Signed Fixed-Point Calculator

FX format: [20 8]

Cory Bledsoe, Rachel Pilarowski, Melvin Pulianthuruthil
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: cbledsoe@oakland.edu, rpilarowski@oakland.edu, pulianthuruthil@oakland.edu

***Abstract*—The purpose of this design project was to implement a signed fixed-point calculator on a FPGA. This calculator is able to add, subtract, multiply and divide two sign and magnitude inputs which come from a keypad. The fixed-point format for this calculator is [20 8]. It can handle numbers in the range of [-999.99, +999.99]. The user can select which operation to do by using the on-board switches and buttons. After the calculation is completed, the result is shown on 7-segment displays.**

I. Introduction

Calculators are some of the most common tools in every occupation. They are fast and efficient, making them a very powerful tool in simple arithmetic applications. In this project, a signed fixed-point calculator was designed and created using a Nexys A7 FPGA Trainer board and VHDL programming language.

The purpose of this project was to gain a higher appreciation for the computing behind a calculator. This project will also serve to strengthen VHDL concepts, reinforce troubleshooting skills and explore the concepts of fixed-point arithmetic. Fixed-point arithmetic is useful because it allows for both positive and negative representation and includes fractional precision.

This calculator will work by taking a sign and magnitude fixed-point number with the format [20 8] from a keypad input and go through the basic arithmetic operations: addition, subtraction, multiplication and division. The user will use switches in order to decide which operation they wish to complete. After the computation is completed, the result will be shown on the on-board 7-segment displays.

II. Methodology

There are a few main components that this calculator will include. The user interface will be a keypad in order to enter the desired numbers and their sign. This calculator can only handle numbers in the range of [-999.99, 999.99]. Because this calculator is using a keypad, the input will need to be converted from BCD to binary. This is a basic calculator which can compute basic arithmetic. Therefore, 4 circuits are created to handle each of the operations: addition, subtraction, multiplication and division. A multiplexor is used to determine which of the 4 arithmetic will be computed. The user can use the switches on-board to control the output of the multiplexor. Lastly, the result will need to be converted from binary to BCD, and then display the result on the on-board 7-segment displays. A high-level block diagram is shown below in Figure 1. More detailed explanations of each component is described in the sections below.

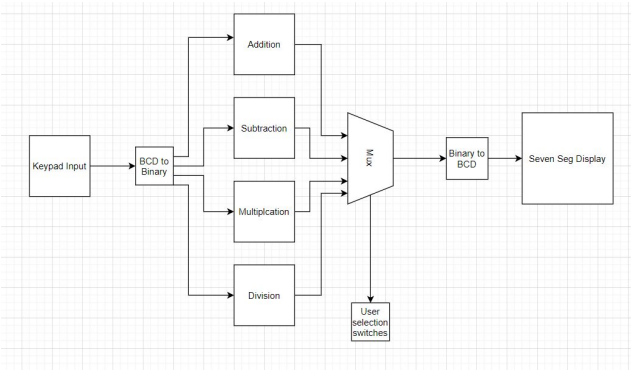

Figure 1: High-Level Block Diagram of the Calculator

A. Keypad and User Interface

The goal is to allow the user to input the digits he or she wants to calculate by using the Arduino 4x4 keypad. There are 8 pins for the keypad, where the first four pins are connected to the four rows, and the last four pins are connected to the four columns. These 8 pins are connected to the Nexys A7 JC PMOD port. The JC PMOD port has 12 pins. Pins 1-4 and 7-10 are used for logic signals, pins 5 & 11 are GND, and pins 6 & 12 are VCC which can provide 3.3V. For this project, only the logic signal pins are used since power is supplied to the board using the USB port. In Figure 2, the row 1-4 pins are connected to the PMOD pins 7 -10, and the column 1-4 pins are connected to the PMOD pins 1-4.

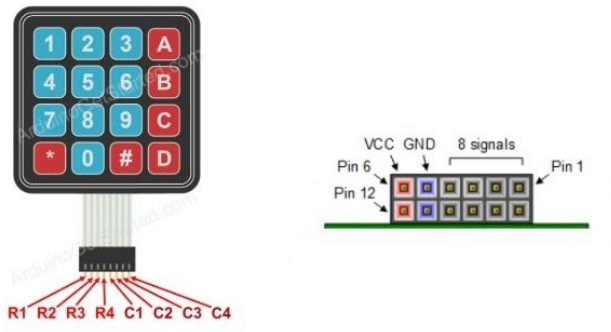

Figure 2: Keypad Connection

The block diagram, in Figure 3, shows how the program works. The decoder keeps the logic of one column pin low and the rest high, and then checks each row in the column for a logic low denoting a button press. If it does find a logic low, it determines the corresponding value and displays it to the 7-segment. If it doesn't find a logic low on any row, then the program keeps the next column low, and repeats this process until it matches a logic low on a column and a row. The Figure 4 shows the number displayed on the 7-segment when a button is pressed on the keypad [1].

