
# Raspberry Pi (7.4)
# - mesure de t°C et RH -

## Mesure de température (t°C) et humidité (RH)

| | | |
|---|---|---|
| **Nom :**<br>**Prénom :**<br>**Classe :**<br>**Date :** | **Appréciation** : | **Note** :<br><br>**/20** |
| **Objectif** :<br>**Utilité** : | | **durée** : 4h |

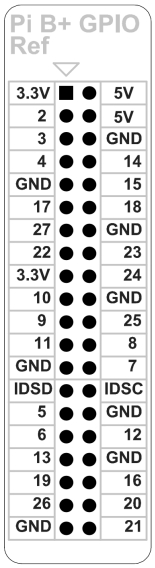
**Matériel** : plaque labdec – composants électroniques

**Prérequis** : Connexion à distance avec SSH, commande GPIO

**Compétences et savoirs principalement visées** :
C2-1, C2-2 (page 3a), C3-2, C3-3 (page 3b à 6)

**Travail à réaliser :**
-
-
-

**Schéma du système :**




## Le capteur de température (t°C) et humidité (RH)

Anglais :


- **Question :**

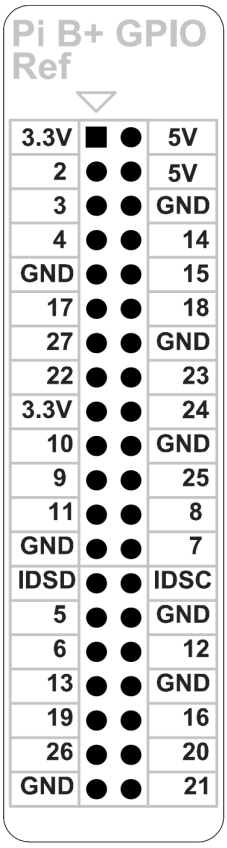
- **Question :**

**Câblage du système**

| AVERTISSEMENT |
| --- |
| **Attention** : en cas d'erreur de branchement, ton Raspberry Pi risque d'être **détruit** !!! Ne mets pas le circuit sous tension **avant** que le professeur l'ai vérifié. |

- Avant de réaliser le câblage physiquement, relie par des traits les éléments ci-dessous en t'aidant du document «ANNEXE – Connexion du module DHT Sodial»

  Note : il y a **3** connexions à réaliser

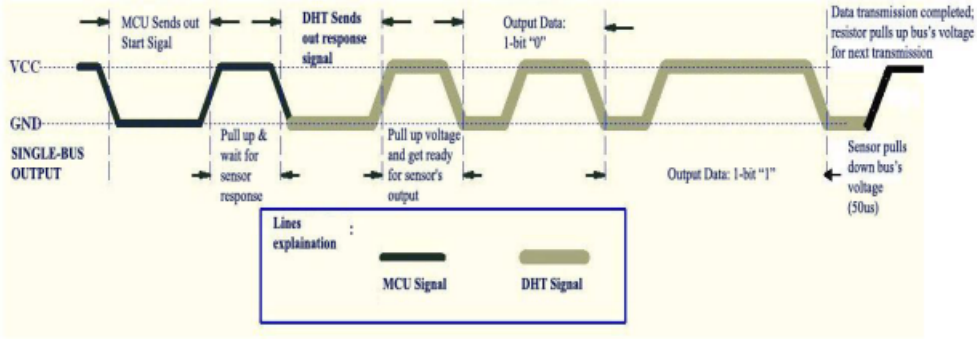




- Une fois que le schéma est complet, réalise ces connexions à l'aide des câbles fournis, puis fait valider par les professeur

**Communication entre le capteur et le Rpi**

le constructeur indique comment le micro-contrôleur (pour nous le Rpi) doit communiquer avec le capteur ; cela est résumé dans ce schéma :




Le **GPIO 17** du RPi (gpio **0** en numérotation wPi) est connecté sur la patte 2 du capteur :

**1)** au départ, il est en mode « sortie » et il « envoie » 1 au capteur, c'est le mode « repos » ;

| *Écris ci-dessous le code bash pour passer le GPIO17 en mode « sortie » et avec la valeur 1* |
|---|
| |

