Librairie I2C

Support :

- PIC24FJ64GA006 (PICTAVE)
- PIC18F44K22
- PIC18F45K22
- PIC18F46K22

Mode :

- Master (mode bloquant avec ‘while’ et non-bloquant)
- Slave (pas implémenté pour le moment)

Fonctionnalités :

- Envoi sur le bus I2C ;
- Réception depuis le bus I2C ;
- Lecture de registres ;
- Scan des esclaves connectés. (pas implémenté pour le moment)

Version	Modification du doc	Auteur	Date
1.1	Création du document	EPEU	08/05/20

Sommaire

TABLE DES MATIERES

1. INITIALISATION

Il Faut impérativement commencer par initialiser la librairie I2C avec la fonction **int I2C_Init(u8 Options)**. Elle permet d'initialiser le bus I2C, les entrées/sorties utilisées, et les registres de configurations. La fonction initialise un timer à 1ms afin de marquer des temps entre 2 actions si nécessaire. Si le timer n'arrive pas à s'initialiser, la fonction retourne **PIC_FAILED**.

La fonction prend un paramètre **Options**. Celui-ci attend :

- **MASTER ;**
- **SLAVE ;**
- **INTERRUPT.**

Il faut au moins définir Master ou Slave en option. L'interruption n'est pas obligatoire mais fortement conseillée.

Pour définir les options voici un exemple :

I2C_init(MASTER | INTERRUPT) ;

Sur cet exemple, le bus a été initialisé en mode maitre avec des fonctionnalités non-bloquantes.

2. ENVOI SUR LE BUS I2C

Pour envoyer des données sur le bus I2C, il faut utiliser la fonction :

int I2C_Write(u8 adresse, u8 *data, u8 sizeData, u8 delay)

Cette fonction prend en paramètres :

- **adresse** : l'adresse de l'esclave vers lequel on envoi la donnée.
- **data** : la donnée à envoyer sous forme d'un tableau.
- **sizeData** : la taille du tableau à envoyer.
- **delay** : le temps à attendre entre chaque octet envoyé. Mettre à 0 si ce n'est pas nécessaire d'attendre. Cette fonctionnalité est utilisée pour la communication avec les mémoires EEPROM I2C par exemple.

En mode bloquant, cette fonction peut partir en boucle infinie dans le cas où aucun esclave n'est connecté sur le bus I2C.

En mode non-bloquant, la fonction réalise une copie de votre tableau afin que vous puissiez réutiliser votre tableau data sans crainte après l'appel de cette fonction. (Plus de détails après)

La fonction retourne **PIC_FAILED** dans les cas où vous êtes configurés en mode esclave i2C, ou alors qu'il n'y a plus assez d'espace pour stocker la copie du message à envoyer. Dans ce cas l'utilisation de la fonction **void I2C_Free(void)** est recommandé afin de libérer de l'espace. La fonction peut retourner **I2C_OVERFLOW** s'il y a trop de requêtes. Il faut alors attendre d'avoir assez d'espace. Si la fonction ne retourne pas d'erreur, alors elle retourne un identifiant unique qui vous permettra de connaitre l'état de votre message grâce à la fonction **int I2C_Get_Status(u8 MsgID)**.

Exemple d'utilisation de la fonction :

```
int Status = I2C_Write(0xA0, « test\0 », 5, 0) ;

if((Status !=PIC_FAILED) && (Status= !I2C_OVERFLOW))

        u8 MsgID = (u8)Status ;
```

L'exemple envoie un message « test » de 5 octets avec sans délai à l'esclave qui se trouve à l'adresse hexadécimale « A0 ». Si la fonction ne retourne pas **PIC_FAILED** ou **I2C_OVERFLOW**, alors il stocke l'identifiant unique.

2.1. PICTAVE

L'utilisation de la fonction d'envoi de données sur le bus réalise une copie de la donnée. Pour cela, la librairie utilise la fonction **malloc** qui permet la création de tableau de manière dynamique : la taille du tableau n'est pas définie de base dans le programme, elle peut varier grâce à des variables.

