Spring 2021

California State University, Northridge

Department of Electrical & Computer Engineering

Lab 1
First Design on ZYNQ GPIO

February 13, 2021

ECE 526L

Written By: Jose Luis Martinez

Introduction

In lab 1 our focus is to be able to create a ZYNQ system in the Vivado software that includes both hardware and software components. We will learn to use the Vitis software that comes with Vivado to program and drive the GPIO. We will learn how to configure the ZYNQ SoC system to verify the functionality of the design and also using the IP integrator to add IP to SoC design.

In this specific lab we will configure a ZYBO/ZedBoard to use the LEDs and switches to test the points mentioned above. We will use the IP integrator to build a Block Design and then export to SDK/xsa so that we can create a software application to drive the GPIO. By doing this we will show that we can do what was mentioned in the previous paragraph.

Requirements

For lab 1 we mostly had to just follow the steps in the lab manual. The things we were tasked to come up with, on our own is making the patterns:

1. All on LEDs
2. All off LEDs
3. Alternating on and off LEDs
4. Ring counter
5. Johnson counter
6. Binary counter
7. Gray counter
8. Fibonacci counter

In addition to making the patterns above we also tasked with controlling which pattern to display by using the switches on our ZYBO/ZedBoard.

Methodology

These are the steps I used to complete this lab:

1. Creating the Block Design in Vivado and exporting hardware
    - 1.1. Make sure to add the ZYBO/ZEDBoard board files to Vivado.
    - 1.2. Create a Vivado RTL project and set the target platform to ZYBO.
    - 1.3. Inside the project manager go to IP Integrator and click Create Block Design.
    - 1.4. Right click inside the diagram window and click add IP.
    - 1.5. Type ZYNQ in the search box and click the first option.
    - 1.6. Run the auto connection.
    - 1.7. Add a GPIO block using the same way.
    - 1.8. Run auto connection again.

1.9. This time make sure to connect the GPIO to the LEDs and switches.
1.10. Click on Regenerate Layout to clean up your Block Design.
1.11. Save your design by going to File->Save Block Design.
1.12. Then validate it by going to Tools->Validate Design.
1.13. Then go to the sources tab and right click your Block Design and select Create HDL Wrapper.
1.14. In your Flow Navigator window from Program and Debug select Generate Bitstream.
1.15. Make sure to select the Open Implemented Design in the last step.
1.16. Finally we can export our hardware configuration by going to File->Export->Export Hardware.
1.17. Make sure to save it somewhere accessible like your project folder and also check the box Include bitstream.

2. Creating a Software Application With Vitis
    - 2.1. Open vitis and make a new software application by going to File->New->Application Project.
    - 2.2. Enter an appropriate name and click Next.
    - 2.3. Go to the Create a new pla… tab and click the + button.
    - 2.4. Select the xsa file you exported and click next.
    - 2.5. Select next again and then select empty application then you can finally click the Finish button.
    - 2.6. Finally if you want you can use the code from the Zynq Book tutorials zip file as a reference. To do so just look for the file and then add it to your sources.
3. Modifying the c file to accomplish the requirements.
    - 3.1. Using the ZYBO LED example source code from the zip file I made an empty c file and pasted the code in there.
    - 3.2. I deleted everything in the main for now.
    - 3.3. I added one more definition for the switches.
    - 3.4. I initialized all of my functions before the main and declared them after the main function.
    - 3.5. I created a init() function.
        - 3.5.1. This function initializes the GPIO, sets the data directions for both LEDs and switches, and has a while loop with a switch statement selecting a pattern based on the state of the switches.
    - 3.6. I created a wait_time(int) function.
        - 3.6.1. This function does as the name says so and it uses a for loop to run a number of times provided by the parameter.
    - 3.7. I created an all_led_on() function.
        - 3.7.1. Sets all LEDs on.
    - 3.8. I created an all_led_off() function.
        - 3.8.1. Sets all LEDs off.
    - 3.9. I created an on_and_off() function.
        - 3.9.1. Sets LEDs on and off with a delay between states.
    - 3.10. I created a ring_counter() function.
        - 3.10.1. Displays a ring counter pattern with a delay before switching to the next state.

3.11. I created a johnson_counter() function.
    3.11.1. Displays a johnson counter pattern with a delay before switching to the next state.
3.12. I created a gray_counter() function.
    3.12.1. Displays a gray counter pattern with a delay before switching to the next state.
3.13. I created a fibo() function.
    3.13.1. Displays a fibo pattern with a delay before switching to the next state.
3.14. I created a binary_counter() function.
    3.14.1. Displays a binary counter pattern with a delay before switching to the next state.
3.15. After all these functions were created and tested I put init() in the main function and the program now works with the criteria specified in the requirements section.

Results

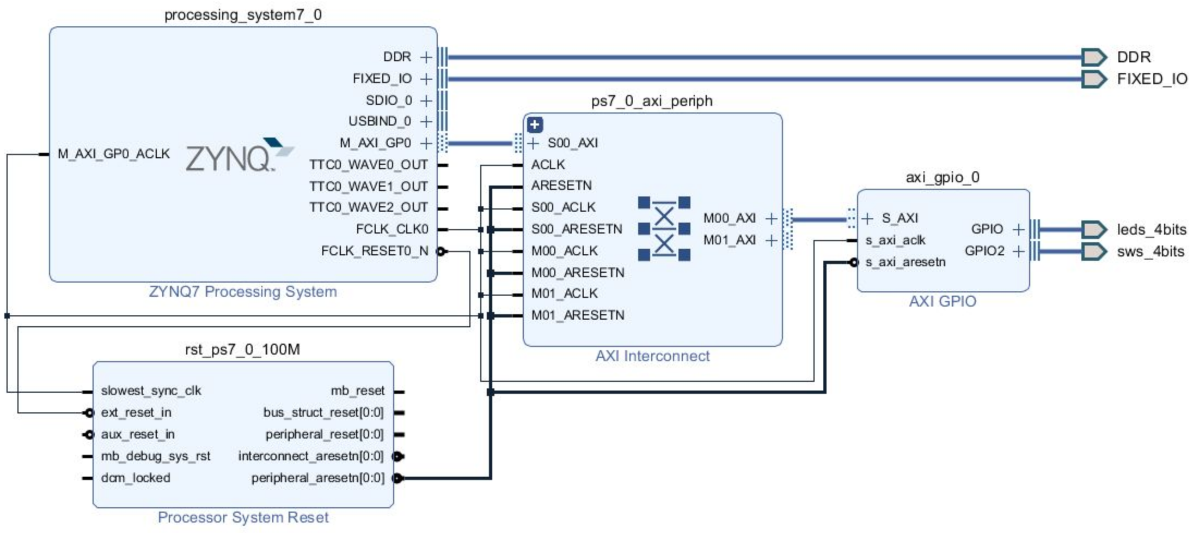
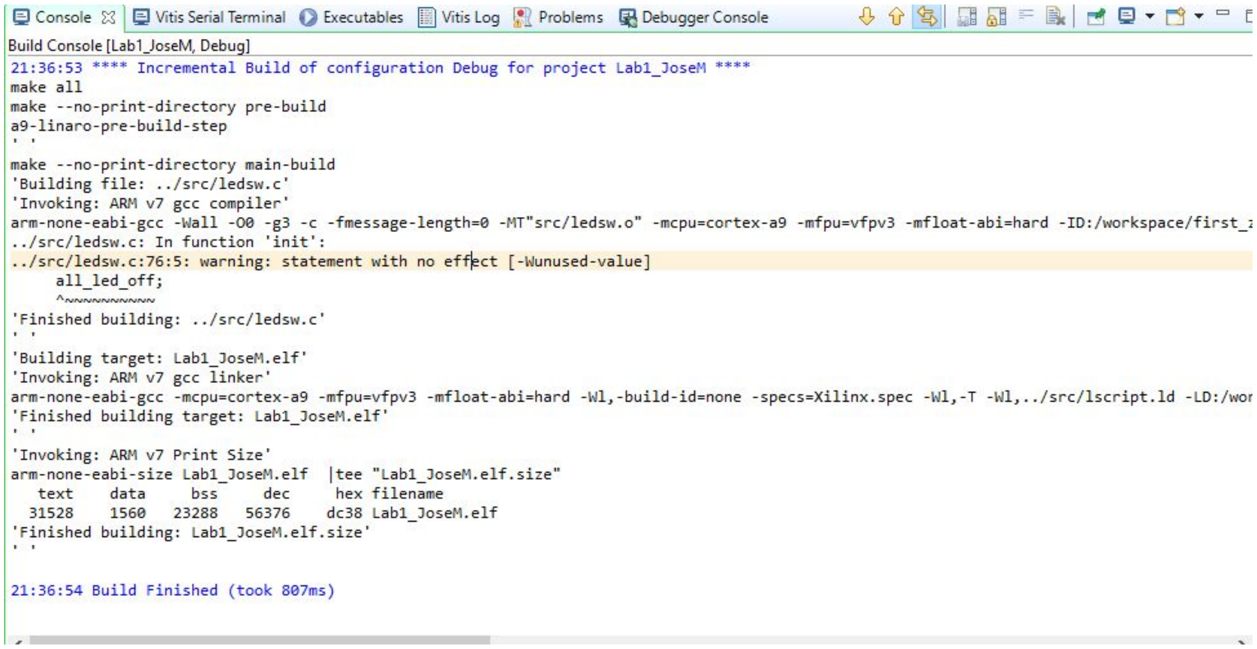


