République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique

Université Mouloud MAMMERI de Tizi-Ouzou

Faculté de Génie Electrique et d'Informatique
DEPARTEMENT D'AUTOMATIQUE

Mémoire de Fin d'Etudes de MASTER PROFESSIONNEL

Spécialité : **Automatique et informatique industrielles**

Présenté par
Salah CHERFAOUI
Anis BOUNOUAR

Mémoire dirigé par **Mr Nabil BENYAHIA** et co-dirigé par **Mr Md O BENSIDHOUM**

Thème

Conception d'une carte électronique à base de microcontrôleurs 16F877A : Application a un drone quadri rotor

Mémoire soutenu publiquement le 27 Septembre 2014 devant le jury composé de :

M Fadhila BOUDJMAA
MMA, UMMTO, President

Mr Md Outahar BENSIDHOUM
MCB, UMMTO, Rapporteur

Mr Ahmed KASRI
MMA, UMMTO, Examinateur

M Ourida HEDJEM
MMA, UMMTO, Examinateur

Remerciements

Au terme de ce travail, nous tenons à exprimer notre vive reconnaissance et nos sincères remerciements à notre promoteur Mr **Nabil BENYAHIA** enseignant à l'UMMTO pour avoir accepté de diriger notre travail, son aide et ses encouragements.

Nous tenons aussi a remercier Mr **Md Outahar BENSIDHOUM** Chef du département d'Automatique à l'UMMTO pour son aide.

On voudrait également remercier tout les enseignants et le personnel administratif de la Faculté de Génie Electrique et d'Informatique en général et du département d'Automatique en particulier.

On remercie les membres du jury, pour l'honneur qu'il nous one fait, en acceptant d'évaluer notre travail.

On remercie également tous ceux qui ont participé de prés ou de loin à l'élaboration de ce travail.

Sommaire

CHAPITRE III : Conception logicielle

Tables des figures

Introduction générale :

L'intérêt pour les drones télécommandés semble grandir de plus en plus notamment pour les applications militaires (intervention en milieux hostiles). De nos jours la technologie ayant évoluée, les systèmes se sont miniaturisés, de nouvelles applications s'ouvrent alors aux drones qui ne sont plus qu'à finalités militaires mais aussi civiles (surveillances des feux de foret, surveillances du trafic autoroutier ….)

La recherche dans le domaine des véhicules aériens autonomes est essentiellement pluridisciplinaire. En effet elle fait intervenir des domaines très variés tels que l'aérodynamique, le traitement du signal et le l'image, la commande automatique, la mécanique, les matériaux composites, l'informatique temps réel….

Dans ce mémoire, nous nous intéressons aux véhicules aériens miniatures et plus particulièrement à un mini hélicoptère à quatre hélices (quadri rotor), nous allons concevoir une carte électronique et une télécommande (connexion via Bluetooth) à base de PICs 16F877A sous logiciel (PROTEUS) et une commande adapté sous Pic c compiler et pour cela notre travail est organisé comme suit :

- Chapitre I : présentation générale du drone quadri rotor ainsi que son principe de fonctionnement.
- Chapitre II : conception matérielle de la carte électronique en citant les différents composants nécessaire à la conception.
- Chapitre III : ce chapitre donne une configuration logicielle et les différents programmes sous forme d'organigrammes.

CHAPITRE I
Généralités sur les drones

CHAPITRE I : Généralités sur les drones

I.1.Introduction :

Dans ce chapitre nous essayons de décrire d'une manière générale le fonctionnement d'un drone quadri rotor, qui est un modèle d'hélicoptère à quatre rotors,

Nous avons aussi mis à l'évidence le problème principale du quadri rotor qui est la stabilisation, on a aussi donné un aspect théorique sur le régulateur PID qui est l'une des méthodes les plus efficaces pour la régulation, et donc la stabilisation de notre drone.

I.2.Définition d'un drone :

Un drone de l'anglais «faux-bourdon» est également appelé **UAV (Unmanned Aerial Vehicle)**, **UAS (Unmanned Aerial System)** ou **RPA (Remotely Piloted Aircraft)** est un aérodyne sans pilote embarqué et télécommandé qui emporte souvent une charge utile,

Il est lui-même un composant d'un système qui est composé :

- D'un ou plusieurs vecteurs aériens équipés de capteurs de détection.
- De liaisons radioélectriques de données entre le vecteur aérien et la partie au sol.

I.3.Historique :

Le concept est né pendant et après la Première Guerre mondiale : des prototypes d'avions sans pilote radiocommandés ont ainsi vu le jour. En 1917, aux États-Unis, le projet Hewitt-Sperry automatic airplane des ingénieurs Elmer Ambrose Sperry, Lawrence Sperry et Peter Cooper Hewitt se développe.

En France, le 2 juillet 1917 le pilote Max Boucher, fait voler un avion Voisin « sans l'intervention de l'homme » sur 1 km. Au début de l'année 1918, Georges Clemenceau, président de la Commission sénatoriale de l'Armée, lance un projet d'« avions sans pilotes ». Le capitaine Boucher améliore son système de pilotage automatique et le 14 septembre, il fait voler pendant 51 minutes sur un parcours de100 km un avion Voisin BN3 .

Le mot anglais «drone » désigne le faux-bourdon ; le nom lui a été donné par dérision dans les années 1930, leur vol lent et bruyant ressemblait plus à celui du bourdon. Le nom est resté et fut repris par l'armée américaine dés 1941.

I.4.Le drone quadri rotor : [2]

I.4.1.Définition :

Un quadri rotor est un aéronef à voilure tournante comportant quatre rotors pour sa sustentation, les quatre rotors sont généralement placés aux extrémités d'une croix. Afin d'éviter à l'appareil de tourner sur lui-même sur son axe de lacet, il est nécessaire que deux hélices tournent dans un sens (A1 et A2) et que les deux autres dans l'autre sens (B1 et B2).

Pour pouvoir diriger l'appareil, il est nécessaire que chaque couple d'hélices tournant dans le même sens soit placé aux extrémités opposées d'une branche de la croix.

La figure ci-dessous démontre le sens de rotation des hélices :

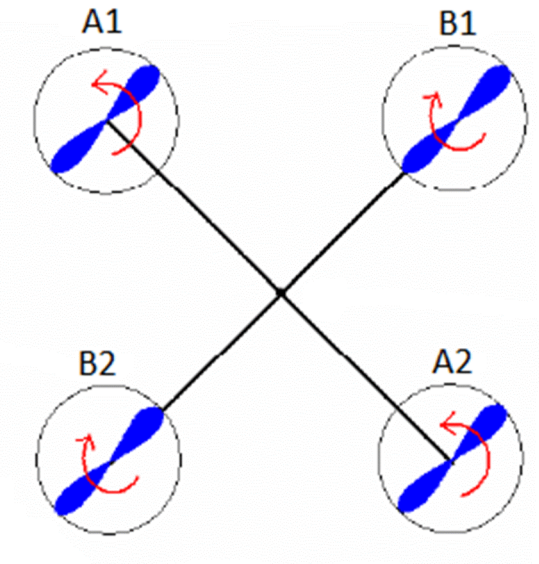


Figure I.1 : sens de rotation des hélices

La commande simultanée des quatre moteurs est utilisée pour modifier l'altitude.

I.4.2.Fonctionnement :

Le fonctionnement d'un quadri rotor est assez particulier. C'est un engin omnidirectionnel à décollage vertical et à atterrissage vertical (VTOL). On distingue quatre mouvements possibles : les gaz, le lacet, le roulis et le tangage. Pour garder le contrôle du lacet, cela implique que deux hélices tournent dans le sens horaire (hélice à pas normal) et les deux autres dans le sens antihoraire (hélices à pas inversé). Ce qui permet à vitesse égale un couple d'anti rotation nul afin que le micro hélicoptère ne tourne pas dans le plan (x,y) représenté sur la figure (I.2).

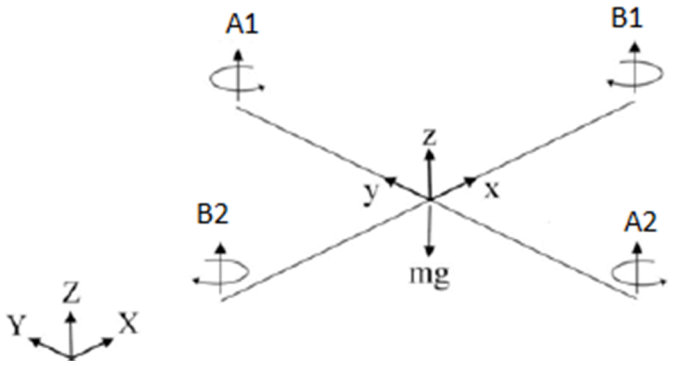


Figure I.2 : Repères du système

I.4.2.1.Décollage (le gaz) :

Le mouvement de gaz correspond tout simplement à la montée/descente du quadri rotor. La montée est obtenue en augmentant la vitesse des quatre moteurs. La descente, qui, elle est plus difficile à doser, s'obtient par la réduction de la vitesse des moteurs.

La figure (I.3) montre l'accélération simultanée des quatre moteurs.

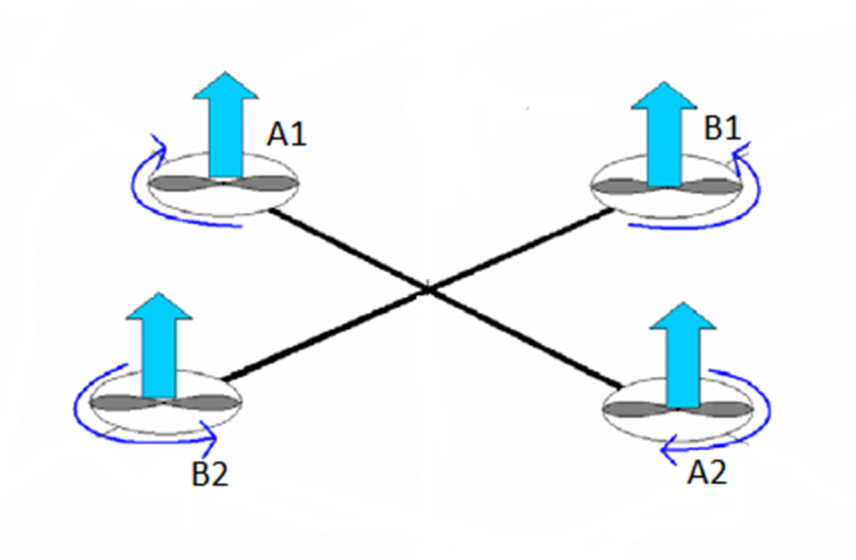


Figure I.3 : décollage du quadri rotor

L'idée est de faire tourner les quatre moteurs du quadri rotor à une vitesse équivalente, Plus la vitesse des hélices est grande plus la force de portance grandit. Lorsque cette force est égale au poids de l'appareil, le vol est dit stationnaire.

I.4.2.2.Déplacement (avant, arrière, gauche, droite) :

Les quatre mouvements (avant, arrière, gauche, droite) sont obtenus en jouant à chaque fois sur les quatre moteurs.

Pour avancer il suffit de réduire la vitesse des moteurs A1et B1 et d'augmenter la vitesse des moteurs A2 et B2 (voir figure I.5), pour reculer il faut réduire la vitesse des moteurs A2 et B2 et augmenter la vitesse des moteurs A1 et B1, ces deux mouvements sont obtenus par le roulis.

Pour aller à gauche ou à droite il suffit de faire varier l'angle de tangage, pour aller à gauche il faut réduire la vitesse des moteurs A1 et B2 et augmenter la vitesse des moteurs A2 et B1, pour prendre à droite il faut réduire la vitesse des moteurs A2 et B1 et augmenter la vitesse des moteurs A1 et B2.

Le roulis et le tangage (roll and pitch) en aéronautique sont des mouvements assez similaires visant à pencher le quadri copter sur l'axe x' ou sur l'axe y' (figure I.4).

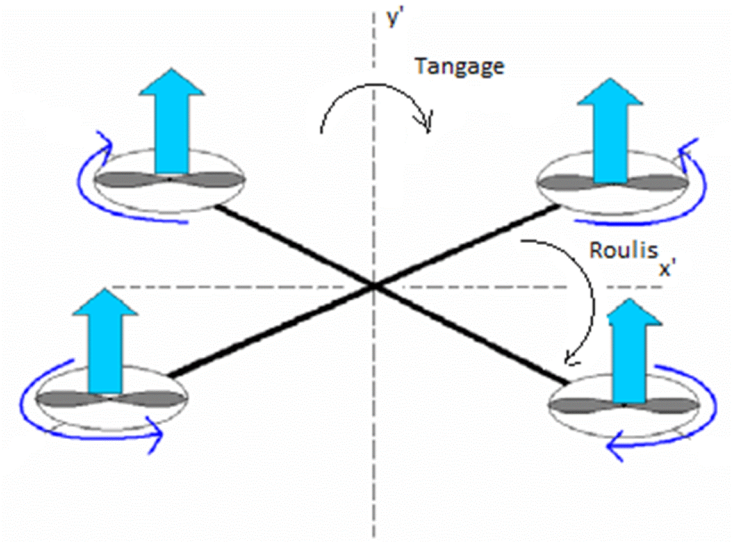


Figure I.4 : Mouvements de roulis et de tangage

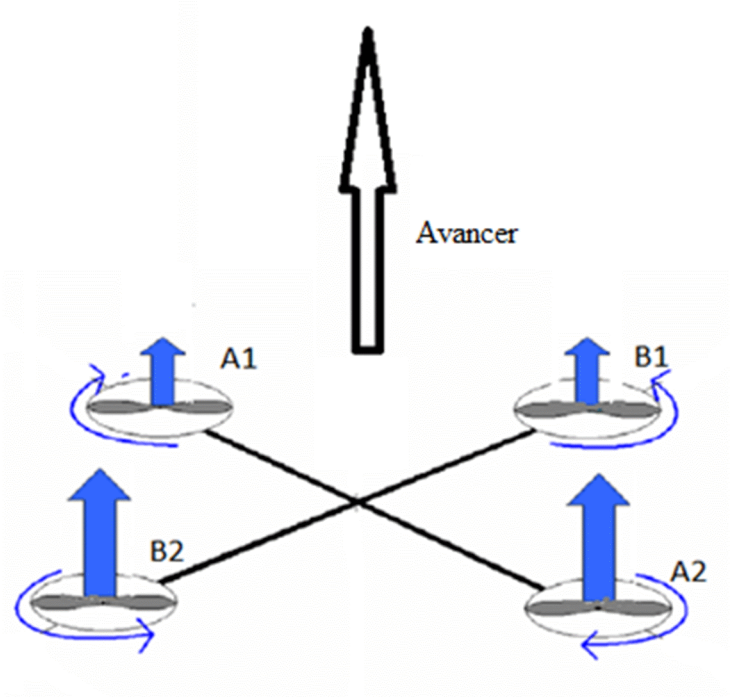


Figure I.5: Réduction de la vitesse des moteurs A1 et B1

I.4.2.3.Rotation

Le mouvement de lacet (yaw) sert à faire tourner le quadri rotor sur lui-même. Il est obtenu en augmentant la vitesse des hélices à pas normal (A1 et A2) et en diminuant proportionnellement la vitesse des hélices à pas inversé (B1 et B2).

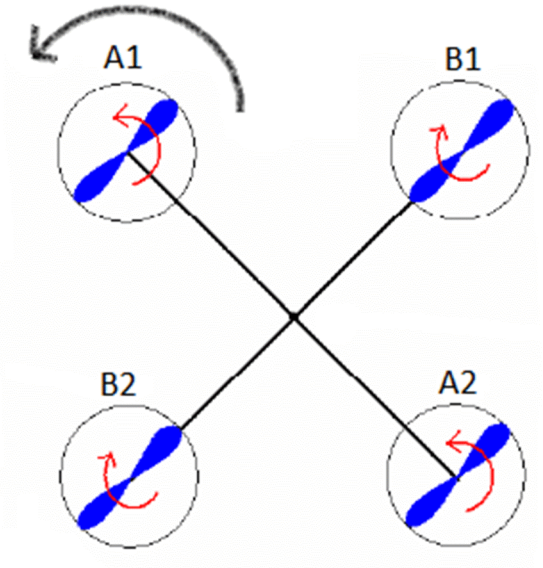


Figure I.6: Mouvement de lacet

I.4.3.La stabilisation :

Le problème de stabilisation est la principale difficulté pour le vol de notre drone, car même si les hélices tournent toutes à la même vitesse, des éléments (vend, poids mal équilibré, taille des bras différente, usure des moteurs….) provoquent des perturbations qui font que sur une simple accélération (volonté de faire monté l'appareil sans qu'il se déplace horizontalement) le drone va se déplacer. Donc le problème consiste à maintenir le drone parallèle au sol lorsqu'aucune commande de la télécommande n'est reçue, il se doit également de savoir pivoter sur ses axes pour pouvoir se déplacer. Ces rotations sont spécifiées par la station de commande au sol (la télécommande).

Ce problème nécessite l'utilisation d'un régulateur nous allons évoquer et utiliser le régulateur PID (Proportionnel Intégral Dérivé).

