

PARTIE DE L' ETUDIANT A - BILLOT ALEXANDRE

1. Rappel du cahier des charges pour l'étudiant A.....	14
Etudiant A.....	14
Acquisitions des données :	14
2. Matériels / Protocoles utilisés.....	16
2.1 Calculateur Panda.....	16
2.3 .NET Micro Framework.....	17
2.3.1 Création d'un projet sous Micro Framework(.NET) avec le Visual Studio 2010.....	18
2.4 Protocole I2C	19
3. Présentation de la carte d'acquisition(capteurs, convertisseur, balance électronique...).....	20
3.1 PCF8591P	20
3.2 DS1621.....	20
3.3 INA125	21
3.4 La balance électronique	21
4. Acquisition des données(Température, Poids, Tension).....	22
4.1 Présentation de l'application "Acquisition"	22
4.1.1 Diagramme de classes	23
4.1.2 Utilisation d'un Thread	23
4.2 Mesure de température	24
4.3 Mesure de poids	25
4.3 Mesure de la tension.....	27
4.4 Intégration module Xbee(Emetteur).....	29
5. Conclusion	30

1. Rappel du cahier des charges pour l'étudiant A

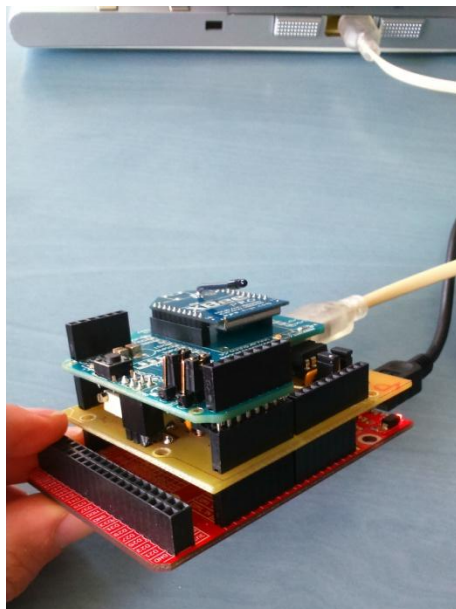
Etudiant A

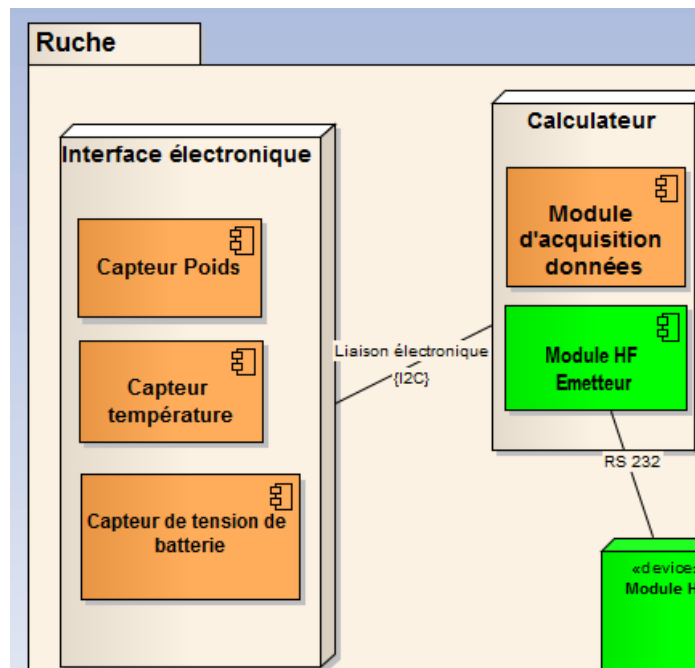
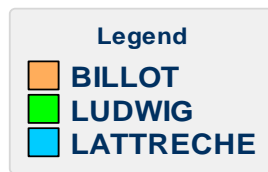
Acquisitions des données :

