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

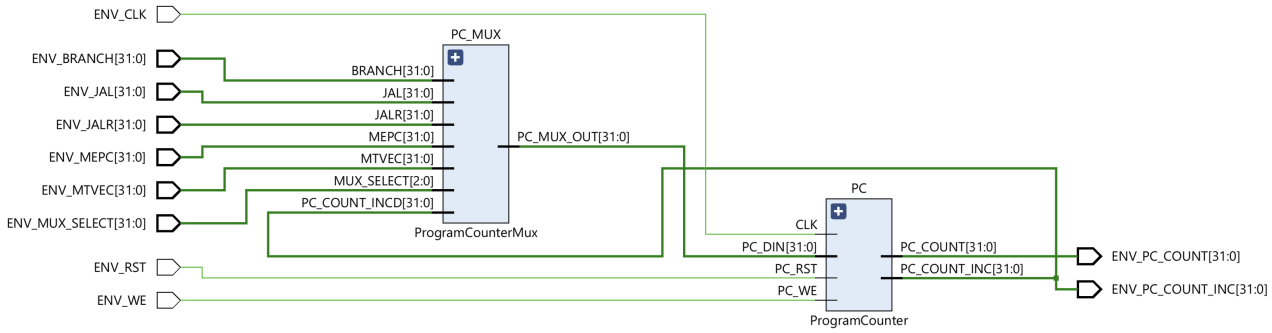

Figure 1: Program Counter Elaborated Design

2.2 Program Counter Multiplexer Elaborated Design

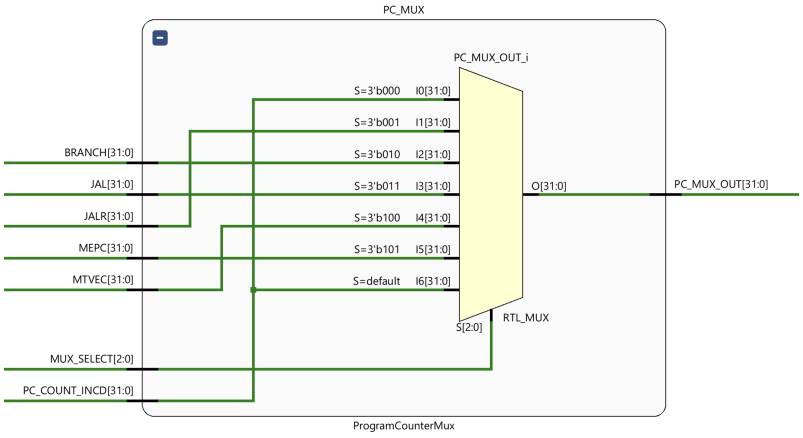

Figure 2: Program Counter Multiplexer Elaborated Design

2.3 Program Counter Main Hardware Elaborated Design

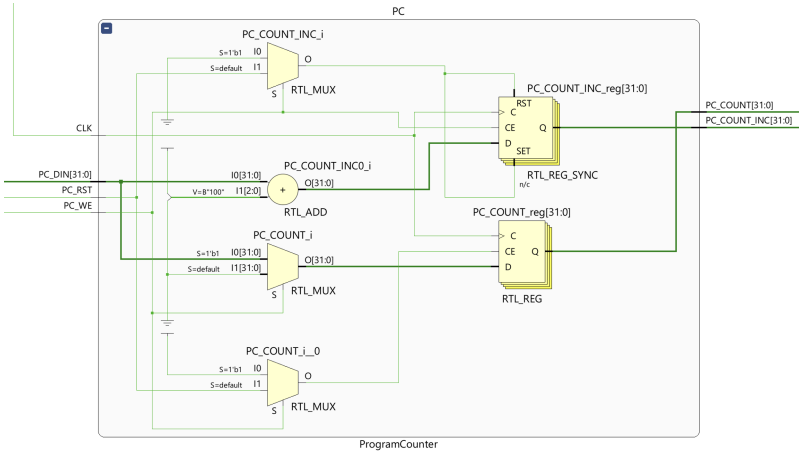

Figure 3: Program Counter Main Hardware Elaborated Design

3 Synthesis Warnings Listing

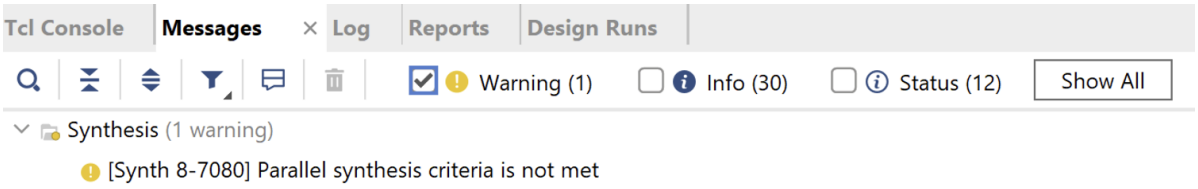

Figure 4: Synthesis Warnings

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

```
1
2  \subsection{Testbench Output} % Third level section
3  Here is the output from the console after running this testbench.
4
5  \begin{lstlisting}[language=Verilog, caption=TCl Output from ProgramCounterEnv\_TB]
6      Setup Complete
7      ReadMux Test Passed