**2)** pour débuter une mesure, il doit passer à 0 pendant 18 ms ; puis il repasse à 1

| *Écris ci-dessous le code bash pour passer le GPIO17 à 0 pendant 18 ms ; puis repasse à 1* |
|---|
| |

**3)** puis il doit « écouter » la réponse du capteur ; pour cela, le GPIO17 doit donc passer en mode « entrée »

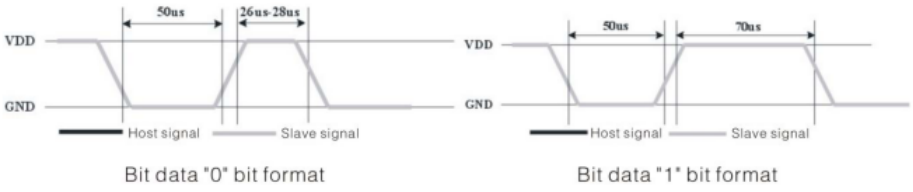
| *Écris ci-dessous le code bash pour passer le GPIO17 en mode «entrée»* |
|---|
| |

**4)** le capteur « envoie » 0, puis repasse à 1, pour indiquer qu'il est prêt à envoyer des données

| *Écris ci-dessous le code bash pour : 1) attendre que le GPIO17 passe à 0, puis 2) attendre qu'il repasse à 1* |
|---|
| |

**5)** le capteur envoie une série de 0 et de 1 ; un bit de donnée est codé de la façon suivante :

- si la durée du 0 > durée du 1, alors c'est un bit à 0,
- si la durée du 1 > durée du 0, alors c'est un bit à 1.




| |
|---|
| le capteur envoie 40 bits dont 16 bits pour à la mesure de l'humidité (RH %), 16 bits pour la mesure de la température (t°C), et 8 bits de parité |

➔ Le code source du programme est donné à la page suivante; l'étape 5 est déjà programmée ; **tu dois compléter** le code pour les étapes 1 à 4 en t'aidant des informations de la page précédente, et en traduisant le code BASH en code C

Remarques :

- les textes en rouge sont les commentaires qui expliquent la fonction réalisée par la ligne de code de code qui suit
- les parties à compléter sont en jaune
- tu trouveras de l'aide sur la programmation en C sur Google (ex : openclassrooms.com, wiringpi.com, ...)
- les fonctions dont tu as besoin sont : **pinMode** , **digitalWrite**, **delayMicroseconds**, **while** et **digitalRead**.

➔ Enregistre ce code source dans un fichier nommé : **dht.c**
➔ compile le programme avec la commande :

**gcc -Wall -o dht dht.c -lwiringPi**

➔ Le compilateur indique qu'il y a des erreurs : lire attentivement les messages puis corrige ces erreurs dans le code source, puis recommence la compilation jusqu'à ce qu'il n'y ai plus d'erreur.

➔ Après compilation, le fichier **dht** doit être présent dans le même dossier que **dht.c**

➔ Lance le programme avec la commande : **./dht**

➔ Recopie ci-dessous le message renvoyé par le programme dht :

| |
|---|
| |
| Message renvoyé par le programme dht |

➔ Suite du TP à la page 7...

