Assignment1 —— Report Keyi Liu s1703084

Part 1: Introduction

I have built up two calculators based on the same coding structure,

— Basic calculator: includes simple arithmetic operations which are addition, subtraction, multiplication and division.

— Advanced Calculator: includes more complex arithmetic operations which are modulation, squareRoot, power and algorithm.

Both calculators satisfy the requirements which are

— building an 8-bit calculator including 2 inputs,

— 4 push buttons that are used to select different arithmetic operations,

— demonstrating calculated results through 7-segment display and LEDs.

The extra features I have achieved on both calculators which are

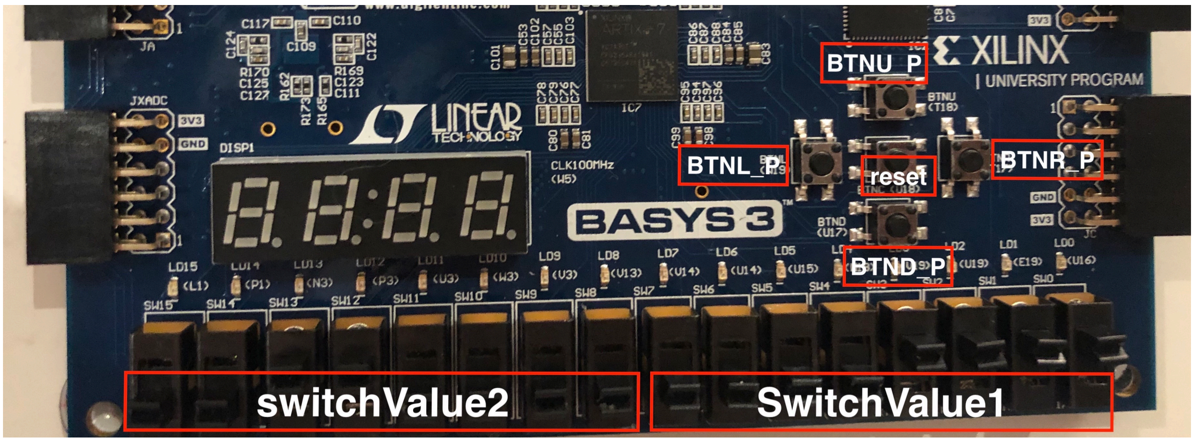
— Checking Input: Every input value can be demonstrated through 7-segment display and LEDs separately,

— Correcting Input: Input values can be demonstrated at any time before applying any arithmetic operations; corrected values can be updated and demonstrated through 7-segment immediately when pressing checking input button again.

— (Advanced Calculator Only) achieving complex arithmetic operations which involve modulation, squareRoot, power and algorithm, decimal result can be rounded off.

— (Basic Calculator Only) negative output can also be demonstrated by 7-segment and LEDs.

Part 2: Instructions Of Calculator Program



Since the two calculators are based on the same structure, I will now use the Basic Calculator as an example to explain the Instruction:

I. Turning on the FPGA(BASYS 3), connecting it to PC with the Micro-USB.
II. Opening the **Vivado** —> **Export Hardware** (Includes bitstream) —> **Launch SDK**.
III. **Program FPGA** to get connection and **Run** the program.
IV. Press BTNL_P, starting the Calculator.
V. By pushing the slide switches to set up two binary inputs, the right 8 switches represent the first input and the left 8 switches represent the second input.
VI. Press BTND_P, checking the decimal number of the first input.
Press BTNU_P, checking the decimal number of the second input.
(**Note: you can correct the inputs by pushing the slide switches and repressing the related button to check again)**
VII. Press BTNR_P, going to the *computation mode,* selecting one of operations below, and the arithmetic operation's decimal result will be demonstrated on 7-segment display and the binary result will be demonstrated on LEDs(lighting represents 1, otherwise represents 0):
- Press BTNR_P again, doing the addition of two inputs (SwitchValue1 + SwitchValue2)
- Press BTNU_P, doing the subtraction of two inputs (SwitchValue1 - SwitchValue2)
- Press BTNL_P, doing the division of two inputs (SwitchValue1 ÷ SwitchValue2)
- Press BTND_P, doing the multiplication of two inputs (SwitchValue1 x SwitchValue2)

(**Note: after starting the Calculator, only one arithmetic operation can be applied, restart the Calculator to apply other operations)**
VIII. Press reset, turning off the calculator.

