Circuitos Digitales y Microcontroladores

GRUPO 6

Chanquía, Joaquín (02887/7)

Forden Jones, Ian E. W. (02543/3)

Ollier, Gabriel (02958/4)

ÍNDICE

1. INTRODUCCIÓN

1.1 - Enunciado

Implementar un registrador de temperatura y humedad relativa ambiente utilizando el sensor DHT11, el módulo RTC DS3231 y el kit del MCU conectado a una PC por medio de la interfaz USB.

El sensor DHT11 estará conectado al terminal PORTC0 del MCU, mientras que el módulo RTC se conectará mediante la interfaz I2C del mismo. Para resolver el problema deberá implementar los drivers para el control del sensor, para el control del módulo RTC y para la comunicación serie asincrónica por UART.

A continuación, se muestran los requerimientos que el sistema debe cumplir:

A) El MCU deberá encuestar al sensor para obtener una medida de la temperatura y la humedad relativa cada 2seg.
B) Utilizando el módulo RTC el MCU completará el registro agregando la fecha y hora actual a cada una de las medidas obtenidas con el sensor.
C) Por último, realizará un formateo de los datos para transmitir el mensaje a una terminal serie en PC. Por ejemplo, el formato puede ser “TEMP: 20 °C HUM: 40% FECHA: 10/06/24 HORA:15:30:56\r\n”
D) El envío de datos se podrá detener o reanudar desde la PC presionando la tecla ‘s’ o ‘S’ sucesivamente.
E) La comunicación serie asincrónica deberá implementarse utilizando interrupciones de recepción y transmisión del periférico UART0.

1.2 - Interpretación

El objetivo del trabajo es crear un sistema que registre y transmita datos de temperatura y humedad relativa del ambiente. Este sistema utilizará un sensor DHT11 para obtener las medidas, un módulo RTC DS3231 para agregar la marca de tiempo a las medidas, y un microcontrolador (MCU) que se conectará a una PC mediante USB para enviar los datos en un formato legible. Se debe conectar físicamente el sensor de temperatura y humedad (DHT11) al pin PORTC0 del microcontrolador. Esto implica que la programación del MCU debe manejar las señales de entrada/salida en este pin para leer los datos del sensor y por parte del módulo de reloj en tiempo real (RTC DS3231) debe conectarse al microcontrolador usando la interfaz I2C, lo que implica que se utilizarán los pines específicos de I2C en el MCU.

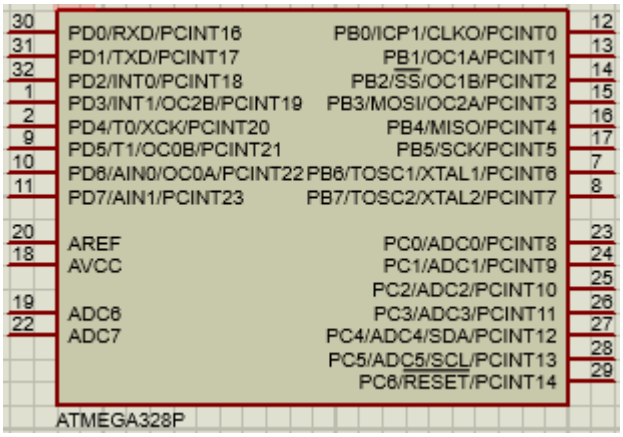
El microcontrolador debe programarse para leer los datos del sensor DHT11 cada 2 segundos. Esto implica el uso del temporizador permita la ejecución periódica de la lectura del sensor. Después de obtener los datos de temperatura y humedad, el MCU debe consultar el RTC para obtener la fecha y hora actual. Estos datos de tiempo deben combinarse con las medidas del sensor para crear un registro completo que incluya tiempo, temperatura y humedad y finalmente los datos obtenidos (temperatura, humedad, fecha y hora) deben organizarse en una cadena de texto con un formato específico para ser transmitidos a través de la interfaz UART a una PC. El formato debe ser: “TEMP: 20 °C HUM: 40% FECHA: 10/06/24 HORA:15:30:56\r\n”. La comunicación UART debe ser manejada usando interrupciones tanto para la recepción como para la transmisión de datos. Esto significa que el microcontrolador

debe estar configurado para generar interrupciones cuando haya datos disponibles para leer o cuando esté listo para enviar datos

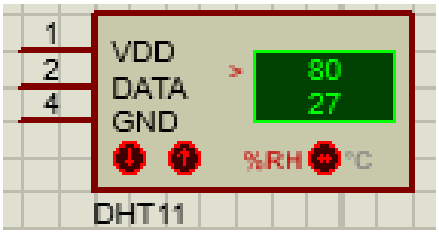
Además, el sistema debe incluir una funcionalidad que permita al usuario de la PC controlar la transmisión de datos ya que, si el usuario presiona la tecla 's' o 'S', la transmisión debe detenerse o reanudarse. Esto implica que el MCU debe estar programado para reconocer estas entradas y responder adecuadamente.

Componentes utilizados:

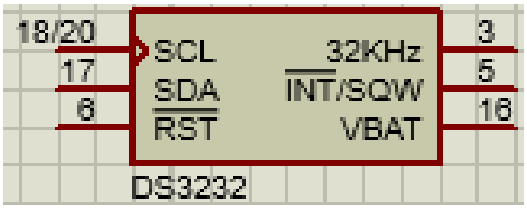
MCU Atmega328p



Sensor DHT11



Módulo RTC DS3232



Aclaración: Si bien en la consigna se pide utilizar el módulo RTC DS3231, el simulador Proteus no tiene disponible este módulo para su uso. Debido a esto en la simulación se utilizará el módulo RTC DS3232 cuyo principio de funcionamiento es igual al del DS3231.

