Floating Point Calculator

Dylan Powell, Brandon Troppens

Electrical and Computer Engineering Department

School of Engineering and Computer Science

Oakland University, Rochester, MI

e-mails:dmpowell@oakland.edu, bjtroppens@oakland.edu

Abstract - **This final project will consist of an AXI4-Lite interface and an AXI peripheral, that acts as a calculator, that supports the mode selections: addition, subtraction, and multiplication. Data will be fed into three slave registers and calculated data will be read through the UART terminal. An AXI Peripheral and AXI interconnect will be constructed such that the digital system can interface with the programmable logic implemented in the ARM processor within the ZYBO Z7-10 Board.**

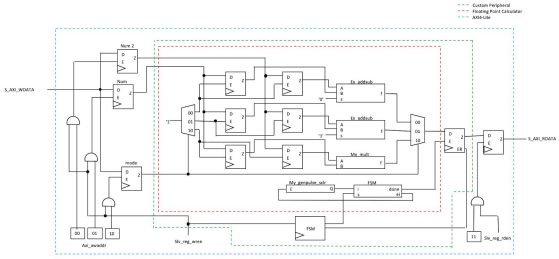
I. Introduction

This report will discuss the inspiration, method, and expected results of this project.
The inspiration for creating a calculator was learning about AXI4-Lite interface architectures. The team found it interesting to learn about AXI peripheral architectures and how they could be interfaced with a software application through utilization of the ARM processor inside the Zybo Z7-10. One of the many benefits of AXI interconnects is the ability to interface between hardware and software through the programmed ARM processor within the Zynq-7000 series PSoC. The scope of this project aims to take advantage of the possibilities given by the advanced extensible interface bus. The AXI peripheral is designed to read three 32-bit words, perform the desired operation, and output a 32-bit word. This will then be passed through the AXI interconnect to be processed on the ARM processor, and the serial data will be received and displayed on the SDK terminal via UART.

II. Methodology

A. Digital Circuit Datapath

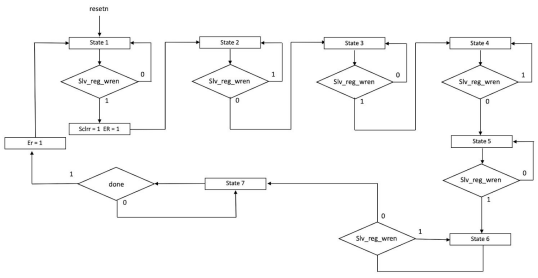
The method chosen to accomplish this includes an AXI4-Lite that has 3 input slave registers, one buffer register and one output slave register.



The 3 input slave registers are fed 2 numbers and an operation. Once the registers have data and are enabled, the 2 numbers and the mode data are sent into the custom peripheral then into the floating point calculation block. Data is then added, subtracted, or multiplied depending on the mode data inputted. An FSM internal to

the calculation block will control when calculation is started and finished along with a done signal to the external FSM. Data is then sent to the buffer register. From here, data is sent to the output register awaiting the external FSM to enable the output slave register which is read by the software application. The SDK software application will then output the data within the SDK terminal via UART.

Below is a flow diagram of the FSM located external to the custom peripheral (external FSM).



The purpose of this finite state machine is to control how data is inputted and outputted to the slave registers. The first step has the purpose to clear whatever data may be left over in the output slave register. Then, steps two through six allow the input slave registers to input data into the floating point calculator correctly. This is controlled by the slv_reg_wren signal, as this is how these slave registers are enabled.

Once the FSM reaches state seven, it will check if the internal floating point calculator FSM has sent out a done bit. Once that is determined to be high, then the output buffer register is enabled.

While these FSMs can be condensed into a single FSM, having them work in tangent allowed for an easier time debugging the circuit. This would provide a much more stable calculation method, even if it is sacrificing processing time for smaller calculations. The consistency in terms of processing time between different calculations was determined to be the best approach for this circuit.

The internal portion of the floating point calculator consists of a demultiplexer, a number of registers, the actual calculation blocks, an internal finite state machine controlled by a counter, and an output multiplexer. The input data for the two numbers are held at registers that are only enabled depending on the intended calculation method. Once data is inputted to select whichever calculation method is desired, the demultiplexer will send an output high signal to the input of a pair of registers. These registers then feed into the respective addition, subtraction, or multiplication circuits. From there, the output of the calculation circuit runs to another multiplexer, which is also controlled by the same input demultiplexer selection scheme. This is where the floating point calculator circuit outputs data to the buffer register.

B. Software Routine

This section will comprise the routine for the software interface portion of the project. The software routine in this application is very similar to those that were learned in the lab portion throughout this course. For this project, three words are written to the input slave registers. Next, after the input is processed, the result is outputted to the last slave register.

