Scientific Calculator (s1945267)

Instructions:

- Launch the Vivado and open the "es3_hw_assignment_1" project. (my es3_hw_assignment_2 had some problems opening it)
- Click on "File", go to "Export" and "Export Hardware", then tick "include bitstream".
- Click on "File" again and go to "Launch SDK".
- Connect the FPGA board through the USB port and turn the power switch on.
- In the Xilinx SDK, click on "Xilinx Tools", go to "Program FPGA" and click "Program".
- Click on the arrow button next to the play button and choose run "ScCalculator" program.

Basic operation of the calculator:

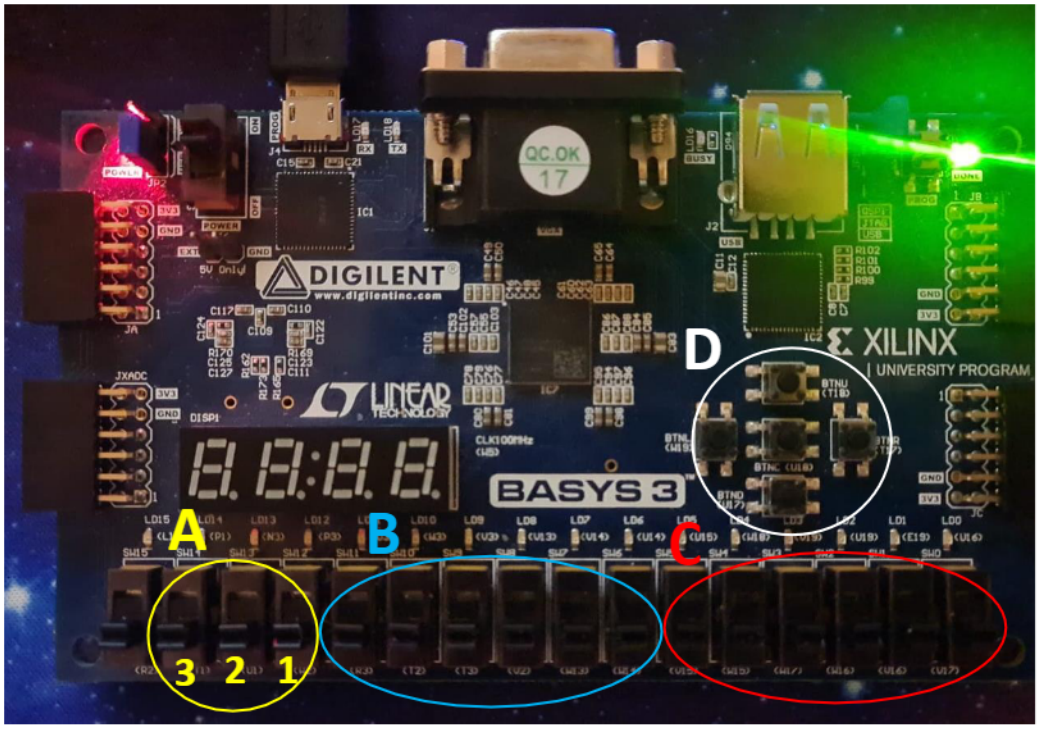


Diagram 1

- Diagram 1 shows the FPGA board which are labelled with 4 different sections, A, B, C and D.
- Section A are the switches that when is turned on or off, the user would then be able to choose four different options for operations or to display the output value using the four buttons at section D.

- The inputs are separated into 6-bit real number inputs by 6 switches, which is section B, and 6- bit imaginary number input by 6 switches, which is section C. (positive and negative)
- All the inputs are displayed at the LEDs but only real number input is displayed at the 7-segment display.
- The button and switches selection for operation would be explained more at main.c section of User's guide.
- After the user has inputted the values while pressing the button, the user must release the button to allow for the calculation to happen.
- The result of the calculation can then be displayed by turning the switch off and press while holding the right button for the real and imaginary values to be displayed.
- Centre button is used for reset.

Programmer's Guide

In this programmer's guide, I will explain all the variables used and how the code structure enables it to become a 6-bit complex number calculator.

main.c

Most of the function of the calculator is written in this "main.c" file which uses the "int main()" function.

The header files that are used in this function is "gpio_init.h" for the GPIO initializations, "seg7_display.h" for the "displayNumber()" function using the 7-segment display, "complex.h" for complex number variables and functions, and "xil_types.h" for integer type definitions.

These are the variables that I am using in my main.c code:

1) s16 num1, num2 & slideSwitchIn
- slideSwitchIn is used to input the 16 slide switches of the FPGA board. num1 is the imaginary number input while num2 is the real number input. All of them are in signed integer (s16) as the calculator uses negative and positive number input.