Fig 1. IP Block Design

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

**********************************************************************
**********/
```

```c
/* Include Files */
#include "xparameters.h"
#include "xgpio.h"
#include "xstatus.h"
#include "xil_printf.h"

/* Definitions */
#define GPIO_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID
#define LED 0x9
#define LED_DELAY 20000000
#define LED_CHANNEL 1
#define SW_CHANNEL 2
#define printf xil_printf

XGpio Gpio;     /* GPIO Device driver instance */

void all_led_on(void);
void all_led_off(void);
void on_and_off(void);
void ring_counter(void);
void johnson_counter(void);
void binary_counter(void);
void gray_counter(void);
void fibo(int n);
void wait_time(int time);
void init();

/* Main function. */
int main(void)
{
	init();
	return 0;
}

void init()
{
	int Status;
	Status = XGpio_Initialize(&Gpio, GPIO_DEVICE_ID);
	XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x0);
}

        XGpio_SetDataDirection(&Gpio, SW_CHANNEL, 0xf);

        while(Status == XST_SUCCESS){
                switch(XGpio_DiscreteRead(&Gpio, SW_CHANNEL)){
                        case 0x0:
                                all_led_on();
                                break;
                        case 0x1:
                                all_led_off();
                                break;
                        case 0x2:
                                on_and_off();
                                break;
                        case 0x3:
                                ring_counter();
                                break;
                        case 0x4:
                                johnson_counter();
                                break;
                        case 0x5:
                                binary_counter();
                                break;
                        case 0x6:
                                gray_counter();
                                break;
                        case 0x7:
                                fibo(8);
                                break;
                        default :
                                all_led_off;
                }
        }

        if (Status != XST_SUCCESS)
        {
                xil_printf("GPIO output to the LEDs failed!\r\n");
        }
}
```

```
void all_led_on(void)
{
	XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, 0xf);
}

void all_led_off(void)
{
	XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, 0x0);
}

void on_and_off(void)
{
	all_led_on();
	wait_time(LED_DELAY);
	all_led_off();
	wait_time(LED_DELAY);
}

void ring_counter(void)
{
	for(int i=1; i<=8; i *= 2)
	{
		XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, i);
		wait_time(LED_DELAY);
	}
}

void johnson_counter(void)
{
	int jNum = 0xf0;
	for(int i=0; i<8; i++)
	{
		XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, jNum >> i);
		wait_time(LED_DELAY);
	}
}

void binary_counter(void)
{
        for(int i=0; i<16; i++)
        {
                XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, i);
                wait_time(LED_DELAY);
        }
}

void gray_counter(void)
{
        int gcNum;
        for(int i=0; i<16; i++)
        {
                gcNum = i ^ (i >> 1);
                XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, gcNum);
                wait_time(LED_DELAY);
        }
}

void fibo(int n)
{
        int fib[2] = {0, 1};
        int temp;
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, fib[0]);
        wait_time(LED_DELAY);
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, fib[1]);
        wait_time(LED_DELAY);
        for(int i=0; i<n-2; i++){
                temp = fib[0] + fib[1];
                fib[0] = fib[1];
                fib[1] = temp;
                XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, temp);
                wait_time(LED_DELAY);
        }
}

void wait_time(int time)
{
        for(int i=0; i<time; i++){}
}
```

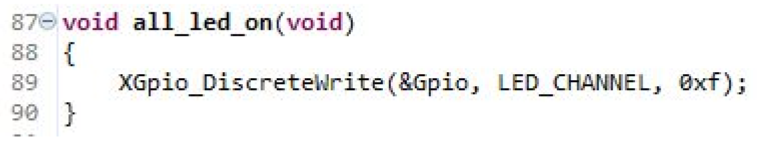


```
Build Console [Lab1_JoseM, Debug]
21:36:53 **** Incremental Build of configuration Debug for project Lab1_JoseM ****
make all
make --no-print-directory pre-build
a9-linaro-pre-build-step
' '
make --no-print-directory main-build
'Building file: ../src/ledsw.c'
'Invoking: ARM v7 gcc compiler'
arm-none-eabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/ledsw.o" -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -ID:/workspace/first_
../src/ledsw.c: In function 'init':
../src/ledsw.c:76:5: warning: statement with no effect [-Wunused-value]
     all_led_off;
     ^~~~~~~~~~~
'Finished building: ../src/ledsw.c'
' '
'Building target: Lab1_JoseM.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:/wor
'Finished building target: Lab1_JoseM.elf'
' '
'Invoking: ARM v7 Print Size'
arm-none-eabi-size Lab1_JoseM.elf  |tee "Lab1_JoseM.elf.size"
   text    data     bss     dec     hex filename
  31528    1560   23288   56376    dc38 Lab1_JoseM.elf
'Finished building: Lab1_JoseM.elf.size'
' '


21:36:54 Build Finished (took 807ms)
```

Fig. 2 Compilation Successful Message

Analysis

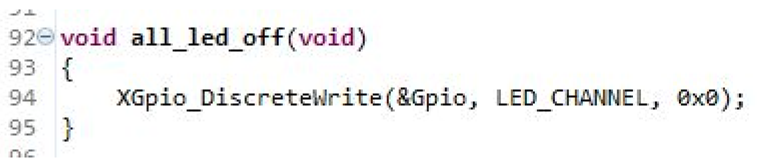
For this lab we were left with the task of designing several patterns to display with LEDs on the ZYBO board. We first had to design each pattern generator and then a way to display each pattern based on the state of the switches on the ZYBO board. A delay function called wait_time(int) was made to facilitate the creation of these said patterns.



