```c
/**
 * LAB ON A BIRD OS V1.0
 * For use on Atmega 128RFA1
 * This code is used to monitor the state of the
 * battery by which it is power, and modify the
 * active state duty cycle to reduce power consumption
 * under low battery power.
 *
 * 2012-05-02 V 1.0
 */
#include <avr/sleep.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <util/delay.h>                     //needed for degubbing when using UART
#include <avr/wdt.h>
#include <avr/eeprom.h>

//****** Interrupt defs
#define battery_measure 1
#define instrument_measure 0
volatile char interrupt_type;               //lets the system know if the interrupt was for
battery or instrument measurements

//****** ADC defs
#define s_p0 0.00887097                     // calibrated scale for 1.8V internal reference
voltage over 10-bit 1024 resolution with 0.178 voltage divider
#define s_p1 0.00456432                     // calibrated scale for 1.8V internal reference
voltage over 10-bit 1024 resolution with 0.3913 voltage divider

//****** LCD defs- used only for debugging
#include "lcd_lib_128RFA1.h"
const int8_t LCD_initialize[] PROGMEM = "LCD Initialized\0";
const int8_t LCD_line[] PROGMEM = "line 1\0";
const int8_t LCD_number[] PROGMEM = "Number=\0";
int8_t lcd_buffer[17];                                      // LCD display buffer
uint16_t count;                                             // a number to display on the
LCD
uint8_t anipos, dir;                                        // move a character around

//****** I2C defs for DS1337
#include "i2cmaster.h"
#define ADDR_clck 0xD0    //Address is 1101 000

#define SDA_PORT PORTD;
#define SCL_PORT PORTD;
#define time_addr_secs  0x00
#define time_addr_mins  0x01
#define time_addr_hour  0x02
#define time_addr_days  0x03
#define time_addr_date  0x04
```

```
#define time_addr_mnth  0x05
#define time_addr_year  0x06

#define alm1_addr_secs  0x07
#define alm1_addr_mins  0x08
#define alm1_addr_hour  0x09
#define alm1_addr_dydt  0x0A

#define alm2_addr_mins  0x0B
#define alm2_addr_hour  0x0C
#define alm2_addr_dydt  0x0D

#define clck_addr_ctrl  0x0E
#define clck_addr_stts  0x0F
    //Control register bits
#define A1IE    0                   //Alarm 1 interupt enable
#define A2IE    1                   //Alarm 2 interupt enable
#define INTCN   2                   //Interrupt control
#define RS1     3                   //Square wave output rate select bit 1
#define RS2     4                   //Square wave output rate select bit 2
#define EOSC    7                   //Enable Oscillator
    //Status register bits
#define A1F     0                   //Alarm 1 flag
#define A2F     1                   //Alarm 2 flag
#define OSF     7                   //Oscillator stop flag
    //Alarm Control Register Bits (Note there is no A2M1-no seconds control on alarm 2)
#define A1M1    7                   //Alarm1 mask bit 1
#define A1M2    7                   //Alarm1 mask bit 2
#define A1M3    7                   //Alarm1 mask bit 3
#define A1M4    7                   //Alarm1 mask bit 4
#define A2M2    7                   //Alarm2 mask bit 2
#define A2M3    7                   //Alarm2 mask bit 3
#define A2M4    7                   //Alarm2 mask bit 4
#define dydt    6                   //date or Dat register bit

//****** UART FILE DESCRIPTOR-used for debugging only
#include "uart.h"
// putchar and getchar are in uart.c
FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);

//****** FUNCTION DEFINITIONS
void init_ei(int ei, char type);
void init_lcd(void) ;
void init_uart(void) ;
void set_clck(void) ;
void set_alm1(uint8_t time[4], char repeat);
int get_time(char unit);
int cnvrt_to_clck(int number, char unit);
int cnvrt_from_clck(int number, char unit);
void write_bit(int logical, int bit, int reg_addr,int dev_addr);
int read_bit( int bit, int reg_addr,int dev_addr);
void cnvt_from_jd(long int JD_abs,double frac_day, int *p_time);
void cnvt_to_jd(int time[7], long *p_JD, double *p_JD_frac);
void add_time  (int time[7], int delta[7], int *p_time_result);
```

```c
double measure_volt(uint8_t channel);
void PRR_sleep(void);
void PRR_wake(void);
void period(double v_batt, int alarm, int *p_delta);
void sleep_point(void);
uint16_t swap_endian16(uint16_t number);
uint32_t swap_endian32(uint32_t number);
void save_data (uint32_t JD, double JD_frac, double measurement);
int main(void);


//**********************************************************
// External interrup intialization
// ei is the interrupt you want to set.
// 'l' is for low level trigger.
// 'a' is for any edge trigger
// 'f' is for falling edge trigger
// 'r' is for rising edge trigger
void init_ei(int ei, char type)
    {int type_select = 0x01;

         if (type=='l'){type_select = 0x00;}
    else if (type=='a'){type_select = 0x01;}
    else if (type=='f'){type_select = 0x02;}
    else if (type=='r'){type_select = 0x03;}

    EIMSK |= (1 << ei);                             //enable the 'ei'-th interrupt pin

    if (ei>3)
    {
    //fprintf(stdout,"Selected correct port: %d \n\r",ei);

        EICRB |= (type_select<<(ei-4)*2);           //set the control
        DDRE  &= ~(1<<(ei));                        //set the pin as an input
        //PORTE |= (1<<(ei));                       //set the input pin high
        PORTE &= ~(1<<(ei));                        //set the input pin low
    }
    else if (ei<=3)
    {
        EICRA |= (type_select<<ei*2);               //set the control
        DDRD  &= ~(1<<ei);                          //set the pin as an input
        //PORTD |= (1<<ei);                         //set the input pin high
        PORTD &= ~(1<<(ei));                        //set the input pin low
    }

    }


//**********************************************************
// LCD setup
void init_lcd(void)
{
    LCDinit();                                      //initialize the display
    LCDcursorOFF();
```

```
    LCDclr();                                       //clear the display
    LCDGotoXY(0,0);
    CopyStringtoLCD(LCD_initialize, 0, 0);
}

//************************************************************
// UART setup
void init_uart(void)
{
  uart_init();                                      // init the UART -- uart_init() is in uart.c
  stdout = stdin = stderr = &uart_str;
  fprintf(stdout,"UART system online....\n\r");     //Tell the user everything's okay so far
}


//************************************************************
// Set the DS1337 Clock
// This function sets the clock to 2012-01-01 00:00:00
void set_clck(void)
{
    i2c_start_wait(ADDR_clck+I2C_WRITE);        //call the clock
    i2c_write(time_addr_secs);                  //tell the clock what register we want to write
    to-the clock will increment the register automatically after each write event
    i2c_write(0x00);                            //write 0 seconds
    i2c_write(0x00);                            //write 0 minutes
    i2c_write(0x00);                            //write 0 hours
    i2c_write(0x00);                            //write 0 day
    i2c_write(0x01);                            //write 1 date
    i2c_write(0x01);                            //write 1 month
    i2c_write(0b00001100);                      //write 12 year
    i2c_stop();                                 //stop the I2C comm
    write_bit(0,OSF,clck_addr_stts,ADDR_clck);  //clear the oscillator stop flag
    //fprintf(stdout,"Seconds after setting clock: %d \n\r",get_time('s'));
    //fprintf(stdout,"Years after setting clock: %d \n\r",get_time('Y'));
}



//************************************************************
// Write a single bit to a control register. This function only modifies the specified bit and
leaves all other alone
// Note that this was written for the DS1337 and may not work for all i2c devices.