2) s16 switch1, switch2 & switch3
- switch1, switch2 & switch3 are the switches named SW12, SW13 and SW14 respectively. The values are inserted using "slideSwitchIn" variable and bitwise with the intended switch. For example, "switch1 = slideSwitchIn && 0x1000 (SW12)". These are the switches that are used to choose which group of operation that we want to use to calculate our inputs. (Table 1 for reference)

3) u8 BTNU, BTND, BTNL & BTNR

- These variables are used for the up button, down button, left button and right button respectively. The button values are read in using XGpio_DiscreteRead() function. The buttons are used for operation selection. (Table 1 for reference)

4) u16 counter
- The counter variable is will be used to input the amount of time needed to display the value of the answer.

5) u16 decideI & decideR
- These variables are used to decide what the type of answers are. Then, it would be called at the seg7_display.c whether to display negative, positive, decimal or whole numbers. For example, decide = 1, then the answer would be displayed in decimal form.

6) u16 deci_R, deci_I
- Supports decideI and decideR in evaluating the number type of the answers.

7) double complex z
- Output for the complex number.

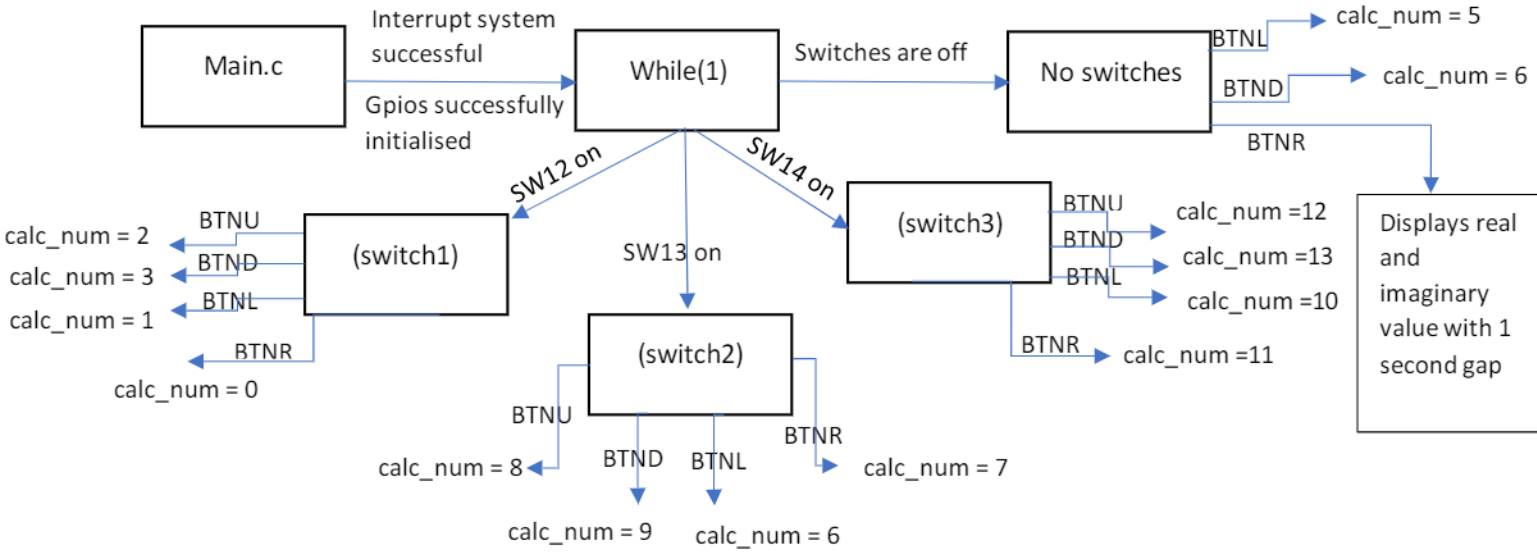
8) double a, b
- a is the variable that will read the real value of z and b is the variable that will read the imaginary value of z.

9) s16 calc_num
- This variable will be assigned with values that will choose which calculation that the user wants.

10) u16 display_num
- Variable that is used for setting the range of the loop when displaying the answer (real and imaginary) using interrupt system.

The functions that I used are "calculation()", "displayNumber()", "input_shift()" and "value_detection()". The "calculation()" function is used to receive the real and imaginary input from the user which would then be calculated and the value will be called back to the main.c function. The displayNumber() function in main.c reads in the real and imaginary number with the "decideR/decideI" variable which would then be displayed at the 7-segment display. The input_shift() function shifts the value from the first 6-bits from "slideSwitchIn" (SW0-SW5) to the next 6-bits value (SW6-SW11). The MSB of the 6-bit would then undergo Two's complement which makes both real and imaginary to have 6-bits value of positive and negative number. The "value_detection()" function would be explain further in calculations.c file.

