Lab Assignment 4

Due: 19 Oct 2020

By: Alex Oswald
EECE 2323

Introduction

During this lab experiment, sequential logic was explored by way of designing and building the processor path connecting an ALU, a zero register, and a register file. The connection of these three components in sequential logic can be boiled down to logic that holds a state and is timed with a clock signal. The following are descriptions of the code for the ALU-RegFile connection as well as the testbench that supported it, and the waveform generated by the testbench. Following, the components were tested in tandem on a Xilinx PYNQ-Z2 FPGA board with inputs and outputs operated via Virtual IO pins as included in the hardware test below.

Verilog Code for `alu_regfile.v`:

```
1.  `timescale 1ns / 1ps
2.
3.  module alu_regfile(
4.      output [8:0] ReadData1,
5.      output [8:0] ReadData2,
6.
7.      input ALUsrc1,
8.      input ALUsrc2,
9.      input [2:0] ALUop,
10.     input [7:0] Instr_i,
11.
12.     input rst,
13.     input clk,
14.     input wr_en,
15.     input [1:0] rd0_addr,
16.     input [1:0] rd1_addr,
17.     input [1:0] wr_addr,
18.     input [8:0] wr_data,
19.
20.     output [7:0] input_1,
21.     output [7:0] input_2,
22.     output [7:0] result,
23.     output ovf,
24.     output take_branch);
25.
26.     reg_file reg_file(
27.         .rst(rst),
28.         .clk(clk),
29.         .wr_en(wr_en),
30.         .rd0_addr(rd0_addr),
31.         .rd1_addr(rd1_addr),
32.         .wr_addr(wr_addr),
33.         .wr_data(wr_data),
34.         .rd0_data(ReadData1),
35.         .rd1_data(ReadData2));
36.
37.     // Initialize ALU & properly link comopnents
38.     eightbit_alu eightbit_alu(
39.         .a(input_1),
40.         .b(input_2),
41.         .sel(ALUop),
42.         .f(result),
43.         .ovf(ovf),
44.         .take_branch(take_branch));
45.
46.     assign input_1 = ALUsrc1 ? 8'b00000000 : ReadData1;
```

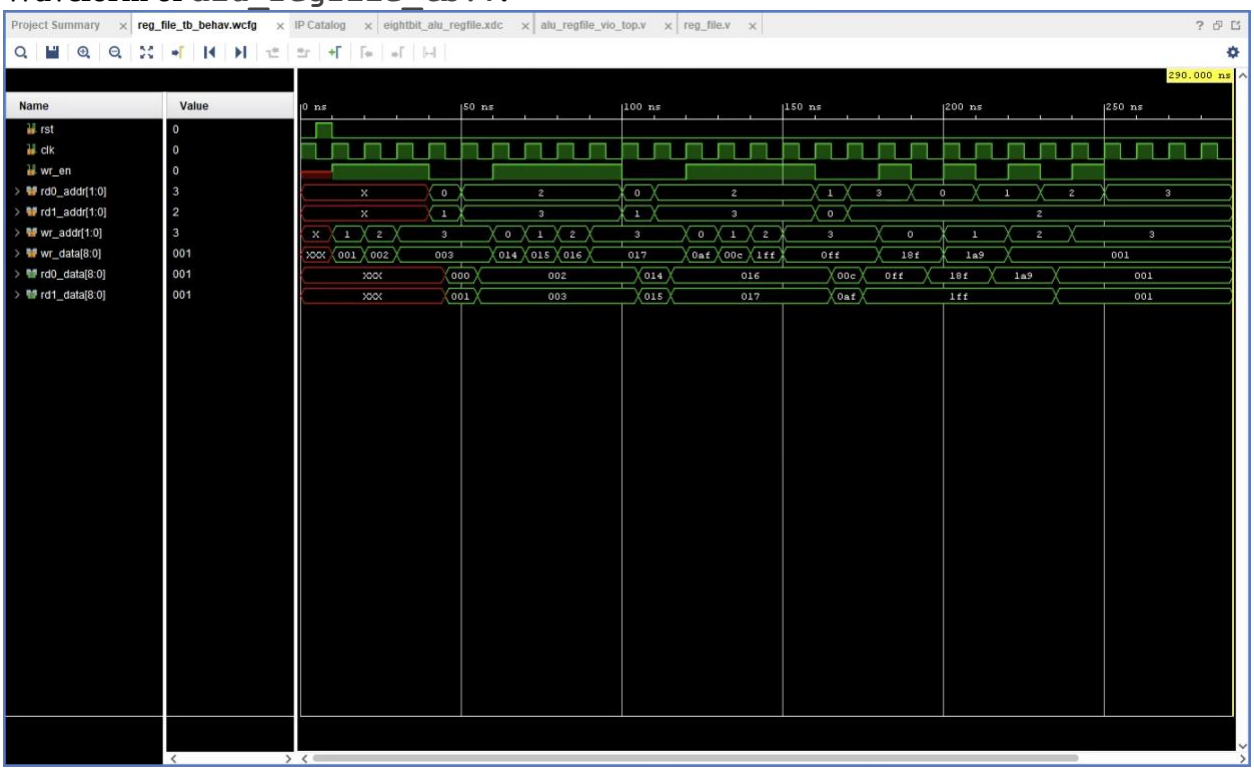
```
47.     assign input_2 = ALUsrc2 ? Instr_i : ReadData2;
48.
49. endmodule
```