*Complète le code C ci-dessous pour les étapes 1 à 4*

```
#include <wiringPi.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <ctype.h>

////////////////////////////////////////////////////////////////////  Programme principal ////////////////////////////////////////////////////////////////////
int main (int argc, char *argv[])
{
  int i, etat=1, pin=0,  duree0,   duree1,   start0,  start1; // pin est le n° du gpio utilisé
  int bit[41];
  unsigned int rh=0, t=0, rhd=0, td=0;


  wiringPiSetup() ;
  piHiPri (99);

  // pin gpio en mode sortie

  // pin gpio => 0

  // attendre 18 ms

  // pin gpio => 1

  // pin gpio en mode entrée

  //on attend que pin gpio passe à 0..

  //on attend que pin gpio repasse à 1..

  //on attend que pin gpio passe à 0..


start0=micros();

// lecture des 40 bits dans un tableau de int bit[40]
for (i = 0 ;  i < 39 ;  i++)
{
  //on attend que pin repasse à 1..
  while (etat == 0 )  { etat = digitalRead(pin); }
  // etat = 1
  start1=micros(); duree0=start1-start0;
  //on attend que pin passe à 0..
  while (etat == 1 )  { etat = digitalRead(pin); }
  // etat = 0
  start0=micros(); duree1=start0-start1;
  // bit = ?
  if ( duree1 > duree0 ) { bit[i] = 1; } else { bit[i] = 0; }
}

//interpretation des bits en valeurs : RH, t , parity
for (i = 0 ;  i < 8 ;  i++) { rh = (rh << 1) + bit[i]; }
for (i = 8 ;  i < 16 ;  i++) { rhd = (rhd << 1) + bit[i] ;}
printf("Humidité = %d.%d\n",rh);
for (i = 16 ;  i < 24 ;  i++) { t = (t << 1) + bit[i]; }
for (i = 24 ;  i < 32 ;  i++) { td = (td << 1) + bit[i]; }
printf("Température = %d.%d\n",t);

return 0;
}
```

**Découverte du fonctionnement**

- Rempli le tableau en suivant le protocole expérimental suivant : prends le capteur entre tes doigts (il va mesurer la température externe de ton corps) et utilise un chronomètre pour pouvoir lancer la commande **dht** toute les 10 secondes

| Temps (seconde) | Température (°C) | Humidité (%) |
|---|---|---|
| 0 | | |
| 10 | | |
| 20 | | |
| 30 | | |
| 40 | | |
| 50 | | |
| 60 | | |

- En déduire le température externe de ton corps : **°C**

**Automatisation**

- Crée un script BASH pour lancer automatiquement le programme dht 6 fois de suite avec un intervalle de 5 secondes.
- Recopie ton script ci-dessous :

```
# !/bin/bash
```

Script d'automation de la mesure de température

- Testes ton script en refaisant l'expérience précédente.

## ANNEXE : Connexion du module DHT Sodial


Le module Sodial intègre le DHT11 et les résistance nécessaires au focntionnement

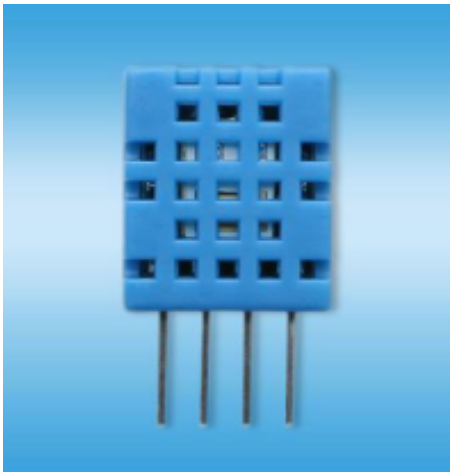
Connexion au RaspberryPi :

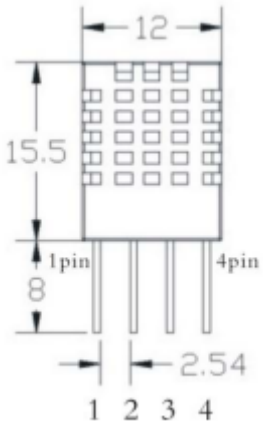
- ✔ la patte + au RPi +5V
- ✔ la patte – au RPi GND
- ✔ la patte out au RPi **gpio** bcm=**17** (wPi=0)

