Simple Eight-bit Signed Calculator

NuJude Mady, Jayanti Rakshit, Sam Walker
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
nujudemady@oakland.edu, jayantirakshit@oakland.edu, swalker4@oakland.edu

***Abstract*—This project serves as a simple eight-bit signed calculator with four operations: addition, subtraction, multiplication, and division. The calculator takes its two inputs as two-digit hexadecimal numbers input via a Personal System/2 (PS/2) keyboard, as well as a switch input from the the Nexys A7-100T board, and outputs a signed four-digit hexadecimal answer on the four rightmost seven segment displays of the board. This was done using various digital logic components coded using Very High-Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) in Vivado. Upon completion, an accurate and user-friendly signed hexadecimal calculator was created.**

I. Introduction

This simple eight-bit calculator takes two two-digit signed numbers as hexadecimal inputs (A and B) from an external PS/2 keyboard. From this keyboard, a decoder, Finite State Machine (FSM), and series of four registers are used to feed user input into the various operations of the calculator. These basic arithmetic operations, including addition, subtraction, multiplication, and division, are determined via four switches on the Nexys board, linked to a four-to-one multiplexer to properly output an answer. The output from the multiplexer is then displayed on the Nexys board's four rightmost seven-segment displays using a seven segment serializer. Figure 1 shows the block diagram of this schematic.

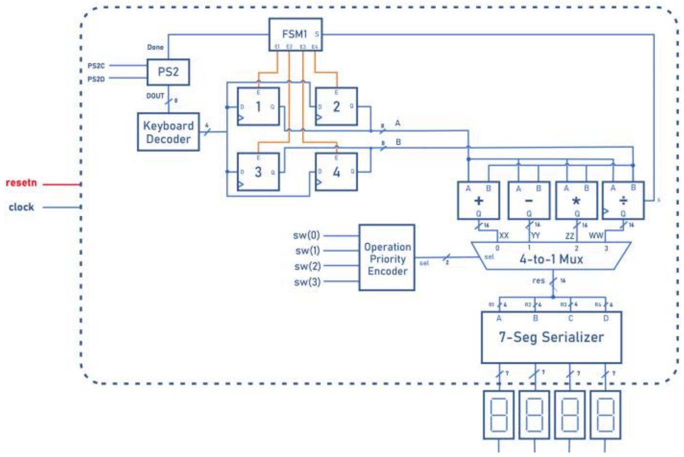

Figure 1: Simple Signed Eight-Bit Calculator Block Diagram

Calculators are vital instruments used for solving algebraic problems that would otherwise take much longer to solve by hand. Topics from class used in this project include basic logic gates, combinational and sequential circuits, FSMs, as well as number systems and computer arithmetic to confirm the calculator is giving appropriate results.

II. Methodology

A. PS/2 Keyboard Input

In designing this calculator, it was decided to use a keyboard as the user interface to obtain the A and B values to perform calculations with. To do this, it was necessary to decode the data obtained from the keyboard/Nexys board interface. The PS/2 keyboard outputs a DOUT signal, as well as a done signal. The DOUT signal is a ten-bit output, with the two most significant bits (MSBs) being the stop bit and the parity bit, which can be ignored for the purpose of this project. The rest of the signal, DOUT (7 downto 0), is the data itself, corresponding to the scan code of each key on the keyboard, as shown in Figure 2. A keyboard decoder was used to turn this eight-bit scan code hexadecimal signal into a four-bit signal, corresponding to the binary representations of a hexadecimal value between 0-F. Further, as each DOUT signal is output, the done signal is maintained for one clock cycle. This done signal goes into an FSM where it acts as the deciding state for whether the next state can be implemented.

