Création d'un projet sous MCUXpresso

Anthony Juton – septembre 2018

Ce document a pour objectif de vous amener à créer votre premier projet dans l'environnement de développement (IDE pour Integrated Development Environnement) MCUXpresso en vous expliquant les différents éléments le constituant.
Il vous aidera à tirer partie des ressources de MCUXpresso pour observer l'exécution de votre programme dans le moindre registre du microcontrôleur.
Nombre d'informations sont issus de la documentation de MCUXpresso : Help > MCUXpresso IDE User Guide.

1. Présentation de l'environnement de développement

MCUXpresso est l'environnement de développement proposé par NXP.

- Il a l'avantage d'être gratuit, compatible avec tous les microcontrôleurs Cortex M de NXP et de proposer un débogueur, nombre d'exemples utilisant tous les périphériques de ces microcontrôleurs et un OS temps réel.
- Solution propriétaire, il n'est compatible qu'avec les microcontrôleurs de NXP.

Cependant, il est basé sur la plateforme Eclipse comme beaucoup d'environnements de développement modernes d'informatique embarquée (ST atollic et TI Code Composer Studio par exemple). Le passage de MCUXpresso à ces environnements de développement, très similaires, sera ainsi aisé.

L'environnement de développement est constitué de nombreuses *« sous-fenêtres »* nommées *vues (views)*.
Window > Perspective > Reset perspective permet de revenir à l'agencement par défaut des vues.

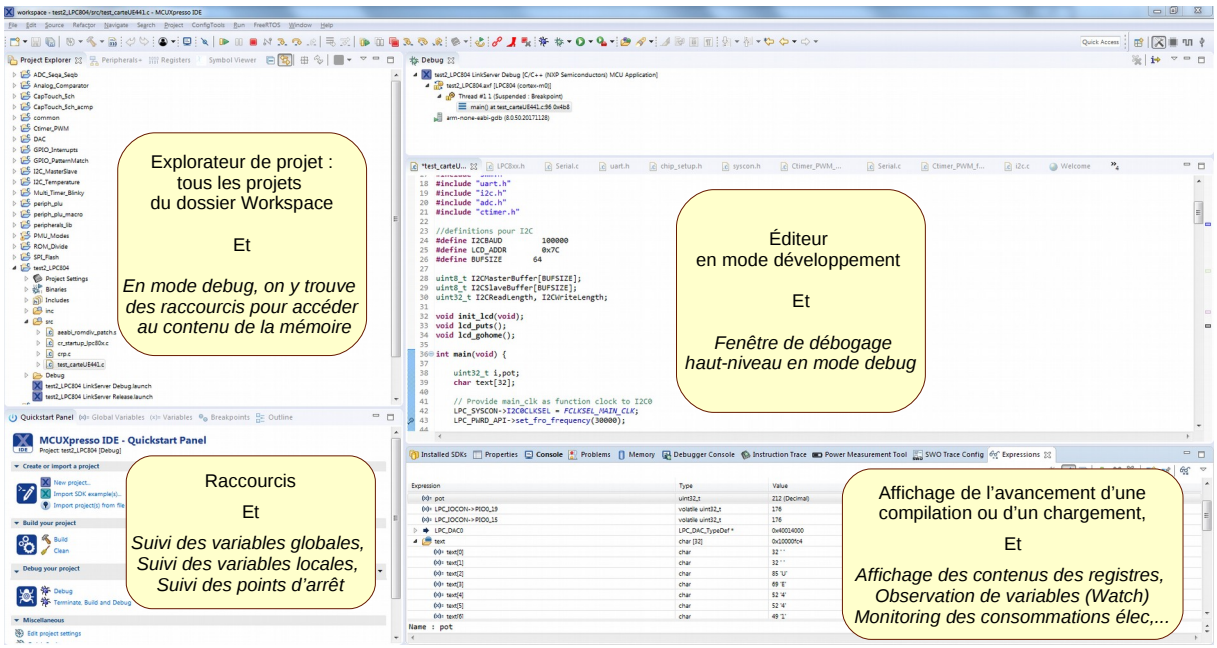


Figure 1: Les différentes vues de l'environnement de développement

Nous détaillerons les contenus de ces vues dans les chapitres suivants.

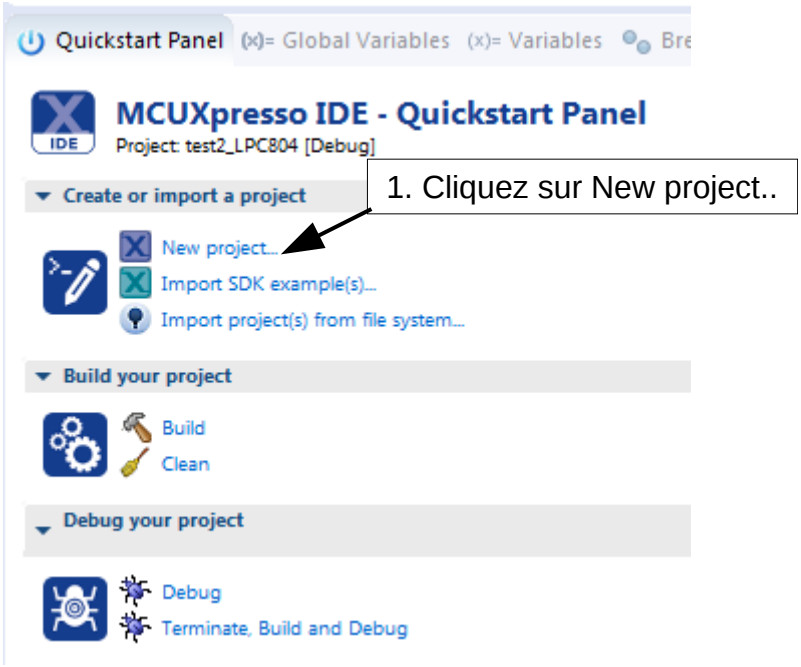
2. Création d'un premier projet allumage d'une led par un bouton

MCUXpresso travaille dans un dossier nommé Workspace, *Mes documents* > *MCUXpresso* sous windows par exemple. Il y crée un dossier par projet.

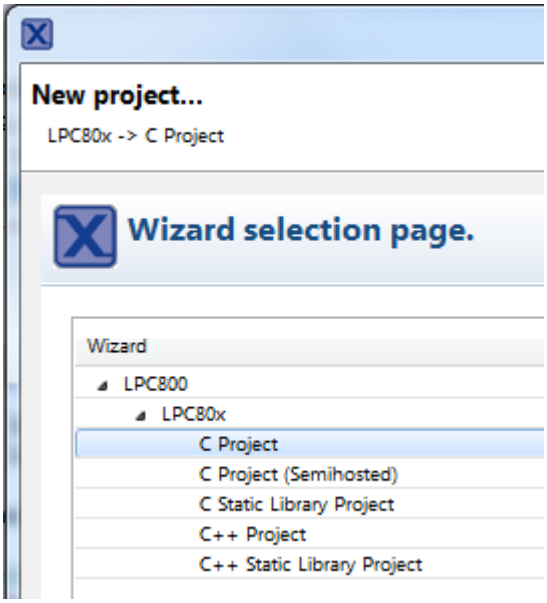
2.1. Création du projet

Un projet minimal est constitué de votre programme, d'une description des registres du microcontrôleur, des bibliothèques de base du langage C et d'un code de démarrage / initialisation.

