```c
/* Author: Adam Grabowski
 * Course: ECE 3829
 * Project: Lab 4
 * Description: Project for Lab 4
 * Functionality:
 * 1 - reading switch and button inputs
 * 2 - output corresponding LED based on switch data
 * 3 - output tone to the AMP2 based on switch and button data
 * 4 - output note to seven segment display based on switch and button data
 * 5 - output tune to the AMP2 on power up or after reset
 */

// Header Inclusions
/* xparameters.h set parameters names
 like XPAR_AXI_GPIO_0_DEVICE_ID that are referenced in you code
 each hardware module as a section in this file.
 */
#include "xparameters.h"

/* each hardware module type as a set commands you can use to
 * configure and access it. xgpio.h defines API commands for your gpio modules
 */
#include "xgpio.h"

/* this defines the recommend types like u32 */
#include "xil_types.h"
#include "xil_printf.h"
#include "xstatus.h"
#include "sleep.h"
#include "xtmrctr.h"

void calc_note(double *freq, u32 *cathode_data, u32 sw_data, u32 btn_data);
void check_switches(u32 *sw_data, u32 *sw_data_old, u32 *sw_changes);
void check_buttons(u32 *sw_data, u32 *sw_data_old, u32 *btn_changes);
void update_LEDs(u32 led_data);
void update_cathode(u32 cathode_data);
void update_anode(u32 anode_data);
void update_amp2(u32 *amp2_data, u32 target_count, u32 *last_count);

// Block Design Details
/* Timer device ID
 */
#define TMRCTR_DEVICE_ID XPAR_TMRCTR_0_DEVICE_ID
#define TIMER_COUNTER_0 0

/* LED are assigned to GPIO (CH 1) GPIO_0 Device
 * DIP Switches are assigned to GPIO2 (CH 2) GPIO_0 Device
 */
#define GPIO0_ID XPAR_GPIO_0_DEVICE_ID
#define GPIO0_LED_CH 1
#define GPIO0_SW_CH 2

// 16-bits of LED outputs (not tristated)
#define GPIO0_LED_TRI 0x00000000
#define GPIO0_LED_MASK 0x0000FFFF
```

```
// 16-bits SW inputs (tristated)
#define GPIO0_SW_TRI 0x0000FFFF
#define GPIO0_SW_MASK 0x0000FFFF

/*  7-SEG Anodes are assigned to GPIO (CH 1) GPIO_1 Device
 *  7-SEG Cathodes are assigned to GPIO (CH 2) GPIO_1 Device
 */
#define GPIO1_ID XPAR_GPIO_1_DEVICE_ID
#define GPIO1_ANODE_CH 1
#define GPIO1_CATHODE_CH 2

// 4-bits of anode outputs (not tristated)
#define GPIO1_ANODE_TRI 0x00000000
#define GPIO1_ANODE_MASK 0x0000000F

// 8-bits of cathode outputs (not tristated)
#define GPIO1_CATHODE_TRI 0x00000000
#define GPIO1_CATHODE_MASK 0x000000FF

// Push buttons are assigned to GPIO (CH_1) GPIO_2 Device
#define GPIO2_ID XPAR_GPIO_2_DEVICE_ID
#define GPIO2_BTN_CH 1

// 4-bits of push button (not tristated)
#define GPIO2_BTN_TRI 0x00000000
#define GPIO2_BTN_MASK 0x0000000F

// AMP2 pins are assigned to GPIO (CH1 1) GPIO_3 device
#define GPIO3_ID XPAR_GPIO_3_DEVICE_ID
#define GPIO3_AMP2_CH 1
#define GPIO3_AMP2_TRI 0xFFFFFFF4
#define GPIO3_AMP2_MASK 0x00000001

// Note frequency encodings
#define FREQ_C3 130.81
#define FREQ_D3 146.83
#define FREQ_E3 164.81
#define FREQ_F3 174.61
#define FREQ_G3 196
#define FREQ_A3 220
#define FREQ_B3 246.94
#define FREQ_C4 261.63
#define FREQ_D4 293.66
#define FREQ_E4 329.63
#define FREQ_F4 349.23
#define FREQ_G4 392
#define FREQ_A4 440
#define FREQ_B4 493.88
#define FREQ_C5 523.25
#define FREQ_D5 587.33

// Anode encoding
#define ANODE_D 0xE
```

```
// Switch and button selection masks
#define SW_SEL_MASK 0b11
#define BTN_SEL_MASK 0b1111

// Cathode encodings
#define CATHODE_OFF 0x40
#define CATHODE_A 0x8
#define CATHODE_B 0x3
#define CATHODE_C 0x46
#define CATHODE_D 0x21
#define CATHODE_E 0x6
#define CATHODE_F 0xE
#define CATHODE_G 0x10

// Power-up note lengths
#define TUNE_LEN 250000000
#define NOTE_1 50000000
#define NOTE_2 100000000
#define NOTE_3 150000000
#define NOTE_4 200000000

// Timer Device instance
XTmrCtr TimerCounter;

// GPIO Driver Device
XGpio device0;
XGpio device1;
XGpio device2;
XGpio device3;

// IP Tutorial  Main
int main() {
	u32 sw_data = 0;
	u32 sw_data_old = 0;
	// bit[3] = SHUTDOWN_L and bit[1] = GAIN, bit[0] = Audio Input
	u32 amp2_data = 0x8;
	u32 target_count = 0xffffffff;
	u32 last_count = 0;
	u32 sw_changes = 0;
	u32 btn_data = 0;
	u32 btn_data_old = 0;
	u32 btn_changes = 0;