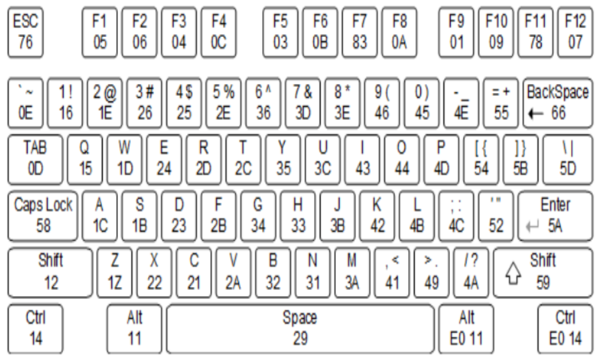

Figure 2: PS/2 Keyboard Scan Codes [1]

As shown in Figure 1, the keyboard FSM is connected to the PS/2 keyboard data input, as well as four registers. The FSM is also taking in 'clock' and 'resetn' inputs, as all synchronous circuits do, meaning everything only happens on the clock tick. These four registers each take in one of the four inputs the user inputs (two inputs for the hexadecimal value of A, two inputs for the hexadecimal value of B). The keyboard FSM was designed to act as an enable for each of the registers, and as the 's' value for the synchronous signed divider, which will be later explained. As shown in Figure 3, the FSM uses the done signal of the PS/2 keyboard as the deciding factor for moving between states. To move from one of the five states to the next, the done signal must be sent, else the machine stays in its current state. In addition, after the first state, when the done signal is '1', the enable for the first register is also '1', allowing the first decoded scan code value of the input A to be captured. The same thing happens with each of the registers. Once state three is over, identified as done being '1' and the fourth register being enabled, the machine moves on to the final state, state four, where the 's' value of the divider is output, before moving back to state zero to start over.

At the end of each FSM cycle, the registers are all full of the data corresponding to the values of A and B. Then, as the definition of registers predicts, they all release their values at the next clock cycle, where values of A and B are separately concatenated and are now ready to use in calculations.

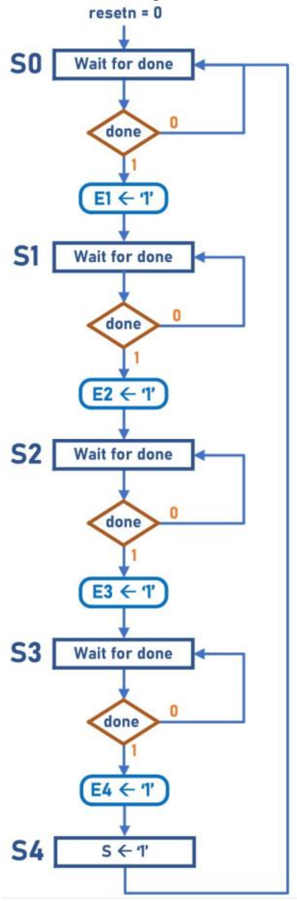

Figure 3: Keyboard FSM

B. Signed Addition and Subtraction

As previously stated, in the specifications of the calculator, the inputs are taken as signed hexadecimal numbers. This makes addition and subtraction relatively simple. For addition, the binary versions of A and B are sign extended, to account for overflow, and then simply added together, using the adder concept learned previously [2] with a carry-in value of '0'.

For subtraction, it is known that subtracting a negative is the same as adding a positive, and that subtracting a positive is the same as adding a negative. To simplify this operation, this logic was used, applying two's complement (2C) to B, and then adding this value to A. If B is originally negative, subtracting it from A is the same as adding the absolute value of B to A, so applying 2C turns the originally negative B positive so its value can be added to A. Similarly, if B is originally positive, applying 2C turns it negative, and by adding this negative version to A, it acts the same as if we subtracted the positive B from A. To implement this, A and B are, once again, sign extended, but B is also inverted. Then, these new values are input to an adder with a carry-in of '1', to act as the second step of applying 2C to B.

The result of these two operations will end up as the mathematically accurate, signed answers one would expect. The answers are also sign extended to be 16 bits, to match the answers given from multiplication, as seen next.

C. Multiplication

When multiplying two numbers, the sign of the output depends on the input. When both inputs are positive, the output is also positive. When one of the inputs is positive but the other is negative, the output is negative. Finally, when both inputs are negative, the output is positive, as the negatives cancel each other out. The magnitude of the values in each of these cases is the same, however. Using this logic, it can be understood that multiplying the positive values of A and B will give the correct magnitude of the answer, and 2C can be applied where appropriate to give the correct signed answer of the multiplication of A and B.

To carry this out, a 2C decider was implemented, to determine if A or B were negative and needed to have their absolute values taken. This was done by creating a component where the MSB of A and B were observed. If the MSB was a '0', the new value of the input was simply the sign extended version of that number. If it was determined that the MSB was a '1', the number was also sign extended, but then underwent 2C to give the positive value of this number.

After this, A and B are now both the nine-bit positive representations of the values input by the user. A 9x9 bit multiplication is carried out, using techniques implemented in the laboratory section of this course [2]. The output of this cross is still positive, even in cases where the sign of the numbers input by the user do not match. To solve this, A and B's original MSBs are observed and if appropriate, 2C is, again, applied to the answer, giving the user the correct, 16-bit, signed answer.

D. *Divison*

As with multiplication, division follows the same rules of the output being negative when the input signs do not match. However, when implementing the dividing component of the calculator, it was decided to simplify things even further and take the absolute value of both A and B regardless of if it was necessary. Then, as implemented in the laboratory section of this course, a division operation was implemented [2]. After the division of the absolute value of A and B was executed, the sign of the answer had to be decided. To carry this out, the simple sign extended version of the answer was found, as well as the 2C version of the answer. To select which was correct, the MSBs of the original inputs of A and B were concatenated and acted as a select for the multiplexer that decided which answer would be taken. If the MSB concatenation/select was 00 or 11, the simple signed extended version was output. If the select was 01 or 10, this means that one of the inputs was negative, meaning the answer should also be negative, meaning the multiplexer would output the sign extended, 2C version of the answer.

In addition, when using the divider design given in the laboratory section of this course, the inputs were, of course, A and B, but there was also an input ‘s’ necessary for the divider’s FSM, which was manually turned on/off by the user using a button. This is impractical to have the user do in a four-operation calculator, so this ‘s’ value was taken care of by the FSM in the keyboard design, as shown in Figure 3. After all values of A and B are input into the registers, ‘s’ is assigned a value of ‘1’ so the divider can operate correctly with the inputs it has. Also, as shown in the laboratory section, the divider uses a series of registers and synchronous circuits, which can only operate corresponding to the clock cycle given. This means that the answer of the divider is only accurate when the done signal is high.

E. *Answer Output*

After determining the values of all four operations using the inputs determined by the user, it is necessary to consider which switch the user flipped up, indicating which operation was intended to be executed. SW0 acts as addition, SW1 acts as subtraction, SW2 acts as multiplication, and SW3 acts as division. The calculator was designed to automatically show the output as soon as the second hexadecimal digit of B was input, so there was no need for an “equal” switch. The switches went into a priority encoder, where their values were transformed into a two-bit output. This two-bit output acts as the select for a four-to-one multiplexer. The inputs of the multiplexer consist of the four results of each of the four operations using A and B. The positions of each operation result into the multiplexer match the priority encoder designed to appropriately choose the value displayed to the user depending on their desired calculation.

The resulting 16-bit output of the multiplexer is then split into four four-bit signals to be input to a seven-segment serializer provided [2]. Within the seven-segment serializer is a decoder to transform the binary value of each four-bit signal to its hexadecimal equivalent. The serializer works to enable the four right most seven-segment displays of the Nexys board, and, finally, show the signed, hexadecimal version of the result of the proper operation.

III. Experimental Setup

This calculator was set up using a PS/2 keyboard, connected to a Nexys A7-100T board, connected to the USB port of a computer running the Vivado software. In this software, all previously mentioned components were coded in VHDL, using brand new code, as well as previous code from laboratory assignments/the course page.

The inputs were obtained via the keyboard, as well as SW0-SW3, and outputs were displayed on the board’s four rightmost seven segment displays. Both the inputs and outputs are represented in the hexadecimal number system, which is simply compressed binary, allowing for a more efficient way to calculate.

Prior to putting all components together, tests were conducted on each to ensure they were working properly and zone in on where errors were formed. For example, each operation file was individually tested, by writing a testbench and utilizing Vivado’s simulation tool. Only after confirming each was working as expected were they added into the final project. By doing this, excessive troubleshooting and error mitigation were not needed, as the source of the problem could easily be identified by examining each signal within the component.

After confirming these operation blocks were working, a separate “test” project was made to ensure they were port-mapped together correctly. This project included all parts besides the keyboard components and the seven-segment serializer components, meaning one had to manually enter the DOUT values for each test case, as well as the done signals that would be otherwise provided by the keyboard. The seven-segment serializer was also not included, so that it could be confirmed that the results were coming through each component correctly. After everything was confirmed to be working correctly, a final project was made, including every single source of every single component, as well as a constraints file to allow for the board to correctly assign each physical component to its corresponding virtual component.

The expected results are that the user will enter the two-digit hexadecimal value of A, then flip the corresponding switch of the operation they would like to carry out, then input the two-digit hexadecimal value of B, and immediately be provided with the mathematically correct, signed, four-digit hexadecimal value on the board’s four rightmost seven-segment displays. After each calculation is carried out, the user is expected to reset/clear the calculator via the CPU reset button on the board.

IV. Results

As previously explained in the experimental setup, testing was conducted in phases. By simulating the intermediate project, without including the keyboard/board interface, results were correct, as shown in Figures 4-7. Upon receiving these results, it was essentially without a doubt that the final, complete project would give the expected results.

The results obtained by this project were as they were expected to be. Upon two signed two-digit hexadecimal inputs, and the corresponding switch being flipped up, the calculator does give the mathematically correct signed hexadecimal outputs displayed on the seven-segment displays. A video of the functioning calculator can be seen here: https://www.youtube.com/watch?v=lcjVxKpi5EU

Digital logic design was the topic of this course, and by designing a system that achieves this, much was learned in the process. The implementation of synchronous circuits was explored even further, as they were only touched on in the last two laboratory assignments. The idea of using signed inputs was also dived into, seeing that the laboratory assignments pertaining to calculations were unsigned and various steps had to be taken to convert these base operations from unsigned to signed, as explained previously. Further, the PS/2 keyboard/Nexys A7-100T board interface and seven-segment serializer had to be understood before implementing them. Previously, the switches on the board and the occasional button on the board were used and outputs were displayed on a single seven-segment display. In this project, inputs were taken using a combination of the switches on the board and an external USB keyboard, and outputs were shown on multiple displays. Upon completion, these concepts became much clearer.

Conclusions

At the end of this project, concepts regarding digital logic design are exponentially more coherent. This project allowed for all concepts learned throughout the semester to come together nicely, from basic binary addition to applying 2C to simple asynchronous circuits to more complex synchronous circuits and FSMs to designing a digital system using various components and implementing them on a Field Programmable Gate Array (FPGA).

Troubleshooting and finding errors within the project were also topics that were crucial to its completion. By being able to single out errors and discrepancies within each component, each part came together much more seamlessly in the end. The way this was done was through writing testbenches and performing simulations in Vivado and ensuring each component was working as expected. When results were unexpected, various signals from within the components were analyzed to pinpoint the exact root of the problem.

References

[1] Brown, A. (2019, July 10). *Nexys A7 Reference Manual*. Nexys A7 Reference Manual - Digilent Reference. https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual

[2] Llamocca, D. VHDL coding for FPGAs. https://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html

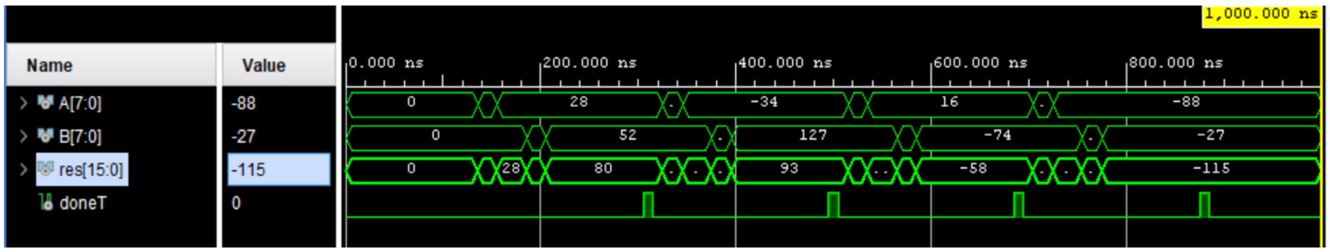

Figure 4: Addition Simulation Results

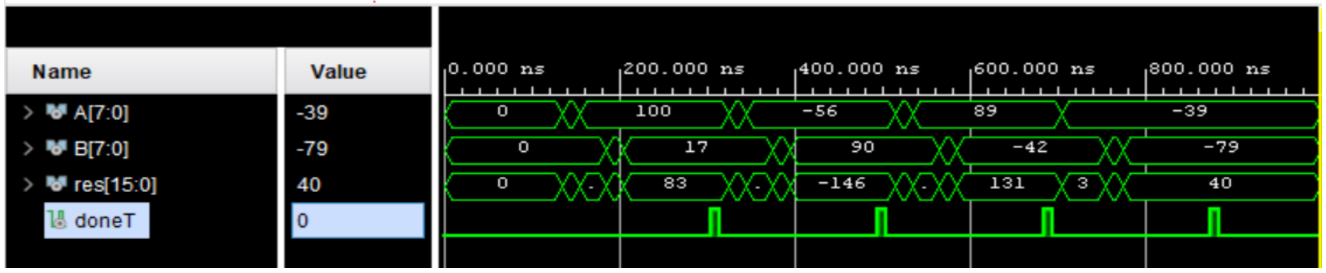

Figure 5: Subtraction Simulation Results

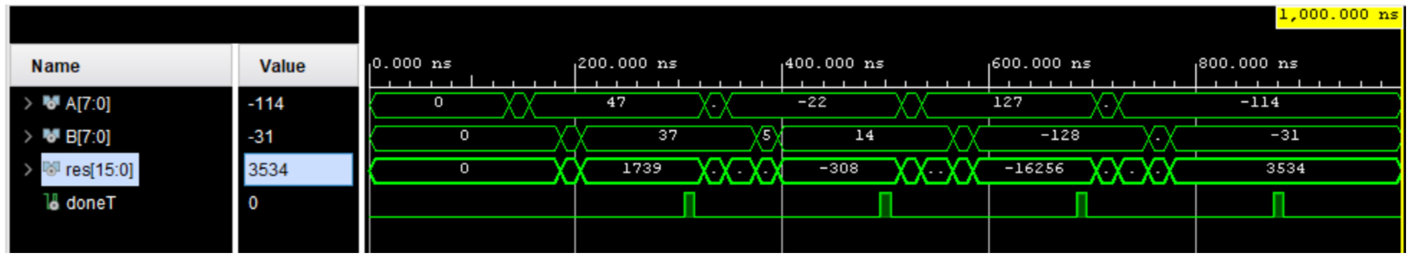

Figure 6: Multiplication Simulation Results

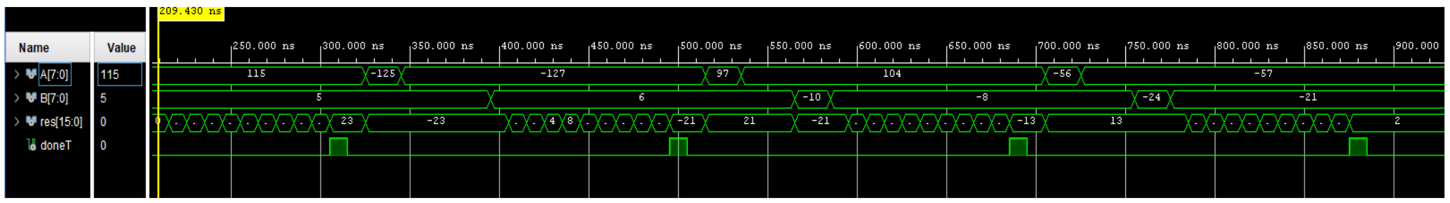

Figure 7: Division Simulation Results