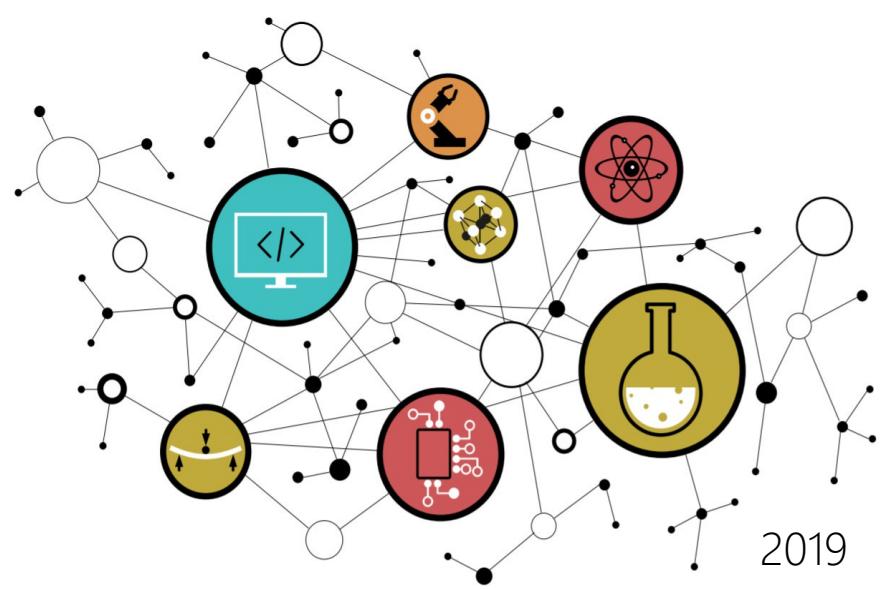


ANNEXES



2019

SOMMAIRE

1. CREATION DE PROJET SOUS MPLABX

2. CREATION D'UNE BIBLIOTHEQUE STATIQUE SOUS MPLABX

3. ENVIRONNEMENT GRAPHIQUE DE MPLABX

4. CHAINE DE COMPILEMENT XC8

5. MCU PIC18F27K40

6. ASSEMBLEUR PIC18

7. GLOSSAIRE SYSTEMES EMBARQUES

8. REGLES DE CODAGE C

1. CREATION DE PROJET SOUS MPLABX

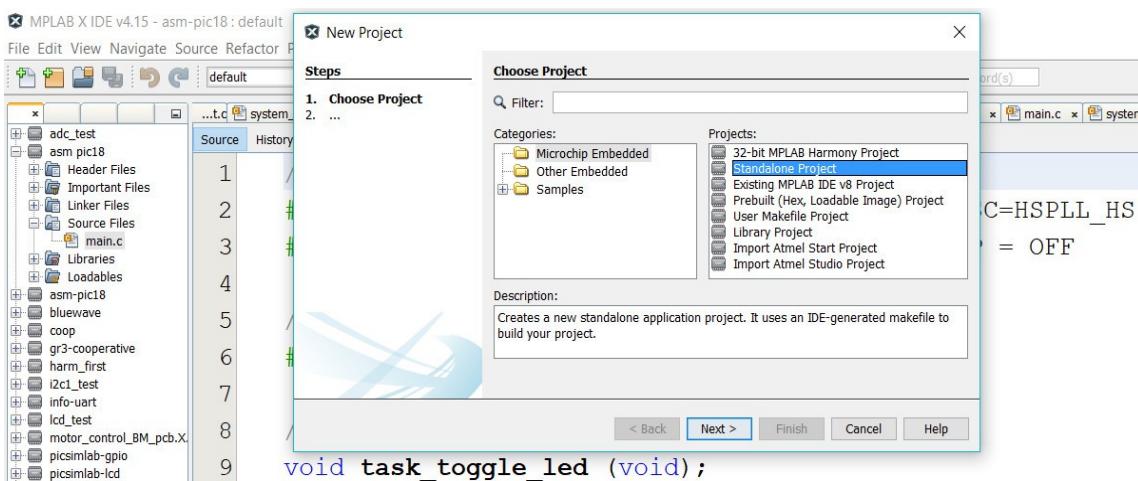
1. CREATION DE PROJET SOUS MPLABX



Il est à noter que la totalité des développements pourrait être réalisée sans environnement de développement ou IDE (Integrated Development Environment). Nous utiliserions alors la console, UNIX ou MS-DOS, pour invoquer et paramétriser la chaîne de compilation XC8 (toolchain 8bits Microchip). De même, l'édition des fichiers sources se ferait depuis un éditeur de texte, par exemple vi, vim, nano, geany, etc sous GNU\Linux ou Notepad++ sous Windows. Néanmoins, durant les phases de développement d'application offrant des arborescences de fichiers conséquentes, un IDE peut s'avérer pratique. Un IDE est dédié à centraliser et lier dans un même outil tous les services nécessaires à l'ingénieur développeur (gestion des outils de compilation, éditeur de texte, interface de debug, etc). Malgré la grande richesse d'outils et d'options graphiques proposés, l'un des principaux travaux de sortie de l'IDE reste de générer le Makefile du projet, de configurer les outils de compilation, puis d'appeler la commande make afin de générer le firmware de sortie, tout en proposant une représentation graphique logique de projets en développement.

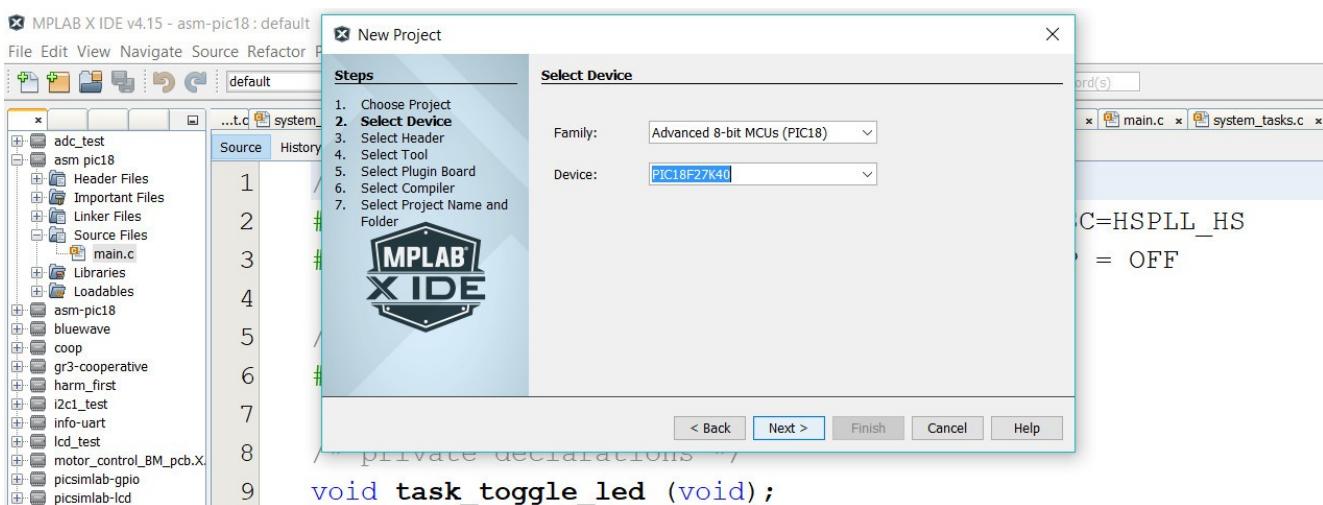
Sélection du type de projet

- Télécharger l'archive *mcu.zip* sur la plateforme moodle, et démarrer ce premier tutoriel !
- Ouvrir MPLABX → File → New Project...
- Sélectionner le type de projet souhaité : *Standalone (développement d'application)*, *Prebuilt (charger un firmware précompilé)*, *Library (développement bibliothèque statique)*, etc
- *Microchip Embedded* → *Standalone Project* → *Next*



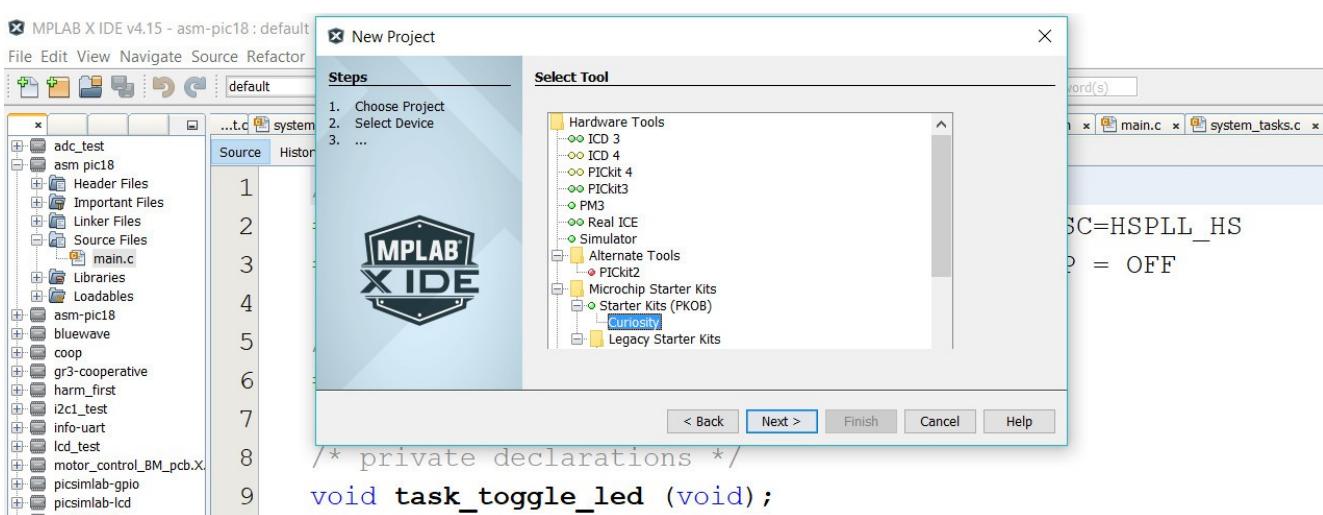
Sélection du processeur cible

- L'étape qui suit est à adapter en fonction du MCU ciblé par les développements. Attention, le nom du MCU doit être impérativement le bon, sous peine de voir apparaître des erreurs au chargement de l'exécutables par la sonde de programmation. Néanmoins, toutes les étapes présentées dans ce tutoriel sont modifiables par la suite après création du projet.
- *Family* → *Advanced 8-bits MCUs (PIC18)*
- *Device* → *PIC18F27K40* → *Next*