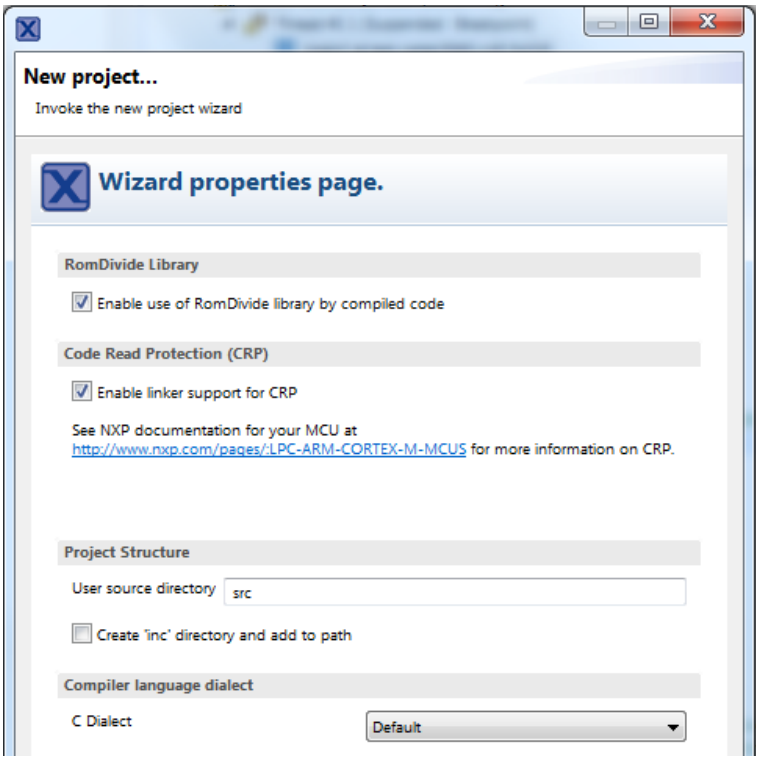
Pour créer un nouveau projet, cliquez sur *New Project…* dans les raccourcis.



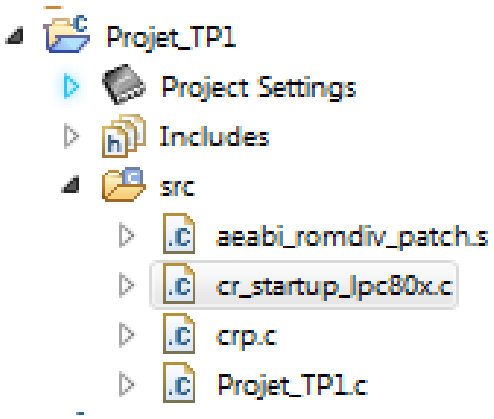
Choisissez alors la carte *LPCXpresso804* dans la liste proposée et cliquez sur *Next*.
Le choix du microcontrôleur permet à l'IDE de fournir le code de démarrage et d'initialisation.
Choisissez ensuite *C project*. Ce choix permet de faire le lien avec les bibliothèques génériques du C (*math.h, stdio.h*,...). Nous aborderons cela lorsque nous parlerons des bibliothèques.



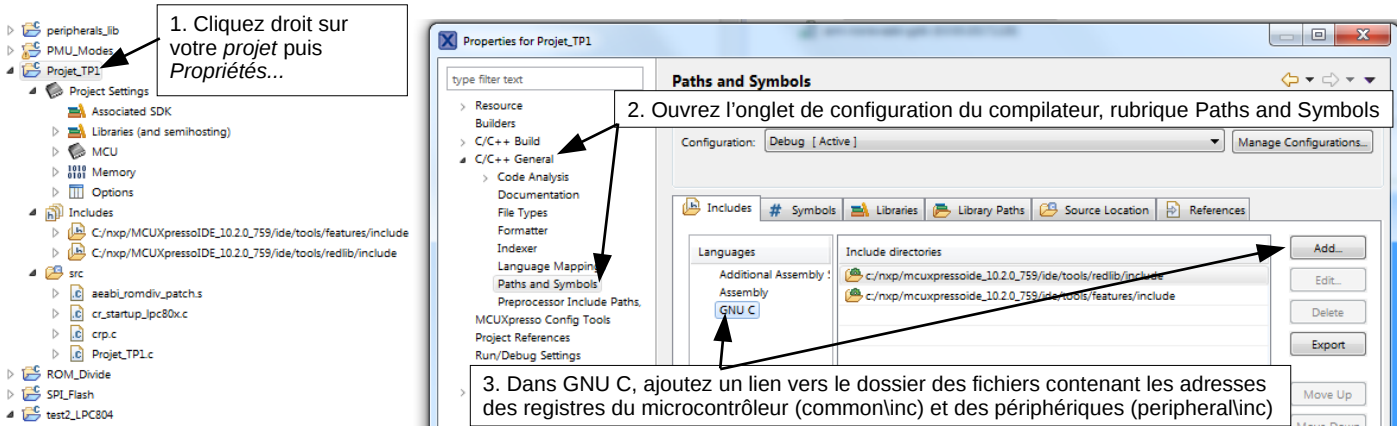
Dans la fenêtre suivante, on choisit les accès à la mémoire : laissez les options par défaut.

