Small Microprocessor

Final Project Report

December 9th, 2023

BJ Blume, Erik Rosenkranz, Jason Rosenkranz
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: bjblume@oakland.edu, erosenkranz@oakland.edu, jrosenkranz@oakland.edu

Abstract: This small microprocessor utilizes an arithmetic logical unit, as well as a control unit, in order to process data inputted by the user. The processed data can then be stored, or shown visually to the user. A Basys 3 Artix-7 FPGA Trainer Board was used to implement the small microprocessor. The functionality of the microprocessor can be described with VHDL.

I. Overview

This report covers how a small microprocessor was created (i.e. the different components and connections) based on the knowledge of VHDL learned in ECE 2700. A picture of the different components and their connections used to make the small microprocessor can be found in the appendix (see Figure 1). This project was chosen to increase understanding and to further explore microprocessors. To expand upon the example shown in class, two additional storage registers are included, as well as developing a detailed functionality table. In addition, to simplify the complexity of inputs for the user into the microprocessor, we enabled the circuit to use signed-and-magnitude. Simply put, this project is a basic Central Processing Unit (CPU) although for completeness, a memory would need to be included.

The small microprocessor can be understood by analyzing each of its key components: the inputs, the outputs, the selector mux, the registers, the Arithmetic Logic Unit (ALU), and the control circuit. All of these components are described in more detail in the Methodology section.

II. Methodology

A. General purpose

For a simple microprocessor, the user will input a (SM) binary number from [-15,15] and control instructions. (As well as a resetting feature and an enable in w.) The input will be stored into a register, then operated on by future inputs and the Arithmetic Logic Unit (ALU). Results of this arithmetic will be output onto four seven segment displays. The output will be a hexadecimal number from [-800,7FF].

B. Inputting value

There will be five user input switches with switch 15 being the sign of the bits and switches 14 through 11 being the number. This goes into a sign and magnitude to two's complement converter, thus the possible values range from negative 15 to positive 15.

C. Outputting resultant

As seen in the diagram (Figure 2), a 12 bit bus will be transformed into an output to be displayed to 4 seven segment displays. The leftmost display will either display nothing or a negative sign, depending on if the number is negative or not. The 12 bit number will be converted into a positive number and its result will be displayed on the remaining 3 displays.

First, the value from the bus will be stored in a register. To be displayed, it will first be converted into sign and magnitude (binary). After the value is converted into sign and magnitude, both components will be used in separate hex to seven segment converters. The sign goes into the specialized hex to seven segment converter, and the remaining 12 bits of magnitude will be partitioned into 3 groups of four bits, each to be converted by use of hex to seven segment converters. The specialized converter will output CACG code for either an empty display or a negative sign, indicating the output sign. The result of the four seven segment converters will go into a MUX, and a counter will control which value is displayed, along with its corresponding seven segment display on the board through use of a decoder.

D. Storage Registers

There are four storage registers. These registers will hold values inputted by the user or resultants from certain operations. There is also a register to hold the first operand of the ALU, since the main data bus can only represent one value at a time. There is also a register to hold the answer generated by the ALU.

E. Multiplexer (Mux)

This microprocessor features a 6 to 1 multiplexor, which sends particular data to the main data bus. The six inputs to this mux are the four storage registers, the input register, and the answer register. This way, the flow of data can be manipulated by changing the selector of the mux, as well as enabling the storage registers.

F. Control Circuit

The control circuit receives new instructions from the user, which are input via nine switches on the FPGA board. Eight of these switches include the instruction for the microprocessor to perform (IR), and the ninth switch is an input w, which when activated, will accept the instruction currently set on the board. There are sixteen distinct functions included in this microprocessor. A detailed functions table is included in the appendix (see Figure 3). Depending on the function specified by the instruction, different actions will be carried out by the control circuit. For instance, operations involving the ALU will take the first four bits of IR as an operation code, and then perform the specified operation. Once an instruction is performed, an LED will light up to indicate that the action specified by the instruction has been completed. The control circuit contains a finite state machine (FSM), which controls most aspects of the microprocessor. The control circuit also contains two registers and a decoder to load in a new instruction, enable/disable the display, and to enable/disable certain registers. A detailed figure of the control circuit is shown in figure 4.

