UNIVERSIDAD
NACIONAL
DE LA PLATA

Facultad de Ingeniería - UNLP
Circuitos Digitales y Microcontroladores (E1305 / E0305)
Curso 2022 - Trabajo Práctico Nº3

Grupo 2

Alumnos: Santana Valentin, Guerrero Nicolás

Índice:

1. Registrador de datos de temperatura y humedad ambiente 3

1.1 Propuesta....3

1.2 Resolución del problema....4

2. Arquitectura: Background/Foreground 4

2.1. Definición....4

2.2 main....5

3. Sensor de temperatura y humedad 5

3.1. Introducción....5

3.2. Desarrollo....5

4. UART(Periférico de comunicación serie asincrónico) 7

4.1. UART: definición....7

4.2. UART: desarrollo....7

5. Eventos de Tareas 8

5.1 Introducción....8

5.2 Eventos de Tareas: Tarea 1....8

5.3 Eventos de Tareas: Tarea 2....9

6. Conclusiones 9

6.1 Problemas encontrados....9

6.2 Validación....10

7. Anexo 11

7.1 Archivos cabecera....11

7.2 Archivos ".c"....13

1. Registrador de datos de temperatura y humedad ambiente

1.1 Propuesta

Implementar un registrador de datos de temperatura y humedad ambiente a partir del sensor digital DHT11. Al iniciar el sistema deberá presentarse en la pantalla de una terminal serie el menú de opciones para que el usuario comande el registrador. Cuando el usuario encienda el registrador, el MCU obtendrá los datos del sensor DHT11 y los presentará en la terminal serie cada 1 segundo hasta que el usuario decida enviar otro comando. Deberá realizar la verificación de los comandos recibidos y en el caso que no corresponda a lo especificado deberá enviar el mensaje “Comando no válido” y seguirá realizando la misma tarea.

Respecto a la arquitectura del programa, la misma deberá ser del tipo Background/Foreground y además se deberá aplicar modularización, abstracción y el modelo productor-consumidor para el periférico UART, es decir, las funciones para el manejo del periférico se deberán implementar, en una biblioteca, usando las interrupciones de Transmisión y Recepción con sus respectivos buffers.

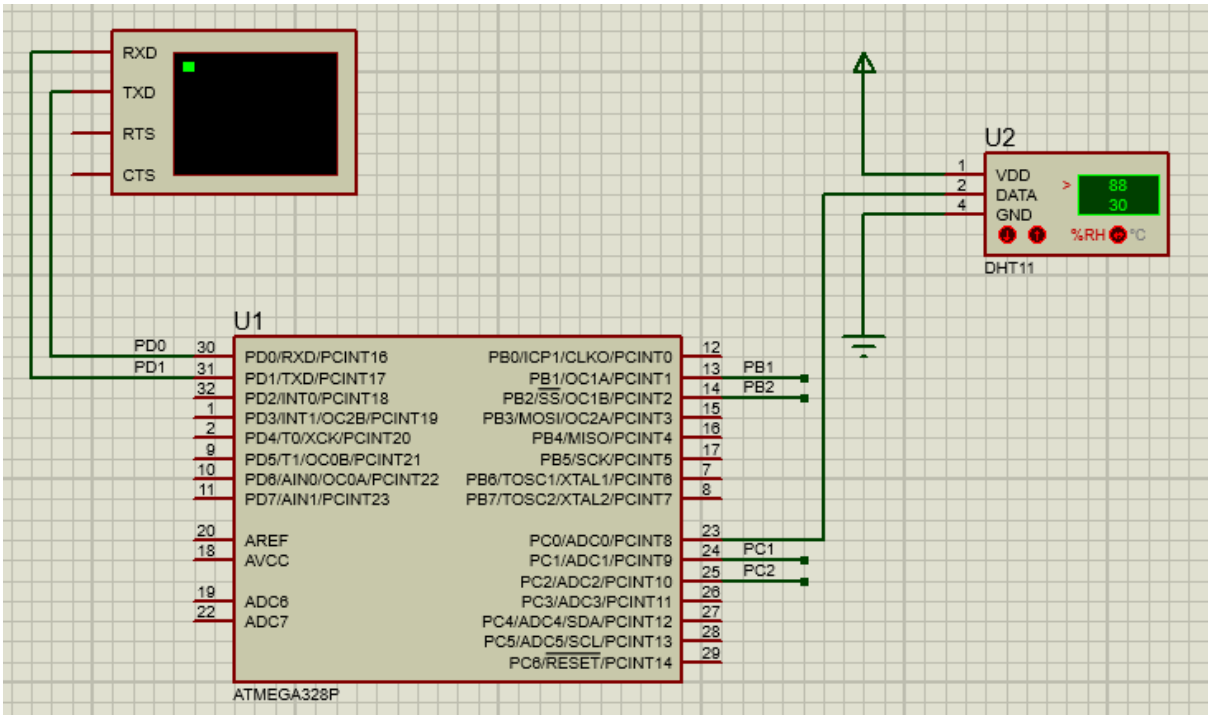

Figura 1.1 Conexión de los componentes en proteus.

