Report

on

COEN 316 Laboratory Experiment #4

**CPU Datapath/Control Unit and Testing**

Submitted to

*SEYED AMIRREZA MOUSAVI*

*Instructor’s name*

Date Performed: November 24th, 2020

Date Submitted: December 8, 2020

By

Jasen Ratnam 40094237

Name student ID

Lab section: DI-X

“I certify that this submission is my original work and meets the Faculty's Expectations of Originality”



40094237 28/11/2020

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# **Objective:**

In this lab, we will use digital logic simulation and synthesis using Modelsim, and Xilinx ISE (Vivado) to become acquainted with the VHDL simulation software tool and FPGA implementation software tools. During this lab, design and model a CPU datapath using VHDL and all the cpu components we have done in the previous labs. All components are put together along with some new components, a control unit and a pipeline, to create a fully functional CPU.

**Introduction:**

The lab consisted of designing a complete CPU using VHDL using different CPU units created in previous labs. We simulate the created processor and test possible operations that may happen on it, using the instructions i cache. The previous components include an ALU, next-address unit, register file unit, and a datapath unit. In this lab, we need to create a pipeline and control unit to join them.

In normal operation, the program counter of the CPU is updated to point to the next instruction after every clock cycle. To point at the next instruction, it’s address is needed that can be the following address of the current one or it can be pointed to a completely different part of the memory by using a jump or branch instruction. Our design needs to be able to calculate the address of the next instruction for every case.

**Part 1:**

In the first part of the lab, we are tasked to build a datapath to join all the CPU components we have created in the previous labs, along with some new components. We are given a partial datapath of the CPU and a detailed textual explanation of how the datapath should function. With this information we created the following complete datapath design. The different colours are used solely to differients the overlapping lines.

****

**Figure 1: CPU datapath.**

With this design, we begin creating the additional components needed in vhdl and then create a datapath code to put all the components together following the design above.

**Part 2:**

In this part of the lab, we are tasked with creating a control unit to control the datapath of the CPU created above based on the instruction in the I\_Chache. We are given 20 instructions to implement in the CPU. The instructions have a 6 bit opcode, a 6 bit func, and 10 control signals to control the flow of data in the datapath of the CPU. We are given a partially completed table with the 20 instructions and their control signals, we completed the table by deriving the values of the 10 control signals using the tables given that give the conditions of each signal to be enabled, The completed table is given below.

**Table 1: 20 instructions with opcode and function fields and control signals.**

| Inst. | op | func | reg\_write | reg\_dst | reg\_in\_src | alu\_src | add\_sub | data\_write | logic\_func | func | branch\_type | pc\_sel |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lui | 001111 | 001111 | 1 | 0 | 1 | 1 | 0 | 0 | 00 | 00 | 00 | 00 |
| add | 000000 | 100000 | 1 | 1 | 1 | 0 | 0 | 0 | 00 | 10 | 00 | 00 |
| sub | 000000 | 100010 | 1 | 1 | 0 | 0 | 1 | 0 | 00 | 10 | 00 | 00 |
| slt | 000000 | 101010 | 1 | 1 | 0 | 0 | 0 | 0 | 00 | 01 | 00 | 00 |
| addi | 001000 | 001000 | 1 | 0 | 1 | 1 | 0 | 0 | 00 | 10 | 00 | 00 |
| slti | 001010 | 001010 | 1 | 0 | 0 | 1 | 0 | 0 | 00 | 01 | 00 | 00 |
| and | 000000 | 100100 | 1 | 1 | 1 | 0 | 1 | 0 | 00 | 11 | 00 | 00 |
| or | 000000 | 100101 | 1 | 1 | 0 | 0 | 0 | 0 | 01 | 11 | 00 | 00 |
| xor | 000000 | 100110 | 1 | 1 | 0 | 0 | 0 | 0 | 10 | 11 | 00 | 00 |
| nor | 000000 | 100111 | 1 | 1 | 0 | 0 | 0 | 0 | 11 | 11 | 00 | 00 |
| andi | 001100 | 001100 | 1 | 0 | 0 | 1 | 0 | 0 | 00 | 11 | 00 | 00 |
| ori | 001101 | 001101 | 1 | 0 | 0 | 1 | 0 | 0 | 01 | 11 | 00 | 00 |
| xori | 001110 | 001101 | 1 | 0 | 0 | 1 | 0 | 0 | 10 | 11 | 00 | 00 |
| lw | 100011 | 100011 | 1 | 0 | 0 | 1 | 0 | 0 | 10 | 10 | 00 | 00 |
| sw | 101011 | 101011 | 0 | 0 | 1 | 1 | 0 | 1 | 00 | 10 | 00 | 00 |
| j | 000010 | 000010 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 | 01 |
| jr | 000000 | 001000 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 00 | 10 |
| bltz | 000001 | 000001 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 11 | 00 |
| beq | 000100 | 000100 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 01 | 00 |
| bne | 000101 | 000101 | 0 | 0 | 0 | 0 | 0 | 0 | 00 | 00 | 10 | 00 |

