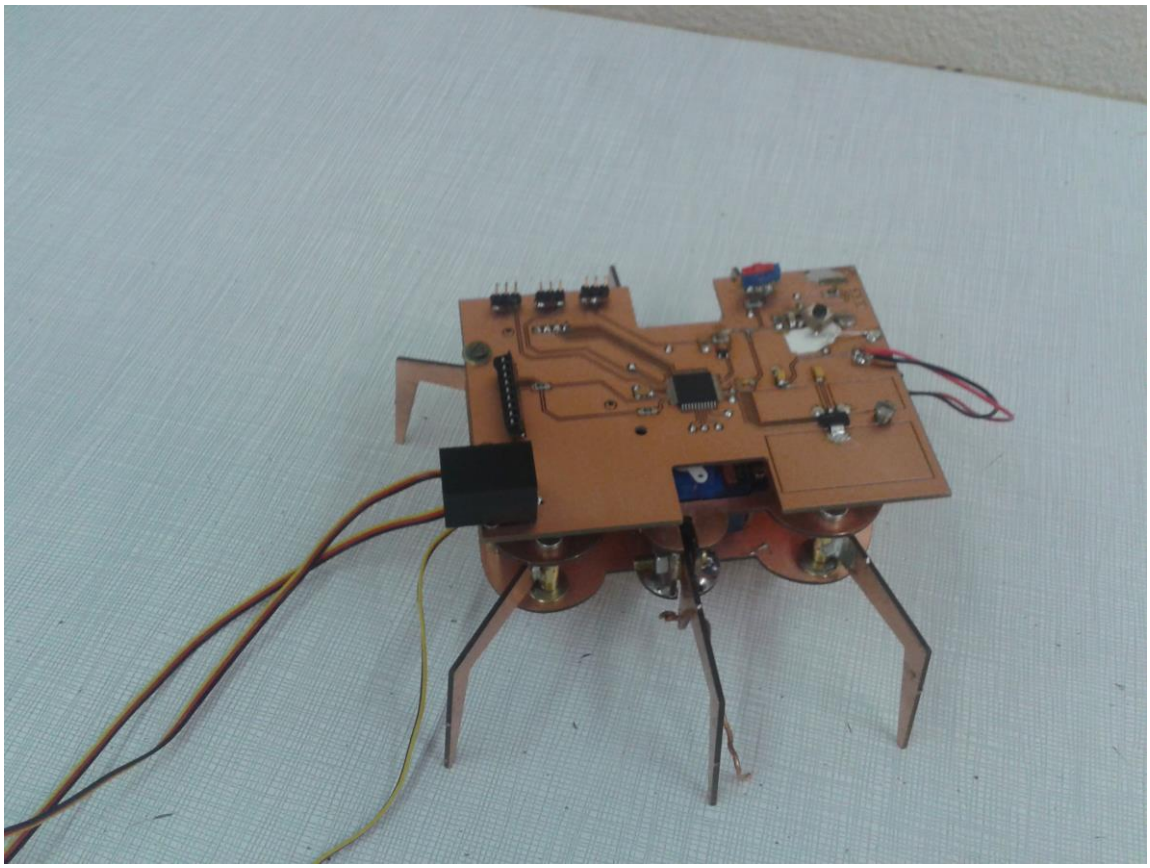
Aix*Marseille université

Département GEII Marseille

Année universitaire : 2012-2013

Conception d'un robot fourmi



Alexis CAU

Stage réalisé du 29 avril au 21 juin
À l'IUT d'Aix-Marseille, département GEII,
28, traverse Charles Susini
Marseille

Tuteur de l'organisme : M. AUBÉPART Fabrice

Tuteur de L'IUT : M. WAGNON Antoine

Remerciements

Je tiens tout d'abord à remercier M. Fabrice AUBEPART, enseignant-chercheur à l'IUT de Marseille pour m'avoir accepté comme stagiaire et pour m'avoir encadré.

Je remercie les techniciens du département Geii, Arnaud DURAND et Georges NOUARI, pour m'avoir conseillé et aider lors de la conception et de la fabrication de la carte électronique.
Par ailleurs, les conseils de mon professeur M. GUMUCHIAN m'ont été bénéfiques autant pour le stage que pour ma vie future et je tiens à l'en remercier. Ainsi que Mr Wagnon pour m'avoir apporté des conseils sur la réalisation du rapport.

L'ambiance était chaleureuse, et le travail intéressant et instructif.

Sommaire

Introduction

Dans le cadre de ma deuxième année de formation au sein du département GEII (Génie Electrique et Informatique Industrielle) de l'IUT d'Aix-Marseille, j'ai effectué un stage en entreprise.

Ce stage d'une durée de huit semaines, du 29 avril au 21 juin 2013, s'est déroulé directement dans le département GEII.

Ce stage possédait trois objectifs. Il m'a permis de tester dans le monde professionnel ce que l'on m'a appris à l'IUT, d'apprendre de nouveaux éléments et de travailler directement sur un projet concret.

Ce rapport est une synthèse de mon travail pendant ces 8 semaines de mon stage. Dans une première partie je présenterais l'organisme d'accueil. Ensuite, j'expliquerais la fabrication mécanique tout comme la réalisation de la carte électronique dont l'élément principal est un microcontrôleur Pic18 crée par la société Microchip. A la suite, je présenterais aussi les programmes conçus enfin de permettre le déplacement du robot.

I. Présentation

I.1 L'entreprise

L'institut Universitaire Technologique de Marseille a ouvert ses portes en 1969 comme élément de l'Université Aix-Marseille III qui deviendra en 2004 l'université Paul Cézanne.

En 2007, les trois universités des régions d'Aix-Marseille qui étaient l'Université de Provence, l'Université de la Méditerranée et l'Université Paul Cézanne, firent une demande de regroupement en Pôle de Recherche et d'Enseignement Supérieur (PRES). Ce processus s'acheva le 1er janvier 2012 avec la création d'une université unique nommée « Aix-Marseille Université (AMU) ».

L'IUT d'Aix-Marseille a été créé le 1er janvier 2013 par la fusion d'Aix-en-Provence et de Marseille ainsi que des départements délocalisé à Salon Provence, Gap, Arles et Digne. L'IUT d'Aix-Marseille est implanté dans sept villes et est devenue le plus gros IUT de France avec 23 diplômes universitaires de technologie (DUT) et 45 licences professionnelles.

Le site de Marseille (quartier de St Jérôme) regroupe plusieurs domaines de formations technologiques organisés en départements : Génie chimique (CH), Gestion des Entreprises et des Administrations (GEA), Génie thermiques et énergie (GTE), techniques de commercialisation (TC), mesures physiques (MP) et génie électrique et informatique industrielle (GEII).

L'IUT GEII propose des formations initiales et par alternances pour le DUT et ces trois licences professionnelles.

Institut universitaire St Jérôme

1 : DUT Génie Electrique et Informatique Industrielle

2 : - DUT Gestion des Entreprises
- DUT Technique de Commercialisation

3 : DUT Mesure Physique

4 : DUT Génie Thermique et Energie

5 : DUT génie Chimique

6 : Administration et amphithéâtre

7 : Cafétéria

I.2 L'objectif du stage

L'objet du stage est la création d'un prototype de robot hexapode qui doit se diriger en direction du Nord magnétique – en utilisant le champ magnétique terrestre - tout en évitant des obstacles. Par ailleurs, il est équipé d'un système de communication sans fil (module Xbee) qui lui permet de recevoir des commandes ou d'envoyer des informations sur un poste informatique distant.
Des robots construits sur le plan de ce prototype, pourront alors être utilisés dans les travaux pratiques en informatique industrielle pour les étudiants de première année par apprentissage du département GEII.
L'objectif de ce stage est de rendre les travaux pratiques de programmation en mplab, ludique, interactif et amusant pour les premières années.

Enfin de répondre à cet objectif, le tuteur de l'organisme à poser ses demandes qui sont :

- La création et montage des pièces mécaniques
- La fabrication de la carte électronique
- Le montage des composants
- La programmation des mouvements du robot
- gestion des déplacements du robot grâce à la connexion sans fil
- La programmation du capteur magnétique par liaison I2C
- La programmation du détecteur d'obstacle avec sortie analogique

Au fil de l'évolution du projet des contraintes sont apparues. Tel que :

- Le coût
- Les délais de livraisons de certains composants
- L'erreur humaine

Grâce à ces données et les informations transmises par mon tuteur de stage, j'ai pu fournir le cahier des charges ci–dessous :

a) Le support mécanique

La structure utilisée est un élément que nous avons pu récupérer d'un site internet. Ce montage a été réalisé par un professeur de lycée.
(**http://innovelectronique.fr/2009/12/08/robot-hexapode-robot-fourmi/**)

b) Carte électronique

- Afin de permettre la réalisation de la carte, nous avons utilisé un circuit imprimé à double face. Celui sera placé comme troisième étage de la structure mécanique

- Le Microprocesseur est le PIC18f4550 TQFP de Microchip. Le microcontrôleur a été imposé par le moniteur : c'est le microcontrôleur utilisé dans les TP de 1ère année au département GEII.

PIC18f4550 TQFP

- La fréquence d'oscillation du microcontrôleur sera définie avec l'horloge externe, réalisé avec le circuit LMTC6900. La fréquence a été fixée à 4Mhz.

- Un capteur magnétique CMPS03 à effet Hall. Il sera relié au pic18 par interface I2C

- Un détecteur d'obstacle GMPY02F de SHARP, il sera connecté à une branche du convertisseur analogique du microcontrôleur.

- Des servomoteurs miniatures HT-55s de Hitec.

- Un module Xbee qui permettra la connexion du pic18 et d'un ordinateur distant par liaison série asynchrone.

- L'utilisation de composants CMS (composant monté sur surface)

- Une pile de 9 V sera utilisée pour alimenter la carte électronique. Cependant les composants de la carte électronique ont besoin d'une tension de 5 V. C'est pour cela que nous utiliserons un régulateur de 5 V en sortie.

c) Programmation

- Programmes écrits en langage C et compatibles avec le compilateur C18
- Utilisation de l'environnement MPLAB
- Aucune contrainte imposée concernant la stratégie de programmation pour le déplacement du robot.

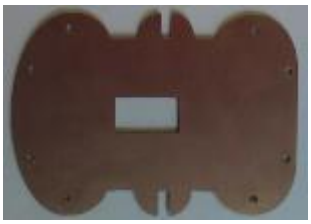
- En revanche, les programmes être réutilisables. Ils devront être correctement commentés.

II La fabrication du robot

II.1 La partie mécanique

La partie mécanique du robot est réalisée à partir d'une structure trouvée sur :
http://innovelectronique.fr/2009/12/08/robot-hexapode-robot-fourmi/

Le robot sera composé de deux supports (haut et bas) et de 6 pattes dont les pattes centrales sont différentes des quatre autres pattes.



Support bas

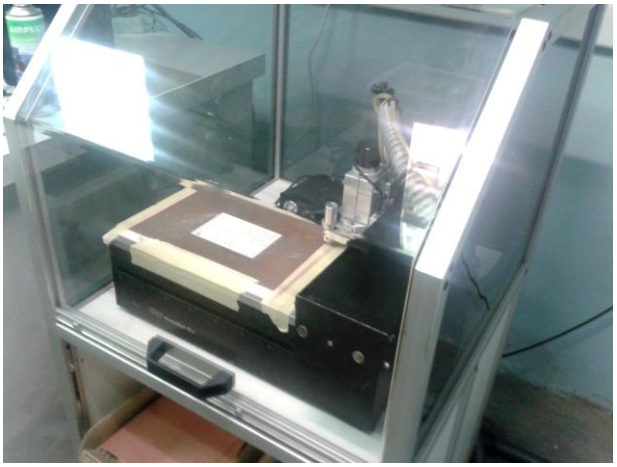
Support haut

Pattes centrales

Pattes avant et arrière

Afin de créer ces éléments, on a utilisé trois logiciels nommés : proteus, Circuitcam et boardmaster pour lancer la machine à graver. Les schémas doivent être de type excellon pour le perçage et gerber pour les contours de la plaque. Proteus crée la forme de la carte, Boardmaster lance l'automate tandis que Circuitcam indique les endroits qui seront percés où coupés.

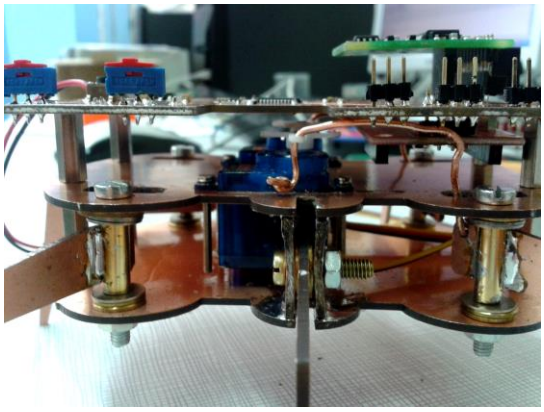
C'est la machine à graver « UPKF protomate 91s » qui permet la réalisation des cartes électroniques et des supports mécaniques ci-dessus.