The flow of the main.c code is shown in this flowchart for better understanding:



The "calc_num" value would then be read by the "calculation()" function and choose the appropriate operation that the user wants. I used void function for the "calculation()" which is corresponded by switch case statement inside it for more options in choosing the intended operation. Void function also enables me to use pointers inside it so that the values could be called back to the "main.c" function after calculating inside it. This enables the calculator to perform repeated operation by the illusion of saving the value inside the global variable "z" that is called inside the void function using a pointer.

The operations that would be performed are shown in the table below:

Switches / Buttons	switch1(SW12)	switch2(SW13)	switch3(SW14)	No switches
BTNU (Up)	Multiplication (calc_num = 2)	Sine (calc_num = 8)	Natural Logarithm (calc_num = 12)	No operation
BTND (Down)	Division (calc_num = 3)	Cosine (calc_num = 9)	Logarithm base 10 (calc_num = 13)	Reverse Division (calc_num = 5)
BTNL (Left)	Subtraction (calc_num = 1)	Power (calc_num = 6)	Tangent (calc_num = 10)	Reverse Subtraction (calc_num = 4)
BTNR (Right)	Addition (calc_num = 0)	Square root (calc_num = 7)	Exponential (calc_num = 11)	Displays real and imaginary result

Table 1

The “slideSwitchIn” values are read by using “XGpio_DiscreteRead() which would then be shifted into the num1 and num2 for inputs. To display the inputs using “LEDs, I used “XGpio_DiscreteWrite()” function with “LEDS_OUT” variable.

I then read the real and imaginary values of z into a and b respectively. This would then enable me to compare the value if it is a whole number, negative, positive, negative decimal or positive decimal using if-else statements. This is done in the “value_detection()” function. It is also a void function that uses pointers to call in values (a, b, decideI & decideR) back to “main.c”.

If the user wants the see the value of the real and imaginary answers, the user can just turn off the switches and press the right button. I used looped the counter by if-else which is possible due to the 'while(1)' function characteristic. If the right button is pressed, the user could see the value of real number and imaginary number both separated by a second. If not, it will display blank ‘ ‘ indicating that the answer has been shown. Before this I displayed ‘0’ for the else part but it has higher tendency to confuse the user as which values are for real and imaginary parts.

calculations.h

This is the header file for “calculations.h”. No global variables are declared in this header file as most of the void function uses only local variables that are to be read and called back and forth from main.c function. However, I defined ‘PI’ so that this definition could be used when doing trigonometric calculations. The file “complex.h” is included to enable local complex variables to be used in the functions. Operation functions are also declared with all having the same parameters. For example, for addition, “void adder(double complex *, s32, s32, u16 *, u16 *);” . The reason for using these parameters will be explained in the “calculations.c” file. Other functions that are declared are “displayNumber()”, “calculation()”, “input_shift()”, and “value_detection()”.

calculations.c

In this file, the void functions are used for operation selection and calculation, shifting the 6-bit inputs and detecting the number types for the answers.

The “calculation()” function in this file has been explained in main.c which uses switch case statement for operation selection. The code for calculating real numbers and imaginary in the operation functions have similar function structure except for division and reverse division function.

An example of the calculation code will be displayed below for better understanding. I will use the “divider()” function’s code as the example as it is has an extra code compared to others.

```
void divider(double complex *z, s32 num2, s32 num1, u16 *deci_I, u16 *deci_R)
	{
	double complex div = num2 + num1 * I;
	double complex result;
	s16 dec_I1, dec_R1;     /*          Same section for decimal detection as before
	if(div == 0){
	    *deci_I = 6;
	    *deci_R = 6;
	}
	else{
	    *z = *z / div;
	}


	    result = *z * 10;
	    dec_I1= cimag(result);
	    dec_R1 = creal (result);
	    if (dec_I1 % 10 != 0){
	        if(dec_I1 > 0){
	            *deci_I = 1;
	        }
	        else if(dec_I1 < 0){
	            *deci_I = 4;
	        }
	    }
	    if (dec_R1 % 10 != 0){
	        if(dec_R1 > 0){
	            *deci_R = 1;
	        }
	        else if(dec_R1 < 0){
	            *deci_R = 4;
	        }
	    }
	}
```

Example code 1

In this code, a local variable "div" is declared as double complex. This is to convert the s16 input of num1 and num2 to complex number as "div" for the purpose of calculating complex number. Then, another local variable, "result", is declared to store in the value of the answer. This is to detect whether the real or imaginary number is positive, negative, decimal or whole number and which are the purpose of "dec_I1" with "*deci_I" for imaginary number and the other two variables for the real number. Notice that dec_I1 and dec_R1 is actually in signed and the value of z* is multiplied by 10 when storing into result variable. This is because signed integers, s16, enables a way to detect remainder by using the 'signed_variable % 10 =! 0' method. I used this method in every other operation so that when every operation is calculated, the display would be accurate with the result that is calculated.

