



Universidad Nacional de La Plata

Trabajo Práctico 3 - Grupo 20

Registrador de temperatura y humedad relativa ambiente

Circuitos Digitales y Microcontroladores

Tarifa, Carla.

Carballo Ormaechea, Lucas.

Índice

1. Problema.....	3
1.1 Interpretación.....	3
2. Resolución.....	4
2.1 Sensor DHT11.....	4
2.1.1 Funcionamiento.....	4
2.1.2 Conexión.....	6
2.1.3 Pseudocódigo DHT11.....	6
2.2 SERIAL PORT.....	7
2.2.1 Utilización.....	7
2.3 Interfaz I2C.....	8
2.3.1 Introducción.....	8
2.3.2 Configuración.....	9
2.4 RTC DS3231.....	11
2.4.1 Introducción.....	11
2.4.2 Pseudocódigo.....	12
2.4.3 Pseudocódigo configuración de RTC con la interfaz I2C.....	13
2.5 Main.....	13
2.5.1Pseudocódigo.....	13
2.6 Background Foreground.....	14
2.6.1 Pseudocódigo de tareas en foreground.....	14
2.6.2 Pseudocódigo de tarea en background.....	15
2.7 Modularización.....	15
2.8 Validación.....	16
2.9 CÓDIGO GitHub.....	19

1. Problema

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 2 seg.
- 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.

1.1 Interpretación

En este proyecto se implementará un registrador de temperatura y humedad , utilizando un sensor DHT11 (de este periférico se leerán los datos para luego ser impresos por la terminal) y el módulo RTC DS3232 (este será encargado de llevar el seguimiento preciso de la fecha y la hora) . Además , se utilizara UART para la comunicación serie asincrónica con dicha terminal.

Requerimientos

- a) Programar el MCU para que lea los datos del sensor DHT11 cada 2 segundos, es decir que, se tendrá que configurar el timer para generar eventos cada 2 segundos y leer el DHT11 en esos eventos.

- b) Leer la fecha y hora del RTC DS323 después de cada lectura del sensor y formar un registro completo.
- c) Formatear las lecturas de temperatura, humedad, fecha y hora en una cadena de texto y enviarla a la PC via UART.
- d) Utilizar un flag para iniciar y detener la transmisión de datos según se reciba el comando 's' o 'S' desde la PC.
- e) Configurar UART0 para que utilice interrupciones tanto en la recepción como en la transmisión de datos.

En pocas palabras se debe configurar el sensor DHT11 (encargado de leer la temperatura y humedad), el RTC DS3231 (modulo para obtener fecha y hora) y configurar UART0 para la transmisión y recepción de datos.

La lógica del programa va a ser leer los datos del sensor DHT11, después leer la fecha y hora, enviar dichos datos a la PC, controlando el envío de datos desde la PC con la teclas S/s, luego a los dos segundos volver a realizar el mismo proceso, siempre controlando el comportamiento de la tecla S/s.

2. Resolución

2.1 Sensor DHT11

2.1.1 Funcionamiento

Para programarlo se tendrá en cuenta que, el MCU envía una Señal de Inicio al DHT11 ,este cambia del Modo de Bajo Consumo de Energía al Modo de Ejecución, y espera a que termine la señal de inicio del MCU. Cuando comienza la comunicación entre el MCU y el DHT11, el programa del MCU pondrá el nivel de voltaje de la línea única de datos a un nivel BAJO. Esta señal de Inicio debe durar al menos 18ms para garantizar que el DHT detecte la señal del MCU. Luego el MCU dejará la línea de nuevo libre y la pondrá en alto , esperando durante 20 a 40 us la respuesta del DHT. En la figura 1 se podrá observar el proceso.

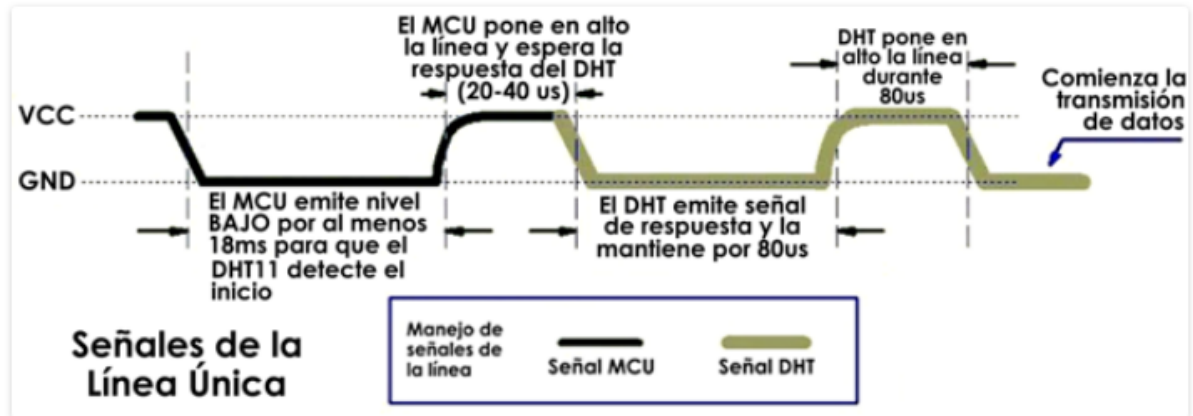


Figura 1. MCU envía señal de inicio y DHT responde.

Una vez que el DHT detecta la Señal de Inicio, enviará una señal de respuesta de nivel de voltaje BAJO durante 80us. Luego, el programa del DHT pone el nivel de voltaje de la línea única de datos de BAJO a ALTO y lo mantiene durante 80us, mientras el DHT se prepara para enviar datos.

Del lado del MCU, cuando la línea única de datos está en el nivel de voltaje BAJO significa que el DHT está enviando una señal de respuesta. Una vez que el DHT envía esa señal de respuesta, pone la línea en ALTO y la mantiene durante 80us. Así se prepara para la transmisión de datos.

