
Reloj de Tiempo Real DS1307

Autor: Ing. Carlos Narváez
Universidad de Oriente
email: cnarvaez@udo.edu.ve

Introducción

En el presente trabajo se describe el Reloj de Tiempo Real DS1307 de Dallas Semiconductors. La aplicación se realiza utilizando un microcontrolador PIC18F452 y el software, utilizando el PICC de CCS. Un reloj de tiempo real es un dispositivo muy útil en cualquier aplicación donde se quiera dejar constancia de fecha y hora de algún evento.

El DS1307 es un RTC cuya interfaz con el microcontrolador se realiza utilizando el bus I²C (Inter-Integrated Circuit), desarrollado por Phillips Semiconductors.

Características Generales del DS1307

El DS1307 es un poderoso reloj calendario en BCD, cuyas características más destacadas son las siguientes:

- Reloj de tiempo real que cuenta los segundos, los minutos, las horas, la fecha, el mes, el día de la semana, y el año, con compensación de años bisiestos, válido hasta el año 2100
- Formato de 12 Horas con indicador AM/PM ó de 24 horas
- Protocolo I²C
- 56 bytes de RAM no volátil, para almacenamiento de datos
- Señal de onda cuadrada programable
- Circuitos internos de respaldo para la alimentación automático
- Bajo consumo de potencia: menor a 500nA en modo respaldo, a 25 grados Centígrados
- Sólo 8 pines

Descripción de los Pines

La figura 1, muestra la asignación de los pines del DS1307 los cuales se describen a continuación:

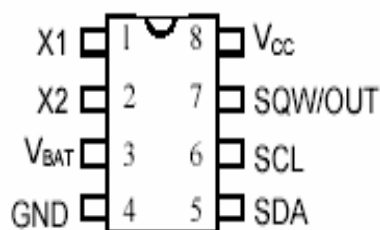


Fig. 1 Asignación de pines del DS1307

VCC, GND: La alimentación DC es proporcionada a este circuito a través de estos pines. VCC es la entrada de +5VDC, mientras que GND es la referencia.

VBAT: Entrada de alimentación de una pila estándar de litio de 3 Voltios. El voltaje debe estar entre 2.5 y 3.5 voltios para una operación apropiada.

SCL: Entrada de reloj para sincronizar la transferencia de datos en la interfaz serial.

SDA: Entrada/salida de datos para la interfaz I²C. Este pin es de drenaje abierto, por lo que requiere de una resistencia pull-up externa.

X1,X2: Conexiones para un cristal de cuarzo estándar de 32.768 Hz.

SQW/OUT: Salida para generar una de cuatro posibles frecuencias de salida: 1Hz, 4KHz, 8KHz ó 32KHz. Este pin también es de drenaje abierto, por lo que requiere de una resistencia pull-up externa.

Circuito Típico de aplicación

La figura 2, muestra un diagrama esquemático de una aplicación típica para este circuito. Se observan las resistencias pull-up en los pines SDA, SCL y SQW/OUT que normalmente son de 4.7 KOhm. Usa un cristal de cuarzo estándar de 32.768Hz. La alimentación es normalmente 5 voltios DC y utiliza una pila de litio de 3V para el respaldo de la alimentación.

