Faculté des Sciences et Techniques
Marrakech

Département de Physique Appliquée

Filière d'Ingénieurs d'Etat en Systèmes Electriques et Télécommunications (ISET)

Développement d'un programme « BOOTLOADER » Avec le PIC *16F887*

Réalisé Par :

- NIDBELKACEM Mouhcine
- RAIOUH Hassana
- RAIOUH Ihsane
- KAMBOU Célestin

Encadré Par :

- Mr H.*AYAD*

Année Universitaire 2013/2014

Remerciements

*N*os remerciements iront tout d'abord à Monsieur *HASSAN AYAD* enseignant à la FSTG de Marrakech, qui nous a encadré et guidé tout au long de ce projet.

*E*nsuite, nous tenons à adresser toute notre gratitude et notre sympathie à tous les enseignants de la Faculté des Sciences et Techniques, surtout ceux de département de Physique Appliquée, qui nous ont consacré de leur temps et de leur énergie afin de faciliter le déroulement de nos études.

Sommaire

Introduction

La programmation d'un microcontrôleur nécessite toujours un programmateur pour transférer le programme du PC au microcontrôleur.

Chaque programme pourra contenir un certain nombre d'erreurs qui obligera le programmeur de reprogrammer son microcontrôleur, ceci implique le déplacement fréquent du microcontrôleur de sa carte d'utilisation finale vers le programmateur et vis versa. De plus, pour certains équipements dotés de microcontrôleurs, le constructeur propose des mises à jour du logiciel (FIRMWARE) pour corriger certains bugs ou apporter des améliorations ou de nouvelles fonctionnalités. L'utilisateur final de tels équipements ne dispose ni de programmateur ni de compétences pour reprogrammer le microcontrôleur ; ce dernier peut aussi être non démontable de la carte industrielle. Pour dépasser tous ces problèmes, on utilise actuellement la technique du << BOOTLOADER>>.

Le sujet de notre projet consiste à l'étude, la conception et la réalisation d'un programme BOOTLOADER avec le PIC 16F887 qui permettra la reprogrammation du PIC plusieurs fois, sans avoir besoin du programmateur. Les programmes pour Pic sont développés sur un PC dans un environnement de développement intégré IDE, comme celui fourni par la société MICROCHIP (MiKroC). Le transfert du code exécutable vers le PIC à partir du PC nécessite un interfaçage matériel PC-PIC.

Cahier Des Charges

→Objectif :

Développement et essai d'un programme « BOOTLOADER » pour PIC de MICROCHIP de type Mid-range disposant de mémoire flash programmable par le pic lui même.

→Travail demandé :

Après avoir bien étudié et compris le fonctionnement du pic 16F887, son langage d'assemblage, les techniques de programmation de la mémoire flash, d'échange de données à travers l'interface asynchrone USART (RS232),
Développer :

- un programme loader à programmer sur le pic dont le rôle est de recevoir de l'extérieur le programme utilisateur via l'interface USART et de l'écrire dans la mémoire programme flash. Ce programme prend toujours la main après la RAZ du pic, puis tente de détecter la présence de la liaison au PC pour recharger un autre programme, si cette liaison n'est pas détectée il rend la main au programme utilisateur déjà chargé précédemment. Ce programme sera écrit en assembleur pour PIC de MICROCHIP.

- Un programme écrit en langage C, au niveau du PC, dont le rôle et de d'établir la connexion avec le pic, de demander à l'utilisateur de fournir le nom du fichier du programme .HEX à charger dans le pic, puis assure le chargement vers le pic, tout en garantissant un contrôle de flux et d'erreur lors de l'échange, ainsi que la protection contre l'écrasement du programme loader.

Chapitre 1 : Point De Vue Matériel

Introduction :

Dans ce chapitre nous expliquerons en quoi consiste la technique du "BOOTLOADER", en citant son principe de fonctionnement ainsi que ses avantages.

1 – BOOTLOADER :

1.1 – Rôle de BOOTLOADER :

Pour modifier un programme dans un microcontrôleur placé dans une carte industrielle, on devra couper l'alimentation du montage, enlever le microcontrôleur, le placer dans le programmateur, le reprogrammer, le remettre dans son montage et peut être, tout recommencer plusieurs fois.

Tout ceci est un travail non valorisant, et en plus présente un grand risque. En effet un moment ou à un autre, le programmeur pourra oublier de couper l'alimentation par exemple, ou alors mettre le microcontrôleur à l'envers, etc .... Ce qui entraine des risques d'endommagement des équipements.

Aussi, parfois, le programmeur ne pourrait pas utiliser le microcontrôleur une fois il a placé le montage sur sa carte d'exploitation, il ne pourrait pas démonter le circuit de son support à cause de la soudure du circuit sur sa platine ou son éventuelle inaccessibilité (circuit difficilement démontable). Cela peut l'empêcher de reprogrammer son microcontrôleur pour une éventuelle mise à jour.

Il nous faut donc utiliser une solution pratique et flexible pour charger une nouvelle version du programme dans le microcontrôleur sans le démonter de sa carte d'exploitation. La technique du BOOTLOADER est faite pour résoudre ce problème.

1.2 – Définition :

Le BOOTLOADER que nous souhaitons concevoir pour les microcontrôleurs de type PIC de MICROCHIP dotés d'une mémoire flash auto programmable, est un programme résidant dans le PIC dans les adresses hautes de la mémoire Flash. Il permet de mettre à jour le programme contenu dans le PIC en écrivant directement dans la mémoire flash, le nouveau programme reçu de l'extérieur (à partir d'un PC par exemple) via l'un des ports, dans ce projet on a choisi l'USART du côté PIC et donc le port RS232 du côté PC, vu que ce type d'interface est l'une des plus répondues pour la gamme de PIC auquel on s'intéresse dans ce projet.

Le BOOTLOADER doit être protégé contre un écrasement. Il ne permet pas modifier les bits de configurations ni l'EEPROM interne.

L'idée de base est la suivante :

- Une fois le nouveau programme utilisateur écrit, assemblé et testé dans un simulateur on obtient un fichier contenant du code exécutable avec l'extension « .HEX »

- Ce fichier « .HEX » est écrit dans un format spécial qu'il faut savoir décoder pour extraire, les instructions à écrire dans le pic et les adresses dans lesquelles il faut les écrire, et les envoyer vers le PIC.
- Le PIC doit être dans un mode de fonctionnement tel qu'il exécute le programme BOOTLOADER et non pas l'ancien programme utilisateur. Il doit donc recevoir les instructions et leurs adresses et les écrire dans la mémoire flash tout en écrasant ou en complétant l'ancien programme utilisateur par le nouveau programme utilisateur
- Une fois le transfert terminé, et après une remise à zéro, le pic redémarre en mode d'exécution du nouveau programme utilisateur qui a été chargé.

Il va sans le dire que cet échange entre le PIC et le PC, doit impliquer un contrôle d'erreurs et un contrôle de flux.

2 – conception de l'interface :

2.1 – choix des matériels :

Le **BOOTLOADER** est un microprogramme qui est programmé une seule fois dans le pic 16F887 à l'aide d'un programmateur. Il utilise une interface **RS232** pour le transfert du programme utilisateur vers la mémoire flash, donc il faut en plus un **Max232** dans notre système. Ça augmente le nombre de composants mais ça facilite énormément la programmation (Idéal pour les cartes de développement). Il suffit de brancher le câble série sur la carte, lancer le transfert. Lorsque le transfert sera terminé, rebooter le PIC et le programme qui vient d'être transféré s'exécute.

