CPE 233 Hardware Assignment 3

Register File and Verification

Report by:
Ethan Vosburg (evosburg@calpoly.edu)

January 28, 2024

Table of Contents

1 Project Description

In this project, a register file was created and tested using a testbench. The testbench was able to verify that the register file worked properly and passed all of the test cases. The register file was able to take in two different addresses and return the correct information as well as take in one address and write to a specific address.

2 Structural Design

2.1 Overall Elaborated Design

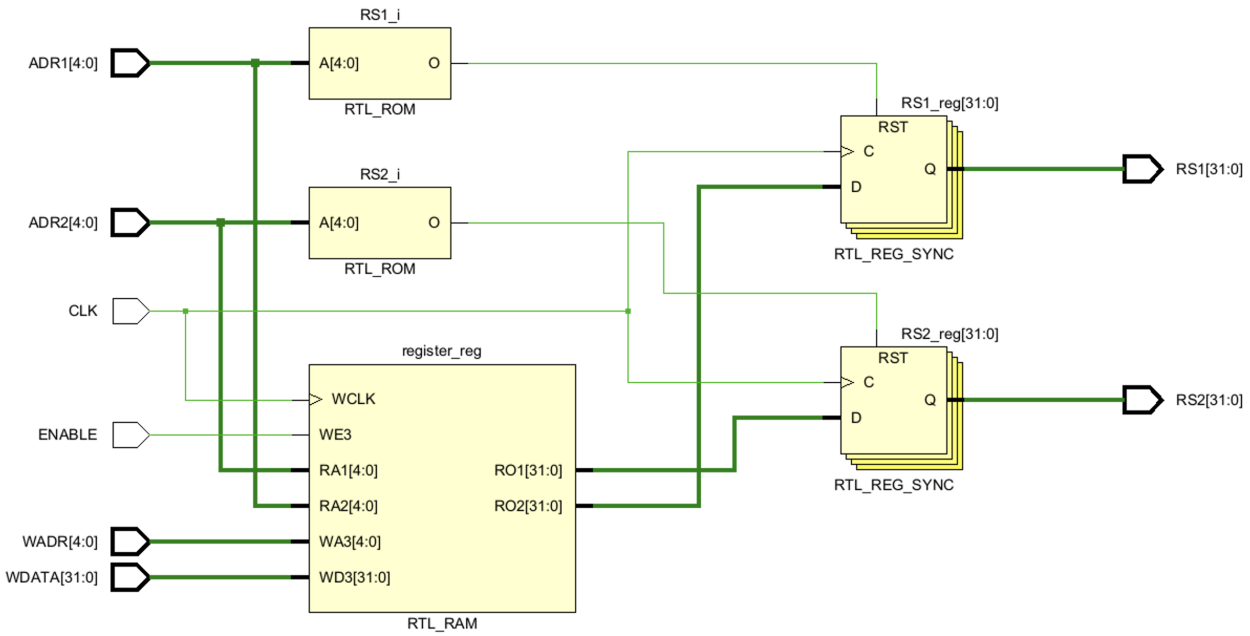

Figure 1: Program Counter Elaborated Design

3 Synthesis Warnings

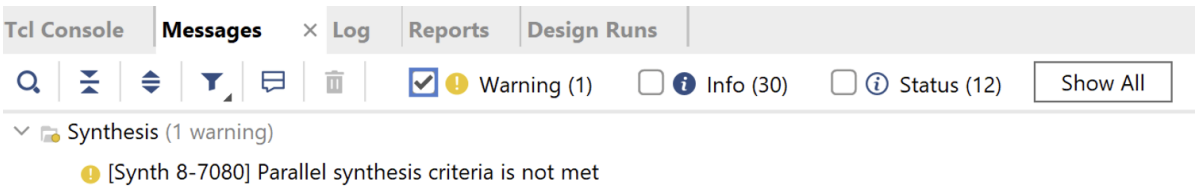

Figure 2: Synthesis Warnings

4 Verification

4.1 Testbench Coverage

When testing this design, there were several test cases that were used to verify that the design was working properly. The test cases are listed below.

1. **Random Bits Test:** This test case was used to verify that the RegFile could store any random number in all of the registers and then provide that number through both of the register outputs.
2. **Zero Bits Test:** This test case was used to verify that the RegFile could store a zero in all of the registers and then provide that number through both of the register outputs.
3. **Enable Write Test:** This test case was used to verify that the enable write bit would allow the RegFile to write to the registers when it was high and not write to the registers when it was low.
4. **Min Bits Test:** This test case was used to verify that the RegFile could store the minimum number in all of the registers and then provide that number through both of the register outputs.
5. **Max Bits Test:** This test case was used to verify that the RegFile could store the maximum number in all of the registers and then provide that number through both of the register outputs.

4.2 Testbench Code

Listing 1: Verilog Testbench for Program Counter

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 01/26/2024 11:06:16 PM
// Module Name: RegFile_TB
// Project Name: RegFile
// Target Devices: Basys 3
// Description: This Testbench tests the RegFile module
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module RegFile_TB();

    // Inputs
    logic ENABLE_TB;        // 1-bit enable
    logic CLK_TB = 0;       // 1-bit clock
    logic [4:0] ADR1_TB;    // 5-bit address
    logic [4:0] ADR2_TB;    // 5-bit address
    logic [4:0] WADR_TB;    // 5-bit address
    logic [31:0] WDATA_TB;  // 32-bit data

    // Outputs
```

```
    logic [31:0] RS1_TB;    // 32-bit data
    logic [31:0] RS2_TB;    // 32-bit data

    // Logic
    logic passed = 1;       // 1 = pass, 0 = fail
    logic simPage = 1;      // 1 = simPage, 0 = no simPage

    // Define Clock
    always #5 CLK_TB = ~CLK_TB;

