Création d’un BOOTLOADER

*Précaution : ce tuto a été fait avec MPLABX v2.15 et XC32 v1.31*

Pour créer une bootloader il est nécessaire d’avoir 1 programme qui sera le bootloader ET un programme qui sera l’application lancée par le bootloader. Donc deux projets distinct.

Le projet Bootloader : Il va permettre de lire un fichier .HEX et l’écrire dans la mémoire FLASH du PIC.

Le projet APP : c’est le projet qui va être écrit dans la mémoire FLASH du PIC lorsqu’un évènement l’ordonnera.

*Le projet Bootloader*

*Un chargeur d'amorçage (ou bootloader) est un* [*logiciel*](http://fr.wikipedia.org/wiki/Logiciel) *permettant de lancer un ou plusieurs* [*systèmes d'exploitation*](http://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27exploitation) *(multi-*[*boot*](http://fr.wikipedia.org/wiki/Boot)*), c'est-à-dire qu'il permet d'utiliser plusieurs systèmes, à des moments différents, sur la même machine. Plus spécifiquement pour un microcontrôleur type PIC le bootloader est un programme qui permet de démarrer un programme (App) lorsqu’un évènement définis est activé (bouton, variable, code ASCII,…). Enfaite le bootloader est le programme MAITRE, c’est lui qui est exécuter avant tout application si et seulement si un bootloader est intégré.*

Dans ce projet il est doit y avoir 3 fonctions principale, c’est-à-dire 3 actions qui vont permettre de reprogrammer le PIC. Ces actions sont : le déclenchement du Bootloader, l’effacement de la mémoire FLASH du PIC et la programmation de la mémoire avec le nouveau programme (APP).

1-Le Bootloader est effectué à chaque démarrage de la carte électronique. Cependant deux chemins peuvent être empruntés selon « le déclenchement du Bootloader ». Le programme bootloader ici décrit à deux choix, l’un est de lancer directement l’application si une variable est à 0 sinon si cette variable est à 1 alors on va entrer dans une boucle qui va permettre de reprogrammé le PIC.

Exemple :

Une variable noté File\_Up permet de déclencher ou non la reprogrammation du PIC.

