[Ricardo Stefano Reyna]
[5/August/2016]
[2101-0521]

FINAL PROJECT – Report

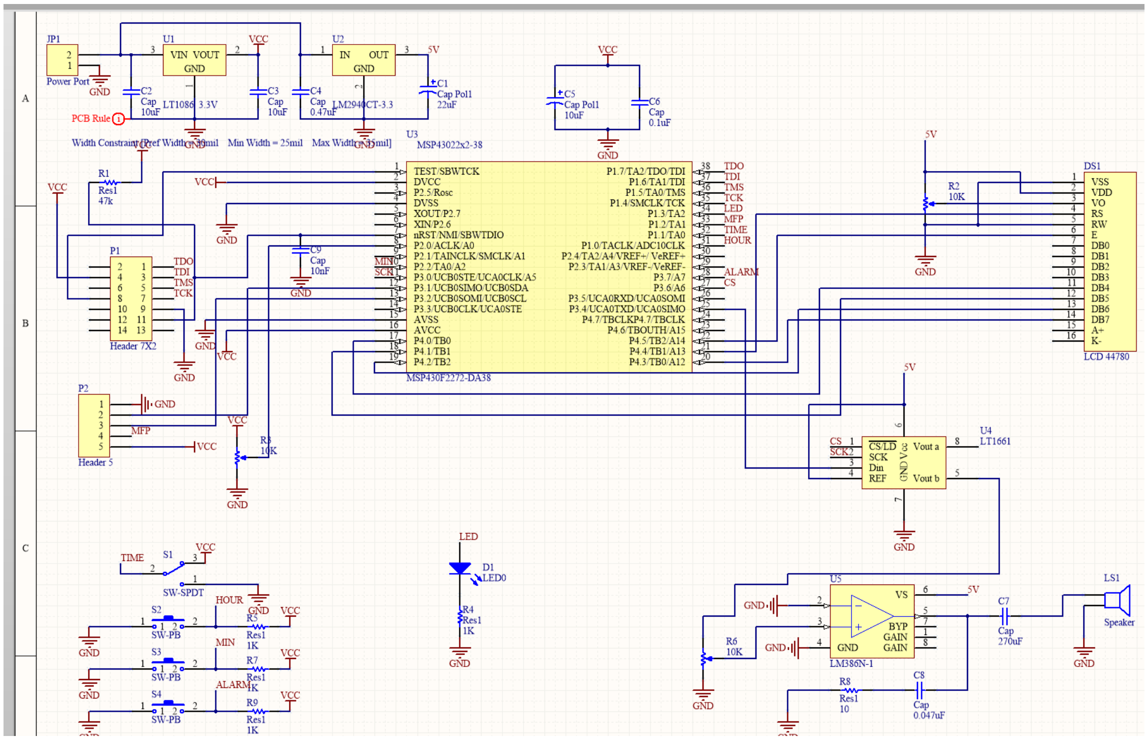
Introduction

This project was a combination of all my previous modules plus the pre-amplifier module done by the EE side of the class as an above and beyond. The project consists of an alarm clock where you can set the time and alarm, based on the LED being on it tells you if it's on or not. Once it turns on the microcontroller will output a sine wave through the 8-ohm speaker.

