Signed Fixed Point Calculator

Joshua Boczar, Alexander Ivanovic, Andrew Korte, and Benley Mathew
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: jjboczar@oakland.edu, aivanovic@oakland.edu, andrewkorte@oakland.edu, bpmathew@oakland.edu

***Abstract*:**

The purpose of our project is to create a fixed-point calculator that can add, subtract, multiply, and divide signed fixed point binary numbers. This calculator is capable of taking a sign and magnitude [12 4] fixed point binary input and performing various arithmetic operations on it. Switches and buttons will be used to enter data and seven segment displays will be used to display the output. Overall, the design was effective and functioned as expected.

I. Introduction

In today's times, calculators are an important part of life, they are very powerful tools and without them the world would not be as advanced as it is. To gain a better understanding and appreciation for how calculators work, our group decided to build one. This report will cover how we created the fixed-point calculator, how we overcame the obstacles presented during the calculator's creation, what results we found from experimenting with our calculator, and what our project is capable of accomplishing. The motivation behind creating this project was to learn how to create an effective calculator in VHDL. Signed fixed-point can be very useful as it allows for positive and negative numbers, as well as the precision that comes along with having fractional bits. This calculator will be capable of taking in a sign and magnitude [12 4] fixed point number, putting it through the arithmetic operations and providing the user with a signed decimal output. Topics that were learned in class that this project covers include calculating signed fixed-point binary numbers, being able to create, simulate, and port map multiple components into VHDL, and being able to troubleshoot and see multiple components in timing diagrams for errors. Another aspect of our project is the user interface. The user will be allowed to use switches to enter in the sign, the data that will go through the arithmetic operation, and the arithmetic operation itself. A button was used to load the data inputs into their respective registers so they can go through the calculation process. After the calculations are complete, the user will be able to view the output in signed decimal via the seven segment display.

II. Methodology

A. Calculator Assembly

There were many different components that needed to come together in order for this calculator to function properly. With each component came a new set of difficulties however, through careful planning and thorough testing everything was able to line up perfectly into a top file. The block diagram can be viewed in figure one which is located on the last page of the report. The block diagram was put at the end of the report due to the sheer size of it. Following the block diagram in figure 1, it can be viewed that the first component that starts off the entire data path is the input interface, after the input interface the data will travel to a two's complement component then to

addition and subtraction, or just straight to the multiplier and divider components. After addition and subtraction the data will go through another two's complement component and then will be stored in a register. After the multiplication and division component, the data will be stored in a register. After all this is complete, the data will go to a MUX where the select line is controlled by an operation selection FSM. After the data is selected, it will go to the seven segment component so the user will be able to see the correct answer. Now that you have an understanding of how the data will flow through the project it will be easier to understand how each of the components work as they are being explained in the later sections.

B. Input Interface

The very first part to this calculator is the input interface. This is the part of the calculator that takes the user input and prepares it so that it is ready to go through the arithmetic operations. The method chosen to accept user data was by having switches and buttons that the user can use to input data and operations. The proper way for the user to input data is to start by using switch 14 to select the sign for the A input (making this switch high will make A negative, and leaving it low will keep A positive), then use switches 11 down to 0 for the A input in binary. After this, the user must press the center button on the Nexys board to load that data into its respective register. The next step is to use switch 15 to select the sign for the B input, after this they can use switches 11 down to 0 to for the B input in binary. Once the proper switches have been selected, the user must then press the center button on the Nexys board to load the B data into its respective register. After the A and B inputs have been loaded into their registers, it is time to pick an operation using switches 2 down to 0. The user must use these switches to select what operation they want to use and then press the center button on the Nexys board to load it into its register. Table 1 shows which switches need to be high in order to achieve the desired operation.

Now that you are familiar with how the user is able to interact with the calculator to achieve the desired result, it is time to explain how the user interface was built and implemented on VHDL. The main component to the Input interface is the state machine. This state machine is used to detect when a button has been pushed and released, based off of this it will enable certain registers. Every time a button is pushed and released, the state machine moves along so that only the correct registers are enabled. Once every register has had data written to it the state machine will enable a done signal enabling a final set of registers that hold the data so it can go through the arithmetic operations. The reason the final row of registers is needed is because the data for A and B is not loaded instantly at the same time so the final register holds that data until it has all been collected so that the arithmetic operations will work as expected. Figure 2 shows a diagram in ASM form of the state machine created for the input interface.