Machine à graver du département GEII

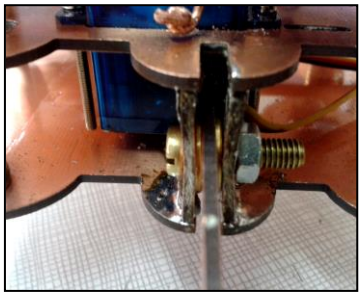
Afin de permettre la création de nos supports, nous utiliserons une plaque en double face.

Les supports haut et bas ont été gommés et vernis afin de faciliter le soudage. Ces supports seront après reliés par des entretoises de 20 cm pour permettre la mise en place des servomoteurs. Les pattes seront toutes reliées au servomoteur par des câbles.
Les pattes non centrales seront soudées à une entretoise qui leurs permettra alors d'avoir un mouvement de pivot. Tandis que les pattes centrales seront visées à des supports soudés verticalement pour permettre le maintien de ces pattes.
Afin d'égaliser la hauteur vis-à-vis du sol et pour éviter le jeu des pattes, on a placé des rondelles de chaque côté.

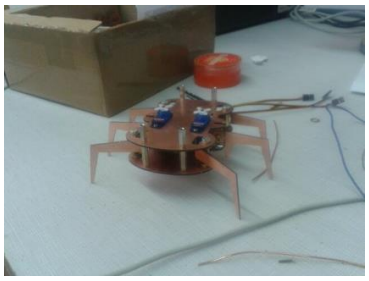


Robots sur le côté

Le robot sera monté ainsi sur 3 étages. Chaque étage possède un rôle particulier. Sur celui du bas, nous encastrerons le servomoteur pour les pattes centrales, celui du milieu maintiendra les pattes gauches et droites et sur celui du haut sera placée la carte électronique.
Les vis utilisées sont majoritairement des vis de 4mm en diamètre. Cependant la difficulté fût de trouver des vis de différentes longueurs. Pour cela une armoire fût mise à disposition avec toutes sortes de vis, écrous et rondelles.
Afin de finir la partie mécanique d'un robot, il a fallu 36 rondelles, 6 écrous et 6 vis.
Ces 36 rondelle ont servi à réduire le jeu des pattes afin d'éviter des mouvements non voulues.
Pour chaque patte avant et arrière, il faut 3 rondelles en haut et en bas. Ce système permet de maintenir une hauteur égale et une stabilité correcte sur le sol. Cependant la rondelle, juste avant et après l'entretoise, devra posséder un diamètre interne égal à 4 mm.
De plus pour éviter tout mouvement horizontal des pattes centrales, c'est le même système fût utilisé avec la contrainte des diamètres en moins.



Patte Centrale



Partie mécanique finie

Les problèmes rencontrés dans le montage mécanique furent surtout au niveau des perçages et des soudures.

II.2 La partie électronique

a) Schéma fonctionnel

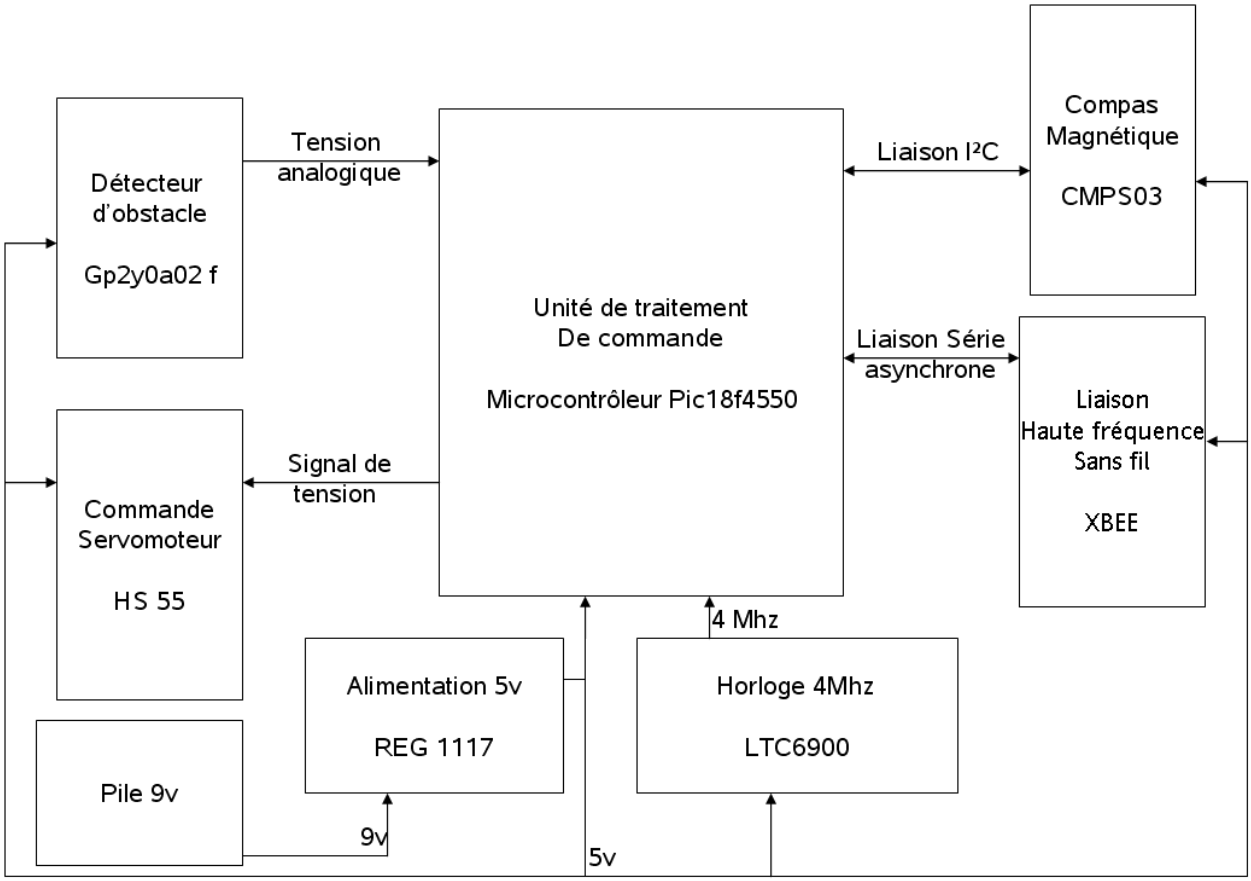
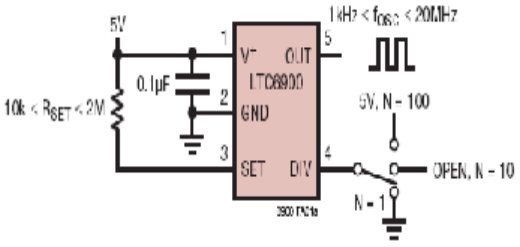


Schéma fonctionnel

L'horloge est réalisée à partir du circuit intégré LTC6900 (de linear Technology). Il a été choisi pour ses faibles dimensions. La fréquence peut être réglée grâce au choix d'une valeur de résistance et du paramètre N qui peut être choisi suivant le câblage.

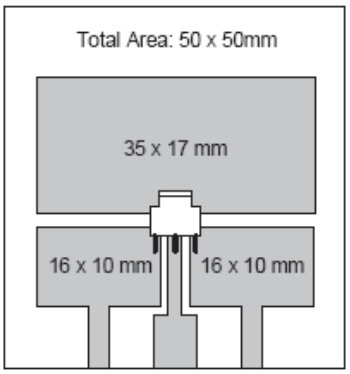
La valeur de RSET à un minimum fixé qui est 10kΩ, N peut prendre les valeurs 1, 10 et 100 et l'on m'a imposé une valeur Fosc de fréquence de 4Mhz. A partir de ces données et en utilisant la relation suivante, bous calculons RSET.

*RSET= (10 MHz * 20 k)/ (N*Fosc)*
*RSET= (20k*10)/ (4N)=*
RSET=50k/N



Câblage LTC6900

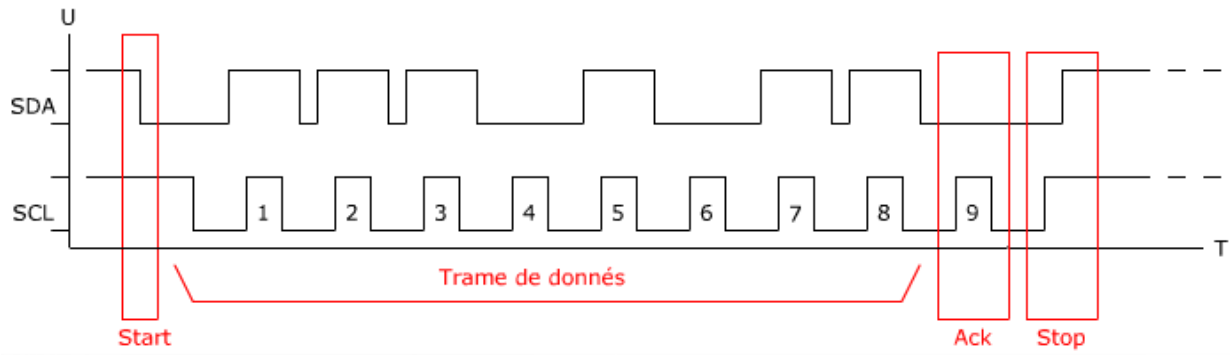
L'alimentation de toute la fonction a été faite à 5 v Celle-ci à été gérée par un régulateur REG1117 en boîtier SOC de TEXAS INSTRUMENTS. J'ai utilisé les données de la documentation spécifique pour créer son empreinte en tenant compte d'une zone de dissipation thermique.



Zone de dissipation thermique

Le compas de champs magnétique a été réalisé par un module intégré cmps03. On utilisera la liaison I²C du capteur pour permettre le déplacer le robot en fonction du champ magnétique.

Il faut envoyer une condition de départ pour initialiser le support I²C. Lorsque les esclaves ont reçu la condition de départ, ils envoient un ack (acquittement) au maître. Après le maître continu la discussion soit en écriture soit en lecture grâce à un signal de donnée et un signal d'horloge. Toutes les transactions possèdent un ack.



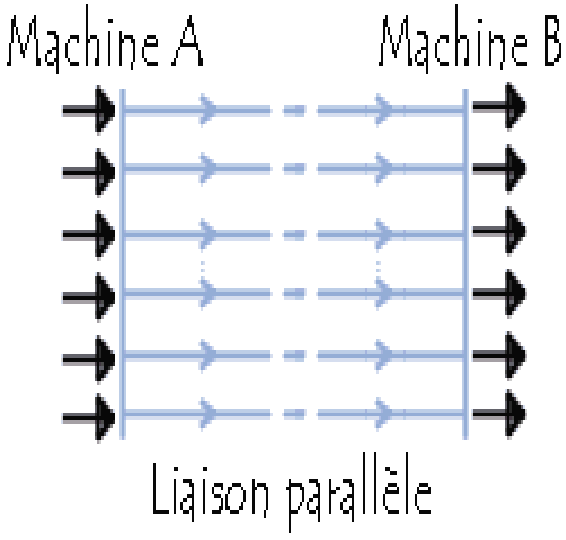
Exemple de Trame de communication

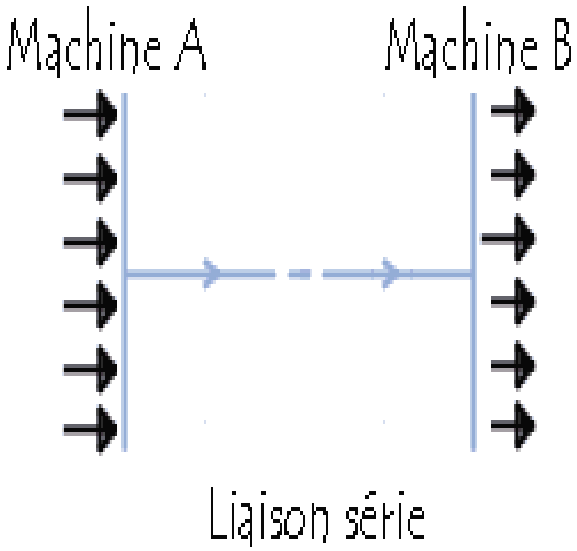
La séquence se décompose ainsi :
- Condition de départ
- Emission de l'adresse esclave, soit de L'EEPROM.
- Acquittement de la part de l'esclave
- Emission par le maître de l'adresse mémoire à laquelle il souhaite inscrire les futures données, acquittement successif de l'esclave
- Transmission des données à écrire, acquittement à chaque réception de paquet par l'esclave
- Condition de stop

Le SDA est le signal de donnée tandis que le SCL est le signal de l'horloge. La première permet de dire avec quel esclave le maître discutera et quel sera son rôle (lire ou écrire).