1.2 Resolución del problema

Se nos fue facilitada una librería que nos permite interactuar con el periférico UART. La misma contenía un archivo cabecera “serialPort.h” junto con su respectiva implementación en el archivo “serialPort.c”. Las funciones que utilizamos en nuestro programa fueron:

void SerialPort_TX_Interrupt_Enable(void) → Activa las interrupciones del transmisor.
void SerialPort_RX_Interrupt_Enable(void) → Activa las interrupciones del receptor.
void SerialPort_TX_Interrupt_Disable(void) → Desactiva las interrupciones del transmisor.
void SerialPort_Send_Data(char) → Transmite un carácter.

Dentro de estas funciones se encuentra la lógica utilizada para activar o desactivar los bits del Registro B de control y estado del UART, en función de lo requerido.

UCSRB:	RXCIE0	TXCIE0	UDRIE0	RXEN0	TXEN0	UCSZ02	RXB80	TXB80

Figura 1.2 UART Control and Status Register B.

2. Arquitectura: Background/Foreground

2.1. Definición

La arquitectura usada para nuestro programa es la denominada Background/Foreground. Esta arquitectura se basa en interrupciones que pueden ser por eventos o temporizadas que se comunicaran a través de flags globales, ya que al estar interactuando con un periférico externo, no necesariamente la cpu va a estar atendiendo al mismo de manera constante, por lo que es posible seguir realizando otras tareas hasta que el periférico genere una interrupción para ser atendido.

Para esto se implementaron distintas interrupciones las cuales:

- **Interrupción de transmisión:** cuando se habilita esta interrupción se encarga de mostrar en la terminal una cadena de caracteres hasta el “\0”.

- **Interrupción de recepción:** cuando se habilita se encarga de recibir cualquier carácter que se presione en el teclado hasta que el usuario presione enter, una vez

hecho esto, se activa un flag global que activa un evento en el programa principal

- **Interrupción de tiempo(timer):** esta interrupción a diferencia de las dos anteriores no es por evento sino temporizada. Se encarga de activar un flag cada un segundo que activará un evento en el programa principal.

2.2 main

Nuestro programa principal se encarga de inicializar tanto el periférico UART como el reloj, llamando a sus respectivas funciones de inicialización. Una vez hecho esto, se espera a que ocurra un evento, los cuales pueden ser dos:

El primer evento espera a que se reciba una cadena de datos a través del receptor de datos del UART. La interrupción RX será la encargada de activar el flag **FLAG_LINEA_RECIBIDA** cuando se termine de recibir datos. En ese momento iniciará la tarea 1 encargada de analizar la cadena de caracteres recibido.

El segundo evento inicia cuando el usuario ingresa el término “ON” en la terminal lo cual activará el flag **FLAG_ON** y el flag **FLAG_ON_reloj**. Se inicia el contador del reloj y se desactiva el último flag mencionado. Cuando el reloj genere la interrupción activará el flag **FLAG_SENSOR** dando por válida la inicialización de la tarea 2 que se encarga de mostrar en pantalla los datos enviados por el sensor. Esta tarea se hará de manera constante cada un segundo hasta que el usuario ingrese en la terminal la palabra “OFF” o “RST”.

3. Sensor de temperatura y humedad

3.1. Introducción

El sensor tiene tres funciones principales: Pedir los datos del ambiente, obtener una respuesta y enviar los datos a quien se lo pida.

Será necesario implementar una función para cada una de sus funciones para el correcto manejo de los datos.

3.2. Desarrollo

De acuerdo a la hoja de datos del fabricante, cuando comienza la comunicación

entre el microcontrolador y el sensor, se debe envíar una señal en bajo al mismo, que deberá durar al menos 18ms para que pueda ser detectado. Seguido a esto, se debe envíar esta vez, una señal en alto, que debe durar entre 20 y 40 us para que el sensor pueda responder. Todo esto se realiza en la función **void Pedido(void)** que establece la lógica necesaria para la primera parte del proceso.

