

**NOM : Correction**

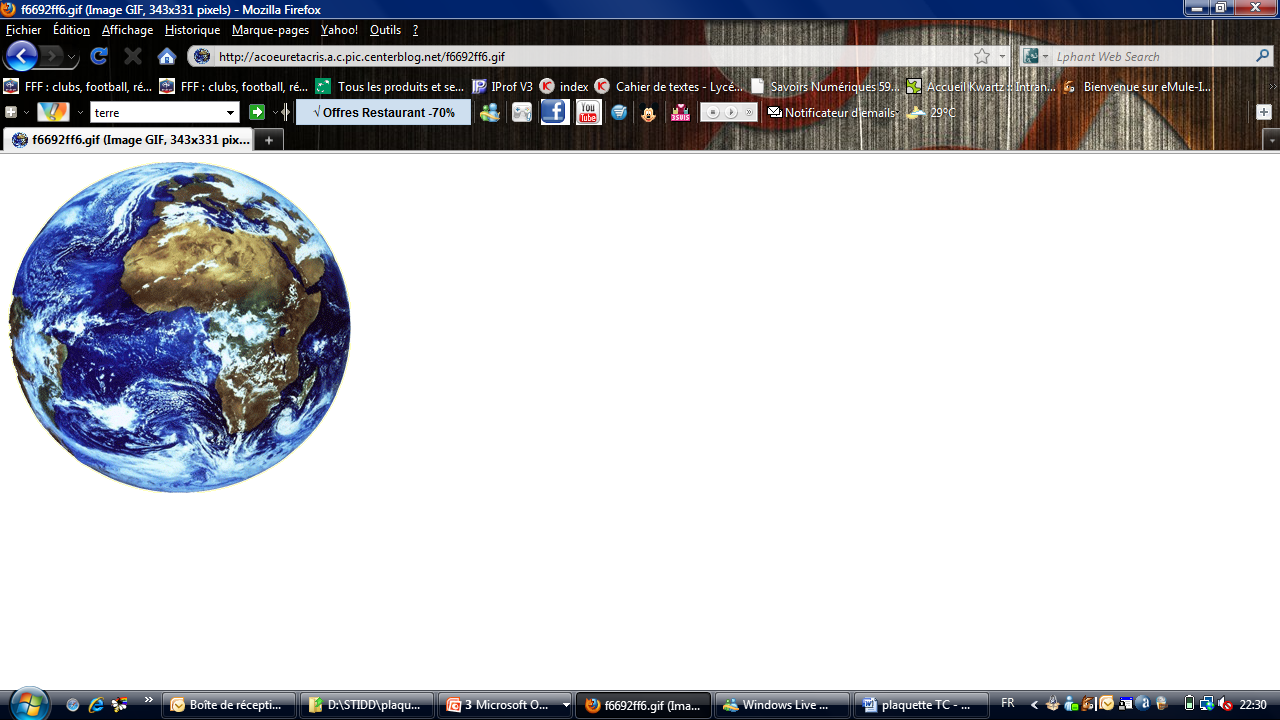
**Prénom :**

**Classe : TSTI2D\_SIN**

**Durée :**

**Note : /**

Baccalauréat STI2D



**Sciences et**

**Technologie de**

**l’Industrie**

**Développement**

**Durable**

**Et du**

Enseignement de Spécialité

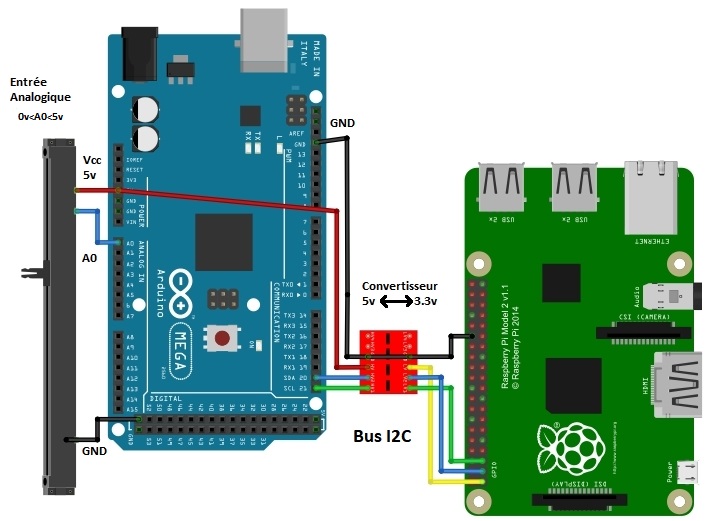
**Système d’Information et Numériques**

TP Liaison i2c

Raspberry Pi – Arduino Méga 2560 R3

Publication dans une page Web de l’entrée A0

Etude Logiciel



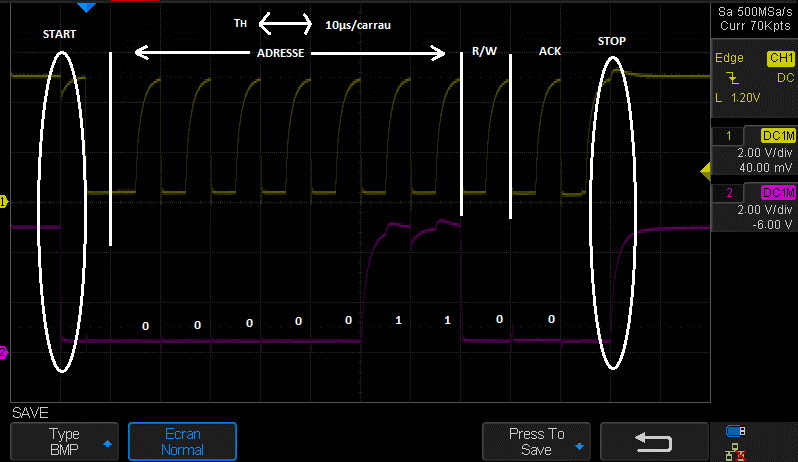
**3 : Etude Logiciel de la communication**

**3.1 : Mise en œuvre de la liaison I2C**

**3.11 : Identification du protocole**

***Indiquer sur le chronogramme suivant :***

* La condition de start
* La condition de stop
* La période du signal d’horloge TH (Oscilloscope calibre 10µs/carreau)
* Repérer l’adresse (indiquer l’état des bits 0 ou 1)
* Repérer le bit R/W (indiquer l’état du bit 0 ou 1)
* Repérer le bit Ack (indiquer l’état du bit 0 ou 1)



**Répondes aux questions :**

* Calculer la période du signal SCL (TH).
* Calculer la fréquence de communication du bus (fH).
* Dans quel état est le bit Ack ?
* Le chronogramme précédent a été relevé coté Raspberry ou coté Arduino ? Justifier votre réponse.
* Calculer la valeur de l’adresse et relever l’état du bit R/W

**Réponse :**

* La période du signal d’horloge (TH) correspond à 1 carreau donc TH =10µs

Puisque l’oscilloscope à son calibre temporel sur 10µs/carreau.

* La fréquence du signal d’horloge est donc de fH =1/TH
* Dans ce relevé l’état du bit « Ack » est actif Ack = 0, indique que le récepteur de l’adresse est actif sur le bus.
* Le chronogramme correspond au côté Arduino puisque les lignes SDA et SCL sont ramenées à 5v.
* L’adresse vaut 03H et le bit R/W = 0 (écriture, write)

**3.12 : Vérification du fonctionnement du bus**

On peut maintenant mettre en œuvre la communication sur le Bus I2C.

Si sur le système d'exploitation de la carte Raspberry on a déjà les ressources pour mettre en œuvre cette communication, sur la carte Arduino il n'y a aucun programme!

On vas donc créer un petit programme dans l’interface de développement "Arduino" qui permettra de configurer uniquement la liaison I2C.

Un programme minimum que l'on utilisera comme base dans la suite de cette étude.