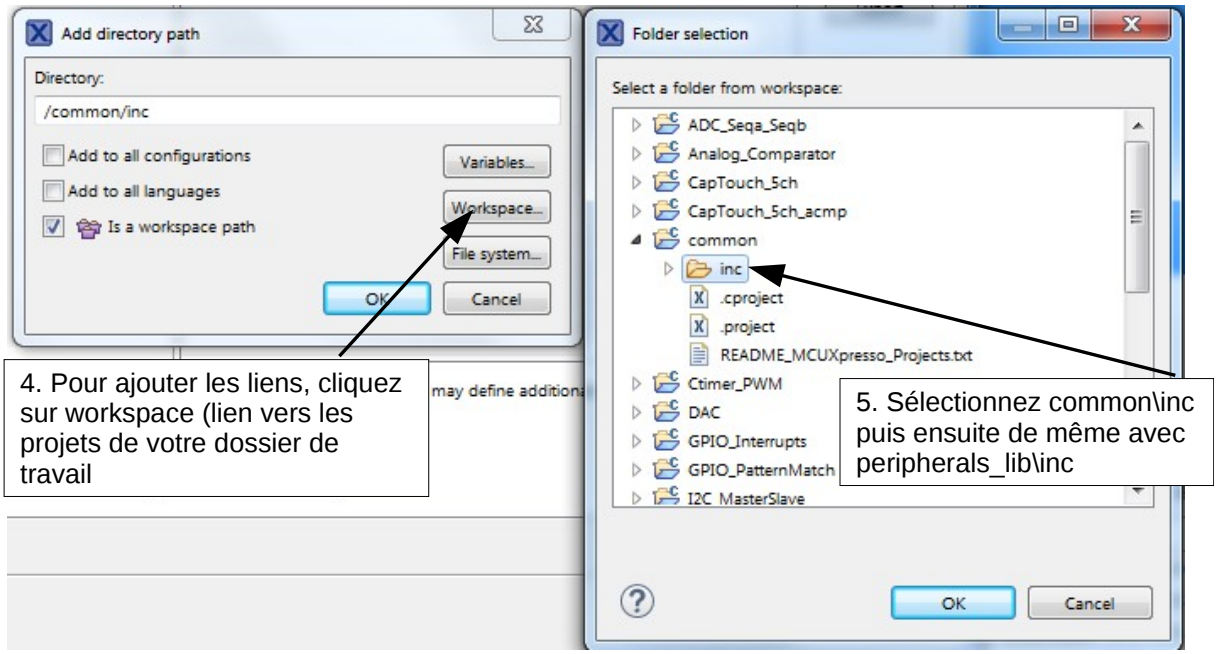


Votre projet est alors créé avec son code de démarrage, propre à votre microcontrôleur LPC804.



Il reste à ajouter les liens avec les fichiers header (*.h) contenant les adresses des registres du microcontrôleur et les définitions des symboles.





Vous pouvez alors rajouter un lien vers le fichiers en-tête contenant les adresses des registres qui nous intéressent pour ce premier TP :

#include "lpc8xx.h" : lien vers les adresses des registres du microcontrôleur.

#include "syscon.h" : lien vers les définitions des symboles liés aux registres systèmes.

Nous verrons les liens vers les bibliothèques (libraries en anglais) un peu plus tard…

2.2. Ecriture de votre programme d'allumage de la led bleue par le bouton user

L'objectif de ce programme est d'allumer la led bleue connectée à la broche P0_11 (ou PIO0_11) lorsque l'on appuie sur le bouton User connecté à la broche P0_13 (noté aussi PIO0_13).

Ces informations sont issus du schéma de la carte LPCXpresso804 :

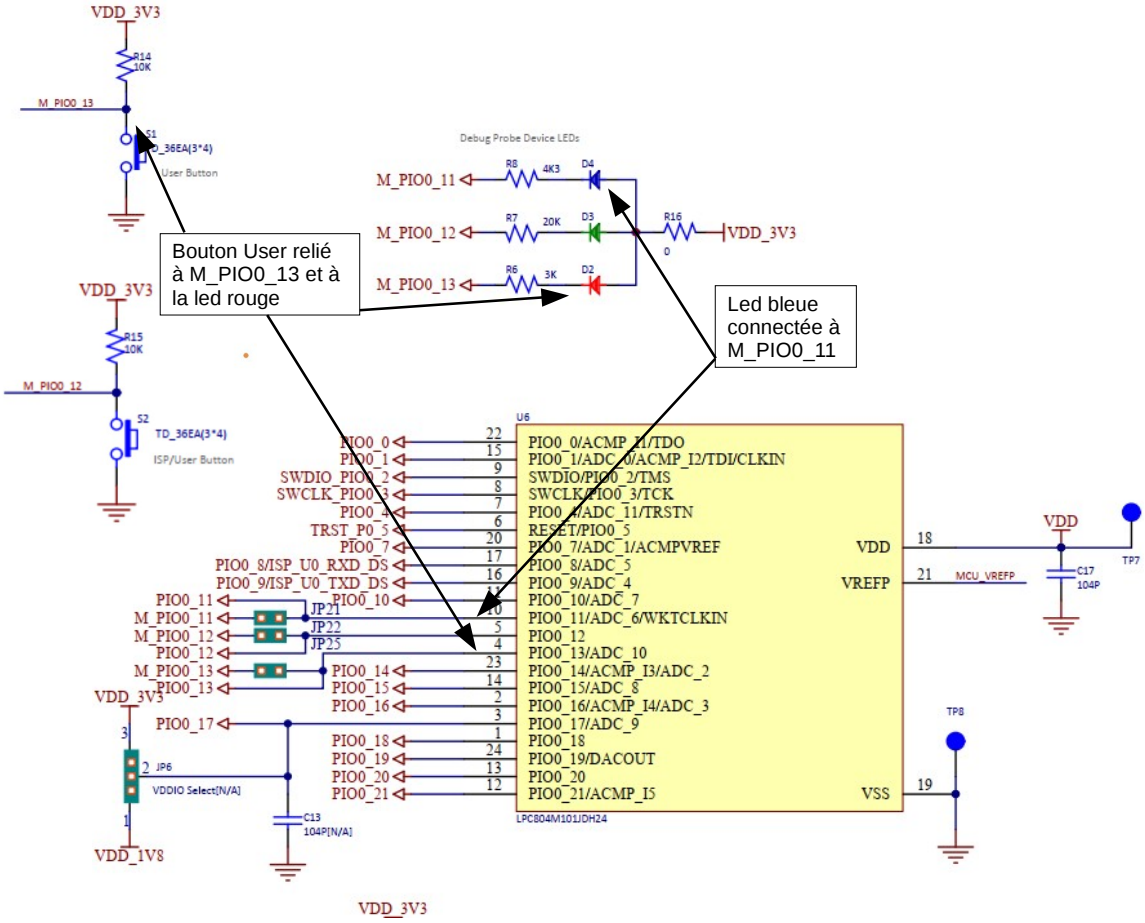


Figure 2: extrait du schéma LPC804_Devboard_b.pdf

→ Que vaut la tension à la broche PIO0_13 lorsque le bouton est relâché, quand il est enfoncé ?

→ Pour une valeur 1 de PIO0_11, quelle est la tension à la sortie PIO0_11, la led est-elle allumée ou éteinte ? Même question pour une valeur 0.

2.2.1. Allumage du périphérique Entrées/Sorties

Pour économiser l'énergie, les périphériques du microcontrôleur, y compris les entrées/sorties (GPIO pour General Purpose Input Output) sont éteints par défaut.

Pour allumer le périphérique, on utilise le registre système de contrôle de l'horloge des périphériques : SYSAHBCLKCTRL0.

6.6.10 System clock control 0 register

The SYSAHBCLKCTRL0 register enables the clocks to individual system and peripheral blocks. The system clock (bit 0) provides the clock for the AHB, the APB bridge, the Arm Cortex-M0+, the SYSCON block, and the PMU. This clock cannot be disabled.

Table 64. System clock control 0 register (SYSAHBCLKCTRL0, address 0x4004 8080) bit description

Bit	Symbol	Value	Description	Reset value
0	SYS		Enables the clock for the AHB, the APB bridge, the Cortex-M0+ core clocks, SYSCON, and the PMU. This bit is read only and always reads as 1.	1
1	ROM		Enables clock for ROM.	1
		0	Disable.	
		1	Enable.	
6	GPIO0		Enables clock for GPIO0 port registers.	0
		0	Disable.	
		1	Enable.	
18	IOCON		Enables clock for IOCON block.	0
		0	Disable.	
		1	Enable.	

