Developing a logic unit to command a system that will

correct air traffic control communications

Dillon J. Capalongo

Mentored by Jarick Cammarato

Science and Mathematics Academy

**Executive Summary**

This project will develop an Arithmetic Logic Unit (ALU) to command a system that performs error detection, location, and correction in air traffic control. Future researchers involved in air traffic communications will apply this research to complete the entire correction system. This will improve the efficiency of secondary surveillance radar involved in data transferring, which is key to supporting more planes in an airspace with a quick refresh rate. Development of the ALU will begin with a thorough planning stage, where diagrams and drawings of it will be produced to understand the component on a fundamental level. Then, custom code will be created based on the specific requirements of the microprocessor and a testbench of it will be simulated in ModelSim to detect and resolve issues. Finally, the ALU will be implemented onto an FPGA, and an interface outside of the computer will be connected so that it can demonstrate its ability to perform addition and subtraction, make comparisons and potentially read and write commands with prototypes of other components. This and the development plans created will be the result of this project, acting as the foundation for further construction of this efficient error-correcting system.

*Keywords:* Arithmetic Logic Unit, ALU, sliding window method, air traffic control, secondary surveillance radar

**Introduction**

Civil aviation is crucial across the entire globe, transporting people from one place to another safely and quickly. An important aspect of aviation that makes safe flight possible is ground support via Air Traffic Control (ATC) towers. These towers receive information regarding an aircraft’s fuel levels, their location, altitude, and more (Stevens, 1988). They use this information to give each plane a flight path integrated into other air traffic. The military also communicate to aircraft in their base’s airspace, interrogating them with a system known as Identification Friend or Foe (IFF). IFF talks to each plane as they enter the airspace, asking if they are friendly or not. If they respond at the proper times, they are considered friendly and can pass, otherwise they are considered a threat (Stevens, 1988). An interrogator system is also used in commercial flights using Mode Select, or Mode S, waveforms in Secondary Surveillance Radar (SSR). Commercial flights use Mode S to distinguish planes from one another and makes it possible to organize their information (Chang et. al., 2000). This radar system, as shown in Figure 1, sends narrow pulses of Mode S but slowly rotates to cover the entire airspace. SSR sends Mode S to a transponder located inside of the aircraft, that receives the command to send its information. This informative reply, known as a downlink, is sent at a different known frequency than the initial waveform, or an uplink, in order to avoid receiving clutter from birds, or even rain (Chang et. al., 2000). The new reply is then interpreted by the radar, and the information is collected and displayed in the ATC tower.

Figure 1 (right): An ATC radar system. SSR is the flat surface on the top of the antenna. This system rotates over time, allowing it to send and receive information to all planes in its airspace.

