Spring 2021

California State University, Northridge Department of Electrical & Computer Engineering

Lab 2
UART and GPIO

March 6, 2021

ECE 526L

Written By: Jose Luis Martinez

Introduction

UART is a serial communication protocol that stands for Universal Asynchronous Receiver/Transmitter(UART). This communication protocol works by the transmitter converting parallel data from a source device into a serialized form that is then sent to a receiving UART. Then the receiving UART will translate the serialized data back into parallel form for the connected device. UART only uses two wires, Tx pin for transmitting UART data to the RX pin of the receiving UART. An Rx pin on a UART devices will start reading when it detects a start bit and it will stop reading when it detects a stop bit. Without a clock both UART devices must operate within 10% of the same BAUD rate.

Objective

For this lab we are to use the UART device on our ZED/ZYBO Board to communicate to our computers information about the status of our buttons and switches. We are also to modify our code to show the status of our switches by turning on the corresponding LEDs.

In addition to displaying our status of our buttons/switches we are also to use UART to display the results of the following tasks:

a. Output sum of the value of dip switches and push buttons
b. Output difference of the value of dip switches and push buttons
c. Output product of the value of dip switches and push buttons
d. Output the remainder of the division of the value of dip switches by push buttons
e. Output the ceiling of square root of value of dip switches (You can use two functions: ceil and sqrt provided by math.h header file. Search online to see how you can add math.h to your SDK project.
f. Write a function to calculate x^y where x represents the value of dip switches and y represents the value of push buttons. You need to pass these two parameters to your function to calculate the results. Use a for loop to perform this task.

Methodology

First of all I created a RTL project in Vivado and configured it to work with the ZYBO board. Once created, inside the Vivado project manager tab I created a Block Design and added the ZYNQ processor along with two axi GPIO modules. I then ran the auto connect option and made one of the GPIO blocks dual channel. The GPIO with dual channels are then connected to the switches and buttons. The other GPIO block is connected to the LEDs on the SOC. Then the design is validated and we right click our design in the Sources tab and create a HDL wrapper. We then have to synthesize and implement our design in order to generate a bitstream. Clicking Generate Bitstream will automatically do these things if they have not been done beforehand. After generating the bitstream I then exported the design including the bitstream file into an xsa file to use in Vitis. Once inside Vitis, I created an Application Project with the xsa file created in Vivado. A C file was then configured to read the status of the buttons and switches and print them with UART using the functions printf() and xil_printf(). Then I modified the code to show the status of the switches on the LEDs. Finally I further modified the code to also print the tasks shown in the objective above.

