# Complex Number Calculator
# EE 552 Project

Noah Aklilu
Marco Castellon
Curtis Wickman
April 7, 1998

# Table of Contents

# 1 Abstract

This document discusses the design, implementation and verification of a complex number calculator capable of performing addition, subtraction and multiplication on 8-bit integer numbers.  The calculator accepts input in Base10 using Reverse Polish Notation (RPN) from a keypad and displays its results in Base10 on a LCD.  The system is composed of an Arithmetic Logic Unit, a Keypad Interface Module, an LCD Interface Module, and implemented using an Altera University Program Board which has a 10K20RC240 FPGA mounted on it.  For user interaction, a keypad and LCD are wired onto a prototype board and connected to the FPGA thorough a ribbon cable patch.

# 2 Project Overview

The specifications for the project ask for a calculator that is capable of performing addition, subtraction and multiplication of 8-bit integer numbers.  The range of operands for an 8-bit implementation is –128 to 127 with the usage of two's complement to represent negative numbers.  For the addition and subtraction the results are also eight bits however multiplication operations can generate a 16-bit result which the calculator must handle properly.  The user will only see numbers in their Base10 implementation and does not have to be concerned with the binary representation used internally by the calculator.

The logic implemented in the calculator chip can broken down into three main modules, the ALU, the Keypad Interface Module (KIM) and the LCD Interface Module (LIM).  The cycle of operation begins when the KIM accepts two operands and an operation from the user.  The ALU then computes the resultant of the operation and makes it available to the LIM, which displays it on the LCD.  The KIM and LIM handle the conversion between Base10 and binary, the ALU only deals with the numbers in their binary form.

# 3 System Design

## 3.1 IALU Complex Number Arithmetic Logic Unit

### 3.1.1 Description

The IALU provides the calculator with its arithmetic capabilities. It accepts four binary signed numbers, representing the real and imaginary portions of the two operands in question, and provides the option of performing addition, subtraction or multiplication operations. The output is two 16-bit numbers, which represent the real and imaginary portions of the result.

### 3.1.2 Data path

The two principal components of the data path are an 8x8 multiplier and an adder/subtracter block that can perform either addition or subtraction. The input to the data path is held in four latches, which correspond to Real A, Imag A, Real B and Imag B. To handle multiplication, these values are switched into the multiplier using two multiplexers that can feed either Real A or Imag A to one input and Real B or Imag B to the other input. The multiplier is pipelined with 3 stages, which allows the values to queued in a particular order such that only two latches are required at the output of the multiplier to hold intermediate values. The latched intermediate values are feed into the adder/subtracter via two multiplexers and the result is finally stored in two latches that represent the Real and Imaginary portions.

Addition and Subtraction is handled by switching in the values of A and B into the adder/subtracter using the two multiplexers, and the control path selects the operation as addition or subtraction. The result can then be latched according to whether it is the Real or Imaginary portion. In the case of overflow or underflow the data path has an error signal which goes high when this occurs.

In order to minimize the effect on signal glitches on the data path, the latches, multiplier and the adder/subtracter are synchronous. This does add more states to the control path and increase the amount of chip space taken up the data path but the gain from synchronous operation is more than that from less chip area.

The approach taken when designing the data path was biased more towards space minimization than modularity.  The first attempt at designing the IALU used a more modular approach with separate addition, subtraction and multiplication sections.  This resulted in a component that took up 82% of the chip.  The current approach produces the IALU in 36% of the chip.

### *3.1.3 Control Path*

The control path takes responsibility for communicating with the calculator state machine as well as handling the operation of the multiplexers and latches in the data path.  When it receives a “start operation” signal from the calculator state machine, the control path goes through the necessary states before it latches the operation result and signals the calculator state machine that it is finished.

The control path does not take any responsibility for an overflow/underflow error signal from the data path and will continue its sequence until it believes it has completed the desired operation.  Considering that the control path is not the originator of the input values, it should not have to deal with an erroneous operation.

## *3.2 LCD Interface Module*

### *3.2.1 Description*

The LCD interface module generates the data for the external LCD Display Unit. It translates the contents of the memory module from a binary representation to the ASCII format; note that this is done for numerical characters only; alphabetic characters are generated directly by the LCD control state machine and sent to the output port.