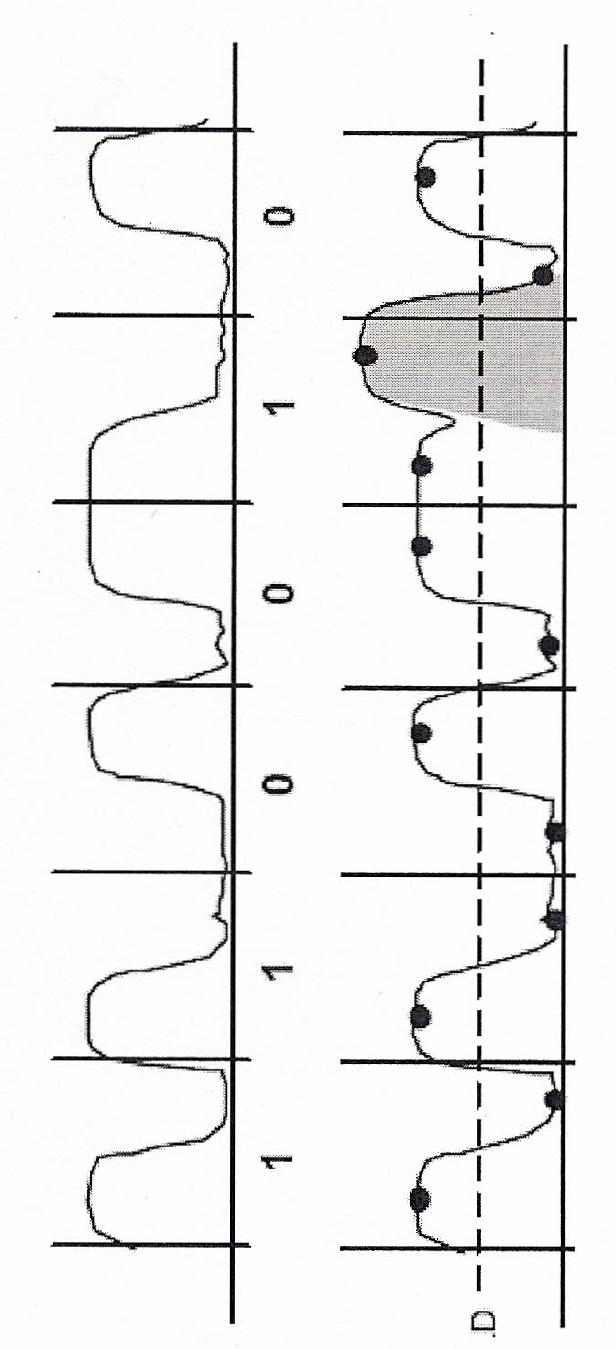
 Even though clutter responses are avoided by SSR, errors can still occur. As previously stated, SSR sends Mode S in separate, narrow paths to target an airplane, but in a congested airspace nearby airplanes can also pick up the command and send their information. The data from the first airplane would not be completely accepted before the next data arrives, overlapping the two data sets and creating a large amount of errors, known as a burst error syndrome. This syndrome as shown in Figure 2, causes the data to be unreadable and requires the transponder to send the data again. This reduces the efficiency and refresh rate of the radar system, which lowers the precision of the information and can even reduce the threshold of planes that the radar can detect (Chang et. al., 2000).

Figure 2 (left): An example of a 1-bit burst error. The top waveform is the data sent correctly; a 1 occurs at the beginning of the clock cycle, and a 0 occurs at the end. The bottom waveform has a noticeable burst error, because it has an inconsistent voltage.

The process of resending data can be avoided if the damaged data can be fixed. To fix the data, there are 3 processes that must occur: error detection, location, and correction. Error detection is already implemented into Mode S, using a field of parity bits that can detect single errors and the characteristics of cyclic codes to check burst errors. Random, single errors are detected by parity bits, which are extra bits added to a code word that keep the number of 1’s in the code even. If the number of 1’s becomes odd as it is sent, then an error has occurred (Castagnoli, Brauer, & Herrmann, 1993). If there are multiple errors, known as a burst error, then there is a chance that multiple 1’s are added, keeping the number even and falsely clearing the code word. Another system, known as Cyclic Redundancy Check (CRC), is needed to detect these burst errors.

CRC was designed to detect any errors through the qualities of cyclic codes. A cyclic code is a code word that can be cycled, or have its bits shifted from their original location, and still be able to function as a multiple of a generator polynomial G(x). This means that no matter how many times the code is cycled, it will still be divisible without producing a remainder. If there are any changes to the original code, then CRC will detect this in the remainder produced by polynomial division (Gertz, 1984). To make a code word cyclic, the message M(x) is shifted left xb bits, where b is the number of bits shifted, to make room for the parity bits, or bits necessary to store the remainder. Then, the result xbM(x) is divided by G(x) to produce the remainder R(x), which is then added to xbM(x) to create the encoded message C(x):

C(x) = R(x) =

This process prepares the Mode S message for transmission, so that either the uplink or downlink code can be tested for consistency through CRC (Gertz, 1984). CRC takes the encoded message and reverses the process to decode it, producing a remainder of 0 if no errors are present.