Cuando el DHT11 detecta el pedido, envía una señal en bajo que dura 80us, luego pasa de bajo a alto y espera el mismo intervalo de tiempo para realizar el envío de los datos. Para ello implementamos una función llamada **uint8_t Respuesta(void),** la cual utiliza una variable auxiliar en forma de timeout, en caso que el DHT11 no haya podido ejecutar correctamente sus tareas. El pseudocódigo de la misma es el siguiente:

uint8_t Respuesta()

Inicializo la variable auxiliar
Mientras la variable auxiliar no exceda el valor y la señal esté en bajo
Hago un delay de 1us
Aumento la variable auxiliar
Si la variable llegó al valor máximo
Retorno 0
(Realizo la misma lógica para la señal en alto)
Retorno 1

Por último, para recibir los datos en el atmega328p realizamos una función llamada **uint8_t Recibir_datos(void)** que, por cada uno de los bits espera a una señal en alto. Cuando llega, si el pulso en alto dura al menos 28us entonces el bit es un lógico alto, sino un lógico bajo. Esta información es guardada en una variable y luego espero a que la señal sea baja. Una vez que todos los bits fueron almacenados retorno el valor.

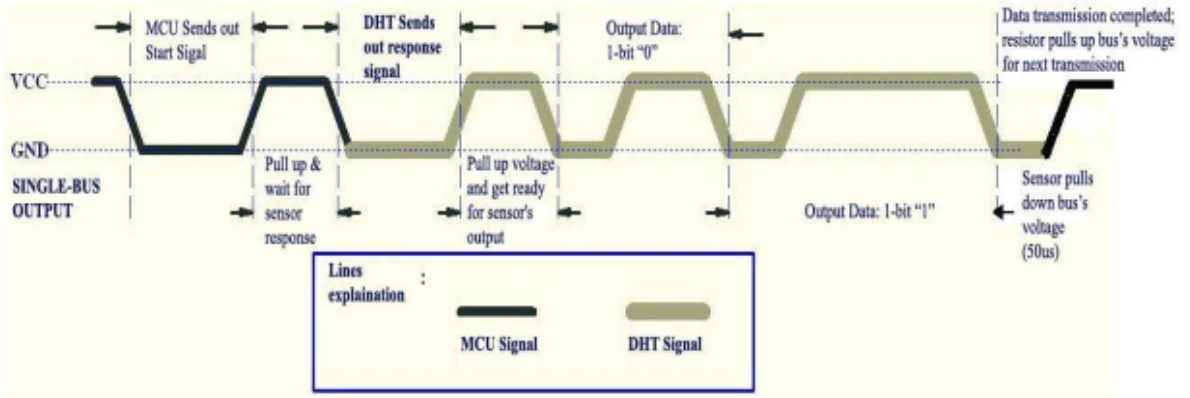

Figura 3.1 Proceso general de comunicación del DHT11.

4. UART(Periférico de comunicación serie asincrónico)

4.1. UART: definición

Es el periférico integrado del MCU que se encarga de realizar la comunicación de forma asincrónica con otro periférico, en nuestro caso, el sensor DHT11. Para esto cuenta con un receptor(RX) y un transmisor (TX) en el cual:

El TX es el encargado de la transmisión de los datos, cuando el flag UDRE está en alto significa que el transmisor está libre y es posible utilizarlo; además si se ingresa un dato mientras se está utilizando, este quedará a la espera por lo tanto actúa como doble buffer.
Una vez que termina de transmitir, vuelve a activar el flag para indicar que está libre.

El RX es el encargado de detectar y decodificar los datos recibidos. Cuando detecta el bit de comienzo este decodifica cada bit hasta encontrar el de parada o “stop”. Una vez hecho esto se transfiere al registro de datos UDR y se activa un flag de “línea recibida” para indicar que completo la recepción de los datos enviados. Además, igual que con TX puede actuar como doble buffer en caso de que se ingresen nuevos datos cuando está en uso.