Verilog Code for `alu_regfile_tb.v`:

```
1.  `timescale 1ns / 1ps
2.
3.  module alu_reg_file_tb();
4.
5.      reg rst, clk, wr_en, ALUsrc1, ALUsrc2;
6.      reg [1:0] rd0_addr, rd1_addr, wr_addr;
7.      reg [2:0] ALUop;
8.      reg [7:0] Instr_i;
9.      reg [8:0] wr_data;
10.     wire ovf, take_branch;
11.     wire [7:0] result, input_1, input_2;
12.     wire [8:0] rd0_data, rd1_data, ReadData1, ReadData2;
13.
14.
15.     alu_regfile uut(
16.         .ReadData1(ReadData1),
17.         .ReadData2(ReadData2),
18.         .ALUsrc1(ALUsrc1),
19.         .ALUsrc2(ALUsrc2),
20.         .ALUop(ALUop),
21.         .Instr_i(Instr_i), // BREAK
22.
23.         .rst(rst),
24.         .clk(clk),
25.         .wr_en(wr_en),
26.         .rd0_addr(rd0_addr),
27.         .rd1_addr(rd1_addr),
28.         .wr_addr(wr_addr),
29.         .wr_data(wr_data), // BREAK
30.
31.         .input_1(input_1),
32.         .input_2(input_2),
33.         .result(result),
34.         .ovf(ovf),
35.         .take_branch(take_branch));
36.
37.     reg_file reg_file_tb(rst,clk,wr_en,rd0_addr,rd1_addr,wr_addr,wr_data,rd0_data,rd1_d
    ata);
38.
39.     // 1) Initial pulse for rst (reset)
40.     initial
41.     begin
42.         #0  rst = 0;
43.         #5  rst = 1;
44.         #5  rst = 0;  // semicolon after #10 ???
45.     end
46.
47.     // 2) Clock signal
48.     initial
49.     begin
50.         #0 clk  = 1;
51.         forever #5  clk = ~clk;
52.     end
```

```
53.
54.     // 3) All Other Tests
55.     initial
56.     begin
57.
58.         ALUsrc1=0; ALUsrc2=0;  // ReadData1 && ReadData2
59.          ALUop = 3'b000; // a + b;
60.         #10 wr_en = 1;  wr_addr = 1;    wr_data = 9'd1;
61.         #10 wr_en = 1;  wr_addr = 2;    wr_data = 9'd2;
62.         #10 wr_en = 1;  wr_addr = 3;    wr_data = 9'd3;
63.         #10 wr_en = 0;  rd0_addr = 0;   rd1_addr = 1;
64.         #10 wr_en = 0;  rd0_addr = 2;   rd1_addr = 3;
65.
66.          ALUop = 3'b001; // ~b;
67.         #10 wr_en = 1;  wr_addr = 0;    wr_data = 9'd20;
68.         #10 wr_en = 1;  wr_addr = 1;    wr_data = 9'd21;
69.         #10 wr_en = 1;  wr_addr = 2;    wr_data = 9'd22;
70.         #10 wr_en = 1;  wr_addr = 3;    wr_data = 9'd23;
71.         #10 wr_en = 0;  rd0_addr = 0;   rd1_addr = 1;
72.         #10 wr_en = 0;  rd0_addr = 2;   rd1_addr = 3;
73.
74.          ALUop = 3'b010; // a & b;
75.         #10 wr_en = 1;  wr_addr = 0;    wr_data = 9'd175;
76.         #10 wr_en = 1;  wr_addr = 1;    wr_data = 9'd12;
77.         #10 wr_en = 1;  wr_addr = 2;    wr_data = 9'd511;
78.         #10 wr_en = 1;  wr_addr = 3;    wr_data = 9'd255;
79.         #10 wr_en = 0;  rd0_addr = 1;   rd1_addr = 0;
80.         #10 wr_en = 0;  rd0_addr = 3;   rd1_addr = 2;
81.
82.         ALUsrc1=0; ALUsrc2=1; // ReadData1 && Instr_i
83.          ALUop = 3'b011; // a | b;
84.         #10 wr_en = 1;  wr_addr = 0;    wr_data = 9'd399;
85.         #10 wr_en = 0;  rd0_addr = 0;   Instr_i = 8'd42;
86.
87.          ALUop = 3'b100; // a >>> 1;
88.         #10 wr_en = 1;  wr_addr = 1;    wr_data = 9'd425;
89.         #10 wr_en = 0;  rd0_addr = 1;   Instr_i = 8'd200;
90.
91.          ALUop = 3'b101; // a << 1;
92.         #10 wr_en = 1;  wr_addr = 2;    wr_data = 9'd1;
93.         #10 wr_en = 0;  rd0_addr = 2;   Instr_i = 8'd23;
94.
95.          ALUop = 3'b110; // 0;
96.         #10 wr_en = 1;  wr_addr = 3;    wr_data = 9'd1;
97.         #10 wr_en = 0;  rd0_addr = 3;   Instr_i = 8'd23;
98.
99.         #10 ALUsrc1=1; ALUsrc2=0; // zero_register && ReadData2
100.                 ALUop = 3'b110; // 0;
101.                #10 Instr_i = 8'd255;
102.
103.                 ALUop = 3'b111; // 0;
104.                #10 Instr_i = 8'd255;
105.
106.                #10 $finish;
107.
108.            end
109.
110.        endmodule
```

