Simple Signed Calculator

ECE 2700 – Final Project Report

List of Authors (Gabriella Juncaj, Jarrod Fortner, Peter Younan)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: gabriellajuncaj@oakland.edu, jmfortner@oakland.edu, pyounan@oakland.edu

***Abstract*— This project involves designing and implementing a Simple Signed Calculator that will perform addition, subtraction, multiplication, and division on two 8-bit signed numbers. The inputs are put into switches, and results are displayed in BCD format (with a sign) on 7-segment displays. This report provides an overview of the project.**

Introduction

The purpose of this project is to build a calculator that works with signed 8-bit numbers, showing how digital logic design concepts can be used in real life. It covers important topics like 2's complement arithmetic, Hex decoding, and hardware interfacing, all of which cover what we've learned in class. Also, it gives us hands-on experience with FPGA implementation, helping us better understand how hardware design and simulation work.

This calculator could have real use in embedded systems where basic arithmetic operations are needed. This project has also been a great opportunity for us to work as a team, combining what we've learned in theory with practical problem-solving.

Methodology

A. Calculator Design

The calculator includes the following components:

- Input Interface: Two 8-bit signed numbers and operation selection entered via switches.
- ALU (Arithmetic Logic Unit): Performs addition, subtraction, multiplication, and division.
- Control Unit: Decodes the operation and directs data flow.
- Output Interface: Displays the results in Hex format (with a sign) on 7-segment displays.

B. Block Diagram

We created a block diagram (figure 1) illustrating the system architecture, showing connections between the switches, ALU, control unit, and 7-segment displays. Vivado will be used to simulate individual components, ensuring functionality before integration. We also implemented the code onto the Nexys Board (figure 2) to perform calculations.

Figure 1: Block diagram

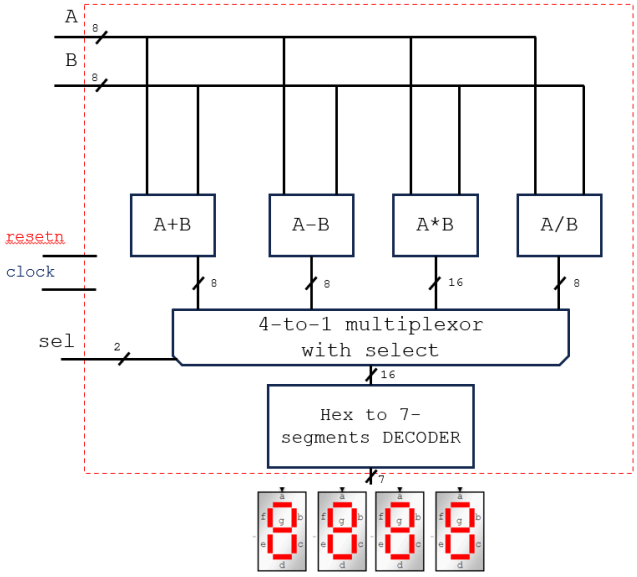
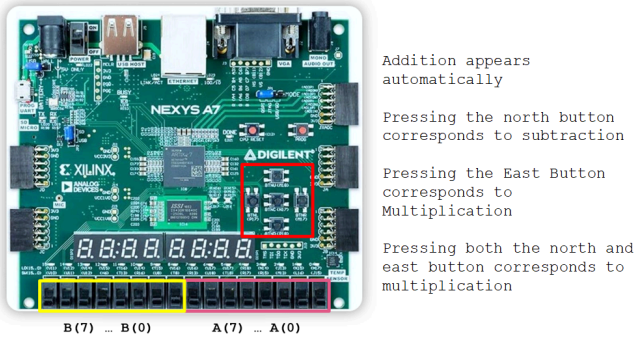


Figure 2: Operations



C. Adder and Subtractor

Both the adder and subtractor operations utilize a nearly identical structure with eight full adders. For the subtraction operation, the xor gates are used to invert one operand based on the control signal - if the control signal is 0, the second operand passes unchanged while if it 1, subtraction is performed using the two's complement method. The control signal in this instance is the port "addsub" which is a single bit input that

determines whether the operation being performed is addition - addsub = 0 - or subtraction - addsub =1. For the calculator, because the adder and subtractor were built as separate components, by default addsub = 0 for addition and 1 for subtraction.