Le détecteur d'obstacle sera réalisé par un GP2Y0A02F, capteur de distance à infrarouge. On utilisera une entrée du convertisseur analogique du pic pour ce capteur.

La liaison haute fréquence est assurée par deux modules Xbee dont un connecté à l'ordinateur. On utilisera une liaison série asynchrone pour cette fonction. Dans cette liaison on peut envoyer un bit à n'importe quel moment. Cependant le récepteur ne pourra pas dire la forme de l'octet. Pour éviter ce problème l'émetteur devra envoyer un bit de Start et bit de stop



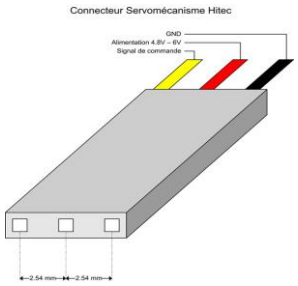


Il existe deux différents types de liaison. La liaison en parallèle qui permet d'envoyer plusieurs bit à la suite mais qui perd de la qualité de signal en haute fréquence, et la liaison série qui est moins rapide que la liaison en parallèle mais qui peut envoyer des bits en Haute fréquence avec une meilleure qualité. Ces liaisons peuvent être transformées, grâce à une horloge et des décalages de registres. Dans notre cas on utilisera une sous espèce de la liaison série.
La liaison série possède deux sous catégories, la liaison série synchrone et la liaison série asynchrone. Dans une liaison série synchrone, le récepteur et l'émetteur sont cadencés à la même horloge. Cependant il peut y avoir des différences de fréquence entre l'émetteur et le récepteur qui peuvent empêcher la lecture du bit par le récepteur. Pour éviter ce problème le temps d'envoie doit être assez long pour que le récepteur l'ai reçu.

Tandis que pour la liaison série asynchrone, l'émetteur et le récepteur ne possède pas la même horloge. Afin de parer à ce problème, l'émetteur envoie un bit de start et de stop afin de commencer une conversation avec le récepteur. Sinon le récepteur ne pourrait pas identifier la place du bit dans l'octet et ainsi faussé le message.
Cette différence entre la liaison série synchrone et asynchrone

Une pile de 9 v sera branchée par deux pins et un cordon d'alimentation à la carte.

La commande des servomoteurs se fait par tension numérique. C'est le temps de mise à 1 dans une période de 20 millisecondes qui permet de régler l'angle de rotation (voir détail dans la partie programmation).



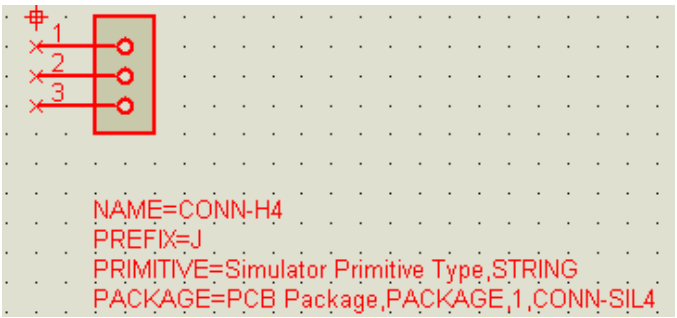
Connecteur HS 55

Photo servomoteur

Pour des applications futures, 1 bouton poussoir (pouvant servir de remise à zéro) et deux micros commutateurs ont été ajoutés à certaines entrées du bouton poussoir.

b) création de la carte électronique

La conception de la carte électronique fut réalisée en utilisant les logiciels ISIS et ARES.
Sur ISIS on crée le schéma électrique suivant les demandes et les caractéristiques des composants. Dans le cadre du stage, il fût nécessaire de créer les composants sur des bases existantes.
On d'abord utilisé cet interface de base pour la création de notre montage ISIS
Ce montage a été repris plusieurs fois à la demande du tuteur de stage. Ce montage n'est fixe que lorsque l'on pense que la carte sur ARES est finie.
Le principe a été de partir d'un montage réalisé autour d'un pic18f4550 que l'on a étudié au département GEII afin d'en modifier les éléments.
Pour mon étude, j'ai été obligée de créer des composants sur ISIS et ARES. Je vais donner en quelques lignes la technique qui permet de créer un composant. On commence par créer un rectangle et on lui ajoute autant de pattes que nécessaire.



Ecran isis création boîtier

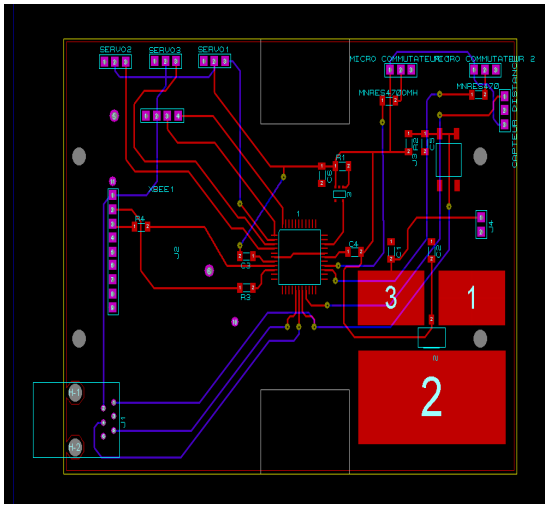
Lorsque l'on a fini de composer la forme du boîtier avec l'une des deux techniques ci-dessus, il faut créer le boitier. Lors de sa création, plusieurs options apparaissent tel que le nom de celui-ci, sa référence ou même la forme du boîtier.
La dernière option est nécessaire afin de donner une forme du composant sur ARES.

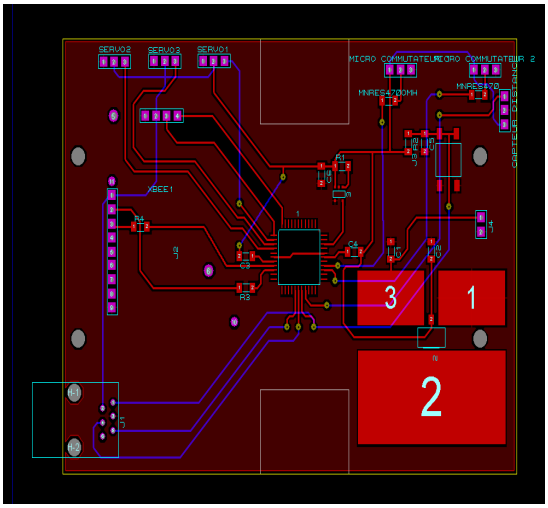
Ce sont les liaisons en vert qui permettent de modifier le fichier ARES.
Ces traits verts permettent de visualiser la liaison entre les composants. Le montage sera nécessaire lors du soudage des composants pour repérer les valeurs de ceux-ci et éviter toute erreur de mélange.

La seconde partie de proteus, est ARES. Ce logiciel crée le circuit imprimé suivant les liaisons établies sur le schéma d'ISIS.
Lorsque l'on place deux composants venant d'ISIS, des liaisons vertes apparaissent pour montrer le chemin que devront emprunter les futures pistes.
Une partie des empreintes de composant sur ARES furent également fabriquées. Pour cela, on a dû soit mesurer les composants à taille réelle soit utiliser les dimensions fournies dans leurs documentations afin de les saisir directement dans ARES.
Le carré bleu correspond à la forme du boitier. Cet élément doit être construis suivant les mesures du composant.

Un exemple de création de composant : le connecteur RJ11

Le circuit imprimé sera réalisé en deux couches recto verso. La face supérieure est nommée « Top » et la face inférieure, la couche « Bottom ». Dans le cas du top, tous les éléments que nous utiliserons seront en rouge. Tandis que le bottom sera en bleu.
Pour relier les composants nous utiliserons des pistes correspondant à la surface utilisée.
La dernière étape est de créer un plan de masse. Ce plan permet de mettre reliée tous les pistes à la masse sans avoir besoin de les liées signal sur toute la surface. Dans notre cas c'est la masse, avec ce plan toutes les liaisons faites à la masse peuvent être supprimées.

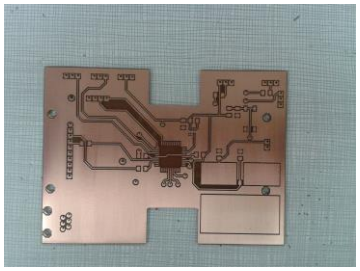


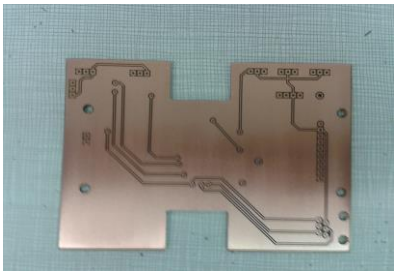


Carte sans plan de masse

Carte avec plan de masse

La partie grise de la carte électronique réelle et la partie noire de cette carte sur ARES signifie qu'il n'y a rien sur cette partie et que le plan de masse ne peut pas passer.

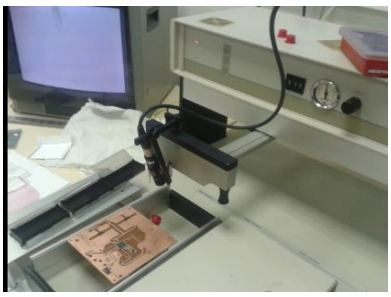




Carte vierge top

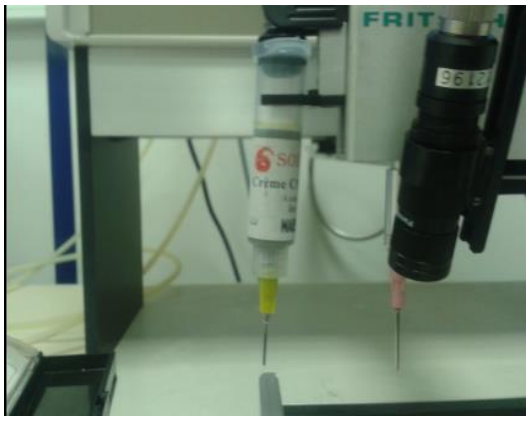
Carte vierge Bottom

A cette étape, on peut passer au soudage. La majorité des composants utilisés est en technologie CMS, il est donc déconseillé toute soudure avant de les avoir placé. Car celles-ci partiraient sous l'effet du four.
Pour souder ces composants il faut utiliser une machine et une crème d'étain spéciale.



Machine

Afin de poser la crème d'étain sur la plaque, il faut utiliser la machine FRITSCH LM 901. Afin de bouger la mèche et de capturer les composants, elle utilise de la pression. Ensuite on pose la carte sur un support que l'on resserre pour éviter de faire bouger celle-ci.



Bras de fonctionnement

Moniteur

On utilise la caméra pour voir où poser la crème. Lorsque celle-ci a été posée, on bouge la caméra pour voir la ventouse et on se déplace jusqu'au cms.

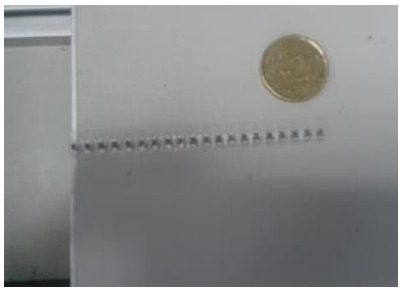


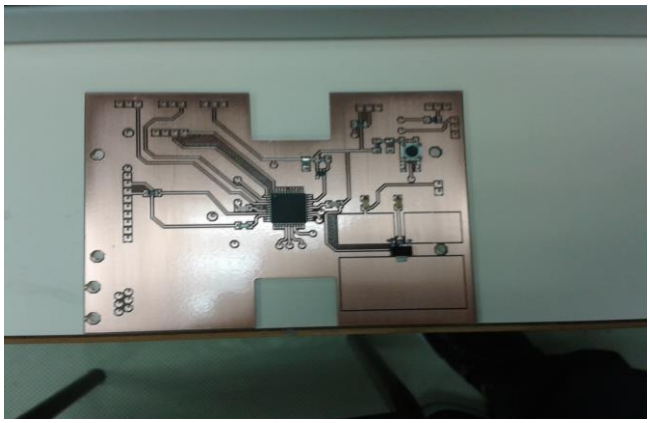
Photo cms

Lorsque l'on est dessus, on le prend avec la ventouse et on le pose là où ses soudures ont été faites. Il difficile de reprendre un composant après ça, car la crème peut faire effet ventouse lorsque l'on essaye de déplacer le composant.
Lorsque l'on a placé tous les composants cms, il faut vérifier qu'il n'y a pas d'étain qui relie deux pattes. Cette crème à la particularité, lorsqu'elle chauffe, de revenir vers le cuivre lorsqu'elle est étalée sur la partie grise. Cependant si elle est reliée entre deux pattes, elle ne pourra pas se retirer.