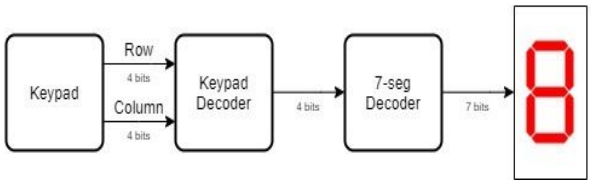

Figure 3: Keypad Block Diagram

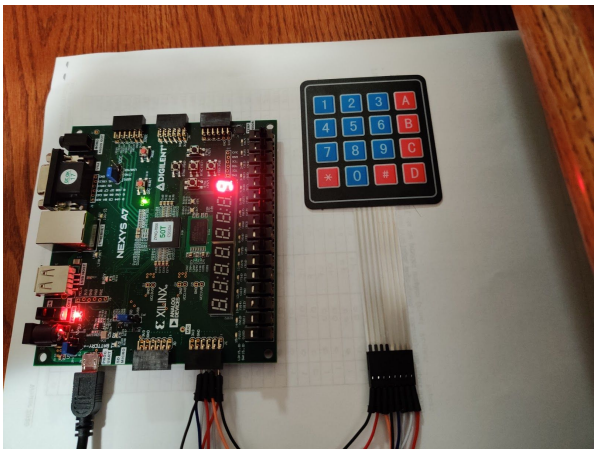

Figure 4: Keypad Demo

B. Addition

For this project, a simple full adder is used for both the addition and subtraction. However, the full adder expects integers in two's complement and the output of the BCD to binary is in sign and magnitude. Therefore, a conversion between two's complement and sign-and-magnitude is needed. Once the result is obtained, the result needs to be converted back to the sign and magnitude format. Figure 5, shown below, is a block diagram for the addition and subtraction circuits. It should be noted that since this is addition, the "Cin" for the full adder is '0'.

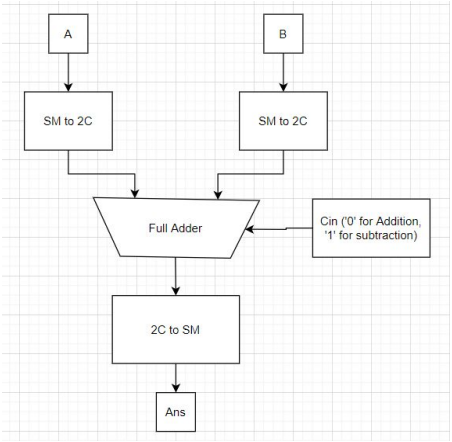

Figure 5: Block Diagram for Addition and Subtraction

The sign and magnitude to two's complement converter works by looking at the sign bit of the input. If it is a '0', or positive, it passes through the input without changing it. However, if it is a '1', or negative, it appends a '0' to the magnitude of the input and converts it into two's complement. The two's complement to sign and magnitude converter works in a similar way. If the number is positive, the input is not changed. However, if it is negative, the input is converted [2].

C. Subtraction

The method of subtraction for this calculator works very similarly to the addition. The same circuit which is shown above in Figure 5 is used for the subtraction. The two inputs are sign and magnitude and it needs to be converted to two's complement so that they can be fed to the full adder. However, Cin is set to '1' so that the inputs are subtracted instead of added together. The result of the full adder is then converted back to sign and magnitude in order to be displayed on the 7-segment displays [2].

D. Multiplication

Fixed point multiplication works the same way as the binary multiplication by using the two's complement system for the multiplications performed. The precision for the multiplication must be taken into account because of having to drop bits as the arithmetic operation for the product can be much longer than the bit lengths from the inputs. When multiplying two's complement numbers, there

must be a sign extension to the operands to meet the number of digits for the multiplication that has been performed.

For this circuit, the input is two fixed point numbers in sign-and-magnitude. The sign bit is sent to an XOR gate to determine the sign of the number. After this, zero's are appended to the MSB of the second input, increasing the number of bits from 19 to 38. After this has been completed, a typical Iterative Multiplier circuit is used. The block diagram and state machine for this circuit is shown below in Figures 6 and 7. These circuits come from Dr. Llamocca's website [2].

