Compte rendu du capteur de température DS1831

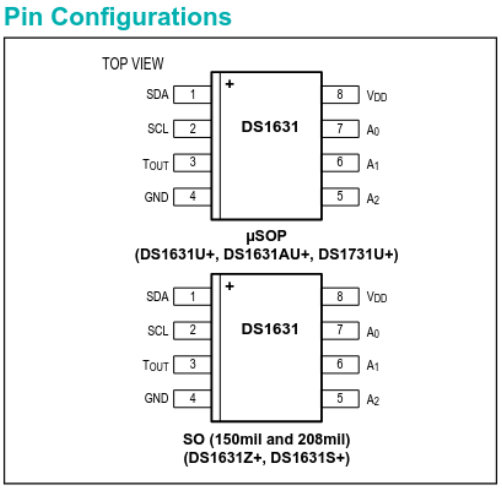
Introduction :

On souhaite créer un thermostat d'une pièce en utilisant le module DS1631 piloté via I2C. Le module DS1631 contient un capteur de température et une fonction de thermostat.

Le but est de relever la température ainsi que d'afficher les températures via le moniteur série et de régler ses paramètres.

Découverte du DS1631

Avant de commencer le câblage du DS1631, il m'a fallu d'en connaître plus sur ce module, grâce à une datasheet j'ai pu relevé les points importants qui m'ont permis ensuite d'utiliser le capteur.



Au début, j'ai cherché les pins du module pour savoir par la suite comment le câbler à l'Arduino.

"VDD" et "GND" sont les pins d'alimentation.

"SDA" et "SCL" sont les pins pour faire passer les informations et de synchronisation avec l'horloge de l'Arduino.

"Tout" enverra du courant ou non si la température se situera dans la plage de température à deux seuil réglé au préalable. Elle sert en sorte de fonction thermostat.

"A0", "A1" et "A2" sont les pins qui servent à régler l'adresse I2C du module.

DC Electrical Characteristics

(V_{DD} = 2.7V to 5.5V; T_A = -55°C to +125°C.)

PARAMETER	SYMBOL	CONDITION	MIN	MAX	UNITS
Supply Voltage	V_{DD}	(Note 1)	2.7	5.5	V
DS1631, DS1631A Thermometer Error (Note 2)	T_{ERR}	0°C to +70°C, 3.0V ≤ V_{DD} ≤ 5.5V		±0.5	°C
		0°C to +70°C, 2.7V ≤ V_{DD} < 3.0V		±1	
		-55°C to +125°C		±2	

Via le tableau ci-dessus qui communique les caractéristiques électriques du module, j'ai trouvé que le module doit être alimenté entre 2.7 Volts et 5.5 Volts. Et que sa plage de température qui peut relever se situe entre -55°C et 125°C.

Dans la documentation technique j'ai compris que pour régler le module il fallait envoyer un octet de 8 bits dans un registre qui s'appelle le "registre de configuration" qui réglera les paramètres du module.

MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	LSB
DONE	THF	TLF	NVB	R1	R0	POL*	1SHOT*
*NV (EEPROM)							

Cette octet est décomposé en plusieurs parties :

"THF" quand il est à 0 veut dire que la température n'est pas au-dessus de celle enregistrée dans le registre "THF". Si il est à 1, cela veut dire que la température est au-dessus de celle enregistrée dans son registre.

"THL" quand il est à 0 veut dire que la température n'est pas en dessous de celle enregistrée dans le registre "THL". Si il est à 1, cela veut dire que la température est en dessous de celle enregistrée dans son registre.

"NVB" n'a aucune fonction.

"POL" contrôle le "Tout", s' il est à 0, il changera "Tout" en 1 si "THF" change d'état (s'active en "HIGH") et rechargera en 0 si "THL" change d'état.

Si "POL" est à 1, il changera "Tout" à 1 si "THL" change d'état (s'active en "LOW") et redeviendra en 0 quand "THF" change d'état.