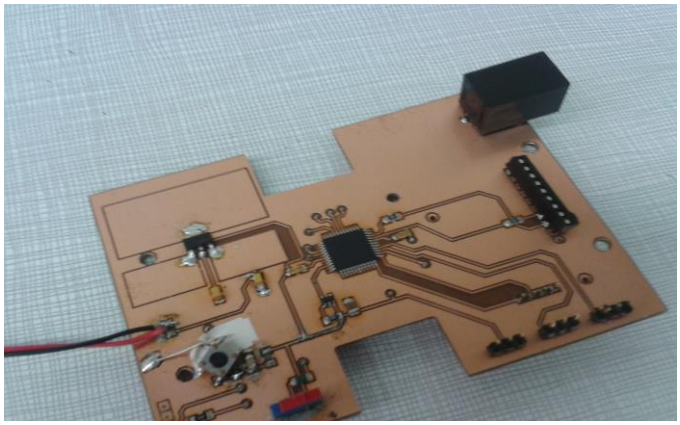
Four

Image four à haute température

Une fois les composants disposés sur la carte, il faut passer à l'étape de soudage. Celle-ci s'effectue en mettant la carte dans un four particulier dont la température est maintenue dans une première phase de chauffe à150°. La température s'élèvera alors dans une seconde phase jusqu'à 231°C. C'est à cette étape que la crème se liquéfie et devient de l'étain.



Carte après le passage au four



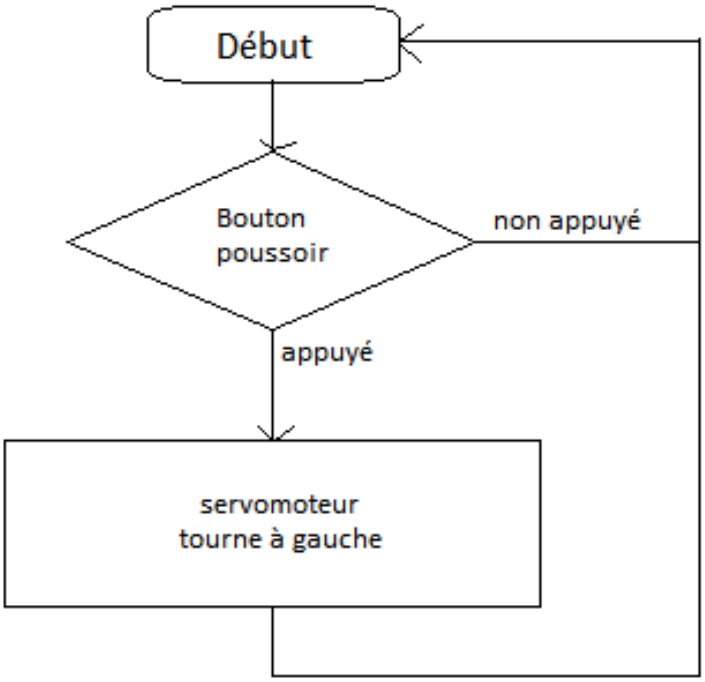
Carte électronique finale

II.3 La partie programmation

a) programmation des servomoteurs

La partie programmation s'est déroulée en plusieurs étapes. Afin de permettre la rotation des servomoteurs et la communication avec les différents modules.

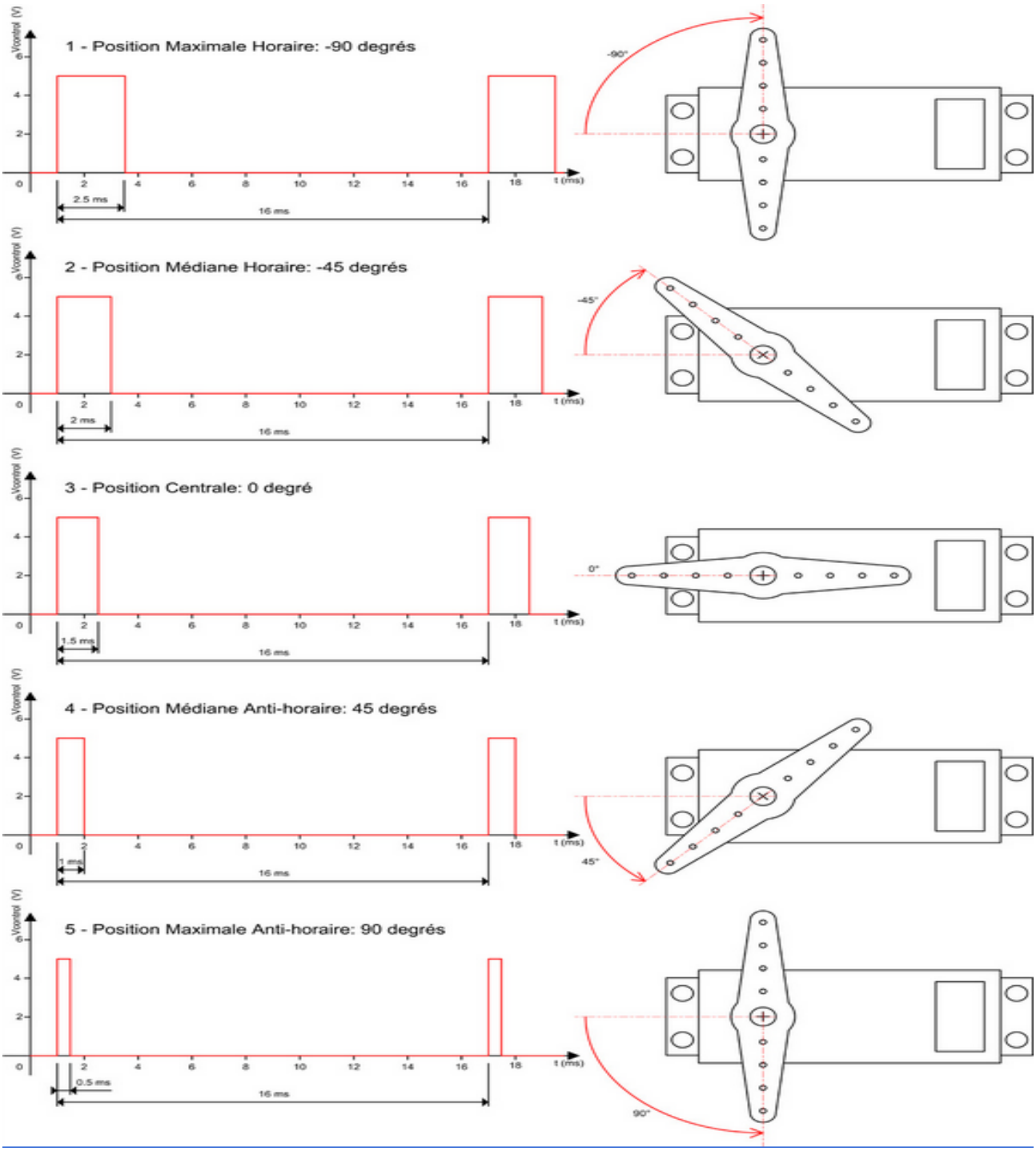
Il y a d'abord eu les tests pour contrôler les servomoteurs. Ceux-ci ont été réalisés en utilisant la carte pic18F conçue au département GEII et utilisée lors des travaux pratiques du module d'informatique industrielle. La figure suivante représente un organigramme de test utilisé.



Programme d'apprentissage des servomoteurs

Pour permettre la rotation à gauche, il faut envoyer une impulsion d'une durée supérieure à 1,5ms. L'angle de rotation est déterminé par la durée de l'état haut de la tension envoyé au servomoteur.

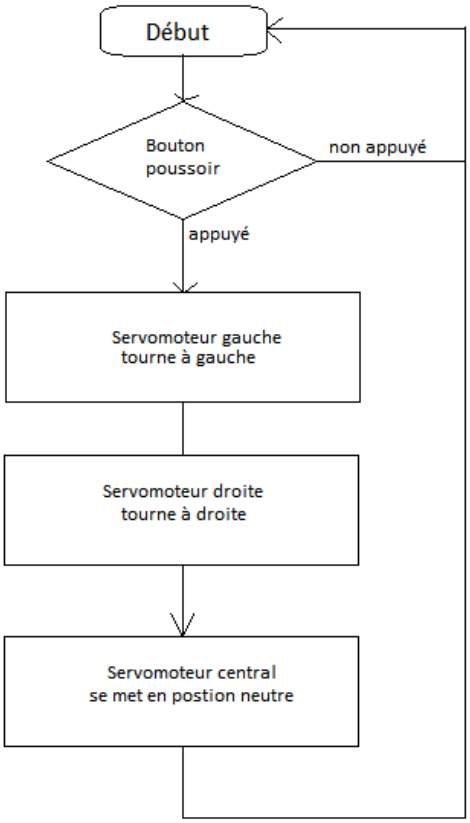
De plus, on ne peut envoyer qu'une seule impulsion au servomoteur toutes les vingt millisecondes. C'est la période que met le servomoteur afin de lire le signal venant du microcontrôleur.



Etat du servomoteur

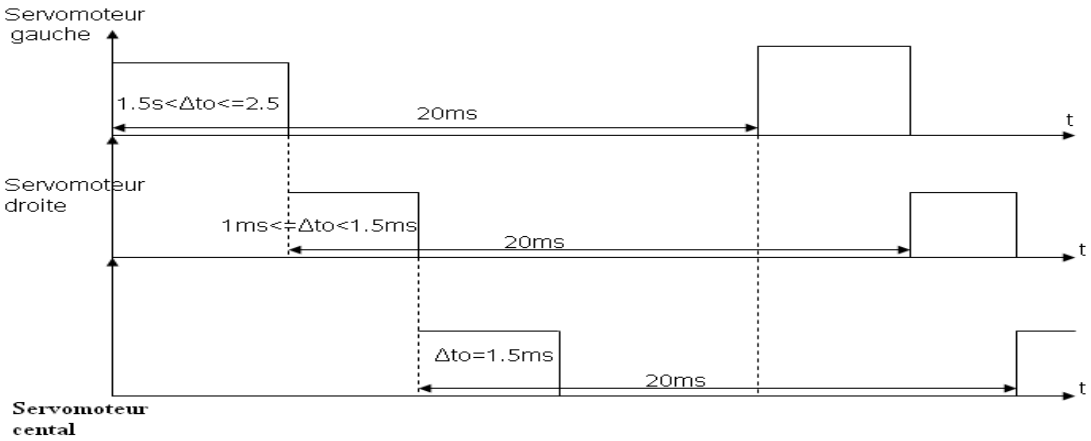
Ce chronogramme montre les différentes rotations avec le temps d'état haut.

Grâce aux tests, j'ai pu commencer à placer plusieurs servomoteurs dans un seul cycle tout en utilisant toujours un bouton poussoir. L'organigramme suivant représente le contrôle de 3 servomoteurs en temps décalé avec des ordres différents.



Programme testant plusieurs servomoteurs

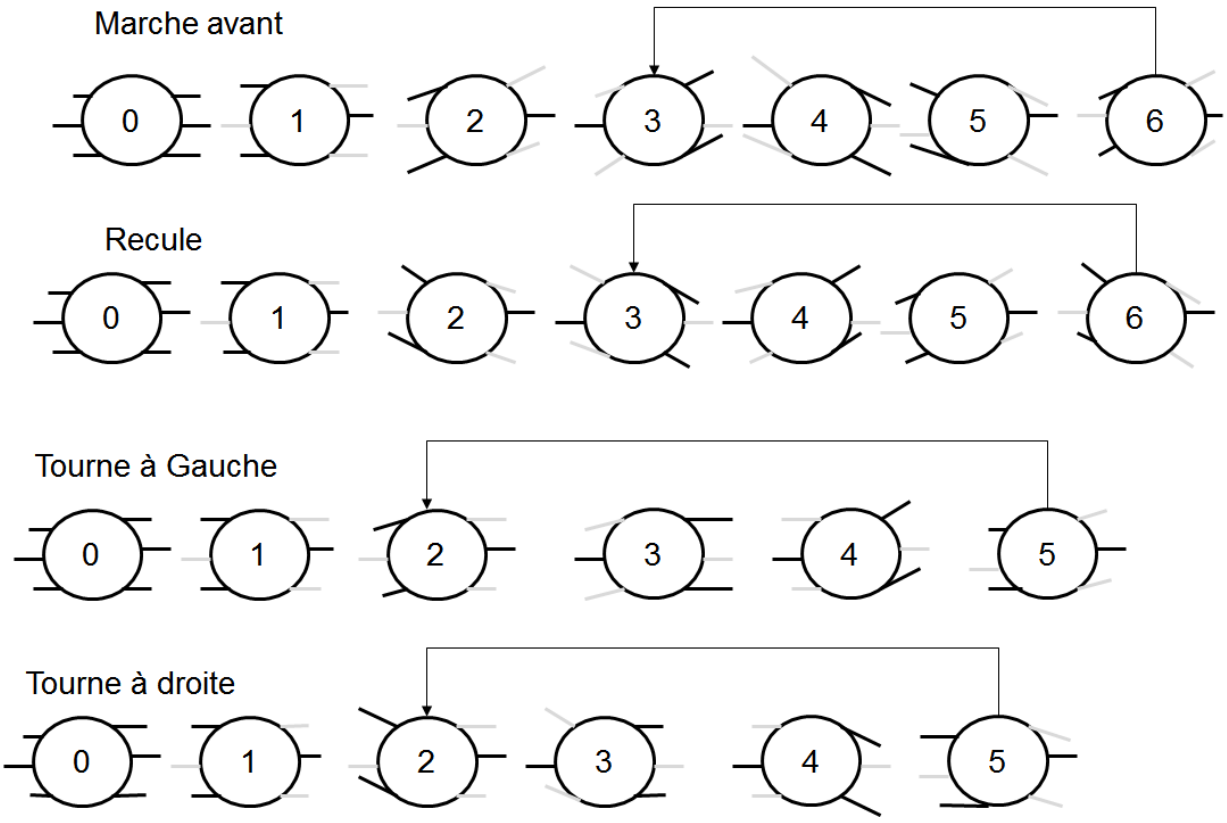
Les chronogrammes suivants permettent de mieux comprendre le principe de fonctionnement :



Chronogramme des servomoteurs

Il existe trois étapes de rotation pour le servomoteur. La rotation sur le côté est déduis suivant la période que l'on envoie au servomoteur. Cependant la position neutre n'est accessible qu'avec un signal de 1,5milliseconde.

