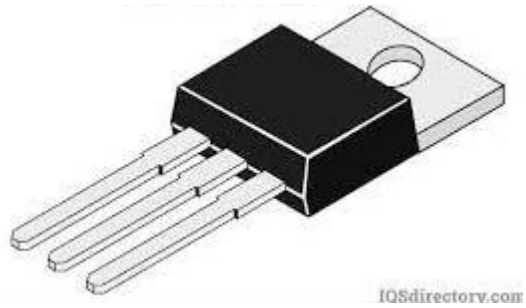
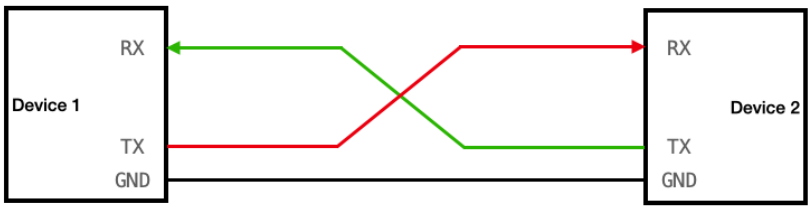
4 DE JULIO DE 2022





BLANCO VALENTIN NICOLAS, PALADINO GABRIEL AGUSTIN
CIRCUITOS DIGITALES Y MICROCONTROLADORES

Interpretación:

Se debería realizar un proyecto en c que a partir de la escritura de comandos en una consola se muestre un menú, se realice la lectura de temperaturas y humedades dada por un sensor cada 1 segundo o se frene dicha lectura.

Para la realización de este se contó con un periférico de medición de temperatura y humedad de nombre DHT11 del cual se leían estos datos para luego ser impresos por la terminal. Además, se utilizó el USART que permitió la comunicación serie asincrónica con la terminal. Por otro lado, los comandos que se podían utilizar sobre la terminal eran ON<Enter> para encender el registrador de humedad y temperatura, OFF<Enter> para apagar el mismo y RST<Enter> para detener el registrador, volver al estado inicial y mostrar el menú. En caso de que el usuario ingrese otro comando se debía desplegar un mensaje de Comando no válido y deberá seguir realizando la tarea previa al comando mal ingresado.

Resolución de problema:

Comunicación serie

Para realizar la comunicación entre el MCU y la terminal de comandos hacemos uso del periférico de comunicación serie que trae incorporado el MCU, el mismo nos permite enviar y recibir datos de manera asincrónica y sincrónica gracias al generador de reloj que trae incorporado. En nuestro caso para resolver el problema utilizamos una conexión full dúplex asincrónica con una tasa de transmisión de 9600 bps. La trama de datos utilizada para la comunicación es 8N1, es decir 8 bits de datos, 1 bit de stop y ningún bit de paridad.

Para configurar el reloj a la tasa de baudios deseada se debe configurar el registro UBRR0L ingresando el numero obtenido al utilizar la siguiente formula extraída del datasheet.

Table 19-1. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRRn Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(\boldsymbol{UBRR}n + 1)}$	$\boldsymbol{UBRR}n = \frac{f_{OSC}}{2BAUD} - 1$

$$UBRR0 = \frac{16MHz}{16*9600} - 1 = 103.1666 \rightarrow se\ redondea\ a\ 103$$

Al redondear el numero se comete un error en el reloj, el mismo se obtiene de la siguiente formula que también se extrajo del datasheet.

$$\text{Error}[\%] = \left(\frac{\text{BaudRate}_{\text{Closest Match}}}{\text{BaudRate}} - 1\right) \bullet 100\%$$

$$Error = \left(\frac{\frac{16MHz}{16 * (103 + 1)}}{9600} - 1\right) * 100 = 0.16\%$$

Para que no haya problemas en la transmisión el máximo error entre relojes debe ser menor al 5%, por lo que no debería haber problemas en la comunicación.

Una vez que configuramos los baudios, hay que indicarle que tipo de trama se va a utilizar, esto lo hicimos a través del registro UCSRC y siguiendo las siguientes imágenes.

UCSRC:	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0

Table 19-7. UCSZn Bits Settings

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Como se puede ver, para enviar datos de 8 bits se debe poner UCSZ01 y UCSZ01 en alto. Luego para no utilizar bit de paridad se debe poner UMP01 y UMP00 en bajo. Y por último para usar un solo bit de stop se debe poner USBS0 en bajo.
Como vamos a utilizar comunicación asincrónica debemos poner en bajo UMSEL01 y UMSEL00.
Por último, UCPOL0 no lo configuramos porque es únicamente para modo sincrónico.
En la siguiente imagen se puede observar cómo quedaría configurado todo el registro UCSRC para lograr el tipo de comunicación deseada.

