Logotipo

Descripción generada automáticamente

TRABAJO PRÁCTICO N°3

Grupo 5

Caciani Toniolo, Melina 02866/1

Larsen, Mateo Emmanuel 02993/7

Niderhaus, Franco 02976/6

UNLP

Facultad de Ingeniería

Departamento de Electrotecnia

Curso 2024 –1er Cuatrimestre

Circuitos Digitales y Microcontroladores (E305)

# ÍNDICE

**INTRODUCCIÓN ……………...…………………………………………… 3**

**PROBLEMA ……………………………………………………………………. 3**

**INTERPRETACIÓN ……………………………………………………… 4**

**RESOLUCIÓN …………………………………………………………………. 5**

**PERIFÉRICOS ………………………………………………………………. 5**

**SENSOR DHT11 …...……………………………………………………. 6**

**MÓDULO RTC…………………………………………………………... 7**

**TERMINAL VISUAL……………………………………………………. 8**

**PSEUDOCÓDIGO………………………………………………………. 8**

**SOFTWARE …………………………………………………………………. 13**

**TIMER ……………………………………………………………… 13**

**ARQUITECTURA ………………………………………………… 14**

**MODULARIZACIÓN……………………………………………………… 15**

**VALIDACIÓN …………………………………………………………………17**

**CONCLUSIÓN ……………………………………………………………… 20**

# INTRODUCCIÓN

Este informe muestra y explica la resolución del Trabajo Práctico N3 de Circuitos Digitales y Microcontroladores, el cual tenía como objetivo la implementación de un registrador de temperatura y humedad relativa ambiente. Para realizar esto se utilizó un sensor DHT11, un módulo RTC DS3231 y el MCU Atmega328p. Además, se hizo uso del software Microchip Studio para la escritura del código y su vinculación con el microcontrolador.

# 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 enumeran los requerimientos que debe satisfacer el sistema:

1. El MCU deberá encuestar al sensor para obtener una medida de la temperatura y la humedad relativa cada 2seg.
2. 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.
3. 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”
4. El envío de datos se podrá detener o reanudar desde la PC presionando la tecla ‘s’ o ‘S’ sucesivamente.
5. La comunicación serie asincrónica deberá implementarse utilizando interrupciones de recepción y transmisión del periférico UART0.

# INTERPRETACIÓN

Este trabajo se basa en realizar un registrador de temperatura y humedad relativa ambiente. Para lograr esto será necesario el uso del sensor DHT11, el cual otorga los valores reales de la temperatura y la humedad. Además, se hará uso del módulo RTC DS3231, este brinda la fecha y hora actual. Estos periféricos deben conectarse al kit del MCU, el cual se conecta a la PC.

El proyecto requiere que el MCU realice varias tareas específicas. Primero, el MCU debe encuestar al sensor DHT11, conectado a PORTC0, cada 2 segundos para obtener medidas de temperatura y humedad relativa. Esto implica que el MCU debe estar programado para realizar lecturas periódicas del sensor, manejando la temporización adecuada para garantizar que las lecturas se realicen exactamente cada 2 segundos. Se consideró más eficiente la utilización del TIMER 1 para lograr interrupciones cada 2 segundos, ya que es posible configurarlo de tal manera que se realicen interrupciones en el tiempo deseado; mientras que con el TIMER0 sería necesario un contador para lograr esto.

Además, el sistema debe utilizar el módulo RTC DS3231 para completar el registro de datos, agregando la fecha y hora actuales a cada medida obtenida del sensor DHT11. Para lograr esto, el MCU debe estar configurado para comunicarse con el módulo RTC a través de la interfaz I2C. Esta comunicación permitirá obtener la fecha y hora actual en cada ciclo de lectura del sensor, asegurando que cada registro de temperatura y humedad tenga una marca de tiempo precisa. Se debe inicializar el RTC con la hora y la fecha actual.

