Spring 2021

California State University, Northridge

Department of Electrical & Computer Engineering

Final Project
Laser with Motion Control

May 17, 2021

ECE 526L

Written By: Jose Luis Martinez

Introduction

For my final project I have chosen to make a **laser with motion control** (lwmc) using the ZYBO Soc. The lwmc will operate differently depending on the current state. States will be based on the value of the current value of the switches. The lwmc will sit idle in the first state, for the 2nd state it will take readings from the joystick and move the laser accordingly. The other states will simply generate a pattern.

My project includes both elements of hardware and software design. The hardware design of the project requires IPs to interface with the PMODS and IPs to generate PWM signals to control the servos. The Software design portion will control the actual logic flow of the project. The software will change the current state according to the switches, set the duty for each PWM IP, take readings from the joystick, and set text to the OLED display based on the state.

Materials/Software

- Zybo
- 2 servos
- Laser module
- OLED PMOD
- Joystick PMOD
- CON 3 PMOD (Optional)
- 5V voltage regulator
- DC Power Supply
- Vivado 2019.1

Methodology

The following are the steps I took to complete my final project:

1. Open Vivado 2019.1
    a. Create a new project.
    b. Make sure to select ZYBO from the boards list.
    c. Once inside vivado I imported IPs provided from Digilent that had all of the IPs I would need.
    d. I also imported the ZYBO constraint files and uncomented 2 ports from pmod port JE.
    e. Create a new block design.
        i. Add Zynq processor and run autoconnection.
        ii. Add axi gpio and run auto connection making sure that SW are selected.
        iii. Add joystick pmod ip and run auto connection.
        iv. Add oled pmod and run auto connection.
        v. Add two PWM ips and run auto connection.
        vi. Add external outputs for each PWM and connect them to what you uncommented in the constraints file.
    f. Save block design.
    g. Validate block design.
    h. Generate HDL wrapper for the block design.
    i. Generate bitstream file.
    j. Launch SDK.
2. Once in sdk create a new application.
    a. Click next and select hello world as the base.

b. I then renamed the helloworld.c to main.c.
c. I then wrote my code as shown in **Fig. 2.**

Results/Code

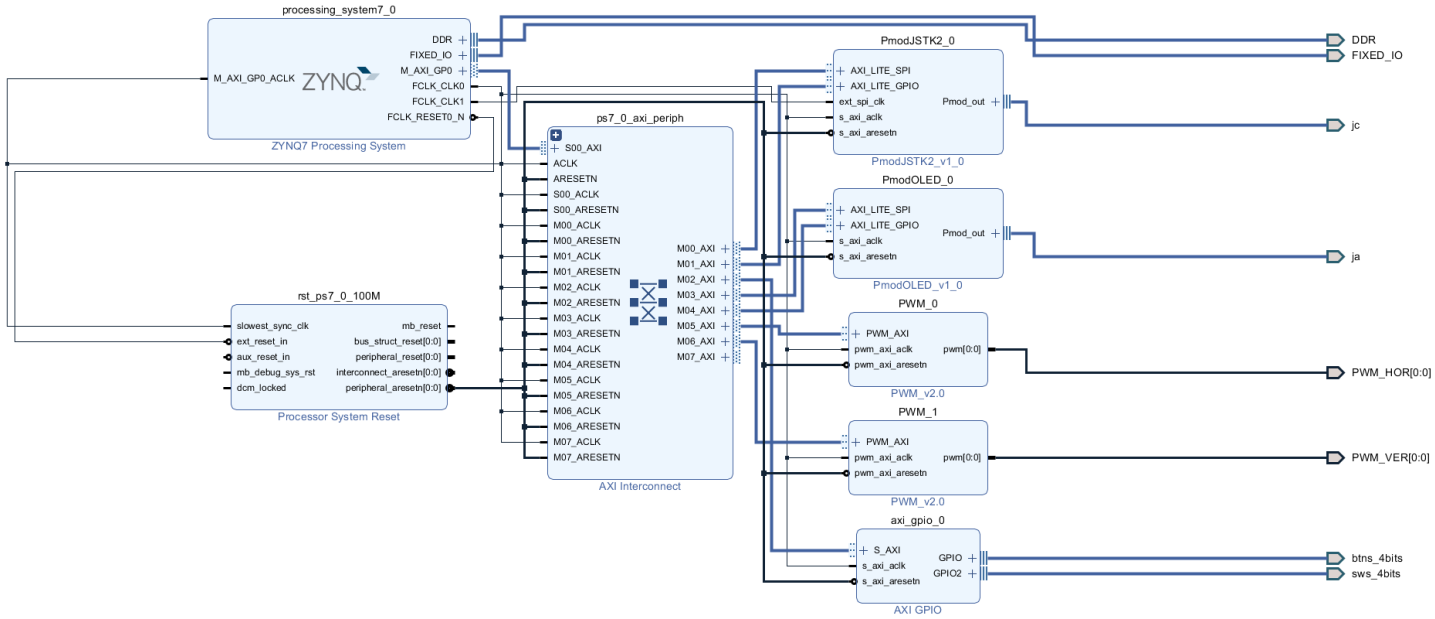

Fig. 1 IP Block Design

```
#define PWM_PAN XPAR_PWM_0_PWM_AXI_BASEADDR
#define PWM_TILT XPAR_PWM_1_PWM_AXI_BASEADDR
#define GPIO_INPUTS XPAR_AXI_GPIO_0_DEVICE_ID
#define BTN_CHANNEL 1
#define SWS_CHANNEL 2
#define PERIOD 2000000
#define MIN_DUTY 90000
#define MAX_DUTY 223000

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "PWM.h"
#include "xparameters.h"
#include "sleep.h"
```

```c
#include "xgpio.h"
#include "PmodJSTK2.h"
#include "math.h"
#include "PmodOLED.h"

XGpio Gpio;
PmodJSTK2 joystick;
PmodOLED myDevice;

_Bool btn = 0;
int pan_value = (MIN_DUTY + MAX_DUTY)/2;
int tilt_value = (MIN_DUTY + MAX_DUTY)/2;
const u8 orientation = 0x0;
const u8 invert = 0x0;

void initLPT();
void driverLoop();
void servoJoy();
void squarePattern();
void circlePattern();
void linePattern();
void crossPattern();
void reset();

int main()
{
    init_platform();
    initLPT();
    driverLoop();

    cleanup_platform();
    return 0;
}

void initLPT(){
      PWM_Set_Period(PWM_PAN, PERIOD);
      PWM_Set_Period(PWM_TILT, PERIOD);
      PWM_Set_Duty(PWM_PAN, (MIN_DUTY + MAX_DUTY)/2, 0);
      PWM_Set_Duty(PWM_TILT, (MIN_DUTY + MAX_DUTY)/2, 0);
      u8 *pat;

      XGpio_Initialize(&Gpio, GPIO_INPUTS);
      XGpio_SetDataDirection(&Gpio, BTN_CHANNEL, 0xf);
```

```
	XGpio_SetDataDirection(&Gpio, SWS_CHANNEL, 0xf);

	OLED_Begin(&myDevice, XPAR_PMODOLED_0_AXI_LITE_GPIO_BASEADDR,
			XPAR_PMODOLED_0_AXI_LITE_SPI_BASEADDR, orientation, invert);

	pat = OLED_GetStdPattern(0);
	OLED_SetFillPattern(&myDevice, pat);
	OLED_SetCharUpdate(&myDevice, 0);

	JSTK2_begin(
		&joystick,
		XPAR_PMODJSTK2_0_AXI_LITE_SPI_BASEADDR,
		XPAR_PMODJSTK2_0_AXI_LITE_GPIO_BASEADDR
	   );
	JSTK2_setInversion(&joystick, 0, 1);

	PWM_Enable(PWM_PAN);
	PWM_Enable(PWM_TILT);
}

void driverLoop(){
	while(1){
		switch(XGpio_DiscreteRead(&Gpio, SWS_CHANNEL)){
			case 0x0:
				reset();
				break;
			case 0x1:
				servoJoy();
				break;
			case 0x2:
				squarePattern();
				break;
			case 0x3:
				linePattern();
				break;
			case 0x4:
				crossPattern();
				break;
			case 0x5:
				circlePattern();
				break;

			default :
```

```
                    reset();
                    break;
            }
      }
}

void reset(){
      pan_value = (MIN_DUTY + MAX_DUTY)/2;
      tilt_value = (MIN_DUTY + MAX_DUTY)/2;

      OLED_ClearBuffer(&myDevice);
      OLED_SetCursor(&myDevice, 0, 0);
      OLED_PutString(&myDevice, "Idle                        ");
      OLED_Update(&myDevice);

      PWM_Set_Duty(PWM_PAN, pan_value, 0); // Sets Duty for PWM 0 ip connected
to pan servo
      PWM_Set_Duty(PWM_TILT, tilt_value, 0); // Sets Duty for PWM 1 ip connected
to tilt servo
      PWM_Enable(PWM_PAN);
      PWM_Enable(PWM_TILT);
      usleep(20000);
}

void servoJoy(){
      JSTK2_Position position;
      position = JSTK2_getPosition(&joystick);

      printf("X: %d, Y: %d\n\r", position.XData, position.YData);
      OLED_ClearBuffer(&myDevice);
      OLED_SetCursor(&myDevice, 0, 0);
      OLED_PutString(&myDevice, "Joystick Control           ");
      OLED_Update(&myDevice);

      pan_value += 60 * (position.XData - 128);
      tilt_value += 60 * (position.YData - 128);

      if(pan_value > MAX_DUTY) pan_value = MAX_DUTY;
      if(pan_value < MIN_DUTY) pan_value = MIN_DUTY;
      if(tilt_value > MAX_DUTY) tilt_value = MAX_DUTY;
      if(tilt_value < MIN_DUTY) tilt_value = MIN_DUTY;

      PWM_Set_Duty(PWM_PAN, pan_value, 0);
```

```
	PWM_Set_Duty(PWM_TILT, tilt_value, 0);

	usleep(20000);
}

void squarePattern(){
	int sqMin = MIN_DUTY + 2*(MAX_DUTY-MIN_DUTY)/8;
	int sqMax = MIN_DUTY + 5*(MAX_DUTY-MIN_DUTY)/8;
	pan_value = sqMin;
	tilt_value = sqMin;

	OLED_ClearBuffer(&myDevice);
	OLED_SetCursor(&myDevice, 0, 0);
	OLED_PutString(&myDevice, "Square Pattern           ");
	OLED_Update(&myDevice);

	PWM_Set_Duty(PWM_PAN, pan_value, 0);
	PWM_Set_Duty(PWM_TILT, tilt_value, 0);

	for(int i = sqMin; i < sqMax; i += 16000){
		pan_value = i;
		PWM_Set_Duty(PWM_PAN, pan_value, 0);
		usleep(20000);
	}

	for(int i = sqMin; i < sqMax; i += 16000){
		tilt_value = i;
		PWM_Set_Duty(PWM_TILT, tilt_value, 0);
		usleep(20000);
	}

	for(int i = sqMax; i > sqMin; i -= 16000){
		pan_value = i;
		PWM_Set_Duty(PWM_PAN, pan_value, 0);
		usleep(20000);
	}

	for(int i = sqMax; i > sqMin; i -= 16000){
		tilt_value = i;
		PWM_Set_Duty(PWM_TILT, tilt_value, 0);
		usleep(20000);
	}
}
```

```
void circlePattern(){
	int sqMin = MIN_DUTY + 2*(MAX_DUTY-MIN_DUTY)/8;
	int sqMax = MIN_DUTY + 5*(MAX_DUTY-MIN_DUTY)/8;
	pan_value = sqMin;
	tilt_value = sqMin;

	OLED_ClearBuffer(&myDevice);
	OLED_SetCursor(&myDevice, 0, 0);
	OLED_PutString(&myDevice, "Circle Pattern           ");
	OLED_Update(&myDevice);

	PWM_Set_Duty(PWM_PAN, pan_value, 0);
	PWM_Set_Duty(PWM_TILT, tilt_value, 0);

	for(int i = sqMin; i < sqMax; i += 32000){
		pan_value = i;
		PWM_Set_Duty(PWM_PAN, pan_value, 0);
		usleep(20000);
	}

	for(int i = sqMin; i < sqMax; i += 32000){
		tilt_value = i;
		PWM_Set_Duty(PWM_TILT, tilt_value, 0);
		usleep(20000);
	}

	for(int i = sqMax; i > sqMin; i -= 32000){
		pan_value = i;
		PWM_Set_Duty(PWM_PAN, pan_value, 0);
		usleep(20000);
	}

	for(int i = sqMax; i > sqMin; i -= 32000){
		tilt_value = i;
		PWM_Set_Duty(PWM_TILT, tilt_value, 0);
		usleep(20000);
	}
}

void linePattern(){
	int sqMin = MIN_DUTY + 2*(MAX_DUTY-MIN_DUTY)/8;
	int sqMax = MIN_DUTY + 5*(MAX_DUTY-MIN_DUTY)/8;
```

```
	OLED_ClearBuffer(&myDevice);
	OLED_SetCursor(&myDevice, 0, 0);
	OLED_PutString(&myDevice, "Line Pattern           ");
	OLED_Update(&myDevice);

	PWM_Set_Duty(PWM_PAN, sqMin, 0);
	PWM_Set_Duty(PWM_TILT, (MIN_DUTY + MAX_DUTY)/2, 0);

	for(int i = sqMin; i < sqMax; i += 8000){
		pan_value = i;
		PWM_Set_Duty(PWM_PAN, pan_value, 0);
		usleep(20000);
	}

	for(int i = sqMax; i > sqMin; i -= 8000){
		pan_value = i;
		PWM_Set_Duty(PWM_PAN, pan_value, 0);
		usleep(20000);
	}
}

void crossPattern(){
	int sqMin = MIN_DUTY + 2*(MAX_DUTY-MIN_DUTY)/8;
	int sqMax = MIN_DUTY + 5*(MAX_DUTY-MIN_DUTY)/8;
	int halfWay = (sqMin + sqMax)/2;
	pan_value = sqMin;
	tilt_value = sqMin;

	OLED_ClearBuffer(&myDevice);
	OLED_SetCursor(&myDevice, 0, 0);
	OLED_PutString(&myDevice, "Cross Pattern           ");
	OLED_Update(&myDevice);

	PWM_Set_Duty(PWM_PAN, pan_value, 0);
	PWM_Set_Duty(PWM_TILT, halfWay, 0);

	for(int i = sqMin; i < halfWay; i += 8000){
		pan_value = i;
		if(pan_value > halfWay) pan_value = halfWay;
		PWM_Set_Duty(PWM_PAN, pan_value, 0);
		usleep(20000);
	}
```

```
for(int i = halfWay; i < sqMax; i += 8000){
      tilt_value = i;
      if(tilt_value > sqMax) pan_value = sqMax;
      PWM_Set_Duty(PWM_TILT, tilt_value, 0);
      usleep(20000);
}

for(int i = sqMax; i > halfWay; i -= 8000){
      tilt_value = i;
      if(tilt_value < halfWay) pan_value = halfWay;
      PWM_Set_Duty(PWM_TILT, tilt_value, 0);
      usleep(20000);
}

for(int i = halfWay; i < sqMax; i += 8000){
      pan_value = i;
      if(tilt_value > sqMax) pan_value = sqMax;
      PWM_Set_Duty(PWM_PAN, pan_value, 0);
      usleep(20000);
}

for(int i = sqMax; i > halfWay; i -= 8000){
      pan_value = i;
      if(tilt_value < halfWay) pan_value = halfWay;
      PWM_Set_Duty(PWM_PAN, pan_value, 0);
      usleep(20000);
}

for(int i = halfWay; i > sqMin; i -= 8000){
      tilt_value = i;
      if(tilt_value < sqMin) pan_value = sqMin;
      PWM_Set_Duty(PWM_TILT, tilt_value, 0);
      usleep(20000);
}

for(int i = sqMin; i < halfWay; i += 8000){
      tilt_value = i;
      if(tilt_value > halfWay) pan_value = halfWay;
      PWM_Set_Duty(PWM_TILT, tilt_value, 0);
      usleep(20000);
}
```

```
        for(int i = halfWay; i > sqMin; i -= 8000){
            pan_value = i;
            if(tilt_value < sqMin) pan_value = sqMin;
            PWM_Set_Duty(PWM_PAN, pan_value, 0);
            usleep(20000);
        }
}
```

Fig. 2 main.c

Video is in the zipped folder.

Analysis

For the Analysis I will be explaining what each function is doing. The main function will call 2 user-made functions called initLPT() and driverLoop(). initLPT() is the function that initializes all of the PMOD devices and PWM modules. It also initializes the GPIO and sets its data direction. driverLoop() is where the main loop is and will run until you turn off the Zybo. The while loop has a case statement where the condition is the value of the switches. At case 0x0 the reset() function is called. The reset function will move the servos to their default position and do nothing else. At case 0x1 the servoJoy() function will be called. The servoJoy() function will read the current position of the joystick and then set the servo's position based on that. At case 0x2, 0x3, 0x4, and 0x5 the joy functions squarePattern(), linePattern(), crossPattern(), or circlePattern(). These functions are responsible for moving the servo in a certain pattern. They use for loops and delays to try to make the pattern smooth and legible. The logic of moving the servos based on position value can be seen in **Fig. 3.** The code to read from the joystick and print to the OLED is shown below in **Fig. 4.** Code to update the Duty can be seen in **Fig. 5**.

```
pan_value += 60 * (position.XData - 128);
tilt_value += 60 * (position.YData - 128);

if(pan_value > MAX_DUTY) pan_value = MAX_DUTY;
if(pan_value < MIN_DUTY) pan_value = MIN_DUTY;
if(tilt_value > MAX_DUTY) tilt_value = MAX_DUTY;
if(tilt_value < MIN_DUTY) tilt_value = MIN_DUTY;
```

Fig. 3 Joystick logic

```
JSTK2_Position position;
position = JSTK2_getPosition(&joystick);

printf("X: %d, Y: %d\n\r", position.XData, position.YData);
OLED_ClearBuffer(&myDevice);
OLED_SetCursor(&myDevice, 0, 0);
OLED_PutString(&myDevice, "Joystick Control          ");
OLED_Update(&myDevice);
```

Fig. 4 OLED and Joystick reading

```
PWM_Set_Duty(PWM_PAN, pan_value, 0);
PWM_Set_Duty(PWM_TILT, tilt_value, 0);
```

Fig. 5 Updating the Duty

Conclusion

In conclusion, I had fun working with this project and I was happy I was able to make my design work. I used both hardware and software design methods I learned in this class for this project. With the use of Vivado I was able to use IPs designed by Digilent in order to program the Hardware logic required. The IPs I used came with drivers that had examples so it was not hard at all to implement the IPs in software. I learned about PWM signals, controlling servos, and how to modify constraints in order to add my own ports in the block design diagram.