The many inputs and outputs of the control circuit are handled by the FSM. Every 10 nanoseconds, the state of the FSM is determined. Each state specifies a certain set of values to be output. Some states will occur in sequence, regardless of the inputs, since the flow of data cannot be completed in just one clock cycle. The ASM diagram for this FSM is shown in Figure 5.

As an example, 4 clock ticks are required to add two values in the storage registers. The first clock tick uses the MUX to select the first operand, and to enable the operand register. The second clock tick uses the MUX to select the second operand, send the op code to the ALU, and enable the answer register. The third clock tick uses the MUX to select the answer and enable the register where the result is to be stored. After the fourth clock tick, the answer will be stored in the correct storage register.

G. ALU

The aforementioned partition of IR will determine a specific operation of the ALU. These instructions will be passed only in certain states of the microprocessor, as outlined by the FSM State diagram. One can also note that A and B go into the ALU which is controlled by the input op. Op is determined by the control circuit and is then converted to sw. The output of the ALU then goes into the ANS, which is controlled by the input E_ans as determined by the control circuit. The mux is also controlled by the control circuit and has a sw size of three bits. The control circuit has an input w and IR which are determined by the behavior of the microprocessor. Once everything is completed in the control circuit, the done bit is set and lights up an LED (Light Emitting Diode).

III. EXPERIMENTAL SETUP

To simulate the code created for this project, Vivado 2019.1 was used with exact specifications matching those of the ECE 2700 laboratory. Doing this enables us to confirm the interconnections and functionality between components. For full analysis on our simulations, please refer to the VHDL code provided. In addition to simulating the testbench, a bitstream of the code was also generated and uploaded to the FPGA board. This way, the functionality of the microprocessor could be analyzed in real-time. The expected results are that the small microprocessor performs the operation based on the user input and the data in the registers. The discussion of the simulation results and if the correct results were found, is in the results section.

IV. *RESULTS*

In looking at the example operation simulation graphs, one can see the results of storing values into two different registers with one number being positive and the other being negative. One can also see an operation of the absolute value completed (calculated by the ALU). Each of the operations takes several clock cycles to perform, as discussed previously in the methodology section. In real-time, however, these operations will appear instantaneous to a human user. A link to a live demonstration of the microprocessor is included in the references section. All of the results obtained are as expected, as each of the functions was verified to be working on the FPGA board.

This project allowed us to utilize many of the topics learned in the ECE 2700 course. The input and output modules utilize knowledge of different number systems, and how to convert between them. The ALU utilizes many of the arithmetic operations studied, as well as some of the knowledge from the lab component of this course. The control circuit utilizes the concept of a FSM to control the flow of data, and carry out complex synchronous processes.

V. CONCLUSIONS

Creating a small microprocessor was a sufficient introduction to using VHDL code through Vivado to create a larger scale project. General functionality of the microprocessor and the steps to program it were better understood through the completion of it.

In creating the microprocessor, two errors found during real time testing were solved by identical solutions. By containing input w and the Q output of the modulo-4 counter into appropriately sized registers, influence of w outside of stage one and the skipping of lit segments AN(1) and AN(3) were respectively avoided.

To improve this project even more, use of a keyboard as input would ease the arduous task of flipping up to 15 switches on the Artix-7 FPGA Board. More functions and higher functionality would also enhance the capabilities of the microprocessor, as well as adding more storage registers.

