Rapport de projet IMA4

Projet CanSat

Burtaire Céline
Gouenard Robin

Sommaire

Introduction

Dans la cadre de notre quatrième année à l'école Polytech'Lille nous devions réaliser un projet de 40 heures. Nous avons choisi de nous intéresser au projet CanSat.

Bien que nous ne pouvions pas participer à la compétition à cause d'un début de projet tardif et d'un emploi du temps plutôt serré nous avons réalisé ce projet dans le but que notre travail soit repris par une équipe l'année prochaine, avec le club Robotech par exemple.

Ce rapport est destiné principalement à servir de support pour la reprise du projet. Nous y expliquerons en particulier comment fonctionnent les différents modules de notre application, ainsi que les prochaines étapes à réaliser pour améliorer le prototype.

Présentation du projet

La compétition CanSat a été créée en 1998 au États-Unis et existe depuis 2009 en France. CanSat, pour "Canette-Satellite" est une sonde spatiale contenue dans un volume équivalent à une canette de soda de 33 cl ou d'1l. Cette sonde est lancée par une fusée ou un ballon et redescend sous parachute en effectuant ses missions.
Ces missions peuvent avoir un objectif scientifique, d'atterrissage de précision ou libre. Il faut obligatoirement réaliser un sondage atmosphérique et une mission libre pour participer au concours.

Dans le cadre notre projet nous nous sommes contentés de réfléchir à la première mission, c'est à dire le sondage atmosphérique.

Nous avons souhaité relever la température, l'altitude et la vitesse de notre sonde et envoyer ces données en temps réel vers un ordinateur au sol.

Cahier des charges

Avant de rentrer au cœur du sujet il a fallu définir notre cahier des charges. Les consignes permettent beaucoup de libertés et c'est à l'équipe chargé de réalisé le CanSat de se fixer des objectifs concrets et réalisables.

Finalement, voilà ce que nous avons décidé de réaliser et la manière que nous avons choisi pour y arriver.

Objectif	Manière
Communication en temps réel	Module ZigBee Pro
Relevé de la température	Capteur de température
Relevé de la vitesse	Accéléromètre
Relevé de l'altitude	Capteur de Pression
Traitement des données	microprocesseur

Le détail de chacune de ces parties est détaillé dans la suite de ce rapport.

Réalisation du projet

Liste des composants

Modules Zigbee/Zigbee Pro

Afin de recevoir en temps réel les données du processeur embarqué, nous avons besoin d'un module de communication sans fils. Nous avons fait le choix d'utiliser des modules XBee qui présentaient plusieurs caractéristiques intéressantes. Ils s'utilisent facilement avec un Arduino car ils utilisent une communication par liaison série. Ils sont de plus relativement performants pour l'utilisation que l'on veut en faire. En effet, deux XBee Pro peuvent normalement communiquer avec une portée de 1500m dans un espace dégagé, ce qui est largement suffisant pour communiquer dans l'air à l'altitude demandée par la compétition (150 mètres pour la compétition française). Nous avons utilisé dans ce projet les Xbee S1 présents à l'école Polytech'Lille, mais on peut utiliser les Xbee Pro qui ont une configuration et implantation identique.

Configuration

Pour que les deux XBee puissent communiquer, il faut d'abord les configurer. Nous avons choisi d'utiliser les commandes AT. On peut à priori utiliser n'importe quel terminal série, mais nous utiliserons X-CTU, celui-ci permettant également de configurer les XBee via leur firmware.

On commence par connecter un XBee au Pc via sa plateforme USB. On effectue ensuite un "Test/Query". L'onglet terminal permet ensuite d'envoyer les commandes de configuration.

Liste des commandes :

+++ (Pas de retour chariot) : passe en mode configuration
ATxx : consulte l'état xx
ATxx YY : fixe l'état xx à la valeur YY

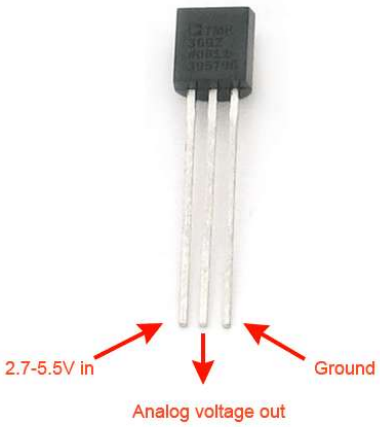
En mode configuration
ATID : l'adresse du réseau (la même pour les deux XBee)
ATMY : l'adresse du dispositif
ATDH : l'adresse du dispositif distant
ATDL : bits de poids faible de l'adresse distante, généralement 0.