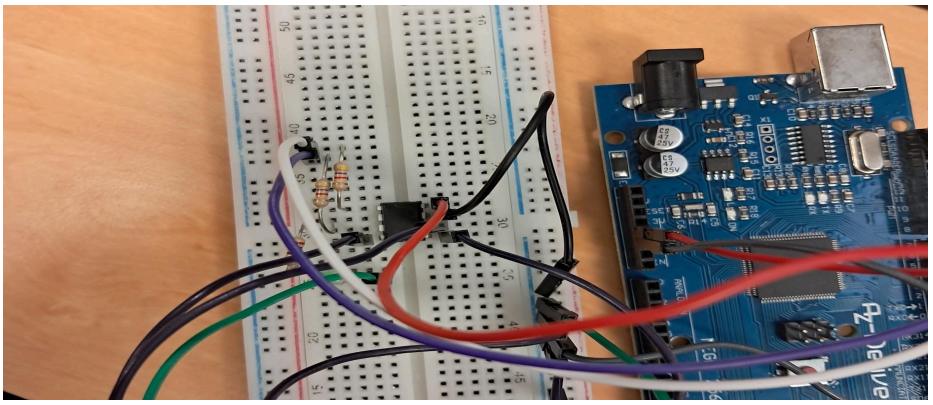
“R1” et "R0” définissent la résolution de la température capture (le nombre de chiffres après la virgule).

Table 6. Resolution Configuration

R1	R0	RESOLUTION (BIT)
0	0	9
0	1	10
1	0	11
1	1	12

Câblage à l’arduino :

Une fois que j’ai compris la base du fonctionnement de ce module il restera plus qu’à câbler le module à l’arduino en respectant les contraintes posées par la documentation technique.



La LED sert à visualiser la sortie “Tout”. J’ai utilisé deux résistances de 330 Ohms pour les bornes “SDA” et “SCL” pour des moyens de sécurité et une résistance de 1k Ohms pour la LED.

J’ai câblé “A0”, “A1” et “A2” pour que le composant utilise l’adresse 0x90.

Programmation du capteur :

Tout le code est disponible sur mon github à ce lien :
https://github.com/BOREKBenoit/DS1631

Maintenant que j'ai câblé le capteur de température à l'Arduino il faut écrire le code pour pouvoir capter la température et l'afficher dans le moniteur série.

Au début du code il faut inclure la librairie Wire qui permet d'avoir toutes les fonctions faites pour pouvoir interagir avec le capteur de température et faire changer ses valeurs de TL et TH par exemple.

```
#include <Wire.h> // Inclus la librairie Wire pour intéragir avec le capteur
#define Addr 0x90 >> 1 // Définit une variable Addr sur 0x90, qui est l'adresse du capteur.
```

Cette partie du code ci-dessous permet d'arrêter une conversion en cours et aussi à tester si la communication de l'Arduino vers le capteur fonctionne.

```
Wire.begin(); // L'arduino ce met en
Wire.beginTransmission(Addr); // L'ar
Wire.write(0x22); // L'arduino va env
int error = Wire.endTransmission(); /
```

Ensuite j'ai dû changer le registre de configuration pour qu'elle soit conforme à mes attentes et aussi j'ai changé les valeurs de TH et TL qui sont les deux seuils pour pouvoir exploiter le "Tout".

```
Wire.beginTransmission(Addr);
Wire.write(0xAC); // L'arduino
Wire.write(0x0C); // L'arduino
Wire.endTransmission();
```

J'ai ensuite accéder au registre "TL" pour pouvoir changer sa valeur sur 2 bits, il en va de même pour "TH".

```
Wire.beginTransmission(Addr);
Wire.write(0xA2); // Accède à TL pour écrire sa nouvelle valeur sur 2 Bits.
Wire.write(0x1A);
Wire.write(0x00);
Wire.endTransmission();
```

Ensuite j'affiche la nouvelle valeur de "TL" ou "TH" en utilisant la fonction "Wire.read()" dont la valeur est attribuée à l'entier "tl".

Je demande au capteur de commencer la conversion de la température en envoyant 0x51.

