Sistemas de Tempo Real: Microcontroladores
2013/14

Práctica 2: Estación meteorológica sencilla

Objetivo

Repasar los conceptos vistos en la Práctica 1 y aprender a gestionar los datos de la EEPROM y el sensor de temperatura y humedad DHT11.

Sobre la memoria EEPROM

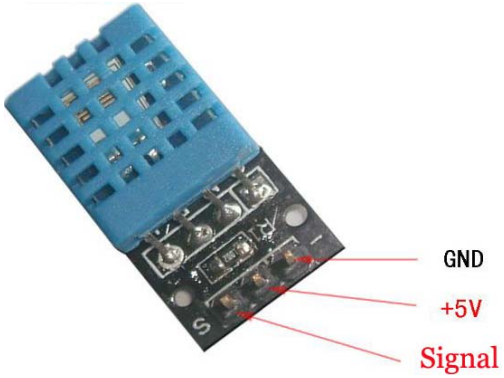
La memoria EEPROM nos permite almacenar datos que se mantendrán después de que se retire la alimentación al Arduino. Su uso principal es doble: para almacenar parámetros de configuración del sistema y guardar datos a modo de *logger*.

La cantidad de memoria EEPROM difiere de una placa Arduino a otra: las placas basadas en el ATmega328 tiene 1 KB mientras que las que poseen el ATmega168 tienen sólo 512 bytes.

La librería *EEPROM* de Arduino nos facilita la gestión de esta memoria: tan sólo precisamos utilizar las funciones *EEPROM.read(dirección)* y *EEPROM.write(dirección, valor_a_almacenar)*. En los ejemplos que acompañan al IDE existen tres que indican cómo utilizar estas tres funciones[1].

Sensor DHT11

El sensor DHT11 permite medir a la vez temperatura y humedad. Como su datasheet indica, no es un sensor rápido ni muy preciso, pero es suficiente para una primera aproximación al data logging.

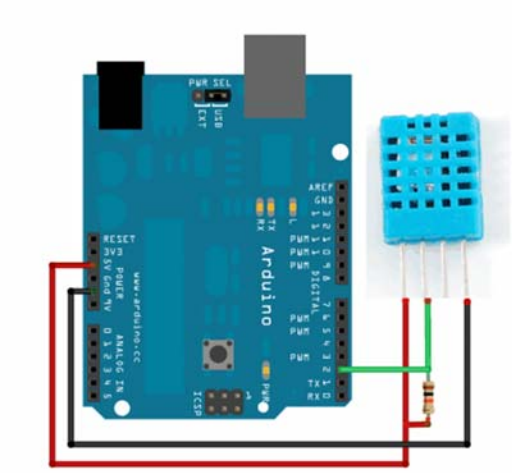


[1] Se puede probar primero el ejemplo *de EEPROM.Write* o *EEPROM.Clear* y luego el de *EEPROM.Read*.

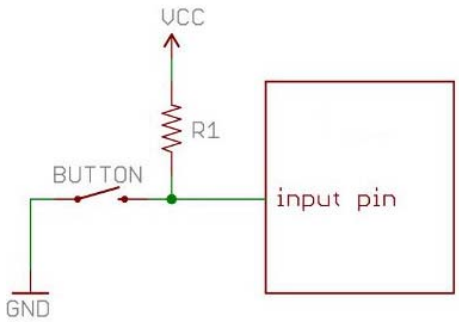
Además, en caso de que la precisión sea un problema, es reemplazable a nivel hardware directamente por el DHT22, el cual ofrece bastante más precisión y para el que el software desarrollado sigue siendo válido. En concreto, estas son las características básicas del DHT11:

Model	DHT11
Power supply	3-5.5V DC
Output signal	digital signal via single-bus
Sensing element	Polymer resistor
Measuring range	humidity 20-90%RH; temperature 0-50 Celsius
Accuracy	humidity +-4%RH (Max +-5%RH); temperature +-2.0Celsius
Resolution or sensitivity	humidity 1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-1Celsius
Humidity hysteresis	+-1%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	size 12*15.5*5.5mm

Por otro lado, su conexión *general* (i.e. cuando no se encuentra soldado a la placa de Keyes) al Arduino es muy sencilla:

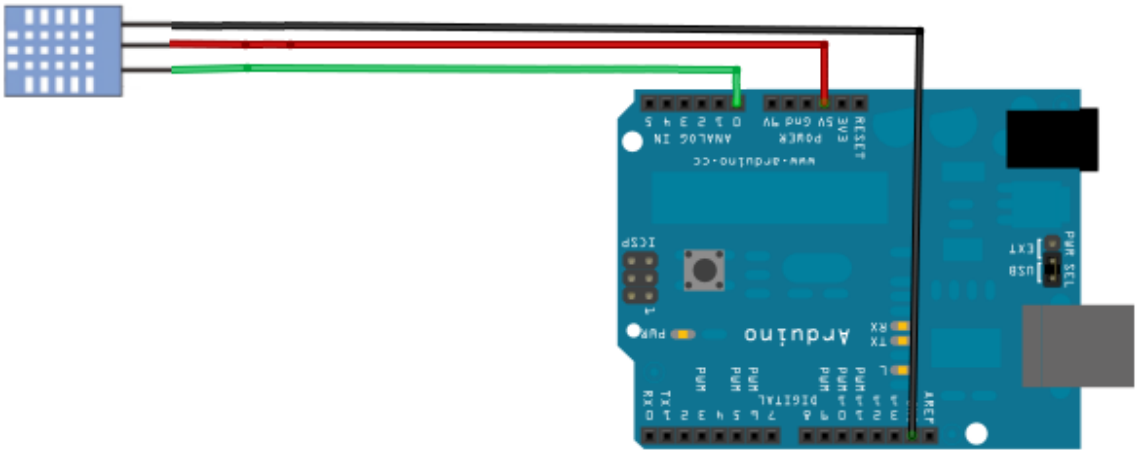


La alimentación (pin1, línea roja) puede ir entre 3V y 5V. El pin 4 va a GND, mientras que el pin 3 no se utiliza. El pin 2 (línea verde) es la línea de datos. Es recomendable colocar una resistencia de 10k entre ésta y VCC para que funcione a modo de resistencia de pull-up. Una resistencia de pull-up sirve para garantizar que el pin digital esté o en HIGH o en LOW, lo cual no se cumpliría si el pin estuviera “al aire”. Por ejemplo, si tenemos el siguiente circuito en el que se utiliza un botón en lugar del sensor



si el botón está en abierto el pin de entrada estará a HIGH. Cuando el botón se cierra, el pin se pone a GND, poniéndose entonces a LOW. En este segundo caso es cuando se justifica el uso de la resistencia de pull-up: si ésta no existiese y VCC y GND estuvieran directamente unidas se produciría un corto-circuito. El valor de la resistencia se escoge en función de la cantidad de corriente que se quiera que fluya hacia el pin de entrada. Los pines de Arduino tienen resistencias de pull-up integradas, las cuales se activan mediante la función *pinMode(número_pin, INPUT_PULLUP)*.

El módulo suministrado para estas prácticas ya contiene la resistencia de pull-up (una resistencia SMD etiquetada como R1), lo cual permite reducir el conexionado al siguiente:



El siguiente código básico permite verificar que el sensor funciona correctamente:

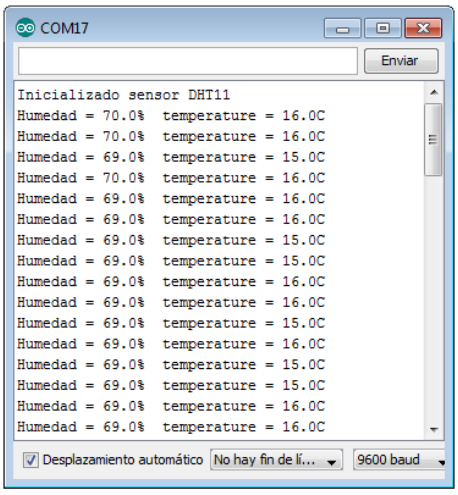
```
#define dht_dpin A0

byte bGlobalErr;
byte datos_dht[5];

void InicializaDHT(){

   //inicialialización del pin de datos
   pinMode(dht_dpin,OUTPUT);
   digitalWrite(dht_dpin,HIGH);

}

void leeDHT(){

  bGlobalErr=0;
  byte dht_in;
  byte i;

  // se lleva a cabo el procedimiento de lectura:
  //se pone el pin a LOW e inmediatamente después a HIGH
  digitalWrite(dht_dpin,LOW);
  delay(20);
  digitalWrite(dht_dpin,HIGH);
  delayMicroseconds(40);

  pinMode(dht_dpin,INPUT); //se configura como entrada el pin de datos del DHT

  dht_in = digitalRead(dht_dpin); //...y se realiza la lectura del sensor dos
                                  //veces tan sólo para verificar que funciona

  if(dht_in){
     bGlobalErr=1;
     return;
   }

  delayMicroseconds(80);

  dht_in = digitalRead(dht_dpin);

  if(!dht_in){
     bGlobalErr=2;
     return;
  }

  delayMicroseconds(80);

  for (i=0; i<5; i++)  // una vez verificado que funciona, se lee los 40 bits enviados por el DHT11
    datos_dht[i] = lee_datos_dht();


  pinMode(dht_dpin,OUTPUT); //se cambia el pin de datos a salida y se pone a HIGH
  digitalWrite(dht_dpin,HIGH);

  //se verifica el checksum
  byte dht_check_sum = datos_dht[0] + datos_dht[1] + datos_dht[2] + datos_dht[3];
  if(datos_dht[4]!= dht_check_sum)
  {
    bGlobalErr=3;
  }
```

```
}

byte lee_datos_dht(){

  byte i = 0;
  byte resultado = 0;

  for(i=0; i< 8; i++){

    while(digitalRead(dht_dpin)==LOW);
    delayMicroseconds(30);
    if (digitalRead(dht_dpin)==HIGH)
         resultado |=(1<<(7-i));

    while (digitalRead(dht_dpin)==HIGH);

  }

  return resultado;

}


void setup(){

  InicializaDHT();
  Serial.begin(9600);
  delay(300);
  Serial.println("Inicializado sensor DHT11");
 delay(700);

}


void loop(){

  leeDHT();
  switch (bGlobalErr){
     case 0:
       Serial.print("Humedad = ");
       Serial.print(datos_dht[0], DEC);
       Serial.print(".");
       Serial.print(datos_dht[1], DEC);
       Serial.print("%  ");

       Serial.print("temperature = ");
       Serial.print(datos_dht[2], DEC);
       Serial.print(".");
       Serial.print(datos_dht[3], DEC);
       Serial.println("C  ");
       break;

     case 1:
        Serial.println("Error 1: la primera verificación de inicialización del DHT ha fallado.");
        break;

     case 2:
        Serial.println("Error 2: la segunda verificación de inicialización del DHT ha fallado.");
```

```
            Serial.print(".");
            Serial.print(datos_dht[3], DEC);
            Serial.println("C  ");
            break;

        case 1:
            Serial.println("Error 1: la primera verificación de inicialización del DHT ha fallado.");
            break;

        case 2:
            Serial.println("Error 2: la segunda verificación de inicialización del DHT ha fallado.");
            break;

        case 3:
            Serial.println("Error 3: detectado fallo al almacenar los datos.");
            break;

        default:
            Serial.println("Error: código de error desconocido.");
            break;

    }

    delay(800);

}
```

El código es directo excepto la parte relacionada con la lectura de datos, tanto la configuración del pin de datos como el formato de los datos transmitidos. **Se recomienda leer el datasheet del DHT11 colgado junto a esta práctica para comprender cómo funciona a bajo nivel el protocolo de comunicaciones del sensor**.

Una vez cargado el programa anterior en el Arduino, podremos ver por el Serial Monitor una salida como la siguiente:



Enunciado de la práctica

Haciendo uso de la EEPROM, del sensor de humedad y de delays, se implementará una estación meteorológica que una vez por minuto almacene los datos de temperatura y humedad en la EEPROM[2]. Dicha EEPROM funcionará como un buffer circular: en el momento que se llene la memoria, se pasará a la primera posición de la misma y se pisarán los datos existentes. En cuanto se reciba el comando 'D' por el puerto serie se enviarán por el buffer todos los datos almacenados hasta el momento en la EEPROM y se resetearán los punteros del buffer circular.

Enviar por correo electrónico (a tiago.fernandez@udc.es) una memoria explicativa acompañada del código fuente.

[2] **Importante**: se debe almacenar tanto la parte entera como la parte real de cada parámetro.