ATND : recherche de nœuds distants. Si un autre XBee est bien configuré, la recherche de nœud renverra son adresse, sinon il ne renverra rien.
ATWR : enregistrement des paramètres dans la mémoire du XBee
ATCN : quitter le mode configuration. ATCN est automatiquement effectuée après 10 secondes d'inactivité.
Pour plus de commandes, voir l'Annexe 1.

Il est nécessaire de configurer les deux modules intelligemment, l'adresse MY de l'un est l'adresse DH de l'autre. Il faut aussi faire attention à la fréquence utilisée.

Une fois les XBee correctement configurés, ils peuvent communiquer entre eux (test ATND). On s'en sert normalement en utilisant le port série, cependant quelques conflits survenaient lors de l'utilisation des broches RX/TX de l'Arduino. Nous avons donc contourné ce problème on émulant un port série virtuel à l'aide de la librairie « soft serial ». Cette librairie permet d'émuler un port série sur n'importe quelle broche numérique de l'Arduino. On se sert alors des mêmes commandes (print, read, …) avec notre port série virtuel qu'avec le port série usuel. On peut ainsi utiliser la communication par Xbee sans risque de conflit.

Capteur de température



Pour relever la température à l'extérieur du satellite, nous utilisons un capteur TMP36 (Annexe 2). Son fonctionnement est très simple, après avoir alimenté le capteur, on peut lire une tension proportionnelle à la température sur sa 3ème broche. Un rapide calcul nous permet alors de déduire la température mesurée :

$$Température = ((V_{mesuré} * 0{,}004882)-0{,}5)*100$$

Capteur de pression

Nous pensions utiliser un capteur de pression en tant qu'altimètre afin de déterminer la hauteur du satellite par rapport au sol. Cela permettrait, par exemple, de déployer un parachute à une altitude choisit.
Le capteur choisit est un MPXHZ6400A (Annexe 3). Son boitier ne nous permettait pas de la testé sans soudure, donc sans conception de carte, cependant les calculs à programmés sont similaires à ceux du capteur de température.

Accéléromètre

Nous avons choisi d'intégrer à notre satellite un accéléromètre dans le but de pouvoir relever ses valeurs lors de la chute. Ce choix s'est avéré ne pas être le meilleur car la conversion accélération/vitesse est imprécise.

Nous avons utilisé un ADXL345 (Annexe 4) en communiquant via I²C, puis via SPI. Ce type de communication n'est cependant pas astucieux pour notre application, car nous utilisons des protocoles par bus, ou par maître/esclave, qui sont plus lourds à mettre en place et qui n'apportent pas vraiment d'avantages lorsque l'on a qu'un seul capteur de ce type.

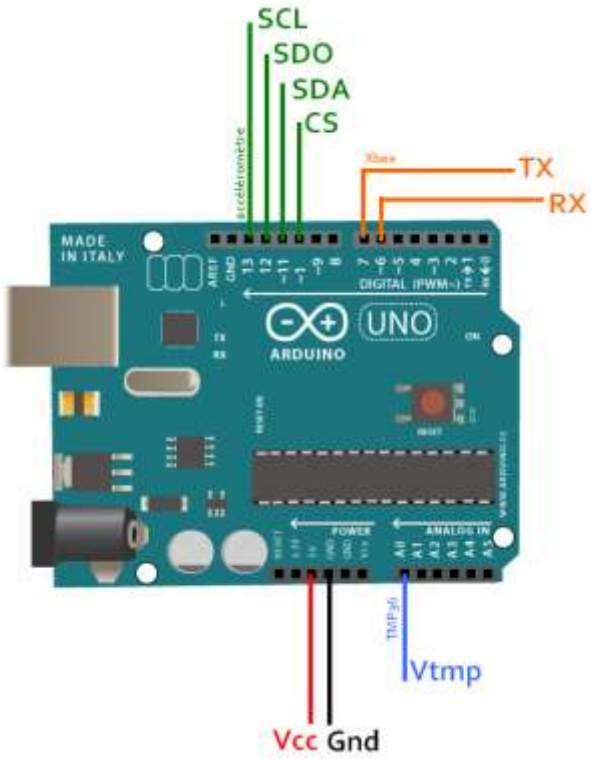
Le principal intérêt de l'accéléromètre est de nous fournir la vitesse de chute du satellite. En effet, dans les consignes de la compétition il nous ait demandé de relevé cette vitesse mais aussi de s'assurer qu'elle se trouve dans une certaine échelle de valeur.
Pour cela, il est nécessaire d'effectuer un calcul pour convertir les données brutes d'accélération en vitesse instantanée. Ce procédé est assez imprécis puisqu'il faut effectuer beaucoup de relevés d'accélération pour en déduire une vitesse correcte. De plus, la nouvelle vitesse est calculée à partir de l'ancienne valeur ce qui implique une accumulation d'erreurs. Cette méthode peut être utilisée pour de courtes chutes, mais les résultats deviendraient inexploitables pour de plus longues chutes.