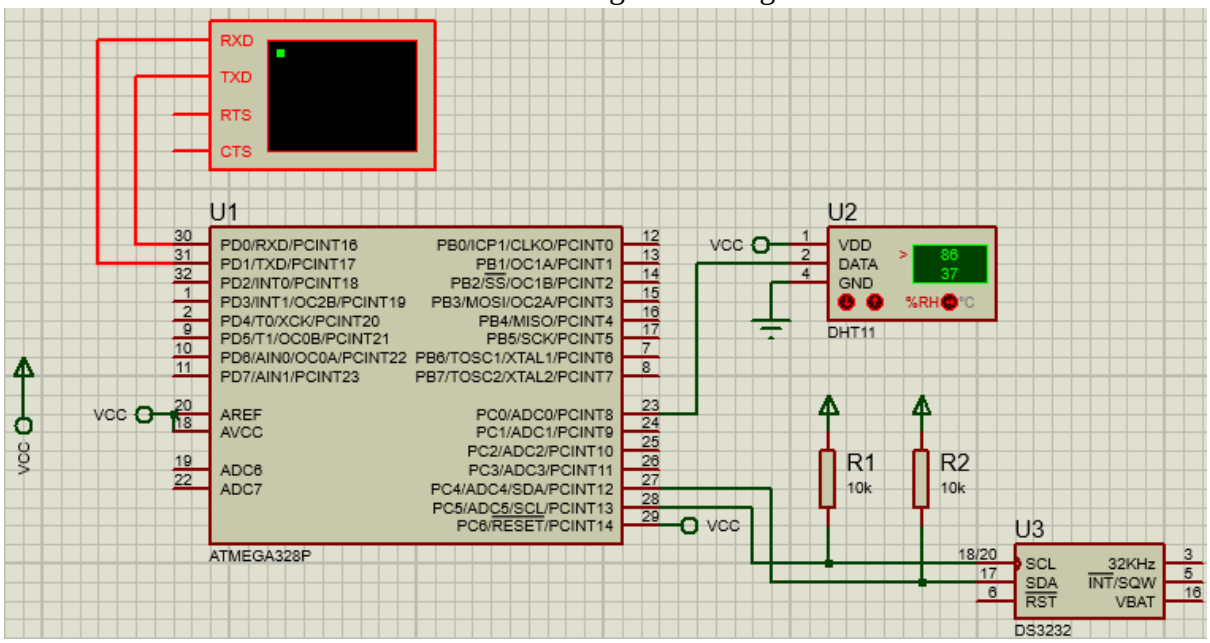
2. RESOLUCIÓN

2.1 Conexionado de los periféricos

Se conectarán el sensor y el RTC al MCU utilizando la información provista en la consigna para el caso del sensor y las etiquetas de los puertos del MCU para el RTC. Para el sensor se tiene que el puerto de datos debe estar conectado al puerto PC0 del MCU y para el RTC se tiene que sus puertos SCL (puerto de clock) y SDA (puerto de datos) deben ir conectados a los puertos PC5 y PC4 respectivamente pues los mismos están etiquetados como SCL y SDA. Además, en el caso del sensor se puede deducir a que se deben conectar las terminales VDD y GND en base a sus nombres, y en el caso del RTC se tiene que, según la hoja de datos, los puertos SCL y SDA que se deben conectar al MCU requieren una resistencia externa de pull-up mientras que los demás puertos del RTC pueden dejarse desconectados.

Para finalizar con el conexionado se tomará la libertad de conectar una terminal virtual al MCU con el fin de visualizar el correcto funcionamiento de los módulos y la correcta comunicación entre el MCU y la PC a la cual se conectará. La terminal virtual requiere que sus puertos RXD y TXD se conecten a aquellos puertos del MCU etiquetados con esos nombres de forma intercalada, es decir se conecta un transmisor con un receptor, los cuales son PD1 y PD0 respectivamente.

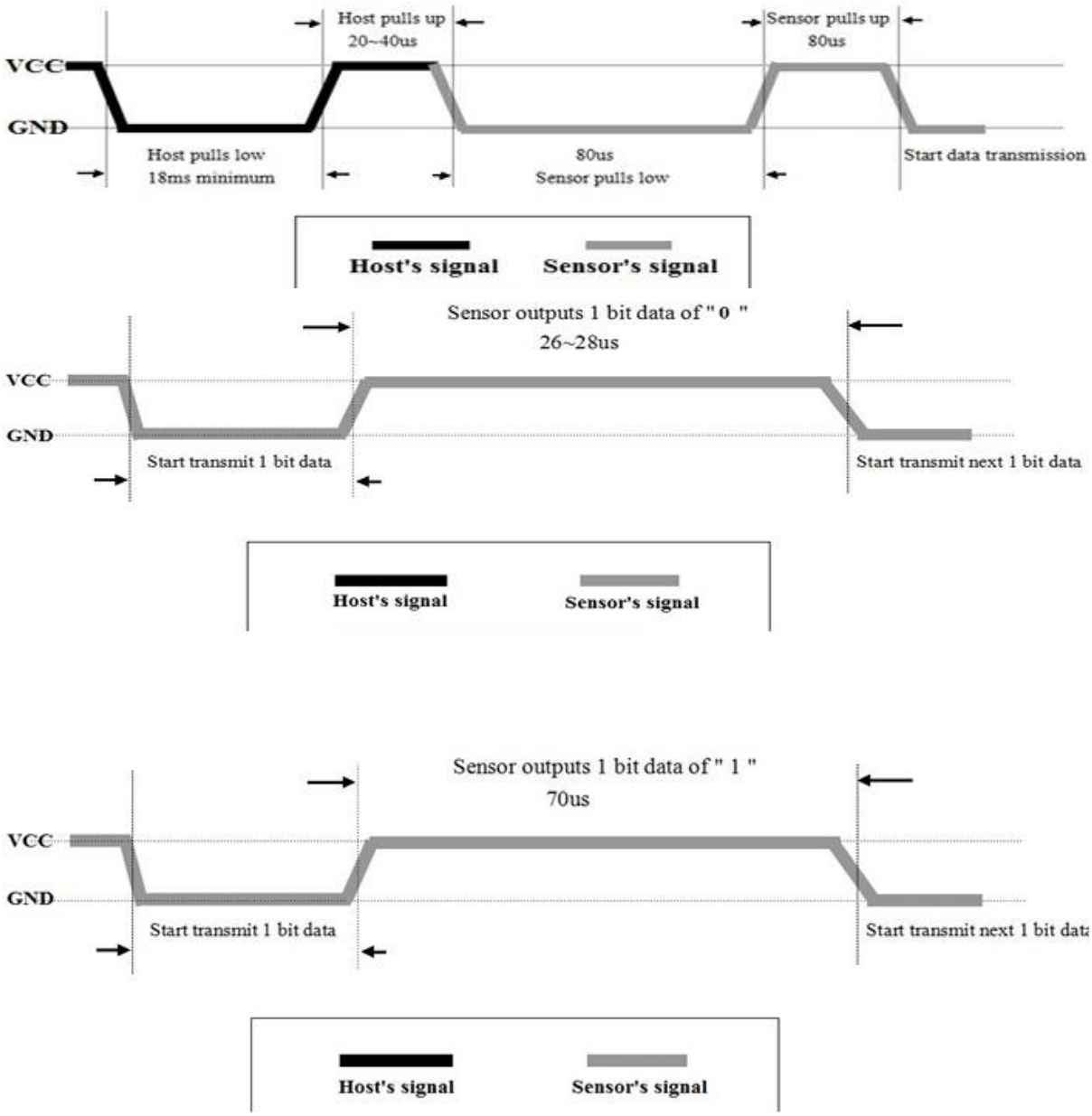
El conexionado final se muestra en la siguiente imagen:



2.2 Funcionamiento de los periféricos

DHT11

Las siguientes imágenes se obtuvieron del datasheet del DHT11:



Las imágenes muestran el método mediante el cual el sensor y el MCU se comunican para realizar la transferencia de datos. Dicha transferencia inicia cuando el MCU pone en bajo, por al menos 18 ms, el valor del puerto por el cual el DHT11 está conectado (el puerto debe haber estado en un nivel alto anteriormente) y luego lo vuelve a poner en un valor alto, tras lo cual, después de un intervalo de entre 20 a 40 us, el sensor procederá a poner en bajo el valor del puerto y luego lo volverá a poner en alto durante 80 us para cada valor. A partir de este momento inicia la transmisión de 40 bits de datos por parte del sensor donde cada bit inicia cuando se coloca un valor bajo en el puerto y después se lo pasa a un valor alto por una cierta cantidad de tiempo la cual determina si el bit representa un 1 (70 us) o un 0 (26 a 28 us). Los bits enviados y que representan se muestran en la siguiente imagen donde el orden de envío es de izquierda a derecha:

0010 0001	**0000 0000**	**0001 1010**	**0000 0000**	**0011 1011**
Integral part of RH	Decimal part of RH	Integral part of T	Decimal part of T	check sum

Remarks: The decimal part of RH and T is always 0000 0000.