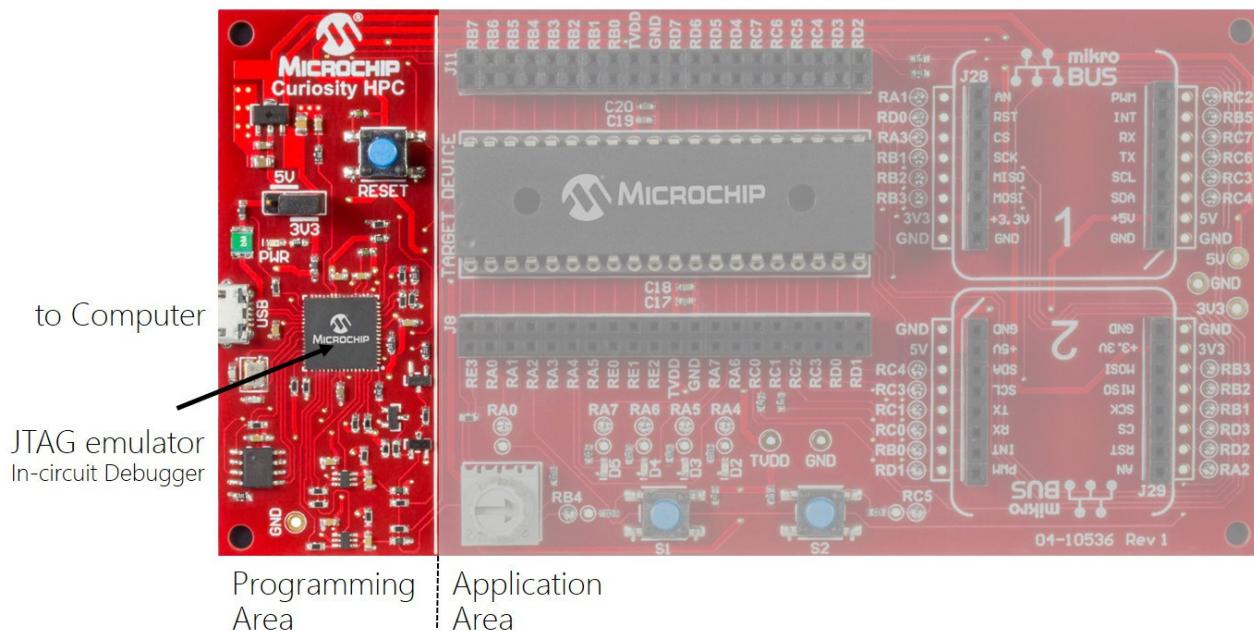
Sélection de la sonde de programmation ou du starter kit

- Sélectionner l'outil de chargement du fichier exécutable de sortie (ELF, HEX, COFF, etc) vers le processeur cible. Nous parlons de sonde JTAG (Join Test Action Group, norme IEEE 1149.1), ou sonde de programmation, ou sonde d'émulation, etc.
- *Starter Kits (PKOB)* → *Curiosity* → *Next*



Annexes

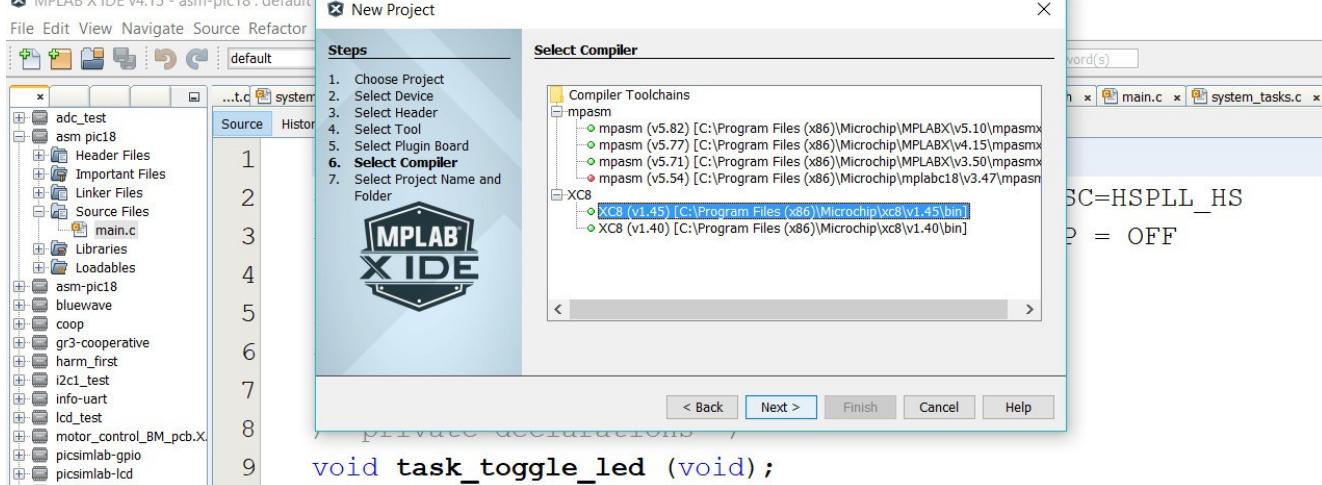
- Depuis sa normalisation en 1990, le JTAG est devenu le plus grand standard dans l'industrie pour la programmation et le test de processeur. Observons la sonde de programmation JTAG présente sur la carte de démarrage ou starter kit Curiosity HPC de Microchip



Sélection de la chaîne de compilation

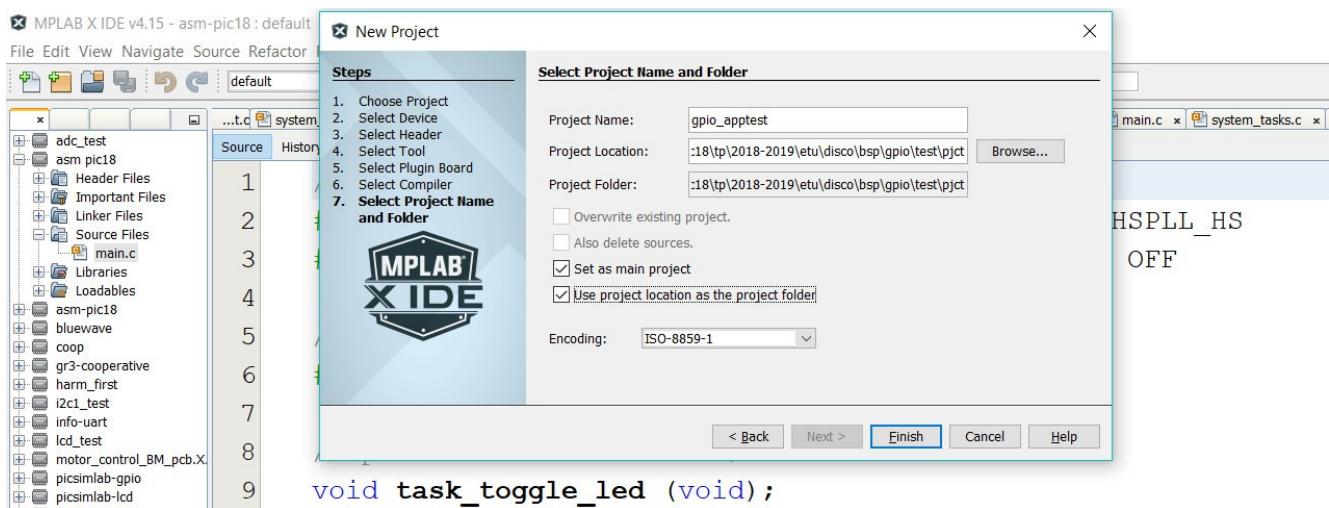
- Sélectionner la chaîne de compilation. Sur une même machine de développement, un certain nombre de toolchain peuvent être installées. De même, pour une même chaîne de compilation dédiée à une famille de CPU, plusieurs versions peuvent également être installées. L'exemple ci-dessous montre que l'IDE MPLABX a reconnu plusieurs toolchain C pour PIC18. Dans le cas présent, 2 versions de XC8, la chaîne de compilation développée par Microchip pour leur famille de CPU PIC18.