Design

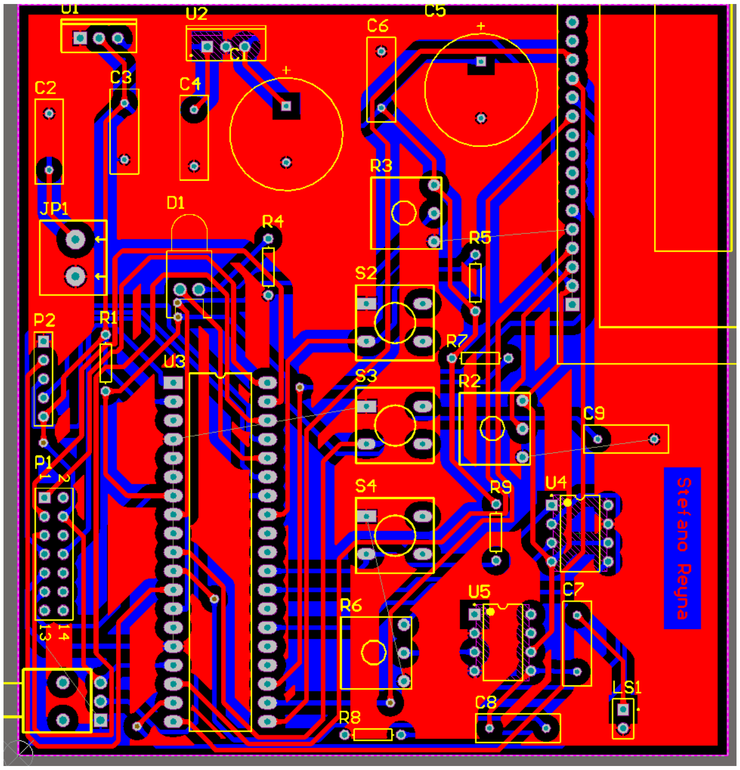
For the design I already had most of it assembled in the breadboard since it was mostly relied on the RTC module which was done the week before the project weeks were due so because of this I already had everything I needed assembled in my breadboard. All I had to add was an LED and the DAC. In order to use the DAC the TA suggested it to combine it with the pre-amplifier module and it would be considered as above and beyond. I started by adding the LED to pin 1.3, and it would go high once the alarm has been set and the alarm button goes low (because the push button is low true). Once done I added the DAC to the same pins as the SPI module (3.0 for clock, 3.4 to data, and 3.6 to CS), and used 5V as the voltage reference. Then I proceeded to build the amplifier which has a gain of 20 where I just followed the diagram given in the data sheet for the LM386. The reason I added this is because I wanted to give a use to my DAC and avoid outputting a simple sine wave out the o-scope. The output of the DAC is connected to the voltage in of logarithmic potentiometer (this has better effects when changing the volume) and the variable pin goes into the input of the amplifier (LM386), while the output goes into the speaker. The last thing I tested were the voltage regulators to see if I can get the LCD and MSP using 9 volts as my direct source. In reality nothing new had to be added other than the LM386 amplifier. Once tested on the breadboard and got everything working perfectly I started writing down my schematic in a sheet of paper to later use it as reference when designing it on Altium.

The following is the end result of the schematic (this is a modification since the previous schematic had a bad connection):



In the top left there's the voltage regulators, under it we have the debugger and RTC header. In the middle is the MSP430, to the right the LCD, and below that the DAC and LM386.

And as for the PCB:



In this case I tried to place the components in a way that mimics the design done in the schematic. This way it would be a lot easier when it comes to soldering, I tried to find a way that when I used auto route it would give preference to the bottom rather than the top layer, but no luck so I decided to just leave both top and bottom automatic.
Once done adding the polypour, making the drills, and gerber files; I submitted the files through the senior website. Once received, scraped, and soldered I had to still fix some connections since some of the traces were routed to places I didn't want to in the first place. After scrapping the traces and re-routing the using wire wrap and adding a few components here and there I managed to get my board working.

Conclusion

Most of the results were as expected, I experienced difficulty during my presentation since my MSP430 chip had gone bad. As a consequence, the data was never being sent properly into the DAC where I needed to get a new chip where it was successful this time around. At the end it inflicted in other minor yet negligible errors with my LCD (adding extra gibberish), but the big scope of everything worked rather well.