RTC DS3231

La comunicación entre el RTC DS3231 y el MCU se realizará mediante la interfaz I2C cuyo protocolo de comunicación es el siguiente:

1) El bus se encuentra en estado IDLE cuando ambas líneas SDA y SCL están en ALTO
2) La comunicación la debe comenzar el maestro con una condición de START
3) Luego debe especificar la dirección del dispositivo esclavo (caso 7bits)
4) A continuación, debe establecer si la transferencia es de lectura/escritura (1 bit más)
5) El esclavo que corresponda a la dirección presentada debe responder con una condición ACK (bit de Acknowledge)
6) A partir de aquí, el dispositivo que corresponda debe enviar los datos según condición R/W
7) Y el dispositivo que recibe dichos datos (maestro o esclavo) debe establecer una confirmación: ACK o NACK. Cada byte recibido debe tener su correspondiente ACK (9no pulso de reloj) y un NACK por parte del maestro (en el caso de la operación read) significa “fin de la comunicación” y el esclavo libera SDA para que el maestro presente la condición de STOP
8) El maestro debe finalizar la comunicación con una condición de STOP

A continuación, se muestran los registros utilizados para configurar la frecuencia del CLOCK (SCL) y la fórmula con la que se determina dicha frecuencia:

$$SCL\ \ frequency = \frac{XTAL}{16 + 2 \times TWBR \times 4^{TWPS}}$$

Aclaración: XTAL es la frecuencia del microprocesador, en este caso 16MHz.

El último registro que se utilizará es el registro de control:

- TWINT: TWI Interrupt flag
- TWEA: TWI Enable Acknowledge bit
  - 1:ACK, 0:NACK
- TWSTA: TWI Start condition bit
- TWSTO: TWI Stop condition bit
- TWWC: TWI Write Collision flag
- TWEN: TWI Enable bit
- TWIE: TWI Interrupt Enable

Teniendo en cuenta el funcionamiento de la interfaz I2C, se explica a continuación el método de comunicación entre el MCU y el DS3231, la cual se divide en 2 etapas:

Etapa de escritura (Slave receiver mode o DS3231 write mode): se debe inicializar al RTC escribiendo una fecha y hora en su RAM interna la cual, una vez finalizada la inicialización, el módulo actualizará constantemente siempre y cuando esté conectado a una fuente de alimentación o tenga instalado una pila cargada. Esta etapa inicia cuando el dispositivo maestro (en este caso, el MCU) envía una señal de START al dispositivo esclavo (en este caso, el RTC) la cual es seguida por un byte el cual contiene la dirección de 7 bits del DS3231 (1101000) más un bit de dirección que indica si la operación es una lectura (1) o escritura (0) y el cual ocupa la posición del bit menos significativo. El esclavo responde con un bit de acknowledge y el maestro, tras recibir ese acknowledge, le envía un byte el cual indica la dirección de memoria del RTC a partir de la cual se empezará a escribir. Una vez el maestro recibe un acknowledge por el byte anterior, el mismo es libre de enviar cuantos bytes de datos

quiera (la dirección de memoria en la que se escribe aumenta en uno cuando el RTC recibe un byte) y debe enviar una señal de STOP para finalizar la operación.

Etapa de lectura (Slave transmitter mode o DS3231 read mode): una vez se haya inicializado el RTC se puede consultar la fecha y horario que hay en su memoria. La operación es muy similar a la escritura excepto por el hecho de que antes de iniciarla se debe escribir la dirección a partir de la cual se debe iniciar la lectura, el bit de dirección se pondrá en 1 y la lectura continuará hasta que el RTC reciba un not acknowledge.

Es importante aclarar que los datos en la RAM del DS3231 están codificados BCD (decimal codificado en binario) y la organización de los datos en memoria es:

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{\text{AM}}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{\text{AM}}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99

USART (Universal Synchronous and Asynchronous Receiver Transmitter)

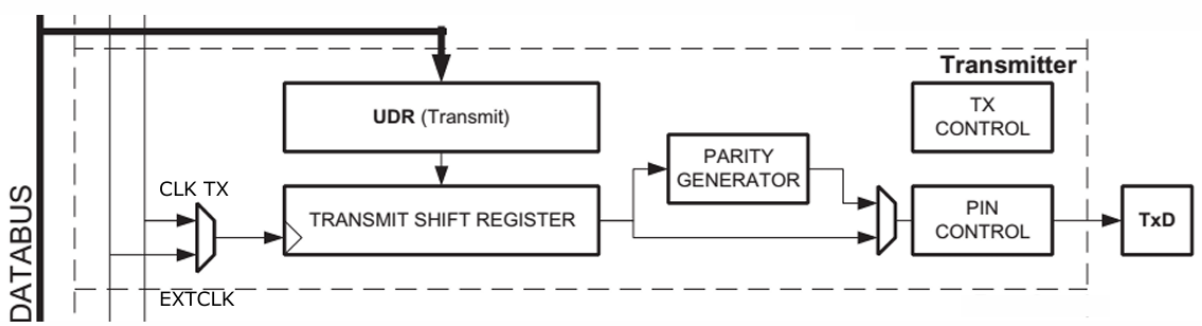
El USART está conformado por 3 elementos: el generador de reloj, el transmisor y el receptor. A continuación, se explicará la configuración y funcionamiento de cada uno.

El generador de reloj se configura utilizando el registro UBRR, cuyo tamaño es de 16 bits y se divide en UBRRH (8 bits más significativos) y UBRRL (8 bits menos significativos), y el bit 1 (U2X0) del registro USCRA. El valor del registro UBRR se usa para calcular la tasa de baudios del reloj mediante la fórmula siguiente:

$$\text{Baud rate} = \frac{\text{Fosc}}{(\text{UBRR}+1) \times 16}$$

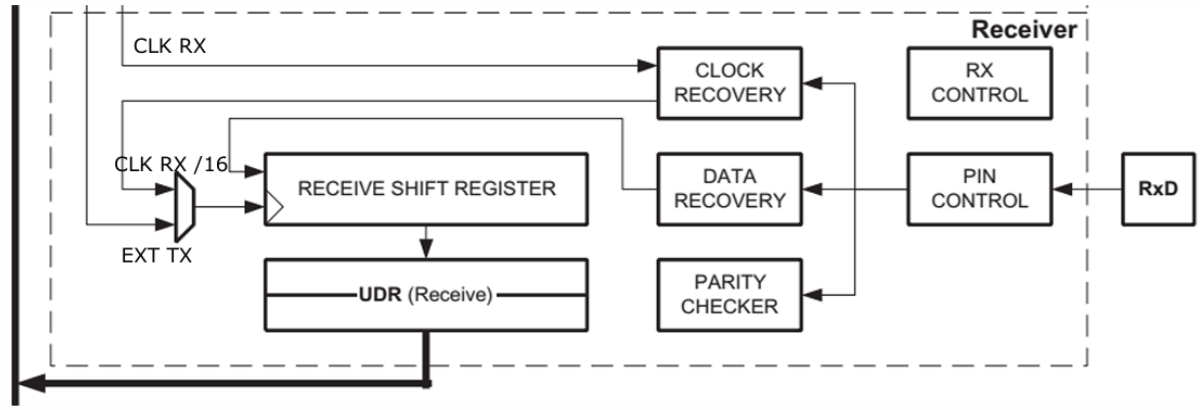
donde Fosc es la frecuencia del microprocesador (16 MHz). La tasa de baudios anterior se duplica si se setea U2X0 en 1.