With the table completed, we begin designing the control unit using vhdl as a process in the datapath code done in part 1 of the lab. The unit is designed as a combinational logic circuit whose inputs are the opcode and function fields of the instruction and the outputs are the 10 control signals from the table above. This control unit and the entire CPU is tested using the following instructions stored in I\_Cache.

| "00000" "00100000000000110000000000000000"; *-- addi r3, r0, 0* "00001" "00100000000000010000000000000000"; *-- addi r1, r0, 0* "00010" "00100000000000100000000000000101"; *-- addi r2,r0,5* **LOOP**: "00011" "00000000001000100000100000100000"; *-- add r1,r1,r2* "00100" "00100000010000101111111111111111"; *-- addi r2, r2, -1* "00101" "00010000010000110000000000000001"; *-- beq r2,r3 (+1)* THERE "00110" "00001000000000000000000000000011"; *-- jump 3 (LOOP)* THERE:"00111" "10101100000000010000000000000000"; *-- sw r1, 0(r0)* "01000" "10001100000001000000000000000000"; *-- lw r4, 0(r0)* "01001" "00110000100001000000000000001010"; *-- andi r4,r4, 0x000A* "01010" "00110100100001000000000000000001"; *-- ori r4,r4, 0x0001* "01011" "00111000100001000000000000001011"; *-- xori r4,r4, 0xB* "01100" "00111000100001000000000000000000"; *-- xori r4,r4,* 0x0000 **others** "00000000000000000000000000000000"; *-- dont care* |
| --- |