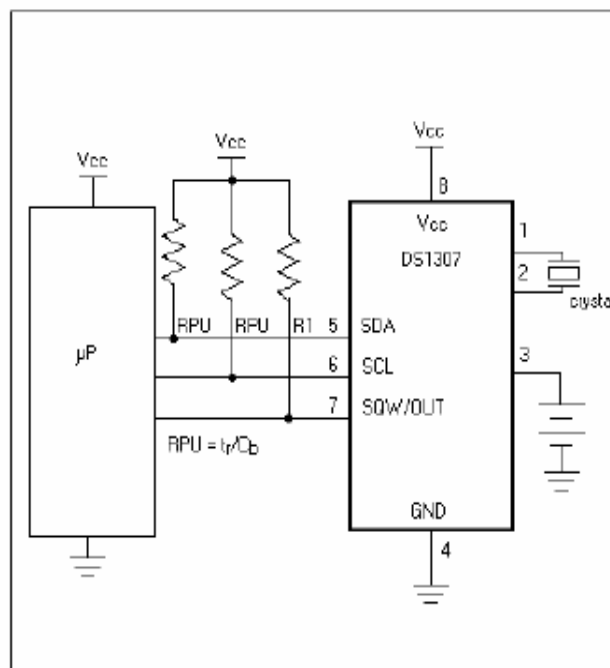


Fig. 2 Esquema de operación típico

Mapa de Direcciones

En la figura 3, se presenta el mapa de direcciones para los registros del Reloj de Tiempo Real y la RAM del DS1307. Los registros se localizan en las direcciones 0x00 hasta 0x07, mientras que la RAM va desde 0x08 hasta 0x3F.

El puntero de direcciones es de 6 bits, de tal manera que cuando se realiza accesos múltiples y se alcanza la dirección 0x3F, ó fin del espacio de la RAM, el puntero regresa a 0 con el siguiente acceso.

00H	SECONDS
	MINUTES
	HOURS
	DAY
	DATE
	MONTH
	YEAR
07H	CONTROL
08H	RAM
3FH	56 x 8

Fig. 3 Mapa de direcciones

Reloj Calendario

La información del tiempo y el calendario esta en formato BCD, y es obtenida leyendo los registros apropiados los cuales se muestran en la figura 4. Los bits marcados con 0 en esta figura son intrascendentes para el funcionamiento del circuito.

El bit 7 del registro 0 es el bit CH (Clock Halt). Cuando este bit se establece a 1, el oscilador se deshabilita, deteniendo el funcionamiento del reloj. Cuando se establece a 0, el oscilador es habilitado, y el circuito funciona normalmente.

El RTC, como se mencionó puede trabajar en formato de 12 horas ó de 24 horas siendo el bit 6 del registro 2 el encargado de determinar esta función. Cuando se establece a 1, funciona en modo 12 horas, y el bit 5 es el indicador AM/PM, siendo 1 cuando es PM. En formato 24 horas, el bit 5 de este registro es la parte más significativa de las decenas de horas.

	BIT7							BIT0	
00H	CH	10 SECONDS			SECONDS				00-59
	0	10 MINUTES			MINUTES				00-59
	0	12 / 24	10 HR / A/P	10 HR	HOURS				01-12 / 00-23
	0	0	0	0	0	DAY			1-7
	0	0	10 DATE		DATE				01-28/29 / 01-30 / 01-31
	0	0	0	10 MONTH	MONTH				01-12
	10 YEAR				YEAR				00-99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	

Fig. 4 Registros DS1307

El registro 7 es el registro de control. Este registro es utilizado para determinar el funcionamiento del pin de salida SQW/OUT. El bit 7 ó OUT determina el nivel de salida cuando la salida de onda cuadrada esta deshabilitada (SQWE = 0). Será alto si el bit OUT = 1 y bajo si OUT = 0. El bit 4 ó SQWE, habilita la salida del oscilador de onda cuadrada, cuya frecuencia dependerá de los bits RS1 y RS0. La tabla 1, muestra las frecuencias que se pueden obtener según la configuración de los bit RS1 y RS0.

RS1	RS0	Frecuencia de salida
0	0	1Hz
0	1	4KHz
1	0	8KHz
1	1	32KHz

Tabla 1 Frecuencia de Salida pin SQW/OUT

Bus I²C

El bus I²C (Inter-Integrated Circuit), fue desarrollado por Phillips Semiconductors con el propósito de comunicar elementos que se encuentren en una misma tarjeta o circuito. Utiliza un protocolo serial sincrónico que solamente requiere de dos líneas, SDA (Serial Data Line) y SCL (Serial Clock Line) las cuales se comportan bidireccionalmente. Estas líneas se conectan al positivo de la fuente de alimentación a través de resistencias pull-up. Cuando ambas líneas están libre permanecen en un estado lógico alto. Para el modo I²C del módulo MSSP estas líneas son típicamente RC3 y RC4 respectivamente. Estas son las mismas líneas usadas por el modo SPI del MSSP. La figura 5. muestra un ejemplo de configuración del bus I²C en un sistema.