4.2. UART: desarrollo

Para inicializarlo utilizamos la función **init_periferico()** ubicado en el archivo periférico.c. Esta se encarga de activar las entradas TX y RX como así también sus respectivas interrupciones. Por último, el **UBRR**(Baud Rate Register) que se encarga de la razón de transmisión, se configura a una tasa de transmisión de 9600 bps ya que nuestra transmisión será de 8 bits, por lo cual su valor **UBRR** será de valor 103 de acuerdo con las siguientes ecuaciones

$$FrecuenciaMCU/((UBRR + 1) * 16) = Baud\ rate$$

$$16MHz/(UBRR + 1) * 16) = 9600$$
$$(UBRR + 1) = 1MHz/9600$$
$$UBRR = 103$$

Error cometido = { (9600 - 1MHz/104) * 100 } / 9600 ≈ 0,2%

Una vez hecho esto, se carga el mensaje de bienvenida para mostrar al usuario y se habilita la interrupción de TX para que el transmisor comience a poner los datos en pantalla.

```
void init_periferico(void){

    UCSR0B = (1<<RXEN0) | (1<<TXEN0);
    UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
    UBRR0L = 103;

    mensaje=msg1;

    SerialPort_TX_Interrupt_Enable();
}
```

Figura 4.1 Función init_periferico.

5. Eventos de Tareas

5.1 Introducción

Existen dos tareas a realizar, la primera se encarga de la lógica de los comandos que el usuario va a ingresar mediante la terminal. La segunda, tiene la función de llamar a las funciones del sensor previamente explicadas y mostrar el resultado en pantalla.

5.2 Eventos de Tareas: Tarea 1

Respecto a la primera (**void tarea1(void)**), su contenido es bastante sencillo. Mediante el buffer, obtiene lo que el usuario ingresó y mediante un *strcmp*, compara la cadena de caracteres con alguno de los comandos disponibles. En caso de haber ingresado “ON”, se activa un flag global para que el reloj y luego por consiguiente el sensor, comiencen su ejecución. En cambio al ingresar “RST” u “OFF”, el flag global previamente descrito se desactiva y el reloj se detiene, con la única diferencia de que el primero envía el mensaje de bienvenida otra vez. En caso de haber puesto en la terminal un comando inexistente, se mostrará en pantalla “Comando no válido”.

5.3 Eventos de Tareas: Tarea 2

La segunda (**void tarea2(void)**), es la que se encarga de la lógica del sensor en caso de haber ingresado “ON” previamente. Esta, llama a la función Pedido() explicada anteriormente y a Respuesta(). Si está devolvió un 0, quiere decir que se realizó un timeout y hubo un error en la respuesta del sensor, por lo que mostrará por pantalla “Error respuesta”. Es necesario remarcar, que tuvimos un problema cuando se ingresa por primera vez a pedir datos al sensor ya que devolvía siempre como resultado un “Error respuesta”, por lo cual se agrego una variable uint8_t que verifique que la primera vez que se ingrese a el sensor, el resultado obtenido no se muestre en pantalla.

En caso contrario, si se dió una contestación, recibo los datos en variables según lo explica la datasheet del dht11. Es decir, guardo los 8 bits de la parte entera y decimal y el checksum.
A continuación, me aseguro que la suma de todos los valores no difieren del checksum. Si son diferentes, transmito por la terminal “Error datos”, de no ser así, concateno los valores obtenidos y los muestro por pantalla en la terminal.

6. Conclusiones

6.1 Problemas encontrados

La resolución final varía un poco de la propuesta inicial ya que tuvimos un problema con el sensor que necesitaba más tiempo para su correcto funcionamiento, por lo cual fue necesario que la información mostrada en pantalla se visualice en 1,2 segundos en vez de 1 segundo.

Por otra parte tuvimos un error con la primera respuesta del sensor ya que siempre se quedaba bloqueado en el módulo **Respuesta(),** encargado de generar la señal del sensor; como no pudimos resolver esto, nuestra solución fue que la primera respuesta que enviaba el sensor no se muestre en pantalla.

6.2 Validación

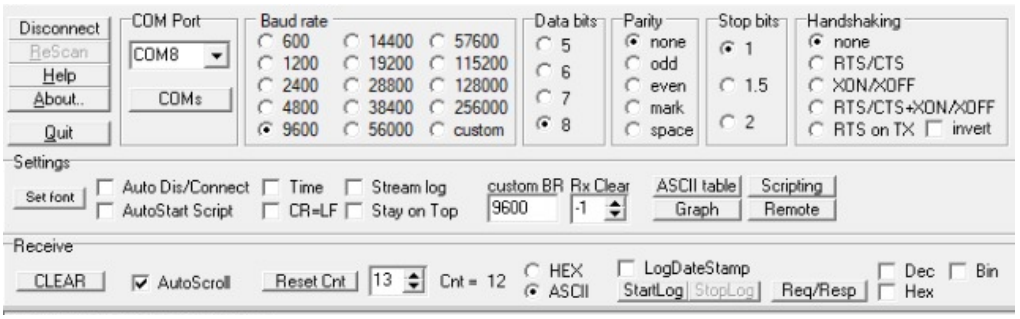

Figura 6.1 Configuración de la terminal.

