PHASE 1

Michael Mollica & Nidhay Patel

ECE 4120 – Spring 2023
03/02/2023

I. Modified Block Diagram

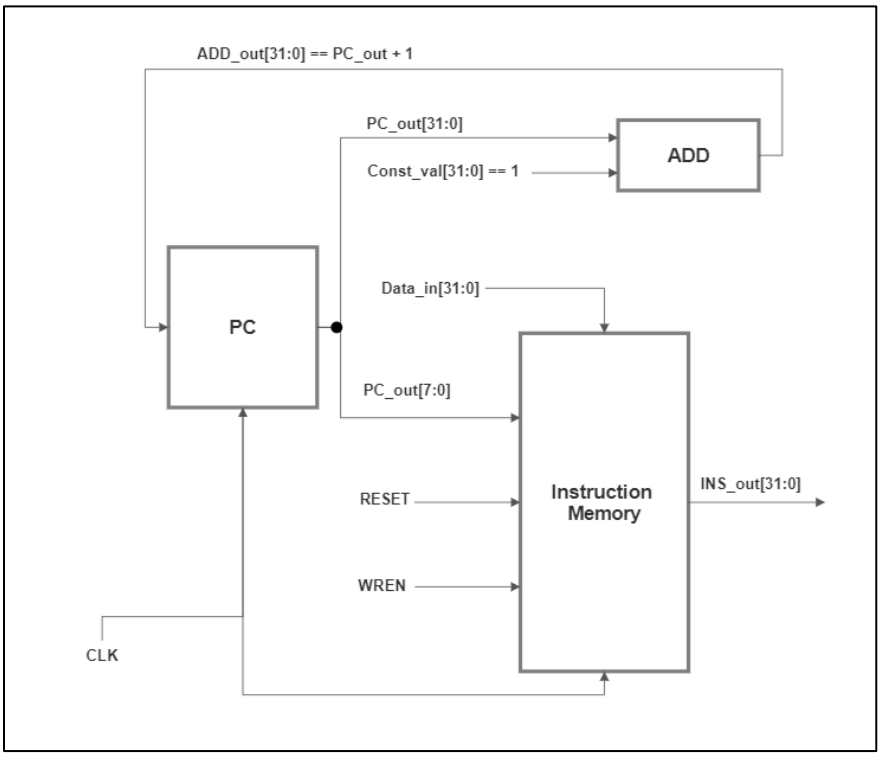

Figure 1: Detailed Block Diagram

II. VHDL Implementation

The VHDL code was written with the mindset of functionality through simplicity. This involved simple behavioral designs for the PC and ADD4 entities, and a port mapping for the final top-level entity Fetch Unit (FU). For the PC, the idea was to create an entity that functioned just like a 32-bit register. That is, at every rising clock edge, the input of the PC is sent to the output. The ADD4 entity was built for the PC to increment by a constant value of 4 for each rising clock edge. However, after all other implementation was finished, the constant value was changed to 1 as adding to a value of 4 was skipping memory locations. This was done by using unsigned addition of the PC output with the value 1 and feeding the output of ADD4 back into the input of PC. Instead of implementing a large full adder circuit using 32, 1-bit adders, the unsigned addition allowed for one simple instruction that will still provide the sum of the two inputs. Finally, the Instruction Memory was generated using the Mega-Function in Quartus, so no modifications were made to that particular circuit. To complete the Fetch Unit, each of the components were port mapped appropriately. However, the Instruction Memory required a few additional signals that were created and fed directly into the respective inputs: Data, Wren, Reset, and Address for testing purposes. Therefore, this design was done using the least amount of hardware and data signals possible to maintain proper functionality (75 total pins).