- XC8 (v1.45) → Next*



Sélection de l'emplacement du projet

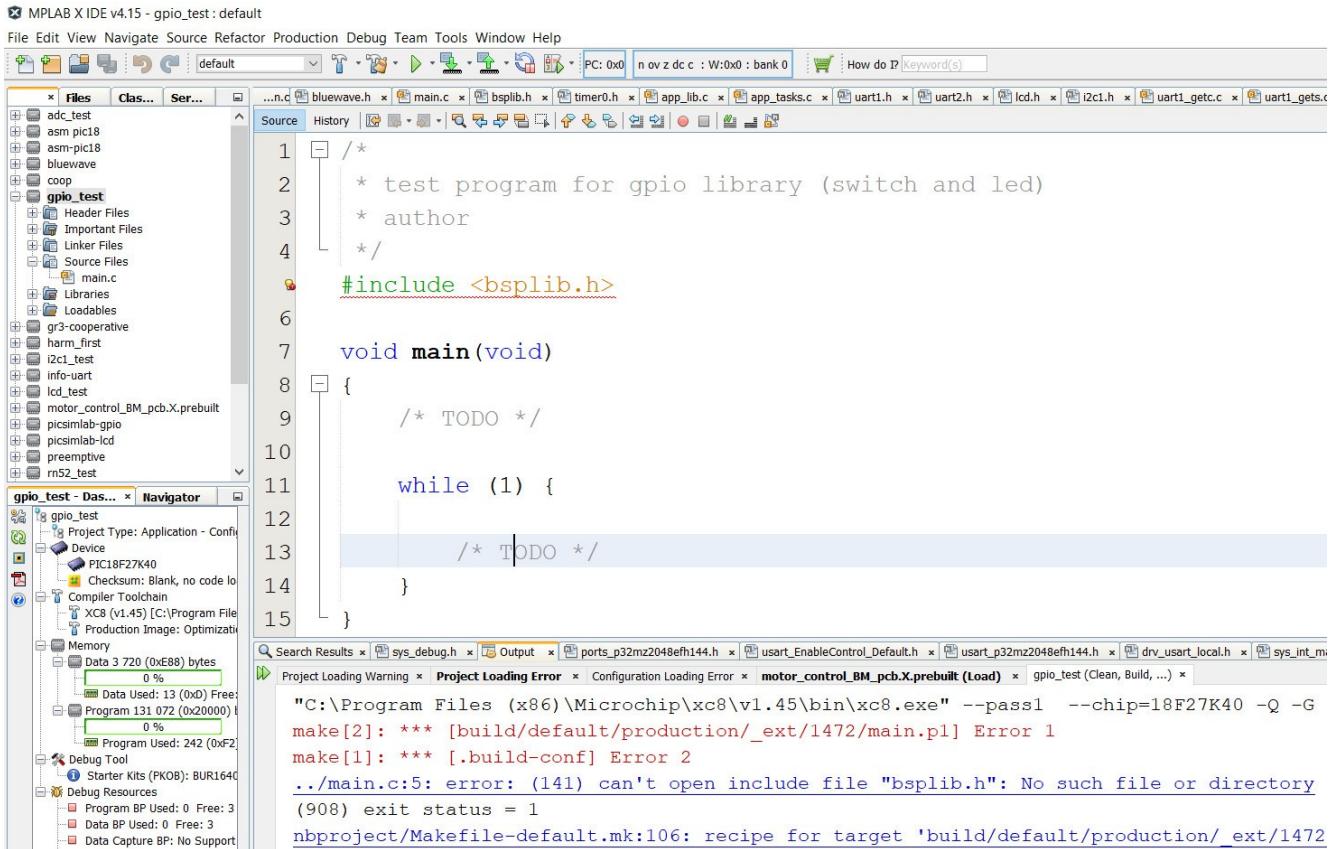
- En première année, l'arborescence du système de fichiers du projet vous est donnée. Construire une telle arborescence demande déjà une certaine maturité en ingénierie afin de la penser parcimonieuse, simple, claire, modulable, etc. Ce travail vous sera demandé durant les projets de 2^{ième} et 3^{ième} année en majeures IA et SATE. Chaque sous projet possède une arborescence locale commune (cf. ci-dessous). Toujours créer vos projets dans le répertoire *pjct* correspondant au travail demandé. De même, toujours donner à vos projets un nom en lien avec les développements en cours. Par exemple, gpio, adc_pjct, uart_test et donc éviter des noms ne permettant pas de connaître le contenu du projet, par exemple test, projet, tp1, exo3, etc :
 - *src* (fichiers sources .c)
 - *include* (fichiers d'en-tête .h)
 - *pjct* ou *test/pjct* (emplacement des fichiers internes de MPLABX, Makefile, fichiers de compilation et de production .obj, .elf, etc)
- *Project Name* → <choisir_un_nom_court_ayant_un_sens>
- *Project Location* → <your_path>/mcu/tp/disco/bsp/gpio/test/pjct (à adapter bien entendu d'un projet à un autre)
- [x] *Use project location as the project folder* → Finish



- La création du projet est maintenant terminée. Reste à inclure les sources, configurer les outils de compilation et s'assurer de la bonne compilation du projet complet. Les développements pourront alors commencer. Vous pouvez d'ailleurs constater sur chaque capture d'écran précédente que de nombreux projets avaient déjà été créés sur mon ordinateur de travail.

Ajouter les fichiers sources

- *Clic droit sur le nom du projet (fenêtre à gauche) → Set as Main Project*
- *Clic droit sur Source Files → Add Existing Items... → <your_path>/test/main.c → Select*
- Compiler le projet. *Clic droit sur le nom du projet → Build* ou cliquer le marteau et le balai



```

1  /*
2   * test program for gpio library (switch and led)
3   * author
4   */
5
6  #include <bsplib.h>
7
8  void main(void)
9  {
10    /* TODO */
11
12    while (1) {
13      /* TODO */
14    }
15}

```

Project Loading Error: "C:\Program Files (x86)\Microchip\xc8\v1.45\bin\xc8.exe" --pass1 --chip=18F27K40 -Q -G
make[2]: *** [build/default/production/_ext/1472/main.p1] Error 1
make[1]: *** [.build-conf] Error 2
./main.c:5: error: (141) can't open include file "bsplib.h": No such file or directory
(908) exit status = 1
nbproject/Makefile-default.mk:106: recipe for target 'build/default/production/_ext/1472'

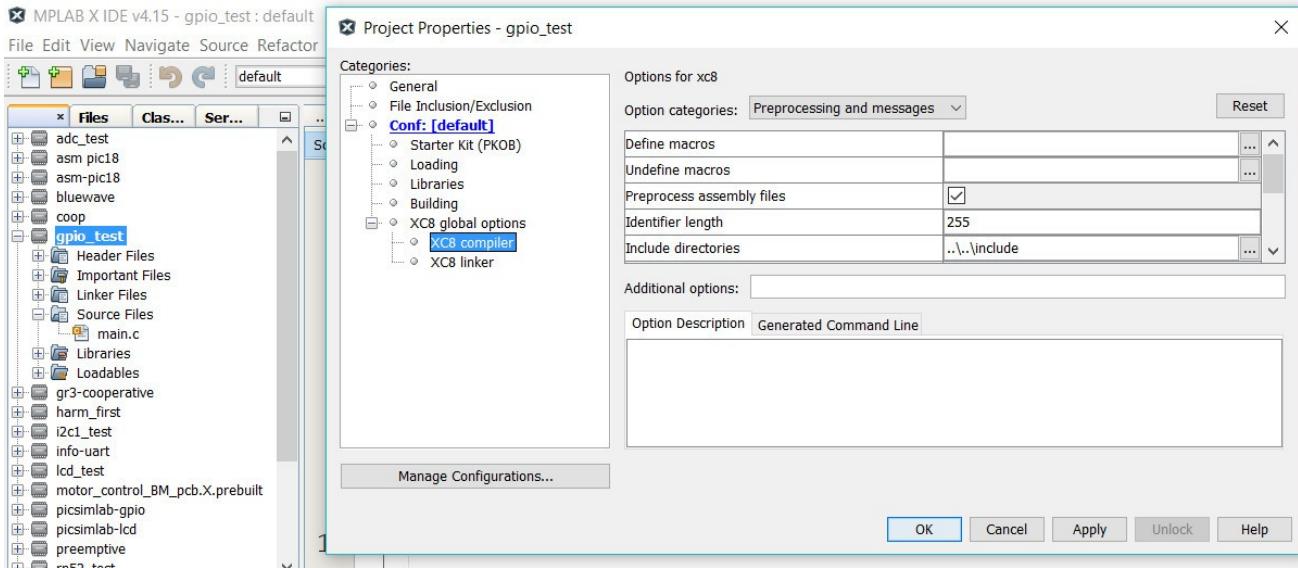
- Première erreur de compilation. Pour être plus précis, il s'agit du préprocesseur du C qui ne trouve pas le fichier *bsplib.h*. En effet, ce fichier a été développé spécifiquement par moi-même (hugo) pour l'application de TP sur la carte Curiosity HPC. La chaîne de compilation ne fera jamais d'hypothèse sur l'emplacement potentiel d'un header applicatif. Les chemins doivent être explicitement précisés à la toolchain C !

Configurer les outils de compilation

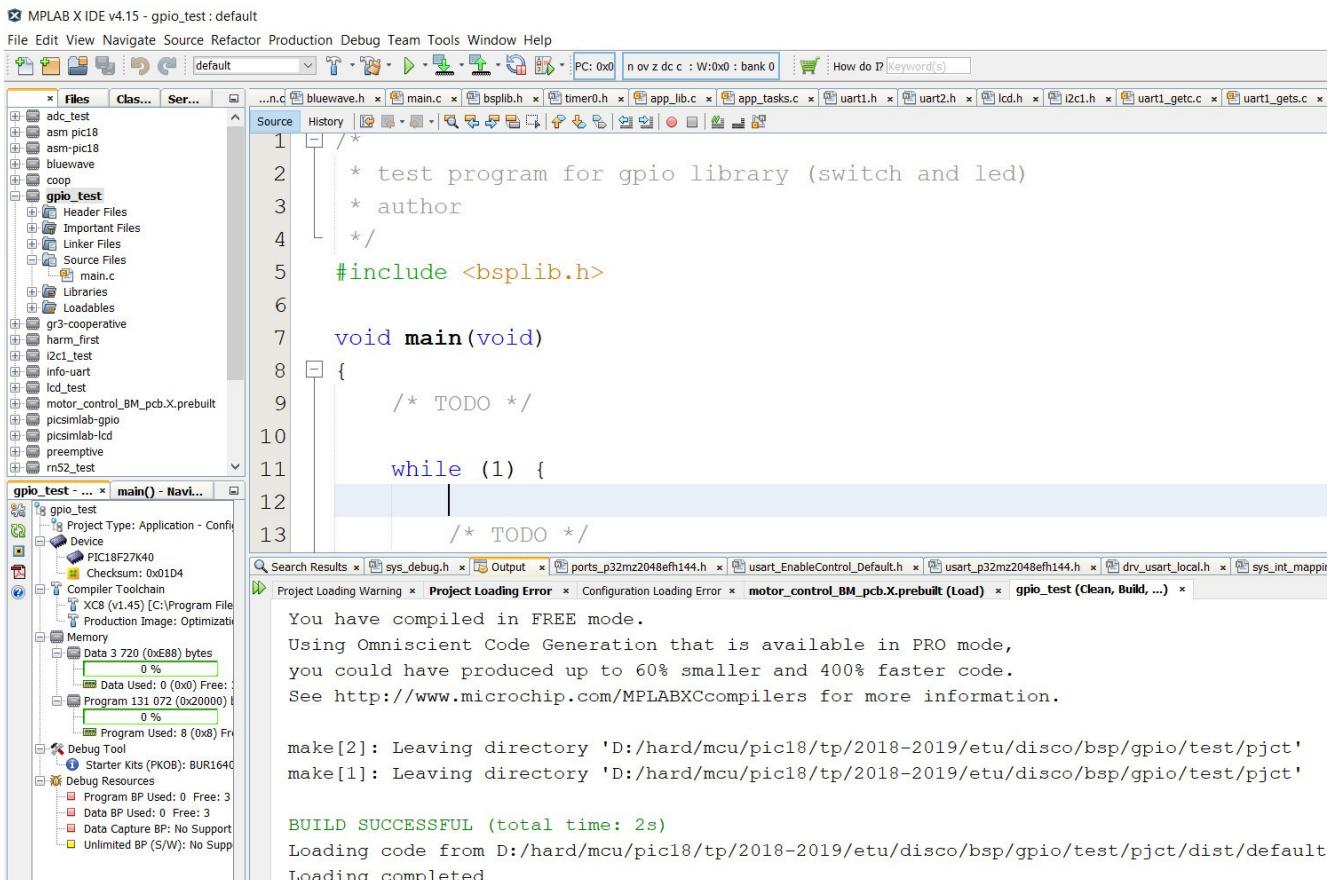
- *Clic droit sur le nom du projet → Properties*
- *Cliquer sur XC8 compiler*
- *Option Categories → Preprocessing and message*
- Vérifier l'emplacement du fichier. Recherche Windows de *bsplib.h* dans le répertoire *disco*

Annexes

- Include directories → ... → Browse... → <your_path>/bsp/ → Apply
- Vous constaterez que l'IDE définit par défaut des chemins relatifs à l'emplacement courant du projet sur la machine. Cela améliore la portabilité des projets de machine en machine.



