CPU Design Project

Youzhe Dou, Yulong Wang, and Howard Xu
Electrical and Computer Engineering Department
Johns Hopkins University
Baltimore, MD, United States

***Abstract*—A 24-bit Central Processing Unit (CPU) completed with a Register File, an Arithmetic and Logic Unit (ALU), and a Control Unit is presented. The design is a classic Von Neumann architecture comprising a simple data path. All simulations are completed using a state-of-the-art CAD environment provided by CADENCE Design System, which is incorporated with hardware programming tools such as VHDL. The CPU has the capability for completing multiplications for any two numbers as long as the result is under 24 bits. Other functions available in the ALU and register include Not, And, Or, Xor, Add, Neg, Sub, Load, Increment, Shift Left, Shift Right, and Clear. The CPU has a max speed of 4ns clock cycle (250Mhz), die area of 62,500 µm^2, circuit density of 88.898%, power consumption of 0.482 mw, and a figure of merit of 8.30*10^-6 Mhz/µW*µm^2. This paper serves as a report for the final project of the class 520.491/691: CAD Design of Digital VLSI Systems taught by Dr. Ralph Etienne-Cummings in Fall 2016 at Johns Hopkins University.**

Keywords—CPU; ALU; Register; 24-bit

I. INTRODUCTION

The microprocessor, also known as the Central Processing Unit (CPU), is the most essential part of a computer, and is a complete computation engine that is fabricated on a single chip. Studying the structure and working principles of a CPU remains one of the most effective way to understand computer architecture in order to make best use of the software tools and computer languages to create programs. [1][2]

Utilizing the CADENCE Design Systems provides a simple design environment free of low-level electrical issues and allows the designers to spend most of their energy on designing the system instead of worrying about low-level electrical failures and issues. [2]

II. DESIGN OF THE CPU

A. Overview

There are many different CPU architectures available. In this class, we were given backgrounds on components of an ALU as well as the data path. We designed a CPU consists of a Register File, ALU and Control Unit with a 24-bit wide data path.

The Register File is composed of AC, DR1 and DR2 registers, and allows load, clear, increment and shifts (left and right) functions.

The ALU was implemented so that it could accomplish Not, And, Or, Xor, Add, Neg (two's complement), and Sub functions.

The Control Unit completes the CPU. It has the ability to decode operations stored in the ROM and utilizes a Finite State Machine to process each operation in a few clock cycles.

B. Register File

The register files used in this project is built from basic building blocks of memory which is D-Flip Flop. These edge-sensitive block is able to hold the "old" value until the next rising edge. And the principal is shown in Fig.1. Bit slice of the 24-bit register is synthesized using behavior description in order to achieve some basic functionalities such as load and clear. Twenty-four instances of the bit slices are then linked together in top module to create the 24-bit register files which are used as DR1, DR2 and accumulator in the future. More complicated functionalities such as shift and increment are implemented in the top module for simplicity.

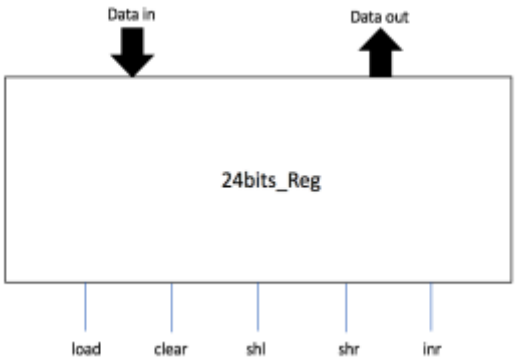

Fig. 1: 24bits_Reg

C. Arithmetic logic unit

The ALU is a combinational circuit that performs several different arithmetic and logical operations. In our case, the operations are listed in Table 1 below.

Table 1: ALU Functions

Operation	Logic	Code
NOT	AC = NOT(AC)	000
AND	AC = AND (DR1, DR2)	001

OR	AC = OR (DR1, DR2)	010
XOR	AC = XOR (DR1, DR2)	011
ADD	AC = ADD (DR1, DR2)	100
NEG	AC = NEG (two's complement, DR2)	101
SUB	AC = SUB (DR1, DR2)	110

With these seven operations listed above, the ALU handles all computations for this microprocessor. Bit slice of the ALU is designed using VHDL with three input pins (3 bits) named "mode" to determine the mode of the operations. The "mode" is decoded by control unit and corresponding task will be performed. The 24-bit ALU is built by linking twenty-four individual bit slice. NEG is achieved in top module for simplicity.

