Simple Digital Calculator

Implementing A Simple Digital Calculator Using An FPGA

Brandon Banchiu, Tasmina Ahmed, Ionatan Crisan
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: bmbanchi@oakland.edu, tasmina121@gmail.com, ivcrisa2@oakland.edu

Abstract - The purpose of this project is to construct a fully functioning, simple digital calculator that performs the four basic mathematical operations, namely: Addition, Subtraction, Multiplication, and Division of eight-bit unsigned numbers. The calculator will consist of a Nexys4 FPGA board which will manage all of the computation and input and output functions of the Digital Calculator. Input will be retrieved from a numeric USB Keypad, and output will be displayed on the Seven-Segment Displays. In order to implement this Digital Calculator the main functionality has been divided into four distinct portions. The calculator will consist of an input block which will accept input from the numeric keypad and interpret this into a meaningful digital signal for the application. These digital signals will consist of two 8-bit expressible numeric inputs and an operator (+, - , *, /). The second portion consists of volatile memory (RAM) which will allow storage of the inputs. The third portion will be the logical portion which will be responsible for the actual calculation of the result. The final portion will be the output portion which will take the calculated result from the FPGA and display a human-readable result on an LCD.

I. INTRODUCTION

This report covers the setup, methodology, results and conclusions of the implementation of the Simple Digital Calculator.

This project was to demonstrate the technical aspects related to constructing a functioning Digital Calculator, specifically input, calculation, and output. All these operations require careful mapping of inputs, outputs and careful organization in order for the calculator to function properly. Although in normal computational experience, the numerical computation seems a very simple process, this project demonstrates that it requires very careful mapping and state definition.

The origin of the idea for this project was learning about the functionality of the Apollo Command Module computers used in the 1960's. These computers were constructed long before Integrated Circuits and Personal Computers were commonplace, and as such seem primitive. However, an explanation found online here, demonstrated the many operations that they were able to implement were quite complex.

Implementing this Calculator demonstrates that despite the ordinary nature of calculating resources at our disposal, the simple process of calculation is a very complex one. Also, the process of data input, and data output are significantly more complex than are observed.

This project allowed us to utilize all the knowledge gained from the Labs in the class. Specific details were gleaned from the ALU creation Lab (Lab 3) as to the construction of the Arithmetic Logic Unit block. Lab 2 informed our implementation of the Multiplication feature of the Digital Calculator. Lab 4 was utilized in constructing the Register Array for storing ASCII values. Lab 5 informed our design of the Division feature of the ALU, and Lab 6 provided us with a simpler way of addressing the registers and in the construction of our Finite State Machine in the Control Circuit.

In addition to these Lab insights and the information learned in class about building Finite State Machines, there were other topics that were utilized in this project that we were required to learn. One of these was the interfacing of a PS/2 Keyboard with the Nexys4, along with the functioning of an LCD and its interface with the Nexys4.

This project could be applied to calculate mathematical values for the user under any circumstances (on the way to the moon, for instance)

II. METHODOLOGY

In order to begin designing the calculator, the first step was to break the problem down into smaller pieces that could each be tackled one at a time by different members of the group. The task was broken down into seven main components, input, input translation, register storage, control signals, arithmetic translation, arithmetic operations, and result output. In this way each component of the system could be built and tested separately, and pieced together at the end. This required defining the appropriate inputs and outputs of each block so that upon completion, the blocks would be known to function correctly and thus would function as expected upon assembly.

A. Input

The task of input was assigned to Ionatan and the solution prescribed was using an USB PS/2 keyboard for input. On the current Nexys4 board the original PS/2 input was removed in favor of the more ubiquitous USB port. Currently most keyboards function using the UART standard where keystrokes are transmitted on demand with the computer or receiver device polling constantly looking for user keystrokes. This takes place asynchronously (no clock is involved). As such, the keypad originally purchased for this task was insufficient as it specifically stated that it was non-synchronous (no clock signal). Currently many computer keyboards also function in this manner (only support the UART protocol). For this task it was critical that a keyboard that still supported the PS/2 protocol be used. Many of the keyboards made in the intermediate period contained circuitry for both the PS/2 and UART protocols.