- Compiler le projet. Clic droit sur le nom du projet → Build



- Le projet compile. Reste maintenant à charger le firmware de sortie sur la cible !

Charger et exécuter le programme sur la cible

- Une fois la compilation du projet validée, il ne reste plus qu'à charger le firmware en mémoire programme flash interne du MCU. Nous disons souvent "flasher" le processeur. Le transfert se fera par la sonde JTAG précédemment présentée. Au chargement, vous constaterez que la fenêtre de sortie de l'IDE commutera de la fenêtre "build, load,..." (compilation) à "Starter Kit on Board" (programmation et exécution). L'IDE est alors en train d'échanger avec la sonde JTAG.
- Clic droit sur le nom du projet → Run*

```
4  /*
5   *include <bsplib.h>
6
7   void main(void)
8   {
9     /* TODO */
10
11   while (1) {
12     /* TODO */
13   }
14 }
15
16
```

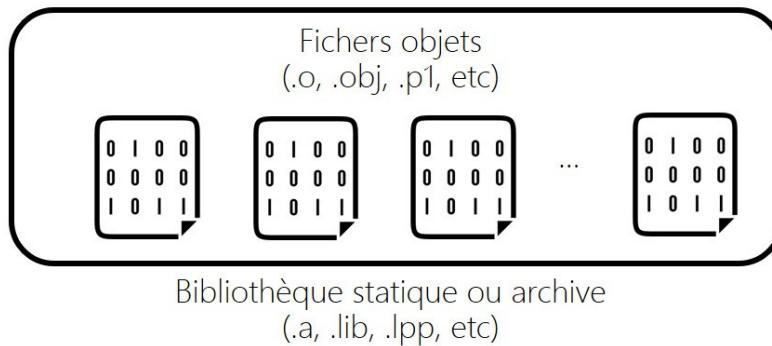
gpio_test - ... x main() - Navi... Project Type: Application - Configuration: PIC18F27K40 Device: PIC18F27K40 Checksum: 0x104 Compiler Toolchain: XC8 (v1.45) [C:\Program Files\Microchip\XC8\Production Image: Optimization] Memory: Data 3 720 (0xE88) bytes Data Used: 0 (0x0) Free: 3 Program 131 072 (0x20000) bytes Program Used: 8 (0x8) Free: 128 Debug Tool: Starter Kits (PKOB): BUR160 Debug Resources: Program BP Used: 0 Free: 3 Data BP Used: 0 Free: 3 Data Capture BP: No Support Unlimited BP (S/W): No Support

gpio_test (Build, Load, ...) x Starter Kit on Board x Target device PIC18F27K40 found. Device ID Revision = a003 Device Erased... Programming... The following memory area(s) will be programmed: program memory: start address = 0x0, end address = 0xffff configuration memory User Id Memory Programming/Verify complete

- L'affichage *Programming/Verify complete* spécifie le bon chargement du firmware dans le MCU cible et le début de son exécution. Il reste maintenant à vérifier le bon fonctionnement de votre application !

2. CREATION D'UNE BIBLIOTHEQUE STATIQUE SOUS MPLABX

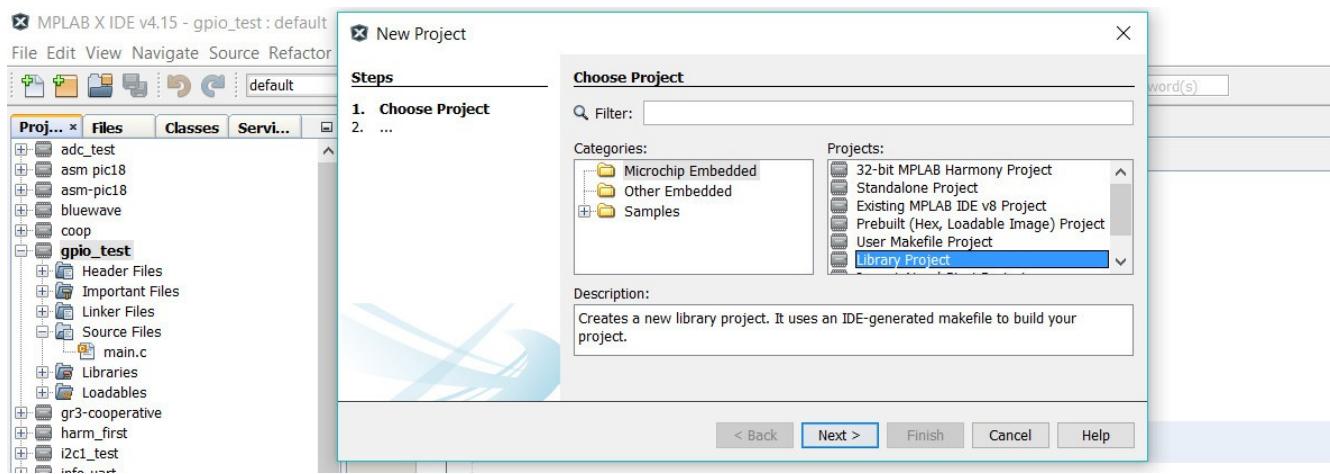
2. CREATION D'UNE BIBLIOTHEQUE STATIQUE SOUS MPLABX



Une bibliothèque statique n'est qu'une archive (concaténation) de fichiers objets pré-compilés. Il s'agit d'ailleurs de l'outil *ar* (archiver) sur système GNU\Linux. Une librairie statique ne comporte pas de fonction *main*, elle n'est qu'un agglomérat de fonctions présentes dans des fichiers au format binaire (ELF, HEX, etc). La fonction *main* est quant à elle le point d'entrée d'une application.

Sélection du générateur de bibliothèque statique

- Ouvrir MPLABX → File → New Project...
- Microchip Embedded → Library Project → Next



Sélection du processeur cible

- Family → Advanced 8-bits MCUs (PIC18)
- Device → PIC18F27K40 → Next

Sélection de la sonde de programmation ou du starter kit

- *Quelque soit le choix réalisé, cela n'influera en rien la bonne génération de la bibliothèque*

Sélection de la chaîne de compilation

- *XC8 (v1.45) → Next*

Sélection de l'emplacement du projet

- Toujours donner à vos projet un nom en lien avec les développements en cours. Dans le cas présent, une bibliothèque ou library. Par exemple, bsp_lib, adclib, lib_uart et donc éviter des noms ne permettant pas de connaître le contenu du projet, par exemple lib, tp1, exo3, etc. Le projet MPLABX pour la génération de la bibliothèque statique intégrant tous les binaires du BSP (Board Support Package) développés durant les TP sera placé dans le répertoire *disco/bsp/lib/pjct*.
- *Project Name → <choisir_un_nom_court_ayant_un_sens>*
- *Project Location → <your_part>/mcu/tp/disco/bsp/lib/pjct*
- *[x] Use project location as the project folder → Finish*

Ajouter les fichiers sources de la bibliothèque

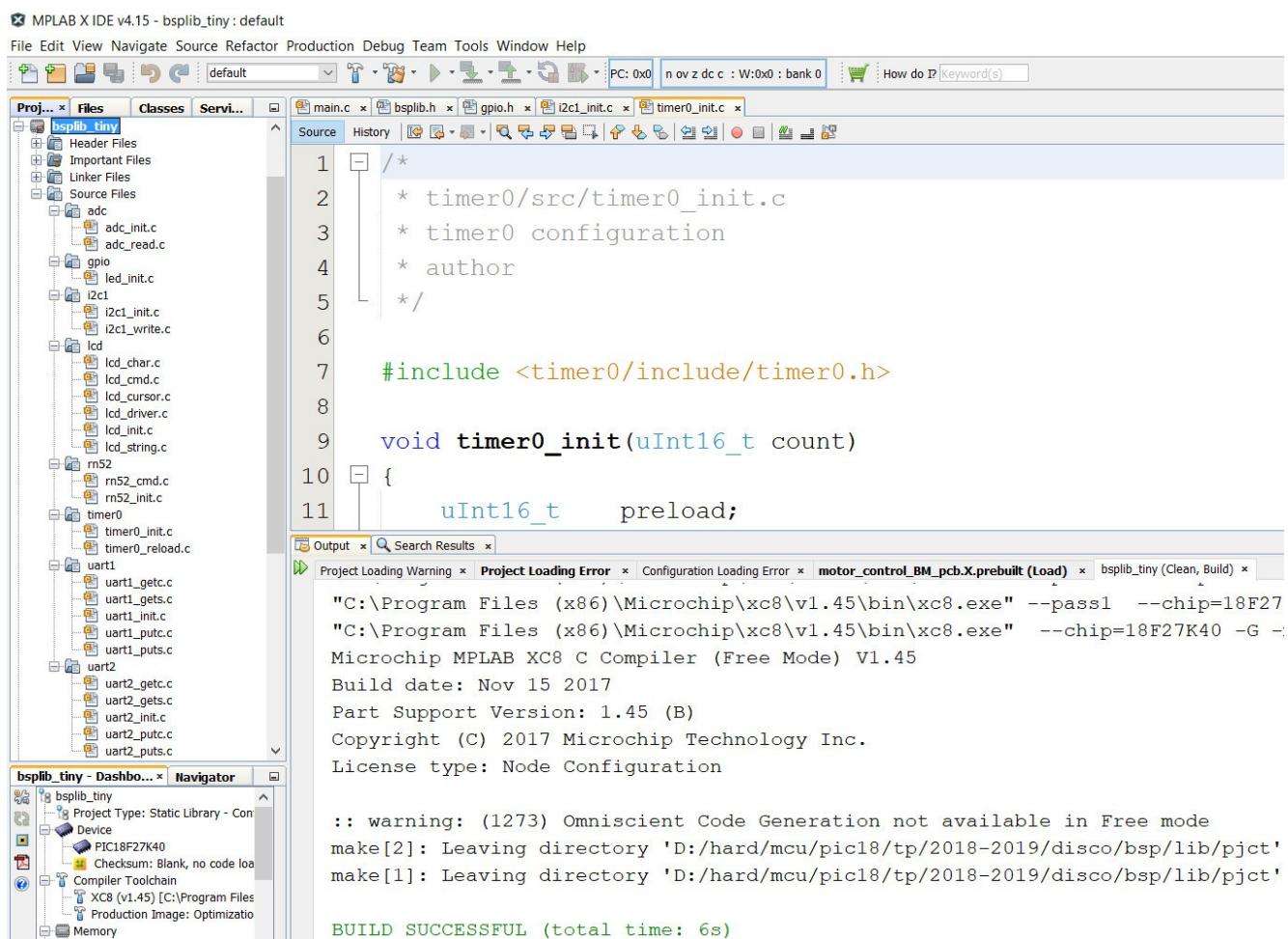
- *Clic droit sur le nom du projet → Set as Main Project*
- *Clic droit sur Source Files → Add Existing Items... → <add_all_needed_sources> → Select*
- Compiler le projet. *Clic droit sur le nom du projet → Build*