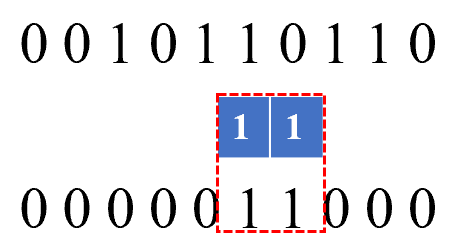
 If an error is present, meaning the remainder is something other than 0, then an error syndrome has occurred, and the sliding window method’s results must locate the error. In short, this method compares the data to its corresponding confidence levels, which are produced in Figure 2, trying to match questionable values to the remainder and locate the error. A 2-bit sliding window example is shown in Figure 3, where the blue remainder, known as a check sum, matches the confidence levels below it, signifying that the two corresponding data bits are errors.

Figure 3 (left): a 2-bit sliding window. The check sum (middle) aligned with the flagged confidence bits (bottom), meaning the corresponding data bits (top) are errors and need to be flipped.

When the error is found, the flagged data bits are then corrected by flipping them to the opposite value. In Figure 3 the corresponding “1 0” would become “0 1”, and the data would be sent off. This plethora of systems work together to create a fully functional error detection, location and correction system (Gertz, 1984).

The focus of this project will not be on the entire system, rather the construction of a component on the microprocessor that commands each system, determining when to run, what action needs to be performed, and what data needs to be sent. This microprocessor will take inputs from the CRC and sliding window systems and perform logic and comparisons to produce instructions for the next action. There are many ways to design and program a microprocessor, and for this project the Microprocessor without Interlocked Pipelined Stages (MIPS) architecture will be the foundation for a custom architecture.

MIPS is a reduced instruction set computer (RISC) instruction set architecture (ISA) designed to run almost entirely through system registers instead of data memory (Dandamudi, 2005). Registers can be viewed as temporary memory, made to hold data in such a way that it can be directly manipulated, moved or compared. Registers are vital to communicating between systems and acting as a base to perform logic on (Dandamudi, 2005). The MIPS architecture implements 32-bit registers, meaning each system can send instructions and data that are in 32-bit intervals. MIPS has 5 different major components of the microprocessor, shown as squares, including the Arithmetic Logic Unit (ALU) in Figure 4.