	double freq = 0.0;
	u32 cathode_data = 0;
	u32 current_count = 0;
	XStatus status;

	// Initialize timer
	status = XTmrCtr_Initialize(&TimerCounter, XPAR_TMRCTR_0_DEVICE_ID);
	if (status != XST_SUCCESS) {
		xil_printf("Initialization Timer failed\n\r");
		return 1;
	}

	// Make sure the timer is working
```

```
status = XTmrCtr_SelfTest(&TimerCounter, TIMER_COUNTER_0);
if (status != XST_SUCCESS) {
	xil_printf("Initialization Timer failed\n\r");
	return 1;
}

// Configure the timer to Autoreload
XTmrCtr_SetOptions(&TimerCounter, TIMER_COUNTER_0, XTC_AUTO_RELOAD_OPTION);

// Initialize your timer values
// Start your timer
XTmrCtr_Start(&TimerCounter, TIMER_COUNTER_0);

// Initialize the GPIO devices
status = XGpio_Initialize(&device0, GPIO0_ID);
if (status != XST_SUCCESS) {
	xil_printf("Initialization GPIO_0 failed\n\r");
	return 1;
}
status = XGpio_Initialize(&device1, GPIO1_ID);
if (status != XST_SUCCESS) {
	xil_printf("Initialization GPIO_1 failed\n\r");
	return 1;
}
status = XGpio_Initialize(&device2, GPIO2_ID);
if (status != XST_SUCCESS) {
	xil_printf("Initialization GPIO_2 failed\n\r");
	return 1;
}
status = XGpio_Initialize(&device3, GPIO3_ID);
if (status != XST_SUCCESS) {
	xil_printf("Initialization GPIO_3 failed\n\r");
	return 1;
}

// Set directions for data ports tristates, '1' for input, '0' for output
XGpio_SetDataDirection(&device0, GPIO0_LED_CH, GPIO0_LED_TRI);
XGpio_SetDataDirection(&device0, GPIO0_SW_CH, GPIO0_SW_TRI);
XGpio_SetDataDirection(&device1, GPIO1_ANODE_CH, GPIO1_ANODE_TRI);
XGpio_SetDataDirection(&device1, GPIO1_CATHODE_CH, GPIO1_CATHODE_TRI);
XGpio_SetDataDirection(&device2, GPIO2_BTN_CH, GPIO2_BTN_TRI);
XGpio_SetDataDirection(&device3, GPIO3_AMP2_CH, GPIO3_AMP2_TRI);

xil_printf("Demo initialized successfully\n\r");

XGpio_DiscreteWrite(&device3, GPIO3_AMP2_CH, amp2_data);

// Show display D (right most display) and set cathode to OFF
update_anode(ANODE_D);
update_cathode(CATHODE_OFF);

// Play tune on power-up or reset
while (current_count < TUNE_LEN) {
	current_count = XTmrCtr_GetValue(&TimerCounter, TIMER_COUNTER_0);
	if (current_count < NOTE_1) {
```

```
                freq = 130.81;
            } else if (current_count < NOTE_2) {
                freq = 196;
            } else if (current_count < NOTE_3) {
                freq = 293.66;
            } else if (current_count < NOTE_4) {
                freq = 440;
            } else {
                freq = 587.33;
            }
            target_count = (1.0/(2.0*freq*10e-9));
            update_amp2(&amp2_data, target_count, &last_count);
        }

        // This loop checks for changes in the input switches and buttons
        // If they changed it updates the outputs to match the switch and button
values
        // target_count = (period of sound)/(2*10nsec)), 10nsec is the processor clock
frequency
        // Example count is middle C (C4) = 191110 count (261.62 Hz)
        while (1) {
            check_switches(&sw_data, &sw_data_old, &sw_changes);
            if (sw_changes) update_LEDs(sw_data);
            check_buttons(&btn_data, &btn_data_old, &btn_changes);
            calc_note(&freq, &cathode_data, sw_data, btn_data);
            if (freq == 0) target_count = 0;
            else target_count = (1.0/(2.0*freq*10e-9));
            if (target_count > 0) update_amp2(&amp2_data, target_count,
&last_count);
            update_cathode(cathode_data);
        }
}

// Calculates the target count and cathode data based on the input switches and
buttons
void calc_note(double *freq, u32 *cathode_data, u32 sw_data, u32 btn_data) {
        u32 sw_sel = sw_data & SW_SEL_MASK;
        u32 btn_sel = btn_data & BTN_SEL_MASK;
        switch (sw_sel) {
        case 0b0:
            switch (btn_sel) {
            case 0b0000:
                *freq = 0;
                *cathode_data = CATHODE_OFF;
                break;
            case 0b0001:
                *freq = FREQ_C3;
                *cathode_data = CATHODE_C;
                break;
            case 0b0010:
                *freq = FREQ_D3;