Après cette étape j'ai commencé à crée les quatre programmes de marche du robot et les ai testés sans passer par d'interrupteur pour les validés.

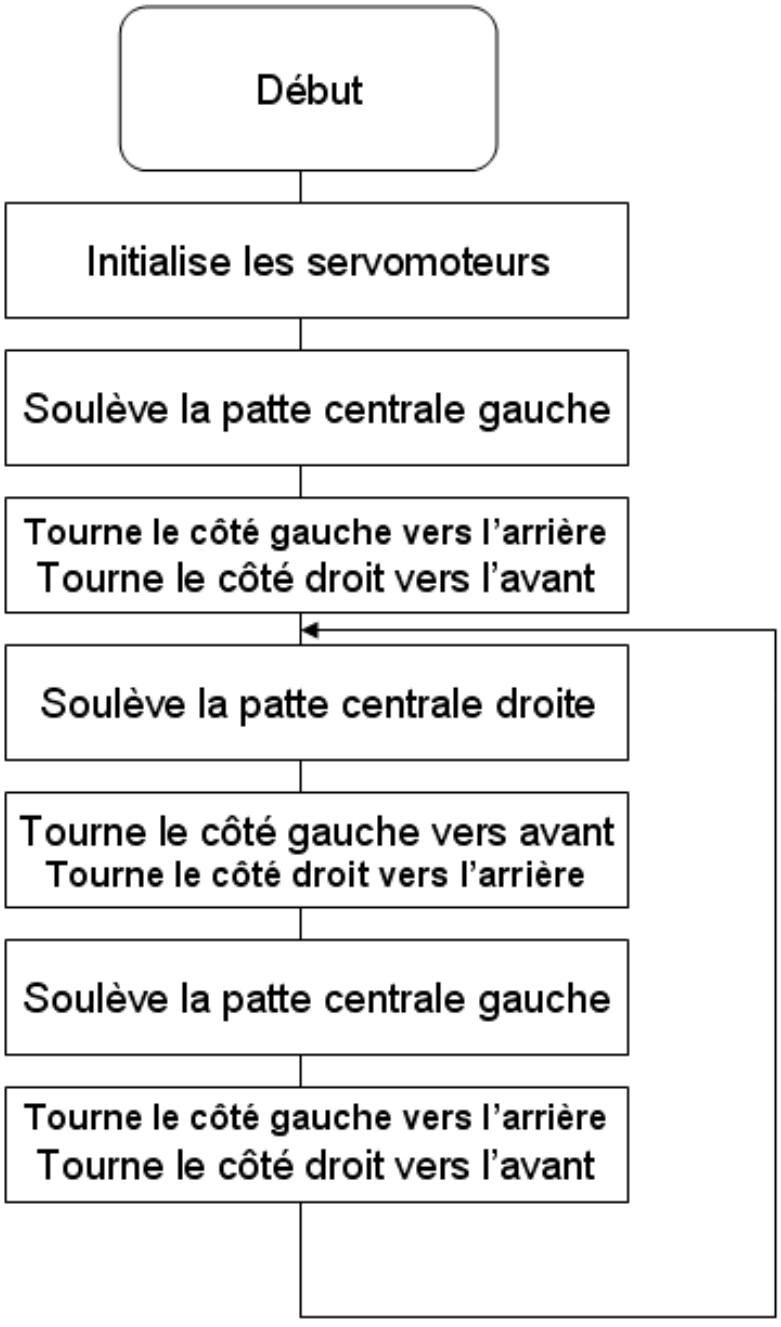


Démarche du robot

Les pattes bleu ne touchent sont en l'air tandis que les pattes noirs représentent les pattes qui touchent le sol.

La programmation suivante permet de faire avancer le robot.

C'est la structure suivante qui permet l'une des démarches qui ont été vues dans la page précédente.



Programme de la marche avant

Cette structure lui permet d'avoir une démarche pouvant ressembler à celle d'un insecte.

Les autres marches du robot utilisent le même procédé avec des rotations des servomoteurs différentes.

b) liaison série asynchrone

Le module Xbee a été testé à partir de l'algorithme suivant. Il s'agit simplement de configurer le périphérique EUSART pour l'utiliser en liaison série asynchrone à la vitesse de 9600 Bauds.

```
Action Principal
Variable : ucCarac (caractère)
Début

  Initport ( )
  InitUSART ( )
  InitInterrupt ( )

Faire indéfiniment
                Attendre que le registre TXREG soit vide
                Mettre la valeur ucCarac dans TXREG
Fin
```

Les fonctions Initport ( ), InitUSART ( ) et InitInterrupt ( ) seront détaillées dans les annexes.

La fonction InitInterrupt est une fonction déclenchée lors de la réception d'un caractère part le périphérique USART. L'algortihme est donnée ci-dessous

```
Interruption
Variable : ucCarac
Début

Si on  une donnée à été écris dans RCREG
Alors
UcCarac prend la valeur de RCREG
La valeur d'UcCarac  augmente de 1
On remet  le bit RCIF à 0

Fin interruption
```

L'algorithme des déplacements du robot est donné ci-dessous. Ce programme permet de faire déplacer le robot à distance suivant le caractère envoyer depuis un pc vie le module Xbee

Action Principal
Variable : ucCarac (caractère)
Début

Initport ( )
InitUSART ( )
InitInterrupt ( )

Faire indéfiniment

Attendre que le registre TXREG soit vide

Mettre la valeur TXREG dans ucCarac
Le robot met tous les servomoteurs en position neutre

Si ucCarac = « z »
Le robot avance

Si ucCarac =« q »
Le robot tourne à gauche

Si ucCarac = « s »
Le robot recule

Si ucCarac = «d »
Le robot tourne à droite

Si un autre caractère est envoyé
Le robot met en position tous les servomoteurs

Fin

Conclusion

Très peu des connaissances apprises au DUT GEII furent utilisées. Les rares éléments apportés que j'ai pu apporter à l'entreprise, sont l'explication sur l'utilisation des servomoteurs et les doutes d'utilisation pour certains composants. La partie programmation est la partie qui m'a permis d'expliquer le fonctionnement des servomoteurs ainsi que pourquoi j'ai utilisé la méthode des délais pour contrôler ceux- ci. J'ai également pu aider d'autres stagiaires dans leur apprentissage de certain logiciel comme ISIS et ARES.

Cependant ce stage m'a été plus avantageux qu'à l'entreprise. En effet, j'ai découvert un nouvel type de composants, des logiciels qui m'ont permis de créer la carte. Des techniques me furent apportées et d'autres furent découvertes. Ce stage m'a permis de voir les étapes de créations d'un robot, autant dans la recherche des composants que dans la programmation de ceux-ci. J'ai pu apprendre une autre manière d'utiliser ISIS tout comme j'ai appris à utiliser les bases d'ARES sous la tutelle de Mr GUMUCHIAN et des techniciens. En plus de ces programmes, j'ai pu connaître l'utilisation de module inconnu ainsi que quelque moyens pour les utiliser tel que les modules XBEE et des deux capteurs utilisés.

Le projet que l'on m'a donné n'a pût être abouti. Cependant c'est un prototype qui a pu révéler plusieurs points négatif qui pourront être évité pour les futurs développements. Les défauts qu'a pu montrer le prototype créé sont le poids des éléments tel que la mécanique et la pile, le délai de livraison, les tiges de cuivres utilisé ainsi que l'utilisation des servomoteurs.

Le robot peut avancer, reculer, tourner à gauche et à droite également. Plusieurs furent commises pendant l'apprentissage mais pourront également être éviter sur les robots crées sur cette base. Ces erreurs sont dans l'utilisation des délais pour ralentir le robot, le placement de certains composants cms et la taille de la carte électronique qui fut créé pour gagner un maximum de place pour éviter de ne pouvoir placer tous les composants. Les éléments restants sont la programmation des capteurs et du master reset. Ces éléments se font en analogique pour le capteur de distance, en I2C pour le capteur magnétique et le master reset utilisera les interruptions. Cependant certain composant auraient pu être choisi.

Comme pour le capteur de distance qui pose problème par sa taille, on aurait pu choisir le GP2Y0D810Z0F qui est largement plus petit mais qui possède une portée plus courte.

D'autre moyen d'utilisation de composants. Par exemple, le capteur cmps03 peut être utilisé avec une liaison I^2C ou en PWM (modulation de largeur de fréquence). La PWM aurait pu être utilisé car elle fait partie des modules d'électronique en deuxième année.

Glossaire

A:

Analogique : signal qui varie suivant une courbe continue. Opposé du signal numérique.
Ack : acknowledge : acquittement

C:

Cms : composant montée sur surface. Utilisable pour une carte nécessitant très peu d'espace.

I:

I²c: Inter Integrated Circuit.

M:

Module : petite montage électronique préfabriquée et programmée.

N :

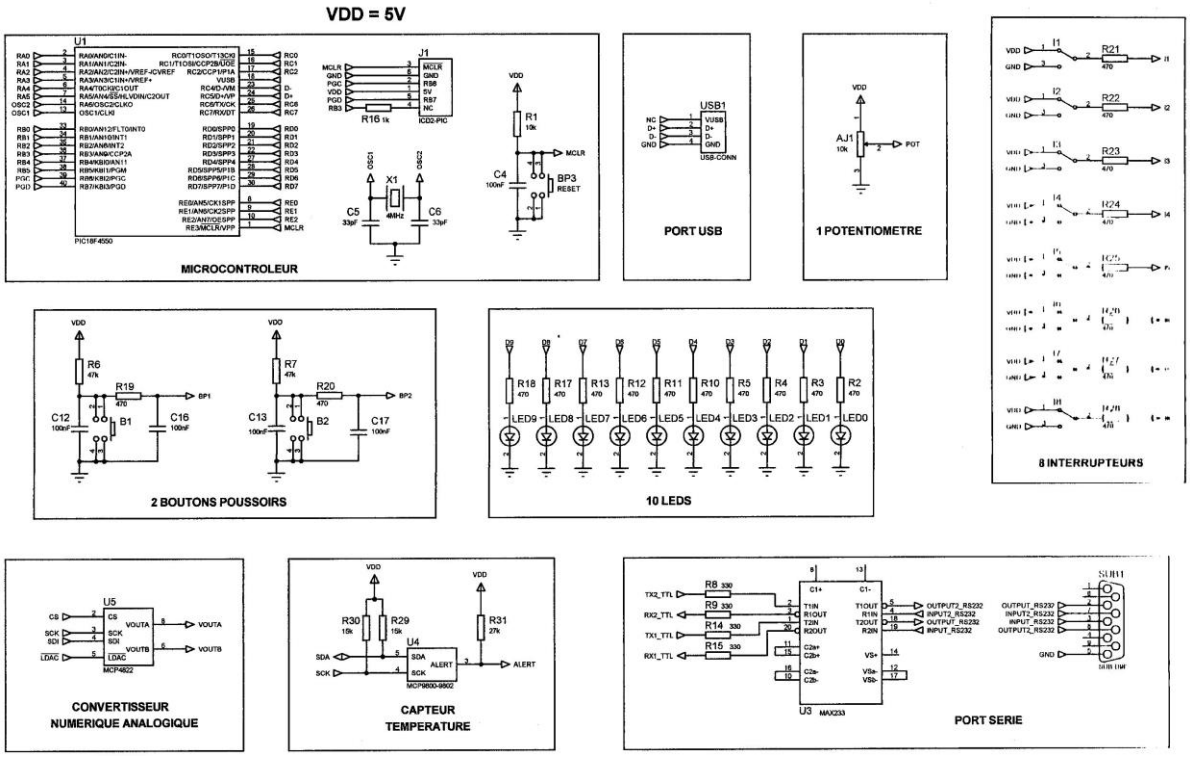
Numérique : signal qui peut seulement être à 0v ou 5v. Opposé du signal analogique