El transmisor es un conversor de datos paralelo a serie que además puede generar un bit de paridad para verificar la correcta transmisión de los datos. Su diagrama es el siguiente:



Los datos para transmitir se deben cargar en el registro UDR un byte a la vez y tanto el envío como la recepción están sincronizados con el reloj configurado anteriormente.

El receptor es un conversor de datos serie a paralelo que, además, contiene un verificador de paridad y su diagrama es el siguiente:



Los registros utilizados para configurar al transmisor y al receptor son los siguientes:

UCSR0A: UART CONTROL AND STATUS REGISTER A

RXC0	TXC0	UDRE0	FE0	DOR0	UPE0	U2X0	MPCM0

Los únicos bits que se utilizarán de UCSR0A son el UDRE0 (USART Data Register Empty 0) que se setea cuando el registro UDR del trasmisor está vacío y el bit RXC0 que se setea cuando hay datos en el registro UDR del receptor que todavía no han sido leídos.

UCSR0B: UART CONTROL AND STATUS REGISTER B (UART0)

RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80

Los bits que se utilizarán de UCSR0B son RXEN0 y TXEN0 los cuales habilitan al receptor y al transmisor respectivamente y el UCSZ02 cuyo funcionamiento se explicará más adelante. Los bits RXCIE0 y TXCIE0 se pueden utilizar para habilitar interrupciones cuando finalicen una recepción o una transmisión respectivamente.

UCSR0C: UART CONTROL AND STATUS REGISTER C

UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0

Los bits que se utilizarán de UCSR0C son UCSZ01 y UCSZ00 los cuales se usan en conjunto con UCSZ02 para determinar el tamaño de los datos que se utilizarán en la transferencia de acuerdo con la siguiente tabla:

UCSZ02	UCSZ01	UCSZ00	Char. size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	1	1	9-bit

2.3 Funcionamiento del software

La resolución de este problema se llevo a cabo con el uso de 5 archivos de tipo .c y sus cabeceras, la separación del código en archivos facilita la reutilización de las funciones en futuras aplicaciones.

- main.c: este archivo contiene la función principal que será ejecutada al momento de correr el programa.
- Timer_ms.c: Contiene funciones relacionadas con la temporización del programa. Este archivo fue reutilizado del trabajo anterior, solo se modificaron los nombres del archivo y de las funciones, además de agregar dos funciones que serán explicadas más adelante.
- DHT.c: Este archivo contiene la funcion encargada de encuestar al sensor de temperatura y humedad y de devolver los datos recibidos por el mismo.
- RTC.c: Este archivo tiene las funciones utilizadas para comunicar el programa con el modulo de fecha y hora RTC
- i2c.c: Este archivo contiene las funciones explicadas en la teoría de la materia para el protocolo de comunicación i2c.

Además de estos archivos se utilizo el archivo serialPort.c dado por la catedra para el uso del módulo UART.

main.c

En este archivo se encuentra la funcion principal del programa asi como la interrupción de lectura del teclado. En el programa principal se llama a las funciones de inicialización del timer, del modulo RTC y del módulo UART.

```
int main(void)
{
        Configuración del Timer
        Inicialización del UART
        Se activa el Transmisor del Puerto Serie
        Se activa el Receptor del Puerto Serie
        Se activa la interrupción de recepcion.
        Se activan las interrupciones globales
        Se inicializa el módulo RTC
        Se establece una fecha y hora inicial al modulo RTC
        while (1) // loop infinito
        {
                Si se produjo una interrupcion y el sistema esta activo{
                        Se piden los datos al sensor DHT
                        if (El sensor devolvio datos){
                                Se envian los datos de temperatura y humedad
                        }else{ Se envia un mensaje informando el error}
                        Se pide la fecha y hora al módulo RTC
                        Se envían los datos de fecha y hora
                }
        }

}
```

Activación o desactivación del sistema:

La acción de activar o desactivar el funcionamiento del sistema se llevo a cabo mediante una interrupción del modulo USART, al recibir un dato por este medio se produce la interrupción y a través de esta misma, si fue presionada la tecla ‘s’ se activa o se desactiva un flag encargado de indicar si se debe o no enviar los datos de los sensores.

```
// Rutina de Servicio de Interrupción de Byte Recibido
ISR(USART_RX_vect){
        char RX_Buffer = UDR0; //la lectura del UDR borra flag RXC
        // Si se presiona 's' se alterna la transmision de datos
        if((RX_Buffer == 's')||(RX_Buffer == 'S')){
                active ^= 1;
        }
        RX_Buffer=0;
}
```

Timer_ms.c

Este archivo fue tomado del trabajo numero 2 y modificado para ser usado en este programa. Se cambio el nombre de algunas funciones pero su funcionamiento no recibió cambios:

- Flag de interrupción: el nombre del flag de interrupción cambio de llamarse FLAG_MEF a FLAG_F, asi como el contador asociado a este flag cambio de cuentaMEF a cuentaF. Por ultimo se modifico el nombre del comparador limite para este flag de topMEF a flag_top.

Funciones agregadas para este programa:

- Timer_Stop(): Se encarga de deshabilitar la interrupción del timer.
- Timer_Start(): Se encarga de habilitar la interrupción del timer.

Estas funciones fueron creadas para la comunicación con uno de los sensores que será explicado más adelante. Y su funcion es la de evitar que se realicen interrupciones del timer en determinados momentos.

Las funciones utilizadas para este programa fueron:

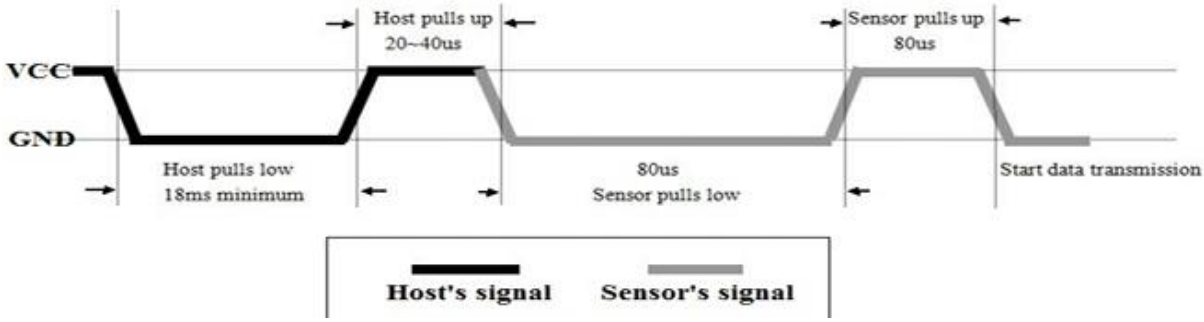
- TimerInit_ms(uint32_t top): esta funcion inicializa la interrupción del timer para que suceda cada 1ms y configura el tiempo de actualización del flag cada 2 segundos (se pasa un valor de 2000 como parámetro).
- GetFlag_ms() y SetFlag_ms(): Con estas funciones se accede al flag del archivo para realizar las funciones del programa cada 2 segundos.
- TempoInit_ms() y TempoCount_ms(): Estas funciones son las que realizan el trabajo de un temporizador y son utilizadas para reportar errores de comunicación con el módulo DHT.

