Basys 3 ALU/Calculator Implementation

Patrick Scott
University of New Hampshire
Electrical Computer Engineering Department
ECE 583

***Abstract*—This Document discusses the purpose and function of the implemented ALU design on the Basys 3 Artix-7 FPGA. The paper will go in to more depth all of it's functions and talk a bit more on the implementation process that went into the design. Along with this it will go into specifics such as motivation for the project along with challenges faced and methods used when developing the final product. Finally the results and conclusion that will bring the entire project together and bring the paper to a close.**

I. Introduction

An ALU also known as a Arithmetic Logic Unit which is contained within the CPU or Central Processing Unit of a computer. It is a combination digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. It's basic operation in a CPU is to have a control input that determines what operation to perform on the given inputs. The output produced from the ALU goes directly to the output which can then be used to determine anything from a branch offset to a location in memory along with setting certain flags such as the zero, negative, carry and overflow flags.

This implementation is somewhat of a interpreted lesser version of such device because it can only operate on 6 bit inputs and the output can only reach up to 9999 on the seven segment display where as a standard ALU in a modern CPU can operate with 64 bit input/output depending on the operation. This limitation is not a big deal seeing as it still realizes the basic function it needs to mock a fully functional ALU. The device can also handle signed 2's compliment input and even basic floating point arithmetic. The floating point implementation differs from the industry standard (IEEE 754), this implementation will be further explained in the Methods and Challenges section.

Also note,the reason this paper is titled with ALU/Calculator Implementation is due to the fact of some of the odd functions realized by the design. For example it contains a factorial function which is really a recursive ALU function, but mostly due to the idea of floating point arithmetic. In an actual ALU the operation between two floating point numbers would be like any other number but the post ALU interpretation would be different to represent a decimal number, where as this system does that post operation conversion with a custom decimal interpretation system. After considering these differences it is at the end of the day a simple calculator that handles everything from input/output to interpreting bit strings with different elements such as decimal point precision or setting a flag to signify the output is to be read as a negative.

II. Motivation

The ALU is the worker of the modern computer and without it there would be no way to preform arithmetic operations to calculate offset's for jumps or even just a simple addition/subtraction that a user inputs. This idea was very interesting to us and the interface with a multi 7-segment display output would provide a nice visual component to act basically as a basic calculator for the user. The implementation of the custom floating point manipulation system was also an interesting system to see through as apposed to the floating point representation standard as defined by IEEE. It is interesting to see how custom implementations stack up against industry standard and where they fall behind in efficiency.

III. Challenges

There were some minor challenges when designing the ALU, but nothing unmanageable. Below are some of the issues encountered while in the design process.

- the first minor challenge was syncing the 7-segment display with the entire design including the output flags and error detection that arose with certain operations.
- The most annoying part of the entire project was the fact that it took about five minutes in between the time of making a change in the code and re-synthesising it onto the board. It made it very frustrating to have to go through that five minute cycle even after the smallest of bug fixes only to see that the fix didn't actually work or worse it further broke the design.
- The final and most time consuming challenge was within the floating point mode of the design. Originally the idea was to use the real data type which is the equivalent of the floating point data type from C. The problems occurred when trying to assign these real numbers to non real data types which caused a synthesis error because this cannot be synthesized by the hardware. In order to get around this barrier a custom floating point interpretation had to be implemented.

IV. Methods

For the most part this project was self explanatory and not very complicated. For example when taking in 2 different integer values and multiplying them it was a simple multiplication

symbol between then follows by a set of code that extracts each decimal place and displays it on the board. But for the fractional portion of the design it took a bit of thinking. The original idea was to implement it using the real data type, but seeing that did not synthesise there needed to be further research. The final design was to perform the operation just as it was a normal decimal input but then interpret the output in a different manor.

In a normal addition/subtraction situation after the operation the most right 3 bits are extracted and given one of eight possible decimal values which is assigned to the ”below radix value” anything else above those 3 bits is assigned to the above radix value. The above radix value is then multiplied by 100 to move it above the decimal placement on the 7-segment and then added to the below radix value to form a full value. For multiplication the output is shifted to the right 3 bits to align the number for the above described process. Division was another story however, the idea of a number going into another number a fraction amount of times deemed to be much harder than anticipated so the final design only shows the whole number portion of the division output.