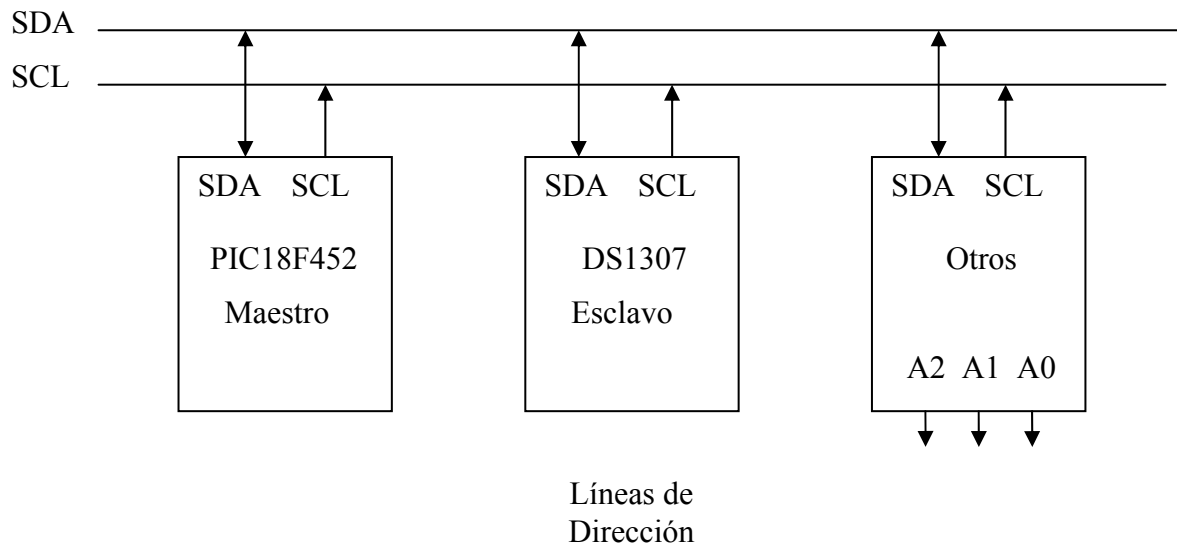


Fig. 5 Esquema de configuración I²C

Usar sólo dos líneas para la comunicación, requiere de una aproximación diferente para el flujo de datos. En SPI, los datos fluyen del maestro al esclavo y simultáneamente del esclavo hacia el maestro. En el modo I²C se usa un sistema secuencial en el cual el flujo de datos ocurre del maestro hacia el esclavo sobre la línea SDA y luego, si es necesario (en caso de que el maestro necesite leer del esclavo), los datos fluyen del esclavo hacia el maestro sobre la misma línea SDA. En cualquiera de los casos el maestro siempre suministra la señal de reloj necesaria para la comunicación Maestro/Esclavo a través de la línea SCL.

Cada dispositivo que se conecta al bus tiene asignada una dirección compuesta de 7 bits. El DS1307 tiene establecida internamente la dirección en b1101000. Para facilitar la programación el bit siguiente, correspondiente a lectura/escritura, se incorpora a los siete bits anteriores, a fin de completar un byte. En el caso de escritura la dirección del DS1307 es entonces 0xD0 y en el caso de lectura 0xD1.

