STI 2D SIN

STI2D - Système d'Information et Numérique

TD TP Cours Synthèse Devoir Evaluation Projet Document ressource

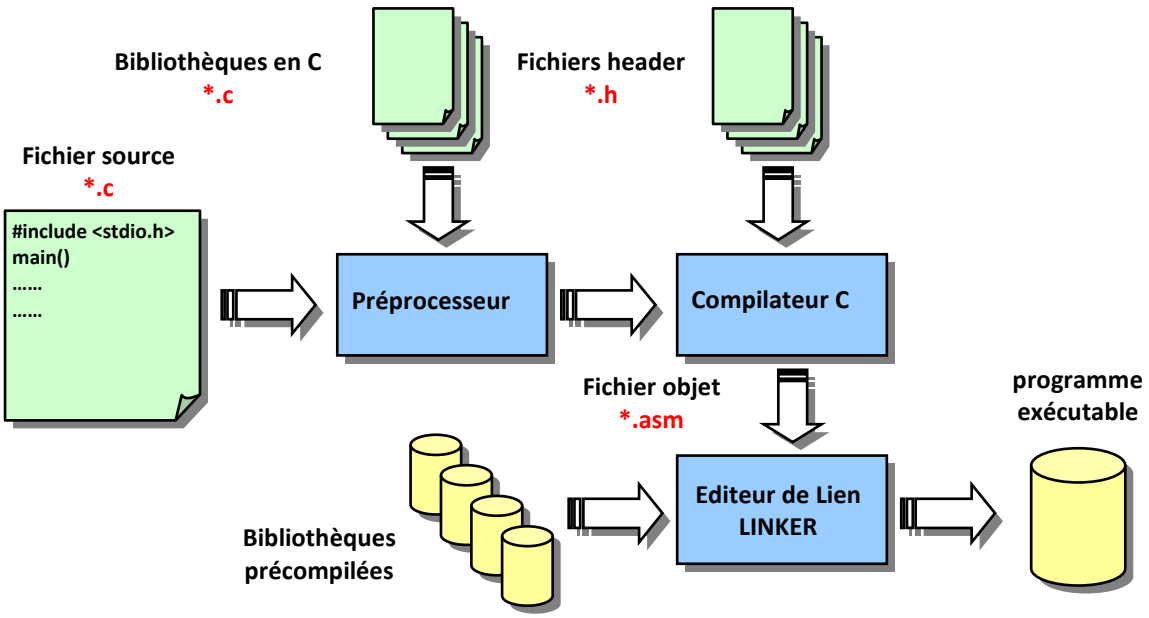
PROGRAMMATION DES MICROCONTROLEURS PIC EN LANGAGE C

1. PRÉSENTATION

Le **langage C** est un **langage structuré**. Créé en **1972** par Denis Ritchie pour le développement du système d'exploitation **UNIX**. Il a fait son entrée, depuis quelques années, dans le monde des microcontrôleurs. Ses avantages pour la programmation des **µC** sont les suivants :

- **La portabilité** : Un programme développé en C est indépendant du microcontrôleur utilisé.
- **Une grande bibliothèque de fonctions** : Le langage C dispose d'un grand nombre de fonctions mathématiques, de gestion de fichiers ou d'entrées/sorties et permet d'éviter les tâches d'écritures pénibles en langage assembleur.
- **Proche du matériel** : Le langage C est très proche de la machine en permettant d'accéder aux adresses des variables. Un programme en C peut contenir des séquences en assembleur.

2. DÉVELOPPEMENT D'UN PROGRAMME EN LANGAGE C



Les étapes qui interviennent pour obtenir le code objet téléchargé dans le microcontrôleur PIC sont :

- **Edition** du fichier source (**programme.c**) avec un éditeur de texte.
- **Interprètation** des directives C contenues dans le fichier source par le **pré-processeur.**
- **Compilation** du fichier source pour obtenir un fichier objet (**programme.asm**). La compilation est la transformation des instructions C en instructions assembleur pour microcontrôleurs PIC.
- **Edition de liens** permettant d'intégrer des fonctions prédéfinies. Le programme auxiliaire éditeur de liens (**Linker**) réunit les différents fichiers objets, affecte des adresses physiques aux variables et génère un fichier exécutable (**programme.hex**) compatible avec le PIC.