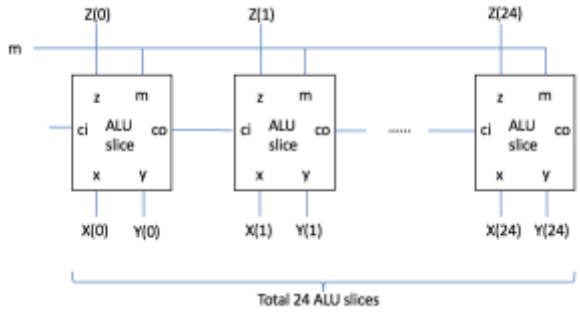

Fig. 2: ALU

D. Control Unit

The Control Unit has two main purposes in this design. The first purpose is to decode the operations stored inside the ROM. From the first 7 bits of the op-code, the control unit will decide the type of operation (register file or ALU) as well as the mode of operation (ADD or shift, etc.). Once the decoding is finished, a Finite State Machine is used to process each operation in a few clock cycles. One specific task will be done in one period and this is done by creating a clock divider which set clocks for all the units in this design and therefore they can be controlled.

As shown below in Fig. 3 is a diagram to demonstrate the opcode decoder. In a 24-bit op-code, the first seven bits are used to store information about the mode and operand while the last seventeen bits contains the address in the ROM. In detail, the first two bits are decoded as the instruction type (register file operation, ALU operation or control unit operation). The next three bits determine the mode of the operations. For example, "001" is addition for ALU and increment for register file. The next two bits works only for register file operations. It selects the operands from accumulator, DR1 and DR2. One feature in our design is that this microprocessor is able to work for tasks other than multiplications of "113 * 32345". Therefore, control unit operations such as "skip the next operation" and "jump back" are essential. This is achieved by increment PC (program counter) twice or assign a specific address to it.

Fig. 3: Op-code Decoder

Examples of Opcode used are shown below in Table 2.

Table 2: Op-code examples

Instruction	Instruction type	First 5 bits of instruction code	Input data	Output destination
MOV	Register file	00 000	From address	DR1 or DR2
INC	Register file	00 001	-	DR1 or DR2
SHL	Register file	00 010	-	DR1 or DR2
CLR	Register file	00 100	DR1 or DR2	-
ADD	ALU	10 100	-	AC
JMP	control	11 001	address	-
SNA	control	11 000	address	-

III. IMPLEMENTATION

A. Register file

The 24-bit Register File is composed of AC, DR1 and DR2 registers and is used to load, clear, increment and shift data. These functions are realized in gate level, expect for shift function which is created in top level using behavior description.

Each Bit-Slice of the register essentially uses a D-Flip-Flop (DFF) with MUXs to accomplish the four functions. As shown below in Fig. 4 and 5 are the schematic and layout of each Bit-Slice, respectively. Assuming we are using the q1 Bit-Slice, the Shift Left function essentially sets the value of q1 equal to q2. Shift Right on the other hand sets the value of q1 equal to q0. Load sets the value of q1 equal to d, data. Increment increments the value of q1 by 1, and Clear sets the value of q1 to 0.

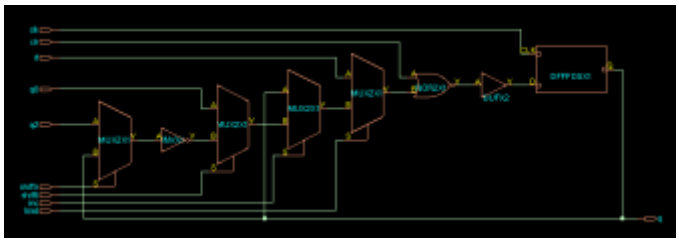

Fig. 4: Register Bit-Slice Schematics

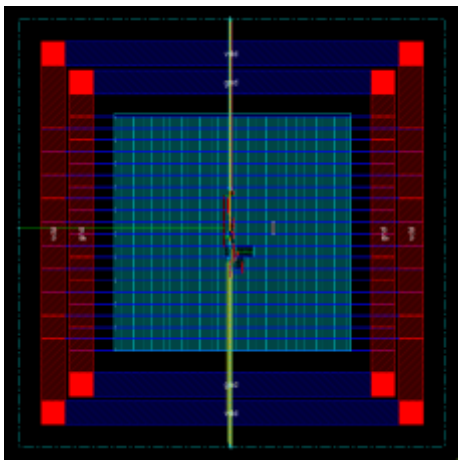

Fig. 5: Register Bit-Slice Layout

24 Bit-Slices together construct the entire Register File. Fig. 6 and 7 below show the schematic and layout of the 24-bit Register, respectively.

