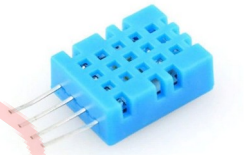
 © 2021

TP Corrigé
Capteur DHT11 : récupérer la température et l'humidité

Le capteur DHT11 est lui capable de mesurer des températures de 0 à +50°C avec une précision de +/- 2°C et des taux d'humidité relative de 20 à 80% avec une précision de +/- 5%. Une mesure peut être réalisée toutes les secondes.

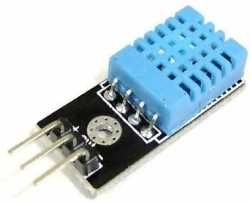


Les capteurs DHTxx communiquent avec le microcontrôleur via une unique broche d'entrée / sortie.

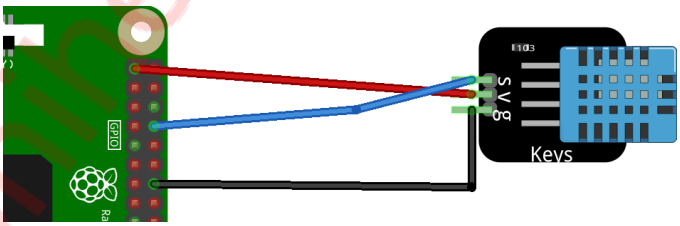
Le brochage du capteur est le suivant :

- La broche n°1 est la broche d'alimentation (5 volts ou 3.3 volts).
- La broche n°2 (S,DATA) est la broche de communication. Celle-ci doit impérativement être reliée à l'alimentation via une résistance de tirage de 4.7K ohms (il s'agit d'une sortie à collecteur ouvert).
- La broche n°3 (NC) n'est pas utilisée et ne doit pas être câblée.
- La broche n°4 est la masse du capteur (GND).

Pour le module DHT11 intégré à un PCB, les broches seront au nombre de trois, puisque le NC est retiré pour faciliter les choses. De plus, ce module inclut la résistant pull-up à la broche DATA.



Le montage à une carte Raspberry sera donc le suivant :



La communication avec un capteur DHT11 se fait en 3 étapes :

- Tout d'abord, le microcontrôleur maitre réveille le capteur en plaçant la ligne de données à LOW pendant au moins 18ms pour le DHT11. Durant ce laps de temps, le capteur va se réveiller et préparer une mesure de température et d'humidité. Une fois le temps écoulé, le maitre va libérer la ligne de données et passer en écoute. Le timeout pour recevoir une réponse est de 1000 us.

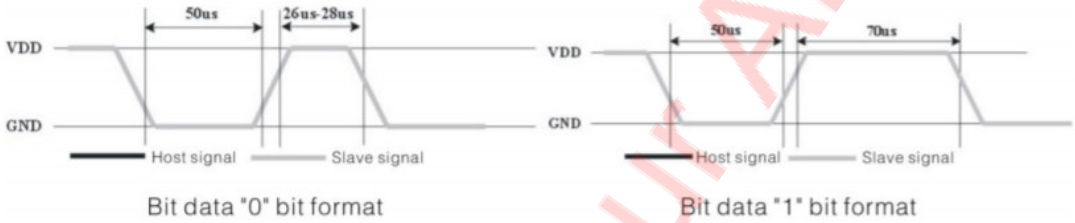
Chiheb Ameur ABID

 © 2021

• Une fois la ligne de données libérée, le capteur répond au maître (pour montrer qu'il est bien réveillé) en maintenant la ligne de données à LOW pendant 80µs puis à HIGH pendant 80µs.

• Le capteur va ensuite transmettre une série de 40 bits (5 octets). Les deux premiers octets contiennent la mesure de l'humidité. Les deux octets suivants contiennent la mesure de la température et le cinquième octet contient une somme de contrôle qui permet de vérifier que les données lues sont correctes.

Le DHT envoie la valeur de l'humidité et celle de température sur 40 en série en commençant par le bit le plus signficatif. Chaque bit de données commence par le niveau de tension basse 50us et la longueur du signal de niveau de tension élevé suivant détermine si le bit de données est à "0" ou "1" comme l'illustre la figure ci-après.



Une manière pour identifier les bits 1 et 0 est la suivante :

- Si la durée de mise à 1 est inférieure à celle de mise à 0, alors il s'agit du bit '0'
- Sinon, c'est-à-dire la durée de mise à 1 est supérieure à celle de mise à 0, alors il s'agit du bit '1'

Le format d'une mesure envoyée par le capteur est illustrée par l'exemple suivant :

Example .: 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

On se propose d'écrire un programme qui affiche à chaque 5 secondes la température et l'humidité capturées à partir du capteur DHT11.

1) Développer la classe CDht11 dont la déclaration est la suivante :

```
class CDht11 {
public:
        /*
         * No need for copying objects of this class
         */
        CDht11(const CDht11&) = delete;
        CDht11& operator=(const CDht11&) = delete;

        /*
         * Constructor
         * Provide  the GPIO pin for data transmission.
         */
```

