CPE 233 Hardware Assignment 2

Program Counter and Verification

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

January 19, 2024

Table of Contents

1 Project Description

In this project, a program counter for the Otter processor was created. The program counter is a register that holds the address of the next instruction to be executed. The program counter is incremented by 4 every clock cycle. The program counter is also able to be reset to 0. The program counter was then tested using a testbench. The testbench went through several test cases to verify that the program counter and the mux associated with it work properly. The testbench was able to be run and passed all of the test cases.

2 Structural Design

2.1 Overall Elaborated Design

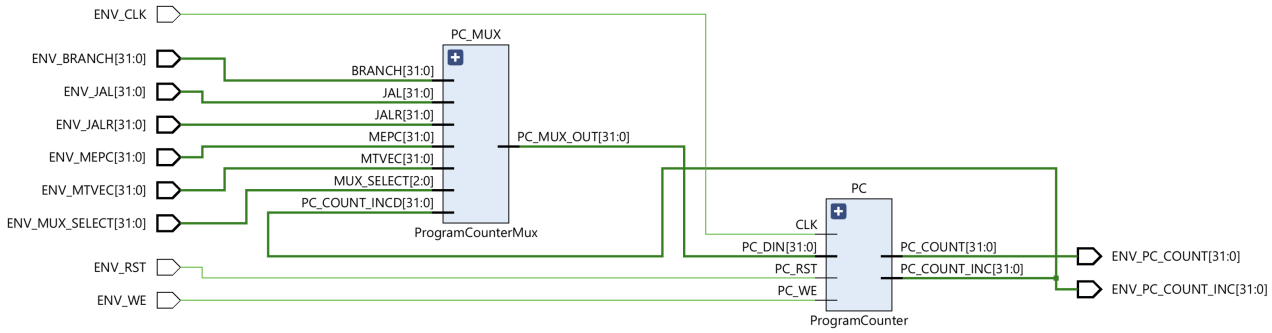

Figure 1: Program Counter Elaborated Design

2.2 Program Counter Multiplexer Elaborated Design

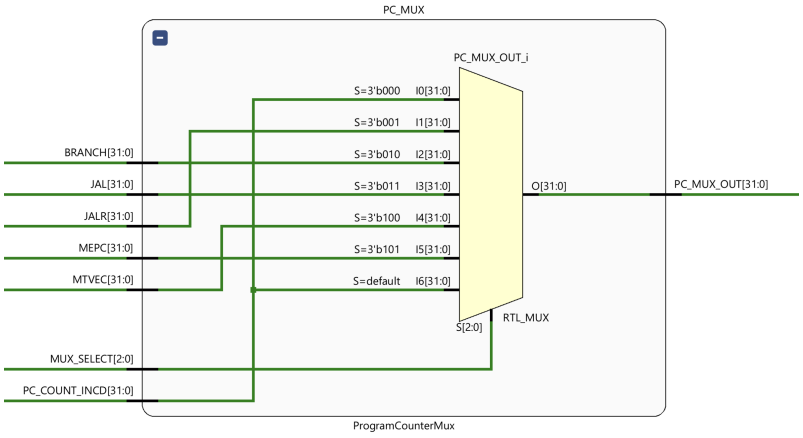

Figure 2: Program Counter Multiplexer Elaborated Design

2.3 Program Counter Main Hardware Elaborated Design

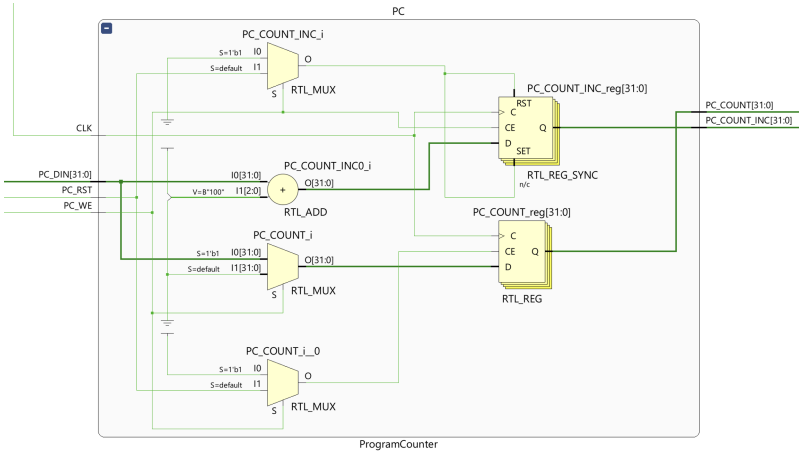

Figure 3: Program Counter Main Hardware Elaborated Design

3 Synthesis Warnings Listing

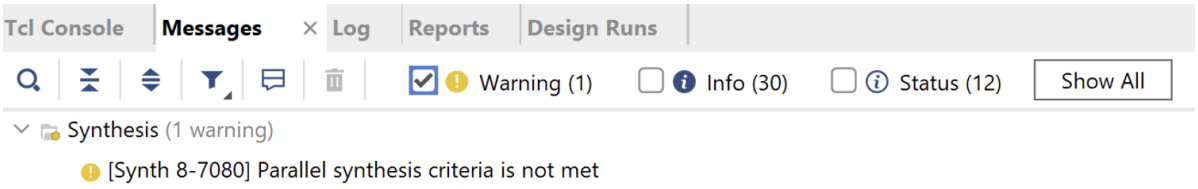

Figure 4: Synthesis Warnings

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. **ReadMux Test:** This test checks that the mux can read all Inputs properly. The test-bench sets the different inputs of the mux to 10 times the select value and then checks that the output is equal to the select value.
2. **MaxMux Test:** This test checks that when the maximum 32-bit value of the program counter is reached, the failure is predictable and expected.
3. **MinMux Test:** This test checks that when the minimum 32-bit value is loaded into all of the inputs, only valid next addresses are outputted.