3. STRUCTURE D'UN PROGRAMME EN C

La saisie d'un programme en "C" répond pratiquement toujours à la même architecture :

```
//*****************************************************************************
//*                          Programme de mise en oeuvre du MOWAY               *
//*****************************************************************************

#include "lib_rf2gh4.h"                     // Librairie de rf
#include "lib_cam_moway.h"          // Librairie caméra
#include "lib_mot_moway.h"          // Librairie commande moteurs
#include "lib_sen_moway.h"          // Librairie capteurs
#include "i2c_problematiques.h"     // Librairie liaison I2C

//*****************************************************
//* Attente d'un certain nombre de secondes : 0 - 255
//*****************************************************

void PAUSE_SECONDE(unsigned char nb_secondes)
{
        unsigned char i;
        for (i=0;i<nb_secondes;i++)
        {
                Delay10KTCYx (100);           //pause de 1 seconde
        }
}

//*****************************************************************************
//*                          Programme Principal                              *
//*****************************************************************************

void main()                             // Début
{
        SEN_CONFIG();
        MOT_CONFIG();
        while(1)                        // Début d'itération Répéter
        {
                LED_BRAKE_OFF();        // Eteindre la DEL de frein
                LED_TOP_RED_ON();       // Allumer la DEL Rouge
                PAUSE_SECONDE(1);       // Pause de 1s
                LED_TOP_RED_OFF();      // Eteindre la DEL Rouge
                LED_TOP_GREEN_ON();     // Allumer la DEL Verte
                PAUSE_SECONDE(1);       // Pause de 1s
                LED_TOP_GREEN_OFF();    // Eteindre la DEL Verte
                LED_FRONT_ON();         // Allumer la DEL Blanche
                PAUSE_SECONDE(1);       // Pause de 1s
                LED_FRONT_OFF();        // Eteindre la DEL Blanche
                LED_BRAKE_ON();         // Allumer la DEL de frein
                PAUSE_SECONDE(1);       // Pause de 1s
        }                               // Fin Boucle Répéter
}                                       // Fin de la fonction "main"
```

Les principales règles de syntaxes sont :

- Le **programme principal** est indiqué par le terme **main(…)**.
- Le programme principal est constitué de **fonctions définies par le programmeur** (déclarées avant le main) ou bien de fonctions **prédéfinies dans les bibliothèques**.
- Les **commentaires** sont placés entre les symboles **/*……*/** ou bien à droite des symboles **//**.
- Toutes les instructions ou actions se terminent par un point virgule **;**.
- Une fonction ou bloc d'instructions commence par **{** et se termine par **}**.
- Attention à la **casse** car le langage C fait la différence entre minuscules et majuscules.

4. VARIABLES ET CONSTANTES

4.1. Différents types

Avant d'utiliser une **variable** ou une **constante**, il faut la déclarer afin d'informer le compilateur de son existence et ainsi lui réservé un espace mémoire suffisant. Leur nom que l'on utilise est un **identificateur**. Leur écriture doit : Utiliser les lettres de **a** à **z**, et de **A** à **Z**, les chiffres de **0** à **9** (hors 1er caractère) ou **_**. Ne contenir ni espace, ni caractère accentué. Elle doit être **représentative du rôle** dans le programme.

Toutes les constantes ou variables utilisées en C sont classées selon des types. Un **type décide de l'occupation mémoire** de la donnée. Pour les déclarer correctement, il faut savoir ce qu'elles vont contenir. On distingue les types suivants :

char a;	Entier signé **8 bits** [**-128** à **+127**]
unsigned char b ;	Entier non signé **8 bits** [**0** à **255**]
int c ;	Entier signé **16 bits** [**-32768** à **+32767**]
unsigned int d ;	Entier non signé **16 bits** [**0** à **65535**]
long e ;	Entier signé **32 bits**[**-2147483648** à **+2147483647**]
unsigned long f ;	Entier non signé **32 bits** [**0** à **4292967295**]
float g ;	Réel signé **32 bits** de valeur absolue comprise entre $\mathbf{3,4.10^{-38}}$ et $\mathbf{3,4.10^{+38}}$
double h ;	Réel signé **64 bits** de valeur absolue comprise entre $\mathbf{1,7.10^{-308}}$ et $\mathbf{1,7.10^{+308}}$
long double i ;	Réel signé **80 bits** de valeur absolue comprise entre $\mathbf{3,4.10^{-4932}}$ et $\mathbf{3,4.10^{+4932}}$

La représentation des différentes bases de numération et du code ASCII sont :

int a = 4 ;	Un nombre seul représente un nombre décimal.
int b = 0b1010 ;	Un nombre précédé de **0b** est un nombre binaire
int p = 0x00FF ;	Un nombre précédé de **0x** est un nombre hexadécimal
char c = 'x'	Un caractère entre **' '** représente son code ASCII

4.2. Les variables globales et locales

Une **variable globale** est **déclarée en en-tête du programme**. Elle est valide pendant toute la durée d'exécution du programme car elle fait l'objet d'une **réservation mémoire permanente** en RAM. Elle peut être utilisée et modifiée par toutes les fonctions du programme y compris le main.