After configuring the digital system, writing a software routine that interfaces with it is critical to ensure that the application is working as intended. The software routine can be best described with the pseudo code as follows:

1. Define the base address.

2. Write a 32-bit word to slave_register_0. This is number 1.

3. Write a 32-bit word to slave_register_1. This is number 2.

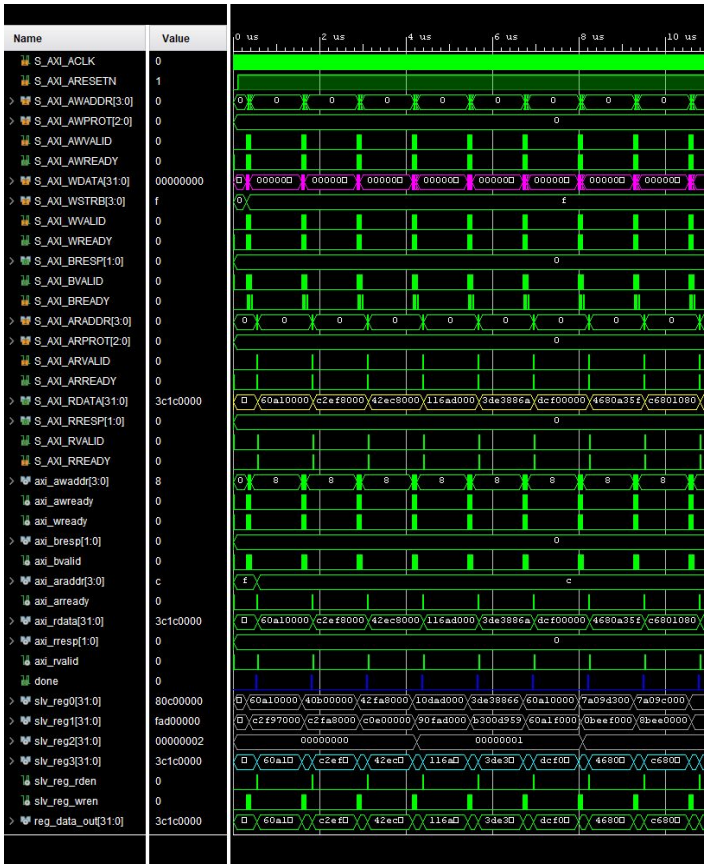
4. Write a 32-bit word to slave_register_2. This is the operation.

5. Display the calculated value from the output register, slave_register_3.

Steps 1-5 describe what the software application routine is, and how it is implemented. Step 1 describes assigning a base address to the processing system. Step 2 describes the first input on the S_AXI_WDATA line, which becomes the first number in the equation. Step 3 describes the second number in the equation. Step 4 describes which function to perform on the imputed numbers one and two. The final step, Step 5, describes how the calculated floating point number is read on the output slave register. Data is also written out the S_AXI_RDATA line.

III. Experimental Setup

Once the entire circuit was constructed, the team had to verify the calculator was working correctly before interfacing it with the software routine. So, a test bench was created for the AXI interface. The test bench essentially does the same thing seen in the pseudo code as seen in the software routine section. Data is inputted into the slave registers to select the numbers to operate with, and also the calculation operation is chosen, whether it be addition, subtraction, or multiplication. A selection of floating point calculations were implemented into the testbench to verify that the AXI interface was working as intended. Below is a picture of the waveform produced by the test bench.



This simulation displays how the input data S_AXI_WDATA maps to each of the three input slave registers. S_AXI_WDATA is seen in the magenta color. Once the data is captured by the first three slave registers, as displayed as the gray signals, it is processed by the Floating Point calculator where it is processed for 11 processing cycles. Once this operation is complete, the floating point calculator will send a done bit out, and is seen colored in dark blue. This will go to the FSM for the peripheral, which will enable the output buffer register. Once this register is enabled, and the output slave register is enabled by the signal slv_reg_rden, data will be outputted from the circuit to the S_AXI_RDATA line, which can be seen by the light blue and yellow signals.

One major challenge that the team had to work through was the SDK application not interfacing with the hardware as intended. The team noticed that the simulation worked as intended,