4. **Reset Test:** This test checks that the reset is working properly. The testbench sets the reset to 1 and then checks that the output of the program counter is 0.
5. **RandNum Test:** This test checks that when random numbers are inputted into the program counter, the output is predictable and expected.
6. **WriteEnable Test:** This test checks that the write enable is working properly. When the write enable is set to 0, the program counter should not change. When the write enable is set to 1, the program counter should increment by 4.

4.2 Testbench Code

Listing 1: Verilog Testbench for Program Counter

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 01/19/2024 07:17:25 PM
// Module Name: ProgramCounterEnv_TB
// Project Name: ProgramCounter
// Target Devices: Basys3
// Description: Test the program counter and multiplexer together.
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////

module ProgramCounterEnv_TB();
    // Inputs
    logic ENV_RST_TB;
    logic ENV_WE_TB;
    logic [31:0] ENV_JALR_TB;
    logic [31:0] ENV_BRANCH_TB;
    logic [31:0] ENV_JAL_TB;
    logic [31:0] ENV_MTVEC_TB;
    logic [31:0] ENV_MEPC_TB;
    logic [31:0] ENV_MUX_SELECT_TB;
    logic ENV_CLK_TB;

    // Outputs
    logic [31:0] ENV_PC_COUNT_TB;
    logic [31:0] ENV_PC_COUNT_INC_TB; //4

    // Logic
    logic pass; // 1 = pass, 0 = fail

    // Instantiate the Program Counter
    ProgramCounterEnv PC(
        .ENV_RST(ENV_RST_TB),
        .ENV_WE(ENV_WE_TB),
        .ENV_JALR(ENV_JALR_TB),
        .ENV_BRANCH(ENV_BRANCH_TB),
        .ENV_JAL(ENV_JAL_TB),
        .ENV_MTVEC(ENV_MTVEC_TB),
        .ENV_MEPC(ENV_MEPC_TB),
        .ENV_MUX_SELECT(ENV_MUX_SELECT_TB),
        .ENV_CLK(ENV_CLK_TB),
        .ENV_PC_COUNT(ENV_PC_COUNT_TB),
        .ENV_PC_COUNT_INC(ENV_PC_COUNT_INC_TB)
    );