// logical      -the value of the bit you want to write
// dev_addr     -the address of the device you want to write to
// reg_addr     -the address of the register in the device
// bit          -the bit (0-7) that you want to write to.
void write_bit(int logical, int bit, int reg_addr,int dev_addr)
{
    int result;
    i2c_start_wait(dev_addr+I2C_WRITE);                     //Find out the current settings
    i2c_write(reg_addr);                                    //of the register you want to
    i2c_rep_start(dev_addr+I2C_READ);                       //modify
    result = i2c_readNak();                                 //
    i2c_start_wait(dev_addr+I2C_WRITE);
```

```
    i2c_write(reg_addr);
    if(logical) {i2c_write(result |  (1<<bit));}            //Set the bit
    else        {i2c_write(result & ~(1<<bit));}            //Clear the bit
    i2c_stop();                                             //Stop I2C comm

}

//***********************************************************
//Reads and returns the logic any bit on the DS1337.
// dev_addr     -the address of the device you want to write to
// reg_addr     -the address of the register in the device
// bit          -the bit (0-7) that you want to read from to.
int read_bit( int bit, int reg_addr,int dev_addr)
{
    int result;
    int the_bit;
    i2c_start_wait(dev_addr+I2C_WRITE);                     //Find out the current settings
    i2c_write(reg_addr);                                    //of the register you want to
    i2c_rep_start(dev_addr+I2C_READ);                       //read from
    result = i2c_readNak();                                 //

    the_bit = ((result & (1<<bit))>>bit);                   //mask and output the bit
    return the_bit;                                         //return the result
}

//***********************************************************
// Set the DS1337 Alarm 1
//  This function sets alarm 1. It can be set to either repeat when the hour, minutes,
// and seconds match, or can be set to only go off once. This repeat functionality is
// is controlled by the 'repeat' variable. The function that uses this alarm MUST CLEAR
// THE ALARM FLAG ON THE CLOCK.
//
//                  'd'     go off when day (1-7), hours, minute and seconds match
//                  'D'     go off when date (1-31), hours, minute and seconds match
// 'repeat'         'h'     go off when hour, minute, and seconds match
//                  'm'     go off when minute and seconds match
//                  's'     go off when seconds match

// 'time'           this variable is an array of for integers. Each integer the time of
//                  that the alarm should be set for.
//                  [date/day,hour,minute,seconds]
//                  note that if the 'repeat' is set as 'd', the first element of 'time'
//                  must represent the week day(1-7). If 'repeat' is  'D', 1st of 'time' must be
//                  the month date (1-31)


void set_alm1(uint8_t time[4], char repeat)
{
    uint8_t set1 = 0;
    uint8_t set2 = 0;
    uint8_t set3 = 0;
    uint8_t set4 = 0;
    uint8_t set5 = 0;
                    //fprintf(stdout,"time[3]: %d \n\r",time[3]);
```

```
                //fprintf(stdout,"time[2]: %d \n\r",time[2]);
                //fprintf(stdout,"time[1]: %d \n\r",time[1]);
                //fprintf(stdout,"time[0]: %d \n\r",time[0]);

    write_bit(0,A1F,clck_addr_stts,ADDR_clck);              //Clear only the alarm 1 flag
    write_bit(1,INTCN,clck_addr_ctrl,ADDR_clck);            //setup INTA pin for logic 1 when
    alarm is met
    write_bit(1,A1IE,clck_addr_ctrl,ADDR_clck);             //setup INTA pin for logic 1 when
    alarm is met

         if(repeat == 'd')                                  //setup alarm to go off when day,
         hour, minute, and seconds match
    {set1 = 0; set2 = 0; set3 = 0; set4 = 0; set5 = 1;}
    else if(repeat == 'D')                                  //setup alarm to go off when date,
    hour, minute, and seconds match
    {set1 = 0; set2 = 0; set3 = 0; set4 = 0; set5 = 0;}
    else if(repeat == 'h')                                  //setup alarm to go off when hour,
    minute, and seconds match
    {set1 = 0; set2 = 0; set3 = 0; set4 = 1; set5 = 0;}
    else if(repeat == 'm')                                  //setup alarm to go off when
    minute, and seconds match
    {set1 = 0; set2 = 0; set3 = 1; set4 = 1; set5 = 0;}
    else if(repeat == 's')                                  //setup alarm to go off when
    seconds match
    {set1 = 0; set2 = 1; set3 = 1; set4 = 1; set5 = 0;}

    write_bit(set1,A1M1,alm1_addr_secs,ADDR_clck);          // write the setup bits to the clock
    write_bit(set2,A1M2,alm1_addr_mins,ADDR_clck);          //
    write_bit(set3,A1M3,alm1_addr_hour,ADDR_clck);          //
    write_bit(set4,A1M4,alm1_addr_dydt,ADDR_clck);          //
    write_bit(set5,dydt,alm1_addr_dydt,ADDR_clck);          //

    i2c_rep_start(ADDR_clck+I2C_WRITE);

    i2c_write(alm1_addr_secs);
    i2c_write((set1<<A1M1)|cnvrt_to_clck(time[3],'s'));     //A1M1 bit or-ed with the seconds
    converted to clock format
    i2c_write((set2<<A1M2)|cnvrt_to_clck(time[2],'m'));     //A1M2 bit or-ed with the minutes
    converted to clock format
    i2c_write((set3<<A1M3)|cnvrt_to_clck(time[1],'h'));     //A1M1 bit or-ed with the seconds
    converted to clock format- note that we do not set the 12/24 bit. This system uses the 24
    hour mode. If you try to set the hour in alarm to a number greated that 23, you could
    accidentally set this bit. Don't do that.
    if (repeat=='d')                                        //For repeating when the day of the
    week matches
        {
            i2c_write((set4<<A1M4)|(1<<dydt)|cnvrt_to_clck(time[0],'d'));   //A1M1 bit or-ed
            with the seconds converted to clock format
        }
    else if (repeat=='D')                                   //For repeating when the date matches
        {
            i2c_write((set4<<A1M4)|cnvrt_to_clck(time[0],'D'));     //A1M1 bit or-ed with the
            seconds converted to clock format
        }
```

```
    i2c_stop();

}

//**************************************************************
// Set the DS1337 Alarm 2
//  This function sets alarm 2. It can be set to either repeat when the hour, minutes,
// and seconds match, or can be set to only go off once. This repeat functionality is
// is controlled by the 'repeat' variable. The function that uses this alarm MUST CLEAR
// THE ALARM FLAG ON THE CLOCK.
