Tennessee Technological University

ECE 4120 – Fundamentals of Computer Design

Department of Electrical and Computer Engineering

Written By: Josh Adair, Cameron Dempsey, Brantlee Garland, Hunter Goodson, and Randy Wilhoite

Phase One: Instruction Fetch Unit

**Table of Contents**

Modified Block Diagram ………………………………………………………………………………………………………………………… 1

Phase One Objectives ………………………………………………………………………………………………..………………………. 1-2

VHDL Implementation Elaboration ……………………………………………………………………………………………………….. 2

Synthesis Results ……………………………………………………………………………………………………………………………….. 2-3

Test Bench Elaboration …………………………………………………………………………………………………………………………. 3

Waveform Elaboration ………………………………………………………………………………………………………….………………. 4

TimeQuest Timing Analyzation ………………………………………………………………………………………………………….. 4-6

Phase One Conclusion …………………………………………………………………………………………………………………………… 6

**Modified Block Diagram:**

Figure 1 below shows the instruction fetch unit block diagram with bit values and any additional connections. You can see from the diagram that PC has a 32 bit input and output, the adder has a 32 bit and constant 1 input with a 32 bit output, and the 8 LSB of PC are input into the instruction memory, with a 32 bit output. The constant required changing to 1 instead of 4 due to the fact that adding four caused addresses to be skipped. Instead of the address going from 4 to 8, 8 to 12, 12 to 16, etc, we wanted it to go from 1 to 2, 2 to 3, etc. This is why we changed the constant to a 1.

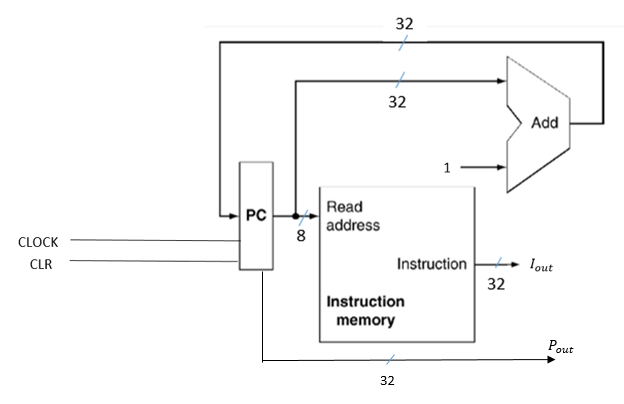


Figure 1. Modified block diagram

**Phase One Objectives:**

Phase one of this project had many objectives, and the big picture once all of these objectives are completed is to fully implement the instruction fetch unit of the MIPS processor. The first objective was to create a program counter (PC) in VHDL. This needed to be a synchronous block with a 32 bit input and a 32 bit output. The second objective was to create an adder in VHDL as a combinational block. The adder is 32 bits with two inputs, a constant 4 and the 32 bit output from the PC. The third objective and final block that needed to be created was the synchronous for instruction memory. The input to this block is the 8 least significant bits being output from the PC. The output should be 32 bits. The fourth objective was to ensure that the MIPS instructions were stored through the test bench in the first five locations of instruction memory. The final objective was to simulate everything we created using Quartus and the TimeQuest timing analyzer in order to obtain relevant data, such as Fmax and timing reports.

**VHDL Implementation Elaboration:**

For the adder, we chose create an entity called Add4 with a 32 bit standard logic vector input A. We also have Q, a 32 bit standard logic vector output. We then created an architecture from Add4 that included a constant value of 4 and a process. Inside the process, we assign the output Q to the sum of 4 plus whatever the input was. The process and architecture are then ended.

For the PC, we chose to create an entity called Program\_Counter with 3 different inputs. The first input is D, a 32 bit standard logic vector. The second and third inputs are CLR and CLK, which adjusts the clear and clock variables. Next we created an architecture of Program\_Counter called pc. Inside is a process that uses both CLK and CLR. If CLR is chosen to be 1, then it is cleared and starts at the beginning. If CLR is chosen to be 0, on the rising edge of the clock, Q is set to D. The process and architecture are then ended.

For the instruction memory, we chose to use the Megafunction using the steps from the notes. After selecting IP Catalog, you then choose basic functions. Next you choose RAM: 1-PORT, VHDL, and change the bits to 32 with 256 words. Ensure the ‘q’ output port is selected. Next we linked the Megafunction to our .mif file by selecting the option and then specifying our filename. After choosing the correct simulation library, the final step is to ensure the VHDL component declaration file option is selected.