VI. REFERENCES

[1] Example video of project working on board, link to video demonstration: https://youtu.be/S45skjf9Sfw

[2] Daniel Llamocca Obregon, National Science Foundation, “VHDL Coding for FPGAs” VHDL Coding for FPGAs (oakland.edu)

APPENDIX

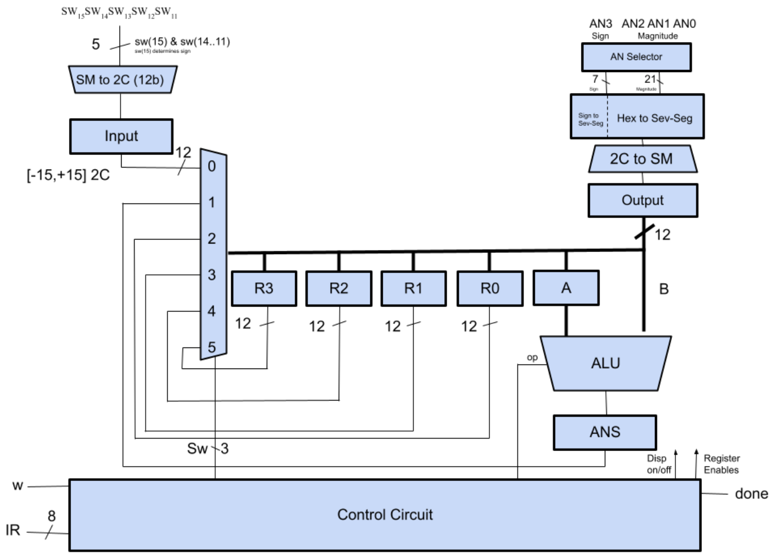

Figure 1: Block Diagram of Microprocessor
For visual simplicity, not all internal signals are shown above (e.g. enable).

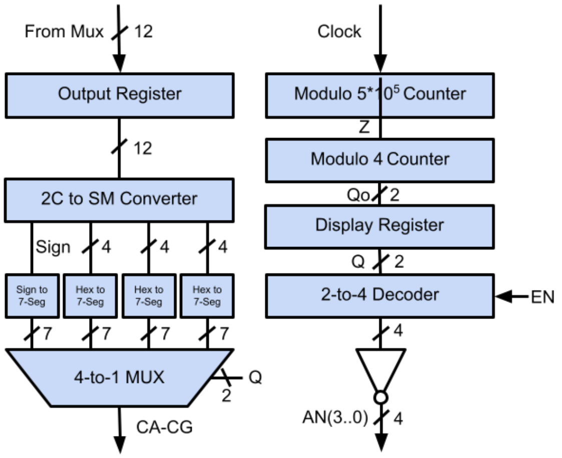

Figure 2: Output Module Diagram

General and Arithmetic Operations		**Bitwise Operations**	
IR Code	**Functionality**	**IR Code**	**Functionality**
$0000R_{A1}R_{A0}XX$	Store input into R_A	$1010R_{A1}R_AR_{B1}R_{B0}$	R_A=R_A OR R_B
$0001R_{A1}R_{A0}XX$	Display the value of R_A on 7-seg display	$1011R_{A1}R_AR_{B1}R_{B0}$	R_A=R_A AND R_B
0010XXXD	7 segment on/off (D = 1/0)	$1100R_{A1}R_AR_{B1}R_{B0}$	R_A=R_A XOR R_B
$0011R_{A1}R_AR_{B1}R_{B0}$	Copy the value of R_B into R_A	$1101R_{A1}R_AR_{B1}R_{B0}$	R_A=R_A NOR R_B
$0100R_{A1}R_AXX$	$R_A=R_A+1$	$1110R_{A1}R_AR_{B1}R_{B0}$	R_A=R_A NAND R_B
$0101R_{A1}R_AXX$	$R_A=R_A-1$	$1111R_{A1}R_AR_{B1}R_B$	R_A=R_A XNOR R_B
$0110R_{A1}R_AXX$	$R_A=-R_A$		
$0111R_{A1}R_AR_{B1}R_{B0}$	$R_A=R_A+R_B$		
$1000R_{A1}R_AR_{B1}R_{B0}$	$R_A=R_A-R_B$		
$1001R_{A1}R_AR_{B1}R_{B0}$	$R_A=\|R_A-R_B\|$		