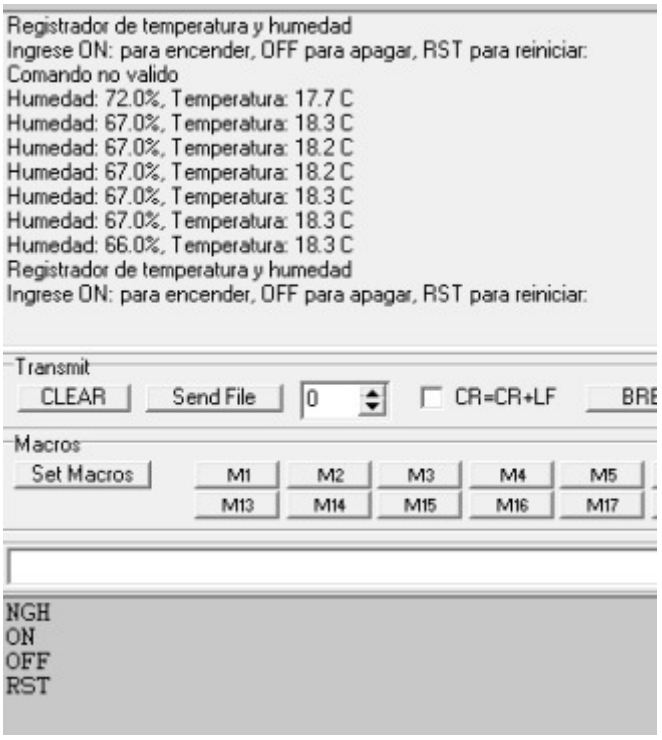

Figura 6.2 Verificación en la terminal.

7. Anexo

7.1 Archivos cabecera

main.h

```
#ifndef MAIN_H
#define MAIN_H

#include "clock.h"
#include "periferico.h"
#include <avr/io.h>
#include <avr/interrupt.h>
uint8_t FLAG_SENSOR;
uint8_t FLAG_ON;
uint8_t FLAG_LINEA_RECIBIDA;
uint8_t FLAG_ON_reloj;

#endif
```

clock.h

```
#ifndef CLOCK_H
#define CLOCK_H

#include <avr/interrupt.h>
#include <avr/io.h>
#include "main.h"

void init_reloj(void);
void ON_reloj(void);
void stop_reloj(void);

#endif
```

periferico.h

```
#ifndef PERIFERICO_H
#define PERIFERICO_H
```

```
#include <avr/io.h>
#include <string.h>
#include <avr/interrupt.h>
#include "main.h"
#include "serialPort.h"
#include "sensor.h"
#include "clock.h"

void init_periferico(void);
void tarea1(void);
void tarea2(void);

#endif
```

sensor.h

```
#ifndef SENSOR_H
#define SENSOR_H
#define DHT11_PIN 0
#define F_CPU 16000000UL

#include <avr/io.h>
#include <stdlib.h>
#include <stdio.h>
#include <util/delay.h>

uint8_t I_RH,D_RH,I_Temp,D_Temp,CheckSum;
uint8_t count;

void Pedido(void);
uint8_t Respuesta(void);
uint8_t Recibir_datos(void);

#endif
```

serialPort.h

```
#ifndef SERIALPORT_H_
	#define SERIALPORT_H_

	// ------------------- Includes ----------------------------

	// Archivo de cabecera del Microcontrolador
	#include <avr/io.h>
```

```
	// Interrupciones del Microcontrolador
	#include <avr/interrupt.h>

	// -------- Prototipos de funciones Publicas --------------

	// Inicializacion de Puerto Serie
	void SerialPort_Init(uint8_t);

	// Inicializacion de Transmisor
	void SerialPort_TX_Enable(void);
	void SerialPort_TX_Interrupt_Enable(void);
	void SerialPort_TX_Interrupt_Disable(void);

	// Inicializacion de Receptor
	void SerialPort_RX_Enable(void);
	void SerialPort_RX_Interrupt_Enable(void);

	// Transmision
	void SerialPort_Wait_For_TX_Buffer_Free(void);  // Pooling -
Bloqueante hasta que termine de transmitir.
	void SerialPort_Send_Data(char);
	void SerialPort_Send_String(char *);
	void SerialPort_Send_uint8_t(uint8_t);
	void SerialPort_send_int16_t(int val,unsigned int field_length);
//This function writes a integer type value to UART
                                          //-32768 y 32767

	// Recepcion
	void SerialPort_Wait_Until_New_Data(void);      // Pooling -
Bloqueante, puede durar indefinidamente!
	char SerialPort_Recive_Data(void);

	//Driver P.C.
	void SerialPort_Write_Char_To_Buffer ( char Data );
	void SerialPort_Write_String_To_Buffer( char *  STR_PTR );
	void SerialPort_Send_Char (char dato);
	void SerialPort_Update(void);
	char SerialPort_Get_Char_From_Buffer (char * ch);
	char SerialPort_Get_String_From_Buffer (char * string);
	char SerialPort_Receive_data (char * dato);
#endif /* SERIALPORT_H_ */
```