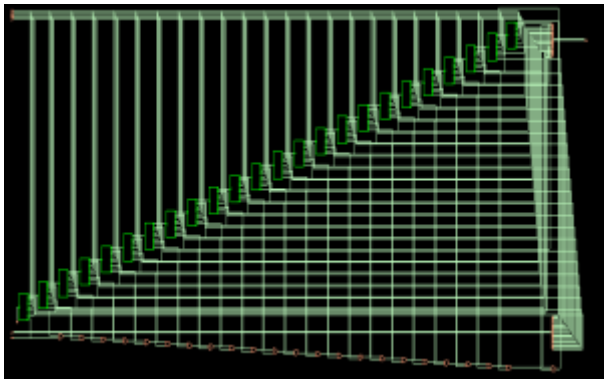

Fig. 6: 24-bit Register File Schematics

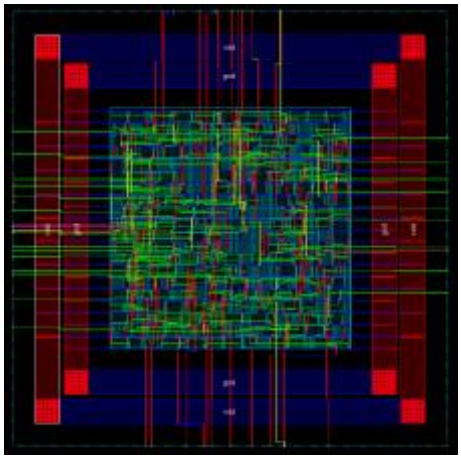

Fig. 7: 24-bit Register File Layout

Fig. 8 below shows the result of the simulation of our Register File. As shown in the graph, data d becomes 000FFF at 10ns. At 20ns, the next raising edge of the clock, with load being high, our result becomes 000FFF, showing the functionality of Load. Another load is performed at 140ns. At 40ns, shl is high, and our result changes to 001FFE from 000FFF, doubles the original value, showing the functionality of Shift Left. At 60ns, shr is high, and our result changes back to 000FFF from 001FFE, halves the last value, showing the functionality of Shift Right. Increment function was tested three times in a roll at 80ns, 100ns, and 120ns. Clear was tested at 160ns to change the result to 000000.

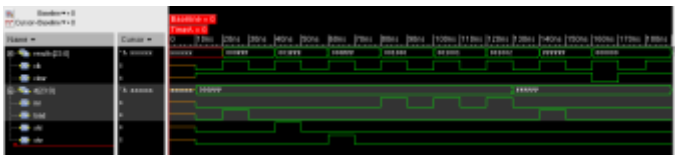

Fig. 8: Register File Simulation

B. ALU

All functions except for NEG are accomplished in the Bit-Slice level of the ALU. Fig. 9 below shows a simulation of the Bit-Slice of the ALU. 0ns to 200ns demonstrates the NOT operation; 200ns to 350ns demonstrates the AND operation; 350ns to 500ns demonstrates the OR operation; 500ns to 700ns demonstrates the XOR operation; 700ns to 950ns demonstrates the ADD operation; and 900ns to the end demonstrates the SUB operation.

Fig. 9: ALU Bit-Slice Simulation

Fig. 10 and Fig. 11 below show the schematic and layout of each Bit-Slice, respectively.