- *Poids de la ruche*
- *Température*
- *Niveaux de batterie*

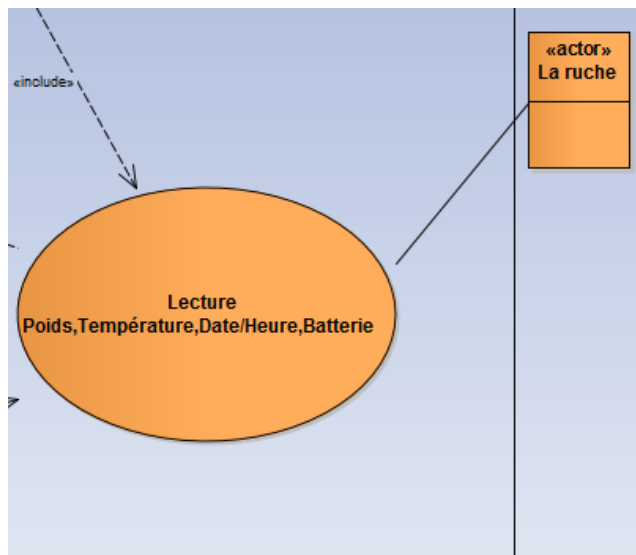
Je suis donc en charge de l'*acquisition des données* de la ruche (Essentiellement **Poids et Température et Tension**). J'ai donc en charge la calculateur Panda, qui permet de gérer facilement des entrées/sorties analogique ou numérique. (**PCF8591P**) Et d'autre part de capteurs tels que le **DS1621**.

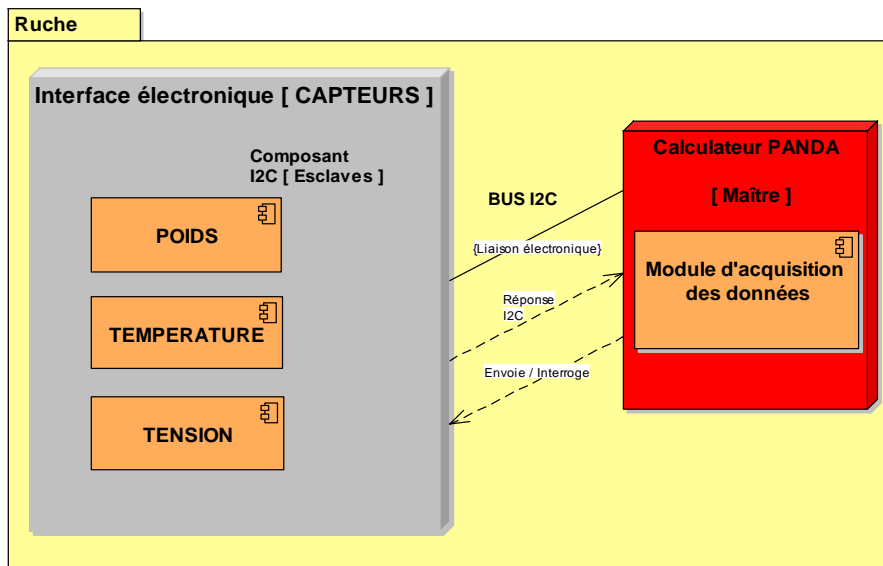
Sur le diagramme de déploiement on peut voir qu'un module(Emetteur) s'intègre au calculateur Panda. Ce module en question est un module XBee, en l'occurrence un émetteur envoyant les informations de la ruche au module récepteur et de là pouvoir les afficher sur PC par exemple.



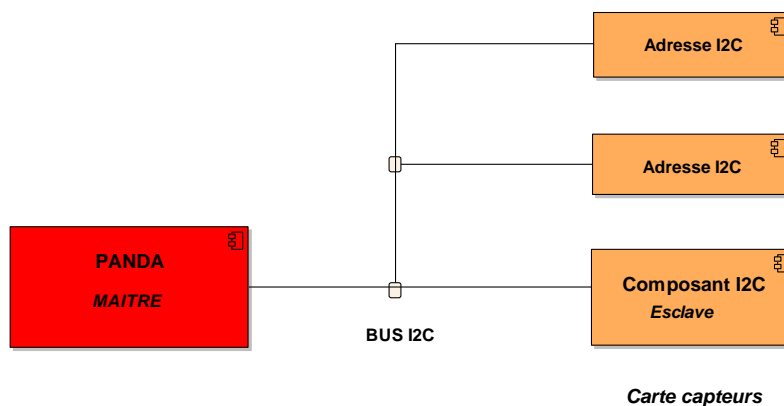


Rappel faisant référence au diagramme des cas d'utilisations :





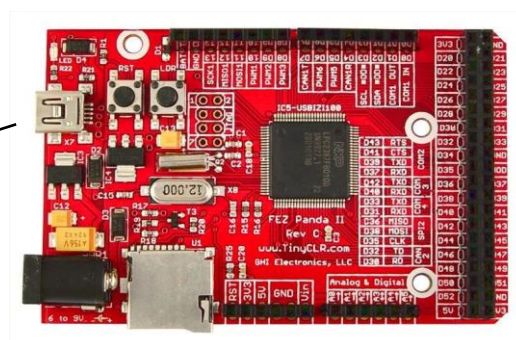
C'est un bus dit Maître/esclave dans la mesure où tout échange sur le bus ne peut se faire qu'à l'initiative du maître. Les esclaves ne font qu'y répondre par un mécanisme d'appel/réponse.