On the Advanced Calculator the only difference is after going to to the *computation mode:*
- Press BTNR_P again, doing the modulation
- Press BTNU_P, doing the squareRoot of first inputs—**sqrt**(SwitchValue1)
- Press BTNL_P, doing the power of two inputs—**pow**(SwitchValue1 , SwitchValue2)
- Press BTND_P, doing the logarithm of two inputs, SwitchValue1 as exponent and SwitchValue2 as base.

Part 3: Programmer's Guide

In this part, I will explain core functionalities and code implementation of the two calculators, some codes are used as an example to be analyzed further

● *Files included in the application project*

In the **src** folder of **Assessment_Calculator** application project involves:

— **arith_operations.c:** methods of all the arithmetic operations will be called in the main.

— **gpio_init.h**: declaration of all the devices on the FPGA will be used for this project.

— **gpio_init.c**: initialization of all the variables which are declared in gpio_init.h.

— **seg7_display.h**: definitions of 7-segment BCD codes and digit selection codes.

— **seg7_display.c**: functions of **displayNumber()** and **calculateDigits()**.

— **timer_interrupt_func.c**: function of the ISR.

— **xinterruptES3.c**: a design example using the Interrupt Controller both with a PowerPC and MicroBlaze processor.

— **platform_config.h**: definition of hardware configuration.

— **platform.h**: containing functions of init_platform() and cleanup_platform().

— **platform.c**: functions of init_platform() and cleanup_platform().

● *Key functions description and explanation*

① *Structure in **int main( )***

— Declarations of all the arithmetic operations will be used.

— Define and initialize all the variables will be used.

— Check the Initialization of GPIOs and Interrupt System.

— Display the decimal number of two inputs separately and input updated if corrected.

— Apply the arithmetic operations and display the finalResult.

② *Variable declarations*

```
s32 slideSwitchIn = 0;  // 16-bit binary number
u16 finalResult = 0;    // the final result of any arithmetic operations
u16 switchValue1;       // the most right 8-bit binary number
u16 switchValue2;       // the most left 8-bit binary number
u16 input = 0;          // the input represent either Value1 or Value2
u16 BTNL_P = 0;         // initialise left button (Aim to execute addition operation)
u16 BTNR_P = 0;         // initialise right button (Aim to execute subtraction operation)
u16 BTNU_P = 0;         // initialise up button (Aim to execute multiplication operation)
u16 BTND_P = 0;         // initialise down button (Aim to execute division operation)
int leftHold = 0;       // variable for while loop to start the calculator
int status;             // declare the status (Aim to check the initialisation)
```

③ *Check the Initialization of GPIOS and Interrupt System*

```
    status = initGpio();// initialise the GPIOs
/***Check if the general-purpose IOs in the hardware are successful initialised. If not, the program is terminated.
    if (status != XST_SUCCESS) {
        print("GPIOs initialisation failed!\n\r");
        cleanup_platform();
        return 0;
    }
    status = setUpInterruptSystem(); // Setup the Interrupt System
/***Check if the interrupt system has been set up. If not, the program is terminated.******/
    if (status != XST_SUCCESS) {
        print("Interrupt system setup failed!\n\r");
        cleanup_platform();
        return 0;
    }
```

④ *Start the Calculator and check inputs*

```
while(1){
    while(leftHold == 0){
            .......
        if(BTNL_P == 1){
            .........
            leftHold=1;
        }
    }
    while(leftHold == 1){
        displayNumber(input);
        XGpio_DiscreteWrite(&LED_OUT,1,input);
        ............
        if(BTND_P == 1){
            while(BTND_P == 1)
            {
            BTND_P = XGpio_DiscreteRead(&P_BTN_DOWN, 1);
            }
            input = switchValue1;
        }
        else if(BTNU_P == 1){
            .....
        }
```

Press BTNL, assign leftHold to 1 to keep BTNL_P = 1 when releasing the button.

Calculator starts, number zero will be shown on 7-segment display as the sign of turning on the Calculator, since I have already initialized the input to 0. while(leftHold ==1) loop means BTNL_P = 1has been kept during the loop.

Press BTND, assign the input to switchValue1, switchValue1 read from slideSwitchIn by slideSwitchIn&00ff. If the switchValue has changed by pushing the slide switches, repress the BTND will get the updated input from 7-segment display and LEDs.

