[Ricardo Stefano Reyna]
[18/July/2016]
[21010521]

RTC/I2C Module Report

Introduction

For this module we worked with the I2C of the microprocessor to communicate with the mcp7940n RTC. The I2C is similar to the SPI except that it just uses two lines (serial clock, and serial data) which is in charge of transmitting and receiving said data. The RCT (other than the control byte which is the same) just needs the address pointing to what part of the RTC you want to configure or read, and the value you'd like to write.

Design

For this design I started making the same connections I did for the ohm meter module in regard of the LCD. The only new lines I had to connect was towards the RTC where VCC and GND were obvious, however the serial clock I connected it to port 3.2 and the serial data to port 3.1 following the code example from TI. Once connected an I2C_init function and the interrupt from the same example was done. Later I added a function to setup up the receive setup (RX) and transmit setup (TX), these are to be used later when either writing or reading the RTC and takes in the control byte. The write function will take in the control byte, address, and value to transmit, will call the TX_setup and store the address and the value in the buffer accordingly. The read function just takes in the control byte and the address, this calls the TX_setup in the same manner, but later calls the RX_setup to receive the data and returns the value in that address. These two functions which I called write_I2C and read_I2C are the base of the whole program, pretty much I can configure do anything with the RTC now. Lastly I added the RTC_init function that calls the I2C_init, disables anything in the control address, and starts the oscillator. Inside my main I made a few modifications to display the time in the LCD screen and see if it was running. After confirming it was working perfectly I added the date and made some formatting fixes, followed by and array of strings that are to be used for the day of the week. I added some helper functions to map a decimal value to hex value, a function to display the time into the LCD, and function that partitions the ADC into 23 (hours) or 59 (minutes or seconds). Once done I tested these functions to see if they worked and proceeded to add a SPDT switch to change from time to alarm which I connected to port 1.1, used the same code as before only changing variables for time and for alarm and tested it. I also added two push buttons in order to change the hours and minutes of the RTC to ports 1.0 and 2.3 respectively. Lastly I connected the MFP pin to port 1.3 and a push button to port 3.5 for the alarm. I set the

MFP high true when the alarm is on which turns my flag true. As long as the flag is true and the button is not pressed the buzzer connected to port 1.4 will make a sound, the same button is used to activate the alarm which is done by setting the alarm enable in the control address.

Conclusion

This module was by far the most interesting one where we at least built a common real world house hold device. Understanding the I2C was tricky at first, but it was because of the SPI module that it started to seem familiar so I started to look up differences between them and when to use them in order to understand it better. The RTC was much simpler to understand since really all I needed was the 12 page of the documentation which had all the register addresses map. It would be interesting to implement a timer mode as well, where you can set the amount of hours or minutes and add it to the current time, take into consideration if it exceeds the 59-minute mark or 23-hour mark, and write it into the second alarm register to activate it.

Appendix

```
#include <msp430.h>


/*
 * main.c
 */
#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_1HZ0x00
```

```
//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);



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();

        P1DIR = 0x08;

        volatile unsigned int sec = 0x40;
        sec |= 0x80;
        volatile unsigned int min = 0x39;
        volatile unsigned int hour = 0x14;
        volatile unsigned int day = 0x04;
        volatile unsigned int date = 0x14;
        volatile unsigned int month = 0x07;
        volatile unsigned int year = 0x16;

        volatile unsigned int al_sec = 0x00;
        volatile unsigned int al_min = 0x42;
        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

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
                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 noice until button is pressed
                                beep(255);
                        }
                        flag = 0;
                        //turn alarm off
                        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 beep(unsigned int note) {
    long delay = (long)(10000/note);  //Note's semiperiod.
    P1OUT |= 0x08;     //Set P1
    delay_us(delay);   //play half
    P1OUT &= ~0x08;    //reset
    delay_us(delay);   //play half
}
```