Cuando el DHT está enviando datos al MCU, cada bit de datos comienza con un nivel de BAJO voltaje de 50us y la longitud de la siguiente señal de nivel de voltaje ALTO determina si el bit de datos es "0" o es "1" (consulte las figuras a continuación)

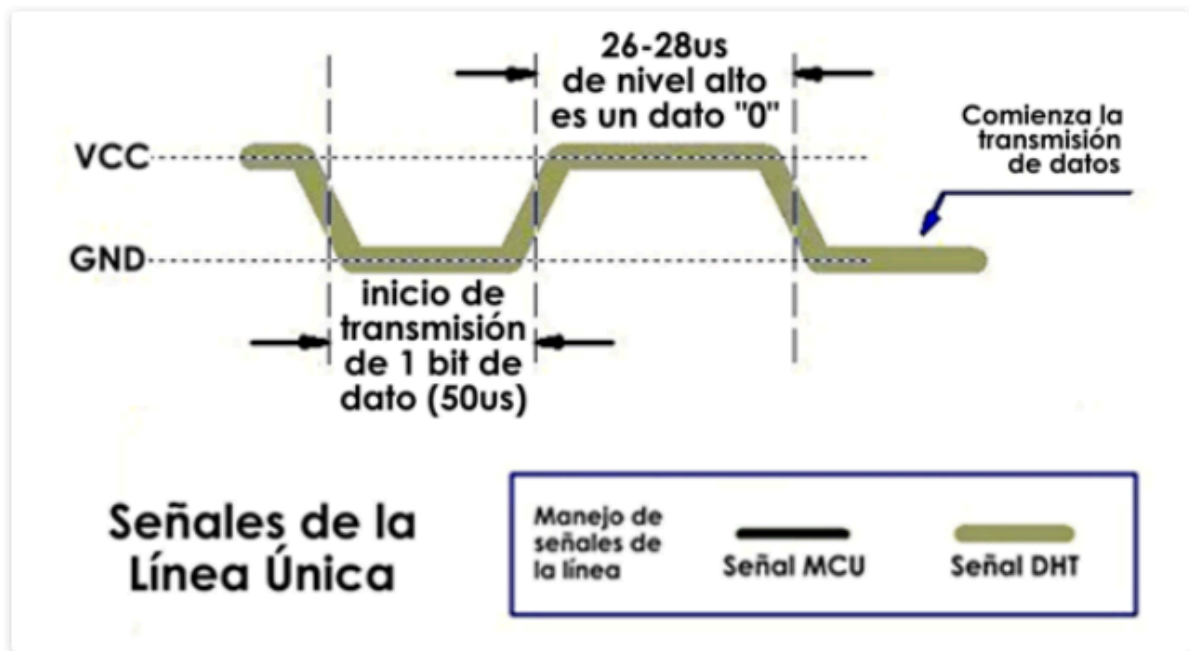


Figura 2. Transmisión del bit ' 0 '.

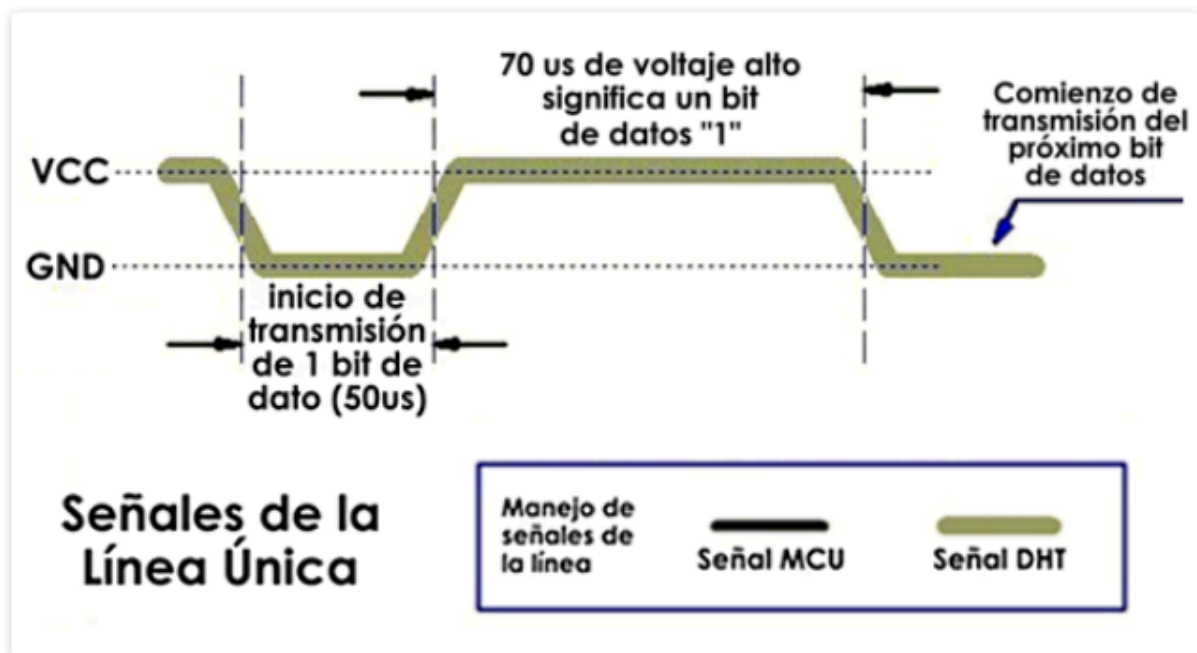


Figura 3. Transmisión del bit ' 1 '.