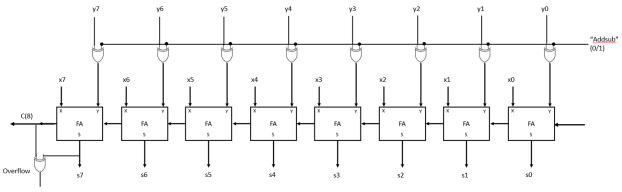
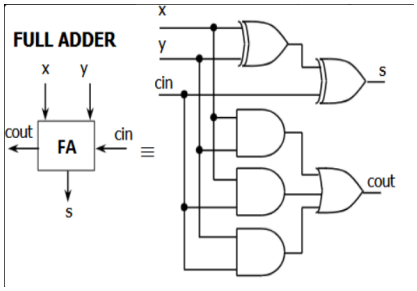


FIGURE 3:ADDER & SUBTRACTOR

FIGURE 4: INDIVIDUAL FULL ADDER COMPONENT



D. Multiplier

When multiplying two signed 8-bit numbers, the sign of the result depends on the signs of the inputs. If both positive, result is positive. If one number is positive and the other is negative, the result is negative. Finally, if both numbers are negative, their signs cancel out, and the result becomes positive.

To handle this behavior, the system first ensures the inputs are converted to their positive forms using a 2's complement (2C) process (figure 4) when necessary. This is achieved by analyzing the most significant bit (MSB) of each input. If the MSB is "0", the number is positive and sign-extended to maintain consistency. If the MSB is "1", the number is negative, so 2C is applied to obtain its absolute value. Once converted, the inputs become 9-bit positive representations.

The unsigned array multiplier (figure 5), extended from a 4-bit multiplier used in a prior lab [1], computes the product of these absolute values. At this stage, the result is always positive, regardless of the original input signs. To ensure the final output has the correct sign, the MSBs of the original inputs are revisited. If the inputs had differing signs, the result is converted back to a negative value using 2C.

The system also includes key components such as input registers to store the numbers, a clock/reset module to synchronize operations, and a 2C unit to handle signed numbers. This design ensures a correct and efficient 16-bit signed output for all input combinations.

FIGURE 5: 2'S COMPLEMENT

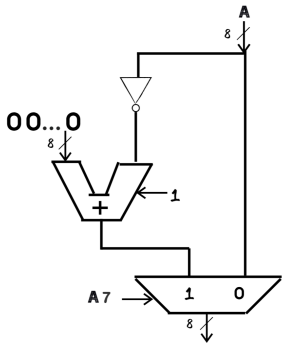
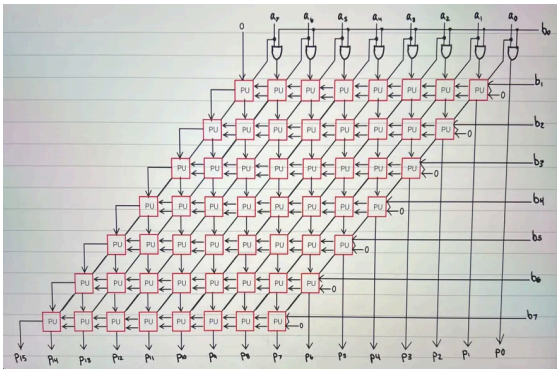


FIGURE 6: UNSIGNED ARRAY MULTIPLIER



E. Divider

In this design (figure 6), we decided to create the component of the division operation that would be computed only if the inputs of the component were unsigned numbers. If both of the inputs had the same sign, they would not need to use the 2's complement for the output numbers as they stay unsigned.

The components that were used in this design each had an important role in the process and it would not work correctly if any of them is missing or wrong.

The register was used to hold the value of the input B and the shift registers were used to hold the values of their inputs and would shift out the next bit based on the

state of the FSM. The adder was used to add the results based on the inputs it would receive. It would also feed the output back into the input so it would be accumulating the results. The counter was used to count up to the number of times the operation had been complete and would do that until it reached the number of bits that the inputs were.