UCSRC:	UMSEL01	UMSEL00	UPM01	UPM00	USBS0	UCSZ01	UCSZ00	UCPOL0
	0	0	0	0	0	1	1	0

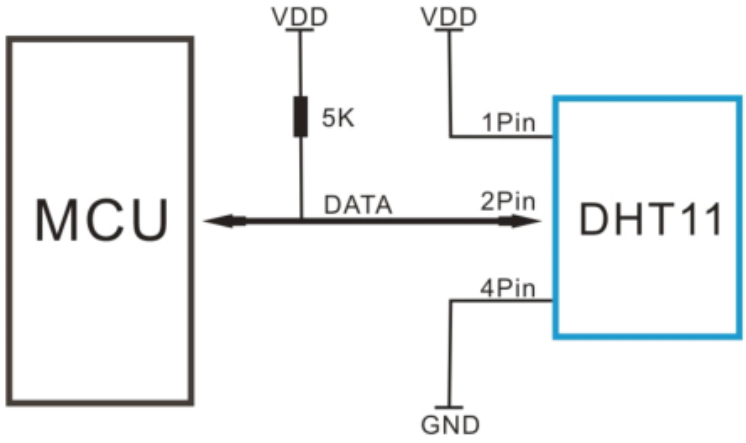
Sensor DHT-11

El dht_11 es un sensor de humedad y temperatura que consta de 4 pines, dos de ellos son de alimentación (se alimenta con 5V), uno de datos y el otro va al aire. El mismo devuelve los datos de manera serie en 40 bits (8 bits de temperatura, 8 bits de temperatura decimal, 8 bits de humedad, 8 bits de humedad decimal y otros 8 bits de sumcheck). Dependiendo del fabricante, el sensor puede enviar los valores decimales o no.

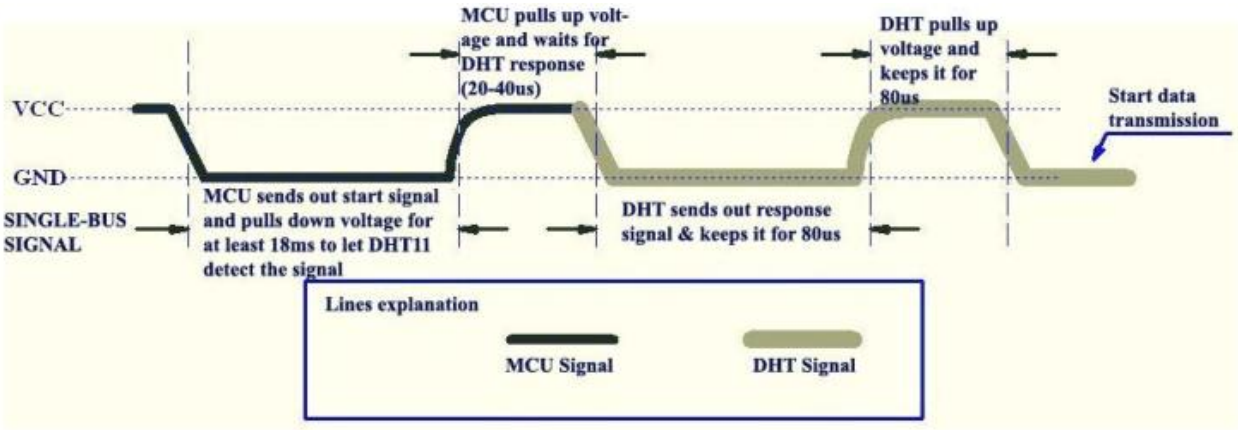
A continuación, se muestran las características principales del mismo como el rango de medida y la precisión.

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 ℃	±5%RH	±2℃	1	4 Pin Single Row

En nuestro caso como pin de datos utilizamos en pin 0 del puerto C (PC0). Para poder usarlo se debe conectar una resistencia de pull-up en la línea de datos. La conexión quedaría de la siguiente manera según el datasheet.

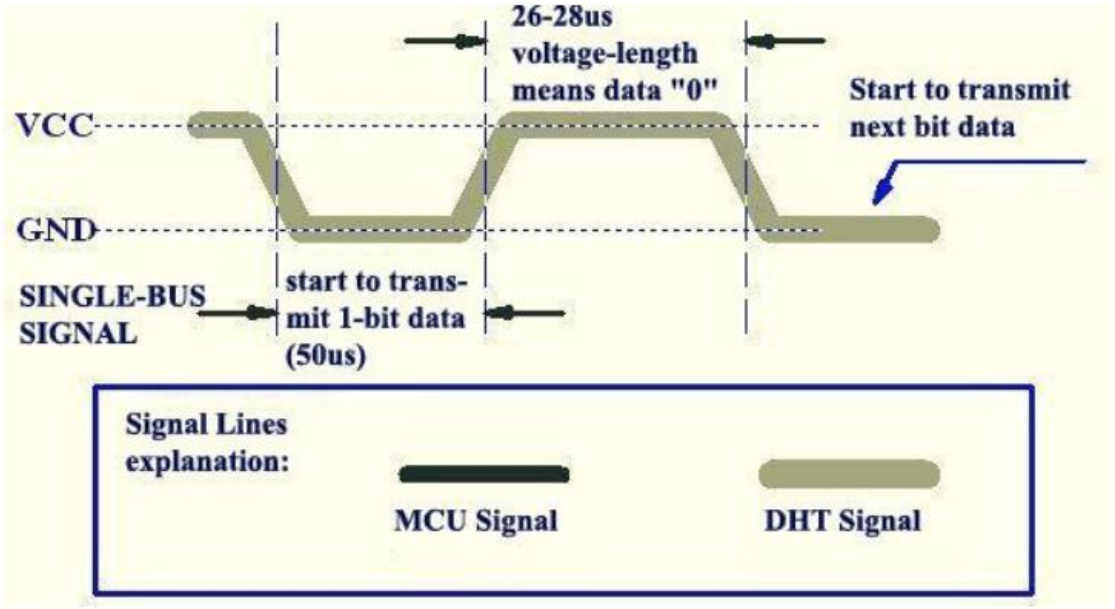


Para poder programarlo nos guiamos por los diagramas de tiempo que se muestran en la hoja de dato.



