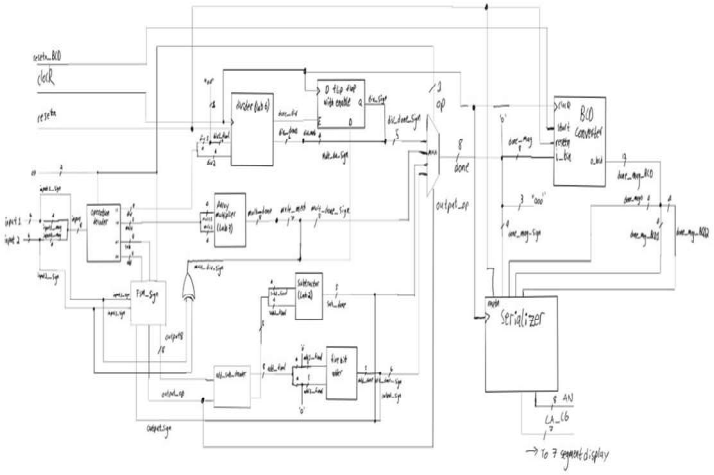
Signed Calculator

List of Authors (Jason Wend, Manuel Muresan, Fady Sto, Karam Matti)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI

e-mails: Manuelmuresan@oakland.edu , Fadysto@oakland.edu , Kmatti@oakland.edu, jgwend@oakland.edu

For the final project we have decided to create a signed calculator. Our calculator will be able to add, subtract, divide, and multiply. As a signed calculator it uses positive and negative numbers. The results are in BCD and our inputs are 6 bits in sign and magnitude binary form. Also, our results will be displayed on the 7-segment display that is on the FPGA board. Positive and negative signs will be displayed, and the answer will be displayed in decimal form.

Introduction



For us to make our project, we will use Vivado to make the signed calculator that will be able to do the four known operations (adding, subtracting, multiplying, and dividing). The answer will be displayed on the 7-segment display in decimal form followed by the positive/negative sign.

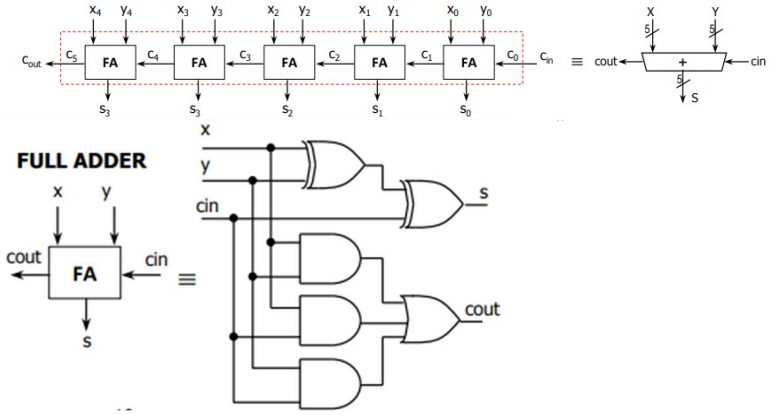
Our main goal in selecting a signed calculator is to deepen our grasp of manipulating binary data. This effort will increase our understanding of registers in addition to seven segment displays. To save our input data, we intend to use several registers, and to display our solution, we intend to employ various seven-segment displays. The three primary functions of any digital system are data receiving/saving, data manipulation, and data presentation. These three functions are covered by our signed calculator. Numerous modules used in this calculator, such as the multiplier and register, were covered in class. The calculator will, however, also make use of some logic circuits that were not discussed in class, like the BCD to binary converter and a unique 5-bit hexadecimal decoder. To build a reliable yet user-friendly calculator, our goal is to apply the fundamental logic circuits that were taught to us in class.