Upon finding a keyboard that met the requirements, the next task was to retrieve the key codes. The keyboard is connected to the Nexys4 board on the USB HID input. This input contains 2 pins, PS2D, and PS2C. The PS2C signal line carries the clock signals which are used to synchronize the keyboard input. The PS2D serial signal line carries the data bits from the keyboard. The PS/2 keyboard protocol consists of 11 bits transacted. Initially when no key is being pressed, the PS/2 data signal remains high (1). Upon pressing a key, the signal goes low (0) which denotes that a key is being pressed. Then 8 bits are transmitted in succession which represents the key's key code. This unique key code identifies to the device, which key (or function for modifiers i.e. Shift) is being pressed. Upon completion of these 8 bits, a parity bit is sent, along with a stop bit which returns the PS/2 Data back to its idle state of high(1).

Utilizing the code from the book that the professor provided to us the keyboard input is captured using filters on the PS2C and PS2D input signals. In addition a mechanical debouncer is utilized to avoid the constant repeating of signals due to a key press (i.e. a key press can be registered multiple times on one press due to the way mechanical switches work). This allows a single key code to be registered. Since this signal is clocked, a Shift Register is utilized to store the 9-bit key code data (with the parity bit included). This output is then sent out of the ps2keyboard block along with a done signal indicating that a complete key code has been transmitted.

B. Input Translation

Upon receiving a complete key code from the keyboard, the next step was to translate the input into a signal that the other components in the design could understand and use. In the original design, an LCD was utilized to display user input and arithmetic output. The LCD needed as an input an 8-bit ASCII signal to display the value entered. As a result, the first step upon receiving the key code from the keyboard was to translate the key code input into an ASCII symbol.

ASCII stands for American Standard Code for Information Interchange. This is a well-established standard that represents numerical, alphabetic, and control signals in computing devices as unique 8-bit values. Utilizing this code allows the data being input to be usable to any device that understands the ASCII standard representation.

Thus upon receiving a complete 8-bit key code (by removing the parity bit from the ps2keyboard signal) from the keyboard, this block converts certain pre-defined keys(numbers, operators and control signals) into ASCII data. Along with that certain keys define an operation, or a control state for the calculator. Thus along with sending out an 8-bit ASCII signal, this Key code to ASCII decoder also sends out control data bits that control the state of the calculator [Deprecated in Final Design].

C. Registers

The next task after the decoding of the keystrokes to ASCII was how to store the values in preparation for their input to the arithmetic portion of the circuit. This task was easily accomplished with the use of 7 registers. Each register stores the value of a particular keystroke in ASCII. Since the calculator calculates at most an 8 bit value, the maximum value of any input or output will be 255 (in

decimal), and thus at most three keystroke inputs are allowed per input value (A or B). In order to simplify the process of computation, three values are required for input. As a result values must be padded with zeroes to exactly three digits.

Three of the registers are used to store the three ASCII values of A (the first input into the Arithmetic Logic Unit), and three registers are used to store the three ASCII values which represent B. In addition, one register is used to store the ASCII value assigned to the operator (+, -, *, /).

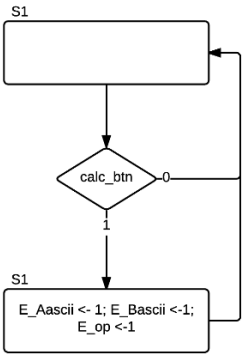
In the original design, the control circuit was responsible for addressing the proper register and enabling it, allowing the register to hold the intended ASCII value. Unfortunately, however, this caused numerous issues due to timing, and the registers were never able to be loaded properly using the control circuit. As a result an alternate method of enabling the registers was attempted which utilized the switches on the Nexys4 board as inputs. This allowed the ASCII values to be properly stored in the correct registers in preparation for calculation.

D. Control Circuit (FSM)

The critical component of the calculator is the control circuit. This component contains the Finite State Machine that activates the calculator functionality.

In the original design, the control circuit accepted the input signals from the Key code to ASCII decoder (enter_in, clr_in, op_in, digit_in) and utilized them in the finite state machine to direct user input, store keystrokes to the registers, and calculate and display the result. Unfortunately this caused some significant timing issues and errors which we were unable to correct.

As a result the control circuit was minimized to just contain the trigger for calculation. Once all inputs are accurately stored in the register, the control circuit triggers the calculation on a BTNC press. Once this occurs, the Finite State Machine converts from an idle state to a calculation state. Signals are sent downstream to ASCII to Binary and an ASCII to Operator decoder that convert the ASCII inputs into Binary inputs that the ALU can use to calculate a result.



E. Arithmetic Translation

While the numerical and operator values are inserted into the registers denoted above, the Arithmetic Logic Unit will be unable to compute the value as it does not understand what the ASCII values represent. As a result, three decoder blocks are built in and connected to the outputs of the registers.

One decoder is connected to the three registers that store the value of input A. This decoder is designed to accept 3 ASCII values and output a single 8-bit binary value representation of the value. This output (A) will be input to the Arithmetic Logic Unit for computation. This decoder also contains an enable signal that is controlled by the Finite State Machine in the Control Circuit. This is required to ensure that all 3 ASCII input values are populated in the registers before the A operand is converted to binary. If not, this would result in incomplete binary value being transmitted to the ALU. Likewise the three registers that store the value of B are connected to an ASCII to Binary decoder which functions similarly.

Finally there is a third decoder connected to the register storing the ASCII value of the operator (+, -, *, /). This decoder accepts as input the ASCII value stored in the operator register and outputs a 2-bit binary signal which the ALU will use to determine which operation to run. The value "00" represents addition, the value "01" represents subtraction, the value "10" represents multiplication, and the value "11" represents division. This decoder also contains an enable bit controlled by the Finite State Machine in the Control Circuit to avoid incomplete values being transmitted to the ALU.

F. Arithmetic Operations

The next circuit component is the Arithmetic Logic Unit. This unit is the block that does the actual computation of

results. Most of this functionality was gleaned from previous labs.

The Arithmetic Logic Unit (ALU) accepts as input three values. Two 8-bit binary values (A and B) are accepted as the operands to use for computation, and a 2-bit binary operator signal is accepted which determines the operation to be performed.

The ALU performs the computation and returns 2 8-bit binary signals. A result signal is returned which represents the result of the computation. Likewise a remainder signal is also returned which represents the remainder of a division operation. This value will be "00000000" for addition, subtraction, and multiplication. It will only contain another value in a division operation where a remainder is returned.

For addition the two 8-bit binary signals are added and the resulting 8-bit binary result value is returned. If the resulting value is greater than 8 bits, the extra bits are removed as this is an unsigned addition. Similarly for subtraction, the two 8-bit binary signals (A and B) are accepted and the operation is performed as an unsigned subtraction. For multiplication, the two operands are reduced to two 4-bit inputs by using only the least significant 4 bits. This is done since using all 8 bits would result in an overflow. By reducing the inputs to 4 bits the two operands can be at most 15 in decimal which would result in a maximum value of 255(decimal), which is the greatest value an 8-bit binary result could have.

Finally for division, the ALU accepts the two 8-bit binary operands and performs the division which results in an 8-bit binary result signal. Also for division, the remainder of the division operation is calculated and output as an 8-bit binary signal.

G. Output

The final step in the digital calculator system is to display the output of the operation.

In the original design, the output was designed to be output on the LCD display. Since the LCD only accepts ASCII input, all outputs were encoded to ASCII values.

However due to issues with character corruption and problems with timing the enabling of register input, the LCD output was removed. Instead the 7-Segment Displays on the Nexys4 board are utilized to show the resulting binary output of the ALU in hexadecimal.

The serializer code given in class is used to display the result of the arithmetic calculation. This component displays the value of the 8-bit binary result signal on 2 segments of the 7-Segment display. It does this by enabling the 2 segments and then displaying the values in quick succession several times a second which to the human eye is seem as a stable output. The values displayed are 2 hexadecimal digits, each digit representing 4 binary bits. The 8-bit binary remainder signal is also displayed on the 7-Segment displays using the same method.