En la imagen se puede ver que primero debemos generar la señal de start para que el sensor nos empiece a enviar los datos. Para esto se configura como salida el pin de datos y se pone en cero el mismo por al menos 18ms, luego se debe configurar como entrada pull-up y se espera

entre 20us y 40us a que el sensor responda. El mismo responde bajando la señal durante 80us y luego la pone en alto durante otros 80us indicando que comenzara la transmisión de datos.



El sensor transmite bit a bit, para cada bit que envía genera una señal en bajo de 50us y luego la pone en alto por un determinado tiempo, el tiempo que esta en alto es el que determina si es un uno o un cero, siendo entre 26us y 28us un cero y 7us un uno.

A continuación, se muestra un pseudocódigo de como quedaría la rutina implementada.

```
Se configura PC0 como salida
Se pone en bajo 18ms
Se configura PC0 como entrada pull-up
Se espera entre 20us y 40us
El sensor pone en bajo la señal por 80us
El sensor pone en alto la señal por 80us
Mientras no lleguemos a 40 bits
    El sensor anuncia el siguiente bit poniendo en bajo la señal por 50us
    El sensor pone en alto la señal por un tiempo determinado
    Chequeo si el tiempo que estuvo en alto es menor que 28us
        Caso correcto envió un cero
        Caso contrario envió un uno
    Se almacena el bit en un arreglo
Fin mientras
Se codifican los bits recibidos y almacenados
```

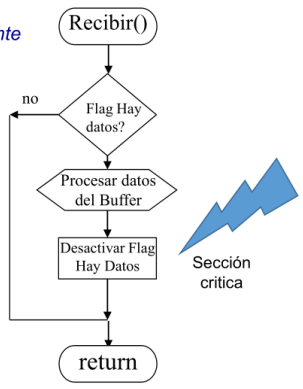
Productor Consumidor y Librería SerialPort

Como la transmisión y recepción de datos se realiza a distintas velocidades, estas se deben hacer en segundo plano, por lo que se utilizo el modelo productor consumidor. Al mismo lo implementamos mediante dos buffers globales, uno llamado Buffer que es el que contiene los datos a transmitir y otro llamado comando que es donde se va almacenando 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 y no se "cuelgue" el programa principal. Para configurar el periférico de comunicación serie utilizamos la librería brindada por la catedra llamada "SerialPort.h". Para configurar el USART como se explicó anteriormente la librería incluye una función llamada SerialPort_Init(*uint8_t*) que recibe como parámetro el valor en hexadecimal UBRR para configurar los baudios. Previamente habíamos calculado que para 9600 baudios UBRR debía ser 103, si lo convertimos a hexadecimal equivale a 67. Luego la misma función se encarga de configurar los demás bits de los registros para usar tramas 8N1. Una vez que termina de inicializar se debe activar el transmisor y receptor del puerto serie, para esto existen las funciones SerialPort_TX_Enable() que pone en uno el bit TXEN0 para activar la transmisión y SerialPort_RX_Enable() que pone en alto el bit RXEN0 para activar la recepción. Prosiguiendo activamos la interrupción del receptor, de modo tal que cuando haya datos nuevos en el registro UDR0 se genere una interrupción para almacenar el dato recibido en el buffer de recepción. Para activar dicha interrupción hacemos uso de la función SerialPort_RX_Interrupt_Enable() que básicamente lo que hace es poner en uno el bit RXCIE0 del registro UCSR0B. Una vez que se quiera transmitir un dato debemos activar la interrupción del transmisor para que el mismo interrumpa cada vez que este libre para enviar un dato, al igual que RX tiene su propia función para activarla llamada SerialPort_TX_Interrupt_Enable(). La librería además tiene la función SerialPort_TX_Interrupt_Disable() que lo que hace es desactivar la interrupción del transmisor, ya que es de suma importancia desactivar la misma cuando se termina de enviar los datos debido a que si no se hace el transmisor queda interrumpiendo constantemente al programa principal.

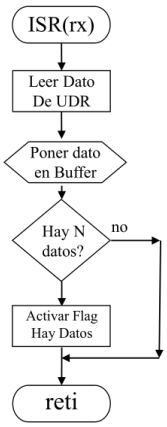
Backgroun Foreground

Para resolver el problema utilizamos la arquitectura Backgroun Foreground para determinar que tarea deberá realizar el MCU. Para ello cada vez que se deba hacer una tarea especifica se debe disparar un flag que le avise al programa principal de que ocurrió un evento. En nuestro caso usamos dos flag que cada uno indica un evento distinto. Uno de ellos (hayComando) se activa cuando se terminó de ingresar un comando por la terminal serie (se termina de ingresar un comando cuando ingresa la tecla enter), de modo tal que cuando se active el mismo el programa principal procese el comando presente en el buffer de recepción y realice las acciones correspondientes. El otro flag (hayDato) indica que se debe transmitir todo lo que hay en el buffer de salida, mientras no se terminen de enviar dichos datos el flag seguirá en uno. Todo lo comentado anteriormente se puede observar en los siguientes diagramas.

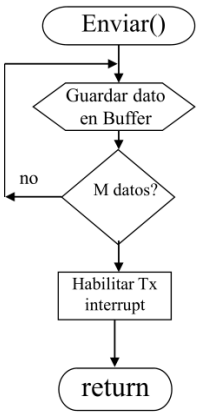
El Consumidor y el Productor se sincronizan mediante un flag!!!



