FPGA calculator for 2 digits basic calculation

Nana Ogawa
University of Utah

Erdembeileg Ariunbold
University of Utah

I. Introduction

Our project is a calculator that takes input from 4*4 keypad and that displays the input, operation, and the output on the FPGA board's 7 segments LED lights.

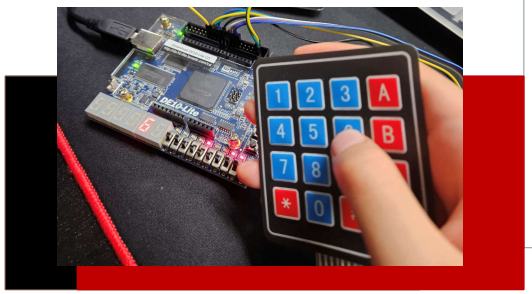
Our project had three phases for implementation.
A) key detection – a detecting what the input is from keypad
B) calculation — the actual calculation part based on input
C) display — display of operands, operator, and the results on the 7-segment LED

We used Verilog code to accomplish all of these phases.

One limitation for our calculator is that it is bounded for 2-digit operands and result. The display of operations were also difficult due to the trait of 7-segment LED trait, so we used letters to represent operations. The operations are also limited to addition, subtraction, multiplication, and division.

II. Design

Here are key designs that we adopted
1, Clock Enable Generator
Since FPGA runs the system by fast clock, we used clock enable generator to pace down the cycle, which is effective for both scanning and input rebouncing.
2, Column Scanning
We scanned the column while receiving input from keypad for the row, which is a common design
3, Two digits limit for operands and result
We bounded for two digits due to the simplicity. If the user press number after two digit is filled, both digits will be cleared.
4, Error Handling for division by 0
When the user divides first operand by 0, we display "EE" on the first and second LED to notify the error.
5, 7 segment display for the result
While limiting operations display, it enabled us to display directly on FPGA itself
6, Key-pads representations
A for addition, B for subtraction, C for multiplication, and D for division. * for clear and # for =



III. Implementation

Here are implementations on each phase.
A) Key Detection
for every "positive edge" of clock enable pulse, each column is scanned. When a key is pressed, on the column that is scanned (i.e. in active low), It pulls down the row of the key to active low as well. As a decoder, the column and row information is then converted to the 4 bits number from 0 to E.
We also implemented the debouncing here. We made a debouncing timer that simply increments each iteration until it gets certain number. When it gets to that number, it activates the condition for the key to be detected.
A) Calculation
We adjust digits and values, and use built-in operations to calculate the result when operand is detected.
We made a variable for rising edge of the key press, so when key is detected, it will be active. We made sure that it is only activated at the very first moment when the key is pressed, by adding condition of the previous key pressed state to be 0, and renewing previous key pressed for every system clock.
When that rising edge is detected, it separates cases based key value. When the key value is number from 0 to 9, it renews operand values, value for display, and keeps track of the digit count. When it is the operation, it activates the calculation based on the operation pressed, and renew variables to prepare for the second operand. When = is pressed, it activates the result display and ending condition for the calculation. The last case is the reset button, which simply initialize all variables.
A) Display
We displayed the values using the 7-segment decoder that is commonly used. The input display was handled along with the value change, to change LED position as digit increases. Division by 0 is the edge case for display of EE, and the 8 bit 11111111 was used for display off.

Key Labels vs Key Value

Key label

1	2	3	A
4	5	6	B
7	8	9	C
E	0	F	D

Key value

1	2	3	×
4	5	6	−
7	8	9	+
*	0	=	÷

IV. Results

Let's see if our design is properly executed
1, debouncing of the input
The calculator did in fact executed the debouncing, which prevented the multiple input of same key at only one press
2, 7 segment display
It displayed the right input, operation, and result on the right positions.
3, calculation/scanning
The calculator showed correct result for all addition, subtraction, multiplication, and division, which suggests the correct implementation of both calculation and scanning
4, reset
It properly reset the values and display when reset is pressed
5, error handling
It properly displayed EE when division by 0 is occured

THE UNIVERSITY OF UTAH

This work was supported by the University of Utah Office of Undergraduate Research.

Office of UNDERGRADUATE RESEARCH THE UNIVERSITY OF UTAH