Configurer les outils de compilation

- *Clic droit sur le nom du projet → Properties*
- *Cliquer sur XC8 compiler*
- *Option Categories → Preprocessing and message*
- *Include directories → ... → Browse... → <your_part>/bsp/ → Apply*

Génération de la bibliothèque statique

- BUG, pour une raison que j'ignore encore, je n'arrive pas à générer de bibliothèque statique incluant des fichiers objets assemblés depuis des fichiers sources assembleur sous XC8.
Nous ne générerons dans un premier temps qu'une bibliothèque évolutive de TP en TP à partir des fichiers sources C développés. 1Pt en plus dans la moyenne de TP à celle ou celui qui me résout le problème !
- Clic droit sur Source Files → New Logical Folder afin de générer une arborescence logique permettant un usage efficace de l'environnement graphique proposé par l'IDE. Bien s'assurer que tous les sources nécessaires sont bien présents (cf. capture d'écran ci-dessous).
- Clic droit sur le nom du projet → Build



```

MPLAB X IDE v4.15 - bsplib_tiny : default
File Edit View Navigate Source Refactor Production Debug Team Tools Window Help
Proj... Files Classes Servi...
bsplib_tiny
Header Files
Important Files
Linker Files
Source Files
  adc
    adc_init.c
    adc_read.c
  gpio
    led_init.c
  I2C
    I2C1
      I2C1_init.c
      I2C1_write.c
  LCD
    LCD_char.c
    LCD_cmd.c
    LCD_cursor.c
    LCD_driver.c
    LCD_init.c
    LCD_string.c
  RN52
    RN52_CMD.C
    RN52_init.c
  Timer0
    timer0_init.c
    timer0_reload.c
  Uart1
    uart1_getc.c
    uart1_gets.c
    uart1_init.c
    uart1_putc.c
    uart1_puts.c
  Uart2
    uart2_getc.c
    uart2_gets.c
    uart2_init.c
    uart2_putc.c
    uart2_puts.c
Output x Search Results x
Project Loading Warning x Project Loading Error x Configuration Loading Error x motor_control_BM_pcb.X.prebuilt (Load) x bsplib_tiny (Clean, Build) x
"C:\Program Files (x86)\Microchip\xc8\v1.45\bin\xc8.exe" --pass1 --chip=18F27
"C:\Program Files (x86)\Microchip\xc8\v1.45\bin\xc8.exe" --chip=18F27K40 -G -
Microchip MPLAB XC8 C Compiler (Free Mode) V1.45
Build date: Nov 15 2017
Part Support Version: 1.45 (B)
Copyright (C) 2017 Microchip Technology Inc.
License type: Node Configuration

:: warning: (1273) Omniscient Code Generation not available in Free mode
make[2]: Leaving directory 'D:/hard/mcu/pic18/2018-2019/disco/bsp/lib/pjct'
make[1]: Leaving directory 'D:/hard/mcu/pic18/2018-2019/disco/bsp/lib/pjct'

BUILD SUCCESSFUL (total time: 6s)

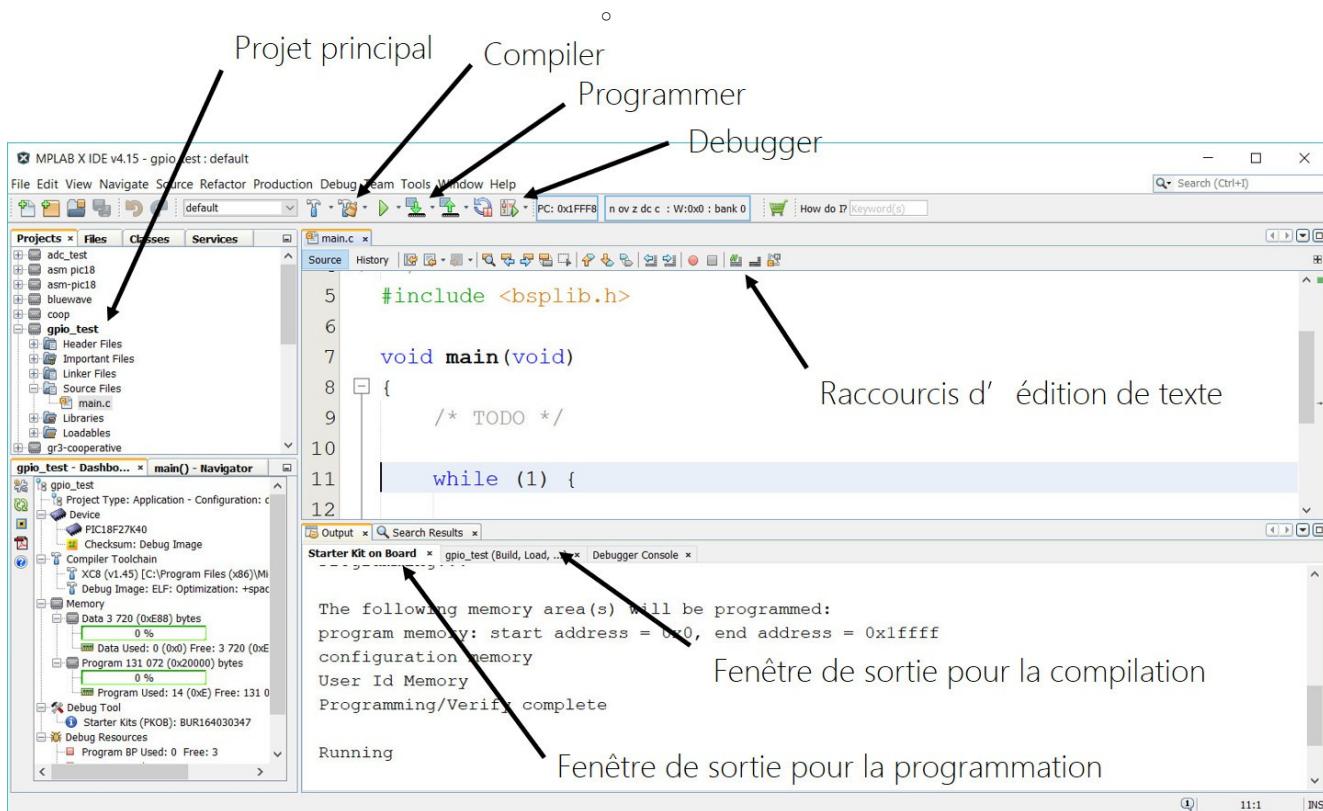
```

- Après génération, la bibliothèque sera rangée par défaut dans le répertoire `pjct/dist/default/production/<lib_name>.lpp`. A copier voire renommer dans `bsp/lib`.
- La bibliothèque complète générée en fin d'enseignement sera le fruit de plusieurs mois de travail. Elle pourra maintenant être utilisée dans une infinité d'applications potentielles ... il ne reste plus qu'à imaginer des projets ... et les réaliser !

