Course Name: Topics in ECE - Embedded Systems

Course Number and Section: **14:332:493:10**

Experiment: [Final Project]

Lab Instructor: Professor Phillip Southard

Date Performed: -

Date Submitted: December 16, 2019

Submitted by: Antoni Chrobot 183008289

Final Project – Guitar Tuner

Github:

Purpose

The purpose of my final project is to create a guitar tuner. My guitar tuner will play 7 different tones, based on user input, through the LINE OUT jack on the Zybo board. The 7 different tones will be played at frequencies which correspond to the standard tuning of a guitar (E A D G B e, as well as drop D). A user will be able to select their desired frequency by setting the slide switches on the board to different values. The guitar tuner will also feature a seven segment display that will display the current string value that is selected.

Theory of Operation

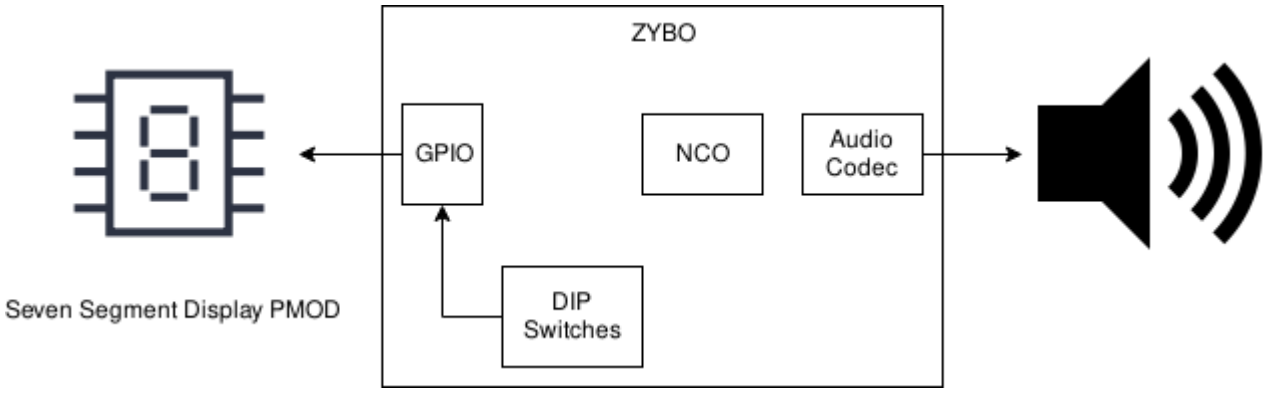
Tone generation

For the tone generation, I have decided to take a simple approach and re-use what I can from lab 3 (lab 5 in the Zynq book tutorials). In that lab we explored the use of custom made IP cores to interact with the audio codec. I generated tones by simply writing to the audio codec at periods calculated to produce a desired frequency. The frequency of the tone will be set by user using the switches on the Zybo board.

