Compte rendu recherches sur le bootloader

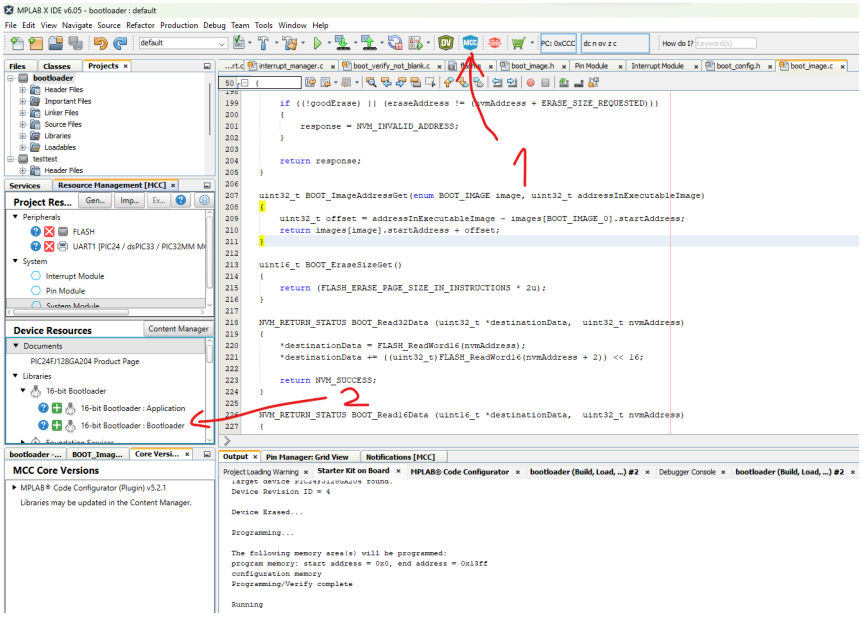
Table des matières

Tuto bootloader et bootloader application

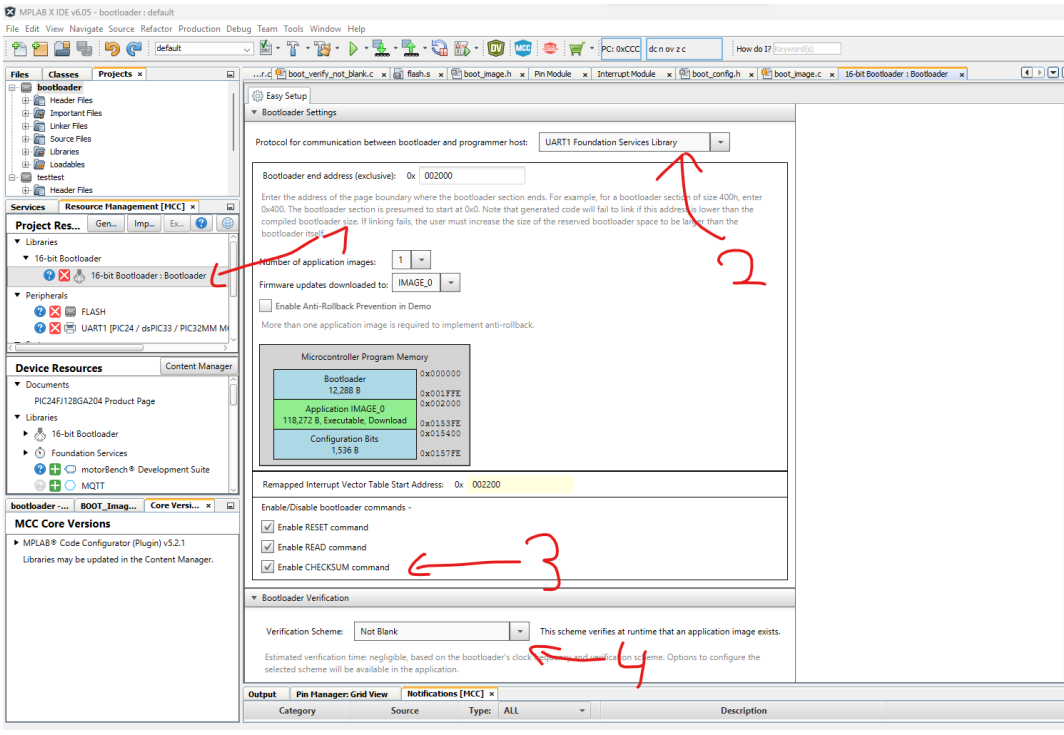
Comment générer le bootloader avec MCC

Une fois sur votre projet, lancez MCC puis allez dans les librairies à votre disposition et sélectionnez « Bootloader : BOOTLOADER »

À partir de là, vous avez accès à la documentation Microship en appuyant sur le point d'interrogation à gauche de la librairie bootloader active.