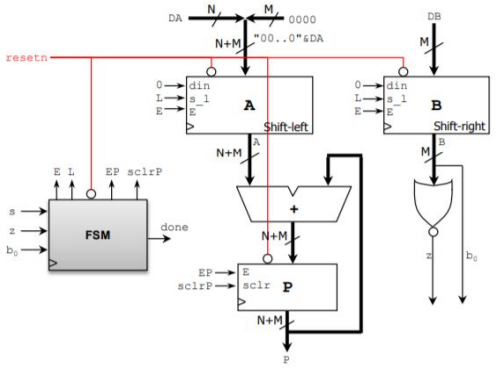

Figure 6: Block Diagram for Iterative Multiplier

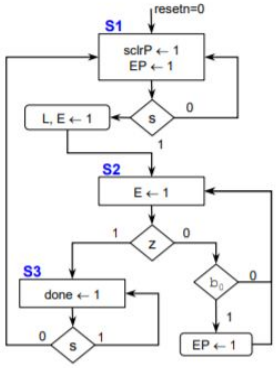

Figure 7: State Machine for Iterative Multiplier

This circuit works by looking at the LSB of the second input, B. If it is '1', then it is added to the cumulative sum. After each iteration, both inputs are shifted. After all bits have been shifted, the cumulative sum is the answer. As discussed before, the answer will have 38 bits whereas the calculator can only handle 19 bits. Therefore, the MSBs are cutoff as well as the LSBs of the result. After this, the result is sent to the binary to BCD converter and displayed on the 7-segment displays.

E. Division

Fixed point division works like a typical binary division with a few extra steps. This calculator's inputs also are in sign and magnitude, therefore, the inputs do not need to be converted to unsigned binary. Also, since the format of this fixed-point calculator is constant, alignment is already completed. However, in order to get some precision for the fractional part of the number, bits need to be added to the end of the dividend. For this calculator, 8 bits are added to the LSB of the dividend since the fixed-point format is [19 8], not including the sign. After these few steps have been completed, the typical iterative divider circuit can be used to complete the actual division. The iterative divider that was used for this project comes from Dr. Llamocca's website [2]. The block diagram and state machine of the iterative divider is shown below in Figures 8 and 9.

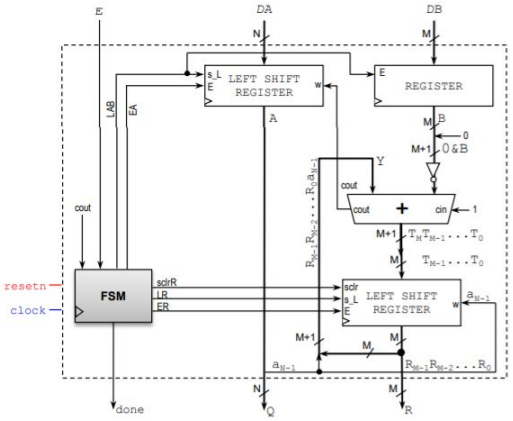

Figure 8: Iterative Divider Block Diagram

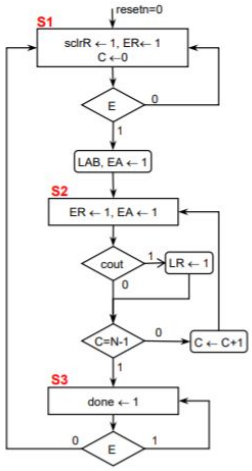

Figure 9: Iterative Divider FSM

This circuit is based off of the hand-division method. Bits of A are compared one by one to the divisor. If the result is greater or equal to B, then B is subtracted from it. On each iteration, one bit of Q is determined. After the division is completed, the last 8 bits of Q are truncated and sent to the Binary to BCD converter to be displayed on the 7-segment displays.