III. Device Selection

Available devices:

Name	Core Voltage	LEs	Total I/Os	GPIOs	Memory Bits	Embedded multiplier 9-bit elen
10M50DAF484C7G	1.2V	49760	360	360	1677312	288

Migration Devices... 0 migration devices selected

Figure 2: Intel FPGA DE10-Lite Board

IV. Flow Summary, RTL, Technology Map

Flow Summary

<<Filter>>

Flow Status	Successful - Wed Mar 01 15:38:44 2023
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	FU
Top-level Entity Name	FU
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	9 / 49,760 (< 1 %)
Total registers	8
Total pins	75 / 360 (21 %)
Total virtual pins	0
Total memory bits	8,192 / 1,677,312 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Figure 3: Flow Summary

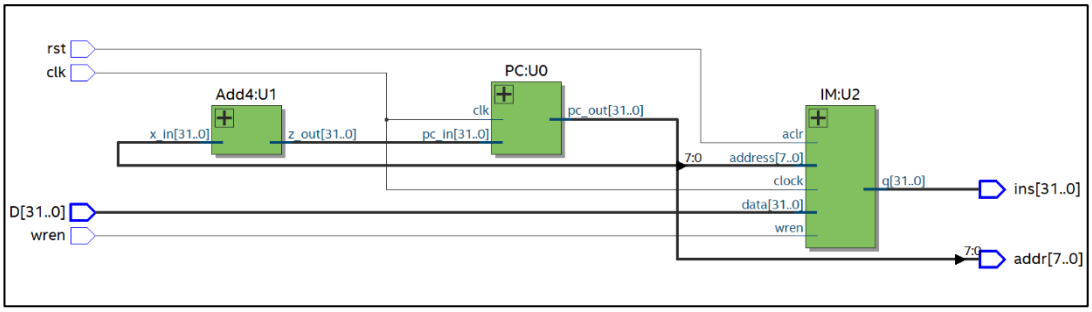

Figure 4: RTL View

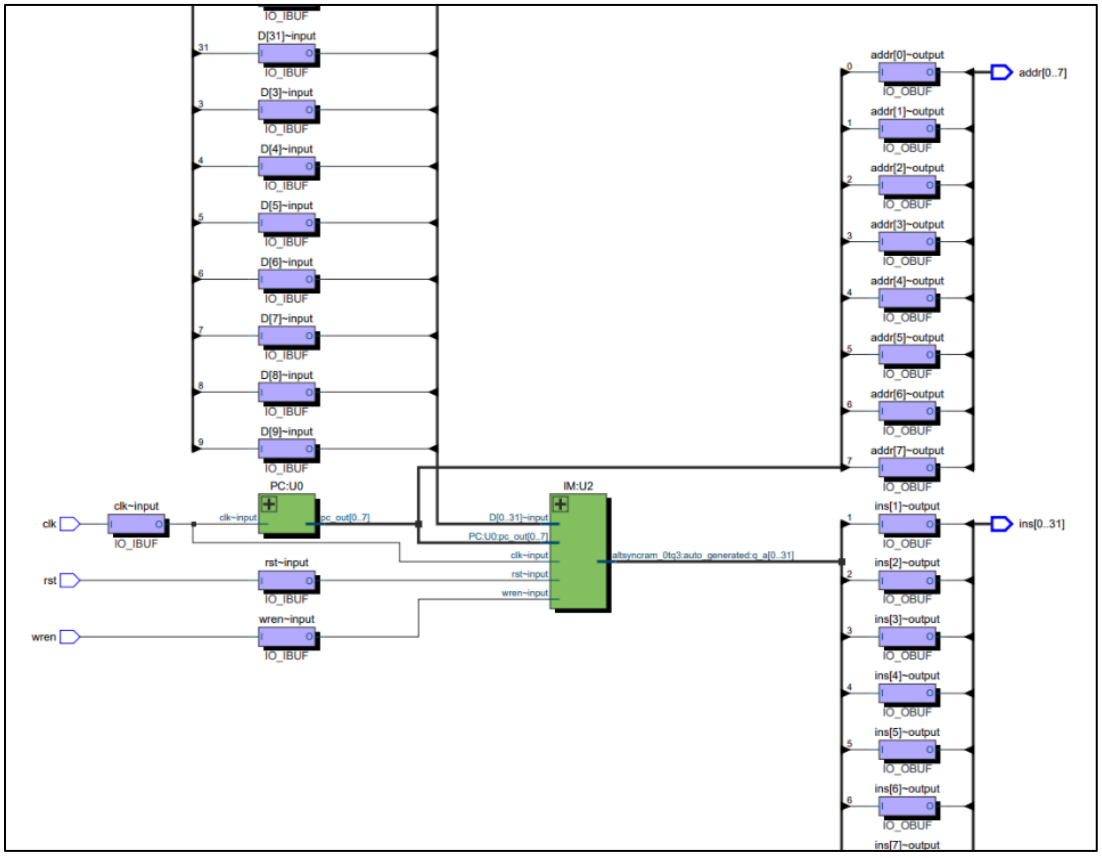

Figure 5: Technology Map (Zoomed-In)

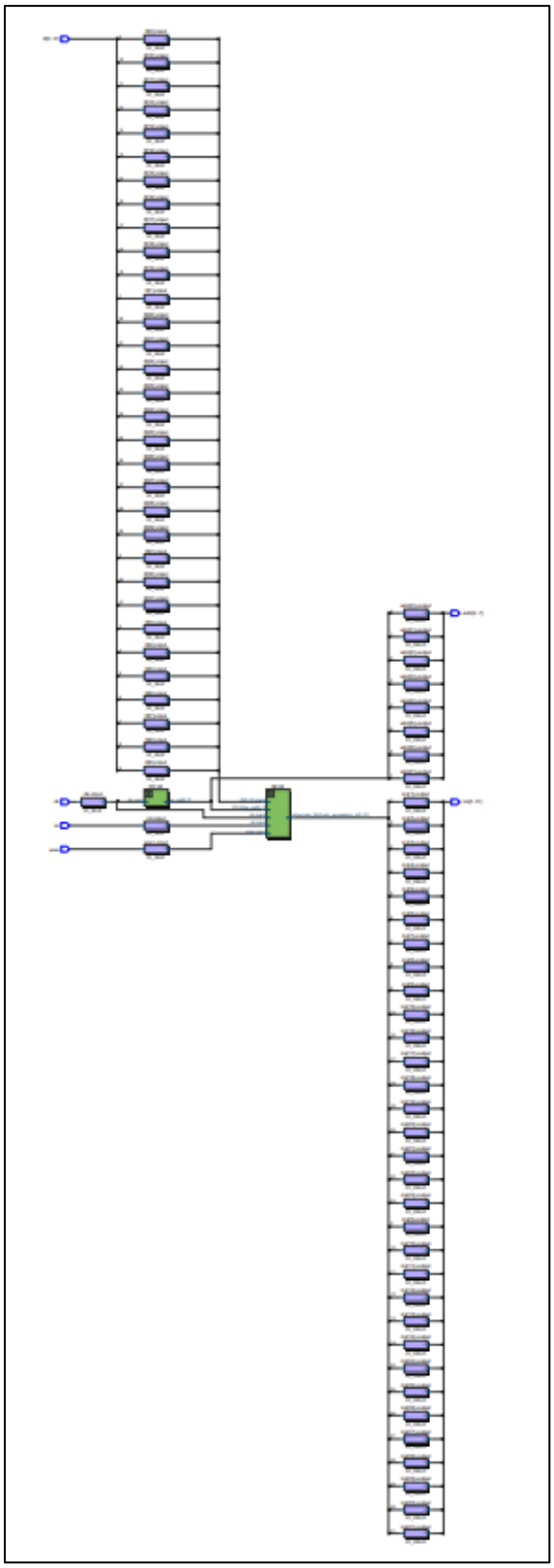

Figure 6: Technology Map (Full-View)