```
void all_led_on(void)
{
    XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, 0xf);
}
```

Fig. 3 all_led_on(void)

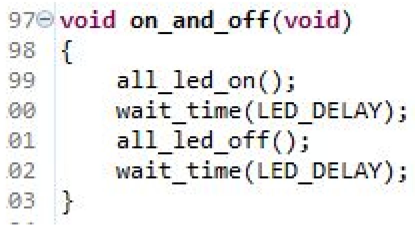
The pattern for turning all of the LEDs on was a simple function called all_led_on() and simply set the LED channel to 0xf to turn all of them on.



```
void all_led_off(void)
{
    XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, 0x0);
}
```

Fig. 4 all_led_off(void)

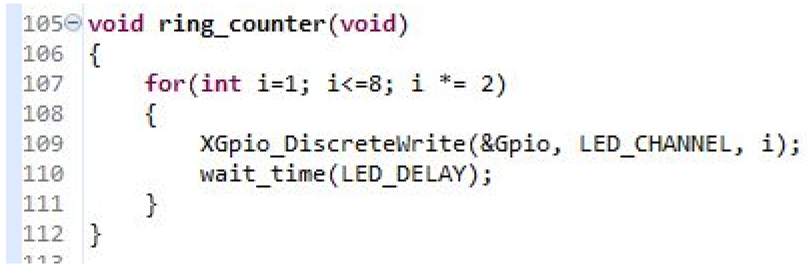
The pattern for turning all of the LEDs off was a simple function called all_led_off() and simply set the LED channel to 0x0 to turn all of them off.



```c
void on_and_off(void)
{
    all_led_on();
    wait_time(LED_DELAY);
    all_led_off();
    wait_time(LED_DELAY);
}
```

Fig. 5 on_and_off(void)

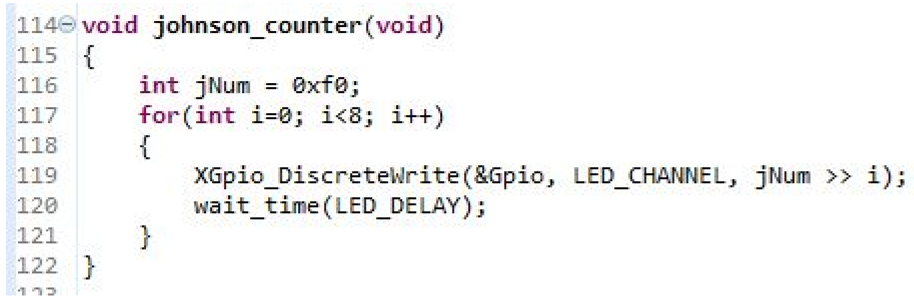
The pattern turning the LEDs on and off simply called all_led_on() and all_led_off(). Each of those calls were also followed by a wait_time(int) to see the results.



```c
void ring_counter(void)
{
    for(int i=1; i<=8; i *= 2)
    {
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, i);
        wait_time(LED_DELAY);
    }
}
```

Fig. 6 ring_counter(void)

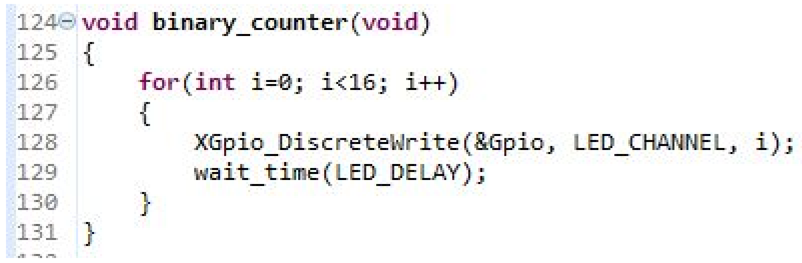
The ring counter pattern was made by having a for loop that runs starts with 1 and ends with 8. The for loop increments by multiplying itself by 2 thus simulating as if a bit was shifting to the left. Inside the loop we simply wrote the current value to the LED GPIO channel and put a delay afterwards.



```c
void johnson_counter(void)
{
    int jNum = 0xf0;
    for(int i=0; i<8; i++)
    {
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, jNum >> i);
        wait_time(LED_DELAY);
    }
}
```

Fig. 7 johnson_counter(void)

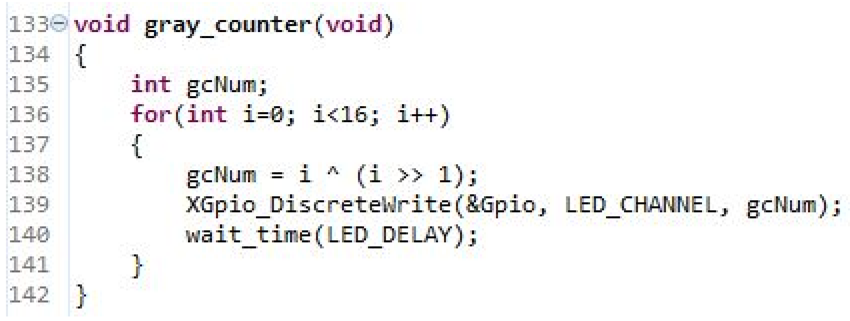
The johnson counter pattern works with a for loop that runs from 0 to 7. It uses a constant of 0xf0 and then shifts right by the current value of the loop variable and sets that value to the LEDs thus simulating a johnson counter.



```
void binary_counter(void)
{
    for(int i=0; i<16; i++)
    {
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, i);
        wait_time(LED_DELAY);
    }
}
```

Fig. 8 binary_counter(void)

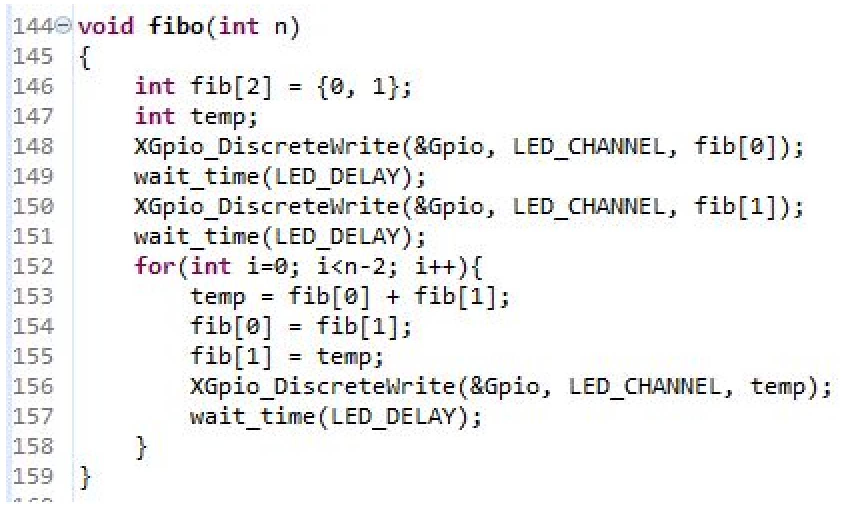
The binary counter works by having a for loop that runs from 0 to 15. The function simply sets the current loop variable to the LEDs and waits a small delay to display the results.