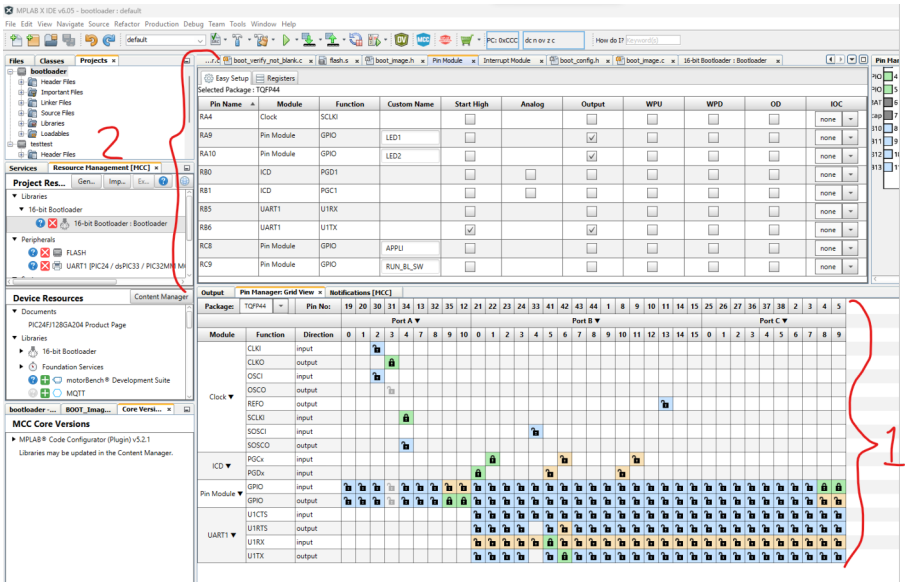
Double-cliquez sur la librairie bootloader active afin d'ouvrir la fenêtre de configuration. Sélectionnez le protocole de communication (2) « UART1 Foundation Library ». Cochez aussi « enable checksum » et sélectionnez la méthode de vérification : Not Blank.



Maintenant, nous allons configurer les pins. Sélectionnez la PIN PC9 en input, PA9 en output, RB5 en U1RX et RB6 en U1TX. Dans la fenêtre PIN Module, au-dessus, précisez que RB6 commence à l'état haut.

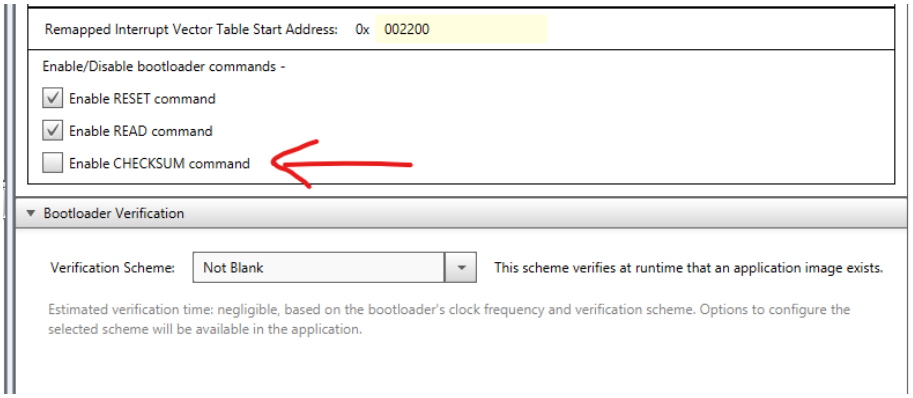
Suite à cela, mettez les labels.

LED1 pour RA9
RUN_BL_SW pour RC9
Les autres labels ne sont pas utiles et ont été initialisés pour des tests.



En suite, dé-sélectionnez « Enable checksum command ». Veuillez générer le projet et le compiler pour voir s'il n'y a pas d 'erreurs.

Suite à cela, il faut modifier le code pour faire clignoter la LED et mettre un bouton pour indiquer que nous voulons rentrer dans le bootloader.