I. Methodology

Section 1: Magnitude

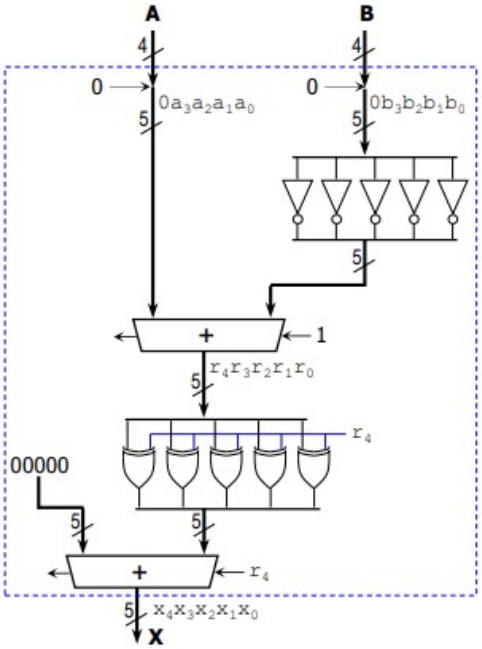
In designing the simple sign and magnitude binary calculator, we decided to approach the actual calculator circuit in two parts: computing sign (positive or negative) and computing magnitude. In computing magnitude, we used many of our labs as individual components. For instance, lab 2 computed the magnitude of input 1 minus input 2, which we used as the "subtractor" for our calculator. We also used one of the 5-bit adders from lab 2 as our "adder, the entirety of lab 3 (the array multiplier) as our "multiplier", and the entirety of lab 6 as our "divider". Circuit diagrams of each of these components are shown below.

Adder:



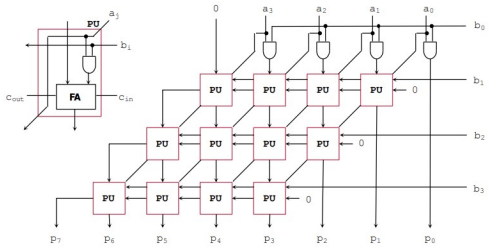
Here the is the five bit adder which is composed of five full adders. The circuitry for the full adders is also shown.

Subtractor:

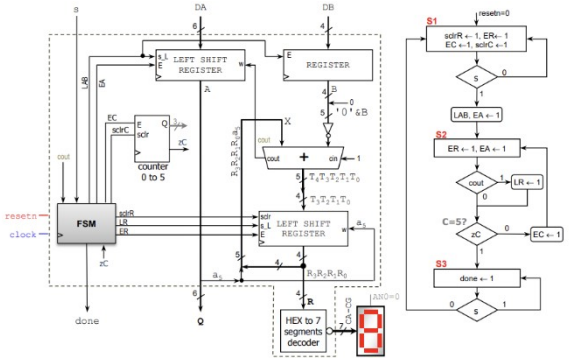


Note that the components marked with a "+" sign are five bit adders.

Multiplier:



Divider:



Note that for the divider, we did not utilize the "Hex to 7 segments decoder" since we were not displaying the remainder of the division. The only outputs we cared about were "done" to indicate that the division has been completed, and "Q", the quotient.

A subtlety in our circuit design lies within the divider component. Because it utilizes a clock and its own FSM, its processes are synchronous and require a clock and a reset button. Because the other operations (addition, subtraction, and multiplication) were entirely asynchronous, this led to complications. Adding a reset button to our inputs partially fixed our problem, but now the correct sign no longer displayed with this computed magnitude, since there was a delay in the division operation. In order to solve this, a D flip-flop with enable was implemented so that it stored the output sign and only displayed it along with the magnitude when the operation was complete.

Section 2: Sign

Computing the sign of the answer involved making entirely new components. For multiplication and division, it was fairly simple. The truth table for the input signs and output sign for them is as follows:

sign1	sign2	sign out
0	0	0
0	1	1
1	0	1
1	1	0

This relationship can be represented by an XOR gate which we included in our circuit. For addition and subtraction, determining the output sign was much more complicated. In a process we called "FSM_sign", we figured out that it is determined by the signs of the inputs, which magnitude was larger, and the inputted operation (addition or subtraction). The truth table is as follows:

mag1<mag2	operation	sign1	sign2	sign output	operation output
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	1	1
0	1	0	1	0	0
0	1	1	0	1	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	1	1
1	0	1	1	1	0
1	1	0	0	1	1
1	1	0	1	0	0
1	1	1	0	1	0
1	1	1	1	0	1