2.1.2 Conexión

De acuerdo a la hoja de datos , el sensor se conectará al PIN 0 del puerto C .En la siguiente figura se podrá observar como es la conexión del sensor al MCU

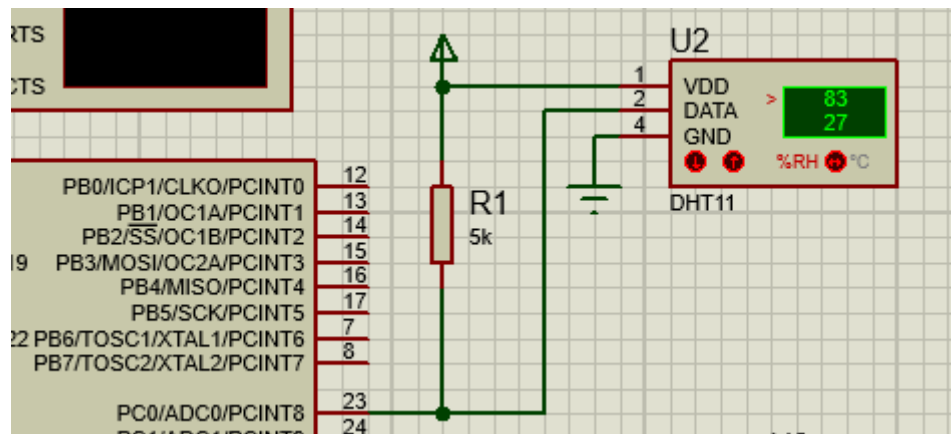


Figura 4. Conexión del DHT11

2.1.3 Pseudocódigo DHT11

Configurar PC0 como salida

Poner en bajo durante 18ms

Configurar PC0 como entrada

Esperar entre 20-40 us

El sensor pone en bajo la señal por 80us

El sensor pone el alto la señal por 80us

Mientras no llegue a 40 bits

 El sensor anuncia el siguiente bit poniendo en bajo la señal por 50 us

 El sensor pone en alto la señal por tiempo determinado

 Chequeo si el tiempo en alto es menor a 28 us

 Si es menor envío cero

 Si no envío un uno

 Guardo el bit en un arreglo

2.2 SERIAL PORT

2.2.1 Utilización

En el proyecto se utilizaron dos buffer globales , uno de ellos llamado Buffer que es el que contiene los datos a transmitir y otro llamado comando que es donde se almacena la información que llega de la terminal . Tanto el transmisor como el receptor actúan mediante interrupciones de modo que no sea bloqueante la interacción con los periféricos. Para configurar el periférico de comunicación serie se utilizó la librería "SerialPort.h" provista por la cátedra.

Dicha librería cuenta con :

1- Inicialización del UART: SerialPort_Init(uint8_t config) → Recibe como parámetro el valor en hexadecimal UBRR para configurar los baudios(de acuerdo a la

teoría vista y los cálculos realizados más adelante el UBRR debe ser 103 , en hexadecimal 67 para 9600 baudios) , luego la misma función se encarga de configurar los demás registros para utilizar tramas 8N1 , es decir 8 bits de datos, sin bit de paridad y con un bit de stop.

2- Habilitación del transmisor (`SerialPort_TX_Enable()`) : Habilita la transmisión de datos poniendo en 1 el bit TXEN0) y receptor (`SerialPort_RX_Enable()`): Habilita la recepción de datos poniendo en 1 el bit RXEN0).

3- Habilitar interrupción para la recepción (`SerialPort_RX_Interrupt_Enable()`) pone el 1 el bit RXCIE0 del registro UCSR0B) de datos , siempre y cuando haya datos nuevos en el registro UDR0 .

4- Habilitar interrupción para la transmisión (`SerialPort_TX_Interrupt_Enable()`) para que este interrumpa cada vez que esté libre para enviar un dato.

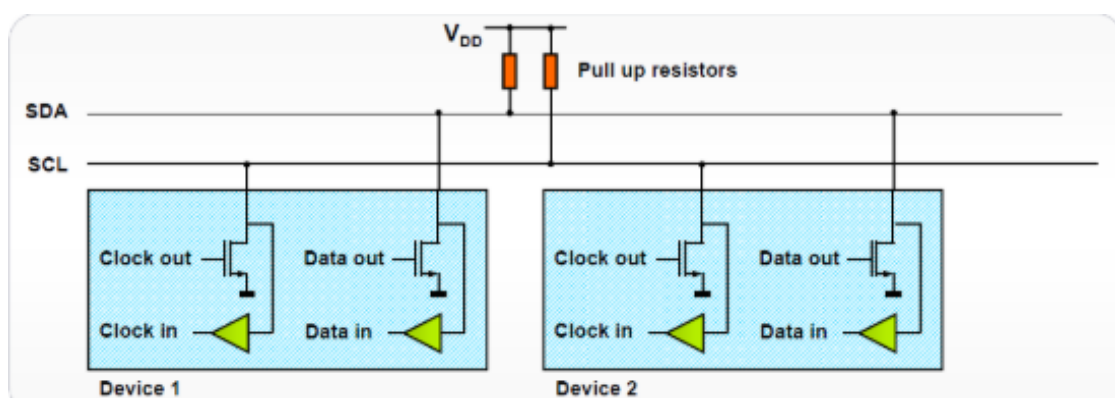
5- Deshabilitar interrupciones : `SerialPort_TX_Interrupt_Disable()` y `SerialPort_RX_Interrupt_Disable()` .

6- Recepción y transmisión de datos : `SerialPort_Send_Data(char data)` → Envía un byte. `SerialPort_Send_String(char *msg)` → Envía una cadena de caracteres. `SerialPort_Recive_Data()` → Recibe un byte.

2.3 Interfaz I2C

2.3.1 Introducción

La interfaz I2C permite conectar dispositivos de manera sincrónica bidireccional . En la siguiente figura se encuentra su esquema de interconexión.



Figura

2.3.1 Esquema de interconexión

Para determinar el valor de las resistencias Pull-up se debe tener en cuenta la corriente máxima y la tasa de transferencia deseada , generalmente se recomienda :

4.7k para modo estándar (fclk=100kHz) , 2.2k para modo fast(fclk=400kHz) y 1k para modo High Speed(fclk=3.4Mhz).