Figure 3: Le registre SYSAHBCLKCTRL0, extrait du User Manual du LPC804 page 59

→ Trouvez dans le fichier lpc8xx.h la définition du registre SYSAHBCTRL0.

→ Trouvez dans le fichier syscon.h la valeur du symbole GPIO0.

→ Expliquez alors la ligne suivante :

```
LPC_SYSCON->SYSAHBCLKCTRL0 |= GPIO0;
```

2.2.2. Configuration des sorties

Chaque broche GPIO peut être configurée en entrée ou en sortie. On trouve le schéma électrique d'une broche dans la documentation :

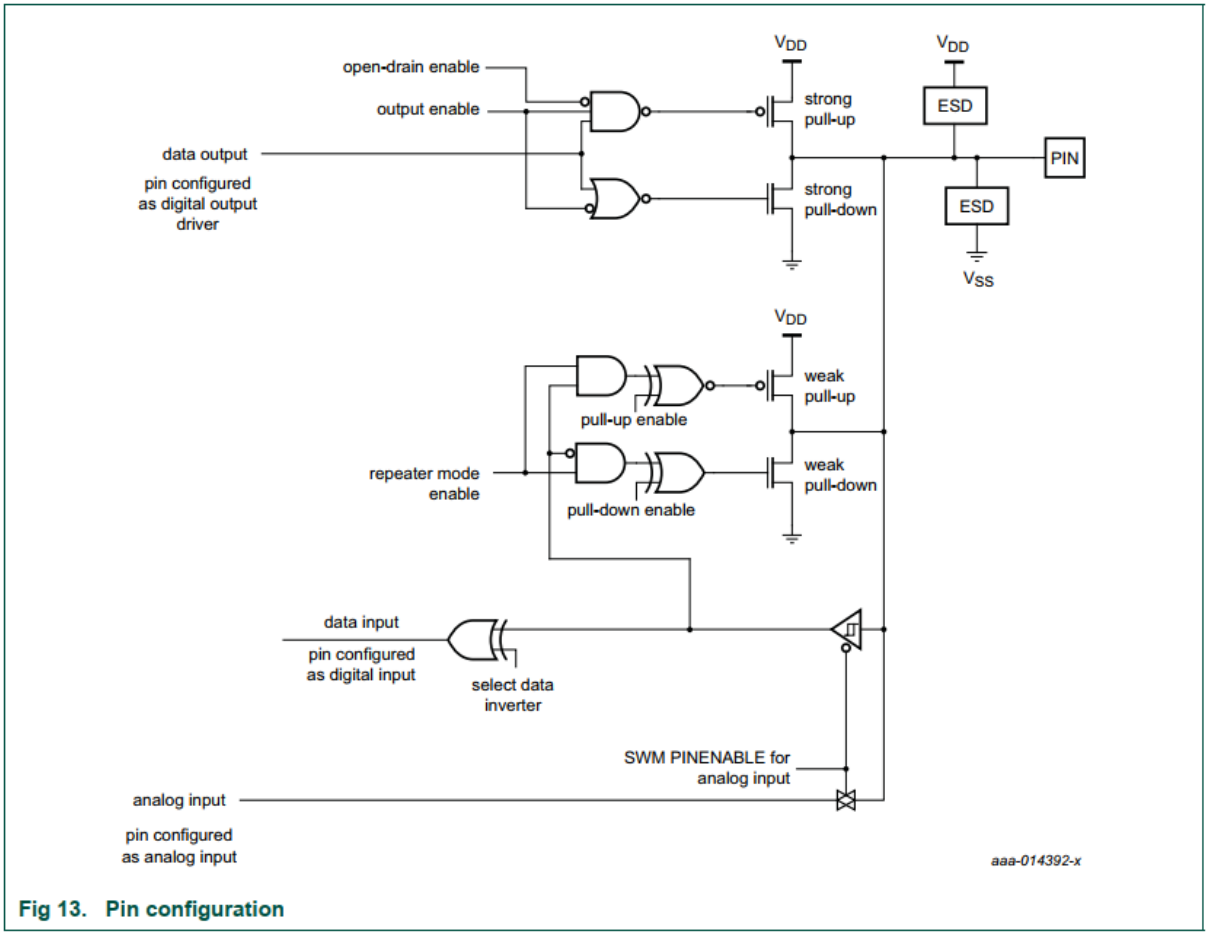


Fig 13. Pin configuration

Figure 4: Schéma électrique interne d'une broche, extrait du User Manual du LPC804 page 95

Par défaut les broches sont configurées en GPIO (nous verrons plus tard qu'elles peuvent aussi être utilisées par le convertisseur analogique numérique ou des périphériques de communication par exemple) et en entrée.

→ Expliquez le fonctionnement d'une broche en entrée en identifiant les composants utilisés par une couleur.

→ Expliquez le fonctionnement d'une broche en sortie en identifiant les composants utilisés par une autre couleur.

Pour passer en sortie, nous utilisons le registre contenant les signaux qui activent la sortie.

10.5.3 GPIO port direction registers

Each GPIO port has one direction register for configuring the port pins as inputs or outputs.

Table 140. GPIO direction port register (DIR0), address 0xA000 2000 bit description

Bit	Symbol	Description	Reset value	Access
30:0	DIRP	Selects pin direction for pin PIO0_n (bit 0 = PIO0_0, bit 1 = PIO0_1, ..., bit 30 = PIO0_30). 0 = input. 1 = output.	0	R/W
31	-	Reserved	0	-

Figure 5: Le registre DIR0, extrait du User Manual du LPC804, page 118

Le registre est évidemment défini dans lpc8xx.h.

→ Vérifiez que l'adresse donnée dans lpc8xx.h correspond bien à l'adresse du registre.

→ Expliquez alors la ligne suivante :

```
LPC_GPIO_PORT->DIR0 |= 0x00000800;
```

2.2.3. Recopie de l'état du bouton sur la led

Il y a plusieurs moyens d'accéder aux valeurs des entrées sur ce microcontrôleur, par bit, byte ou mot. Nous utiliserons les bytes.

On trouve l'état de PIO0_13 dans le registre B[13] et on change l'état de PIO0_11 en écrivant dans le registre B[11].

10.5.1 GPIO port byte pin registers

Each GPIO pin has a byte register in this address range. Bytes read in this range will be 0 if the pin is LOW or 0x01 if the pin is HIGH, regardless of direction, masking, or alternate function, except that pins configured as analog I/O always read as zeros. Writes will set or clear the pins output bit based on bit 0 of the byte written. Software typically reads and writes bytes to access individual pins, but can read or write halfwords to sense or set the state of two pins, and read or write words to sense or set the state of four pins.