7.2 Archivos ".c"

main.c

```
#include "main.h"

 int main(void)
 {
	init_periferico();
	init_reloj();//Inicia el reloj
	sei();
	while(1)
	{
		if(FLAG_LINEA_RECIBIDA){
			tarea1();
			FLAG_LINEA_RECIBIDA=0;
		}
		if(FLAG_ON){
			if(FLAG_ON_reloj){
				ON_reloj();
				FLAG_ON_reloj=0;
			}
			if(FLAG_SENSOR){
				FLAG_SENSOR=0;
				cli();
				tarea2();
				sei();
			}
		}
	}
	return 0;
 }
```

clock.c

```
#include "clock.h"

static uint8_t counter = 0;

void init_reloj(void)
{
	DDRC = 0xFF;
	TCCR0A = (1 << WGM01);
	OCR0A = 156; //0.01 sec
	TCCR0B = (1 << CS02) | (1 << CS00); // prescaler 1024
}
```

```
void ON_reloj(void)
{
	TIMSK0 = (1 << OCIE0A);
	counter=0;
}

void stop_reloj(void)
{
	TIMSK0 = (0 << OCIE0A);
}

ISR(TIMER0_COMPA_vect)
{
	counter++;
	if(counter > 120){
		FLAG_SENSOR=1;
		counter = 0;
	}
}
```

periferico.c

```
#include "periferico.h"

//mensajes de bienvenida y despedida
static char msg1[] = "Registrador de temperatura y humedad\rIngrese ON:
para encender, OFF para apagar, RST para reiniciar: \r";
static char msg2[] = "Comando no valido\r";
static char data[5];
static char RX_Buffer[32];
static char * mensaje;

void init_periferico(void){

	UCSR0B = (1<<RXEN0) | (1<<TXEN0);
	UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
	UBRR0L = 103;

	mensaje=msg1;
	SerialPort_TX_Interrupt_Enable();
}

void tarea1(){
	if(strcmp(RX_Buffer,"ON")==0){
```

```
			FLAG_ON=1;
			FLAG_ON_reloj=1;
		}
		else if(strcmp(RX_Buffer,"RST")==0){
			mensaje=msg1;
			FLAG_ON=0;
			stop_reloj();
			SerialPort_TX_Interrupt_Enable();
		}
		else if(strcmp(RX_Buffer,"OFF")==0){
			FLAG_ON=0;
			stop_reloj();
			} else{
			mensaje=msg2;
			SerialPort_TX_Interrupt_Enable();
		}

		//SerialPort_Wait_For_TX_Buffer_Free();		// Espero a que el
canal de transmisión este libre (bloqueante)
}

void tarea2(){

		Pedido();

		if(Respuesta()==0){
			if(count>1){
				mensaje="Error respuesta\r";
				SerialPort_TX_Interrupt_Enable();
			}
		}


		else{
		I_RH=Recibir_datos();
		D_RH=Recibir_datos();
		I_Temp=Recibir_datos();
		D_Temp=Recibir_datos();
		CheckSum=Recibir_datos();


		if ((I_RH + D_RH + I_Temp + D_Temp) != CheckSum)
		{
			mensaje="Error datos \r";
		}
```

```
	else
	{
		char Humedad[60]="Humedad: ";
		strcat(Humedad,itoa(I_RH,data,10));
		strcat(Humedad,".");
		strcat(Humedad,itoa(D_RH,data,10));
		strcat(Humedad,"%, ");

		char Temperatura[30]="Temperatura: ";
		strcat(Temperatura,itoa(I_Temp,data,10));
		strcat(Temperatura,".");
		strcat(Temperatura,itoa(D_Temp,data,10));
		strcat(Temperatura," C\r");
		strcat(Humedad,Temperatura);
		mensaje=Humedad;
		SerialPort_TX_Interrupt_Enable();
	}

	}

}

ISR(USART_RX_vect){
	static volatile char rx_data=0;
	static short int index=0;
	rx_data=UDR0;
	if(rx_data!='\r')
	RX_Buffer[index++]=rx_data;
	else{
		RX_Buffer[index]='\0';
		index=0;
		FLAG_LINEA_RECIBIDA=1;
	}
}

ISR (USART_UDRE_vect) //interrupción de transmisión
{
	static short int index=0;
	if (mensaje[index] != '\0')
	{
		SerialPort_Send_Data(mensaje[index]);
		index++;
	}
	else
	{
		index=0;
```

```
            SerialPort_TX_Interrupt_Disable();
            SerialPort_RX_Interrupt_Enable();
      }
}
```