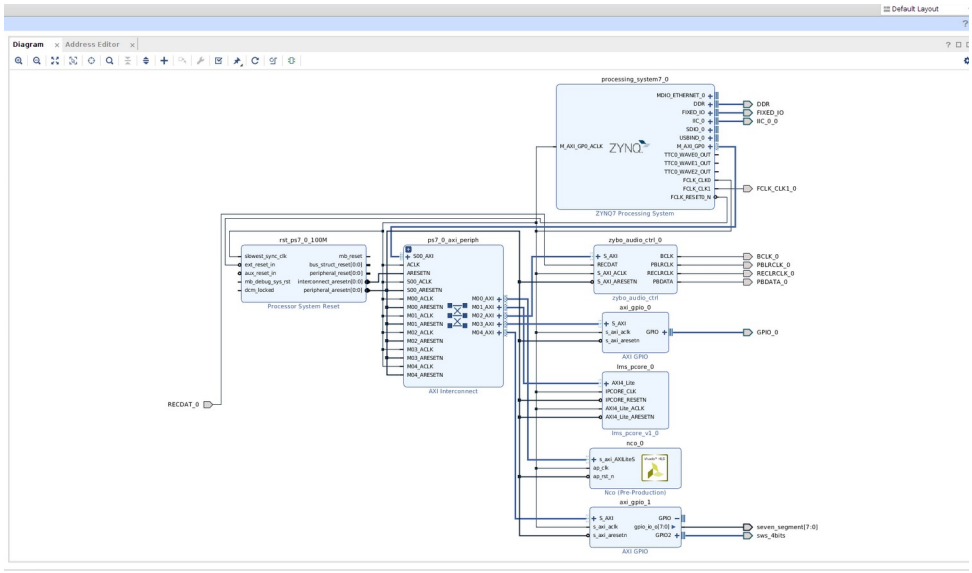
Seven Segment Display

The seven segment display is a rather simple device. It contains seven LED segments. The seven segment display has 12 pins, two for power, two for ground, and the remaining eight pins are used for turning on or off each of the seven segments. To drive the seven segment display, I used a GPIO IP with 8 outputs. By writing the to the GPIO peripheral, I can easily display letters to the seven segment display.

System Block Diagram



Principal IP Integrator Block Diagram



Design

Below is my main function. It first initializes the GPIO and NCO peripheral. Then it enters two for loops which illuminate the segments on the SSD pmod one at a time. This is for trouble shooting to make sure each segment works properly. Finally, we enter an infinite while loop. Inside the loop, we check the value of the switches on the board. Based on the values of the switch, we write to the SSD pmod and generate a tone.

```
int main(void)
{
	xil_printf("Entering Main\r\n");
	//Configure the IIC data structure
	IicConfig(XPAR_XIICPS_0_DEVICE_ID);

	//Configure the Audio Codec's PLL
	AudioPllConfig();

	xil_printf("SSM2603 configured\n\r");


	/* Initialise GPIO and NCO peripherals */
	gpio_init();
	nco_init(&Nco);

	u32 switch_inputs;
	u32 prev_input;
	u16 delay_time = 0xFFFF;

	for(int i=0; i<8; i++)
	{
		XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL,1<<i);
		usleep(250000);
	}
	for(int i=0; i<8; i++)
		{
			XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL,(1<<i)|0x80 );
			usleep(250000);
		}


	while(1)
	{
		switch_inputs = XGpio_DiscreteRead(&Gpio, SWITCH_CHANNEL);

		if(switch_inputs != prev_input)
		{
			switch(switch_inputs)
			{
				case 0x0: delay_time = 0xFFFF;
					XGpio_DiscreteWrite(&Gpio,
SSD_CHANNEL,0x7F);
					break;
				case 0x1: delay_time = E_delay;
					XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL, E);
					break;
				case 0x2: delay_time = A_delay;
					XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL, A);
```