Los datos obtenidos deben ser formateados adecuadamente para su transmisión a una terminal en la PC. El formato sugerido para los datos es “TEMP: 20 °C HUM: 40% FECHA: 10/06/24 HORA:15:30:56\r\n”.

El envío de estos datos debe poder detenerse o reanudarse desde la PC, mediante la presión de la tecla 's' o 'S'. Esto significa que el MCU debe estar programado para detectar y responder a estos comandos de la PC.

Finalmente, la comunicación serie asincrónica debe implementarse utilizando interrupciones de recepción y transmisión del periférico UART0 del MCU. Las interrupciones permitirán que el MCU responda de inmediato a las señales de la PC sin necesidad de estar constantemente verificando el estado de la comunicación.

Un detalle sin aclarar es qué sucede si se desconecta alguno de los periféricos. Se decidió que al desconectarse el sensor DHT11, se emita un mensaje de error informando lo sucedido, en cambio, si se desconecta el RTC sigue funcionando sin dar valores certeros para la fecha y hora.

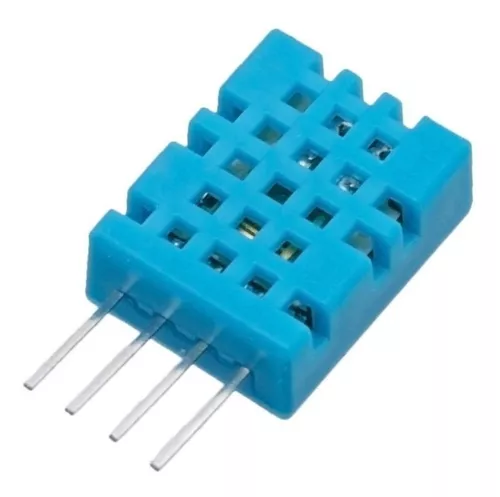
# RESOLUCIÓN

* **Periféricos**

**Diagrama, Esquemático

Descripción generada automáticamente**

**Sensor DHT11**

****

Para su correcto funcionamiento, se conectó el PORTC0 al pin de DATA del sensor, además de conectar sus correspondientes terminales a tierra y a Vcc. También fue necesaria la realización de un Pull Up. (En Proteus).

* Inicialización de la comunicación: el MCU inicia la comunicación configurando el pin del DHT11 como salida y enviando una señal LOW durante 18 milisegundos, seguida de una señal HIGH durante 20 a 40 microsegundos. Esto despierta al sensor y lo prepara para enviar datos.
* Espera de la respuesta del sensor: tras la inicialización, el pin se configura como entrada. El MCU espera que el DHT11 responda con una secuencia de señales LOW y HIGH. Esta secuencia de espera se realiza en tres etapas: primero, el MCU espera a que el pin se ponga en LOW, luego en HIGH, y nuevamente en LOW. Si alguna de estas etapas excede el tiempo de espera predeterminado, se considera que hubo una falla en la comunicación.
* Lectura de datos: una vez que el sensor responde correctamente, el MCU lee 5 bytes de datos del DHT11. Estos bytes contienen la humedad y la temperatura medidas, así como una suma de comprobación para verificar la integridad de los datos. La humedad se almacena en el primer byte y la temperatura en el tercer byte.
* Verificación de datos: después de leer los datos, el MCU verifica la suma de comprobación. Esta verificación se realiza sumando los valores de humedad y temperatura y comparándolos con el valor de la suma de comprobación proporcionada por el sensor. Si los valores no coinciden, se registra una falla en la lectura.

**Módulo RTC DS3231**

Un circuito electrónico

Descripción generada automáticamente con confianza baja

En Proteus, se utilizó el módulo RTC DS3232, ya que el DS3231 no estaba disponible. Fue necesario agregar un Pull Up en cada entrada, las cuales se conectaron a los puertos PC4 y PC5.