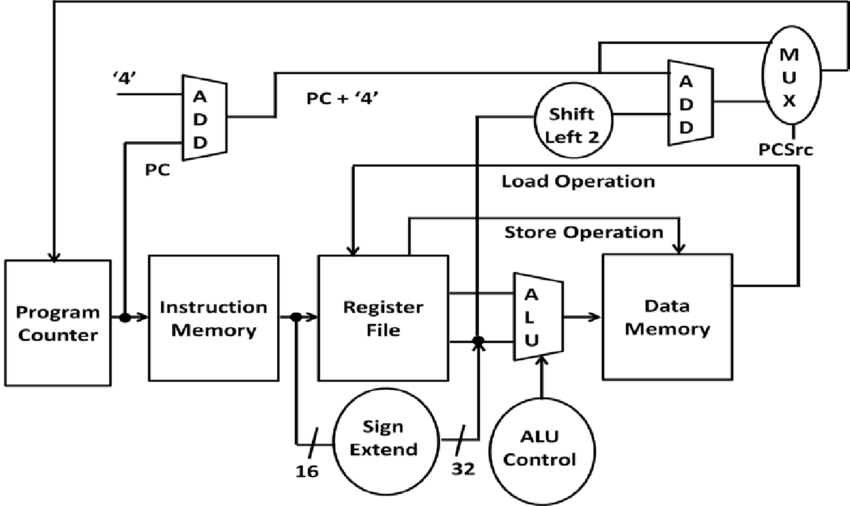
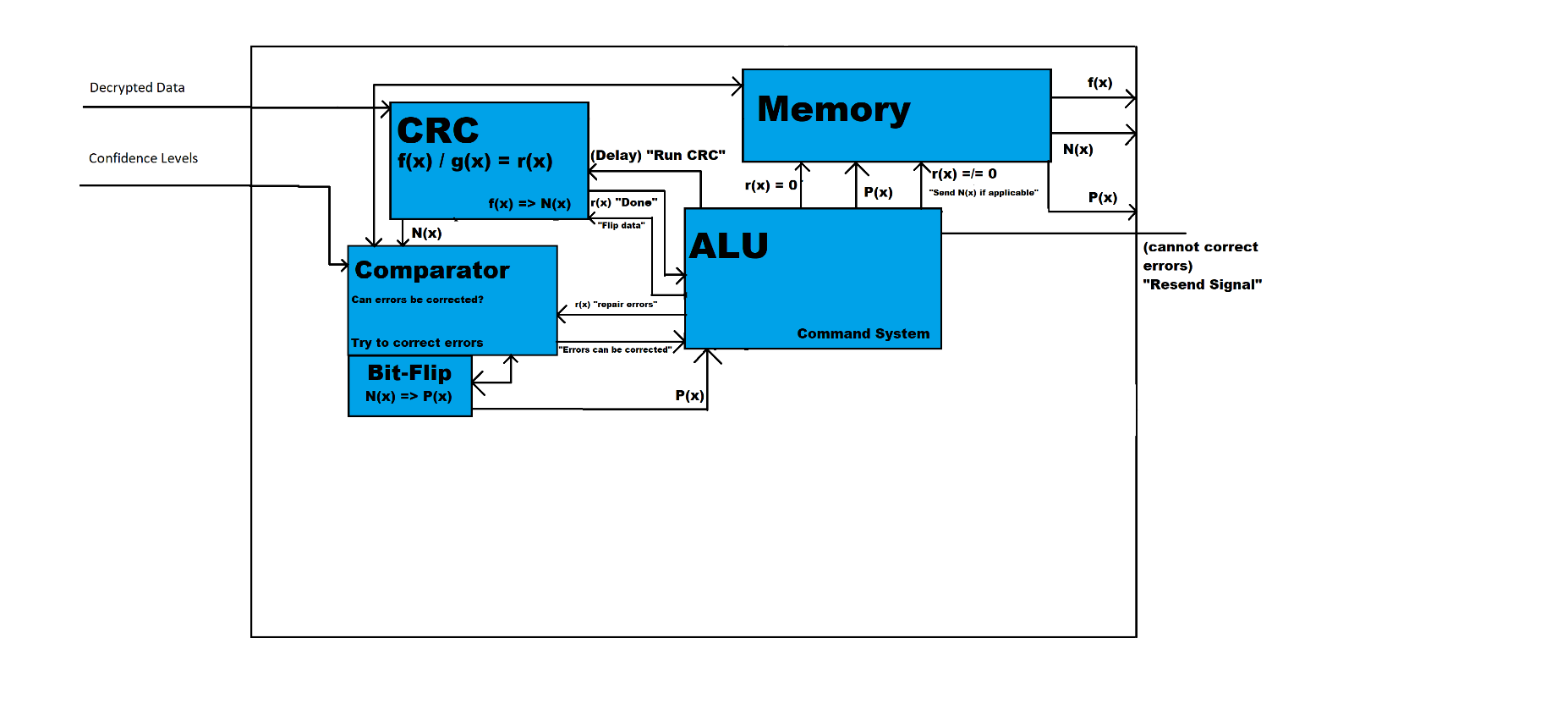


Figure 4 (left): A basic MIPS architecture design. Alterations to this default design may occur as the exact requirements of the error correction system are realized.

This project will be focusing on the ALU, and its interactions with other components of the microprocessor. The ALU must perform logical operations for the microprocessor, including addition and subtraction, reading and writing commands, and making comparisons. This occurs through manipulation/comparison of two operands, with an operative input from the ALU Control, producing an output that the processor can use (Gertz, 1984). The ALU can be designed in different ways based on what is needed from it. For example, this microprocessor may not need to multiply or divide to determine if the data has been successfully manipulated by the sliding window method.

**Materials and Methods**

As stated previously, this project will focus on planning, implementing and validating a custom ALU for use in the microprocessor of an error-correcting system. Planning the ALU will require multiple programs and systems to correct or manipulate the code to be used on an Alchitry Au board, a Field Programmable Gate Array (FPGA) digital logic board. Before anything can be programmed onto the FPGA, the functions that need programmed and the logic behind them will be organized into a full Firmware Development Plan (FDP). An FDP is a loose planning system that breaks down the functions and requirements of a system into a descriptive list of each, allowing conflicts and decisions to appear more obvious than they would inside the program. An FDP includes but is not limited to block diagrams of each system, as shown in Figure 5, activity diagrams, state machines, logic tables, interfaces, initial values or preset inputs, and any other problems that must be resolved through code.