### *3.2.2 Data Path*

The data path performs all the necessary steps to convert a 16-bit signed binary number to 5 BCD digits, which then get converted into their corresponding ASCII representation by just adding the prefix ‘3’ (0011 in binary) as the 4 most significant bits of the data byte. The issue at hand is the generation of the BCD digits from the signed binary number.

The first step in the process consists in transforming the binary number from signed to unsigned. Since the data received by the LCD data path can be either negative or positive, the most significant bit of the word is checked. When this bit is a ‘1’, the two’s complement of the number is generated; that is, the number is now unsigned while its absolute value remains unchanged. At the same time a flag is asserted to let the control path know that the received value was negative. With this information, the control path will know that it must display a negative sign on the screen. Now, if the most significant bit of the input was not a ‘1’, the output is just a buffered version of the input. The output of this inverting module becomes the input to the Parallel-to-Serial converter.

The binary-to-BCD converter requires its input data to be serial; therefore, it is imperative to convert the output from the two’s complement inverter. This is accomplished with a parallel-to-serial converter, which is implemented with a state machine. The bits of the 16-bit word, starting with the most significant one, are shifted out on each consecutive rising edge of the clock signal. When the least significant bit is shifted out, a flag is asserted to advice the binary-to-BCD converter that it will be receiving the last bit of the word. The state machine will wait in the Finished state until it receives an acknowledge signal from the LCD control path at which time it will go back to the idle state.

The binary-to-BCD converter was designed around an algorithm described in the application note “Serial Code Conversion between BCD and Binary” found at the Xilinx web page. The conversion is performed using a modified shift register. The binary data is shifted serially into the converter. Upon shifting each data bit, the contents of the register are double, thereby vacating the least significant position for the next data bit to be shifted in. The conversion ends when all the bits of the binary input have been entered.

At this time, the BCD digits will be available at the output in parallel form.

For the BCD digit to remain valid as it is doubled, a digit equal to or greater that 5 must not be double but converted into the proper BCD representation of its doubled value at the same time that a ‘1’ is shifted into the next higher digit.

It was previously stated that the digits, at the end of the conversion, are available in the ASCII format. This is accomplished by prefixing each 4-bit digit with the value “0011”.

When the conversion is finished, the digits as well as an instruction word generated in the control path, become the inputs to a data multiplexer. By using this data multiplexer, the control path can select which data to send out to the LCD unit at any particular time.

There are three control signals from the control path used to select the data to be sent out.

### *3.2.3 Control Path*

Upon reset, the control path issues a command sequence to configure the LCD unit. The LCD unit is configured for 8-bit interface, and dual-line mode. After initialization, the control path waits until the higher level calculator state machine issues the signal to write to the screen. The control path generates the read/not-write, register-select and enable control signals for the LCD unit. The control path is a state machine that takes care of all the timing to meet the set-up and hold times as specified in the LCD unit data sheet. It will also select which byte, either a data byte or an instruction byte, will be sent to the LCD unit. After all the digits have been written to the screen, the control path will reset the cursor to its base position, which is at the top left hand corner of the display. At this time, it will assert a done signal for one clock cycle and then move back to the idle state.

## *3.3 Keypad Interface Module*

### *3.3.1 Description*

The keypad interface module handles input from the external keypad and translates this input for use by the other modules. The converted user input is stored in the memAB module for access by other modules.

### *3.3.2 Data Path*

The data path handles the acquisition of the key presses including debouncing of keys. Input is taken from the keypad by using the drive and sense lines. The sense lines are pulled up using 10K resistors and the drive lines are initially at ground. When a key press occurs the drive line is shorted to the sense line and the sense line is pulled low. The hardware used for implementing the connections to the keypad is shown in Figure 1.

When a key is pressed and the sense lines change, the data path cycles through the drive lines driving one line high at a time in order to determine which key was pressed. Once the sense line goes back to 0 then we will have determined what key was pressed. This key is then debounced and registered with the control path. The control path has an output, which determines when the data path will read the next key.

### *3.3.3 Control Path*

The control path for the keypad controls the memory block, and signals the calculator state machine when a complete number has been acquired. Once the control path has a new key, it asserts the set, change sign, and write lines to the memory block after which it waits to acquire the next key press. Once an Enter key is received, the control path signals to the calculator state machine that a full acquire has been completed.