Appendix

```
#include <msp430.h>
#include <stdint.h>



/*
 * Author: Stefano Reyna
 * Final Project
 */
#define HOME	0x02	// Home
#define CLEAR	0x01	// Clear screen and CR
#define DOWN		0xC0	// Second line
#define RIGHT	0x14	//Move right

#define		CNTRL_BYTE	0x6F
#define		RTCC_EN_OSC_START	0x80
#define		RTCC_EN_SQWE		0x40
#define		RTCC_MFP_1HZ		0x00

//Address map
#define		RTC_SEC					0x00
```

```
#define         RTC_MIN                         0x01
#define         RTC_HOUR                0x02
#define         RTC_DAY                         0x03
#define         RTC_DATE                0x04
#define         RTC_MONTH               0x05
#define         RTC_YEAR                0x06
#define         RTC_CNTRL               0x07
#define         RTC_CAL                         0x08

#define         RTC_ALARM0_SEC          0x0A
#define         RTC_ALARM0_MIN          0x0B
#define         RTC_ALARM0_HOUR         0x0C
#define         RTC_ALARM0_DAY          0x0D
#define         RTC_ALARM0_DATE         0x0E
#define         RTC_ALARM0_MONTH        0x0F

void lcd_command(char uf_lcd_x);
void lcd_char(char uf_lcd_x);
void lcd_init(void);
void lcd_string(char *str);

void i2c_init();
void rtc_init();
void write_I2C(unsigned int Slave_Add,unsigned int Add, unsigned int Val);
unsigned int read_I2C(unsigned int Slave_Add,unsigned int Add);
void Setup_TX(unsigned int Add);
void Setup_RX(unsigned int Add);

unsigned int map_dec_hex(unsigned int dec);
void write_to_lcd(unsigned int val);
unsigned int adc_div(unsigned int partition, int divs);
void adc_setup(void);
unsigned int adc_sam20(void);
void delay_us(unsigned int us);
void beep(unsigned int note);
void SPI_setup(void);
void SPI_write(unsigned int data);
void output_sound(int freq_value);
void delay(void);

int sine_t[16] = {512,707,873,984,1023,984,873,707,512,316,150,39,0,39,150,316,};
char *days_of_week[7] = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

char uf_lcd_temp;
char uf_lcd_temp2;
char uf_lcd_x;

int RXByteCtr, Res_Flag = 0;   // enables repeated start when 1
unsigned int TXData;
unsigned int *PRxData;
unsigned int *PTxData;
unsigned int TxBuffer[128];
unsigned int RxBuffer;
unsigned int TXByteCtr, RX = 0;

int main(void) {
   WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer
          lcd_init();
          adc_setup();
          rtc_init();
          SPI_setup();

          P1DIR = 0x08;

          volatile unsigned int sec = 0x50;
          sec |= 0x80;
          volatile unsigned int min = 0x39;
          volatile unsigned int hour = 0x14;
          volatile unsigned int day = 0x05;
          volatile unsigned int date = 0x05;
          volatile unsigned int month = 0x08;
          volatile unsigned int year = 0x16;

          volatile unsigned int al_sec = 0x00;
          volatile unsigned int al_min = 0x40;
          volatile unsigned int al_hour = 0x14;
          volatile unsigned int al_day = 0xF4; // The first bit is high so the MFP is high true
          volatile unsigned int al_date = 0x14;
          volatile unsigned int al_month = 0x07;

          volatile unsigned int day_num = 0x00;
          int day_temp = 0;

          //Set alarm
          write_I2C(CNTRL_BYTE, RTC_ALARM0_SEC, al_sec);
          write_I2C(CNTRL_BYTE, RTC_ALARM0_MIN, al_min);
          write_I2C(CNTRL_BYTE, RTC_ALARM0_HOUR, al_hour);