F.. BCD to Binary and Binary to BCD Converters

When a key is pressed, the keypad decoder will output a corresponding BCD number. In order to do arithmetics for the calculator, these BCD numbers need to be converted to a binary number. There are three main decoders for this converter: Sign Decoder, Integer Decoder and Fraction Decoder. Sign decoder will output a one bit binary number which is either a '1' if B was pressed or a '0' if A was pressed.The integer decoder has 3 components which are used to obtain a 11 bit binary number representing the 3 digit integer: 1^{th} Decoder, 10^{th} Decoder and 100^{th} Decoder. The 1^{th} will receive a 4 bit BCD value between 0 and 9, and it outputs an 11 bit value between 0 and 9. Similarly, the 10^{th} decoder will output an 11 bit value from 0 and 90, but the numbers go up by 10. Lastly, the 100^{th} decoder will output an 11 bit value from 0 and 900, but the numbers go up by 100. Then, these three binary numbers are added together to obtain the final integer value in the binary number. The fractional decoder is similar to an LUT, and it contains 10 decoders. The two fractional BCD numbers are concatenated together, and this 8 bit number is used as the input. Then, by looking at the upper four bit of the input, the program selects a decoder that will give the 8 bit binary number of the fractions. The first decoder holds values from 0.00 to 0.09, the last decoder holds values from 0.90 to 0.99. The remaining decoders in between hold the remaining binary values of the fractions. Once the three numbers are processed, they are concatenated together to form a 20 bit binary number which will be used for the arithmetic operations. Bit 19 of the output is the sign of the number, bits 18 to 8 are the integer, and bits 7 to 0 are the fraction. Figure 10 is the block diagram of the BCD to binary converter circuit.

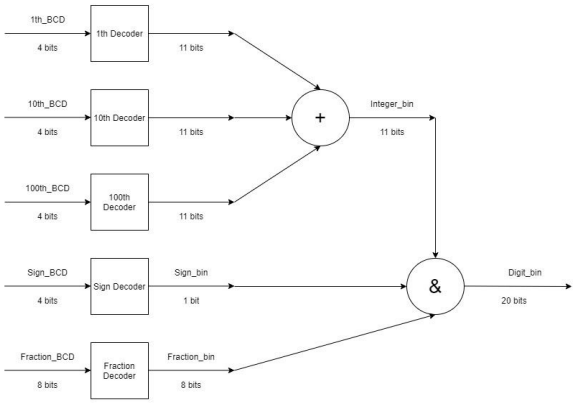

Figure 10: BCD to Binary Converter Block Diagram

Once the arithmetic operation outputs a binary result, it needs to be converted back to BCD number to display the result on the 7-segment. The BCD of the sign is determined by taking bit 19 of the result. A double method is used to extract the three BCD numbers for the integer. Since there are 11 bits, four BCD numbers are used, and they are all initialized with 0s. But, the integer number range of the calculator doesn't go over 999. Therefore, only three BCDs are required. The process starts by left shifting both the BCDs and the binary number, and the MSB of the binary number is shifted into the LSB of the first BCD. Then, the program checks whether a BCD value is greater than 4. If it is, the particular BCD is added with 3 and it gets updated. If not, the numbers are left shifted again. This process continues until all the binary number bits are shifted into the BCDs. Next, the fractional BCD numbers need to be extracted. The concatenated 8 bit fractional binary number is multiplied by 10. The upper 4 four bits will be 0's, bits from 11 to 8 is the BCD value of the first decimal place number. The bits from 7 to 0 are, then, multiplied by 10. Bits 11 to 8 of the new product is the BCD value of the second decimal place number. This process can be continued if there were more fractional numbers. Figure 11 is a block of the fractional binary to BCD converter. The converter will output the six BCD numbers which will be displayed on the 7-segment.

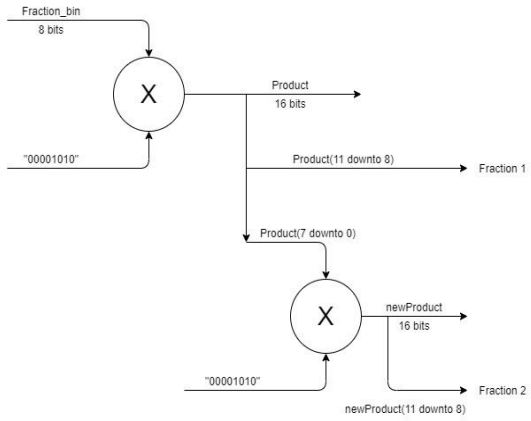

Figure 11: Fractional BCD to Binary converter

G. Seven Segment Displays

In order to display the final result, seven 7-segments are used. The program uses a similar technique that Dr. Llamocca has implemented in the program called Serializer: Four 7-segment displays. Only one 7-segment on-board can be enabled at a time. Therefore, a very small delay needs to be implemented to enable each 7-segment. The FSM in Figure 12 is programmed to enable each 7-segment on-board at a time. A pulse generator is used to output a '1' every 1 ms, which is used to transition the states in the FSM. Even if there is a 1 ms delay to enable each 7-segment, the human eyes would not perceive it. Since there are only seven states in the FSM, only the first seven 7-segments are enabled. The first 7-segment displays the sign of the result. If the number is negative, a minus is displayed, and if it is positive, nothing is displayed on the first 7-segment. The next 3 7-segments display the three digit integer. The fifth 7-segment displays the decimal point. Lastly, the sixth and the seventh 7-segment display the two fractional numbers.