Waveform of `alu_regfile_tb.v`:



Hardware tests for VIO:

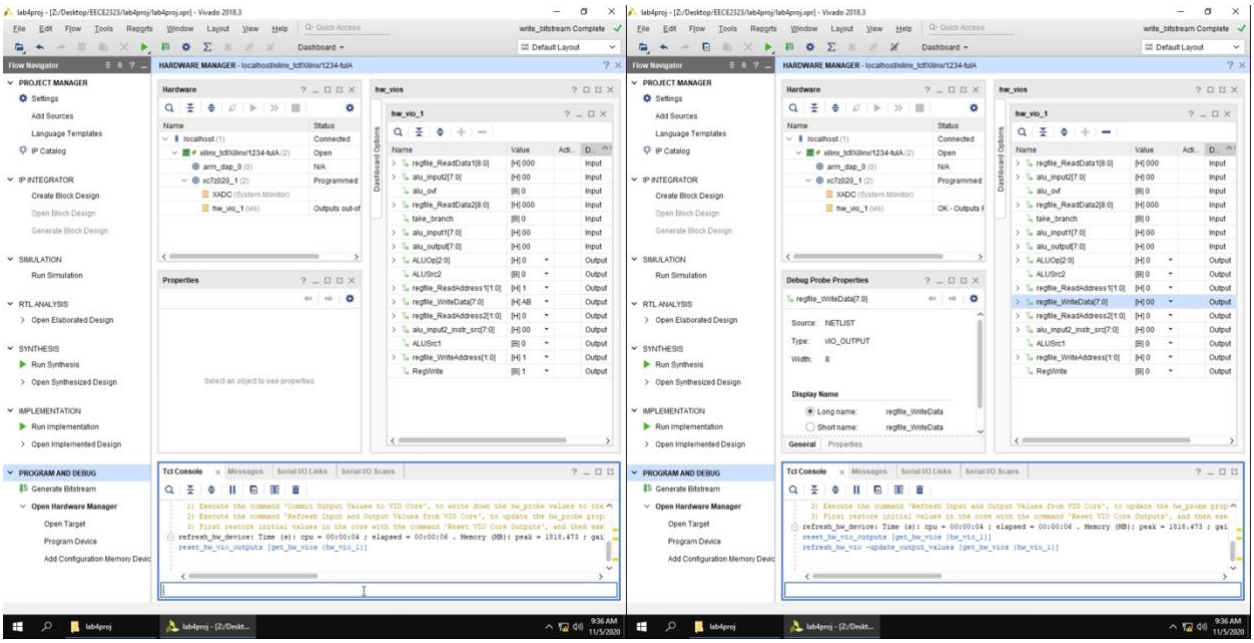