Le PID prend pour une entrée l'angle voulu et l'angle mesuré, par exemple lorsque la télécommande au sol n'envoie rien cette erreur correspond au défaut de parallélisme au sol. Des opérations mathématiques sont appliquées à cette erreur et déterminent les vitesses à envoyer aux quatre moteurs.

Le régulateur PID est un algorithme qui prend en charge une erreur. Il fait la somme d'une multiplication de cette erreur, d'une intégral de cette erreur et d'une dérivée de cette erreur, puis renvoie une nouvelle valeur.

Notre régulateur est dit en boucle fermée (figure I.7) car il s'auto régule lorsque l'erreur d'angle est grande, les moteurs concernés accélèrent et l'erreur se réduit. La sortie de l'algorithme influe donc sur son entrée.

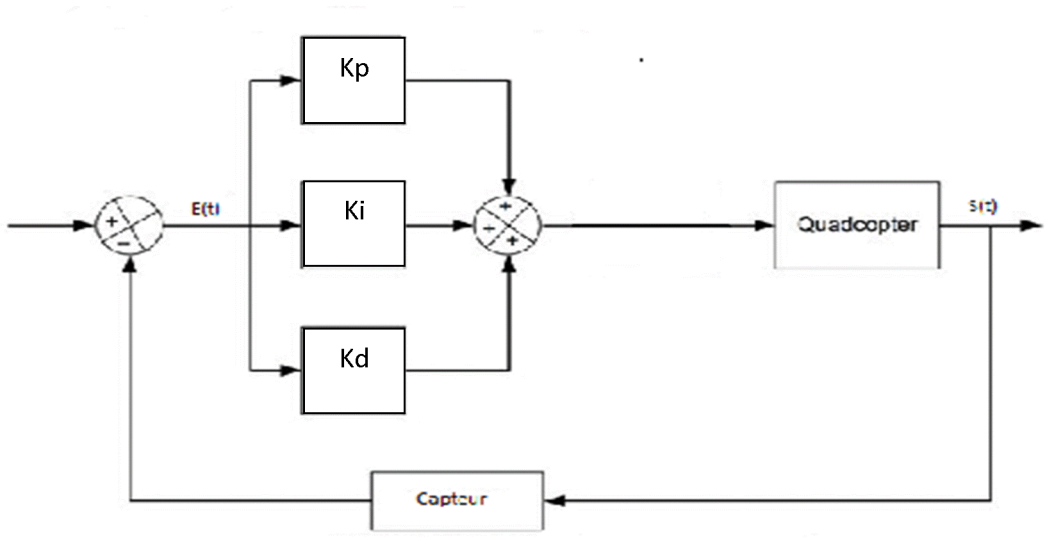


Figure I.7 : Schéma du régulateur PID

Action proportionnelle (P):

La sortie du régulateur est proportionnelle à l'erreur, elle est donnée par la formule suivante:

$$S(t) = Kp * E(t) = \frac{100}{Bp} * E(t)$$

Kp est le gain proportionnel

Bp%=Kp/100 est la bande proportionnel, elle est définie comme la variation en pourcentage de l'entrée du régulateur E nécessaire pour que la sortie S varie de 100%.

Action intégrale (I):

L'entrée de la commande est proportionnelle à l'intégrale de l'erreur.

L'erreur est intégrée et divisée par un gain Ki

$$S(t) = Ki * \int E(t)dt$$

Ki=1/Ti

Action dérivée (D):

C'est une action qui amplifie les variations brusque de la consigne. Elle a une action opposée à l'action intégrale. Cette fonction est remplie par l'opérateur mathématique : 'dérivé par rapport au temps'.

Elle est donnée par l'équation suivante :

$$S(t) = Kd * \frac{d}{dt} E(t)$$

Une fois le code du régulateur PID implémenté, il s'agit de déterminer les trois gains. Comme chaque quadri rotor a des caractéristiques particulières il n'existe pas de constante PID universelles.

Une méthode de détermination rapide, nommée méthode de Ziegler et Nichols permet d'obtenir des Ki et Kd corrects a partir de la seul valeur de Kp.

I.5.Conclusion :

Dans ce chapitre nous avons donné une présentation des drones quadri rotor, ainsi que le mode de fonctionnement de ces derniers qui est assez particulier.

CHAPITRE II
Conception matérielle

II.1.Introduction :

Dans ce chapitre nous nous intéressons à la conception matérielle de nos cartes électroniques (pour le drone et la télécommande).

Le choix du matériel est d'une importance majeur dans le développement d'un projet, de son rendement et de sa rentabilité.

II.2.Définition d'un microcontrôleur :

Un microcontrôleur est un composant électronique autonome dote :

- D'un microprocesseur.
- D'une RAM.
- D'une ROM.
- D'interfaces d'E/S parallèles et séries (RS232, I2C…..).
- D'interfaces d'E/S analogique.
- De Timers.

Les microcontrôleurs ont une architectures très similaires quelque soit leurs constructeurs, la différence réside dans le langage de programmation.

II.3.Le PIC 16F877A : [5]

Le 16F877A comme tout les PICs (Microcontrleur du constructeur Microchip) est conçue sur une architecture dite HARVARD et RISC, cette architecture est basée sur deux bus de données ; un bus pour les données et un autre pour les instructions. Cela permet d'exécuter l'instruction à l'adresse courante et de décoder l'instruction suivante (structure de type pipeline)

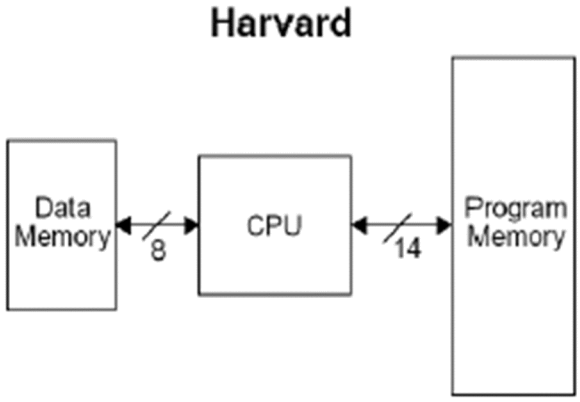


Figure II.1 : Architecture HARVARD du PIC 16F877A.

II.3.1.Caractéristiques du PIC 16F877A :

Le PIC 16F877A est constitue essentiellement des éléments suivants :

- Une mémoire programme de type EEPROM flash de 8K mots de 14 bits.
- Une RAM donnée de 368 octets.
- Une mémoire EEPROM de données de 256 octets (EEPROM DATA).
- Cinq ports d'entrée/ sortie, A (6 bits), B (8 bits), C (8 bits), D (8bits) et E (3bits).
- Convertisseur analogique 10 bits a 8 entrées (port A et E).
- USART, Port série universel, mode asynchrone (RS232) et mode synchrone.
- SSP, Port série synchrone supportant I2C.
- Un port parallèle esclave (psp).
- Trois TIMERS avec leurs Prescalers, TMR0, TMR1, TMR2.
- Deux modules CCP1 et CCP2 pour la comparaison, la Capture et la génération de la PWM (modulation de largeur d'impulsion).
- Un chien de garde (WDT).
- 14 sources d'interruption.
- Fonctionne a 20 MHZ maximum.
- Fonctionnement en mode sleep pour réduction de la consommation d'énergie.
- Programmation par mode ICSP (*In Circuit Serial Programming*) 12V ou 5V.
- Tension de fonctionnement de 2 a 5V.
- Jeux de 35 instructions.

II.3.2.Brochage et fonction des pattes :

- Brochage :

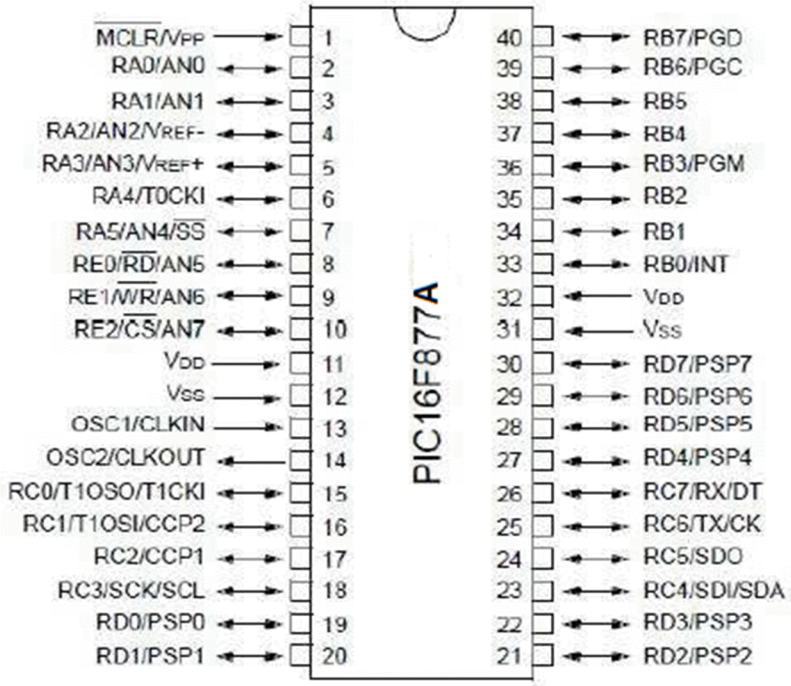


Figure II.2 : Brochage du PIC 16F877.

- Fonction des pattes :

/MCLR/Vpp : entrée du reset manuel ou la tension de programmation.
RA0 a RA7: sont les entrées/sorties du port A (bidirectionnel).
AN0 a AN7 : sont les entrées analogiques.
Vref- et Vref+ : entrées des tensions de référence pour le convertisseur analogique.
RE0 a RE2 : entrées/sorties du port E (bidirectionnel).
TOCKI : entrée d'horloge externe du timer0.
/RD : entrée, lecture en mode PSP.
/WR : entrée, écriture en mode PSP.
/CS : entrée, chip select en mode PSP.
VDD : alimentation positive du circuit.
VSS : est la masse du circuit.
OSC1/CLKIN et OSC2/CLKOUT : entrée /sortie d'oscillateur d'horloge.
RC0 a RC7 : entrées/sorties du port C (bidirectionnel).
T1OSI et T1OSO : sont l'entrée et sortie de l'oscillateur du timer1 respectivement.
T1CLKI : entrée d'horloge externe pour le timer1.
SCK : E/S d'horloge en mode SPI.
SCL : E/S d'horloge pour le bus I2C.
CCP1 et CCP2 : E/S pour les modes (compare, capture et PWM).
RD0 a RD7 : E/S du port D (bidirectionnel).
PSP0 a PSP7 : entrées/sorties du port parallèle psp.
RB0 a RB7 : entrées/sorties du port B (bidirectionnel).
SDI et SDO : entrée et sortie de données en mode SPI.
SDA : entrée/sortie de données pour le bus I2C.
INT : entrées d'interruption externe.
RX : est une entrée, pour la réception asynchrone SCI.
TX : sortie, pour l'émission asynchrone SCI.
DT : entrée/sortie de données synchrone SCI.
CK : entrée/sortie de l'horloge synchrone SCI.

II.3.3.Architecture interne du PIC 16F877A : [5]

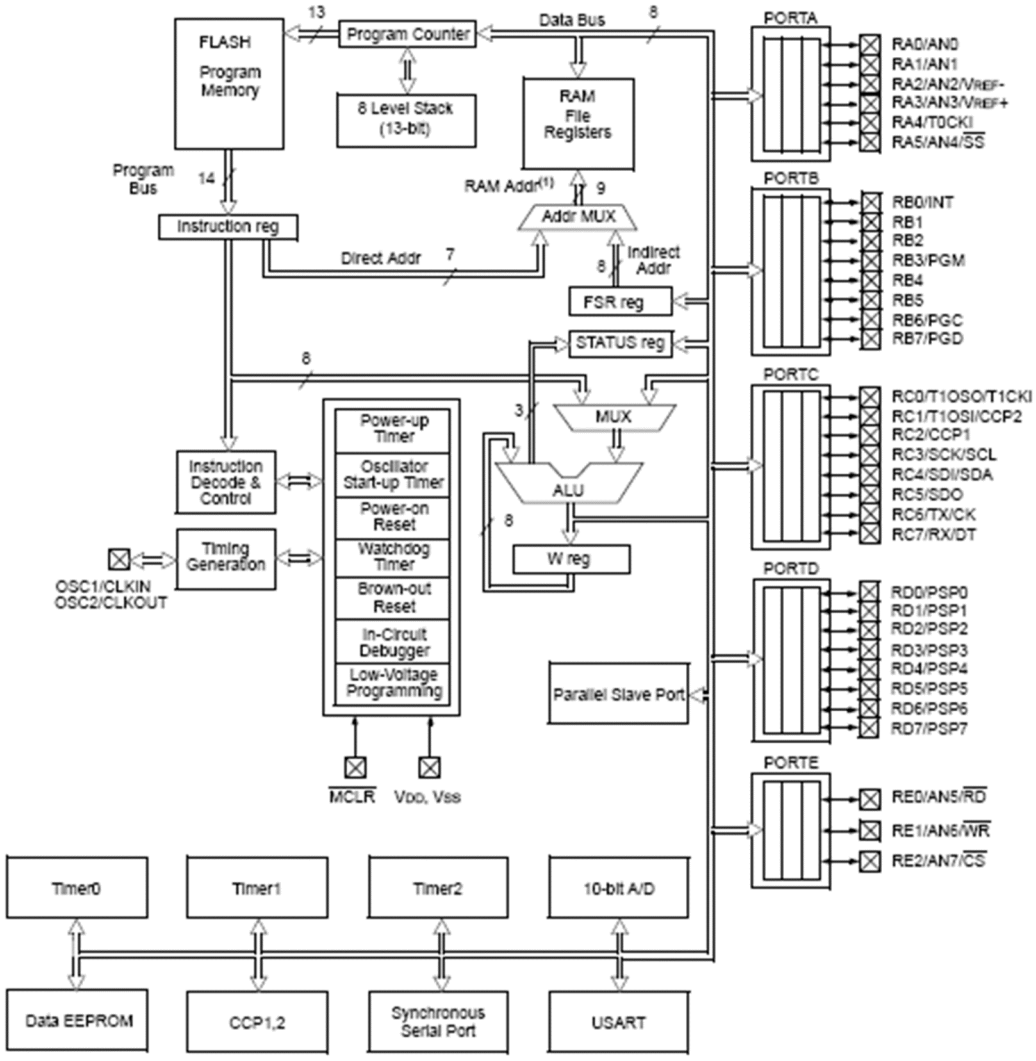


Figure II.3. Architecture interne du PIC 16F877A.

Le Microprocesseur du PIC se constitue des éléments suivants :

· Une unité arithmétique et logique (ALU).
· Un système de génération d'horloge a partir du quartz externe (timing génération)
· Une pile de 8 niveaux (Stack).
· Un compteur de programme (program counter).
· Registre d'instruction, il contient le code de l'instruction à exécuter.
· Un système d'initialisation a la mise sous tension (power up-timer, power on reset, Low-votage programming ……).
· Un chien de garde (watchdog).
· Un registre de travail nomme W, il est utilise pour l'adressage.
· Un registre d'etat.
· Un bus de données (data bus).
· Un bus pour le programme (program bus).
· Registres spéciaux (FSR), représentent l'ensemble des registres de configuration.

II.4.Gyro-accéléromètre 3 axes :

II.4.1.Définition :

Le gyro-accéléromètre 3 axes est un composant combinant un gyroscope 3 axes et un accéléromètre 3 axes, il est très utile pour détecter la vitesse angulaire pour pouvoir calculer l'angle de rotation.

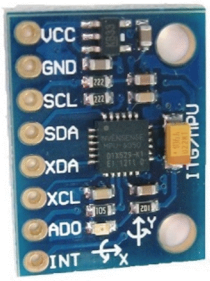


Figure II.4 : Le gyro-accéléromètre MPU6050

II.4.1.1.Le gyroscope :

Le gyroscope est un capteur de position angulaire, il donne la position angulaire selon deux ou trois axes de son référentiel par rapport à un référentiel inertiel (galiléen).

II.4.1.2.L'accéléromètre :

L'accéléromètre est un capteur qui, fixé à un mobile ou tout autre objet, permet de mesurer l'accélération linéaire de ce dernier ;il s'agit en fait de 3 accéléromètres qui calculent les 3 accélérations linéaire selon 3 axes orthogonaux.

Bien que l'accélération linéaire soit définie en m/s^2, la majorité des documentations sur ces capteurs exprime l'accélération en « g ».

II.4.2.caractéristiques techniques du gyro-accéléromètre 3 axes :

- Capteur de vitesse angulaire tri-axiale (gyroscope) avec une sensibilité allant jusqu'à 131 LSB/dps et une gamme a pleine échelle de ±250, ±500, ±1000, et ±2000dps.
- Accéléromètre tri-axiale pleine échelle programmable de ±2g, ±4g, ±8g, ±16g.
- Capteur de température à sortie numérique.
- Algorithmes embarqués pour la calibration de la boussole sans besoin d'intervention de l'utilisateur

II.5.Le Bluetooth HC05 :

II.5.1.Définition :

Le Bluetooth est un standard de communication permettant l'échange bidirectionnel de données à très courte distance et utilisant des ondes radio UHF (Ultra Haute Frequence) ;

Son objet est de simplifier les connexions entre les appareils électroniques en supprimant les liaisons filaires.

II.5.2.Caractéristiques technique :

Dans la version la plus répondue la liaison Bluetooth exploite les caractéristiques suivantes :

- Très faible consommation d'énergie.
- Très faible portée (sur un rayon de l'ordre d'une dizaine de mètres).
- Faible débit.
- Très bon marché et peu encombrant.

II.6.Les communications :

II.6.1.Communication I2C :

Le bus de communication I2C permet d'établir une liaison entre deux ou plusieurs composants.

Il a été crée dans le but d'établir des échanges d'information entre circuits intégrés se trouvant sur une même carte.son nom, d'ailleurs, traduit son origine : Inter Integrate Circuit, ou I, I, C, ou plus communément I2C.

Le bus I2C est constitué de 2 uniques lignes bidirectionnelles :

- La ligne SCL (Serial Clock Line), qui véhicule l'horloge de synchronisation, elle est gérée par le maitre

- La ligne SDA (Serial Data line), qui véhicule les bits transmis, elle est à un moment donné pilotée par celui qui envoi une information (maitre ou esclave)

Fonctionnement :

Le périphérique qui gère la communication est le maître, c'est lui qui génère l'horloge (SCL) et qui envoie les données (SDA) mis à part l'acknowledge (acquittement en français).

L'acquittement est un 'bit' envoyé par le composant esclave pour indiquer qu'il a bien reçu toutes les données ; si c'est le cas l'esclave impose le niveau 0, sinon la résistance de pull-up maintient la ligne à 1, on dit alors qu'il n'y a pas d'acknowledge.
(NACK qui veut dire "no acknowledge" en anglais)

Au début de la communication SDA passe de 1 à 0 alors que SCL reste à 1, c'est le StartBit. Après avoir imposé la condition de départ, le maître passe SCL à 0 puis l'applique

sur SDA le bit de poids fort. Il verrouille la donnée en appliquant pendant un instant un niveau 1 sur la ligne SCL.

Lorsque SCL revient à 0, il recommence l'opération avec le bit inférieur jusqu'à ce que l'octet complet soit transmis. Il redéfinit ensuite SDA comme une entrée et scrute son état ; l'esclave doit alors imposer un niveau 0 pour signaler au maître que la transmission s'est effectuée correctement, c'est l'acknowledge, la communication peut donc continuer. Si l'esclave n'envoie pas l'acknowledge les résistances de pull-up maintiennent la ligne à 1.

La communication peut alors être arrêtée, ou reprendre à zéro (cela dépend de la configuration), c'est le rôle du bit de STOP StopBit. Le StopBit indiquant la fin de la transmission par le maître est effectué en appliquant un passage de 0 à 1 de SDA alors que SCL reste lui à 1. Le premier octet envoyé est l'adresse, il est composé de 7bits variables selon le composant et du bit de read/write (0 pour write, 1 pour read).

Le second octet peut être le byte de contrôle sur certains composants, ou directement la donnée.

Exemple de communication I2C :

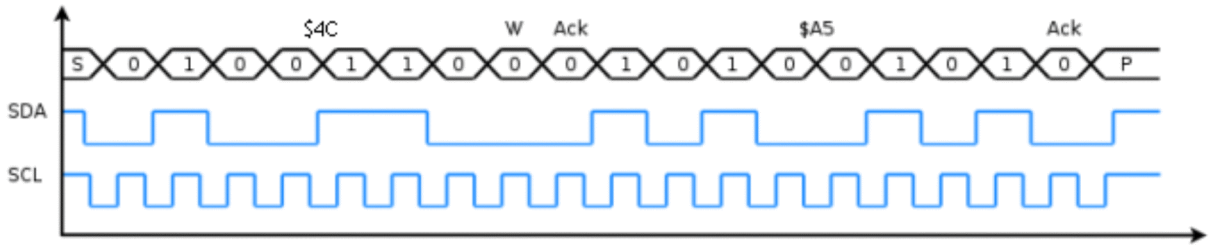


Figure II.5 : Communication I2C.

La communication commence par le StarBit
Puis l'adresse, (sur 8bits $4C) avec bit de read/write à 0
L'acknowledge (Ack)
Un octet de données ($A5)
De nouveau L'acknowledge (Ack)
Et enfin le StopBit

II.6.2.Communication USART (RS-232) :

Le bus RS-232 est un en bus de communication de type serie sur trois fils minimum (électrique, mécanique, protocole)

Fonctionnement :

Pour pouvoir dialoguer avec le PC, notre microcontrôleur utilise son module USART signifie (Universal Synchronous Asynchronous Reciever Transmitter).

C'est donc un module qui permet d'envoyer et de recevoir des données en mode série, soit de façon synchrone, soit asynchrone. Le module USART de notre PIC gère uniquement deux broches, à savoir RC6/TX/CK et RC7/RX/DT.

Une liaison série synchrone nécessite une connexion dédiée à l'horloge, donc il reste une seule ligne pour transmettre les données. Alors qu'en mode asynchrone on n'a pas besoin d'une ligne d'horloge, il nous restera alors deux lignes pour communiquer, chacune étant dédiée à un sens de transfert. Nous pourrons donc envoyer et recevoir des données en même temps.

Les liaisons RS-232 sont des liaisons asynchrones très utilisées en informatique. Elle nécessite que l'émetteur et le récepteur soit informé de la vitesse de transfert choisie (dans notre cas 9600 baud).

Puisque le récepteur connaît la vitesse du transfert il peut se passer de signal de synchronisation.

Trois lignes sont nécessaires à cette liaison.

1. TX : transmission de donnés.
2. RX : récepteur de donné.
3. GND : masse

II.7.La PWM (Pulse Wite Modulation)

La PWM (modulation de la largeur d'impulsion) est un signal binaire de frequence fixe dont le rapport cyclique peut être modulé par logiciel.

Il y a deux paramètres qui définissent un signal PWM :

- La durée d'un cycle complet (la frequence de répétition).
- Le rapport cyclique.

On a :

Tc : durée d'un cycle.

Rc : le rapport cyclique.

Th : durée de l'etat haut

Tb : durée de l'etat bas

Donc :

Tc=Th+Tb.

La frequence du signal (en Hertz)=1/Tc.

Rc=Th/Tc (rapport cyclique en %).

La figure (II.6) est exemple d'une PWM avec un rapport cyclique de 50%

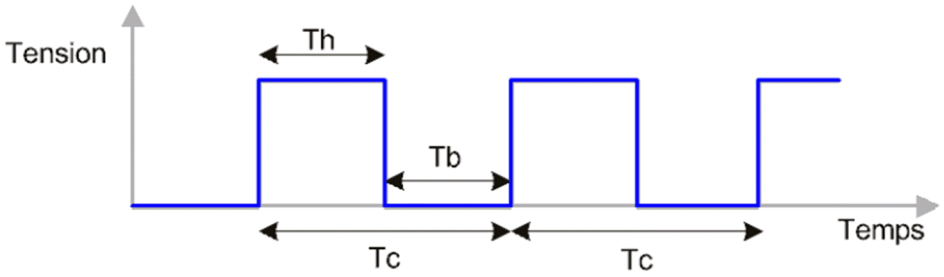


Figure II.6 : Modulation de la largeur d'impulsion.

II.8.Schéma synoptique de la carte :

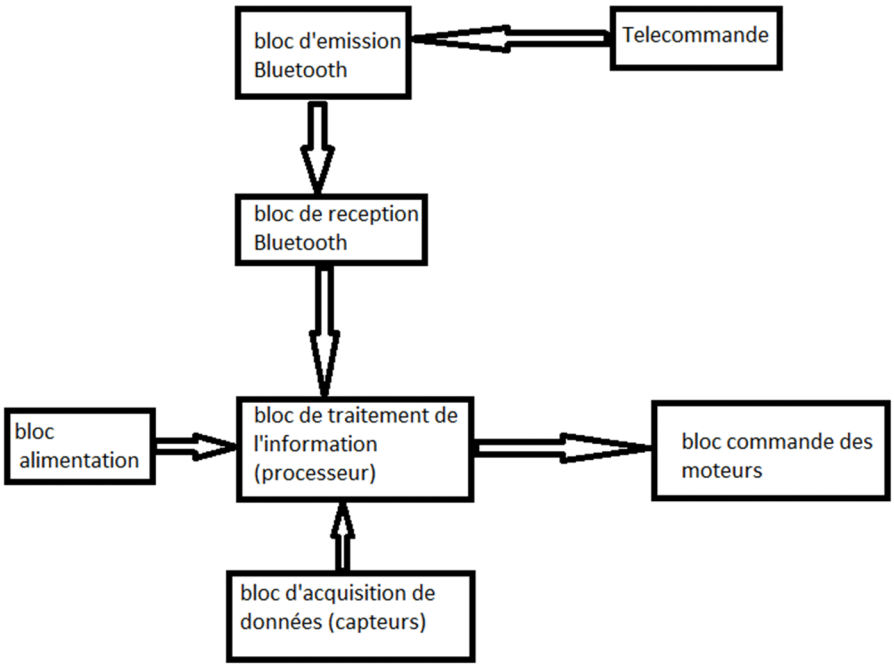


Figure II.7 : Schéma synoptique de la carte.

Ce schéma synoptique est traduit par la schéma de la carte électronique sur la figure (II.8) sur laquelle les différents bloc sont représentés en détails

Cette carte se compose de différents blocs qui sont :

- Bloc d'alimentation
- Bloc de traitement de l'information
- Emission Bluetooth
- Réception Bluetooth
- Bloc réception de données
- Bloc commande des moteurs

II.9 la carte électronique du drone :

II.9.1.Schéma de la carte :

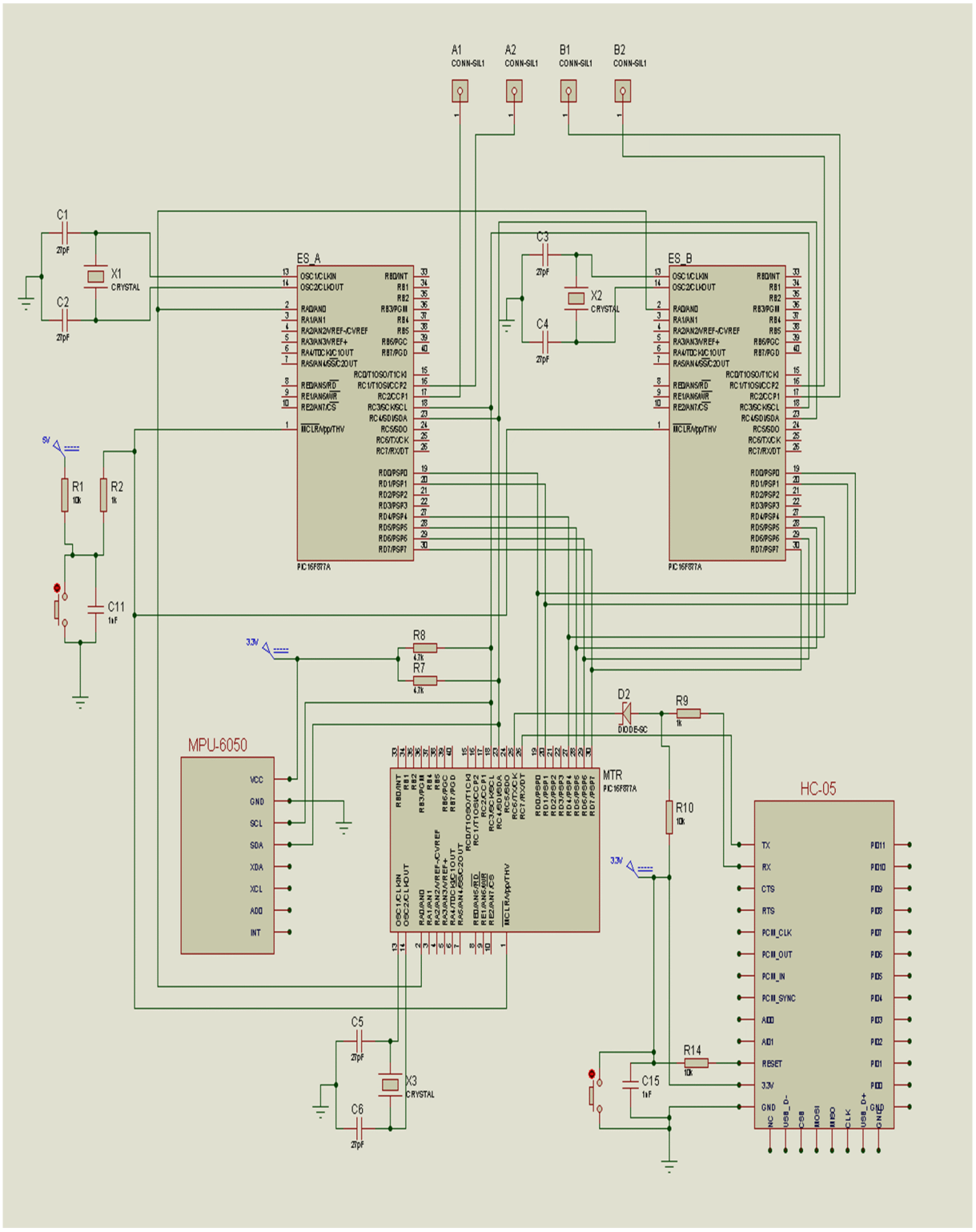


Figure II.8 : Schéma de la carte sous ISIS

II.9.2 : Constitution de la carte :

Les Microcontrôleurs :

- Le PIC maitre pour le traitement des données et la commande des PICs esclaves. Il est représenté su la figure (II.9)

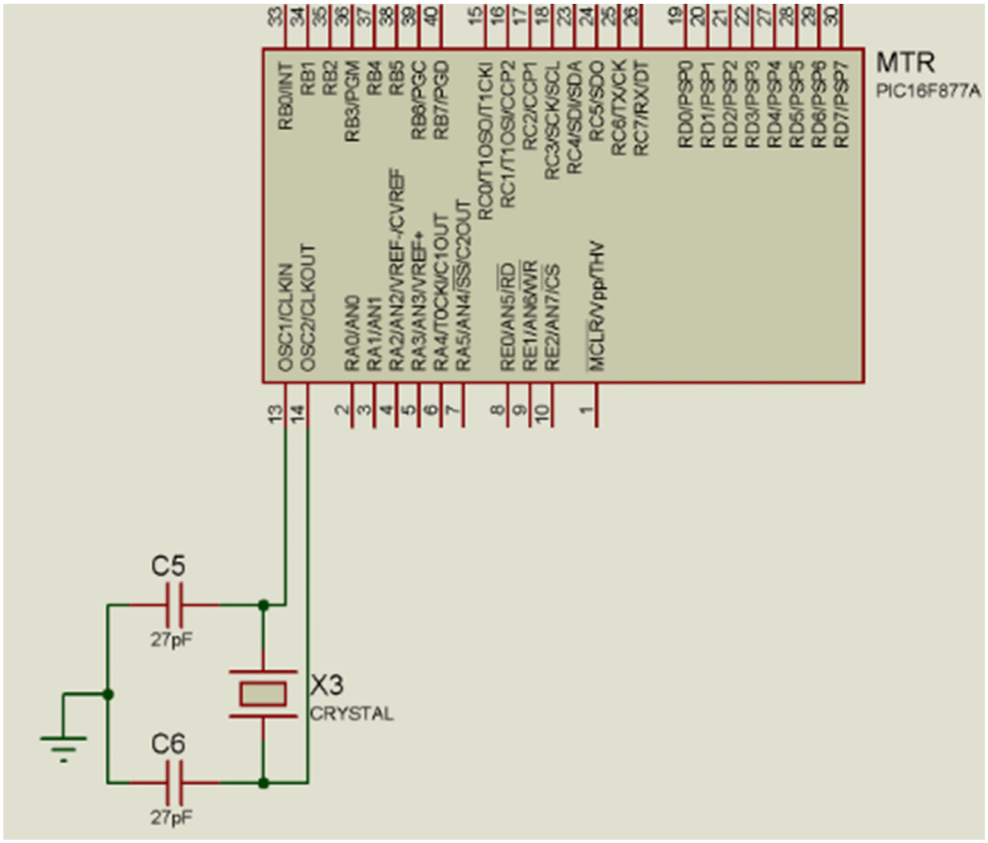


Figure II.9 : Schéma du PIC maitre.

- Les PICs esclaves qui commandent les moteurs A1, A2, B1 et B2.

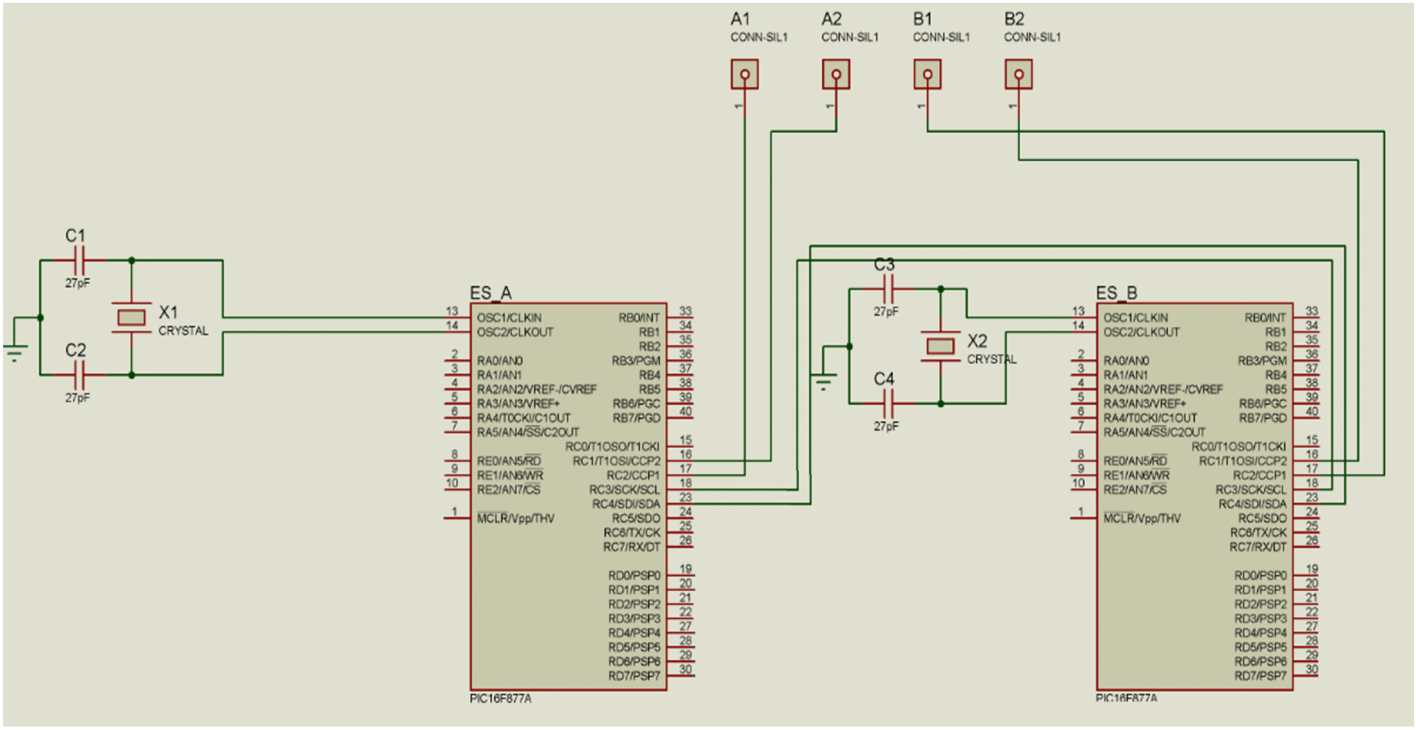


Figure II.10 : Schéma des PICs esclaves.

Le gyro_accéléromètre MPU6050 :

La figure (II.11) représente le MPU 6050 pour recueillir l'information et la transmettre au PIC maitre.

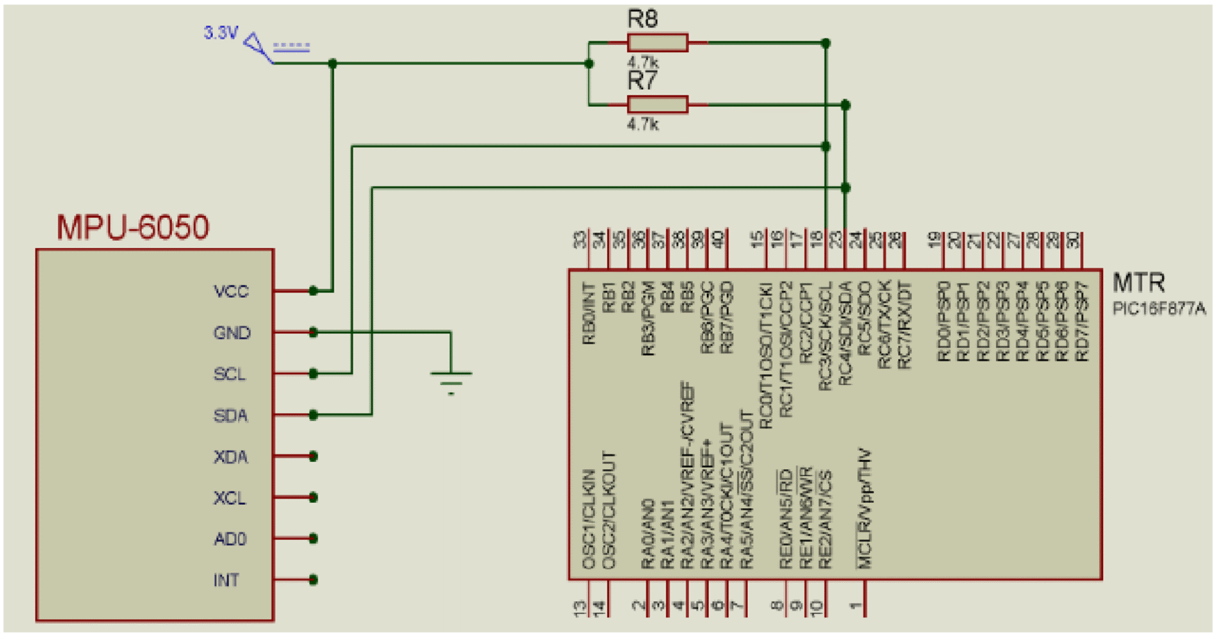


Figure II.11 : Le MPU 6050

Le récepteur Bluetooth HC05

La figure (II.12) représente le Bluetooth pour la réception de la commande de la télécommande vers la carte.

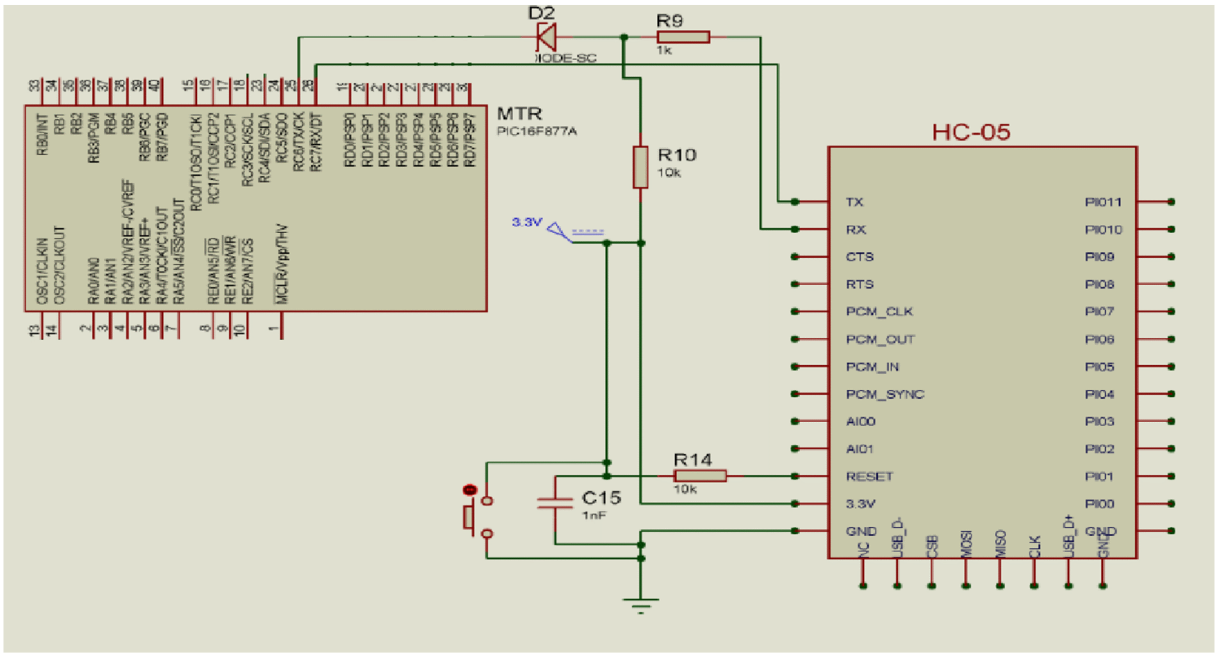


Figure II.12 : Le récepteur Bluetooth

Les circuits d'alimentation :

Pour le bon fonctionnement de notre carte nous avons besoin de deux tensions 5 volts (Régulateur de tension LM7805) pour l'alimentation les PICs et 3.3 volts(régulateur de tension LM317S) pour le Bluetooth et les capteurs.

- Le LM7805 transforme une tension continue de 9V en tension continue de 5.

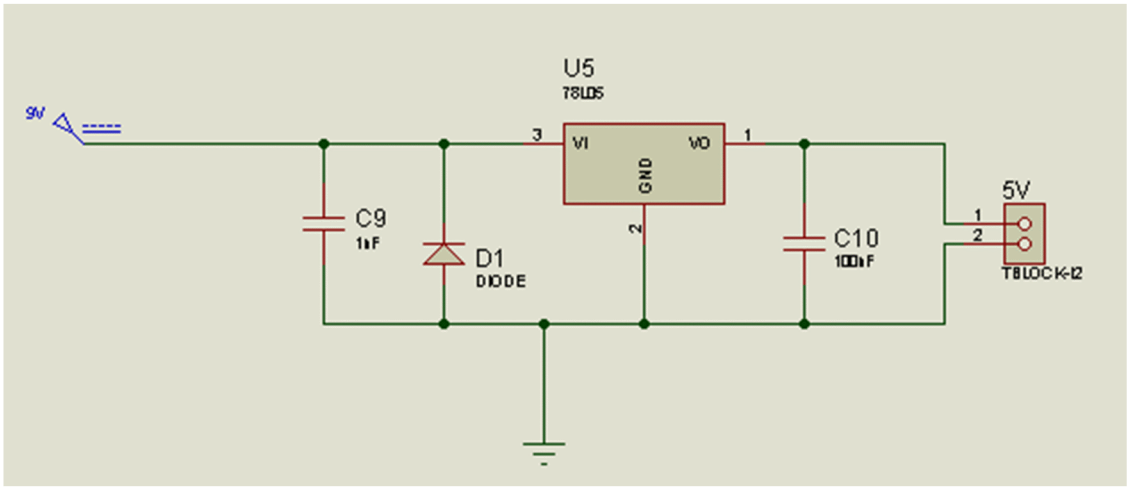


Figure II.13 : Le circuit d'alimentation LM7805.

Le LM7805 est un régulateur de tension qui délivre une tension de 5 volts pour une entrée entre 7.5 volts et 36 volts, donc pour avoir la tension voulu il suffit d'alimenter le régulateur comme le montre la figure (II.13)

Pour éviter le bruit (parasites), nous avons placé en amont du composant un condensateur polarisé (C9), ainsi qu'une diode en parallèle pour éviter les chocs électriques dus à un branchement inversé de la batterie.

- Le LM317S transforme la tension de 9V en 3.3V en ajoutant des résistances :

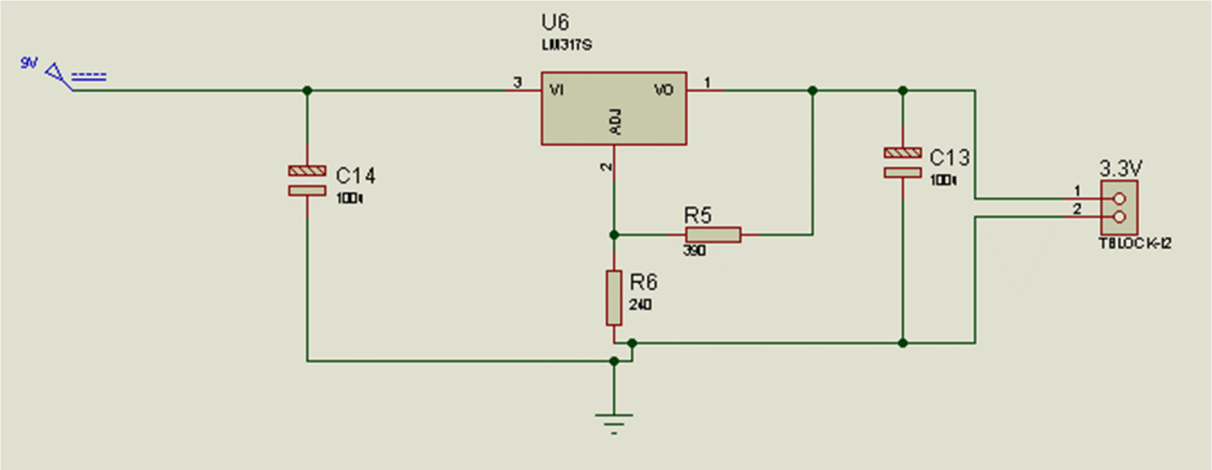


Figure II.14 : Le circuit d'alimentation LM3117S.

Circuit reset :

Le circuit reset permet d'effacer le contenu des différents ports et du registre d'état, qui est le seul registre à être effacer par le reset, la figure (II.15) représente le circuit.

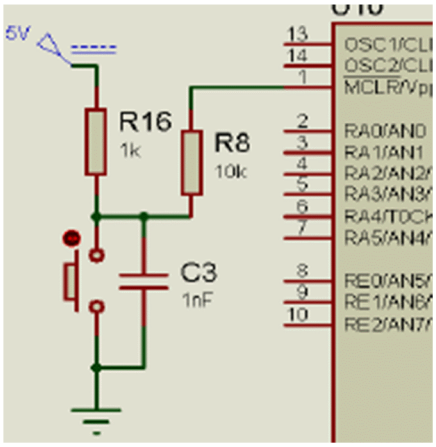


Figure II.15 : Le circuit reset.

Oscillateurs :

Il existe plusieurs oscillateurs :

LP : quartz faible puissance

XT : quartz ou oscillateur externe

HS : quartz ou oscillateur externe rapide

RC : résistance /capacité externe

RCIO : résistance/capacité externe (RA6 est port //)

Dans notre cas on a utilise un quartez 20MHz, ce qui permet de cadencer le PIC a une Frequence de 5MHZ, ce qui fait 5 million d'instructions par seconde. Cette vitesse est assurée par le circuit suivant :

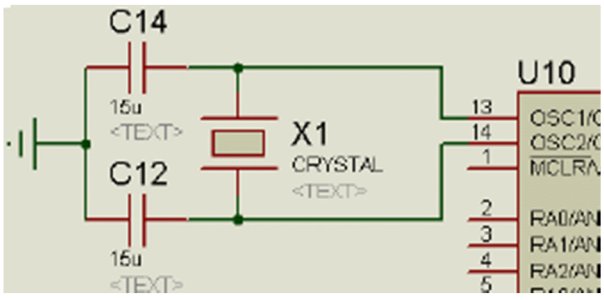


Figure II.16 : le circuit d'horloge.

II.10.La télécommande :

II.10.1.Schéma de la carte de la télécommande :

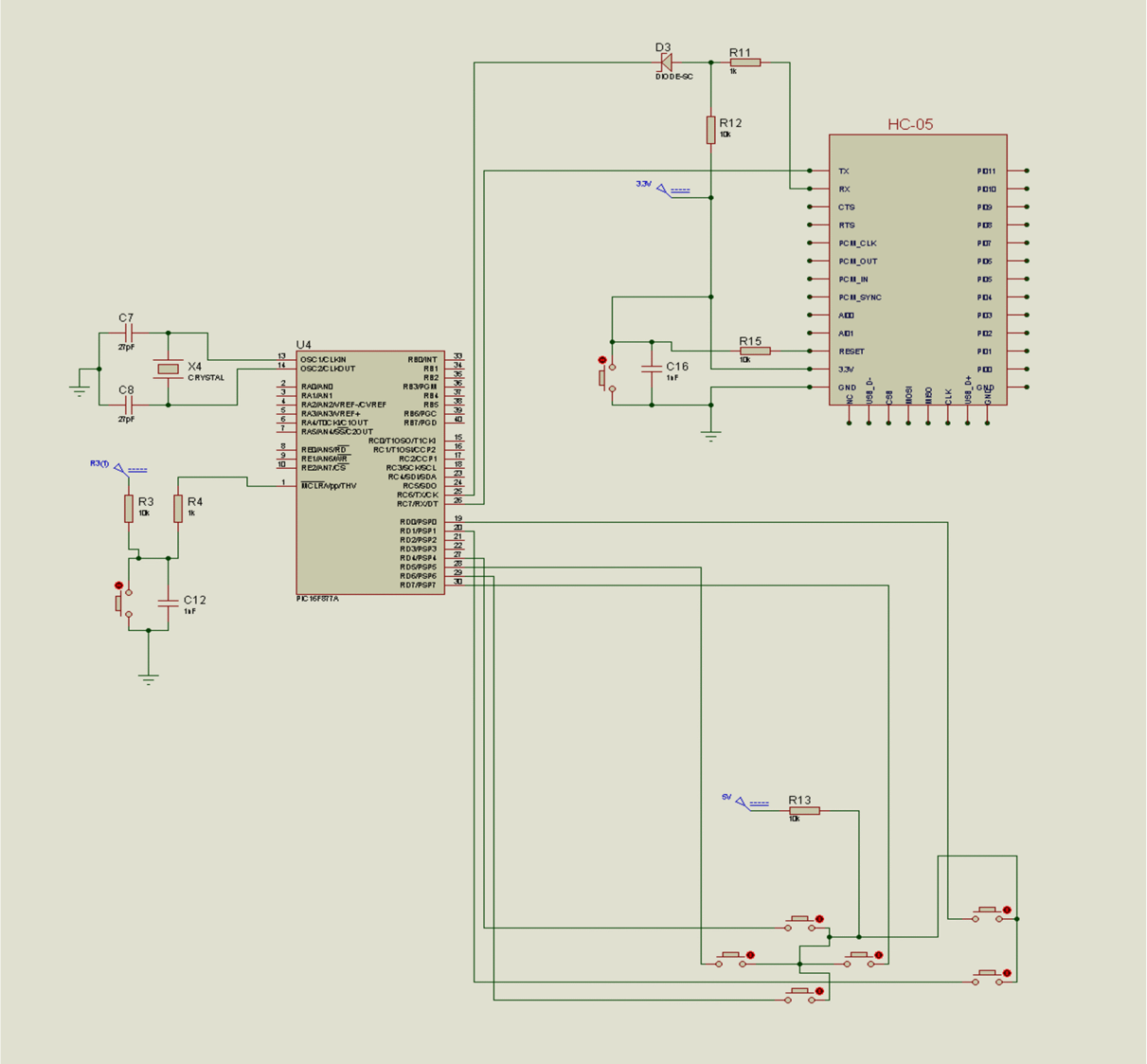


Figure II.17 : la carte de la télécommande.

II.10.2.Constitution de la carte de la télécommande :

La télécommande est constituée de six boutons poussoir pour les différentes directions, d'un circuit reset comme pour la carte figure (II.15), d'un Oscillateur avec un quartz de 20 MHz figure (II.16), d'un émetteur Bluetooth pour envoyer la commande.

II.11.Conclusion :

Dans ce chapitre nous avons présenté la conception matérielle de notre drone en donnant sa configuration électronique.qui elle, sans doute est une partie primordiale pour le bon fonctionnement de notre drone

CHAPITRE III
Conception logicielle de la carte

III.1.Introduction :

Une fois la conception matérielle terminée, nous allons concevoir dans ce troisième chapitre un programme sous forme d'organigrammes pour la commande de notre drone.

III.2.Organisation du programme élaboré :

La commande développée est divise en trois parties :

III.2.1.Partie déclaration des variables et des fonctions :

Cette partie comporte :

- Définition des constantes et des variables.
- Déclaration des fonctions.

II.2.2.Programme du maitre (MTR):

Ce programme comporte la gestion des deux esclaves, la lecture sur le capteur (MPU6050) et la réception des données émises par la télécommande via le Bluetooth.

II.2.3.Programme de la télécommande :

Ce programme permet de géré les commandes émises par l'utilisateur et les transformer en caractère, puis les envoyés par le port série RS-232 au Bluetooth qui transmet l'information au microcontrôleur principal (maitre)

II.2.4.Programme de l'esclave A (ES_A) :

Cet esclave contrôle la vitesse de rotation des deux moteurs A1 et A2 qui tournent dans le même sens (sens anti horaire).

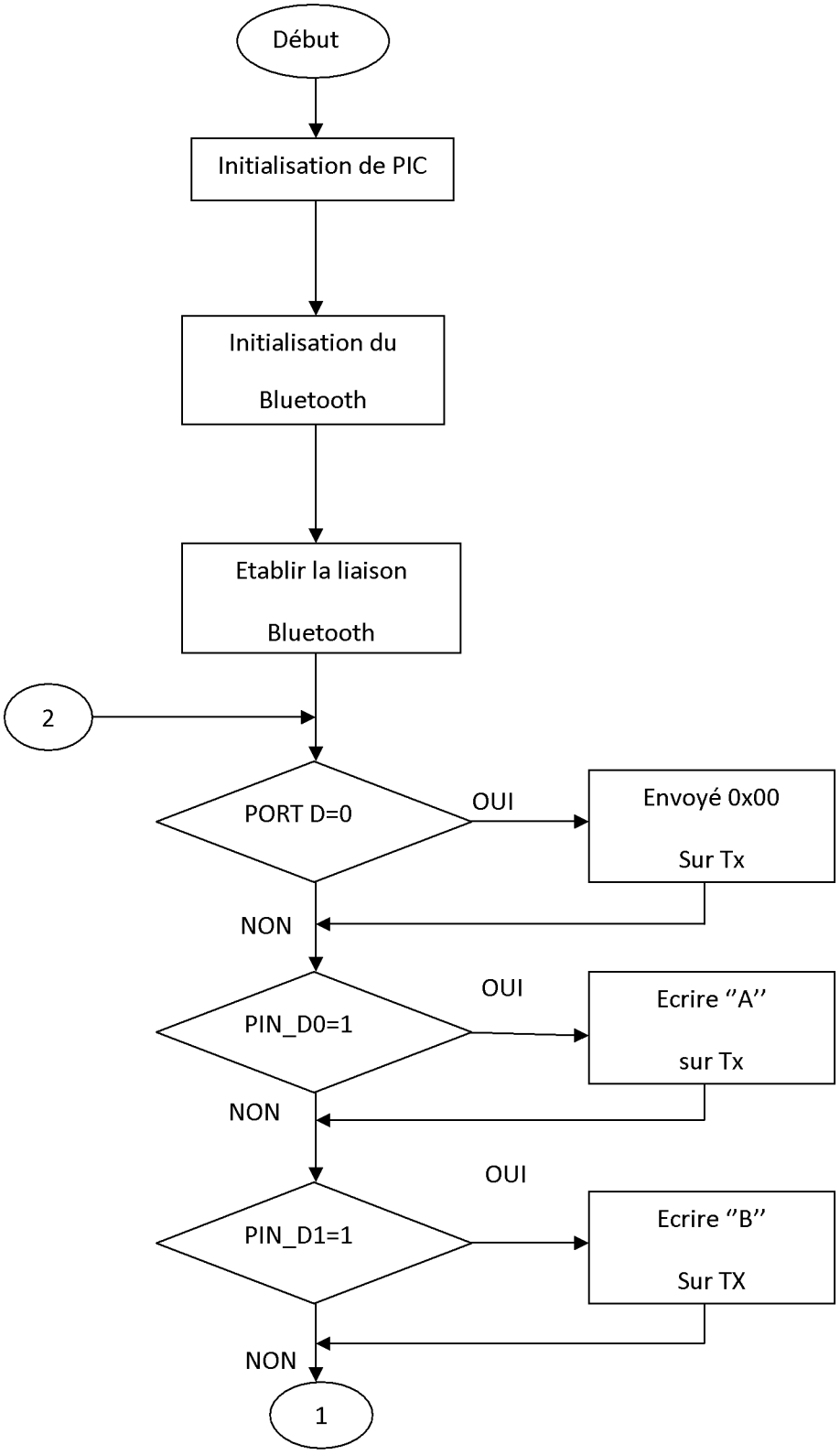
II.2.5.Programme de l'esclave B (ES_B) :

Cet esclave contrôle la vitesse de rotation des deux moteurs B1 et B2 qui tournent dans le même sens (sens horaire).

III.3.Organigrammes des programmes a exécuté :

Organigramme de la télécommande :

e



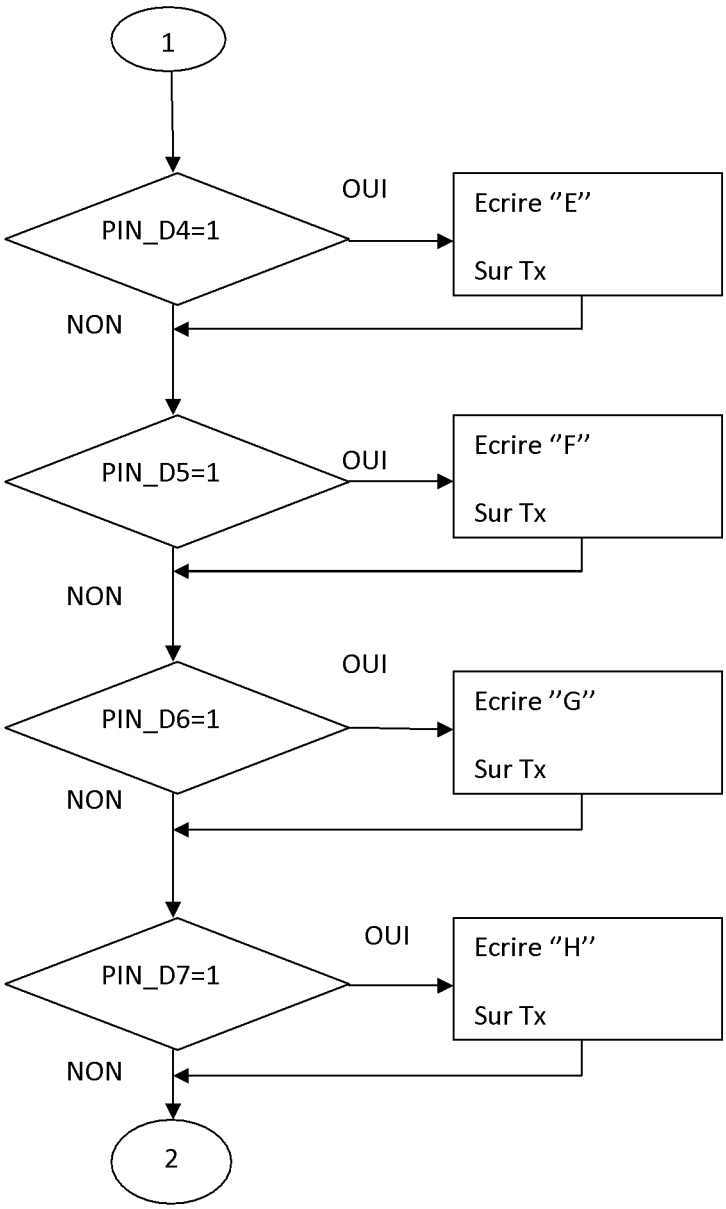
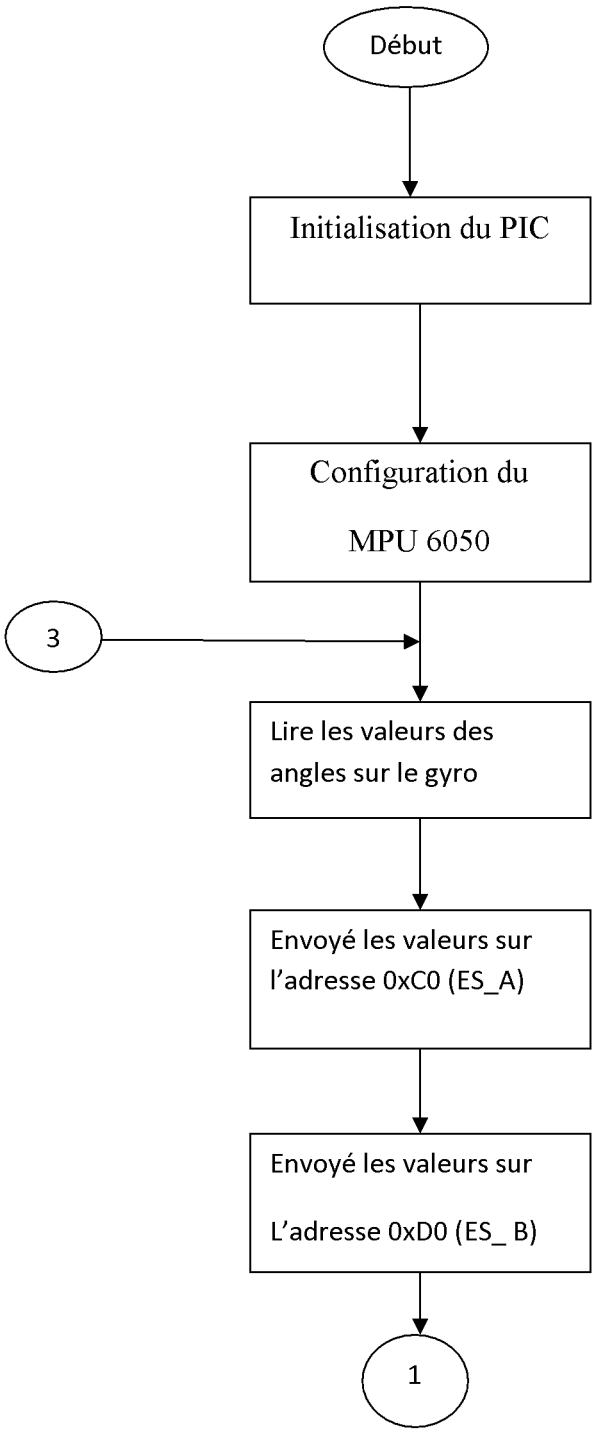


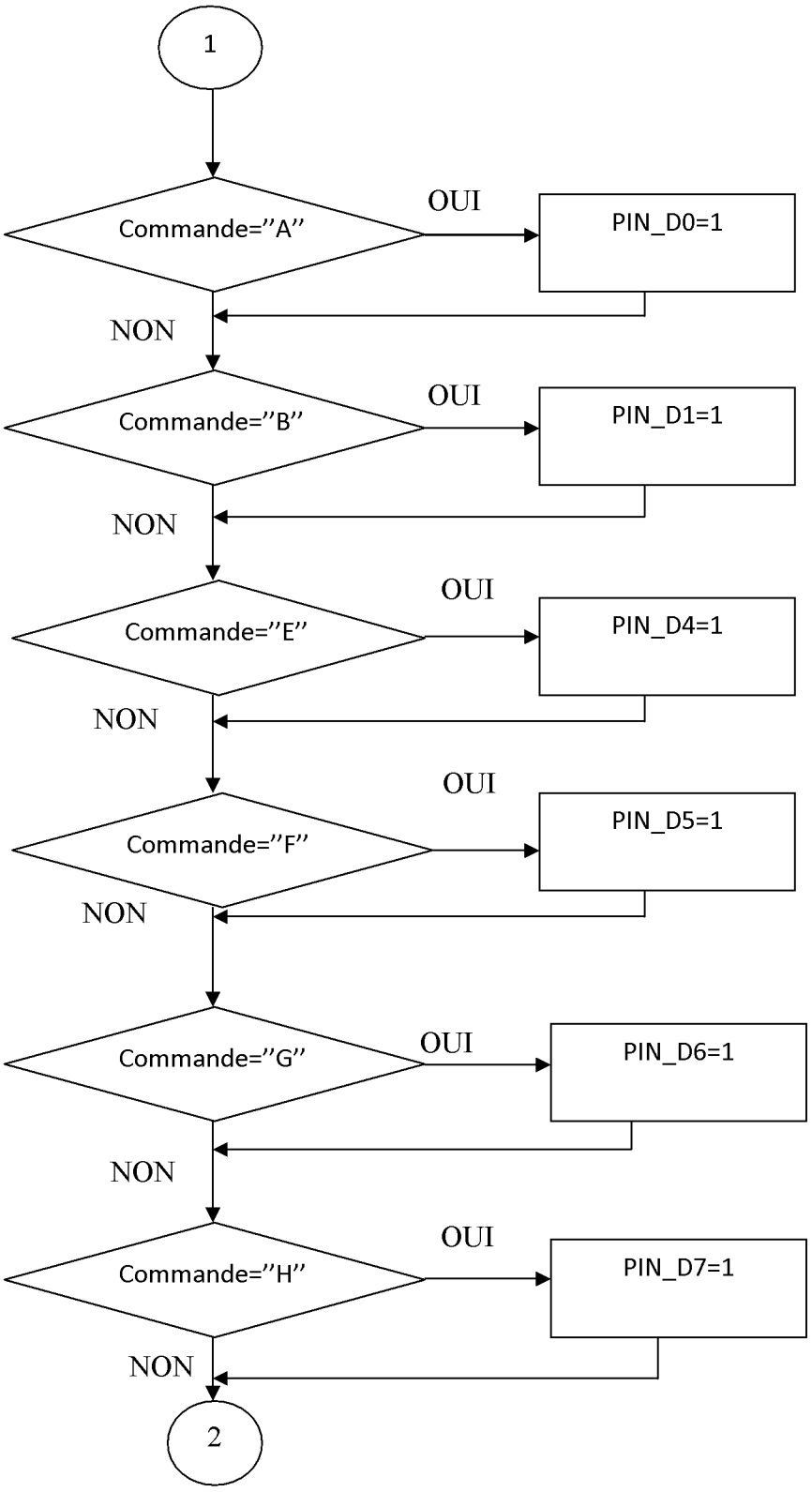
Figure III.1 : Organigramme de la télécommande

Lorsqu'on appui sur une touche (Avant, arrière…)le PIC de la télécommande reçois un signal (+5 volts) sur le Port D, Le PIC envoi un caractère différent selon la touche appuyé vers le Bluetooth par le bus Tx de RS_232 qui le transmet vers le PIC maitre du drone.

- **Organigramme du PIC maitre :**

Le maitre va configurer le MPU6050 avant de le lire a chaque 3.28 ms car 1cycle=0.2us 256cycle=51.2us On a utilisé un presdiviseur de 64 donc 256cycle=51.2us*64=3.2768ms Donc on obtient une frequence d'échantillonnage de 305.18Hz par interruption du Timer0. Le maitre calcul la moyenne de chaque 20 valeur pour minimiser l'erreur, pui il envoi la valeur des angles dans chaqu'un des esclaves par bus I2C en sélectionnant l'adresse de chaque esclave. Il active le pin du Port D spécifier à chaque caractère reçu depuis la télécommande.




1
Commande=''A''
OUI
PIN_D0=1
NON
Commande=''B''
OUI
PIN_D1=1
NON
Commande=''E''
OUI
PIN_D4=1
NON
Commande=''F''
OUI
PIN_D5=1
NON
Commande=''G''
OUI
PIN_D6=1
NON
Commande=''H''
OUI
PIN_D7=1
NON
2

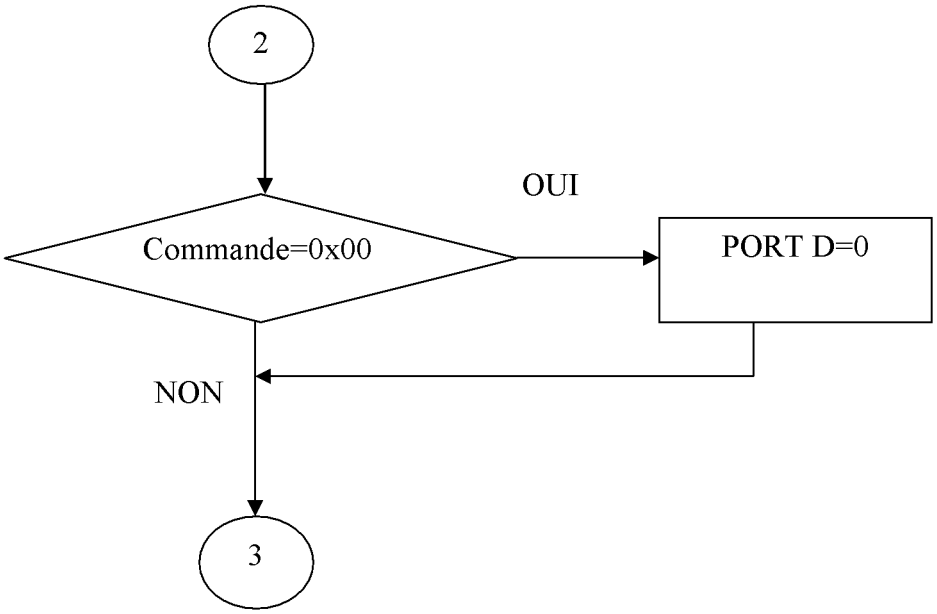
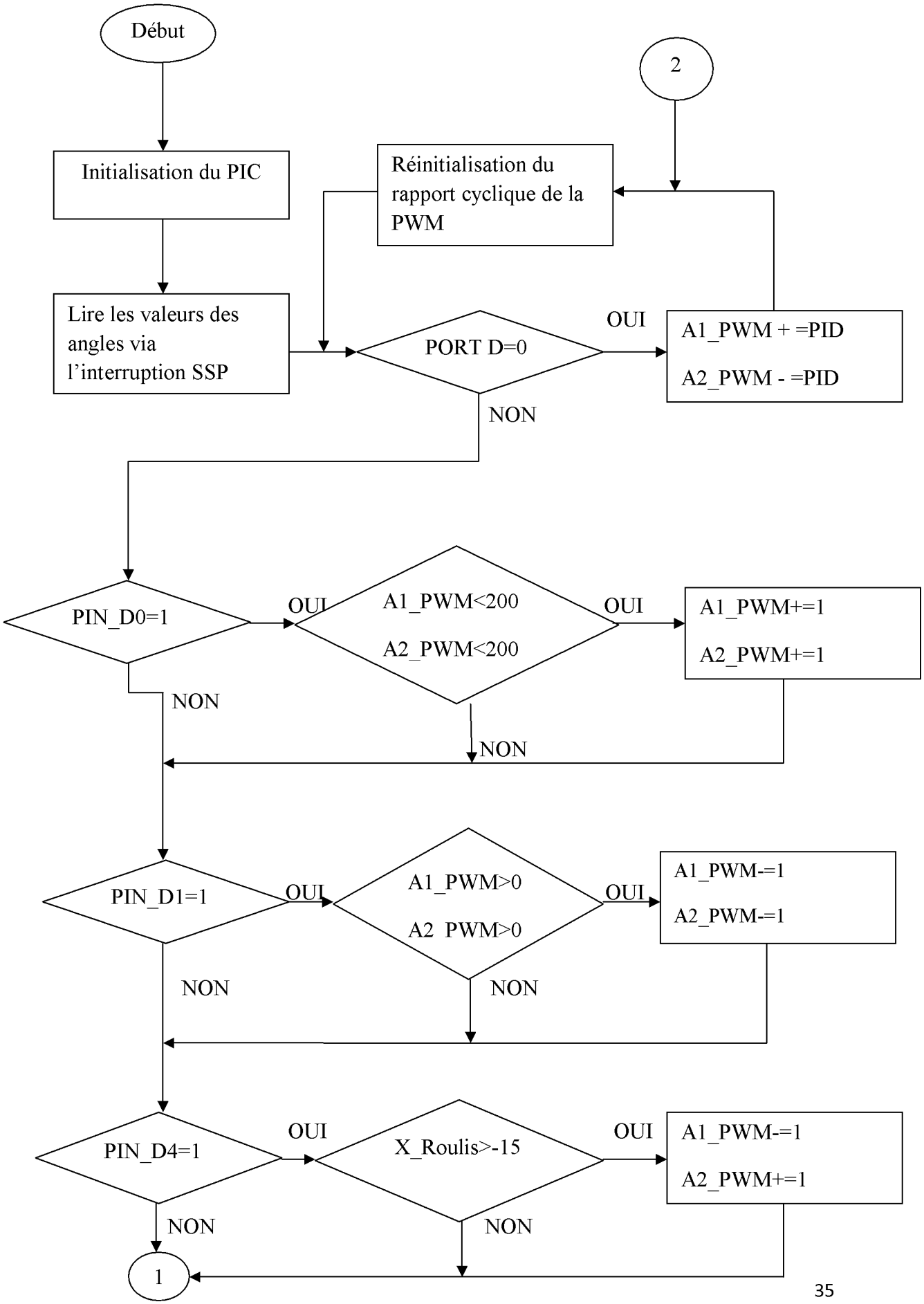


Figure III.2 : Organigramme du maitre

- **Organigramme de l'esclave A :**



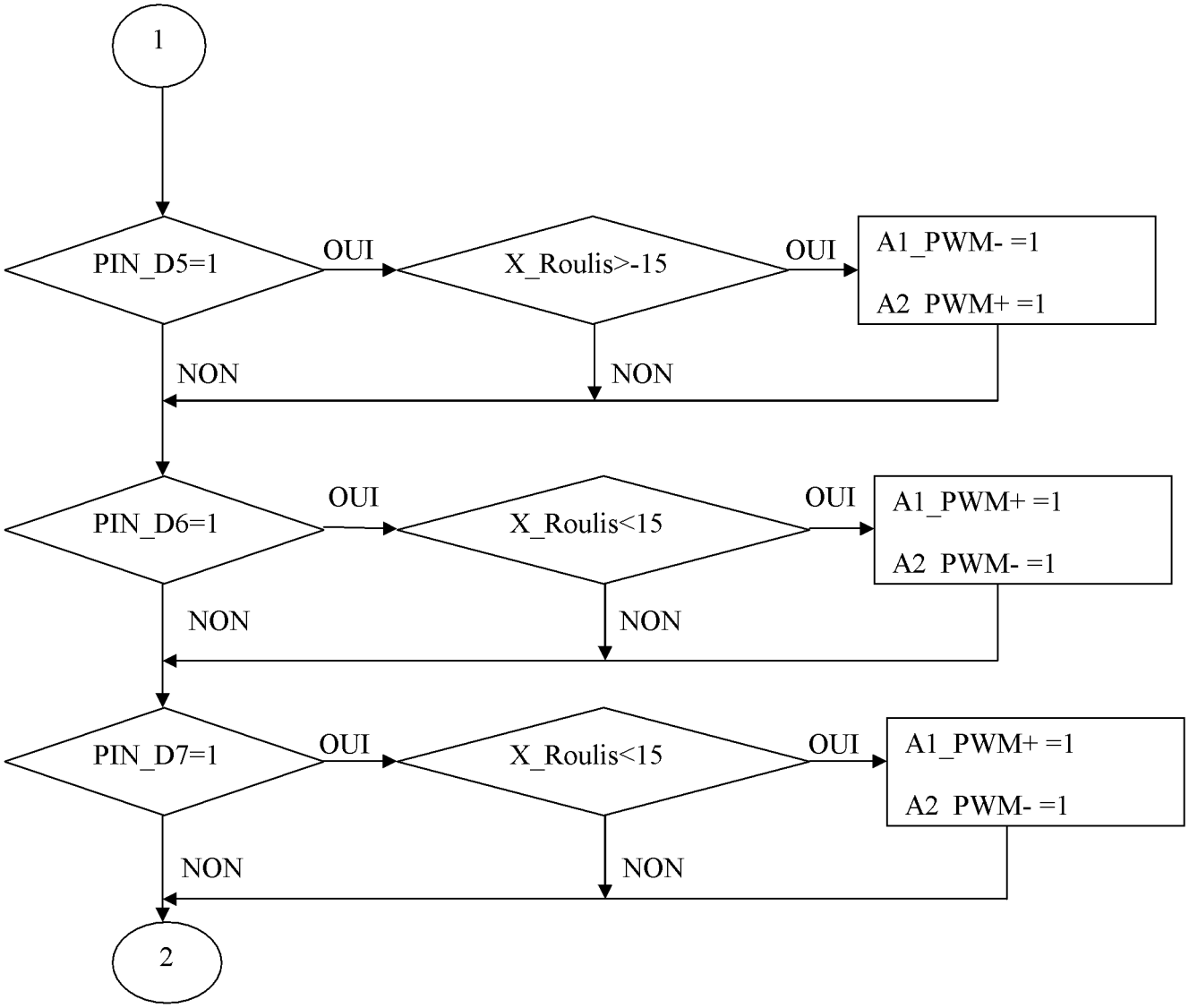
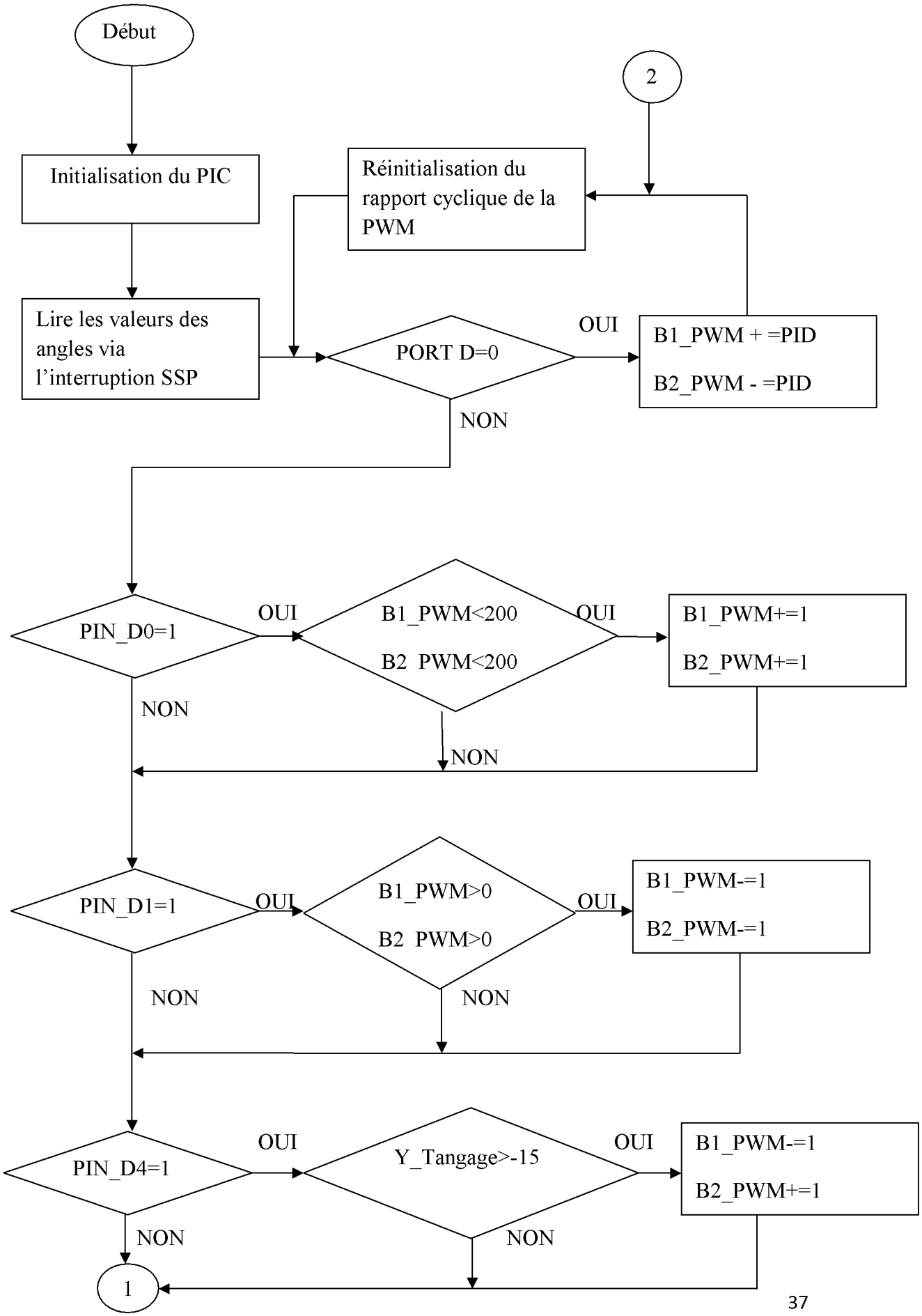


Figure III.3 : Organigramme de l'esclave A.

- **organigramme de l'esclave B :**



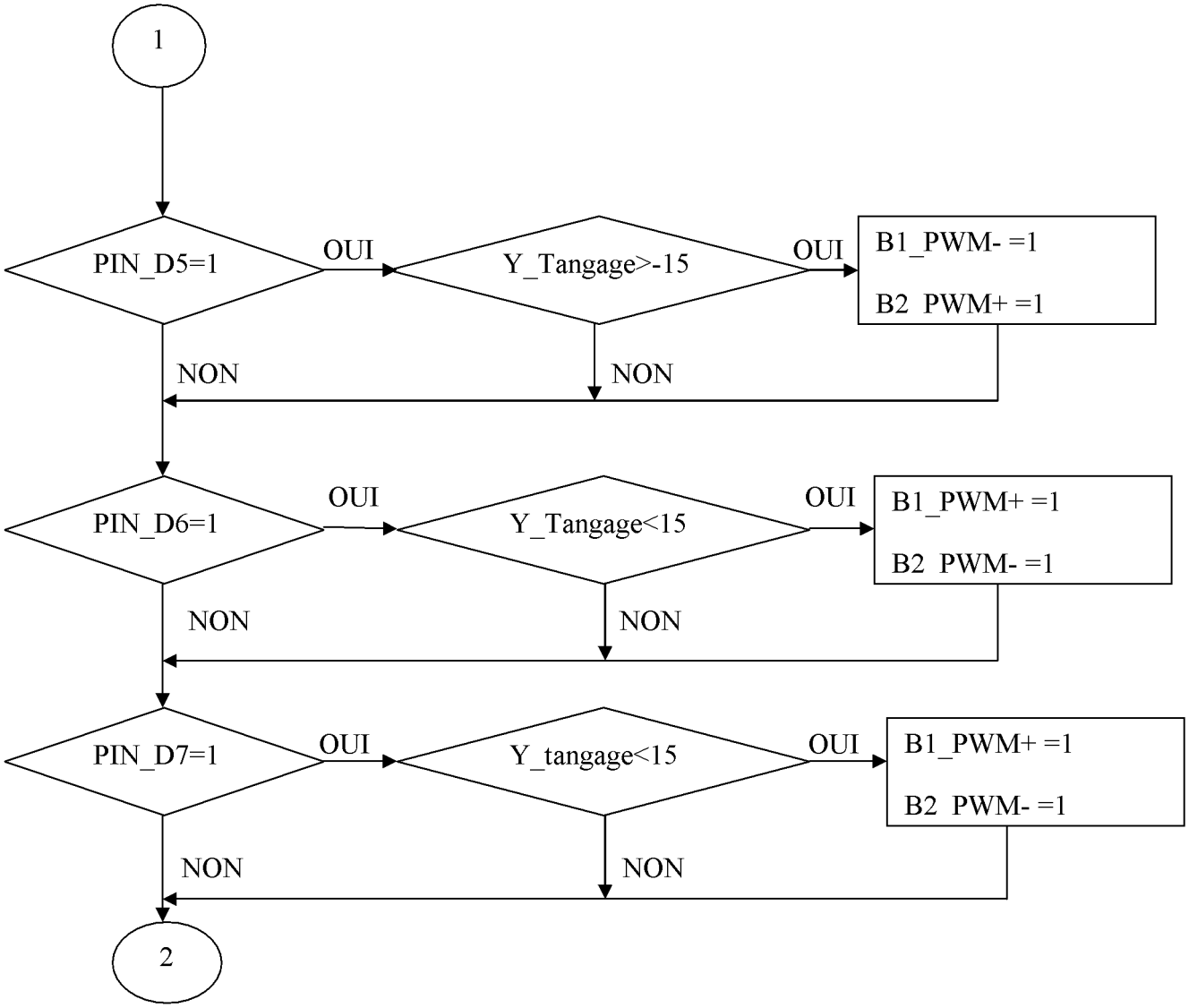


Figure III.4 : Organigramme de l'esclave B.

Les esclaves A et B reçoivent les commandes par le Port D et les valeurs des angles par bus I2C.

En recevant la commande ils agissent sur la PWM pour commandé les moteurs tout en préservant un angle (tangage et roulis) situé entre -15° et +15°

En n'ayant reçu aucune commande les PICs vont calculer l'erreur de parallélisme au sol et la corriger (par PID) en cas de non parallélisme.

III.4.Conclusion :

Dans ce chapitre nous avons essayé d'expliquer le fonctionnement de notre drone en présentant le programme de gestion du drone sous forme d'organigrammes.

Conclusion générale :

Ainsi nous avons lors de ce travail pu concevoir un modèle de cartes pour un quadri rotor télécommandé à base de PICs 16F877A, ainsi qu'une commande adéquate. Ce travail nous a permis de relever un défie de conception électronique, mais aussi d'apprendre de nouvelles notions dans ce domaine.

L'élaboration de ce projet regroupe plusieurs parties, notamment l'étude et le choix de l'ensemble des éléments constituant le système et la programmation, ainsi nous avons eu l'occasion au cours de ce travail d'étudié, concevoir et utilisé plusieurs matériels et logiciels (programmation en c, conception et simulation avec Proteus, les commandes AT……).

Pour la suite de notre projet il est nécessaire de se demander quelles performances doit réaliser le drone. En effet le drone doit de posséder une bonne stabilité mais aussi une bonne manœuvrabilité. Dans notre projet nous nous sommes seulement intéressés à la commande.

Au terme de ce travail quelques problèmes comme le calcule des paramètres PID qui nécessite d'avoir un système existant.

ANNEXES

ANNEXE A : Les programmes élaborés

- **Programme du maitre :**

```
#include <16F877A.h>
#device adc=8
#FUSES NOWDT                    // pas de Watch Dog Timer
#FUSES HS                       // Osc a grand vitesse (20mhz)
#FUSES NOPUT                    // No Power Up Timer
#FUSES NOPROTECT                // pas de protection de la lecture du programme
#FUSES NODEBUG                  // pas de Debug en mode ICD
#FUSES NOBROWNOUT               // pas de brownout reset
#FUSES NOLVP                    // No low voltage prgming, B3(PIC16) used for I/O
#FUSES NOCPD                    // pas de protection de la EEPROM
#FUSES WRT_50%                  // 50% de la memoire du programme est proteger
#use delay(clock=20000000)
#use i2c(master, fast, sda=PIN_C4, scl=PIN_C3)
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
#DEFINE ACCEL_XOUT_H 0x3B
#DEFINE ACCEL_XOUT_L 0x3C
#DEFINE ACCEL_YOUT_H 0x3D
#DEFINE ACCEL_YOUT_L 0x3E
#DEFINE ACCEL_ZOUT_H 0x3F
#DEFINE ACCEL_ZOUT_L 0x40
#DEFINE GYRO_XOUT_H 0x43
#DEFINE GYRO_XOUT_L 0x44
#DEFINE GYRO_YOUT_H 0x45
#DEFINE GYRO_YOUT_L 0x46
#DEFINE GYRO_ZOUT_H 0x47
```

```
#DEFINE GYRO_ZOUT_L 0x48
#DEFINE WHO_AM_I 0x68
#DEFINE PWR_MGMT_1 0x6B
#DEFINE ESCLV1 0xC0
#DEFINE ESCLV2 0xD0

char commande=0x00;

int16 gyro_x;                    // declaration des variables sur 16 bits
int16 gyro_y;
int16 gyro_z;

int8 gyro_x_L;                   // declaration des valeurs sur 8 bits
int8 gyro_x_H;
int8 gyro_y_L;
int8 gyro_y_H;
int8 gyro_z_L;
int8 gyro_z_H;

float vit_ang_x;                 // declaration des variables en valeur reel
float vit_ang_y;
float ang_cal_x;
float ang_cal_y;
float ang_x;
float ang_y;
```

```
int count=0;


void ecrire_reg(int8 gyro_adrss, int8 reg_adrss, int8 valeur)
{
  i2c_start();                                    // debut de la transmission
  i2c_write(gyro_adrss);                          // envoyer l'adresse du capteur
  i2c_write(reg_adrss);                           // envoyer l'adresse du registre
  i2c_write(valeur);                              // envoyer la valeur a ecrire
  i2c_stop();                                     // arreter la transmission
}
int8 lire_reg(int8 gyro_adrss, int8 reg_adrss)
{
  int valeur;
  i2c_start();                                    // commencer la transmission I2C
  i2c_write(gyro_adrss);                  // pour ecrire dans le gyro l'adresse du registre
  i2c_write(reg_adrss);                           // le registre a lire
  i2c_start();                                    // on redemare la connection I2C
  i2c_write(gyro_adrss + 1);              // maintenent on met le gyro en mode lecture
  valeur = i2c_read();                            // on va lire dans le gyro
  i2c_stop();                          // on arrete la communication I2C
  return valeur;                       // on renvoie la valeur du registre
}
void ecrire_esc(int8 esclave_adrss, int8 valeur)
{
  i2c_start();
  i2c_write(esclave_adrss);
  i2c_write(valeur);
```

```
  i2c_stop();
}
void moyenne()                                    // calcule la moyenne de chaque 20
valeurs
{
count+=1;
ang_x+=ang_cal_x;
ang_y+=ang_cal_y;

 if(count==20)                                    // envoi des donnees chaque 65,536 ms
 {
  ecrire_esc(ESCLV1,ang_x/20);
  ecrire_esc(ESCLV2,ang_y/20);
  count=0;
 }
}
void initialisation()
{
  ecrire_reg(WHO_AM_I,PWR_MGMT_1,0x80);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_64);
// activer TMR_0 et le diviseur sur 64
}
#INT_TIMER0                                       // interruption sur TMR_0
void tmr_0()                    ///////1cycle=0,2us////256cycle = 51,2us * 64 = 3276,8us =
3,2768ms ///// freq = 305,17578125hz /////
```

```
{
gyro_x_L=lire_reg(WHO_AM_I, GYRO_XOUT_L);
                                                    // lire les valeurs sur le gyro en 8 bits
gyro_x_H=lire_reg(WHO_AM_I, GYRO_XOUT_H);
gyro_y_L=lire_reg(WHO_AM_I, GYRO_YOUT_L);
gyro_y_H=lire_reg(WHO_AM_I, GYRO_YOUT_H);
gyro_z_L=lire_reg(WHO_AM_I, GYRO_ZOUT_L);
gyro_z_H=lire_reg(WHO_AM_I, GYRO_ZOUT_H);

gyro_x=make16(gyro_x_H,gyro_x_L);                    // convertir les valeurs sur 16 bits
gyro_y=make16(gyro_y_H,gyro_y_L);
gyro_z=make16(gyro_z_H,gyro_z_L);

vit_ang_x=gyro_x/131;                                // calcule de la vitesse angulaire °/s
vit_ang_y=gyro_y/131;

ang_cal_x=vit_ang_x*0.0032768;                  // temps d'echentiollannage = 0.0032768s;
ang_cal_y=vit_ang_y*0.0032768;

moyenne();                                              // appel de la fonction moyenne
}
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
```

```
setup_timer_2(T2_DISABLED,0,1);
setup_comparator(NC_NC_NC_NC);
setup_vref(FALSE);
enable_interrupts(GLOBAL);
enable_interrupts(INT_TIMER0);
initialisation();
while(1)
{
commande=getc();                    // recevoir le charactere envoyer par la telecommande

if(commande=='A')                           // activer le pin correspondant au charactere
{
output_high(PIN_D0);
}
else if(commande=='B')
{
output_high(PIN_D1);
}
else if(commande=='E')
{
output_high(PIN_D4);
}
else if(commande=='F')
{
output_high(PIN_D5);
}
else if(commande=='G')
```

```
  {
  output_high(PIN_D6);
  }
  else if(commande=='H')
  {
  output_high(PIN_D7);
  }
  else{
     output_d(0x00);
     }
  }
}
```

- **Programme de la télécommande :**

```
#include <16F877A.h>
#device adc=8

#FUSES NOWDT                    //No Watch Dog Timer
#FUSES HS                       //High speed Osc (> 4mhz)
#FUSES NOPUT                    //No Power Up Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NODEBUG                  //No Debug mode for ICD
#FUSES NOBROWNOUT               //No brownout reset
#FUSES NOLVP                    //No low voltage prgming, B3(PIC16) or B5(PIC18) used for I/O
#FUSES NOCPD                    //No EE protection
#FUSES WRT_50%                  //Lower half of Program Memory is Write Protected

#use delay(clock=20000000)
```

```
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
void main()
{
  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_psp(PSP_DISABLED);
  setup_spi(SPI_SS_DISABLED);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  while(1)
  {
  if(input(PIN_D0))
  {
  putc('A');
  }
  else if(input(PIN_D1))
  {
  putc('B');
  }
  else if(input(PIN_D4))
  {
  putc('E');
  }
  else if(input(PIN_D5))
```

```
  {
  putc('F');
  }
  else if(input(PIN_D6))
  {
  putc('G');
  }
  else if(input(PIN_D7))
  {
  putc('H');
  }
  else{
      putc(0x00);
      }
      delay_ms(10);
  }
}
```

- **Programme esclave A :**

```
#include <16F877A.h>
#device adc=8

#FUSES NOWDT                    // pas de Watch Dog Timer
#FUSES HS                       // Osc a grand vitesse (20mhz)
#FUSES NOPUT                    // No Power Up Timer
#FUSES NOPROTECT                // pas de protection de la lecture du programme
#FUSES NODEBUG                  // pas de Debug en mode ICD
```

```
#FUSES NOBROWNOUT                  // pas de brownout reset
#FUSES NOLVP                       // No low voltage prgming, B3(PIC16) used for I/O
#FUSES NOCPD                       // pas de protection de la EEPROM
#FUSES WRT_50%                     // 50% de la memoire du programme est proteger

#use delay(clock=20000000)
#use i2c(Slave,Fast,sda=PIN_C4,scl=PIN_C3,address=0xC0)          // uttilisation de I2C
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
// uttilisation de rs232
int A1_pwm;
int A2_pwm;
int incl=15;                                                    // valeur de l'inclinaison
int x_roulis;                                                   // valeur de l'angle roulis
int erreur_prec=0;
int erreur=0;
int somme_erreur=0;
int K = 25 ,Td = 0.5,Ti = 0.02;                                 // valeur des parametre PID

int PID(int mesure,int consigne)                                // la fonction PID
{
int commande;
erreur=consigne-mesure;
somme_erreur+=erreur;
commande=K*erreur+Td*(erreur-erreur_prec)+Ti*somme_erreur;
erreur_prec=erreur;
return commande;
}
```

```
#INT_SSP                                        // interuption sur I2C
void  int_i2c()
{
  x_roulis =  i2c_read();                       // lecture de roulis
}
#INT_TIMER2
void int_tmr2()
{
  if(input(PIN_D0))                             // test du botton haut
  {
    if(A1_pwm<200)
      {
       A1_pwm+=1;
      }
    if(A2_pwm<200)
      {
       A2_pwm+=1;
      }
  }
  else if(input(PIN_D1))                        // test du botton bas
  {
    if(A1_pwm>0)
      {
       A1_pwm-=1;
      }
    if(A2_pwm>0)
```

```
    {
     A2_pwm-=1;
    }
  }
else if(input(PIN_D4))                          // test du botton avant
{
  if(x_roulis>-incl)
    {
     A1_pwm-=1;
     A2_pwm+=1;
    }
}
else if(input(PIN_D5))                          // test du botton gauche
{
   if(x_roulis>-incl)
    {
     A1_pwm-=1;
     A2_pwm+=1;
    }
}
else if(input(PIN_D6))                          // test du botton arriere
{
if(x_roulis<incl)
    {
     A1_pwm+=1;
     A2_pwm-=1;
    }
```

```
 }
 else if(input(PIN_D7))                                  // test du botton droite
 {
  if(x_roulis<incl)
     {
      A1_pwm+=1;
      A2_pwm-=1;
     }
 }
 else{                                                   // l'auto_regulation par le PID
     A1_pwm += PID(x_roulis,0);
     A2_pwm -= PID(x_roulis,0);
    }
if(A1_pwm<0)
{
A1_pwm=0;
}
if(A2_pwm<0)
{
A2_pwm=0;
}
if(A1_pwm>255)
{
A1_pwm=255;
}
if(A2_pwm>255)
{
```

```
 A2_pwm=255;
 }
      set_pwm1_duty(A1_pwm);                    // reinitialisation du rapport cyclique
      set_pwm2_duty(A2_pwm);
}
void main()
{

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_comparator(NC_NC_NC_NC);
  enable_interrupts(GLOBAL);
  enable_interrupts(INT_TIMER2);
  enable_interrupts(INT_SSP);
  setup_vref(FALSE);
  setup_ccp1(CCP_PWM);
  setup_ccp2(CCP_PWM);
  setup_timer_2(T2_DIV_BY_1 ,255 ,1);
  while(1)
  {
  }
}
```

- **Programme de l'esclave B :**

```
#include <16F877A.h>
#device adc=8
```

```
#FUSES NOWDT                                    // pas de Watch Dog Timer
#FUSES HS                                       // Osc a grand vitesse (20mhz)
#FUSES NOPUT                                    // No Power Up Timer
#FUSES NOPROTECT                // pas de protection de la lecture du programme
#FUSES NODEBUG                                  // pas de Debug en mode ICD
#FUSES NOBROWNOU                                // pas de brownout reset
#FUSES NOLVP                    // No low voltage prgming, B3(PIC16) used for I/O
#FUSES NOCPD                                    // pas de protection de la EEPROM
#FUSES WRT_50%                  // 50% de la memoire du programme est proteger


#use delay(clock=20000000)
#use i2c(Slave,Fast,sda=PIN_C4,scl=PIN_C3,address=0xD0)          // uttiliser le I2C
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)
          // uttiliser le rs232


int B1_pwm;
int B2_pwm;


int incl=15;                                    // valeur de l'inclinaison
int y_tangage;                                  // valeur de l'angle roulis
int erreur_prec=0;
int erreur=0;
int somme_erreur=0;
int K = 25 ,Td = 0.5,Ti = 0.02;                 // valeur des parametre PID



int PID(int mesure,int consigne)                // la fonction PID
```

```
{
int commande;
erreur=consigne-mesure;
somme_erreur+=erreur;
commande=K*erreur+Td*(erreur-erreur_prec)+Ti*somme_erreur;
erreur_prec=erreur;
return commande;
}
#INT_SSP                                        // interuption sur I2C
void  int_i2c()
{
  y_tangage = i2c_read();                       // lecture de roulis
}
#INT_TIMER2
void int_tmr2()
{
  if(input(PIN_D0))                             // test du botton haut
  {
    if(B1_pwm<200)
      {
       B1_pwm+=1;
      }
    if(B2_pwm<200)
      {
       B2_pwm+=1;
      }
  }
```

```
else if(input(PIN_D1))                          // test du botton bas
{
  if(B1_pwm>0)
    {
     B1_pwm-=1;
    }
  if(B2_pwm>0)
    {
     B2_pwm-=1;
    }
 }
else if(input(PIN_D4))                          // test du botton avant
{
 if(y_tangage>-incl)
    {
     B1_pwm-=1;
     B2_pwm+=1;
    }
}
else if(input(PIN_D5))                          // test du botton gauche
{
  if(y_tangage>-incl)
    {
     B1_pwm-=1;
     B2_pwm+=1;
    }
}
```

```
 else if(input(PIN_D6))                                  // test du botton arriere
 {
 if(y_tangage<incl)
    {
     B1_pwm+=1;
     B2_pwm-=1;
    }
 }
 else if(input(PIN_D7))                                  // test du botton droite
 {
  if(y_tangage<incl)
    {
     B1_pwm+=1;
     B2_pwm-=1;
    }
 }
 else{
    B1_pwm += PID(y_tangage,0);                          // l'auto_regulation par le PID
    B2_pwm -= PID(y_tangage,0);
   }
 if(B1_pwm<0)
{
B1_pwm=0;
}
if(B2_pwm<0)
{
B2_pwm=0;
```

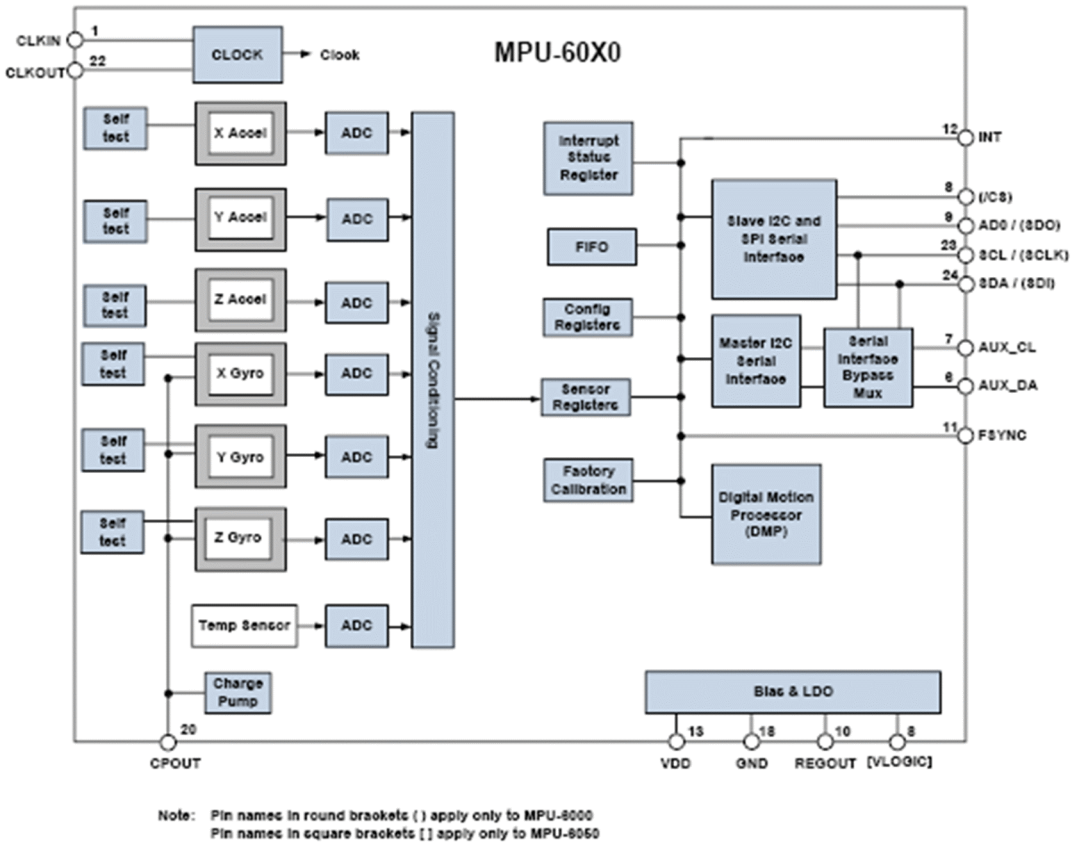
```
}
if(B1_pwm>255)
{
B1_pwm=255;
}
if(B2_pwm>255)
{
B2_pwm=255;
}
      set_pwm1_duty(B1_pwm);                    // reinitialisation du rapport cyclique
      set_pwm2_duty(B2_pwm);
}

void main()
{

  setup_adc_ports(NO_ANALOGS);
  setup_adc(ADC_OFF);
  setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
  setup_comparator(NC_NC_NC_NC);
  enable_interrupts(GLOBAL);
  enable_interrupts(INT_TIMER2);
  enable_interrupts(INT_SSP);
  setup_vref(FALSE);
  setup_ccp1(CCP_PWM);
  setup_ccp2(CCP_PWM);
  setup_timer_2(T2_DIV_BY_1 ,255 ,1);
```

```
  while(1)
  {
  }
}
```

ANNEXE B : Datasheet

1.Datasheet MPU 6050

Block Diagram



Note: Pin names in round brackets ( ) apply only to MPU-6000
Pin names in square brackets [ ] apply only to MPU-6050

Register Map

The register map for the MPU-60X0 is listed below.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0D	13	SELF_TEST_X	R/W	XA_TEST[4-2]			XG_TEST[4-0]				
0E	14	SELF_TEST_Y	R/W	YA_TEST[4-2]			YG_TEST[4-0]				
0F	15	SELF_TEST_Z	R/W	ZA_TEST[4-2]			ZG_TEST[4-0]				
10	16	SELF_TEST_A	R/W	RESERVED		XA_TEST[1-0]		YA_TEST[1-0]		ZA_TEST[1-0]	
19	25	SMPLRT_DIV	R/W	SMPLRT_DIV[7:0]							
1A	26	CONFIG	R/W	-	-	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	R/W	-	-	-	FS_SEL [1:0]		-	-	-
1C	28	ACCEL_CONFIG	R/W	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]				
1F	31	MOT_THR	R/W	MOT_THR[7:0]							
23	35	FIFO_EN	R/W	TEMP _FIFO_EN	XG _FIFO_EN	YG _FIFO_EN	ZG _FIFO_EN	ACCEL _FIFO_EN	SLV2 _FIFO_EN	SLV1 _FIFO_EN	SLV0 _FIFO_EN
24	36	I2C_MST_CTRL	R/W	MULT _MST_EN	WAIT _FOR_ES	SLV_3 _FIFO_EN	I2C_MST _P_NSR	I2C_MST_CLK[3:0]			
25	37	I2C_SLV0_ADDR	R/W	I2C_SLV0 _RW	I2C_SLV0_ADDR[6:0]						
26	38	I2C_SLV0_REG	R/W	I2C_SLV0_REG[7:0]							
27	39	I2C_SLV0_CTRL	R/W	I2C_SLV0 _EN	I2C_SLV0 _BYTE_SW	I2C_SLV0 _REG_DIS	I2C_SLV0 _GRP	I2C_SLV0_LEN[3:0]			
28	40	I2C_SLV1_ADDR	R/W	I2C_SLV1 _RW	I2C_SLV1_ADDR[6:0]						
29	41	I2C_SLV1_REG	R/W	I2C_SLV1_REG[7:0]							
2A	42	I2C_SLV1_CTRL	R/W	I2C_SLV1 _EN	I2C_SLV1 _BYTE_SW	I2C_SLV1 _REG_DIS	I2C_SLV1 _GRP	I2C_SLV1_LEN[3:0]			
2B	43	I2C_SLV2_ADDR	R/W	I2C_SLV2 _RW	I2C_SLV2_ADDR[6:0]						
2C	44	I2C_SLV2_REG	R/W	I2C_SLV2_REG[7:0]							
2D	45	I2C_SLV2_CTRL	R/W	I2C_SLV2 _EN	I2C_SLV2 _BYTE_SW	I2C_SLV2 _REG_DIS	I2C_SLV2 _GRP	I2C_SLV2_LEN[3:0]			
2E	46	I2C_SLV3_ADDR	R/W	I2C_SLV3 _RW	I2C_SLV3_ADDR[6:0]						
2F	47	I2C_SLV3_REG	R/W	I2C_SLV3_REG[7:0]							
30	48	I2C_SLV3_CTRL	R/W	I2C_SLV3 _EN	I2C_SLV3 _BYTE_SW	I2C_SLV3 _REG_DIS	I2C_SLV3 _GRP	I2C_SLV3_LEN[3:0]			
31	49	I2C_SLV4_ADDR	R/W	I2C_SLV4 _RW	I2C_SLV4_ADDR[6:0]						
32	50	I2C_SLV4_REG	R/W	I2C_SLV4_REG[7:0]							
33	51	I2C_SLV4_DO	R/W	I2C_SLV4_DO[7:0]							
34	52	I2C_SLV4_CTRL	R/W	I2C_SLV4 _EN	I2C_SLV4 _INT_EN	I2C_SLV4 _REG_DIS	I2C_MST_DLY[4:0]				
35	53	I2C_SLV4_DI	R	I2C_SLV4_DI[7:0]							
36	54	I2C_MST_STATUS	R	PASS_ THROUGH	I2C_SLV4 _DONE	I2C_LOST _ARB	I2C_SLV4 _NACK	I2C_SLV3 _NACK	I2C_SLV2 _NACK	I2C_SLV1 _NACK	I2C_SLV0 _NACK
37	55	INT_PIN_CFG	R/W	INT_LEVEL	INT_OPEN	LATCH _INT_EN	INT_RD _CLEAR	FSYNC_ INT_LEVEL	FSYNC _INT_EN	I2C _BYPASS _EN	-
38	56	INT_ENABLE	R/W	-	MOT_EN	-	FIFO _OFLOW _EN	I2C_MST _INT_EN	-	-	DATA _RDY_EN

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3A	58	INT_STATUS	R	-	MOT_INT	-	FIFO _OFLOW _INT	I2C_MST _INT	-	-	DATA _RDY_INT
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT[7:0]							
49	73	EXT_SENS_DATA_00	R	EXT_SENS_DATA_00[7:0]							
4A	74	EXT_SENS_DATA_01	R	EXT_SENS_DATA_01[7:0]							
4B	75	EXT_SENS_DATA_02	R	EXT_SENS_DATA_02[7:0]							
4C	76	EXT_SENS_DATA_03	R	EXT_SENS_DATA_03[7:0]							
4D	77	EXT_SENS_DATA_04	R	EXT_SENS_DATA_04[7:0]							
4E	78	EXT_SENS_DATA_05	R	EXT_SENS_DATA_05[7:0]							
4F	79	EXT_SENS_DATA_06	R	EXT_SENS_DATA_06[7:0]							
50	80	EXT_SENS_DATA_07	R	EXT_SENS_DATA_07[7:0]							
51	81	EXT_SENS_DATA_08	R	EXT_SENS_DATA_08[7:0]							
52	82	EXT_SENS_DATA_09	R	EXT_SENS_DATA_09[7:0]							
53	83	EXT_SENS_DATA_10	R	EXT_SENS_DATA_10[7:0]							
54	84	EXT_SENS_DATA_11	R	EXT_SENS_DATA_11[7:0]							
55	85	EXT_SENS_DATA_12	R	EXT_SENS_DATA_12[7:0]							
56	86	EXT_SENS_DATA_13	R	EXT_SENS_DATA_13[7:0]							
57	87	EXT_SENS_DATA_14	R	EXT_SENS_DATA_14[7:0]							
58	88	EXT_SENS_DATA_15	R	EXT_SENS_DATA_15[7:0]							
59	89	EXT_SENS_DATA_16	R	EXT_SENS_DATA_16[7:0]							
5A	90	EXT_SENS_DATA_17	R	EXT_SENS_DATA_17[7:0]							
5B	91	EXT_SENS_DATA_18	R	EXT_SENS_DATA_18[7:0]							
5C	92	EXT_SENS_DATA_19	R	EXT_SENS_DATA_19[7:0]							
5D	93	EXT_SENS_DATA_20	R	EXT_SENS_DATA_20[7:0]							
5E	94	EXT_SENS_DATA_21	R	EXT_SENS_DATA_21[7:0]							
5F	95	EXT_SENS_DATA_22	R	EXT_SENS_DATA_22[7:0]							
60	96	EXT_SENS_DATA_23	R	EXT_SENS_DATA_23[7:0]							
63	99	I2C_SLV0_DO	R/W	I2C_SLV0_DO[7:0]							
64	100	I2C_SLV1_DO	R/W	I2C_SLV1_DO[7:0]							

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
65	101	I2C_SLV2_DO	R/W	I2C_SLV2_DO[7:0]							
66	102	I2C_SLV3_DO	R/W	I2C_SLV3_DO[7:0]							
67	103	I2C_MST_DELAY_CTRL	R/W	DELAY_ES_SHADOW	-	-	I2C_SLV4_DLY_EN	I2C_SLV3_DLY_EN	I2C_SLV2_DLY_EN	I2C_SLV1_DLY_EN	I2C_SLV0_DLY_EN
68	104	SIGNAL_PATH_RESET	R/W	-	-	-	-	-	GYRO_RESET	ACCEL_RESET	TEMP_RESET
69	105	MOT_DETECT_CTRL	R/W	-	-	ACCEL_ON_DELAY[1:0]		-		-	
6A	106	USER_CTRL	R/W	-	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	-	FIFO_RESET	I2C_MST_RESET	SIG_COND_RESET
6B	107	PWR_MGMT_1	R/W	DEVICE_RESET	SLEEP	CYCLE	-	TEMP_DIS	CLKSEL[2:0]		
6C	108	PWR_MGMT_2	R/W	LP_WAKE_CTRL[1:0]		STBY_XA	STBY_YA	STBY_ZA	STBY_XG	STBY_YG	STBY_ZG
72	114	FIFO_COUNTH	R/W	FIFO_COUNT[15:8]							
73	115	FIFO_COUNTL	R/W	FIFO_COUNT[7:0]							
74	116	FIFO_R_W	R/W	FIFO_DATA[7:0]							
75	117	WHO_AM_I	R	-	WHO_AM_I[6:1]						-

Datasheet : Bluetooth

PINs description

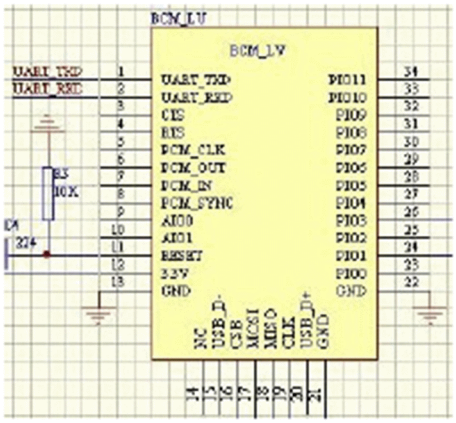


Figure 3 PIN configuration

The PINs at this block diagram is same as the physical one.

PIN Name	PIN #	Pad type	Description	Note
GND	13 21 22	VSS	Ground pot	
1V8	14	VDD	Integrated 1.8V (+) supply with On-chip linear regulator output within 1.7-1.9V	
VCC	12	3.3V		
AIO0	9	Bi-Directional	Programmable input/output line	
AIO1	10	Bi-Directional	Programmable input/output line	
PIO0	23	Bi-Directional RX EN	Programmable input/output line, control output for LNA(if fitted)	
PIO1	24	Bi-Directional TX EN	Programmable input/output line, control output for PA(if fitted)	
PIO2	25	Bi-Directional	Programmable	

			input/output line	
PIO3	26	Bi-Directional	Programmable input/output line	
PIO4	27	Bi-Directional	Programmable input/output line	
PIO5	28	Bi-Directional	Programmable input/output line	
PIO6	29	Bi-Directional	Programmable input/output line	CLK_REQ
PIO7	30	Bi-Directional	Programmable input/output line	CLK_OUT
PIO8	31	Bi-Directional	Programmable input/output line	
PIO9	32	Bi-Directional	Programmable input/output line	
PIO10	33	Bi-Directional	Programmable input/output line	
PIO11	34	Bi-Directional	Programmable input/output line	
RESETB	11	CMOS Input with weak internal pull-down		
UART_RTS	4	CMOS output, tri-stable with weak internal pull-up	UART request to send, active low	
UART_CTS	3	CMOS input with weak internal pull-down	UART clear to send, active low	
UART_RX	2	CMOS input with weak internal pull-down	UART Data input	
UART_TX	1	CMOS output, Tri-stable with weak internal pull-up	UART Data output	
SPI_MOSI	17	CMOS input with weak internal pull-down	Serial peripheral interface data input	
SPI_CSB	16	CMOS input with weak internal pull-up	Chip select for serial peripheral interface, active low	
SPI_CLK	19	CMOS input with weak	Serial peripheral interface	

		internal pull-down	clock	
SPI_MISO	18	CMOS input with weak internal pull-down	Serial peripheral interface data Output	
USB_-	15	Bi-Directional		
USB_+	20	Bi-Directional		
1.8V	14		1.8V external power supply input	Default : 1.8V internal power supply.
PCM_CLK	5	Bi-Directional		
PCM_OUT	6	CMOS output		
PCM_IN	7	CMOS Input		
PCM_SYNC	8	Bi-Directional		

Bibliographie

[1] **A. Frenot - A. Gossmann – R. Guillerm**

Stabilisation d'un Quadri rotor.

ENSICA Toulouse France

[2] Rémi CAZZARO

Modélisation et stabilisation d'un Micro Hélicoptère à Quatre rotors

CNAM de TOULOUSE

[3] E:\etudes\memoire drone\Quadricopter 2- La documentation blog de julien dumortier.htm

[4] E:\etudes\memoire drone\programmation des microcontrôleur en C compilateur CCS PIC_files\ca-pub-0827645159344239.js

[5] La programmation des PIC par BIGONOFF-seconde partie.

La gamme MID-Range par l'étude des A6F87X.

[6] La programmation des PIC

Par: Christian TAVERNIER

[7] Datasheet MPU-600 and MPU-6050 Product specifiction Revision 3.4 (19/08/2013).