In determining the output sign, we also needed to figure out which operation, addition or subtraction, was really occurring. For instance, If the user inputs 5–(-7), the way we compute the answer is by saying "since magnitude 1 is less than magnitude 2 (mag1<mag2 = 1), the sign of input one is positive (sign1 = 0) and the sign of input 2 is negative

(input2 = 1), and the inputted operation is subtraction (operation = 1), then the true operation that occurs is addition of the magnitudes (operation output = 0) and the sign of the answer is positive (sign output = 0)". The calculator would compute 5 +7, which is equal to 12, and then the answer is positive. In implementing this circuit, we used a VHDL process and if-then statements to code the logic, however it would have been much more realistic and simpler to use logic gates and comparators to compute the outputs. The computed operation from this component is sent to a decoder which selects which operation process the inputs go to. After, the computed sign is attached to the end of the bits.

Section 3: signal selection and conversion

Another important aspect of our circuit that has been modified is our "mux". This process serves to select the proper operation to output like a 4-to-1 multiplexer, however it also uses the output operation from the "FSM_sign" as another select line to choose between addition and subtraction, since the true operation that was being performed changes. We also inserted zeros between the signs and magnitudes of the addition, subtraction, and division signals to make them all 8 bits at the output signal (the multiplication signal was already 8 bits).

BCD:

After the operation has been resolved by the corresponding functional block, the result obtained is unsigned, and we can add the sign bit in the MSB location, converting it to sign and magnitude. The sign is kept separate because our multiplication and division blocks were originally obtained from lab2 and lab6 respectively, and they were performing unsigned operations. We chose to implement a Binary Coded Decimal (BCD) converter which supplies the four separate 4-bit signals, each of which will represent one character of the 7-segment display array. This component was obtained from Dr. Llamocca's website [1]. It uses a 20-bit shift register, a counter, an adder, and an algorithm in order to convert the input bit stream into the decimal equivalent separated by order of magnitude. When the whole input has been shifted to the final position of the shift register, we are left with three 4-bit binary coded decimal representation for the hundreds, tens, and one's place. Some minor pre-processing is done to our input, by removing the sign bit and replacing it with a leading 0. This is done to maintain the 8-bit unsigned binary input to the BCD converter. This converter takes the 8-bit magnitude input and converts it into the decimal representation, with each digit being represented by four bits. Each of the 4-bit signals generated by the BCD converter can now be separated and passed along to the next step, which is the serializer.

Serializer:

The Serializer used was also obtained from Dr. Llamocca's website [1]. The purpose of this component is to display four separate characters on the 7-segment array. Since the FPGA board has all 7-seg input pins tied together, we need the serializer to enable the four 7-segment displays to work in sequence, briefly allowing current to flow for only one digit at a time. The serializer uses counters and the clock signal to make sure the information displayed corresponds to the correct digit. The Finite State Machine inside the serializer repeats the cycle once the final digit has been displayed, returning to the first digit. The serializer input passes through a decoder which contains all the possible characters which can be displayed on a 7-segment display. The left digit is only used to display the horizontal center segment, indicating a negative result. Even though originally configured to work with Hexadecimal inputs, it still works in our case, as unsigned binary and Hex characters are the same for 0-9. The only difference is the addition of a special case which is only used by the segment displaying the negative symbol. The information available to the 7-segment displays is the same for all the digits, but only the correct one will be allowed to display that information. The 7-bit signal output by the Hexto7seg decoder is synchronized with the index of the corresponding digit. This is achieved by the select line, which controls the multiplexer. The mux is what completes the circuit for each individual element of the 7-seg array. The 2-bit select signal has four possible cases: 00, 01, 10, and 11. Each case corresponds to activating one of the digits on the array.