Après réflexion, nous pensons qu'il serait plus judicieux d'utiliser l'altitude pour déduire la vitesse du satellite. En effet, en dérivant la position relevé et connaissant son point de départ et d'arrivé, la vitesse calculée serait plus précise. Cela permettrait aussi d'économiser un capteur plutôt coûteux.

Programmation

Programmation sur Arduino

Nous avons au départ programmé nos tests sur Arduino. C'est-à-dire, en utilisant le langage et les plateformes Arduino qui ont l'avantage d'être très simple d'utilisation, même pour les plus novices.
Notre programme de test du schéma global est disponible en Annexe 6 et dans l'archive de notre projet.
Ci-dessous la figure pour le branchement des capteurs sur les bonnes sorties. Il faut bien sur les alimenter séparément.



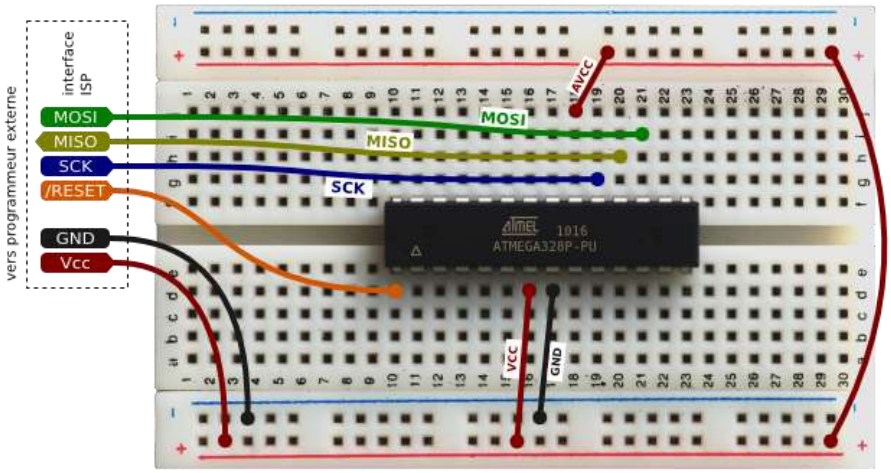
Informations complémentaires sur les plateformes Arduino sur le site : http://www.arduino.cc/

Ce programme peut être retranscris en langage C/C++ pour être moins volumineux ou pour une programmation plus classique du microprocesseur.

Programmation sur ATMEGA328p

Afin minimiser la place et le poids de la carte électronique, nous pensions implémenter l'ATMEGA 328p directement sur notre carte, sans plateforme Arduino. Dans le cas d'une telle solution, il n'aurait pas été pratique de garder la programmation via Arduino. En effet il faudrait mettre le processeur sur l'Arduino avant chaque programmation, puis le déplacer vers la carte. Pour chaque modification du programme il y aurait eu un risque d'endommager l'ATMEGA 328p.

Une des solutions est donc de programmer le microprocesseur directement sur la carte. On utilise pour cela un programmateur externe tel qu'un ARVisp mkII d'Atmel. Il suffit de connecter le microprocesseur comme indiqué sur la figure ci-dessous.