El DS3231 se comunica con el MCU a través del bus I2C. En el código, la dirección del DS3231 en el bus I2C es por defecto 0x68. Para facilitar la comunicación con el módulo, se utilizan funciones auxiliares que convierten entre BCD y decimal. Estas conversiones son necesarias porque el DS3231 utiliza BCD para almacenar la hora y la fecha. La función BCD\_to\_DEC convierte de BCD a decimal, mientras que la función DEC\_to\_BCD convierte de decimal a BCD.

* Configuración de la hora/fecha: la función DS3231\_SetTime se encarga de establecer la hora en el DS3231. Esta función inicia la comunicación I2C y envía la dirección del DS3231 en modo escritura. Luego, especifica el registro inicial, que es 0x00 para empezar a escribir desde los segundos. Posteriormente, se escriben los valores de segundos, minutos, horas, día de la semana, día del mes, mes y año, todos convertidos de decimal a BCD. Finalmente, la comunicación I2C se termina.
* Obtención de la hora/fecha: la función DS3231\_GetTime permite leer la hora actual del DS3231. El proceso comienza iniciando la comunicación I2C y enviando la dirección del DS3231 en modo escritura. Se especifica nuevamente el registro inicial, 0x00, para comenzar a leer desde los segundos. Después, la comunicación se detiene y se reinicia en modo lectura. Los valores de segundos, minutos, horas, día de la semana, día del mes, mes y año se leen y se convierten de BCD a decimal. La comunicación I2C finaliza.

**Terminal visual**

En Proteus, se conectaron los pines RXD y TXD de la terminal con los puertos TXD y RXD del MCU, respectivamente.

La comunicación serie asincrónica debía implementarse utilizando interrupciones de recepción y transmisión del periférico UART0. Para esto se hizo uso de la librería *serialPort.h.* A esta librería se le aplicaron algunas modificaciones para que pueda implementarse utilizando interrupciones de transmisión y recepción.

Se creó una interrupción para que cada vez que el buffer esté libre, active un flag que habilita el envío de un nuevo caracter del mensaje a la terminal. Dentro de ella era necesario deshabilitar las interrupciones de transmisión y que se vuelvan a habilitar luego del envío del caracter.

También fue necesaria la implementación de una interrupción que detecte cuando se escribe un dato, en este caso, “S” o “s” utilizada para pausar la comunicación.

* **Pseudocódigo de los drivers para el control de los periféricos**
* Funcionamiento del driver para el sensor DHT11, incluyendo la inicialización, la espera de respuestas y la lectura de datos del sensor.

// Definiciones y variables globales

Definir DHT11\_PIN como 0

Definir DHT11\_DDR, DHT11\_PORT, DHT11\_PIN\_INPUT como registros de puerto

Declarar temperature, humidity, check\_sum, fallo como variables volátiles de 8 bits

Función read\_dht11:

Inicializar un arreglo de 5 elementos a 0

Inicializar timeout como variable de 16 bits

// Iniciar comunicación con el sensor

Configurar DHT11\_PIN como salida

Establecer DHT11\_PIN a LOW

Esperar 18 ms

Establecer DHT11\_PIN a HIGH

Esperar 20-40 us

Configurar DHT11\_PIN como entrada

// Esperar la respuesta del sensor

Inicializar timeout en 10000

Mientras DHT11\_PIN esté en HIGH:

Decrementar timeout

Si timeout llega a 0:

Establecer fallo a 1

Terminar función

Inicializar timeout en 10000

Mientras DHT11\_PIN esté en LOW:

Decrementar timeout

Si timeout llega a 0:

Establecer fallo a 1

Terminar función

Inicializar timeout en 10000

Mientras DHT11\_PIN esté en HIGH:

Decrementar timeout

Si timeout llega a 0:

Establecer fallo a 1

Terminar función

// Leer los 5 bytes de datos

Para i desde 0 hasta 4:

data[i] = Llamar a read\_byte()

// Asignar valores leídos a variables

Asignar data[0] a humidity

Asignar data[2] a temperature

Asignar data[4] a check\_sum

// Verificar la suma de comprobación

Si (humidity + temperature) es diferente de check\_sum:

Establecer fallo a 1

De lo contrario:

Establecer fallo a 0

Función read\_byte:

Inicializar result a 0

Para i desde 0 hasta 7:

Mientras DHT11\_PIN esté en LOW:

Esperar

Esperar 30 us

Si DHT11\_PIN está en HIGH:

Establecer el bit (7 - i) de result a 1

Mientras DHT11\_PIN esté en HIGH:

Esperar

Retornar result

* Funciones de configuración y lectura del tiempo en el DS3231 usando I2C.

