Circuitos Digitales y Microcontroladores - UNLP
24/6/2024

Trabajo Práctico N°3
Registrador de temperatura y humedad relativa ambiente

Grupo 4:

Graciano Gonzalez, Fausto. Legajo: 03028/2
Zanetti, Bruno. Legajo: 02975/5

FACULTAD DE INGENIERÍA

UNIVERSIDAD NACIONAL DE LA PLATA

Índice

1. Enunciado general

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

Utilizando el simulador Proteus y el microcontrolador ATMEGA328P, se deberá implementar un registrador de temperatura y humedad relativa del ambiente haciendo uso de periféricos externos, como el sensor DHT11 con el cual se obtendrán los datos de temperatura y humedad y el módulo RTC DS3231 con el cual se obtendrá la fecha y hora del momento de la medición. El sensor DHT11 estará conectado al MCU mediante el PORTC0 del mismo y el RTC se conectará haciendo uso de la interfaz I2C. Se deberá implementar los drivers necesarios para este registrador, lo que incluye el control del sensor, el control del módulo RTC y para la comunicación serie asincrónica por UART.

Una vez realizado el registrador, con este se deberá encuestar al sensor para obtener la información del mismo cada 2 segundos, esto se resolverá haciendo uso de un temporizador TIMER1. Luego mediante la consola (UART), se deberá mostrar un mensaje el cual contenga la siguiente información de acuerdo a lo medido: “TEMP: 20 °C HUM: 40% FECHA: 10/06/24 HORA:15:30:56\r\n”.
También debe ser posible mediante la consola detener y reanudar el reporte de información mediante las teclas claves “s” o “S”.
Estas dos últimas acciones se deberán realizar mediante una comunicación serie asincrónica, la cual se debe implementar mediante interrupciones de recepción y transmisión del periférico UART0, para esto se nos provee de una librería implementada por la cátedra para hacer uso del periférico.

1.3 Esquema eléctrico

1.3.1 MCU

Mediante el software Proteus, se incorporaron los componentes necesarios mediante el menú de búsqueda “Library > Pick Parts”, en este caso se selecciona el MCU y se agrega en la hoja de proyecto. En este trabajo se utilizó el “ATmega328P”. Se muestra la representación del MCU en la Figura 1.3.1 a continuación.

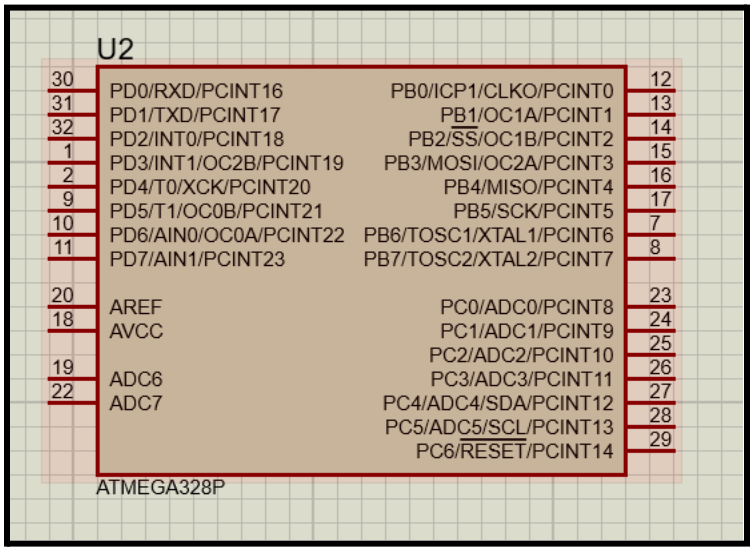

Figura 1.3.1 Microcontrolador ATmega 328P en Proteus.

1.3.2 DHT11

Se realizó el agregado del sensor DHT11, el cual para la transferencia de información, se conectó su pin DATA al PORTC0 del MCU.

El pin VDD del sensor se conectó a una terminal VCC y el pin GND se conectó a una terminal GND respectivamente.

La implementación se realizó creando las librerías “DHT11.c” y “DHT11.h” en las cuales se implementan las funciones realizables con el mismo y el conexionado para la recepción de la información.