La interfaz sigue el siguiente protocolo de comunicación :

- 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 7 bits)
- 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 (master o slave) debe establecer una confirmación: ACK o NACK
- 8)El maestro debe finalizar la comunicación con una condición de STOP.

En pocas palabras , cada byte debe tener su correspondiente ACK(9no pulso de reloj) , si el esclavo no puede leer/escribir otro byte(está ocupado) puede forzar al master a esperarlo manteniendo SCL en bajo y se puede producir un NACK si es que el esclavo no está conectado y/o disponible . En este caso el master debe terminar la transferencia con un STOP o reintentar con un START y si el master realiza una operación de lectura con un NACK significa que es el fin de la comunicación y el esclavo libera SDA para que el master presente STOP.

2.3.2 Configuración

Se utilizó la configuración master mode , implementada en la teoría 17.Esta utiliza los registros TWBR(Registro de velocidad de bits TWI que selecciona el factor de división para el generador de velocidad de bits), TWCR(Registro de Control), TWSR(Registro de estado), TWDR(Registro de datos), TWAR(Registro de direcciones TWI (esclavo)) y TWAMR(Registro de máscara de dirección TWI (esclavo)) para configurar el hardware y transmitir y recibir datos.

A continuación se explican las funciones utilizadas

- **INICIO**

- El prescaler TWI en 1.

Al configurar TWSR con 0x00: Significa que el prescaler es 1 →
TWSR = 0x00;

- Configuración del TWI (I2C)

Para que funcione a una frecuencia de 100kHz con una frecuencia de CPU de 16 MHz.

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

$$100,000 = 16,000,000 / (16 + 2 \cdot TWBR \cdot 1) \rightarrow \mathbf{TWBR=72}$$

- Habilitar el TWI

-Configurar el bit TWEN en el registro de control de TWI (TWCR). $\rightarrow TWCR = (1 \ll TWEN);$

- **START**

- Función diseñada para enviar una condición de inicio en el bus I2C.
- Configurar el registro de control TWI (TWCR)

-TWSTA $\rightarrow 1$ Para iniciar una condición de inicio en el bus I2C.

-TWINT $\rightarrow 1$ para borrar el indicador de interrupción e inicia la condición de inicio.

-TWEN $\rightarrow 1$ para Habilitar el periférico TWI.

-TWCR = $(1 \ll TWSTA) | (1 \ll TWEN) | (1 \ll TWINT);$

- Esperando que se transmita la condición de inicio.

-Este bucle espera hasta que la condición de inicio se ha transmitido y el hardware TWI está listo para la siguiente operación.

-while $((TWCR \& (1 \ll TWINT)) == 0);$

- **STOP**

- Diseñada para enviar una condición de parada en el bus I2C.

- Configurar el registro de control TWI (TWCR)

-TWSTO \rightarrow Este bit se establece para iniciar una condición de parada en el bus I2C.

-TWINT $\rightarrow 1$ para borrar el indicador de interrupción e inicia la condición de inicio.

-TWEN \rightarrow 1 para Habilitar el periférico TWI.
-TWCR = $(1 \ll \text{TWINT}) \mid (1 \ll \text{TWEN}) \mid (1 \ll \text{TWSTO})$;

- **WRITE**

- Diseñada para transmitir datos a través del bus I2C
- Cargar datos -TWDR = data;
- Iniciar la transmisión de datos
-TWCR = $(1 \ll \text{TWEN}) \mid (1 \ll \text{TWINT})$;
- Esperar mientras se complete la transmisión
-while $((\text{TWCR} \& (1 \ll \text{TWINT})) == 0)$;

- **READ**

- Compruebe si se debe enviar ACK o NACK
 - if (isLast == 0)
 - ACK : Cuando el dispositivo recibe un byte de datos correctamente, enviará un bit ACK (nivel bajo) en el noveno ciclo de reloj.
Indica al emisor que el byte fue recibido y que ya podrá enviar el siguiente byte. En el caso de una transmisión de lectura, el dispositivo receptor (generalmente el maestro) genera el bit ACK para indicar que desea continuar leyendo más datos.
 $\text{TWCR} = (1 \ll \text{TWEN}) \mid (1 \ll \text{TWINT}) \mid (1 \ll \text{TWEA})$;
 - NACK: Se envía un bit NACK (Nivel Alto) en el noveno ciclo de reloj, para indicar que no se recibió el byte de datos correctamente, o que no desea recibir más datos.
 $\text{TWCR} = (1 \ll \text{TWEN}) \mid (1 \ll \text{TWINT})$;
- Espere a que se complete la recepción
-while $((\text{TWCR} \& (1 \ll \text{TWINT})) == 0)$;
- Devolver datos recibidos
return TWDR;

2.4 RTC DS3231

2.4.1 Introducción

El periférico RTC DS3231 es un reloj de tiempo real de alta precisión que mantiene un registro de segundos, minutos, horas, día, mes y año.



Figura 2.4.1. Módulo DS3231

- La fecha es ajustada automáticamente a final de mes para meses con menos de 31 días, incluyendo las correcciones para año bisiesto.
- Es capaz de generar señales de reloj cuadradas de frecuencia configurable.
- Puede generar interrupciones en el microcontrolador.
- El módulo se comunica con el microcontrolador a través del protocolo I2C.