V. Test Bench & .MIF

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity testb is
5   end entity;
6
7   architecture Behavior of testb is
8      signal D         : std_logic_vector(31 downto 0);
9      signal clk       : std_logic := '0';
10     signal wren, rst : std_logic;
11     signal Q         : std_logic_vector(31 downto 0);
12     signal addr      : std_logic_vector(7 downto 0);
13
14     component FU is
15     port( D    : in std_logic_vector(31 downto 0);
16           addr : buffer std_logic_vector(7 downto 0);
17           clk  : in std_logic;
18           wren : in std_logic;
19           rst  : in std_logic;
20           ins  : out std_logic_vector(31 downto 0)
21          );
22     end component;
23
24     begin
25        DUT0 : FU port map(D, addr, clk, wren, rst, Q);
26
27     clk <= not clk after 50ns;
28     rst <= '0', '1' after 1000ns;
29
30     D <= x"00000020",              -- ADD
31          x"00000022" after 125ns, -- SUB
32          x"00000002" after 225ns, -- SWL
33          x"23000000" after 325ns, -- LW
34          x"2B000000" after 425ns; -- SW
35     wren <= '1', '0' after 500ns;
36
37  end architecture;
```

Figure 7: Test Bench Code

Test Bench Explanation: The first section of code (lines 4-5) is to create the test bench entity which is empty. Next is the creation of the behavioral architecture of the test bench entity (line 7). The first step in the architecture is to create signals for each of the components that will be in use later in the test bench file (lines 8-12). The following list explains each signal and its function:

- Signal D → Fetch Unit Data input signal. This signal carries the instruction to be stored in the Instruction Memory.
- Signal CLK → This is the clock signal for all sequential circuits in the Fetch Unit (PC and Instruction Memory).
- Signal WREN → Write Enable for the Instruction Memory, active HIGH.
- Signal RST → Reset for the Instruction Memory that asserts a output of 0, active HIGH.

- Signal Q → Fetch Unit output signal. This signal outputs the instruction stored in a given input address.
- Signal ADDR → Address signal that is sent to the Instruction Memory which is mapped to the lower 8 bits of the PC output.

After the signal declarations, the component FU is defined (lines 14-22). FU is the component for the top-level entity of the Fetch Unit, which is the main device under test. To capture the Fetch Unit as the device under test (DUT), a port mapping is done with the signals created earlier in the test bench architecture (line 25). Below the port mapping is where the assertion of signals is done to test the Fetch Unit functionality. This includes setting the clock period, asserting reset and write enable signals, and asserting the data input signal to the Fetch Unit at various times on the waveform (lines 27-35). This test bench loads 5 different instructions into the Instruction Memory. All other instructions are loaded into memory using a .mif file as seen in the figure below.

```
DEPTH = 256;
WIDTH = 32;
ADDRESS_RADIX = BIN;

DATA_RADIX = HEX;