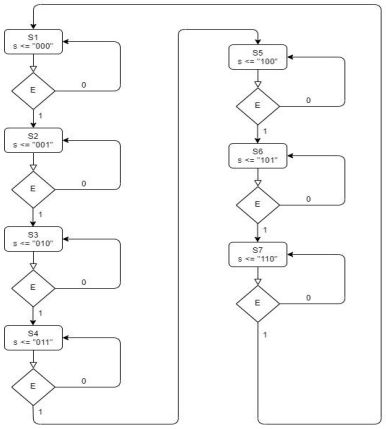

Figure 12: FSM of the 7-Segment Display

III. Experimental Setup

In order to verify that the calculator would work correctly, each component was simulated separately with its own test bench. This way, when combining all of the components together, it was guaranteed to function as expected. Since the keypad decoder code was provided by Digilent, a simulation was not created, and the code was directly tested on the hardware. There were no issues found during the hardware test.

Figure 13, below, shows the timing diagram resulting from simulating the addition circuit. The inputs are two sign and magnitude numbers with the FX format [20 8]. However, it was impossible to show the inputs with this format and be in sign and magnitude. Nevertheless, since fixed point addition is very similar to normal addition, if the result of normal binary is correct, it can be assumed that the fixed point addition would work correctly. One example that is shown in Figure 13 is -3200 + 10792 = 7592.

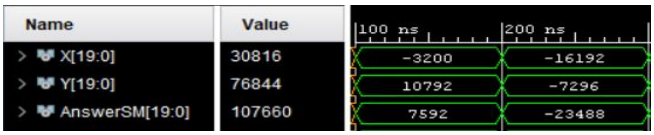

Figure 13: Addition Simulation

The subtraction simulation is very similar to that of the addition. The inputs are the same two sign and magnitude numbers in FX format [20 8]. The subtraction simulation is shown below in Figure 14. An example from the simulation shown below is -16192 - -7296 = -8896. Since the binary works correctly, it can be assumed that when changed to the correct fixed point format that it would be the correct result.

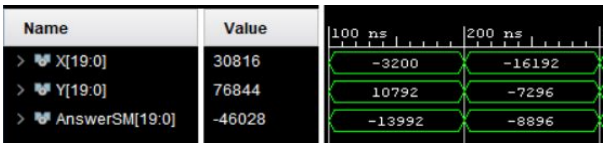

Figure 14: Subtraction Simulation

Figure 15 below shows the timing diagram for the multiplication circuit. This circuit uses a state machine for the control. So, unlike with the subtraction and addition, the answer is not shown instantaneously. It is an iterative multiplier, so the answer takes a few clock cycles. It is important to note that the sign is not simulated in the simulation below. The multiplication circuit took two numbers in [19 8] fixed point format. The MSB of the two inputs was sent to a separate XOR gate. Therefore, the results can be shown in the timing diagram with the proper radix. The example shown below in Figure 15 is 8.5 * 3.0 = 25.5, which is correct.

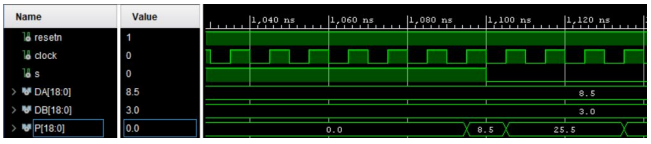

Figure 15: Multiplication Simulation

Figure 16 below shows the timing diagram resulting from the simulation of the division circuit. Similar to the multiplication circuit, the sign is not considered inside of the iterative divider. Therefore, the two inputs have a fixed point format of [19 8], which does not include the sign bit. Therefore, the timing diagram could include the radix in the proper position. The example shown below in the timing diagram is 1418/2 = 709, which is correct.

