8-bit Simple Signed Calculator: Keyboard and Display

Authors: Rita Brikho, Ramsin Yaqo, Ansam Ghareeb,
and Randa Matti

Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI

E-mails: Rzbrikho@oakland.edu, Ramsinyaqo@oakland.edu, Ansamghareeb@oakland.edu, Randamatti@oakland.edu

Abstract

This project is aimed to provide an 8-bit simple calculator whose input will be taken from a keyboard and the result will be displayed on seven segments of FPGA. This simple calculator is capable of addition, subtraction, multiplication, and many other functions. This calculator is close to real life application as the user can easily input the values using the standard computer keyboard.

Introduction

Our goal for this is to make a simple 8-bit calculator which is similar to a real-world calculator. The calculator does the following operations: addition, subtraction, multiplication, division, exponent and absolute. Two 2-digit inputs are taken from the keyboard which will be used to operate on. After the input is taken from the
user, an operation code is given which determines which operation is to be done. After the operation is done the result is taken through a serializer which then displays the output at the 8 7-segment displays on the FPGA named Nexys-A7. This whole thing is achieved using different paths. For example, the data path contains the operations, mux (determines which operations is done) and the serializer
This project has the potential to be used in a variety of real-life situations. By adding a function for scientific notation and providing the output in decimal format, this calculator could be converted to a simple scientific calculator. Another use is to solve elementary mathematics problems, such as those encountered in teaching or accountancy.

Methodology

The goal for this project is to visualize the calculator on the Nexys-A7 board. In order to do that, the essential task of designing the code needs to be completed. The whole project is simplified into a block diagram which is explained in Figure 1