**3.121: Ouvrir le logiciel "Arduino"** et copier/coller les lignes suivantes dans l'interface utilisateur.

**/\* Programme test de la liaison I2C entre la carte Raspberry et la carte Arduino.**

**\* Auteur:**

**\* Date:**

**\* Le Raspberry est maître de la communication et l'Arduino esclave.**

**\* Utilisation de la librairie Wire:**

**\* Pour information:**

**\* https://www.arduino.cc/en/Reference/Wire**

**\* http://www.mon-club-elec.fr/pmwiki\_reference\_arduino/pmwiki.php?n=Main.LibrairieWire**

**\* Programme minimum pour déclarer le bus i2c.**

**\* L'utilisation de la librairie "Wire"**

**\* L'adresse de la carte Arduino est déclaré en hexadécimale.**

**\* Elle est déclarée dans cette exemple à l'adresse N°3, première adresse libre du bus.**

**\* Rem: Le programme principal ne fait rien.**

**\***

**\*/**

**#include <Wire.h>**

**#define SLAVE\_ADDRESS 0x03 // Déclaration de l'adresse de la carte Arduino**

**// Initialisation**

**void setup() {**

**Wire.begin(SLAVE\_ADDRESS); // Initialisation de la liaison i2c**

**}**

**// Programme principal**

**void loop() {**

**delay(100);**

**}**

**3.122: Sauvegarder votre programme** après avoir complété, le nom de l'auteur et la date, sous le nom:

***test\_i2c\_VotreNom.ino***

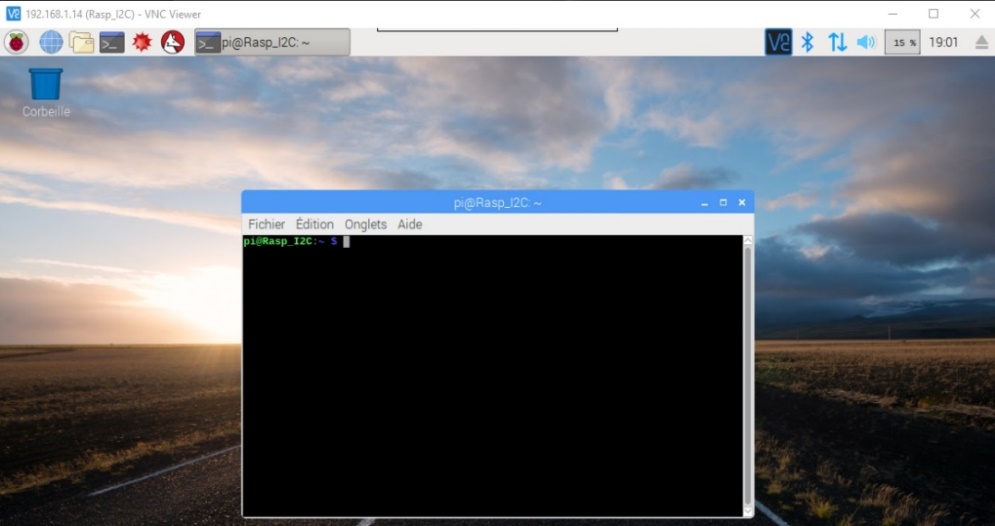
**3.123:** **Téléverser votre programme** dans la carte Arduino.

N'oublier pas de définir dans l'interface Arduino le type carte utilisé (la cible) et le port de communication qui doit être le même que dans votre gestionnaire de périphériques de votre système d'exploitation.

***Voilà c'est fini, on peut (enfin) tester la communication***

**3.124:** **Ouvrir le logiciel "VNC Viewer"** et connectez-vous sur votre carte Raspberry.

Sur votre bureau ouvrir le logiciel "**LXTerminal**".



Dans cette console, taper la commande suivante: "**i2cdetect -y 1**"



Cette commande dans le terminale vas déclencher une procédure d'analyse du bus par la carte Raspberry qui je le rappel ici est maître de la communication.

Cette procédure envoi sur le bus les adresses (adresse: 03H à 77H soit adresse: 3 à 119 en décimale) successivement avec un bit R/W à "0" (écriture) et interprète l'état du bit Ack.

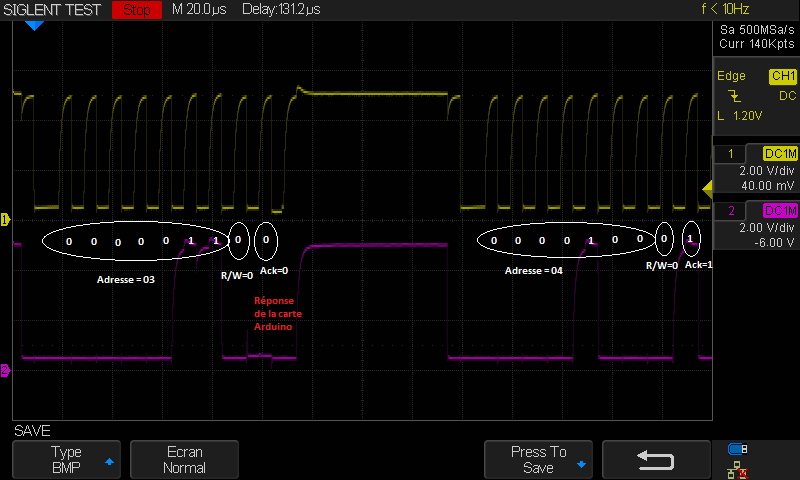
Si un périphérique est connecté au bus et voit son adresse passé sur le bus, il imposera un état bas pendant la période Ack.

Nous devrons donc obtenir avec ce programme test une réponse à l'adresse 03H (adresse de déclaration de la carte Arduino) et uniquement cette adresse s'il n'y a pas d'autre composant i2c connecté au bus.

Vous devez obtenir le résultat suivant:



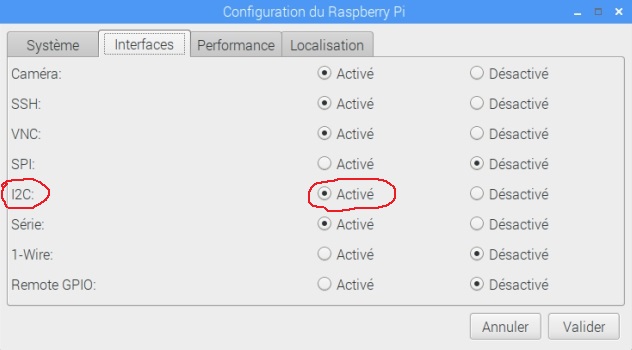
***Voici une image relevée à l'oscilloscope de la communication sur le bus:***