Se muestra su conexión en la Figura 1.3.2

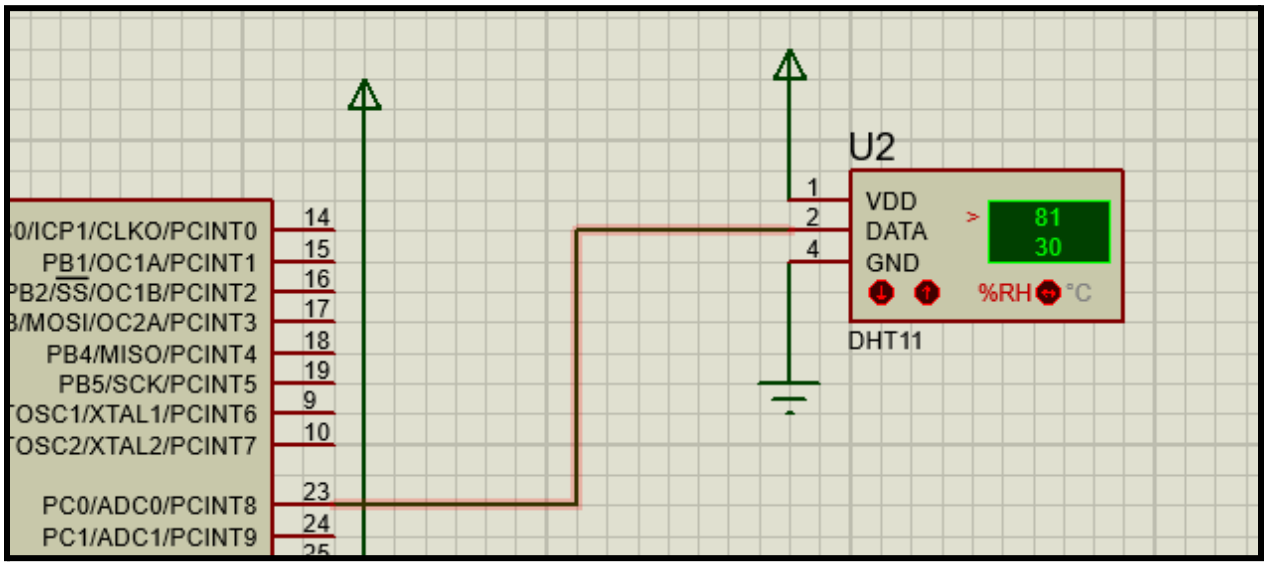

Figura 1.3.2 Conexionado de sensor DHT11.

1.3.3 RTC DS3231

Se realizó el agregado del módulo RTC DS3231, el cual se conectó su pin SCL al PORTC5 del MCU y su pin SDA al PORTC4 del MCU ambos conectados con un pull-up externo.

Para dar mayor modularización al programa, creamos la librería “RTC.c” y “RTC.h” en las cuales se implementan las funciones realizables y la conexión del mismo mediante la interfaz I2C.

Se muestra su conexión en la Figura 1.3.3.