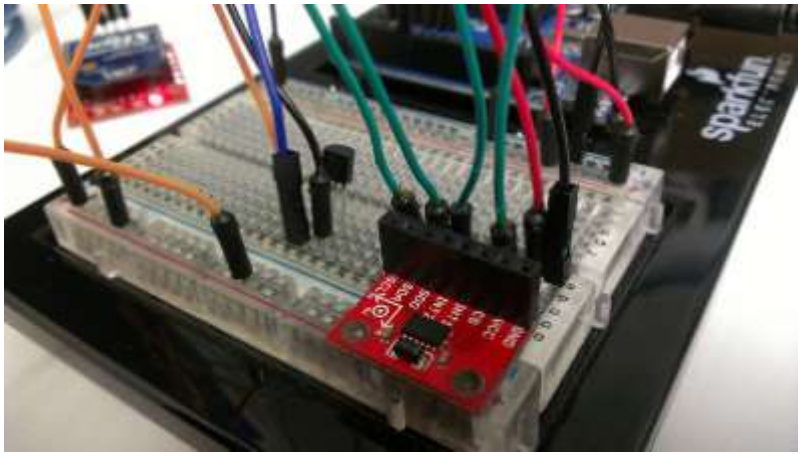


Bien que le principe soit simple, nous n'avons pas réussi à tester le programmateur externe, à cause d'un certain nombre de conflits avec le « avrdude ». Nous pensons que ce problème peut être résolu en « burnant le bootloader » de l'ATMEGA328p au préalable. Cette opération à la particularité d'être faisable avec un logiciel tel qu'AVR-Studio , ou bien d'être également via ISP à l'aide d'une autre plateforme Arduino.

Réalisation de la carte

Phase de tests préalables

Avant de concevoir la carte électronique, il a fallu tester tous les capteurs un à un et finalement le montage dans son ensemble (figure ci-dessous).
Cette phase permet à la fois d'optimiser le programme mais aussi e visualiser les besoins physiques de la carte électronique ensuite.



Ensuite, si nous avions réussis à programmer l'ATmega328P, nous aurions pu effectuer un montage de test avec le micro contrôleur en suivant le même brochage que sur l'Arduino (voir Annexe 5).

Phase de définition

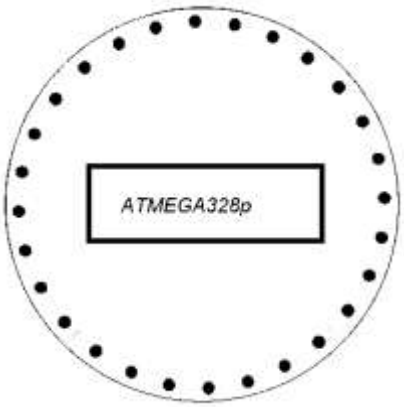
Une fois les essais sur plaque de tests effectués il faut penser à l'architecture de la carte. C'est-à-dire, visualiser le nombre de cartes à réaliser et les composants nécessaires à leur élaboration.
Nous avons décidé de réaliser deux cartes rondes :

1/ Une carte réutilisable et adaptable

La carte principale réunit le microprocesseur et les modules Zigbee. Cela permet d'avoir les éléments nécessaires chaque année au projet sur une unique carte.
En tirant des pistes pour chacune des sorties de l'ATMEGA, nous avons ainsi une carte modulable et sur laquelle nous pouvons brancher d'autres cartes avec de nombreux capteurs.
La gestion de l'alimentation s'effectue aussi sur cette carte avec un régulateur de tension en amont du montage. L'alimentation des autres composants est effectuée grâce aux sorties de l'ATMEGA.

2/ Une carte attribuée à la mission

La seconde carte regroupe l'ensemble des capteurs. Elle est reliée à la première via un cercle de connecteurs.

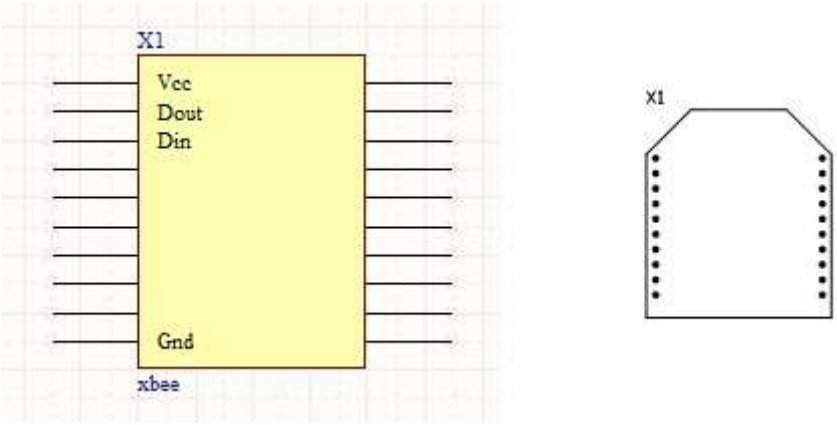


Nous pensons que relier chacune des sorties à un connecteur sur le rebord de la carte est une configuration permettant une grande modularité.