8      MaxMux Test Passed
9      MinMux Test Passed
10     Reset Test Passed
11     RandNum Test Passed
12     WriteEnable Test Passed
13     All Tests Passed
14     $finish called at time : 300 ns
```

Running this testbench results in this simulation wavefrom.

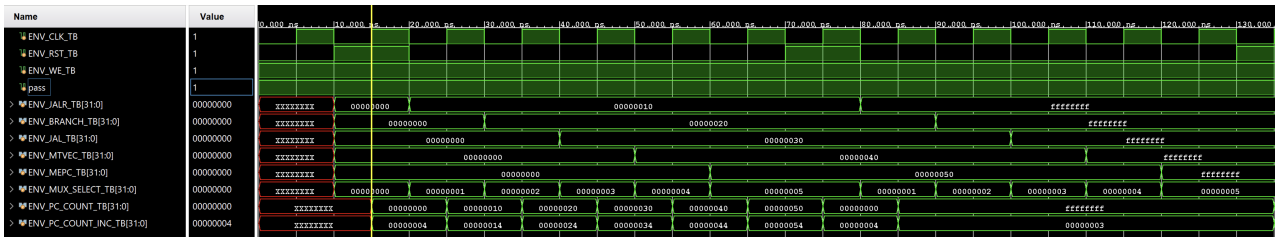

Figure 5: Program Counter Simulation 0ns - 130ns

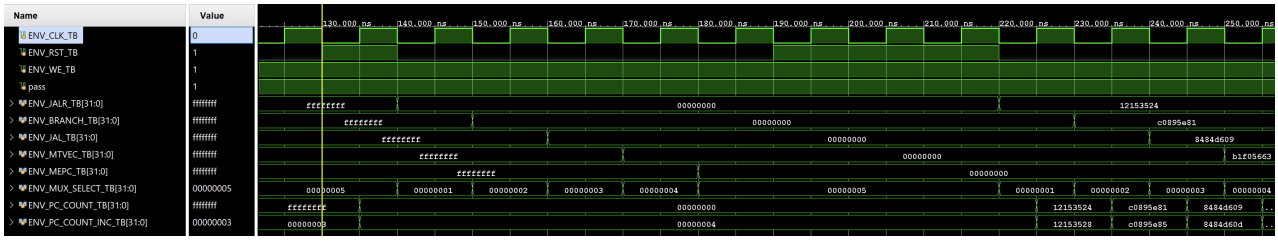

Figure 6: Program Counter Simulation 130ns - 250ns

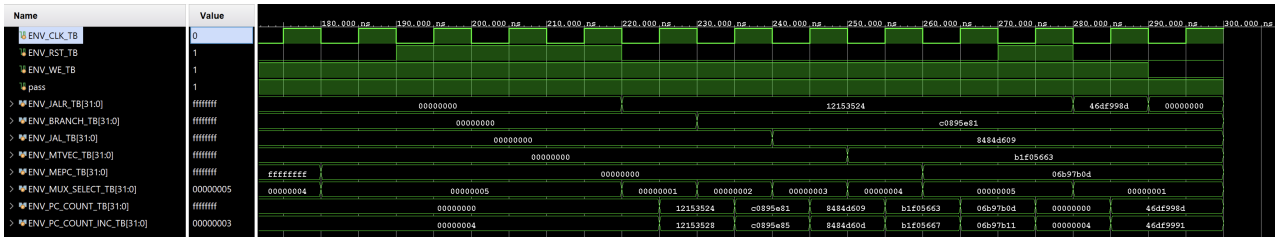

Figure 7: Program Counter Simulation 180ns - 300ns

5 Source Code

5.1 Program Counter

Listing 2: 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 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: 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 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: 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.