Table 1: A table of the arithmetic operations and their corresponding switch inputs

Switch Input (2 down to 0)	**Operation**
001	**Addition**
010	**Subtraction**
011	**Multiplication**
100	**Division**

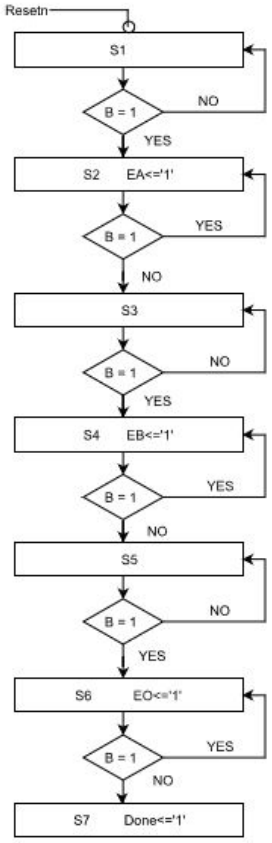


Figure 2: The state machine used in the Input Interface, where B = 1 checks to see if a button has been pressed, and EA,EB,EO are the enable lines for the A,B, and Operation registers respectively.

C. Operation Selection State Machine

The operation selection state machine is responsible for taking the data from the register that holds the operation input and converting it to a number that will act as the select line for the MUX that determines which output will be displayed on the seven segment display. Looking back, it might have been easier to create a decoder rather than a state machine, but the state machine worked just fine so it was not changed. Figure 3 shows how the state machine was created for the operation selector.

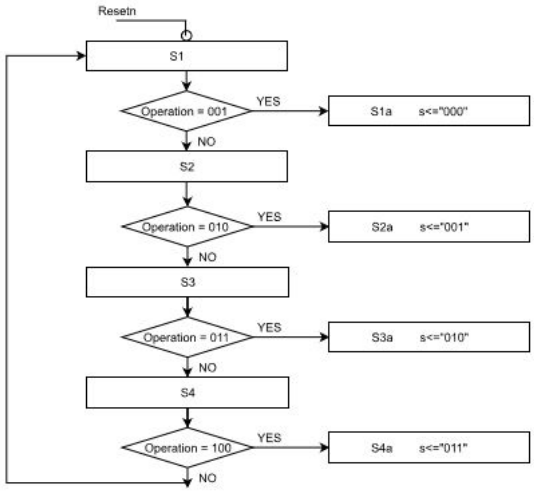


Figure 3: Operation Selection State Machine

D. Addition and Subtraction

When beginning the project, the addition, subtraction, multiplication, and division components were designed individually and simulated to make sure their operation was correct. For the addition and subtraction a zero was concatenated to the MSB to allow for signed operation. The sign of the number is determined with switch inputs, 1 for negative and 0 for positive. This means that the addition and subtraction inputs are 13 bits long, 9 bits for the integer and 4 bits for the fractional portion.

E. Multiplication

The inputs to the multiplication and division are 12 bit numbers, 8 integer and 4 fractional. The multiplication works by creating two temporary variables that are double the size of the input numbers. In this case the two variables are 24 bits long. One of the variables is filled with zeros and the other is the B input number with 12 zeros concatenated to the MSB. Next, a For Loop is used to multiply the A input and the temporary B variable. The A input is checked to see if a bit is equal to one. If so the temporary B variable is added to the all zeros variable. The temp B variable is then shifted to the left. If the bit in the A input is zero then the temp B variable is just shifted left and the process continues. The For Loop is executed N times with N equal to the size of the 2 inputs. For example the For Loop is executed 12 times for this project as the inputs are 12 bits long. The output for the multiplier is different from that of the other operations. To preserve the range of inputs, the outputs of the multiplier are five integer numbers and two fractional numbers.

F. Division

Division in fixed point is very similar to division in normal binary, however there are a few extra steps that must be accounted for. Normally, the input for a signed fixed point division would need to go through a two's complement process however, since the input to the calculator will be in sign and magnitude that will not be needed as the sign bits will simply go through an XOR gate and if the output is a 1, a negative sign will appear on the seven segment display. The same method will be used for multiplication. The next step to doing fixed point division is adding four zeros to the LSB. These extra four zeros will provide us with an extra four bits of precision. After this actual dividing process works the same as it does for normal binary, meaning that you need to keep subtracting the numerator from the remainder until the remainder gets too small, after this the decimal point was placed four bits to the left of the LSB since the format we chose was [12 4] and the output was sent to a Binary to BCD convertor so that it could be displayed on the seven segment display. The method we decided to used to implement the divider into VHDL was the iterative divider. The iterative divider utilizes shift registers and a state machine, it essentially shifts the input and subtracts the output from it. The iterative divider used was largely based off of the one that can be found on Dr. Llamoca's website. A picture of the iterative divider can be found in figure 4 and the state machine for it can be found in figure 5.