DHT.c

Este archivo cuenta con una única función que se encarga de realizar la comunicación con el sensor de temperatura y humedad DHT. En su archivo de cabecera se define una estructura de datos que se puede usar para guardar los datos del sensor de una forma más organizada.

```
typedef struct {
        uint8_t T;
        uint8_t RH;
} dht_data;
```

La comunicación con el sensor se realiza siguiendo el esquema especificado en la hoja de datos de este. Esta comunicación debe



Para la recepción de los datos al igual que la etapa de comunicación inicial con el sensor se utilizo la hoja de datos para diferenciar un bit 0 de un 1. Ambos comienzan con una entrada en bajo por un tiempo indefinido y cambian a un valor alto por 28 us en el caso de un 0 y por 70 us en el caso de un 1.

El siguiente código resuelve la decodificación esperando a un pullup por parte del sensor y mediante el uso de la funcion delay se espera por 30 us, esto hace que el valor de PINC0 en ese momento sea el del bit que se debe recibir.

El armado de la respuesta se realiza guardando el bit recibido al final de un dato de 32 bits y realizando un corrimiento de 1 bit a la izquierda exceptuando en el ultimo bit recibido.

dht_data GetTyRH(){
// Mensaje del programa
Se establece el PINC0 como salida
Se pone un valor 1 en PINC0 por 2 ms
Se pone un valor 0 en PINC0 por 20 ms
Se pone un valor 1 en PINC0
Se establece PINC0 como entrada
// Respuesta del sensor
Se inicia un temporizador de 2 segundos
Se espera el pull down del sensor
Se espera el pull up del sensor
Se espera el pull down del sensor
Si en alguna de estas esperas pasaron los 2 segundos del temporizador se vuelve de la funcion con un código de error

// Comienza la transmision
for(i=0;i<32;i++){
Se espera el pullup del sensor
Se espera por 30 us
Se añade el bit actual del pin en la respuesta
Se realiza un desplazamiento a la izquierda si no es el ultimo bit
Si el bit fue un 1 se espera el pulldown

}
Se separa los datos de la respuesta y se devuelven los mismos
}

i2c.c

Este archivo contiene una recopilación de las funciones explicadas en la teoría de la materia para el uso de la interfaz i2c. Las funciones que contiene el mismo son:

- I2C_Init(): Inicializa la interfaz de comunicación, estableciendo los bits del preescalador en 0 y configurando la frecuencia del SCL.
- I2C_Start(): Esta funcion inicia una comunicación por el la interfaz i2c y espera la confirmación de una conexión exitosa.
- I2C_Write(unsigned char data): El objetivo de esta función es el de escribir un dato a través de esta interfaz. Para ello se carga el mismo en el registro TWDR y se inicia una transferencia de datos. Luego se espera a la finalización de esta.
- I2C_Read(unsigned char isLast): Esta función realiza la lectura de un byte mediante este sistema y recibe un parámetro que debe ser un 1 en caso de que el siguiente sea el ultimo byte a leer o un 0 en caso contrario. Luego de iniciar la comunicación y enviar el bit ACK en caso de que no sea el ultimo byte a leer, se espera a la finalización de la lectura y se devuelve el dato leído.
- I2C_Stop(): Esta funcion finaliza la comunicación en la interfaz I2C.

RTC.c

En este archivo se realiza las funciones necesarias para la comunicación con el modulo de fecha y hora RTC. En su archivo cabecera se encuentra la definición de una estructura capaz de almacenar todos los datos que devuelve el sensor.

```
typedef struct {
    uint8_t sec;
    uint8_t min;
    uint8_t hour;
    uint8_t weekDay;
    uint8_t date;
    uint8_t month;
    uint8_t year;
}rtc_data;
```

El archivo cuenta con 3 funciones:

RTC_Init():

Esta funcion hace uso de la funcion i2C_Init() para inicializar el protocolo de comunicación i2c.

RTC_SetDateTime(rtc_data *data):

Esta funcion es la encargada de establecer un valor de fecha y hora recibida por los parámetros en el módulo RTC, para ello primero establece una conexión con el mismo enviando el código para realizar una escritura y luego la dirección de inicio de escritura de los datos, que en este caso es el de los segundos que están ubicados en la posición 0x00. Dada la estructura del modulo los datos deben guardarse en formato BCD y para ello al estar todos estos guardados en variables de 8 bits se hace la siguiente conversión (siendo data la estructura recibida por la función):

((data->sec / 10) << 4) + (data->sec % 10)

A continuación se presenta un pseudocodigo de la función:

```
void RTC_SetDateTime(rtc_data *data){
    Desactivar la interrupción del timer antes de escribir la fecha
    Iniciar la comunicación I2C
    Conectarse al DS3231 enviando su ID en el bus I2C en modo escritura
    Solicitar la dirección de memoria de los segundos en 00H
    Escribir los datos con una conversión de binario a BCD
    Detener la comunicación I2C después de configurar la fecha
    Habilitar la interrupcion del timer  después de configurar la fecha
}
```

Se debe desactivar la interrupción del timer ya que al momento de realizar la conexión con el modulo si esta era interrumpida por el timer podía haber problemas en la comunicación.

RTC_GetDateTime(rtc_data *data):

Esta función se usa para obtener los datos de fecha y hora que se encuentran almacenados en el modulo RTC. Para ello primero se realiza una conexión i2c con el modulo RTC en modo escritura para establecer el punto inicial de lectura y una vez pasado este punto se realiza otra comunicación para realizar la lectura de los datos. Luego de esto se realiza la conversión de BCD a binario siguiendo la tabla de la hoja de datos del sensor para la conversión de las decenas de la siguiente forma:

*data->sec = ((((data->sec) & 0x70) >> 4) * 10) + ((data->sec) & 0x0F);*
*data->min = ((((data->min) & 0x70) >> 4) * 10) + ((data->min) & 0x0F);*
*data->hour = ((((data->hour) & 0x30) >> 4) * 10) + ((data->hour) & 0x0F);*
*data->date = ((((data->date) & 0x30) >> 4) * 10) + ((data->date) & 0x0F);*
*data->month = ((((data->month) & 0x10) >> 4)*10)+((data->month)&0x0F);*
*data->year = ((((data->year) & 0xF0) >> 4) * 10) + ((data->year) & 0x0F);*

Esta diferencia se dio ya que algunos de estos datos usan bits de las decenas para almacenar otra información, esto puede verse en la siguiente tabla:

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99

*void RTC_GetDateTime(rtc_data *data){*
Desactivar la interrupcion del timer
Iniciar la comunicación I2C
Conectarse al DS3231 enviando su ID en el bus I2C en modo escritura
Solicitar la dirección de memoria de los segundos en 00H
Detener la comunicación I2C
Iniciar otra comunicación I2C
Conectarse al DS3231 enviando su ID en el bus I2C en modo lectura

Leer los datos del modulo RTC
Detener la comunicación I2C después de leer la fecha
Convertir los datos de BCD a binario