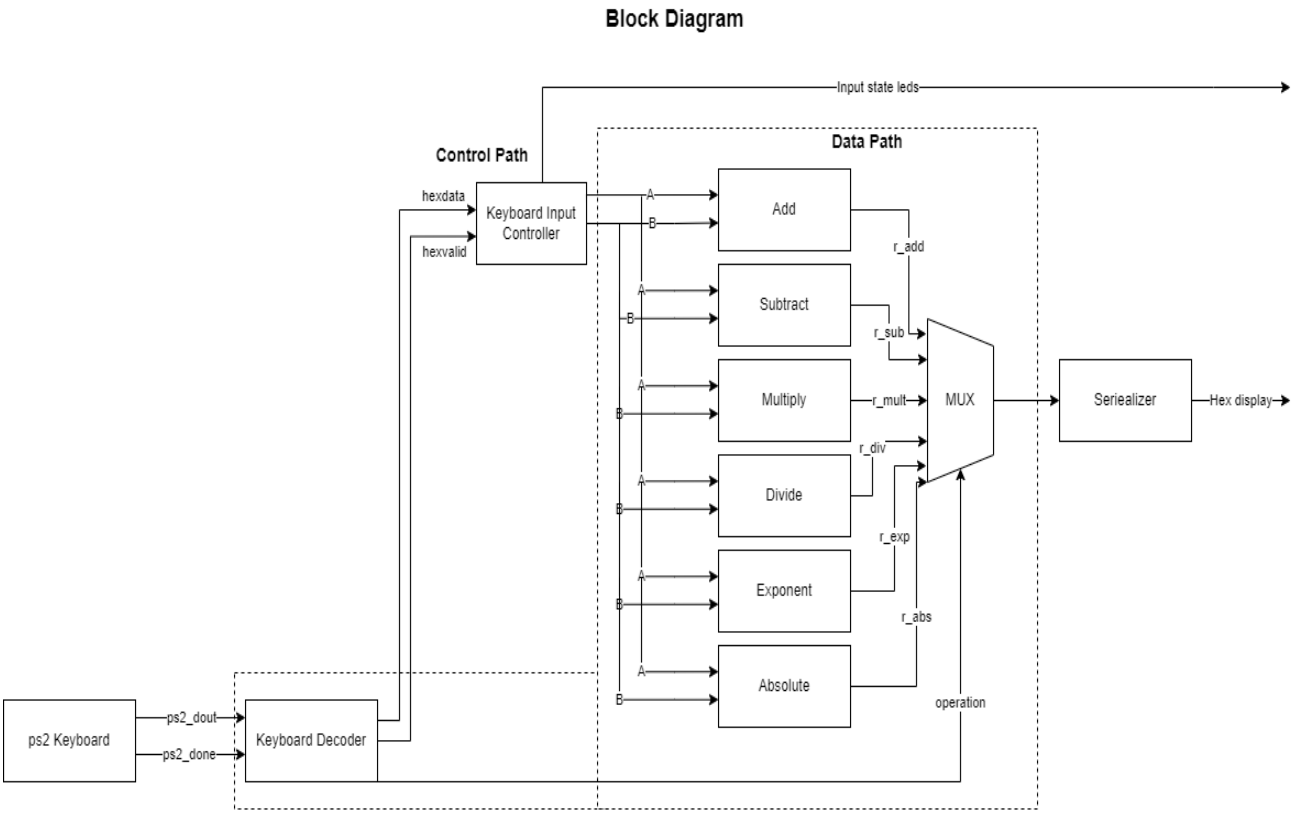

Figure 1: Block diagram of 8-bit calculator

Each block diagram has its purpose which computes the input to and outputs the respective result.

- Ps2 Keyboard

  The ps2 block is the main controller for keyboard protocol. It works based on two lanes, ps2_c(clock) and ps2_data. When a button is pressed on the keyboard the protocol starts.

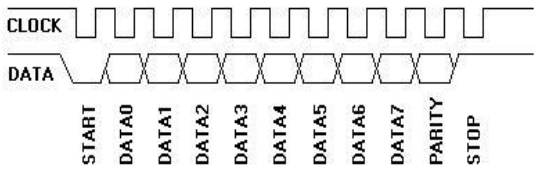

Figure 2: Communication between the host and the device.

It has a starting bit, parity bit, ending bit and the 8-bit data. The starting and ending bits indicate the start and end of the protocol respectively. The parity indicated whether transmission is ok or not. When the button is pressed on the keyboard (excluding special keys) the protocol scans the keyboard. The protocol waits for 100ms and then rescans whether the key is released or not. If the key is released f0 code is sent out. The module determines when f0 is received and the button is released. The module is implemented with the 2 lanes to determine and to extract the data from the keyboard. The module then sends out two outputs, one is ps2_dout which is the 8-bit data and ps2_done.
Ps2_done displays whether the data is valid or not (high for validity) This module is given by the lab.

- Keyboard Decoder

  The decoder waits for the output from the ps2 keyboard. It has a series of conditions to determine the input whether it is data or operation. If Ps2_done from the keyboard comes as low (0) then the decoder starts over again till the ps2_done is one.

If the condition is met, the input is run through several combinations as shown in Figure 3 stored to check whether the input is data or operations.

- Note:
- Valid Inputs: 0-9, A-F
- Valid Operations: F1 - F6

- Operations:
- F1 => Add
- F2 => Subtract
- F3 => Multiply
- F4 => Divide
- F5 => Absolute
- F6 => Exponent

Figure 3: Combinations stored in the keyboard decoder

For example, if the input is F7 or G which is not in the stored combinations the decoder will run the two conditions simultaneously and will show the data as invalid. If the input matches the data stored in the input combinations, then
It will assign Hex Value = '1' or if it matches with the operation combination then it will assign it to operation.