Table 138. GPIO port byte pin registers (B[0:30], addresses 0xA000 0000 (B0) to 0xA000 001E (B30)) bit description

Bit	Symbol	Description	Reset value	Access
0	PBYTE	Read: state of the pin PIO0_n, at byte offset n, regardless of direction, masking, or alternate function, except that pins configured as analog I/O always read as 0. Write: loads the output bit of the pin.	external	R/W
7:1		Reserved (0 on read, ignored on write)	0	-

Figure 6: Les registres GPIO->B[0..30], extrait du User Manual du LPC804, page 117

→ Expliquez alors la ligne suivante :

```
LPC_GPIO_PORT->B0[11] = LPC_GPIO_PORT->B0[13];
```

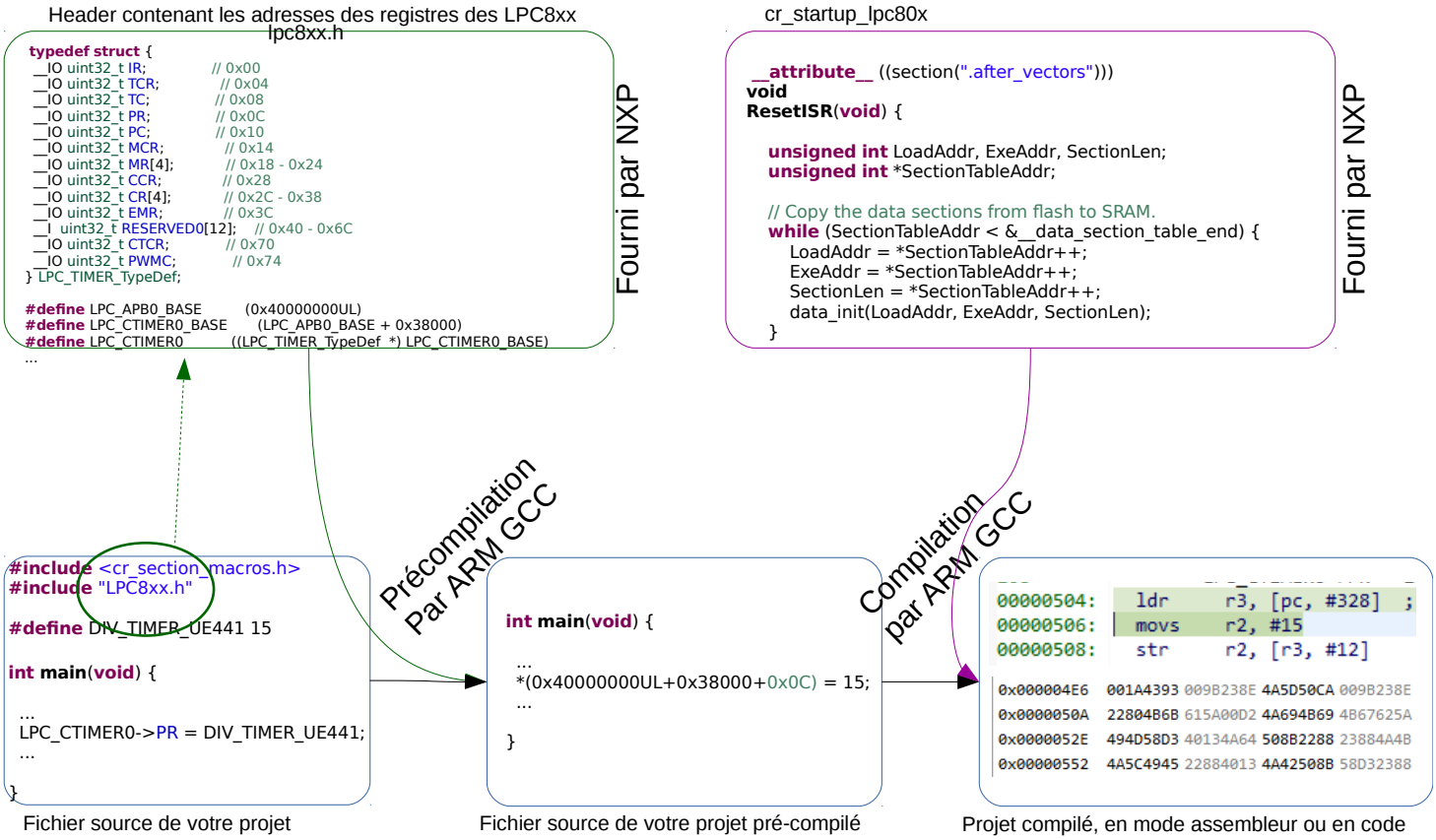
2.2.4. Compilation

Votre programme est complet, rajoutez l'incrémentation d'une variable i pour avoir une variable à surveiller.

```
int main(void) {
  static int i=0;
  LPC_SYSCON->SYSAHBCLKCTRL0 |= GPIO0;
  LPC_GPIO_PORT->DIR0 |= 0x00000800;
  while(1) {
        LPC_GPIO_PORT->B0[11] = LPC_GPIO_PORT->B0[13];
        i++;
  }
  return 0 ;
}
```

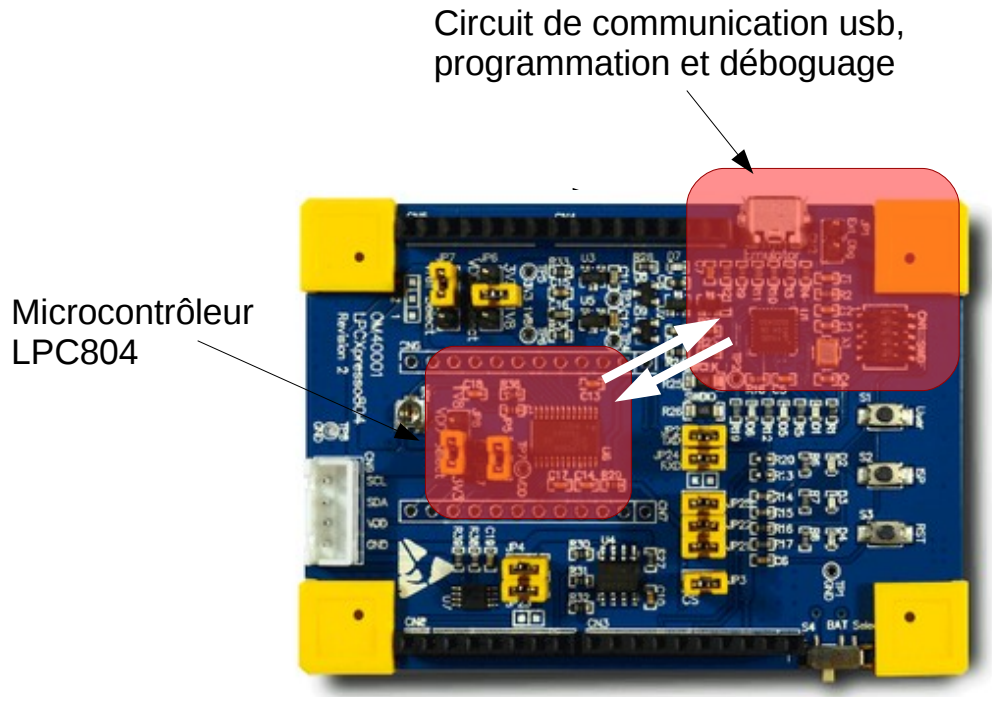
Vous pouvez alors le compiler en utilisant le raccourci *Build*.