//
//                  'd'     go off when day (1-7), hours, minute and seconds match
//                  'D'     go off when date (1-31), hours, minute and seconds match
// 'repeat'         'h'     go off when hour, minute, and seconds match
//                  'm'     go off when minute and seconds match
//Note that there is no seconds available on alarm 2

// 'time'           this variable is an array of for integers. Each integer the time of
//                  that the alarm should be set for.
//                  [date/day,hour,minute]
//                  note that if the 'repeat' is set as 'd', the first element of 'time'
//                  must represent the week day(1-7). If 'repeat' is  'D', 1st of 'time' must be
//                  the month date (1-31)
void set_alm2(uint8_t time[3], char repeat)
{
    uint8_t set2 = 0;
    uint8_t set3 = 0;
    uint8_t set4 = 0;
    uint8_t set5 = 0;
            //fprintf(stdout,"time[2]: %d \n\r",time[2]);
            //fprintf(stdout,"time[1]: %d \n\r",time[1]);
            //fprintf(stdout,"time[0]: %d \n\r",time[0]);

    write_bit(0,A2F,clck_addr_stts,ADDR_clck);          //Clear only the alarm 2 flag
    write_bit(1,INTCN,clck_addr_ctrl,ADDR_clck);        //setup INTA pin for logic 1 when alarm
    is met
    write_bit(1,A2IE,clck_addr_ctrl,ADDR_clck);         //setup INTA pin for logic 1 when alarm
    is met

        if(repeat == 'd')                               //setup alarm to go off when day, hour,
        minute, and seconds match
    {set2 = 0; set3 = 0; set4 = 0; set5 = 1;}
    else if(repeat == 'D')                              //setup alarm to go off when date,
    hour, minute, and seconds match
    {set2 = 0; set3 = 0; set4 = 0; set5 = 0;}
    else if(repeat == 'h')                              //setup alarm to go off when hour,
    minute, and seconds match
    {set2 = 0; set3 = 0; set4 = 1; set5 = 0;}
    else if(repeat == 'm')                              //setup alarm to go off when minute,
    and seconds match
    {set2 = 0; set3 = 1; set4 = 1; set5 = 0;}


    write_bit(set2,A2M2,alm2_addr_mins,ADDR_clck);      // write the setup bits to the clock
```

```
    write_bit(set3,A2M3,alm2_addr_hour,ADDR_clck);      //
    write_bit(set4,A2M4,alm2_addr_dydt,ADDR_clck);      //
    write_bit(set5,dydt,alm2_addr_dydt,ADDR_clck);      //
    i2c_rep_start(ADDR_clck+I2C_WRITE);
    i2c_write(alm2_addr_mins);
    i2c_write((set2<<A2M2)|cnvrt_to_clck(time[2],'m'));             //A1M2 bit or-ed with the
    minutes converted to clock format
    i2c_write((set3<<A2M3)|cnvrt_to_clck(time[1],'h'));             //A1M1 bit or-ed with the
    seconds converted to clock format- note that we do not set the 12/24 bit. This system uses
    the 24 hour mode. If you try to set the hour in alarm to a number greated that 23, you
    could accidentally set this bit. Don't do that.
    if (repeat=='d')                                                //For repeating when the
    day of the week matches
        {
            i2c_write((set4<<A2M4)|(1<<dydt)|cnvrt_to_clck(time[0],'d'));   //A1M1 bit or-ed
            with the seconds converted to clock format
        }
    else if (repeat=='D')                                           //For repeating when the
    date matches
        {
            i2c_write((set4<<A2M4)|cnvrt_to_clck(time[0],'D'));             //A1M1 bit or-ed
            with the seconds converted to clock format
        }
    i2c_stop();

}

//************************************************************
// Get the current time of the unit specified by 'unit'
int get_time(char unit)
{
    uint8_t unit_addr = 0x00;
    int result;
         if (unit == 's') {unit_addr = time_addr_secs;}
    else if (unit == 'm') {unit_addr = time_addr_mins;}
    else if (unit == 'h') {unit_addr = time_addr_hour;}
    else if (unit == 'd') {unit_addr = time_addr_days;}
    else if (unit == 'D') {unit_addr = time_addr_date;}
    else if (unit == 'M') {unit_addr = time_addr_mnth;}
    else if (unit == 'Y') {unit_addr = time_addr_year;}

    i2c_start_wait(ADDR_clck+I2C_WRITE);
    i2c_write(unit_addr);
    i2c_rep_start(ADDR_clck+I2C_READ);
    result = cnvrt_from_clck(i2c_readNak(),unit);
return result;
}


//************************************************************
// Convert numbers to format used in DS1337
// 'number' is the number you want converted to DS1337 format
// unit is the units of the number you want converted
int cnvrt_to_clck(int number, char unit)
```

```
{
uint8_t tens;
uint8_t ones;
uint8_t converted = 0xFF;

    if (unit == 'Y'){number = number-2000;}
                                                                          //go from this century
    to a 0-99 year base
    if((unit == 's')|(unit == 'm')|(unit == 'h')|(unit == 'd')|(unit == 'D')|(unit == 'M')|(unit
     == 'Y'))
    {
        tens = (number/10);
        ones = number-(tens*10);
        converted = ((tens<<4)|ones);
    }
return converted;
//use the following code in your main program to test this function functionality
//fprintf(stdout,"The converted value is %d \n\r",cnvrt_to_clck(23,'s'));
}


//************************************************************
// Convert numbers from format used in DS1337
//This function converts any of the DS1337 clock register values (0x00-0x06) to a normal time
number. The function always outputs hours in the 24 hr format,
//although it can accept the hour register when it is in the 12 hour with AM/PM format.