******

***Rem: si vous n'avez pas de réponse de la carte plusieurs problèmes sont possibles.***

***1: Vérifier votre montage (c'est fondamentale!!!), mesurer en statique la valeur les résistors de pull-up.***

***2: Avez-vous bien autorisé le fonctionnement du bus i2c dans la configuration de la carte Raspberry***

******

***3: Votre carte Arduino est bien configuré dans l'interface (bonne cible, bon port).***

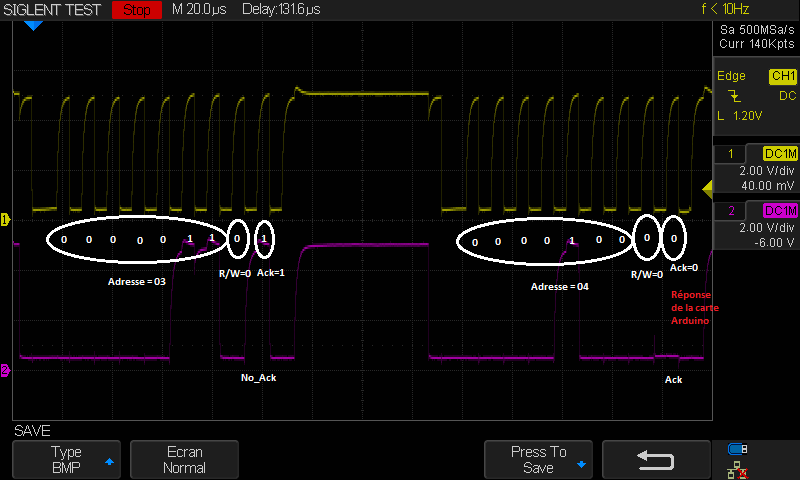
***4: au repos les deux lignes SDA et SCL sont à 5v coté Arduino et à 3,3v coté Raspberry?***

Modifier le programme "**Raspeberry\_i2c\_Arduino\_test\_VotreNom.ino"** en choisissant **une autre adresse**, dans l’exemple suivant c’est l’adresse N°4.

Coller ci-dessous votre image du résultat de l’exécution de la commande "**i2cdetect -y 1**"



***Voici une image relevée à l'oscilloscope de la communication sur le bus:***

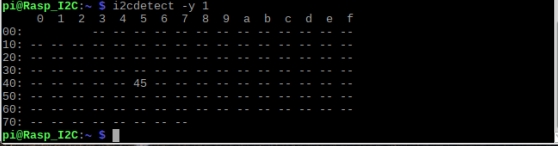
******

**Vous possédez maintenant un bus de communication opérationnel entre votre carte Raspberry (serveur du système) et votre carte Arduino.**

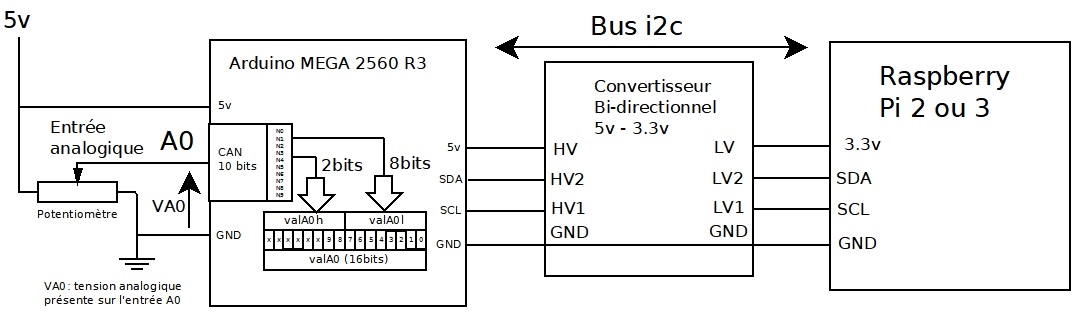
Autre exemple à l'adresse 45H soit 69 décimale déclarée dans votre programme "*Raspeberry\_i2c\_Arduino\_test\_VotreNom.ino*".

****

**Résultat de l’exécution de la commande:**

****

***3.2: Transmettre la grandeur "A0" vers la carte Raspberry.***



Nous avons maintenant un bus I2C opérationnel. Pour transmettre cette grandeur vers la carte Raspberry,

on va écrire deux programmes :

Nous avons maintenant un bus I2C opérationnel.

Pour transmettre cette grandeur vers la carte Raspberry, on va écrire deux programmes :

**Le premier** résidera dans la carte Arduino.

Ce programme répondra à une demande de la carte Raspberry (Maître de la communication) en envoyant les deux octets constitutifs de la valeur numérique de l’entrée A0 soit valA0h et valA0l, la carte Arduino étant esclave, elle doit répondre immédiatement au maître et le fera grâce à une interruption du bus i2c qui déclenchera le sous-programme de transmission d’une donnée (d'un octet). Le programme principal de la carte Arduino fera l’acquisition de l’entrée analogique A0 (valA0) périodiquement.

Le programme écrira dans la console Arduino la valeur de l'entrée analogique et les valeurs transmises, pour vérifier le bon fonctionnement du programme.

