Autumn 2019

EL 203 : Embedded Hardware Design

Project Report : Q-10 (Group - 3)

Four Function Calculator

Assigned by : Prof. Biswajit Mishra

Date : November 19th, 2019

- **Problem Statement**

To make a 4-function hand-held decimal signed calculator which can add, subtract, multiply and divide, with an 16x2 LCD screen and an input keypad containing 16 buttons representing 0 to 9 numbers, +, -, *, / and = respectively. The answer should be displayed in the following decimal manner: $\pm$**xxxxx.xx** (x represents any unsigned decimal digit). The operation should be as following manner: operand_operator_operand and the result is to be displayed on the screen.

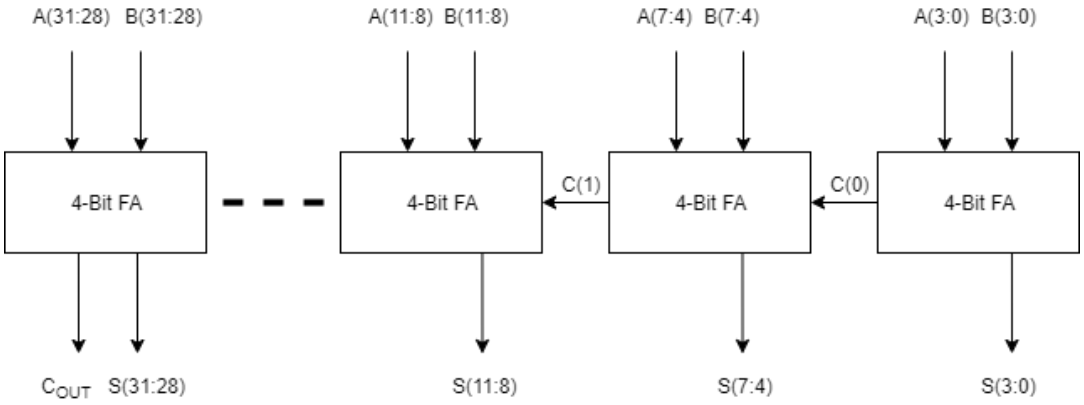
- **Design and Implementation (Software- VHDL Codes)**

As it is a fixed point representation, we reserved the first digit for the sign of the number entered by the user, the remaining 7 digits as five integer digits and 2 fractional digits (precision of 2). That's why the representation is in $\pm$xxxxx.xx format. An important assumption in the terms of input that's made is that when the user enters anything, the first bit would be sign bit. If he starts off with just numbers, there won't be any output. This just simplifies the process. For example, suppose user enters 9 X -6. So first, the user has to start off with + and then enter 9. Then he enters operator and that's X. Finally, he enters -6, so first bit would be sign bit which is -. This helps in calling the specific ports easily for our different functionalities. We represent the negative operands as 32-bit 2's complement binary numbers. We also know how many digits before and after the decimal point are entered by the user while entering the number on the keypad. The user enters an 8 digit decimal number but every calculation is done in the binary form. So for that, we convert the decimal number into BCD and then convert the BCD number into binary number. After that, the calculation is done. For each conversion and calculation, different modules are used. The main four functions are explained below:

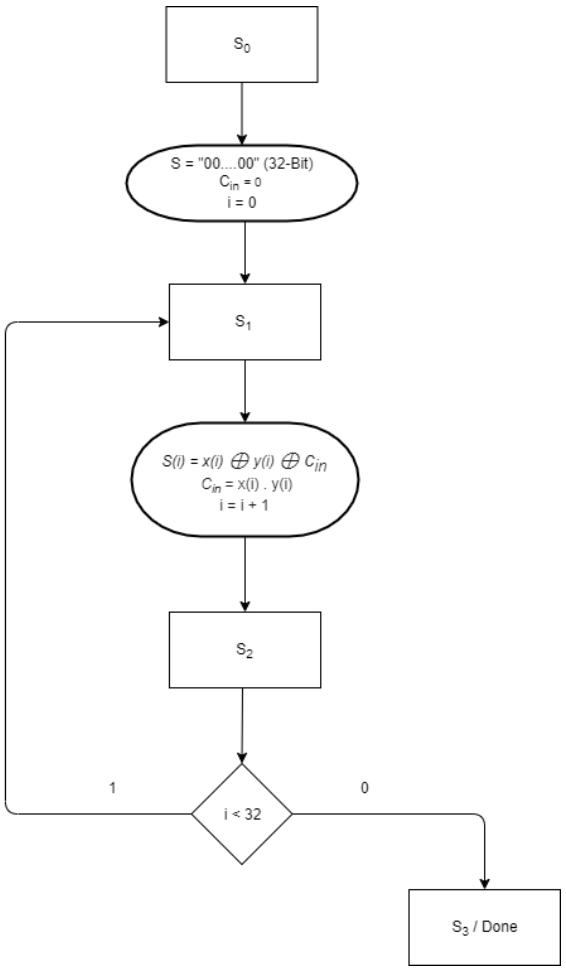
1. **Addition:**

We need to get the binary sum of A+B. Now, we know the signs of A and B therefore we definitely know which module to call, addition or subtraction. If both the signs, signA and signB are the same and the operator is +, then we will call the adder module and our final answer will be having the sign as same as the sign of A or B. So, in this module, we only add the numbers and not subtract them. This module returns the 32-bit binary result and the carry-out. We have used the hierarchical structure to implement the 32-bit adder. First, we made a module for 1 bit full adder, then we made a module for 4-bit full adder using 1-bit full adder and finally made a module for 32-bit full adder using 4-bit full adder.