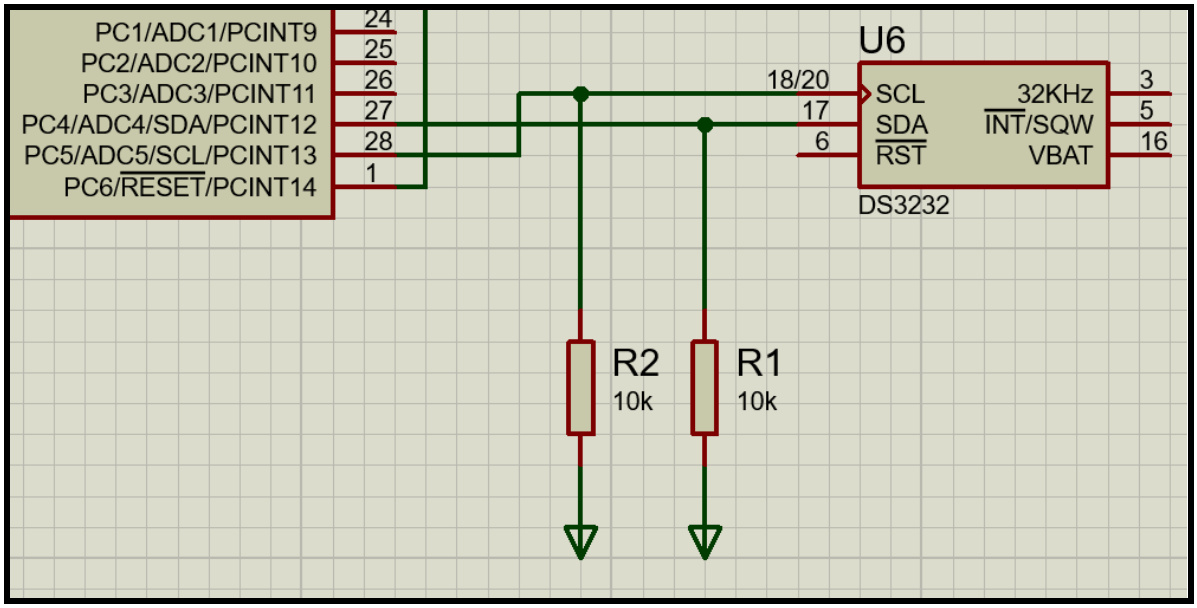

Figura 1.3.3 Conexionado del módulo RTC DS3231.

1.3.4 UART0

Se realizó el agregado del UART, mediante el dispositivo “VIRTUAL TERMINAL” del Proteus, se conectó su pin RXD con el PORTD1/TXD del MCU y su pin TXD con el PORTD0/RXD del MCU.

Para dar funcionamiento al mismo utilizamos la librería “serialPort.h” y “serialPort.c”, proporcionada por la cátedra.

Se muestra su conexión en la Figura 1.3.4.

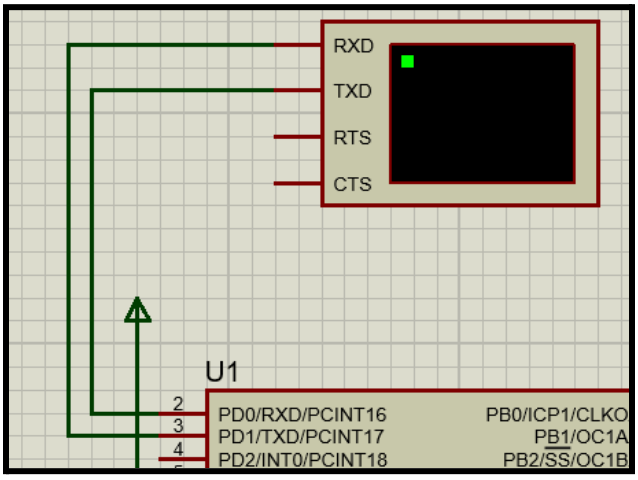

Figura 1.3.4 Conexionado del UART0.

1.4 Resolución del problema

1.4.1 Implementación del DHT11

La implementación del DHT11 fue modularizada creando su archivo .c y .h respectivamente.

Para la obtención de los datos mediante el sensor DHT11 se implementó una función llamada “leerDHT()” la cual como su nombre lo indica, realiza la lectura de los datos pasados por el DHT11 al MCU. Esta puede devolver tres tipos de resultados, la correcta transmisión de información, que hubo un error en la transmisión, o que no se encuentre conectado ningún DHT11, este último se detectara mediante el uso de un timeout, en el que caso que pase determinado tiempo sin detectar al sensor, se mostrará un mensaje en particular que dirá "Se requiere conectar un DHT11”.

Para crear la función leerDHT() se utilizó una datasheet (link en la bibliografía) con la cual pudimos obtener e implementar la manera en que se comunica el sensor con el MCU mediante el puerto PORTC0 para enviar los datos con pulsos, es decir, envía los datos de la temperatura y la humedad bit a bit. El MCU recibe del DHT un total de 40 bits de información la cual se divide en 16 bits de información de humedad + 16 bits de información de temperatura + 8 bits de chequeo de suma.