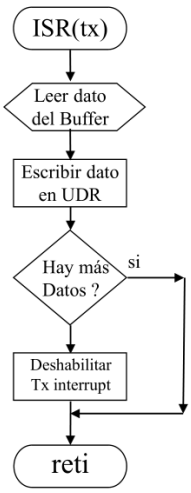
Tarea en Background- consumidor



Tarea en Foreground- productor



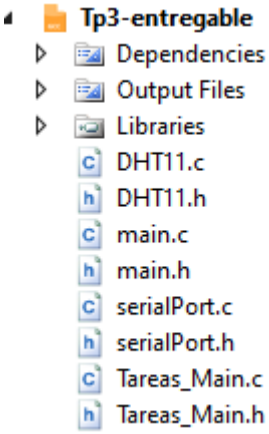
Tarea en Background - productor



Tarea en Foreground - consumidor

Modularización

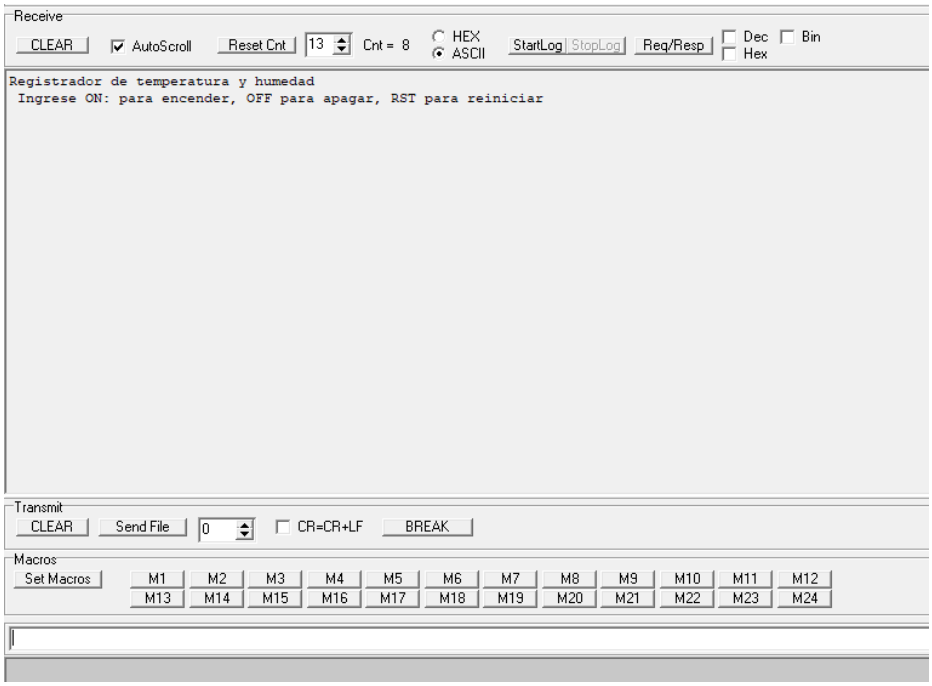
A continuación se muestra una imagen donde se pueden ver los distintos archivos que componen la solución del problema.



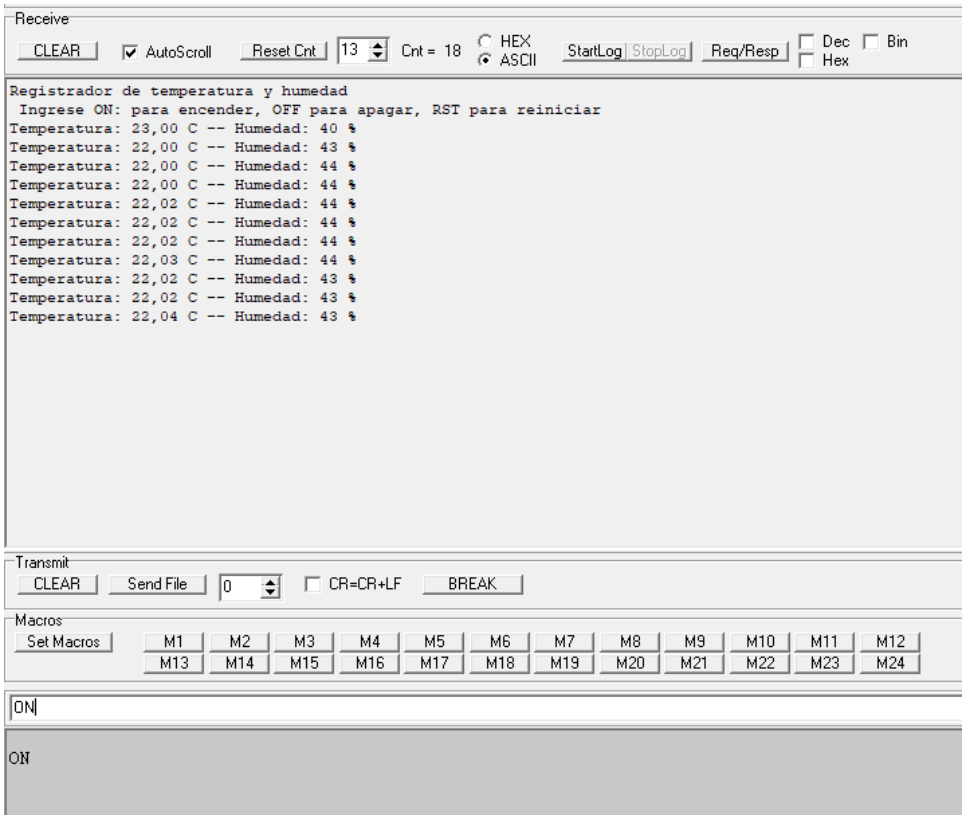
- DHT11 se encarga de hacer funcionar el sensor, para ello posee un método llamado dht11_iniciaryleerdato(int *, int *,int *) que devuelve los datos medidos a través de los parámetros.
- serialPort es la librería brindada por la catedra para configurar y utilizar el puerto serie del MCU.
- Tareas_Main contiene dos métodos, uno para llenar el buffer de salida con los datos brindados por el sensor llamado Tareas_Main_Tarea1(), y el otro método se encarga de procesar el comando presente en el buffer de entrada llamado Tareas_Main_leercomando(char *s) que recibe como parámetro el buffer de entrada.