On peut schématiser la compilation par le schéma suivant :



3. Debug du programme

Une fois votre programme compilé, vous pouvez l'envoyer dans le microcontrôleur et surveiller son exécution via le débogueur intégré sur la carte LPCXpresso804.



3.1. Programmation et vérification du bon fonctionnement

Branchez votre carte via le port USB puis cliquez sur le raccourci Debug pour envoyer le programme dans le microcontrôleur et établir une communication « de surveillance » entre le PC et le microcontrôleur.

Appuyez sur F8 (exécution) et vérifiez le bon fonctionnement du programme.

Si vous avez une alimentation usb (un chargeur de téléphone), vous pouvez vérifier que le programme est bien exécuté par le microcontrôleur, indépendamment de sa communication avec le PC.

3.2. Debug haut-niveau

Arrêtez le programme (*Run* > *Supend* ou icône « pause »).

Voyons comment observer le fonctionnement du programme.

Ajoutez un point d'arrêt par un double clic devant la première instruction de votre boucle infinie.

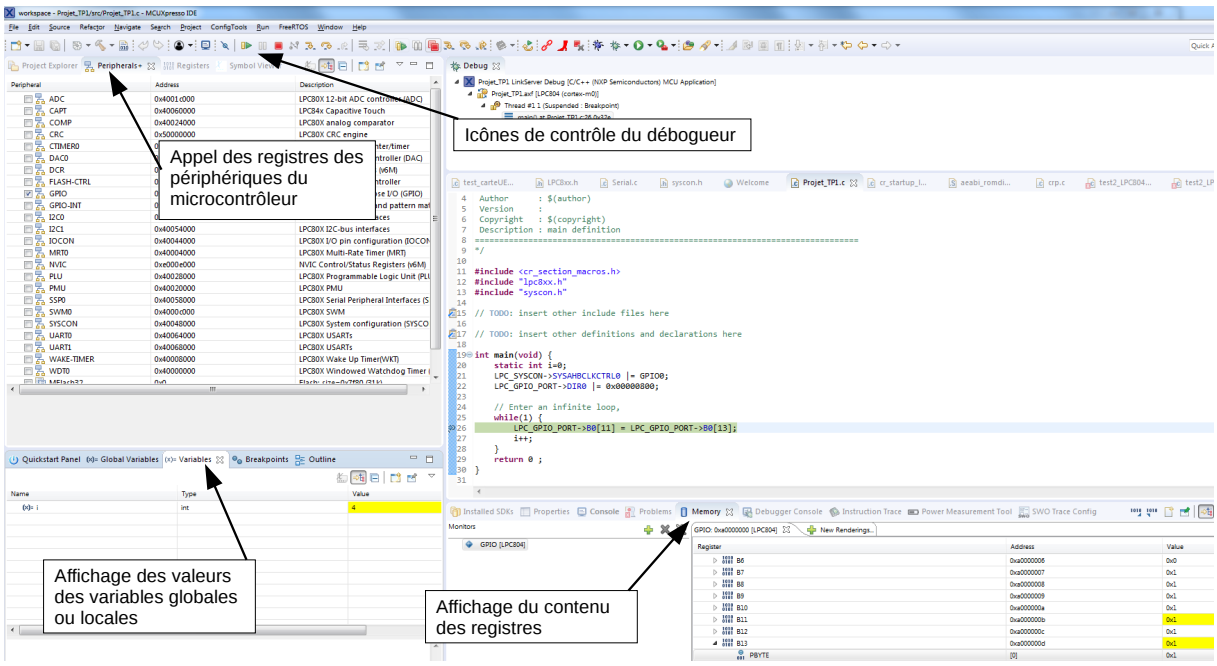
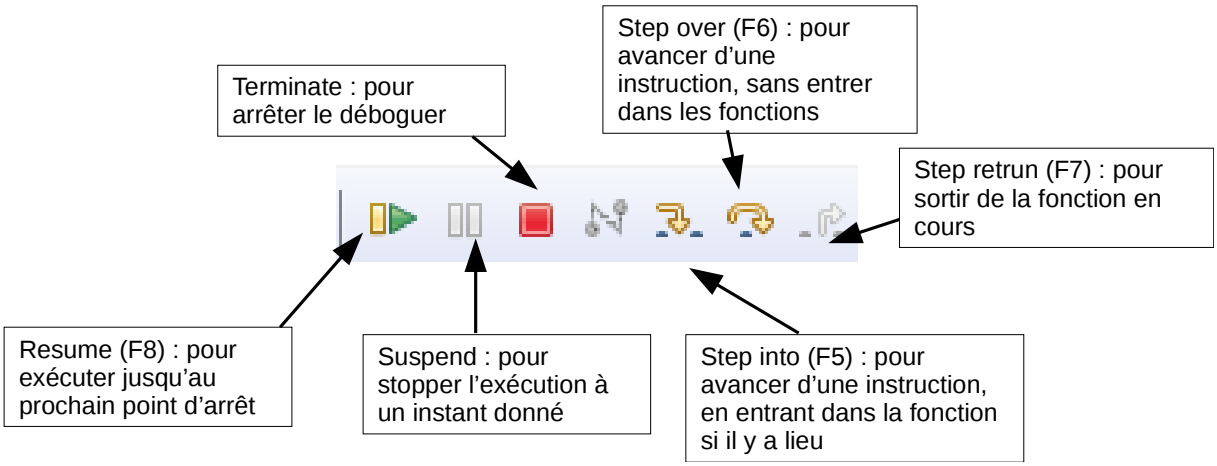


Figure 7: Perspective de déboguage

Utilisez les icônes de contrôle pour faire évoluer l'exécution de votre programme :



→ Observez l'évolution du registre B13 (noté B0[13] dans le programme) du périphérique GPIO. Quelle est son adresse dans le débogueur ? Cela correspond-il à ce qu'indique la documentation ?

3.3. debug niveau instruction

Jusqu'à présent, nous sommes resté au niveau du langage C, « haut niveau ». Il est également possible de faire du débogage au niveau des instructions processeur.

Cliquez sur l'icône Instruction Stepping Mode, en haut à droite 

Observez l'exécution de votre programme instruction processeur par instruction processeur.

→ Quel est le code assembleur (les instructions du processeur) qui correspond à votre boucle infinie.