*R_A and R_B are two-digit codes. Since there are four registers, they can be referenced using 00, 01, 10, or 11
*X specifies "don't care" conditions (they can be either 0 or 1, and it will not affect the functionality

Figure 3: Control Circuit Functions Table

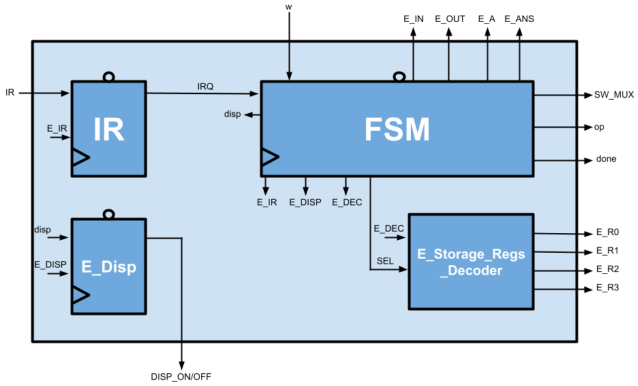

Figure 4: Control Circuit Diagram

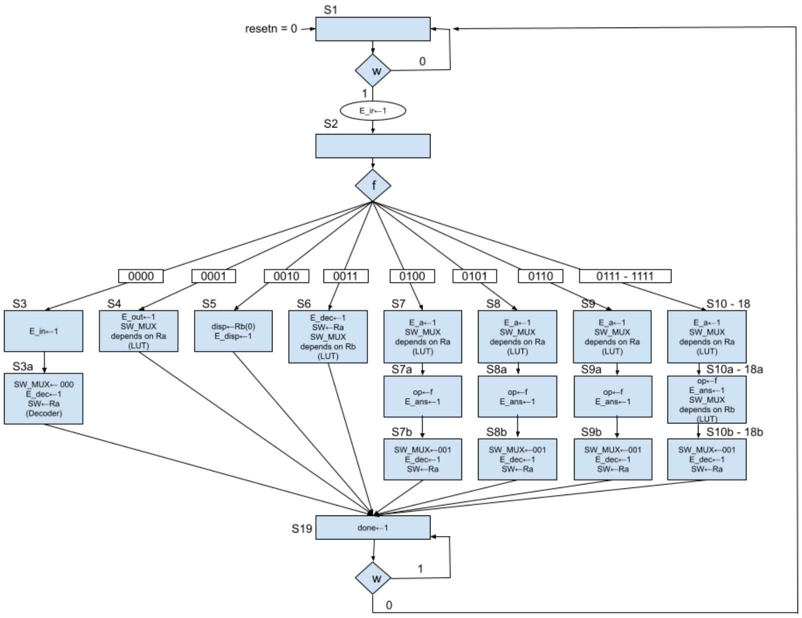

Figure 5: ASM Diagram (for the FSM in the control circuit)