    // Define Clock
    always #70 simPage = ~simPage;

    // Instantiate the Unit Under Test (UUT)
    RegFile uut (
        .ENABLE(ENABLE_TB),
        .CLK(CLK_TB),
        .ADR1(ADR1_TB),
        .ADR2(ADR2_TB),
        .WADR(WADR_TB),
        .WDATA(WDATA_TB),
        .RS1(RS1_TB),
        .RS2(RS2_TB)
    );

    // Begin simulation code
    initial begin

        $display("----------------------------------------\n");
        $display("RegFile Testbench\n");
        $display("----------------------------------------\n");

        setup();            // Setup Task
        randomBits();       // Random bits Task
        zeroBits();         // Zero bits Task
        enableWrite();      // Enable Write Task
        minBits();          // Min bits Task
        maxBits();          // Max bits Task

        // Display overall pass/fail
        $display("----------------------------------------\n");
        if (passed == 1) begin
            $display("OVERALL PASS\n");
        end else begin
            $display("OVERALL FAIL\n");
        end
        $display("----------------------------------------\n");
        $finish;
    end

    task setup();
        // Initialize Logic
        ENABLE_TB = 0;      // Initialize ENABLE_TB
        ADR1_TB = 0;        // Initialize ADR1_TB
        ADR2_TB = 0;        // Initialize ADR2_TB
        WADR_TB = 0;        // Initialize WADR_TB
        WDATA_TB = 0;       // Initialize WDATA_TB
        RS1_TB = 0;         // Initialize RS1_TB
        RS2_TB = 0;         // Initialize RS2_TB

        $display("Setup Complete");

    endtask

    task randomBits();
        // Random bits
        logic [31:0] testBits [0:31];


        //enable write input
```

```
99          ENABLE_TB = 32'h0000_0001;
100
101         // Load Random Bits
102         for (int i = 0; i < 31; i++) begin
103             testBits[i] = $random;
104             WADR_TB = i;
105             WDATA_TB = testBits[i];
106             #10;
107             // Display Case
108             // $display("WADR_TB[%0d] = %h", i, WADR_TB);
109         end
110
111         // Read random bits
112         // Check x0 returns 0
113         if (RS1_TB != 32'h0000_0000) begin
114             $display("Error: x0 test RS1_TB[%0d] = %h", 0, RS1_TB);
115             $display("Error: x0 test testBits[%0d] = %h", 0, testBits[0]);
116             passed = 0;
117         end
118         // Check x0 returns 0
119         if (RS2_TB != 32'h0000_0000) begin
120             $display("Error: x0 test RS2_TB[%0d] = %h", 0, RS2_TB);
121             $display("Error: x0 test testBits[%0d] = %h", 0, testBits[0]);
122             passed = 0;
123         end
124
125
126         for (int i = 1; i < 31; i++) begin
127             // Set ADR1 and ADR2 to i and allow for propagation
128             ADR1_TB = i;
129             ADR2_TB = i;
130             #10;
131
132             // Check RS1 returns expecred value
133             if (RS1_TB != testBits[i]) begin
134                 $display("Error: Random Read Test RS1_TB[%0d] = %h", i, RS1_TB);
135                 $display("Error: Random Read Test testBits[%0d] = %h", i, testBits[i]);
136                 passed = 0;
137             end
138
139             // Check RS2 returns expecred value
140             if (RS2_TB != testBits[i]) begin
141                 $display("Error: Random Read Test RS2_TB[%0d] = %h", i, RS2_TB);
142                 $display("Error: Random Read Test testBits[%0d] = %h", i, testBits[i]);
143                 passed = 0;
144             end
145             #10;
146
147             // Display Case
148             // $display("ADR1_TB[%0d] = %h", i, ADR1_TB);
149             // $display("ADR2_TB[%0d] = %h", i, ADR2_TB);
150         end
151
152         if (passed == 1) begin
153             $display("Random Bits Test Passed");
154         end else begin
155             $display("Random Bits Test Failed");
156         end
157
158     endtask
159
160     // Task to test zero bits
161     task zeroBits();
162         //enable write input
163         ENABLE_TB = 32'h0000_0001;
164
165         WADR_TB = 0;
166         WDATA_TB = 32'h0000_0000;
167         ADR1_TB = 0;
168         ADR2_TB = 0;
```

```
169         #10;
170 
171         // Load Zero Bits
172         // Check x0 returns 0
173         if (RS1_TB != 32'h0000_0000) begin
174             $display("Error: x0 Load All Zeros RS1");
175             passed = 0;
176         end
177         // Check x0 returns 0
178         if (RS2_TB != 32'h0000_0000) begin
179             $display("Error: x0 Load All Zeros RS2");
180             passed = 0;
181         end
182 
183 
184         WADR_TB = 0;
185         WDATA_TB = 32'hffff_ffff;
186         ADR1_TB = 0;
187         ADR2_TB = 0;
188         #10;
189 
190         // Load Zero Bits
191         // Check x0 returns 0
192         if (RS1_TB != 32'h0000_0000) begin
193             $display("Error: x0 Load All F RS1");
194             passed = 0;
195         end
196         // Check x0 returns 0
197         if (RS2_TB != 32'h0000_0000) begin
198             $display("Error: x0 Load All F RS2");
199             passed = 0;
200         end
201 
202         if (passed == 1) begin
203             $display("Zero Bits Test Passed");
204         end else begin
205             $display("Zero Bits Test Failed");
206         end
207 
208     endtask
209 
210     // Max Bits Task
211 
212     task maxBits();
213         //enable write input
214         ENABLE_TB = 32'h0000_0001;
215 
216         // Load Max Bits
217         for (int i = 0; i < 31; i++) begin
218             WADR_TB = i;
219             WDATA_TB = 32'hffff_ffff;
220             #10;
221             // Display Case
222             // $display("WADR_TB[%0d] = %h", i, WADR_TB);
223         end
224 
225         // Read Max bits
226         for (int i = 1; i < 31; i++) begin
227             // Set ADR1 and ADR2 to i and allow for propagation
228             ADR1_TB = i;
229             ADR2_TB = i;
230             #10;
231 
232             // Check RS1 returns expecred value
233             if (RS1_TB != 32'hffff_ffff) begin
234                 $display("Error: Max Read Test RS1_TB[%0d] = %h", i, RS1_TB);
235                 passed = 0;
236             end
237 
238             // Check RS2 returns expecred value
```