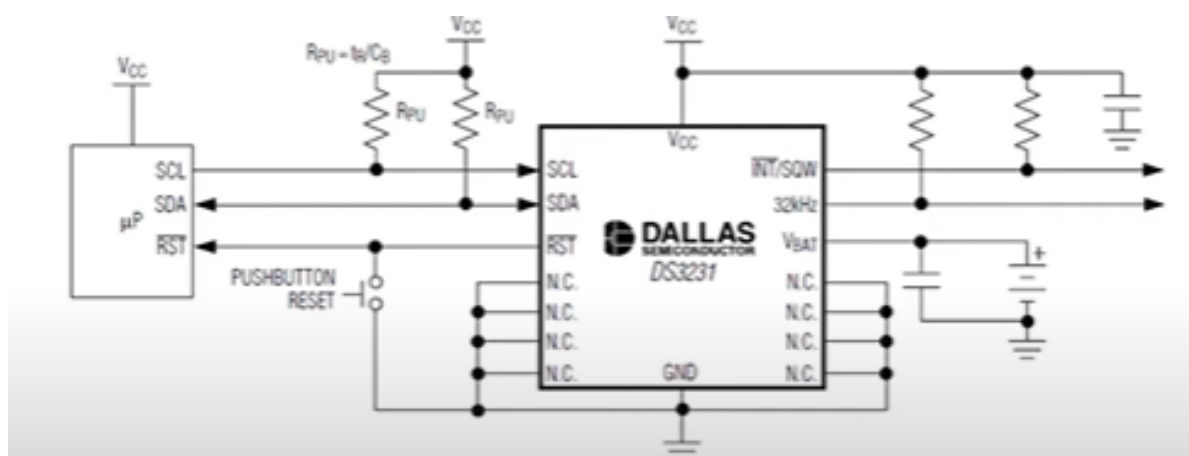


Figura 2.4.2 Conexiones del módulo

Conexión del Módulo:

- Vcc, GND → Son los pines de alimentación.

-SDA, SCL → Son del protocolo I2C y van conectados al microcontrolador.

2.4.2 Pseudocódigo

Para obtener la fecha y hora del módulo RTC DS3231 a través de la comunicación I2C. Se deberá crear una función que pase los datos BCD leídos del RTC a decimales. En este proyecto se encuentra con el nombre `bcd_to_dec()`.

- Pseudocódigo `bcd_to_dec()`
 - Separar los dos dígitos que están codificados en el valor BCD.
 - Cada uno se convierte a su equivalente decimal.
 - Se suman los valores decimales de los dos dígitos
 - Retorno dicho valor

Cabe aclarar que en el dato del año, vamos a sumarle el 24, así inicia desde el 2024. Lo mismo hacemos con día y mes, que si el valor que nos devuelve el RTC es 00, inicializamos en 01, ya que no existe el día/mes 00.

Creamos dos funciones distintas:

- `config_errorFecha(val)` → si dicho valor es 0, devuelve 1.
- `bcd_anio_dec(val)` → retorna la suma de 24 + el valor del año.

2.4.3 Pseudocódigo configuración de RTC con la interfaz I2C

Crear una variable de la estructura `DateTime`.
Iniciar la Comunicación I2C.
Escribir la Dirección del dispositivo RTC (DS3231) y bit de escritura
Escribir la dirección del primer registro (El de segundos).
Reiniciar la Comunicación para Leer
Escribir la Dirección y el Bit de Lectura
Leer los Datos de los Registros Consecutivos(seg,min,hora,dia,mes, anio)
Finalizar la Comunicación
Retorno la Estructura `DateTime`

2.5 Main

En este archivo se llaman al resto de las librerías. A continuación se detalla su funcionamiento:

Formato del buffer para imprimir de la siguiente manera:

"TEMP: C HUM: % FECHA: / / HORA : : \r\n"

Además se configuraron las interrupciones para recibir y enviar datos de la terminal. La interrupción para enviar datos, chequea que hayDatos se igual a 1, y procede a enviar todos los caracteres del buffer. Una vez que termina, hayDato pasa a valer 0, y se deshabilita la interrupción.

La interrupción para recibir datos, chequea el dato que llega, guardo si es S o s y espero por un enter. En caso de ser s/S alterno el valor del flag.

2.5.1 Pseudocódigo

```
int main(void) {
```

```
    Establecer un baud rate de 9600 bps con formato de datos 8N1.
```

```
    Habilitar tanto la transmisión como la recepción de datos en el puerto serie
```

```
    Habilitar la interrupción para la recepción de datos.
```

```
    Inicializar el módulo RTC DS3231.
```

```
    while infinito (while(1))
```

```
        Si el flag es xFF // El flag va alternando entre los valores [ x00 o xFF ] dependiendo si se presiona la S o s.
```

```
            Colocar hay datos en 1
```

```
            Actualizar buffer con dichos datos
```

```
            Habilitar interrupcion para imprimir dicho buffer.
```

```
            Delay de 2 segundos como indica el enunciado.
```

2.6 Background Foreground

Para resolver el problema se utilizó dicha arquitectura para determinar que tarea realizar el MCU. Se utilizaron dos flag para avisar al programa principal que ocurrió un evento, el primero (hayDato) indica que se debe transmitir todo lo hay en el buffer, mientras que no se terminen de enviar dichos datos el flag seguirá en uno y el segundo(flag) indica si se detiene o se reinicia la transmisión de datos de acuerdo a la tecla ingresada.

2.6.1 Pseudocódigo de tareas en foreground

- Interrupción de recepción de datos en la terminal (Esta interrupción se encarga de recibir los datos de la terminal, procesar los datos recibidos y alternar en valor de flag si se recibe un comando 's' o 'S')

```
Interrupción (USART_RX_vect:)
```

```
    Deshabilitar interrupción de transmisión UART
```

```
    Leer dato recibido en RX_BUFFER
```

```
    Si RX_BUFFER no es '\r'
```

```
        Guardar RX_BUFFER en comando[cant]
        Incrementar cant
Sino:
        comando[cant] = '\n'
Si comando[0] es 's' o 'S':
    Alternar valor de flag
```

- Interrupción de transmisión de datos (Esta interrupción se encarga de enviar los datos de manera secuencial desde el buffer a la terminal y deshabilitar la interrupción de transmisión una vez que se han enviado todos los datos.