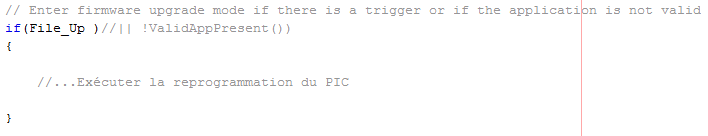
Si File\_Up = 1

Lancer l’application

Reprogrammer le PIC

Oui

Non



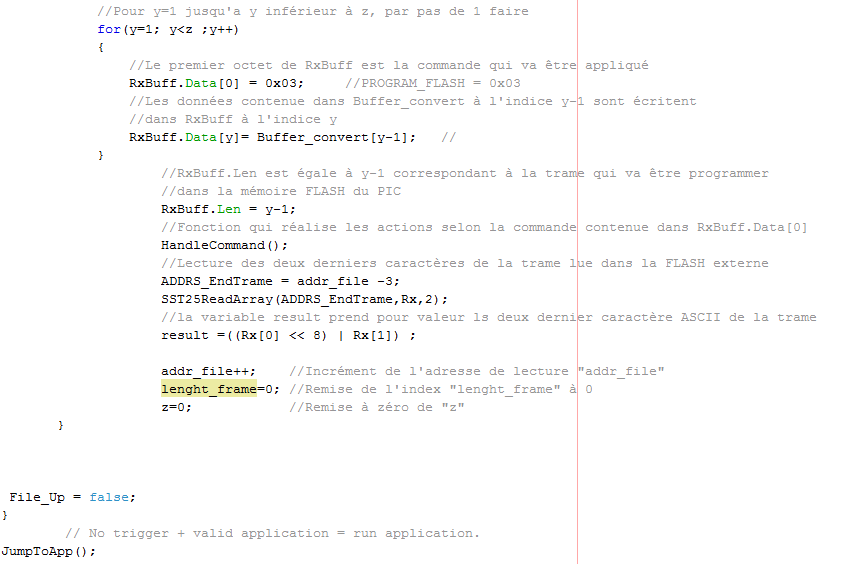
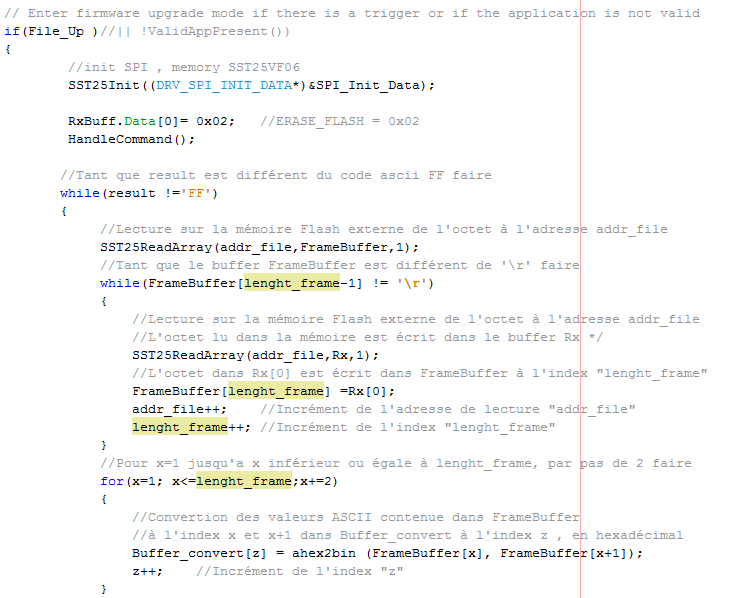
2- *L’effacement de la mémoire FLASH du PIC*: Si l’évènement de déclenchement (File\_Up =1) est égale à 1 alors on entre dans la boucle du bootloader qui va permettre de reprogrammer le micro. Avant tout l’ancienne application (App) écrite dans la mémoire FLASH du microcontrôleur doit être effacée. Pour faire ceci, la fonction EraseFlash est appeler par le biais d’un command (ici attribuer à 0x02 ).

Code :



3- L*a programmation de la mémoire avec le nouveau programme (APP)*: Une fois la mémoire FLASH du PIC effacer l’écriture de la nouvelle application en mémoire FLASH du PIC peut être effectuée.

Code :



***FrameBuffer = trame lue dans la flash externe entre le caractère ‘ :’ et ‘/r’.***

***Ex : FrameBuffer[x] = ‘0’***

***FrameBuffer[x+1] = ‘2’***

***Buffer\_convert[z]=0x02***

***Ici CMD=0x03***

***RxBuff.Data [0]= CMD.***

***RxBuff.Data[1]=0x02***

***RxBuff.Data[y]=0x00***

Explication du SOFT :

Le SOFT bootloader se base sur le projet « UART\_Btl\_Explorer16.X ». Dans ce projet, la reprogrammation du PIC est effectuer via une liaison UART (RS322) et une application via un PC. Tous le procéder est expliqué dans l’application Note AN1388 de microchip (Voir page 1). En résumer on Flash le bootloader sur le micro, on lance l’application sur la PC afin d’envoyer les commandes EraseFlash, Program et Verify. Le but a donc été de remplacer cette application PC directement sur la bootloader.

Ce qui veut dire que lorsque File\_Up = 1 on simule la commande ERASE\_FLASH puis PROGRAM\_FLASH (La vérification n’est pas encore effectuée malgré les CRC en fin de trame 20/01/2015)

ERASE\_FLASH : L’effacement de la flash du PIC ce fait dans la fonction *HandleCommand().* La flash du micro est effacé de l’adresse *APP\_FLASH\_BASE\_ADDRESS* à *APP\_FLASH\_END\_ADDRESS.*

PROGRAM\_FLASH : La programmation de la flash est faite dans la fonction *HandleCommand().* Avant cette programmation on charge dans un buffer ligne par ligne les trames lues dans la Flash externe. Le chargement de ces lignes est fait dans le code ci-dessus (p3).