Press BTNU to check the second input read from slideSwitchIn by slideSwitchIn>>8 using the same structure as above.

⑤ *computation mode*

```
if(XGpio_DiscreteRead(&P_BTN_RIGHT,1) == 1){
    while(1){
        displayNumber(finalResult);
        XGpio_DiscreteWrite(&LED_OUT,1,finalResult);
    if(XGpio_DiscreteRead(&P_BTN_RIGHT,1) == 0){
        while(1){
        if(XGpio_DiscreteRead(&P_BTN_RIGHT,1) == 1 ){
            while(1){
                finalResult = addition(switchValue1,switchValue2);
                displayNumber(finalResult);
                XGpio_DiscreteWrite(&LED_OUT,1,finalResult);
                }
            }
            ................
```

Inside while(leftHold == 1) loop, press BTNR means going to computation mode. Number zero will be shown as the sign of the computation mode begins.

When inside the computation mode, assign BTNR_P to 0 while releasing the BTNR button. When pressing BTNR again, the calculator will do the addition, the result will display and hold on 7-segment and LEDs after releasing button.

Other different arithmetic operations are applied by other buttons, the code structure is same as addition.

⑥ *Complex arithmetic operations in advanced Calculator*

Add #include "**math.h**" in main.c and arith_operations.c which involves the functions of power, square root and logarithm. I will use the logarithm method in arith_operations.c as an example to explain further:

```
float logarithm(float exponent, float base){
    float resultForLog;
    //use the definition of Change of base
    resultForLog = log(base)/log(exponent);
    return resultForLog;
}
```

The header file "**math.h**" has the method of **log()**. To achieve using the two variables to construct method of logarithm(exponent, base), the idea comes from the change of base formula. SquareRoot uses the function of **sqrt()** and power uses the function of **pow(a,b)** in which ***a*** represents base and ***b*** represents index(or exponent). However, the method of modulation is using the % to execute instead of using functions from **math.h**.

⑦ *Rounding off the final result in Advanced Calculator*

Since the finalResult type of squareRoot and algorithm are **float,** which means decimal results will exist. The FPGA(BASYS 3) will automatically round the decimal result. For example, if the calculated result should be 7.83, the display will show 7 as finalResult. For better estimating, I choose rounding off when decimal result occurs. using rounding off the finalResult of the logarithm as an example below.

```
else if(XGpio_DiscreteRead(&P_BTN_DOWN,1) == 1){
    while(1){
        finalResult = logarithm(switchValue1, switchValue2);
        if(finalResult-(int)finalResult > 0.5){
        displayNumber(finalResult+1);
        XGpio_DiscreteWrite(&LED_OUT,1,finalResult+1);
        }else{
            displayNumber(finalResult);
            XGpio_DiscreteWrite(&LED_OUT,1,finalResult);
        }
    }
}
}
```

⑧ *Displaying the negative result in Basic Calculator*

The code of displaying negative result are written inside **void displayNumber()** of file **seg7_display.c**. there are slight adjustments that should be made in the seg7_display.h and seg7_display.c: change all declaration type of **numbe**r from **u16** to **s16** to allow signed value. Considering the capacity of the 7-segment display is 4 and Dash will occupy one position to represent minus. The structure is similar as **if(number <= 9999)** and the codes shows below.

```
else if (number < 0 && number >= -999) {
    absoluteValue = -number;
    calculateDigits(absoluteValue);
    digits[3-numOfDigits] = NUMBER_DASH;
    count=4;
    while (count >= 4 - numOfDigits)
            {
                digitToDisplay = digits[count-1];
                digitNumber = count;
                count--;
                while (digitDisplayed == FALSE);
                digitDisplayed = FALSE;
            }
}
```

⑨ *GPIOs used on the FPGA*

All the devices used on the FPGA are declared in the file **gpio_init.h**, once the device has been declared, related initialization need to be achieved inside the **Xstatus initGpio(void)** of the file **gpio_init.c**. an example code shows below, all the device initialization share the same structure, the only difference is to find the unique device name and deviceID. The deviceID can be checked in **xparameters.h**(located in the folder **include**/folder **microblaze_0**/folder **Assessment_Calculator_bsp**).

```
status = XGpio_Initialize(&SEG7_SEL_OUT, 1);
if (status != XST_SUCCESS)
{
    return XST_FAILURE;
}
```