2. Matériels / Protocoles utilisés

2.1 Calculateur Panda

Un mini connecteur USB permet de relier le module au PC afin de debugger l'application.



Le choix a été tourné vers le calculateur Panda car c'est très certainement le système de développement embarquée le plus petit et le plus économique sur le marché. La carte permet de gérer très facilement des Entrées/Sorties et des ports de communication, j'utilise ici en l'occurrence celui destiné à l'I2C (SDA / SCL).

D3	I2C	(open drain pin) I2C Interface SCL	JP1
D2	I2C	(open drain pin) I2C Interface SDA	JP1

Pour qu'un **microcontrôleur** soit capable de **piloter** un produit ou un système, il faut y placer un programme permettant de piloter les actionneurs, l'interface IHM en utilisant les informations issues des **capteurs**.

Les applications sont développables sous un environnement appelé .NET Micro Framework et écrites sous le langage "**C#**" avec simplicité.

2.3 .NET Micro Framework

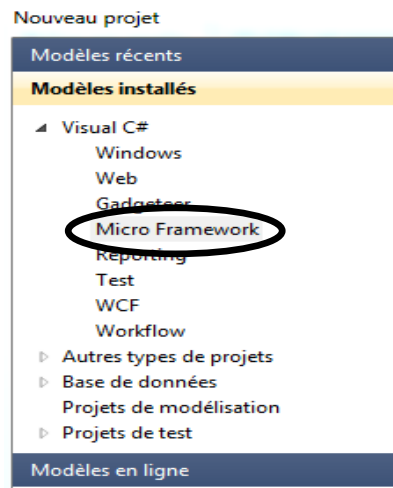


.NET Micro Framework est une plate-forme open source qui vous permet d'écrire du code C# applications gérées en utilisant Visual Studio pour les appareils embarqués contraints de ressources. .NET Micro Framework peut être utilisée pour **construire des dispositifs embarqués** sur les appareils dotés de ressources limitées se exécutant sur un microcontrôleur avec seulement quelques centaines de kilo-octets de RAM et le stockage. Les développeurs peuvent **utiliser Visual Studio, C# et .NET** connaissances pour écrire rapidement des applications embarquées sans avoir à vous soucier de la complexité de chaque microcontrôleur. Aucune connaissance approfondie de la conception de matériel n'est nécessaire pour commencer à écrire du code pour les périphériques physiques.

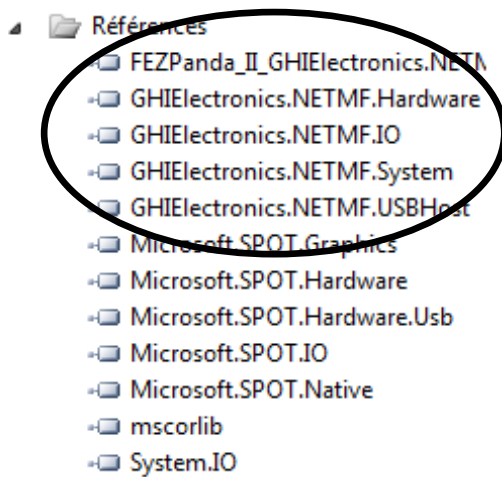
L'application sera donc créée sous un projet .NET Micro Framework avec le logiciel **Microsoft Visual Studio 2010**



2.3.1 Création d'un projet sous Micro Framework(.NET) avec le Visual Studio 2010



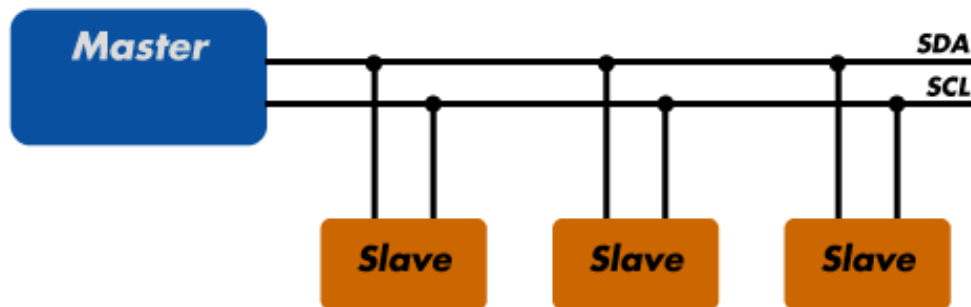
Ajout de dll indispensable pour le développement du module embarqué.