Results

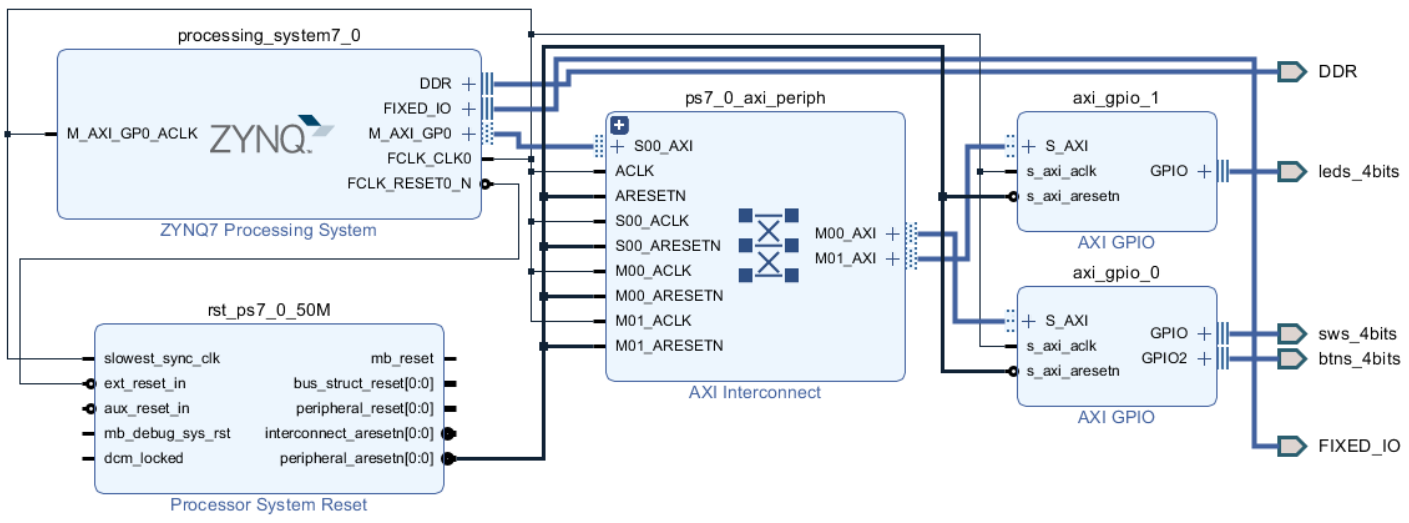


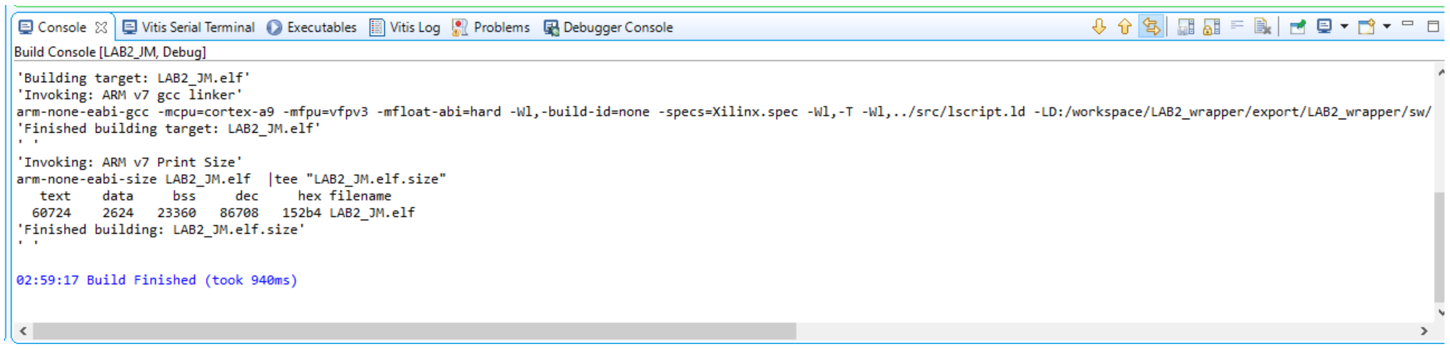
Fig. 1 Zynq Block Design

```
/*
 * Lab2.c
 *
 *  Created on:  3 March 2021
 *      Author:  Jose Luis Martinez
 *     Version:       1

********************************************************************************
***********/

/* Include Files */
#include "xparameters.h"
#include "xgpio.h"
#include "xstatus.h"
#include "xil_printf.h"
#include "platform.h"
#include <math.h>
#include <stdio.h>

/* Definitions */
#define GPIO_DEVICE_ID_INPUTS  XPAR_AXI_GPIO_0_DEVICE_ID     /* GPIO device that
LEDs are connected to */
#define GPIO_DEVICE_ID_LED  XPAR_AXI_GPIO_1_DEVICE_ID
#define LED_CHANNEL 1                                        /* GPIO port for
```

```
LEDs */
#define SW_CHANNEL 1                                        /* GPIO port for SWs
*/
#define BTN_CHANNEL 2                                       /* GPIO port for
buttons */
#define DELAY 200000

XGpio inputs, outputs;

_Bool init();
void button_status();
int powerF(int base, int power);

int main()
{
	init();
	return 0;
}

_Bool init()
{
	int input_status;
	int output_status;
	int sw_values = 0;
	int btn_values = 0;

	input_status = XGpio_Initialize(&inputs, GPIO_DEVICE_ID_INPUTS);
	output_status = XGpio_Initialize(&outputs, GPIO_DEVICE_ID_LED);

	XGpio_SetDataDirection(&inputs, SW_CHANNEL, 0xf);
	XGpio_SetDataDirection(&inputs, BTN_CHANNEL, 0xf);
	XGpio_SetDataDirection(&outputs, LED_CHANNEL, 0x0);
	init_platform();

	while((input_status == XST_SUCCESS) && (output_status == XST_SUCCESS))
	{
		sw_values = XGpio_DiscreteRead(&inputs, SW_CHANNEL);
		btn_values = XGpio_DiscreteRead(&inputs, BTN_CHANNEL);
		XGpio_DiscreteWrite(&outputs, LED_CHANNEL, sw_values);

		xil_printf("Switch value: %d\r\n", sw_values);
		xil_printf("Button value: %d\r\n", btn_values);
		xil_printf("Sum: %d\r\n", sw_values + btn_values);
```