**Results:**

**The complete VHDL code source for the register file:**

| **library** IEEE; **use** IEEE.std\_logic\_1164.**all**; **use** IEEE.std\_logic\_signed.**all**;  **entity** cpu **is**  **port**(   *--asynchronous reset and a clock input.*  clk, rst: **in** std\_logic;  rs\_out : **out** std\_logic\_vector(31 **downto** 0);  rt\_out : **out** std\_logic\_vector(31 **downto** 0);  *--output ports from register file*  pc\_out : **out** std\_logic\_vector(31 **downto** 0);   zero, overflow : **out** std\_logic); **end** cpu;     **architecture** cpu\_arch **of** cpu **is**  *--components:*  *--component of instruction cache*  **component** icache\_comp **is**  **port**  (  address : **in** std\_logic\_vector(4 **downto** 0);  instruction : **out** std\_logic\_vector(31 **downto** 0));  **end** **component**;   *--component of a 5 bit mux*   **component** mux5\_comp **is**  **port**  (  control : **in** std\_logic;  regD : **in** std\_logic\_vector(4 **downto** 0);  regT : **in** std\_logic\_vector(4 **downto** 0);  reg : **out** std\_logic\_vector(4 **downto** 0));  **end** **component**;    *--component of the REGFILE*  *-- 32x32 register file*  **component** regFile\_comp **is**   **port**  ( din : **in** std\_logic\_vector(31 **downto** 0);   reset : **in** std\_logic;  clk : **in** std\_logic;  write : **in** std\_logic;  read\_a : **in** std\_logic\_vector(4 **downto** 0);  read\_b : **in** std\_logic\_vector(4 **downto** 0);  write\_address : **in** std\_logic\_vector(4 **downto** 0);  out\_a : **out** std\_logic\_vector(31 **downto** 0);  out\_b : **out** std\_logic\_vector(31 **downto** 0));  **end** **component**;    *--component of sign extending*  **component** signex\_comp **is**  **port**  ( func : **in** std\_logic\_vector(1 **downto** 0);  immediate : **in** std\_logic\_vector(15 **downto** 0);  signExtended : **out** std\_logic\_vector(31 **downto** 0));   **end** **component**;    *--component of a 32 bit mux*   **component** mux32\_comp **is**  **port**  ( control : **in** std\_logic;  reg : **in** std\_logic\_vector(31 **downto** 0);  immediate : **in** std\_logic\_vector(31 **downto** 0);  value : **out** std\_logic\_vector(31 **downto** 0));  **end** **component**;   *--component of the ALU*   **component** alu\_comp **is**   **port**  ( x : **in** std\_logic\_vector(31 **downto** 0);  y : **in** std\_logic\_vector(31 **downto** 0); *-- two input operands*  add\_sub : **in** std\_logic; *-- 0 = add , 1 = sub*  logic\_func : **in** std\_logic\_vector(1 **downto** 0 ); *-- 00 = AND, 01 = OR , 10 = XOR , 11 = NOR*  func : **in** std\_logic\_vector(1 **downto** 0 ) ; *-- 00 = lui, 01 = setless , 10 = arith , 11 = logic*  output : **out** std\_logic\_vector(31 **downto** 0) ;  overflow : **out** std\_logic ;  zero : **out** std\_logic);  **end** **component**;   *--component of the data cache*   **component** dcache\_comp **is**  **port**  (  clock : **in** std\_logic;  reset : **in** std\_logic;  registerData : **in** std\_logic\_vector(31 **downto** 0);  write : **in** std\_logic;  muxsel : **in** std\_logic;  ld\_st : **in** std\_logic\_vector(31 **downto** 0);  Dout: **out** std\_logic\_vector(31 **downto** 0));  **end** **component**;   *--component of the nextAddress block*   **component** nextAddr\_comp **is**   **port**  (  rt : **in** std\_logic\_vector(31 **downto** 0);  rs : **in** std\_logic\_vector(31 **downto** 0);   pc : **in** std\_logic\_vector(31 **downto** 0);  target\_address : **in** std\_logic\_vector(25 **downto** 0);  branch\_type : **in** std\_logic\_vector(1 **downto** 0);  pc\_sel : **in** std\_logic\_vector(1 **downto** 0);  next\_pc : **out** std\_logic\_vector(31 **downto** 0));  **end** **component**;   *--component of pc regsiter*  **component** pc\_comp **is**   **port**(   clock : **in** std\_logic;  reset : **in** std\_logic;  value : **in** std\_logic\_vector(31 **downto** 0);  lower : **out** std\_logic\_vector(4 **downto** 0);  pc\_value : **out** std\_logic\_vector(31 **downto** 0));  **end** **component**;   *--Entities to connect the components to the vhd code*  **for** instructioni : icache\_comp **use** **entity** work.icache(icache\_arch);  **for** write : mux5\_comp **use** **entity** work.Mux5(mux5\_arch);  **for** registr : regFile\_comp **use** **entity** work.regfile(regfile\_arch);  **for** se : signex\_comp **use** **entity** work.signextend(se\_arch);  **for** check : mux32\_comp **use** **entity** work.mux32(mux32\_arch);  **for** a\_l\_u : alu\_comp **use** **entity** work.alu(alu\_arch);  **for** data : dcache\_comp **use** **entity** work.dcache(dcache\_arch);  **for** nextpc : nextAddr\_comp **use** **entity** work.next\_address(arch\_next);  **for** pcpass : pc\_comp **use** **entity** work.pc(pc\_arch);    *--define the signals*  **signal** pc\_in : std\_logic\_vector(31 **downto** 0);  **signal** pc\_nxt : std\_logic\_vector(31 **downto** 0);  **signal** lower\_pc : std\_logic\_vector(4 **downto** 0);   **signal** instruction : std\_logic\_vector(31 **downto** 0);  **signal** reg\_one, reg\_two : std\_logic\_vector(31 **downto** 0);  **signal** write\_reg : std\_logic\_vector(4 **downto** 0);  **signal** immediate\_se : std\_logic\_vector(31 **downto** 0);  **signal** reg\_value : std\_logic\_vector(31 **downto** 0);  **signal** alu\_out : std\_logic\_vector(31 **downto** 0);  **signal** alu\_overflow, alu\_zero : std\_logic;  **signal** data\_out : std\_logic\_vector(31 **downto** 0);   *--control unit signals*  **signal** branch\_type, pc\_sel, logic, the\_func : std\_logic\_vector(1 **downto** 0);  **signal** reg\_write,alu\_src, addsub, data\_write, reg\_in\_src, reg\_dst : std\_logic;     **begin**   pcpass : pc\_comp **port** **map**(clk, rst, pc\_nxt, lower\_pc, pc\_in);   *--------next pc------------*  nextpc : nextAddr\_comp **port** **map**(reg\_one, reg\_two, pc\_in, instruction(25 **downto** 0), branch\_type, pc\_sel, pc\_nxt);    *-----------send pc address to icache and get the instruction components*  instructioni : icache\_comp **port** **map**(lower\_pc, instruction);    *--------get the address to write into*  write : mux5\_comp **port** **map**(reg\_dst, instruction(15 **downto** 11), instruction(20 **downto** 16), write\_reg);   *----------control unit-------*  **process**(instruction)   **begin**  *--set tje signals depending on the opcode of the instructions*  *--lui*  **if**(instruction(31 **downto** 26) = "001111") **then**  reg\_write <= '1';   reg\_dst <= '0';   reg\_in\_src <= '1';   alu\_src <= '1';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "00";  branch\_type <= "00";   pc\_sel <= "00";   *--addi*  **elsif**(instruction(31 **downto** 26) = "001000") **then**  reg\_write <= '1';   reg\_dst <= '0';   reg\_in\_src <= '1';   alu\_src <= '1';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "10";  branch\_type <= "00";   pc\_sel <= "00";   *--slti*  **elsif**(instruction(31 **downto** 26) = "001010") **then**  reg\_write <= '1';   reg\_dst <= '0';   reg\_in\_src <= '0';   alu\_src <= '1';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "01";  branch\_type <= "00";   pc\_sel <= "00";   *--andi*  **elsif**(instruction(31 **downto** 26) = "001100") **then**  reg\_write <= '1';   reg\_dst <= '0';   reg\_in\_src <= '0';   alu\_src <= '1';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "11";  branch\_type <= "00";   pc\_sel <= "00";   *--ori*  **elsif**(instruction(31 **downto** 26) = "001101") **then**  reg\_write <= '1';   reg\_dst <= '0';   reg\_in\_src <= '0';   alu\_src <= '1';   addsub <= '0';   data\_write <= '0';   logic <= "01";  the\_func <= "11";  branch\_type <= "00";   pc\_sel <= "00";   *--xori*  **elsif**(instruction(31 **downto** 26) = "001101") **then**  reg\_write <= '1';   reg\_dst <= '0';   reg\_in\_src <= '0';   alu\_src <= '1';   addsub <= '0';   data\_write <= '0';   logic <= "10";  the\_func <= "11";  branch\_type <= "00";   pc\_sel <= "00";   *--lw*  **elsif**(instruction(31 **downto** 26) = "100011") **then**  reg\_write <= '1';   reg\_dst <= '0';   reg\_in\_src <= '0';   alu\_src <= '1';   addsub <= '0';   data\_write <= '0';   logic <= "10";  the\_func <= "10";  branch\_type <= "00";   pc\_sel <= "00";   *--sw*  **elsif**(instruction(31 **downto** 26) = "101011") **then**  reg\_write <= '0';   reg\_dst <= '0';   reg\_in\_src <= '1';   alu\_src <= '1';   addsub <= '0';   data\_write <= '1';   logic <= "00";  the\_func <= "10";  branch\_type <= "00";   pc\_sel <= "00";   *--nor*  **elsif**(instruction(31 **downto** 26) = "100111") **then**  reg\_write <= '1';   reg\_dst <= '1';   reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "11";  the\_func <= "11";  branch\_type <= "00";   pc\_sel <= "00";   *--jump*  **elsif**(instruction(31 **downto** 26) = "000010") **then**  reg\_write <= '0';  reg\_dst <= '0';  reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "00";  branch\_type <= "00";   pc\_sel <= "01";   *--bltz*  **elsif**(instruction(31 **downto** 26) = "000001") **then**  reg\_write <= '0';   reg\_dst <= '0';   reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "00";  branch\_type <= "11";   pc\_sel <= "00";   *--beq*  **elsif**(instruction(31 **downto** 26) = "000100") **then**  reg\_write <= '0';  reg\_dst <= '0';  reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "00";  branch\_type <= "01";   pc\_sel <= "00";   *--bne*  **elsif**(instruction(31 **downto** 26) = "000101") **then**  reg\_write <= '0';   reg\_dst <= '0';   reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "00";  branch\_type <= "10";   pc\_sel <= "00";    *--if opcode is 0*  **elsif**(instruction(31 **downto** 26) = "000000") **then**   *--add*  **if**(instruction(5 **downto** 0) = "100000") **then**  reg\_write <= '1';   reg\_dst <= '1';   reg\_in\_src <= '1';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "10";  branch\_type <= "00";  pc\_sel <= "00";  *--sub*  **elsif**(instruction(5 **downto** 0) = "100010") **then**  reg\_write <= '1';   reg\_dst <= '1';   reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '1';   data\_write <= '0';   logic <= "00";  the\_func <= "10";  branch\_type <= "00";  pc\_sel <= "00";   *--slt*  **elsif**(instruction(5 **downto** 0) = "101010") **then**  reg\_write <= '1';   reg\_dst <= '1';   reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "01";  branch\_type <= "00";  pc\_sel <= "00";   *--and*  **elsif**(instruction(5 **downto** 0) = "100100") **then**  reg\_write <= '1';   reg\_dst <= '1';   reg\_in\_src <= '1';   alu\_src <= '0';   addsub <= '1';   data\_write <= '0';   logic <= "00";  the\_func <= "11";  branch\_type <= "00";  pc\_sel <= "00";   *--or*  **elsif**(instruction(5 **downto** 0) = "100101") **then**  reg\_write <= '1';   reg\_dst <= '1';   reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "01";  the\_func <= "11";  branch\_type <= "00";  pc\_sel <= "00";   *--xor*  **elsif**(instruction(5 **downto** 0) = "100110") **then**  reg\_write <= '1';   reg\_dst <= '1';   reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "10";  the\_func <= "11";  branch\_type <= "00";  pc\_sel <= "00";   *--jump register*  **else**  reg\_write <= '0';   reg\_dst <= '0';   reg\_in\_src <= '0';   alu\_src <= '0';   addsub <= '0';   data\_write <= '0';   logic <= "00";  the\_func <= "00";  branch\_type <= "00";  pc\_sel <= "10";   **end** **if**;  **end** **if**;  **end** **process**;   *--Do operation depending on the control values*  registr : regFile\_comp **port** **map**(data\_out,rst,clk,reg\_write,instruction(20 **downto** 16), instruction(25 **downto** 21), write\_reg,reg\_one, reg\_two);   *------------Sign Extend*  se : signex\_comp **port** **map**(the\_func, instruction(15 **downto** 0), immediate\_se);   *-----------determine if we use immediate or register*  check : mux32\_comp **port** **map**(alu\_src, reg\_two, immediate\_se, reg\_value);   *-----------alu*  a\_l\_u : alu\_comp **port** **map**(reg\_one, reg\_value, addsub, logic, the\_func, alu\_out, alu\_overflow, alu\_zero);    *------------dcache*  data : dcache\_comp **port** **map**(clk, rst, alu\_out, data\_write, reg\_in\_src, reg\_one, data\_out);   rs\_out <= reg\_two(31 **downto** 0);  rt\_out <= reg\_one(31 **downto** 0);  pc\_out <= pc\_nxt(31 **downto** 0);  zero <= alu\_zero;  overflow <= alu\_overflow; **end** cpu\_arch; |
| --- |