int cnvrt_from_clck(int number, char unit)
{
    int converted = 0xFF;

    if((unit == 's')|(unit == 'm'))
    {
        converted = (((number>>4)&0x7)*10+(number&0x0F));
        //10*number from bits (4 5 6) + the number represented by the lower 4 bits
    }
    else if(unit == 'h')
    {
        if(number&(1<<6))                                                         // if
        for some reason, the clock is set to 12 hr mode
        converted = (((number>>4)&0x01)*10+(number&0x0F))+12*((number&(1<<5))>>5);
        //10*number from bit (4) + the number represented by the lower 4 bits + 12 hours if it
        is PM
        else                                                                      // For
        the 24 hr set case....
        converted = (((number>>4)&0x03)*10+(number&0x0F));
        //10*number from bit (4) + the number represented by the lower 4 bits + 12 hours if it
        is PM
    }
    else if(unit == 'd')
    {
        converted = number;
        //Nothing else is held in the day register
    }
    else if(unit == 'D')
```

```
    {
        converted = (((number>>4)&0x03)*10+(number&0x0F));
        //10*number from bits (4 5) + the number represented by the lower 4 bits
    }
    else if(unit == 'M')
    {
        converted = (((number>>4)&0x01)*10+(number&0x0F));
        //10*number from bits (4) + the number represented by the lower 4 bits. Ignore the MSB
        because it contains the century bit.
    }
    else if(unit == 'Y')
    {
        converted = ((number>>4)*10+(number&0x0F))+2000;
        //10*number from upper 4 bits + the number represented by the lower 4 bits. Ignore the
        MSB because it contains the century bit. Add 2000 to get us to this century

    }
return converted;

}



//************************************************************
void  init_mcu(void)
{
    DDRG = 0x01;                            //Set G0 as an output-This is my debugging pin to
    show wake and sleep cycles
    PORTG |= (1<<0);                        // set PORTG0 high
    DDRD |= (1<<2);                         //Set up trigger output for turning on and off
    battery
    PORTD &= ~(1<<2);                       //Set to low voltage;

    init_uart();                            //initialize the uart system
    i2c_init();                             //initialize the I2C system
    wdt_disable();                          //diable the watchdog timer
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);    //Set up the Power Down Sleep mode

    write_bit(0,A2IE,clck_addr_ctrl,ADDR_clck);     //turn off the output of alarm 1 because it
    stays on from the last time you set it
    write_bit(0,A2IE,clck_addr_ctrl,ADDR_clck);     //turn off the output of alarm 2 because it
    stays on from the last time you set it
}

//************************************************************
//Entry point and task scheduler loop
int main(void)
{
    init_mcu();
    //while(1){measure_volt(0);_delay_ms(2000);}                //line used for debugging

    //DECLARE SOME LOCAL VARIABLES
    uint8_t the_alarm[4] = {0,0,0,15};
    int curr_time[7] = {0,0,0,0,0,0};       int *p_curr_time;   p_curr_time = (int *)&curr_time
    ;        //initialize current time and point to the array
```

```
    int futr_time[7]  = {0,0,0,0,0,0,0};    int *p_futr_time;   p_futr_time = (int *)&futr_time
    ;        //initialize furture time and point to the array
    int delta_time[7] = {0,0,0,0,0,0,0};    int *p_delta_time;  p_delta_time = (int *)&
    delta_time;      //initialize the time between current and future
    uint8_t startupstate = 0;
    double v_meas = 0;

    //initialize the measured battery voltage
    double inst_meas;

    uint16_t curr_mem_loc = 27;
                                                                                //start at end of
    second expected wakt time location
    uint32_t EWT_1, EWT_2;
                                                                                     //expected
    wakeup times of alarm 1 and alarm 2
    double EWT_1_frac, EWT_2_frac, LMT, LMT_frac;
                                                              //expected wake up time fractions of
    a day, last measurement time- whole day, and last measurement fraction of a day

    long int *p_EWT_1;      p_EWT_1      = (long int *)&EWT_1;
                                                //pointers to expected wake times
    double   *p_EWT_1_frac; p_EWT_1_frac = (double *)&EWT_1_frac;
                                             //pointers to expected wake times
    long int *p_EWT_2;      p_EWT_2      = (long int *)&EWT_2;
                                                //pointers to expected wake times
    double   *p_EWT_2_frac; p_EWT_2_frac = (double *)&EWT_2_frac;
                                             //pointers to expected wake times

    double alrm1_dt, alrm2_dt;
                                                                                 //differenece
    between EWTs and Current Time-used to findout if we slept through an alarm

    long int    JD_time = 0;
                                                                                   //the Julian date
    double      JD_time_frac = 0.0;
                                                                            //the fraction of the
    day
    long int    *p_JD_time;         p_JD_time= (long int *)&JD_time;
                                          //pointer to whole Julian Day
    double      *p_JD_time_frac;    p_JD_time_frac= (double *)&JD_time_frac;
                                  //pointer to fractional Julian Day

    //fprintf(stdout,"Startup bit in EEPROM: %d
    \n\r",eeprom_read_byte(0));                              //Output to UART so we know what
    our startup state is-for debugging

    //FIGURE OUT WHAT THE START UP CONDITION IS
      if((eeprom_read_byte(0))==0){startupstate=1;}
      //read the byte that containts the start bit
    else
        {
            //fprintf(stdout,"OSF bit is: %d
            \n\r",read_bit(OSF,clck_addr_stts,ADDR_clck));         //for debuggiung
```

```
        if(read_bit(OSF,clck_addr_stts,ADDR_clck)){startupstate=2;}
                                        //this is true of the clock has lost power
        else
        {
            curr_time[0] = get_time('Y');   curr_time[1] = get_time('M');   curr_time[2] =
            get_time('D');
            curr_time[3] = get_time('d');   curr_time[4] = get_time('h');   curr_time[5] =
            get_time('m');
            curr_time[6] = get_time('s');

            cnvt_to_jd(curr_time, p_JD_time, p_JD_time_frac);
                                                    //convert the current time to Julian
            data. Save to p_JD_time and p_JD_time_frac

            EWT_1 = eeprom_read_dword((uint32_t*)3);
            EWT_1_frac = ((double) eeprom_read_dword((uint32_t*)7))/1000000;
            EWT_2 = eeprom_read_dword((uint32_t*)11);
            EWT_2_frac = ((double) eeprom_read_dword((uint32_t*)15))/1000000;
            LMT = eeprom_read_dword((uint32_t*)19);
            LMT_frac = ((double) eeprom_read_dword((uint32_t*)23))/1000000;

            alrm1_dt = (EWT_1 - JD_time)+(EWT_1_frac - JD_time_frac);
                                          //did we sleep through alarm 1?
            alrm2_dt = (EWT_2 - JD_time)+(EWT_2_frac - JD_time_frac);
                                          //did we sleep through alarm 2?