Ce programme doit réaliser les opérations suivantes :

**1 :** **Mesurer VA0 et stocker la valeur numérique de l’entrée analogique A0 (50 fois par seconde)**

La tension analogique présente sur l’entrée A0 (VA0 sur le schéma réalisée avec un potentiomètre) est convertie en une **valeur numérique « valA0 »** grâce à un **CAN** (**C**onvertisseur **A**nalogique **N**umérique 10 bits) interne au micro contrôleur de la carte MEGA 2560 R3 par l'exécution de l’instruction : **valA0 = analogRead(A0);**

Cette **valeur numérique valA0** sera stockée en mémoire dans une variable de type int (integer, entier) codée sur 16bits.

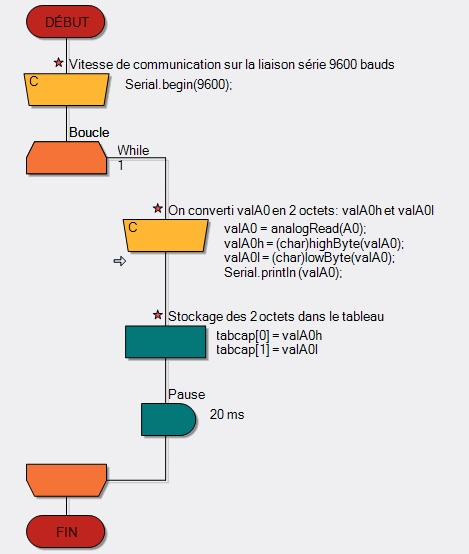
La valeur valA0 étant codée sur 16bits (CAN 10 bits oblige), pour la transmettre, il faudra la séparer en deux octets car sur le bus i2c on transmet des octets (voir schéma de principe ci-dessus) :

**valA0h** (octet de poids fort de la conversion, 2 bits significatifs)

**valA0l** (octet de poids faible, 8 bits significatifs).

Les deux octets seront stockés dans un tableau de deux valeurs afin d'être récupérés par la procédure de transmission sur le bus i2c.

**Voici une représentation graphique de cette procédure (sous Flowcode v7)**

****

**2 : Développer la procédure d’émission de donnée du bus I2C. "sendData"**

Cette procédure et un sous-programme il sera exécuté sur une interruption du bus i2c (start).

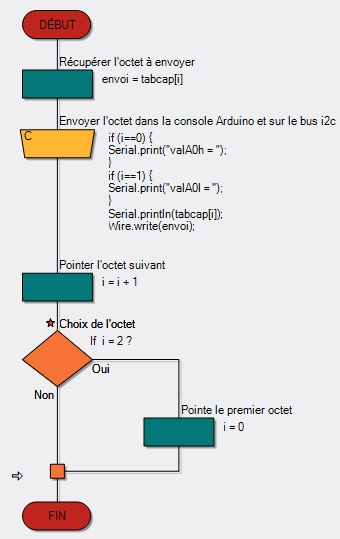
Il faudra exécuter deux fois la procédure pour envoyer successivement les deux octets du résultat de la conversion (1 octet par transmission).

La procédure d'émission doit pouvoir distinguer quelle octet elle doit transmettre?

On utilisera un index "i" qui pointera la donnée dans le tableau.

La procédure gérera ce pointeur (i) afin d'envoyer alternativement les deux octets du tableau (valA0h et valA0l), à chaque sollicitation de la carte Raspberry.

**Voici une représentation graphique de ce sous-programme Émission bus i2c :sendData**



**Le deuxième** résidera dans la carte Raspberry.

***Pour copier le programme dans la carte Raspberry j'utilise le logiciel WinSCP, il faut copier le fichier python dans le répertoire home/pi.***

On va écrire un petit script en python qui permettra de récupérer les deux octets valA0h et valA0l.

On calculera la valeur valA0 à partir des valeurs reçues : valA0h et valA0l

Le résultat du calcul et les valeurs reçues seront écrits dans la console (LXTerminal) à chaque exécution de ce script.

**Présentation du programme écrit en python:**

Il permet de réceptionner les deux octets (valA0h et valA0l) de la grandeur valA0 via le bus i2c.

« valA0 » contient le résultat de la conversion de l'entrée analogique A0 sur une carte Arduino MEGA

« valA0h » correspond à l'octet de poids fort

« valA0l » correspond à l'octet de poids faible

Le programme réalise 2 lectures sur le bus (pour récupérer les 2 octets), une lecture est décomposée en 2 temps :

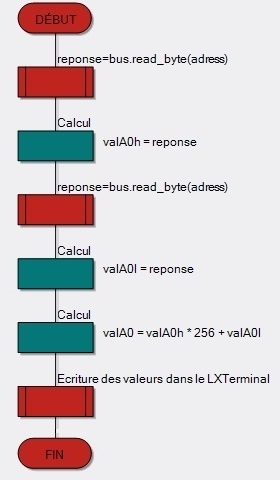
**1 temps :** Emission de l'adresse de la carte arduino avec un bit R/W = 1 pour lui signifier que l'on veut récupérer un octet**.**

La carte Arduino doit imposer un niveau bas pendant la période Ack pour signaler sa présence.