          write_I2C(CNTRL_BYTE, RTC_ALARM0_DAY, al_day);
          write_I2C(CNTRL_BYTE, RTC_ALARM0_DATE, al_date);
          write_I2C(CNTRL_BYTE, RTC_ALARM0_MONTH, al_month);
          write_I2C(CNTRL_BYTE, RTC_CNTRL, 0x10); //turn it on

          //set time
          write_I2C(CNTRL_BYTE, RTC_SEC, sec);
          write_I2C(CNTRL_BYTE, RTC_MIN, min);
          write_I2C(CNTRL_BYTE, RTC_HOUR, hour);
          write_I2C(CNTRL_BYTE, RTC_DAY, day);
          write_I2C(CNTRL_BYTE, RTC_DATE, date);
          write_I2C(CNTRL_BYTE, RTC_MONTH, month);
          write_I2C(CNTRL_BYTE, RTC_YEAR, year);

          //Test code, unnecessary
          write_to_lcd(hour);
          lcd_char(':');
          write_to_lcd(min);
          lcd_char(':');
          sec &= 0x7F; //Get rid of the oscillator bit
          write_to_lcd(sec);

          lcd_command(HOME);

          int flag = 0; //flag that tells me alarm is on
          P1OUT &= 0xF7;
```

```
while(1) {

        int time_pin = P1IN & 0x02;
        //time mode
        if (time_pin) {
                sec = read_I2C(CNTRL_BYTE, RTC_SEC);
                min = read_I2C(CNTRL_BYTE, RTC_MIN);
                hour = read_I2C(CNTRL_BYTE, RTC_HOUR);
                day = read_I2C(CNTRL_BYTE, RTC_DAY);
                date = read_I2C(CNTRL_BYTE, RTC_DATE);
                month = read_I2C(CNTRL_BYTE, RTC_MONTH);
                year = read_I2C(CNTRL_BYTE, RTC_YEAR);

                //Throw it to the LCD
                //time
                write_to_lcd(hour);
                lcd_char(':');
                write_to_lcd(min);
                lcd_char(':');
                sec &= 0x7F;
                write_to_lcd(sec);
                //date
                lcd_command(DOWN);
                write_to_lcd(date);
                lcd_char('/');
                month &= 0x1F; //get rid of leap year
                write_to_lcd(month);
                lcd_char('/');
                write_to_lcd(year);

                lcd_command(RIGHT);

                day_num = read_I2C(CNTRL_BYTE, RTC_DAY);
                day_num &= 0x07;
                //map the day which goes from 1 to 7 as 0 to 6
                lcd_string(days_of_week[--day_num]);

                int pin1_in = P1IN & 0x01;
        if(!pin1_in) {
                //get hours
                unsigned int wr_hr = adc_sam20();
                unsigned int temp_hr = adc_div(wr_hr, 23);
                unsigned int new_temp_hr = map_dec_hex(temp_hr);
                write_I2C(CNTRL_BYTE, RTC_HOUR, new_temp_hr);
        }
        int pin2_in = P2IN & 0x04;
        if (!pin2_in) {
                //get minutes
                unsigned int wr_min = adc_sam20();
                unsigned int temp_min = adc_div(wr_min, 59);
                unsigned int new_temp_min = map_dec_hex(temp_min);
                write_I2C(CNTRL_BYTE, RTC_MIN, new_temp_min);
        }
}
//alarm mode
else {

        al_sec = read_I2C(CNTRL_BYTE, RTC_ALARM0_SEC);
        al_min = read_I2C(CNTRL_BYTE, RTC_ALARM0_MIN);
        al_hour = read_I2C(CNTRL_BYTE, RTC_ALARM0_HOUR);
        al_day = read_I2C(CNTRL_BYTE, RTC_ALARM0_DAY);
        al_date = read_I2C(CNTRL_BYTE, RTC_ALARM0_DATE);
        al_month = read_I2C(CNTRL_BYTE, RTC_ALARM0_MONTH);

        //Throw it to the LCD