Once the FDP has been created, ALU development will commence. The first iterations of code will be designed in ModelSim, creating test benches that can simulate running the code as if it was on the FPGA. This will speed up the process of creating new iterations of code and reduce the risk of damaging the FPGA due to critical errors. On a computer, Vivado will manipulate the validated code so that it can enter Alchitry Loader, a software that will communicate with the FPGA via a USB-C cable. The ALU will also output values onto a separate display from the computer. Using a bread board, LEDs, resistors and other wiring, a prototype of the ALU’s display system will be created. This system will help make outputs more noticeable; for example, an LED could light up when the ALU receives a message that an error is present.

Figure 5 (above): Iteration 3 of the full CPU system’s block diagram. Arrows indicate transfer of data or messages, and the blue boxes indicate a componet that will need programming.

Once the first version of the ALU has proven successful, the remaining task will be validating, or error-correcting, the code and finalizing the interface. ModelSim can break down the code into its waveforms over time, as shown in Figure 6. This will be a form of discovering errors in the system.

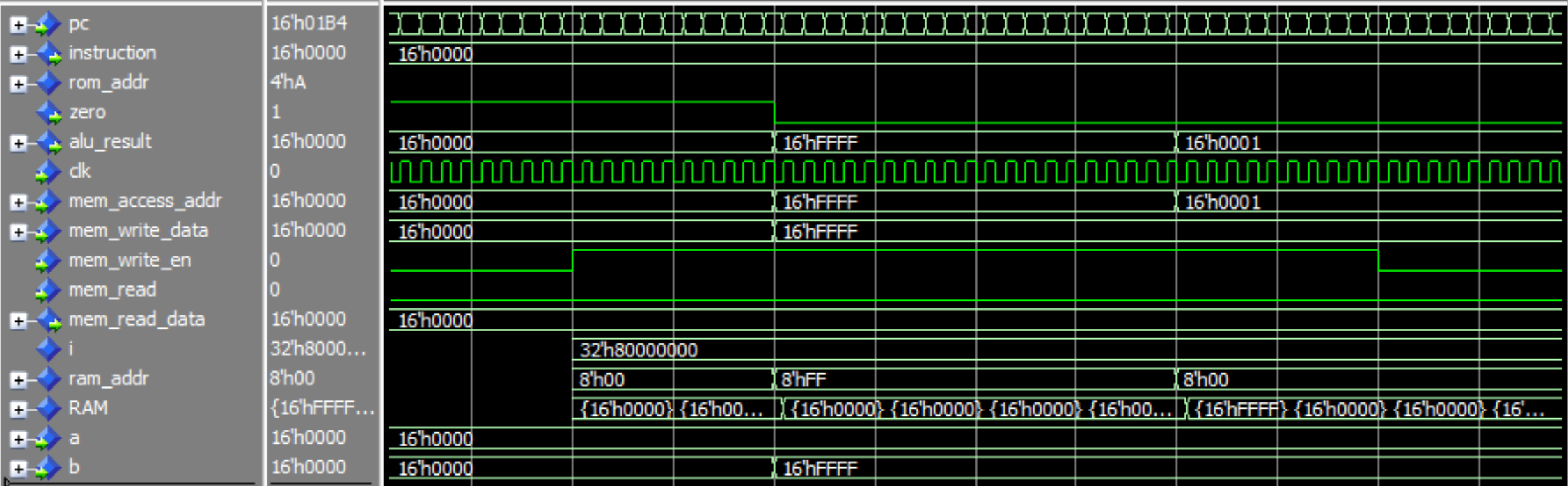


Figure 6 (above): A simulation of a portion of MIPS architecture. In this example, the data 16’FFFF was stored in the address 16’FFFF.

GitHub will also aid in the validating process, by making iterations of code publicly accessible. This allows the code to be tested and reviewed on many platforms and keeps a detailed timeline of objectives relative to programming.