**2 temps :** Réception de l'octet.

**Chaque lecture sera encadrée par un start et un stop :**

**Voici une représentation graphique de ce programme Python Réception bus i2c**



**3.21 Procédure Arduino (Ouvrir le logiciel Arduino)**

Le programme de la carte Arduino sera sauvegardé sous le nom : ***«*A0\_i2c\_VotreNom.ino*»***

Copier/coller le listing suivant dans l'interface Arduino et **téléverser le**.

***Listing procédure "******A0\_i2c\_VotreNom.ino"***

/\* Nom du programme : ***A0\_i2c\_VotreNom.ino***

\* Programme test de la liaison I2C entre la carte Raspberry et la carte Arduino.

\* Auteur :

\* Date :

\* Le Raspberry est maitre de la communication et l'Arduino esclave.

\* Utilisation de la librairie Wire:

\*  **https://www.arduino.cc/en/Reference/Wire**

\* **http://www.mon-club-elec.fr/pmwiki\_reference\_arduino/pmwiki.php?n=Main.LibrairieWire**

\* Programme minimum pour déclarer le bus i2c:

\* L'adresse de la carte Arduino est déclaré en hexadécimale.

\*/

#include <Wire.h>

#define SLAVE\_ADDRESS 0x10// Déclaration de l'adresse de la carte Arduino ici 16 en décimale

int valA0 = 0; // valeur de conversion de l'entrée A0 sur 16 bits

byte valA0h = 0; // Octet de poids fort de valA0

byte valA0l = 0; // Octet de poids faible de valA0

int tabcap[2] = {valA0h,valA0l}; //tableau de 2 valeurs

int i=0; //index du tableau

// Initialisation

void setup() {

Wire.begin(SLAVE\_ADDRESS); // Initialisation de la liaison i2c

Wire.onRequest(sendData); // init sous-programme émission sur le bus i2c

Serial.begin(9600); // Fixe la vitesse de communication sur le port série (USB ici) à 9600 baud.

}

// Programme principal

void loop() {

valA0 = analogRead(A0); // Conversion de l'entrée A0, converti en un integer sur 16 bits (int val=).

Serial.println (valA0); // Ecriture de la valeur "val" sur le terminal (via la liaison série USB).

valA0h = (char)highByte(valA0); // Isolation de l'octet de poids fort.

tabcap[0] = valA0h; // Stockage dans le tableau, attention [0] correspond à la première case du tableau.

valA0l = (char)lowByte(valA0); // Isolation de l'octet de poids faible.

tabcap[1] = valA0l; // Stockage dans le tableau, attention [1] deuxième case.

delay(20);

}

// Sous-programme émission bus i2c

void sendData(){

int envoi = tabcap[i]; // envoi contient l'octet à transmettre

if (i==0) { // écrire dans la console Arduino l'octet transmis

Serial.print("valA0h = "); // son nom

}

if (i==1) {

Serial.print("valA0l = "); // son nom

}

Serial.println(tabcap[i]); // sa valeur

Wire.write(envoi); // émission de l'octet sur le bus

i=i+1; // pointe l'octet suivant

if (i==2) { // initialiser le pointeur

i=0; // si on dépasse le tableau

}

}

Ouvrir la console (*cliquer sur l’icône* ) dans l'interface Arduino :



**Vous obtenez dans la console la valeur de valA0 qui défile (ici 689).**

**Actionner votre potentiomètre pour vérifier son bon fonctionnement.**

**(Contrôler la plage de variation théoriquement 0 à 1023, CAN 10bits).**



**3.22 Procédure Raspberry (python)**

Maintenant que le programme Arduino fonctionne on s'occupe de celui de la carte Raspberry.

Copier/coller le listing qui suis dans un éditeur de texte.

Sauvegarder ce fichier sous le nom : « A0\_i2c\_VotreNom.py »

**Utiliser WinSCP pour stocker ce fichier dans le répertoire "home/pi/" de votre carte Raspberry**

***Listing procédure "A0\_i2c\_VotreNom.py"***

# Nom du programme : ***A0\_i2c\_VotreNom.py***

**import smbus**

**import time**

**valA0h = 0 # Declaration et initialisation des variables du programme**

**valA0l = 0**

**valA0 = 0**

**bus = smbus.SMBus(1) # Declaration du bus i2c et initialisation de l'adresse de la carte Arduino (esclave) ici 0x10 soit 16 en decimale**

**address = 0x10**

**reponse = bus.read\_byte(address) #Lecture du premier octet valA0h**

**valA0h = reponse**

**reponse = bus.read\_byte(address) #Lecture du deuxieme octet valA0l**

**valA0l = reponse**

**valA0 = valA0h\*256 + valA0l # On calcul la valeur 16bits valA0**

**print "valA0 =", valA0 # On affiche dans le console le resultat du calcul et les deux valeurs recues valA0h et valA0l**

**print "valA0h =", valA0h**

**print "valA0l =", valA0l**

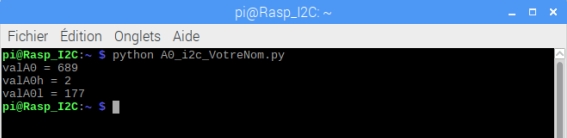
**3.23 Essais**

Il ne reste plus qu'à exécuter ce script dans la console Raspberry.