*Cette liste d'**assemblies** sont les plus utilisées*

L'ajout de référence comme on peut le voir permet d'avoir des classes issues du constructeur **Fez Panda** et également des classes génériques issues du **Framework NET** de Microsoft. Et viens aussi pour le développement de l'application des classes personnelles qui seront nécessaires afin de facilité la lisibilité.

2.4 Protocole I2C



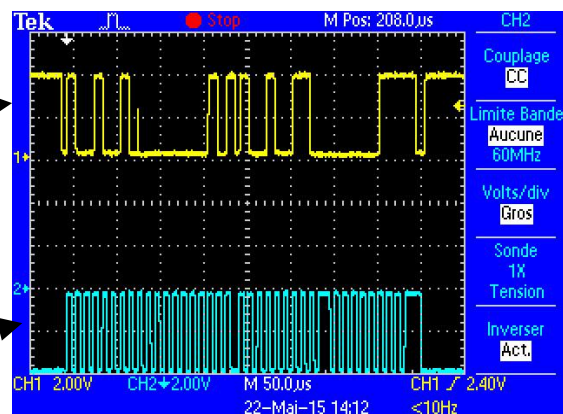
I2C a été développé par Philips pour permettre à de nombreux chipsets de communiquer sur un bus à 2 fils dans les matériels domestiques (TV, en général). I2C a un maître et plusieurs esclaves sur le même bus. Plutôt que de choisir l'esclave via une broche, I2C utilise l'adressage logiciel.

Avant de transférer des données, le maître envoie l'adresse sur 7 bits de l'esclave avec lequel il souhaite communiquer. Il envoie également un bit indiquant si le maître veut envoyer ou recevoir des données. L'esclave qui repère son adresse sur le bus confirmera sa présence et le maître pourra alors **envoyer/recevoir des données**. Le maître va débuter sa transaction avec "start ", avant d'envoyer quoi que ce soit d'autre, et terminera avec la condition "stop". Les pilotes I2C NETMF sont basés sur les transactions. *Si nous voulons lire la valeur d'un registre sur un capteur, nous aurons besoin d'envoyer le numéro du registre, puis nous lirons le registre. Ce sont donc deux transactions : **écrire puis lire**.* Tel va être l'idée pour effectuer des lectures afin de récupérer les informations issues des capteurs.

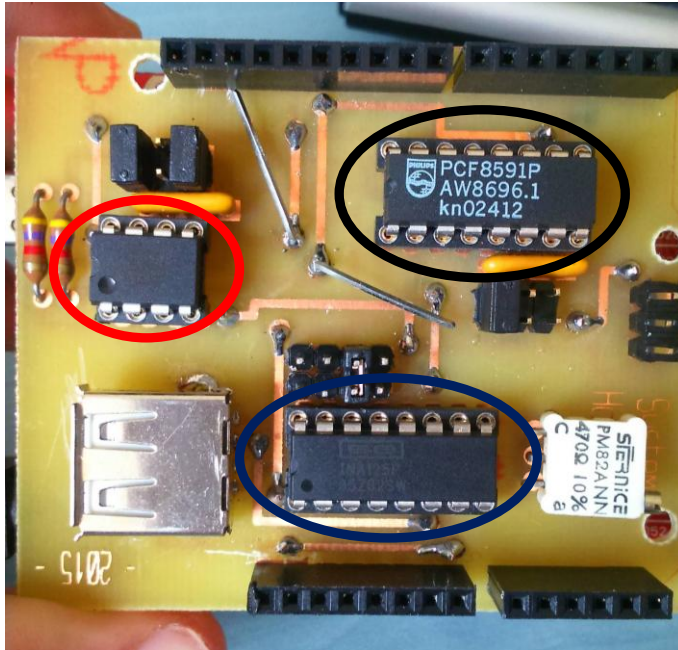
[Exemple d'une trame I2C récupérer directement sur le calculateur Panda à l'aide de l'oscilloscope](#)

- SCL (Serial Clock Line) : ligne d'horloge de synchronisation bidirectionnelle.