Pour récupérer la température captée par le capteur je vais devoir lui envoyer l'instruction d'envoyer la température convertie avec le code hexadécimal 0xAA.

La lecture de la température se fait sur 2 bits, un pour la partie entière et l'autre pour la partie décimale. Mais avant de traduire le binaire vers les décimales il faut effectuer une conversion que l'on peut voir dans la photo ci-dessous.

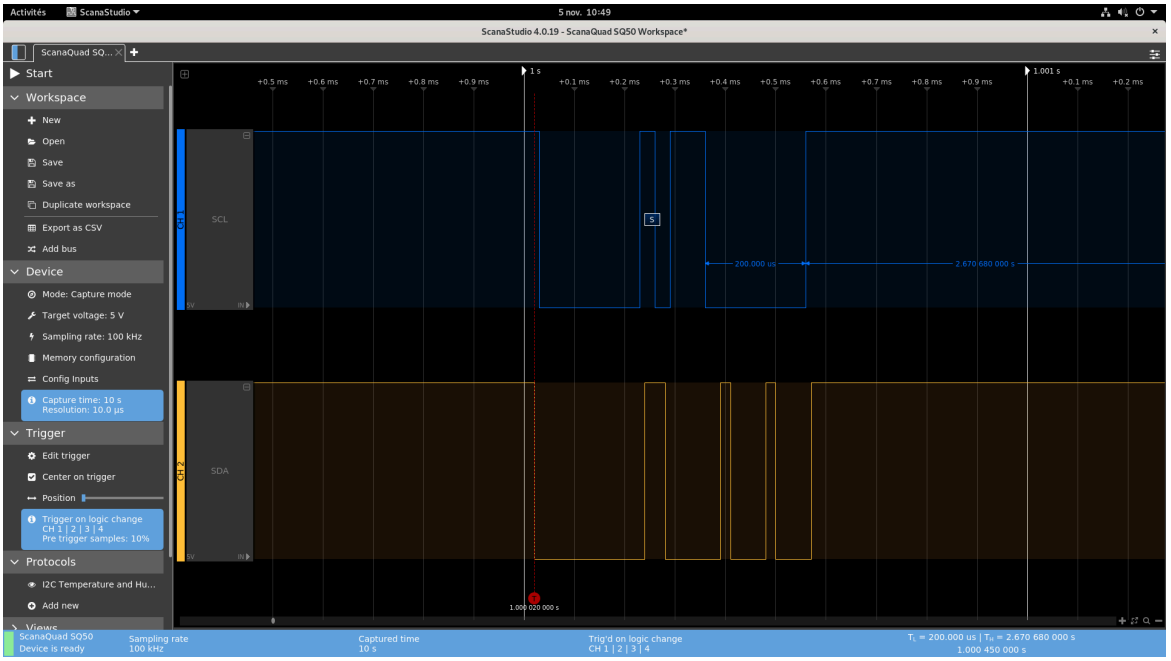
```
// Partie du code qui sert à convertir la température du binaire vers les décimales.


float partie_decimale = 0.0;

if((T_LSB & 0b10000000) == 0b10000000){
  partie_decimale = partie_decimale + 0.5;
  }
if((T_LSB & 0b01000000) == 0b01000000){
  partie_decimale = partie_decimale + 0.25;
  }
if((T_LSB & 0b00100000) == 0b00100000){
  partie_decimale = partie_decimale + 0.125;
  }
if((T_LSB & 0b00010000) == 0b00010000){
  partie_decimale = partie_decimale + 0.0625;
  }
```

Pour finir j'affiche la température traitée par l'Arduino, j'ai aussi mis une pause de 2,5ms pour laisser un temps au capteur de convertir une température légèrement différente.

Analyseur logique :

Une fois que le capteur de température fonctionne correctement, j'ai commencé à capter les trames des températures (SDA) et l'horloge qui sert à synchroniser le capteur à l'Arduino (SCL).