Chiheb Ameur ABID

 © 2021

```
        explicit CDht11(const uint8_t gpioPin);

        /*
         * This method starts a measurement and returns the data <temp,humidity>
         */
        std::tuple<uint8_t, uint8_t> Measure();

private:
        /*
        * Specify which pin is connected to data line of DHT11
        */
        const uint8_t _gpioPin;
        /*
         * Signal the sensor that we want to read the data.
         */
        void SendStartSignal();

        /*
         * Wait for the GPIO pin to get low. Returns the microseconds waited until the pin
         * got low.
         */
        int WaitForLow();

        /*
         * Wait for the GPIO pin to get high. Returns the microseconds waited until the
         * pin got high.
         */
        int WaitForHigh();

        // Extracts the relevant data from the binary raw data.
        std::tuple<uint8_t,uint8_t> ProcessData(uint64_t Data);

        // Calculates the checksum for the data to make sure there was no error during
        // data transmission.
        uint8_t CalculateParity(uint8_t HumidityHigh, uint8_t HumidityLow,
                        uint8_t TemperatureHigh, uint8_t TemperatureLow);

};
```

Le constructeur permet de spécifier la broche connectée à la ligne de transmission de données avec le DHT11. Initialement, cette broche est définie comme étant une sortie avec la valeur ‘1’.

```
CDht11::CDht11(const uint8_t gpioPin) :_gpioPin(gpioPin) {
        gpioSetMode(gpioPin, PI_OUTPUT);
        gpioWrite(gpioPin, 1);
}
```

La méthode WaitForLow() renvoie le temps d’attente en microsecondes pour que la ligne de données passe à ‘0’ par le DHT11.

```
int CDht11::WaitForLow() {
        auto StartTime = gpioTick();
        while (gpioRead(_gpioPin)) {
                if (FREQUENCY < gpioTick() - StartTime) {
                        throw std::runtime_error(
                                        "Timeout while waiting for pin to get low.");
                }
        }
        return gpioTick() - StartTime;
}
```

Chiheb Ameur ABID

 © 2021

La méthode WaitForLow() renvoie le temps d'attente en microsecondes pour que la ligne de données passe à '0' par le DHT11.

```
int CDht11::WaitForHigh() {
	auto StartTime = gpioTick();
	while (!gpioRead(_gpioPin)) {
		if (FREQUENCY < gpioTick()- StartTime) {
			throw std::runtime_error(
				"Timeout while waiting for pin to get high.");
		}
	}
	return gpioTick() - StartTime;
}
```

Cette méthode réveille le capteur.

```
void CDht11::SendStartSignal() {
	gpioWrite(_gpioPin, 0);
	std::this_thread::sleep_for(std::chrono::milliseconds(20));
	gpioWrite(_gpioPin, 1);
}
```

Calculer la somme des 4 premiers octets reçus pour la vérification de la validité des données.

```
uint8_t CDht11::CalculateParity(uint8_t HumidityHigh, uint8_t HumidityLow,uint8_t TemperatureHigh,
uint8_t TemperatureLow) {
	return static_cast<uint8_t>(HumidityHigh + HumidityLow + TemperatureHigh
			+ TemperatureLow);
}
```

La méthode qui effectue la récupération des données à partir du DHT11. Elle retourne un couple dont la première composante est la valeur de la température ambiante, tandis que la deuxième composante est la valeur de l'humidité.

```
std::tuple<uint8_t,uint8_t> CDht11::Measure() {
	uint64_t data = 0;
	SendStartSignal();
	gpioSetMode(_gpioPin, PI_INPUT);
	try {
		WaitForLow();
		WaitForHigh();
		WaitForLow();
		for (int i = 0; i < 40; i++) {
			data <<= 1;
			int LowTime = WaitForHigh();
			int HighTime = WaitForLow();
			if (LowTime < HighTime) {
				data |= 0x1;
			}
		}
		WaitForHigh();
	} catch (...) {
		gpioSetMode(_gpioPin, PI_OUTPUT);
		gpioWrite(_gpioPin, 1);

	}
	gpioSetMode(_gpioPin, PI_OUTPUT);
	gpioWrite(_gpioPin, 1);
	return ProcessData(data);
}
```

Cette méthode, appelée par la méthode Measure(), extrait la température et l'humidité à partir de la trame reçue.

Chiheb Ameur ABID

 © 2021

```
std::tuple<uint8_t,uint8_t> CDht11::ProcessData(uint64_t Data) {
        uint8_t HumidityHigh = (Data >> 32) & 0xFF;
        uint8_t HumidityLow = (Data >> 24) & 0xFF;
        uint8_t TemperatureHigh = (Data >> 16) & 0xFF;
        uint8_t TemperatureLow = (Data >> 8) & 0xFF;
        uint8_t Parity = Data & 0xFF;
        if (Parity!= CalculateParity(HumidityHigh, HumidityLow, TemperatureHigh,TemperatureLow)) {
                throw std::runtime_error("Parity checksum failed.");
        }
        return std::make_tuple(TemperatureHigh,HumidityHigh);
}
```

2) Écrire le programme principal pour afficher la température et l'humidité sur la console en utilisant la classe `CDht11`.

```
#include "CDht11.h"
#include <iostream>
#include <pigpio.h>
#include <thread>
#include <chrono>
#include <tuple>

int main(int argc, char *argv[]) {
        if (gpioInitialise() < 0) {
                std::cout << "Can't initialize pigpio..." << std::endl;
                exit(-1);
        }

        CDht11 Dht11(14); // Spécifier la GPIO14 comme ligne de données
        for (int i = 0; i < 10; ++i) {
                std::this_thread::sleep_for(std::chrono::seconds(5));
                std::tuple<int,int> data = Dht11.Measure();
                std::cout << "Temperature: " << std::get<0>(data) << " °C" << std::endl;
                std::cout << "Humidity: " << std::get<1>(data) << "%" << std::endl;
                std::cout << std::endl;
        }

        gpioTerminate();
        return 0;
}
```

Chiheb Ameur ABID