En otros dispositivos como el caso particular de una memoria 24LC256, la dirección de 7 bits esta compuesta de la siguiente manera: Los cuatro bits más significativos corresponden al valor binario 1010, dispuesto por el fabricante para este tipo de dispositivo. Los tres siguientes corresponden a los bits de selección del dispositivo, lo cual se realiza por hardware usando los pines A0, A1 y A2. Si por ejemplo estos pines son puestos a común (cero) el valor binario de la

dirección del dispositivo es b1010000. Luego, en caso de escritura la dirección de la memoria es entonces 0xa0 y en el caso de lectura 0xa1.

La figura 6 y la figura 7, muestran la actividad del maestro y del esclavo cuando el maestro realiza una operación de escritura sobre el esclavo y cuando el maestro realiza una operación de lectura sobre el esclavo. En ambas operaciones, la parte sombreada corresponde a la actividad del maestro sobre el bus y la no sombreada a la actividad del esclavo sobre el bus. El bit de acuse de recibo (ACK) siempre lo genera el dispositivo que recibe los datos, excepto cuando el maestro se comporta como receptor, éste no debe generar el último acuse de recibo (ACK), en su lugar debe enviar un no acuse de recibo (NACK).

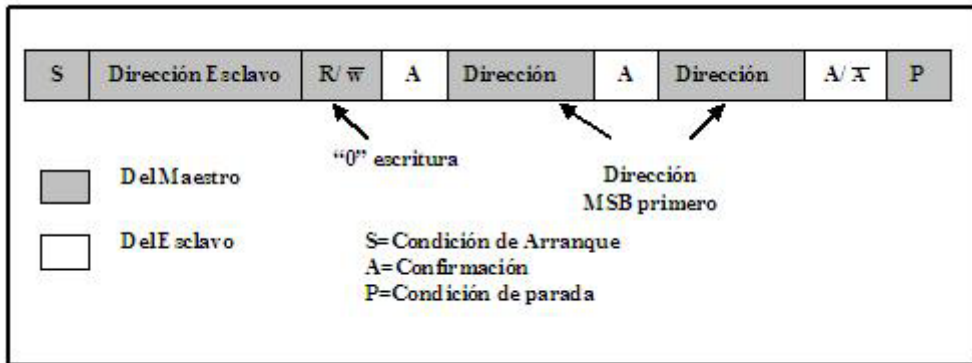


Fig. 6 Operación de escritura

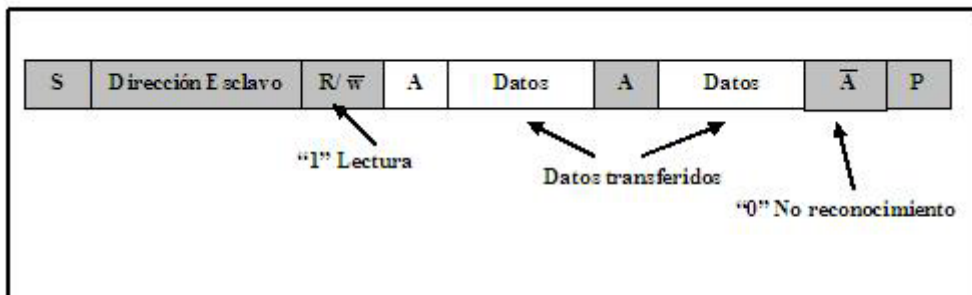


Fig. 7 Operación de lectura

El esquema general del protocolo I²C es:

1. El maestro aplica la condición de start al bus. Esto alerta a los esclavos que algo está por venir y entonces, estos sincronizan el flujo de datos.
2. El maestro entonces envía un byte con la dirección del esclavo que va a ser activado. Todos los esclavos reciben este byte, pero sólo el esclavo cuya dirección se iguale entra en actividad, los otros ignoran la comunicación hasta ver una nueva condición de start en el bus. Los 7 bits más significativos de este byte es la dirección del esclavo, el bit menos significativo controla la dirección de la próxima transferencia (uno o más bytes). Un "0" indica una transferencia de maestro a esclavo, y un "1" indica una transferencia de esclavo a maestro. Este bit normalmente se denomina R/W.
3. Los datos entonces son transferidos según el bit R/W.
4. Los datos pueden ser transferidos en ambas direcciones durante un intercambio reenviando el pulso de start y cambiando el bit de R/W en el byte de dirección.
5. Finalmente, el maestro debe enviar la condición de stop al bus para concluir la transferencia.