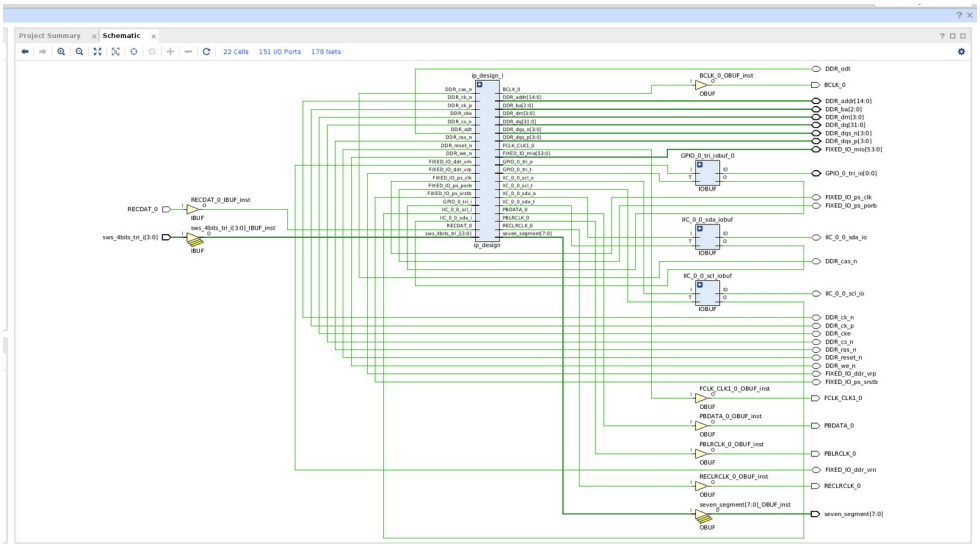
```
                                        break;
                                case 0x3: delay_time = D_delay;
                                        XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL, D);
                                        break;
                                case 0x4: delay_time = G_delay;
                                        XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL, G);
                                        break;
                                case 0x5: delay_time = B_delay;
                                        XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL, B);
                                        break;
                                case 0x6: delay_time = e_delay;
                                        XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL, e);
                                        break;
                                case 0x7: delay_time = drop_D_delay;
                                        XGpio_DiscreteWrite(&Gpio, SSD_CHANNEL, D);
                                        break;
                                default: delay_time = 0xFFFF;
                                        XGpio_DiscreteWrite(&Gpio,
SSD_CHANNEL,0x7F);
                                        break;


                        }

                }

                if(delay_time != 0xFFFF)
                {

                        Xil_Out32(I2S_DATA_TX_L_REG, 0x0000FFFF);
                        Xil_Out32(I2S_DATA_TX_R_REG, 0x0000FFFF);
                        usleep(delay_time>>1);
                        Xil_Out32(I2S_DATA_TX_L_REG, 0x00007FFF);
                        Xil_Out32(I2S_DATA_TX_R_REG, 0x00007FFF);
                        usleep(delay_time>>1);
                        Xil_Out32(I2S_DATA_TX_L_REG, 0x0);
                        Xil_Out32(I2S_DATA_TX_R_REG, 0x0);
                        usleep(delay_time>>1);
                        Xil_Out32(I2S_DATA_TX_L_REG, 0x00007FFF);
                        Xil_Out32(I2S_DATA_TX_R_REG, 0x00007FFF);
                        usleep(delay_time>>1);
                }

        }


        xil_printf("GPIO and NCO peripheral configured\r\n");

        /* Display interactive menu interface via terminal */
        menu();
    return 1;
}
```

Test

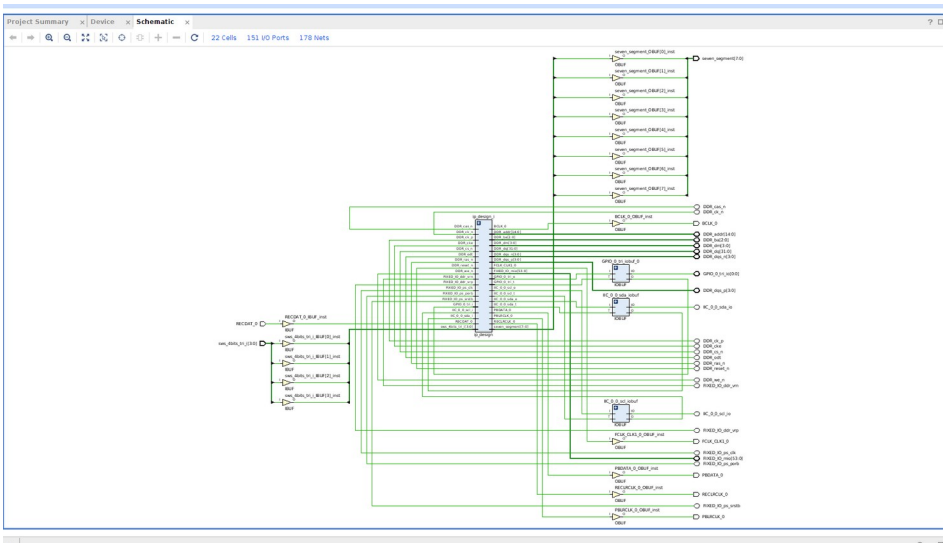
Testing this projected consisted of visually inspecting the seven segment display pmod to make sure it was displaying the proper values, and listening to the audio output at the LINE OUT jack using headphones. To make sure the proper frequency was being played, I borrowed a friends headphones and played a tone from an online tone generator, and compared that to my headphones which were connected to the LINE OUT port on the board. The tones sounded identical.