The FSM was used to control the circuit and would send signals based on the inputs and state that it was in. The states are as shown in figure 4. If it received the start signal, the FSM would then move onto the next state. In state 2, It would repeat the process until the adder outputs a 0, which would be the result of the feedback of the adder. After that it would do that until the counter has reached the number it goes up to and that will make that operation complete and move to the next state.

FIGURE 7: DIVIDER FSM

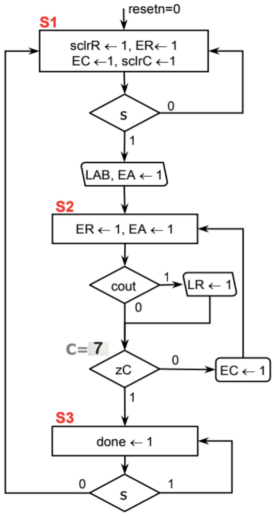
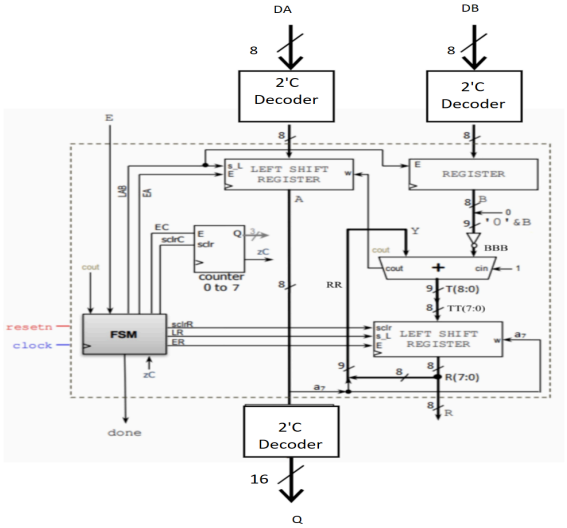


FIGURE 8: UNSIGNED DIVIDER



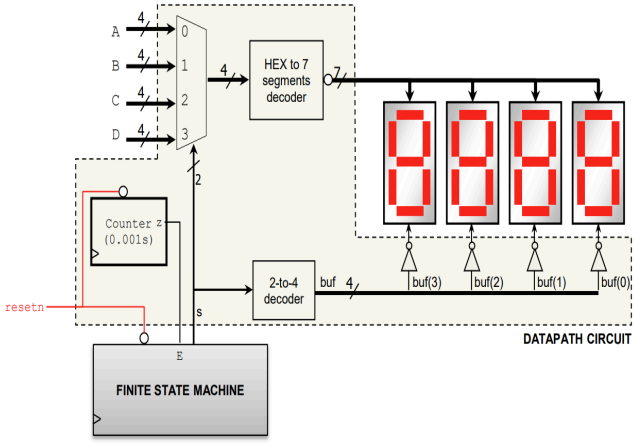
F. *SERIALIZER*

For displaying the output of the operation that you select, you need to be able to display different numbers on each of the segments of the display. That is where this component comes into play (figure 8). The serializer gets the input signal and breaks it down into parts and displays the value of the part that is selected.

The MUX being used in the divider is getting a input signal that is 16-bits and is getting split up 4-bits per MUX input with the select being 2-bits. The clock was set to 1ms so that it could flip through the lights fast enough that they look like they are staying on, but not fast enough that the segments would look like they are flickering.

The Hex to 7 segment decoder would get a 4-bit output from the MUX and decode that into the related Hex value which would be shown on the segment. The 2-to-4 decoder will get the same value as the select of the MUX and decode that value into 4-bits(Ex:01—-0010,11—-1000). This output would get fed into the segments connected to a not gate right before because the displays are active low.

FIGURE 9: SERIALIZER



EXPERIMENTAL SETUP

Our calculator was set up and tested on a Nexys A7-100T FPGA board, with inputs provided through the board's switches (A(0)-A(7) and B(0)-B(7)) to enter two 8-bit signed numbers and select the operation (add, subtract, multiply, or divide). The outputs were displayed on 7-segment displays. The design was implemented using VHDL code written in Vivado, with a combination of new code and components from previous labs (4 and 6).