```
void gray_counter(void)
{
    int gcNum;
    for(int i=0; i<16; i++)
    {
        gcNum = i ^ (i >> 1);
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, gcNum);
        wait_time(LED_DELAY);
    }
}
```

Fig. 9 gray_counter(void)

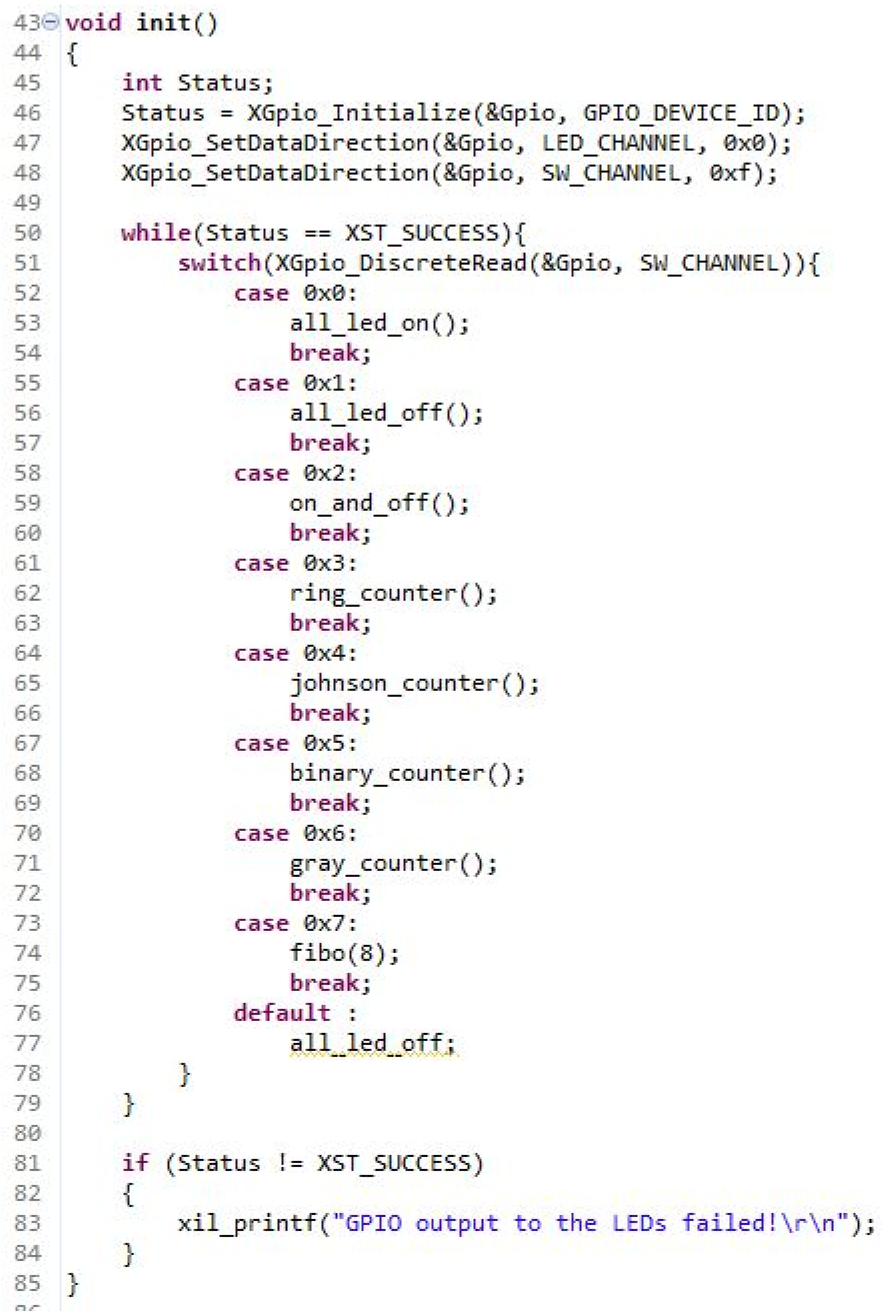
The gray counter works the same way as the binary counter except that it converts the current loop variable to its gray code counterpart.



```
void fibo(int n)
{
    int fib[2] = {0, 1};
    int temp;
    XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, fib[0]);
    wait_time(LED_DELAY);
    XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, fib[1]);
    wait_time(LED_DELAY);
    for(int i=0; i<n-2; i++){
        temp = fib[0] + fib[1];
        fib[0] = fib[1];
        fib[1] = temp;
        XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, temp);
        wait_time(LED_DELAY);
    }
}
```

Fig. 10 fibo(int)

The fibonacci counter works by displaying the first 2 numbers in the fibonacci sequence. It then runs a for loop the remainder of the fibonacci sequence specified by the parameter. Inside the for loop the current value of the fibo sequence is added from the previous value and then it is displayed with the LEDs.



```
void init()
{
    int Status;
    Status = XGpio_Initialize(&Gpio, GPIO_DEVICE_ID);
    XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x0);
    XGpio_SetDataDirection(&Gpio, SW_CHANNEL, 0xf);

    while(Status == XST_SUCCESS){
        switch(XGpio_DiscreteRead(&Gpio, SW_CHANNEL)){
            case 0x0:
                all_led_on();
                break;
            case 0x1:
                all_led_off();
                break;
            case 0x2:
                on_and_off();
                break;
            case 0x3:
                ring_counter();
                break;
            case 0x4:
                johnson_counter();
                break;
            case 0x5:
                binary_counter();
                break;
            case 0x6:
                gray_counter();
                break;
            case 0x7:
                fibo(8);
                break;
            default :
                all_led_off;
        }
    }