```
239             if (RS2_TB != 32'hffff_ffff) begin
240                 $display("Error: Max Read Test RS2_TB[%0d] = %h", i, RS2_TB);
241                 passed = 0;
242             end
243             #10;
244 
245             // Display Case
246             // $display("ADR1_TB[%0d] = %h", i, ADR1_TB);
247             // $display("ADR2_TB[%0d] = %h", i, ADR2_TB);
248         end
249 
250         if (passed == 1) begin
251             $display("Max Bits Test Passed");
252         end else begin
253             $display("Max Bits Test Failed");
254         end
255 
256     endtask
257 
258     task minBits();
259         //enable write input
260         ENABLE_TB = 32'h0000_0001;
261 
262         // Load Min Bits
263         for (int i = 0; i < 31; i++) begin
264             WADR_TB = i;
265             WDATA_TB = 32'h0000_0000; // Write minimum value
266             #10;
267             // Display Case
268             // $display("WADR_TB[%0d] = %h", i, WADR_TB);
269         end
270 
271         // Read Min bits
272         for (int i = 1; i < 31; i++) begin
273             // Set ADR1 and ADR2 to i and allow for propagation
274             ADR1_TB = i;
275             ADR2_TB = i;
276             #10;
277 
278             // Check RS1 returns expected value
279             if (RS1_TB != 32'h0000_0000) begin // Check for minimum value
280                 $display("Error: Min Read Test RS1_TB[%0d] = %h", i, RS1_TB);
281                 passed = 0;
282             end
283 
284             // Check RS2 returns expected value
285             if (RS2_TB != 32'h0000_0000) begin // Check for minimum value
286                 $display("Error: Min Read Test RS2_TB[%0d] = %h", i, RS2_TB);
287                 passed = 0;
288             end
289             #10;
290 
291             // Display Case
292             // $display("ADR1_TB[%0d] = %h", i, ADR1_TB);
293             // $display("ADR2_TB[%0d] = %h", i, ADR2_TB);
294         end
295 
296         if (passed == 1) begin
297             $display("Min Bits Test Passed");
298         end else begin
299             $display("Min Bits Test Failed");
300         end
301 
302     endtask
303 
304     //Enable Write Task
305     task enableWrite();
306         // Random bits
307         logic [31:0] testBits [0:31];
308 
```

```
309         //enable write input
310         ENABLE_TB = 32'h0000_0001;
311 
312         // Load Random Bits
313         for (int i = 0; i < 31; i++) begin
314             testBits[i] = $random;
315             WADR_TB = i;
316             WDATA_TB = testBits[i];
317             #10;
318             // Display Case
319             // $display("WADR_TB[%0d] = %h", i, WADR_TB);
320         end
321 
322         // Disable write input
323         ENABLE_TB = 32'h0000_0000;
324         #10
325 
326         // Attempt to zero out registers
327         for (int i = 0; i < 31; i++) begin
328             WADR_TB = i;
329             WDATA_TB = 32'h0000_0000; // Write minimum value
330             #10;
331             // Display Case
332             // $display("WADR_TB[%0d] = %h", i, WADR_TB);
333         end
334 
335         for (int i = 1; i < 31; i++) begin
336             // Set ADR1 and ADR2 to i and allow for propagation
337             ADR1_TB = i;
338             ADR2_TB = i;
339             #10;
340 
341             // Check RS1 returns expecred value
342             if (RS1_TB != testBits[i]) begin
343                 $display("Error: Enable Bits Test RS1_TB[%0d] = %h", i, RS1_TB);
344                 $display("Error: Enable Bits Test testBits[%0d] = %h", i, testBits[i]);
345                 passed = 0;
346             end
347 
348             // Check RS2 returns expecred value
349             if (RS2_TB != testBits[i]) begin
350                 $display("Error: Enable Bits Test RS2_TB[%0d] = %h", i, RS2_TB);
351                 $display("Error: Enable Bits Test testBits[%0d] = %h", i, testBits[i]);
352                 passed = 0;
353             end
354             #10;
355 
356             // Display Case
357             // $display("ADR1_TB[%0d] = %h", i, ADR1_TB);
358             // $display("ADR2_TB[%0d] = %h", i, ADR2_TB);
359         end
360 
361         if (passed == 1) begin
362             $display("Enable Write Test Passed");
363         end else begin
364             $display("Enable Write Test Failed");
365         end
366 
367     endtask
368 
369 
370 endmodule
```

4.3 Testbench Output

Running this testbench produced the following output in the TCL code console:

Listing 2: TCl Output from ProgramCounterEnv_TB

```
----------------------------------------

RegFile Testbench

----------------------------------------

Setup Complete
Random Bits Test Passed
Zero Bits Test Passed
Enable Write Test Passed
Min Bits Test Passed
Max Bits Test Passed
----------------------------------------