sensor.c

```
#include "sensor.h"

static uint8_t c=0;
uint8_t count=0;

void Pedido()
{
      DDRC |= (1<<DHT11_PIN); //como salida
      PORTC &= ~(1<<DHT11_PIN);
      _delay_ms(20);
      PORTC |= (1<<DHT11_PIN);
      _delay_us(40);
}

uint8_t Respuesta()
{
      uint8_t aux=0;
      DDRC &= ~(1<<DHT11_PIN);
      PORTC |= (1<<DHT11_PIN); //pull up

      while(((aux<90)&&((PINC & (1<<DHT11_PIN))==0))){
            _delay_us(1);
            aux++;
      }
      if(aux==90){
            count++;
            return 0;
      }
      aux=0;
      while((aux<90)&&(PINC & (1<<DHT11_PIN))){
            _delay_us(1);
            aux++;

      }
      if(aux==90){
            count++;
            return 0;
      }
```

```
	return 1;
}

uint8_t Recibir_datos()
{
	for (int q=0; q<8; q++)
	{
		while((PINC & (1<<DHT11_PIN)) == 0);  //Se fija si es alto o
bajo el pulso
		_delay_us(30);
		if(PINC & (1<<DHT11_PIN))// Si el pulso en alto dura al
menos 30ms
		c = (c<<1)|(0x01);    // Entonces es logico alto
		else            // Sino logico bajo
		c = (c<<1);
		while(PINC & (1<<DHT11_PIN));
	}
	return c;
}
```