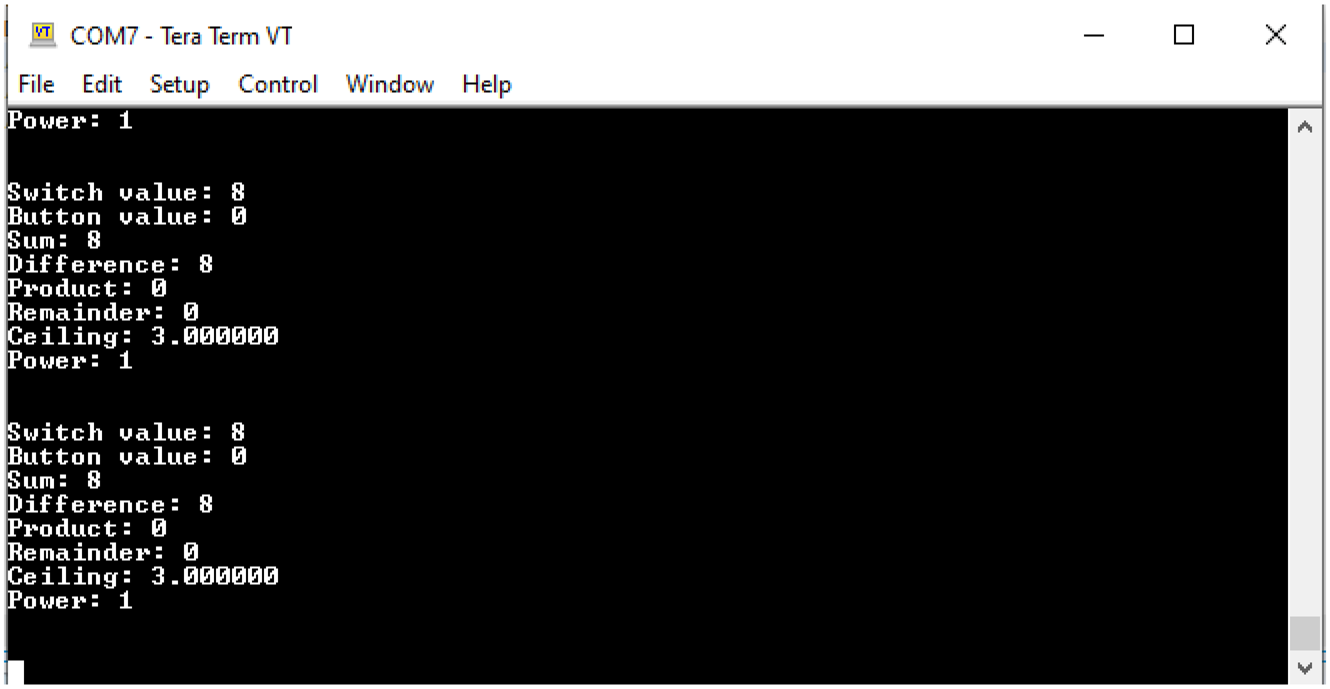
```c
                xil_printf("Difference: %d\r\n", sw_values - btn_values);
                xil_printf("Product: %d\r\n", sw_values * btn_values);
                xil_printf("Remainder: %d\r\n", sw_values % btn_values);
                printf("Ceiling: %lf\r\n", ceil(sqrt(sw_values)));
                xil_printf("Power: %d\r\n\r\n\r\n", powerF(sw_values, btn_values));
                usleep(DELAY);
        }

        cleanup_platform();
        return 0;
}

int powerF(int base, int power)
{
        int result = 1;

        for(int i=0; i<power; i++)
        {
                result = result * base;
        }

        return result;
}
```