    initial begin
```

```
52          // Initialize Logic
53          ENV_CLK_TB = 0; // Instaintiate clock with 0
54          pass = 1;       // Assume pass until fail
55
56          // Run Tests
57          setup();        // Initialize all inputs with values
58          reset();        // Reset the test
59          ReadMux();      // Verify Mux woerks properly by reading all inputs
60          reset();        // Reset the test
61          MaxMux();       // Verify Mux works properly with 32'hffff_ffff values
62          reset();        // Reset the test
63          MinMux();       // Verify Mux works properly with 32'h0000_0000 values
64          reset();        // Reset the test
65          ResetTest();    // Verify reset case works on the program counter
66          reset();        // Reset the test
67          RandNum();      // Verify Mux works properly with random values
68          reset();        // Reset the test
69          WriteEnable();  // Verify program counter works properly with write enable
70          if (pass == 1) $display("All Tests Passed");
71          $finish;        // Finish the test
72
73
74
75      end
76
77      // Setup all inputs with normal values
78      task setup();
79          ENV_RST_TB = 0;
80          ENV_WE_TB = 1;
81          #10
82          ENV_RST_TB = 0;
83          ENV_JALR_TB = 0;
84          ENV_BRANCH_TB = 0;
85          ENV_JAL_TB = 0;
86          ENV_MTVEC_TB = 0;
87          ENV_MEPC_TB = 0;
88          ENV_MUX_SELECT_TB = 0;
89          $display("Setup Complete");
90      endtask // setup
91
92      // Read all inputs to verify mux works properly
93      task ReadMux();
94          // Load the program counter with the select value multiplied by 10
95          ENV_JALR_TB = 32'h0000_0010;
96          ENV_MUX_SELECT_TB = 1;
97          #10
98          // Verify the random number was loaded into the program counter and that the
99          // program counter incremented output was incremented by 4
100         if (ENV_PC_COUNT_INC_TB != 32'h0000_0014 || ENV_PC_COUNT_TB != 32'h0000_0010) begin
101             $display("JALR Read Failed");
102             pass = 0;
103         end
104
105         // Repeat for all inputs
106         ENV_BRANCH_TB = 32'h0000_0020;
107         ENV_MUX_SELECT_TB = 2;
108         #10
109         if (ENV_PC_COUNT_INC_TB != 32'h0000_0024 || ENV_PC_COUNT_TB != 32'h0000_0020) begin
110             $display("BRANCH Read Failed");
111             pass = 0;
112         end
113
114         ENV_JAL_TB = 32'h0000_0030;
115         ENV_MUX_SELECT_TB = 3;
116         #10
117         if (ENV_PC_COUNT_INC_TB != 32'h0000_0034 || ENV_PC_COUNT_TB != 32'h0000_0030) begin
118             $display("JAL Read Failed");
119             pass = 0;
120         end
121
```

```
122         ENV_MTVEC_TB = 32'h0000_0040;
123         ENV_MUX_SELECT_TB = 4;
124         #10
125         if (ENV_PC_COUNT_INC_TB != 32'h0000_0044 || ENV_PC_COUNT_TB != 32'h0000_0040) begin
126             $display("MTVEC Read Failed");
127             pass = 0;
128         end
129 
130         ENV_MEPC_TB = 32'h0000_0050;
131         ENV_MUX_SELECT_TB = 5;
132         #10
133         if (ENV_PC_COUNT_INC_TB != 32'h0000_0054 || ENV_PC_COUNT_TB != 32'h0000_0050) begin
134             $display("MEPC Read Failed");
135             pass = 0;
136         end
137 
138         // Determine if the test passed or failed
139         if (pass == 1) begin
140             $display("ReadMux Test Passed");
141         end else begin
142             $display("ReadMux Test Failed");
143         end
144 
145     endtask // ReadMux
146 
147 
148     // Verify the program counter works properly with 32'hffff_ffff values
149     task MaxMux();
150         // Load the program counter with the max 32bit value
151         ENV_JALR_TB = 32'hffff_ffff;
152         ENV_MUX_SELECT_TB = 1;
153         #10
154         // Verify the program counter incremented outputs are correct meaning they are
155         // carryed over properly but the address is now incorrect
156         if (ENV_PC_COUNT_INC_TB != 32'h0000_0003 || ENV_PC_COUNT_TB != 32'hffff_ffff) begin
157             $display("JALR Max Read Failed");
158             pass = 0;
159         end
160 
161         // Repeat for all inputs
162         ENV_BRANCH_TB = 32'hffff_ffff;
163         ENV_MUX_SELECT_TB = 2;
164         #10
165         if (ENV_PC_COUNT_INC_TB != 32'h0000_0003 || ENV_PC_COUNT_TB != 32'hffff_ffff) begin
166             $display("BRANCH Max Read Failed");
167             pass = 0;
168         end
169 
170         ENV_JAL_TB = 32'hffff_ffff;
171         ENV_MUX_SELECT_TB = 3;
172         #10
173         if (ENV_PC_COUNT_INC_TB != 32'h0000_0003 || ENV_PC_COUNT_TB != 32'hffff_ffff) begin
174             $display("JAL Max Read Failed");
175             pass = 0;
176         end
177 
178         ENV_MTVEC_TB = 32'hffff_ffff;
179         ENV_MUX_SELECT_TB = 4;
180         #10
181         if (ENV_PC_COUNT_INC_TB != 32'h0000_0003 || ENV_PC_COUNT_TB != 32'hffff_ffff) begin
182             $display("MTVEC Max Read Failed");
183             pass = 0;
184         end
185 
186         ENV_MEPC_TB = 32'hffff_ffff;
187         ENV_MUX_SELECT_TB = 5;
188         #10
189         if (ENV_PC_COUNT_INC_TB != 32'h0000_0003 || ENV_PC_COUNT_TB != 32'hffff_ffff) begin
190             $display("MEPC Max Read Failed");
191             pass = 0;
```

