8-Bit Calculator

Final Report

List of Authors (Jaskaran Singh, John Nasir, Nhi Vu, Alyssa Lafever)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: jaskaransingh@oakland.edu, jnasir@oakland.edu, nvu@oakland.edu, anlafever@oakland.edu

***Abstract*—The purpose of this project was to build an eight-bit calculator. The calculator is able to perform addition, subtraction, multiplication, and division of signed eight-bit numbers. The numbers and math operator are inputted through a keyboard and the output is displayed on the seven-segment display on a Nexys board.**

I. Introduction

The goal for this project was to create a simple calculator that performs basic math functions such as addition, subtraction, multiplication, and division of signed eight-bit binary numbers. The motivation for the project was to use skills learned from the Digital Logic class to create a useful device on an FPGA.

Creating the calculator involved using several topics that were covered in the Digital Logic class. These topics include: reading inputs from a keyboard, using registers to store data, converting numbers from binary-coded decimal (BCD) to binary, performing signed binary math operations, displaying numbers on a seven-segment display, and creating a finite state machine (FSM) to control the data path circuit.

Additionally, creating the calculator involved using a topic that was not taught in class and had to be learned while creating this project. This topic was converting numbers from keyboard keycode to two's complement binary.

II. Methodology

The calculator was designed using structural VHDL. Using this method involved dividing tasks that the calculator performs into components that do smaller sub-tasks. These sub-tasks are performed by easily designable digital components, such as adders, subtractors, multipliers and dividers, that would be created by individual team members.

The components in the top level of the structural VHDL design are shown in the block diagrams of Figure 1 and Figure 2 at the end of this report. These figures had to be shrunk to fit them in this report, so the reader will need to zoom in to view the diagrams.

Users are able to interact with the calculator by feeding an input from a keyboard and viewing the results of their desired math operation on a seven-segment display on a Nexys board.

A. FSM

The FSM controls all of the data path components in the calculator. When the user inputs a number, operand A, and presses the Enter key, the FSM sends a signal to a demultiplexer and RegisterA to save the signed binary value of operand A. All of the register's in this project were based on Dr. Llamocca's register design [1]. The FSM then clears the DigitSaver, a five-bit input shift register to prepare the data path circuit for the next input. Simultaneously, the FSM sends a signal to convert operand A to its BCD value and display its signed decimal value on the seven-segment display.

The user can then input the math operator. When the user inputs the operator and presses the Enter key, the FSM sends a signal to save the operator as an encoded two-bit binary number in OpSave, a two-bit register. Simultaneously, the FSM clears the input shift register and display.

Next, the user can input a second number, operand B. When the Enter key is pressed after inputting operand B, the FSM will send signals to save the value for the signed binary value of operand B in RegisterB. The FSM will then clear the input shift register and send a signal to convert the saved signed binary value of operand B into BCD and display its signed decimal value on the seven-segment display.

The user can then press the Enter key again to tell the FSM to send a signal to complete the math operation. After the math operation has been completed and the result has been saved, the FSM sends a signal to convert the signed binary result to BCD and display its signed decimal value on the seven-segment display.

When the user next presses the Enter key, the FSM clears the display and all of the other values saved in the registers so it can return to its original state. Once everything is clear, the user can start inputting a new math operation.

B. Input

The first component, my_ps2keyboard, of the calculator receives the number or operator from the keyboard. This component was adapted from Dr. Llamocca's design [1]. Each key that is pressed on the keyboard will send a serial eight-bit keycode which the component outputs simultaneously through an eight-bit bus. The keycode for the possible inputs are listed in Table 1.

This eight-bit keycode is then saved into the DigitSaver input register. If the data represents a number, the keycode is then converted to a signed eight-bit binary number. This binary number is then saved into the appropriate operand register after the Enter key is detected in the EnterDetector. This procedure is followed for both of the operands.

Table 1. Keycodes for Keyboard Keys

Key	Hex Keycode	Keycode
0	45	0100 0101
1	16	0001 0110
2	1E	0001 1110
3	26	0010 0110
4	25	0010 0101
5	2E	0010 1110
6	36	0011 0110
7	3D	0011 1101
8	3E	0011 1110
9	46	0100 0110
+	55	0101 0101
-	4E	0100 1110
x	22	0010 0010
/	4A	0100 1010
Enter	5A	0101 1010

If the data in the DigitSaver is an operator, the OpEncoder converts the eight-bit keycode into a two-bit binary number which represents the appropriate math operation. Once the EnterDetector detects the keycode for the Enter key, the two-bit binary number is saved into OpSave, a two-bit register.