El compilador C de CCS, incorpora funciones de alto nivel que nos permiten el manejo del bus I²C. El bus I²C puede implementarse usando el hardware del microcontrolador, en cuyo caso es

necesario usar las patitas RC3 y RC4 destinadas para ello y usar la opción FORCE_HW de la directiva `#use i2c()` o puede implementarse por software usando patitas distintas, en cuyo caso el compilador genera el código necesario. Para ello deberá indicar las patitas escogidas en la directiva `#use i2c()` y no usar la opción FORCE_HW.

Hardware

El hardware esta basado en un microcontrolador PIC18F452 y se muestra en la fig. 8. Al pin RB6 se conecta un switch normalmente abierto el cual al estar presionado en el momento de arranque, causa que se entre en el modo de configuración, donde se podrá suministrar la fecha y la hora vía teclado del PC utilizando Hyperterminal de Windows. El diodo y la resistencia que se conectan entre la batería y VCC sirve para la carga de esta cuando se esta operando normalmente.

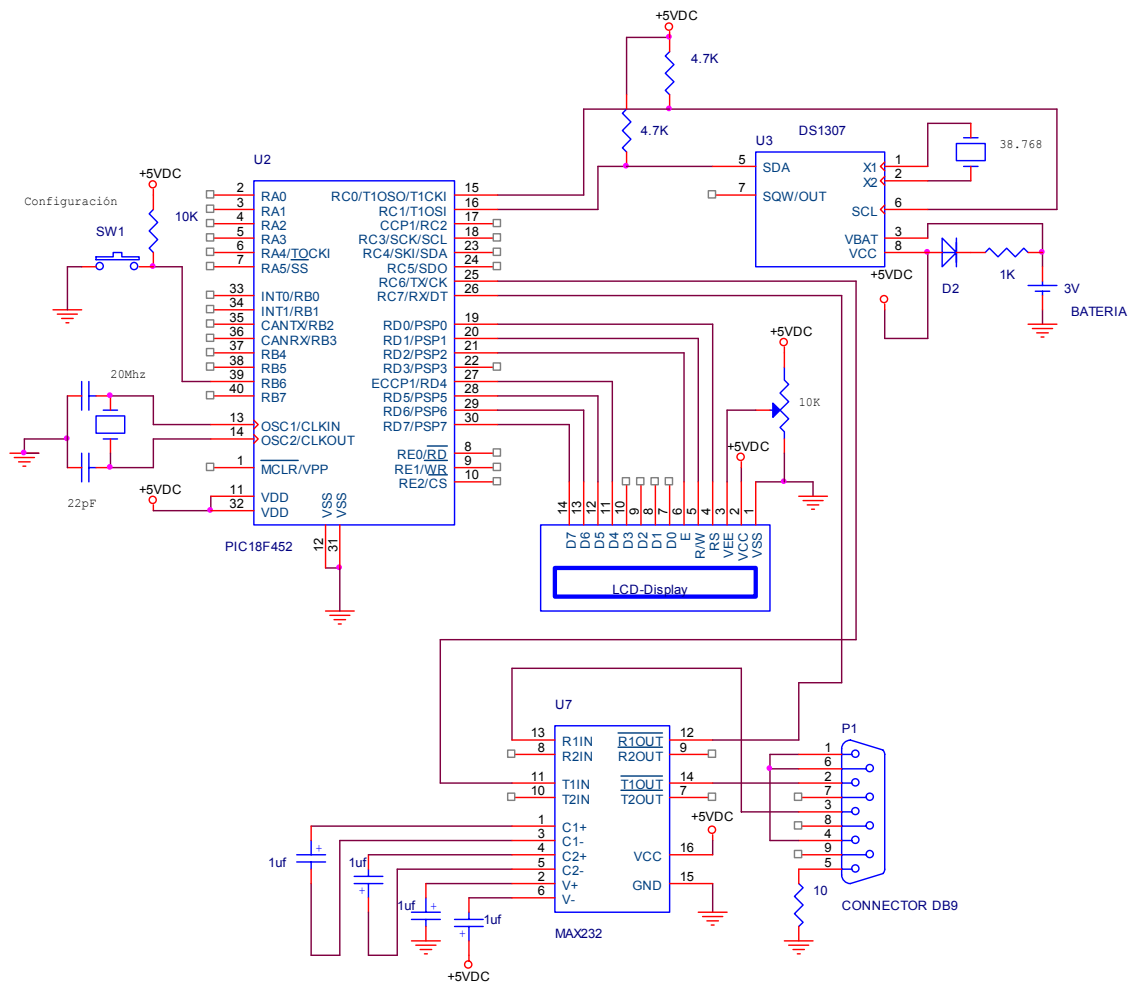


Fig. 8 Hardware Reloj de Tiempo Real

Software

El software esta escrito en lenguaje C utilizando el compilador PICC de CCS. Se anexan una serie de rutinas en DS1307.h las cuales son una adaptación propia de rutinas suministradas por PCM programmer.

```
////////////////////////////////////
////                               RTC.C                               ////
////                               ////
////////////////////////////////////
#include <18f452.h>
#fuses HS,NOWDT,NOPROTECT,PUT,BROWNOUT,NOLVP
#use fast_io(A)
#use fast_io(B)
#use fast_io(C)
#use fast_io(D)
#use fast_io(E)
#define M_SDA PIN_C1
#define M_SCL PIN_C0
#use i2c(master,sda=M_SDA, scl=M_SCL, RESTART_WDT, FAST)
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6, RCV=PIN_C7,ERRORS)

#byte PORTA = 0xF80
#byte PORTB = 0xF81
#byte PORTC = 0xF82
#byte PORTE = 0xF84

#define ALL_OUT 0
#define ALL_IN 0xff

/*****
* Variables Globales
*****/
int DD, MM, AA, HH, MI;
short escaped;

/*****
* Prototipos
*****/
void init_i2c_bus(void);
void LCD_Setup(void);
void displays(int b);
void desplegar_time(void);
int getnum(void);

#include "lcd-pd.h"
#include "ds1307.h"

/*****
* Programa Principal
*****/
void main() {