    if (Status != XST_SUCCESS)
    {
        xil_printf("GPIO output to the LEDs failed!\r\n");
    }
}
```

Fig. 11 init(void)

Lastly, to put all of these together the init() function initializes the GPIO. Then sets the LED channel data direction to write and set the switch channel data direction to read. A while loop follows this and its condition to run is that GPIO successfully initializes. Inside this while loop there is a switch statement that reads the values of the switches and selects its cases based on that. Each case runs only one of the patterns and the default case turns all LEDs off.

Conclusion

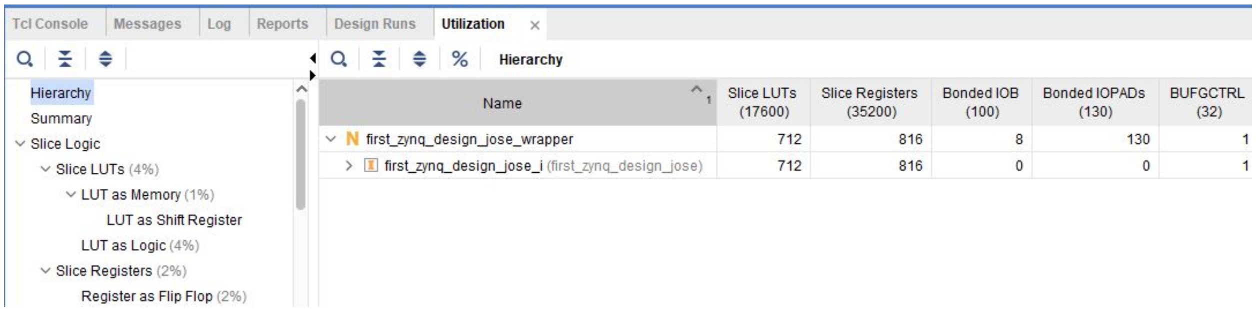
In conclusion we learned how to create a project and target a specific board in vivado, learned how to create a Block Design using the IP Integrator, and finally exporting that hardware design to an SDK/xsa file for use in creating application software. We were also tasked in the second week to design various patterns and display them to the ZYBO built in LEDs using GPIO. By doing this we also learned how to control the ZYBO board using the c language.

Questions

Week 1:

1. What is the utilization of the part for this design in terms of FPGA resources? You can find this information from two reports: post synthesis and post implementation. Is the device utilization the same in both reports? If not, why?

Looking at the reports in **Fig. 12** and **Fig. 13** we can see that they are not using the same resources. The post implementation utilization report uses less slice LUTs and slice registers compared to the post synthesis utilization report.



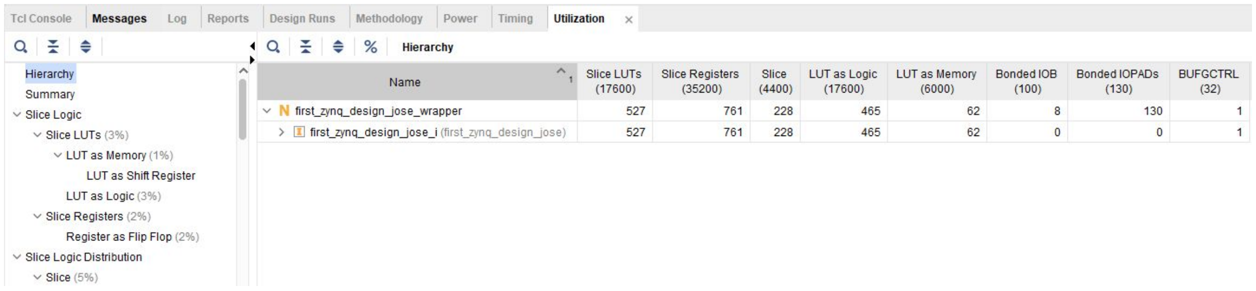
Tcl Console | Messages | Log | Reports | Design Runs | Utilization

Hierarchy

- Hierarchy
- Summary
- Slice Logic
  - Slice LUTs (4%)
    - LUT as Memory (1%)
      - LUT as Shift Register
    - LUT as Logic (4%)
  - Slice Registers (2%)
    - Register as Flip Flop (2%)

Name	Slice LUTs (17600)	Slice Registers (35200)	Bonded IOB (100)	Bonded IOPADs (130)	BUFGCTRL (32)
first_zynq_design_jose_wrapper	712	816	8	130	1
first_zynq_design_jose_i (first_zynq_design_jose)	712	816	0	0	1

Fig. 12 Post Synthesis Utilization Report



Tcl Console | Messages | Log | Reports | Design Runs | Methodology | Power | Timing | Utilization

Hierarchy

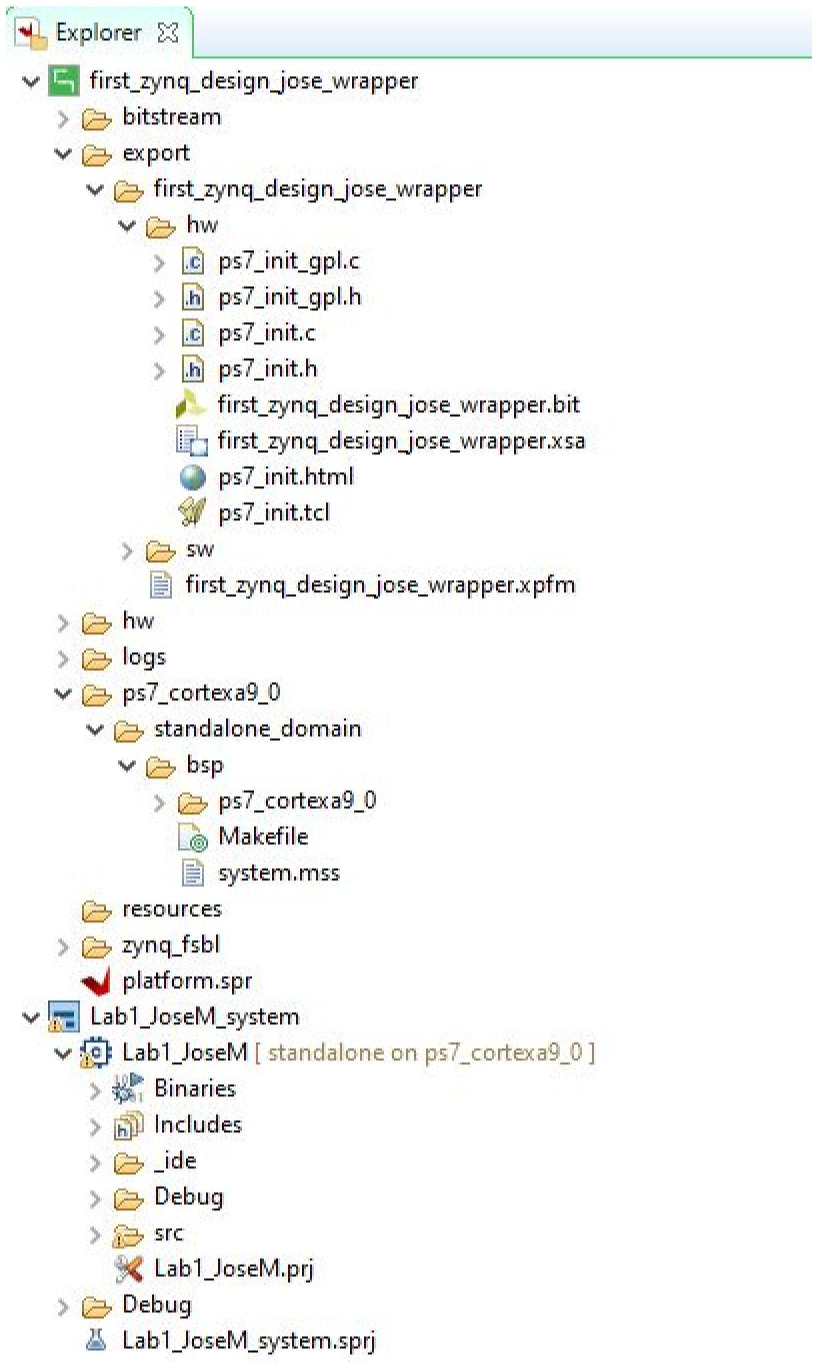
- Hierarchy
- Summary
- Slice Logic
  - Slice LUTs (3%)
    - LUT as Memory (1%)
      - LUT as Shift Register
    - LUT as Logic (3%)
  - Slice Registers (2%)
    - Register as Flip Flop (2%)
- Slice Logic Distribution
  - Slice (5%)

Name	Slice LUTs (17600)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	Bonded IOB (100)	Bonded IOPADs (130)	BUFGCTRL (32)
first_zynq_design_jose_wrapper	527	761	228	465	62	8	130	1
first_zynq_design_jose_i (first_zynq_design_jose)	527	761	228	465	62	0	0	1

Fig. 13 Post Implementation Utilization Report

2. When SDK project environment is set up completely, you will see three main folders in the project Explorer. What are these three main folders and what are they related to?

Since I am using the 2019.2 version of Vivado they have changed the SDK method to a different one using an xsa file instead. So if we look at **Fig. 15** we can see that we have two main folders, one being first_zynq_design_jose_wrapper and the other being Lab1_JoseM_system. The first_zynq_design_jose_wrapper folder is where the configured hardware is located. It also contains all of the source code required to program the Soc. The Lab1_JoseM_system folder is where we create our application software.

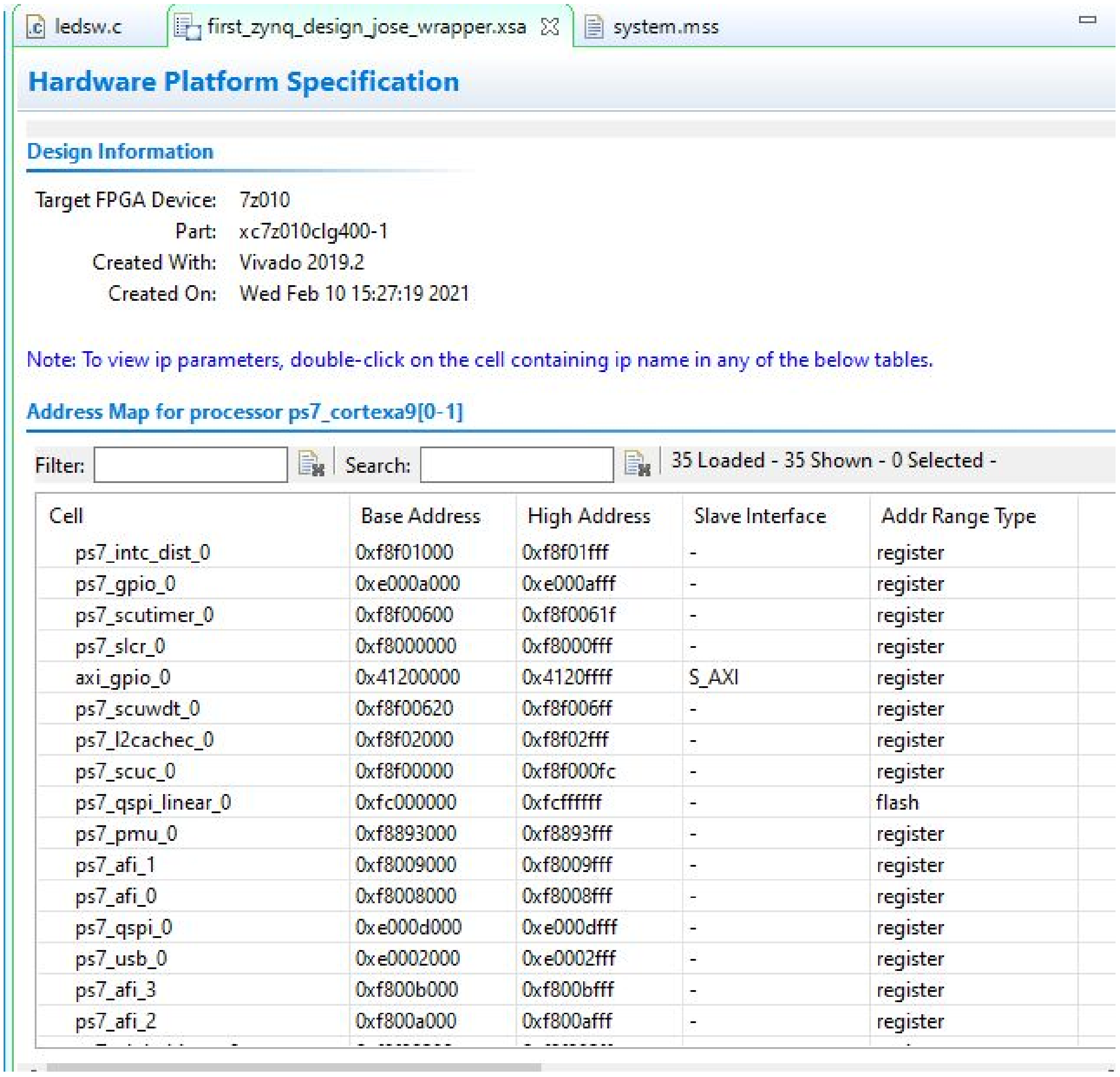