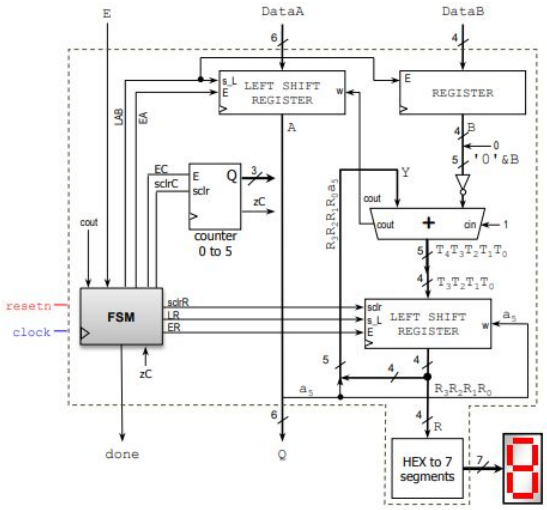


Figure 4: Iterative Divider Schematic

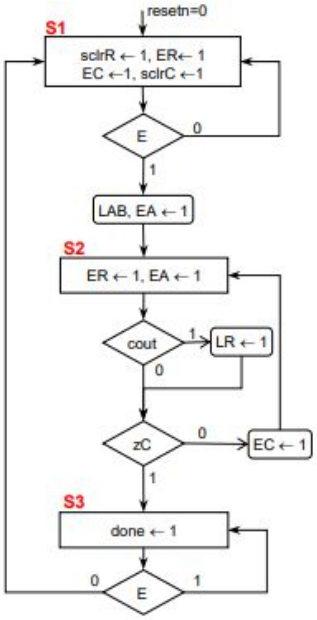


Figure 5: Iterative Divider State Machine

G. Basic Arithmetic Information

Next was the design of the Binary to BCD converter. A generic Binary to BCD converter was used to convert the outputs of the Adder, Subtractor, and Divider. The multiplier has a special Binary to BCD converter built inside the component as the number of numbers after the decimal is different than that of the other three operations. After that was created as a group, the calculation components were connected and simulated along with the converter to find if the main components of the signed fixed-point calculator were operating correctly. The range of inputs for the operations is -128 to 127.

H. Seven-segment display interface

After the data goes through the arithmetic operations, the output will be displayed on seven segment displays. The outputs to the operations are put through a Binary to BCD converter. This makes it easier to display on the seven segs. If the number had been left in binary a large decoder would have been need to show the correct values on the seven segment. The display works by port mapping decoders to each of the BCD numbers and then taking the decoder output to a display controller. The display controller then turns the correct display on and displays the correct number. A decimal point is also included in the display. This was achieved by adding an extra bit to the MSB of the normal seven bits that are sent to the displays. This extra bit controls whether or not the decimal point turns on for that display. For addition, subtraction, and division the decimal point for the 5th display is turned on to show 4 numbers after the decimal and three numbers before the decimal. For multiplication the decimal point for the 3rd display is turned on to show two numbers after the decimal and five before the decimal.

I. Bin to BCD and BCD to Bin

In a normal Bin to BCD converter the binary number is shifted to the left and then groups of four bits are checked to see if they are greater than four. If so that you add three to the bits. This method is called double-dabble. For the fractional numbers you multiply the binary number by 10 and grab the top 4 bits. This is repeated till the desired amount of fraction numbers. These numbers will all be in BCD form. A BCD to Binary convert was made to convert the Keypad inputs. The converter was taken out after it was determined that the Keypad would not be used and that the inputs would be in binary. The way the converter worked was, the integer values were multiplied by 1, 10, or 100 depending on what spot they represented. Then they were added together. For the fractional numbers it was more complicated. First, the numbers were treated as integer numbers and converted to binary using the previously stated way. Next, the number was multiplied by 256. Lastly, that

number was divided by 10,000. An example would be, 0.5678 = 5678 => 5*1000 + 6*100 + 7*10 + 8*1 = 5678(in bin) => (5678*256)/10,000. The final result would be the binary equivalent to the decimal fraction.