Interrupción USART_UDRE_vect:

Si hay dato para enviar:

Enviar buffer[i] por UART

Incrementar i

Si buffer[i] es '\0':

Reiniciar i a 0

Resetear hayDato

Deshabilitar interrupción de transmisión UART

2.6.2 Pseudocódigo de tarea en background

Esta tarea se lleva a cabo en el programa principal:

Si flag está habilitado:

Esperar 2000 ms

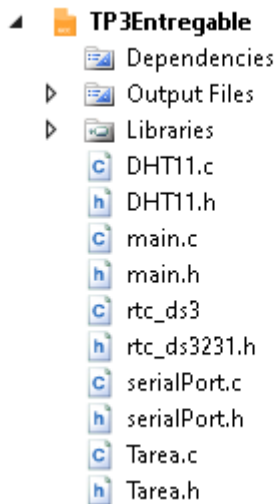
Establecer hayDato a 1

Llamar a ArmarBuffer para preparar datos

Habilitar interrupción de transmisión UART

2.7 Modularización

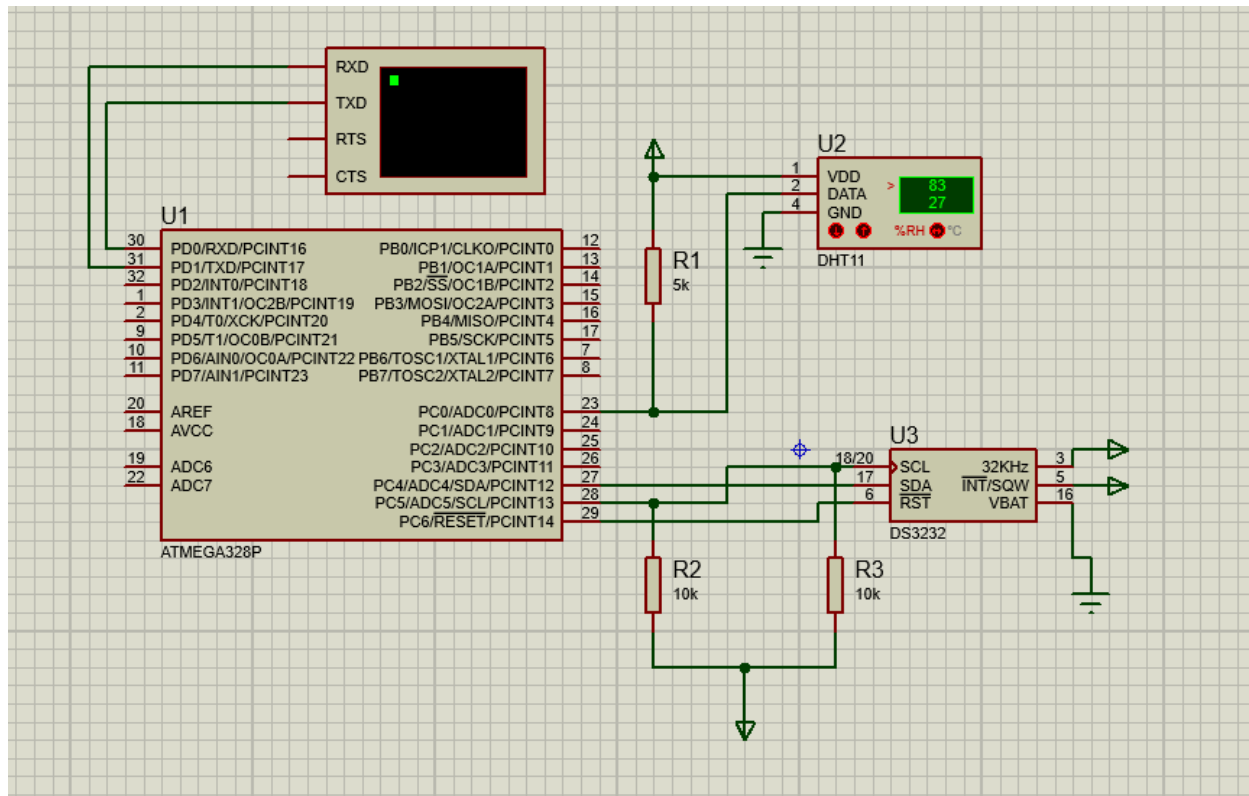
A continuación se muestra una imagen donde se podrán observar los distintos archivos utilizados para la solución del problema.