C. Adder

In a one-bit full-adder, the sum of two bits is calculated by taking the XOR of the input ports X, Y, and carry-in. The carry out is calculated by the following equation.

$$\text{Carry Out} = X.Y + X.(\text{Carry In}) + (\text{Carry In}).Y$$

The full-adder then outputs the one-bit result of the addition and the carry-out bit.

For the nine-bit adder, there are two inputs, A and B, and one output R. By linking nine full-adders through their carry-in and carry-out ports, it becomes possible to add two sign-extended eight-bit signed numbers. This sum is then sign extended to give a 16-bit output.

D. Subtractor

The subtractor is similar to the adder. The main differences between the subtractor and the adder are that there is a "not gate" that inverts the B input and the carry-in for the subtractor is 1 rather than 0. These differences between the subtractor and adder allow operand B to be subtracted from operand A while using a design that is similar to the adder.

E. Multiplier

For the multiplier, first two signed eight-bit numbers are inputted and are converted into their eight-bit unsigned magnitude. After this process, the eight unsigned bits go into the multiplication function and are outputted as the 16-bit unsigned result of the multiplication of the magnitudes of the two inputs. This output is then inputted into an XOR gate with the XOR of the most significant bit (MSB) of each eight-bit input. This output is used as the input to a 16-bit adder with a carry-in equal to the output of the XOR of the MSB of both signed inputs. The other input to the adder is a 16-bit zero. This adder turns the unsigned result of the multiplication into the signed 16-bit result.

F. Divider

The last math operation component is a divider. The divider operation is closely based on lab six; however, in the project, the number of bits for the inputs are changed to eight and the number of bits for the result output is 16. There is also an enable input to start the division operation and an output to signal when the division is complete.

Two signed eight-bit numbers are inputted and their magnitude is taken to convert them to unsigned. After the conversion, one input will go into a normal register and the other input will go into a left shift register. The divider operation includes an FSM, register, an n-bit adder, another left shift registers: one that has a synchronous clear in it, and a counter. The result of the division is an integer only, the remainder from the division is not shown.

G. Math Unit

The math unit is the top-file for the four math operators, addition, subtraction, multiplication, and division. It contains two eight-bit inputs for operands A and B, a two-bit input to indicate which math operation needs to be completed, and a one-bit input, EM, used to enable the divider.

The eight-bit signed binary numbers A and B go into the math unit and undergo addition, subtraction, multiplication and division simultaneously to obtain four 16-bit results.

A four-to-one bus multiplexer is then used to output the result of the desired math operation. The inputs to the multiplexer are the results from each math operation. The selector of the multiplexor is the two-bit operator input that indicates which 16-bit result is to be outputted.

The 16-bit output of the multiplexor is the desired results and is saved into Register R after the divider outputs a done signal through done_D. This done_D signal is used to enable the register because the division operation takes the longest time out of all of the math operations. Thus, every math operation is complete and has a result that is ready to be saved by the time that done_D signals that the division is complete.

H. Output

The final step of the calculator is to display the numbers inputted and the final answer on the seven-segment display on the Nexys board. The result, R, and Operands A and B are inputted into a multiplexor. In order to input Operand A and B in the multiplexor, the eight-bit binary numbers need to be sign extended to become a 16-bit binary number. Thus, the output of the multiplexor is always 16 bits. The output of this multiplexor is determined by the FSM.

The multiplexer's signed 16-bit binary number output goes into an XOR gate along with the MSB of the number.

This XOR flips the bits from 1 to 0 and vice versa if it detects that the number is negative. This result is then inputted into an adder along with a 16-bit zero and a carry-in equal to the MSB of the signed 16-bit number. The output of the adder is the magnitude of the signed 16-bit number. Also, the MSB of the signed 16-bit number is saved into a D Flip Flop.

The output of the adder is then inputted into Dr. Llamocca's bin2bcd, a component that converts the number from binary to BCD [1]. This component has a output that indicates when the conversion is done and a 24-bit output that outputs the number in BCD. The 24-bit outputted number then goes into a register to be stored.

Since only seven bits can be outputted to the seven-segment display at any time, only one seven-segment digit will be lit at a time. Thus, and modified version of Dr. Llamocca's serializer with a built in BCD to seven segment display decoder is used to accomplish this [1]. The 24-bit output of the 24-bit register and the output of the D Flip Flop then go into the serializer to be displayed onto the seven-segment display. All of the digits of the number will be cycled through quickly by the serializer by turning displays on and off while outputting only the data for the seven-segment display that is on. The numbers will be cycled fast enough so that it looks like they are all being displayed at once.