→ Donnez alors le contenu de la mémoire programme correspondant à cette boucle infinie : dans la vue Memory, il est possible d'afficher le contenu d'une zone de la mémoire à l'aide de l'icône *Add Memory monitor :*

On donne le plan d'adressage mémoire du LPC804 :

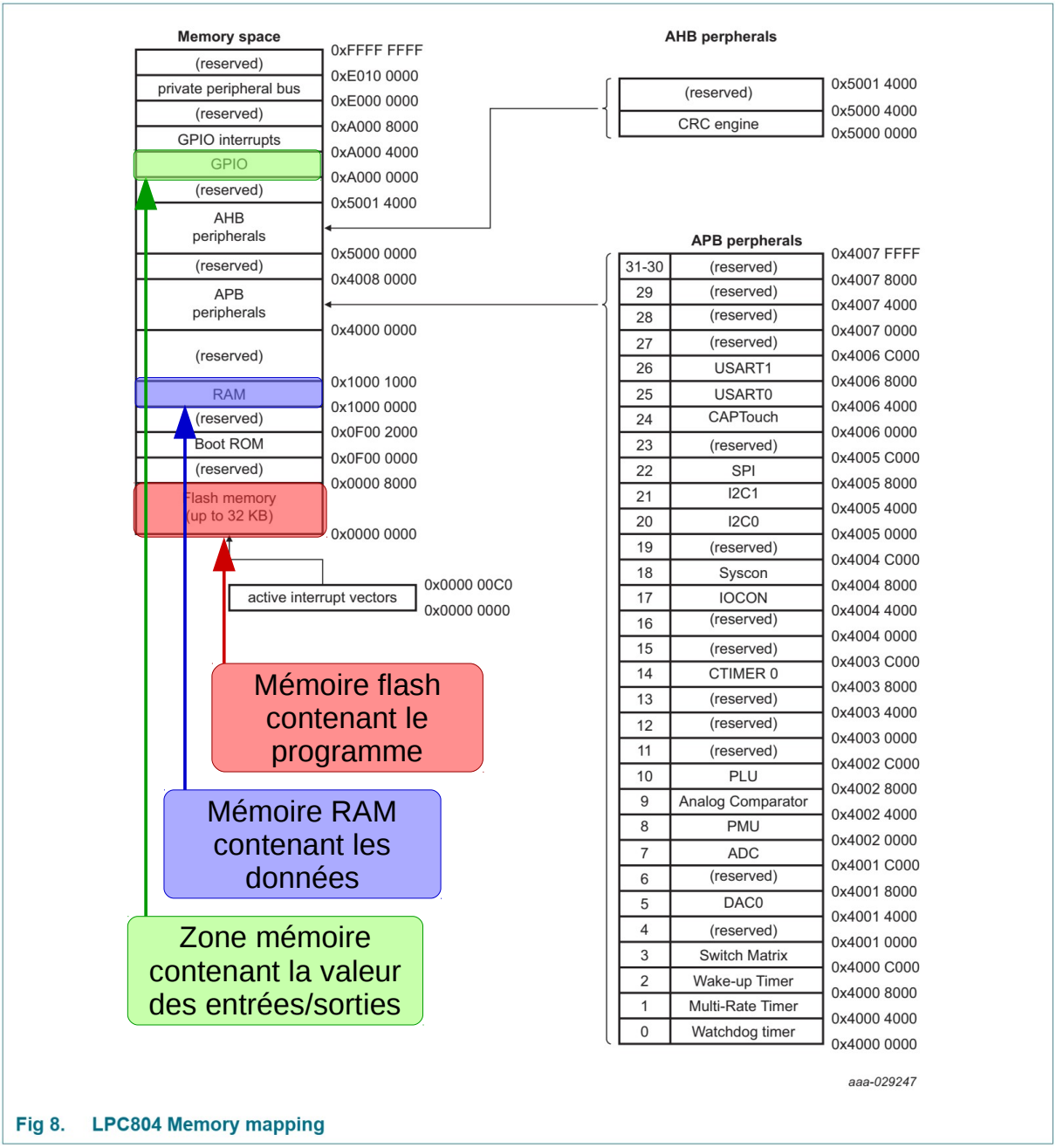


Figure 8: Plan d'adressage mémoir du LPC804

4. Création d'un projet avec des bibliothèques

Les fonctions sont incontournables en informatique embarquée :

- Pour améliorer la lisibilité du code, en limitant la fonction principale (main) à des appels de fonctions explicites. Le main devient alors lisible pour un non spécialiste.

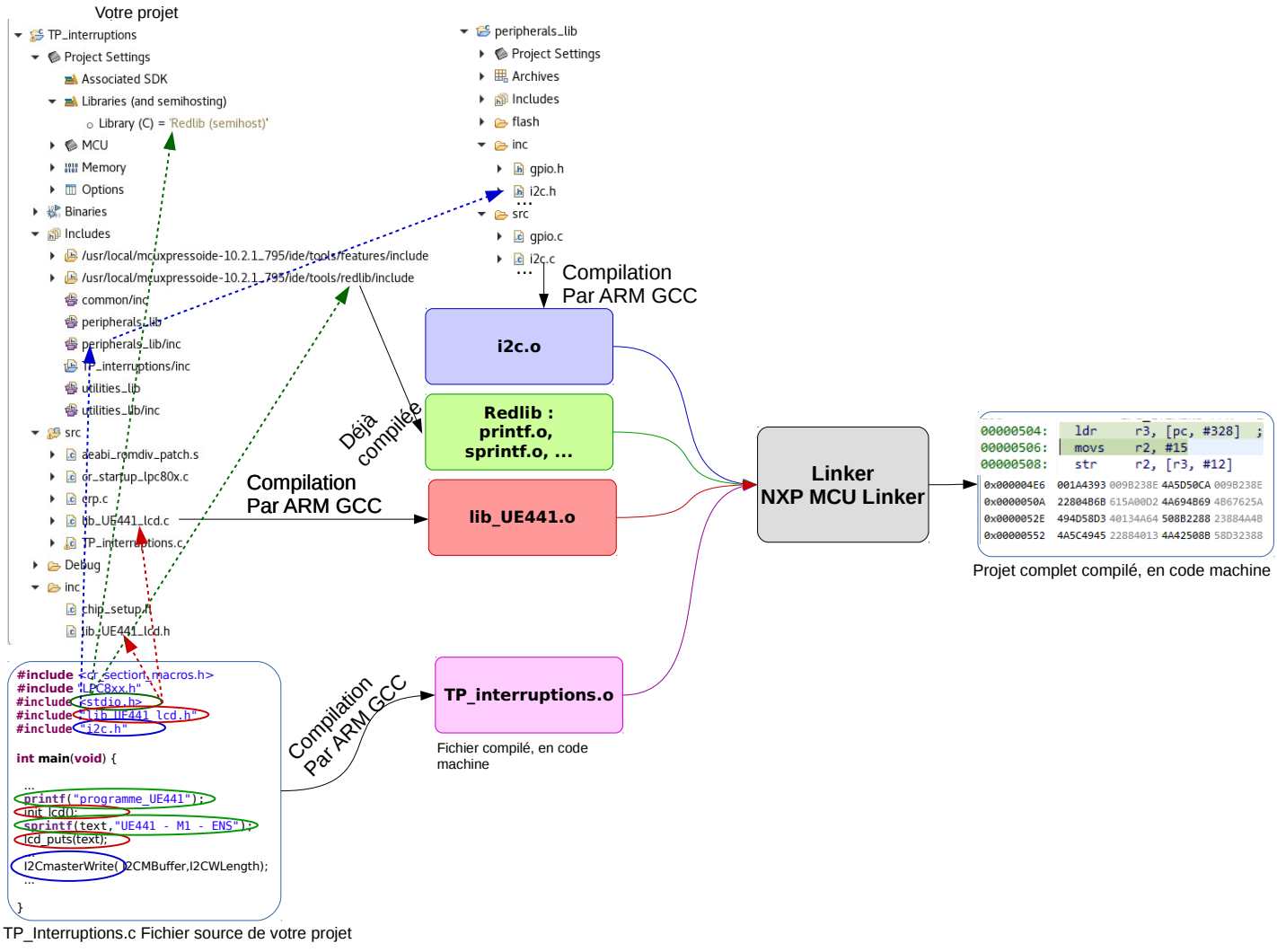
- Pour réutiliser des éléments de code précédemment validés
  - par vous,
  - par un membre de votre équipe (bibliothèque *lib_UE441_lcd.h*),
  - par le fabricant (bibliothèques *i2c*, *spi* de nxp)
  - par des tiers (bibliothèque *stdio* [printf…] et *math* [sin, cos…])