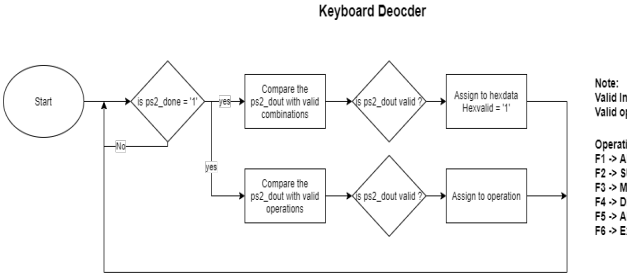

Figure 4: Block diagram of the keyboard decoder

- Keyboard Input Controller

This block is a finite state machine. Whenever the code runs it goes into a state of loops till the hex value ='1' is achieved. This is added because we can only enter one key at a time. To get multiple hex digits we use fsm . At the start fsm is at state 0 till we get hex value ='1' then goes to state 1 to get the seconds data. Since we have 2 inputs, we take them as A and B respectively. We assigned state 0 and 1 to A and state 2 and 3 to B to the data. Figure 5 shows how the fsm goes through the states.

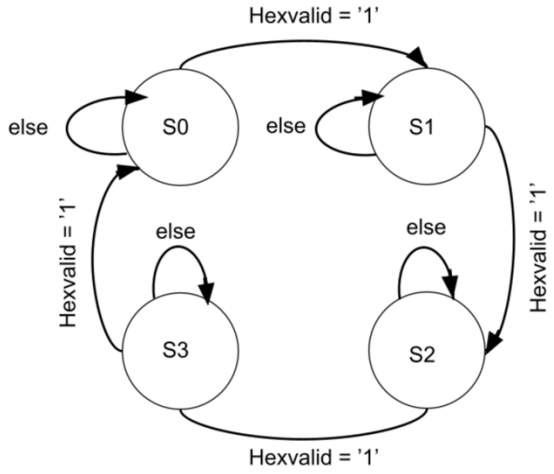

Figure 5: Block diagram of the keyboard Input Controller (FSM)

S0 → Gets the first digit of first input from keyboard → A(3 - 0) = Hex data
S1 → Gets the second digit of first input from keyboard → A(7 - 4) = Hex data
S2 → Gets the first digit of second input from keyboard → B(3 - 0) = Hex data
S3 → Gets the second digit of second input from keyboard → B(7 - 4) = Hex data

- Operation Blocks

The operation block contains the six operations (addition, subtraction, multiplication, division, exponent and absolute). Initially the data is in VHDL and since we are using numeric_std we cannot perform numeric operations on the input. For this we convert VHDL into unsigned using the function, then to integer. After the inputs are converted into integers the respective operations are done and the result is then again converted back to STD_logic_vector. It is important to note that the inputs are 8 bits each and the output STD_logic_vector is 32 bits.

- Addition

To implement the addition operation the above operation is used to convert VHDL to integer, perform the addition operations and back to STD_logic_vector which is in 32 bits.

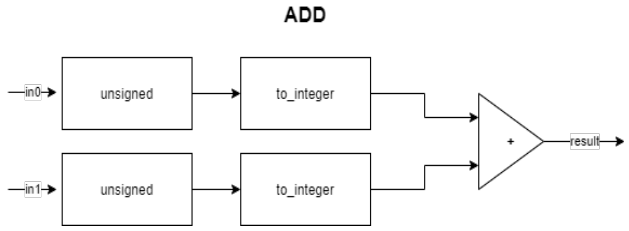

Figure 6: Block diagram of Addition

- Subtract

To implement the subtraction operation the above operation is used to convert VHDL to integer, perform the operation and back to STD_logic_vector

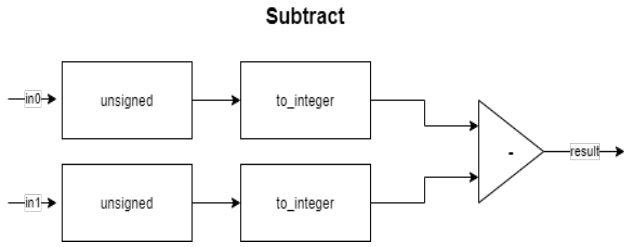

Figure 7: Block diagram of Subtraction

- Multiply

To implement the multiplication operation the above operation is used to convert VHDL to integer, perform operation and back to STD_logic_vector

