

Bootloader

Rapport technique



Branche :	POBJ
Nom du document :	2203_Bootloader-Rapport_MRI_v1.0.0.pdf
Réalisé par :	Meven Ricchieri
A l'attention de :	Mr. Bovey
Date de début :	9 février 2023
Date de fin :	2 juin 2023

Table des matières

1	Introduction	2
1.1	Avancement.....	2
1.2	Objectif	2
2	Documentation.....	3
2.1	Emplacement du bootloader dans le PIC32	3
2.2	Communication entre le PC et le MCU.....	4
2.3	Format de fichier HEX.....	4
3	Analyse réelle	6
3.1	Méthode	6
3.2	Schéma de mesure	6
3.3	Mesures	6
3.4	Analyse.....	7
4	Algorithme	9
5	Conclusion	10
6	Sources	11
7	Annexes	11
7.1	Journal de travail	11

1 Introduction

Dans le cadre de ce projet de POBJ numéro 2203, ma mission consiste à poursuivre le développement d'un logiciel visant à remplacer l'utilitaire existant de Microchip, PIC32 Bootloader Application V1.2. L'objectif principal de ce logiciel est de remédier à une limitation de l'utilitaire actuel, l'absence d'une fonctionnalité permettant de présenter une liste de programmes PIC32 et de permettre aux utilisateurs de choisir celui à introduire dans le microcontrôleur. Le logiciel offrira la possibilité de choisir le mode d'envoi du programme au MCU, que ce soit via USART ou USB.

1.1 Avancement

Ci-dessous se trouve une liste des éléments déjà réalisés :

- Procédure de mise en place du programme de bootloader dans le PIC32, documentée et fonctionnelle
- Procédure de génération de fichiers HEX destinés à un bootloader, documenté et fonctionnelle
- Interface graphique du logiciel PC remplaçant celui de Microchip, réalisée et documentée

1.2 Objectif

L'objectif actuel est donc de réaliser le mécanisme permettant d'envoyer les fichiers HEX au MCU tout en respectant le protocole de communication de Microchip.

J'ai donc commencé par effectuer différentes recherches d'informations sur le bootloader, le type de fichier HEX à envoyer au MCU et le mode d'envoi. En parallèle de cela, j'ai analysé les trames envoyées par le PC lors de l'utilisation du logiciel fourni par Microchip, PIC32 Bootloader Application V1.2, afin de mieux comprendre comment cela fonctionne réellement.

2 Documentation

2.1 Emplacement du bootloader dans le PIC32

Les bootloaders de taille réduite sont placés dans la mémoire flash d'amorçage du PIC32. Le fait de placer l'application bootloader dans la mémoire Flash d'amorçage permet à l'utilisateur de disposer d'une mémoire Flash de programme complète pour l'application de l'utilisateur.

Dans le cas de bootloaders dépassant la taille de la Flash de démarrage du PIC32, le bootloader est divisé en deux parties. La table des vecteurs d'interruption (IVT) et le code de démarrage C sont placés dans la Flash d'amorçage, et la partie restante du chargeur d'amorçage est placée dans le programme Flash.