// Definir dirección del DS3231 en el bus I2C

Definir DS3231\_ADDRESS como 0x68

// Función para convertir de BCD a decimal

Función BCD\_to\_DEC(valor):

Retornar (valor / 16 \* 10) + (valor % 16)

// Función para convertir de decimal a BCD

Función DEC\_to\_BCD(valor):

Retornar (valor / 10 \* 16) + (valor % 10)

// Función para establecer la hora en el DS3231

Función DS3231\_SetTime(tiempo):

Iniciar comunicación I2C

Escribir dirección DS3231 en modo escritura

Escribir 0x00 (registro de segundos)

Escribir tiempo.seconds en BCD

Escribir tiempo.minutes en BCD

Escribir tiempo.hours en BCD

Escribir tiempo.day en BCD

Escribir tiempo.date en BCD

Escribir tiempo.month en BCD

Escribir tiempo.year en BCD

Terminar comunicación I2C

// Función para obtener la hora del DS3231

Función DS3231\_GetTime(tiempo):

Iniciar comunicación I2C

Escribir dirección DS3231 en modo escritura

Escribir 0x00 (registro de segundos)

Terminar comunicación I2C

Iniciar comunicación I2C nuevamente

Escribir dirección DS3231 en modo lectura

Leer y convertir tiempo.seconds de BCD a decimal con ACK

Leer y convertir tiempo.minutes de BCD a decimal con ACK

Leer y convertir tiempo.hours de BCD a decimal con ACK

Leer y convertir tiempo.day de BCD a decimal con ACK

Leer y convertir tiempo.date de BCD a decimal con ACK

Leer y convertir tiempo.month de BCD a decimal con ACK

Leer y convertir tiempo.year de BCD a decimal con NACK

Terminar comunicación I2C

* Operaciones básicas del protocolo I2C, incluyendo la inicialización, las condiciones de START y STOP, la escritura de datos y la lectura de datos con y sin ACK.

// Función para inicializar I2C

Función I2C\_Init:

Configurar prescaler a 1

Configurar Bit Rate para 400kHz

Habilitar el módulo TWI

// Función para iniciar la condición de START en I2C

Función I2C\_Start:

Iniciar condición de START

Esperar a que se complete la condición de START

// Función para generar la condición de STOP en I2C

Función I2C\_Stop:

Generar condición de STOP

Esperar a que se complete la condición de STOP

// Función para escribir un byte de datos en I2C

Función I2C\_Write(dato):

Cargar dato en el registro de datos del TWI

Iniciar la transmisión

Esperar a que se complete la transmisión

// Función para leer un byte de datos con ACK en I2C

Función I2C\_Read\_ACK:

Iniciar la recepción y enviar un ACK

Esperar a que se complete la recepción

Retornar el dato recibido

// Función para leer un byte de datos con NACK en I2C

Función I2C\_Read\_NACK:

Iniciar la recepción y no enviar un ACK

Esperar a que se complete la recepción

Retornar el dato recibido

* Funciones principales para inicializar y manejar la comunicación serie UART, incluyendo la configuración, transmisión y recepción de datos.