OVERALL PASS

----------------------------------------

$finish called at time : 3980 ns
```

4.4 Simulation Results

Running this testbench produced the following simulation results:

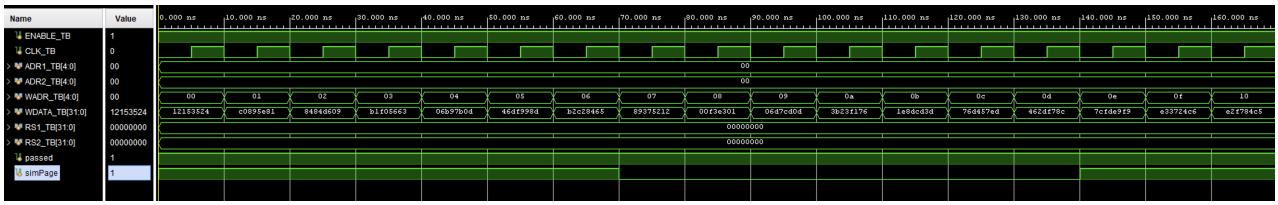

Figure 3: RegFile Simulation 0ns - 140ns

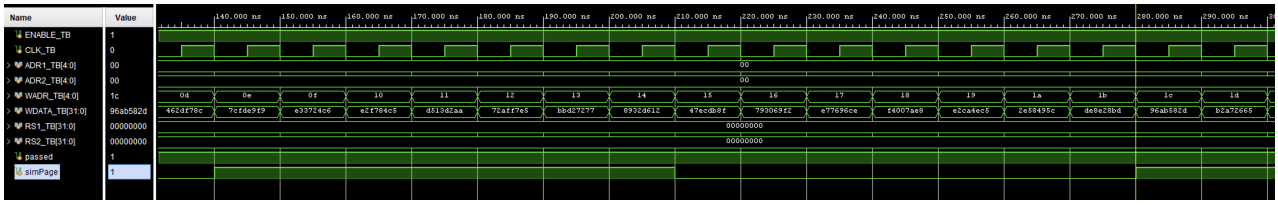

Figure 4: RegFile Simulation 140ns - 280ns

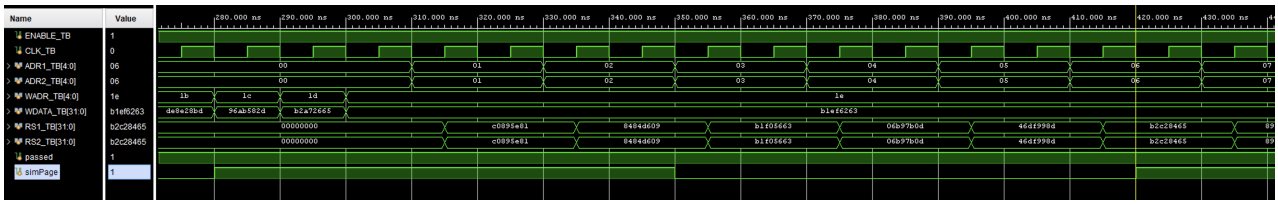

Figure 5: RegFile Simulation 280ns - 420ns

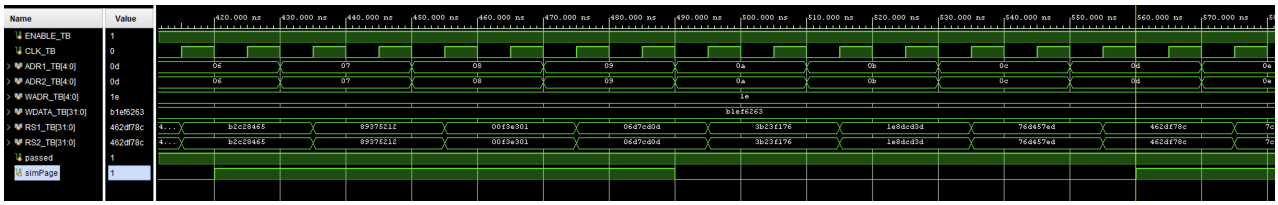

Figure 6: RegFile Simulation 420ns - 560ns

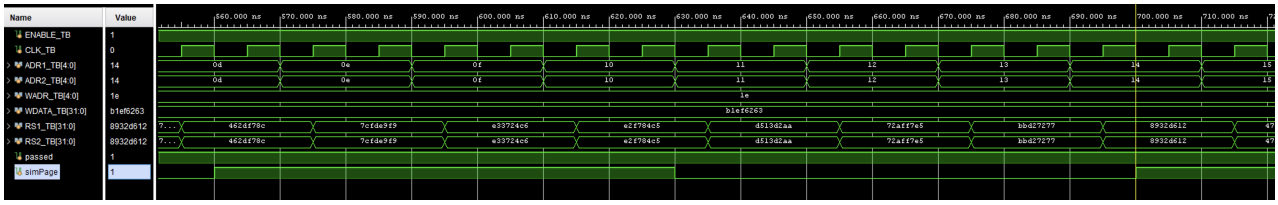

Figure 7: RegFile Simulation 560ns - 700ns

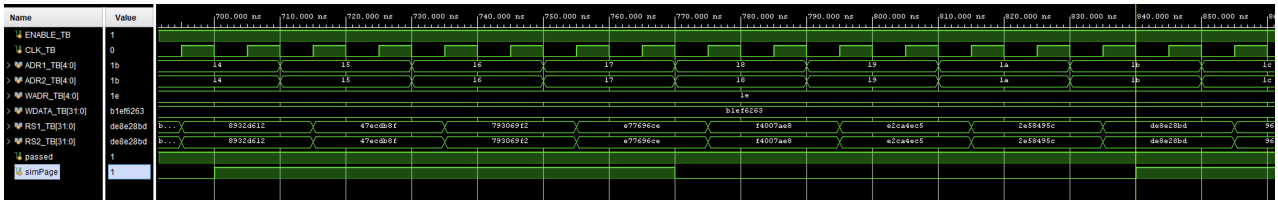

Figure 8: RegFile Simulation 700ns - 840ns

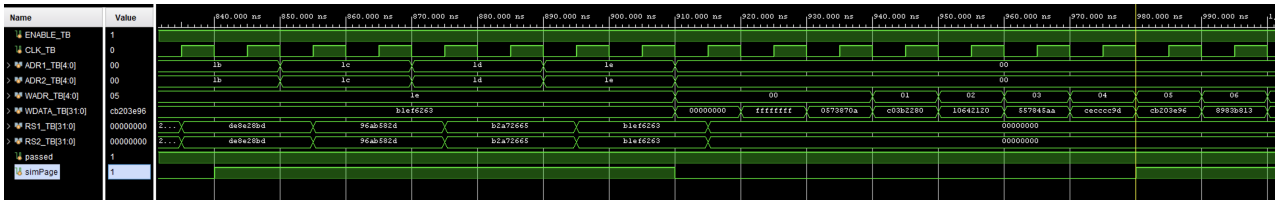

Figure 9: RegFile Simulation 840ns - 980ns

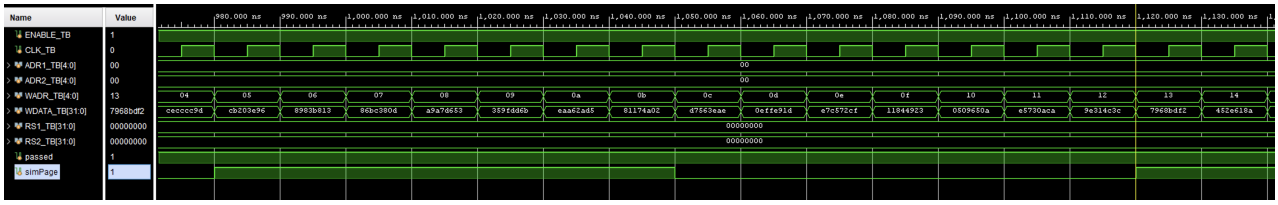

Figure 10: RegFile Simulation 980ns - 1120ns