```
192         end
193
194         // Determine if the test passed or failed
195         if (pass == 1) begin
196             $display("MaxMux Test Passed");
197         end else begin
198             $display("MaxMux Test Failed");
199         end
200     endtask // maxMux
201
202
203     // Verify the program counter works properly with 32'h0000_0000 values
204     task MinMux();
205         // Load the program counter with the min 32bit value
206         ENV_JALR_TB = 32'h0000_0000;
207         ENV_MUX_SELECT_TB = 1;
208         #10
209         // Verify the program counter incremented outputs are correct and should lead to
210         // another valid address
211         if (ENV_PC_COUNT_INC_TB != 32'h0000_0004 || ENV_PC_COUNT_TB != 32'h0000_0000) begin
212             $display("JALR Min Read Failed");
213             pass = 0;
214         end
215
216         // Repeat for all inputs
217         ENV_BRANCH_TB = 32'h0000_0000;
218         ENV_MUX_SELECT_TB = 2;
219         #10
220         if (ENV_PC_COUNT_INC_TB != 32'h0000_0004 || ENV_PC_COUNT_TB != 32'h0000_0000) begin
221             $display("BRANCH Min Read Failed");
222             pass = 0;
223         end
224
225         ENV_JAL_TB = 32'h0000_0000;
226         ENV_MUX_SELECT_TB = 3;
227         #10
228         if (ENV_PC_COUNT_INC_TB != 32'h0000_0004 || ENV_PC_COUNT_TB != 32'h0000_0000) begin
229             $display("JAL Min Read Failed");
230             pass = 0;
231         end
232
233         ENV_MTVEC_TB = 32'h0000_0000;
234         ENV_MUX_SELECT_TB = 4;
235         #10
236         if (ENV_PC_COUNT_INC_TB != 32'h0000_0004 || ENV_PC_COUNT_TB != 32'h0000_0000) begin
237             $display("MTVEC Min Read Failed");
238             pass = 0;
239         end
240
241         ENV_MEPC_TB = 32'h0000_0000;
242         ENV_MUX_SELECT_TB = 5;
243         #10
244         if (ENV_PC_COUNT_INC_TB != 32'h0000_0004 || ENV_PC_COUNT_TB != 32'h0000_0000) begin
245             $display("MEPC Min Read Failed");
246             pass = 0;
247         end
248
249         // Determine if the test passed or failed
250         if (pass == 1) begin
251             $display("MinMux Test Passed");
252         end else begin
253             $display("MinMux Test Failed");
254         end
255
256     endtask // MinMux
257
258
259     // Verify the program counter works properly with reset
260     task ResetTest();
261         // Send a reset signal
```