// Variable global para indicar si el puerto está libre

Definir flagLibre como 0

// Inicialización de Puerto Serie

Función SerialPort\_Init(config):

Deshabilitar UART

Configurar UART para 8 bits de datos

Configurar baud rate según config

// Inicialización de Transmisor

Función SerialPort\_TX\_Enable:

Habilitar transmisión UART

Función SerialPort\_TX\_Interrupt\_Enable:

Habilitar interrupción de transmisión UART

Función SerialPort\_TX\_Interrupt\_Disable:

Deshabilitar interrupción de transmisión UART

// Inicialización de Receptor

Función SerialPort\_RX\_Enable:

Habilitar recepción UART

Función SerialPort\_RX\_Interrupt\_Enable:

Habilitar interrupción de recepción UART

// Transmisión de Datos

Función SerialPort\_Send\_Data(data):

Enviar dato a través de UART

Función SerialPort\_Send\_String(msg):

Definir i como 0

Mientras msg[i] no sea el fin de cadena:

Si flagLibre está activo:

Enviar msg[i]

Incrementar i

Desactivar flagLibre

Habilitar interrupción de transmisión UART

// Recepción de Datos

Función SerialPort\_Wait\_Until\_New\_Data:

Esperar hasta que el buffer de recepción esté completo

Función SerialPort\_Receive\_Data:

Retornar dato recibido a través de UART

Función SerialPort\_Send\_uint8\_t(num):

Esperar hasta que el buffer de transmisión esté libre

Enviar centenas de num como carácter

Restar 100 a num

Esperar hasta que el buffer de transmisión esté libre

Enviar decenas de num como carácter

Restar 10 a num

Esperar hasta que el buffer de transmisión esté libre

Enviar unidades de num como carácter

// Enviar un valor int16\_t a través de UART

Función SerialPort\_send\_int16\_t(val, field\_length):

Inicializar arreglo str de 5 caracteres a 0

Definir i como 4, j como 0

Mientras val no sea 0:

Almacenar dígito en str[i]

Dividir val por 10

Decrementar i

Si field\_length es -1:

Ajustar j a la posición del primer dígito significativo

De lo contrario:

Ajustar j a 5 - field\_length

Si val es negativo:

Esperar hasta que el buffer de transmisión esté libre

Enviar carácter '-'

Para cada dígito en str desde j hasta 4:

Esperar hasta que el buffer de transmisión esté libre

Enviar dígito como carácter

* **Software**

**TIMER**

Configuración:

Se configura un Timer1 en modo CTC (TCCR1B |= (1 << WGM12)).

Se configura el prescaler en 1024 ( TCCR1B |= (1 << CS12) | (1 << CS10) ).

Para generar interrupciones cada 2 segundos, configuramos el OCR1A = 31249. Este valor se obtuvo a partir de la siguiente ecuación:

**ARQUITECTURA**

**Background – Foreground**

* Foreground (Interrupciones y Flags)
* Interrupción de Temporizador (TIMER1\_COMPA\_vect): Se utiliza para activar el flag que indica cuándo es necesario encuestar el sensor DHT11.
* Interrupción de Recepción UART (USART\_RX\_vect): Maneja la recepción de datos por UART, cambiando el estado de send\_data cuando se recibe 's' o 'S', provocando que se pause o reanude el envío de datos según corresponda.
* Interrupción de Transmisión UART (USART\_UDRE\_vect): Indica cuándo el buffer de transmisión está libre para enviar más datos.

Estas últimas dos son las mencionadas anteriormente en el apartado de *Terminal Visual*.

Pseudocódigo

// Interrupción de recepción UART

Al recibir un dato por UART {

Leer el dato recibido y almacenarlo;

Si el dato es 's' o 'S' {

Invertir el estado de send\_data;

}

}

// Interrupción de transmisión UART

Al estar listo el buffer de transmisión {

Indicar que el buffer de transmisión está libre;

Deshabilitar esta interrupción;

}

// Interrupción de comparación del TIMER1

Al llegar al valor de comparación en TIMER1 {

Activar un flag para indicar la lectura del sensor DHT11;

}

* Background (Bucle Principal)
* Lectura de Sensores: El bucle principal comprueba el flag para realizar la lectura del sensor DHT11. Tras completar la lectura, establece el flag dht11\_ready indicando que los datos están listos.
* Envío de Datos: El bucle principal también verifica si send\_data está habilitado y si el sensor ha proporcionado datos (dht11\_ready). Si ambas condiciones se cumplen, envía los datos formateados por el puerto serie y resetea el flag de preparación (dht11\_ready).