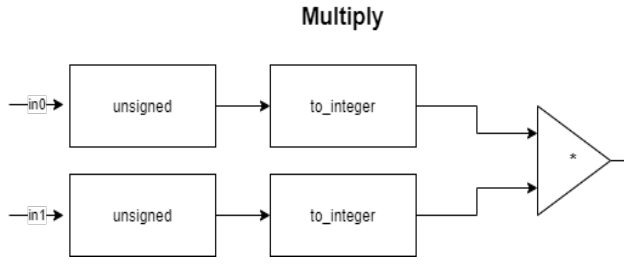

Figure 8: Block diagram of Multiplication

- Divide

To implement the division operation, the same data type conversion function is utilized. Input divided by 0 cannot be done so we generate an error. But the division operation is not as same as the other operations due to it giving an error in the simulation "divided by 0 is not possible ". Thus, we use a clock and reset switch meaning it's a sequential logic. Simple combination can be used but it gives the error. Changing the input into sequential and clock dependent removes the error in the simulation (/0 which is the unknown state).

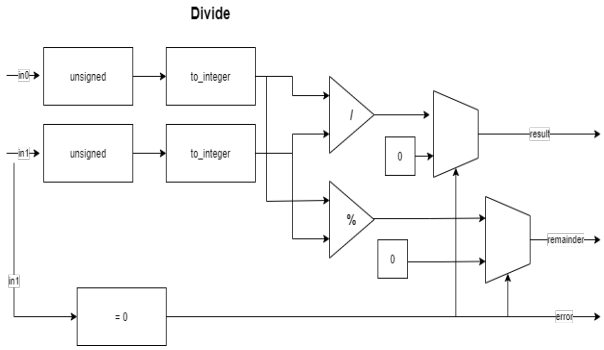

Figure 9: Block diagram of Division

- Exponent

To implement the exponential operation the above operation is used to convert VHDL to integer, perform the operation and back to STD_logic_vector. This logic is valid for simulation only.
Input 1 is set to k structure and is given a variety of options which are selected by the operator. The operator decides which of the constants is to be sent. "**" cannot be used for variables thus cannot synthesize the design. To synthesize the operation, it is a must to give a constant which is why we used k structures.

There are two limitations to this operation. 1) If the output exceeds 32 bits we generate "0" at the output since we haven't implemented an error. 2) The max power we can use is 31.

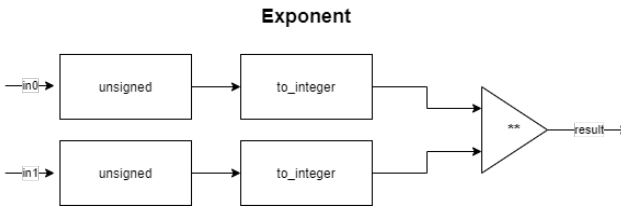

Figure 10: Block diagram of Exponent

- Absolute

To implement the subtraction operation the above operation is used to convert VHDL to integer, perform the operation and back to STD_logic_vector

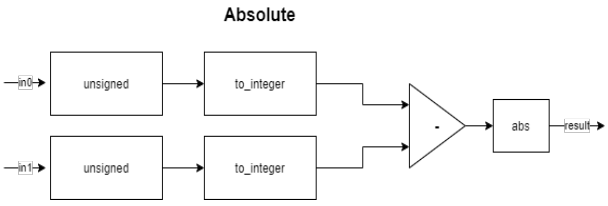

Figure 11: Block diagram of Absolute

- Mux

All the six operation blocks are connected to the mux. The mux decides which operations are to be determined using the operation code given. For example, code = "0" we will use add and if code =" 5" we will use exponential operations.

We have 4 operations bits for the code, but originally, we were using 3 bits. The extra bit is reserved for the future if we want to add more operations. Since we have 6 operations, we are using 3 bits for the code. Figure 12 shows the selection of codes that the mux uses to determine the operation.