Fig. 2 Lab2.c



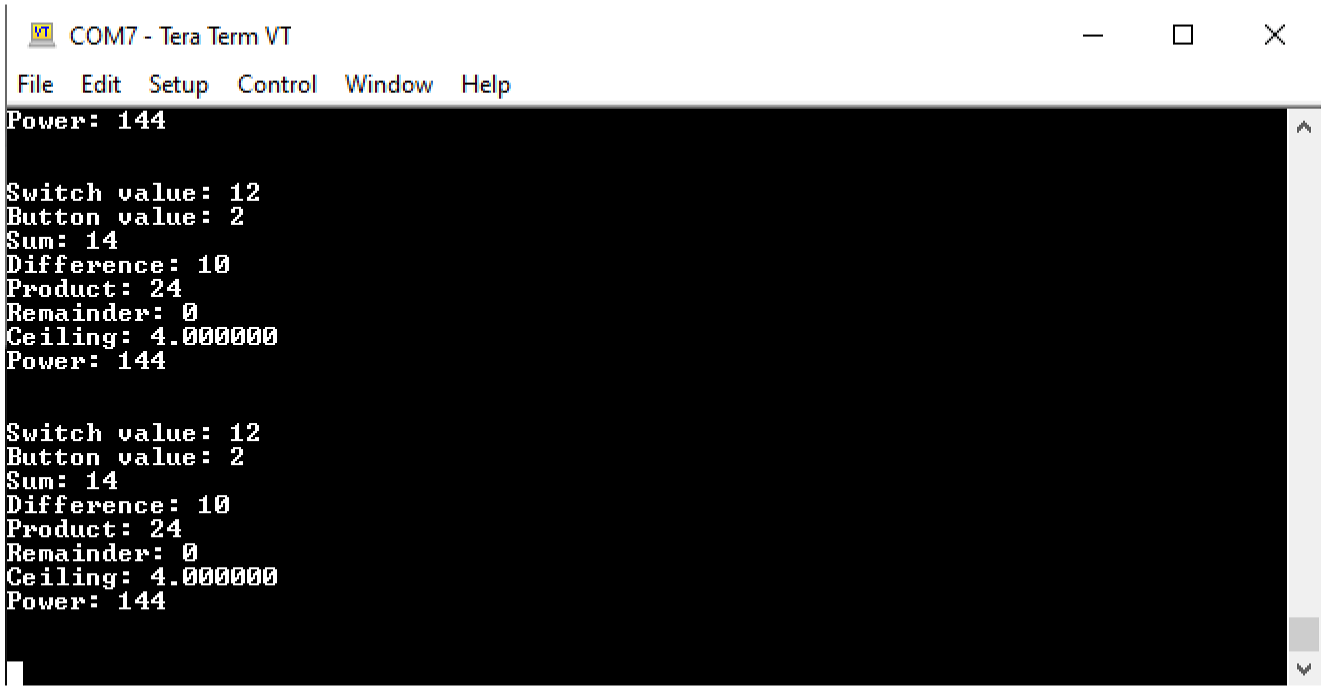
```
Build Console [LAB2_JM, Debug]
'Building target: LAB2_JM.elf'
'Invoking: ARM v7 gcc linker'
arm-none-eabi-gcc -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -Wl,-build-id=none -specs=Xilinx.spec -Wl,-T -Wl,../src/lscript.ld -LD:/workspace/LAB2_wrapper/export/LAB2_wrapper/sw/
'Finished building target: LAB2_JM.elf'
' '
'Invoking: ARM v7 Print Size'
arm-none-eabi-size LAB2_JM.elf  |tee "LAB2_JM.elf.size"
   text    data     bss     dec     hex filename
  60724    2624   23360   86708   152b4 LAB2_JM.elf
'Finished building: LAB2_JM.elf.size'
' '

02:59:17 Build Finished (took 940ms)
```