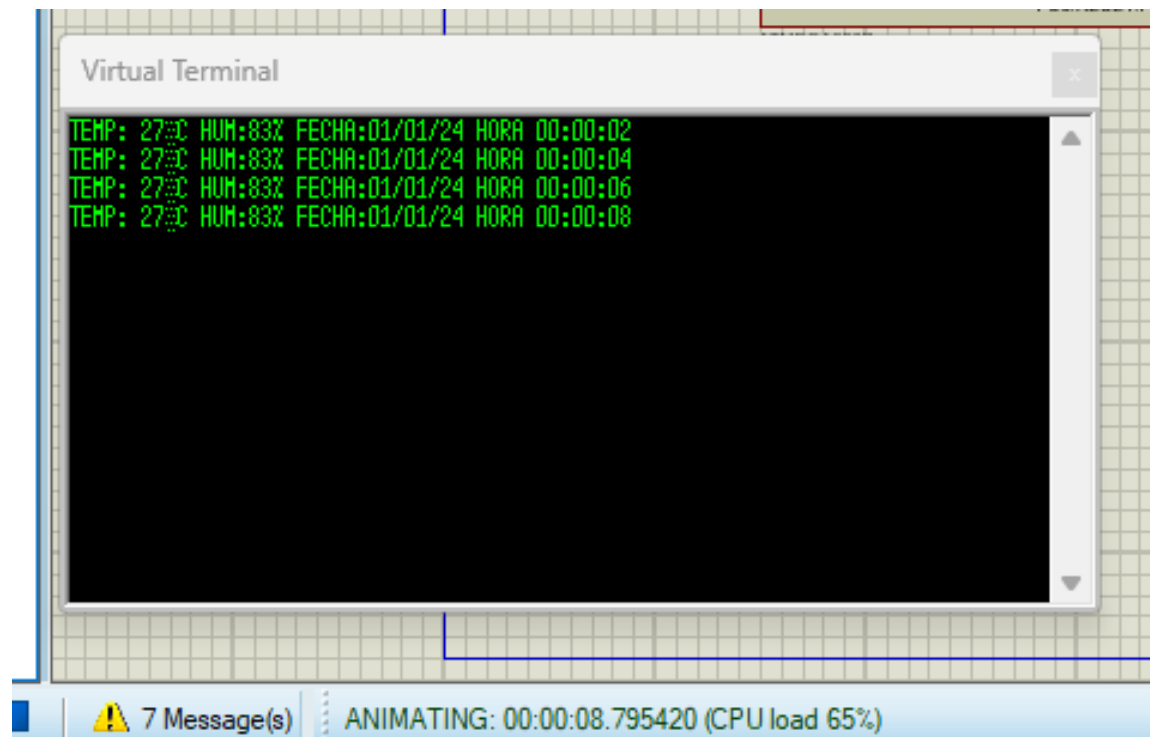
- DHT11 se encarga de la configuración del sensor a través de los métodos `DHT11_init()` y `DHT11_checkResponse()`; y devuelve los datos medidos por el sensor a través del método `DHT11_read(uint8_t *hE, uint8_t *tE)`.
- serialPort es la librería brindada por la cátedra para configurar y utilizar el puerto serie del MCU
- Tarea utiliza los archivos DHT11 y `rtc_ds3231` para armar un buffer con los datos obtenidos de dichos módulos. Para que los datos se guarden correctamente en el buffer se deben convertir a string, dividiendo primero por 10 y luego realizando el módulo. Por ejemplo, si se recibe el 23, guardo el 2 en la primera posición (correspondiente a la temperatura) y el 3 en la segunda posición.
- `rtc_ds3231` se encarga de la configuración de la interfaz I2C a través de los siguientes módulos: `TWI_Init()`, `TWI_Start()`, `TWI_Stop()`, `TWI_Write()` y `TWI_Read()`; y cuenta con el módulo `RTC_DS3231_Get()`, que es el encargado de iniciar y terminar la comunicación I2C para poder obtener los datos de fecha y hora.

2.8 Validación

Para poder validar el programa realizado, se recreó en proteus dicho modelo, el cual cuenta con un módulo DS3232, un sensor DHT11 y SerialPort. Dentro del directorio de GitHub se podrá encontrar el archivo como [TP3Entrega.pdsprj](#)



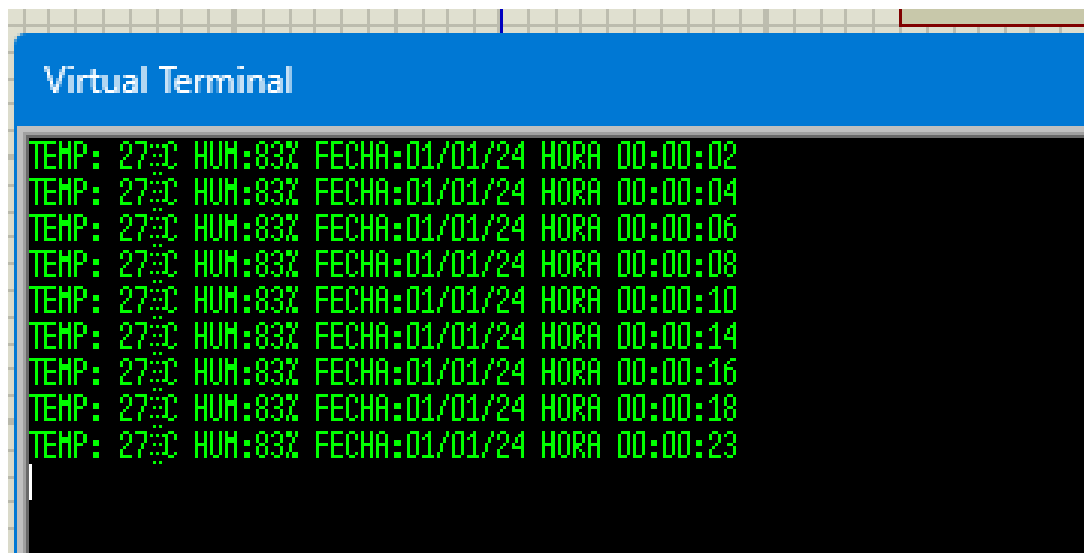
Iniciamos la simulación y dejamos correr 8 segundos (Se imprime cada 2 seg):



Se presiona 'S/s' para detener el envío de datos, y luego se vuelve presionar 'S/s' para volver a poner en marcha.

-En el segundo 10 pausamos.

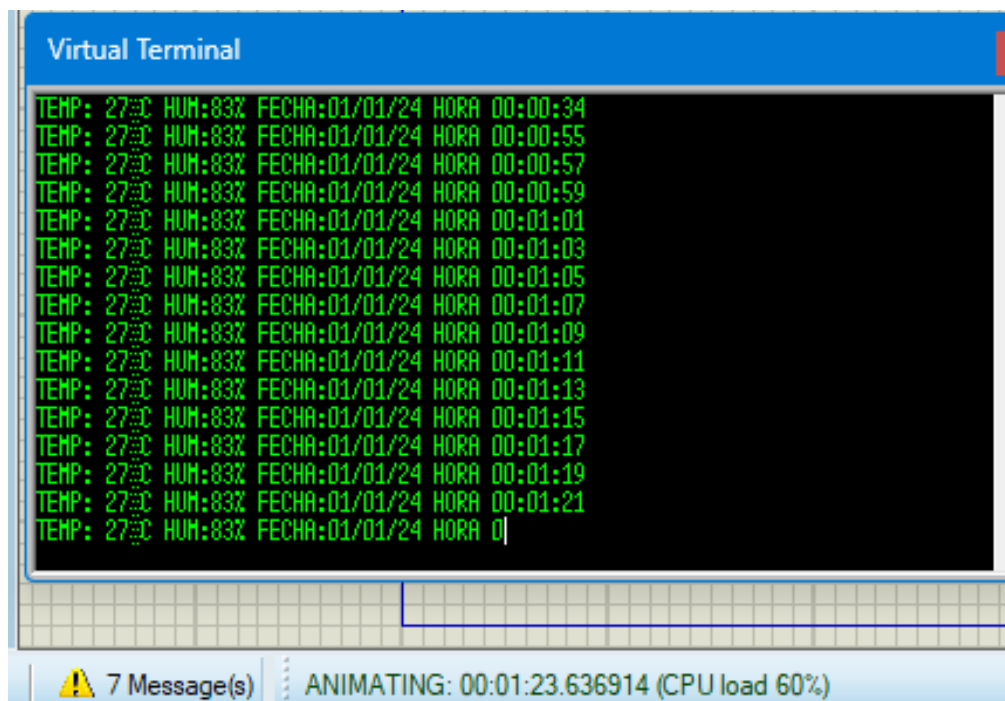
- En el segundo 14 volvemos a poner en marcha.
- En el segundo 18 pausamos.
- En el segundo 23 volvemos a poner en marcha.