Validación

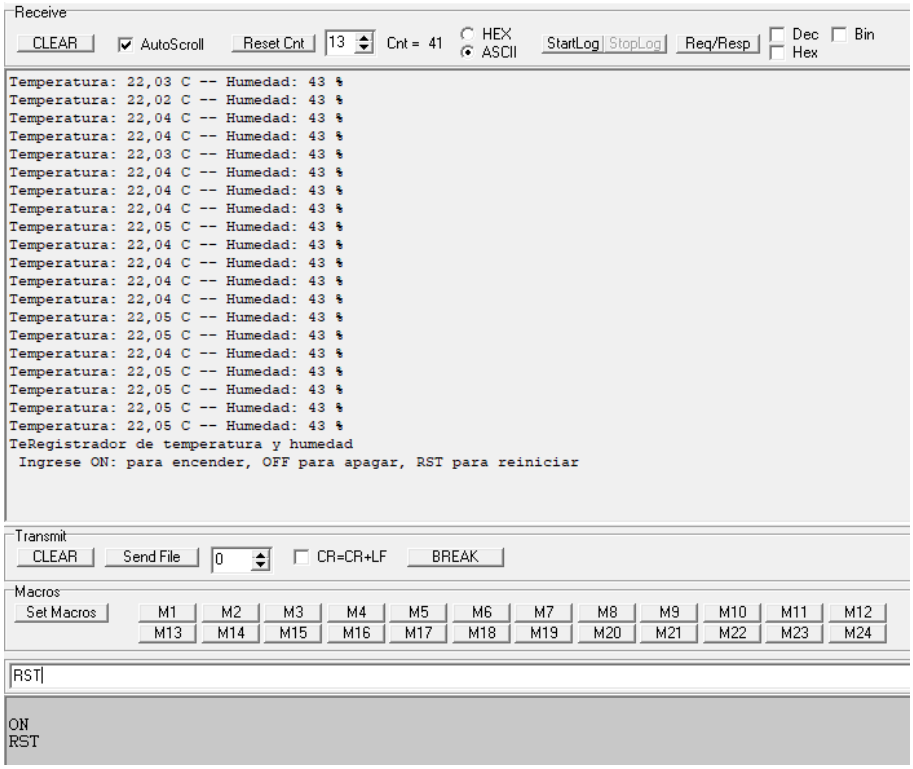
Como se puede ver en la siguiente imagen apenas se conecta el MCU con la terminal se muestra el menú de opciones y queda a la espera de recibir un comando.



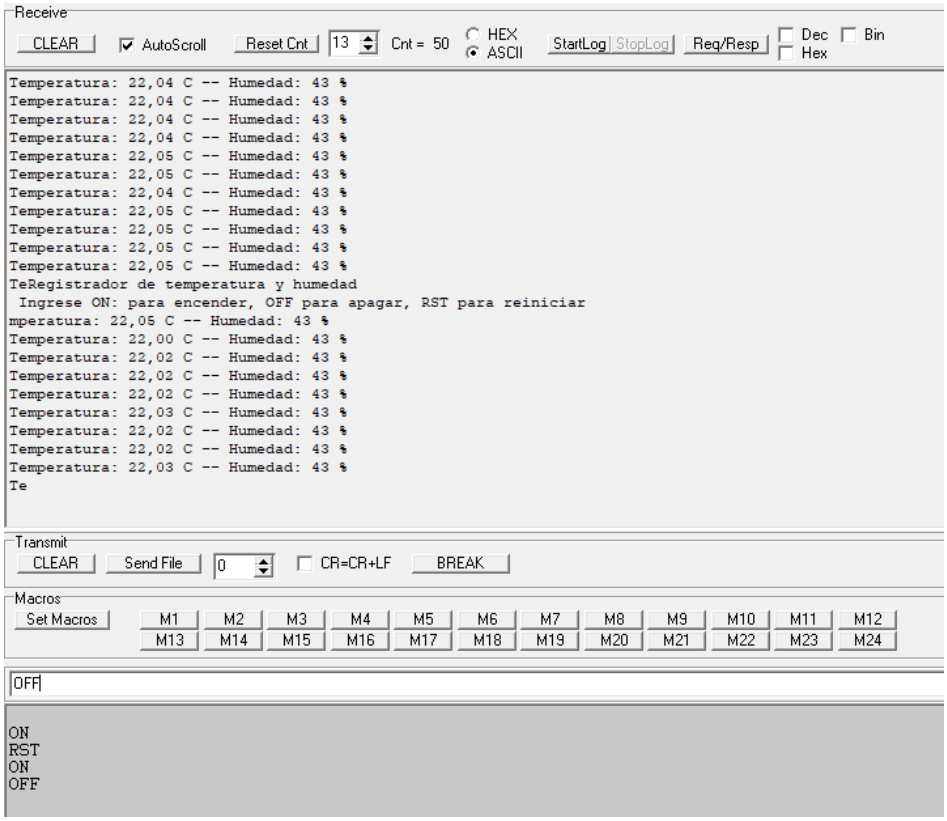
Como primera prueba ingresamos el comando ON para que el MCU nos empiece a enviar los datos leídos por el sensor cada 1 segundo. A continuación, se puede observar el correcto funcionamiento del comando en cuestión.