    int i;
    port_b_pullups(true);
    set_tris_D(ALL_OUT);
    LCD_setup();
    disp_serial=false;
    disp_lcd = true;
    delay_ms(2000);
    init_i2c_bus();
    ds1307_init();

    if(!bit_test(PORTB,6)) { //boton en RB6 presionado
        escaped = 0;
        printf("\r\nConfiguración Reloj de Tiempo Real");
        printf("\n\rDia:");
        i = getnum();
        if(!escaped)
            gca_ds1307_regs[4] = i;
        if(!escaped) {
            printf("\r\nMes:");
            gca_ds1307_regs[5] = getnum();
        }
    }
}
```

```

        if(!escaped) {
            printf("\n\rAño:");
            gca_ds1307_regs[6] = getnum();
        }
        if(!escaped) {
            printf("\r\nHora:");
            gca_ds1307_regs[2] = getnum();
        }
        if(!escaped) {
            printf("\r\nMinutos:");
            gca_ds1307_regs[1] = getnum();
            printf("\n\r");
            gca_ds1307_regs[0] = 0;
            gca_ds1307_regs[3] = 0;
            ds1307_set_date_time();
        }
        if(escaped)
            printf("\r\nEscape!!!");
    }

while(1) {
    delay_ms(1000);
    ds1307_read_date_time();
    MI = gca_ds1307_regs[1];
    HH = gca_ds1307_regs[2];
    DD = gca_ds1307_regs[4];
    MM = gca_ds1307_regs[5];
    AA = gca_ds1307_regs[6];
    desplegar_time();

}

/*****
* desplegar date_time
*****/
void desplegar_time(void)
{
    printf(displays,"\rFecha  Hora      ");
    printf(displays,"\n%02d/%02d/20%02d %02d:%02d    ",DD,MM,AA,HH,MI);
}

/*****
* Inicializa el bus i2c
*****/
void init_i2c_bus(void) {
    output_float(m_scl);
    output_float(m_sda);
}

/*****
* Toma un numero de 8 bits desde la consola
* return con cualquier tecla no numerica (excepto backspace)
*****/
int getnum(void)
{
    int val=0;
    char c, buff[2];
    int n=0;

    do {
        c = getchar();
        if (c == 0x1b) //escape
            escaped = 1;
        else if (c>='0' && c<='9')
        {
            if (n < sizeof(buff))
            {
                buff[n++] = c;
                putchar(c);
            }
        }
        else if (c=='\b' || c==0x7f) //backspace
        {
            if (n > 0)
            {
                n--;
            }
        }
    } while(1);
}