# ANNEXE : DHT11

## Detailed Specifications:

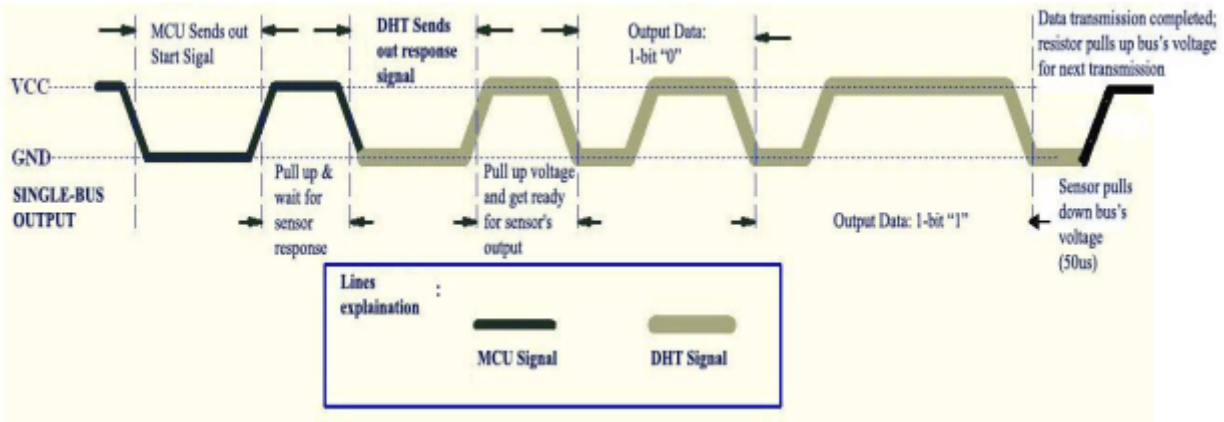
| Parameters | Conditions | Minimum | Typical | Maximum |
|---|---|---|---|---|
| **Humidity** | | | | |
| **Resolution** | | 1%RH | 1%RH | 1%RH |
| | | | 8 Bit | |
| **Repeatability** | | | ±1%RH | |
| **Accuracy** | 25℃ | | ±4%RH | |
| | 0-50℃ | | | ±5%RH |
| **Interchangeability** | Fully Interchangeable | | | |
| **Measurement Range** | 0℃ | 30%RH | | 90%RH |
| | 25℃ | 20%RH | | 90%RH |
| | 50℃ | 20%RH | | 80%RH |
| **Response Time (Seconds)** | 1/e(63%)25℃, 1m/s Air | 6 S | 10 S | 15 S |
| **Hysteresis** | | | ±1%RH | |
| **Long-Term Stability** | Typical | | ±1%RH/year | |
| **Temperature** | | | | |
| **Resolution** | | 1℃ | 1℃ | 1℃ |
| | | 8 Bit | 8 Bit | 8 Bit |
| **Repeatability** | | | ±1℃ | |
| **Accuracy** | | ±1℃ | | ±2℃ |
| **Measurement Range** | | 0℃ | | 50℃ |
| **Response Time (Seconds)** | 1/e(63%) | 6 S | | 30 S |






**Pin sequence number: 1 2 3 4 (from left to right direction).**

| Pin | Function |
|---|---|
| 1 | VDD----power supply |
| 2 | DATA--signal |
| 3 | NULL |
| 4 | GND |



Data format:

The 8bit humidity integer data + 8bit the Humidity decimal data +8 bit temperature integer data + 8bit fractional temperature data +8 bit parity bit.

◎Parity bit data definition

"8bit humidity integer data + 8bit humidity decimal data +8 bit temperature integer data + 8bit temperature fractional data" 8bit checksum is equal to the results of the last eight.

Example 1: 40 data is received:

| 0011 0101 | 0000 0000 | 0001 1000 | 0000 0000 | 0100 1101 |
|---|---|---|---|---|
| High humidity 8 | Low humidity 8 | High temp. 8 | Low temp. 8 | Parity bit |

Calculate：

0011 0101+0000 0000+0001 1000+0000 0000= 0100 1101

Received data is correct：

Humidity：0011 0101=35H=53%RH

Temperature：0001 1000=18H=24℃

Example 2: 40 data is received:

| 0011 0101 | 0000 0000 | 0001 1000 | 0000 0000 | 0100 1001 |
|---|---|---|---|---|
| High humidity 8 | Low humidity 8 | High temp. 8 | Low temp. 8 | Parity bit |