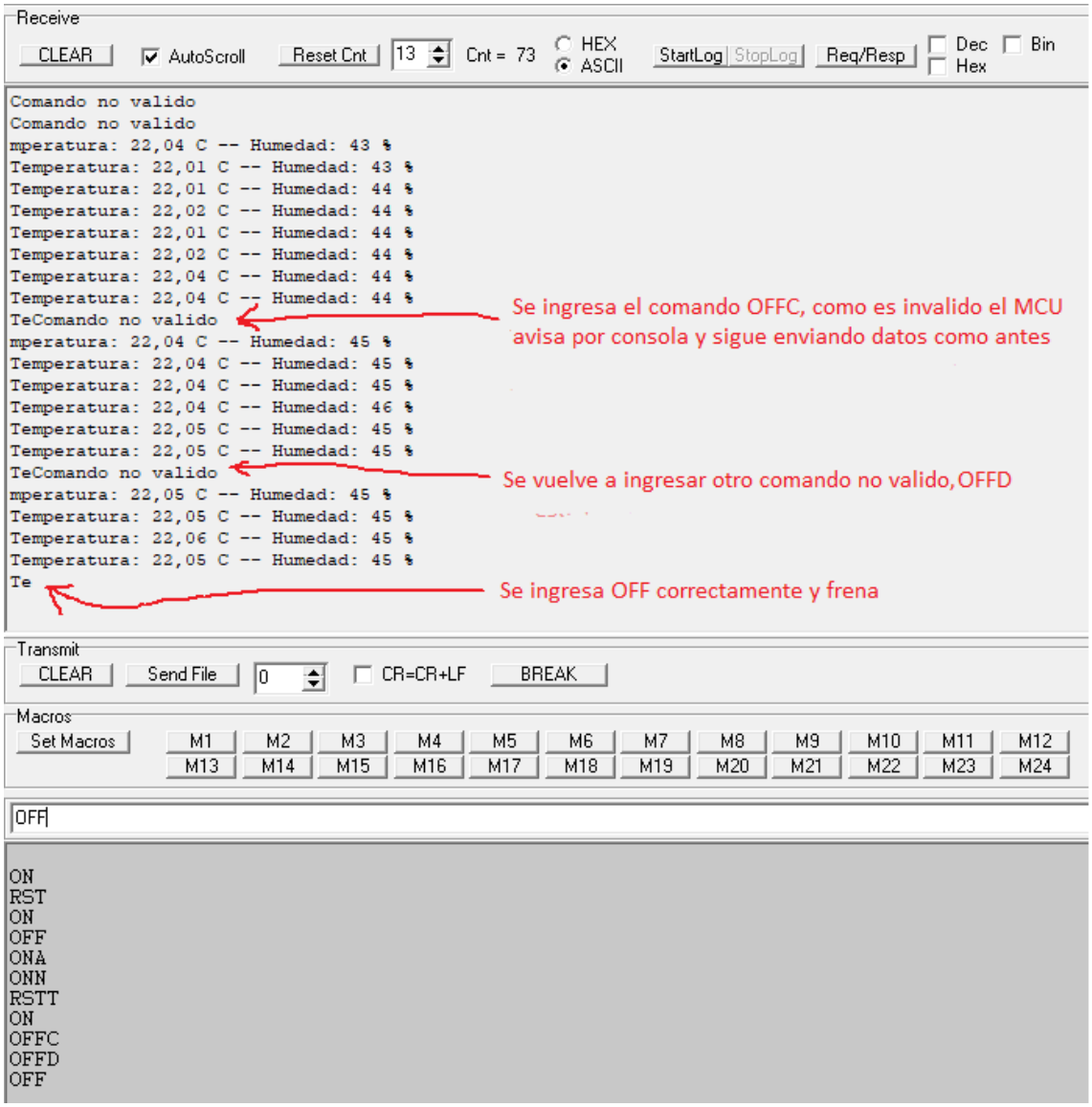
Como segunda prueba se ingresa el comando RST, para que el MCU deje de enviar los datos del sensor, muestre el menú principal y quede a la espera de un nuevo comando.



Como tercera prueba volvimos a mandar el comando ON para que empiece nuevamente a imprimir en pantalla los datos del sensor. Luego se ingresó el comando OFF para que corte la impresión como se puede ver a continuación.



Como ultima prueba se ingreso un comando invalido para corroborar que muestra el mensaje de comando invalido. Luego se puso a funcionar con ON y se ingresaron comandos inválidos para comprobar que muestre el cartel de comando invalido, pero siga realizando la tarea anterior.