Figure 1: Reset & Refresh VIO.

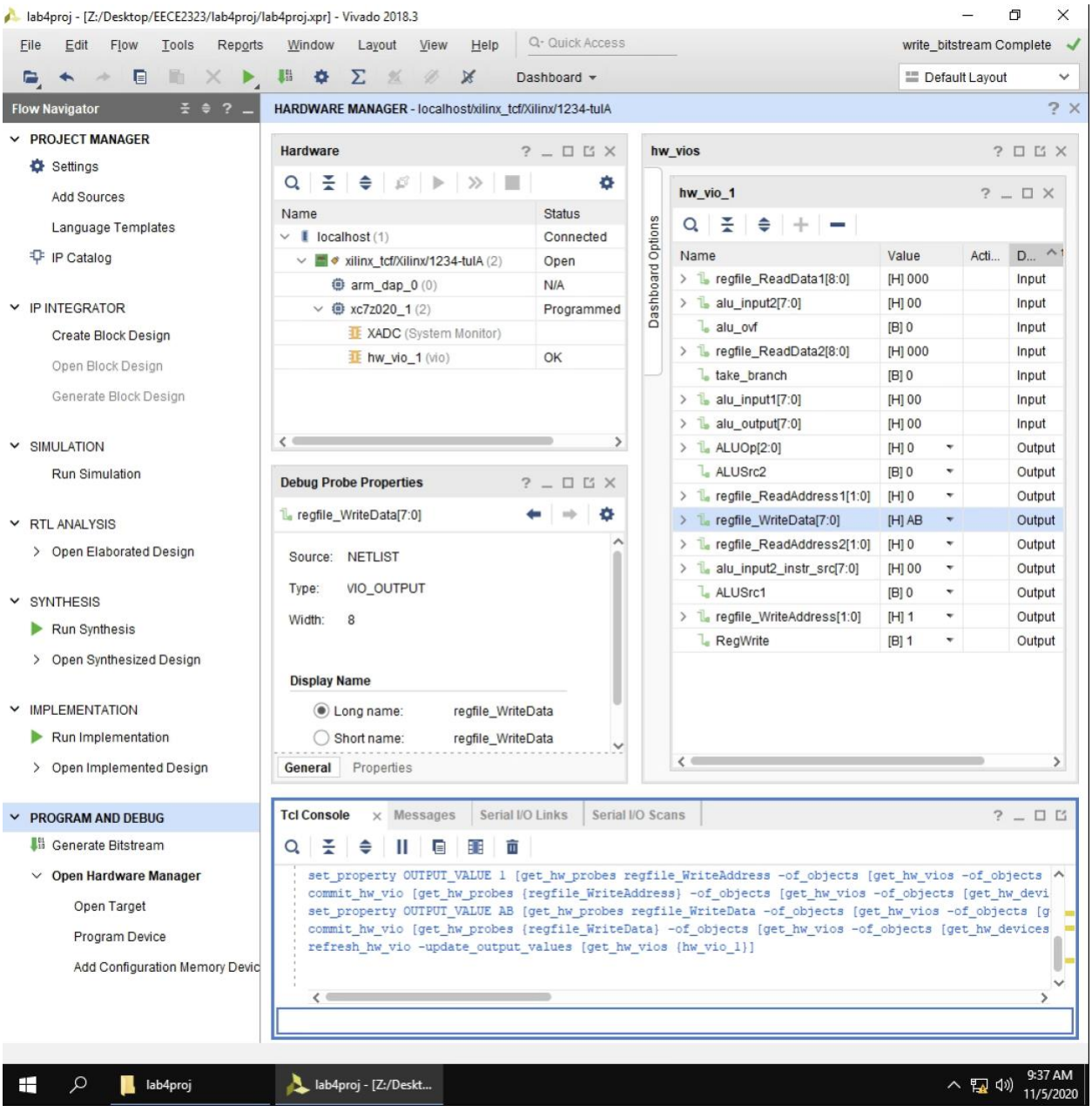


Figure 2: Write 8'hAB to reg 1.