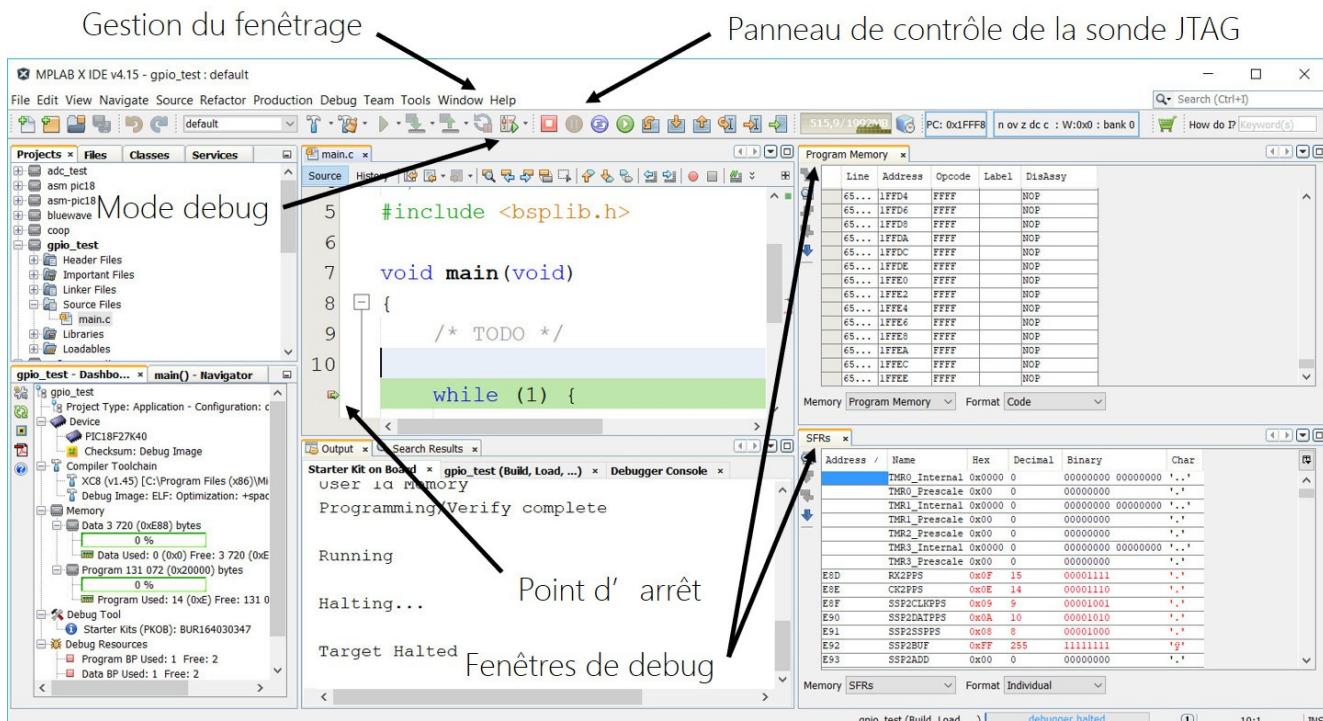
3. ENVIRONNEMENT GRAPHIQUE DE MPLABX

3. ENVIRONNEMENT GRAPHIQUE DE MPLABX

Environnement de développement



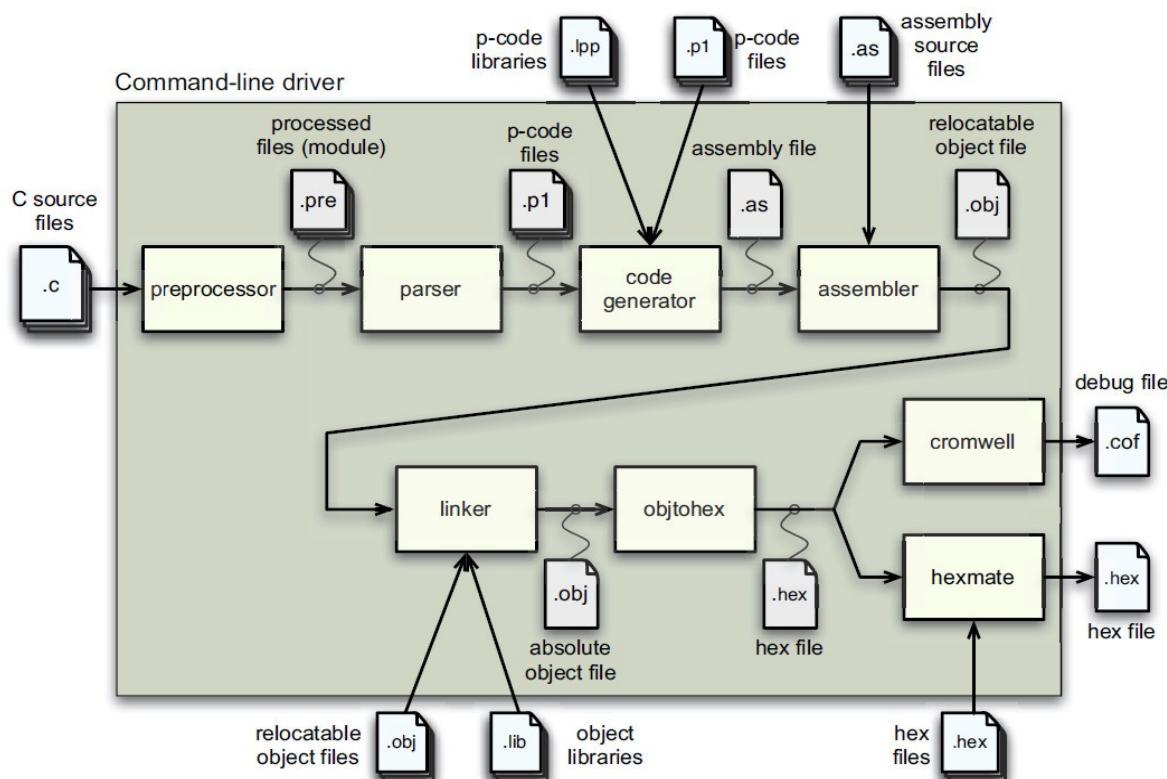
Environnement de debug



4. CHAINE DE COMPILEMENT XC8

4. CHAINE DE COMPILEATION XC8

Séquence de compilation, fichiers de construction et de production

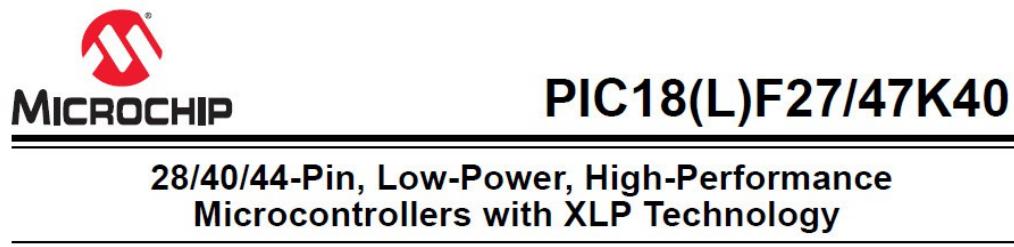


Tailles des types de donnée

Type	Size (bits)	Arithmetic Type
bit	1	Unsigned integer
signed char	8	Signed integer
unsigned char	8	Unsigned integer
signed short	16	Signed integer
unsigned short	16	Unsigned integer
signed int	16	Signed integer
unsigned int	16	Unsigned integer
signed short long	24	Signed integer
unsigned short long	24	Unsigned integer
signed long	32	Signed integer
unsigned long	32	Unsigned integer
signed long long	32	Signed integer
unsigned long long	32	Unsigned integer

5. MCU PIC17F27K40

4. MCU PIC18F27K40



Description CPU et mémoire

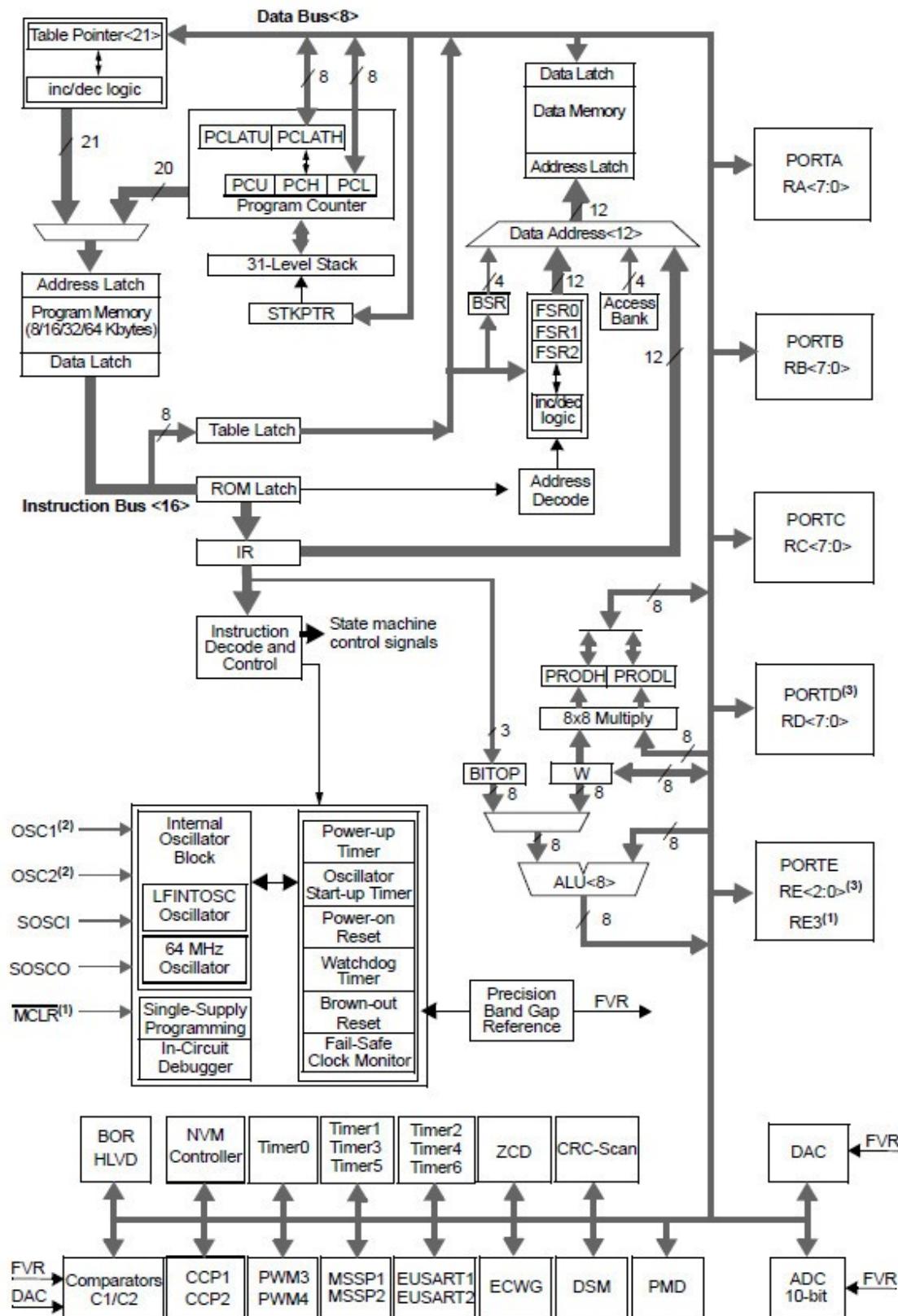
Core Features

- C Compiler Optimized RISC Architecture
- Operating Speed:
 - DC – 64 MHz clock input over the full V_{DD} range
 - 62.5 ns minimum instruction cycle
- Programmable 2-Level Interrupt Priority
- 31-Level Deep Hardware Stack
- Three 8-Bit Timers (TMR2/4/6) with Hardware Limit Timer (HLT)
- Four 16-Bit Timers (TMR0/1/3/5)
- Low-Current Power-on Reset (POR)
- Power-up Timer (PWRT)
- Brown-out Reset (BOR)
- Low-Power BOR (LPBOR) Option
- Windowed Watchdog Timer (WWDT):
 - Watchdog Reset on too long or too short interval between watchdog clear events
 - Variable prescaler selection
 - Variable window size selection
 - All sources configurable in hardware or software