                *cathode_data = CATHODE_D;
                break;
            case 0b0011:
                *freq = FREQ_E3;
```

```
                *cathode_data = CATHODE_E;
                break;
        case 0b0100:
                *freq = FREQ_F3;
                *cathode_data = CATHODE_F;
                break;
        case 0b0101:
                *freq = FREQ_G3;
                *cathode_data = CATHODE_G;
                break;
        case 0b0110:
                *freq = FREQ_A3;
                *cathode_data = CATHODE_A;
                break;
        case 0b0111:
                *freq = FREQ_B3;
                *cathode_data = CATHODE_B;
                break;
        }
        break;
case 0b1:
        switch (btn_sel) {
        case 0b0000:
                *freq = 0;
                *cathode_data = CATHODE_OFF;
                break;
        case 0b0001:
                *freq = FREQ_C4;
                *cathode_data = CATHODE_C;
                break;
        case 0b0010:
                *freq = FREQ_D4;
                *cathode_data = CATHODE_D;
                break;
        case 0b0011:
                *freq = FREQ_E4;
                *cathode_data = CATHODE_E;
                break;
        case 0b0100:
                *freq = FREQ_F4;
                *cathode_data = CATHODE_F;
                break;
        case 0b0101:
                *freq = FREQ_G4;
                *cathode_data = CATHODE_G;
                break;
        case 0b0110:
                *freq = FREQ_A4;
                *cathode_data = CATHODE_A;
                break;
        case 0b0111:
                *freq = FREQ_B4;
                *cathode_data = CATHODE_B;
                break;
        }
```

```
			break;
		case 0b10:
			switch (btn_sel) {
			case 0b0000:
				*freq = 0;
				*cathode_data = CATHODE_OFF;
				break;
			case 0b0001:
				*freq = FREQ_C5;
				*cathode_data = CATHODE_C;
				break;
			case 0b0010:
				*freq = FREQ_D5;
				*cathode_data = CATHODE_D;
				break;
			}
			break;
		}
}

// Reads the value of the input switches and outputs if there were changes from last
time
void check_switches(u32 *sw_data, u32 *sw_data_old, u32 *sw_changes) {
		*sw_data = XGpio_DiscreteRead(&device0, GPIO0_SW_CH);
		*sw_data &= GPIO0_SW_MASK;
		*sw_changes = 0;
		if (*sw_data != *sw_data_old) {
			// When any bswitch is toggled, the LED values are updated and report
the state over UART
			*sw_changes = *sw_data ^ *sw_data_old;
			*sw_data_old = *sw_data;
		}
}

// Reads the value of the input buttons and outputs if there were changes from last
time
void check_buttons(u32 *btn_data, u32 *btn_data_old, u32 *btn_changes) {
		*btn_data = XGpio_DiscreteRead(&device2, GPIO2_BTN_CH);
		*btn_data &= GPIO2_BTN_MASK;
		*btn_changes = 0;
		if (*btn_data != *btn_data_old) {
			// When any bswitch is toggled, the LED values are updated and report
the state over UART
			*btn_changes = *btn_data ^ *btn_data_old;
			*btn_data_old = *btn_data;
		}
}

// Writes the value of led_data to the LED pins
void update_LEDs(u32 led_data) {
		led_data = (led_data) & GPIO0_LED_MASK;
		XGpio_DiscreteWrite(&device0, GPIO0_LED_CH, led_data);
}

// Writes the value of cathode_data to the cathode pins
```

```
void update_cathode(u32 cathode_data) {
	cathode_data = (cathode_data) & GPIO1_CATHODE_MASK;
	XGpio_DiscreteWrite(&device1, GPIO1_CATHODE_CH, cathode_data);
}

// Writes the value of anode_data to the anode pins
void update_anode(u32 anode_data) {
	anode_data = (anode_data) & GPIO1_ANODE_MASK;
	XGpio_DiscreteWrite(&device1, GPIO1_ANODE_CH, anode_data);
}

// If the current count is - last_count > target_count toggle the amp2 output
void update_amp2(u32 *amp2_data, u32 target_count, u32 *last_count) {
	u32 current_count = XTmrCtr_GetValue(&TimerCounter, TIMER_COUNTER_0);
	if ((current_count - *last_count) > target_count) {
		// Toggling the LSB of amp2 data
		*amp2_data = ((*amp2_data & 0x01) == 0) ? (*amp2_data | 0x1) :
(*amp2_data & 0xe);
		XGpio_DiscreteWrite(&device3, GPIO3_AMP2_CH, *amp2_data );
		*last_count = current_count;
	}
}
```