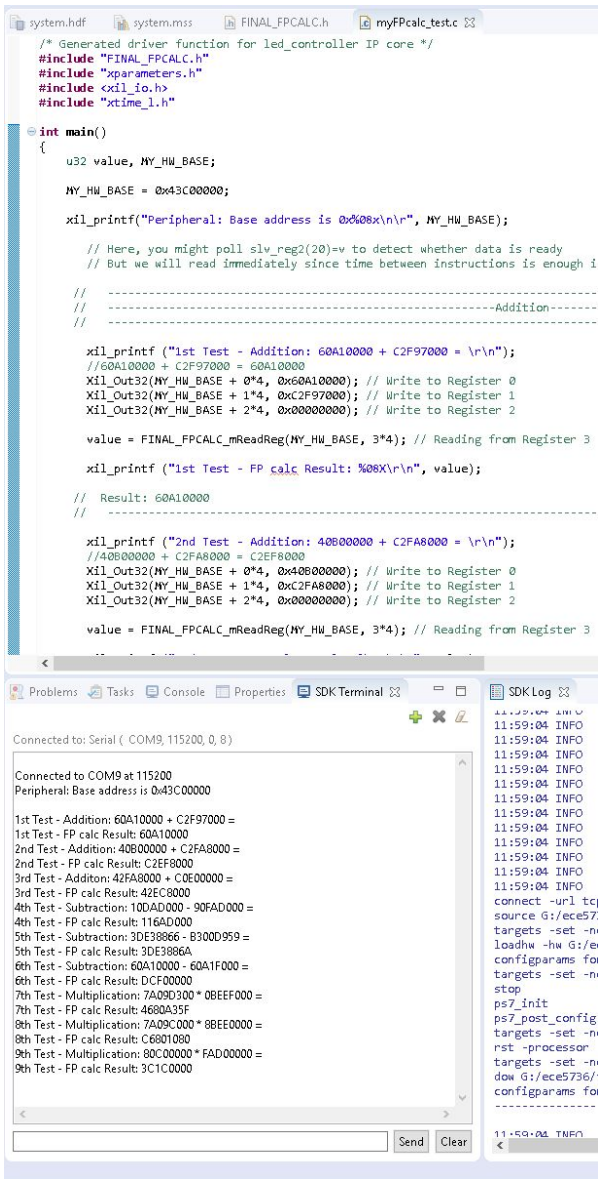
however the results of the software application were not the same as the simulation. One such critical task that had to be reworked was the finite state machine of the floating point calculator peripheral. An inconsistent trigger signal, in combination with incorrect design check iterations lead to mistakes that would only be seen upon close inspection. Many mistakes such as these were explored further with repeated testbench simulations and analysis of proper signal flow.

Another hurdle that the team had to work through was identifying normal operation from overflow and underflow in terms of the multiplier circuit. Addition, subtraction, and multiplication all calculate with the same number of bits on the input and the output, so extra bits are not allowed. Similarly, underflow cases have to be considered as well, as their representation is not possible out of range with the associated number of bits. When it came to the multiplier circuit, the team figured the result could not be implemented when cases of overflow or underflow were inputted.

The team attempted one method where the adder/subtractor circuit would take the carry bits of the most significant bit and the most significant bit plus one and xored them together to determine if there was overflow. Another method that was attempted was to sign extend the most significant bit of both the signal ex, which is the output of an adder/subtractor, and sign extending the bias. From there, the two most significant bits would be analyzed to determine if the number was negative, or if the number was too big for the final output to read. Both methods proved to be unsuccessful when implemented in the multiplier circuit. With that, an improvement to this calculator circuit would be to include the detection and proper output for cases of overflow and underflow.

IV. Results

The calculations were first tested in the Vivado test bench waveforms using a test bench simulation file. Next the team verified the calculations using the UART terminal on the SDK software. To verify the floating point calculations were correct, they were compared with hand calculations. The hand calculations followed the same procedure that was used for Homework #4 and in Unit 1 notes. In the end, the terminal displayed the same data that was hand calculated. The SDK Terminal interface produced the following results, given the different input conditions:



One major lesson that the team learned throughout this project was the importance of testbench simulations. The team constructed multiple testbench simulations; one for the add and subtract circuit, one for the multiplication circuit, one for the floating point calculator top level circuit, and also one for the AXI interconnect. Throughout the development of this project, testbenches at each step proved essential to the proper configurations of the modules. Some modules had to be tweaked differently than what was initially designed to produce desired results. There were multiple occurrences of having to start fresh with a module that does not seem to be producing the expected output, but the code looked correct. Many small errors and overlooks can result in a faulty circuit block that proves tough to debug. This is sometimes the best option, especially when no errors are generated through synthesizing the code or running simulations. Once acquainted with the system and how it should operate, being able to go through each signal in a testbench waveform is critical to ensure that everything was working as intended.

V. Conclusions

This project reinforced and enhanced the methods behind calculations using floating point numbers. During the semester, the course required students to hand calculate, but not implement a calculator for floating point numbers. It also reinforced knowledge on how to make custom peripherals. With this knowledge leads to other possibilities of more complex and complicated designs. Throughout this course and through this project, the team learned the proper way to interface this custom peripheral with a software routine by an AXI interconnect. Due to time constraints and other restrictions, the divider calculation method could not be implemented. Adding the divider portion would be an improvement to this circuit.

VI. References

[1] Llamocca, Daniel. “Unit 1 - Computer Arithmetic “. ECE-5736 Reconfigurable Computing. N.p., n.d. Web. 8 June 2020.

[2] Llamocca, Daniel. “Unit 5 - Embedded Systems in PSoC “. ECE-5736 Reconfigurable Computing. N.p., n.d. Web. 8 June 2020.