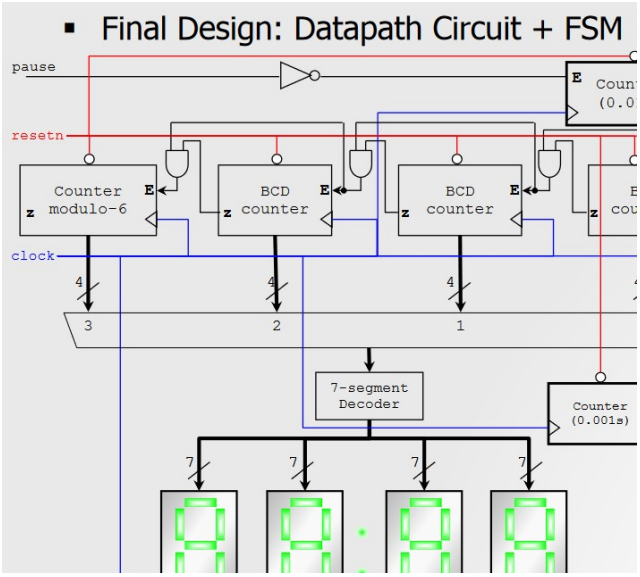


Figure 2- Example of 4-digit 7-segment display

The illustration in Figure 2 was obtained from the Tutorials/VHDLFPGA/Unit 7 notes available on the list of examples and VHDL code from Dr. Llamocca [3]. This

design was adapted to work in our situation, with the major difference being the source of the BCD numbers in our case comes from the operations being performed instead of counters.

```
34    if (s = "11") then
35        case CA_CG is
36            when "0001" =>  --negative number
37                leds_sig <= "0000001"; -- "-"
38            when others =>  --positive numbers
39                leds_sig <= "0000000"; --nothing
40        end case;
41    else
42        case CA_CG is
43            when "0000" =>  --0
44                leds_sig <= "1111110";
45            when "0001" =>  --1
46                leds_sig <= "0110000";
47            when "0010" =>  --2
48                leds_sig <= "1101101";
49            when "0011" =>  --3
50                leds_sig <= "1111001";
51            when "0100" =>  --4
52                leds_sig <= "0110011";
53            when "0101" =>  --5
54                leds_sig <= "1011011";
55            when "0110" =>  --6
56                leds_sig <= "1011111";
57            when "0111" =>  --7
58                leds_sig <= "1110000";
59            when "1000" =>  --8
60                leds_sig <= "1111111";
61            when "1001" =>  --9
62                leds_sig <= "1111011";
63            when others =>
64                leds_sig <= "1111111";
65        end case;
```

Figure 3- Hex-to-7seg display decoder

The decoder used to interpret the timing diagram outputs "LEDs" can be seen in Figure3. A few sample calculations are simulated in the experimental section below.

II. Experimental Setup

When all the project files were finished, we started testing our project using Vivado's simulation feature. We discovered quite a few things were not working as intended right away. There were a few minor issues that were resolved right away, including some typos in the top file's port mapping and the BCD decoder.

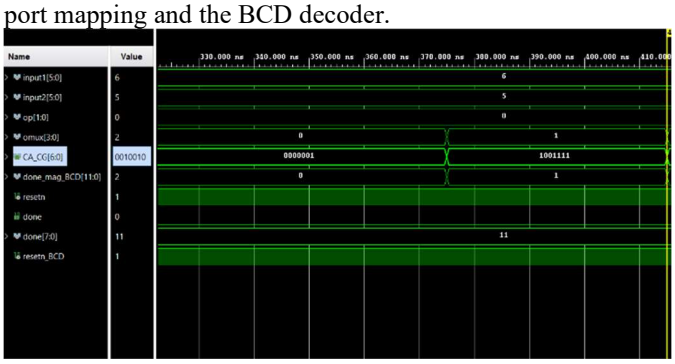


Figure 4: Addition

Example of operation 00, which represents addition. 6+5=11. The 7-seg display is active low, and therefore we see the bits of CA_CG as a flipped version of the sequence specified in the decoder.