A screenshot of a 'Virtual Terminal' window with a blue title bar. The terminal displays a series of log entries in green text on a black background. Each entry contains sensor data: temperature (TEMP: 27°C), humidity (HUM: 83%), date (FECHA: 01/01/24), and time (HORA). The times shown are 00:00:02, 00:00:04, 00:00:06, 00:00:08, 00:00:10, 00:00:14, 00:00:16, 00:00:18, and 00:00:23.

```
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:02
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:04
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:06
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:08
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:10
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:14
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:16
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:18
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:23
```

Dejamos correr la simulación para mostrar que funciona el minuterio del DS3231:

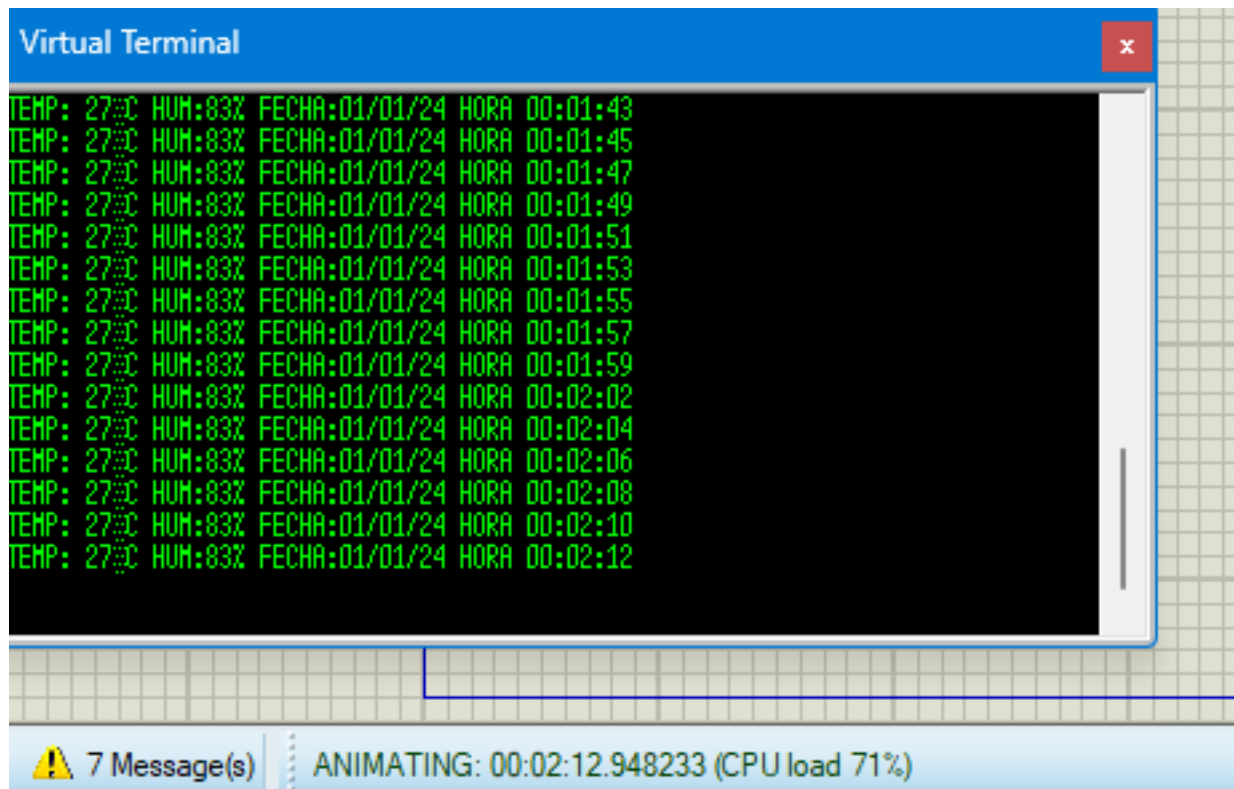


A screenshot of a 'Virtual Terminal' window showing a continuation of the log entries. The times progress from 00:00:34 to 00:01:21. The final line shows the time as 0. Below the terminal window, a status bar indicates '7 Message(s)' and 'ANIMATING: 00:01:23.636914 (CPU load 60%)'.

```
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:34
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:55
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:57
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:00:59
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:01
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:03
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:05
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:07
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:09
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:11
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:13
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:15
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:17
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:19
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 00:01:21
TEMP: 27°C HUM: 83% FECHA: 01/01/24 HORA 0
```

7 Message(s) ANIMATING: 00:01:23.636914 (CPU load 60%)

Luego de 2 minutos:



2.9 CÓDIGO GitHub

<https://github.com/LucasCarba/CDyM-TP3-G20/tree/7167e964a719a1e462726ca05fe4bccb2d9a8ef8>