## *3.4 Memory Module*

### *3.4.1 Description*

There were two separate memory modules built. The first memory module holds the imaginary and real part of the two operands. The second memory module holds the imaginary and the real part of the result. The second memory module is simply composed of an instantiation of the ram_lpm module whereas the first memory module is quite complex and will be called memAB module henceforth.

The memAB module must be more complex in order to handle the calculator functions. The first requirement of the block is that it must hold the actual values. This was done using a ram lpm module and requires no explanation here. The other three operations required of this block are the ability to set the initial value, update the current value, and to change the sign of the value. Figure 2 shows the data path for the memAB module and the three operations mentioned above are described below.

### 3.4.2 Set the Initial Value

The “Ram Input” select mux controls which input is connected to the block. If the input to this module is set, then the memAB modules knows that the control path is setting the initial value of the ram block. Once the data to be written in ram is present at the inputs, then the control path asserts the write line in order to load the values into memory. At this point the setting of the initial value is complete

### 3.4.3 Update the Current Value

This operation is needed in order to find the new value of the decimal representation, which is currently displayed on the screen. Essentially every number past the first causes the existing value in memory to be multiplied by ten and the new number is added to the sum. This operation occurs when the set input line and sign input lines are both low.

This operation generates the new value by multiplying the current value by 10, shifting the value to the left by three and adding it to the current value shifted left by 1. This corresponds to a multiplication by 10 after which this value is added to the current value using a second adder and the result is fed to the input of the ram block. It should be noted that the ram input mux will only load the resulting value into the ram block if the shifting caused a change in sign.

### 3.4.4 Sign Change of the Current Value

This operation simply changes the sign of the current value of the ram. This is accomplished by subtracting the value from 0 and then adding 1. Since there are already two adders for the update operation we added some muxes to control the inputs to the adder/ subtractors. The carry in for the addition of 1 was connected to the change sign line so that a 1 was added only when changing the sign of the value.

## 3.5 Calculator Module

### 3.5.1 Description

### 3.5.2 Data Path

The data path for the calculator is just the creation of all the modules and the connections between them.

### *3.5.3 Control Path*

The Calculator Control path controls all of the modules within the calculator.  It controls the address lines to the memory blocks and controls when the next key is requested.

# 4 System Verification

The system was simulated using Maxplus2, with the entire system being tested.  The simulations are setup such that they emulate a complete session including operations with the KIM, ALU and LIM.  The results from this simulation are shown in Appendix C: System Simulation.  The simulations were run for an addition operation as well as a multiplication operation.

For actual hardware verification, the calculator is programmed into an Altera board, which is connected to the prototype board using a ribbon cable connection.  The prototype board has a 16x2 Sharp LCD and a Grayhill 4x4 keypad mounted on it as well as any other support hardware such as pull-up resistors.  The system clock is running at 10KHz due to the timing requirements of the LCD, to achieve clock division some additional logic was added onto the Altera FPGA in order to step down the 25MHz clock available on the board.

# 5 Conclusion

The project progressed quite well, and the group was able to overcome many hurdles including chip area, performing binary to BCD conversion for the LCD and others. The group was able to provide support to other groups who were stumped by similar hurdles, and currently the LIM and KIM are being used by other groups.

The tools used for the project, primarily Maxplus2, were adequate but could be improved or replaced. It was a common occurrence to have Maxplus2 lock up machines or leak memory until machines were unusable.