FIGURE 1: BOOTLOADER PLACEMENT

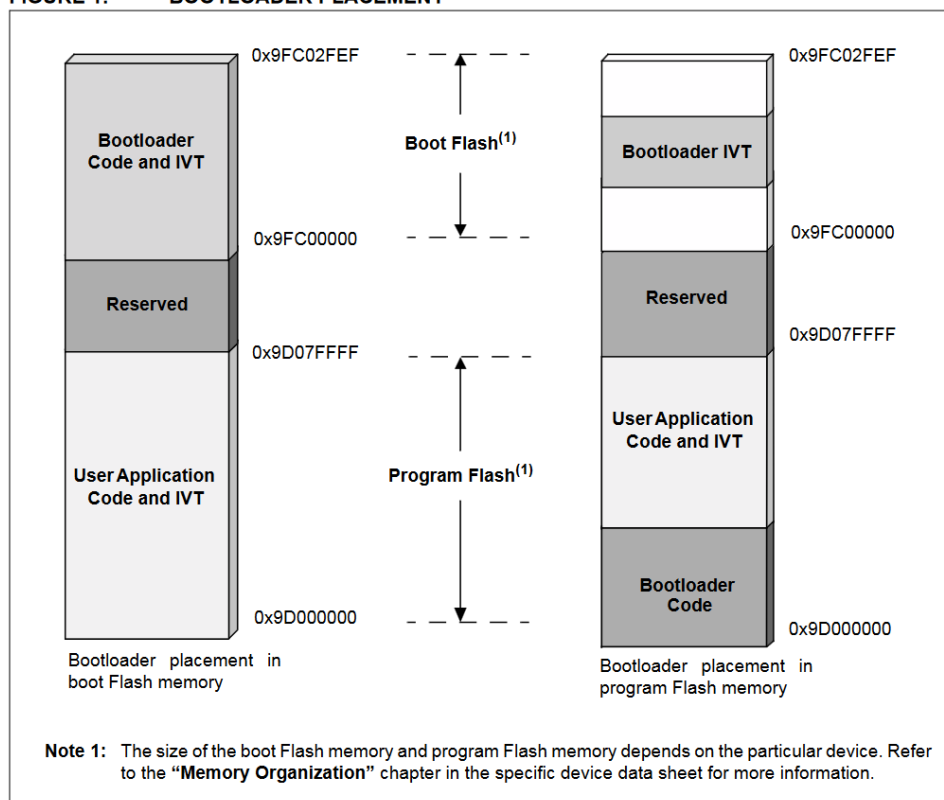


Figure 1 – Bootloader placement

2.2 Communication entre le PC et le MCU

L'application bootloader est implémentée à l'aide d'un framework. Le logiciel du bootloader communique avec l'application hôte du PC à l'aide d'un protocole de communication prédéfini. Le framework de démarrage fournit des fonctions d'interface de programmation d'application (API) pour gérer les trames liées au protocole provenant de l'application PC.

FIGURE 2: BOOTLOADER ARCHITECTURE (UART, USB HID, AND ETHERNET)

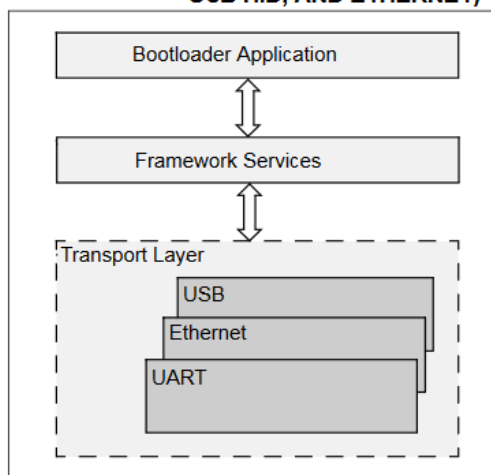


Figure 2

2.3 Format de fichier HEX

Le format de fichier demandé par le bootloader est un type bien connu, il s'agit de l'Intel Hex Format.

EXAMPLE 3-1: HEX FILE RECORD FORMAT

```

:BBAAAATTHHH.....HHCC
:100F90000E1022000E1023000C1121000C1524004D

:      Record Start Character
BB      two digit byte count specifying the number of data bytes in
        this record.
AAAA    Four digit starting address of this data record
TT      Two digit record type
        00 = data record
        01 = End of File record
        02 = Segment Address Record
        04 = Extended Linear Address record
HH      Data Bytes
CC      Two digit checksum calculated as 2's complement of all
        preceding bytes in data record except the colon.
  
```

Figure 3

Cependant, le bootloader prévoit des caractères de contrôle supplémentaires qui peuvent poser des problèmes s'ils sont mal interprétés.