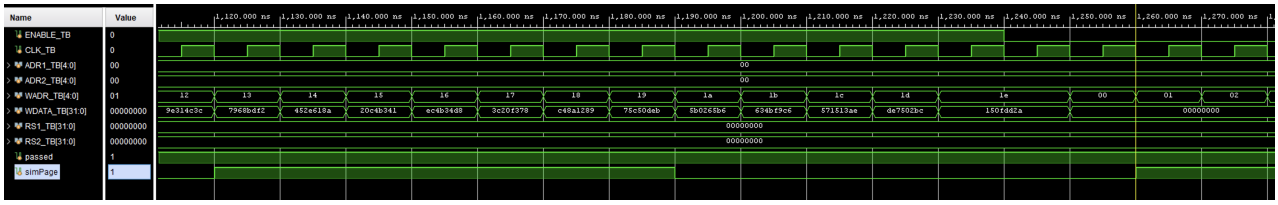

Figure 11: RegFile Simulation 1120ns - 1260ns

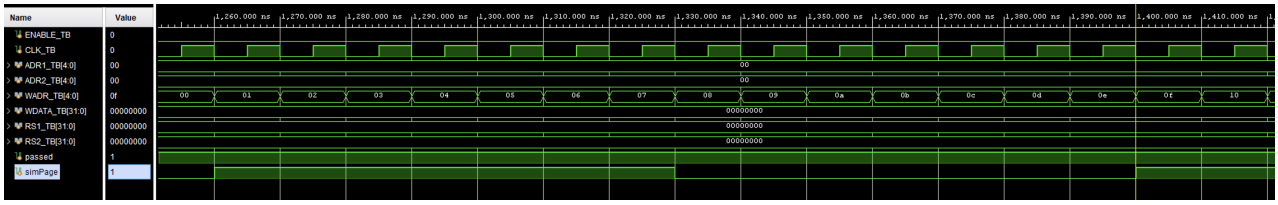

Figure 12: RegFile Simulation 1260ns - 1400ns

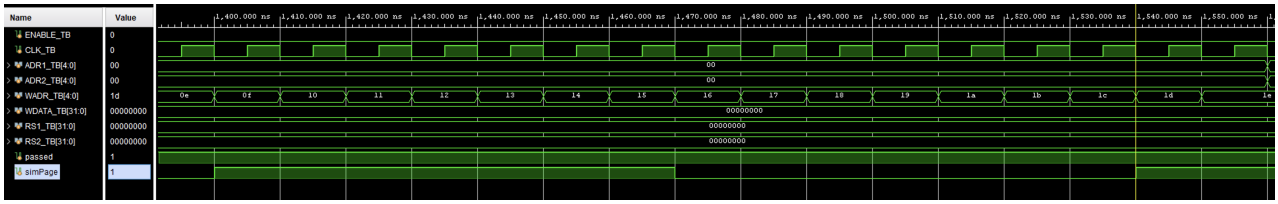

Figure 13: RegFile Simulation 1400ns - 1540ns

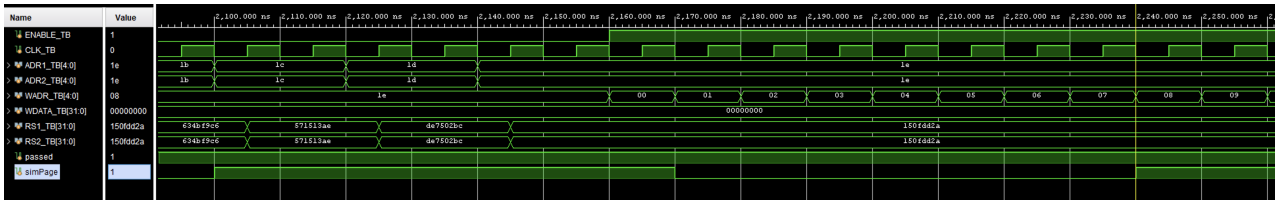

Figure 18: RegFile Simulation 2100ns - 2240ns

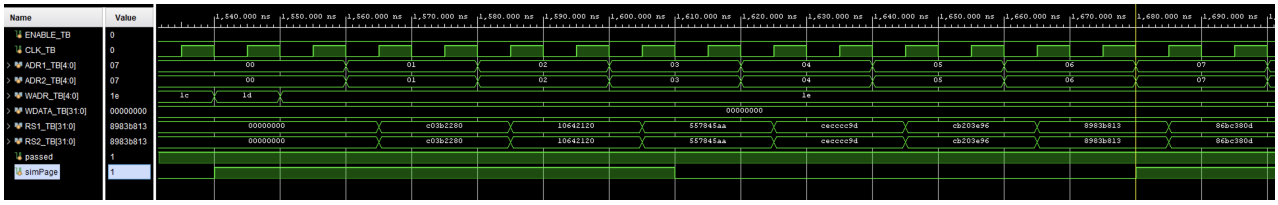

Figure 14: RegFile Simulation 1540ns - 1680ns

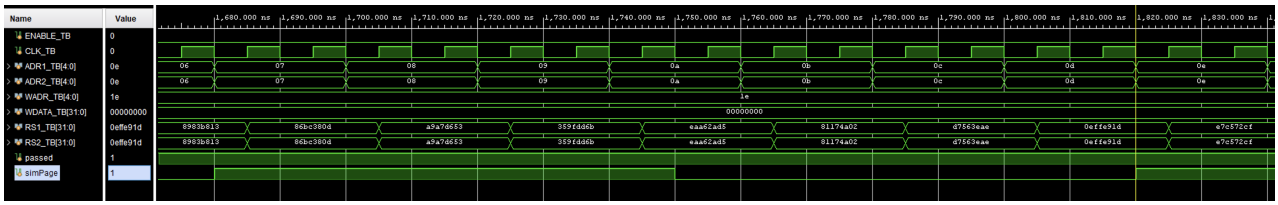

Figure 15: RegFile Simulation 1680ns - 1820ns

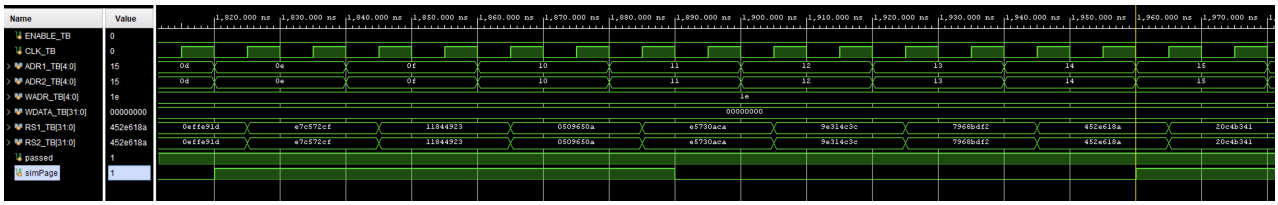

Figure 16: RegFile Simulation 1820ns - 1960ns

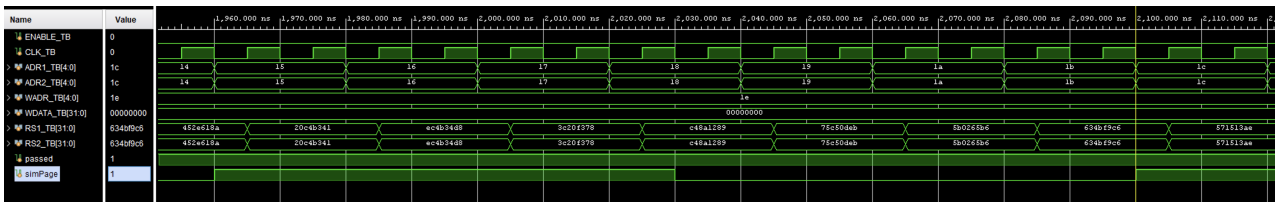

Figure 17: RegFile Simulation 1960ns - 2100ns

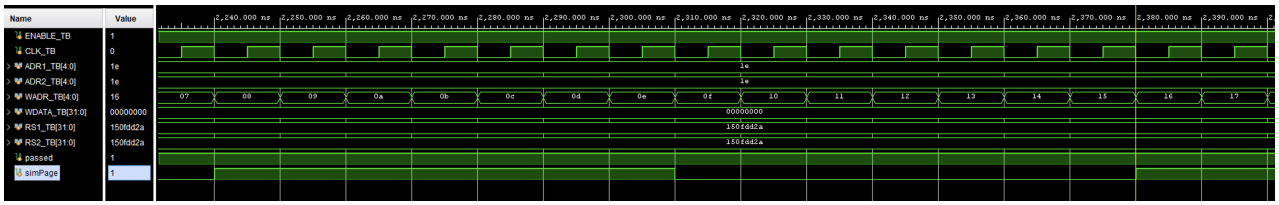

Figure 19: RegFile Simulation 2240ns - 2380ns

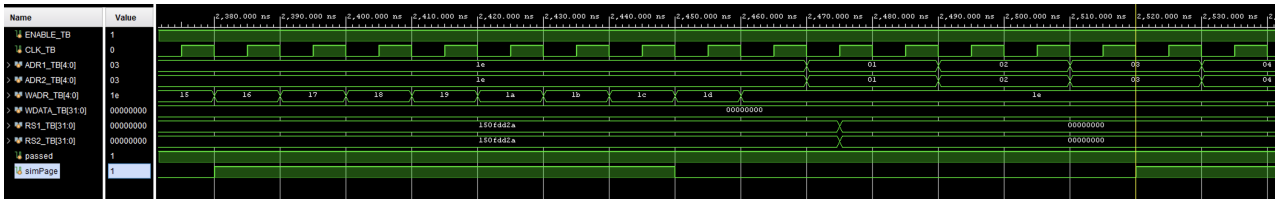