# 6 Appendix A : Figures

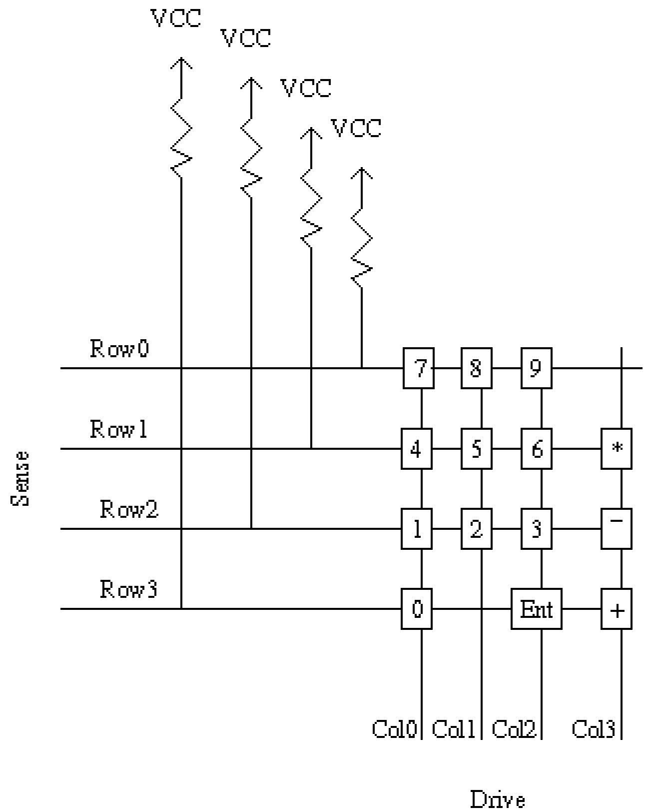


**Figure 1: Keypad Schematic**

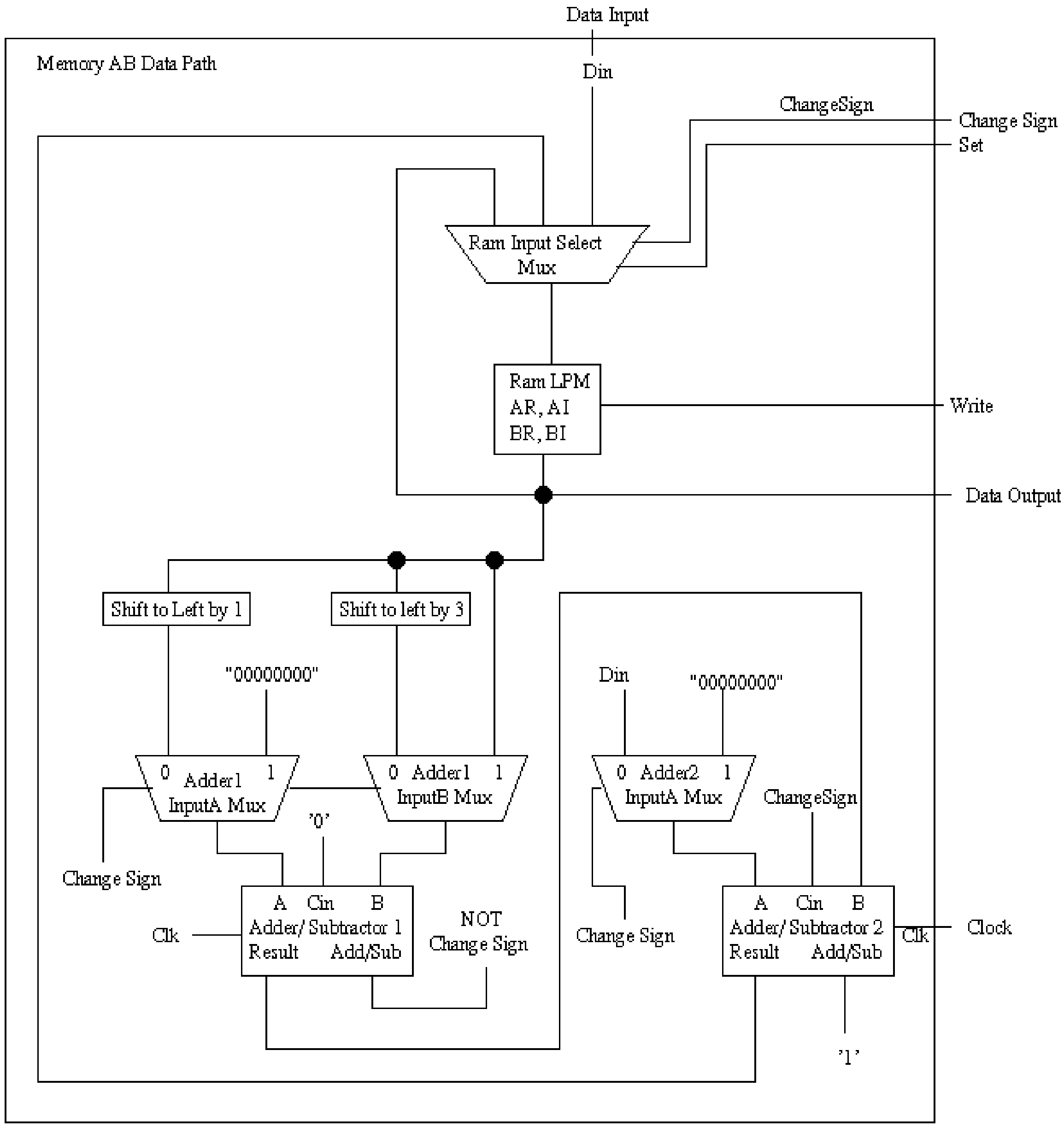