Once the ALU has been fully created, an interface will be prototyped through a breadboard with components that can act as inputs and outputs of the system. This prototype will then be implemented into a shield for the ALU, as shown in Figure 7. The shield is the final version of the interface, that allows the ALU to be experimented with outside of a testbench.

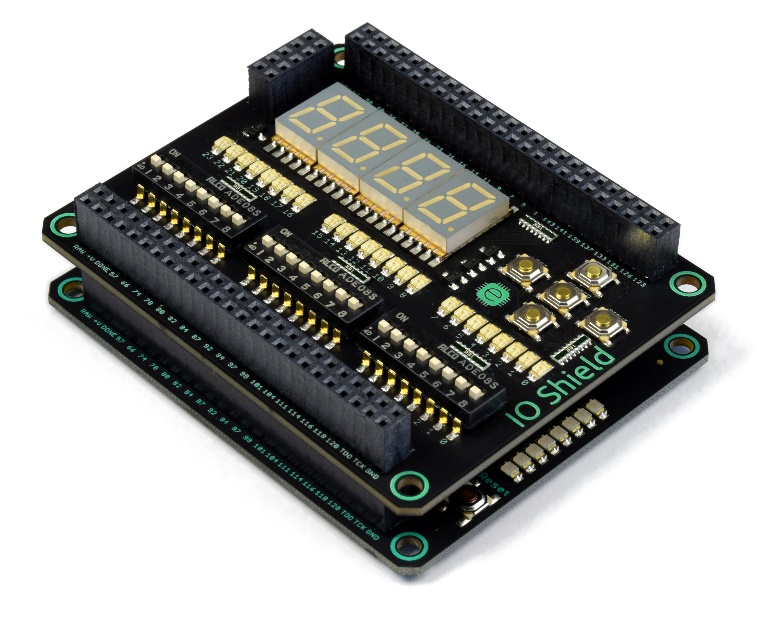


Figure 7 (left): An example of a shield for an FPGA. It contains a series of buttons and LEDs that can display or send information to the ALU, which is seen below it.

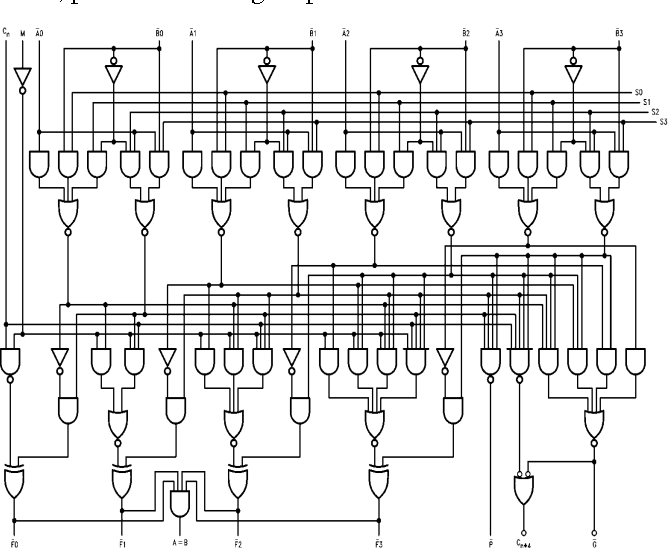
Other aspects of the microprocessor will be designed to help simulate the ALU’s functionality. An FDP will be created for the entire processor, focusing on what interacts with the ALU most frequently. The CPU that creates the entire error detecting and correcting system may also be designed in an FDP, and other components of the microprocessor may also begin development if time allows.

**Results**

This project will produce a custom ALU that will perform logic and comparisons for an error correcting system. The ALU will be connected to an FPGA shield, and it will receive inputs and display outputs through it. It will also demonstrate addition and subtraction and perform comparisons between data sent from instruction sets. The ALU may not be able to make decisions significant to the error-correcting system because those systems may not be completely developed. Therefore, preset inputs and instructions will be generated that will run through the component and produce relevant outputs that can be understood. For example, it might receive two preset operands and an operation and then display the operation being performed and the result.