        //time
        write_to_lcd(al_hour);
        lcd_char(':');
        write_to_lcd(al_min);
        lcd_char(':');
        sec &= 0x7F;
        write_to_lcd(0x00);
        //date
        lcd_command(DOWN);
        write_to_lcd(al_date);
        lcd_char('/');
        month &= 0x1F;
        write_to_lcd(al_month);
        lcd_char('/');
        write_to_lcd(year);

        lcd_command(RIGHT);

        day_num = read_I2C(CNTRL_BYTE, RTC_DAY);
        day_num &= 0x07;

        lcd_string(days_of_week[--day_num]);

        int pin1_in = P1IN & 0x01;
        if(!pin1_in) {
                unsigned int wr_hr = adc_sam20();
                unsigned int temp_hr = adc_div(wr_hr, 23);
                unsigned int new_temp_hr = map_dec_hex(temp_hr);
                write_I2C(CNTRL_BYTE, RTC_ALARM0_HOUR, new_temp_hr);
        }
        int pin2_in = P2IN & 0x04;
        if (!pin2_in) {
                unsigned int wr_min = adc_sam20();
                unsigned int temp_min = adc_div(wr_min, 59);
                unsigned int new_temp_min = map_dec_hex(temp_min);
                write_I2C(CNTRL_BYTE, RTC_ALARM0_MIN, new_temp_min);
        }
        if (!(P3IN & 0x80)) {
                //set alarm on whe nbutton is pressed
                P1OUT |= 0x08;
                write_I2C(CNTRL_BYTE, RTC_CNTRL, 0x10);
        }
}

al_day = 0xF0 | day;
write_I2C(CNTRL_BYTE, RTC_ALARM0_DAY, al_day);
write_I2C(CNTRL_BYTE, RTC_ALARM0_DATE, date);
write_I2C(CNTRL_BYTE, RTC_ALARM0_MONTH, month);
write_I2C(CNTRL_BYTE, RTC_ALARM0_SEC, sec);
```

```
        int pin_al = P1IN & 0x04;
        if(pin_al) {
            //if MFP pin is high set flag
            flag = 1;
        }
        if (flag) {
            while(P3IN & 0x80) {
                //make a noise until button is pressed
                output_sound(1); //sound
            }
            flag = 0;
            //turn alarm off
            P1OUT &= 0xF7;
            write_I2C(CNTRL_BYTE, RTC_CNTRL, 0x00);
        }
        lcd_command(HOME);
    }

    return 0;
}

unsigned int adc_div(unsigned int partition, int divs) {
    if(partition > 1023) {
        partition = 1023; //limit, just in case
    }
    //hours
    if(divs == 23) {
        return partition/44;
    }
    //minutes
    else {
        return partition/17;
    }
}

unsigned int adc_sam20(void) {
    volatile unsigned int sum = 0;
    volatile unsigned int value;
    ADC10CTL0 |= ENC + ADC10SC;         // Start the conversion and enable conversion
    __bis_SR_register(CPUOFF + GIE);    // call ISR, low power mode0
    value = ADC10MEM;                                     // Save measured val
    return value;
}

void write_to_lcd(unsigned int val) {
    char array[2] = {}; //only need 2 digits
    unsigned int temp = 0x0F & val; //grab the last digit
    temp += '0'; // int to char
    array[1] = temp; //save last digit
    val >>= 4; //grab first digit
    val += '0'; //int to char
    array[0] = val; //save first
    lcd_string(array);
}

//This function maps de to hex, i.e. 45 becomes 0x45,
unsigned int map_dec_hex(unsigned int dec) {
    return dec + ((dec/10) * 6);
}

void lcd_string(char *str) {
    //This writes words yo
    while (*str != 0) {
        lcd_char(*str);
        *str++;
    }
}

void lcd_init(void){
    lcd_command(0x33);
    lcd_command(0x32);