P:

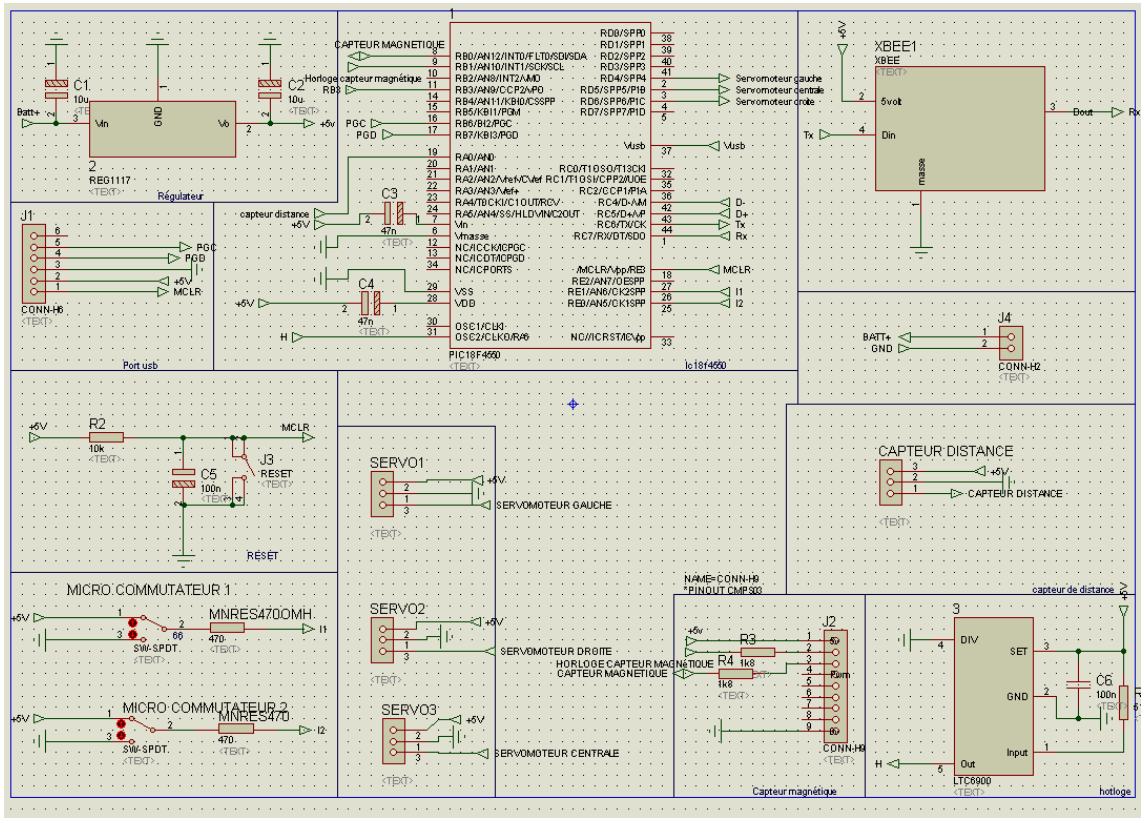
Pwm: Pulse Width Modulation. C'est la modulation de largeur de fréquence.

Annexes

Câblage ISIS :



Montage pris comme exemplaire



Montage ISIS effectué

Programme Test des servomoteurs :

Port.c

```
#include "../Sources/main.h"

void Initport(void)
{
                DIR_BP1=1;                                              // BP1 et BP0 en entrée
                DIR_BP0=1;
                DIR_servo1=0;                                   // servo1, 2 et 3 en sortie
                DIR_servo2=0;
                DIR_servo3=0;
}

void Bougerservo(void)
{
        if(BP0==0 && BP1==1)
        {
         servo1=1;
         Delay10TCYx(70);//attend 0.7ms
         servo1=0;
         servo2=1;
         Delay10TCYx(200);//attend 2ms
         servo2=0;
         servo3=1;
         Delay10TCYx(150);//attend 1.5ms
         servo3=0;*/
         Delay100TCYx(158);// attend 20-1-2-1.2= 17.5ms .

        }

        if(BP1==0 && BP0==1)
        {
         servo1=1;
         Delay10TCYx(210);
         servo1=0;
         servo2=1;
         Delay10TCYx(100);
         servo2=0;
         servo3=1;
         Delay10TCYx(200);
         servo3=0;*/
         Delay100TCYx(149);
   }

        if((BP1==0) && (BP0==0))
        {
         servo1=1;
         Delay10TCYx(150);
         servo1=0;
         servo2=1;
         Delay10TCYx(100);
         servo2=0;
         servo3=1;
         Delay10TCYx(200);
         servo3=0;*/
         Delay100TCYx(155);
   }
}
```

Port.h

```
#define BP0 PORTDbits.RD0
#define DIR_BP0 TRISDbits.TRISD0
#define BP1 PORTDbits.RD1
#define DIR_BP1 TRISDbits.TRISD1
#define servo1 PORTDbits.RD7
#define DIR_servo1 TRISDbits.TRISD7
#define servo2 PORTDbits.RD6
#define DIR_servo2 TRISDbits.TRISD6
#define servo3 PORTDbits.RD5
#define DIR_servo3 TRISDbits.TRISD5

void Initport(void);
void Bougerservo(void);
void Bougerservoinit(void);
```

main.h

```
#include <p18cxxx.h>
#include <Delays.h>
#include "../Sources/port.h"
```

config.h

```
#pragma config FOSC = HS
#pragma config PLLDIV =1
#pragma config CPUDIV = OSC1_PLL2
#pragma config USBDIV = 2
#pragma config MCLRE = ON
#pragma config XINST = ON
#pragma config DEBUG = ON
#pragma config CCP2MX = ON
#pragma config PWRT = OFF
#pragma config ICPRT = OFF
#pragma config LVP = OFF
#pragma config BOR = OFF
#pragma config BORV = 3
#pragma config WDT = OFF
#pragma config PBADEN = OFF
#pragma config FCMEN = OFF
#pragma config IESO = OFF
#pragma config LPT1OSC = OFF
#pragma config STVREN = OFF
```

main.c

```
#include "../Sources/main.h"
#include "../Sources/config.h"
#include <stdio.h>

void main(void)
{
        Initport();
        while(1)
        {
    Bougerservo();

        }

}
```

Programme démarche du robot :

```
#include "../Sources/config.h"
#include <stdio.h>
#include <p18cxxx.h>
#include <Delays.h>
int i;

void marche(void);
void recule(void);
void droite(void);
void gauche(void);
void Initport(void);

#define servocentre PORTDbits.RD0
#define DIR_servocentre TRISDbits.TRISD0

#define servogauche PORTDbits.RD1
#define DIR_servogauche TRISDbits.TRISD1

#define servodroite PORTDbits.RD2
#define DIR_servodroite TRISDbits.TRISD2


void Initport(void)
{
        DIR_servocentre=0;
        DIR_servogauche=0;
        DIR_servodroite =0;                     servo1,2 et 3 en sortie
}

void main(void)
{

  Initport();


while(1)
{
  marche();
}
}


void marche(void)
{
        for (i=0; i<36; i++)// régule la vitesse de rotation
  {
    servocentre=1;
    Delay10TCYx(130);// met le servomoteur central en position neutre avec 1.3ms
    servocentre=0;
    servogauche=1;
    Delay10TCYx(150);// met le servomoteur gauche en position neutre avec 1.5ms
    servogauche=0;
    servodroite=1;
    Delay10TCYx(150);// met le servomoteur droite en position neutre avec 1.5ms
    servodroite=0;
    Delay100TCYx(157);// période de 15.5ms
   }
```

```
            for (i=0; i<36; i++)//fais tourner le servomoteur du centre à fauche
  {
    servocentre=1;
    Delay10TCYx(100);// donne une rotation anti-horaire au servomoteur centrale avec une mise à haut de
2.1ms
    servocentre=0;
    Delay100TCYx(190);// période de 17.9ms
  }

                        for (i=0; i<36; i++)
  {
    servodroite=1;
    Delay10TCYx(200);// donne une rotation horaire au servomoteur droite avec une mise à haut de 0.7ms
    servodroite=0;
    servogauche=1;
    Delay10TCYx(200);// donne une rotation anti-horaire au servomoteur droite avec une mise à haut de
0.7ms
    servogauche=0;
   Delay100TCYx(160);// période de 17ms
  }

            for (i=0; i<36; i++)// fais tourner le servomoteur centrale à droite
  {
    servocentre=1;
    Delay10TCYx(200);// donne une rotation horaire au servomoteur centrale avec une mise à haut de
1ms
    servocentre=0;
    Delay100TCYx(180); // période de 19ms
  }


  for (i=0; i<36; i++)// donne une rotation antihoraire au deux servomoteur gauche et droite avec une mise
à haut de 1ms
  {
    servogauche=1;
    Delay10TCYx(100);
    servogauche=0;
    servodroite=1;
    Delay10TCYx(100);
    servodroite=0;
    Delay100TCYx(180); // période de 18ms
  }
            for (i=0; i<36; i++)// fais tourner le servomoteur centrale à droite
  {
    servocentre=1;
    Delay10TCYx(100);// donne une rotation horaire au servomoteur centrale avec une mise à haut de
1ms
    servocentre=0;
    Delay100TCYx(190); // période de 19ms
  }

  for (i=0; i<36; i++)
  {
    servodroite=1;
    Delay10TCYx(200);// donne une rotation horaire au servomoteur droite avec une mise à haut de 0.7ms
    servodroite=0;
    servogauche=1;
    Delay10TCYx(200);// donne une rotation anti-horaire au servomoteur droite avec une mise à haut de
0.7ms
```

```
    servogauche=0;
   Delay100TCYx(160);// période de 17ms
  }
}

void recule(void)
{
            for (i=0; i<36; i++)// régule la vitesse de rotation
  {
    servocentre=1;
    Delay10TCYx(130);// met le servomoteur central en position neutre avec 1.3ms
    servocentre=0;
    servogauche=1;
    Delay10TCYx(150);// met le servomoteur gauche en position neutre avec 1.5ms
    servogauche=0;
    servodroite=1;
    Delay10TCYx(150);// met le servomoteur droite en position neutre avec 1.5ms
    servodroite=0;
    Delay100TCYx(157);// période de 15.5ms
   }


            for (i=0; i<36; i++)//fais tourner le servomoteur du centre à fauche
  {
   servocentre=1;
   Delay10TCYx(100);// donne une rotation anti-horaire au servomoteur centrale avec une mise à haut de
2.1ms
   servocentre=0;
   Delay100TCYx(190);// période de 17.9ms
  }

                    for (i=0; i<36; i++)
  {
   servodroite=1;
   Delay10TCYx(100);// donne une rotation horaire au servomoteur droite avec une mise à haut de 0.7ms
   servodroite=0;
   servogauche=1;
   Delay10TCYx(100);// donne une rotation anti-horaire au servomoteur droite avec une mise à haut de
0.7ms
   servogauche=0;
  Delay100TCYx(180);// période de 17ms
  }

            for (i=0; i<36; i++)// fais tourner le servomoteur centrale à droite
  {
   servocentre=1;
   Delay10TCYx(200);// donne une rotation horaire au servomoteur centrale avec une mise à haut de
1ms
   servocentre=0;
   Delay100TCYx(180); // période de 19ms
  }


  for (i=0; i<36; i++)// donne une rotation antihoraire au deux servomoteur gauche et droite avec une mise
à haut de 1ms
  {
   servogauche=1;
   Delay10TCYx(200);
   servogauche=0;
   servodroite=1;
```

```
    Delay10TCYx(200);
    servodroite=0;
    Delay100TCYx(160); // période de 18ms
   }
            for (i=0; i<36; i++)// fais tourner le servomoteur centrale à droite
   {
    servocentre=1;
    Delay10TCYx(100);// donne une rotation horaire au servomoteur centrale avec une mise à haut de
1ms
    servocentre=0;
    Delay100TCYx(190); // période de 19ms
   }

   for (i=0; i<36; i++)
   {
    servodroite=1;
    Delay10TCYx(100);// donne une rotation horaire au servomoteur droite avec une mise à haut de 0.7ms
    servodroite=0;
    servogauche=1;
    Delay10TCYx(100);// donne une rotation anti-horaire au servomoteur droite avec une mise à haut de
0.7ms
    servogauche=0;
   Delay100TCYx(180);// période de 17ms
   }
}


void droite(void)
{
                                for (i=0; i<36; i++)// régule la vitesse de rotation
   {
     servocentre=1;
     Delay10TCYx(130);// met le servomoteur central en position neutre avec 1.3ms
     servocentre=0;
     servogauche=1;
     Delay10TCYx(150);// met le servomoteur gauche en position neutre avec 1.5ms
     servogauche=0;
     servodroite=1;
     Delay10TCYx(150);// met le servomoteur droite en position neutre avec 1.5ms
     servodroite=0;
     Delay100TCYx(157);// période de 15.5ms
    }


            for (i=0; i<36; i++)//fais tourner le servomoteur du centre à fauche
   {
    servocentre=1;
    Delay10TCYx(100);// donne une rotation anti-horaire au servomoteur centrale avec une mise à haut de
2.1ms
    servocentre=0;
    Delay100TCYx(190);// période de 17.9ms
   }

                        for (i=0; i<36; i++)
   {
    servogauche=1;
    Delay10TCYx(100);// donne une rotation anti-horaire au servomoteur droite avec une mise à haut de
0.7ms
    servogauche=0;
   Delay100TCYx(190);// période de 17ms
```

```
  }

            for (i=0; i<36; i++)// fais tourner le servomoteur centrale à droite
  {
   servocentre=1;
   Delay10TCYx(200);// donne une rotation horaire au servomoteur centrale avec une mise à haut de
1ms
   servocentre=0;
   Delay100TCYx(180); // période de 19ms
  }


  for (i=0; i<36; i++)// donne une rotation antihoraire au deux servomoteur gauche et droite avec une mise
à haut de 1ms
  {
   servodroite=1;
   Delay10TCYx(100);
   servodroite=0;
   Delay100TCYx(190); // période de 18ms
  }
            for (i=0; i<36; i++)// fais tourner le servomoteur centrale à droite
  {
   servocentre=1;
   Delay10TCYx(100);// donne une rotation horaire au servomoteur centrale avec une mise à haut de
1ms
   servocentre=0;
   Delay100TCYx(190); // période de 19ms
  }

}

void gauche(void)
{
                              for (i=0; i<36; i++)// régule la vitesse de rotation
  {
    servocentre=1;
    Delay10TCYx(130);// met le servomoteur central en position neutre avec 1.3ms
    servocentre=0;
    servogauche=1;
    Delay10TCYx(150);// met le servomoteur gauche en position neutre avec 1.5ms
    servogauche=0;
    servodroite=1;
    Delay10TCYx(150);// met le servomoteur droite en position neutre avec 1.5ms
    servodroite=0;
    Delay100TCYx(157);// période de 15.5ms
   }


            for (i=0; i<36; i++)//fais tourner le servomoteur du centre à fauche
  {
   servocentre=1;
   Delay10TCYx(100);// donne une rotation anti-horaire au servomoteur centrale avec une mise à haut de
2.1ms
   servocentre=0;
   Delay100TCYx(190);// période de 17.9ms
  }

                    for (i=0; i<36; i++)
  {
   servogauche=1;
```

```
    Delay10TCYx(200);// donne une rotation anti-horaire au servomoteur droite avec une mise à haut de
0.7ms
     servogauche=0;
    Delay100TCYx(180);// période de 17ms
   }

            for (i=0; i<36; i++)// fais tourner le servomoteur centrale à droite
   {
     servocentre=1;
     Delay10TCYx(200);// donne une rotation horaire au servomoteur centrale avec une mise à haut de
1ms
     servocentre=0;
     Delay100TCYx(180); // période de 19ms
   }


   for (i=0; i<36; i++)// donne une rotation antihoraire au deux servomoteur gauche et droite avec une mise
à haut de 1ms
   {
     servodroite=1;
     Delay10TCYx(200);
     servodroite=0;
     Delay100TCYx(180); // période de 18ms
   }
            for (i=0; i<36; i++)// fais tourner le servomoteur centrale à droite
   {
     servocentre=1;
     Delay10TCYx(100);// donne une rotation horaire au servomoteur centrale avec une mise à haut de
1ms
     servocentre=0;
     Delay100TCYx(190); // période de 19ms
   }

}
```