```

```

        putchar('\b');
        putchar(' ');
        putchar('\b');
    }
}
else
    c = 0;
} while (c && !escaped);
    val = (((buff[0]-'0')*10) + (buff[1] - '0'));
return(val);
}

// DS1307.h Routines
//i2c addresses
#define DS1307_I2C_WRITE_ADDR 0xd0
#define DS1307_I2C_READ_ADDR 0xd1
// DS1307 register offsets
#define DS1307_SECONDS_REG 0
#define DS1307_MINUTES_REG 1
#define DS1307_HOURS_REG 2
#define DS1307_DAY_OF_WEEK_REG 3
#define DS1307_DATE_REG 4
#define DS1307_MONTH_REG 5
#define DS1307_YEAR_REG 6
#define DS1307_CONTROL_REG 7

#define DS1307_DATE_TIME_BYTE_COUNT 7 // includes bytes 0-6
#define DS1307_NVRAM_START_ADDR 8
// We disable the SQWV output, because it uses a lot of battery current
// when it's enabled.
#define DS1307_CONTROL_REG_INIT_VALUE 0x80 // Disable SQWV, set Out = O.C.
// #define DS1307_CONTROL_REG_INIT_VALUE 0x13 // 32.768 KHz output

//-----
// GLOBAL VARIABLES
// This is a global array to hold data which is passed to/from the
// functions that set the ds1307 date and time.
char gca_ds1307_regs[DS1307_DATE_TIME_BYTE_COUNT];
//-----
// FUNCTION PROTOTYPES
void ds1307_set_date_time(void);
void ds1307_read_date_time(void);
void ds1307_write_byte(char addr, char value);
char ds1307_read_byte(char addr);
char bcd2bin(char bcd_value);
char bin2bcd(char binary_value);
void ds1307_init(void);

/*****
/*  inicializa DS1307 */
/*****/
void ds1307_init(void)
{
    ds1307_read_date_time();
    ds1307_set_date_time();
}

/*****
/*  Set Date Time */
/*****/
void ds1307_set_date_time(void)
{
    char i;
    for(i = 0; i < 7; i++)
    {
        gca_ds1307_regs[i] = bin2bcd(gca_ds1307_regs[i]);
    }
    gca_ds1307_regs[DS1307_SECONDS_REG] &= 0x7f;
    gca_ds1307_regs[DS1307_HOURS_REG] &= 0x3f;
    disable_interrupts(GLOBAL);
    i2c_start();
    i2c_write(DS1307_I2C_WRITE_ADDR);
    i2c_write(DS1307_SECONDS_REG);