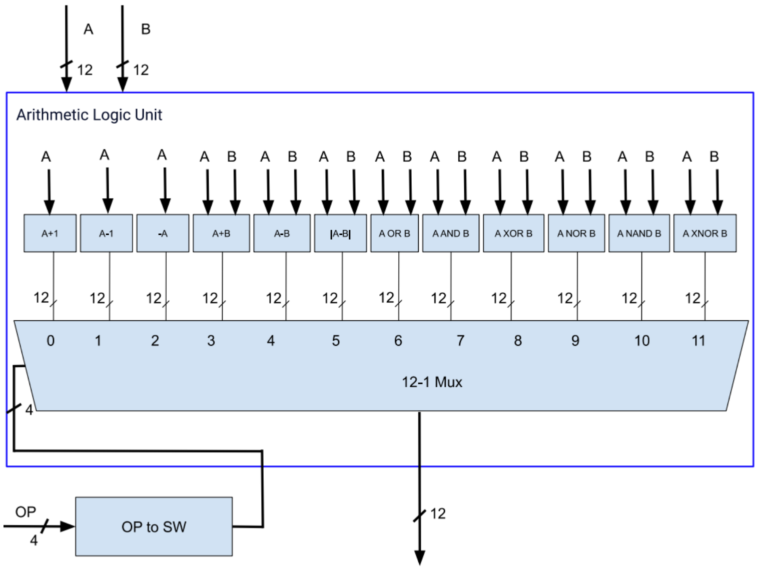

Figure 6: Arithmetic Logic Unit (ALU) Diagram

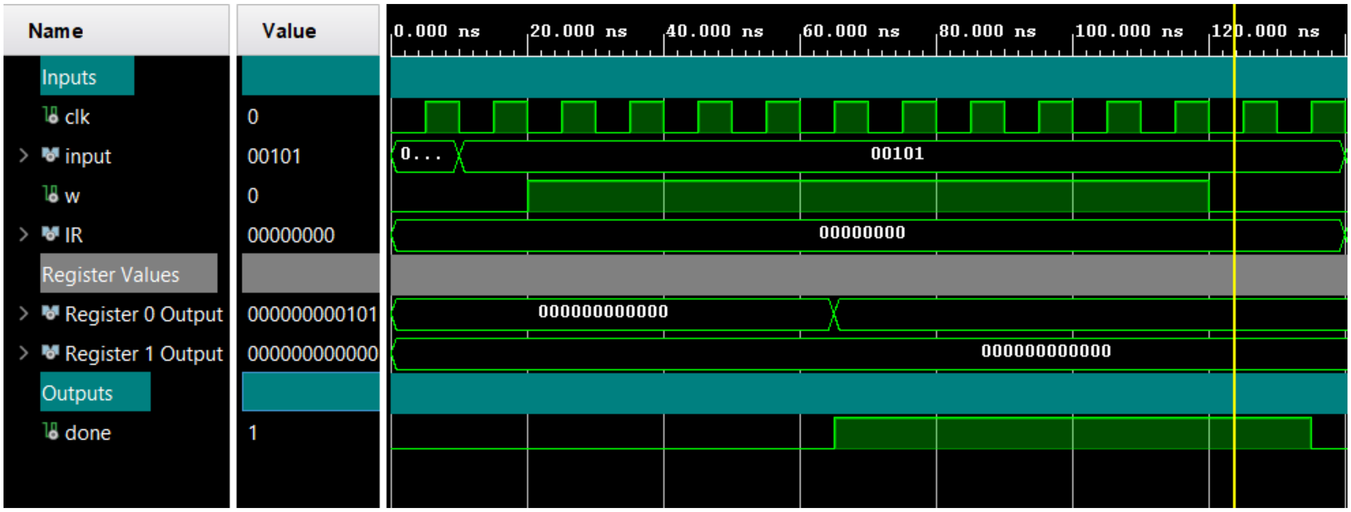

Figure 7: Example Operation Simulation Store +5 into Register 0