- SDA (Serial Data Line) : ligne de données bidirectionnelle



3. Présentation de la carte d'acquisition (capteurs, convertisseur, balance électronique...)



3.1 PCF8591P



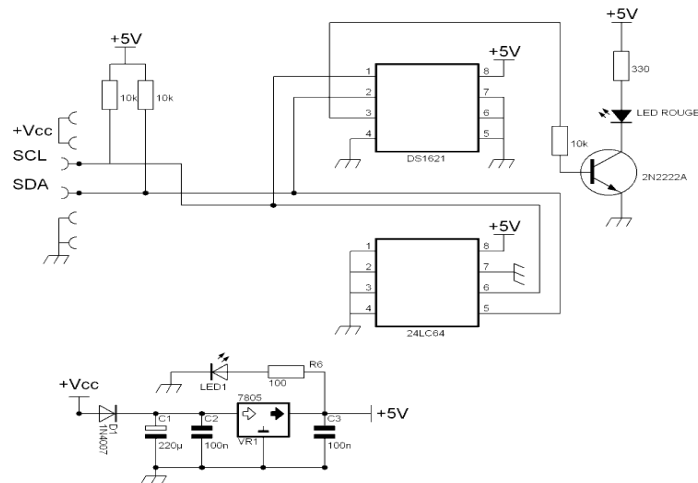
Le PCF8591 est un quadruple convertisseur analogique/numérique 8 bits combiné avec un convertisseur numérique analogique 8 bits. Les convertisseurs analogique/numérique sont de type approximation successives. La référence de tension est externe (comprise entre V_{ss} et V_{dd}), et la fréquence d'échantillonnage est contrôlée par l'horloge du bus I²C. Les entrées sont configurables logiquement, soit en entrées simple, soit en entrées différentielles.

3.2 DS1621



Le capteur de température **DS1621** permet de mesurer des températures comprises entre **-55°C et +125°C**, avec une résolution de 0.5°C. La donnée mesurée est transmise en I²C sur 9 bits (un octet contenant les 8 MSB et un octet contenant le LSB)

Interface Température :



3.3 INA125

Amplificateur d'instrumentation avec une précision de tension de référence. L'INA125 est une faible puissance, haute précision instrumentation, l'amplificateur à une référence de tension de précision. Amplification sur un seul circuit intégré.


3.4 La balance électronique

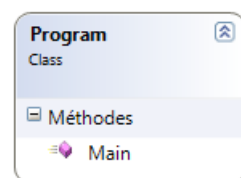
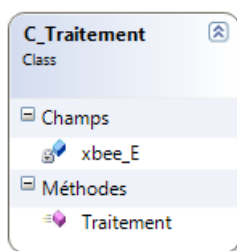
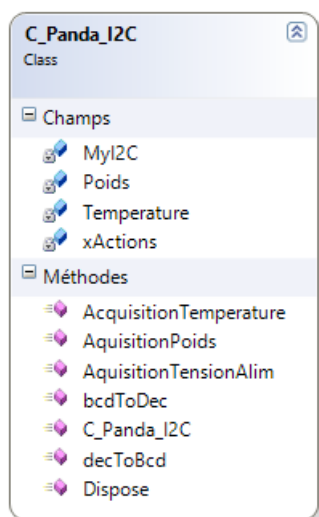
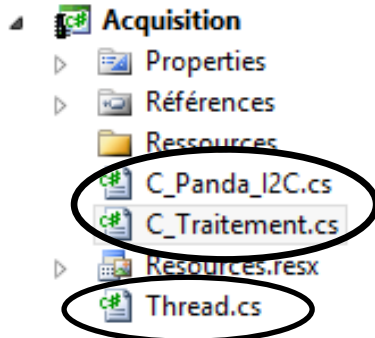
La balance électronique utilisé lors du projet faisant office de ruche pour la pesée, il y'a donc un capteur, ici de pesage vu qu'il est question de mesurer le poids.