**Synthesis Results:**

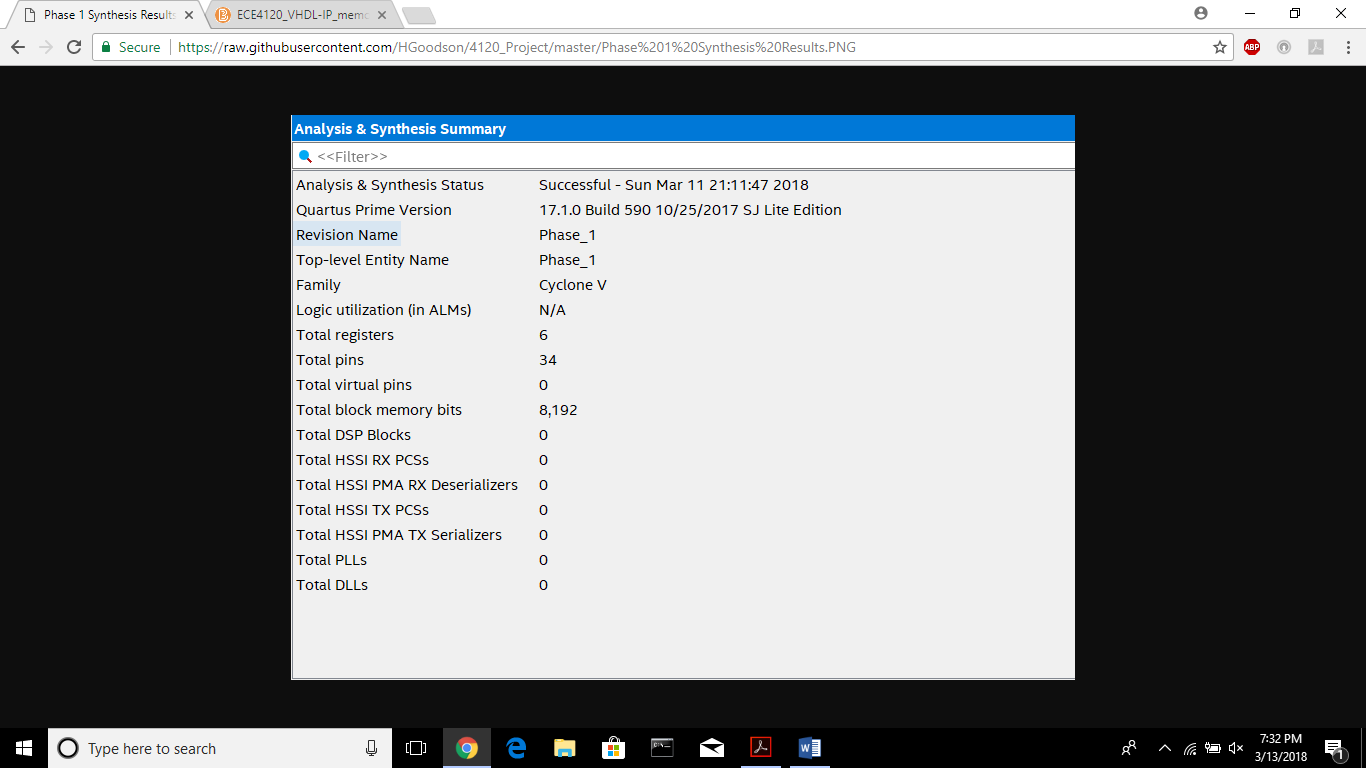


Figure 2. Synthesis results from Quartus.

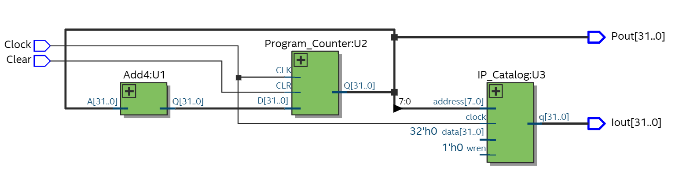


Figure 3. RTL Snapshot View

**Test Bench Elaboration:**

Our testbench is shown below. There are 16 instructions, but out test bench only lasts long enough to load three instructions. There is also only three instructions read as well.

library ieee;

use ieee.std\_logic\_1164.all;

entity tb\_Phase\_1 is

end entity tb\_Phase\_1;

architecture tb\_Phase\_1\_arch of tb\_Phase\_1 is

signal Clear: STD\_LOGIC := '0';

signal Clock : STD\_LOGIC := '0';

signal Iout : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

signal Pout : STD\_LOGIC\_VECTOR(31 DOWNTO 0);

component Phase\_1 IS

port ( Clear, Clock :in STD\_LOGIC;

Iout, Pout : out STD\_LOGIC\_VECTOR(31 DOWNTO 0));

END component ;

Begin

--Each time the clock transitions, an read occurs.