                //fprintf(stdout,"alrm1_dt: %f \n
                \r",alrm1_dt);                                  //debugging line
                //fprintf(stdout,"alrm2_dt: %f \n
                \r",alrm2_dt);                                  //debugging line
                 if (alrm1_dt<=0 && alrm2_dt>0) {startupstate = 3;}
                                                       //missed the alarm1
            else if (alrm2_dt<=0 && alrm1_dt>0) {startupstate = 4;}
                                                  //missed the alarm2
            else if (alrm1_dt<=0 && alrm1_dt<=0){startupstate = 5;}
                                                  //missed both alarms
            else if (alrm1_dt>0  && alrm2_dt>0) {startupstate = 6;}
                                                  //didn't miss any alarms
            }
    }

    //fprintf(stdout,"The startup state is: %d
    \n\r",startupstate);                                    //debugging line

    //fprintf(stdout,"Initial memory location: %d \n  \r", eeprom_read_word((uint16_t
    *)1));            //debugging line

    switch (startupstate)                                              //This switch block
    executes different actions depending on the power up state
    {
        case 1:                                                        //FIRST POWER UP AFTER
        PROGRAMMING
            eeprom_write_byte(0,1);                                    //set the initial
            startup bit
            curr_mem_loc = eeprom_read_word((uint16_t *)1);            //check current mem
```

```
        location
        set_clck();                                            //this sets the clock
        and resets the OSF bit of the clock
    break;
    ///////////////////////
    case 2:                                                    //FULL SYSTEM LOSS OF
    POWER
        curr_mem_loc = eeprom_read_word((uint16_t *)1);        //check current mem
        location
        set_clck();                                            //this sets the clock
        and resets the OSF bit of the clock
    break;
    ///////////////////////
    case 3:                                                    //CPU LOSS OF POWER,
    CLOCK OKAY, SLEPT THROUGH A BATTERY MONITOR ALARM
        curr_mem_loc = eeprom_read_word((uint16_t *)1);        //check current mem
        location
        v_meas  = measure_volt(0);                             //measure the voltage
        on pin F0

        period(v_meas, 1, p_delta_time);                       //Set alarm 1 for
        battery
        //fprintf(stdout,"seconds in period from case 3: %d \n \r",
        delta_time[6]);                    //debugging line
        add_time(curr_time,delta_time,p_futr_time);
        the_alarm[0] = futr_time[2];    //date
        the_alarm[1] = futr_time[4];    //hr
        the_alarm[2] = futr_time[5];    //min
        the_alarm[3] = futr_time[6];    //sec
        //fprintf(stdout,"CT : %d:%d:%d \n \r",
        curr_time[4],curr_time[5],curr_time[6]);               //debugging line
        //fprintf(stdout,"EWT: %d:%d:%d \n \r",
        futr_time[4],futr_time[5],futr_time[6]);               //debugging line
        set_alm1(the_alarm,'m');
    break;
    ///////////////////////
    case 4:                                                    //CPU LOSS OF POWER,
    CLOCK OKAY, SLEPT THROUGH A INSTRUMENT MONITOR ALARM
        v_meas  = measure_volt(0);
        //measure the voltage on pin F0
        period(v_meas, 2, p_delta_time);
        //Set alarm 2 for instrument
        //fprintf(stdout,"seconds in period from case 4: %d \n \r", delta_time[6])
        //debugging line
        add_time(curr_time,delta_time,p_futr_time);
        the_alarm[0] = futr_time[2];    //date
        the_alarm[1] = futr_time[4];    //hr
        the_alarm[2] = futr_time[5];    //min   //note that alarm 2 does not have a
        seconds interval
        set_alm2(the_alarm,'m');
        //NEED TO ADD CODE HERE TO TAKE MEASUREMENT OF SOME INSTRUMENT.
        inst_meas = v_meas;
        //Just and example measurment
        save_data(JD_time,JD_time_frac,inst_meas);
```

```
        break;
        //////////////////////
        case 5:                                                     //CPU LOSS OF POWER,
        CLOCK OKAY, SLEPT THROUGH BOTH ALARMS
            v_meas  = measure_volt(0);
            //measure the voltage on pin F0
            period(v_meas, 1, p_delta_time);
            //Set alarm 1 for battery
            add_time(curr_time,delta_time,p_futr_time);
            the_alarm[0] = futr_time[2];    //date
            the_alarm[1] = futr_time[4];    //hr
            the_alarm[2] = futr_time[5];    //min
            the_alarm[3] = futr_time[6];    //sec
            set_alm1(the_alarm,'m');

            period(v_meas, 2, p_delta_time);
            //Set alarm 2 for instrument
            add_time(curr_time,delta_time,p_futr_time);
            the_alarm[0] = futr_time[2];    //date
            the_alarm[1] = futr_time[4];    //hr
            the_alarm[2] = futr_time[5];    //min
            //note that alarm 2 does not have a seconds interval
            set_alm2(the_alarm,'m');

            //NEED TO ADD CODE HERE TO TAKE MEASUREMENT OF SOME INSTRUMENT.
