Floating Point Calculater

List of Authors (Najib Dubaisi, Yupei Liang, Karam Naama, Taylor Bodenmiller)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: Najibdubaisi@oakland.edu, yupeiliang@oakland.edu, naama@oakland.edu,
tmbodenmiller@oakland.edu

***Abstract*— This project is to show how to code a calculator using the VHDL language. With using a keyboard connected to the Nexy's 4 board as an input and an LED screen to show the answer. This project will result in a four function calculator (Add, Subtract, Multiply and division).**

I. INTRODUCTION

The IEEE-754 standard floating-point is a technical standard for floating-point computation developed in 1985. The most common format used is the 32bits floating-point. 32 bits floating point contains a sign bit, 8 bits biased exponent and 23 bits of significand.

The purpose about this project is to create a floating-point calculator, which is the development of customizable floating point units: Adders, subtractors, multipliers and dividers. Since using the options on the Nexys board as a method of input to the calculator, a keyboard will be used as the signed numerical input.

This report is intended to detail the design and implementation of our floating-point calculator. The decision of a project idea, the planning of project, and putting the plan into action will all be covered in this report.

II. METHODOLOGY

a. Toplevel

This project consists of many parts that are connected together to make a calculator that will add, subtract, multiples and divide. This module allows for the selection to use the other modules by using the switches on the board.

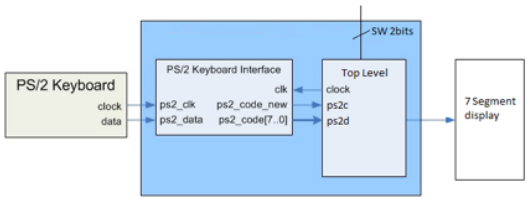

Figure 1: Overall design

b. Adder/Subtractor

First of all, starting with building an adder and a subtractor with using the arithmetic algebra of adding/subtracting numbers from Unit 2 from the class notes. Following the rules of adding/subtracting from the notes the coding was made to be implemented in the single precision case inputting 23 bits of significant and 8 bits of biased exponent [1]. Below is visual representation of the module built. shown in figure 1.

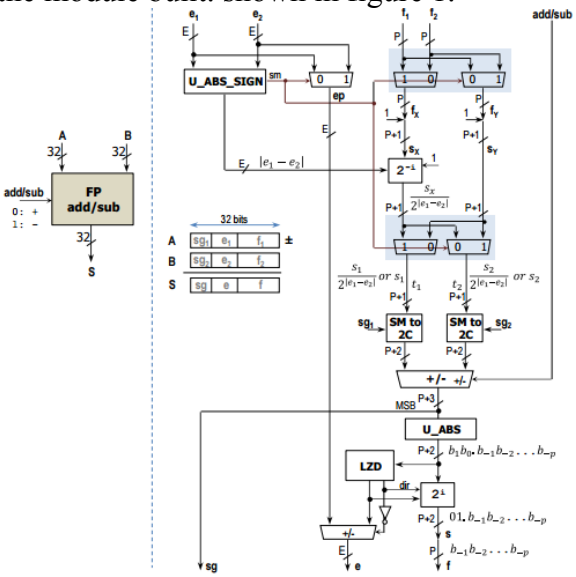

Figure 2: addition and subtraction circuit[3]

c. Multiplier/Divider

The second part of the project was to conduct a multiplier and a divider. With the same procedure of following the arithmetic algebra of multiplying and dividing numbers. With using similar parts for connecting the adder/subtractor they were used to build the multiplication and division circuits. You can find the block diagrams for both multiplication and division circuit in figures 2 and 3 respectively.

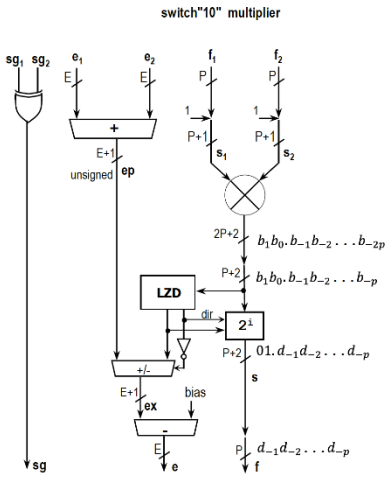

Figure 3: Multiplier circuit[2]

Multiplier: unsigned multiplier that can process multiplication functions by control flow data through the multiplexer, NOR gate, barrel shaft, and Leading Zero Detector. As the both Exponent contains bias which is 127 in decimal are added during the first add process. The bias is needed to be subtracted at the end of the calculate.