Memory

- 128K Bytes Program Flash Memory
- 3728 Bytes Data SRAM Memory

Architecture interne du MCU PIC18F27K40



6. ASSEMBLEUR PIC18

6. ASSEMBLEUR PIC18

Format binaire des instructions

Byte-oriented file register operations

15	10	9	8	7	0
OPCODE	d	a	f (FILE #)		

d = 0 for result destination to be WREG register
 d = 1 for result destination to be file register (f)
 a = 0 to force Access Bank
 a = 1 for BSR to select bank
 f = 8-bit file register address

Example Instruction

ADDWF MYREG, W, B

Byte to Byte move operations (2-word)

15	12	11	0
OPCODE	f (Source FILE #)		
15	12	11	0
1111	f (Destination FILE #)		

f = 12-bit file register address

MOVFF MYREG1, MYREG2

Bit-oriented file register operations

15	12	11	9	8	7	0
OPCODE	b (BIT #)	a	f (FILE #)			

b = 3-bit position of bit in file register (f)
 a = 0 to force Access Bank
 a = 1 for BSR to select bank
 f = 8-bit file register address

BSF MYREG, bit, B

Literal operations

15	8	7	0
OPCODE	k	(literal)	

k = 8-bit immediate value

MOVLW 7Fh

Control operations

CALL, GOTO and Branch operations

15	8	7	0
OPCODE	n<7:0> (literal)		
15	12	11	0
1111	n<19:8> (literal)		

n = 20-bit immediate value

GOTO Label

15	8	7	0
OPCODE	S	n<7:0> (literal)	
15	12	11	0
1111	n<19:8> (literal)		

S = Fast bit

CALL MYFUNC

15	11	10	0
OPCODE	n	<10: 0> (literal)	

BRA MYFUNC

15	8	7	0
OPCODE	n<7:0> (literal)		

BC MYFUNC

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word					Status Affected	Notes		
				MSb				LSb				
BYTE-ORIENTED OPERATIONS												
ADDWF	f, d, a	Add WREG and f	1	0010	01da	ffff	ffff	C, DC, Z, OV, N	1, 2			
ADDWFC	f, d, a	Add WREG and CARRY bit to f	1	0010	00da	ffff	ffff	C, DC, Z, OV, N	1, 2			
ANDWF	f, d, a	AND WREG with f	1	0001	01da	ffff	ffff	Z, N	1, 2			
CLRF	f, a	Clear f	1	0110	101a	ffff	ffff	Z	2			
COMF	f, d, a	Complement f	1	0001	11da	ffff	ffff	Z, N	1, 2			
CPFSEQ	f, a	Compare f with WREG, skip =	1 (2 or 3)	0110	001a	ffff	ffff	None	4			
CPFSGT	f, a	Compare f with WREG, skip >	1 (2 or 3)	0110	010a	ffff	ffff	None	4			
CPFSLT	f, a	Compare f with WREG, skip <	1 (2 or 3)	0110	000a	ffff	ffff	None	1, 2			
DECF	f, d, a	Decrement f	1	0000	01da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4			
DECFSZ	f, d, a	Decrement f, Skip if 0	1 (2 or 3)	0010	11da	ffff	ffff	None	1, 2, 3, 4			
DCFSNZ	f, d, a	Decrement f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2			
INCF	f, d, a	Increment f	1	0010	10da	ffff	ffff	C, DC, Z, OV, N	1, 2, 3, 4			
INCFSZ	f, d, a	Increment f, Skip if 0	1 (2 or 3)	0011	11da	ffff	ffff	None	4			
INFSNZ	f, d, a	Increment f, Skip if Not 0	1 (2 or 3)	0100	11da	ffff	ffff	None	1, 2			
IORWF	f, d, a	Inclusive OR WREG with f	1	0001	00da	ffff	ffff	Z, N	1, 2			
MOVF	f, d, a	Move f	1	0101	00da	ffff	ffff	Z, N	1			
MOVFF	f _s , f _d	Move f _s (source) to 1st word	2	1100	ffff	ffff	ffff	None				

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
		f _d (destination) 2nd word		1111	ffff	ffff	ffff		
MOVWF	f, a	Move WREG to f	1	0110	111a	ffff	ffff	None	
MULWF	f, a	Multiply WREG with f	1	0000	001a	ffff	ffff	None	1, 2
NEGF	f, a	Negate f	1	0110	110a	ffff	ffff	C, DC, Z, OV, N	
RLCF	f, d, a	Rotate Left f through Carry	1	0011	0lda	ffff	ffff	C, Z, N	1, 2
RLNCF	f, d, a	Rotate Left f (No Carry)	1	0100	0lda	ffff	ffff	Z, N	
RRCF	f, d, a	Rotate Right f through Carry	1	0011	00da	ffff	ffff	C, Z, N	
RRNCF	f, d, a	Rotate Right f (No Carry)	1	0100	00da	ffff	ffff	Z, N	
SETF	f, a	Set f	1	0110	00da	ffff	ffff	None	1, 2
SUBFWB	f, d, a	Subtract f from WREG with borrow	1	0101	0lda	ffff	ffff	C, DC, Z, OV, N	
SUBWF	f, d, a	Subtract WREG from f	1	0101	11da	ffff	ffff	C, DC, Z, OV, N	1, 2
SUBWFB	f, d, a	Subtract WREG from f with borrow	1	0101	10da	ffff	ffff	C, DC, Z, OV, N	
SWAPF	f, d, a	Swap nibbles in f	1	0011	10da	ffff	ffff	None	4
TSTFSZ	f, a	Test f, skip if 0	1 (2 or 3)	0110	01la	ffff	ffff	None	1, 2
XORWF	f, d, a	Exclusive OR WREG with f	1	0001	10da	ffff	ffff	Z, N	
BIT-ORIENTED OPERATIONS									
BCF	f, b, a	Bit Clear f	1	1001	bbba	ffff	ffff	None	1, 2
BSF	f, b, a	Bit Set f	1	1000	bbba	ffff	ffff	None	1, 2

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
BTFSC	f, b, a	Bit Test f, Skip if Clear	1 (2 or 3)	1011	bbba	ffff	ffff	None	3, 4
BTFSS	f, b, a	Bit Test f, Skip if Set	1 (2 or 3)	1010	bbba	ffff	ffff	None	3, 4
BTG	f, b, a	Bit Toggle f	1	0111	bbba	ffff	ffff	None	1, 2
CONTROL OPERATIONS									
BC	n	Branch if Carry	1 (2)	1110	0010	nnnn	nnnn	None	4
BN	n	Branch if Negative	1 (2)	1110	0110	nnnn	nnnn	None	
BNC	n	Branch if Not Carry	1 (2)	1110	0011	nnnn	nnnn	None	
BNN	n	Branch if Not Negative	1 (2)	1110	0111	nnnn	nnnn	None	
BNOV	n	Branch if Not Overflow	1 (2)	1110	0101	nnnn	nnnn	None	
BNZ	n	Branch if Not Zero	1 (2)	1110	0001	nnnn	nnnn	None	
BOV	n	Branch if Overflow	1 (2)	1110	0100	nnnn	nnnn	None	
BRA	n	Branch Unconditionally	2	1101	0nnn	nnnn	nnnn	None	
BZ	n	Branch if Zero	1 (2)	1110	0000	nnnn	nnnn	None	
CALL	k, s	Call subroutine 1st word	2	1110	110s	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		
CLRWDT	—	Clear Watchdog Timer	1	0000	0000	0000	0100	TO, PD	
DAW	—	Decimal Adjust WREG	1	0000	0000	0000	0111	C	
GOTO	k	Go to address 1st word	2	1110	1111	kkkk	kkkk	None	
		2nd word		1111	kkkk	kkkk	kkkk		

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			LSb		
NOP	—	No Operation	1	0000	0000	0000	0000	None	
NOP	—	No Operation	1	1111	xxxx	xxxx	xxxx	None	
POP	—	Pop top of return stack (TOS)	1	0000	0000	0000	0110	None	
PUSH	—	Push top of return stack (TOS)	1	0000	0000	0000	0101	None	
RCALL	n	Relative Call	2	1101	1nnn	nnnn	nnnn	None	
RESET		Software device Reset	1	0000	0000	1111	1111	All	
RETFIE	s	Return from interrupt enable	2	0000	0000	0001	000s	GIE/GIEH, PEIE/GIEL	
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
RETURN	s	Return from Subroutine	2	0000	0000	0001	001s	None	
SLEEP	—	Go into Standby mode	1	0000	0000	0000	0011	TO, PD	
LITERAL OPERATIONS									
ADDLW	k	Add literal and WREG	1	0000	1111	kkkk	kkkk	C, DC, Z, OV, N	
ANDLW	k	AND literal with WREG	1	0000	1011	kkkk	kkkk	Z, N	
IORLW	k	Inclusive OR literal with WREG	1	0000	1001	kkkk	kkkk	Z, N	
LFSR	f, k	Move literal (12-bit) 2nd word to FSR(f) 1st word	2	1110	1110	00ff	kkkk	None	
				1111	0000	kkkk	kkkk		
MOVLB	k	Move literal to BSR<3:0>	1	0000	0001	0000	kkkk	None	
MOV LW	k	Move literal to WREG	1	0000	1110	kkkk	kkkk	None	
MULLW	k	Multiply literal with WREG	1	0000	1101	kkkk	kkkk	None	

Jeu d'instructions PIC18

Mnemonic, Operands		Description	Cycles	16-Bit Instruction Word				Status Affected	Notes
				MSb			Lsb		
RETLW	k	Return with literal in WREG	2	0000	1100	kkkk	kkkk	None	
SUBLW	k	Subtract WREG from literal	1	0000	1000	kkkk	kkkk	C, DC, Z, OV, N	
XORLW	k	Exclusive OR literal with WREG	1	0000	1010	kkkk	kkkk	Z, N	
DATA MEMORY ↔ PROGRAM MEMORY OPERATIONS									
TBLRD*		Table Read	2	0000	0000	0000	1000	None	
TBLRD*+		Table Read with post-increment		0000	0000	0000	1001	None	
TBLRD*-		Table Read with post-decrement		0000	0000	0000	1010	None	
TBLRD+*		Table Read with pre-increment		0000	0000	0000	1011	None	
TBLWT*		Table Write	2	0000	0000	0000	1100	None	
TBLWT*+		Table Write with post-increment		0000	0000	0000	1101	None	
TBLWT*-		Table Write with post-decrement		0000	0000	0000	1110	None	
TBLWT+*		Table Write with pre-increment		0000	0000	0000	1111	None	

Note:

- When a PORT register is modified as a function of itself (e.g., MOVF PORTB, 1, 0), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.
- If this instruction is executed on the TMR0 register (and where applicable, 'd' = 1), the prescaler will be cleared if assigned.
- If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.
- Some instructions are two-word instructions. The second word of these instructions will be executed as a NOP unless the first word of the instruction retrieves the information embedded in these 16 bits. This ensures that all program memory locations have a valid instruction.