TABLE 3: CONTROL CHARACTER DESCRIPTIONS

Control	Hex Value	Description
<SOH>	0x01	Marks the beginning of a frame
<EOT>	0x04	Marks the end of a frame
<DLE>	0x10	Data link escape

Figure 4

En effet, certains octets du champ de données peuvent imiter les caractères de contrôle SOH et EOT. Le caractère d'échappement de la liaison de données (DLE, Data Link Escape) est utilisé pour échapper à ces octets qui pourraient être interprétés comme des caractères de contrôle. Le bootloader accepte toujours l'octet suivant un <DLE> comme des données et envoie toujours un <DLE> avant tout caractère de contrôle.

L'application hôte du PC peut envoyer les commandes énumérées dans le tableau ci-dessous au bootloader. Le premier octet du champ de données (suivant le caractère de contrôle qui ne fait pas partie des données) contient la commande.

TABLE 4: COMMAND DESCRIPTION

Command Value in Hexadecimal	Description
0x01	Read the bootloader version information
0x02	Erase Flash
0x03	Program Flash
0x04	Read CRC
0x05	Jump to application

Figure 5

3 Analyse réelle

3.1 Méthode

Afin de mieux comprendre la documentation lue, j'ai effectué différentes mesures sur les trames envoyées par l'utilitaire de Microchip au MCU lors de la programmation via USART. J'ai réalisé ces mesures en utilisant un analyseur logique afin de décoder plus facilement les données qu'avec un oscilloscope. J'ai donc branché deux sondes sur l'USART du microcontrôleur, une sur la sortie de TX et l'autre sur l'entrée RX. De ce fait je peux observer tous les signaux transitant sur ces lignes. Une fois l'installation terminée, j'ai lancé la programmation du PIC32 via l'utilitaire de Microchip.