Fig. 3 Compilation Successful



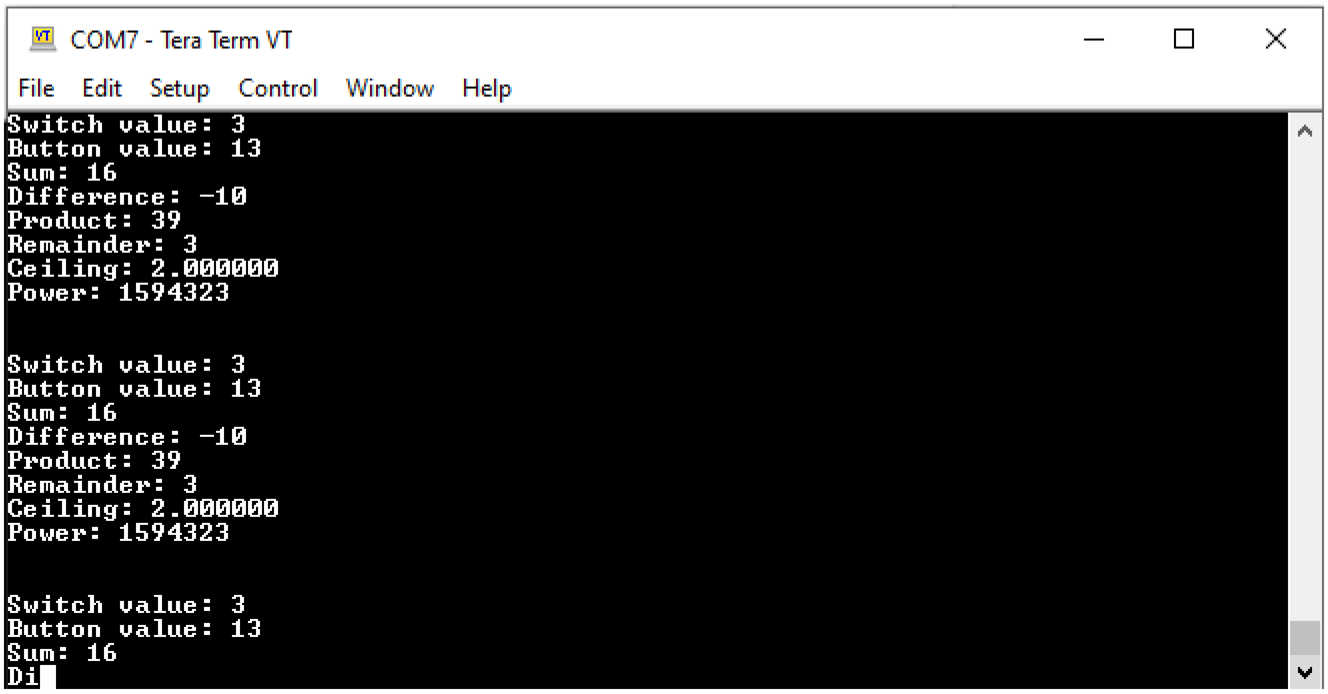
```
Power: 1


Switch value: 8
Button value: 0
Sum: 8
Difference: 8
Product: 0
Remainder: 0
Ceiling: 3.000000
Power: 1


Switch value: 8
Button value: 0
Sum: 8
Difference: 8
Product: 0
Remainder: 0
Ceiling: 3.000000
Power: 1
```

Fig. 4 Serial Output (sw=8, btn=0)



```
Power: 144


Switch value: 12
Button value: 2
Sum: 14
Difference: 10
Product: 24
Remainder: 0
Ceiling: 4.000000
Power: 144


Switch value: 12
Button value: 2
Sum: 14
Difference: 10
Product: 24
Remainder: 0
Ceiling: 4.000000
Power: 144
```

Fig. 5 Serial Output (sw=12, btn=2)



```
Switch value: 3
Button value: 13
Sum: 16
Difference: -10
Product: 39
Remainder: 3
Ceiling: 2.000000
Power: 1594323


Switch value: 3
Button value: 13
Sum: 16
Difference: -10
Product: 39
Remainder: 3
Ceiling: 2.000000
Power: 1594323


Switch value: 3
Button value: 13
Sum: 16
Di
```

Fig. 6 Serial Output (sw=3, btn=13)