7. GLOSSAIRE

SYSTEMES EMBARQUES



7. GLOSSAIRE SYSTEMES EMBARQUES

A

- **ABI** : Application Binary Interface
- **ADC** : Analog to Digital Converter
- **ALU** : Arithmetic and Logical Unit
- **AMD** : Advanced Micro Devices
- **ANSI** : American National Standards Institute
- **API** : Application Programming Interface
- **APU** : Accelerated Processor Unit
- **ARM** : société anglaise proposant des architectures CPU RISC 32bits
- **ASCII** : American Standard Code for Information Interchange

B

- **BP** : Base Pointer
- **BSL** : Board Support Library
- **BSP** : Board Support Package

C

- **CCS** : Code Composer Studio
- **CEM** : Compatibilité ElectroMagnétique
- **CISC** : Complex Instruction Set Computer
- **CPU** : Central Processing Unit
- **CSL** : Chip Support Library

D

- **DAC** : Digital to Analog Converter
- **DDR** : Double Data Rate
- **DDR SDRAM** : Double Data Rate Synchronous Dynamic Random Access Memory
- **DMA** : Direct Memory Access
- **DSP** : Digital Signal Processor
- **DSP** : Digital Signal Processing

E

- **EDMA** : Enhanced Direct Memory Access
- **EUSART** : Enhanced Universal Synchronous Asynchronous Receiver Transmitter



- **EMIF** : External Memory Interface
- **EPIC** : Explicitly Parallel Instruction Computing

F

- **FPU** : Floating Point Unit
- **FLOPS** : Floating-Point Operations Per Second
- **FMA**: Fused Multiply-Add

G

- **GCC** : Gnu Collection Compiler
- **GLCD** : Graphical Liquid Crystal Display
- **GNU** : GNU's Not UNIX
- **GPIO** : General Purpose Input Output
- **GPP** : General Purpose Processor
- **GPU** : Graphical Processing Unit

I

- **IA-64** : Intel Architecture 64bits
- **I2C** : Inter Integrated Circuit
- **ICC** : Intel C++ Compiler
- **IDE** : Integrated Development Environment
- **IDMA** : Internal Direct memory Access
- **IRQ** : Interrupt ReQuest
- **ISR** : Interrupt Software Routine
- **ISR** : Interrupt Service Routine

L

- **L1D** : Level 1 Data Memory
- **L1I** : Level 1 Instruction Memory (idem L1P)
- **L1P** : Level 1 Program Memory (idem L1I)
- **Lx** : Level x Memory
- **LCD** : Liquid Crystal Display
- **LRU** : Least Recently Used

M

- **MAC**: Multiply Accumulate
- **MCU** : Micro Controller Unit
- **MIMD** : Multiple Instructions on Multiple Data
- **MIPS** : Mega Instructions Per Second



- **MMU** : Memory Management Unit
- **MPLABX** : Microchip LABoratory 10, IDE Microchip
- **MPU** : Micro Processor Unit ou GPP
- **MPU** : Memory Protect Unit

O

-
- **OS** : Operating System

P

-
- **PC** : Program Counter
 - **PC** : Personal Computer
 - **PIC18** : Famille MCU 8bits Microchip
 - **PLD** : Programmable Logic Device
 - **POSIX** : Portable Operating System Interface, héritage d'UNIX (norme IEEE 1003)
 - **PPC** : Power PC

R

-
- **RAM** : Random Access Memory
 - **RISC** : Reduced Instruction Set Computer
 - **RS232** : Norme standardisant un protocole de communication série asynchrone
 - **RTOS** : Real Time Operating System

S

-
- **SDK** : Software Development Kit
 - **SIMD** : Single Instruction Multiple Date
 - **SIP** : System In Package
 - **SOB** : System On Board
 - **SOC** : System On Chip
 - **SOP** : Sums of products
 - **SP** : Stack Pointer
 - **SP** : Serial Port
 - **SPI** : Serial Peripheral Interface
 - **SRAM** : Static Random Access Memory
 - **SSE** : Streaming SIMD Extensions
 - **STM32** : STMicroelectronics 32bits MCU

T

- **TI** : Texas Instruments
- **TNS** : Traitement Numérique du Signal
- **TSC** : Time Stamp Counter
- **TTM** : Time To Market

U

- **UART** : Universal Asynchronous Receiver Transmitter
- **USB** : Universal Serial Bus

V

- **VHDL** : VHSIC Hardware Description langage
- **VHSIC** : Very High Speed Integrated Circuit
- **VLIW** : Very Long Instruction Word

8. REGLES DE CODAGE C

8. REGLES DE CODAGE C

Ce document a pour objectif de fixer un cadre et des règles de codage en langage C pour les travaux en Systèmes Embarqués à l'ENSICAEN. L'objectif étant de standardiser, clarifier, cadrer et faciliter le partage de codes sources au sein d'une équipe, d'un projet voire d'une entreprise. Les règles de codage imposées sont inspirées du "Linux Kernel Coding Style" et du "GNU Coding Standard". Voici le sommaire du document :

1. Indentation
2. Dénomination
3. Commentaire
4. Divers

INDENTATION

- ***Tabulation*** : 4 ou 8 caractères (à régler dans les préférences de l'éditeur de texte)

```
switch (data) {  
case 'A':  
case 'B':  
    /* comment */  
    data <= 20;  
    break;  
default:  
    break;  
}
```

- ***Nombres de niveaux d'indentation*** : Éviter plus de 3 niveaux d'indentation imbriqués.
Facilite la lisibilité du code
- ***Nombres de colonnes par page*** : 80 caractères/colonnes par page (à régler dans les préférences de l'éditeur de texte)
- ***procédures*** : cas spécial pour les fonctions, ouvrir l'accolade au début de la ligne suivante

```
int function (int x, int y)  
{  
    int z ;  
  
    ...  
    return 0;  
}
```

- **Structures de contrôle :** règles spécifiant l'indentation ainsi que les espaces à respecter pour l'écriture de structures de contrôle

```

do {
    if (a == b) {
        ...
    } else if (a > b) {
        ...
    } else {
        ...
    }

    /* single statement */
    if (condition)
        action1();
    else
        action2();

} while (condition);

```

DENOMINATION

- **Fonctions :** Toujours débuter par une minuscule. Même si cela est à éviter, pour donner un nom complexe à un objet, utiliser comme séparateur un "_"
- **Variables locales :** Toujours débuter par une minuscule. Les variables servant de compteur de boucle peuvent avoir un nom sans sens précis, par exemple "i, j, k". Sinon, toujours donner à une variable un nom court permettant d'identifier clairement son rôle. Déclarez vos variables locales en début de fonction (portabilité C89/CANSI)
- **Variables globales :** Toujours débuter par une minuscule. Toujours donner à une variable globale un nom permettant d'identifier clairement son rôle (séparateur "_"). Éviter tant que faire se peut les variables globales (ressources partagées à manipuler et protéger avec précaution)

```

float temp_ctrl = 0;

void lm4567_codec_init (void)
{
    temp_ctrl++;
    ...
}

```

- **Définition de type :** Toujours définir un nom permettant d'identifier clairement le type des variables déclarés

```

task_struct foo;      // Bien !
tstruct foo;         // Pas Bien !

```

- **Macros** : Toujours en Majuscule. Utiliser comme séparateur un "_" pour les noms complexes. Pour les macros fonctions, appliquer les même règles de dénomination que pour les fonctions classiques

```
#define UART_BAUDRATE 0xFFFF
#define mul(x, y) x * y
```

COMMENTAIRES

- *Minimiser l'usage de commentaires dans le code. Un code bien écrit, avec des noms d'interfaces bien choisies se suffit à lui-même. Une lecture de code s'adresse avant tout à un ingénieur développeur. Penser sinon à expliquer ce que votre code fait et non comment il le fait !*
- *C99 style* : A éviter "://"
- *C89 style* : A préférer "/* ... */" (solution portable)
- *Balise pour la documentation de code (exemple de Doxygen)* : Insérer dans vos cartouches quelques tags standards, par exemple ceux de Doxygen. Attention, seuls les fichiers d'en-tête doivent contenir des balises pour la génération de documentation. Doxygen permet la génération automatique de documentation vers différents formats (HTML, PDF, CHM ...). Ne pas utiliser de caractères accentués dans vos commentaires

```
/**
 * @file example.h
 * @brief demo header file
 * @author your_name
 */
int temp_conv;

/**
 * @struct str_payload_buffer
 * @brief latest converted datas from temperature sensor
 */
typedef struct {
    int size
    int* conv_tab
} str_payload_buffer;

/**
 * @brief read data codec LM4567 controller
 * @param reset reset LM4567 controller
 * @param conv converted value
 * @return null if data conversion error
 */
handle_spi lm4567_codec_read (char reset, float conv);
```

- Quelques tags supplémentaires (cf. www.doxygen.org):

```
/**  
 * @example example insertion  
 * @warning warning insertion  
 * @li bullet point insertion  
 * @mainpage main page comments insertion  
 * @image picture insertion  
 * @include code insertion  
 */
```

DIVERS

Valeur de retour : Essayer de faire en sorte que la valeur de retour d'une fonction soit représentative de la validité de son bon traitement. Par exemple, retourne zéro (ou pointeur nul) en cas d'échec et différent de zéro en cas de succès. Les résultats des procédures seront retournés par pointeur via les paramètres d'entrée







SYSTEMES EMBARQUES

Annexes











Choisissez un travail que vous aimez
et vous n'aurez pas à travailler
un seul jour de votre vie.

Confucius