3.2 Schéma de mesure

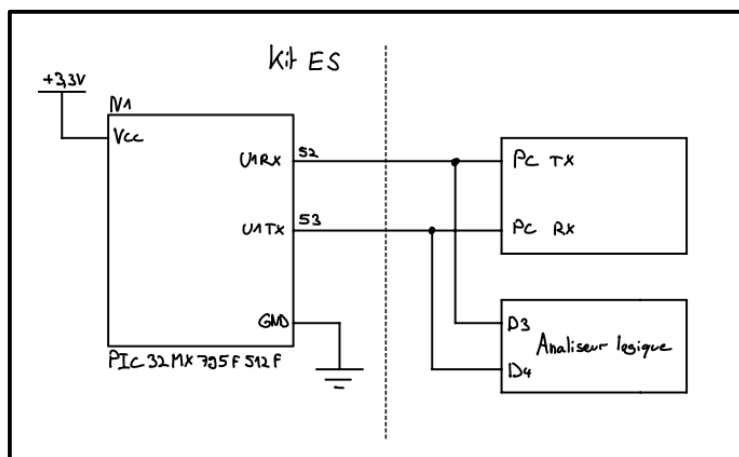


Figure 6 – Schéma de mesure

3.3 Mesures

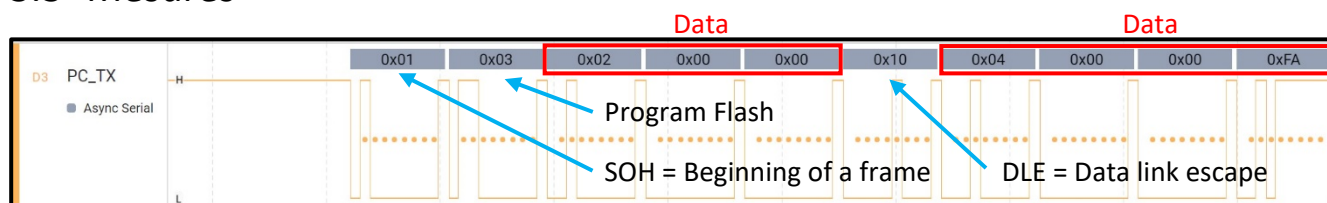


Figure 7 - Mesure 1, ligne 1 du fichier HEX

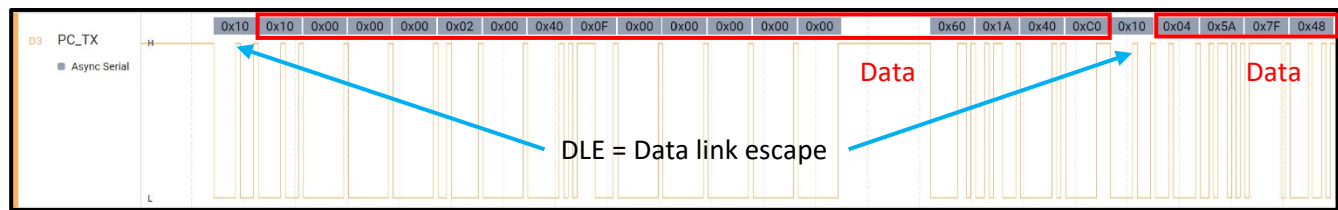


Figure 8 - Mesure 2, ligne 15 du fichier HEX

3.4 Analyse



A l'aide du fichier HEX du programme, nous pouvons constater que le logiciel effectue quelques traitements avant d'envoyer les bytes au microcontrôleur via l'interface sélectionnée. Dans le fichier HEX, les bytes de contrôle et de commandes ne sont pas présentes. De ce fait le logiciel de Microchip effectue lui-même ces modifications avant de tout envoyer au PIC.

Toutefois, la communication nécessite encore de la recherche, car le PC (host) s'arrête d'envoyer des données tous les temps de temps afin de recevoir une réponse du PIC. Cette partie doit être clarifiée avant de pouvoir générer un algorithme permettant de lire le HEX file, le traiter puis tout envoyer au PIC. Je n'ai jusqu'à présent trouvé la raison pour laquelle il se stoppe.