La comunicación entre el MCU y el DHT11 se implementó mediante el siguiente procedimiento:

1. **Inicialización del Sensor**:
    - Se configura el pin PC0 como salida y se pone en alto.
    - Se pone el pin PC0 en bajo durante 20 ms para enviar una señal de inicio.
    - Se vuelve a poner el pin en alto y se configura como entrada para escuchar la respuesta del sensor.
2. **Respuesta del Sensor**:
    - El sensor envía una respuesta de señal baja seguida de una señal alta, indicando que está listo para enviar datos.
    - El microcontrolador espera estas señales y verifica que se reciban dentro de un tiempo límite para comprobar que no hay errores ni bucles.
    - Cada bit de datos que el DHT11 envía comienza con una señal baja seguida de una señal alta. La duración de la señal alta determina si el bit es un 0 o un 1.
    - Los 40 bits son almacenados en una variable llamada data[5] con la cual para almacenar la información sin errores se utilizó dos bucles FOR anidados.
3. **Verificación de Checksum**:
    - Se verifica que la suma de los primeros 4 bytes sea igual al quinto byte (checksum) para validar la integridad de los datos.

- En caso que el Checksum sea el correcto, se actualizan los valores de humedad y temperatura, que son variables del DHT.c.

4. **Información Obtenida**
   - Para poder sacar los datos obtenidos por la comunicación entre el sensor con el MCU se implementaron unos Getters llamados getHumedad() y getTemperatura().

Aqui un Pseudocodigo de la funciones implementadas:

```
  int LeerDHT() {
// Recepción de datos data = [0, 0, 0, 0, 0];
se inicializa timeout en cero;

// Iniciar comunicación con el sensor
Configurar pin como salida
Poner pin en alto
Esperar 1 ms
Poner pin en bajo
Esperar 20 ms
Poner pin en alto
Configurar pin como entrada

// Esperar hasta que el pin se vuelva bajo
se reinicia timeout a 0
MIENTRAS (pin está en alto) Y (no supere tiempo TIMEOUT) {
  Esperar 1 µs
  Incrementar timeout
}
SI (supero tiempo límite) {
 RETORNAR error de timeout;
}

// Esperar hasta que el pin se vuelva alto
 se reinicia timeout a 0
MIENTRAS (pin está en bajo) Y (no supere tiempo TIMEOUT) {
  Esperar 1 µs
 Incrementar timeout
}
SI (supero tiempo límite) {
 RETORNAR error de timeout;
}
// Esperar hasta que el pin se vuelva bajo
 se reinicia timeout a 0
MIENTRAS (pin está en alto) Y (no supere tiempo TIMEOUT) {
Esperar 1 µs
 Incrementar timeout
```

```
 }
SI (supero tiempo límite) {
 RETORNAR error de timeout;
}


// Leer los 5 bytes de datos
 FOR i DESDE 0 HASTA 4 {
  FOR j DESDE 7 HASTA 0 {
// Esperar hasta que el pin se vuelva alto
  MIENTRAS (pin está en bajo) {
   Esperar 1 µs
}
  Esperar 30 µs
  SI pin está en alto {
   se guarda en vector data informacion traida según bit que corresponda
                         //Si la duración es mayor a 30us, el bit es un 1.
}
// Esperar hasta que el pin se vuelva bajo
  MIENTRAS (pin está en alto) {
   Esperar 1 µs
}
 } } //Fin de los dos FOR anidados
// Verificar checksum
SI (se verifica correctamente el CheckSum) {
// Decodificar datos
se asigna a humedad posición cero de data;
se asigna a temperatura posición dos de data;
RETORNAR lectura exitosa
SINO RETORNAR lectura incorrecta;
 }
} //fin leerDHT
```

1.4.2 Configuración del TIMER1

Para la implementación del timer se utilizó el TIMER1, se configuró para el uso del modo CTC () asignando al registro TCCR1A = 0, usando un Preescaler de 64 asignando al registro TCCR1B |= (1 << WGM12) | (1 << CS11) | (1 << CS10), el cual fue elegido empíricamente, y suponiendo que la frecuencia del MCU es de 16Mhz.