Each component of the project was tested individually by whoever was assigned to it before combining them. This was done by having each group member create a testbench

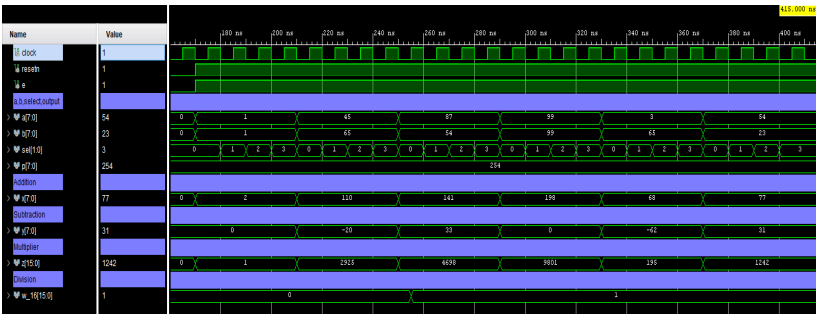
to simulate and verify the functionality of their operation (addition, subtraction, multiplication, or division). This step-by-step approach made it easier to find and fix errors in specific parts of the project. Once all the operations were confirmed to work as expected, we port mapped them together into a single design.`

Before finalizing the project, we created a test setup that included all components except the switches and 7-segment displays. In this setup, we manually entered test values to verify that the data passed through each component correctly. Once everything was working, the full design was loaded onto the board, along with a constraints file to map the FPGA pins to physical components like switches and displays.

In the final version, the user can input two 8-bit signed numbers using the switches, select an operation, and view the result immediately on the 7-segment displays in hexadecimal format. The expected results are accurate calculations for all operations, with special cases like overflow and division by zero handled correctly. After each calculation, the user can reset the calculator using the reset button on the board.

Results

Figure 10: Result Waveform



Circuit - Expected Example Values - Signed Decimal (Binary)								
	A	B	X (Addition)	Y (Subtraction)	Z (Multiplication)	W (Division)	sel	P (MUX Out)
Case 1	1 (00000001)	1 (00000001)	2 (00000010)	0 (00000000)	1 (0000000000000001)	1 (00000001)	1	1 (0000000000000001)
Case 2	45 (00101101)	65 (01000001)	110 (01101110)	-20 (10010100)	2925 (0000101101101101)	0 (00000000)	2	-20 (1000000000010100)
Case 3	87 (01010111)	54 (00110110)	141 (010001101)	-87 (01010111)	4698 (0001001001011010)	1 (00000001)	3	4698 (0001001001011010)
Case 4	99 (01100011)	99 (01100011)	198 (011000110)	-99 (101100011)	9801 (0010011001001001)	1 (00000001)	4	1 (0000000000000001)
Case 5	3 (00000011)	65 (01000001)	68 (001000100)	-3 (10000011)	195 (0000000011000011)	0 (00000000)	3	195 (0000000011000011)
Case 6	54 (00110110)	23 (00010111)	77 (001001101)	31 (00011111)	1242 (0000010011011010)	2 (00000010)	2	31 (0000000000011111)

Figure 11: Calculator Expected Value Table

Conclusions

Wrapping up this project has helped us have a much clearer understanding of digital logic design. It tied together everything we learned during the semester, from binary math and 2's complement to working with asynchronous and synchronous circuits, FSMs, and building a full digital system. Using an FPGA to implement the design gave a hands-on perspective of how these pieces fit together. Troubleshooting was also a key part of making the project work. Identifying and fixing issues within individual components made everything run smoother in the end. Writing testbenches and running simulations in Vivado helped check each piece for accuracy. When something didn't work as expected, signals were traced to find the problem, allowing the system to come together.

References

[1] Abd Alrahman Al Nounou, ECE-2700: Digital Logic Design Fall 2024, Electrical And Computer Engineering Department, Laboratory 3, Oakland University.

[2] Abd Alrahman Al Nounou, ECE-2700: Digital Logic Design Fall 2024, Electrical And Computer Engineering Department, Laboratory 6, Oakland University