InitInterrupt() du Xbee :

```
   void InitInterrupt (void)
{
   INTCONbits.GIE = 1;                  // Validation de toutes sortes d'interruption
   INTCONbits.PEIE = 1;          // Validation des interruptions des périphériques
   RCONbits.IPEN = 1;            // Valide les niveaux d'interruption
}
```

Interrupt.h du Xbee

```
   #include <p18cxxx.h>

   ////////////////////////////////////////////////////////
   //                    Variables globales                                    //
   ////////////////////////////////////////////////////////

   extern volatile unsigned char ucCarac;


   #pragma interrupt Interrupt_High          // L'interruption prioritaire correspond
                                                                  // au label
   Interrupt_High de l'assembleur

   #pragma interruptlow Interrupt_Low        // L'interruption Non prioritaire correspond
                                                                  //au label Interrupt_Low
   de l'assembleur
```

```
//*****************************************************************************
// Ecriture du code Assembleur dans un fichier C
//*****************************************************************************
#pragma code high_vector  =  0x08              // Place le code qui suit à l'adresse 0x0008
void interrupt_at_high_vector(void)            // Fonction pour déclarer l'interruption prioritaire
{
        _asm                                                    // Début d'écriture du code
assembleur à l'adresse 0x0008
                GOTO  Interrupt_High                    // Aller à la fonction : Interrupt_High
        _endasm                                                 // Fin d'écriture du code
assembleur
}
//*****************************************************************************

//*****************************************************************************
// Ecriture du code Assembleur dans un fichier C
//*****************************************************************************
#pragma code low_vector  =  0x18               // Place le code qui suit à l'adresse 0x0018
void interrupt_at_low_vector(void)             // Fonction pour déclarer l'interruption non prioritaire
{
        _asm                                                    // Début d'écriture du code
assembleur à l'adresse 0x0018
                GOTO  Interrupt_Low                     // Aller à la fonction : Interrupt_Low
        _endasm                                                 // Fin d'écriture du code
assembleur
}
//*****************************************************************************

#pragma code                                   // Libere le compilateur de l'obligation d'écrire à une adresse
précise

//*****************************************************************************
// Fonction d'interruption Prioritaire
//*****************************************************************************
void Interrupt_High(void)
{

}
//*****************************************************************************


//*****************************************************************************
// Fonction d'interruption Non Prioritaire
//*****************************************************************************
void Interrupt_Low(void)
{
        if(PIR1bits.RCIF == 1)
        {
                ucCarac = RCREG;
                ucCarac = ucCarac + 1;
                // RaZ de RCIF optionnelle puisque remis à '0'
                // automatiquement lors de la lecture du registre
                // Néanmoins reste dans la logique de ce qui a été
                // utilisé auparavant dans les interruptions
                PIR1bits.RCIF = 0;
        }
}
//*****************************************************************************
```

InitUSART() du Xbee :

```
void InitUsart(void)
{
	// transmission de donnée de 8 bit
	TXSTAbits.TX9 = 0;
	// Validation du module de transmssion
	TXSTAbits.TXEN = 1;
	// USART pour serie asynchrone :
	TXSTAbits.SYNC = 0;
	// Selection du High Baud Rate
	// cf. Tableaux p.249 et 250 pour Fosc = 1MHz
	TXSTAbits.BRGH = 1;

	// Validation du Port série
	RCSTAbits.SPEN = 1;
	// réception de donnée de 8 bit
	RCSTAbits.RX9 = 0;
	// réception de données en continue
	RCSTAbits.CREN = 1;

	// Choix d'utilisation de 16bit pour SPBRG
	BAUDCONbits.BRG16 = 1;

	// Vitesse de transmission de 9600 Bauds
	// Fosc = 1 MHz;
	SPBRG = 25;

	//Préparation au déclanchement d'interruption
	//Validation de l'interruption provenant de l'USART
//	PIE1bits.TXIE = 1;
	PIE1bits.RCIE = 1;
//	IPR1bits.TXIP = 0; // niveau bas
	IPR1bits.RCIP = 0; // niveau bas
//	PIR1bits.TXIF = 0;
	PIR1bits.RCIF = 0;
}
```

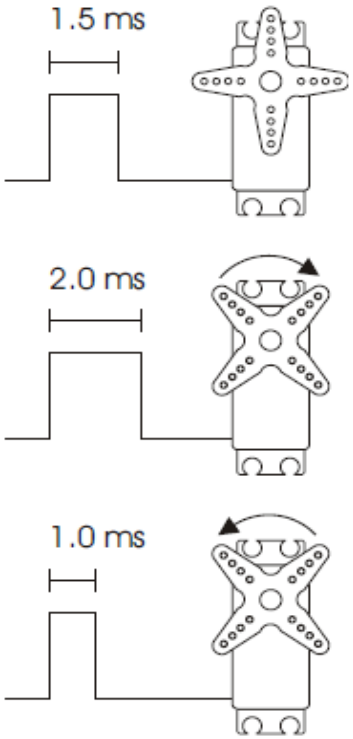
Initport() du Xbee :

```
void InitPort(void)
{
	// Commande servomoteurs
	// Broches RC0 et RC1 configurées en "sortie" :
	TRISCbits.TRISC0 = 0;
	TRISCbits.TRISC1 = 0;
	// Initialisation à '0 de RC0 et RC1
	PORTCbits.RC0 = 0;
	PORTCbits.RC1 = 0;

	// Broches RC7 et RC6 configurées en "entrée"
	// cf. doc. p.243 :
	TRISCbits.TRISC7 = 1;
	TRISCbits.TRISC6 =1;

	// Broche RA0 configurée en "entrée"
	// Signal analogique capteur de distance
	TRISAbits.TRISA0 = 1;

}
```

How Servos and the Servo Controller Work

Radio Control (RC) hobby servos are small actuators designed for remotely operating model vehicles such as cars, airplanes, and boats. A servo might typically move the control surface of an airplane or the steering mechanism in a car. A servo contains a small motor and gearbox to do the work, a potentiometer to measure the position of the output gear, and an electronic circuit that controls the motor to make the output gear move to the desired position. Because all of these components are packaged into a compact, low-cost unit, servos are great actuators for robots.

An RC servo has three leads: two for power and ground, and a third for a control signal input. The control signal is a continuous stream of pulses that are 1 to 2 milliseconds long, repeated approximately fifty times per second, as shown below. The width of the pulses determines the position to which the servo moves. The servo moves to its neutral, or middle, position when the signal pulse width is 1.5 ms. As the pulse gets wider, the servo turns one way; if the pulse gets shorter, the servo moves the other way. Typically, a servo will move approximately 90 degrees for a 1 ms change in pulse width, but the exact correspondence between pulse width and servo varies from one servo manufacturer to another.



The Pololu servo controller performs the processor-intensive task of simultaneously generating 16 independent servo control signals. The servo controller can generate pulses from 0.25 ms to 2.75 ms, which is greater than the range of most servos, and which allows for a servo operating range of over 180 degrees.

Internally, the servo controller maintains a servo position value that is two times the pulse width, measured in microseconds. Thus, the 1.5 ms neutral position, which is 1500 microseconds long, is represented internally as 3000. The internal values thus range from 500 to 5500. Various interface modes allow the user to set the position value for each servo in multiple ways, which are described in depth in the "Using the Servo Controller" section.

LTC6900

Low Power, 1kHz to 20MHz Resistor Set SOT-23 Oscillator

FEATURES

- **One External Resistor Sets the Frequency**
- **1kHz to 20MHz Frequency Range**
- **500µA Typical Supply Current, V_S = 3V, 3MHz**
- Frequency Error ≤1.5% Max, 5kHz to 10MHz (T_A = 25°C)
- Frequency Error ≤2% Max, 5kHz to 10MHz (T_A = 0°C to 70°C)
- ±40ppm/°C Temperature Stability
- 0.04%/V Supply Stability
- 50% ±1% Duty Cycle 1kHz to 2MHz
- 50% ±5% Duty Cycle 2MHz to 10MHz
- Fast Start-Up Time: 50µs to 1.5ms
- 100Ω CMOS Output Driver
- Operates from a Single 2.7V to 5.5V Supply
- Low Profile (1mm) ThinSOT™ Package

APPLICATIONS

- Portable and Battery-Powered Equipment
- PDAs
- Cell Phones
- Low Cost Precision Oscillator
- Charge Pump Driver
- Switching Power Supply Clock Reference
- Clocking Switched Capacitor Filters
- Fixed Crystal Oscillator Replacement
- Ceramic Oscillator Replacement

DESCRIPTION

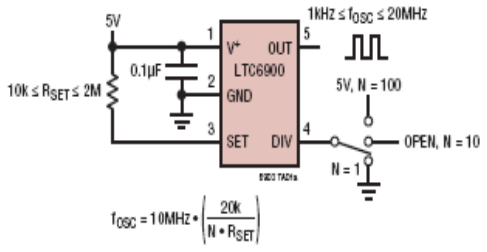
The LTC®6900 is a precision, low power oscillator that is easy to use and occupies very little PC board space. The oscillator frequency is programmed by a single external resistor (R_{SET}). The LTC6900 has been designed for high accuracy operation (≤1.5% frequency error) without the need for external trim components.