III. EXPERIMENTAL SETUP

The setup used to verify the functioning of the project consisted of a Nexys4 FPGA, a USB keyboard with support for the PS/2 protocol, the Xilinx Project software and initially an LCD.

The software utilized was Xilinx ISE 14.7. This was the software utilized throughout the class to design, synthesize, implement and program the FPGA board. The configuration of the software was left in the default state.

The hardware tools utilized for the project was a USB Keyboard. The Keyboard was required to support the PS/2 protocol. Originally all keyboards had support for this protocol which has two input lines a PS2 Clock Signal, and a PS2 Data signal. The keyboard was connected the PS2C and PS2D pins on the board which are present on the USB HID (Human Interface Design) port of the Nexys4 board. Newer keyboards no longer support this, which made it imperative to find a keyboard which still supported the older protocol in order for the system to function correctly.

Another hardware tool which was originally utilized but later scrapped due to issues was an LCD screen, the HD44780. This LCD screen was originally connected to the pins of PMOD Header JA which allowed the LCD screen to be controlled by the signals set on the board. The LCD did not function as expected with the backlight flickering and the screen displaying corrupted characters despite correct ASCII input. As a result the decision was made to display the output of the calculations on the 7-Segment Displays instead.

With these components hooked up to the board, and using the switches to identify the register input, the system was able to function as predicted which is the calculation of mathematical outputs based on keyboard inputs.

IV. RESULTS

In order to verify that the system was functioning properly, we observed the results in several different ways.

Initially the LCD was used to verify the results of our experiment, however due to no results being displayed at all, this feature was removed.

The system was also tested using the LED's for feedback on the keyboard key code values being output, the register values, and the ALU result values. In order to verify that each component of the design was functioning properly, very simple modifications were performed to display the 8 bit input and output values (key codes, ASCII, and result values). By mapping these to the LEDs the system was able to be tested for proper functionality.

Testbenches were also run on all of the components to test that each component of the system functioned correctly in simulation.

Using these various methods of measuring input and output values allowed us to design and verify that the system functioned as designed, and to verify that accurate calculations could be performed.

The results that were compiled demonstrated the complexities of designing a digital calculator system. The calculator design was simplified to allow proper implementation. For instance, simply the addition of a sign would have significantly changed the circuitry and bit sizes of the inputs and outputs. In addition, both inputs and outputs are limited in size to minimize complexity. With 8 bits, unsigned binary values can be at most 255 for both input and output. This can result in false results for addition and subtraction based on the inputs provided. These are limitations that must be communicated to the user before use to avoid improper results being communicated to them.

The main complexity of the Digital Calculator was due to signal translation. Encoding and decoding keyboard, ASCII and binary values is critical to designing a proper system in which all components communicate in a way that other components can understand. Encoders and decoders were utilized for these tasks.

The use of registers was also a very important part of the design. Storing the correct values at the correct time is critical to proper system functionality. If the value is stored too soon, or too late, the value will be incorrect or corrupt, and the system will not function as designed.

Likewise, the process of calculation had to be timed appropriately using the control circuit's finite state machine, otherwise the values would either not exist or would be incorrect.

All of these results were observed and accounted for in the final design. Throughout the testing of the system, these factors had to be accounted for either by careful timing, or by setting up switches or states to ensure that values were being transmitted and stored in the registers.

The only results that were unexplainable was those associated with the LCD functionality. Despite our best efforts to display numerical values on the LCD, we were unable to display either input, or output on it, hence its removal from the design.

V. CONCLUSIONS

In conclusion, this project successfully demonstrated a working 4 function simple 8-bit calculator. This project demonstrated the complexity of the very simple operations involved in adding, subtracting, multiplying, and dividing two 8-bit numbers. The process requires very careful recording of inputs and outputs, translation of signals, and output of the results.