Pour utiliser cette fonction, il faut définir une taille de « tas » (**heap** en anglais) car celle-ci est définie de bas à 0. Pour définir cette taille, il faut se rendre dans l'onglet dashboard de votre projet, puis cliquer sur la clé à molette. (**FIGURE 2-1**)

Une fois dans les propriétés du projet, sélectionner **xc16-ld** et dans **General** remplacer la valeur de **Heap size** par la valeur de votre choix. (**FIGURE 2-2**)

Attention, cette valeur n'est pas infinie et ne dois pas être trop élevé pour laisser de la place à la **pile**. Dimensionnez cette valeur à la taille nécessaire pour votre projet.

FIGURE 2-1

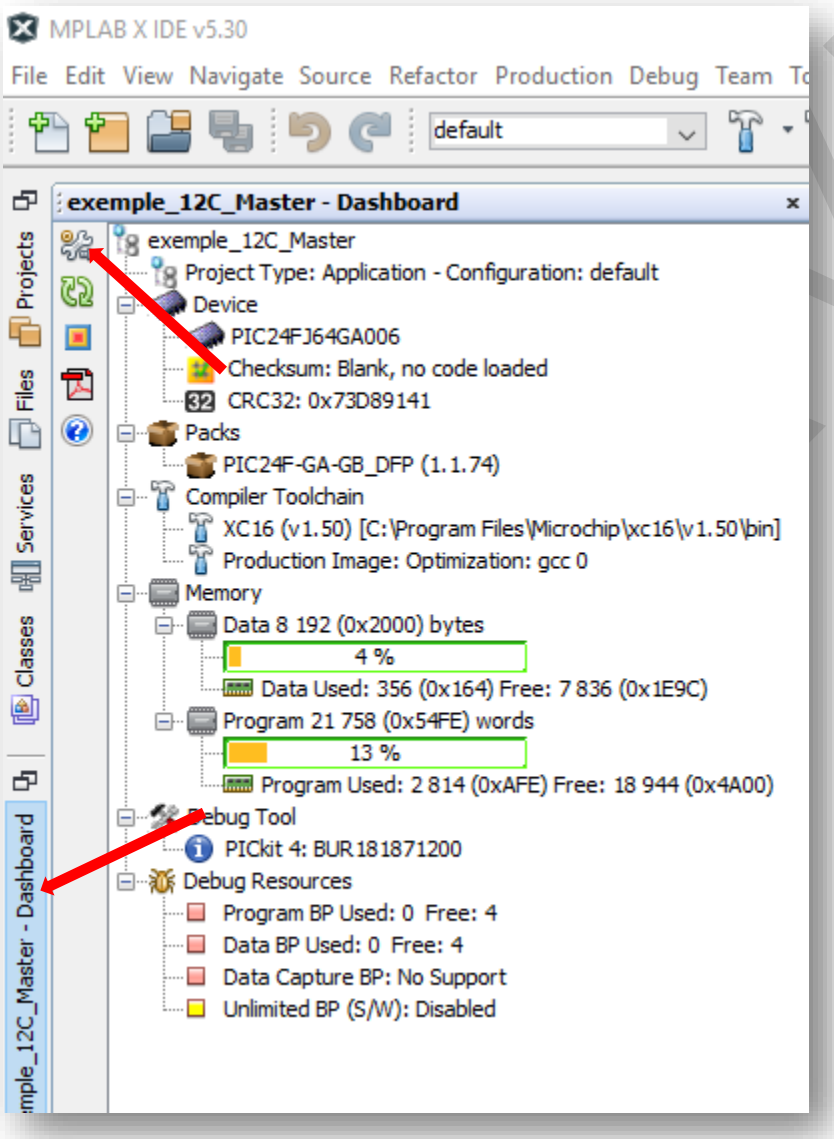
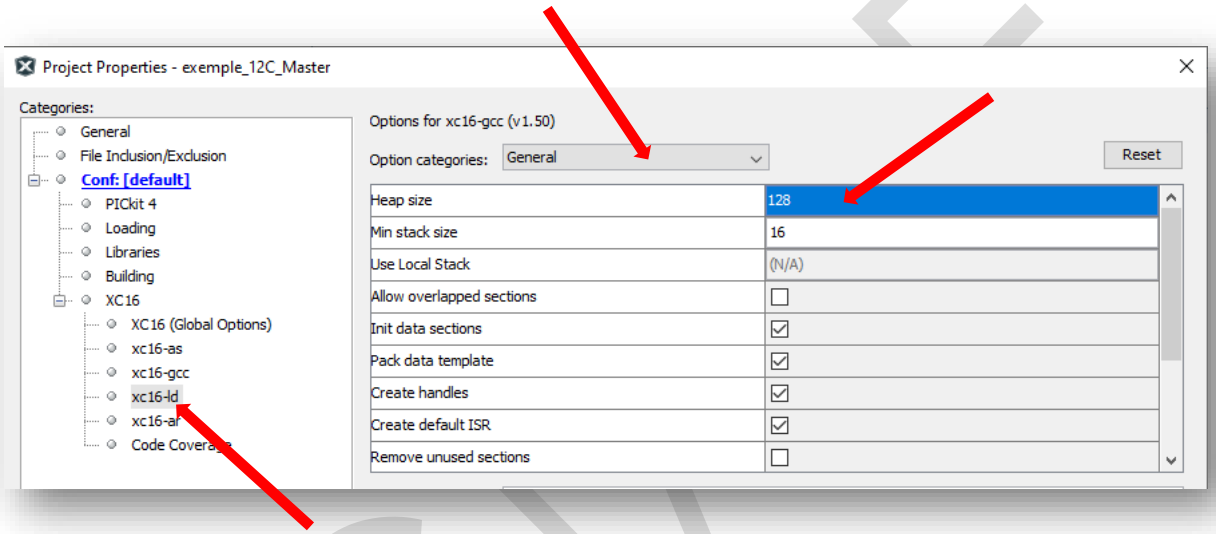