```
Explorer
first_zynq_design_jose_wrapper
    bitstream
    export
        first_zynq_design_jose_wrapper
            hw
                ps7_init_gpl.c
                ps7_init_gpl.h
                ps7_init.c
                ps7_init.h
                first_zynq_design_jose_wrapper.bit
                first_zynq_design_jose_wrapper.xsa
                ps7_init.html
                ps7_init.tcl
            sw
            first_zynq_design_jose_wrapper.xpfm
    hw
    logs
    ps7_cortexa9_0
        standalone_domain
            bsp
                ps7_cortexa9_0
                Makefile
                system.mss
    resources
    zynq_fsbl
    platform.spr
Lab1_JoseM_system
    Lab1_JoseM [ standalone on ps7_cortexa9_0 ]
        Binaries
        Includes
        _ide
        Debug
        src
        Lab1_JoseM.prj
    Debug
    Lab1_JoseM_system.sprj
```

Fig. 15 Vitis Explorer Tab

3. Open the two files system.hdf and system.mss. What do these files represent?

The system.mss contains all of the drivers for peripherals, processor information, and information of what os is in use. In my case the system.hdf file is replaced with first_zynq_design_jose_wrapper.xsa shown in **Fig. 16.** This file contains the information about the target FPGA device and also has the address map of the components in the Soc.



ledsw.c | first_zynq_design_jose_wrapper.xsa | system.mss

Hardware Platform Specification

Design Information

Target FPGA Device:	7z010
Part:	xc7z010clg400-1
Created With:	Vivado 2019.2
Created On:	Wed Feb 10 15:27:19 2021

Note: To view ip parameters, double-click on the cell containing ip name in any of the below tables.

Address Map for processor ps7_cortexa9[0-1]

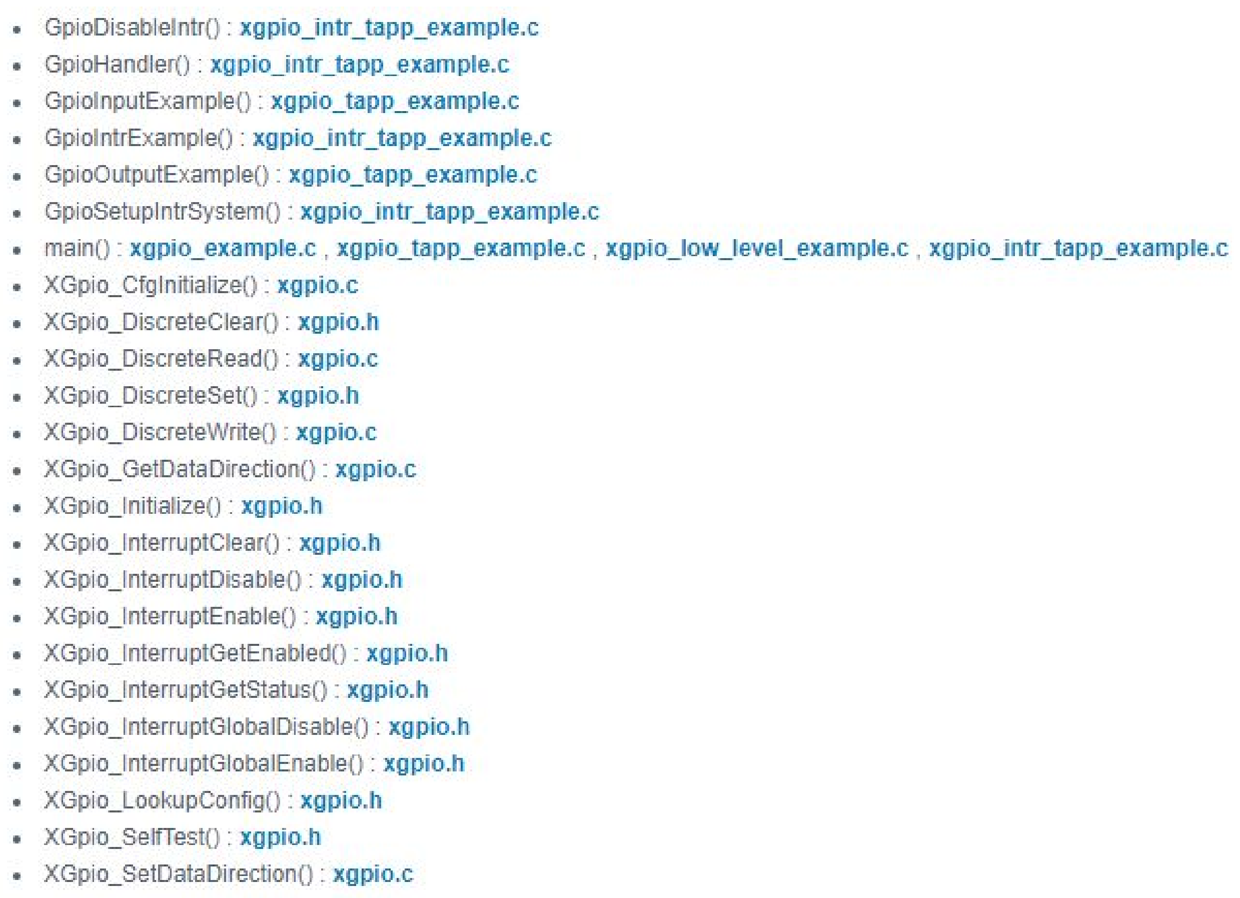
Filter: Search: 35 Loaded - 35 Shown - 0 Selected -

Cell	Base Address	High Address	Slave Interface	Addr Range Type
ps7_intc_dist_0	0xf8f01000	0xf8f01fff	-	register
ps7_gpio_0	0xe000a000	0xe000afff	-	register
ps7_scutimer_0	0xf8f00600	0xf8f0061f	-	register
ps7_slcr_0	0xf8000000	0xf8000fff	-	register
axi_gpio_0	0x41200000	0x4120ffff	S_AXI	register
ps7_scuwdt_0	0xf8f00620	0xf8f006ff	-	register
ps7_l2cachec_0	0xf8f02000	0xf8f02fff	-	register
ps7_scuc_0	0xf8f00000	0xf8f000fc	-	register
ps7_qspi_linear_0	0xfc000000	0xfcffffff	-	flash
ps7_pmu_0	0xf8893000	0xf8893fff	-	register
ps7_afi_1	0xf8009000	0xf8009fff	-	register
ps7_afi_0	0xf8008000	0xf8008fff	-	register
ps7_qspi_0	0xe000d000	0xe000dfff	-	register
ps7_usb_0	0xe0002000	0xe0002fff	-	register
ps7_afi_3	0xf800b000	0xf800bfff	-	register
ps7_afi_2	0xf800a000	0xf800afff	-	register

Fig. 16 first_zynq_design_jose_wrapper.xsa file

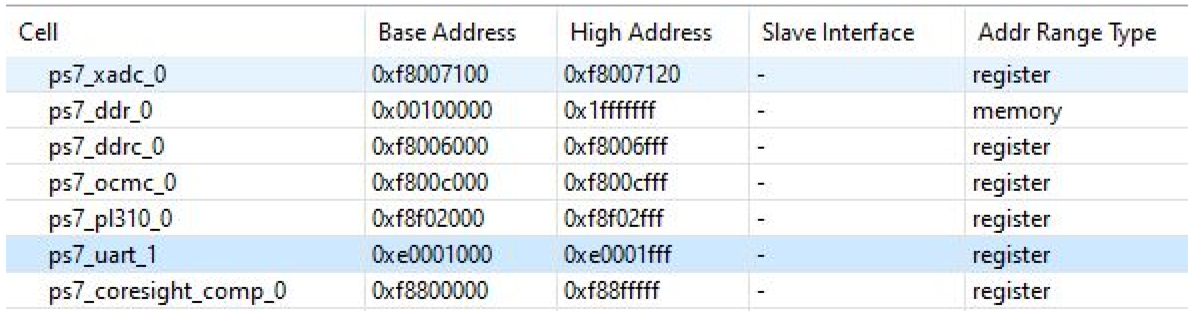
4. Review system.hdf and system.mss and answer the following questions:
   a. As you know, processor peripherals are memory mapped. What is the UART address space?
   b. Find out what functions are available for GPIO operations to use to develop software applications?

a) According to **Fig. 18,** the address space for UART is from 0xe0001000 to 0xe0001000.
b) GPIO functions are located in **Fig. 17.**



- GpioDisableIntr() : xgpio_intr_tapp_example.c
- GpioHandler() : xgpio_intr_tapp_example.c
- GpioInputExample() : xgpio_tapp_example.c
- GpioIntrExample() : xgpio_intr_tapp_example.c
- GpioOutputExample() : xgpio_tapp_example.c
- GpioSetupIntrSystem() : xgpio_intr_tapp_example.c
- main() : xgpio_example.c , xgpio_tapp_example.c , xgpio_low_level_example.c , xgpio_intr_tapp_example.c
- XGpio_CfgInitialize() : xgpio.c
- XGpio_DiscreteClear() : xgpio.h
- XGpio_DiscreteRead() : xgpio.c
- XGpio_DiscreteSet() : xgpio.h
- XGpio_DiscreteWrite() : xgpio.c
- XGpio_GetDataDirection() : xgpio.c
- XGpio_Initialize() : xgpio.h
- XGpio_InterruptClear() : xgpio.h
- XGpio_InterruptDisable() : xgpio.h
- XGpio_InterruptEnable() : xgpio.h
- XGpio_InterruptGetEnabled() : xgpio.h
- XGpio_InterruptGetStatus() : xgpio.h
- XGpio_InterruptGlobalDisable() : xgpio.h
- XGpio_InterruptGlobalEnable() : xgpio.h
- XGpio_LookupConfig() : xgpio.h
- XGpio_SelfTest() : xgpio.h
- XGpio_SetDataDirection() : xgpio.c

Fig. 17 GPIO functions



Cell	Base Address	High Address	Slave Interface	Addr Range Type
ps7_xadc_0	0xf8007100	0xf8007120	-	register
ps7_ddr_0	0x00100000	0x1fffffff	-	memory
ps7_ddrc_0	0xf8006000	0xf8006fff	-	register
ps7_ocmc_0	0xf800c000	0xf800cfff	-	register
ps7_pl310_0	0xf8f02000	0xf8f02fff	-	register
ps7_uart_1	0xe0001000	0xe0001fff	-	register
ps7_coresight_comp_0	0xf8800000	0xf88fffff	-	register

Fig. 18 UART Address Space

5. Mark up the blocks that have been used in this lab in the following Zynq SoC block diagram.

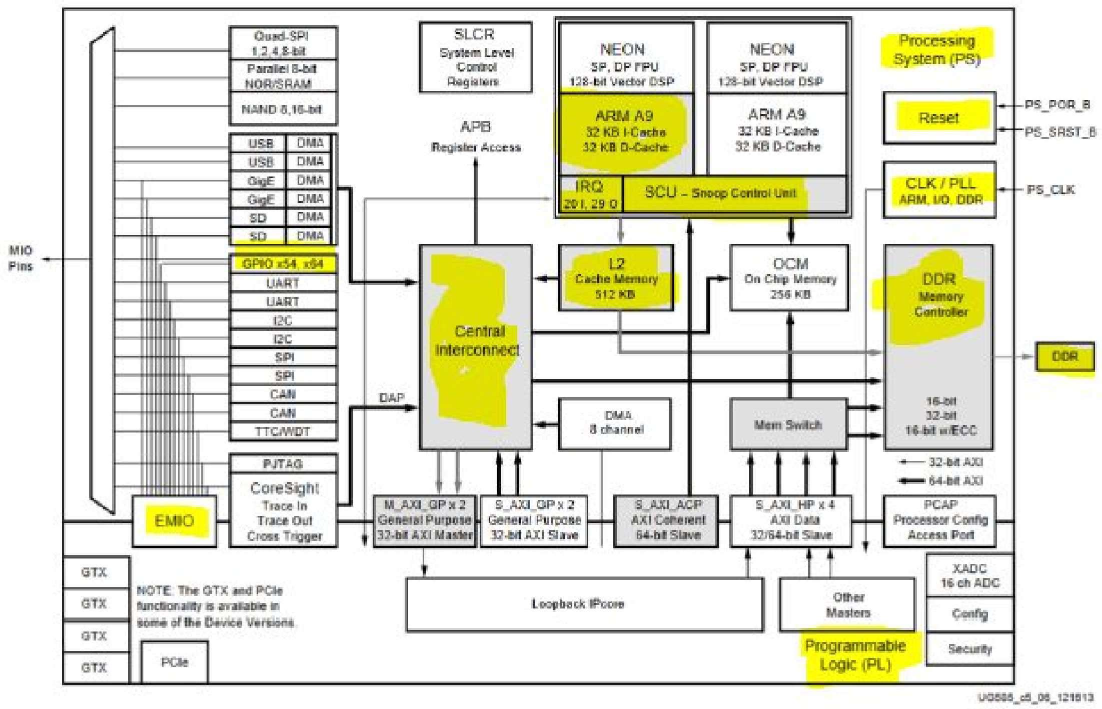


Fig. 19 Blocks Used

6. Review the C source file (LED_test_tut_1C.c) and answer the following questions:
   a. What is the value of the variable "XPAR_AXI_GPIO_0_DEVICE_ID"?
   b. What are the functions in this source file?
   c. Explain the tasks of each function you listed above.
   d. Explain how the task of each function you listed above is performed.
   e. Explain all the arguments of the functions you listed above.

a) 0

b) Functions in this file are main(), XGpio_Initialize(&Gpio, GPIO_DEVICE_ID), XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x0), and XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, 0xf).

c) Their tasks are as follows:
   i) main(): Serves as the starting point in your application.
   ii) XGpio_Initialize(&Gpio, GPIO_DEVICE_ID): initializes the GPIO so it is ready to use.
   iii) XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x0): sets the channel of the specified GPIO to either write or read.
   iv) XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, 0xf): sets a value to the specified channel in the GPIO.

d) They perform their tasks are as follows:
   i) main(): Anything in the main will be run sequentially from top to bottom.
   ii) XGpio_Initialize(&Gpio, GPIO_DEVICE_ID): Takes in the address of the GPIO and its ID to initialize it.

iii) XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x0): Will set the channel to either read or write depending on the parameters.

iv) XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, 0xf): Will write a value based on the parameters to the given channel from a GPIO.

e) The arguments for each function are as follows:

i) main(): Does not have any parameters.

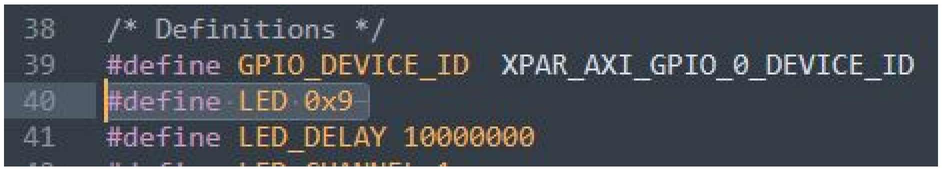
ii) XGpio_Initialize(&Gpio, GPIO_DEVICE_ID): The first parameter is the address of the GPIO you are trying to initialize and the second contains the ID of the GPIO.

iii) XGpio_SetDataDirection(&Gpio, LED_CHANNEL, 0x0): The first parameter is the address of the GPIO, the second parameter the channel, and last one is the direction that you are setting the channel to.

iv) XGpio_DiscreteWrite(&Gpio, LED_CHANNEL, 0xf): The first parameter is the address of the GPIO, the second parameter is the channel within the GPIO, and last parameter is value that you want to write to that channel.

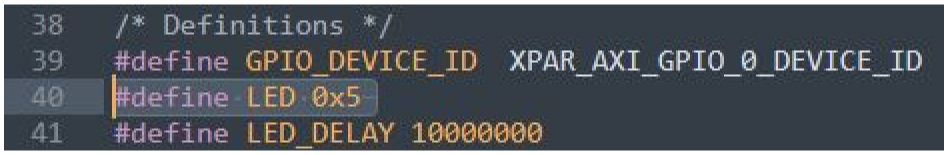
7. How do you change the LED pattern on the board? Create a custom pattern for LED on the board. Explain your answer by presenting the original and modified source code.

You can simply change the pattern on the board by modifying the constant LED. When you compare **Fig. 20** and **Fig. 21** you can see only the value of LED has been changed. So now instead of the 1001->0110 pattern, the new pattern will be 0101->1010.