4. Acquisition des données(Température, Poids, Tension)

4.1 Présentation de l'application "Acquisition"

 Solution 'Acquisition' (1 projet) Solution sous Visual Studio 2010 avec ici 3 classes personnelles (*C_Panda_I2C* ; *C_Traitement* ; *Thread*)



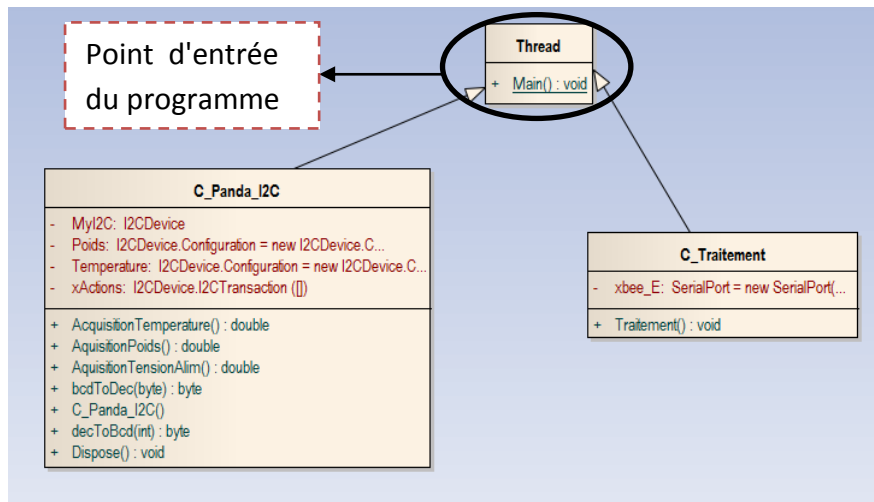
Déclaration des attributs

```
I2CDevice.Configuration Temperature = new I2CDevice.Configuration(0x4D, 400);  
I2CDevice.Configuration Poids = new I2CDevice.Configuration(0x49, 100); //PCF8594  
I2CDevice MyI2C;  
I2CDevice.I2CTransaction[] xActions; //Création d'opérations
```

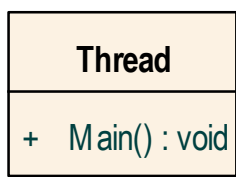
Initialise une instance de la I2CDevice.Configuration classe.

Par exemple « 0x49 » Contient l'adresse du dispositif I² C. Et « 100 » contient la fréquence d'horloge utilisée lors des communications avec le dispositif I2C, dans Khz.

4.1.1 Diagramme de classes



4.1.2 Utilisation d'un Thread



System.Threading .NET Framework facilite l'utilisation des threads. Les threads permettent de créer un certain parallélisme dans les exécutions et apportent de la puissance à une application.

On a là le point d'entrée du programme. On initialise l'objet *Thread(NewThread)* en passant la méthode à exécuter *Traitement()* puis on exécute le Thread.

```
public static void Main()
{
    //Point d'entrée du programme
    C_Traitement aquisition = new C_Traitement();
    Thread newThread = new Thread(() => aquisition.Traitement());
    newThread.Start(); //Exécution du thread
}
```

Toute les 3 secondes une acquisition s'effectuera(Délai). On appelle ça une méthode de sommeil.

```
Thread.Sleep(3000);
```

4.2 Mesure de température



La mesure de la température s'effectue grâce au capteur de température **DS1621**

Afin d'obtenir les informations issues du capteur par communication I2C, il faut effectuer 2 transactions, **Lire** puis **Ecrire**.

```
xActions = new I2CDevice.I2CTransaction[3];
```

Actions

On crée 3 transactions, 2 en écriture et 1 en lecture. Les 2 commandes correspondant à l'écriture ont des commandes du capteur comme les valeurs hexadécimale **EE** et **AA** faisant en décimale **238** et **170**.

```
byte[] commande1 = new byte[1] { 238 };  
byte[] commande2 = new byte[1] { 170 };
```

```
public double AcquisitionTemperature()  
{  
    byte[] temp = new byte[2]; //Besoin de 2 octets  
    double realtemp = 0.0; //15 chiffres de précision(virgule flottante)  
  
    MyI2C.Config = Temperature;  
    xActions = new I2CDevice.I2CTransaction[3]; //Création de 3 transactions  
  
    byte[] commande1 = new byte[1] { 238 }; //Commande EE(Début de conversion)  
    byte[] commande2 = new byte[1] { 170 }; //Commande AA(Lecture de la température)  
  
    //Transactions  
    xActions[0] = I2CDevice.CreateWriteTransaction(commande1); //Représente une transacti  
    xActions[1] = I2CDevice.CreateWriteTransaction(commande2);  
    xActions[2] = I2CDevice.CreateReadTransaction(temp); //Représente une transaction I2C  
  
    MyI2C.Execute(xActions, 1000); //Accès au bus I2C avec un délai de 1sec  
  
    realtemp = temp[0];  
    if (temp[1] != 0) //(opérateur d'inégalité)  
        realtemp = realtemp + 0.5;  
  
    return realtemp;  
}
```