            inst_meas = v_meas;
            //Just and example measurment
            save_data(JD_time,JD_time_frac, inst_meas);
        break;
        case 6:
            //just go back to sleep. taken care of in the first line of the while loop
        break;
    }

//main task scheduler loop
sei();
//Start up interrupts
while(1)
{
  if ((startupstate==3)|(startupstate==4)|(startupstate==5)|(startupstate==6)){sleep_point();
  startupstate=0;}     //Go to sleep if this was a power up and the case was 3 4 5 or 6
  else                                                        //Get the current time
  curr_time[0] = get_time('Y');                               //Get the current year
  curr_time[1] = get_time('M');                               //Get the current month
  curr_time[2] = get_time('D');                               //Get the current date
  curr_time[3] = get_time('d');                               //Get the current day of week
  curr_time[4] = get_time('h');                               //Get the current hours
  curr_time[5] = get_time('m');                               //Get the current minutes
  curr_time[6] = get_time('s');                               //Get the current seconds
  cnvt_to_jd(curr_time, p_JD_time, p_JD_time_frac);           //convert the current time
  to Julian data. Save to p_JD_time and p_JD_time_frac

  //fprintf(stdout,"curr_time min: %d \n\r",curr_time[5]);        //debugging line
```

```
    //fprintf(stdout,"curr_time hrs: %d \n\r",curr_time[4]);        //debugging line
    //fprintf(stdout,"curr_time date: %d \n\r",curr_time[2]);       //debugging line

    //Set a constant delta time for debugging purposes
    //delta_time[0] = 0;
    //delta_time[1] = 0;
    //delta_time[2] = 0;
    //delta_time[3] = 0;    //day
    //delta_time[4] = 0;    //hr
    //delta_time[5] = 0;    //min
    //delta_time[6] = 5;    //sec
    PORTD |= (1<<2);                                                //Turn off battery
    _delay_ms(20);
    v_meas  = measure_volt(0);                                      //measure the voltage on
    pin F0
    PORTD &= ~(1<<2);           //Turn on battery

    //fprintf(stdout,"Batt 1:%f \n\r",v_meas);                      //debugging line
    period(v_meas, 1, p_delta_time);                                //Decide ont the sleep
    period for alarm 1-battery measurement
    save_data(JD_time,JD_time_frac,v_meas);                         //save the voltage of the
    battery
    add_time(curr_time,delta_time,p_futr_time);                     //add the delta time to the
    current time

    the_alarm[0] = futr_time[2];                                    //date
    the_alarm[1] = futr_time[4];                                    //hr
    the_alarm[2] = futr_time[5];                                    //min
    the_alarm[3] = futr_time[6];                                    //sec

    set_alm1(the_alarm,'m');                                        //set the alarm 1
    cnvt_to_jd(futr_time, p_EWT_1, p_EWT_1_frac);                   //convert the expected
    wakeup time to Julian Date
    eeprom_write_dword((uint32_t*) 3,EWT_1);                        //Write the whole date EWT
    to EEPROM
    eeprom_write_dword((uint32_t*) 7,(uint32_t) (EWT_1_frac*1000000));  //Write the fractional
    date EWT to EEPROM

    //fprintf(stdout,"Current JD: %lu + %lf \n \r",JD_time,JD_time_frac);
    //fprintf(stdout,"Current EWT: %lu + %lf \n \r",EWT_1,EWT_1_frac);
    //fprintf(stdout,"CT : %d:%d:%d \n \r", curr_time[4],curr_time[5],curr_time[6]);
    //fprintf(stdout,"EWT: %d:%d:%d \n \r", futr_time[4],futr_time[5],futr_time[6]);
    //_delay_ms(2);

    sleep_point();                                                  //Put the system to sleep
  }

}

//***********************************************************
//void save_data(uint8_t delta_days,double delta_frac, double measurement, uint32_t JD, double
JD_frac)
void save_data(uint32_t JD, double JD_frac, double measurement)
    {
```

```
    uint16_t curr_mem_loc = eeprom_read_word((uint16_t *)1);                        //check
    current mem location

        if (curr_mem_loc>=4084){curr_mem_loc = 0x001B;}                             //if at the
        end of the memory, wrap back to the begining
        eeprom_write_dword((uint32_t*)curr_mem_loc, JD);                            //save
        whole JD
            curr_mem_loc = curr_mem_loc+4;                                          //increment
            mem location
        eeprom_write_dword((uint32_t*)curr_mem_loc, (uint32_t) (JD_frac*1000000));  //save
        fraction JD*10^6
            curr_mem_loc = curr_mem_loc+4;                                          //increment
            mem location
        eeprom_write_float((float*)curr_mem_loc, measurement);                      //save data
            curr_mem_loc = curr_mem_loc+4;                                          //increment
            mem location
        eeprom_write_word((uint16_t*)1, curr_mem_loc);                              //update
        current mem location
        eeprom_write_dword((uint32_t*)19, JD);                                      //write the
        last saved measurement time
        eeprom_write_dword((uint32_t*)23, (uint32_t) (JD_frac*1000000));            //write the
        last saved measurement time
    }


//************************************************************
//Changes the endianness of a 16 bit number
uint16_t swap_endian16(uint16_t number)
    {
        uint16_t result = 0;
        //fprintf(stdout,"%x \n \r",number);
        //fprintf(stdout,"%x \n \r",(number>>8));
        //fprintf(stdout,"%x \n \r",(number<<8));
        result = ((number >> 8) | (number << 8));
        //fprintf(stdout,"THE RESULT: %x \n \r ",result);
        return result;
    }

//************************************************************
//Changes the endianness of a 32 bit number
uint32_t swap_endian32(uint32_t number)
    {
        uint32_t result;
        uint16_t hold1;
        uint16_t hold2;

        fprintf(stdout,"%lu \n \r",number);
        //fprintf(stdout,"%lu \n \r",(number & 0xFFFF0000));
        //fprintf(stdout,"%lu \n \r",(number & 0x0000FFFF));
        //fprintf(stdout,"%u \n \r",(uint16_t) ((number & 0xFFFF0000)>>16));
        //fprintf(stdout,"%u \n \r",(uint16_t) (number & 0x0000FFFF));

        hold1 = (uint16_t) ((number & 0xFFFF0000)>>16);
        hold2 = (uint16_t) (number & 0x0000FFFF);
```

```
        //fprintf(stdout,"%x \n \r",hold1);
        //fprintf(stdout,"%x \n \r",hold2);
        //fprintf(stdout,"%x \n \r",((hold2>>8)|(hold2<<8)));
        //fprintf(stdout,"%lu \n \r",(((uint32_t) ((hold2>>8)|(hold2<<8)))<<16));

        result = ((((uint32_t) ((hold2>>8)|(hold2<<8)))<<16) | ((uint32_t) ((hold1>>8)|(hold1<<8
        ))));
        //fprintf(stdout,"%f \n \r",result);
        return result;
    }


//*************************************************************
//This function decides the period between sleep and wake events
//based on the input voltage of the battery and which alarm
//is to be set
void period(double v_batt, int alarm, int *p_delta)
{
    uint8_t secs = 0;
    uint8_t mins = 0;

switch (alarm)
    {
        case 1:                         //ALARM 1 CONTROLS BATTERY MEASUREMENTS
            if (v_batt >= 7.9)                              {secs = 10;}
            else if ((v_batt < 7.9) & (v_batt > 7.7))   {secs = 20;}
            else if (v_batt < 7.7)                          {secs = 30;}
        break;                          //ALARM 2 CONTROLS INSTRUMENT MEASUREMENTS
        //////////////////////////
        case 2:                         //NOTE THAT ALARM 2 DOES NOT HAVE A SECONDS REGISTER
            if (v_batt >= 8)                            {mins = 1;}
            else if ((v_batt < 8) & (v_batt > 7))   {mins = 2;}