Realizamos la siguiente ecuación para calcular los ciclos necesarios para generar una interrupción cada 1ms.

Ciclos necesarios para generar interrupciones = $(1ms \ * \ 16MHz)/ \ Preescaler$

El resultado de la ecuación es de C = 250, para que la interrupción se de en estos ciclos, al registro OCR1A le ingresamos C-1(249).

Por último para habilitar las interrupciones del timer, al registro se le asigna TIMSK1 |= (1<<OCIE1A).

1.4.3 ISR del TIMER1

Para la interrupción del TIMER1 se creo una funcion la cual realiza una unica operacion, que es la de contar 2 segundos, esto se realizará mediante un contador llamado count, el cual actualizará una variable flag llamada “Flag_tiempo” en 1 cuando llegue a el valor indicado, comunicando así al programa que encueste el sensor para obtener la información de la temperatura y la humedad, y al RTC para la fecha y la hora. Para luego plasmar todo en el periférico UART. Para verificar el valor de la flag, se utiliza una función llamada “chequeoFlag()”, que la misma retorna 1 en caso de que el flag está en alto y 0 caso contrario.

Aquí un Pseudocódigo del ISR:

```
ISR(TIMER1_COMPA_vect){
  incremento count en 1;

  si count = 2000(2 segundos en milisegundos){
      cambia el Flag_tiempo a 1;
      se reinicia el contador count a 0;
  }
}
```

1.4.4 Implementación del RTC DS3231

La implementación del periférico RTC DS3231 fue modularizada creando el “RTC.c” y “RTC.h”.

Para la obtención de los datos de fecha y hora se implementaron dos funciones llamadas “RTC_GetDate()” y “RTC_GetTime()” respectivamente.

Además de estas dos funciones principales, se utilizaron dentro de estas otras funciones creadas las cuales son:

- I2C_Start(): Inicia una condición de inicio en el bus I2C. Configura los bits necesarios en el registro de control del TWI y espera hasta que el hardware I2C indique que la condición de inicio ha sido generada correctamente.
- I2C_Write(): Envía un byte de datos a través del bus I2C. Coloca el dato en el TWDR, inicia la transmisión (TWEN=1 y TWINT=1) y espera hasta que el hardware I2C indique que la transmisión se ha completado correctamente
- I2C_Stop(): Genera una condición de parada en el bus I2C. Configura los bits necesarios en el registro de control del TWI para iniciar la condición de parada.
- bcdToDec(): Toma un número en formato BCD y lo convierte a su equivalente en formato decimal. Se hace extrayendo el dígito de las decenas y multiplicándose por 10, luego agregando el dígito de las unidades.
- I2C_readNack() y I2C_readAck(): La función I2C_ReadAck lee un byte de datos desde el bus I2C y envía un reconocimiento (ACK) para indicar que el byte ha sido recibido correctamente. Configura los bits necesarios en el registro de control del TWI para iniciar la operación de lectura, espera hasta que el

hardware I2C indique que la recepción ha finalizado, y luego devuelve el byte recibido.
La función I2C_ReadNack lee un byte de datos desde el bus I2C y envía una negativa de reconocimiento (NACK), indicando que no se desean más datos. Configura los bits necesarios en el registro de control del TWI para iniciar la operación de lectura, espera hasta que el hardware I2C indique que la recepción ha finalizado, y luego devuelve el byte recibido.

- I2C_Init(): Inicialización de la frecuencia de reloj(SCL) y Habilitar el periférico con el bit TWEN.