C_Panda_I2C.cs

On appelle la méthode *AcquisitionTemperature()* issue de la classe **C_Panda_I2C**.

```
temp = I2C.AcquisitionTemperature();
```

Variable locale I2C

La méthode *ToString()* convertit un objet dans sa représentation de chaîne afin qu'il soit approprié pour l'affichage.

J'utilise la méthode *Substring()* qui permet de retourner une chaîne de caractères située à la position et longueur spécifiées dans la chaîne de caractères de l'objet. (0, 4)

```
temp_s = temp.ToString();  
temp_s = temp_s.Substring(0, 4);  
Debug.Print("La température est de " + temp_s + " C.");
```

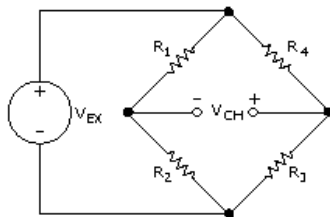
La température retournée, affichée dans la fenêtre de sortie :

La température est de 23.5 C.

4.3 Mesure de poids



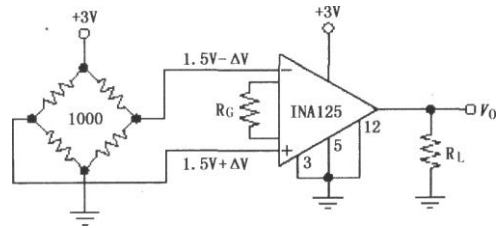
Concernant la mesure d'une masse, je dispose d'une balance électronique avec à l'intérieur un montage en pont de **Wheatstone** avec ici 4 **jauges de contraintes** (résistances).



La **jauge de contrainte** permet de mesurer la déformation lorsque que l'on applique une force, une charge exercée sur un matériau, la résistance électrique varie en fonction de la charge.

Mais une fois le pont alimenté, le signal de sortie sera très faible c'est pourquoi un amplificateur est nécessaire. L'amplificateur **INA125** va amplifier afin d'avoir une tension

beaucoup plus élevée. Puis convertir la valeur analogique en numérique(**CAN**).



```
public double AquisitionPoids()
{
    byte[] poids = new byte[1];
    double masse = 0.0;

    MyI2C.Config = Poids;
    xActions = new I2CDevice.I2CTransaction[2]; //Création tr

    byte[] commande = new byte[1] { 1 }; //Mettre 1 pour lire

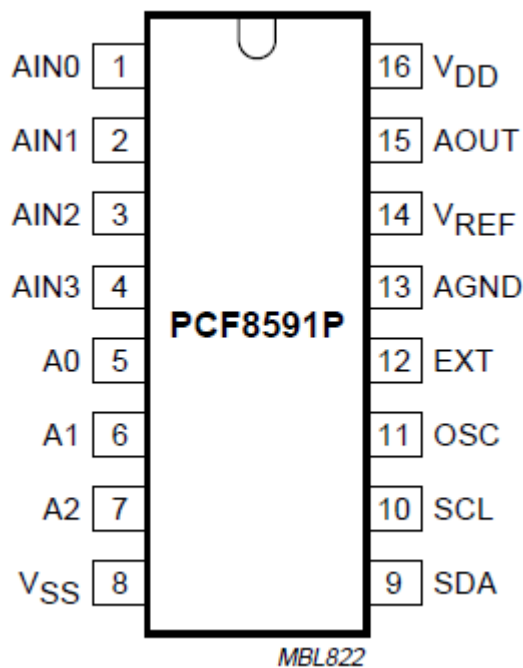
    xActions[0] = I2CDevice.CreateWriteTransaction(commande);
    xActions[1] = I2CDevice.CreateReadTransaction(poids);

    MyI2C.Execute(xActions, 1000);

    masse = (( 4.7 * poids[0]) / 255) ; //Calcul d'acquisition
    return masse;
}
```

C_Panda_I2C.cs

2 Transactions sont créer (*Ecriture et Lecture*). La commande correspond à la broche effectuant la conversion pour la mesure du poids.