The LTC6900 operates with a single 2.7V to 5.5V power supply and provides a rail-to-rail, 50% duty cycle square wave output. The CMOS output driver ensures fast rise/fall times and rail-to-rail switching. The frequency-setting resistor can vary from 10kΩ to 2MΩ to select a master oscillator frequency between 100kHz and 20MHz (5V supply). The three-state DIV input determines whether the master clock is divided by 1, 10 or 100 before driving the output, providing three frequency ranges spanning 1kHz to 20MHz (5V supply). The LTC6900 features a proprietary feedback loop that linearizes the relationship between R_{SET} and frequency, eliminating the need for tables to calculate frequency. The oscillator can be easily programmed using the simple formula outlined below:

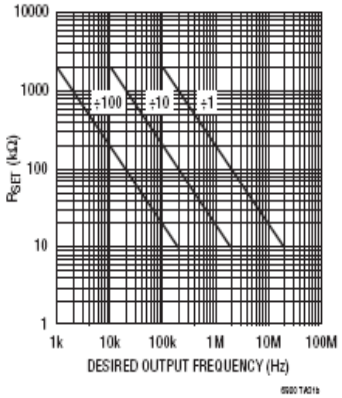
$$f_{OSC} = 10MHz \bullet \left(\frac{20k}{N \bullet R_{SET}}\right),\ N = \begin{cases} 100, & DIV\ Pin = V^+ \\ 10, & DIV\ Pin = Open \\ 1, & DIV\ Pin = GND \end{cases}$$

TYPICAL APPLICATION

Clock Generator



R_{SET} vs Desired Output Frequency



6900fa

Régulateur 1117 :

Burr-Brown Products from Texas Instruments

REG1117
REG1117A

SBVS001D – OCTOBER 1992 – REVISED JULY 2004

800mA and 1A Low Dropout Positive Regulator 1.8V, 2.5V, 2.85, 3.3V, 5V, and Adjustable

FEATURES

- FIXED AND ADJUSTABLE VERSIONS
- 2.85V MODEL FOR SCSI-2 ACTIVE TERMINATION
- OUTPUT CURRENT:
  REG1117: 800mA max
  REG1117A: 1A max
- OUTPUT TOLERANCE: ±1% max
- DROPOUT VOLTAGE:
  REG1117: 1.2V max at I_O = 800mA
  REG1117A: 1.3V max at I_O = 1A
- INTERNAL CURRENT LIMIT
- THERMAL OVERLOAD PROTECTION
- SOT-223 AND DDPAK SURFACE-MOUNT PACKAGES

APPLICATIONS

- SCSI-2 ACTIVE TERMINATION
- HAND-HELD DATA COLLECTION DEVICES
- HIGH EFFICIENCY LINEAR REGULATORS
- BATTERY-POWERED INSTRUMENTATION
- BATTERY MANAGEMENT CIRCUITS FOR NOTEBOOK AND PALMTOP PCs
- CORE VOLTAGE SUPPLY: FPGA, PLD, DSP, CPU

DESCRIPTION

The REG1117 is a family of easy-to-use three-terminal voltage regulators. The family includes a variety of fixed- and adjustable-voltage versions, two currents (800mA and 1A) and two package types (SOT-223 and DDPAK). See the chart below for available options.

Output voltage of the adjustable versions is set with two external resistors. The REG1117 low dropout voltage allows its use with as little as 1V input-output voltage differential.

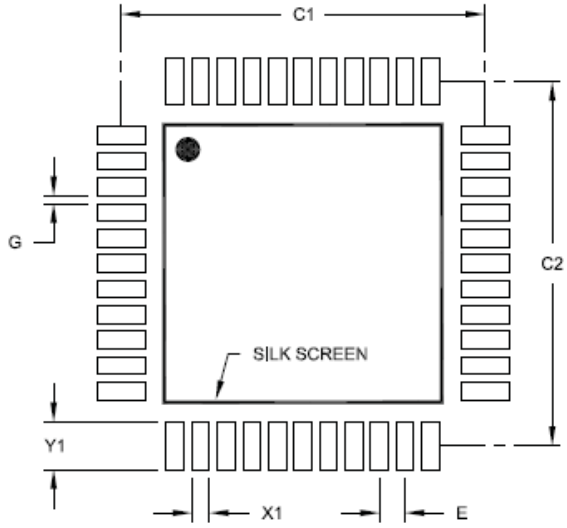
Laser trimming assures excellent output voltage accuracy without adjustment. An NPN output stage allows output stage drive to contribute to the load current for maximum efficiency.

	800mA		1A	
VOLTAGE	SOT-223	DDPAK	SOT-223	DDPAK
1.8V			✓	✓
2.5V			✓	✓
2.85V	✓			
3.3V	✓	✓		
5V	✓			✓
Adjustable	✓		✓	✓

TEXAS INSTRUMENTS
www.ti.com

44-Lead Plastic Thin Quad Flatpack (PT) – 10x10x1 mm Body, 2.00 mm [TQFP]

Note:	For the most current package drawings, please see the Microchip Packaging Specification located at http://www.microchip.com/packaging



RECOMMENDED LAND PATTERN

Units		MILLIMETERS		
Dimension Limits		MIN	NOM	MAX
Contact Pitch	E		0.80 BSC	
Contact Pad Spacing	C1		11.40	
Contact Pad Spacing	C2		11.40	
Contact Pad Width (X44)	X1			0.55
Contact Pad Length (X44)	Y1			1.50
Distance Between Pads	G	0.25		

Notes:

1. Dimensioning and tolerancing per ASME Y14.5M

BSC: Basic Dimension. Theoretically exact value shown without tolerances.

Microchip Technology Drawing No. C04-2076A

Connexion USART avec module XBEE :

2. RF Module Operation

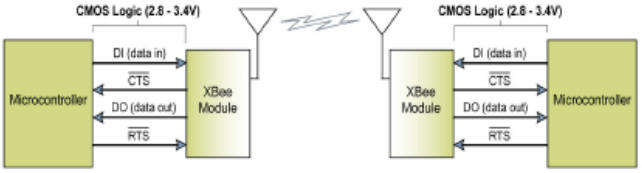
Serial Communications

The XBee®/XBee-PRO® RF Modules interface to a host device through a logic-level asynchronous serial port. Through its serial port, the module can communicate with any logic and voltage compatible UART; or through a level translator to any serial device (For example: Through a Digi proprietary RS-232 or USB interface board).

UART Data Flow

Devices that have a UART interface can connect directly to the pins of the RF module as shown in the figure below.

Figure 2-01. System Data Flow Diagram in a UART-interfaced environment
(Low-asserted signals distinguished with horizontal line over signal name.)

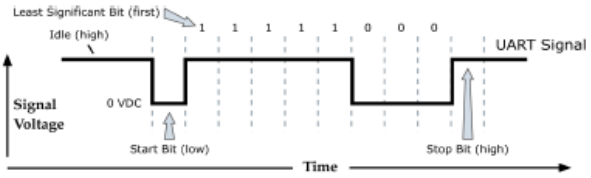


Serial Data

Data enters the module UART through the DI pin (pin 3) as an asynchronous serial signal. The signal should idle high when no data is being transmitted.

Each data byte consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following figure illustrates the serial bit pattern of data passing through the module.

Figure 2-02. UART data packet 0x1F (decimal number "31") as transmitted through the RF module
Example Data Format is 8-N-1 (bits - parity - # of stop bits)



Serial communications depend on the two UARTs (the microcontroller's and the RF module's) to be configured with compatible settings (baud rate, parity, start bits, stop bits, data bits).

The UART baud rate and parity settings on the XBee module can be configured with the BD and SB commands, respectively. See the command table in Chapter 3 for details.

Bouton Poussoir :

Tactile Switches, .236 [6] x .236 [6], Surface Mount

MATERIAL SPECIFICATIONS:

Contacts .....Copper alloy, silver plate
Case.....PPS
Actuator Button .....PPS
Cover.....Steel, tin plate
Terminals .....Copper alloy, silver plate

ENVIRONMENTAL SPECIFICATIONS

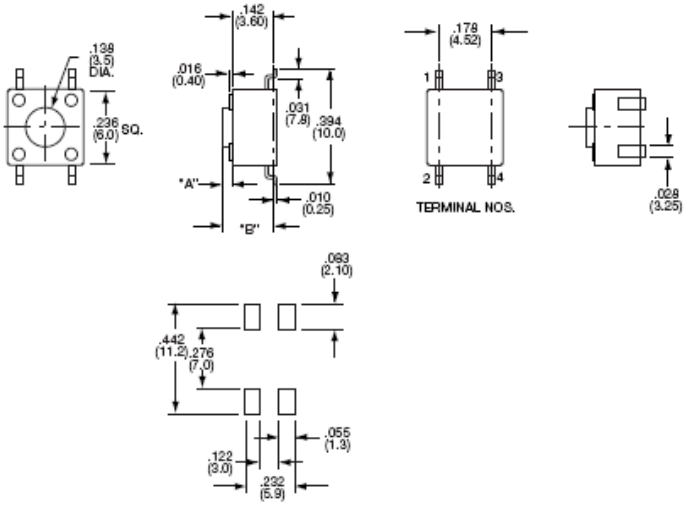
Operating Temperature..... -4°F to +185°F (-20°C to + 85°C)
Storage Temperature.....-22°F to +185°F (-30°C to +85°C)
Solder Heat Resistance .....IR Soldering EIA-364-56
Procedure 5 Level 3, 500°F (260°C)

TYPICAL PERFORMANCE CHARACTERISTICS:

Contact Rating .....50 mA @ 12 VDC
Initial Contact Resistance .....100 Milliohms max.
Insulation Resistance .....100 Megohms min. @ 100 VDC
Dielectric Strength .....500 VAC for 1 minute
Actuation Force .....See Below
Actuator Travel..... .010"(.254)
Life Expectancy .....Up to 100,000 Cycles

6 x 6 mm

FSMJSMA



Bulk Pack Part Number		Tape & Reel Part Number		Actuator Length	Switch Height	Actuation Force
Tyco Electronics	Alcoswitch	Tyco Electronics	Alcoswitch	Dimension A	Dimension B	
2-1437565-7	FSM2JSMA	3-1437565-0	FSM2JSMATR	.028 (0.70)	.169 (4.30)	160 ± 50 grams
4-1437565-1	FSM4JSMA	4-1437565-2	FSM4JSMATR	.055 (1.40)	.196 (5.00)	160 ± 50 grams
4-1437565-9	FSM6JSMA	5-1437565-0	FSM6JSMATR	.134 (3.40)	.276 (7.00)	160 ± 50 grams
2-1437565-8	FSM2JSMAA	2-1437565-9	FSM2JSMAATR	.028 (0.70)	.169 (4.30)	260 ± 50 grams
1571563-1	FSM4JSML	1571563-2	FSM4JSMLTR	.055 (1.40)	.196 (5.00)	100 ± 50 grams
1571563-3	FSM4JSMAA	1571563-4	FSM4JSMAATR	.055 (1.40)	.196 (5.00)	260 ± 50 grams
—	FSM6JSMAA	—	FSM6JSMAATR	.134 (3.40)	.276 (7.00)	260 ± 50 grams
3-1437566-8	FSM8JSMA	—	FSM8JSMATR	.232 (5.90)	.374 (9.50)	260 ± 50 grams
1571527-1	FSM2JSML	1571527-2	FSM2JSMLTR	.028 (0.70)	.169 (4.30)	100 ± 50 grams

Catalog 1308390
Issued 9-04
www.tycoelectronics.com

Dimensions are in inches and millimeters unless otherwise specified. Values in parentheses or brackets are metric equivalents.

Dimensions are shown for reference purposes only. Specifications subject to change.

USA: 1-800-522-6752
Canada: 1-905-470-4425
Mexico: 01-800-733-8926
C. America: 52-55-5-729-0425
South America: 55-11-3611-1514
Hong Kong: 852-2735-1628
Japan: 81-44-844-8013
UK: 44-141-810-8967

Sources

Base mécanique : : http://innovelectronique.fr/2009/12/08/robot-hexapode-robot-fourmi/

Exemple de marche du robot : http://www.ac-limoges.fr/sti_ge/spip.php?article36

Liaison série asynchrone : http://www.commentcamarche.net/contents/1131-transmission-de-donnees-les-modes-de-transmission

Bus I²c: http://www.atmicroprog.com/cours/I2C/i2c.php

Servomoteur : http://www.pololu.com/file/0J35/usc01a_guide.pdf

CMPS03 : http://www.robot-electronics.co.uk/htm/cmps3tech.htm