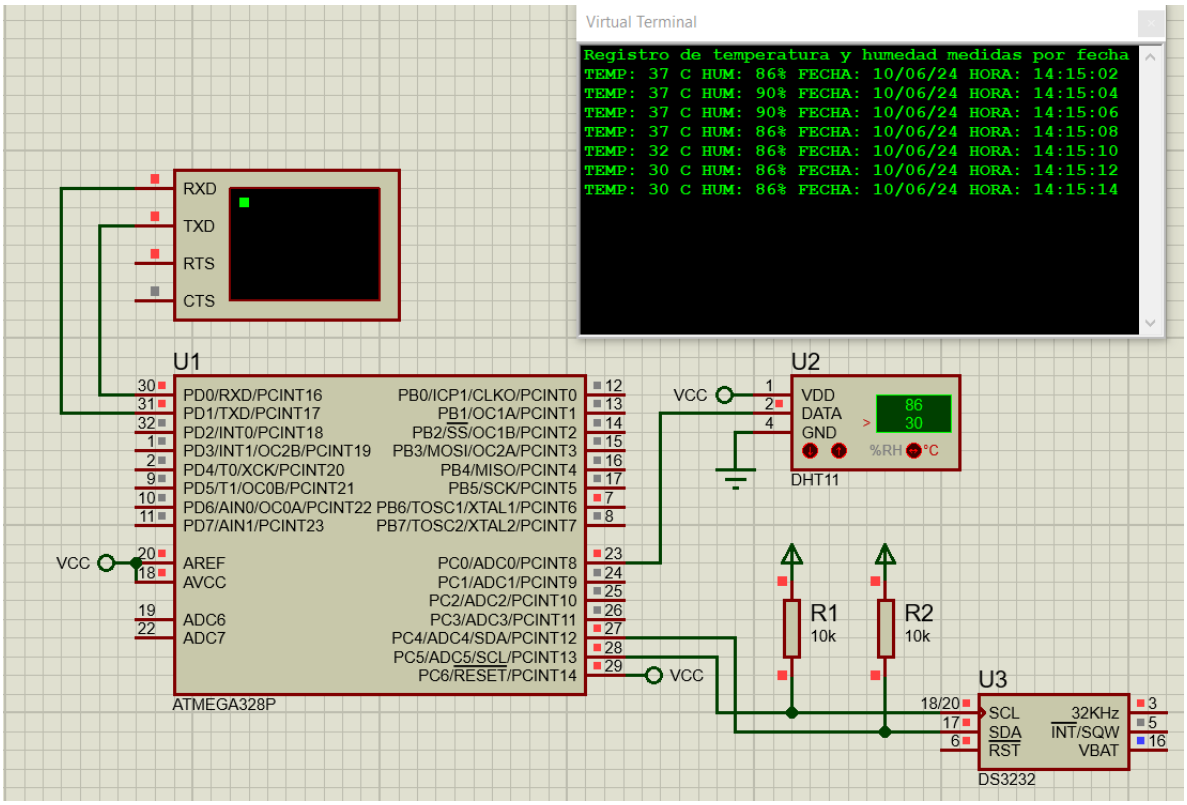
Habilitar la interrupcion del timer después de leer la fecha
}

3. VALIDACIONES

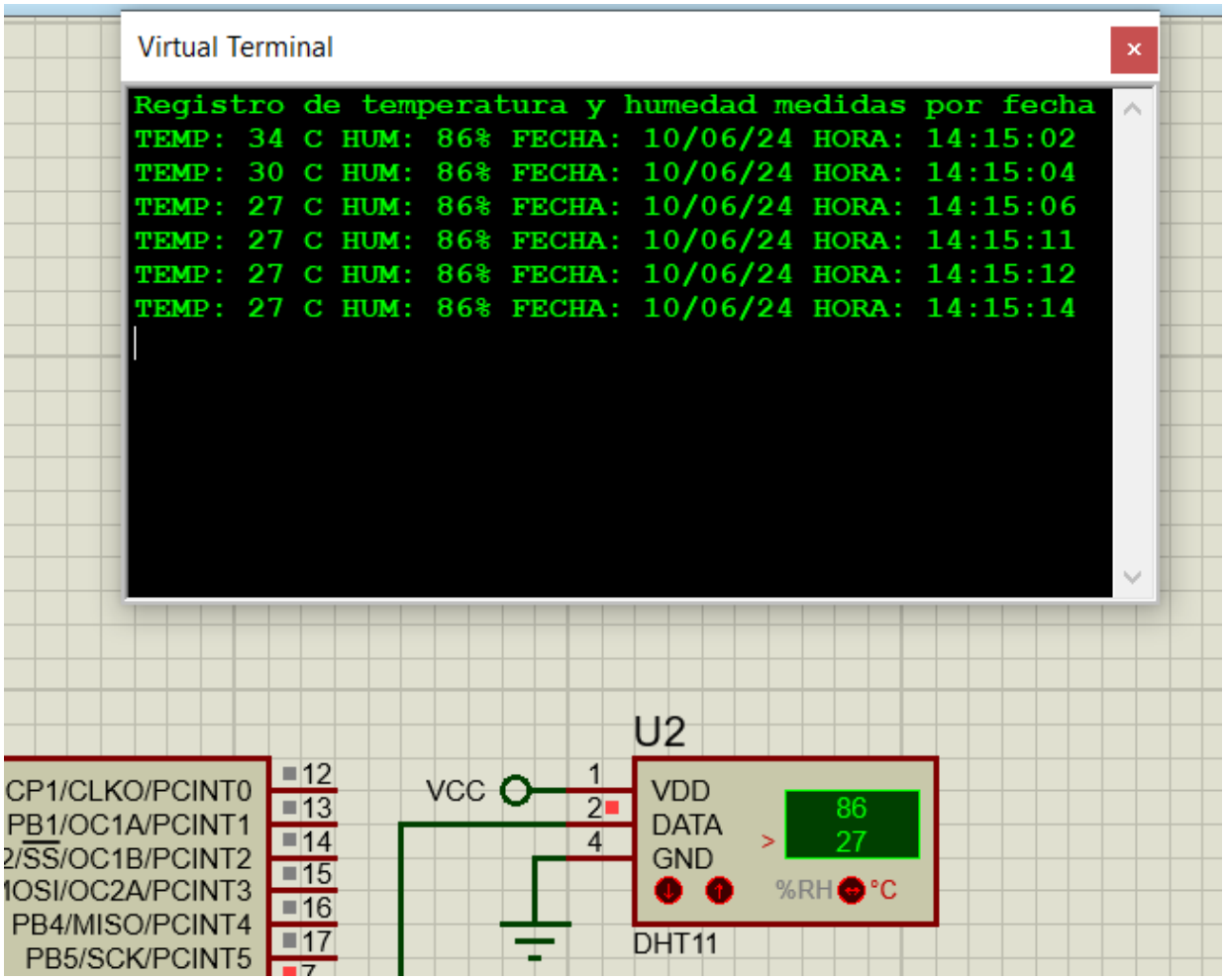
Al correr el programa, se observa desde la terminal virtual el inicio de la lista de registros de temperatura y humedad, dando inicio a las transmisiones de información cada dos segundos, tal como requería el enunciado. La lectura de datos del sensor DHT11 y la transmisión de estos se realizan de forma constante y continua mientras se ejecute el programa. Cada transmisión incluye valores precisos de temperatura y humedad, seguidos de la fecha y hora actuales obtenidas del RTC DS3231. Estos datos son enviados al puerto serie, permitiendo su monitoreo en tiempo real desde la terminal.

El sistema cuenta con una funcionalidad adicional de control a través del puerto serie al presionar la tecla 's' o 'S' en la terminal, la transmisión de datos se detiene de inmediato. Al pulsar nuevamente la tecla 's' o 'S', la secuencia de lecturas y transmisiones se reanuda sin pérdida de precisión en el tiempo, garantizando la continuidad de los registros.