Aqui un pseudocódigo del I2C_Init():

```
static void I2C_Init(void) {
   Configura el preescaler de TWI a 1(0x00).
   Establece el bit rate para determinar la velocidad del I2C(255).
   habilita el módulo TWI(0x04).

 }
```

Aqui un pseudocódigo de la función RTC_GetTime:

```
void RTC_GetTime(char* timeStr) {

      Enviar START y esperar confirmación;
      Enviar la dirección del RTC con el bit de escritura(0 en este caso);
      Envíar la dirección del registro de segundos (0x00) en el DS3232.
      Enviar STOP;

      Enviar START y esperar confirmación;
      Envíar la dirección del RTC con el bit de lectura (1 en este caso).
      Leer tiempo (segundos, minutos, horas)
      Enviar STOP;
      Formatear tiempo concatenando horas, minutos y segundos;
}


Aqui un pseudocódigo de la función RTC_GetData:
void RTC_GetDate(char* dateStr) {
      Enviar START y esperar confirmación;
      Enviar la dirección del RTC con el bit de escritura(0 en este caso);
      Envíar la dirección del registro del día del mes(0x04) en el DS3232.
      Enviar STOP;

      Enviar START y esperar confirmación;
      Envíar la dirección del RTC con el bit de lectura (1 en este caso).
      Leer fecha (día, mes, año);
```

```
      Enviar STOP;
      Formatear fecha concatenando dia, mes y año;
}
```

1.4.5 Implementación del periferico UART

Para la implementación del periférico UART se utilizó las librerías brindadas por la cátedra “serialPort.c” y “serialPort.h”, las cuales tienen dentro distintas funciones que permiten enviarle a la consola mensajes, y otras funciones relacionadas con el control del buffer. Para poder frenar y reanudar el registrador de temperatura y humedad mediante la pulsación de la tecla “s” o “S”, se agregó a la librería de la cátedra una interrupción “ISR(USART_RX_vect)”, la cual tiene como función comprobar el dato dentro del registro UDR0, en caso que sea “s” o “S”, se va a realizar un toggle a un flag creado en el main del programa llamado “SuspendFlag”. Este Flag es encuestado dentro del main el cual si se encuentra en alto, indica que el reporte de datos está frenado, en caso contrario debe seguir su normal funcionamiento de encuestar al sensor cada 2 seg y de obtener la fecha y hora del RTC.
Para imprimir la información completa en el periférico UART se utilizaron 2 funciones, por un lado “SerialPort_Send_String()” con el cual le enviamos a la consola un string, y “SerialPort_Send_uint8_t()” con el que enviamos la información de la temperatura y humedad.

Aqui un pseudocódigo de la ISR:

```
ISR(USART_RX_vect) {
      asigno a variable receivedChar registro UDR0;
      si (receivedChar es igual a “s” o “S”) {
            realizó negación de la Suspendflag;
      }
}
```

Aquí un pseudocódigo del Main para imprimir la información en el periférico UART:

```
// Inicializar variables
// variables del timer1
se declara variable count inicializada en 0;
se declara variable Flag_tiempo inicializada en 0;
se declaran strings para imprimir por consola;
se declaran variables para obtener la fecha y hora del RTC;
se declara flag suspendFlag inicializada en 0;

int main(void) {
Inicializar RTC;
Inicializar Timer1;
Inicializar Puerto Serie de consola a 9600 bps;
Activar el Transmisor(TX) del Puerto Serie;
Activar el Receptor(RX) del Puerto Serie;
```

```
Activar la interrupción del receptor(RX);
Activar las interrupciones globales;
// Bucle principal
 MIENTRAS (1) {
  si(suspendFlag es cero) {
      leer datos del sensor DHT11;
      SI (lectura es correcta) {
            Enviar datos de humedad y temperatura por el puerto serie;
            Desactivar interrupciones globales;
      Obtener y enviar la fecha y hora actual por el puerto serie;
} SINO {
  Se envia al UART un mensaje de error. }

Activar interrupciones globales;
Inicializar contadores;
// Esperar hasta que el flag sea 1
MIENTRAS (chequeoFlag no sea 1)) {
// Esperar 2 segundos antes de la siguiente lectura
    }
   }
 }
 RETORNAR 0;
}
```

2. Validación

2.1 Validación tiempo del timer

Se verificó que el tiempo de espera para un nueva consulta al sensor sea de 2 segundos, se puede observar un diferencia de 0.0003/0.0002seg, para comprobar que este error sea aceptable, hicimos un cálculo del mismo para poder decidir si es un error los suficientemente pequeño como para no generar problemas.

$$\frac{Tiempo\ teórico(2\ seg) - Tiempo\ resultante(1.9997\ seg)}{Tiempo\ teorico(2\ seg)} * 100\% = 0.015\% < 1\%$$

como el resultado es menor que 1% lo consideramos aceptable.