Pseudocódigo

mientras (1): // bucle infinito

si el flag de lectura del sensor DHT11 está activado:

resetear el flag

marcar que los datos del DHT11 están listos

leer datos del sensor DHT11

si el envío de datos está habilitado y los datos del DHT11 están listos:

resetear el flag de datos del DHT11

reiniciar temporizador

obtener la hora actual del RTC

definir un buffer para el mensaje

si no hay fallo en el sensor:

formatear mensaje con datos de temperatura, humedad, fecha y hora

si hay fallo en el sensor:

formatear mensaje de fallo del sensor

enviar mensaje formateado por el puerto serie

* **Modularización**

La modularización del programa se realiza dividiendo el código en múltiples archivos y módulos, cada uno responsable de una funcionalidad específica, lo que facilita el mantenimiento y la escalabilidad del programa. Este programa fue modularizado en 11 archivos:

1. *main.c:* Función principal del programa.

Responsabilidades:

* + - Inicializar el temporizador usando *timer.c*
    - Inicializar el I2C usando *i2c.c*
    - Inicializar el UART usando *serialPort.c*
    - Recibir los datos del DS3231 y del DHT11
    - Enviar los datos recibidos a la terminal

1. *dht11.c:* Implementa la lógica del sensor de temperatura DHT11.

Funciones: *read\_dht11(), read\_byte()*

Responsabilidades:

* Iniciar la comunicación con el sensor
* Recibir respuestas del sensor
* Verificar la suma de comprobación

1. *dht11.h*: Declara las funciones y variables globales relacionadas con el sensor DHT11.
2. *ds3231.c:* Contiene las funciones para manejar el módulo RTC DS3231. Funciones:*DS3231\_SetTime(DS3231\_Time\*time), DS3231\_GetTime(DS3231\_Time \*time), BCD\_to\_DEC(uint8\_t val), DEC\_to\_BCD(uint8\_t val).*

Responsabilidades:

* Establecer y terminar la comunicación I2C mediante *idc.c*
* Convertir de decimal a BCD y viceversa
* Inicializar la hora/fecha actual en el sensor
* Obtener la hora/fecha del sensor

1. *ds3231.h:* Declara las funciones y variables globales relacionadas con el módulo RTC.
2. *i2c.c:* Implementa las funciones para manejar la comunicación I2C. Funciones: *I2C\_Init(), I2C\_Start(), I2C\_Stop(), I2C\_Write(uint8\_t data), I2C\_Read\_ACK(), I2C\_Read\_NACK().*

Responsabilidades:

* Inicializar y configurar la comunicación I2C
* Realizar la comunicación I2C

1. *i2c.h:* Declara las funciones y variables globales relacionadas con el I2C. Provee una interfaz para interactuar con el I2C desde otros módulos.
2. *timer.c:* Contiene las funciones para configurar y manejar el temporizador. Incluye la lógica para generar interrupciones periódicas. Funciones: *configurarTimer(), reiniciarTimer().* Interrupción: *ISR(TIMER1\_COMPA\_vect).*

Responsabilidades:

* Configurar el temporizador para generar interrupciones periódicas.
* Habilitar las interrupciones con *sei().*
* Genera una interrupción cada 2 segundos.

1. *timer.h:* Declara las funciones y variables globales relacionadas con el temporizador. Provee una interfaz para interactuar con el temporizador desde otros módulos.
2. *serialPort.c:* contiene las funciones para la comunicación UART.