Divider: This module module takes in and processes two numbers. The data is first broken up into its proper sign, exponent, and significant bits. By operating in 2C arithmetic the unbiased exponents are subtracted. An unsigned iterative divider is used to divide the significant bits with respect to four fractional bits. The team would like to implement eight fractional bits of precision for future implementation. Before the actual division, the dividend is extended by four bits. The iterative divider then grabs individual bits of the dividend and compares them to the divisor. If this result is greater than the divisor, the results is subtracted from the divisor and a "1" is added to the quotient. A Barrelshift with shift instruction for a lead zero detector is is then used to normalize the quotient data. The lead Zero Detector is then used to normalize the exponent data and the bias is then added back to finally adjust the exponent [2]. Below is a visual representation of this.

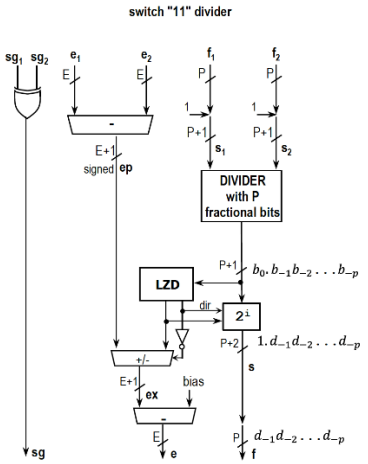

Figure 4: Divider circuit[2]

d. Keyboard as input

Finally, downloading the code on the board, but because of the lack of number of switches, a computer keyboard was used to make it easier to input numbers.

e. Counter and Multiplexer

In order to display the floating-point number and keep updating, a counter is used in the top level. To reach this goal, the done signal from the keyboard is the key to make this function come true. The counter would increment each time after the user release the key as the keyboard output "done" during this process. The counter outputs "00" to mux, to display the first input. When the user inputting the second number, the counter counts 8 and outputs "01" to the mux to display the second number. The third stage is that the counter counts 15 which means the project is able to obtain all 64-bit floating point form A and B and ready for outputs. In this stage, the mux treat the stage as others and outputs the result. And the 7

segment display automatically moves into the output state.

Stage	Counter	Counter output	Display
Input A	0-7	00	A
Input B	8-14	01	B
Output S	15	Others	S

Figure 5: Counter Table

f. Pulse Generator

In this project, we incorporated a pulse generator into our keyboard control. The job of our pulse generator is simply to divide the clock from 100 Mhz, which is standard for this type of FPGA board, into one second intervals.

h. ASCII to Bianry Look Up Table

As the keyboard outputs 9 bits during each cycle and outputs ASCII code and only 0 to 9 and A to F are needed in this process. Thus, a look up table is needed to convert the ASCII code to the actually binary value.

```
with din select
    hexo <= "0000" when x"45", --0
            "0001" when x"16", --1
           "0010" when x"1E",  --2
              "0011" when x"26", --3
              "0100" when x"25", --4
              "0101" when x"2E",   --5
              "0110" when x"36",  --6
              "0111" when x"3D",  --7
              "1000" when x"3E",  --8
              "1001" when x"46",  --9
              "1010" when x"1C", -- A
              "1011" when x"32", -- b
              "1100" when x"21", -- C
              "1101" when x"23", -- d
              "1110" when x"24", -- E
              "1111" when x"2B", -- F
              "----" when others;
```

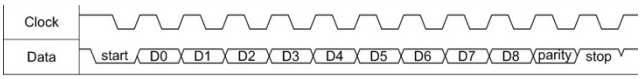

Figure 6: Data flow and LUT

i. Top Level Design

A top- level design is put together to combine all of the components into one file in the Xilinx software. In this part of the project, our team had to disable the keyboard input and individually test each component to make sure they were operating as designed.

III. Experimental Setup

The team was challenged with taking the actual input from the key board and simulation results from that. Instead, individual testbench were made to test each component. They proved to worked efficiently and were then implemented to the top level design and tested on the board for accuracy. A test bench was also made to test the operation of the keyboard PS/2 control block. After verification of this module working properly it was then implemented to the board and the remaining bugs were worked out. After each individual component was verified working properly the team combined the modules into the toplevel design previously mentioned.

a. Hardware

NEXYS 4 DDR Artix- 7 FPGA board: VHDL code from Xilinx software was uploaded to this board. Required.