    lcd_command(0x2C);
    lcd_command(0x0C);
    lcd_command(0x01);
}
```

```
void lcd_command(char uf_lcd_x){
	P4DIR = 0xFF;
	uf_lcd_temp = uf_lcd_x;
	P4OUT = 0x00;
	__delay_cycles(1000);
	uf_lcd_x = uf_lcd_x >> 4;
	uf_lcd_x = uf_lcd_x & 0x0F;
	uf_lcd_x = uf_lcd_x | 0x20;
	P4OUT = uf_lcd_x;
	__delay_cycles(1000);
	uf_lcd_x = uf_lcd_x & 0x0F;
	P4OUT = uf_lcd_x;
	__delay_cycles(1000);
	P4OUT = 0x00;
	__delay_cycles(1000);
	uf_lcd_x = uf_lcd_temp;
	uf_lcd_x = uf_lcd_x & 0x0F;
	uf_lcd_x = uf_lcd_x | 0x20;
	P4OUT = uf_lcd_x;
	__delay_cycles(1000);
	uf_lcd_x = uf_lcd_x & 0x0F;
	P4OUT = uf_lcd_x;
	__delay_cycles(1000);
}

void lcd_char(char uf_lcd_x){
	P4DIR = 0xFF;
	uf_lcd_temp = uf_lcd_x;
	P4OUT = 0x10;
	__delay_cycles(1000);
	uf_lcd_x = uf_lcd_x >> 4;
	uf_lcd_x = uf_lcd_x & 0x0F;
	uf_lcd_x = uf_lcd_x | 0x30;
	P4OUT = uf_lcd_x;
	__delay_cycles(1000);
	uf_lcd_x = uf_lcd_x & 0x1F;
	P4OUT = uf_lcd_x;
	__delay_cycles(1000);
	P4OUT = 0x10;
	__delay_cycles(1000);
	uf_lcd_x = uf_lcd_temp;
	uf_lcd_x = uf_lcd_x & 0x0F;
	uf_lcd_x = uf_lcd_x | 0x30;
	P4OUT = uf_lcd_x;
	__delay_cycles(1000);
	uf_lcd_x = uf_lcd_x & 0x1F;
	P4OUT = uf_lcd_x;
	__delay_cycles(1000);
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {
	__bic_SR_register_on_exit(CPUOFF);	// Return to active mode
}

void adc_setup(void) {
	// ADC
	ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; //0x1018, 16xADC10CLks, ADC10 on, and ADC10 interupt enable
	ADC10AE0 |= 0x01; //Enable reg 0
}

void i2c_init() {
	WDTCTL = WDTPW + WDTHOLD;		// Stop WDT
	P2DIR |= (1 << 1);
	P2SEL |= (1 << 1);		// SMCLK Output
	P3DIR = 0x0F;						// disable CC2500 //TODO remember to reenable the radio
	P3SEL |= 0x06;			// Assign I2C pins to USCI_B0
}

void rtc_init() {
	i2c_init();
	write_I2C(CNTRL_BYTE, RTC_CNTRL, 0x00);
```

```
    unsigned int temp = read_I2C(CNTRL_BYTE,RTC_SEC);
    if(!(temp & 0x80)) {
        //Oscilator bit of RTCC is off. Turn on to start RTCC function
        write_I2C(CNTRL_BYTE, RTC_SEC, RTCC_EN_OSC_START);
    }
}

void write_I2C(unsigned int Slave_Add,unsigned int Add, unsigned int Val) {
    Setup_TX(Slave_Add);
    Res_Flag = 0;
    TxBuffer[0] = Add;
    TxBuffer[1] = Val;
    PTxData = TxBuffer; // TX array start address
    TXByteCtr = 2; // Load TX byte counter
    while (UCB0CTL1 & UCTXSTP); // Ensure stop condition got sent
    UCB0CTL1 |= UCTR + UCTXSTT; // I2C TX, start condition
    __bis_SR_register(CPUOFF + GIE);
    // Enter LPM0 w/ interrupts
    while (UCB0CTL1 & UCTXSTP);         // Ensure stop condition got sent
}

unsigned int read_I2C(unsigned int Slave_Add,unsigned int Add) {
    Setup_TX(Slave_Add);
    Res_Flag = 1;
    PTxData = &Add; // TX array start address
    TXByteCtr = 1; // Load TX byte counter
    while (UCB0CTL1 & UCTXSTP); // Ensure stop condition got sent
    UCB0CTL1 |= UCTR + UCTXSTT; // I2C TX, start condition
    __bis_SR_register(CPUOFF + GIE); // Enter LPM0 w/ interrupts
    while (UCB0CTL1 & UCTXSTP);         // Ensure stop condition got sent

    Res_Flag = 0;
    Setup_RX(Slave_Add);
    PRxData = &RxBuffer; // Start of RX buffer
    RXByteCtr = 1; // Load RX byte counter
    while (UCB0CTL1 & UCTXSTP); // Ensure stop condition got sent
    UCB0CTL1 |= UCTXSTT; // I2C start condition
    while (UCB0CTL1 & UCTXSTT);      // Start condition sent?