The variables of "*deci_I" and "*deci_R" will be called back to main.c and called back to "value_detection()" function that uses if-else statements to decide which number type the real/imaginary value of the result is. I used pointers in this function so that the value could be return back to the main function and be read by the "displayNumber()" to display the correct form. If the value is a decimal, the pointer value of imaginary or real number would be multiplied by 10 as the 7-segment display could not display a decimal number but can be manipulated to display a "decimal illusion" number. By this method, the code in seg7_display.c is structured in a way that displays decimal number even if the number is not decimal.

However, I did not insert the if else (div==0) part on other calculations except reverse division. The if else statement is used in this way to overcome zero error when dividing by zero. This will return the deci_I and deci_R as 5 to display '----'. All the methods are written in the same way except that the "div" equation is written suitable for other calculations.

For trigonometric calculations, ccos(), csin() and ctan() functions could only calculate in radian input. Since the task said that the input should be in degrees, the defined PI in calculator.h is used to convert the input as degrees into radians by multiplying the input with PI divided by 180. This enables the user to input and calculate the trigonometric calculations by using degrees.

seg7_display.c

This file contains methods to display the number on the 7-segment display that are whole numbers, positive and negative decimal numbers, negative values and also if the value is too big to be displayed on the display.

It starts with including the headers such as "seg7_display.h" and "gpio_init.h". The variables that are declared are "digitDisplayed", "digits[4]", "numOfDigits", "digitToDisplay", "digitNumber" and "negative". All are declared as u8 except for the "negative" variable. The "negative" variable uses s16 since "calculateDigits()" function could not accept a negative value, so a modulus of the negative number is read into it. This is by using the "negative" variable equals to (-1) * number which makes it a modulus of the answer.

For the "displayNumber()" function, "if-else" statements are used for multiple choice in setting the range of the number value. For the whole number, the if value is within the range of less than 9999 but bigger than -999. This is because the maximum number that could be displayed as a whole number in the 7-segment display is a 4-digit number with 9999. Moreover, -999 is the largest negative number that could be displayed. Then, "calculateDigits()" will then be performed inside the if function with an assigned value of "count = 4". The "while" loop inside if statement then extracts digits depending on the value. Another "while" loop using (digitDisplayed == FALSE) is inside the "while" loop to ensure the interrupt is occurred and ISR to finish executing digit display instructions.

For displaying decimals, more cases are added at the "digitDisplay()" function. The cases added enables numbers with a decimal point to be displayed by using the 7-segment display. Decimal numbers would be displayed if decide = 1 and negative decimal would be displayed when decide = 3. For displaying negative decimal, the code is similar to displaying decimal but "NUMBER_DASH" section similar to the displaying negative code is added. This adds a dash '-' in front of the decimal number making it a negative decimal number. Negative number is displayed if decide is equals to

For the "else" statement, I used the code that contains "NUMBER_DASH" within while loops that would output four dashes (-) at the 7-segment display. This is to represent a number that could be calculated but too big to be displayed at the 7-segment display. This

would also be displayed when decide = 5 indicating that the number is dividing by zero which is uncountable.

The void "calculateDigits" function would then calculate the number of digits needed to be displayed by using "if-else" statements. It checks from one digit long until four digits long.

The void "displayDigit()" uses a switch statement that have different cases for the "digitToDisplay" variable. These statements would detect which number to be displayed by assigning each number to "XGpio_DiscreteWrite" with the "SEG7_HEX_OUT" and digit values.

seg7_display.h

This file header defines all the variables that are being used to display the numbers that uses BCD codes. Since number 10 and 11 cannot be used as integers in this code, "NUMBER_BLANK" and "NUMBER_DASH" are used respectively.

timer_interrupt_func.c

This code uses file header "seg_7_display.h" that have declared "void hwTimerISR". This is used to have some interrupt time to wait for the digit to be displayed, as it uses "displayDigit()" function inside the "hwTimerISR" function.

x_interruptES3.c

This code initializes the interrupt system that is used in the "main.c" function.

gpio_init.h

This is the header file to define all the XGpio variables that are going to be initialized in the "gpio_init.c" file. The variables that are defined in this file are button variables (eg : P_BTN_LEFT), slide switches, led outputs and the numbering display formats.

gpio_init.c

This file contains all the function to initialize the GPIOs that were defined in the "gpio_init.h" files. It uses "initGpio()" function which uses if else statements that determines if the GPIOs are successfully initialized or not. If successful, the GPIOs are then going to be transferred into the main.c file.