Funciones: *SerialPort\_Init(uint8\_t config), SerialPort\_Send\_String(buffer), SerialPort\_TX\_Enable(), SerialPort\_RX\_Enable(), SerialPort\_TX\_Interrupt\_Enable(), SerialPort\_Send\_String(char \* msg), SerialPort\_RX\_Interrupt\_Enable(), SerialPort\_TX\_Interrupt\_Disable().*

Responsabilidades:

* Inicializar transmisor, receptor y puerto serie
* Permitir la transmisión de caracteres a la terminal

1. *serialPort.h:* Declara las funciones y variables globales relacionadas con la comunicación UART.

# VALIDACIÓN

Para demostrar el correcto funcionamiento del programa, se verificarán los siguientes puntos:

1. Que muestre la temperatura, humedad, fecha y hora en el formato deseado.
2. Que la temperatura aumente o descienda según corresponda.
3. Que la humedad aumente o descienda según corresponda.
4. Que al presionar “S” o “s”, se pausa la transmisión.
5. Que de error al desconectar el sensor DHT11.
6. Que las interrupciones ocurren cada dos segundos.

Mostraremos estos puntos al realizar la simulación en Proteus. También pueden verse realizando la simulación con el MCU y los sensores reales en el siguiente video:

//link

1. Imagen que contiene Interfaz de usuario gráfica

   Descripción generada automáticamente
2. Interfaz de usuario gráfica

   Descripción generada automáticamente
3. Imagen de la pantalla de una computadora

   Descripción generada automáticamente con confianza baja
4. (PAUSAR POR 12 SEGUNDOS)

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. Diagrama, Esquemático

   Descripción generada automáticamente

# CONCLUSIÓN

En este trabajo práctico se desarrolló un registrador de temperatura y humedad relativa utilizando el sensor DHT11, el módulo RTC DS3231 y el microcontrolador Atmega328p. A través del uso de Microchip Studio, se implementaron varios aspectos clave que destacan las ventajas de ciertas arquitecturas y métodos de comunicación. El uso de estas arquitecturas y métodos de comunicación, junto con la modularización del código, permitió implementar un sistema robusto, eficiente y fiable para el registro y transmisión de datos de temperatura y humedad con sus correspondientes marcas de tiempo.

1. **Arquitectura Background y Foreground**:
   * **Background**: La arquitectura background permitió la ejecución de tareas de menor prioridad en segundo plano sin interferir con las tareas críticas. Esto fue fundamental para el manejo de la comunicación con el sensor DHT11 y el módulo RTC DS3231, asegurando que las lecturas de temperatura y humedad, así como la actualización de la fecha y hora, se realizaran de manera eficiente.
   * **Foreground**: Las tareas de mayor prioridad, como el procesamiento de interrupciones y la actualización de datos críticos, se gestionaron en primer plano. Esto garantizó la respuesta rápida del sistema a eventos importantes, como la recepción de comandos desde la PC para detener o reanudar el envío de datos.
2. **Comunicación Asincrónica UART**:
   * La implementación de la comunicación serie asincrónica mediante el periférico UART0 y el uso de interrupciones para la recepción y transmisión de datos permitió una comunicación eficiente y no bloqueante con la PC. Esto significa que el microcontrolador podía seguir ejecutando otras tareas mientras esperaba o enviaba datos, mejorando la eficiencia general del sistema y reduciendo el tiempo de inactividad.
3. **Comunicación Serie Sincrónica I2C**:
   * El uso de la comunicación sincrónica I2C para interactuar con el módulo RTC DS3231 permitió una transferencia de datos fiable y sincronizada entre el microcontrolador y el RTC. Aseguró una transferencia de datos precisa y coherente, esencial para mantener la integridad temporal en el registro de datos.
4. **Modularización**:
   * La modularización del código permitió una mayor organización y claridad, facilitando la reutilización y el mantenimiento del software. Cada módulo fue diseñado para gestionar una tarea específica, como la lectura del sensor, la comunicación con el RTC, o la transmisión de datos mediante UART. Esto no solo mejoró la legibilidad del código, sino que también hizo más sencillo realizar pruebas unitarias, ya que cada módulo podía ser verificado de manera independiente.