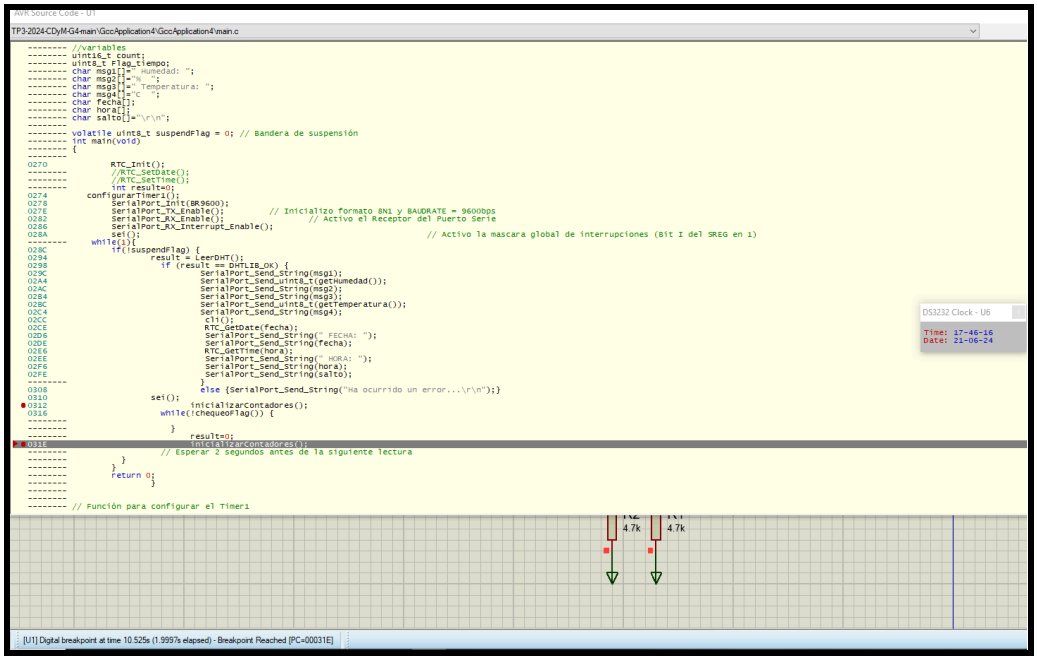

Figura 2.1.1 Validación de consulta cada dos segundos .

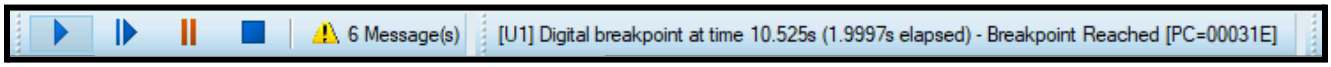

Figura 2.1.2 Tiempo del simulador Proteus..

2.2 Validación de reporte

En la siguiente figura 2.2.1 se verifica el funcionamiento del registrador, en donde se comienza detectando una humedad del 86% y luego al cambiar manualmente el DHT11, se detecta una humedad del 66%. Además se puede observar como la consulta al sensor se realiza cada dos segundos.