**Figure 2: memAB Data Path**

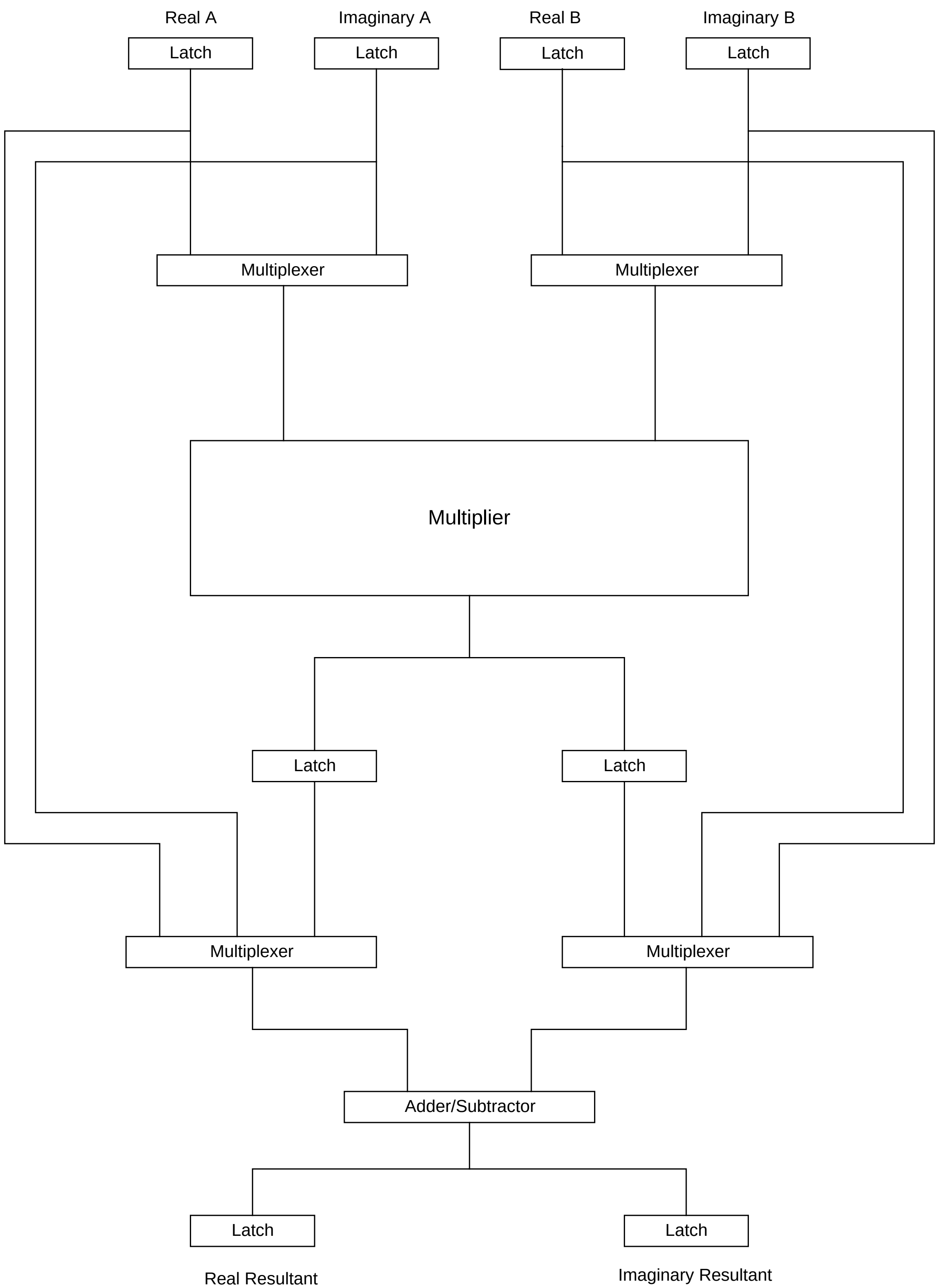


**Figure 3: IALU Data Path**

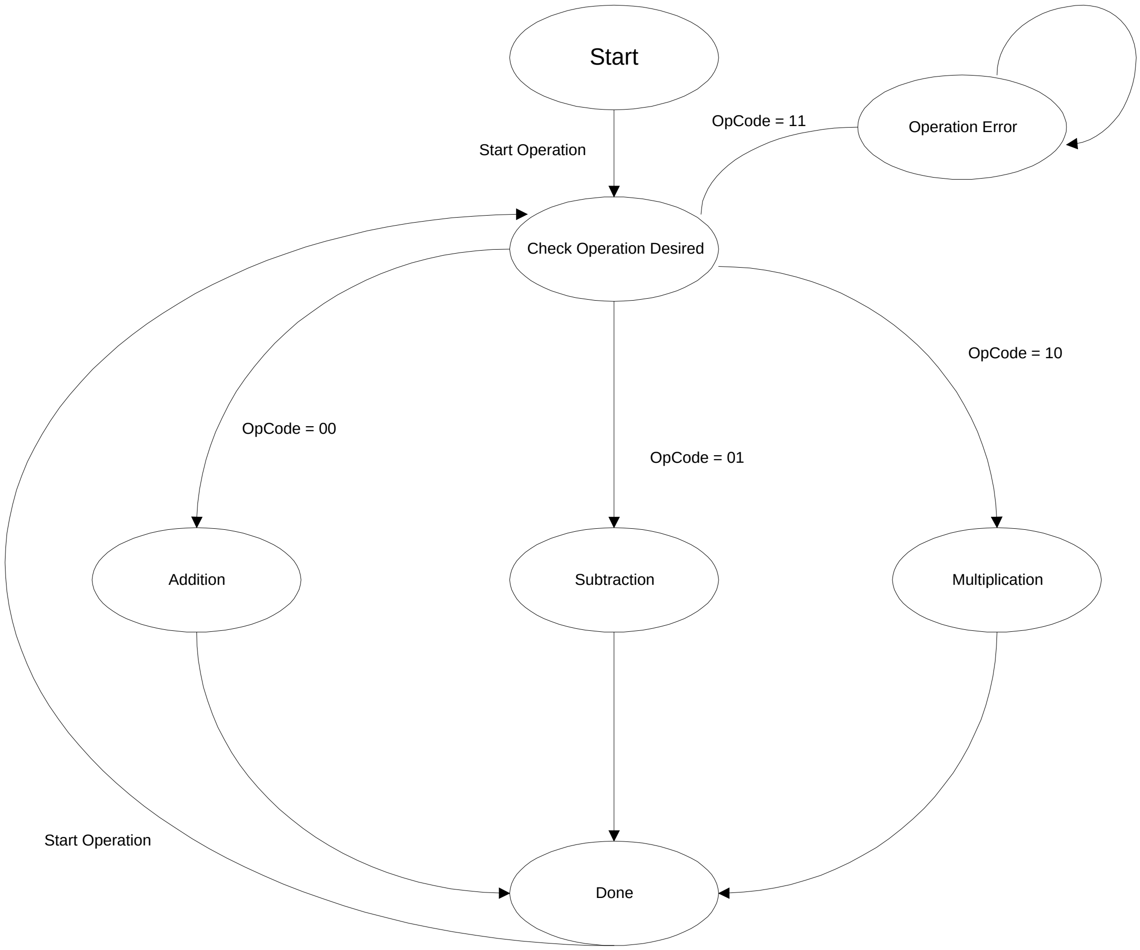


**Figure 4: Simplified IALU Control Path**

# 7 Appendix B: VHDL Code

# 8 DataSheets and Application Notes

# 9 Appendix C: System Simulation