```

```
        UCB0CTL1 |= UCTXSTP;            // No Repeated Start: stop condition
        __bis_SR_register(CPUOFF + GIE);
        // Enter LPM0 w/ interrupts
        while (UCB0CTL1 & UCTXSTP);        // Ensure stop condition got sent

        return RxBuffer;
}

void Setup_TX(unsigned int Add) {
        _DINT();
        RX = 0;
        IE2 &= ~UCB0RXIE;                        // Disable RX interrupt
        while (UCB0CTL1 & UCTXSTP); // Ensure stop condition got sent
        UCB0CTL1 |= UCSWRST;                // Enable SW reset
        UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC; // I2C Master, synchronous mode
        UCB0CTL1 = UCSSEL_2 + UCSWRST;           // Use SMCLK, keep SW reset
        UCB0BR0 = 12; // fSCL = SMCLK/12 = ~100kHz
        UCB0BR1 = 0;
        UCB0I2CSA = Add; // Slave Address
        UCB0CTL1 &= ~UCSWRST; // Clear SW reset, resume operation
        IE2 |= UCB0TXIE; // Enable TX interrupt
}

void Setup_RX(unsigned int Add) {
        _DINT();
        RX = 1;
        IE2 &= ~UCB0TXIE;     // Disable TX interrupt
        UCB0CTL1 |= UCSWRST; // Enable SW reset
        UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC; // I2C Master, synchronous mode
        UCB0CTL1 = UCSSEL_2 + UCSWRST; // Use SMCLK, keep SW reset
        UCB0BR0 = 12; // fSCL = SMCLK/12 = ~100kHz
        UCB0BR1 = 0;
        UCB0I2CSA = Add; // Slave Address
        UCB0CTL1 &= ~UCSWRST; // Clear SW reset, resume operation
        IE2 |= UCB0RXIE; // Enable RX interrupt
}

#pragma vector = USCIAB0TX_VECTOR
```

```
__interrupt void USCIAB0TX_ISR(void) {
        if(RX == 1) {                           // Master Recieve?
                *PRxData = UCB0RXBUF;
                __bic_SR_register_on_exit(CPUOFF);      // Exit LPM0
        }
        else {
                if (TXByteCtr) {                        // Check TX byte counter
                        UCB0TXBUF = *PTxData++;         // Load TX buffer
                        TXByteCtr--;                    // Decrement TX byte counter
                }
                else {
                        if(Res_Flag == 1) {
                                Res_Flag = 0;
                                PTxData = TxBuffer;             // TX array start address
                                __bic_SR_register_on_exit(CPUOFF);
                        }
                        else {
                                UCB0CTL1 |= UCTXSTP;            // I2C stop condition
                                IFG2 &= ~UCB0TXIFG;             // Clear USCI_B0 TX int flag
                                __bic_SR_register_on_exit(CPUOFF);      // Exit LPM0
                        }
                }
        }
}

void delay_us(unsigned int us) {
   unsigned int i;
   for (i = 0; i<= us/2; i++);
     __delay_cycles(1);
}

void SPI_setup(void) {
        P3OUT = 0x50;                   // Set slave reset
        P3DIR |= 0x50;                  //
        P3SEL |= 0x31;                  // P3.0,4,6 USCI_A0 option select
        UCA0CTL0 |= UCCKPL + UCMSB + UCMST + UCSYNC;  // 3-pin, 8-bit SPI master
        UCA0CTL1 |= UCSSEL_3;           // SMCLK
        UCA0BR0 |= 2;                   // /2
        UCA0BR1 = 0;                    //
        UCA0MCTL = 0;                     // No modulation
        UCA0CTL1 &= ~UCSWRST;                 // **Initialize USCI state machine**
        P3OUT &= ~0x50;                     // Now with SPI signals initialized,
        P3OUT |= 0x50;                                                              // reset slave
}

void SPI_write( unsigned int data) {
   uint8_t byte2Transmit = data & 0x003F;
   byte2Transmit <<= 2;
   data >>= 6 ;
   uint8_t byte1Transmit = data & 0x000F;
   byte1Transmit  |= 0xF0;
   P3OUT &= 0b10111111;
   delay();
   UCA0TXBUF = byte1Transmit;         //Byte to SPI TXBUF
   while(!(IFG2 & UCA0TXIFG)); //USCI_A0 TX buffer ready?
   UCA0TXBUF = byte2Transmit;
   while((UCB0STAT & BIT0));
   delay();
   P3OUT += 0x40;
}

void output_sound(int freq_value) {
        int i;
        for (i = 0; i < 16; i++) {
                SPI_write(sine_t[i]);
                delay_us(freq_value);
        }
}

void delay(void) {
   volatile unsigned int i;
   for(i = 0; i < 1; i++);
}
```