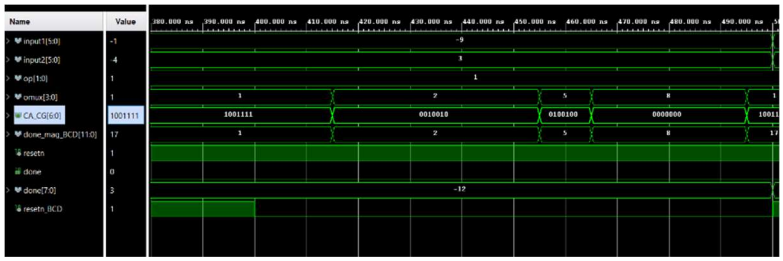


Figure 5: Subtraction

Subtraction example showing the result of –9 - (3) = -12.

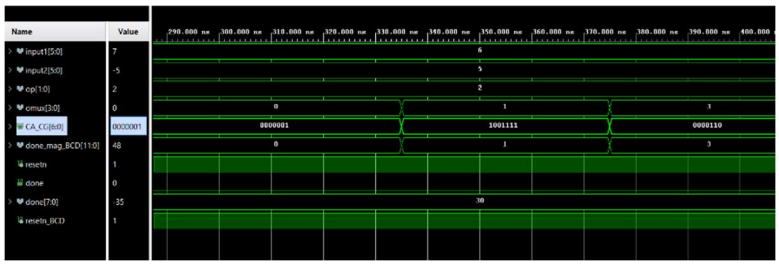


Figure 6: Multiplication

Multiplication example showing the result for input1 = 6, input 2= 5, and the result in sign and magnitude =30. The CA_CG display output is toggling between the 4 different 7-segment display characters, displaying a different digit on each one.

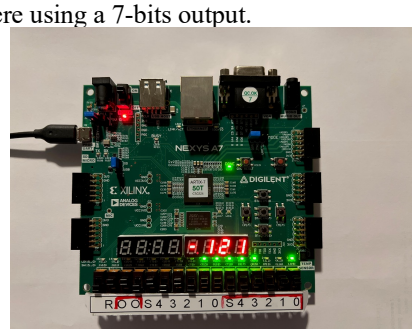
Figure 7: Division

The division operation requires two reset operations. We have the slide switch reset as well as the push button reset switch which was used for the previous operations.

III. Results

The results for our project were like expected a calculator does all the four operations and you can go up to 127 also displaying the sign if the number was positive or negative in addition displaying numbers in decimal form on the FPGA bored by using four 7-segment display. It was hard to make the FPGA bored display the answer in decimal form because as we all know the FPGA bored uses hex decimal numbers. We used a converter from a Binary to BCD and that

converter will allow us to display the answer in a decimal form. The result was explainable and make since if the highest and lowest numbers were (121, -121) after that the results will to be unexplainable and that's only because we were using a 7-bits output.



IV. Conclusion

Following the project's completion, our team was able to draw a few key findings. One thing we discovered early on was the value of creating a thorough yet understandable block diagram. Without a visual tool like the block diagram to determine which issue each group member should focus on, project coordination would have been exceedingly challenging. Similar to the block diagram, our team saw the value of using the error codes that Vivado provided after a failed implementation. We were able to identify precisely where and how the project was failing thanks to the error codes. Overall, our team was quite happy with how our project turned out. One of the issue our team faced was not be able to go higher than 121 but based on our design and calculations we should of be able to go higher than 121.The improvement that can be added to this project is to make the input an advanced keyboard connected to the FPGA board that keyboard should include numbers and all the operations that our calculator can do.

The materials used for this project were Lab 2 for the add operation, lab 3, for the subtract operation, Multiplier for the multiply operation, and lab 6 for the division operation. Finally, we also used a converter from Binary to BCD we got it from Dr. Llamocca's website page.

V. References

[1] D. Llamocca, VHDL coding for fpgas. [Online]. Available: http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html. [Accessed: 02-Dec-2022].

[2] Fall 2021 - ECE2700: Digital Logic Design. (n.d.). Retrieved December 2, 2022, from http://www.secs.oakland.edu/~llamocca/Fall2021_ece2700.html

[3] D. Llamocca, "Digital System Design - Oakland University," *Tutorials/VHDLFPGA/Unit7*. [Online]. Available: http://secs.oakland.edu/~llamocca/Tutorials/VHDLFPGA/Unit%207.pdf. [Accessed: 04-Dec-2022].