CONTENT
BEGIN
[00000000..11111111] : 00000000;
 00000000 : 00000020;
 00000100 : 00000022;
 00001000 : 00000002;
 00001010 : 23000000;
 00010000 : 2B000000;
END;
```

Figure 8: .MIF File

.MIF Explanation: This .mif file is created for a memory unit of 256-bit depth (8-bit addressing) and 32-bit width (32-bit words), which meet the requirement for the Instruction Memory. Each memory address is loaded with all zeros initially except for locations 0x00 to 0x04. Those addresses are loaded with the same 5 instructions used in the test bench. However, this .mif file is **not** functional in the test bench but could be used in later phases when uploaded to the FPGA.

Adder Enabling: Finally, due to the ADD4 entity design, the PC output signal is always added to the constant signal equal to 1. Additionally, the PC output is initialized to 0x00000000 in the PC entity so that instruction zero is always executed first.

VI. Waveform Analysis

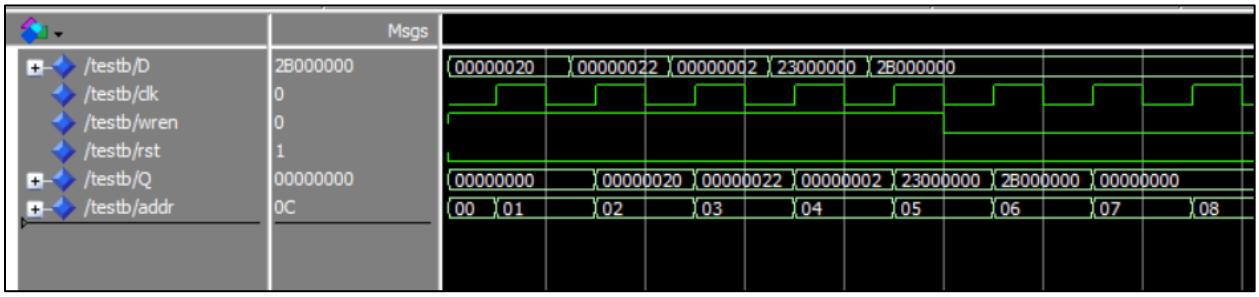

Figure 9: Timing Diagram

Waveform Explanation: The signal D is used to write the instruction to the Instruction Memory as shown in the first line of the figure above. That data is written at every rising clock edge into the Instruction Memory and then read at the next rising clock edge as seen in the signal Q. The signal ADDR at the bottom of Figure 9 is the address at which the instructions are being read from at each rising clock edge. For example, instruction 0x00000020 is loaded into the Instruction Memory at address 0x00 at the first rising clock edge and then output at the next rising clock edge in signal Q. Additionally, the Figure shows that the ADDR signal is properly incrementing by 1 at each rising clock edge. The waveform also shows the correct output when the WREN signal falls to LOW as output is no longer equal to the currently written data, but rather it is equal to the data in the respective address locations in memory (In this case, all other locations are 0x00000000). Below is a table of inputs, their expected outputs, and the results:

Table 1: Waveform Results

Address	Input	Expected Output (at next rising clock edge)	Actual Output
0x00	0x00000020	0x00000020	0x00000020
0x01	0x00000022	0x00000022	0x00000022
0x02	0x00000002	0x00000002	0x00000002
0x03	0x23000000	0x23000000	0x23000000
0x04	0x2B000000	0x2B000000	0x2B000000

VII. Timing Analysis

a) $F_{\max} = 345.9\ MHz$

b) $T_{su} = 0.026\ ns$

Setup Time Requirement:

Data Arrival Time:

➔ Launch Delay + Data Delay = 4.143 ns + 2.664 ns = 6.807 ns

Data Required Time:

➔ Setup Relationship + Latch Edge + Clock Pessimism - uT_{su} + Clock Uncertainty

➔ 2.9 ns + 3.865 ns + 0.097 ns + (– 0.026 ns) + (-0.02 ns) = 6.816 ns

Slack Time:

Data Required – Data Arrival: 6.816 – 6.807 = 0.009 ns

As Data Required > Data Arrival, it meets the setup requirement.

Hold Time Requirement:

Data Arrival Time:

➔ Launch Clock Delay + Data Delay = 3.526 ns + 1.009 ns = 4.535 ns

Data Required Time:

➔ Latch Edge + Clock Pessimism + uT_h + Clock Uncertainty

➔ 4.153 ns + (-0.059 ns) + 0.165 ns + 0.00 = 4.259 ns

Slack Time:

Data Arrival – Data Required: 4.535 – 4.259 = 0.276 ns

As Data Arrival > Data Required, it meets the hold requirement.

Slow 1200mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	345.9 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

Figure 10: Max Frequency

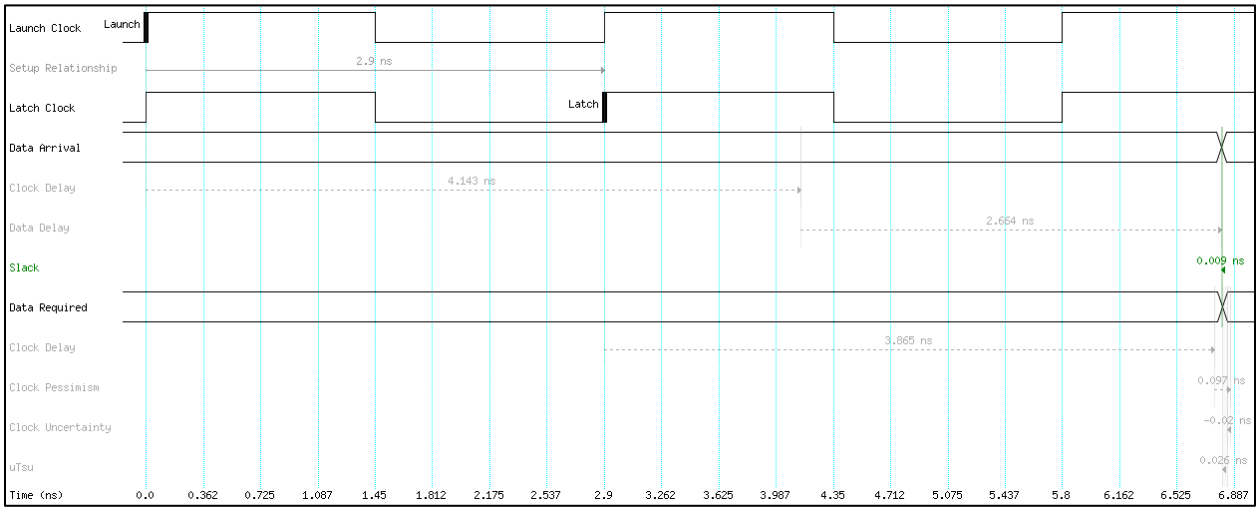

Figure 11: Setup Time Slack

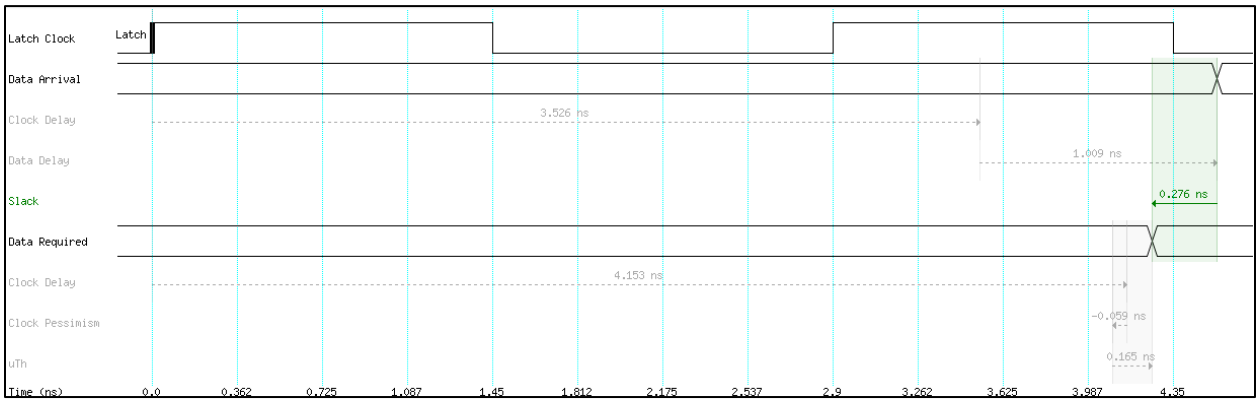

Figure 12: Hold Time Slack

VIII. Conclusion

The Fetch Unit is comprised of three different hardware units: Program Counter, Adder, and Instruction Memory. The Fetch Unit's functionality is to retrieve the instructions stored in the Instruction Memory at the address supplied by the Program Counter and increment the Program Counter after each instruction fetch. This design uses a total of 75 pins of the FPGA and operates at a maximum clock frequency of 345.9 MHz without violating setup and hold time requirements.