La configuration minimale nécessaire pour établir la connexion de notre kit au PC est comme suit (**Fig. 1.1)** :

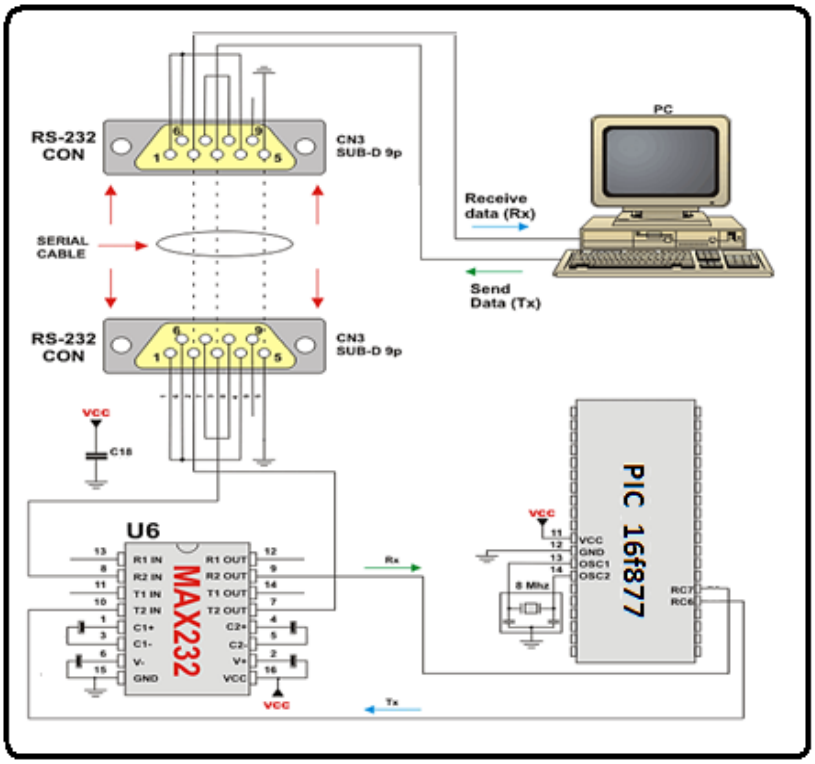

Fig. 1.1 : circuit Bootloader

→Principe de fonctionnement de chaque composant :

a) Microcontrôleur type PIC 16F887 :

Le 16F877 est un PIC de la série « Mid-range » qui est en particulier facile en programmation (jeu d'instruction réduit) permettant de faire une communication série grâce au module intégré USART. Les PIC de la série inférieure sont un peu justes en performance et en capacité mémoire pour accueillir un programme issu d'un compilateur C. Les gammes supérieures (16 ou 32 bits) supportent sans problème la programmation du BOOTLOADER. Le 16F887 (F comme « Flash ») convient parfaitement : mémoire programme de taille suffisante (8K), nombreux périphériques intégrés, fréquence de fonctionnement jusqu'à 20 MHz.

→Mémoires:

Le pic 16F877 contient trois types de mémoire qui sont : mémoire programme, mémoire ROM, mémoire EEPROM. Dans notre circuit nous allons étudier la mémoire Flash qui doit être programmée par le pic lui même à travers le programme BOOTLOADER.

→PORTC (RC6 et RC7):

Les deux broches RC6 et RC7 du port C peuvent être utilisées comme des broches de transfert pour la liaison série synchrone (CLK, DT) ou asynchrone (TX, RX). Dans notre projet elles seront utilisées avec l'interface intégrée USART pour communiquer avec le PC à partir duquel l'utilisateur envoie son programme.

→USART :

Le module USART travaille en deux modes synchrone et asynchrone mais nous dans notre projet nous travaillerons par module USART en mode asynchrone.

USART asynchrone est une liaison série asynchrone, comme son nom l'indique en émettant les bits les uns à la suite des autres, mais sans fournir le signal d'horloge qui a permis de les générer.

→Brochage du PIC16F887 (Fig. 1.3):

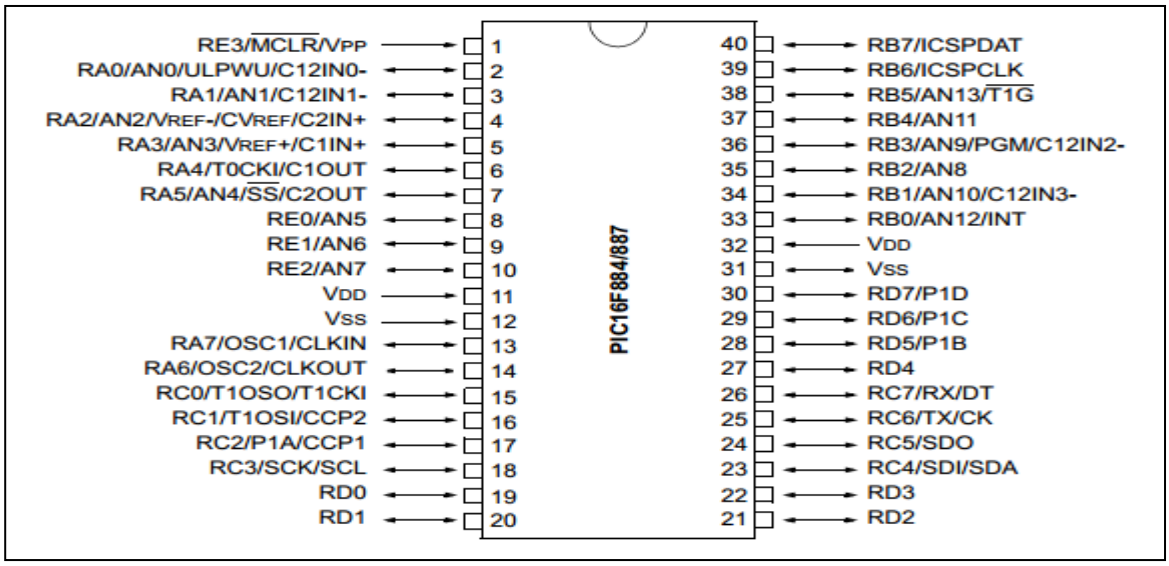

Fig. 1.3 : brochage du PIC 16F877

b) MAX 232 (Fig. 1.4):

Presque tous les appareils numériques que nous utilisons nécessitent soit des niveaux TTL ou logique CMOS. Par conséquent, la première étape pour la connexion d'un périphérique sur le port RS-232 est de transformer les niveaux de tensions utilisés par RS-232 en 0 et 5 Volts. Cela se fait par le circuit d'adaptation MAX 232.

Le MAX232 inclut deux récepteurs transformant les signaux RS232 en signaux TTL, et deux émetteurs transformant les signaux TTL en signaux RS232. Pour fournir les niveaux utilisés par la norme RS232 à partir du +5V, le circuit utilise un convertisseur élévateur de tension basé sur le principe de pompe de charge de condensateurs.

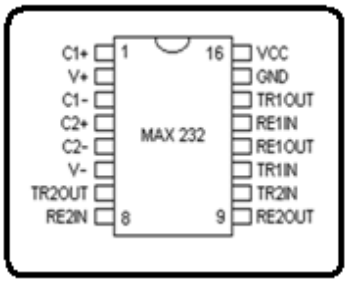

Fig. 1.4 : structure du MAX 232

Le schéma ci-dessous (**Fig. 1.5**) présente la liaison entre le port RS-232 et le MAX232 avec les deux pins TX, RX, sachant que le premier est utilisé pour la transmission et le deuxième pour la réception.

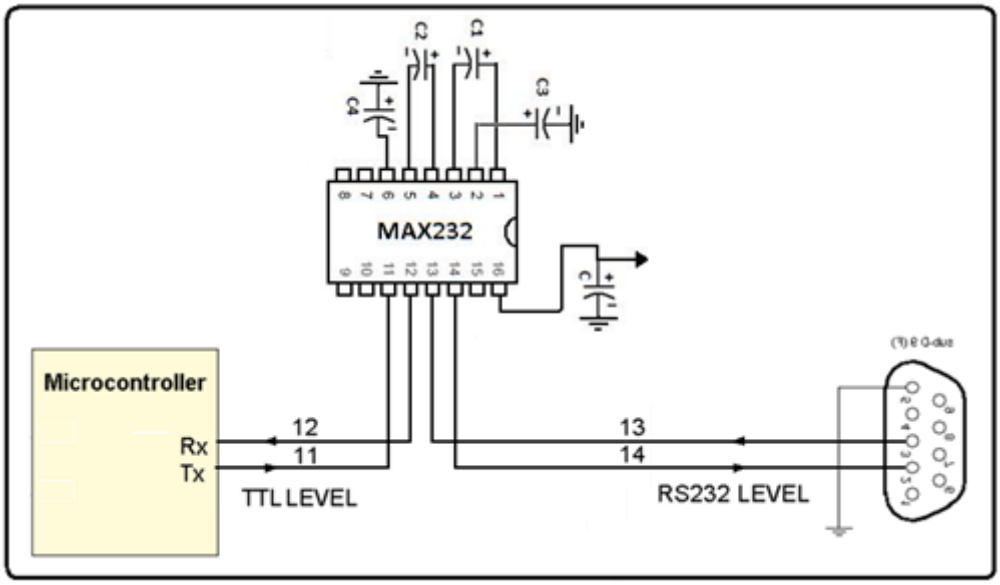

Fig. 1.5 : liaison port RS232-MAX 232

c) RS 232 :

Nous utilisons un câble nul-modem pour lier le PC avec le PIC

- Nous notons que le brochage du port RS232 (**Fig. 1.6)** utilisé dans notre circuit est un brochage nul modem croisé c'est-à-dire le pin 2 du port du PC avec le pin 3 du port de la carte, le pin 3 du port pc avec le pin 2 du port de la carte, comme s'est présenté dans le schéma de liaison.

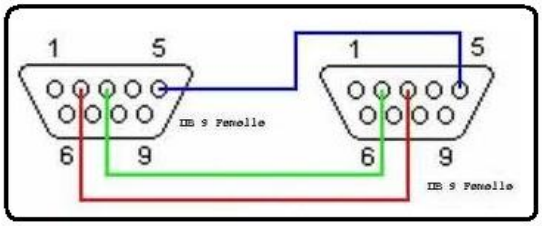

Fig. 1.6 : liaison port RS232-MAX 232

2.2 – le circuit du BOOTLOADER:

Une fois nous avons étudié les différents composants utilisés dans notre réalisation, nous présentons notre circuit de BOOTLOADER comme suit (**Fig. 1.7**):

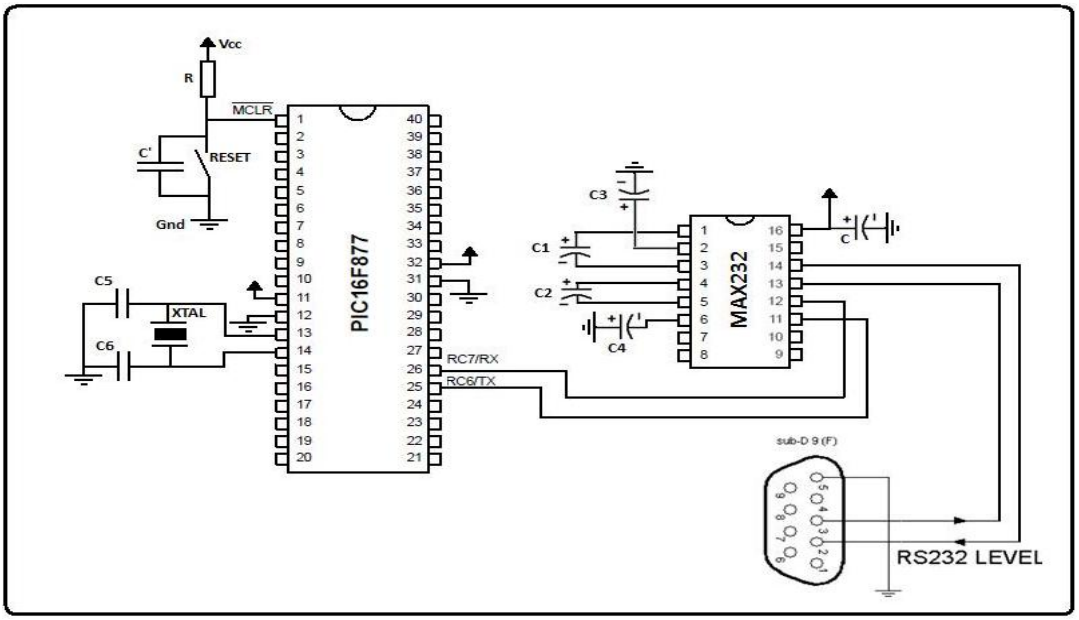

Fig. 1.7 : Circuit globale du Bootloader

Chapitre 2 : Point De Vue Logiciel

Introduction :

Maintenant que nous avons traité l'ensemble des éléments utilisés dans notre circuit du BOOTLOADER, nous présenterons dans le présent chapitre les organigrammes de programmation côtés PIC et PC avec la simulation pour la vérification du bon fonctionnement.

1- communication PIC - PC:

La communication PC – PIC nécessite deux phases :

- Connexion
- Transfert

Le contrôle de flux est réalisé par l'acquittement groupé.
La synchronisation est assurée par la communication unidirectionnelle.

Le PIC entre dans une boucle d'essai de connexion avec le PC d'une durée de 3 secondes.

Si le PIC reçoit la confirmation du coté PC il envoie un acquittement au PC pour que ce dernier commence le transfert des données.

Après chaque donnée (4 octets ADRH-ADRL-DATAH-DATAL) le PIC envoie un acquittement pour chaque octet envoyé qui est l'octet lui même en plus d'un octet RES qui prend la valeur 1 (indique la bonne écriture de la donnée en mémoire) ou 0 pour indiquer l'échec d'écriture en mémoire flash.

Le PC analyse les octets acquittement reçus et les compare avec ceux envoyés ; en cas de non correspondance il renvoie la même donnée (4 octets), ce ci est aussi le cas si le résultat d'écriture en mémoire a échoué. Sinon il passe à l'envoi de la donnée suivante (4 octets). Toutefois, ce renvoi est limité par le PC à 5 fois maximum ; au delà de 5 tentatives le PC arrête le transfert vers le PIC et annonce l'échec de l'opération. Une fois toutes les données transférées le programme de transfert du côté PC s'arrête, il faut rebooter le PIC. Après ce reboot le pic tente, pendant 3 sec, à nouveau de se connecter au PC, mais sans succès puisque le programme PC est arrêté et le pic passe alors en mode d'exécution du programme utilisateur qui vient d'être transféré et écrit en mémoire flash.

Si le PIC ne reçoit pas la confirmation de synchronisation au bout de 3 secondes il passe directement à l'exécution du programme utilisateur qui avait été transféré précédemment en mémoire flash.

On présente ci-dessus (Fig. 2.1) les différents scénarios de communication entre le PIC et le PC.

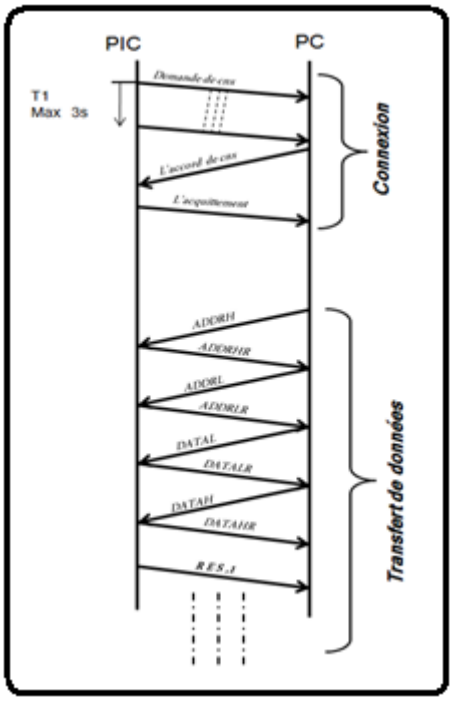

Fig. 2.1.a Transfert normal

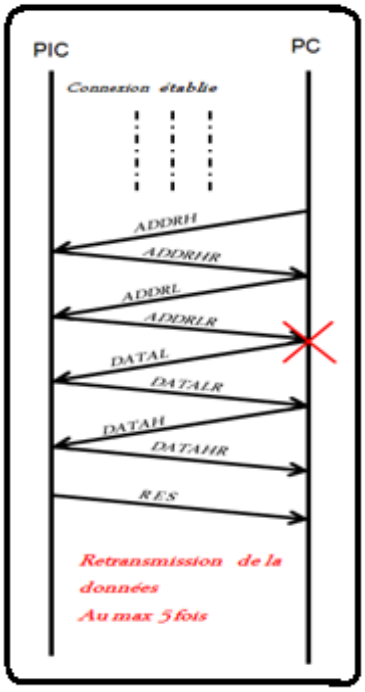

Fig. 2.1.b Erreur de transmission

- Dans le cas du transfert normal l'acquittement groupé et positif RES prend la valeur 1.
- Dans le cas d'échange erronée l'acquittement groupé de la donnée est positif RES prend la valeur 1 mais le PIC demande le renvoi de la donnée car il y a une erreur de transmission.

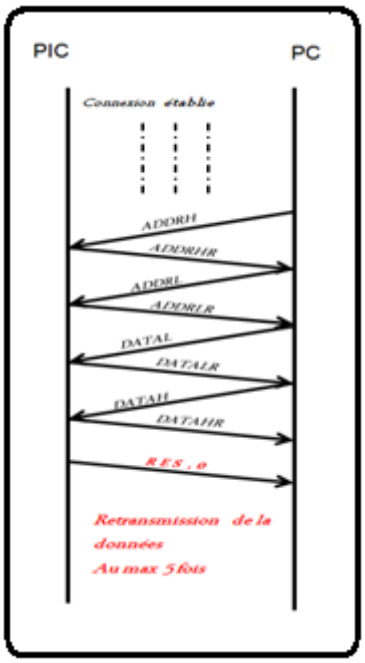

Fig. 2.1.c Erreur d'enregistrement

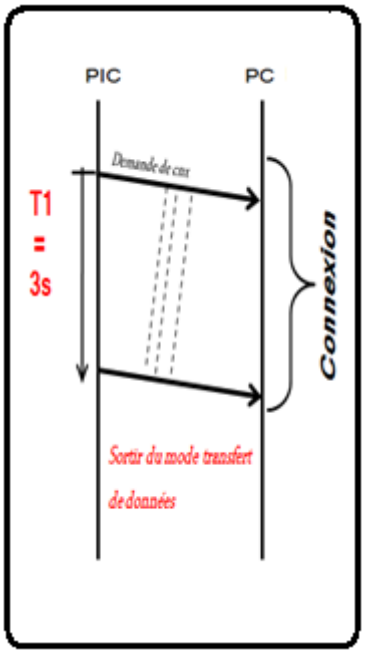

Fig. 2.1.d Echec de connexion

- Dans le cas ou RES prend la valeur 0, cela indique que le PIC a fait une erreur lors de l'enregistrement de la donnée dans sa mémoire Flash. Pour cette raison il demande la retransmission de la donnée même si l'échange de données a été fait sans erreur.

- Dans le cas Echec de connexion le PC ne répond pas à al demande de connexion du PIC, le PIC sort du programme du transfert et passe en mode d'exécution du programme utilisateur déjà téléchargé précédemment dans le pic.

2-Développement du programme BOOTLOADER :

Avant de donner les organigrammes qui illustrent le fonctionnement de notre BOOTLOADER, il faut préciser la nature des informations échangées entre PC et PIC et aussi l'organisation adoptée pour la mémoire flash du pic.

Le BOOTLOADER n'est qu'un programme installé par le programmateur dans la mémoire flash, qui va jouer par la suite le rôle de programme qui installe les programmes utilisateurs sans avoir besoin du programmateur. En aucun moment le programme utilisateur ne doit s'installer dans la zone réservée au BOOTLOADER pour ne pas l'écrasé.

Une organisation de la mémoire est alors nécessaire et des protections doivent être intégrées dans les logiciels pour éviter d'écrire par dessus le BOOTLOADER.

La taille de la mémoire Flash du pic 16F887 est de 8ko=2^{13} l'adressage se fait sur 13 bits (ADRRH=5 bits, ADRRL=8 bits), l'adresse de l'instruction donc codées sur deux octets ADRRH et ADRRL et l'instruction (la donnée) de largeur 14 bits sera aussi codée sur 2 octets (DATAH et DATAL).

L'organisation de la mémoire flash sera comme suit (**Fig. 2.2**):

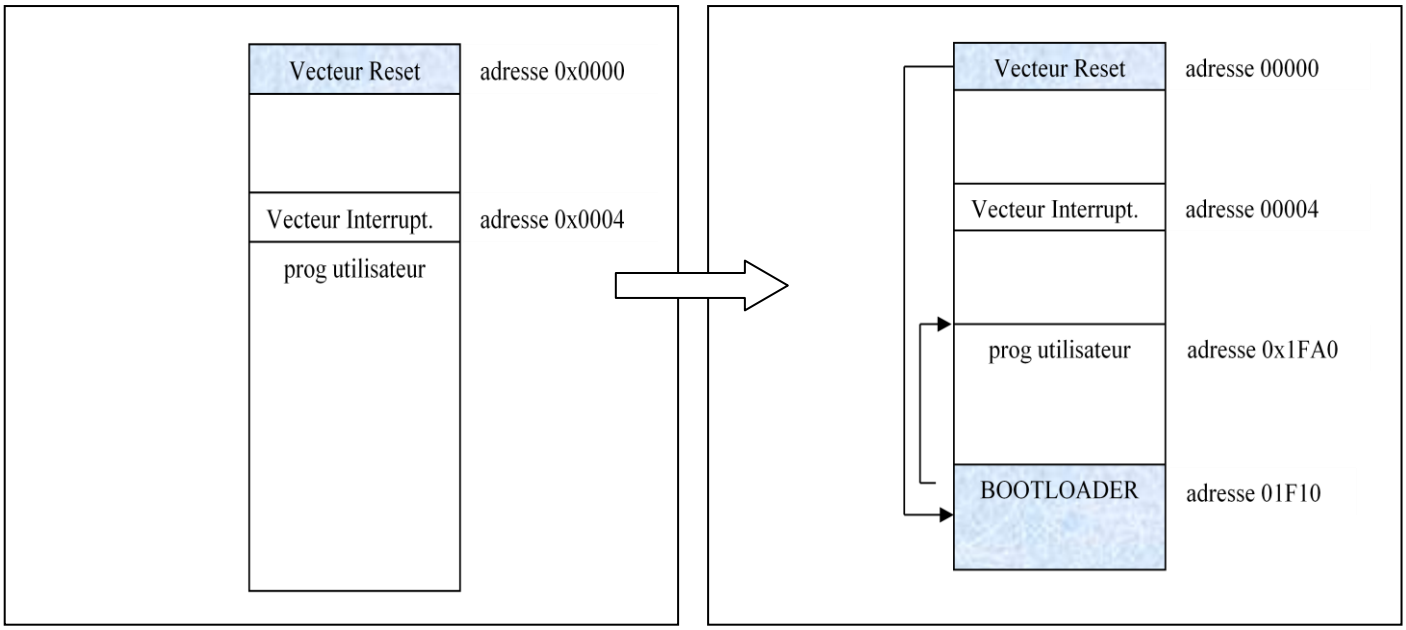

Fig. 2.2 : Emplacement mémoire du BOOTLOADER

2-1 Registres et Organigrammes :

SPBRG

(Adresse 99h)
Registre du Générateur de Baud de l'USART

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
Registre du Générateur de Baud (utilisé par l'USART)							
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

RCREG

(Adresse 1Ah)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
Registre de réception de l'USART							
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

TXSTA

(Adresse : 98h)
le attachments
Registre de statut et de contrôle d'émission (USART)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	**TX9**	**TXEN**	**SYNC**	**-**	**BRGH**	**TRMT**	**TX9D**
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 1	**TRMT** : bit de statut du registre à décalage d'émission (TSR = Transmit Shift Register) 1 = TSR vide 0 = TSR plein

RCSTA

(Adresse : 18h)
Registre de statut et de contrôle de réception (USART)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	**RX9**	**SREN**	**CREN**	**ADDEN**	**FERR**	**OERR**	**RX9D**
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 4	**CREN** : bit de validation de réception continue Mode asynchrone : 1 = Active la réception continue 0 = Désactive la réception continue Mode synchrone : 1 = Active la réception continue jusqu'à ce que le bit CREN soit effacé (CREN prime sur SREN 0 = Désactive la réception continue

PIR1

(adresse : 0Ch)
Registre de signalisation d'interruptions de périphériques

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF	**ADIF**	**RCIF**	**TXIF**	**SSPIF**	**CCP1IF**	**TMR2IF**	**TMR1IF**
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

bit 5	**RCIF** : USART Receive Interrupt Flag bit = bit « drapeau » de l'interruption de réception de l'USART 1 = Le buffer de réception de l'USART est plein 0 = Le buffer de réception de l'USART est vide

→Emission / Réception (liaison bidirectionnelle) :

Nous utiliserons les paramètres suivants :

- **8 bits de données**
- **pas de bit de parité**
- **1 bit d'arrêt (Stop)**
- **pas de contrôle de flux** (liaison 3 fils : RX, TX, GND)

avec **utilisation des interruptions de l'USART** du µC.

Le fil RX de la liaison RS232C est relié à la broche RB1/RX/DT du PIC 16F628A, à travers une interface (MAX232 ou équivalent).

Le fil TX de la liaison RS232C est relié à la broche RB2/TX/CK du PIC 16F628A, à travers une interface (MAX232 ou équivalent).

- Configuration du registre TRISB
  - broche RB1/RX/DT à configurer en entrée
  - broche RB2/TX/CK à configurer en entrée (bien que ce soit une sortie)

- Initialisation du registre SPBRG et du bit BRGH

  La vitesse de transmission est 9600 bauds / seconde, avec un quartz de 4 MHz :

  - SPBRG = D'25' (valeur du registre SPBRG)
  - BRGH = 1 (bit 2 du registre TXSTA)

- SYNC = 0 (bit 4 du registre TXSTA) : mode asynchrone
- SPEN = 1 (bit 7 du registre RCSTA) : utilisation du port série

- Autorisation des interruptions
    - GIE = 1 (bit 7 du registre INTCON) : autorisation globale des interruptions
    - PEIE = 1 (bit 6 du registre INTCON) : autorisation des interruptions des périphériques
    - TXIE = 1 (bit 4 du registre PIE1) : autorisation de l'interruption d'émission de l'USART
    - RCIE = 1 (bit 5 du registre PIE1) : autorisation de l'interruption de réception de l'USART

- Autorisation de la réception et de la transmission du port série
    - CREN = 1 (bit 4 du registre RCSTA) : autorise la réception
    - TXEN = 1 (bit 5 du registre TXSTA) : autorise la transmission

- Dans la routine d'interruption :
    - Sauvegarde du contexte
    - Ecriture dans le registre TXREG des 8 bits de données à transmettre
    - Lecture des 8 bits de données reçus (registre RCREG)
        - Si OERR = 1 (erreur d'overrun) : CREN à mettre 0 puis à 1
    - Restauration du contexte

→ Vitesse de transmission :

La liaison RS232C est caractérisée par sa vitesse de transmission (en bauds par seconde) :

300, 1200, 2400, 4800, 9600, 19 200, 38 400 etc ...

Il est bien clair que la vitesse de réception est identique à la vitesse de transmission.

Le choix se fait avec :

- SPBRG (registre 8 bits : valeur 0 à 255)
- BRGH (bit 2 du registre TXSTA)

→Formules :

BRGH = 1 : SPBRG = (Fosc/(16 x Vitesse de transmission)) - 1

BRGH = 0 : SPBRG = (Fosc/(64 x Vitesse de transmission)) - 1

ou

BRGH = 1 : Vitesse de transmission = Fosc/(16(SPBRG + 1))

BRGH = 0 : Vitesse de transmission = Fosc/(64(SPBRG + 1))

→Exemple

Le PIC 16F628A utilise un quartz externe de 4 MHz.

On désire une vitesse de transmission de 9600 bauds par seconde.

Si BRGH = 0 :

SPBRG = (4 000 0000 /(64 x 9600)) - 1 = 5,51

=> SPBRG = 6 (entier le plus proche)

=> Vitesse de transmission = 4 000 000 /(64(6 + 1)) = 8 929 bauds par seconde

=> 7 % d'erreur ce qui est inacceptable.

Si BRGH = 1 :

SPBRG = (4 000 0000 /(16 x 9600)) - 1 = 25,04

=> SPBRG = 25

=> Vitesse de transmission = 4 000 000 /(16(25 + 1)) = 9615 bauds par seconde

=> 0,16 % d'erreur ce qui est satisfaisant.

En définitive :

- SPBRG = D'25'
- BRGH = 1

On présente ci-dessus l'organigramme principal (**Fig. 2.3**) de notre BOOTLOADER qui contient des sous programmes:

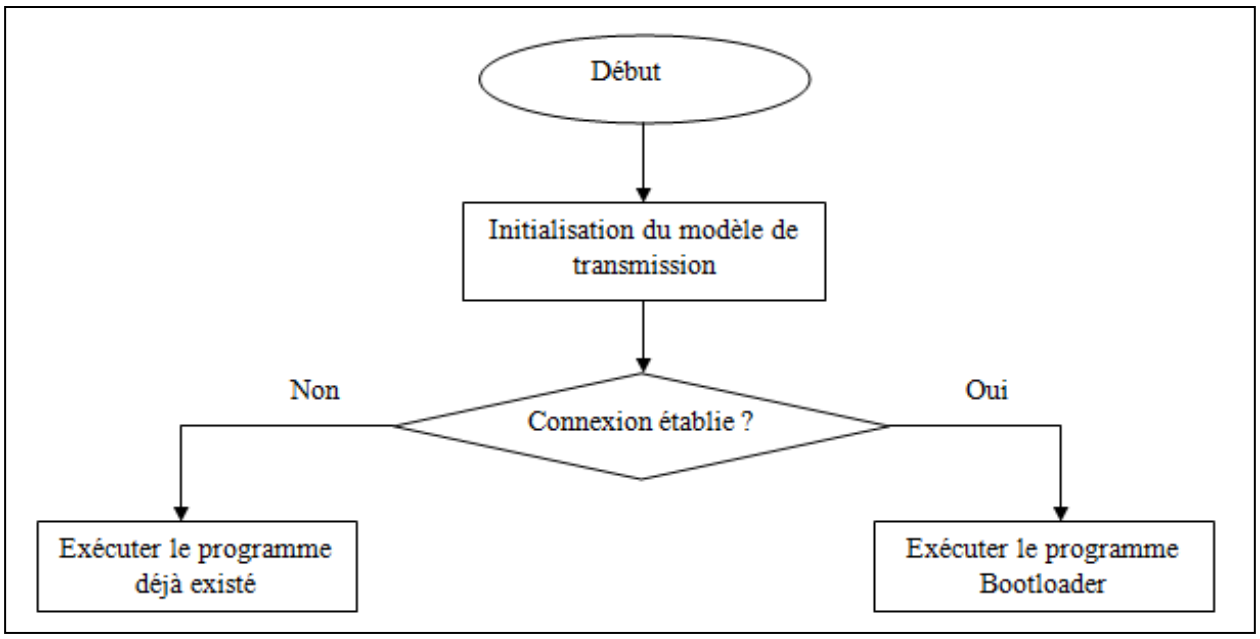

Fig. 2.3 : organigramme principale du PIC

→**Configuration :** nous commençons notre programme par la configuration des éléments suivants :

- Sélection du type d'oscillateur
- Watchdog timer
- Délai de démarrage à la mise en tension
- Autorisation d'écriture en mémoire Flash
- Débogage sur circuit
- CP1/CP2 détermination de la zone protégée contre la lecture
- Protection en lecture de la mémoire EEPROM
- Utilisation du pin RB3/PGM comme broche de programmation
- Le reset du PIC en cas de la chute de tension

→**Démarrage sur RESET :** pour chaque appui sur le bouton reset le pic commence l'exécution à partir de l'adresse 0x00000.

→**Le programme BOOTLOADER :** nous avons placé notre programme BOOTLAODER à partir de l'adresse 0x1F00 vers la dernière partie de la mémoire Flash .pour une raison de protection du programme BOOTLOADER, parce que l'utilisateur essaye en générale d'écrire son programme principale dans les premiers adresses après 0X000.comme ca l'utilisateur écrit librement son programme <<main>> sans affectation du BOOTLOADER. Nous avons essayé de minimiser le maximum possible la capacité de notre programme BOOLOADER pour ne pas occuper une grande taille dans la mémoire.

→**Initialisation du module de transmission :**

- Mettre USART en service
- Réception, émission sur 8 bits
- Réception continu
- Mode USART asynchrone en haute vitesse
- Déterminer la valeur de Baud rate 9600 BAUDS

- Pas de bits de parité
- Bit de START et de stop

→**Essai de connexion au PC :** Pour cette phase nous présenterons l'organigramme (**Fig. 2.4)** du sous programme qui permette la vérification de la connexion entre PIC et PC.

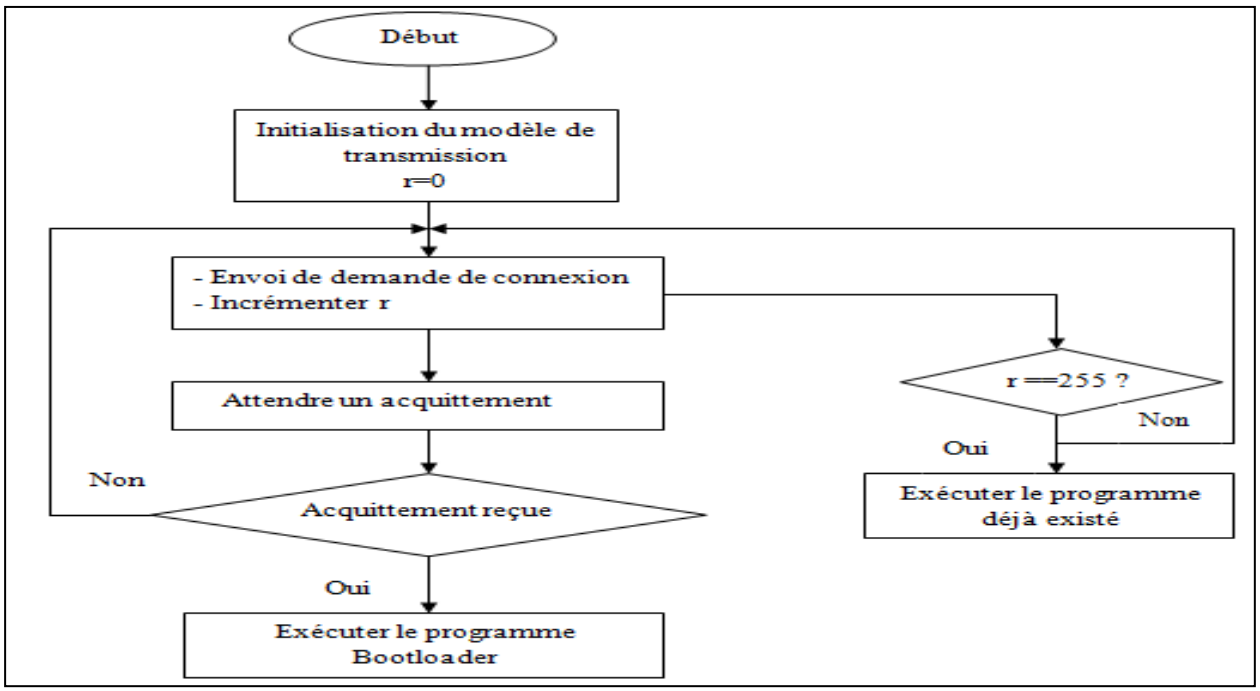

Fig. 2.4 : organigramme d'essai de connexion

- Une fois nous avons initialisé le module de transmission, le PIC commence son essai de connexion avec le PC par l'envoie d'une demande de connexion.
- Si le PIC reçoit la confirmation, le pic envoie un accusé de réception pour que le PC commence le transfert des données.
- Si le PIC ne reçoit pas l'accusé de connexion au bout de 5 secondes le PIC passe en mode d'exécution du programme se trouvant à partir de l'adresse 0x1FA0.

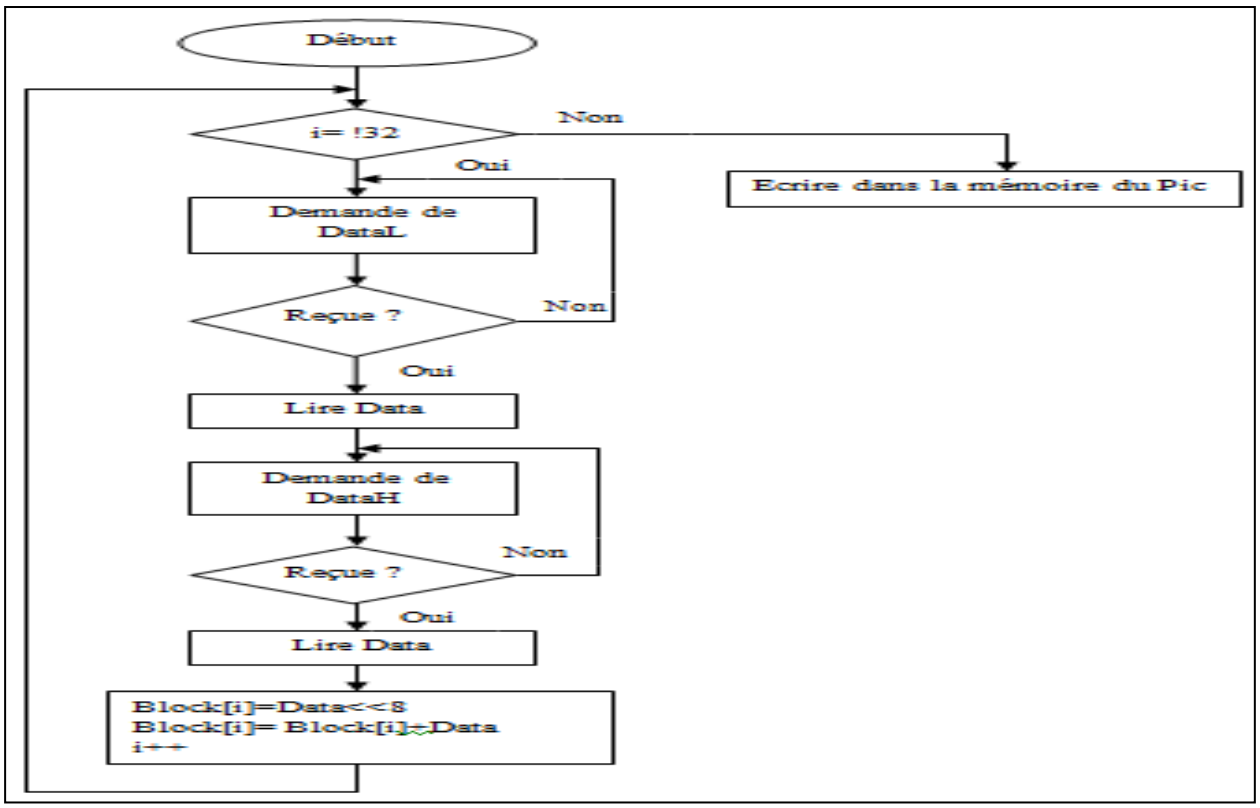

Fig. 2.5: organigramme de la fonction Bootloader

- La donnée envoyé par le PC se compose de : adresse poids faible ADDRL, adresse poids fort ADDRH, octet poids faible de l'instruction DATAL, octet poids fort de l'instruction DATAH.
- Le PIC attend au premier temps la réception du poids fort de l'adresse une fois la réception se fait correctement le PIC renvoie cette adresse car l'échange des données se fait d'une manière synchrone sinon il attend la réception de l'adresse.
- La synchronisation se fait aussi lors de l'envoie du poids faible de l'adresse ainsi l'envoie des octets faible et fort.
- Une fois le PIC reçoit l'ensemble des éléments de la donnée il écrit la donnée dans sa mémoire FLASH et fait une comparaison entre la donnée reçu et la donnée enregistrée.
- Si la donnée est bonne le PIC envoie un acquittement positif pour continuer le transfert des donnée mais si le PIC trouve que la donnée est erronée, il envoie un rejet au PC pour le renvoie de la donnée à nouveau.

3-Programme de transfert du côté PC :

Le programme de transfert côté PC est développé en langage de programmation C; le choix du compilateur est fait suivant la simplicité de la programmation de port RS232, puisqu'il travail sous dos il nous donne la main directement pour l'utilisation du port après certaine configuration des registres correspondantes au port RS232, par contre les autres compilateurs qui travaillent sous Windows, nécessite plusieurs commande pour dépasser la sécurité établie par le système d'exploitation.

Nous envoyons au PIC des fichiers de type .HEX, Ce fichier possède une structure organisée en ligne. Chaque ligne contient des champs, illustrés ici par des exemples:
06 0000 00 8A 15 0A 16 00 20 1B

- Les deux points indiquent le début de la ligne.
- 1 octet **06** précise le nombre de donnée dans la ligne.
- Les deux 2 dégites **00** indiquent qu'on a une donnée**.**
- Les 4 digits= 2 octets indiquent l'adresse de départ.
- Pour le **1B** 1 octet de checksum

4- Simulation :

Pour tester le bon fonctionnement du montage. Nous avons procédé à des simulations à l'aide du logiciel « ISIS PROTEUS ». Ce dernier est un progiciel de conception assistée par ordinateur de circuits électroniques. C'est un système de simulation du circuit, basé sur des modèles de composants électroniques réalisés dans PSPICE.

La simulation du BOOTLOADER nécessite une communication PC – PIC, le simulateur ISIS ne permet pas cette liaison, on a proposé de le faire testé en deux partie isolé PC et PIC.

→Test au niveau PIC :

On propose de faire un test qui contient :

- Le contrôle de flux : ceci est testé à l'aide d'un terminal virtuel disponible dans le simulateur qui joue le rôle du PC dans cette application, les données seront transmises manuellement et ils seront visualisés par un autre terminal virtuel.
- Un bouton pour un démarrage sur RESET lié au $\overline{MCLR}$
- Un bouton pour définir le mode d'utilisation du PIC: mode transfert / exécution

On donne le schéma utilisé pour la simulation (**Fig. 2.7**) :

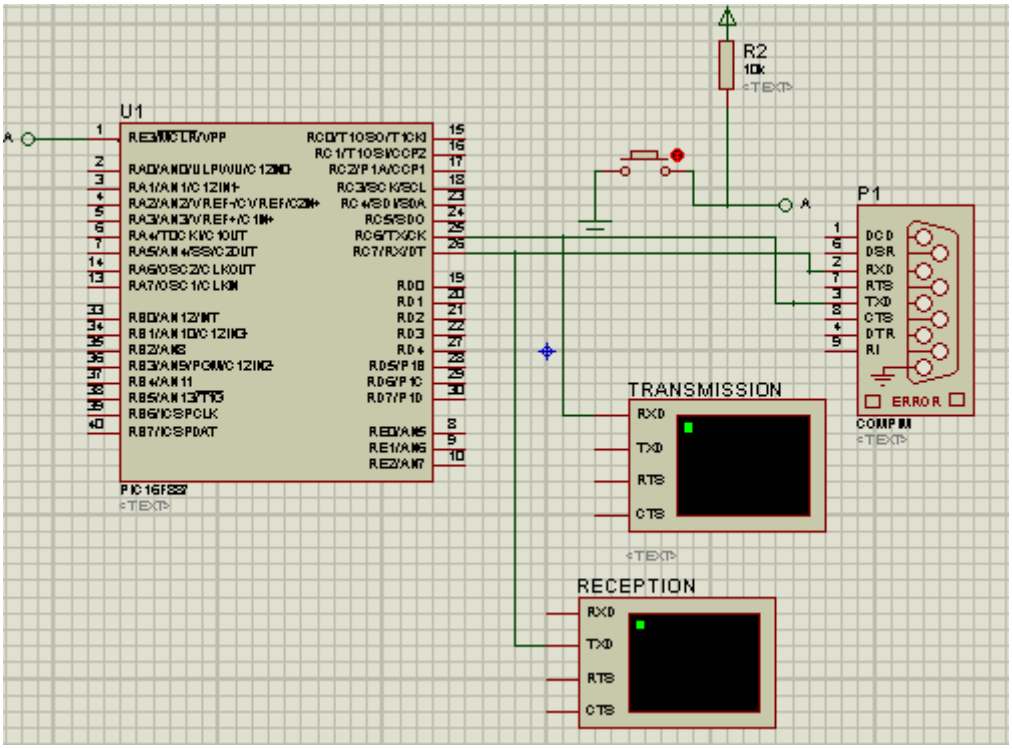

Fig. 2.7 : simulation ISIS

Nous traitons les différents cas possibles :

- ***Pic en mode exécution* (Fig. 2.8*) :***

Mode exécution équivalent à PC non connecté en faisant passer le RESET de 0 à 1, on remarque l'envoi du caractère **« g »** ce qui veut dire que compteur programme du PIC est en programme utilisateur, et l'écran de la réception n'affiche rien même si on donne des valeurs à charger.

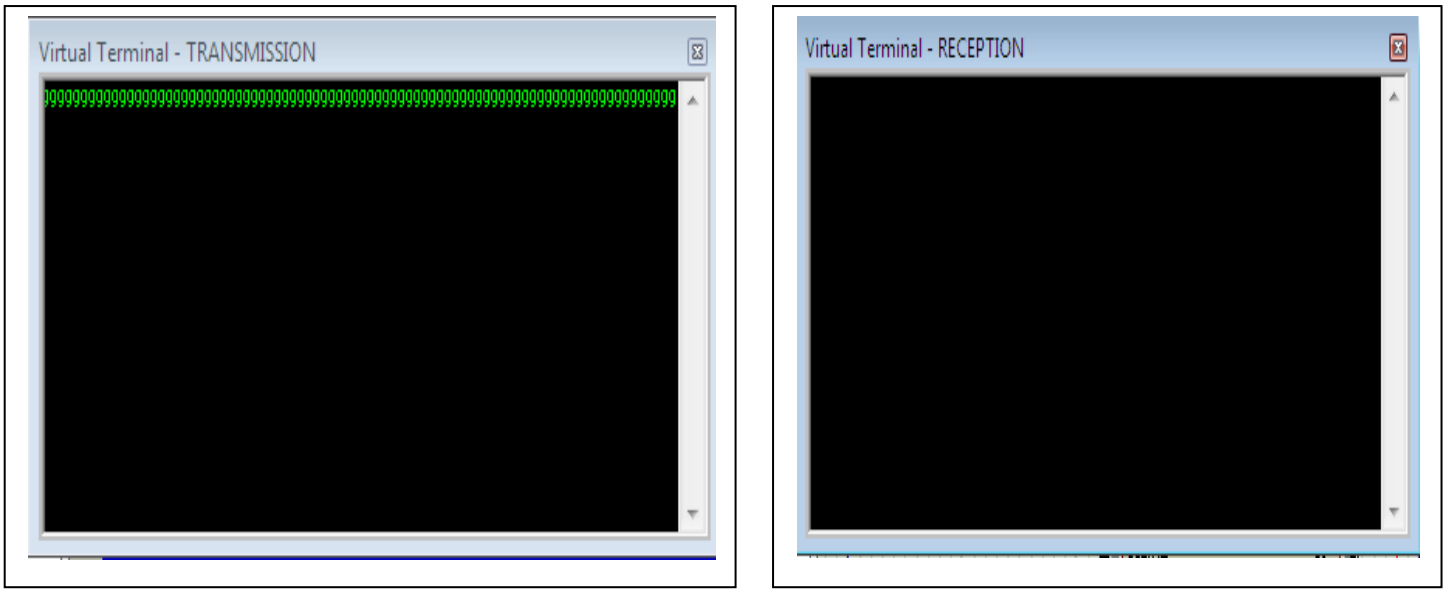

Fig. 2.8 : simulation ISIS (mode d'exécution)

Conclusion

Ce projet nous a permis de découvrir une nouvelle technique qui s'avère indispensable durant la programmation des microcontrôleurs sans utiliser un programmateur qui est celle du BOOTLOADER, pour ce faire on a essayé de suivre les étapes suivantes :

- Elaboration d'un programme BOOTLOADER pour un pic 16F887
- choix des composants matériels pour l'interfaçage PC-PIC,
- réalisation du schéma dans un logiciel de simulation " ISIS PROTEUS",
- programmation du microcontrôleur en langage C.
- Comprendre la structure des fichiers .HEX contenant le code exécutable à écrire dans la mémoire du PIC et réalisation d'un programme de transfert en C.
- simulation et test global du BOOTLOADER.

Les tests de simulation et les essais pratiques que nous avons effectués après avoir implémentés les deux programmes celui du BOOTLOADER et celui de l'utilisateur on a remarqué qu'il y a bien un transfert de données entre le PC-PIC mais malheureusement on avait toujours un problème d'adressages vu que notre application reste inexécutable et on ne voyait pas son influence au niveau du kit.

Webographie

http://paulfmcgowan.com/?page_id=64
http://www.youtube.com/watch?v=_PUh_Q__1qc
http://www.youtube.com/watch?v=o183dhOIWfk
http://elfugao.pagesperso-orange.fr/electronique/bootloader/bootloader.htm
http://www.picprojects.net/serialbootloader/
http://www.mikroe.com/forum/download/file.php?id=5886

Annexe

```
static unsigned block[32], rcv_1, rcv_2;

void Susart_Init(unsigned short brg_reg) org  0x1DCF {
  unsigned short i;

  RCSTA = 0x90; //d'activation du port série etActive la réception continue
  TXSTA = 0x26; // Registre de statut et de contrôle d'émission (USART)
  TRISC.B7 = 1;
  TRISC.B6 = 0;

  while (PIR1.RCIF) // Registre de signalisation d'interruptions de périphériques
    i = RCREG;  // Registre de réception de l'USART

  SPBRG = brg_reg; // Registre du Générateur de Baud de l'USART
}


void Susart_Write(char data_) org 0x1DB5  {

  while (!TXSTA.TRMT)// bit de statut du registre à décalage d'émission
    ;
  TXREG = data_; // Registre de transmission de l'USART
}

unsigned short Susart_Data_Ready() org 0x1DA2 {
  return (PIR1.RCIF); // Registre de signalisation d'interruptions de périphériques
}

unsigned short Susart_Read() org 0x1D8C {
  unsigned short rslt;
  rslt = RCREG; // Registre de réception de l'USART

  if (RCSTA.OERR) {  // Registre de statut et de contrôle de réception (USART) OERR :
bit d'indication de dépassemen ,1 = Erreur de dépassement (peut être effacé en effaçant le
bit CREN)
     RCSTA.CREN = 0;  // 1 = Active la réception continue
     RCSTA.CREN = 1;  // 0 = Désactive la réception continue
  }
  return rslt;
}

//-------------------------------------------------------------------------}
```

```
// After bootloading nops are replaced with page setting and goto to the location
// of the main of loaded program.
void Start_Program() org 0x1FA0  {
  //asm nop;
  //asm nop;
  //asm nop;
  //asm nop;
  portd=0xFF;
}

//-------------------------------------------------------------------------------}
// After bootloading reset vector is adjusted before writing it to new location
// (address of Start_program, 0x1FA0)
void AdjustGoto()             org 0x1D50 {

    rcv_1    = block[3];          // rcv variables are used as temps
    rcv_2    = block[4];          // for saving block[3..4]

    block[4] = block[2];           // page setting and goto to
    block[3] = block[1];           // the main of user's (loaded) program
    block[2] = block[0];           // (assuming that page is set to zero)

    block[0] = 0x118A;             // ensuring that PCLATH page bits are cleared
    block[1] = 0x120A;             // as after reset (requred by code block above)
}

//-------------------------------------------------------------------------------}
// This procedure will recieve the (start address DIV 32) in the ROM where
// the 32 words will be written to.
// eg. To write to location 0x0000 - Flash_Write_Address(0x00);
//     To write to location 0x0020 - Flash_Write_Adderss(0x01);
//     To write to location 0x1FA0 - Flash_Write_Adderss(0xFD);
void Flash_Write_Address(unsigned beginorg)  org 0x1DF2 {

    unsigned total;
    unsigned temp;
    char loop;
    char SaveIntCon;

    loop = 0;
    while (loop != 32) {
        total = (beginorg << 5) + loop;
        temp  = block[loop];
        SaveIntCon = INTCON;
        EEADR   = (char)(total);
```

```
        EEADRH  = (char)(total >> 8);
        EEDATA  = (char)(temp);
        EEDATH  = (char)(temp >> 8);
        EECON1.EEPGD = 1;
        EECON1.WREN  = 1;
        INTCON.GIE   = 0;
        EECON2 = 0x55;
        EECON2 = 0xAA;
        EECON1.WR    = 1;
        asm nop;
        asm nop;
        INTCON = SaveIntCon;
        EECON1.WREN = 0;
        loop++;
    }
}
//-------------------------------------------------------------------------}
// This function will send a char over the USART until another char is recieved
// or until a timeout is reached. If the correct char is recieved the function will
// return false else it will return true

unsigned short Susart_Write_Loop(char send, char recieve)    org 0x1ED2 {
    char  stop, r;

    r    = 0;
    stop = 0;
    while (stop != 1) {
        boot16_Delay_5ms();
        Susart_Write(send);
        boot16_Delay_5ms();
        r++;
        if ( Susart_Data_Ready() ) {
          if (Susart_Read() == recieve)  stop = 1;
          }
        if (r == 255) stop = 1;
    }

    return  r;

}

//-------------------------------------------------------------------------}
// This procedure will recover the bootlocation 0x0000, 0x0001 and 0x0002 to point
// to the bootloaer's main.
// It will also move the reset vector of the program that is uploaded to a new
```

```
// location. In this case it is address of Start_program, 0x1FA0.
void Write_Begin()              org 0x1E56    {

    AdjustGoto();               // Set page and goto to user's main

    Flash_Write_Address(0xFD);      // write 0xFD shl 5 = 0x1FA0
    block[0] = 0x160A;              // page setting
    block[1] = 0x158A;              // page setting
    block[2] = 0x2E7D;              // points to main procedure
    block[3] = rcv_1;               // rcv variables are used as temps
    block[4] = rcv_2;               // for saving block[3..4]
}

//-------------------------------------------------------------------------------}
// Handles comunication between PIC and PC
void Start_Bootload()           org 0x1F10  {

    char     i;
    unsigned j;

   i = 0;
   j = 0;


    for ( ; ; )  {
        if (i == 32) {          // If 32 words recieved then write to flash
            if (j == 0)  Write_Begin();
            Flash_Write_Address(j);
            i = 0;
            j++;
        }

        Susart_Write('y');                  // Ask for rcv_2
        do ; while (Susart_Data_Ready() != 1);
        rcv_2 = Susart_Read();                  // Read rcv_2

        Susart_Write('x');                  // Ask for rcv_1
        do ; while (Susart_Data_Ready() != 1);
        rcv_1 = Susart_Read();                  // Read rcv_1


        block[i] = rcv_1 << 8;              // Save rcv_1rcv_2 in block[i]
        block[i] = block[i] + rcv_2;
        i++;
                    // Next word
```

```
        }
    }



    //--------------- Boootloader main for PIC16's
    void main() org 0x0100 {
      // 16Mhz --> 103          look for asynchronous high speed table
      //  8Mhz -->  51
      //  4Mhz -->  25
      //           |
      //           |
      //           |
      //          \ /
      //           |
      UART1_Init(9600);              // Initialize UART module at 9600 bps
     Delay_ms(100);                 // Wait for UART module to stabilize
      trisd=0;
      portd=0;
      Susart_Init(51);                        // Init USART at 9600
      if (Susart_Write_Loop('g', 'r') != 255) { // Send 'g' for ~5 sec, if 'r'
       Start_Bootload();                      //   received start bootload
      }
      else {
       Start_Program();                       //   else start program
      }

}
```