V. Results

The final design can preform the following operations, addition, subtraction, multiplication, division, modulus, shift right, shift left, factorial and an option for fractional input. For both modes they operate as 2’s complimented input which will allow for negative and positive inputs. As for outputs the board will display up to a 4 decimal digit output with three flag LED’s that depict if an output is negative, overflows or has an error in the operation (divide by zero). In decimal mode the inputs can range from -32 to 31 and in decimal mode they can range from -4 to 3.87.

Going from left to right starting with S0 the inputs are as follows, S0 = integer/fractional input with high being fractional, S1-S3 are the control bits to determine the operation (000-111), S4-S9 being operand B, S5-S15 being operand A. In fraction mode the top 3 bits are above the radix and the signed (2’s comp.) bit and the 3 lower bits are the fractional component. Other than the 7 segment display the only other outputs are the 3 flags, LED[13] is negative, LED[14] is ERROR, and LED[15] is overflow.

To the right are some sample outputs of some various operations

VI. Conclusion

Overall This was a great project that was a good wrap up of the material covered across the entire semester. It covered many of the topics discussed in the course and was a very interesting design to work on. Despite all of the timing issues involved with the synthesis process and minor issues with the floating point arithmetic the final implementation is a clean smooth design that has many functions and operations available to the user.

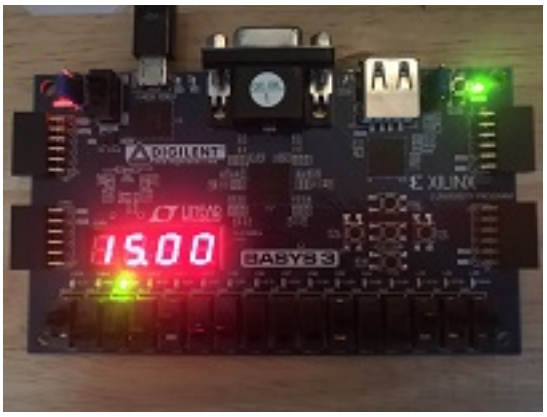

Fig. 1. -3.75 / .25 = -15 Note, the negative flag is set

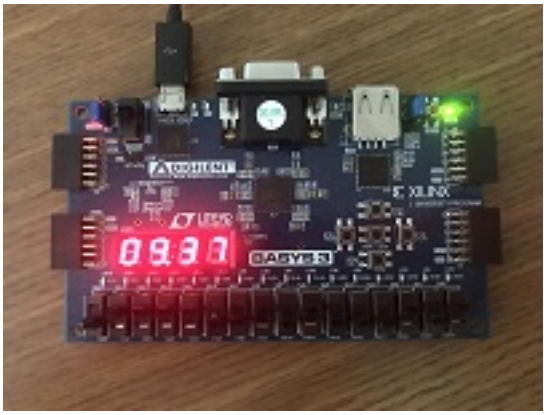

Fig. 2. 3.75 x 2.5 = 9.37

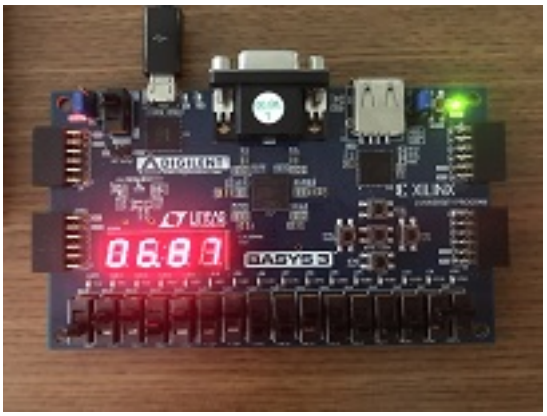

Fig. 3. 3.37 - (-3.5) = 6.87