```
262         ENV_RST_TB = 1;
263         #10
264         ENV_RST_TB = 0;
265 
266         // Verify the program counter was reset to 0
267         if (ENV_PC_COUNT_INC_TB != 32'h0000_0004 || ENV_PC_COUNT_TB != 32'h0000_0000) begin
268             $display("Reset Test Failed");
269             pass = 0;
270         end
271 
272         // Determine if the test passed or failed
273         if (pass == 1) begin
274             $display("Reset Test Passed");
275         end else begin
276             $display("Reset Test Failed");
277         end
278 
279     endtask // ResetTest
280 
281     task RandNum();
282         // Load a random number into the program counter
283         int rand_num;
284         rand_num = $random;
285         ENV_JALR_TB = rand_num;
286         ENV_MUX_SELECT_TB = 1;
287         #10
288         // Verify the random number was loaded into the program counter
289         if (ENV_PC_COUNT_INC_TB != rand_num + 4 || ENV_PC_COUNT_TB != rand_num) begin
290             $display("JALR Rand Read Failed");
291             pass = 0;
292         end
293 
294         // Repeat for all inputs
295         rand_num = $random;
296         ENV_BRANCH_TB = rand_num;
297         ENV_MUX_SELECT_TB = 2;
298         #10
299         if (ENV_PC_COUNT_INC_TB != rand_num + 4 || ENV_PC_COUNT_TB != rand_num) begin
300             $display("BRANCH Rand Read Failed");
301             pass = 0;
302         end
303 
304         rand_num = $random;
305         ENV_JAL_TB = rand_num;
306         ENV_MUX_SELECT_TB = 3;
307         #10
308         if (ENV_PC_COUNT_INC_TB != rand_num + 4 || ENV_PC_COUNT_TB != rand_num) begin
309             $display("JAL Rand Read Failed");
310             pass = 0;
311         end
312 
313         rand_num = $random;
314         ENV_MTVEC_TB = rand_num;
315         ENV_MUX_SELECT_TB = 4;
316         #10
317         if (ENV_PC_COUNT_INC_TB != rand_num + 4 || ENV_PC_COUNT_TB != rand_num) begin
318             $display("MTVEC Rand Read Failed");
319             pass = 0;
320         end
321 
322         rand_num = $random;
323         ENV_MEPC_TB = rand_num;
324         ENV_MUX_SELECT_TB = 5;
325         #10
326         if (ENV_PC_COUNT_INC_TB != rand_num + 4 || ENV_PC_COUNT_TB != rand_num) begin
327             $display("MEPC Rand Read Failed");
328             pass = 0;
329         end
330 
331         if (pass == 1) begin
```

```
            $display("RandNum Test Passed");
        end else begin
            $display("RandNum Test Failed");
        end
    endtask // RandNum

    task WriteEnable();
        // Load a random number into the program counter
        int rand_num;
        rand_num = $random;
        ENV_JALR_TB = rand_num; // Load the random number into the JALR
        ENV_MUX_SELECT_TB = 1;
        ENV_WE_TB = 1;          // Enable the write
        #10
        // Verify the random number was loaded into the program counter
        if (ENV_PC_COUNT_INC_TB != rand_num + 4 || ENV_PC_COUNT_TB != rand_num) begin
            $display("JALR Write Failed");
            pass = 0;
        end

        ENV_WE_TB = 0;          // Disable the write
        ENV_JALR_TB = 0;        // Reset the JALR
        #10
        // Verify the 0 was not loaded into the program counter
        if (ENV_PC_COUNT_INC_TB != rand_num + 4 || ENV_PC_COUNT_TB != rand_num) begin
            $display("JALR Write Disable Failed");
            pass = 0;
        end

        if (pass == 1) begin
            $display("WriteEnable Test Passed");
        end else begin
            $display("WriteEnable Test Failed");
        end
    endtask // WriteEnable


    // Reset the test
    task reset();
        ENV_RST_TB = 1;
        #10
        ENV_RST_TB = 0;
    endtask // reset

    // Toggle Clock
    always #5 ENV_CLK_TB = ~ENV_CLK_TB;

endmodule
```

4.3 Testbench Output

Here is the output from the console after running this testbench.

Listing 2: TCl Output from ProgramCounterEnv_TB