Calculate：

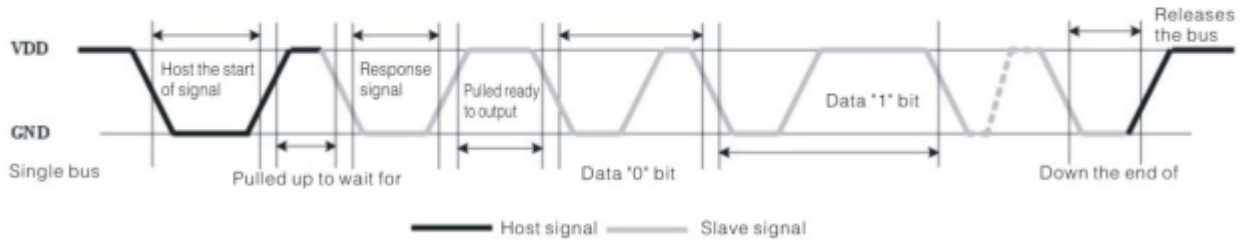
0011 0101+0000 0000+0001 1000+0000 0000 = 0100 1101

01001101≠0100 1001

The received data is not correct, give up, to re-receive data.

◎Data Timing Diagram

User host (MCU) to send a signal, DHT11 converted from low-power mode to high-speed mode, until the host began to signal the end of the DHT11 send a response signal to send 40bit data, and trigger a letter collection. The signal is sent as shown.




Data Timing Diagram

Note: The host reads the temperature and humidity data from DHT11 always the last measured value, such as twice the measured interval of time is very long, continuous read twice to the second value of real-time temperature and humidity values.

◎Peripherals read steps

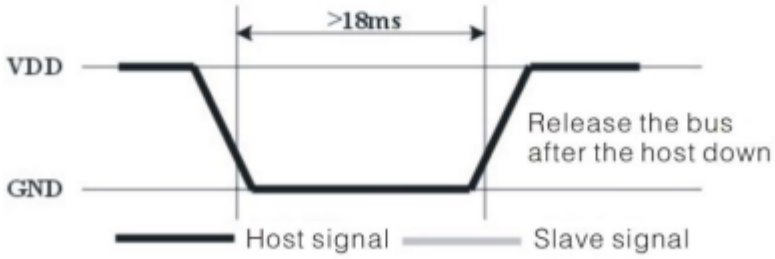
Communication between the master and slave can be done through the following steps (peripherals (such as microprocessors) read DHT11 the data of steps).

Step 1:

After power on DHT11 (DHT11 on after power to wait 1S across the unstable state during this period can not send any instruction), the test environment temperature and humidity data, and record the data, while DHT11 the DATA data lines pulled by pull-up resistor has been to maintainhigh; the DHT11 the DATA pin is in input state, the moment of detection of external signals.

Step 2:

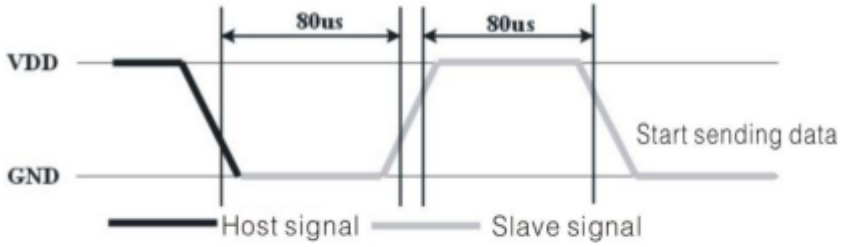
Microprocessor I / O set to output at the same time output low, and low hold time can not be less than 18ms, then the microprocessor I / O is set to input state, due to the pull-up resistor, a microprocessor/ O DHT11 the dATA data lines also will be high, waiting DHT11 to answer signal, send the signal as shown:




Host sends a start signal

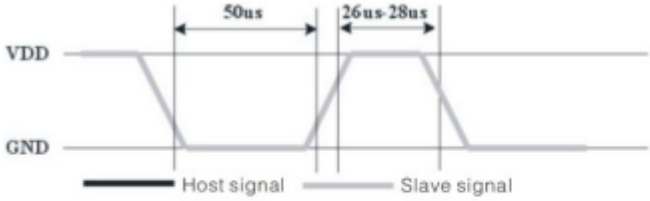
Step 3:

DATA pin is detected to an external signal of DHT11 low, waiting for external signal low end the delay DHT11 DATA pin in the output state, the output low of 80 microseconds as the response signal, followed by the output of 80 micro-seconds of high notification peripheral is ready to receive data, the microprocessor I / O at this time in the input state is detected the I / O low (DHT11 response signal), wait 80 microseconds highdata receiving and sending signals as shown:

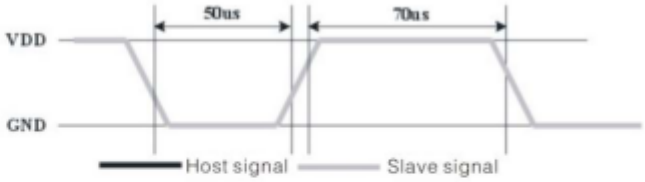



Step 4:

Output by DHT11 the DATA pin 40, the microprocessor receives 40 data bits of data "0" format: the low level of 50 microseconds and 26-28 microseconds according to the changes in the I / O levellevel, bit data "1" format: the high level of low plus, 50 microseconds to 70 microseconds. Bit data "0", "1" signal format as shown:




Bit data "0" bit format




Bit data "1" bit format

End signal:

Continue to output the low 50 microseconds after DHT11 the DATA pin output 40 data, and changed the input state, along with pull–up resistor goes high. But DHT11 internal re–test environmental temperature and humidity data, and record the data, waiting for the arrival of the external signal.

## ANNEXE : code source complet

```
#include <wiringPi.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <ctype.h>


////////////////////////////////////////////////////////////////////////////////////////////////////
//  dht
//
// testing dht22 sensor
//
//  Syntaxe : dht
//  pin : sortie raspberry pi 18
//
//


////////////////////////////////////////////////////////////////////////// Main ////////////////////////////////////////////////////////////////////////////
int main (int argc, char *argv[])
{
  int i, etat=1, pin=6;
  int  duree0,   duree1,   start0,  start1;
  int bit[41];
  unsigned int rh=0, t=0, rhd=0, td=0;


  wiringPiSetup() ;
  piHiPri (99);

  // pin en mode sortie
  pinMode (pin, OUTPUT) ;
  // pin => 0
  digitalWrite(pin,0);
  // attendre 18 ms
  delayMicroseconds(18000) ;
  // pin => 1
  digitalWrite(pin,1);
  // pin en mode entrée
  pinMode (pin, INPUT) ;

  //on attend que pin passe à 0..
  while (etat == 1 )    { etat = digitalRead(pin);  }

  //on attend que pin repasse à 1..
  while (etat == 0 )    { etat = digitalRead(pin);  }

start1=micros();  stop0=start1;

// lecture des 40 bits dans un tableau de int bit[40]
for (i = 0 ;  i < 40 ;  i++)
{

  //on attend que pin passe à 0..
  while (etat == 1 )  { etat = digitalRead(pin); delayMicroseconds(5); }
  // etat = 0
  start0=micros(); duree1=start0-start1;
  //on attend que pin repasse à 1..
  while (etat == 0 )  { etat = digitalRead(pin); delayMicroseconds(5); }
  // etat = 1
  start1=micros(); duree0=start1-start0;
  // bit = ?
  if ( duree1 > duree0 ) { bit[i] = 1; } else { bit[i] = 0; }

}

//interpretation des bits en valeurs : RH, t , parity
for (i = 0 ;  i < 8 ;  i++) { rh = (rh << 1) + bit[i] }
for (i = 8 ;  i < 16 ;  i++) { rhd = (rhd << 1) + bit[i] }
printf("Humidité = %d",rh);
for (i = 16 ;  i < 24 ;  i++) { t = (t << 1) + bit[i] }
for (i = 24 ;  i < 32 ;  i++) { td = (td << 1) + bit[i] }
printf("Température = %d",t);


return 0;
}
```