Analysis

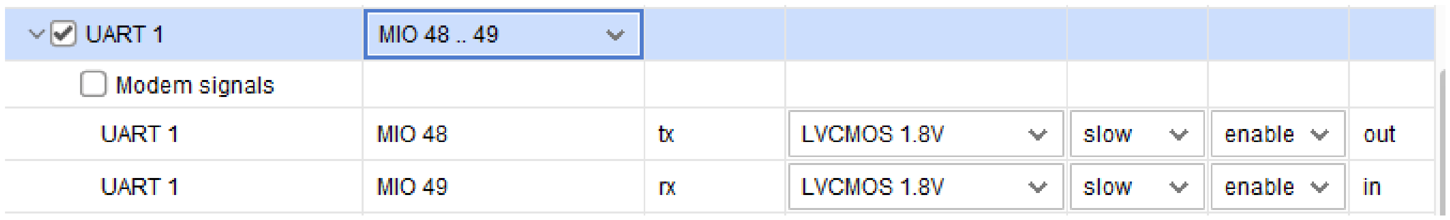
The Block Design configuration shown in **Fig. 1** shows us which blocks we are using for this design. Two axi GPIO blocks allow us to communicate with the LEDs, buttons, and switches that are present on the ZYBO board. Using that I was able to write code to drive my UART connection to the computer. If we look at **Fig. 2** I declared all my constants in the beginning and wrote most of the code in the init function. Inside the init function, both GPIOs are initialized and their data directions are also set. Once that is done then our program will run in a while loop as long as the GPIOs initialized correctly. Inside of the while loop, I used the XGpio_DiscreteRead() function to read the status of the buttons and switches. Then I used the XGpio_DiscreteWrite() function to write the status of the switches to the LEDs. Then using the values of the switches and buttons obtained above I performed the tasks and printed the results. As we can see from **Fig. 4, Fig. 5,** and **Fig. 6** we can see that my program is working as intended.

Conclusion

In conclusion, I designed a Block Design, modified it to only include UART, and wrote code to calculate and display the results using UART to my computer. I learned about BAUD rate and the rules involved with UART. I also learned how to configure the ZYNQ processor to include the systems I need and configure the clock speed. From my results I was able to accomplish the requirements for this lab.

Questions

1. What IO pins are used for UART transmit and receive pins?

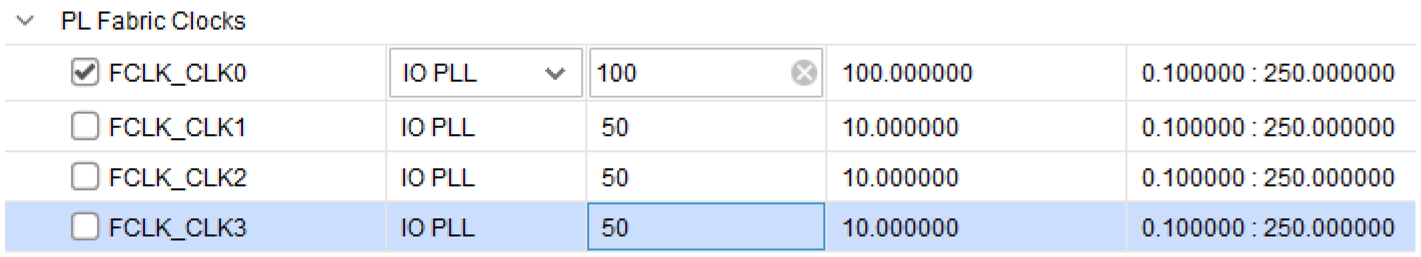


✓ UART 1	MIO 48 .. 49					
Modem signals						
UART 1	MIO 48	tx	LVCMOS 1.8V	slow	enable	out
UART 1	MIO 49	rx	LVCMOS 1.8V	slow	enable	in

Fig. 7

From **Fig. 7** we can see that pin 48 is used for Tx and pin 49 is used for Rx.

2. The lab instruction refers to the 100 MHz clock as a reference clock for the design. Where is the source of the clock?