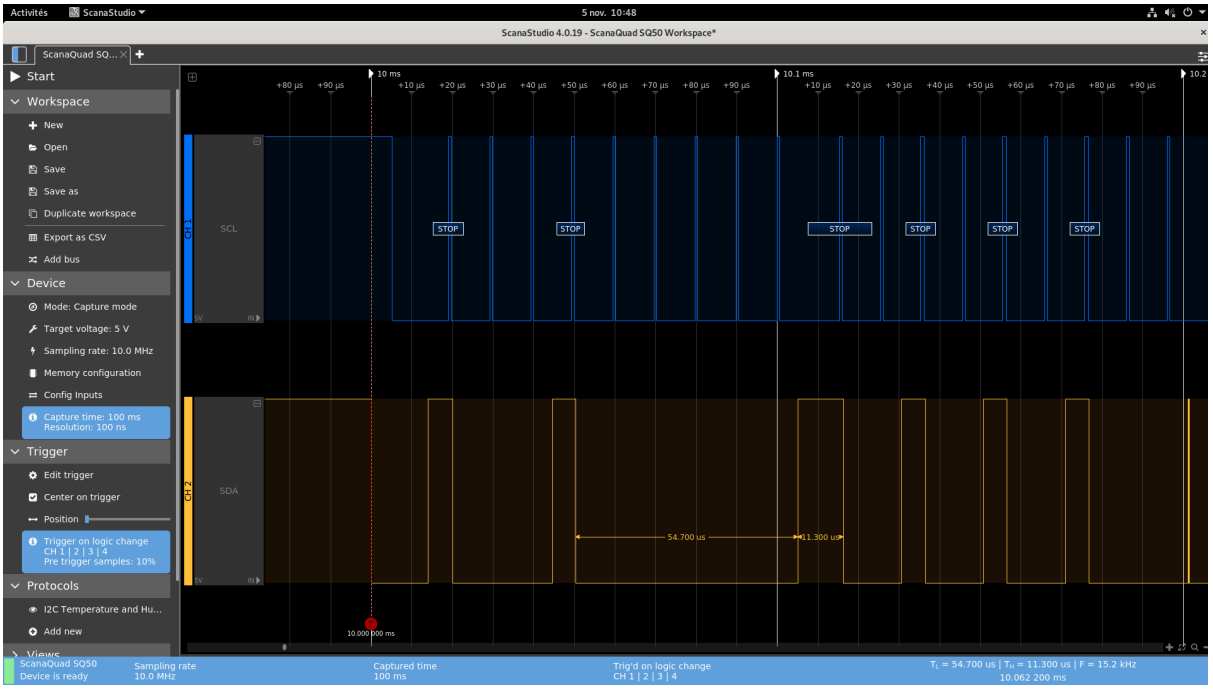
J'ai utilisé un analyseur logique ainsi que de son logiciel "ScanaStudio" pour capturer ces trames, j'ai mis un certain temps à comprendre comment le logiciel fonctionne mais il y avait un paramètre "Sampling rate" avec un choix de fréquences que je n'arrivais pas à appréhender et pendant ce temps mon analyseur logique m'affiche des valeurs complètement absurdes.



Sur la capture d'écran on voit que le SCL ne ressemble pas à ce qu' une horloge devrait ressembler c'est-à-dire une multitudes d'impulsions. Et les valeurs de SDA n'étaient pas les mêmes que celles de mon moniteur série s'affichent.

Avec un peu d'aide j'avais compris mon erreur et le "Sampling rate" est en fait la fréquence que l'analyseur va créer des points, ce paramètre règle la résolution de capture des trames.

En changeant donc cette résolution, le logiciel m'affiche cette fois-ci les bonnes valeurs.



On peut voir que l’horloge est correctement affichée et les valeurs du SDA sont les bonnes.

Utilisation du Picoscope :

Pour continuer la capture des trames j’ai utilisé un Picoscope qui est un moyen plus précis pour capturer les changements de tension.

Pour ce faire j’ai brancher mon Picoscope sur le câble SCL ou SDA et relier le Picoscope à mon ordinateur avec son logiciel installer.