Une **variable locale** est déclarée **à l'intérieur d'une fonction**. Son **existence est limitée à la durée d'exécution de cette fonction**. Elle est donc ignorée par les autres fonctions ou du main. Elle peut (bien que cette façon de procéder soit déconseillée), porter le même nom qu'une variable globale ou qu'une autre variable locale se trouvant dans une autre fonction.

5. LES FONCTIONS

Un programme en C est un ensemble de fonctions :

- La **fonction principale main( )** qui est la première fonction appelée lors de l'exécution du programme.
- Les **fonctions écrites** par le programmeur qui doivent être déclarées avant leur appel.
- Les **fonctions prédéfinies** issues des bibliothèques (sous forme de fichiers comportant l'extension **.h**) du compilateur (dont le code n'est pas écrit par le programmeur mais inséré dans le programme par l'éditeur de liens). Ces fonctions seront ajoutées au programme exécutable lors de **l'édition de liens**. Pour incorporer dans un programme un fichier .h, on utilise la commande `#include <fichier.h>` placée habituellement en début de fichier.

La syntaxe d'écriture d'une fonction est :

```
<type de la valeur de retour> nom_fonction(<type des paramètres>,<paramètres>)
{
        définition des variables locales ;
        instructions ;
}
```

Remarques :

- La liste des paramètres reçus peut-être **vide**.
- La valeur de retour peut être de n'importe quel type : **int**, **float**… ou **void** si la fonction ne retourne pas de valeur,
- Une variable définie dans une fonction n'existe que dans celle-ci (variable locale).

Exemple :

```
#include <pic16f6x.h>     //Fichier de définition des adresses des registres des 16F6x
#include <delay.h>                // fonctions prédéfinies pour temporisations logicielles

//Directives d'assemblage
__CONFIG(MCLRDIS & LVPDIS & PWRTEN & BORDIS & \      //Le \ permet le passage à la ligne
UNPROTECT & DATUNPROT & WDTDIS & XT);

//Définitions des fonctions écrite par le programmeur
void PORTB_en_sortie()
{
        TRISB=0;                        //PORTB en sortie
}
void Allumer_LED_PORTB()
{
        PORTB=0xFF;                     //RB0 à RB7 mis à 1
}
void Eteindre_LED_PORTB()
{
        PORTB=0x00;                     //RB0 à RB7 mis à 0
}

void main()                             //Fonction principale
{
        PORTB_en_sortie();              //Appel fonction écrite par le programmeur
        while(VRAI)                     //Répéter toujours
        {
                Allumer_LED_PORTB();    //Appel fonction écrite par le programmeur
                DelayMs(250);           //Appel fonction prédéfinie
                Eteindre_LED_PORTB();   //Appel fonction écrite par le programmeur
                DelayMs(250);           //Appel fonction prédéfinie
        }
}
```

```
void Allumer_LED_PORTB()
{
        PORTB=0xFF;          //RB0 à RB7 mis à 1
}
```

La liste des paramètres reçus est vide

La fonction ne retourne pas de valeur

La fonction reçoit 2 paramètres de type entier

```
int Ma_fonction_somme(int a, int b)
{
        int resultat;           //Déclaration d'une variable locale appelée resultat
        resultat = a+b;         // a et b ont été déclarées et initialisées lors de l'appel de la fonction
        return resultat;        // return permet de retourner une valeur à la fonction appelante
}
```

La fonction retourne une valeur de type entier

6. LES OPÉRATEURS

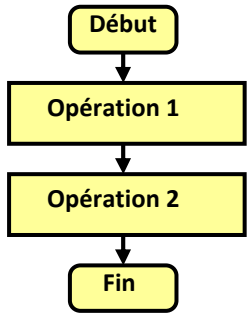
	Opérateur	Fonction	Equivalence
Opérateurs arithmétiques	**+**	Addition	**c = a + b ;**
	–	Soustraction	**c = a – b ;**
	*****	Multiplication	**c = a * b ;**
	/	Division entière	**c = a / b ;**
	%	Reste de la division entière	**c = a % b ;**
Affectations	**=**	Affectation ordinaire	**a = b ;**
	+=	Additionner de	**a + = b ⇒ a =a + b ;**
	–=	Soustraire de	**a – = b ⇒ a =a – b ;**
	***=**	Multiplier par	**a * = b ⇒ a =a * b ;**
	/=	Diviser par	**a / = b ⇒ a =a / b ;**
	%=	Reste	**a % = b ⇒ a = a % b ;**
	– –	Soustraire de 1 (Décrémentation)	**a – – ⇒ a = a – 1 ;**
	++	Additionner 1 (Incrémentation)	**a ++ ⇒ a = a + 1**
Opérateurs binaires	**&**	ET	**c = a & b**
	|	OU	**c = a | b**
	^	OU exclusif	**c = a ^ b**
	~	NON	**b = ~ a**
	>>	Décalage à droite des bits	**c = a << b** (a décalé b fois à droite)
	<<	Décalage à gauche des bits	**c = a >> b** (a décalé b fois à gauche)
Tests	**&&**	ET logique	**if (a && b)**
	||	OU Logique	**if (a || b)**
	==	Egal à	**if (a == b)**
	!=	Différente de	**if (a != b)**
	>	Supérieur à	**if (a > b)**
	>	Inférieur à	**if (a < b)**
	>=	Supérieur ou égal à	**if (a >= b)**
	<=	Inférieur ou égal à	**if (a <= b)**