Ouvrir la console (LXTerminal) et taper la ligne de commande suivante :

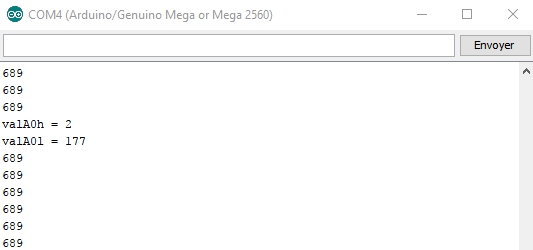
**"*python A0\_i2c\_VotreNom.py*"**

**Vous devez obtenir l'affichage des 3 valeurs dans le LXTerminal**

****

**Vous obtenez en concordance de temps dans la console Arduino :**

***Résultat des 2 passages dans le sous-programme « sendData »***



**Un relevé temporel des 2 lectures successives sur le bus i2c a donné :**

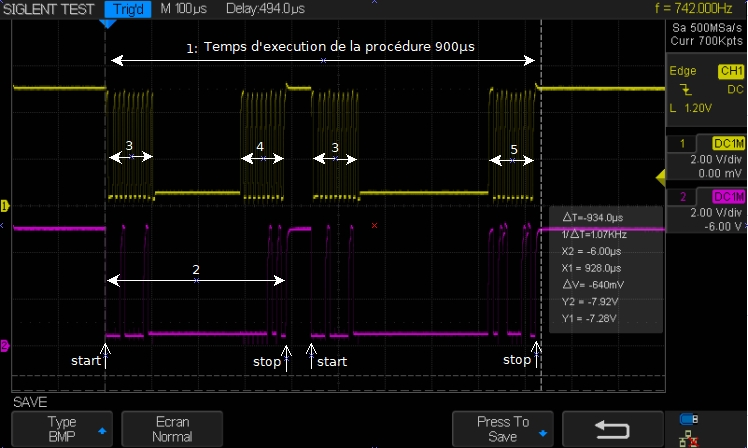
1 : Le temps d'exécution de la procédure python est d'environ 900 µs.

2 : Temps de transmission d'un octet (première lecture)

3 : Temps d’émission de l'adresse (ici 10H soit 16 en décimale), voir listing programme ligne 9.

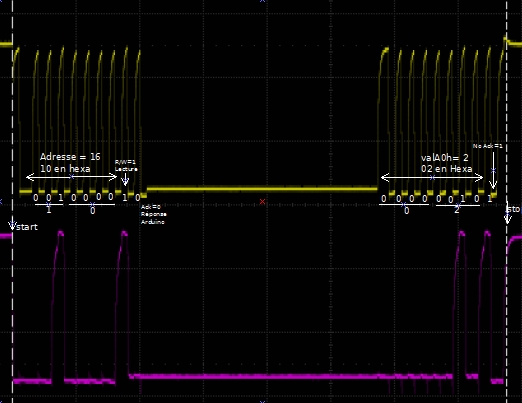
4 : Réception du premier octet (valA0h)

5 : Réception du deuxième octet (valA0l)

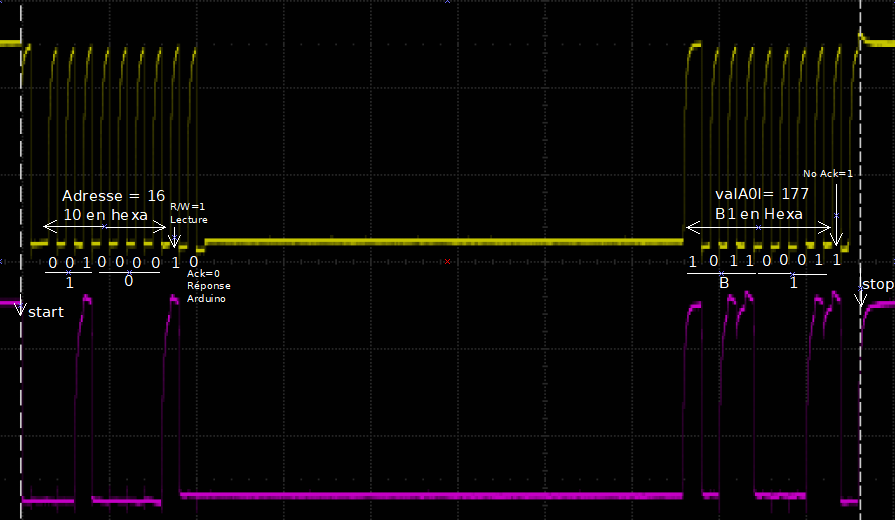
****

**Analyse des 2 lectures enregistrées :**

**2 : Réception du premier octet (valA0h)**



**2 : Réception du deuxième octet (valA0l)**

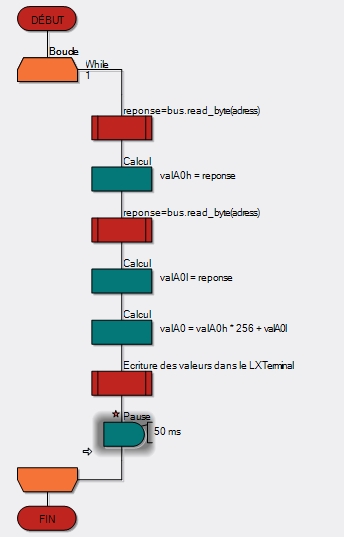
****

**3.24 Essais Multiples :**

La procédure python étant opérationnel, on va la rendre périodique.