```
    Setup Complete
    ReadMux Test Passed
    MaxMux Test Passed
    MinMux Test Passed
    Reset Test Passed
    RandNum Test Passed
    WriteEnable Test Passed
    All Tests Passed
    $finish called at time : 300 ns
```

Running this testbench results in this simulation wavefrom.

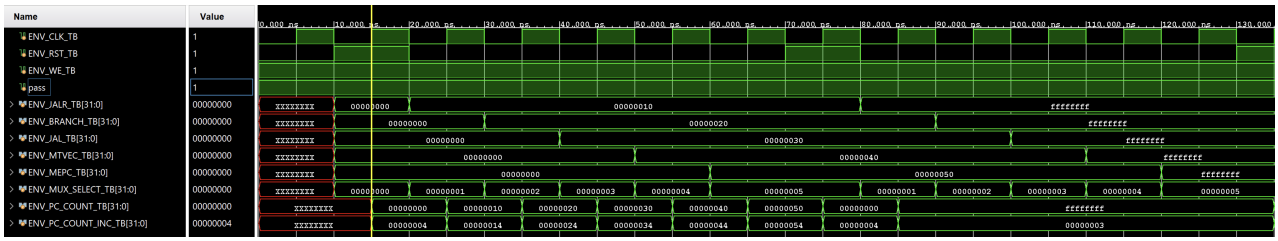

Figure 5: Program Counter Simulation 0ns - 130ns

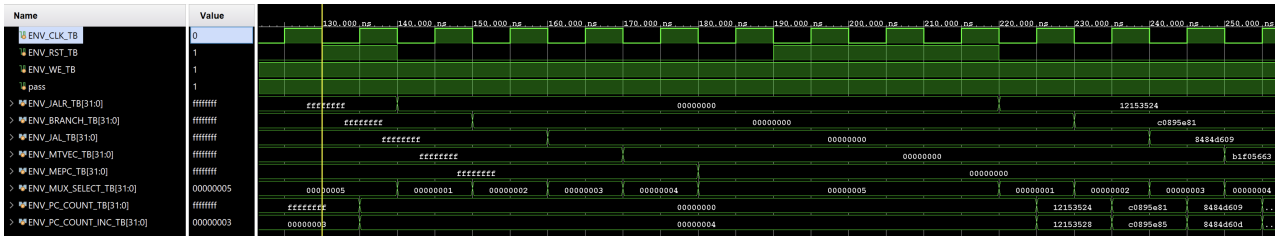

Figure 6: Program Counter Simulation 130ns - 250ns

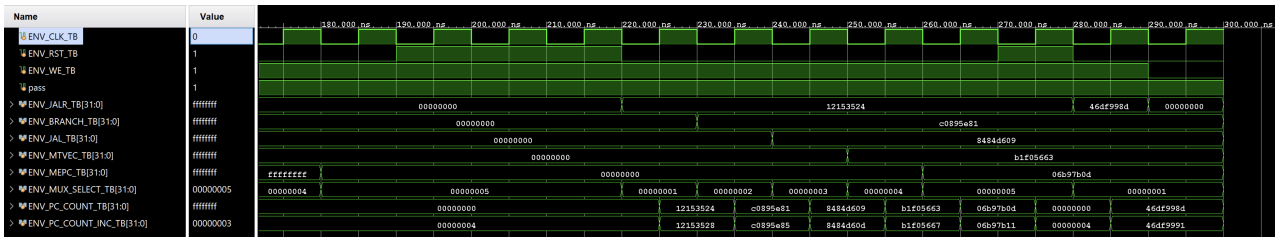

Figure 7: Program Counter Simulation 180ns - 300ns

5 Source Code

5.1 Program Counter

Listing 3: Verilog Code for Program Counter

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 01/19/2024 03:52:57 PM
// Module Name: ProgramCounter
// Project Name: Program Counter
// Target Devices: Basys3
// Description: Take in a 32 bit value and store it in a register while
//              incrementing it by 4.
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////


module ProgramCounter(
    // Inputs
    input PC_RST,                       // Reset
    input PC_WE,                        // Write Enable
    input [31:0] PC_DIN,                // Data In
    input CLK,                          // Clock

    // Outputs
    output logic [31:0] PC_COUNT,       // Data Out
    output logic [31:0] PC_COUNT_INC    // Data Out Incremented by 4
    );

    always_ff @( posedge CLK ) begin
        if (PC_RST) begin
            PC_COUNT <= 32’h0000_0000;  // Reset the Program Counter to 0
            PC_COUNT_INC <= 32’h0000_0004;  // Reset the Program Counter to 4
        end else if (PC_WE) begin
            PC_COUNT <= PC_DIN;         // Write the Data In to the Program Counter
            PC_COUNT_INC <= PC_DIN + 4; // Write the Data In to the Program Counter and increment by
                4
        end
        else
            PC_COUNT <= PC_COUNT;       // Do nothing and keep the Program Counter the same
    end

endmodule
```