            else if (v_batt < 7)                        {mins = 3;}
        break;
    }
        *(p_delta)   = 0;                           //save the delta time to the pointer location
        *(p_delta+1) = 0;
        *(p_delta+2) = 0;
        *(p_delta+3) = 0;
        *(p_delta+4) = 0;
        *(p_delta+5) = mins;
        *(p_delta+6) = secs;
        //fprintf(stdout,"in period function delay is: %d \n \r", secs);
}


//*************************************************************
//This function puts the system to sleep
void sleep_point(void)
{
    //init_ei(4,'l');       //set up external interrupt 4 for low level interrupt
    init_ei(5,'l');     //set up external interrupt 4 for low level interrupt
    sleep_enable();
```

```c
    //fprintf(stdout,"Going to sleep \n\r");_delay_ms(5);
    PRR_sleep();                                //set up the power reduction registers
    PORTG |= (1<<0);                            //Debugging pin to tell me when the system goes
    to sleep
    sleep_cpu();                                //slepp the system
    sleep_disable();                            //disable sleep when the system wakes up
    PORTG &= ~(1<<0);                           //Debugging pin to tell me when the system is
    awake
    PRR_wake();                                 //reeneable various systems that were shut down
    by PRR_sleep
    //fprintf(stdout,"Waking up \n\r");_delay_ms(5);

}

//************************************************************
//This function shuts down many of the system prior to sleep
//for reductions in power
void PRR_sleep(void)
{
    ADCSRA &= ~(1<<ADEN);                   //shut down the ADC- this must be done- see bit0
    explanation in section 12.6.2 of datasheet
    PRR0 = 0b11111101;//|(1<<PRUSART0)      //Shut down a whole bunch of peripherals
    PRR1 |= 0b01111000;                     //Shut down even more
}

//************************************************************
//This function reinitialized the systems that were shut down
//by PRR_sleep
void PRR_wake(void)
{
    PRR0 &= ~(0b00000001);          //enable ADC
    ADCSRA |= (1<<ADEN);            //restart ADC
    i2c_init();                     //restart I2C

}

//************************************************************
double measure_volt(uint8_t channel)
{
    uint8_t mux = 0;
    char Ain_l ;                        //raw A to D number-low bits
    uint16_t Ain;                       //10 bit precision raw A to D number
    double volt;                        //voltage of battery
    double scale = s_p0;                //set scale for ADC

   ADMUX = (1<<REFS0) ;                 //RIGH adj /INTERNAL Aref

         if (channel == 0) {mux = 0x00;scale = s_p0;}
    else if (channel == 1) {mux = 0x01;scale = s_p1;}
    else if (channel == 2) {mux = 0x02;}
    else if (channel == 3) {mux = 0x03;}
    else if (channel == 4) {mux = 0x04;}
    else if (channel == 5) {mux = 0x05;}
    else if (channel == 6) {mux = 0x06;}
```

```
    else if (channel == 7) {mux = 0x07;}

    ADMUX = ((ADMUX & 0xE0)|mux);                   //use selected port as the ADC input
    ADCSRA = ((1<<ADEN) | (1<<ADSC)) + 7 ;          //enable ADC and set prescaler to
    1/128*16MHz=125,000, and clear interupt enable, and start a conversion

    while (ADCSRA & (1<<ADSC)){}                    //don't move forward until the ADC is
    complete
    Ain_l = ADCL;
    Ain   = ADCH*256L+Ain_l;                        //cast 256 as long so its product w/ ADCH
    can be longer than 8 bit
    volt  = Ain * scale;

    ADCSRA |= (1<<ADSC) ;                           //start another conversion

//  while (ADCSRA & (1<<ADSC)){}                    //don't move forward until the ADC is
complete
//  Ain_l = ADCL;
//  Ain   = ADCH*256L+Ain_l;                        //cast 256 as long so its product w/ ADCH
can be longer than 8 bit
//  volt  = Ain * scale;

//  ADCSRA |= (1<<ADSC) ;                           //start another conversion

//  printf("Voltage: %f\n\r",volt);                     //results to hyperterm
//  printf("ADC output: %x\n\r",Ain);                   //results to hyperterm

    return volt;

}

//************************************************************
//This function adds a delta amount of time to a time array
// passed to it and saves the result to the location specified
// by the pointer p_time_result
void add_time(int time[7], int delta[7], int *p_time_result)
{

    long int    JD_time = 0;                            //the Julian date of the time that was
    passed
    double      JD_time_frac = 0.0;                     //the fraction of the day of the time
    that was passed
    long int    the_sum = 0;                            // the julian date sum of the time and
    delta
    double      the_sum_frac = 0;                       // the fractional julian date sum of
    the time and delta

    int *p_time;            p_time = (int *)&time;                          //pointers
    long *p_JD_time;        p_JD_time= (long *)&JD_time;                    //pointers
    double *p_JD_time_frac; p_JD_time_frac= (double *)&JD_time_frac;        //pointers

    cnvt_to_jd(time, p_JD_time, p_JD_time_frac);                            //convert current time
    to Julian date
    //fprintf(stdout,"CURRENT JD: %ld + %f  \n\r",JD_time,JD_time_frac);
```

```
    double delta_days = delta[3]+(double)delta[4]/24 + (double)delta[5]/1440 + (double)delta[6]/
    86400;
    double delta_days_frac = delta_days - (int) delta_days;
                                                    //change fractions of a day
    delta_days = (int) delta_days;
                                                                             //change in whole days
    //fprintf(stdout,"delta_days: %f + %lf  \n\r",delta_days,delta_days_frac);

    the_sum_frac = JD_time_frac + delta_days_frac;                             //sum results
    the_sum = JD_time + delta_days + (int) the_sum_frac;                       //incase there is
    rollover from the fractions
    the_sum_frac = the_sum_frac - (int) the_sum_frac;                          //incase there is
    rollover from the fractions

    //fprintf(stdout,"The sum: %ld + %f  \n\r",the_sum,the_sum_frac);

    cnvt_from_jd(the_sum,the_sum_frac, p_time_result);                         //convert sum to
    normal date

//  fprintf(stdout,"Current: %d - %d - %d  %d:%d:%d
\n\r",time[0],time[1],time[2],time[4],time[5],time[6]);
//  fprintf(stdout,"Future:  %d - %d - %d  %d:%d:%d
\n\r",*p_time_result,*(p_time_result+1),*(p_time_result+2),*(p_time_result+4),*(p_time_result+5),
*(p_time_result+6));
}

//************************************************************
//External clock interrupt routine
ISR(INT4_vect)
{
    EIMSK &= ~(1 << 4);             //disable ei interrupt pin
    EIFR |= (1<<4);                 //clear flag by writing 1. See Section 16.2.4 in datasheet

    interrupt_type = battery_measure;               //instrument measurement interrupt
    //fprintf(stdout,"ALARM 2 JUST WENT OFF-INSTRUMENT \n \r");
    //THIS JUST WAKES UP THE CPU
}

//************************************************************
//External clock interrupt routine
ISR(INT5_vect)
{
    EIMSK &= ~(1 << 5);             //disable ei interrupt pin
    EIFR |= (1<<5);                 //clear flag by writing 1. See Section 16.2.4 in datasheet

    write_bit(0,A1F,clck_addr_stts,ADDR_clck);          //Clear only the alarm 1 flag

    interrupt_type = instrument_measure;                //battery measurment interrupt
    //fprintf(stdout,"ALARM 1 JUST WENT OFF-BATTERY \n \r");
    //THIS JUST WAKES UP THE CPU
}
```

```c
//************************************************************
// --COMPUTES THE GREGORIAN CALENDAR DATE (YEAR,MONTH,DAY) GIVEN THE JULIAN DATE (JD).