III. Experimental Setup

The software that we used was the computer design software Vivado. It was used for all of the simulation and code created and tested. The hardware tools we used and incorporated into the signed fixed point calculator included the Nexys DDR Board, and the 7 segment display operating on the board. Another feature of Vivado that was used included creating the actual digital block diagrams in VHDL that were then printed out and uploaded for visual use. The expected results of the simulation and actual components in real time is to see numbers that are being inputted to be outputted as a signed BCD number. Components that had errors within their simulation were explored and corrected. Testing and simulating the entire Vivado created calculator progressed well. Soon, after the testbench was created and the simulation was checked for any errors, the actual calculator will was synthesized and generated through the Nexys DDR board for further testing. Timing diagrams or the simulated results will be provided and shown during the final presentation. After all of the Vivado simulations were run, the project was loaded onto the actual board and tested. Figure 6 shows a timing simulation showing that all of the operations function as expected. Note that for the subtraction operation that the B input is negative which resulted in a double negative which caused the subtractor to act as an adder. This showed that our subtractor can work under any condition it is given. The reason the negative sign does not show up in front of the B input is because the negative sign was taken care of through switch 14. Figure 7 shows that the project is functioning as expected on the Nexys board. The test ran on figure 7 was -110.125 divided by 20.0625 which equals -5.4891. Because there are only 4 bits of precision, the fractional numbers are a little bit off, but that is to be expected.

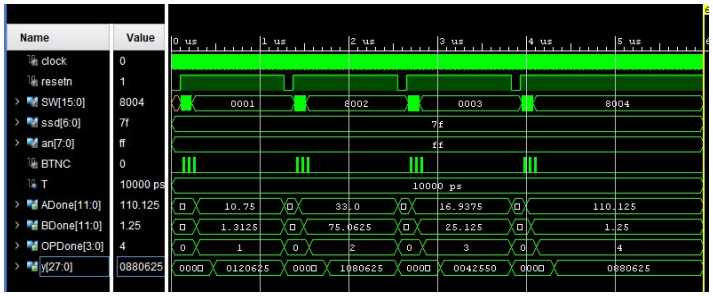
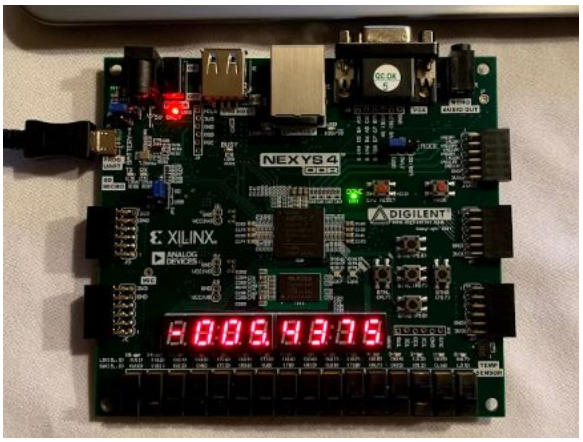


Figure 6: A timing diagram for all operations



.**Figure 7:** A picture of the project working flawlessly on the Nexys 4 DDR board

IV. Results

After completing the fixed point calculator, we were able to get values that were converted from signed fixed point into binary coded decimal. All the operations were completed using the Nexys 4 DDR board, the switches provided, and the seven segment display also provided. We tried using the keypad interface to input our values but we were unsuccessful in

integrating the keypad into our actual circuit and program. When the keypad was integrated, the values that were outputted were either inconsistent or wrong entirely. Either the keypad itself was malfunctioning or it was the actual program not being able to link with the Nexys 4 DDR board and the code within VHDL.

The result of our keypad not being able to operate was to replace the keypad with the use of registers and switches to complete the arithmetic operations and overall calculations. A great learning experience overall was not only being able to complete signed fixed point arithmetic, but was also being able to complete the arithmetic in VHDL while also accounting for multiple different cases that ranged from completing the actual arithmetic to making sure other components that were also being created operated properly together inside the program. Making sure the display was operating correctly was a task that needed to be addressed. As simple as it was, it was extremely important to make sure the seven segment display showed the correct outputted values. Without the values being displayed, how would we have known if the actual calculator was working properly other then through the use of the test bench and simulation.