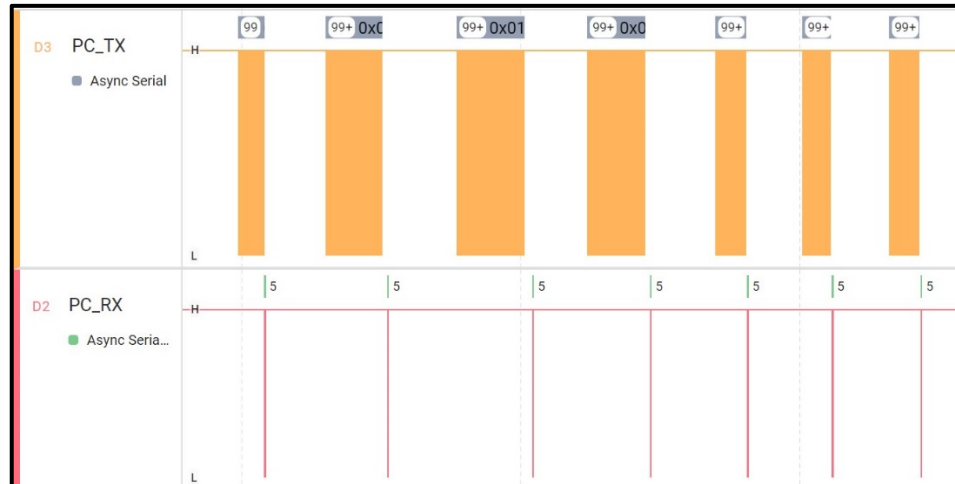


Figure 9 - Frames

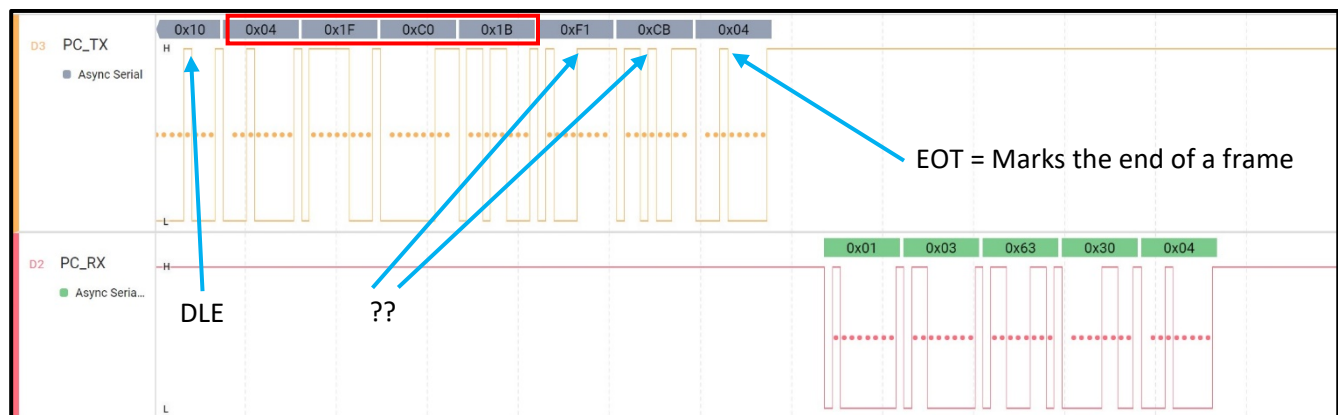


Figure 10 - Mesure 3

Sur cette dernière capture, nous pouvons apercevoir les dernières bytes de data du code HEX suivi de 2 bytes pour l'instant inconnu suivi eux par un caractère de contrôle indiquant au PIC qu'il s'agit une fin de frame. Le PIC répond à chaque fois le même message.

4 Algorithme

J'ai tout de même débuté l'élaboration d'un algorithme, permettant de tester le système avant de le mettre en place proprement. Je l'ai créé en Python avec l'aide de mon camarade Ali Zoubir.

Actuellement, ce code permet d'ouvrir un fichier HEX, de trouver les données qui peuvent être comprise comme des commandes et ajouter un DLE en devant.

```
1  import re
2
3  # Ouvre fichier
4  with open ("TE_ThermoLm92Uart_withBootloader.hex", "r") as file_to_convert:
5      data = file_to_convert.read().splitlines()
6      # Transforme la liste en un seul string
7      data = ''.join(str(e) for e in data)
8      # Supprime tous les ':' du string
9      data = data.replace(':', '')
10     # Créer une liste de paire de char à partir du string
11     data = re.findall('..', data)
12     #Initialise string de sortie
13     converted = ""
14     # Teste toutes les paires de char du string
15     for pair in data:
16         if(pair == '01'):
17             pair = '1001'
18         elif(pair == '04'):
19             pair = '1004'
20         elif (pair == '10'):
21             pair = '1010'
22         # Ajoute la paire transformée au string final
23         converted += pair
24
25     # Créer fichier en écriture
26     file_converted = open ("TE_ThermoLm92Uart_withBootloader_converted.hex", "w")
27     # Ecris le string dans le fichier
28     file_converted.write(converted)
```

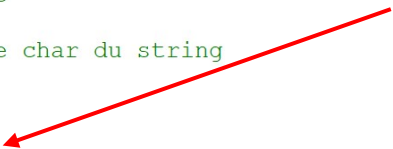


Figure 11 - Algorithme

5 Conclusion

Pour conclure, j'ai effectué diverses recherches afin de trouver des informations pertinentes au niveau du bootloader et du protocole de communication utilisé afin de démarrer correctement le logiciel PC. J'ai également effectué diverses mesures pour confirmer la fiabilité des informations trouvées.

