6  module programCounter_tb;
7
8  // Inputs
9          reg clk; reg reset; reg incriment; reg oTriEn;
10 // Ouputs
11         wire data_out;
12 // Declare module
13         programCounter test(clk, reset, incriment, oTriEn, data_out);
14 // Testbench
15
16         always #10 clk = ~clk;
17
18         initial
19         begin
20                 clk = 0; reset = 0; incriment = 0; oTriEn = 0;
21                 #20; reset = 1;
22                 #20; reset = 0;
23                 #20; incriment = 1;
24                 #20; incriment = 0;
25                 #20; oTriEn = 1;
26                 #20; oTriEn = 0;
27                 #20; incriment = 1;
28                 #100; incriment = 0;
29                 #20; oTriEn = 1;
30                 #20; oTriEn = 0;
31
32         end
33
34 endmodule
```