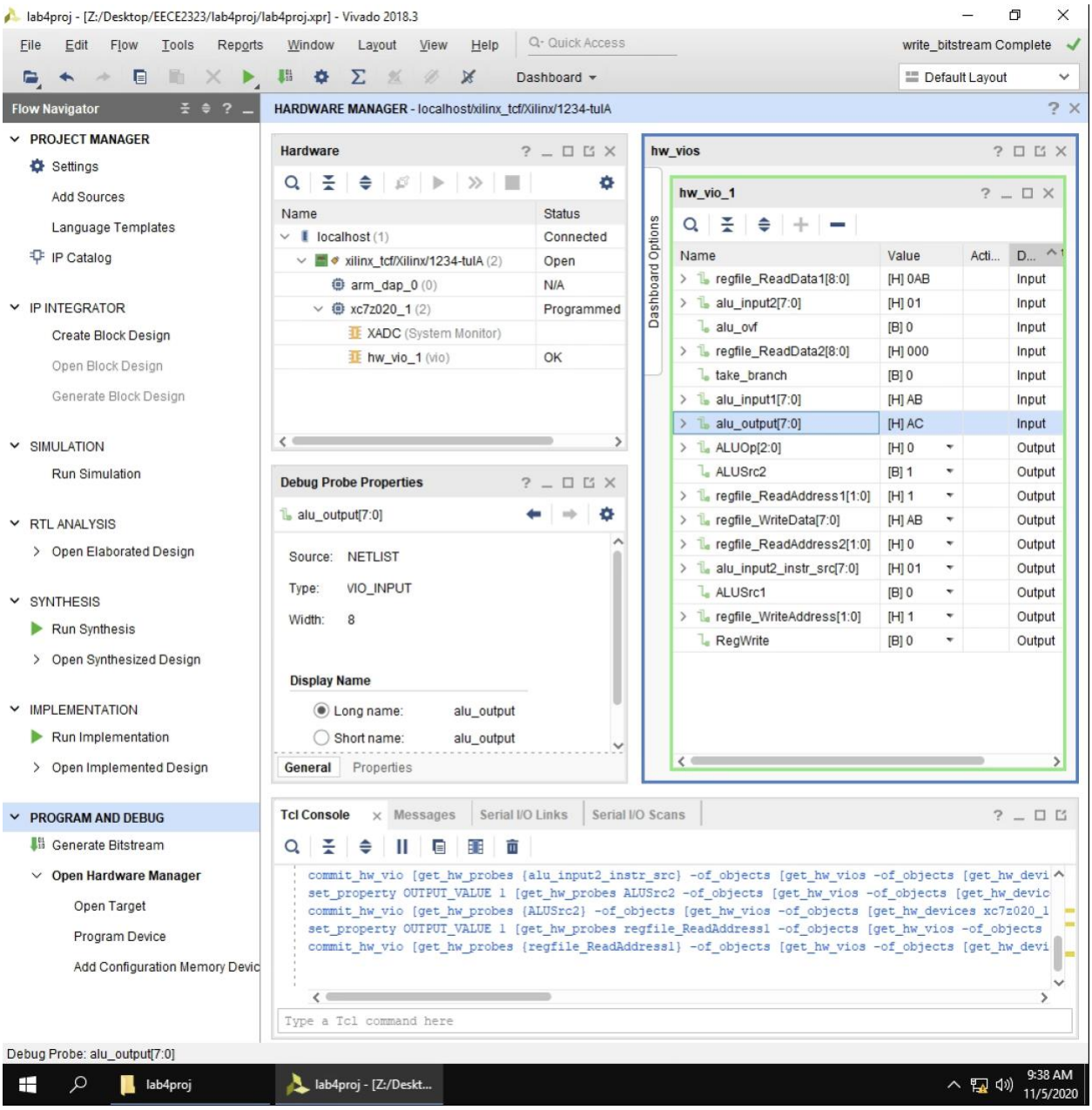


Figure 3: Read reg 1. Do 8'hAB+1. (1 is from the immediate value)