J'ai compris une bonne partie de tout cela mais au niveau du protocole, je n'ai pas entièrement saisi toute la manière de communiquer avec le PIC. Le logiciel de Microchip génère des pauses en envoyant 3 bytes dont 2 sont à déterminer et le dernier est un byte de contrôle de fin de frame.

Malheureusement, en raison de mes nombreux autres travaux, je n'ai pas pu consacrer suffisamment de temps à ce projet sur lequel j'aurais aimé m'investir davantage.

Lausanne, le 2 juin 2023

Meven Ricchieri



6 Sources

- <https://ww1.microchip.com/downloads/en/Appnotes/01388B.pdf>
- <https://microchipdeveloper.com/ipe:sqtp-hex-file-format>
- <https://ww1.microchip.com/downloads/en/DeviceDoc/40001779B.pdf>

7 Annexes

7.1 Journal de travail

Journal de travail	
Février	Prise de connaissance du projet
Mars	Lecture du projet précédent
Avril	Recherche d'information sur le bootloader et test du logiciel fourni par Microchip
Mai	Recherche d'information sur le protocole de communication et mesures sur le logiciel de Microchip
Juin	Mesures des trames et rédaction du rapport

Projet ETML-ES – Modification

Note: Les textes explicatifs en italique peuvent être supprimés

A remplir par l'initiateur

PROJET:	2203 Bootloader		
Entreprise/Client:	-	Département:	SLO
Demandé par (Prénom, Nom):	PBY	Date:	-
Objet (No ou réf, pièce, PCB...)	2203		
Version à modifier:	v1		

A remplir par l'exécutant

Auteur (ETML-ES):	Meven Ricchieri	Filière:	SLO
Nouvelle version:	V1B	Date:	05.05.2022

1 Référence conception

Disponibles dans le dossier du projet :

K:\ES\PROJETS\SLO\2203_BootloaderPic32\doc

2 Détail des modifications

#	Description	Fait	Approuvé
1	Suivre la documentation pour créer un nouveau projet compatible Bootloader	BGR	
2	Créer la partie graphique d'une application C# capable de charger 3 apps dans le Bootloader	BGR	
3	Coder l'application C# pour charger 3 apps dans le Bootloader		
4	Recherche d'information sur le bootloader et le protocole de communication	MRI	
5	Réalisation d'un algorithme permettant de modifier un fichier HEX		
6			
7			
8			

3 Remarques

La suite consiste à reprendre l'application créée puis la coder pour permettre de charger des applications dans le Bootloader et d'en afficher la version.

La partie graphique est inspirée de l'application fournie par microchip (disponible en annexe).

4 Convention de nommage et liens

Le nom de ce fichier doit être unique et doit donc contenir le numéro du projet et un numéro consécutif de modification avec le format suivant :

aaii_MOD_nn.docx

ou

NomProjet_MOD_nn.docx

avec :

- MOD : pour modification
- aaii : numéro de projet, exemple 1708 pour projet de 2017 no 08
- NomProjet : Si le projet n'est pas numéroté ou mandat de client.
- nn : numéro de modification. La première est 01

Exemples :

- **1708_MOD_01.docx** 1ere modification pour le projet 1708
- **1708_MOD_02.docx** 2e modification pour le projet 1708
- **CapteurVolets_MOD_01.docx** Cas de projet externe

Le schéma et/ou les documents de production de la pièce ou du PCB se référeront à ce document dans les cartouches.

Si un nouveau projet reprend un design d'un autre projet, créer un document de **modification numéro 00**. Ainsi, on pourra décrire les modifications initiales dans le fichier.

Exemple :

- **1803_MOD_00.docx** Modification initiale pour le nouveau projet 1803 à partir d'un autre projet (par ex. 1708)

4.1 Stockage du fichier

Ce fichier sera stocké à la racine du dossier **/doc** d'un projet.

Ainsi, tous les fichiers de modifications des pièces ou PCBs faisant partie du projet sont centralisés dans le même répertoire. La numérotation devient implicite.