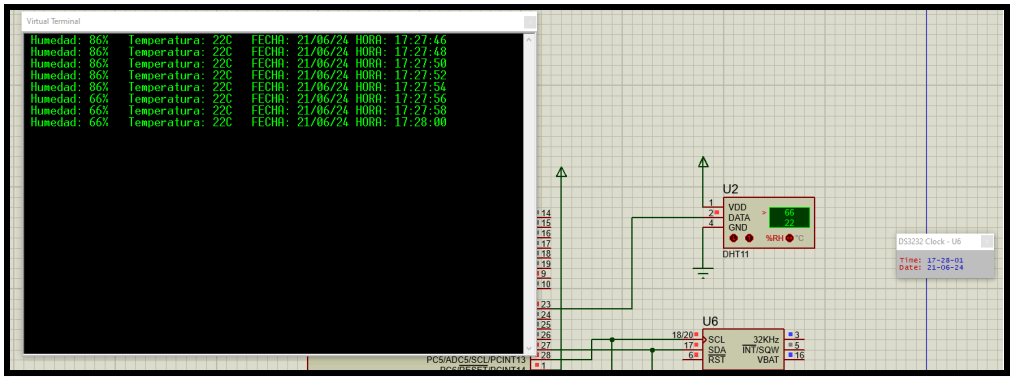

Figura 1.3.4 Validación de reporte de temperatura y humedad ambiental.

2.3 Validación de error

En la siguiente figura 2.3.1 se verifica el mensaje en caso de que ocurra un error como por ejemplo que no esté conectado el DHT11 como se muestra en la imagen.

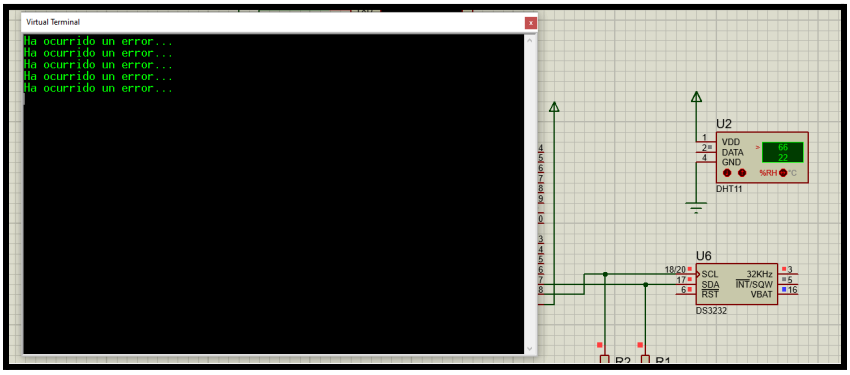

Figura 2.3.1 Validación de reporte de temperatura y humedad ambiental.

2.4 Simulación completa con video

Adjuntamos en el siguiente link un video mostrando la detección de la temperatura y humedad ambiental, y como esta se ve variada al aplicarle un estímulo.

Video Muestra del programa

3. Conclusión

3.1 Desarrollo de la conclusión

El funcionamiento del programa cumple todo lo solicitado en los incisos anteriores, el uso de la modularización nos fue muy útil para trabajar con varios periféricos externos ya que nos brinda una fácil comprensión del código y además nos ayuda a encontrar errores de una manera más simple.

El sistema demostró su veracidad y precisión mediante la simulación en Proteus, validando que las medidas de temperatura y humedad se actualizan cada dos segundos como se esperaba. También se verificó el correcto manejo de errores, como la falta de conexión del sensor DHT11, asegurando una respuesta adecuada del sistema en situaciones anómalas.

La validación del sistema fue importante para garantizar el correcto funcionamiento del programa, especialmente en la precisión del temporizador. Con el uso de una interrupción de 1ms se pudo permitir la ejecución de tareas periódicas en distintos tiempos mayores como cada 2000 ms. Esto permite agregar nuevas funciones, si en un futuro así se quisiera, con tiempos diferentes, en cambio si hubiéramos optado por interrupciones más largas, esa flexibilidad se vería reducida.

4. Bibliografía

4.1 Bibliografía

❖ Digital relative humidity & temperature sensor DHT11.Datasheet.
https://www.electronicoscaldas.com/datasheet/DHT11_Aosong.pdf

❖ Atmel Corporation (2015). ATmega328P Datasheet.
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

❖DS3231(2015).Maxim Integrated Products, Inc. Datasheet.
Datasheet RTC DS3231

5. Anexo

5.1 Anexo

Adjuntamos en el siguiente link un PDF con el código Main C del programa.
Programa Principal

Adjuntamos en el siguiente link el repositorio GITHUB con todos los archivos del programa.
https://github.com/FaustoGraciano/TP3-2024-CDyM-G4/tree/main