En la siguiente imagen se observa la simulación del programa donde se muestra el funcionamiento del sensor DHT11 ante la variación de los valores de temperatura y humedad relativa. A lo largo de la ejecución, se toman 7 muestras, modificando los parámetros de temperatura y humedad entre cada lectura, además el módulo RTC se encarga de agregar la fecha y hora actual a cada medida obtenida del sensor, utilizando como referencia el valor inicial establecido para estos parámetros.



Luego, se verifica el funcionamiento de la interrupción de la secuencia utilizando el pulsador 's' o 'S' desde la terminal. En la siguiente imagen, se observa la simulación mostrando tres muestras registradas, se produce una interrupción tras presionar 's' y luego, se reanudan las mediciones al presionar nuevamente 's', registrando tres nuevas muestras. La imagen ilustra cómo el sistema pausa y retoma la secuencia de registro de datos, manteniendo la precisión en la medición de temperatura y humedad junto con la hora actual.



Estas características del funcionamiento se ven demostradas en las siguientes pruebas, una muestra el programa funcionando en el kit entregado por la catedra y el otro muestra una simulación realizada en el programa Proteus.

Prueba de funcionamiento en el kit:
https://drive.google.com/file/d/1kdYzIkoZzETIbXYhQh3WYCdGlBnJP8sn/view?usp=sharing

Prueba de funcionamiento en Proteus:
https://drive.google.com/file/d/1MePNvI1a489etK1y3rvmsm-csSWoVIFp/view?usp=sharing

Verificación de tiempo:

La validación de la temporización del programa se realizará mediante el contador de ciclos que nos proporciona el simulador Proteus8, considerando una frecuencia del reloj del MCU en 16 MHz, se verificará el tiempo transcurrido desde que se setea el flag del timer en 0, ya que un requisito explicito es que el MCU encueste al sensor para obtener una medida de la temperatura y la humedad relativa cada dos segundos y pasado ese tiempo se habrá encendido el flag hasta el siguiente.

Para esto, detengo el programa luego de desactivar el flag del timer (imagen 1) hasta que se envían los valores serie a la terminal y se vuelve a activar el flag (imagen 2), la diferencia en los valores de los ciclos de reloj entre ambas capturas fue de 32,160,131 ciclos. Para una frecuencia de 16 MHz, este valor equivale a 2.01 segundos, cumpliendo así con el requisito de encuestar al sensor cada dos segundos.

Como se menciona en la explicación del código se demuestra que, al momento de comunicarse con el sensor, el timer se desactiva para evitar interferencias que puedan generar errores. Esto puede ocasionar que el valor obtenido sea ligeramente superior al establecido.