- **Block Diagram:**

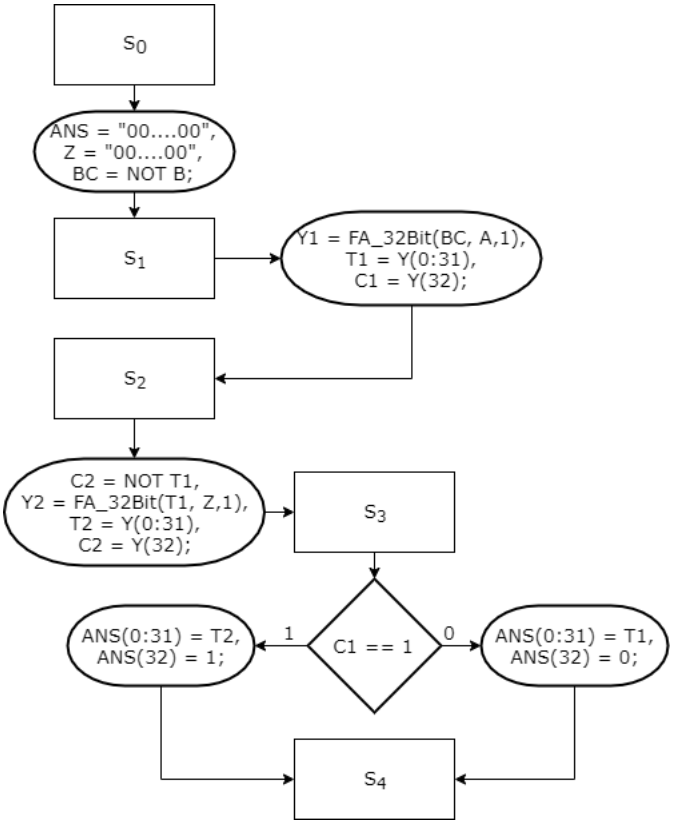


- **SM Chart:**

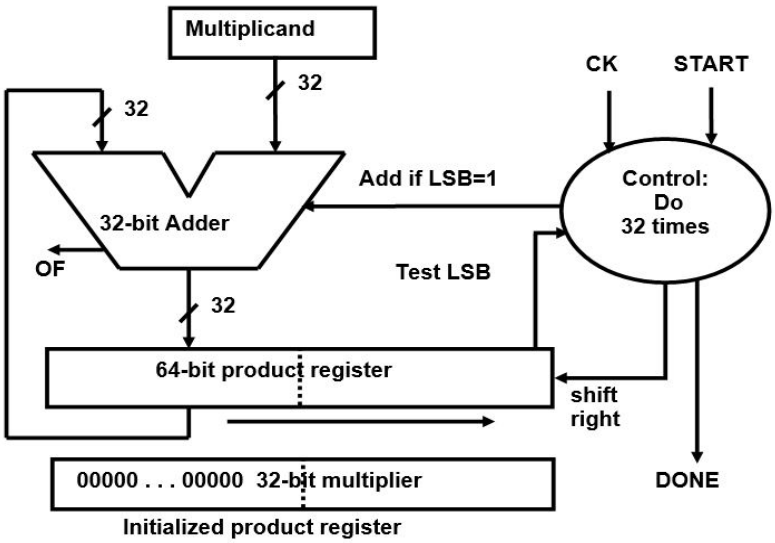


2. **Subtraction:**
   In this module, we will enter two 32-bit numbers A and B and will get a 32-bit result. For the calculations, we use 2's complement method. Using the adder module, we do $A + \overline{B}$ where $\overline{B}$ is the 2's complement of B. If A<B then one addition is that we need to do the 2's complement of the final answer too.

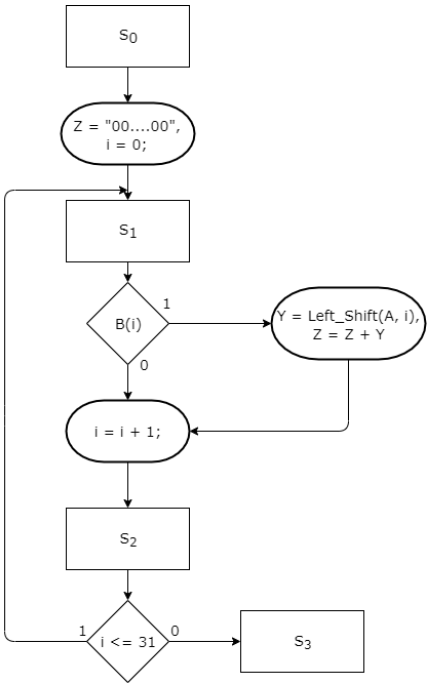
   - **SM Chart:**



3. **Multiplication:**
   When we pass the two numbers A and B, we will be having their signs too. If both of the numbers are positive or both negative, the result would be positive; or else, it'd be negative. The block diagram for the 32-bit multiplier is as shown below:

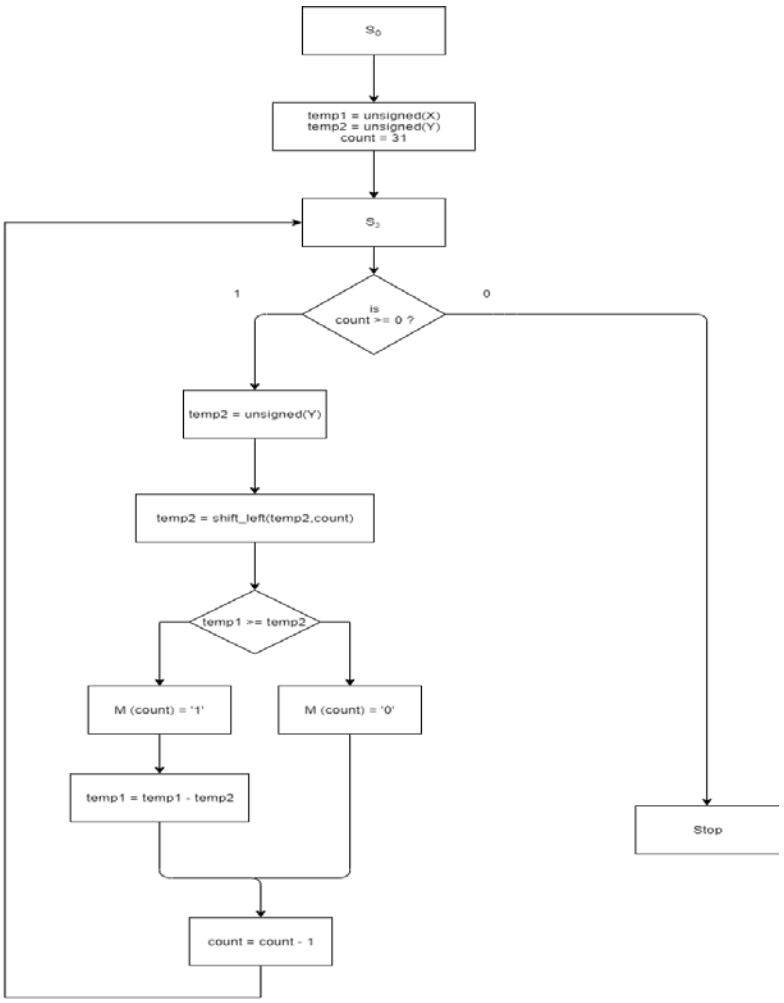


- **SM Chart:**



4. **Division:**
   The logic remains the same as the addition and multiplication modules. We just need to calculate the quotient that we get by dividing the numbers. To divide the numbers, we will use the method of shifting and subtraction.

   - **SM Chart**:

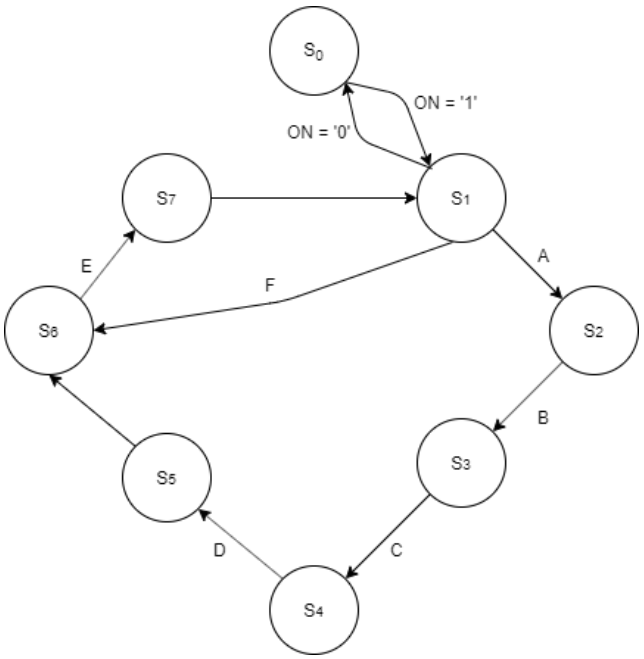


- **Problems:**

  There were many coding problems that we encountered during the making of the codes. We had to make different modules for each operations. However, we couldn't port map every modules so we had to make conditions which made the code lengthier. Moreover, making division module isn't easy to make. For much higher values, the data goes out of bound and therefore no output is shown in the simulator. We faced many problems like this. But in the terms of learning, we learnt a great deal.

The final product consists of an input keypad containing 16 buttons representing 0 to 9 numbers, +, -, *, / and = respectively and a 16x2 LCD screen. When the user enters a number (including sign), the LCD display clears when an operator is entered. The output would be loaded onto a register that sends its value in real time. The display would then show the final value on the display, and the cycle repeats when a new key is pressed. The Final state diagram and SM chart is shown here.

- **State Diagram:**



Where,
S0 = OFF, S1 = IDLE, S2 = BCD2BIN, S3 = LOAD MEMORY, S4 = CALCULATE, S5 = BIN2BCD, S6 = BCD2ASCII, S7 = DISPLAY

A = bcd_in <= 9, B = send = '1', C = +,-,*,/ Operaters,
D = Output Value Sent To Binary2BCD Converter,
E = When Output Is Ready, The Display Will Replace The Inputs With The Output Value Until A New Input Is Registered.
F = Module Sends Keypad Signals To The BCD2ASCII Converter For The LCD, And Also Sends Them To Be converter To a Single Binary Value.

- **Team Details and Contribution:**
  - Vaidik Patel (201701195) - 25%
  - Pavan Baldaniya (201701196) - 25%
  - Darshan Prajapati (201701197) - 20%
  - Dharmin Solanki (201701198) - 30%