**The TCL script file:**

| # TCL script **for** running vivado **in** batch mode **to** synthesize  # **To** run the script first source the Vivado env **file**: # source /CMC/tools/xilinx/Vivado\_2018.2/Vivado/2018.2/settings64\_CMC\_central\_license.csh # #**Then** issue the following command from the Linux prompt: # vivado -log tedcircuit.log -mode batch -source tedcircuit\_script.tcl  # read **in** the VHDL source code files **and** the xdc constraints **file**  set\_property **SEVERITY** {Warning} [get\_drc\_checks UCIO-1]  read\_vhdl { ../Code/cpu.vhd ../Code/cpuCircuit.vhd } read\_xdc CPUcircuit.xdc  # the -top refers **to** the top level VHDL **entity** name # the -part specfies the target Xilinx FPGA  synth\_design -top CPUCircuit -part xc7a100tcsg324-1  opt\_design place\_design route\_design  # **generate** the bitsteam **file** write\_bitstream -**force** CPUCircuit.bit |
| --- |

**The log file is available in the appendix.**

Unfortunately, while performing the synthesis with Xilinx Vivado with the command line script I faced an error that I could find a fix for. The synthesis of the program throws an error that the complaining and simulation of the code did not show.

The error is :

ERROR: [Synth 8-2948] no **architecture** 'pc\_arch' **for** **entity** 'pc\_comp' [/nfs/home/j/j\_ratn/COEN316/Code/cpu.vhd:150]

This is a synthesis error that happens on line 150 of the cpu.vhd code:

| pcpass : pc\_comp **port** **map**(clk, rst, pc\_nxt, lower\_pc, pc\_in); |
| --- |

This line port maps the inputs and outputs of the CPU to the PC component of the system. The error says that the component has no access to architecture of pc “pc\_arch” but it is declared on line 127:

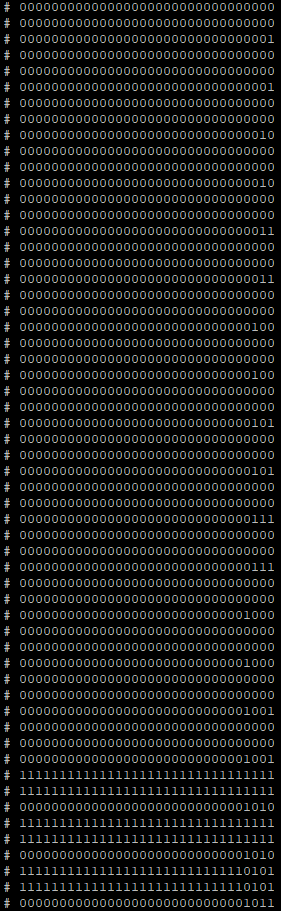
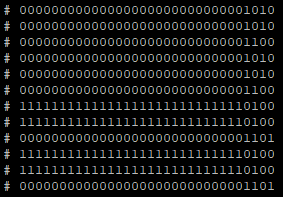
| **for** pcpass : pc\_comp **use** **entity** work.pc(pc\_arch); |
| --- |

And we know that the PC code is situated in the work folder since it was compiled using vcom on the command panel.