Codigo:

DHT11.h:

```
#ifndef DHT11_H_
#define DHT11_H_

void dht11_iniciaryleerdato(int *, int *,int *); //Funcion para iniciar el
dispositivo dht11 para realizar una lectura de la temperatura y humedad actual;


#endif /* DHT11_H_ */
```

DHT11.C:

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>

int i;
int h=0;
int hd=0;
int t=0;
int td=0;
int sumcheck=0;

void dht11_iniciaryleerdato(int *temp , int *tempD, int *hum){
	int cant;
	h=0;
	hd=0;
	t=0;
	td=0;
	sumcheck=0;

	//configuro el pin C0 como salida y lo pongo en alto;
	DDRC |= (1<<PORTC0);
	PORTC |=(1<<PORTC0);
	_delay_ms(20);

	//Hago 0 la señal por al menos 18 ms (start signal)
	PORTC &= ~(1<<PORTC0);
	_delay_ms(18);

	//configura como entrada pull up c0 y pongo en alto la señal y en un lapso
de 20 a 40us el sensor deberia responder
	DDRC &=~(1<<PORTC0);
	PORTC |=(1<<PORTC0);

	//espero a que el sensor responda bajando la señal por 80us y despues
subiendola por 80us
	while(PINC & (1<<PORTC0));//mientras siga en alto espero a que el sensor
la haga cero
	while((PINC & (1<<PORTC0))==0);//mientras este en cero espero a que el
sensor la haga uno

	//Empieza la transmision de datos

	//humedad entero
	for(i=0; i<8; i++){
		cant=0;
```

```
		while(PINC & (1<<PORTC0)); //Espero los 50us que tarda en enviar un
bit
		while((PINC & (1<<PORTC0))==0);//Una vez que esta en alto esta
trasmitiendo el bit de informacion
		while(PINC & (1<<PORTC0)){
			_delay_us(1);
			cant++;
		}
		if(cant < 29)
			h = (h<<1);
		else
			h = (h<<1)|(0x01);
	}

	//humedad decimal
	for(i=0; i<8; i++){
		cant=0;
		while(PINC & (1<<PORTC0));
		while((PINC & (1<<PORTC0))==0);
		while(PINC & (1<<PORTC0)){
			_delay_us(1);
			cant++;
		}
		if(cant < 29)
			hd = (hd<<1);
		else
			hd = (hd<<1)|(0x01);
	}

	//temperatura entero
	for(i=0; i<8; i++){
		cant=0;
		while(PINC & (1<<PORTC0));
		while((PINC & (1<<PORTC0))==0);
		while(PINC & (1<<PORTC0)){
			_delay_us(1);
			cant++;
		}
		if(cant < 29)
			t = (t<<1);
		else
			t = (t<<1)|(0x01);
	}

	//temperatura decimal
	for(i=0; i<8; i++){
		cant=0;
		while(PINC & (1<<PORTC0));
		while((PINC & (1<<PORTC0))==0);
		while(PINC & (1<<PORTC0)){
			_delay_us(1);
			cant++;
		}
		if(cant < 29)
			td = (td<<1);
		else
			td = (td<<1)|(0x01);
	}

	//sumcheck
	for(i=0; i<8; i++){
		cant=0;
```

```
		while(PINC & (1<<PORTC0));
		while((PINC & (1<<PORTC0))==0);
		while(PINC & (1<<PORTC0)){
			_delay_us(1);
			cant++;
		}
		if(cant < 29)
			sumcheck = (sumcheck<<1);
		else
			sumcheck = (sumcheck<<1)|(0x01);
	}

	*temp = t;
	*tempD = td;
	*hum = h;

};
```

Tareas_Main.h:

```
#ifndef TAREAS_MAIN_H_
#define TAREAS_MAIN_H_

#include "DHT11.h"
#include "serialPort.h"

void Tareas_Main_leercomando(char []);//función de chequeo de string insertado
por el usuario sobre la consola que establece el modo a ejecutar
void Tareas_Main_Tarea1();//Funcion para guardar en el buffer la lectura de
temperatura y humedad


#endif /* TAREAS_MAIN_H_ */
```

Tareas_Main.c:

```
extern char Buffer[10];
extern int modo;
extern int haydato;
extern int modo;
extern int modoAnt;


void Tareas_Main_Tarea1(){
	int temp,temp_dec,hum;
	dht11_iniciaryleerdato(&temp, &temp_dec, &hum); //Leo la temperatura,
temperatura decimal junto a la humedad
	Buffer[13]='0'+temp/10;
	Buffer[14]='0'+ temp%10;

	Buffer[16]='0'+temp_dec/10;
	Buffer[17]='0'+ temp_dec%10;

	Buffer[33]='0'+hum/10;
	Buffer[34]='0'+hum%10; //Almaceno datos en el buffer para luego imprimir
por la usart
}

void Tareas_Main_leercomando(char *s){
	if (s[0]=='O'){
		if (s[1]=='N' || s[1]=='F'){
			if (s[1]=='N'){
				if (s[2]=='\n'){
```

```
                                        modo=1; //set modo 1 en caso de ser ON<ENTER>
                                }
                                else{
                                        modoAnt=modo; //Guardo modo anterior para luego
de ejecutar un comando invalido siga con la ejecución anterior
                                        modo=4; //set modo 4 en caso de no ser
ON<ENTER>
                                }
                        }
                        else{
                                if (s[2]=='F'){
                                        if (s[3]=='\n'){
                                                modo=2; //set modo 2 en caso de no ser
OFF<ENTER>
                                        }
                                        else{
                                                modoAnt=modo;
                                                modo=4; //set modo 4 en caso de no ser
OFF<ENTER>
                                        }
                                }
                                else{
                                        modoAnt=modo;
                                        modo=4;
                                }
                        }
                }
                else{
                        modoAnt=modo;
                        modo=4;