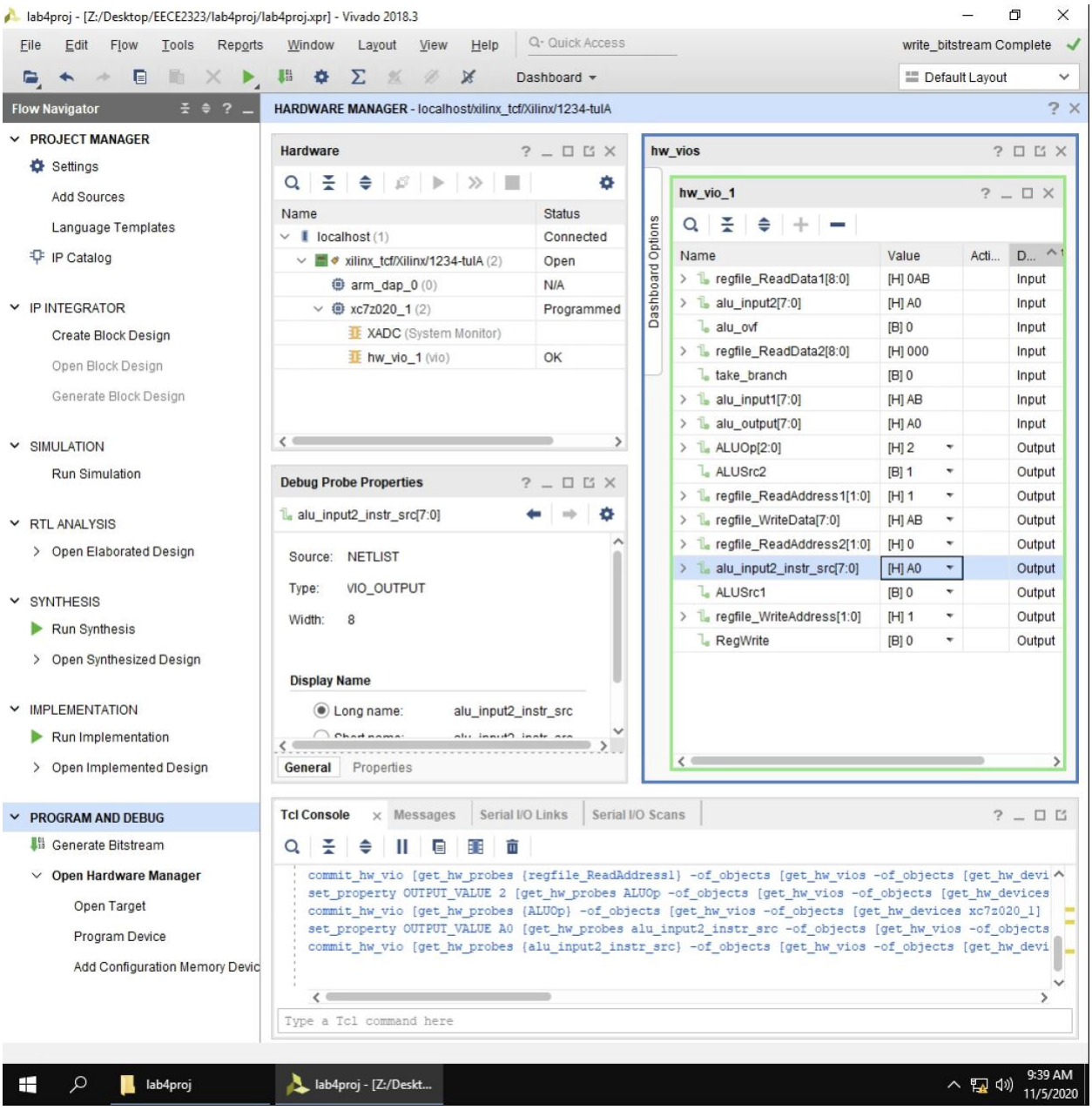


Figure 4: AND 8'hAB with 8'hA0. (8'hA0 is from immediate value)