III. Experimental Setup

For this project, the software, Vivado and hardware components, Nexys A-7 and a keyboard, were used to test and verify the functionality of the calculator. In Vivado, a testbench was created to simulate some of the components of the calculator and a testbench was created to test the overall design. After simulating, the Nexys board was programmed and the calculator was tested with the keyboard through a USB connection. The outputs were displayed on the seven-segment display of the Nexys board.

IV. Results

The adder and subtractor were tested by using a testbench to simulate the output. For the adder, the cases that were tested were the ones that were believed would easily show whether the adder functioned properly and the ones that would most likely give an incorrect result. This included cases that would result in overflow during the addition of the eight-bit inputs. Through testing, it was found that the output was correct in each case. For the subtractor's testbench, cases that were similar to that of the adder's testbench were tested. In each case, it was found that the answer was correct.

The multiplier was tested by using a testbench to simulate the result of all allowed inputs. At first the multiplier wasn't functioning properly. The outputted result of the multiplications involving one negative number were one number higher than the expected result. After troubleshooting the code, the multiplier successfully outputted the correct answer for each input. The divider also had its own test bench that was similar to that of lab 6. The results of the divisions were correct.

When testing the full calculator with a test bench, it was discovered that the negative sign didn't display on the seven-segment display. After trying some troubleshooting techniques to isolate the problem and seeking guidance from the instructor, it was discovered that the problem was due to missing two ports in the port map of the FSM of the calculator.

The calculator does have a few constraints for what inputs it can handle. The calculator can only handle inputted operand with values between -128 and 127. Also, the calculator cannot handle an input of zero or inputs that will result in an answer of zero. Additionally, when inputting a negative number, it must be entered in as a three digit number (e.g. -15 should be entered as -015).

Conclusions

The math unit of the calculator was created and simulated successfully. The keyboard input into the calculator and the resulting output onto the Nexys board were also successfully designed. Some of the code did have errors at first but they were resolved so that the calculator functioned properly.

It was more challenging than expected to design the input and output components. Many different designs were created before the functioning final one was discovered.

After the data path circuit design was selected, there was some challenge with figuring out how the FSM needed to function. These challenges were overcome by drawing a box to represent the FSM that had arrows pointing at or away from the box for every input and output. A version of this box is part of the block diagram in Figure 1.

The states were then defined and listed in Table 2, along with notes that showed which conditions would cause errors. An algorithmic state machine was then made with the aid of the box drawing and the state definitions in Table 2. Writing the code for the FSM was simple after that and required little troubleshooting.

Overall, after working together as a team and solving problems systematically, the input, math unit, output, and FSM were able to be designed successfully to create an eight-bit calculator.

Table 2. State Definitions

State	Description	Notes
S1	Receive Operand A	
S2	Save Operand A	We'll be stuck in S2 if Operand A is 0
S3	Display Operand A on 7Seg; Receive Operator, Save Operator	
S4	Receive Operand B	
S5	Save Operand B	We'll be stuck in S5 if Operand B is 0
S6	Display Operand B on 7Seg; Receive Calculate Command	
S7	Execute Math Operation, Convert Output to BCD	We'll be stuck in S7 if Result R is 0
S8	Display Output on 7Seg	