```

```

        for(i = 0; i < 7; i++)
        {
            i2c_write(gca_ds1307_regs[i]);
        }
        i2c_write(DS1307_CONTROL_REG_INIT_VALUE);
        i2c_stop();
        enable_interrupts(GLOBAL);
    }

/*****
/*      Read Date Time      */
/*****/
void ds1307_read_date_time(void)
{
    char i;
    disable_interrupts(GLOBAL);
    i2c_start();
    i2c_write(DS1307_I2C_WRITE_ADDR);
    i2c_write(DS1307_SECONDS_REG);
    // Start reading at the Seconds reg
    i2c_start();
    i2c_write(DS1307_I2C_READ_ADDR);
    // Read the 7 bytes from the ds1307. Mask off unused bits.
    gca_ds1307_regs[DS1307_SECONDS_REG] = i2c_read() & 0x7f;
    gca_ds1307_regs[DS1307_MINUTES_REG] = i2c_read() & 0x7f;
    gca_ds1307_regs[DS1307_HOURS_REG] = i2c_read() & 0x3f;
    gca_ds1307_regs[DS1307_DAY_OF_WEEK_REG] = i2c_read() & 0x07;
    gca_ds1307_regs[DS1307_DATE_REG] = i2c_read() & 0x3f;
    gca_ds1307_regs[DS1307_MONTH_REG] = i2c_read() & 0x1f;
    gca_ds1307_regs[DS1307_YEAR_REG] = i2c_read(0);
    // No ACK on last byte
    i2c_stop();
    enable_interrupts(GLOBAL);
    // Now convert the data from BCD to binary. Do it after reading the bytes,
    // so that the i2c reads can be done quickly.
    for(i = 0; i < 7; i++)
    {
        gca_ds1307_regs[i] = bcd2bin(gca_ds1307_regs[i]);
    }
}

/*****
/*      Read one byte at the specified address.      */
/*      */
/*      This function is used to access the control byte or the NVRAM byte*/
/*****/
char ds1307_read_byte(char addr)
{
    char retval;
    disable_interrupts(GLOBAL);
    i2c_start();
    i2c_write(DS1307_I2C_WRITE_ADDR);
    i2c_write(addr);
    i2c_start();
    i2c_write(DS1307_I2C_READ_ADDR);
    retval = i2c_read(0);
    // We're only reading a single byte, so no ACK on it.
    i2c_stop();
    enable_interrupts(GLOBAL);
    return(retval);
}

/*****
/*      Write one byte to the DS1307.      */
/*      */
/*      This function is used to access the control byte or the NVRAM byte*/
/*****/
void ds1307_write_byte(char addr, char value)
{
    disable_interrupts(GLOBAL);
    i2c_start();
    i2c_write(DS1307_I2C_WRITE_ADDR);
    i2c_write(addr);
    i2c_write(value);
    i2c_stop();
    enable_interrupts(GLOBAL);
}

```

```

/*****
/* This function converts an 8 bit binary value to an 8 bit BCD value*/
/* The input range must be from 0 to 99. */
*****/
char bin2bcd(char binary_value)
{
    char temp;
    char retval;
    temp = binary_value;
    retval = 0;
    while(1)
    {
// Get the tens digit by doing multiple subtraction of 10 from the binary value
        if(temp >= 10)
        {
            temp -= 10;
            retval += 0x10;
        }
        else
// Get the ones digit by adding the remainder.
        {
            retval += temp;
            break;
        }
    }
    return(retval);
}

/*****
/* This function converts an 8 bit BCD value to an 8 bit binary value*/
/* The input range must be from 00 to 99. */
*****/
char bcd2bin(char bcd_value)
{
    char temp;
    temp = bcd_value;
    temp >>= 1;
// Shifting upper digit right by 1 is same as mult. by 8
    temp &= 0x78; // Isolate the bits for the upper digit
// Now return: (Tens *8) + (Tens *2) + Ones
    return(temp + (temp >> 2) + (bcd_value & 0x0f));
}

```