V. Improvements That Could Be Made

The biggest improvement that could be made is the addition of a keypad interface. The design and implementation of the keypad did not end as expected. Interfacing with the keypad was much more difficult than what was originally assumed. The problem was with debouncing. Because of how the keypad worked timing for the debouncer circuit was very hard to calculate. The keypad is active low along with the fact that, not every bit for the keypad is changes when a new button was pushed. A finite state machine and multiplexor were created in order to connect the keypad and 7-segment display together on the Nexys DDR Board. The keypad uses a row and column system with pull down resistors to achieve an active low interface. The keypad was going to input numbers in BCD and therefore these inputs would have needed to be put through a BCD to Bin converter. A lot of the work was completed for the keypad however it never worked so it had to be excluded from the final version of this project. It appeared that bouncing keys caused the problem however the addition of a debouncing component caused the problem to worsen. The keypad interface component would have consisted of a decoder that would read keypad data, a state machine to detect button presses, and registers to store the data. The Keypad was not included in the final design of the project but a separate demo project was built to show how the Keypad works. The BCD to Binary converter was also not included in the project as the inputs are not input using binary values through switches. Another small improvement that could be made would be to have functionality that would allow the user to see the inputs as they are inputting them. As of now, the only thing appearing on the seven segment display is the final Answer. Overall, the group is very pleased with the final product and we are very proud of our work.

Conclusions

Overall, we were able to create a fixed point calculator using VHDL and the Nexys 4 DDR board. This calculator was able to convert fixed point into BCD using arithmetic operations that worked through components created in VHDL. Port Mapping each component to one another was not an issue, but making sure the number of bits was correct for each

component was a slight task we had to overcome. The biggest issue we ran into was not being able to use the keypad with our calculator. Halfway through the project, we found that the keypad was not operating correctly and wasnt outputting the correct values onto the seven segment display even though the simulation was outputting the correct values. In the end, we used switches on the board for activating are arithmetic operations along with using registers to complete the calculations overall functions without the keypad interface. Issues that remain to be solved include making the keypad interface operate with the overall program and device. Another issue that could have made our fixed point calculator better is to be able to see the inputted values being displayed on the board as we were typing them in. It is a small addition, but an overall improvement that could be accomplished if more time was permitted. Despite these challenges, the calculator functioned as expected.

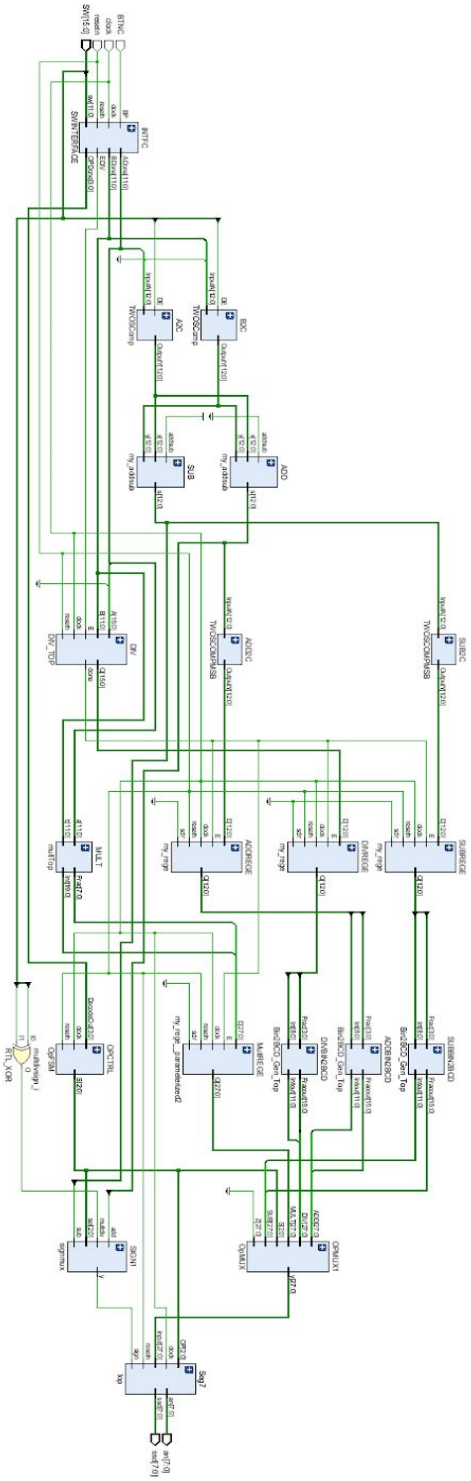


Figure 1: A block diagram of the overall top file for the project. It has been rotated sideways so that it could fit on the page.

References

[1]http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html
[2]https://reference.digilentinc.com/reference/pmod/pmodkypd/reference-manual?_ga=2.3320496.1623223115.1555982345-604595610.1551111620
[3]https://moodle.oakland.edu/course/view.php?id=221052