FPGA Project Report

Clock Divider and Debouncer

When a mechanical input changes state, the signal will fluctuate for a short amount of time. The debouncer is used to ensure that only a single signal received from the mechanical input (i.e. the keypad buttons) will be acted upon.
First, a clock divider function is implemented, which takes the system clock (running at a frequency of 50 MHz) as an input, and gives an output signal at a frequency of 500 Hz.
This 500 Hz signal is one of the inputs to the debouncer function, along with the physical button press and an enable bit. Shift registers are implemented, which store the button press value. If the value of button press is the same for four counts of the divided clock signal (8 ms in total), then the output register is set to the button press value.

Sign-Magnitude to Two's Complement

The SignMag_or_TwosComp function switches the form of a 15-bit input from sign-magnitude to two's complement, and vice versa.
First, it is checked whether the input is zero. If this is the case, the output is set to zero - this prevents the creation of a negative zero.
If the sign bit of the input is 1, all the bits are inverted, the result is incremented by 1, and the MSB of the output is set to 1.

Arithmetic Logic Unit

The ALU takes two numbers (between -99 and 99) as 8-bit inputs, with a 4-bit operand input. These 8-bit inputs are stored in 16-bit registers, where the 8th (sign) bit of the inputs are transferred to the MSB of the 16-bit registers. The two 16-bit numbers are converted from sign-magnitude form into 16-bit two's complement form, using the SignMag_or_TwosComp function.
For the addition and subtraction operations, the two 16-bit numbers are added together or subtracted from each other, and the result of the operation is stored in a 16-bit register in two's complement form. This 16-bit result is converted back into sign-magnitude form using the SignMag_or_TwosComp function, and this is the output of the ALU.
For multiplication, the 16th bit (the sign bit) of both the numbers are added. This means that if the sign bits are both the same, the 16th bit of the result will be 0 (i.e. positive), and if one of the numbers is negative, the result will also be negative. The first 15 bits of the answer are then determined by multiplying the first 8 bits of each number.
For division, it is checked whether the second 16-bit number input is equal to zero (i.e. whether division by 0 is being attempted). If this is the case, the answer is set to the decimal value of 65535, which is unique to the error code. If division by 0 is not occurring, the 15th bit of the result is set to 1 - this is the division flag. The sign bit of the result is determined by adding the sign bits of the inputs, as for multiplication. The **second** set of seven bits in the answer (bits 7 to 13) store the result of the integer division of number 1 by number 2. The **first** set of seven bits (bits 0 to 6) store the remainder, or stay 0 if there is no reminder.

Binary to BCD

The binary to BCD function converts a 16-bit binary number into four separate 4-bit binary numbers representing each digit of the decimal number. First, the validity of the number is checked - if it is equal to the error value (65535), all the 4-bit outputs are set to 14. Next, the 4-bit sign output is determined, depending on the value of the MSB of the input. The binary to BCD conversion is done by left-shifting the bits of the 16-bit number into the ones, tens, hundreds and thousands columns.

This left-shifting starts from the 14th bit, as this is the maximum number of bits that will be used by the result of any calculation. If any column reaches a value of 5 or more, that column gets 3 added to it before left-shifting continues. Binary to BCD conversion is complete once all 14 bits have been shifted into the ones, tens, hundreds and thousands columns.

BCD Handler

The BCD handler uses the binary to BCD function to control to a greater extent what will be shown on the seven-segment displays. The BCD handler has an enable bit input, to provide an extra layer of control - action is only taken when this input shows 1. If the enable bit shows 0, all outputs are 0.
Once again, first it is checked if the input 16-bit binary number is equal to the error value. If so, all 5-bit outputs are set to 14, which equates to “EEEEE” on the display.
Once normal operation is confirmed, the sign, thousands-digit and ones-digit outputs of the binary to BCD converter are set to the corresponding outputs of the BCD handler. The tens-digit and hundreds-digit outputs are not set yet, as they change if the division operation is used.
The division flag (15th bit of the input to BCD handler) indicates whether division is occurring. If so, there are two cases for the result - the quotient is two digits and the remainder is one digit, or the quotient is one digit and the remainder is two digits.
In the first case, the remainder is fully displayed in the ones digit. Hence the tens digit is set to 13, which is equivalent to “r” on the seven segment display. The two-digit quotient needs to be displayed using the thousands and hundreds digits.
In the second case, the remainder needs to be displayed using the tens and ones digits. Hence the hundreds digit is set to 13 and the thousands digit displays the one-digit quotient.

BCD to Binary

The BCD to Binary function converts two 5-bit numbers (representing ones and tens digits) into a single 8-bit binary number. When the user inputs a number, the tens digit will be entered before the ones digit. If the number input is less than 10, they will only input a single digit. The default output when a key is not pressed is 31 (11111 in binary, since keypad is active low).
If neither of the inputs to the BCD to Binary function are 31, the user input is a 2-digit number. In this case, the result is (10*tens-digit-input + ones-digit-input).
If the tens-digit input is 31, the user input is a 1-digit number. In this case, the result is simply equal to the ones-digit input.
In both the above cases, the sign (8th digit) of the output is determined by the negative flag input.

Button_8To4_Converter

This function converts an 8-bit signal from keypad to a 5-bit signal to represent a button. Each 8-bit sequence from the keypad is an expression in a case statement. As long as the validPress input is true, the 8-bit sequence will result in the output register storing the corresponding 5-bit sequence. In the default case (where no button is being pressed) or if the button press is not valid, the 5-bit output is equal to 31 (11111 in binary).

KeypadReader

At every positive edge of the clock (every 2ms), KeypadReader reads in rows of keypad and outputs columns of keypad. First column is being made high at all times when the buttons have not been pressed. It keeps reading columns when the pin is active low.

Button Type
This is a simple function that takes the 5-bit signal representing a button, and depending on the value of this input, categorises it into a button type (number, operator, clear, negative, equal) - as long as validPress is true.

SevenSegmentDisplay
This function is used to display a number or character onto a single HEX display. Each 5-bit sequence representing a number/character is an expression in a case statement. As long as the enableSignal input is true, the 5-bit sequence will result in a 7-bit sequence being sent to the HEX display. In the default case, the 7-bit sequence is equal to 127 (1111111 in binary), resulting in the HEX display being blank.

LED_Lighting
This function turns on red and green LEDs depending on the numbers input into the calculator and the operations being completed.
Each number entered will result in the first five green LEDS lighting up in the corresponding binary sequence.
Each time an operation button is pushed on the keypad, a different green LED is turned on.

Register5Bit_A/Register5Bit_B
This register stores the number inputs that are to be operated on.

Register5Bit_Oper
This register stores the 5-bit sequence representing the operation to be carried out on the two numbers.

FSM
FSM stands for finite state machine and accounts for all the possible pathways that can result in unwanted outcomes. It takes in a valid button press and outputs which type of button was pressed. The undesired outcomes involve pressing ‘+’ operand for the first button press, or pressing any operands more than once consecutively (except two negative operand between two digits) and so on. When any errors are encountered, the FSM is transitioned into error state, where it turns on all displays and error flag.