Addition	0
Subtraction	1
Multiplication	2
Division	3
Absolute	4
Exponential	5

Figure 12: Codes for each operation

- Serializer

Serializer is a 7-segment display which displays the output at the end of the code. The code for the serializer was already given however it was valid for a 4-digit display. Some modifications are added which display the output for 8 digits.

Simulations

The simulations for this project are done on Vivado which is a chip development and high-level synthesis program.

We used this program to verify our model. We first simulate the ps2 input. Using the keyboard, we first input 5, which will be the 1st input unit digit. When it is entered it will go into state 1. Then we input 0 which will be the 1st input tens digit. This will complete the first input. and the FSM will be at state 2 Then we will input 6 which will be the 2nd input unit digit. then we will input 0 which will complete the second input. Now we will input f1 which will indicate the subtraction operation.

After inputting our 2 inputs A and B and using the subtractions operator we get the following result in figure 13.

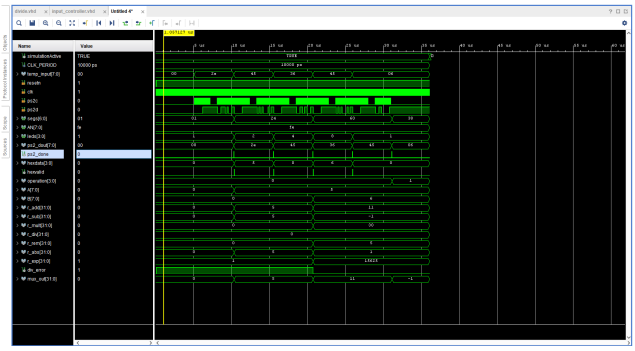

Figure 13: Simulation

Experimental Setup

The solution was devised and was designed successfully in the implementation phase. The keyboard is used to input data and all relevant information for the task. That includes two 8-bit numbers and the 3bit select line for choosing the desired operation of the ALU. Select lines are the input at the MUX which selects which input will be given on the output as shown in the simulations in Figure 13. The data was processed by the operators

as their instances were called in the top model and data of all the operations were available at the input side. The select line would decide which function is selected and that data will go on to the output. The seven-segment display will then display the result on the boards.

Results

The 8-bit calculator performed admirably. The user would enter two 8-bit values, 1 and 2, as inputs. The Keyboard input controller (FSM) would be used to accomplish this. The user must then utilize the select line attached to the mux to select the desired math operation. '0' represents addition, '1' represents subtraction, '2' represents multiplication, '3' represents division, '4' represents absolute, and '5' represents exponential. If the user chooses something else, an error will be sent to the output. The result would be displayed on the seven-segment display in hexa-decimal.

Visit this link as it demonstrates the project: https://www.youtube.com/watch?v=GUyvZex0MJ0

Conclusions

The assigned assignment was completed successfully on the FPGA Nexys-A7 board. On the boards, the output was displayed on a seven-segment display. The keyboard input controller controls the data input and selects the processes to be performed. Hands-on learning is used to implement the task, which includes the use of a display, a board, and codding. This can aid in the execution of the many duties on the board.

Despite making a calculator that is unique in many ways, there are plenty of improvements that can be done. One of those could be to add support for negative exponents or taking roots, as that will make the calculator more advanced. Another suggestion is adding the ability to use constants like e and π.

References

[1] *VHDL coding for fpgas*. [Online]. Available: http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html.]

[2] A. Chapweske, “PS/2 Mouse/Keyboard Protocol,” *The PS/2 mouse/keyboard protocol*. [Online]. Available: http://www.burtonsys.com/ps2_chapweske.htm.

[3] A. Brown, “Nexys A7 Reference Manual,” Nexys A7 Reference Manual - Digilent Reference. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual.

[4] A. Brown, “Nexys A7 Reference Manual,” *Nexys A7 Reference Manual - Digilent Reference*. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual.