Unsigned Calculator

List of Authors (Sivasakthi Muthukumar, Polly Jane Bates, Gerard Griest, Nikki Lipski)

Electrical and Computer Engineering Department

School of Engineering and Computer Science

Oakland University, Rochester, MI

e-mails: smuthukumar@oakland.edu, pjbates@oakland.edu, gtgriest@oakland.edu, nlipski@oakland.edu

***Abstract*—It is possible to do four functions such as addition, subtraction, multiplication, and division by hand with positive integers as large as 255, but this is very cumbersome. The goal of the project is to create a simple unsigned calculator that performs four operations with 2 8-bit unsigned inputs. Using many circuits learned from class, such as full adders, counters, and decoders, the project goal is achieved. Our simple unsigned calculator requires an input keyboard and outputs data in BCD, which will be fed to the seven-segment displays on the Nexys-A7 board.**

I. INTRODUCTION

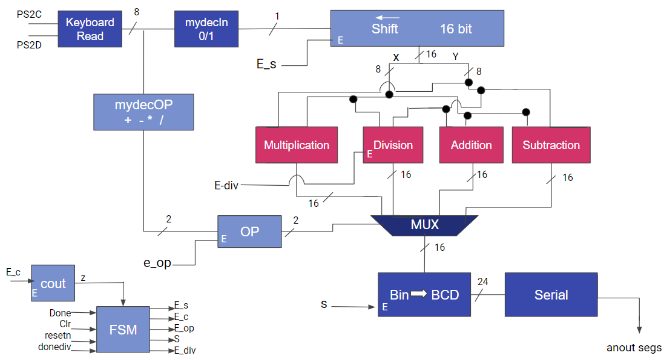
The main goal of this project is to make an unsigned calculator that will perform addition, subtraction, multiplication, and division given two unsigned 8-bit binary numbers. The numbers will be inputted through a keyboard. Calculators are essential in order to accurately solve algebraic problems. They help eliminate mistakes that often occur as a result of human error, as well as save time during calculations. The information needed in order to create the addition, subtraction, and multiplication functions on the calculator was taught in class. Contrarily, the steps needed to create the division function on the calculator needed additional research. The calculator required several components covered in class such as counters, shift registers, registers, and decoders.

II. METHODOLOGY

A. Goal

The objective of this project is to create an unsigned calculator that shows calculations on the 7-segment display of the Nexys-A7 board. The input is taken from a keyboard, and the calculated output is displayed on the FPGA board. Coding wise, this is achieved through the block diagram displayed below. The block diagram can be further broken down into individual blocks.

B. Block Diagram



C. Keyboard[1]

The keyboard serves as the controller that takes in and records the user input. When a key is pressed and held on a keyboard, the protocol is scanned, and the scan code runs once every 100ms to measure whether the key has been released by the user. Once the key is released, a key-up code is sent, as well as the recorded scan code. This component allows the system to understand what and when a key is pressed and released. Specifically, for this project, the numbers along with the operators from the keyboard's keypad were used.

D. Keyboard Decoder

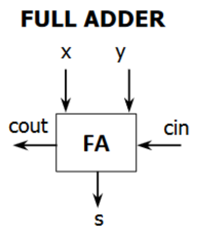
The keyboard decoder takes in the scan code from a keyboard and differentiates the input key from data ('0' or '1') and an operation (+, -, *, /) value. When the key-up value from the keyboard is a logic high, the scan code will go through one of two decoders- the data and operator decoder. The data decoder decodes the scan code as either a '1' or '0' for the 8-bit binary inputs. The default setting for this decoder is to output a '1' should the user press a wrong button. The output of this decoder is sent to a 16-bit left shift register which holds the two 8-bit binary input values. The operator decoder decodes the scan code as a two-bit value that corresponds with one of the four operators. These two bits are sent to a 2-bit register for storage. The default operation is addition if a non function key is pressed.

E. PS/2 Keyboard Controller[1]

The PS/2 keyboard input controller registers the inputs entered into the calculator. These values are read as scan code which is then processed in the keyboard decoder. The calculator can have one input entered at a time. So, to accomplish this, a finite state machine must be utilized.

F. Operation Blocks[1]

Addition is achieved using multiple full adders in series with each other. As in the diagram below the respective inputs (x and y) are inputted into each full adder and the respective output (s) are given out. X(n) and Y(n) would go in full adder (n), each one giving out a different cout which becomes full adder(n+1)'s cin. The full adder itself has a boolean equation of s = (x XOR y) XOR cin) and cout = (x AND y) OR (x AND cin) OR (y AND cin). The initial cin needs to be set to '0'.



Subtraction is much the same as addition using the same full adders in series, however, there is a slight alteration in the inputs as the y input needs to be negated and the initial cin needs to be set to '1' in order to get the proper output. This negation and setting cin to '1' is equivalent to finding the 2's complement of the input y. From there, subtracting y from x is treated the same as the addition of the 2's complement of y to x. Thus the rest of the subtraction process will mirror the addition process. If the difference between the two inputs is negative, the output on the display will be incorrect, however, if the binary equivalent of the output is considered in 2's complement the result is consistent with the negative difference.

Multiplication is carried out by an array multiplier. The product requires at most 16 bits. The array multiplier is an asynchronous circuit that takes in two 8 bit unsigned values and computes a 16 bit product. The propagation delay with this design is smaller compared to the combinational multiplier or iterative multiplier.

Division is carried out with an iterative restoring design, although the remainder is not important for the function of this calculator. This design is synchronous and requires a finite state machine, two left shift registers, a subtractor, and a register. When the divisor is 0, the output will be the maximum valued output possible. If the quotient is not an integer, the result is the floor function of the quotient. The division block also generates a "done" pulse when the final quotient has been reached. Division was not covered in class, however, Lab 6 (a square root function) follows a similar design. The division circuit was acquired thanks to Dr. Llamocca's design[3].

G. Multiplexer

The operations for addition, subtraction, multiplication, and division are inputted into the

MUX as the select lines. The MUX will determine which operation to conduct depending on the two bits generated from the operation decoder. Addition will be indicated by the "+" key on the keyboard which will correspond to select lines equal to 00, subtraction will be indicated by the "-" key and 01, multiplication will be indicated using the "*" key and 10, and division will be inputted using the "/" key and 11.

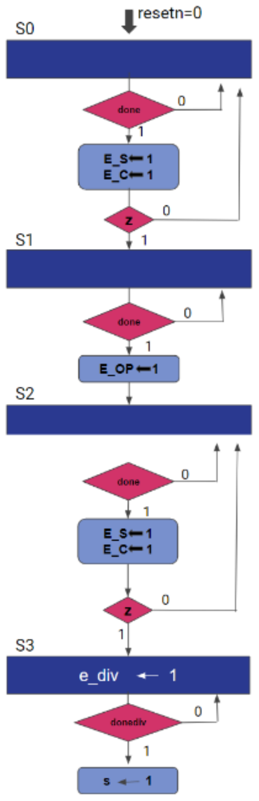
H. Binary to BCD Converter[1]

The binary to BCD converter receives the result of the computation and transforms the value to BCD. The converter loads in the value to be converted depending on the state of the FSM.

I. Counters[1]

Counters were used to track how many bits of the input had been entered. The mod 8 counter was enabled by the FSM and the signal generated when the counter reached the maximum count of 8 (z) was fed back to the FSM. The actual count of the counters was not of importance except for debugging.

J. Finite State Machine



All the major operations required the use of a Finite State Machine (FSM) in order to properly give the correct inputs to the corresponding components at the proper times. The FSM receives the key-up signal from the keyboard, a done signal from the divider, a clock signal, resetn, and the z from the mod 8 counter. The FSM outputs enable the shift register, counter, operation register, binary to BCD converter, and divider. For this calculator, four states were defined from S0 to S3.

S0 is the beginning state and is initiated when the reset button is pressed. In S0, whenever a key is pressed on the keyboard, the mod 8 counter increases its count and the input binary value is shifted into the shift register. Once 8 bits representing the first input are recorded, the state becomes S1. In S1, the operator is recorded and stored in a 2-bit register. After the operator is recorded, the circuit enters S2. S2 operates the same way as S0, receiving 8 bits of the second input and storing them in the shift register. After the second 8-bit input is typed in, S3 begins. In S3, the divider is enabled, and once the divider has finished, the binary-to-BCD converter is enabled. If the converter is enabled at the start of S3, it will capture an incorrect output value, as the divider will not have had time to reach the final answer. As opposed to most FSMs seen in class, the FSM used in this project does not loop back to an earlier state. It only resets to S0 when the resetn equals 0, which can only occur with a manual button press.

K. 7-Segment Display/Serializer[1]

The 7-segment hex display takes the BCD value and converts them to its respective seven-segment display values. This is achieved via with-select statements that pair the BCD value input with a 7-segment value output. That value will then display as the decimal value on the board. The 7-segment display on the Nexys-A7 board is inverted logically, as a "0" will result in an "on" segment, and a "1" will be an "off" segment. The display has 7 segments (a,b,c,d,e,f,g), each turning on or off corresponding to the 7-bit active low input. Additionally, to display unique digits on multiple seven-segment displays, the data must be multiplexed using a serializer. For example, to display "1234" on 4 seven-segment displays, the "1" is turned on for only one display momentarily, then the "2" is turned on and off on the next display. This continues with the "3" and "4". If the displays are turned on and off fast enough, the human eye will not notice that only one digit can only be displayed at one time; it will appear as if all the seven-segment displays are perpetually on.

III. Experimental Setup

To make this calculator, a Nexys-A7 board had to be paired with a keyboard. Each input was put into the keyboard, while the outputs appeared on the board's display. The two 8-bit input values will be displayed on the Nexys-A7 board using the 16 LEDs, where a '1' is represented by a LED that is on, and a '0' is represented by an LED that is off. The operator that is stored will be displayed using the RGB LEDs on the board. The addition operator is symbolized with the RGB LEDs off, the subtraction operator will light up green, the multiplication operator will light up dark blue, and the division operator will light up light blue. The implemented software for this calculator is Vivado, which uses the programming language VHDL. This code would be uploaded to the Nexys-A7 board in order to implement the calculator. The expected result is that if one were to input values on the keyboard along with an operation, then the correctly calculated solution would appear on the board's display.

Several versions of the project were created during development. One version served to check the math functions and multiplexer as it consisted of the four math blocks and the multiplexer. A second version tested the input process and FSM. This circuit contained the FSM, register for the operation, shift registers, and keyboard decoders. The third iteration was the final circuit without the keyboard reader. This circuit helped debug the input process, math process, and output process collectively. Test benches were developed for each of these intermediate designs to confirm their functionality.

IV. Results

As intended, the unsigned four-function calculator was able to calculate all four functions being fed into it and produce a correct and viable result on the 7-segment display. Two 8-bit unsigned numbers and the proper operation were put into the keyboard while a decimal representation of the calculated BCD values was displayed on the 7-segment displays. This was achieved via an FSM in tandem with the PS/2 keyboard controller. The four functions worked as expected with the exception of subtraction when the difference was negative. This result was expected as no steps were taken to accommodate negative numbers in this design. The most difficult part of designing the calculator stemmed from converting the final answer to BCD. The binary to BCD converter was not understood well initially and was assumed to periodically check the input value. This resulted in the output on the seven-segment displays to read all zeros. After taking a closer look at the design of the converter, the converter only captures the input once in its beginning state. Thus, the converter needed to be enabled once all the math blocks had finished their calculations. A video demonstration of the calculator can be found using this link https://youtu.be/Fa8NBjneTu0.

V. Conclusions

In conclusion, this project has been very beneficial for applying all the information that has been learned during this semester. By utilizing the material that was taught during classes and labs, in addition to supplementary research, this calculator was able to function properly. The calculator was made by using concepts such as a finite state machine, a serializer, and a counter, among other components. Physical components used for this project included the Nexys-A7 board and a keyboard. The calculator can accurately add, subtract, multiply, and divide 8-bit unsigned numbers. Further improvements to the calculator could include the ability to edit the input before calculation, adding the ability to accept larger numbers, negative numbers, and hex/decimal numbers, improving the result for a negative difference, or adding more functions. Some of these improvements would require a simple tweak to the existing design while some would require a good amount of reconstruction.

VI. References

[1] Llamoca, D. (2013). VHDL coding for fpgas. Retrieved April 12, 2023, from https://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html

[2] Master XDC file for Vivado Designs. (n.d.). Retrieved April 12, 2023, from https://moodle.oakland.edu/pluginfile.php/8345475/mod_page/content/6/Nexys-A7-100T-Master.xdc

[3] Divider Implementation. (n.d.). Retrieved April 17, 2023, from https://www.secs.oakland.edu/~llamocca/dig_library/arith/Divider%20Implementation.pdf