Phase de CAO

Pour créer le schematic et le PCB de la carte, nous utilisons le logiciel Altium Designer, présent à Polytech'Lille.
Les composants que nous utilisons n'existe pas sur ce logiciel, il a donc fallu créer une bibliothèque de composants, disponible dans l'archive de notre projet (lien disponible sur notre page wiki : http://projets-ima.plil.net/mediawiki/index.php?title=Projet_CanSat).

La figure ci-dessous est un exemple avec le Xbee :



Seules les broches que nous utilisons ont été configurées.

Phase d’assemblage

Malheureusement, nous n’avons pas pu réaliser de carte. Beaucoup de détails étaient manquant pour faire tirer la carte et la machine est tombée en panne lorsque nous aurions pu finaliser le typon.
Cette étape n’est pas la plus difficile puisque beaucoup de contraintes et de réflexions ont déjà été éliminées. Cependant, rien ne nous assure que les phases de tests auraient immédiatement bien fonctionnées, au contraire, il aurait sans doute fallu faire les cartes une seconde fois afin de corriger des erreurs grossières.

Conclusion

Le prototype

Notre sonde n'a malheureusement pas pu être lancée en chute libre.

Nous pensons que le sondage atmosphérique obligatoire a été réalisé au cours de ce projet (envoie de la température et de la vitesse). L'étape de réalisation de la carte peut être finalisée en quelques heures.

Cependant nos analyses et erreurs pourront servir à une équipe qui souhaiterait participer à la compétition l'année prochaine.

Le projet

Avec du recul, nous pouvons dire qu'il y a eu quelques lacunes du côté gestion de projet. Ce que nous avons réalisé nous a beaucoup aidé pour nous préparer à de futurs projets, cependant le résultat n'est pas celui escompté.

Nous avions beaucoup de projets et d'idées en tête pour la réalisation de ce projet et très peu ont pu voir le jour. L'utilisation d'un outil de gestion de projet n'aurait pas été de trop afin de se cadrer.

La configuration du Xbee, les codes pour les capteurs et la bibliothèque de composants pour Altium designer pourront être réutilisés dans d'autres projets, ce sont des composants communs au sein des projets du département IMA et nous espérons que ce travail sera utile à d'autres.

Finalement, ce projet était très intéressant et nous sommes déçus de ne pas l'avoir exploité comme nous l'espérions. Nous pensons voir avec le club Robotech pour fonder une équipe pour la compétition 2014.