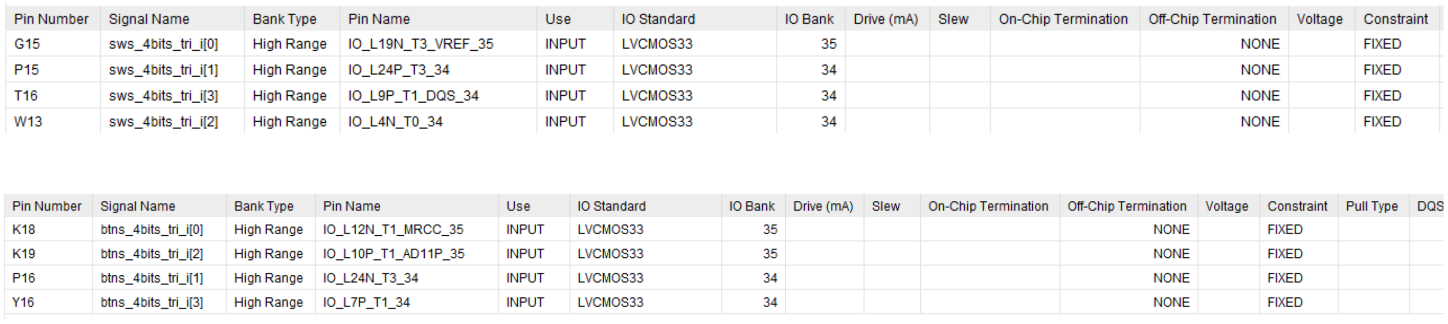
PL Fabric Clocks

✓ FCLK_CLK0	IO PLL	100	100.000000	0.100000 : 250.000000
FCLK_CLK1	IO PLL	50	10.000000	0.100000 : 250.000000
FCLK_CLK2	IO PLL	50	10.000000	0.100000 : 250.000000
FCLK_CLK3	IO PLL	50	10.000000	0.100000 : 250.000000

Fig. 8 Clock Configuration

The zybo has 4 programmable clocks and one of those clocks is being used for the ZYNQ processor.

3. Prove that push buttons and DIP switches are routed to the pins described in the Zedboard user guide. Hint: Take a look at the I/O report under "Report" tab in Vivado which shows where the signals are routed. Include all pin locations for push buttons and DIP switches.



Pin Number	Signal Name	Bank Type	Pin Name	Use	IO Standard	IO Bank	Drive (mA)	Slew	On-Chip Termination	Off-Chip Termination	Voltage	Constraint
G15	sws_4bits_tri_i[0]	High Range	IO_L19N_T3_VREF_35	INPUT	LVCMOS33	35				NONE		FIXED
P15	sws_4bits_tri_i[1]	High Range	IO_L24P_T3_34	INPUT	LVCMOS33	34				NONE		FIXED
T16	sws_4bits_tri_i[3]	High Range	IO_L9P_T1_DQS_34	INPUT	LVCMOS33	34				NONE		FIXED
W13	sws_4bits_tri_i[2]	High Range	IO_L4N_T0_34	INPUT	LVCMOS33	34				NONE		FIXED

Pin Number	Signal Name	Bank Type	Pin Name	Use	IO Standard	IO Bank	Drive (mA)	Slew	On-Chip Termination	Off-Chip Termination	Voltage	Constraint	Pull Type	DQS
K18	btns_4bits_tri_i[0]	High Range	IO_L12N_T1_MRCC_35	INPUT	LVCMOS33	35				NONE		FIXED		
K19	btns_4bits_tri_i[2]	High Range	IO_L10P_T1_AD11P_35	INPUT	LVCMOS33	35				NONE		FIXED		
P16	btns_4bits_tri_i[1]	High Range	IO_L24N_T3_34	INPUT	LVCMOS33	34				NONE		FIXED		
Y16	btns_4bits_tri_i[3]	High Range	IO_L7P_T1_34	INPUT	LVCMOS33	34				NONE		FIXED		

Fig. 9 Switch and Button Pins

4. What are the values that your see on the terminal output after pushing each button and dip switches? What if you push two buttons or two dip switches simultaneously? Write all your observations. For instance:
   SW0: 1
   SW1: 2

See **Fig. 4, 5, & 6.**

5. Modify this implementation by adding LEDs as the third GPIO to the design. The LEDs should represent the corresponding values on dip switches. Modify your software application accordingly. Implement the design and verify the working design with your instructor.
6. What parts of the block design are mapped to the processor and what parts are mapped to the FPGA?

The ZYNQ7 Processing System, Processor System Reset, and AXI Interconnect are all mapped to the processor. Both AXI GPIO blocks and the LEDS, switches, and buttons are all mapped to the FPGA.

7. Modify your software application to do the perform the following tasks:
   a. Output sum of the value of dip switches and push buttons
   b. Output difference of the value of dip switches and push buttons
   c. Output product of the value of dip switches and push buttons
   d. Output the remainder of the division of the value of dip switches by push buttons
   e. Output the ceiling of square root of value of dip switches (You can use two functions: ceil and sqrt provided by math.h header file. Search online to see how you can add math.h to your SDK project.
   f. Write a function to calculate x^y where x represents the value of dip switches and y represents the value of push buttons. You need to pass these two parameters to your function to calculate the results. Use a for loop to perform this task.
8. Explain what you learned in this lab.

See conclusion.