serialPort.c

```
#include "serialPort.h"


#define TX_BUFFER_LENGTH 32
#define RX_BUFFER_LENGTH 32

volatile static unsigned char TXindice_lectura=0, TXindice_escritura=0;
volatile static unsigned char RXindice_lectura=0, RXindice_escritura=0;

static char TX_Buffer [TX_BUFFER_LENGTH];
static char RX_Buffer [RX_BUFFER_LENGTH];

// Inicialización de Puerto Serie
void SerialPort_Init(uint8_t config){
	// config = 0x67 ==> Configuro UART 9600bps, 8 bit data, 1 stop @
F_CPU = 16MHz.
	// config = 0x33 ==> Configuro UART 9600bps, 8 bit data, 1 stop @
F_CPU = 8MHz.
	// config = 0x25 ==> Configuro UART 9600bps, 8 bit data, 1 stop @
F_CPU = 4MHz.
	UCSR0B = 0;
	UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
	//UBRR0H = (unsigned char)(config>>8);
```

```
	UBRR0L = (unsigned char)config;
}


// Inicialización de Transmisor

void SerialPort_TX_Enable(void){
	UCSR0B |= (1<<TXEN0);
}

void SerialPort_TX_Interrupt_Enable(void){
	UCSR0B |= (1<<UDRIE0);
}

void SerialPort_TX_Interrupt_Disable(void)
{
	UCSR0B &=~(1<<UDRIE0);
}


// Inicialización de Receptor

void SerialPort_RX_Enable(void){
	UCSR0B |= (1<<RXEN0);
}

void SerialPort_RX_Interrupt_Enable(void){
	UCSR0B |= (1<<RXCIE0);
}


// Transmisión

// Espera hasta que el buffer de TX este libre.
void SerialPort_Wait_For_TX_Buffer_Free(void){
	// Pooling - Bloqueante hasta que termine de transmitir.
	while(!(UCSR0A & (1<<UDRE0)));
}

void SerialPort_Send_Data(char data){
	UDR0 = data;
}

void SerialPort_Send_String(char * msg){ //msg -> "Hola como andan hoy?"
20 ASCII+findecadena, tardo=20ms
	uint8_t i = 0;
```

```
	//'\0' = 0x00
	while(msg[i]){ // *(msg+i)
		SerialPort_Wait_For_TX_Buffer_Free(); //9600bps formato 8N1,
10bits, 10.Tbit=10/9600=1ms
		SerialPort_Send_Data(msg[i]);
		i++;
	}
}


// Recepción

// Espera hasta que el buffer de RX este completo.
void SerialPort_Wait_Until_New_Data(void){
	// Pooling - Bloqueante, puede durar indefinidamente!
	while(!(UCSR0A & (1<<RXC0)));
}


char SerialPort_Recive_Data(void){
	return UDR0;
}


void SerialPort_Send_uint8_t(uint8_t num){

	SerialPort_Wait_For_TX_Buffer_Free();
	SerialPort_Send_Data('0'+num/100);

	num-=100;

	SerialPort_Wait_For_TX_Buffer_Free();
	SerialPort_Send_Data('0'+num/10);

	SerialPort_Wait_For_TX_Buffer_Free();
	SerialPort_Send_Data('0'+ num%10);
}

/*************************************************************
	This function writes a integer type value to UART
	Arguments:
	1)int val   : Value to print
	2)unsigned int field_length :total length of field in which the
value is printed
	must be between 1-5 if it is -1 the field length is no of digits
in the val
```

```
*****************************************************************/
void SerialPort_send_int16_t(int val,unsigned int field_length)
{
	char str[5]={0,0,0,0,0};
	int i=4,j=0;
	while(val)
	{
	str[i]=val%10;
	val=val/10;
	i--;
	}
	if(field_length==-1)
		while(str[j]==0) j++;
	else
		j=5-field_length;

	if(val<0) {
		SerialPort_Wait_For_TX_Buffer_Free();
		SerialPort_Send_Data('-');
		}
	for(i=j;i<5;i++)
	{
	SerialPort_Wait_For_TX_Buffer_Free();
	SerialPort_Send_Data('0'+str[i]);
	}
}
//*********


void SerialPort_Write_Char_To_Buffer ( char Data )
{
	// Write to the buffer *only* if there is space
	if (TXindice_escritura < TX_BUFFER_LENGTH){
		TX_Buffer[TXindice_escritura] = Data;
		TXindice_escritura++;
	}
	else {
		// Write buffer is full
		//Error_code = ERROR_UART_FULL_BUFF;
	}
}

void SerialPort_Write_String_To_Buffer( char *  STR_PTR )
{
	unsigned char i = 0;
	while ( STR_PTR [ i ] != '\0')
```

```
	{
		SerialPort_Write_Char_To_Buffer ( STR_PTR [ i ] );
		i++;
	}
}

void SerialPort_Send_Char (char dato)
{
	SerialPort_Wait_For_TX_Buffer_Free(); // Espero a que el canal de
transmisión este libre (bloqueante)
	SerialPort_Send_Data(dato);
}

void SerialPort_Update(void)
{
	static char key;

	if ( UCSR0A & (1<<RXC0) ) { // Byte recibido. Escribir byte en
buffer de entrada
		if (RXindice_escritura < RX_BUFFER_LENGTH) {
			RX_Buffer [RXindice_escritura] =UDR0; // Guardar dato
en buffer
			RXindice_escritura++; // Inc sin desbordar buffer
		}
		//else
		//Error_code = ERROR_UART_FULL_BUFF;
	}
	// Hay byte en el buffer Tx para transmitir?
	if (TXindice_lectura < TXindice_escritura){
		SerialPort_Send_Char ( TX_Buffer [TXindice_lectura] );
		TXindice_lectura++;
	}
	else {// No hay datos disponibles para enviar
		TXindice_lectura = 0;
		TXindice_escritura = 0;
	}
}

char SerialPort_Get_Char_From_Buffer (char * ch)
{
	// Hay nuevo dato en el buffer?
	if (RXindice_lectura < RXindice_escritura){
		*ch = RX_Buffer [RXindice_lectura];
		RXindice_lectura++;
		return 1; // Hay nuevo dato
	}
```

```
	else {
		RXindice_lectura=0;
		RXindice_escritura=0;
		return 0; // No Hay
	}
}

char SerialPort_Get_String_From_Buffer (char * string)
{
	char rxchar=0;

	do{
		if(SerialPort_Get_Char_From_Buffer (&rxchar)){
			*string=rxchar;
			string++;
		}
		else{
			rxchar='\n'; //empty string
		}
	}while(rxchar!='\n');
	*string='\0'; //End of String
	return 1;
}

char SerialPort_Receive_data (char * dato)
{
	if ( (UCSR0A & (1<<RXC0))==1) {
		*dato=UDR0;
		return 1;
	}
	return 0; //no data
}
```