MPLAB X IDE v5.30
File Edit View Navigate Source Refactor Production Debug Team To
default
exemple_12C_Master - Dashboard
exemple_12C_Master
Project Type: Application - Configuration: default
Device
PIC24FJ64GA006
Checksum: Blank, no code loaded
CRC32: 0x73D89141
Packs
PIC24F-GA-GB_DFP (1.1.74)
Compiler Toolchain
XC16 (v1.50) [C:\Program Files\Microchip\xc16\v1.50\bin]
Production Image: Optimization: gcc 0
Memory
Data 8 192 (0x2000) bytes
4 %
Data Used: 356 (0x164) Free: 7 836 (0x1E9C)
Program 21 758 (0x54FE) words
13 %
Program Used: 2 814 (0xAFE) Free: 18 944 (0x4A00)
Debug Tool
PICkit 4: BUR181871200
Debug Resources
Program BP Used: 0 Free: 4
Data BP Used: 0 Free: 4
Data Capture BP: No Support
Unlimited BP (S/W): Disabled
Projects
Files
Services
Classes
mple_12C_Master - Dashboard

FIGURE 2-2



2.2. PIC18F4XK22

L'utilisation de la fonction d'envoi de données sur le bus réalise une copie de la donnée. Le PIC18 est un microcontrôleur 8 bit qui utilise le compilateur XC8 contrairement à la carte Pictave qui utilise le compilateur XC16. Les mallocs sont ne sont pas compatible avec le compilateur XC8 donc on ne peut pas utiliser la même méthode qu'avec la Pictave. Pour remplacer le malloc, la fonction utilise des fonctions qui se trouvent dans le fichier **memory**. Cette fonction va stocker les données dans un tableau dont on peut régler la taille. Pour cela il faut changer la valeur de la constante : **MALLOC_SIZE** qui se trouve dans le fichier **memory.h**. Cette valeur doit également être adaptés à votre projet.

3. RECEVOIR DEPUIS LE BUS I2C