NEXYS 4 DDR Artix- 7 FPGA board included the USB to ps-2 converter. A normal USB keyboard is required.

b. Software

As required for ECE378, our team used the Xilinx Vivado software program. In this program, we coded using a language called VHDL. This is the language the class was taught throughout the duration of the course, and is the only feasible language to use at this point to complete this project. The blueprint coding for our components, found on Dr. Llamocca's website, were written in VHDL.

IV. Results

This floating point calculated exactly as designed. The keyboard entry proved to work efficient as input to the arithmetic components and the resulting values matched as expected. Below is simulation wave forms for each of the arithmetic components showing their functionality. The inputs used are denoted as "a" and "b", the output is denoted as "b".

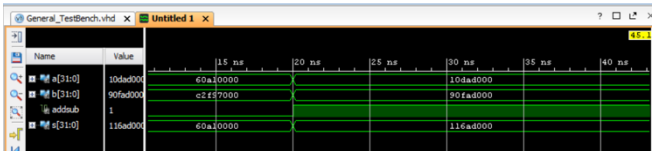

Figure 7: Add(addsub=0)/subtract(addsub=1) result

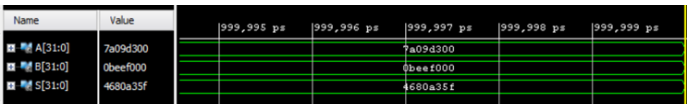

Figure 8: Multiplier Result

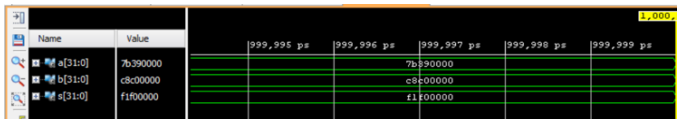

Figure 9: Divider Result

There was also success in LED functionality with regard to Key board input. The program successful input the proper hex values from user input from the keyboard and displayed the results within the LED's found on the FPGA. This also proved successful after arithmetic computation as well.

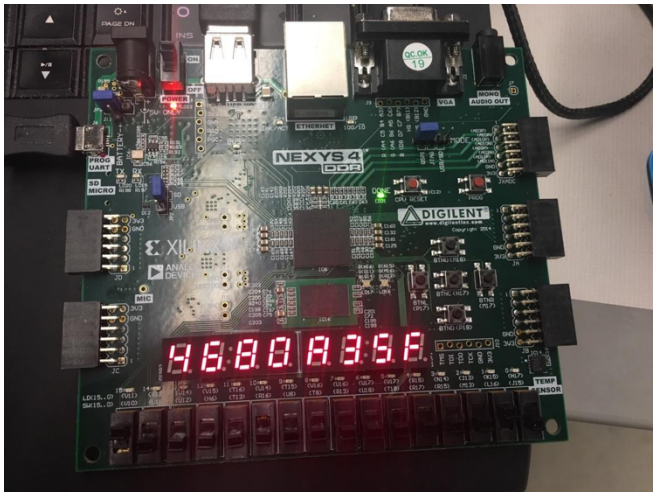

Figure 10: LED's properly display input values

V. Conclusion

This project successfully demonstrated the working of floating point number arithmetic through a functional VHDL software design implemented on a NEXYS FPGA while using a PS/2 keyboard as the input data source. The team built upon their knowledge of floating point arithmetic. While designing and building this project the team learned how to take and test the functionality of individual components and then the proper design of a control system to bring it all together. After successful completion overall understanding of 2C arithmetic, the needs for bit alignment shifts for accurate representation, and general code/hardware debugging skills were improved. There are improvements to the actual project the team would like to have implemented. This includes adding more fractional bits of precision to the divider component. Converting the output hex value to the appropriate floating point representation and displaying it on a LCD or external monitor.

VI. References

[1] Daniel Llamocca, Notes-Unit2, "Computer Arithmetic", Winter 2017.
[2] Daniel Llamocca, Notes-Unit6, "Special-Purpose Arithmetic Circuits and Techniques", Winter 2017
[3] Daniel Llamocca, "Laboratory 3", Winter 2017