The main functions of the calculator function as designed however many improvements could be made. The calculator could make the task of inputting data simpler so users would not be tasked with the work required to store data in specific registers. Also the calculator functions could be expanded to include values that require greater than 8 bits to represent. Finally, the data output could be improved to allow users to see the values they input and the output could be translated into decimal values that are more understandable.

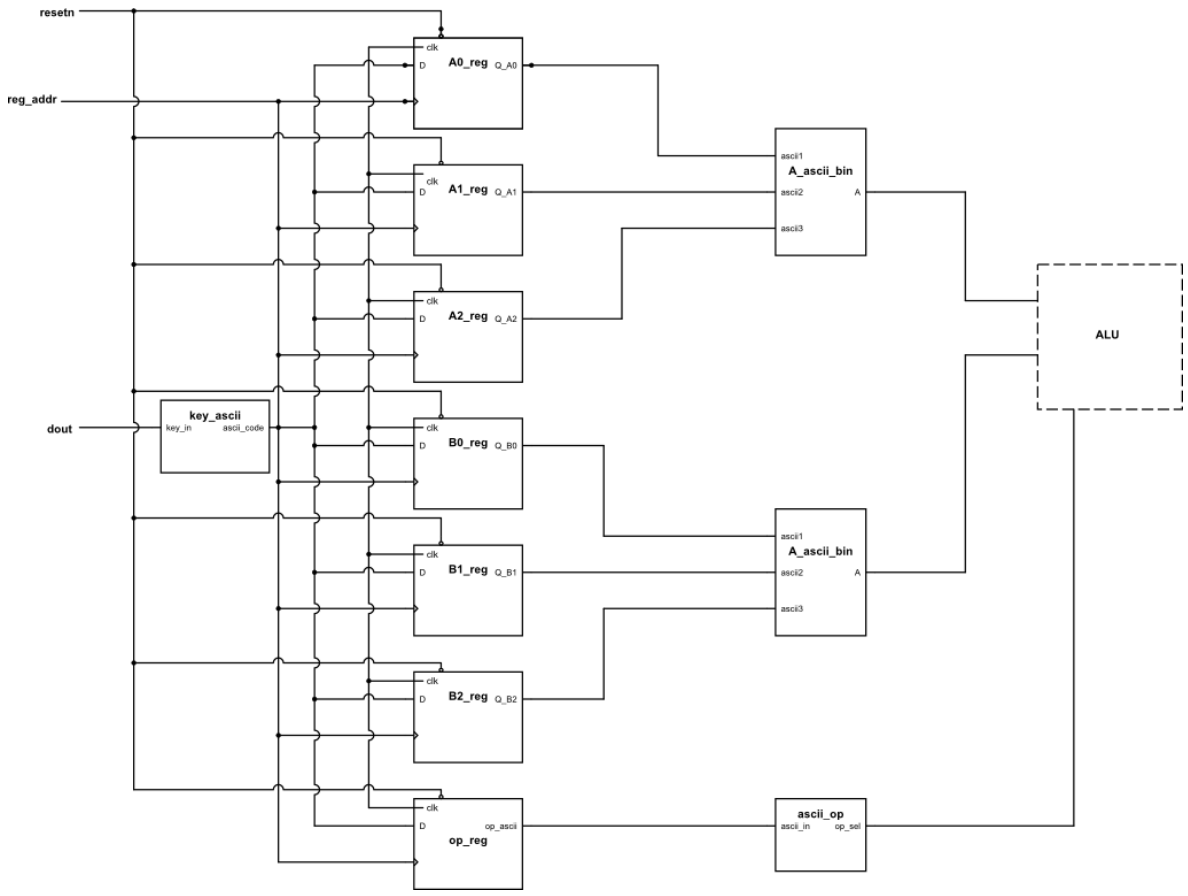
Overall however this project was very informative and interesting, informing us all as to the complexities involved in the design, creation, and implementation of a digital system.

VI. REFERENCES

[1] Haskell, Richard, and Darrin M. Hanna. Digital Design Using Digilent FPGA Boards. Rochester Hills, MI: Learning by Example, 2009. Print.

VII. DIAGRAMS

Datapath Circuit



Top-Level Design