On résume, le projet Bootloader permet de reprogrammer un microcontrôleur de type PIC32MX dans le cas où File\_Up =1.

EUUuuuhhh NON si on laisse en l’état rien ne se programmera…

En effet, si on dit à quelqu’un de reconstruire sa maison sans plans , sans directive et surtout sans lieu de reconstruction comment est-ce possible de le faire correctement ? C’est ici que les fichiers LINKER font leurs entrés.

Les LINKER :

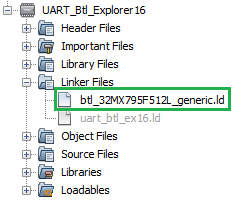
Les fichiers linker (.ld) permettent de faire le lien entre deux programme ou plutôt décrire le chemin de ces deux programmes dans la mémoire FLASH du microcontrôleurs. En clair il indique l’endroit où doit être écrit le bootloader et l’application.

OK et alors on en fait quoi ?

Et bien il faut ajouter à nos deux programme un fichier linker bien précis.

Dans le programme du bootloader sous « H:\Bible C\Bootloader PIC32mx795F512L\Bootloader\_externFlash\src » on utilise le microcontrôleur PIC32MX695F512L.

Il faut donc intégrer à ce projet le linker correspondant. Les différents Linker ce trouve sous « H:\Bible C\Bootloader PIC32mx795F512L\PIC32MX\_others ».



Un fois ceci fait il faut le faire également dans l’application que l’on souhaite charger dans le PIC (AVEC LE MEME FICHIER !!!).

Le fichier btl\_32MX…………\_generic.ld est composé de centaine de lignes. Seulement une dizaine sont utiles dans notre cas (je ne connais pas les autres...) :

OPTIONAL("libmchp\_peripheral.a")

OPTIONAL("libmchp\_peripheral\_32MX795F512L.a")

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* For interrupt vector handling

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PROVIDE(\_vector\_spacing = 0x00000001);

\_ebase\_address = 0x9FC01000;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Memory Address Equates

\* \_RESET\_ADDR -- Reset Vector

\* \_BEV\_EXCPT\_ADDR -- Boot exception Vector

\* \_DBG\_EXCPT\_ADDR -- In-circuit Debugging Exception Vector

\* \_DBG\_CODE\_ADDR -- In-circuit Debug Executive address

\* \_DBG\_CODE\_SIZE -- In-circuit Debug Executive size

\* \_GEN\_EXCPT\_ADDR -- General Exception Vector

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\_RESET\_ADDR = 0xBFC00000;

\_BEV\_EXCPT\_ADDR = (0xBFC00000 + 0x380);

\_DBG\_EXCPT\_ADDR = (0xBFC00000 + 0x480);

\_DBG\_CODE\_ADDR = 0xBFC02000;

\_DBG\_CODE\_SIZE = 0xFF0 ;

\_GEN\_EXCPT\_ADDR = \_ebase\_address + 0x180;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Memory Regions

\*

\* Memory regions without attributes cannot be used for orphaned sections.

\* Only sections specifically assigned to these regions can be allocated

\* into these regions.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

MEMORY

{

kseg0\_program\_mem (rx) : ORIGIN = 0x9D000000, LENGTH = 0x6000 /\* All C Files will be located here \*/

kseg0\_boot\_mem : ORIGIN = 0x9FC00490, LENGTH = 0x0 /\* This memory region is dummy \*/

exception\_mem : ORIGIN = 0x9FC01000, LENGTH = 0x1000 /\* Interrupt vector table \*/

kseg1\_boot\_mem : ORIGIN = 0xBFC00000, LENGTH = 0x490 /\* C Startup code \*/

debug\_exec\_mem : ORIGIN = 0xBFC02000, LENGTH = 0xFF0

config3 : ORIGIN = 0xBFC02FF0, LENGTH = 0x4

config2 : ORIGIN = 0xBFC02FF4, LENGTH = 0x4

config1 : ORIGIN = 0xBFC02FF8, LENGTH = 0x4

config0 : ORIGIN = 0xBFC02FFC, LENGTH = 0x4

kseg1\_data\_mem (w!x) : ORIGIN = 0xA0000000, LENGTH = 0x20000

sfrs : ORIGIN = 0xBF800000, LENGTH = 0x100000

configsfrs : ORIGIN = 0xBFC02FF0, LENGTH = 0x10

}

Ces différentes lignes sont des adresses de mémoire FLASH du PIC.