7. STRUCTURES ALGORITHMIQUES FONDAMENTALES

7.1. Structure linéaire ou séquence

Cette structure se caractérise par une **suite d'actions à exécuter successivement**.

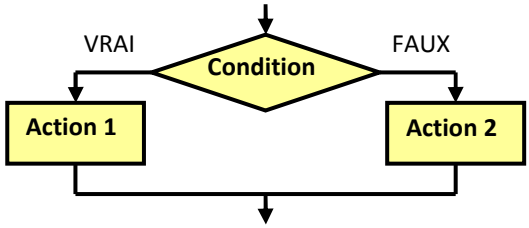
Organigramme	Algorithme	Exemple en langage C

	**Début**
	Faire Opération 1
	Faire Opération 1
	Fin
		`void main()`
		`{`
		`Operation1();`
		`Operation2();`
		`}`

7.2. Structure alternative ou sélection

Cette structure n'offre que deux issues possibles s'excluant mutuellement. Elle définit une **fonction de choix** ou de **sélection** entre l'exécution de l'un ou de l'autre des deux traitements. Également désignées par **structures conditionnelles**, elles sont représentatives de **saut** ou rupture de séquence.

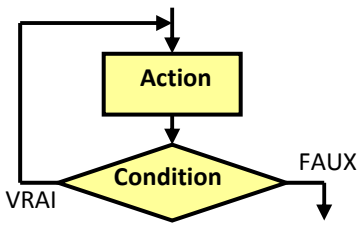
Organigramme	Algorithme	Exemple en langage C

	**Si** condition **vraie**
	Alors Faire Action 1
	Sinon Faire Action 2
	Fin Si
		`if (Condition = = VRAI)`
		`{`
		`Action1();`
		`}`
		`else`
		`{`
		`Action2();`
		`}`

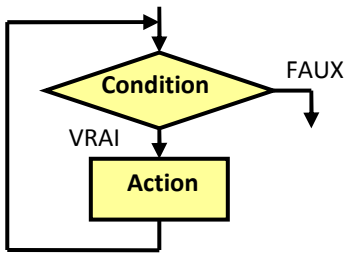
7.3. Structure répétitive ou itérative

Cette structure **répète l'exécution** d'un traitement.

Organigramme	Algorithme	Exemple en langage C

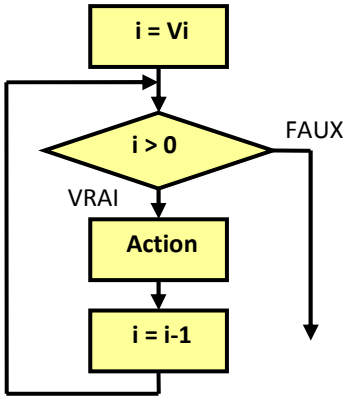
	**Faire**
	Action
	Tant que Condition **vraie**
		`do`
		`{`
		`Action();`
		`}`
		`while (Condition==VRAI);`

Organigramme	Algorithme	Exemple en langage C

	**Tant que** Condition **vraie**
	Faire Action
	Fin Tant que
		`while (Condition==VRAI);`
		`{`
		`Action();`
		`}`

Dans cette structure la sortie de la boucle d'itération s'effectue lorsque le **nombre souhaité de répétition est atteint**. D'où l'emploi d'une variable de boucle (indice **i**) caractérisée par : Sa valeur initiale (**Vi**), sa valeur finale et son pas de variation.

Organigramme	Algorithme	Exemple en langage C

	**Pour** i = Vi à i = 0 par pas de 1
	Faire Action
	Fin Pour
		`for (i=Vi;i>0;i--)`
		`{`
		`Action();`
		`}`
		
		`i=Vi`
		`while (i>0);`
		`{`
		`Action();`
		`i=i-1;`
		`}`