// The Julian date in this system is assummed to start at noon Universal Time on January 1,4713
BCE.
//Some of this code was taken from the USNO website:
// http://aa.usno.navy.mil/faq/docs/JD_Formula.php
//The algorithm is based on Fliegel, H. F. & van Flandern, T. C. 1968, Communications of the
ACM, 11, 657.
//Modified by Michael Shafer here for use with C code.

// JD_abs       the integer of the Julian Date
// frac_day     the fraction of the day
// *p_time      the pointer to the location of the array where you want the time saved
void cnvt_from_jd(long int JD_abs,double frac_day, int *p_time)
 {
    long int I,J,K,L, N;
    //fprintf(stdout,"***JD_abs: %ld \n\r",JD_abs);
    //fprintf(stdout,"***frac_day: %f \n\r",frac_day);

    int hour, secs, mins;
    double frac_hour = 0;double frac_mins = 0;

    hour = (int)((frac_day+0.5) * 24);frac_hour = (frac_day+0.5)*24 - hour;
    if (hour>=24){JD_abs++;hour = hour-24;}                                    //This is
    needed to increment the day if over 24 hours. The Julian day starts at noon, but the
    calendar day starts at midnight. This just shifts the result so the outputed day is correct
    mins = (int)(frac_hour * 60); frac_mins = frac_hour*60 - mins;
    secs = (int)(frac_mins * 60);                                      //frac_sec =
    frac_mins*60 - secs;

    L= JD_abs+68569;
    N= 4*L/146097;
    L= L-(146097*N+3)/4;
    I= 4000*(L+1)/1461001;
    L= L-1461*I/4+31;
    J= 80*L/2447;
    K= L-2447*J/80;
    L= J/11;
    J= J+2-12*L;
    I= 100*(N-49)+I+L;

//Based on algorithm on http://aa.usno.navy.mil/faq/docs/JD_Formula.php
//This site has a Julian date converter that can be used for comparison


//  fprintf(stdout,"DAY OF THE WEEK: %ld \n\r", JD_abs%7);

    *(p_time) = I;          //YEAR= I
    *(p_time+1) = J;        //MONTH= J
    *(p_time+2) = K;        //DAY= K
    *(p_time+3) = JD_abs%7;     //day of week start 0 for monday and 6 for sunday
    *(p_time+4) = hour;
    *(p_time+5) = mins;
    *(p_time+6) = secs;
```

```c
//  fprintf(stdout,"Year: %ld \n\r",I);
//  fprintf(stdout,"Month: %ld \n\r",J);
//  fprintf(stdout,"Date: %ld \n\r",K);
//  fprintf(stdout,"Day: %ld \n\r",JD_abs%7);
//  fprintf(stdout,"Hour: %d \n\r",hour);
//  fprintf(stdout,"Min: %d \n\r",mins);
//  fprintf(stdout,"Sec: %d \n\r",secs);

 }
//************************************************************
//    INTEGER FUNCTION JD (YEAR,MONTH,DAY)
//---COMPUTES THE JULIAN DATE (JD) GIVEN A GREGORIAN CALENDAR
//   DATE (YEAR,MONTH,DAY).
//Some of this code was taken from the USNO website:
// http://aa.usno.navy.mil/faq/docs/JD_Formula.php
//The algorithm is based on Fliegel, H. F. & van Flandern, T. C. 1968, Communications of the
ACM, 11, 657.
//Modified by Michael Shafer here for use with C code and inclusion of of fractions of a day

//*p_time           Location of time you want converted
//*p_JD             Location where you want the Julian date saved. This location must be
defined as a long integer
//*p_JD_frac        Location where you want the Julian date day fractions saved. This loation
must be defined as a double

//void cnvt_to_jd(int *p_time, long *p_JD, double *p_JD_frac)
void cnvt_to_jd(int time[7], long *p_JD, double *p_JD_frac)
{

//fprintf(stdout,"year in cnvt function: %d \n\r",*p_time);
//fprintf(stdout,"year in cnvt function: %d \n\r",time[0]);
    //double JDN;
    long int JDN;
    long int I,J,K;

    I= time[0];         //YEAR
    J= time[1];         //MONTH
    K= time[2];         //DAY
    int H = time[4];    //HOUR
    int M = time[5];    //MINUTE
    int S = time[6];    //DAY


    JDN = (   K-32075                          \
            + 1461*(I+4800+(J-14)/12)/4        \
            + 367*(J-2-(J-14)/12*12)/12        \
            - 3*((I+4900+(J-14)/12)/100)/4     \
            );

    JDN = JDN-1;
    //fprintf(stdout,"JD befor frac: %lu \n\r",JDN);
    double frac_day = (double) H/ (double) 24 + (double)M /(double)1440 + (double) S/(double)
86400 + 0.5; //The 0.5 is because the result of JDN should have a 0.5 on it, but it is an
```

```
    integer
    JDN = (long int) JDN + (int)frac_day;           //incase the fraction of the day plus the
    computed Julian date carries us over to the next day
    frac_day = frac_day - (int) frac_day;           //if there is carry over, we need to zero
    the fraction of the day
    //fprintf(stdout,"Year: %d - %d - %d  %d:%d:%d
    \n\r",time[0],time[1],time[2],time[4],time[5],time[6]);
    //fprintf(stdout,"JD_longint: %ld \n\r",(long int)JDN);
    //fprintf(stdout,"frac_day: %f \n\r",frac_day);
    //fprintf(stdout,"JD_uint32_t: %lu \n\r",(uint32_t)JDN);
    //fprintf(stdout,"frac_day: %f \n\r",frac_day);

    *p_JD = (long int) JDN;
    *p_JD_frac =  frac_day;
}
```