                }
        }
        else{
                if (s[0]=='R'){
                        if (s[1]=='S'){
                                if (s[2]=='T'){
                                        if (s[3]=='\n'){
                                                modo=3; //set modo 1 en caso de ser
RST<ENTER>

                                        }
                                        else{
                                                modoAnt=modo;
                                                modo=4; //set modo 4 en caso de no ser
RST<ENTER>

                                        }
                                }
                        }
                        else{
                                modoAnt=modo;
                                modo=4;
                        }
                }
                else{
                        modoAnt=modo;
                        modo=4;
                }
        }
}
```

Main.h:

```
#ifndef INCFILE1_H_
#define INCFILE1_H_


#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "DHT11.h"
#include "serialPort.h"
#include "Tareas_Main.h"
#define BR9600 (0x67)    // 0x67=103 configura BAUDRATE=9600@16MHz


#endif /* INCFILE1_H_ */
```

Main.c:

```
#include "main.h"

volatile char RX_Buffer=0;
char comando[10];
volatile char nuevoComando=0;

volatile char Buffer[] = "Temperatura: XX,XX C -- Humedad: XX %\n\r\0";//13 14 16
17 33 34
static char msg1[]="Registrador de temperatura y humedad \n\r Ingrese ON: para
encender, OFF para apagar, RST para reiniciar\n\r\0";//Mensaje utilizado en el
modo 3 y de tamaño 105
static char msg2[]="Comando no valido\n\r\0"; //mensaje utilizado en el modo 4
volatile int haydato=0; //entero para definir si se debe procesar un dato para
imprimirlo en la terminar
volatile char cant=0;
volatile char transmitiendo=0;

volatile int modoAnt;
volatile int modo=0;

int main(void){
	//modo 1(ON): muestra temperatura y humedad cada un segundo
	//modo 2(OFF): frena la transmision de humedad y temperatura
	//modo 3(RST): transmite el menu y frena la transmision de datos
	//modo 4: indica comando no valido

	SerialPort_Init(BR9600);
	SerialPort_TX_Enable();            // Activo el Transmisor del Puerto
Serie
	SerialPort_RX_Enable();            // Activo el Receptor del Puerto Serie
	SerialPort_RX_Interrupt_Enable();    // Activo Interrupción de recepcion.
	sei();                               // Activo la mascara global de
interrupciones (Bit I del SREG en 1)

	//Configuramos por defecto en modo 3
	modo=3;
	haydato=1; //aviso a la interrupción que hay dato para imprimir
	SerialPort_TX_Interrupt_Enable();//habilito interrupción para enviar datos
a la terminal
```

```
	while (1){
		if (nuevoComando){
			Tareas_Main_leercomando(comando); //Compruebo el string
enviado por el usuario sobre la terminal
			if(modo==3){
				haydato=1;
				SerialPort_TX_Interrupt_Enable();//habilito int para
imprimir msg1 si comando fue RST(ENTER)
			}
			else if(modo==4){
				haydato=1;
				SerialPort_TX_Interrupt_Enable(); //habilito int para
imprimir msg2 si comando no fue valido
			}
			nuevoComando=0;
			cant=0;
		}
		if(modo==1){
			_delay_ms(1100); //demoro 1.1 segundos para cada impresión de
la temperatura y humedad
			Tareas_Main_Tarea1();
			haydato=1;
			SerialPort_TX_Interrupt_Enable();//habilito int para imprimir
temp y hum
		}
	}
}

ISR(USART_RX_vect){ //interrupción para recibir datos de terminal
	SerialPort_TX_Interrupt_Disable();//deshabilito int para que mientras
inserte comando no imprima datos como temp y hum
	if(cant==0){
		modoAnt=modo;
		modo=2; //activo modo 2 mientras se este leyendo para no ejecutar
modo1 cuando no se debe imprimir
	}
	RX_Buffer=SerialPort_Recive_Data(); //Recibo datos de la terminal para
procesar
	if(RX_Buffer!='\r'){
		comando[cant]=RX_Buffer;
		cant++;//si usuario no dio enter voy leyendo caracteres
	}else{ //si usuario dio enter termino procesamiento de lectura y vuelvo al
modo anterior
		comando[cant]='\n';
		nuevoComando=1; //aviso al programa principal que llego un nuevo
comando
		modo=modoAnt;
	}
}

ISR(USART_UDRE_vect){ //interrupción para enviar datos a terminal
	if(haydato){
		if(modo==1){ //imprimo la temperatura y humedad actual almacenada
en Buffer mientras no llegue a fin de cadena
			static int i=0;
			SerialPort_Send_Data(Buffer[i]);
			i++;
			if(Buffer[i]=='\0'){
				i=0;
				haydato=0;
```

```
                        SerialPort_TX_Interrupt_Disable(); //llegado a fin de
línea deshabilito int para impresión sobre la terminal
                }
            }else if(modo==3){ //imprimo menu mientras no llegue a fin de
cadena
                static int j=0;
                SerialPort_Send_Data(msg1[j]);
                j++;
                if(msg1[j]=='\0'){
                    j=0;
                    haydato=0;
                    SerialPort_TX_Interrupt_Disable();
                }
            }else if(modo==4){ //imprimo comando invalido letra por letra
                static int z=0;
                SerialPort_Send_Data(msg2[z]);
                z++;
                if(msg2[z]=='\0'){
                    z=0;
                    haydato=0;
                    SerialPort_TX_Interrupt_Disable();
                    modo=modoAnt; //vuelvo a modo anterior

                }
            }
        }
}
```