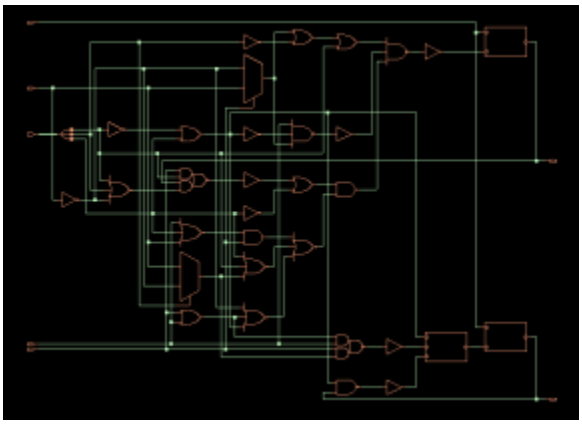

Fig. 10: ALU Bit-Slice Schematics

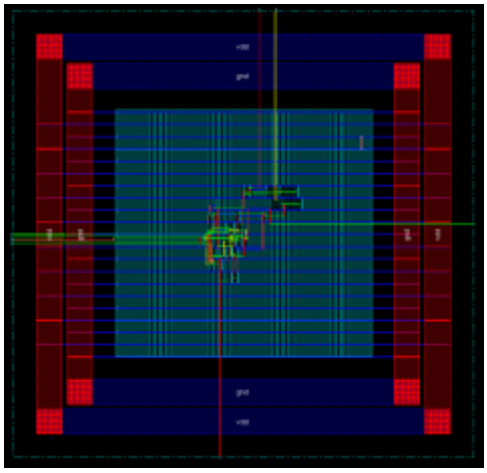

Fig. 11: ALU Bit-Slice Layout

The full ALU Simulations are shown below in Fig. 12-18, with the operations being NOT, AND, OR, XOR, ADD, NEG, and SUB accordingly.

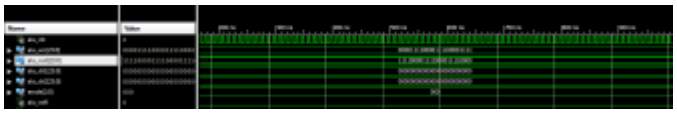

Fig. 12: ALU NOT Simulation

Fig. 13: ALU AND Simulation

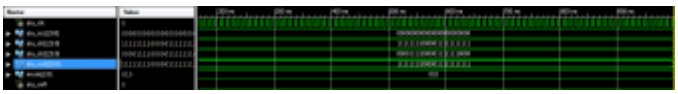

Fig. 14: ALU OR Simulation

Fig. 15: ALU XOR Simulation

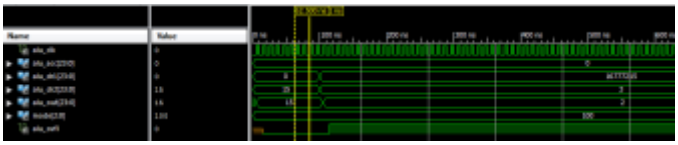

Fig. 16: ALU ADD Simulation

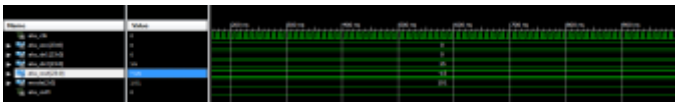

Fig. 17: ALU NEG Simulation

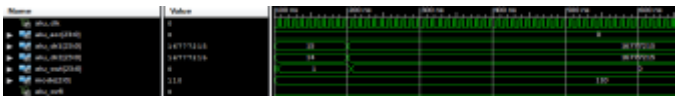

Fig. 18: ALU SUB Simulation

Fig. 19 and Fig. 20 below show the schematic and layout of the entire ALU, respectively.

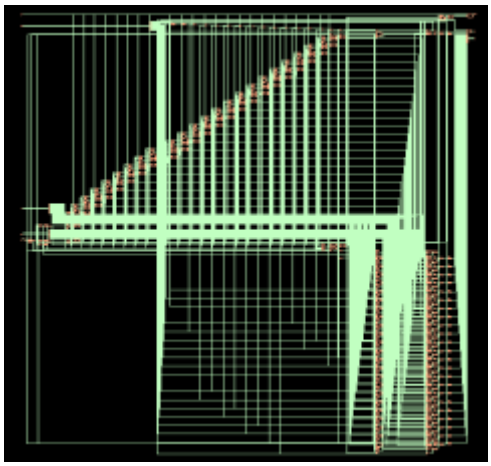

Fig. 19: ALU Schematics

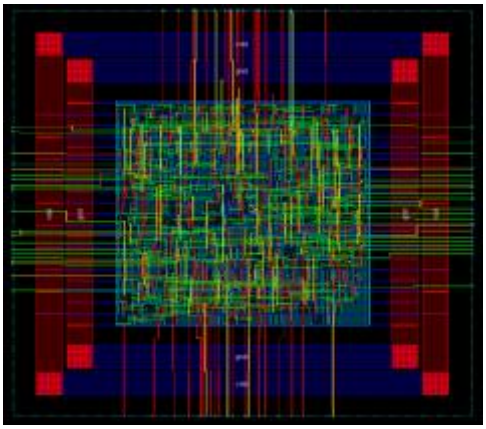

Fig. 20: ALU Layout

C. Timing control

The timing control was accomplished by dividing the master clock to creat individual clock for each functional blocks. Therefore only one functional block will be enables during one clock cycle and it is very clear to design such system using a timing flow chart as shown below in Fig. 21.