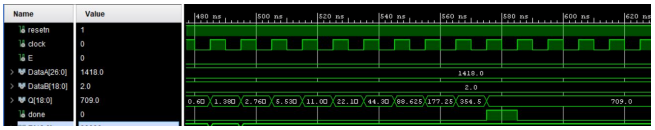

Figure 16: Division Simulation

The timing diagram of BCD to binary converter is shown in Figure 17 below. For the simulation, the converter received the following input: +123.50. The output of the converter is "00000111101110000000". This binary number is in the format [20 8]. The MSB is '0' meaning the number is positive, "00001111011" is equal to 123, and "10000000" is equal to 0.50. Therefore, it is concluded that the converter is working properly.

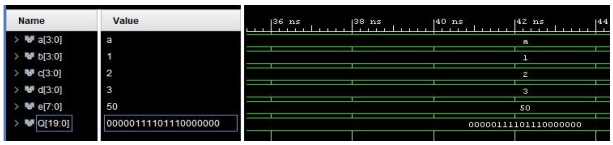

Figure 17: BCD to Binary Converter Simulation

The timing diagram of binary to BCD converter is shown in Figure 18 below. For the simulation, the converter received the following input: "00110110000110001111". This binary number is in the format [20 8]. The MSB is '0' meaning the number is positive, "01101100001" is equal to 865, and "10001111" is equal to 0.56 when rounded to two decimal places. These are also the values that were outputted by the converter as shown in Figure 18. The converter shows the fractional BCD numbers as 5 and 5, even though the number it should output is 5 and 6. This is because of the rounding errors that happened during the conversion process. If the calculator can handle more digits, then the result would be more accurate. Therefore, it is concluded that the converter is working properly.

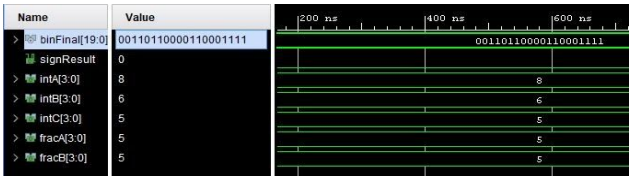

Figure 18: Binary to BCD Converter Simulation

Since all of the components were working separately on their own and have been simulated, it was decided that a simulation of the fully integrated project was not necessary.

IV. Results

For the final implementation, several switches are used in order to enter multiple digits from the keypad. For example, switch 1 needs to be high in order to enter the sign of the first input. Then, switch 2 needs to be high in order to enter the 100th digit for the first input and so on until all the digits have been entered. Since the format is fixed, the user doesn't have to enter a decimal point. These inputs are then converted from BCD to binary. The user then selects which operation to perform on the inputs using two switches on-board. "00" is entered for addition, "01" for subtraction, "10" for multiplication and "11" for division. The arithmetic is then performed and the result is shown on the 7-segment displays. It was found that due to the multiple conversions and only being able to show 2 values after the decimal point that there are some rounding errors. The error could be within +/-0.05 on any operation. However, this is acceptable. The final implementation of the project is shown below in Figure 19.

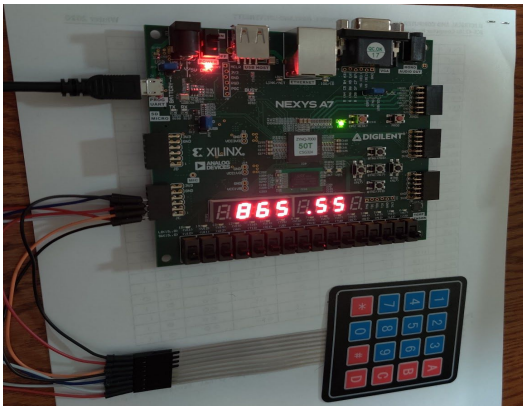

Figure 19: Final Implementation

V. Improvements and Conclusions

There are several improvements that can be made to this calculator in the future, such as getting the inputs of the keypad without using switches on-board. Another improvement is to display the two input numbers after the user inputs them. It is not possible to use the 7-segments to display the result the range was bigger. Also, if the range of fraction is larger, the method implemented to convert BCD to binary is inefficient and long. This final project was a great learning experience that was reinforced from the lectures and the labs from this semester. Calculators are one of the most common tools used globally, and this project allowed us to learn the amount of work that has been put into developing this simple, yet complex tool.

VI. References

[1] Yu, Michelle. "Pmod KYPD." *Digilent Documentation*, Digilent Inc. SUN 12 Apr. 2020, https://reference.digilentinc.com/doku.php?id=reference/pmod/pmodkypd/start.

[2] Llamocca, Daniel. "VHDL Coding for FPGAs." *VHDL Coding for FPGAs*. MON 20 Apr. 2020, http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html