Implementation

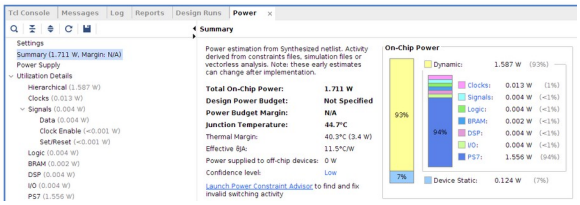
Elaboration schematic



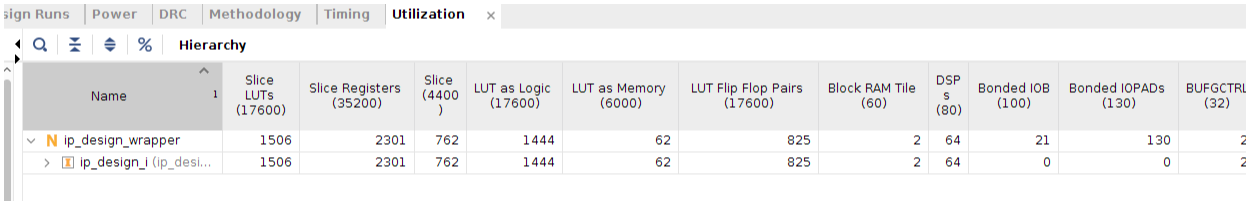
Synthesis schematic



On-chip power graphs



Utilization table



Name	Slice LUTs (17600)	Slice Registers (35200)	Slice (4400)	LUT as Logic (17600)	LUT as Memory (6000)	LUT Flip Flop Pairs (17600)	Block RAM Tile (60)	DSPs (80)	Bonded IOB (100)	Bonded IOPADs (130)	BUFGCTRL (32)
ip_design_wrapper	1506	2301	762	1444	62	825	2	64	21	130	2
ip_design_i (ip_desi...	1506	2301	762	1444	62	825	2	64	0	0	2

Discussion

This project was at times really frustrating, but in the end very rewarding. It was a challenge to implement a completed project. My final result is much different that my original plan. I had originally planned to use a microphone pmod and perform an FFT on the signal, and then display to the user how far off the recorded sound's frequency was from the nearest guitar string frequency. After learning there was no microphone pmod available, I instead opted to the use the MIC IN jack through the audio codec on the board. I had made some progress with this approach; I was able to read samples from the audio codec. Unfortunately I was unable to interpret those results, to see if I was getting real audio samples or just electrical noise. I also have a lack of experience in digital signal processing and decided that doing an FFT on signals from the audio codec was not the simplest solution to my project.

At this point I decided to make a guitar tuner that plays tones at frequencies of the guitar strings. I already had confidence with reading/writing to/from the audio codec, so I thought this approach would be more achievable. I tested this approach by writing a non-zero value to the audio codec, delaying, writing a zero value to the audio codec, and delaying again. When I plugged in my headphones to the LINE OUT jack, I heard a tone! Having established that I can generate tones, I mapped out how long I would have to delay for each guitar string, and stored the values in a header file.

After testing the tone generation and the SSD pmod, I then put the two pieces together. Inside my main function I created an infinite while loop. Inside the while loop I would check the values of the on-board switches. If they had changed from the previous value, I would set a variable 'delay_time' based on the values of the switches. I also wrote to the SSD pmod GPIO so that it would display the given note that was being played.