Figure 20: RegFile Simulation 2380ns - 2520ns

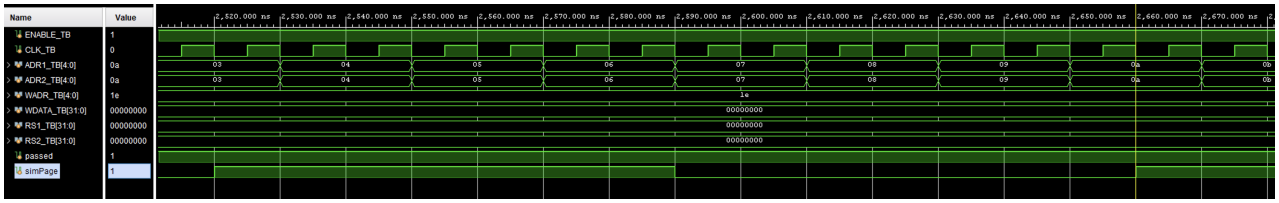

Figure 21: RegFile Simulation 2520ns - 2660ns

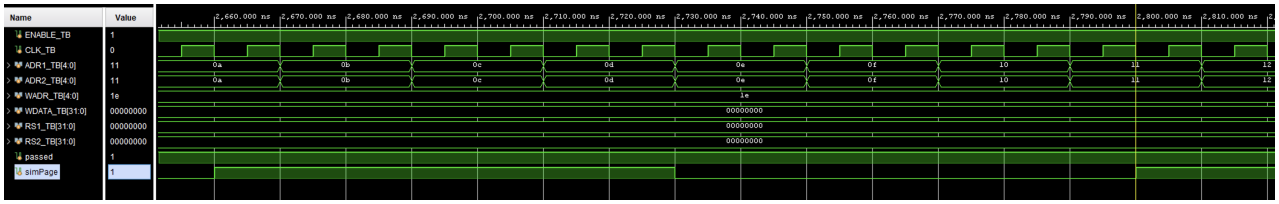

Figure 22: RegFile Simulation 2660ns - 2800ns

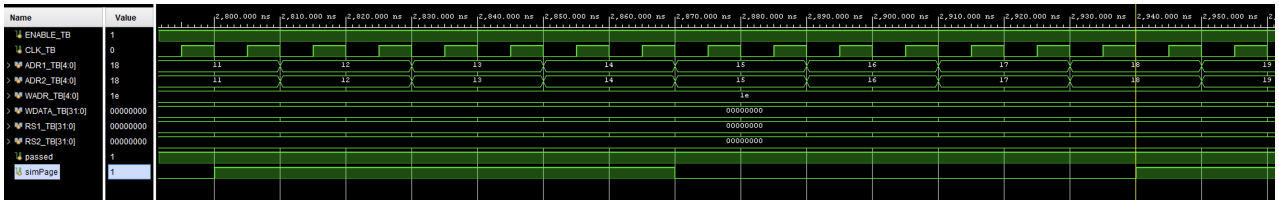

Figure 23: RegFile Simulation 2800ns - 2940ns

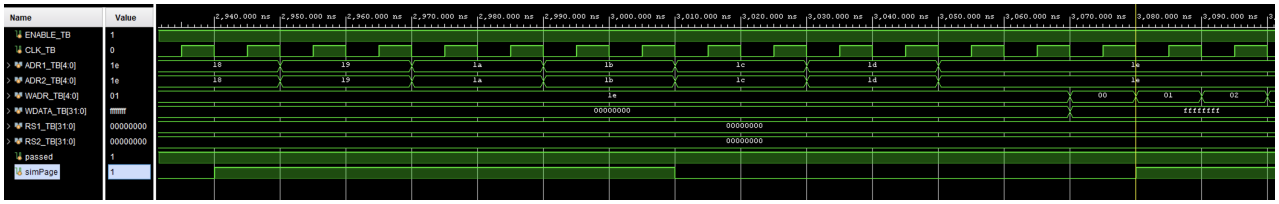

Figure 24: RegFile Simulation 2940ns - 3080ns

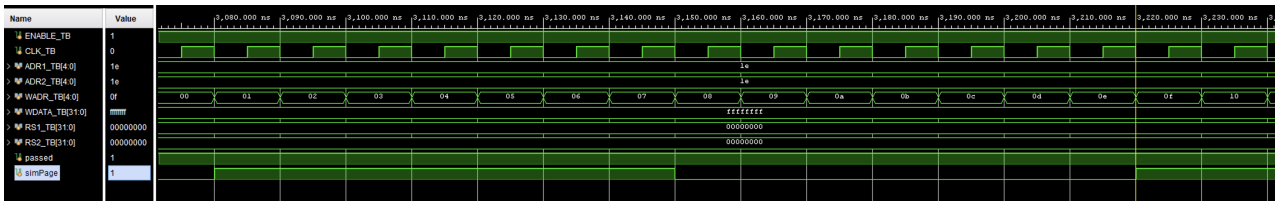

Figure 25: RegFile Simulation 3080ns - 3220ns

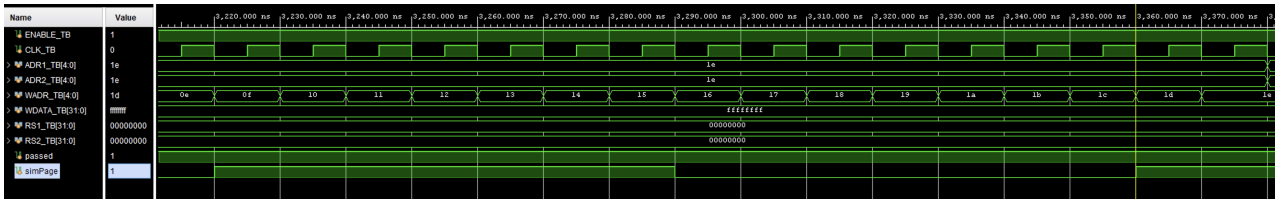

Figure 26: RegFile Simulation 3220ns - 3360ns

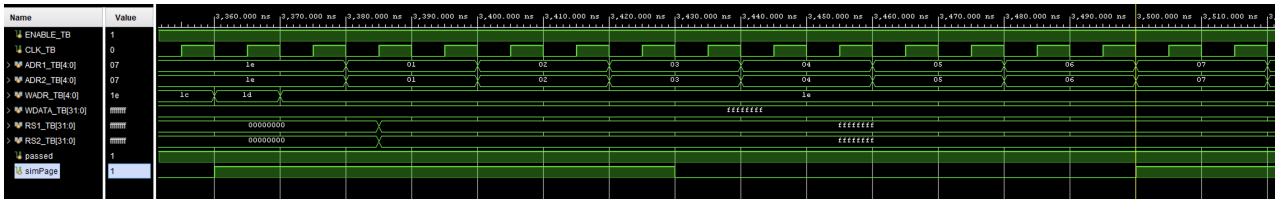

Figure 27: RegFile Simulation 3360ns - 3500ns

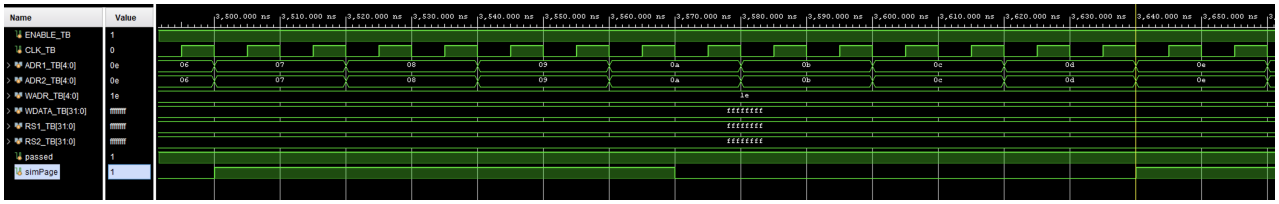

Figure 28: RegFile Simulation 3500ns - 3640ns

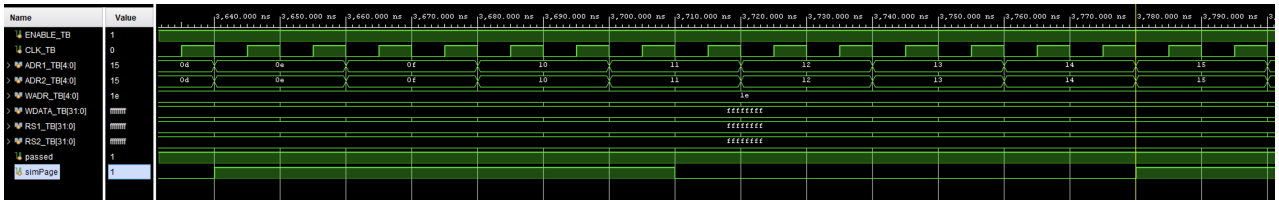

Figure 29: RegFile Simulation 3640ns - 3780ns