DUT: Phase\_1 port map (Clear,Clock, Iout(31 DOWNTO 0), Pout(31 DOWNTO 0));

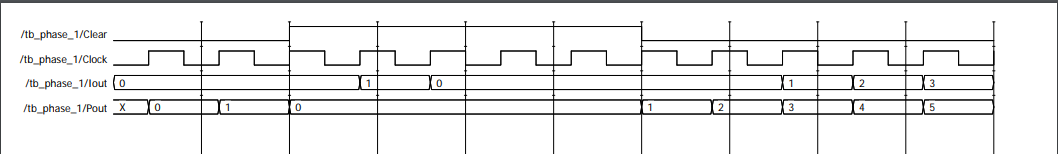
Clock <= not Clock after 20ns;

Clear <= '1' after 100ns,

'0' after 300ns;

end tb\_Phase\_1\_arch;

**Waveform Elaboration:**



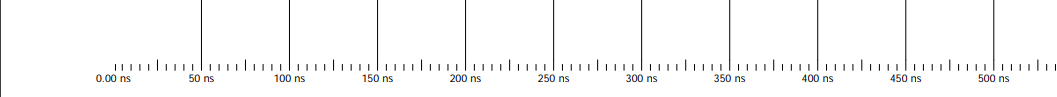


Figure 4. Waveform obtained from implementation

1. A .mif file was used, and the reading from memory is clearly shown on the waveform in red.
2. The waveform shows the successful implementation.

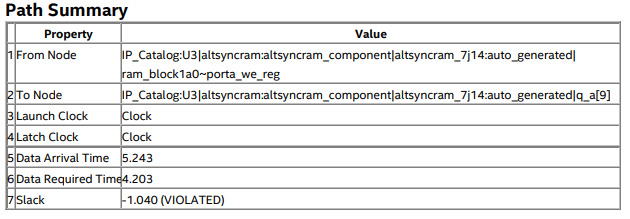
**TimeQuest Timing Analyzation:**

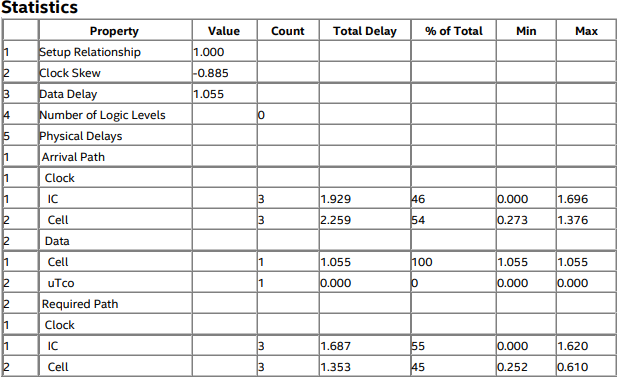
After completing the TimeQuest analysis, we found the following data shown below.

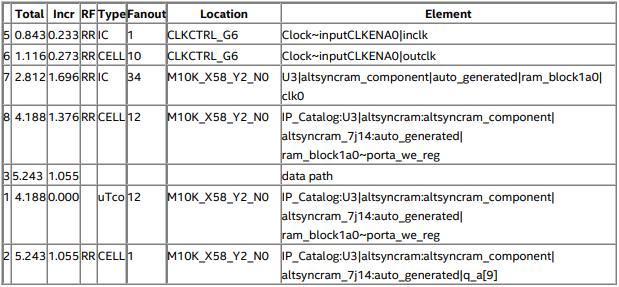
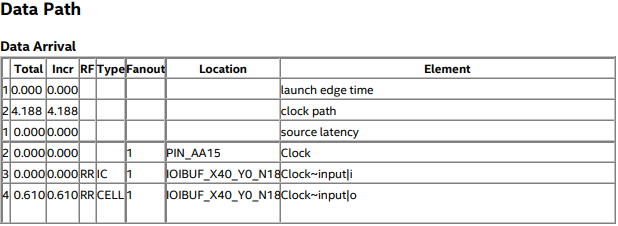
1. Fmax value

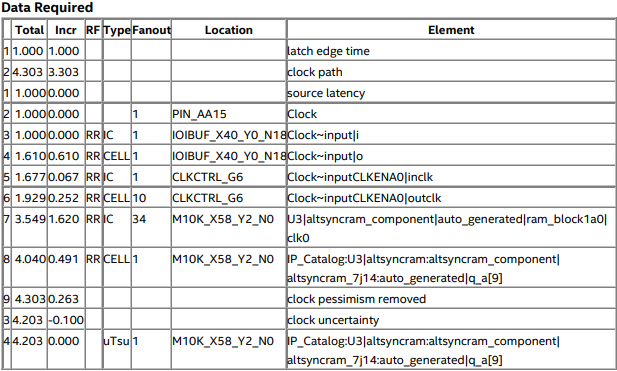


1. Shown is the timing report.









**Phase One Conclusion:**

We were able to complete phase one of the project successfully and met all project constraints. We created our synchronous PC block, combinational adder block, and synchronous instruction memory block. We only used the Megafunction to implement the instruction memory. For the adder, it was required to change the constant to be 1 instead of 4. This was due to receiving complications when adding 4 because the addresses were being increased too much. Changing the constant to 1 fixed our problem. Both the PC and instruction memory have the same clock signal. We were able to use the top module to port map all of the components. Our project compiled and worked successfully, providing us the required data and results via Quartus/ModelSim and TimeQuest.