On l'intègre simplement dans une boucle et on ajoute une temporisation afin de régler la fréquence des lectures.

**Voici une représentation graphique de ce programme Python Réception bus i2c qui boucle :**

****

**Copier/coller le listing qui suis dans un éditeur de texte.**

**Sauvegarder ce fichier sous le nom:** « **A0\_i2c\_boucle\_VotreNom.py »**

**Utiliser WinSCP pour stocker ce fichier dans le répertoire "home/pi/" de votre carte Raspberry**

***Listing procédure "A0\_i2c\_boucle\_VotreNom.py"***

# Nom du programme : ***A0\_i2c\_boucle\_VotreNom.py***

***# -\*- coding: ISO-8859-1 -\*- # compatibilité du jeu de caractère***

***import smbus***

***import time***

***valA0h = 0 # Déclaration et initialisation des variables du programme***

***valA0l = 0***

***valA0 = 0***

***bus = smbus.SMBus(1) # Déclaration du bus i2c et initialisation de l'adresse de la carte Arduino (esclave) ici 0x10 soit 16 en décimale***

***address = 0x10***

***while 0==0: #boucle infinie la condition est toujours vraie ( 0 = 0 )***

***reponse = bus.read\_byte(address) #Lecture du premier octet valA0h***

***valA0h = reponse***

***reponse = bus.read\_byte(address) #Lecture du deuxième octet valA0l***

***valA0l = reponse***

***valA0 = valA0h\*256 + valA0l # On calcul la valeur 16bits valA0***

***print "valA0 =", valA0 # On affiche dans le console le résultat du calcul et les deux valeurs reçues valA0h et valA0l***

***print "valA0h =", valA0h***

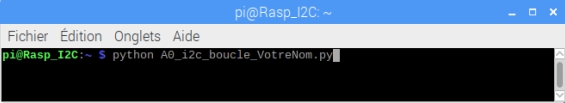
***print "valA0l =", valA0l***

***time.sleep(0.05) # Temporisation (50ms) avant de relancer une acquisition ici 20 fois par seconde***

**Il ne reste plus qu'à exécuter ce script dans la console Raspberry.**

**Ouvrir la console et taper la ligne de commande suivante :**

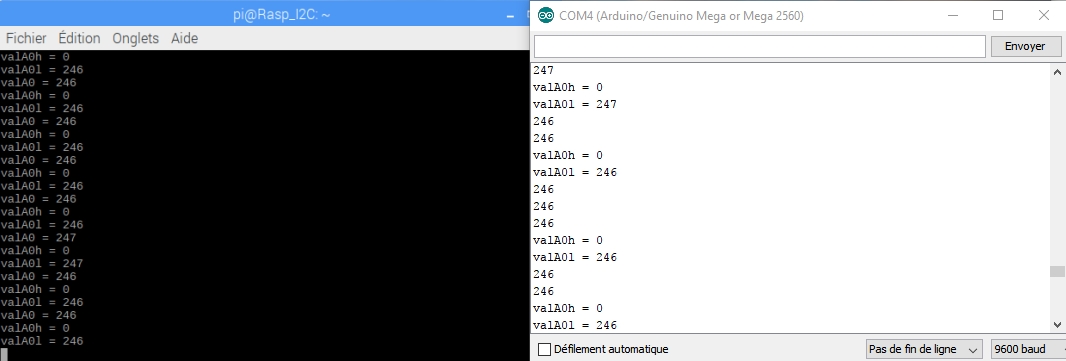
**"*python A0\_i2c\_boucle\_VotreNom.py*"**



**Vous devez obtenir l'affichage des 3 valeurs successivement dans le LXTerminal de la carte Raspberry.**

**Coté Arduino l'affichage dans la console montre une succession d'affichage de la donnée (valA0) suivi lorsque que la carte Raspberry l’interroge des valeurs de valA0h et valA0l respectivement.**

**Un résumé dans cette image qui concatène les deux consoles :**

****

***Vous avez peut-être remarqué, dans le listing python du programme* A0\_i2c\_boucle\_VotreNom.py, j'ai utilisé des caractères accentués dans mes commentaires.**

**Chose que je n'ai pas faite dans le premier programme, j'ai juste rajouté dans le programme, à la première ligne, la déclaration suivante :**

***# -\*- coding: ISO-8859-1 -\*- # compatibilité du jeu de caractère***

***J'ai trouvé la solution sur ce site :* https://info.sio2.be/python/1/9.php**