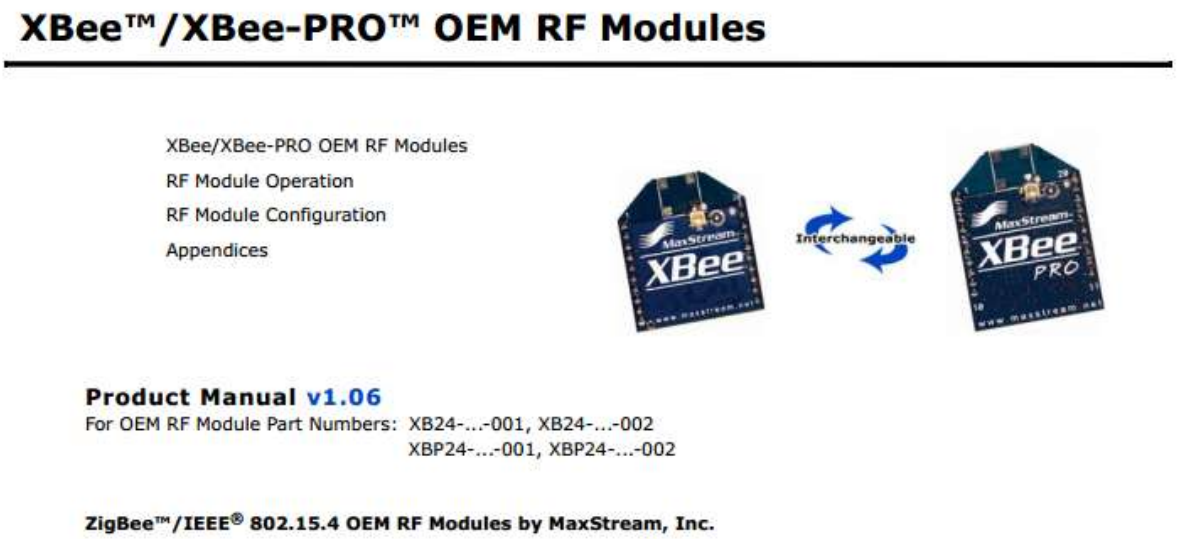
Bibliographie

http://www.cnes-jeunes.fr/
http://www.cansatcompetition.com/Main.html
http://www.planete-sciences.org/espace/-CanSat-
https://fr.wikipedia.org/wiki/CanSat
http://www.chicoree.fr/w/Arduino_sans_Arduino
http://www.chicoree.fr/w/Arduino_sur_ATmega328P
https://www.sparkfun.com/tutorials/240
http://www.arduino.cc/fr/
http://bildr.org/2012/03/stable-orientation-digital-imu-6dof-arduino/

Annexes

Annexe 1: Xbee/Xbee Pro

Datasheet: http://www.picaxe.com/docs/xbe001.pdf



XBee™/XBee-PRO™ OEM RF Modules

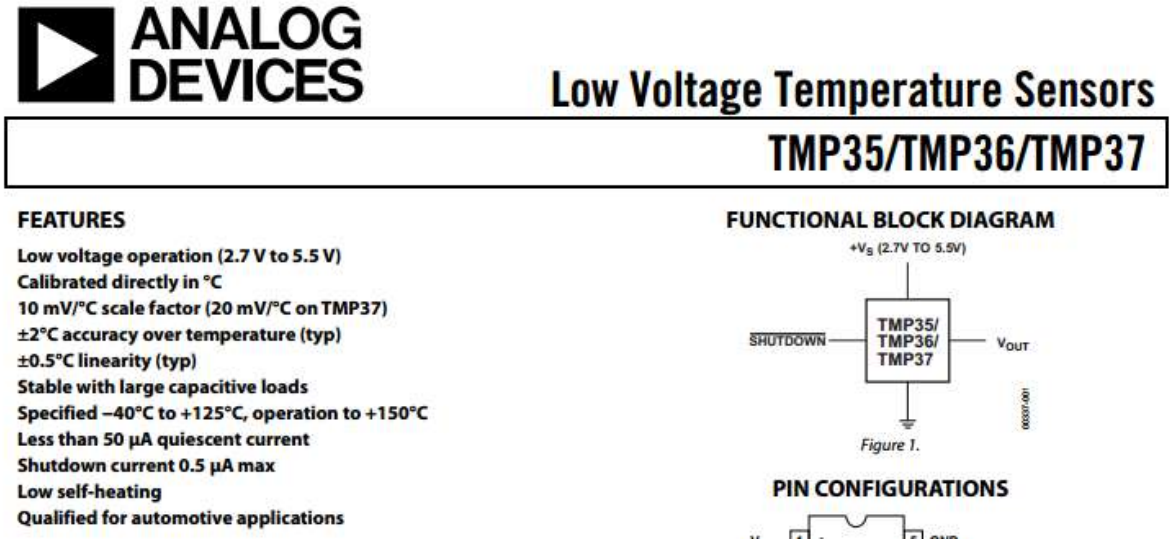
XBee/XBee-PRO OEM RF Modules
RF Module Operation
RF Module Configuration
Appendices

Product Manual v1.06
For OEM RF Module Part Numbers: XB24-...-001, XB24-...-002
XBP24-...-001, XBP24-...-002

ZigBee™/IEEE® 802.15.4 OEM RF Modules by MaxStream, Inc.

Annexe 2: TMP36

Datasheet: http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf



ANALOG DEVICES

Low Voltage Temperature Sensors

TMP35/TMP36/TMP37

FEATURES

- Low voltage operation (2.7 V to 5.5 V)
- Calibrated directly in °C
- 10 mV/°C scale factor (20 mV/°C on TMP37)
- ±2°C accuracy over temperature (typ)
- ±0.5°C linearity (typ)
- Stable with large capacitive loads
- Specified −40°C to +125°C, operation to +150°C
- Less than 50 µA quiescent current
- Shutdown current 0.5 µA max
- Low self-heating
- Qualified for automotive applications