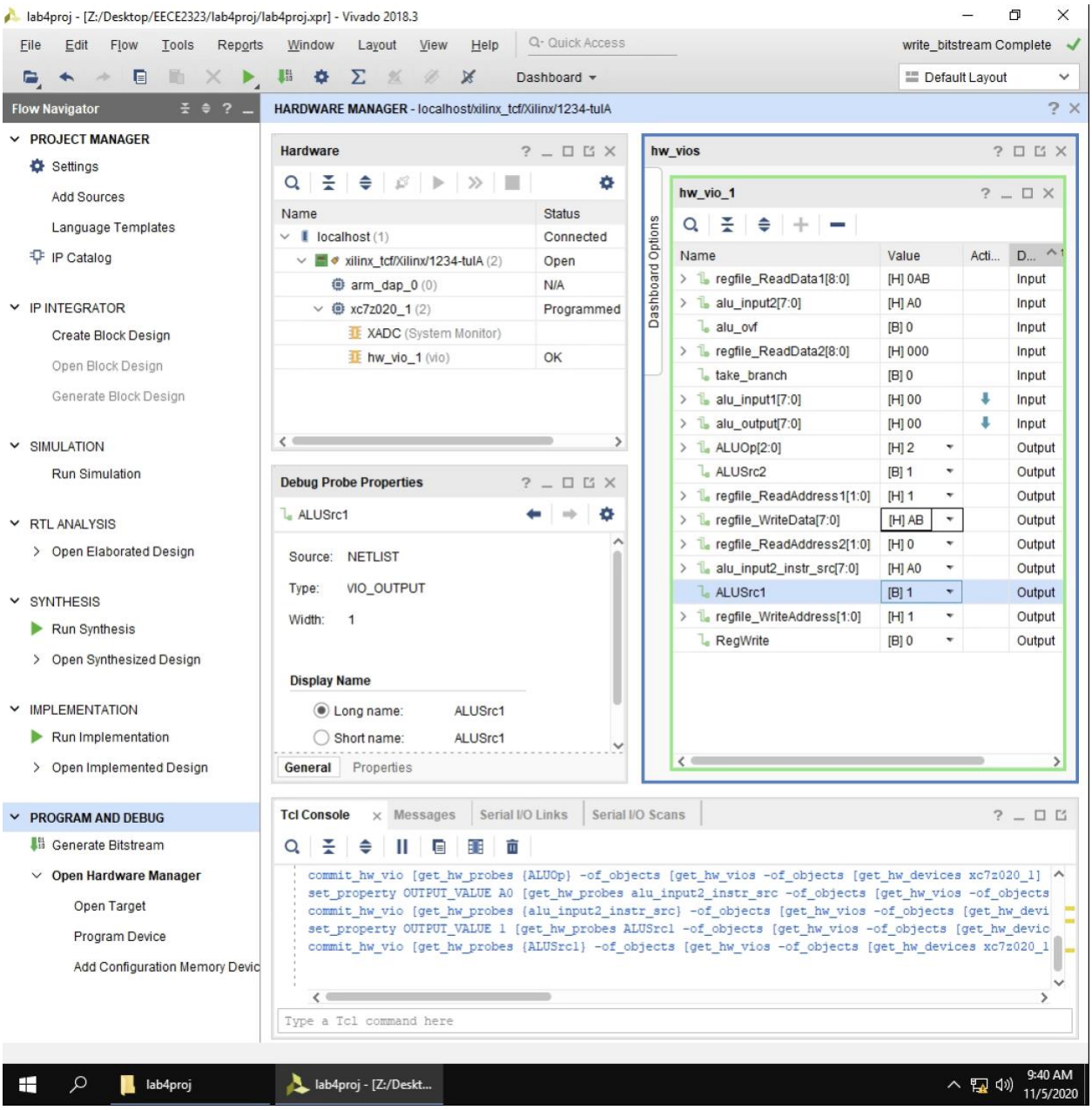


Figure 5: AND 8'hAB with zero register. (clear reg 1)

Conclusion

During this lab assignment, sequential logic was explored by way of designing and building the processor path connecting an ALU, a zero register, and a register file. This assignment demonstrated some of the functions of one of the central parts of a MIPS 8-bit processor. These components were designed with a clock cycle at the heart of its operations, and the individual components were connected together through initializing each of the component files and linking the outputs one with the inputs of another to model the wires between them. To extend academic inquiry, one might examine how to construct data memory using Verilog.