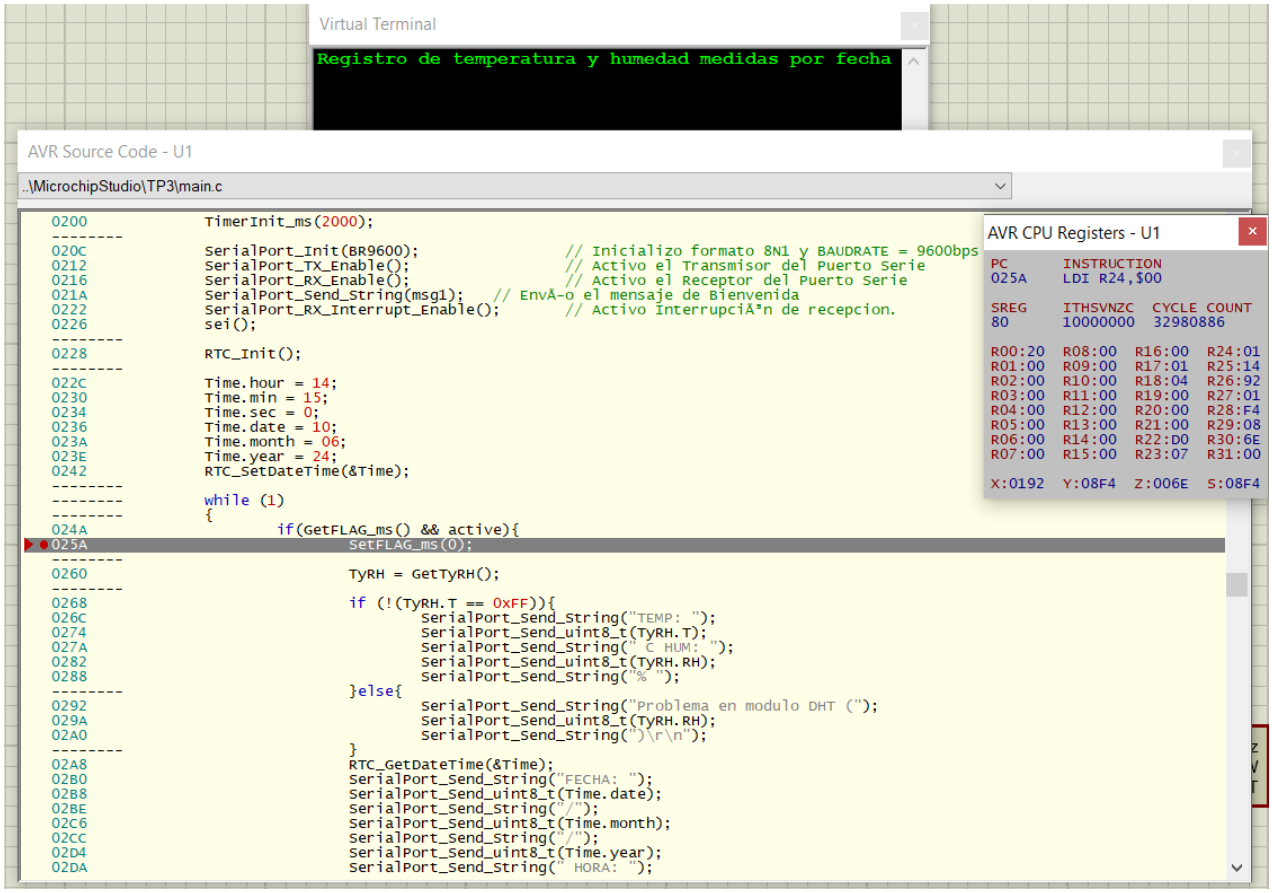


Imagen 1

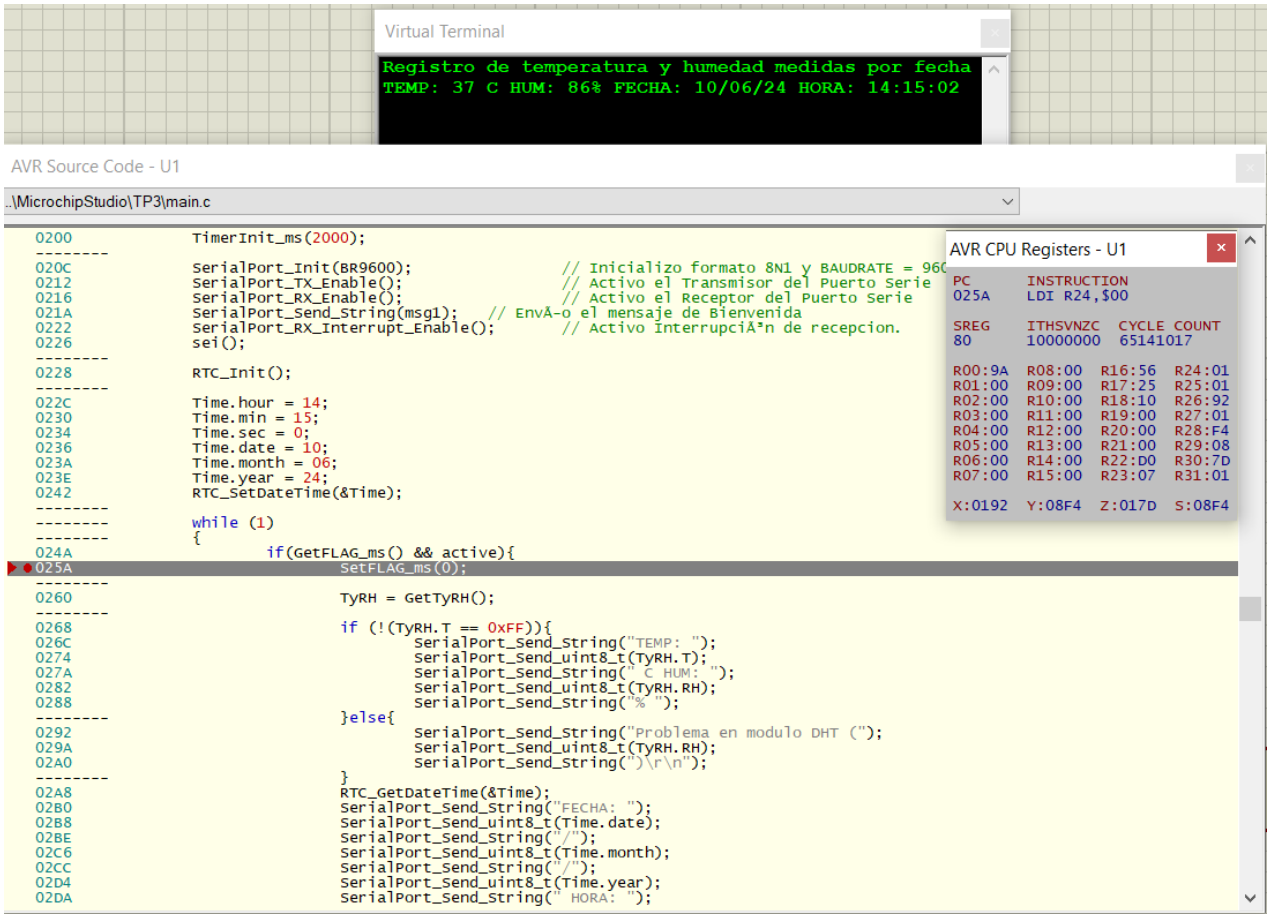

Virtual Terminal
Registro de temperatura y humedad medidas por fecha
TEMP: 37 C HUM: 86% FECHA: 10/06/24 HORA: 14:15:02
AVR Source Code - U1
..\MicrochipStudio\TP3\main.c
AVR CPU Registers - U1
PC INSTRUCTION
025A LDI R24,$00
SREG ITHSVNZC CYCLE COUNT
80 10000000 65141017
R00:9A R08:00 R16:56 R24:01
R01:00 R09:00 R17:25 R25:01
R02:00 R10:00 R18:10 R26:92
R03:00 R11:00 R19:00 R27:01
R04:00 R12:00 R20:00 R28:F4
R05:00 R13:00 R21:00 R29:08
R06:00 R14:00 R22:D0 R30:7D
R07:00 R15:00 R23:07 R31:01
X:0192 Y:08F4 Z:017D S:08F4
0200 TimerInit_ms(2000);
020C SerialPort_Init(BR9600); // Inicializo formato 8N1 y BAUDRATE = 960
0212 SerialPort_TX_Enable(); // Activo el Transmisor del Puerto Serie
0216 SerialPort_RX_Enable(); // Activo el Receptor del Puerto Serie
021A SerialPort_Send_String(msg1); // EnvÃ-o el mensaje de Bienvenida
0222 SerialPort_RX_Interrupt_Enable(); // Activo InterrupciÃ³n de recepcion.
0226 sei();
0228 RTC_Init();
022C Time.hour = 14;
0230 Time.min = 15;
0234 Time.sec = 0;
0236 Time.date = 10;
023A Time.month = 06;
023E Time.year = 24;
0242 RTC_SetDateTime(&Time);
while (1)
{
024A if(GetFLAG_ms() && active){
025A SetFLAG_ms(0);
0260 TyRH = GetTyRH();
0268 if (!(TyRH.T == 0xFF)){
026C SerialPort_Send_String("TEMP: ");
0274 SerialPort_Send_uint8_t(TyRH.T);
027A SerialPort_Send_String(" C HUM: ");
0282 SerialPort_Send_uint8_t(TyRH.RH);
0288 SerialPort_Send_String("% ");
}else{
0292 SerialPort_Send_String("Problema en modulo DHT (");
029A SerialPort_Send_uint8_t(TyRH.RH);
02A0 SerialPort_Send_String(")\r\n");
}
02A8 RTC_GetDateTime(&Time);
02B0 SerialPort_Send_String("FECHA: ");
02B8 SerialPort_Send_uint8_t(Time.date);
02BE SerialPort_Send_String("/");
02C6 SerialPort_Send_uint8_t(Time.month);
02CC SerialPort_Send_String("/");
02D4 SerialPort_Send_uint8_t(Time.year);
02DA SerialPort_Send_String(" HORA: ");

Imagen 2

4. CONCLUSIÓN

El desarrollo de este proyecto nos permitió cumplir con los requerimientos establecidos en la consigna, logrando implementar un sistema de registro de temperatura y humedad relativa del ambiente utilizando el sensor DHT11 y el módulo RTC DS3231. La programación del MCU se realizó exitosamente para leer los datos del sensor cada dos segundos, obtener la fecha y hora actual del RTC, y transmitir los datos a una PC a través de la interfaz UART. Las pruebas realizadas en el simulador Proteus y en el kit físico confirmaron que el sistema puede encuestar al sensor cada dos segundos y manejar adecuadamente las interrupciones de comunicación, con un tiempo transcurrido entre encuestas de aproximadamente 2.01 segundos, cumpliendo así con el requisito de la consigna.

Además de alcanzar los objetivos técnicos del proyecto, este trabajo nos brindó una valiosa oportunidad para practicar la integración y conexión de un programa con distintos módulos y funciones. La implementación de los drivers para el control del sensor DHT11 y del módulo RTC, así como para la comunicación serie asincrónica por UART, nos permitió profundizar en el manejo de periféricos y en la programación de microcontroladores. Aprendimos a gestionar las interrupciones y a coordinar la comunicación entre múltiples componentes.

Este proyecto también destacó la importancia de una planificación y diseño adecuados al trabajar con hardware y software. La necesidad de asegurar una correcta conexión de los componentes y una programación eficiente nos ayudó a mejorar nuestras habilidades en la resolución de problemas. Ayudandonos a ver que las pruebas en un kit real difieren de las simulaciones que se realizaron.

El siguiente repositorio en GitHUB contiene el código implementado y el entorno de pruebas en proteus

https://github.com/joacochanquia/CDyM-TP3-Grupo6.git