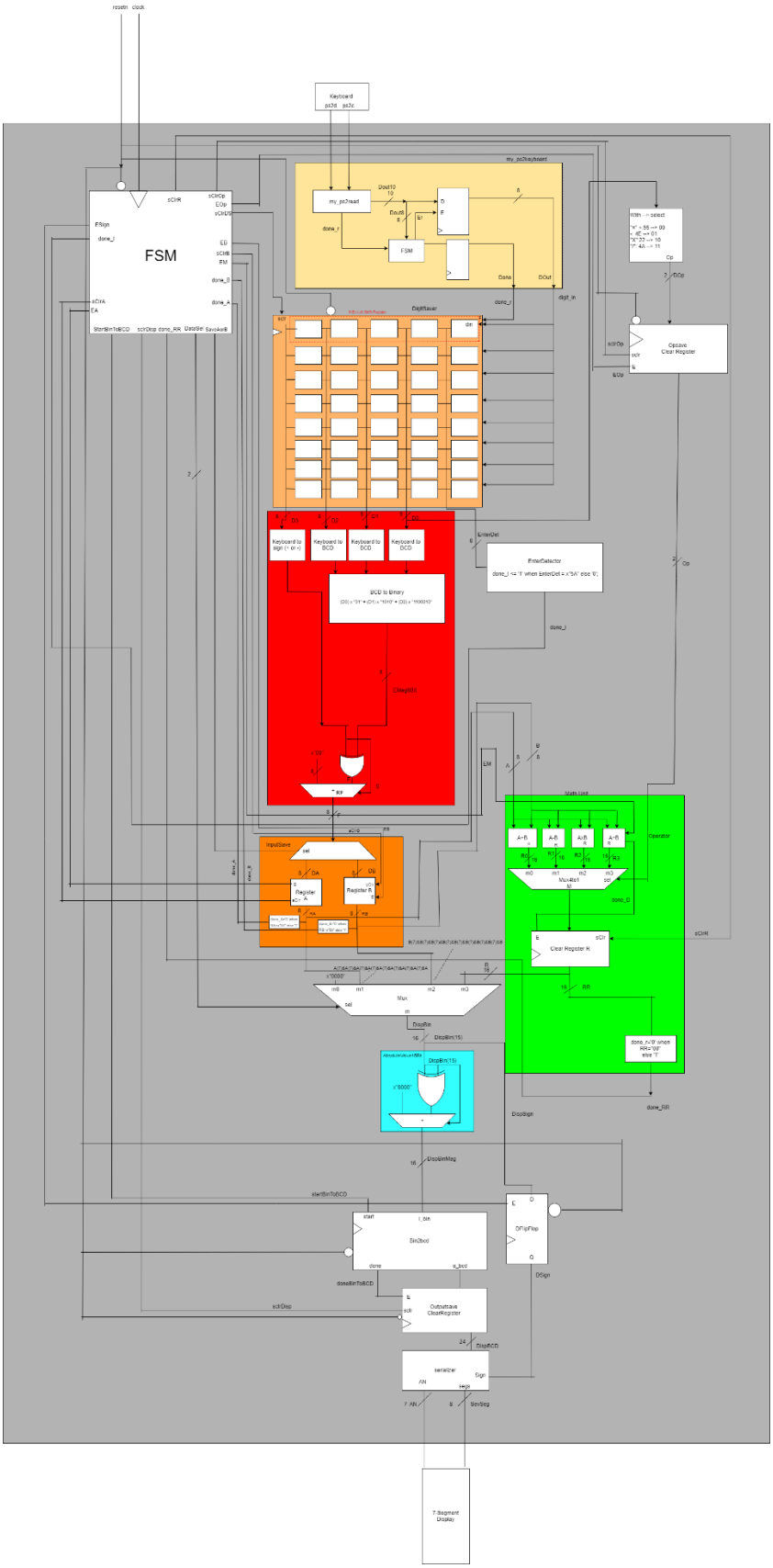

Figure 1. Block Diagram of 8-Bit Calculator

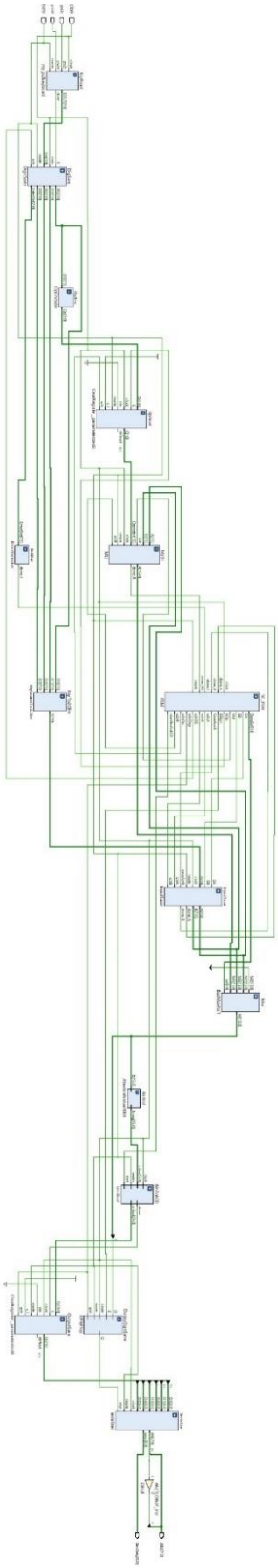

Figure 2. Vivado's Block Diagram of 8-Bit Calculator

REFERENCES

[1] D. Llamocca “VHDL Coding for FPGAs,” Reconfigurable Computing Research Laboratory, http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html