J'utilise les broches 1 et 2 du convertisseur PCF8591P dans le cadre de l'application, donc les entrées analogiques AIN0 ET AIN1.

La broche 2(AIN1) correspond à la conversion du poids. Le canal 1 y correspond avec pour valeur 01(1).

00	channel 0	AIN0	channel 0
01	channel 1	AIN1	channel 1
10	channel 2	AIN2	channel 2
11	channel 3	AIN3	channel 3

```
byte[] commande = new byte[1] { 1 };
```

```
masse_s = masse.ToString();|  
masse_s = masse_s.Substring(0, 4);  
Debug.Print("Le poids est de " + masse_s + "Kg.");
```

La poids retourné, affiché dans la fenêtre de sortie : Le poids est de 0.05Kg.

4.3 Mesure de la tension

Pour la mesure de la Tension le convertisseur est toujours utilisé. Cette fois la broche 1 est utilisé. L'entrée analogique AIN0. Commande 0x00 pour lire la tension.

AIN0	channel 0
AIN1	channel 1
AIN2	channel 2
AIN3	channel 3

```

public double AquisitionTensionAlim() // Lecture de la tension
{
    double tensionalims = 0;
    MyI2C.Config = Poids;
    xActions = new I2CDevice.I2CTransaction[2];

    byte[] commande = new byte[1] { 0x00 }; //Mettre 0 pour lire
    xActions[0] = I2CDevice.CreateWriteTransaction(commande);

    byte[] alims = new byte[1];
    xActions[1] = I2CDevice.CreateReadTransaction(alims);

    MyI2C.Execute(xActions, 1000);

    tensionalims = ( 4.7 * alims[0]) / 255; //Calcul d'acquisition
    return tensionalims;
}

```

```

tensionAlim_s = tensionAlim.ToString();
tensionAlim_s = tensionAlim_s.Substring(0, 3);
Debug.Print("La tension d'alim est de " + tensionAlim_s + "V.");

```

La tension retournée, affichée dans la fenêtre de sortie : La tension d'alim est de 4.68V.

Affichage de l'ensemble des informations(Température, Poids et Tension) dans la fenêtre de sortie :

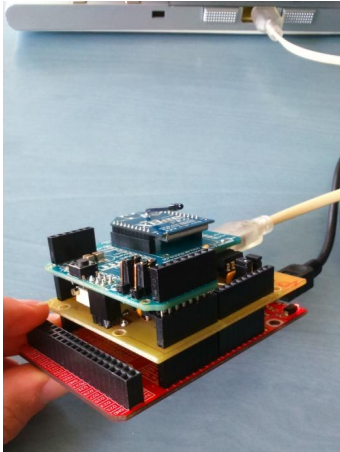
```

data = masse_s + " " + temp_s + " " + tensionAlim_s;
En cours d'acquisition ...
Le poids est de 0.05Kg.
La temperature est de 23.5 C.
La tension d'alim est de 4.68V.

```

Affichage des données sous forme de chaîne.

4.4 Intégration module Xbee(Emetteur)



Création d'un port série pour la communication avec le module Xbee

```
SerialPort xbee_E = new SerialPort("COM1", 9600, Parity.None, 8, StopBits.One);
```

Si le port est ouvert on affiche un message indiquant que le module sans est bien connecté.

```
xbee_E.Open();  
if (xbee_E.IsOpen == true)  
{  
    Debug.Print("Module sans fil connecté !");  
}
```

Module sans fil connecté !
En cours d'acquisition ...

5. Conclusion

Je dirais que ce projet a été très intéressant et formateur. Il m'a permis de développer et renforcer nombre de mes compétences et d'en acquérir de nouvelles. En effet à partir de la réflexion du cahier des charges jusqu'à la réalisation de mes applications cela a nécessité une réflexion du cahier des charges jusqu'à la réalisation de mon application. Des difficultés ont été rencontrées comme pour par exemple la communication i2c utiliser avec le calculateur Panda, ce qui était vraiment nouveau pour moi.

Sinon il a été très enrichissant pour moi de participer à la réalisation d'un long projet. Cela m'a permis d'apprendre la démarche pour gérer un projet en répondant au mieux aux besoins d'un client et apprendre à gérer mon temps sur une échelle de plusieurs mois.