Dans boot_demo.c veuillez coller :

```
#include "../pin_manager.h"
void updateLeds()
{
    #define LED_RATE 0x8000
    static int count=0;
    count++;
    if ((count % LED_RATE)==0)
    {
        LED1_SetLow();

        if ((count/LED_RATE) & 1)
            LED1_SetHigh();

    }
}
```

à la suite de :

```
static bool inBootloadMode = false;
static bool executionImageRequiresValidation = true;
static bool executionImageValid = false;

static bool EnterBootloadMode(void);
```

Cela permettra de faire clignoter la LED1

En suite, modifiez

```
static bool EnterBootloadMode(void)
{

            /* Replace with your own logic here for when to stay in boot load mode. */
            // return false;
    return (RUN_BL_SW_GetValue()==0);
}
```

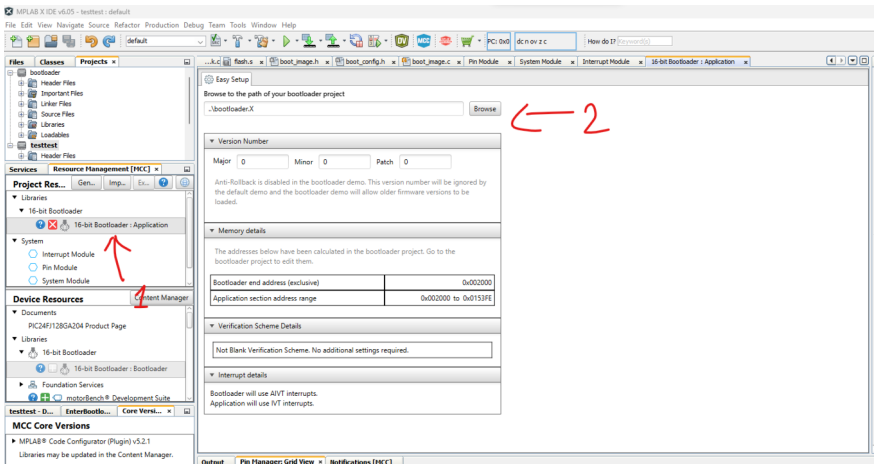
afin que par l'appuit du boutons S1, on indique que l'on veut entreer dans le bootloader.

Vous pouvez téléverser sur la carte.

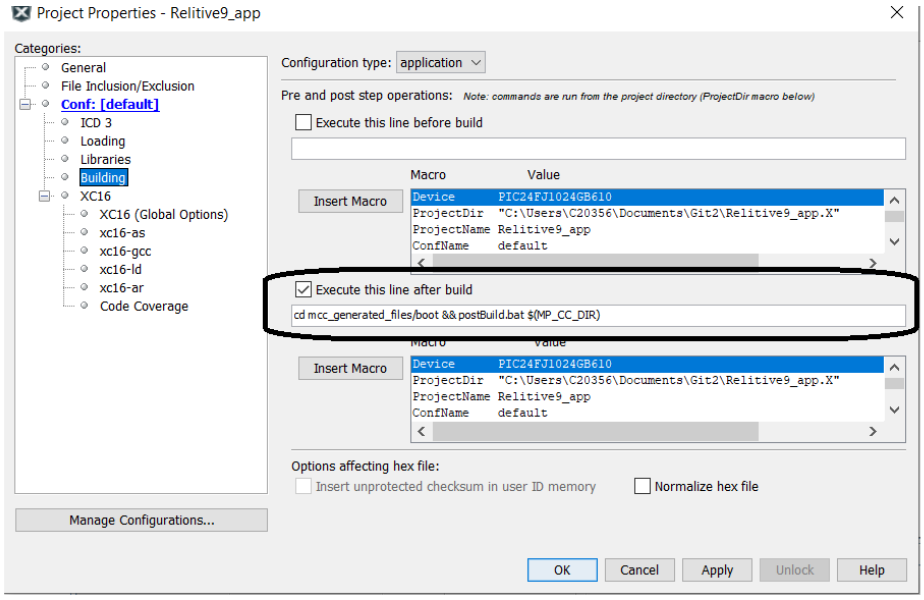
Normalement, au lancement, la LED 1 devrait clignoter.