5.2 Program Counter Multiplexer

Listing 4: Verilog Code for Program Counter

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 01/19/2024 03:52:57 PM
// Module Name: ProgramCounterMux
// Project Name: Program Counter
// Target Devices: Basys3
// Description: Determine which signal to use to drive the program counter taking
//              in multiple signals and outputting one.
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////

module ProgramCounterMux(
    // Inputs
    input [31:0] PC_COUNT_INCD,     // Program Counter Incremented by 4
    input [31:0] JALR,              // Jump and Link Register
    input [31:0] BRANCH,            // Branch jump
    input [31:0] JAL,               // Jump and Link
    input [31:0] MTVEC,             // Machine Trap Vector
    input [31:0] MEPC,              // Machine Exception Program Counter
    input [2:0] MUX_SELECT,        // Select which signal to output

    // Outputs
    output logic [31:0] PC_MUX_OUT
);

    // Begin Multiplexer Code Block
    always_comb begin
        case (MUX_SELECT)
            0: PC_MUX_OUT = PC_COUNT_INCD;
            1: PC_MUX_OUT = JALR;
            2: PC_MUX_OUT = BRANCH;
            3: PC_MUX_OUT = JAL;
            4: PC_MUX_OUT = MTVEC;
            5: PC_MUX_OUT = MEPC;
            6: PC_MUX_OUT = PC_COUNT_INCD;          // Default to PC_COUNT_INCD
            7: PC_MUX_OUT = PC_COUNT_INCD;          // Default to PC_COUNT_INCD
            default: PC_MUX_OUT = PC_COUNT_INCD;    // Default to PC_COUNT_INCD
        endcase
    end

endmodule
```

5.3 Program Counter Environment

Listing 5: Verilog Code for Program Counter

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Ethan Vosburg
//
// Create Date: 01/19/2024 03:52:57 PM
// Module Name: ProgramCounterEnv
// Project Name: Program Counter
// Target Devices: Basys3
// Description: Testing envoriment for the prorgram counter allowing for the
//              multiplexer and program counter to be tested together.
//
// Revision:
// Revision 0.01 - File Created
//
//////////////////////////////////////////////////////////////////////////////////

module ProgramCounterEnv(
    // Inputs
    input ENV_RST,                          // Reset
    input ENV_WE,                           // Write Enable
    input [31:0] ENV_JALR,                  // Jump and Link Register
    input [31:0] ENV_BRANCH,                // Branch jump
    input [31:0] ENV_JAL,                   // Jump and Link
    input [31:0] ENV_MTVEC,                 // Machine Trap Vector
    input [31:0] ENV_MEPC,                  // Machine Exception Program Counter
    input [2:0]  ENV_MUX_SELECT,            // Select which signal to output
    input ENV_CLK,                          // Clock

    // Outputs
    output logic [31:0] ENV_PC_COUNT,       // Data Out
    output logic [31:0] ENV_PC_COUNT_INC    // Data Out Incremented by 4
);
    // Logic
    logic [31:0] mux_pc;                    // Multiplexer connection to PC

    // Instantiate the Program Counter
    ProgramCounter PC(
        .PC_RST(ENV_RST),
        .PC_WE(ENV_WE),
        .PC_DIN(mux_pc),
        .CLK(ENV_CLK),
        .PC_COUNT(ENV_PC_COUNT),
        .PC_COUNT_INC(ENV_PC_COUNT_INC)
    );

    // Instantiate the Program Counter Multiplexer
    ProgramCounterMux PC_MUX(
        .PC_COUNT_INCD(ENV_PC_COUNT_INC),
        .JALR(ENV_JALR),
        .BRANCH(ENV_BRANCH),
        .JAL(ENV_JAL),
        .MTVEC(ENV_MTVEC),
        .MEPC(ENV_MEPC),
        .MUX_SELECT(ENV_MUX_SELECT),
        .PC_MUX_OUT(mux_pc)
    );

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.