I understand the error and the cause of the error but i could not find a way to fix the error, since the architecture is available but the program somehow does not have access to it.

| #*-----------------------------------------------------------* # Vivado v2018.2 (64-bit) # SW Build 2258646 **on** Thu Jun 14 20:02:38 MDT 2018 # IP Build 2256618 **on** Thu Jun 14 22:10:49 MDT 2018 # Start **of** session at: Mon Nov 30 10:02:04 2020 # **Process** ID: 61661 # Current directory: /nfs/home/j/j\_ratn/COEN316/CPUCIRCUIT\_SCRIPT # Command line: vivado -log cpucircuit.log -mode batch -source CPU\_script.tcl # Log **file**: /nfs/home/j/j\_ratn/COEN316/CPUCIRCUIT\_SCRIPT/cpucircuit.log # Journal **file**: /nfs/home/j/j\_ratn/COEN316/CPUCIRCUIT\_SCRIPT/vivado.jou #*-----------------------------------------------------------* source CPU\_script.tcl # set\_property **SEVERITY** {Warning} [get\_drc\_checks UCIO-1] # read\_vhdl { ../Code/cpu.vhd ../Code/cpuCircuit.vhd } # read\_xdc CPUcircuit.xdc # synth\_design -top CPUCircuit -part xc7a100tcsg324-1 Command: synth\_design -top CPUCircuit -part xc7a100tcsg324-1 Starting synth\_design Attempting **to** get a license **for** feature 'Synthesis' **and**/**or** device 'xc7a100t' INFO: [Common 17-349] Got license **for** feature 'Synthesis' **and**/**or** device 'xc7a100t' INFO: Launching helper **process** **for** spawning children vivado processes INFO: Helper **process** launched **with** PID 61688  *---------------------------------------------------------------------------------* Starting RTL Elaboration : Time (s): cpu = 00:00:03 ; elapsed = 00:00:06 . Memory (MB): peak = 1468.590 ; gain = 86.727 ; free physical = 79561 ; free virtual = 128919 *---------------------------------------------------------------------------------* INFO: [Synth 8-638] synthesizing module 'CPUCircuit' [/nfs/home/j/j\_ratn/COEN316/Code/cpuCircuit.vhd:10] INFO: [Synth 8-3491] module 'cpu' declared at '/nfs/home/j/j\_ratn/COEN316/Code/cpu.vhd:5' bound **to** instance 'U1' **of** **component** 'cpu' [/nfs/home/j/j\_ratn/COEN316/Code/cpuCircuit.vhd:31] INFO: [Synth 8-638] synthesizing module 'cpu' [/nfs/home/j/j\_ratn/COEN316/Code/cpu.vhd:17] ERROR: [Synth 8-2948] no **architecture** 'pc\_arch' **for** **entity** 'pc\_comp' [/nfs/home/j/j\_ratn/COEN316/Code/cpu.vhd:150] ERROR: [Synth 8-285] failed synthesizing module 'cpu' [/nfs/home/j/j\_ratn/COEN316/Code/cpu.vhd:17] ERROR: [Synth 8-285] failed synthesizing module 'CPUCircuit' [/nfs/home/j/j\_ratn/COEN316/Code/cpuCircuit.vhd:10] *---------------------------------------------------------------------------------* Finished RTL Elaboration : Time (s): cpu = 00:00:05 ; elapsed = 00:00:08 . Memory (MB): peak = 1513.215 ; gain = 131.352 ; free physical = 79573 ; free virtual = 128931 *---------------------------------------------------------------------------------* RTL Elaboration failed INFO: [Common 17-83] Releasing license: Synthesis 5 Infos, 0 Warnings, 0 Critical Warnings **and** 4 Errors encountered. synth\_design failed ERROR: [Common 17-69] Command failed: Synthesis failed - please see the console **or** run log **file** **for** details INFO: [Common 17-206] Exiting Vivado at Mon Nov 30 10:02:30 2020... |
| --- |

**Simulation and implementation results using port maps:**

* The Modelsim simulation: rs\_out, rt\_out and pc\_out are examined
  +  

**Conclusion:**

In conclusion, the experiment was done somewhat successfully since the results obtained in the modelsim step is accurate compared to the theoretical results. We learned and understood all the steps to do a simulation of a register file.This is shown by the Modelsim simulations testing with the DO file for all possible operations and it produced the correct outputs. The lab was successful in designing a complete CPU for simulation. All of the combinations of inputs were tested and the correct outputs were obtained from the Ichache to the D-cache. But the experiment failed while performing synthesis with the Xilinx Vivado from the command line script, it threw an error in the synthesis step saying the architecture of the PC component is not available. I was not able to solve this error in time unfortunately.