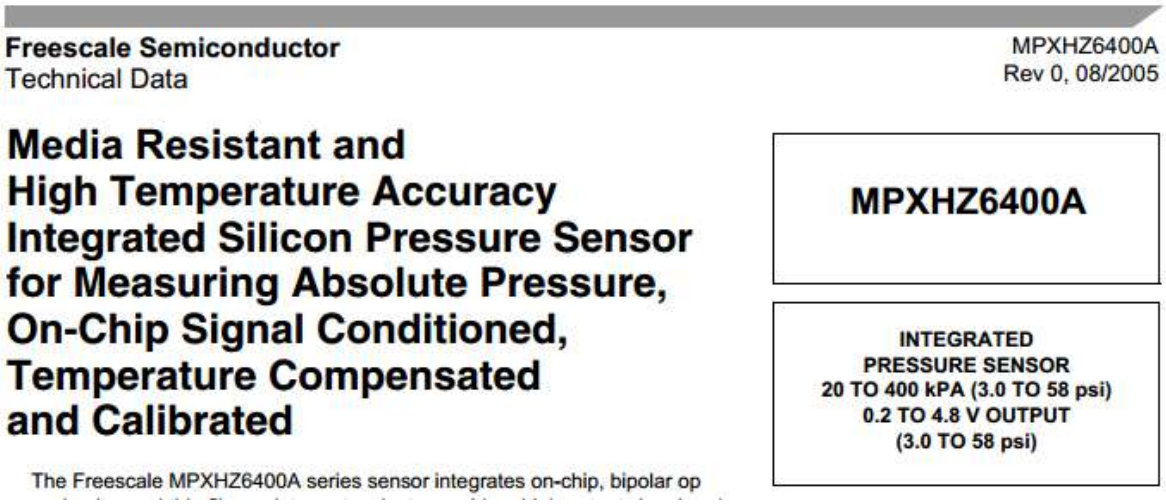
FUNCTIONAL BLOCK DIAGRAM

Figure 1.

PIN CONFIGURATIONS

Annexe 3: MPXHZ6400A

Datasheet: http://www.freescale.com/files/sensors/doc/data_sheet/MPXHZ6400A.pdf



Freescale Semiconductor
Technical Data

MPXHZ6400A
Rev 0, 08/2005

Media Resistant and High Temperature Accuracy Integrated Silicon Pressure Sensor for Measuring Absolute Pressure, On-Chip Signal Conditioned, Temperature Compensated and Calibrated

The Freescale MPXHZ6400A series sensor integrates on-chip, bipolar op

MPXHZ6400A

INTEGRATED PRESSURE SENSOR
20 TO 400 kPA (3.0 TO 58 psi)
0.2 TO 4.8 V OUTPUT
(3.0 TO 58 psi)

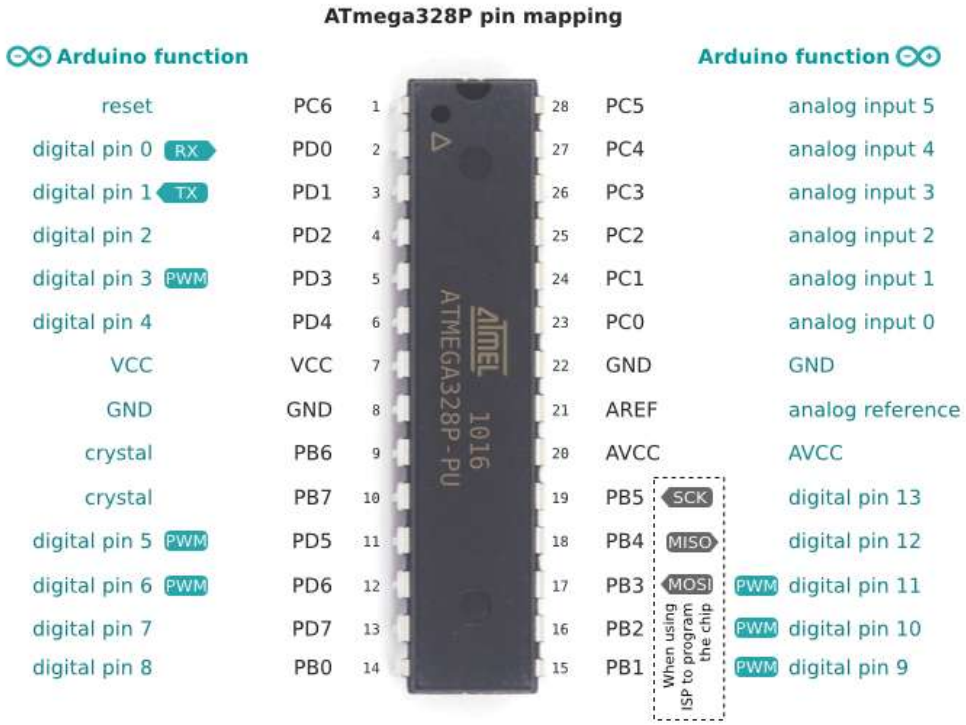
Annexe 4: ADXL 345

Datasheet: https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf
Carte Sparkfun : https://www.sparkfun.com/products/9836

L'ADXL345 est intégré sur une carte Sparkfun (figure à gauche).