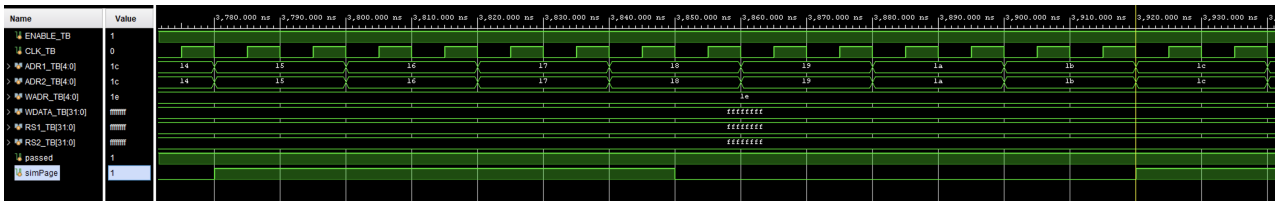

Figure 30: RegFile Simulation 3780ns - 3920ns

5 Source Code

5.1 Register File

Listing 3: Verilog Code for Register File

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 01/26/2024 06:24:51 PM
// Module Name: RegFile
// Project Name: RegFile
// Target Devices: Basys 3
// Description: Create a Register for the ALU
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module RegFile(
    // Inputs
    input ENABLE,
    input CLK,
    input [4:0] ADR1,
    input [4:0] ADR2,
    input [4:0] WADR,
    input [31:0] WDATA,

    // Outputs
    output logic [31:0] RS1,
    output logic [31:0] RS2
    );

    // Declare register array
    logic [31:0] register [0:31];

    always_ff @( posedge CLK ) begin
        if (ENABLE == 1) begin
            // Write data to register
            register[WADR] <= WDATA;
        end

        // Present data from 2 desire
        RS1 <= register[ADR1];
        RS2 <= register[ADR2];

        // Add logic for zero register
        if (ADR1 == 0) begin
            RS1 <= 32’h0000_0000;
        end
        if (ADR2 == 0) begin
            RS2 <= 32’h0000_0000;
        end
    end
endmodule
```

6 Conclusion

The program counter is a very important module and will be used heavily in the Otter processor. This counter keeps track of which line of machine code to feed the rest of the processor and it makes allows branching and many other capabilities. In this project, a program counter was created and tested using a testbench. The testbench was able to verify that the program counter and the mux associated with it worked properly and passed all of the test cases. All code for this assignment can be found here.