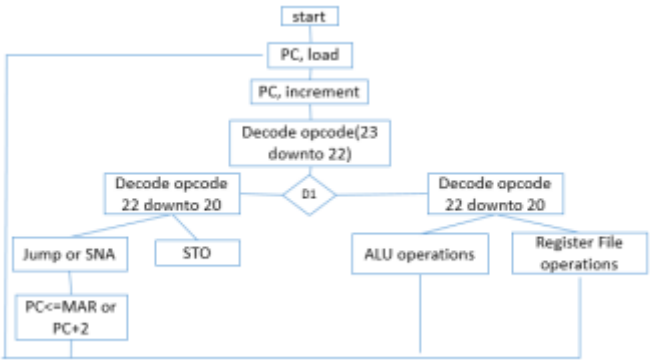

Fig. 21: Timing flow chart

An finite state machine is used here to process each op-code according to the flow chart described above. A maximum of eight clock cycles is used for one operation and there are empty cycles for easier operations such as load or increment. A demonstration of the process is shown below in Fig. 22.

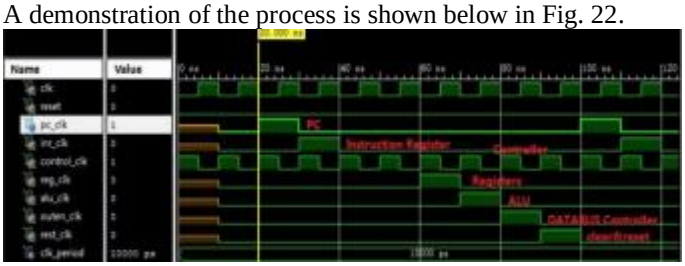

Fig. 22: Clock cycles

IV. Multiplication Function

To prove the functionality of our CPU, we ran a simulation that computes 113*32345. We utilized various functions that the Register File and ALU provides, and added a RAM as the source of our data. The system is outlined below in Fig. 23.

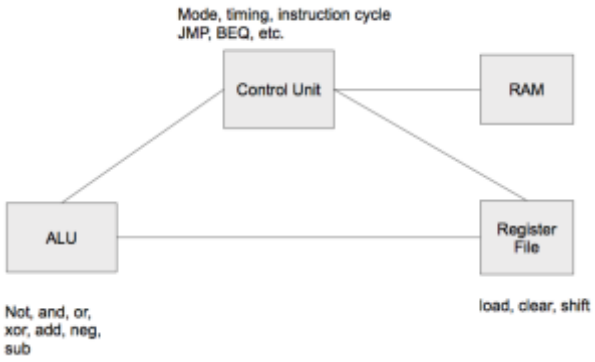

Fig. 23: System outline

A. Algorithm and Pseudo Code

Binary multiplication is accomplished using a "shift and add" algorithm. The Pseudo code for this "shift and add" algorithm is shown below, and Fig. 24 gives an example for the algorithm:

1. Load data1 into DR1
2. Load data2 into DR2
3. Copy DR2 into flag register
4. Clear DR2
5. Right shift flag register
6. If flag register pointer = 1 ->step 7, else -> step 8
7. Add DR1 and DR2
8. Left shift DR1
9. Copy AC to DR2
10. If flag register is empty -> end, else -> step 5

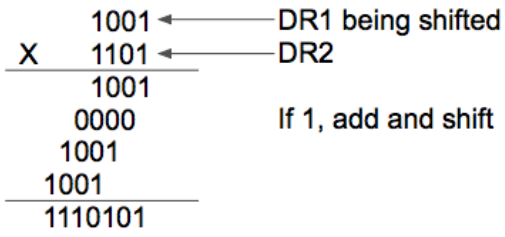

Fig. 24: Example for the algorithm

B. Translating Pseudo Code to ROM Data

("00000000000000000000000")
("00000000000000000000000")
("00000100000000000000000")--Copy data1 into DR1
("00000110000000000000000")--Copy data2 into DR2
("11100000000000000000000")--Copy DR2 into flag register
("00000110000000000000000")--Clear DR2
("11101000000000000000110")--right shift flag register
("11110000000000000000111")--BEQ1 (skip if that bit is '0')
("10100000000000000001000")--ADD DR1 DR2
("00010100000000000000000")--left shift DR1
("11000110000101010101010")--copy AC to DR2
("00000000000000000000000")
("11111000000101010101010")--BEQ2 (are all bit finished?) step 6 if 1, or end if 0

C. Results

The Simulation of this calculation is shown below in Fig. 25. We used a clock period of 100ps for the sake of simulation. The result of the calculation, 00110111110001010100 1001, is equivalent to 3654985 in decimal, which is the correct result of the multiplication 113*32345.