Comment générer le bootloader application avec MCC

Ouvrir le code de votre application dans MPLAB, mettez le main projet et lancez MCC. Sélectionnez la librairie Bootloader : application
Une fois celle-ci activée, double cliqué dessus et sélectionner en haut de la fenêtre qui vient de s'ouvrir le bouton browse. Sélectionnez le dossier contenant votre bootloader et ouvrez le. Vérifiez que les adresses de l'application coïncident avec celles du bootloader



En suite, faite un clic droit sur votre projet et sélectionnez 'Properties', puis building et cocher la case 'Execute this line after the build'. En suite, entrez la ligne de code :
cd MCC_generated_files/boot && postBuild.bat $(MP_CC_DIR) ${ProjectDir} ${ImageDir} ${ImageName}



Vous pouvez générer le MCC et compiler votre projet, veuillez notez le chemin vers votre document .HEX

Code pour communiquer sur le RS 232

Pour ce script python, j'ai utilisé la librairie « serial ». Le bootloader peut attendre des commandes sur l'UART, dans le programme, vous pouvez retrouver une catégorie « bootloader commands » ? Celle-ci répertorie toutes les commandes compréhensibles par le bootloader,
Une commande est décomposée suivant le schéma suivant : un nom avec un byte qui correspond à son « étiquette » en suite, vous avez la commande test à rentrer dans le main après la vérification « if __name__ == '__main__':» ensuite, il y aura de renseigné si la commande a été testée fonctionnelle ou non et pour finir, le nombre de bytes à passer en paramètres à la fonction main à la suite de la commande.

main() : attend un premier paramètre string et un deuxième qui est un entier
main("090000000000000000000",37)
Le main fonctionne en ouvrant un port COM qu'il faut renseigner dans
    ser = serial.Serial('COM3', 38400 , EIGHTBITS ,PARITY_NONE, STOPBITS_ONE, timeout=1000) # open serial port
Il écrit la commande passée en paramètre et lit ce que le bootloader lui renvoie, il affichera dans la console ce qu'il a lu du bootloader
Ensuite, il ferme le port.

application_offset_addr() n'est pas utilisée, mais permet de mettre un offset sur l'adresse d'écriture de la data

application_transmit() se base sur la lecture du document.HEX en exploitant le schéma d'écriture suivant du .HEX :
- 1 byte : taille de la donnée à transmettre
- 2 bytes : adresse d'écriture
- n bytes : data
- dernier byte : checksum

Cette fonction attend deux paramètres en entrée : l'intégralité du .HEX à la suite (sans '\n', ce qui se fait en Ctrl +shift + Alt + flèche du bas) et l'adresse d'offset qui n'est pas utilisée, mais laissée dans le doute

Étapes d'utilisation

Dans cette partie, nous allons aborder le processus total. Pour commencer, téléverser votre code sur la carte, puits lancer le programme python. Si celui-ci affiche « com3 » sans progresser, alors killer la console, appuyez sur le bouton reset de la carte et relancer. La fin du programme python est marquée par l'affichage d'un tableau conséquent.

Pour lancer une commande bootloader, commenté la fonction application_transmit() et initié un main() avec votre commande et les valeurs à lire. Quand vous lancez, si il est juste affiché

« COM3 », fermer la console et relancer. Si le problème persiste, c'est que la valeur de données à lire est fausse et qu'il faut y aller à tâtons.

Vous retrouverez le descriptif des commandes dans la doc bootloader et sur les screens dans le ficher « screen_bootloader-commands »

Sources de problèmes

Lors de l'exécution de tout le processus en mode debugger, nous avons remarqué qu'après le reset via MPLAB, le problème était lors de la méthode de vérification et donc ne nous faisait jamais passer dans le mode application, mais juste dans le bootloader ; le chemin suivit est :

-boot_demo.c : ligne 108 executionImageValid mène a boot_verify_not_blank.c. La bool BOOT_ImageVerify(enum BOOT_IMAGE image) renvoie false ce qui empèche d'entrée dans l'application.

Pistes non explorées

Lors de nos recherches, nous avons vu qu'il y avait des pistes à explorer :
- Se renseigner sur HEXMATE
- Regarder dans la doc de XC 1.6
- Jeter un œil plus en détail au post build script

Autre

Lien vers Unified Host application (qui ne marche pas)

https://www.microchip.com/en-us/software-library/dspic33-pic24-bootloader

lien vers un dépôt Git d'un bootloader maison et la vidéo explicative

https://github.com/BroadwellConsultingInc/BootloaderPIC16F15214.git

https://www.youtube.com/watch?v=OfW4hHFVy3U