**Le fichier Linker du Bootloader** va rester intact à l’exeption de la ligne :

kseg0\_program\_mem (rx) : ORIGIN = 0x9D000000, LENGTH = 0x6000 //C’est la taille du bootloader 6\*4096 =>multiple de 4ko

Où LENGTH doit correspondre à la taille en hexadecimal du fichier HEX du bootloader. Si ce n’est pas fait et que le bootloader est plus grand que la valeur de LENGHT à cette ligne l’application sera écrite sur le programme bootloader et donc ça ne fonctionnera pas. (Bien entendu une protection est intégrer si cela ce produit).

**Le fichier Linker de l’app** va être modifié selon le linker du bootloader. Les lignes

\_ebase\_address = prend la valeur de kseg0\_program\_mem + LENGTH du linker bootloader =>0x9D000000+0x6000=0x9D006000

\_RESET\_ADDR = prend la valeur de \_ebase\_address +0x1000 +0x970 => 0x9D006000+0x1000 +0x970

\_BEV\_EXCPT\_ADDR = prend la valeur de \_RESET\_ADDR + 0x380;

\_DBG\_EXCPT\_ADDR = prend la valeur de \_RESET\_ADDR + 0x480);

kseg0\_program\_mem (rx) : ORIGIN = RESET\_ADDR + 0x490, LENGTH = (longueur de l’app – taille du boot -0x1000-0x970-0x490)

kseg0\_boot\_mem : ORIGIN = \_ebase\_address + 0x1000 , LENGTH = 0x970 /\* This memory region is dummy \*/

exception\_mem : ORIGIN = \_ebase\_address, LENGTH = 0x1000 /\* Interrupt vector table \*/

kseg1\_boot\_mem : ORIGIN = \_RESET\_ADDR, LENGTH = 0x490 /\* C Startup code \*/

Lorsque les deux Linker sont bien configurer il faut ajouter dans le projet bootloader les variables correspondant aux adresses inscrites dans le LINKER de l’app.

C’est variable sont à définir dans le Bootloader.h sous cette forme :

#define APP\_FLASH\_BASE\_ADDRESS \_ebase\_address du linker APP (Attention)

#define APP\_FLASH\_END\_ADDRESS PROGRAM\_FLASH\_END\_ADRESS

#define PROGRAM\_FLASH\_END\_ADRESS (kseg0\_program\_mem+BMXPFMSZ-1) du ld Boot

#define USER\_APP\_RESET\_ADDRESS \_RESET\_ADDR du linker APP (Attention)

Exemple:

Pour les linker en page 5 les variables écrites dans le Bootloader.h seront:

#define APP\_FLASH\_BASE\_ADDRESS 0x9D006000

#define APP\_FLASH\_END\_ADDRESS PROGRAM\_FLASH\_END\_ADRESS

#define PROGRAM\_FLASH\_END\_ADRESS (0x9D000000+BMXPFMSZ-1)

#define USER\_APP\_RESET\_ADDRESS 0x9D006000+0x1000 +0x970

A cet instant et seulement après avoir fait ceci on peut démarrer les étapes afin de reprogrammer le PIC :

-Exporter le fichier HEX de l’application que l’on veut programmer.

-L’intégrer sur le serveur FTP

-Envoyer vien l’interface Web ChargeData la trame qui permet de charger le fichier sur la mémoire externe (SST25) via l’application Bubble\_M\_V2.4.

-Attendre que la carte reboot toute seul et constater si c’est OK ou non.

Lorsque le programme s’écrit dans le flash la LED centrale clignote BLEU, et Vert lorsque le fichier est écrit en mémoire.