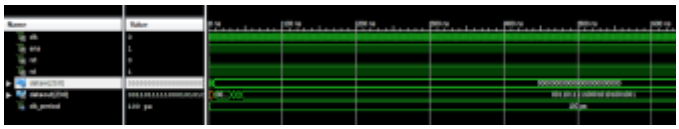

Fig. 25: Simulation of multiplication.

Fig. 26 and Fig. 27 below show the schematic and layout of the entire CPU, respectively.

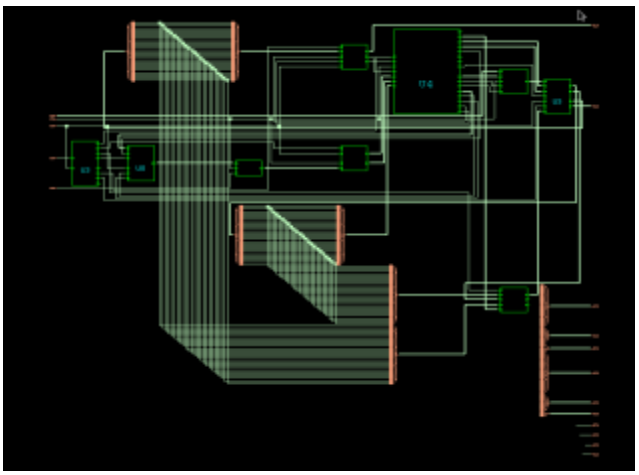

Fig. 26: CPU Schematics

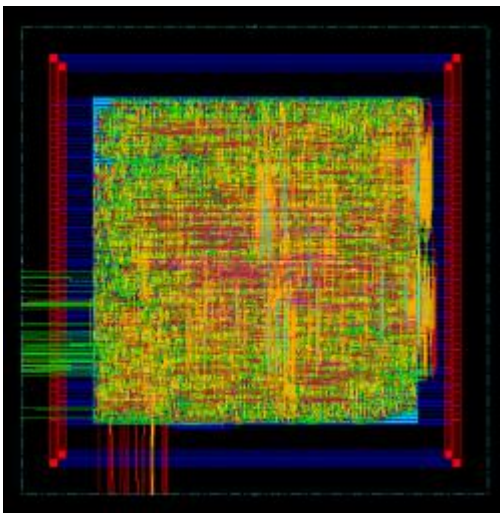

Fig. 27: CPU Layout

The CPU has a max speed of 4ns clock cycle (250Mhz), die area of 62,500 μm2, circuit density of 88.898%, power consumption of 0.482 mw, and a figure of merit of 8.30*10^-6 Mhz/μW*μm2.

V. Conclution

The project successfully designed and simulated a 24-bit Central Processing Unit (CPU) completed with a Register File, an Arithmetic and Logic Unit (ALU), and a Control Unit in a classic Von Neumann architecture comprising a simple data path. The CPU successfully performed the multiplication of 113*32345. The project significantly enhanced our team's understanding about computer architecture as well as the process an integrated circuit is designed using tools like CADENCE.

Acknowledgment

The authors would like to thank Dr. Ralph Etienne-Cummings for the teaching and guidance throughout the past semester for the successful completion of this project. We would also like to thank Adam Khalifa and John Rattray, our teaching assistants, as well as all the friends who are taking this class with us for all the help and support.

References

[1] S. Moslehpour and S. Karatalpu, "Spice modeling of ALU," Computers in Education Journal, v 18, n 1, January/March 2008, pp. 23-35.

[2] D. Schuurman, "Step-by-Step Design and Simulation of a Simple CPU Architecture," SIGCSE '13, Proceeding of the 44th ACM technical symposium on Computer science education, March 06 - 09, 2013, pp. 335-340

[3] 1. Pedroni, Volnei A. (2008). Digital electronics and design with VHDL. Morgan Kaufmann. p. 329. ISBN 978-0-12-374270-4

[4] 2. Latches and Flip Flops (EE 42/100 Lecture 24 from Berkeley) "...Sometimes the terms flip-flop and latch are used interchangeably...")