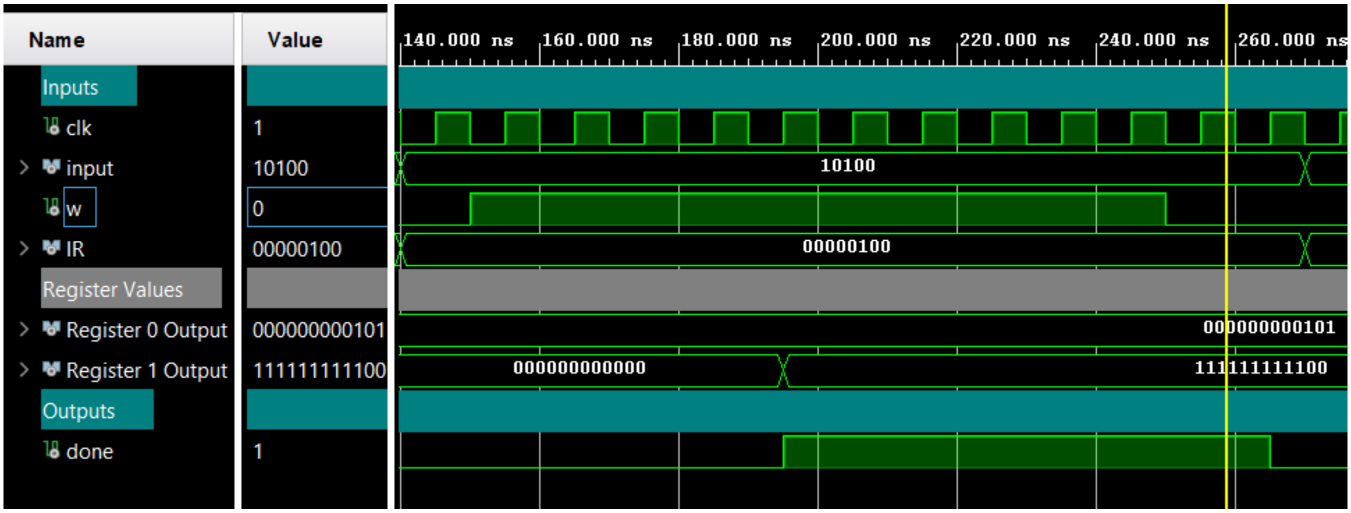

Figure 8: Example Operation Simulation Store -4 into Register 1

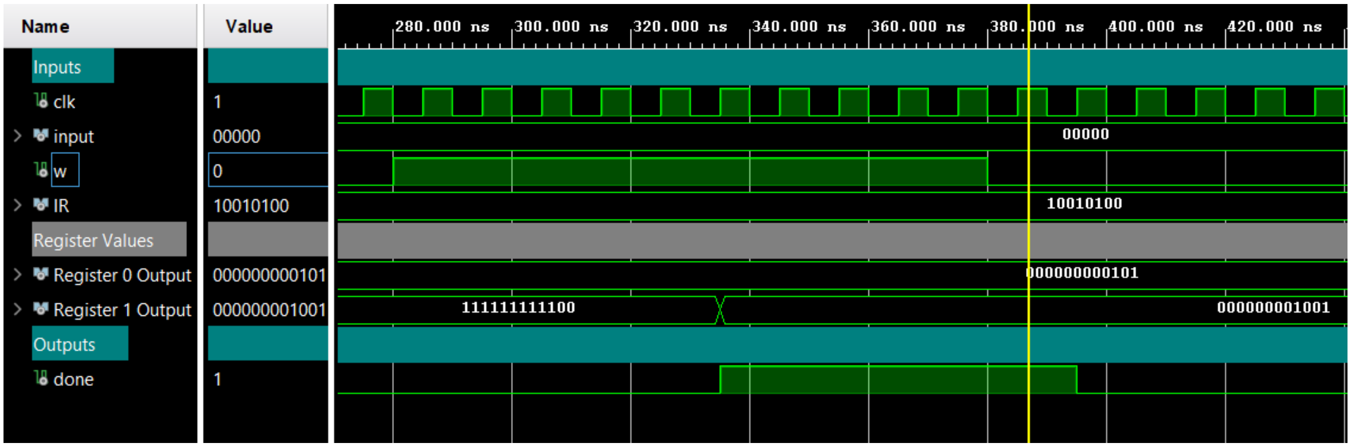

Figure 9: Example Operation Simulation Register 1 = |Register 1 - Register 0|