A firmware development plan for the ALU and potentially other components will also be completed. The ALU’s logic gates will be displayed and explained through a VHDL ModelSim program. This logic system, similar to the example shown in Figure 8, will detail the inputs and outputs and how they are manipulated in binary coding. More general block diagrams will be completed for most of the other systems in the microprocessor, including the instruction memory. These diagrams will help explain how certain preset inputs to the ALU are created, and why they are necessary.

Figure 8 (left): A logic gate diagram of an ALU. Each shape describes a different Boolean operation, including NOT, AND, XOR, and others. The inputs and outputs will also be described with this diagram.



The microprocessor that contains the ALU will also see significant design and development. As stated before, this may not be completed in its entirety, though the register file and ALU control will begin development, which are the most closely connected systems to the ALU. The ALU control will receive preset instructions and produce an operation to perform, and the register file will send operands to the ALU when necessary.

Finally, the support circuitry that is necessary for each system and the interface to communicate to each other will be created, which will allow the ALU and other components to communicate. These communications will then be displayed on a breadboard, instruction sets will be inputted to display that each system can communicate and produce a relevant output. It is Caimportant to note that development time is not exact, and the extent that these objectives are completed will vary based on new obstacles and problems that occur as each system is debugged. The idea of a sliding window error-correcting system has never been attempted before, and underlying issues could be discovered as this system is developed.

**Application**

Each of these systems act as a crucial puzzle piece to the sliding window error-correcting system. This process of error correction is new, so the more that is laid out and designed, the easier it will be for other scientists to use this work as a foundation for an official system used in airports to increase efficiency in radar. Making SSR more consistent by correcting errors is key to allowing the system to rotate faster, increasing the refresh rate of data and increasing the threshold of planes that a radar can contain in an airspace. This project acts as a proof of concept that future studies will use as evidence to continue production of this system. Without a full design of the system, there would be no evidence that combining CRC and sliding window techniques is effective. This will be the start that this system needs to prove that is can be done, so that it can be developed later on and tested for improvement in error correction rates.

**Timeline**

As states previously, this project does not have a final stopping point, as writing custom code will take a varying amount of time and there are many levels of work to be completed. There may be underlying challenges that will slow progress, or the custom code may be easier to write than anticipated. There are milestones, though, that can break down the ALU production. Rough dates can be applied to these milestones and divided over the course of the school year.

|  |  |
| --- | --- |
| **Goal** | **Approximate Date (Month/Date)** |
| Finalize the Firmware Development Plan for the ALU. This includes a document describing the ALU’s inputs and outputs, a block diagram of the ALU and the microprocessor it resides in, and a detailed description of what occurs inside the ALU and the actions it performs. | 11/11 |
| Finish the 1st iteration of the ALU code. | 11/27 (thanksgiving break) |
| Create a test bench and demonstrate its functionality on ModelSim. | 12/12 |
| Run tests and finalize the code and test bench for the ALU. | 1/6 |
| Implement the test bench onto the FPGA and demonstrate proper inputs and outputs with a prototype interface. | 2/10 |
| Connect the IO Shield to the FPGA and demonstrate the ALU taking inputs and outputs from that interface. | 3/6 |
| (IF TIME) Finish FDPs for other components of the microprocessor and begin designing and programming the ALU Control and Register File. Demonstrate ability to understand the functionality of each component and show progress made on development. | N/A |

**References**

Castagnoli, G., Brauer, S., & Herrmann, M. (1993). Optimization of cyclic redundancy-check codes with 24 and 32 parity bits. *IEEE Transactions on Communications,* *41*(6), 883-892.

Chang, E., Hu, R., Lai, D., Li, R., Scott, Q., & Tyan, T. (2000, December 15). Air Traffic Control Radar Beacon System (ATCRBS). Retrieved October 15, 2019.

Dandamudi, S. P. (2005). *Guide to RISC Processors for Programmers and Engineers*. New York, NY: Springer.

Gertz, J. L. (1984). Fundamentals of Mode S Parity Coding. Lexington, MA.

Stevens, M. C. (1988). *Secondary surveillance radar*. Boston, MA: Artech House.