Annexe 5: ATMEGA328p

Datasheet : http://www.atmel.com/Images/doc8161.pdf
Figure ci-dessous : Pin mapping de l'ATmega328p en parallèle avec les fonctions arduino



Annexe 6 : Programme de tests CANSAT.ino

```
/***** ZIGBEE *****/
//Librairie pour utiliser le port série virtuel
#include <SoftwareSerial.h>

// Software Serial RX
const int rxpin = 6;
// Software Serial TX
const int txpin = 7;
SoftwareSerial mySerial(rxpin, txpin);


/****** TEMPERATURE *****/
// Temperature on pin A0
int temperaturePin = 0;
int mesure = 0;


/***** ACCELEROMETRE *****/
//Librairie pour la connexion en SPI
#include <SPI.h>

//Definition de l'adresse de l'accelerometre
#define Address 0x53
```

```
//Definition des registres de l'accelerometre
#define ADXL345_POWER_CTL 0x2D
#define ADXL345_DATA_FORMAT 0x31
#define ADXL345_DATAX0 0x32
#define ADXL345_DATAX1 0x33
#define ADXL345_DATAY0 0x34
#define ADXL345_DATAY1 0x35
#define ADXL345_DATAZ0 0x36
#define ADXL345_DATAZ1 0x37


void ecritureSPI(byte registre, byte data)
{
  digitalWrite(10,LOW);
  SPI.transfer(registre);
  SPI.transfer(data);
  digitalWrite(10,HIGH);
}

byte lectureSPI(byte registre)
{
  digitalWrite(10,LOW);
  SPI.transfer(0x80 | registre);
  byte data = SPI.transfer(0xFF);
  digitalWrite(10,HIGH);

  return data;
}

void valeurs_acc(int *x, int *y, int *z)
{
  *x =(int) ((lectureSPI(ADXL345_DATAX1)<<8)|lectureSPI(ADXL345_DATAX0))+511;
  *y =(int) ((lectureSPI(ADXL345_DATAY1)<<8)|lectureSPI(ADXL345_DATAY0))+511;
  *z =(int) ((lectureSPI(ADXL345_DATAZ1)<<8)|lectureSPI(ADXL345_DATAZ0))+511;
}

// Convertit la valeur de la tension du capteur en température
float tmp_convert(int pin)
{
 float temperature;
 temperature = analogRead(pin) * .004882814;
 temperature = (temperature - 0.5) * 100;
 return temperature;
}

void aff_tmp(void) {
  if(mesure==10) {
  float temperature;
  temperature = tmp_convert(temperaturePin);
  mySerial.println(temperature);
  mesure=0;
  }
  else mesure++;
}

void aff_acc(void) {
  int x, y, z;
  valeurs_acc(&x, &y, &z);
```

```
  Serial.print("X : ");
  Serial.println(x);
  Serial.print("Y : ");
  Serial.println(y);
  Serial.print("Z : ");
  Serial.println(z);
  Serial.println("");

  mySerial.print("X : ");
  mySerial.println(x);
  mySerial.print("Y : ");
  mySerial.println(y);
  mySerial.print("Z : ");
  mySerial.println(z);
  mySerial.println("");
}


void setup()
{
  Serial.begin(9600);
  // Initialisation du Software Serial
  mySerial.begin(9600);
  mySerial.println("Debut du programme");

  pinMode(10,OUTPUT);          //CS
  pinMode(11,OUTPUT);          //SDA
  pinMode(12,INPUT);           //SDO
  pinMode(13,OUTPUT);          //SCL

  SPI.begin();
  SPI.setDataMode(SPI_MODE3);
  ecritureSPI(ADXL345_DATA_FORMAT, 0x01);
  ecritureSPI(ADXL345_POWER_CTL, 0x08);
}


void loop()
{
  aff_acc();
  aff_tmp();
  delay(500);
}
```