```
/* Definitions */
#define GPIO_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID
#define LED 0x9
#define LED_DELAY 10000000
```

Fig. 20 Original Code

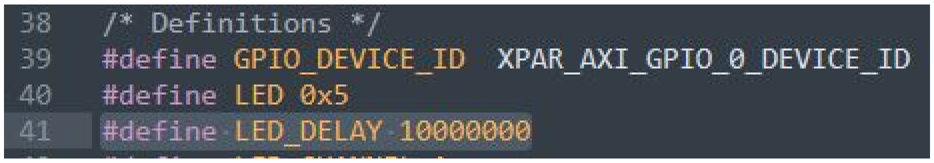


```
/* Definitions */
#define GPIO_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID
#define LED 0x5
#define LED_DELAY 10000000
```

Fig. 21 Modified code

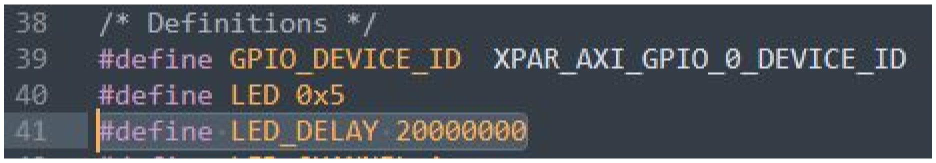
8. How do you change the LED blinking rate? Explain your answer by presenting the original and modified source code.

To change the blinking rate of the pattern all you have to do is modify the LED_DELAY value. Looking at **Fig. 22** and **Fig. 23,** I modified the original value of LED_DELAY to be 20000000 instead of 10000000, essentially doubling the time it takes to blink.



```
/* Definitions */
#define GPIO_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID
#define LED 0x5
#define LED_DELAY 10000000
```

Fig. 22 Original Code



```
/* Definitions */
#define GPIO_DEVICE_ID  XPAR_AXI_GPIO_0_DEVICE_ID
#define LED 0x5
#define LED_DELAY 20000000
```

Fig. 23 Modified Code

Week 2:

1. Explain your algorithm to design the Gray code counter in task 1. Your Gray code counter should count all 256 combinations!

The algorithm for the gray code counter includes a for loop counting from 0 to 7 because of the ZYBO board only having 4 bits. However modifying the for loop to run 256 times will still work. Inside the for loop we have a single statement that will convert our values into grey code by taking the current value and XORing it bitwise with the current value shifted once to the right. Using this statement only modifying the for loop value will be necessary to count all 256 combinations. The code for this can be found in **Fig. 9.**

2. Include the block design as well as the software application for all designs in an organized manner.

See the **Results** section for both block diagram and source code.