Un projet consistant fait donc appel au moins à trois types de bibliothèques :

- Les bibliothèques standard du C (math.h, stdio.h…),
- Les bibliothèques du fabricant du microcontrôleur (i2c, timers,…)
- Les bibliothèques personnelles (afficheur lcd, capteur particulier…)

A celles-ci peuvent s'ajouter des bibliothèques propres à la carte de développement (BSP : Board Support Package) ou des bibliothèques développées pour un capteur ou un actionneur par des tiers.

Le linker est l'élément de la suite de compilation qui va permettre de faire le lien entre votre fichier source et les codes des fonctions qu'il appelle. Le projet final comprendra votre code source et les codes des fonctions utilisées :

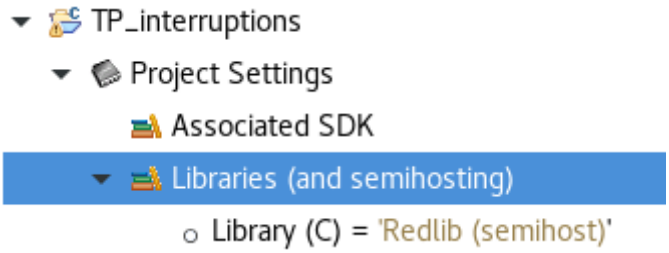


Dans votre projet, vous devez donc, pour le linker,

- indiquer la bibliothèque standard du C que vous utilisez (ici Redlib), et ajouter un lien dans vos programmes qui l'utilisent : #include <stdio.h> (et #include <math.h>...)
- indiquer un lien vers peripherals_lib et faire un lien vers les bibliothèques dans vos programmes qui les utilisent (#include « i2c.h »)
- inclure l'en-tête (lib_UE441.h) et la source (lib_UE_441.c) de votre ou vos bibliothèque(s) personnelles.

4.1. Les bibliothèques standard du C

Dans votre projet, un clic droit sur *Project Settings* > *Libraries* vous permet de choisir le paquet de bibliothèques standard que vous souhaitez.



Plusieurs paquets sont proposés. Certains (ceux notés semihost) proposent une sortie vers la console de MCUXpresso en mode debug via la fonction printf. Le paquet *Redlib-semihost* est complet (printf, math.h), avec des occupations mémoire de fonctions relativement compactes (math.h est en simple précision par exemple).

4.2. Les bibliothèques des périphériques

NXP propose un certain nombre de bibliothèques pour les périphériques du LPC804 et beaucoup plus pour ses microcontrôleurs plus complexes. Une bibliothèque pour le périphérique de communication i2c (communication avec le lcd notamment) est fourni.

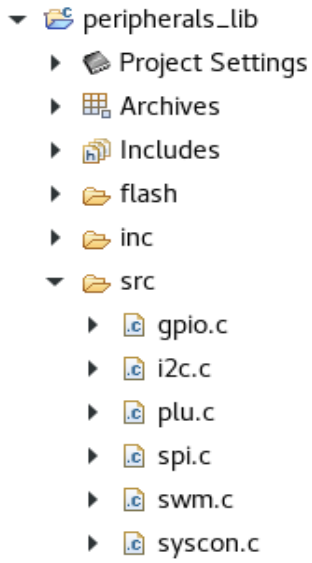
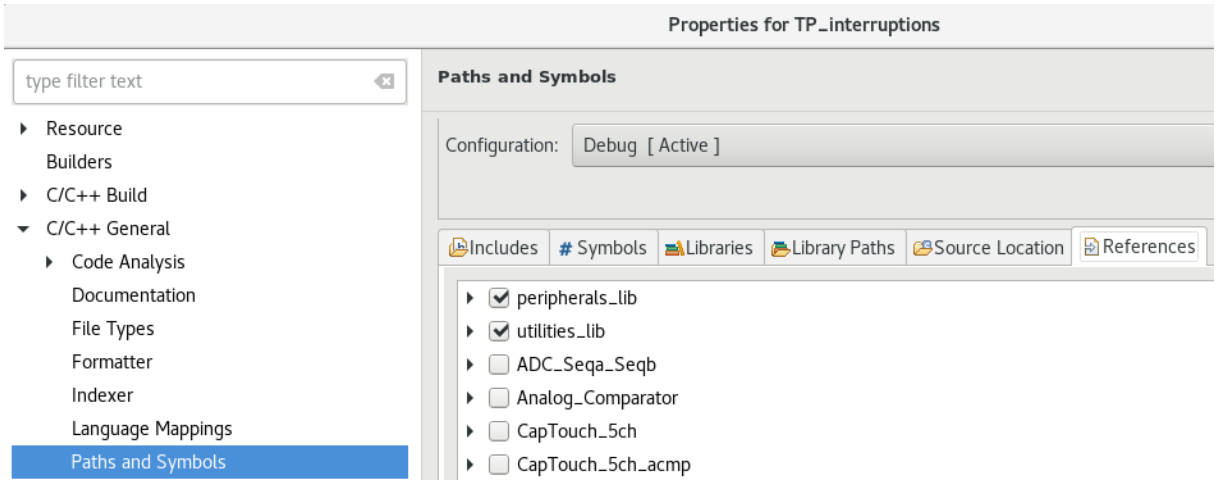


Figure 9: Les bibliothèques de peripherals_lib

Pour utiliser ces bibliothèques, il faut ajouter le projet peripherals_lib aux bibliothèques de votre projet. Ainsi, si vous utilisez la fonction i2cWriteMaster, le linker ajoutera le code extrait de i2c.o (résultat de la compilation de i2c.c) à votre projet compilé.
Pour ajouter des bibliothèques à votre projet,

- Ouvrez les *Propriétés* de votre projet (clic droit sur le nom du projet et *Propriétés*),
- Choisissez la rubrique *C/C++ General > Paths and Symbols*,
- Dans l'onglet *References*, ajoutez *peripherals_lib.* Profitez-en pour ajouter *utilities_lib* qui possède des fonctions pour utiliser la console de MCUXpresso.



4.3. Ajouter une bibliothèque personnelle

La bibliothèque *lib_UE441_lcd* vous est fournie sous forme de 2 fichiers :

- *lib_UE441_lcd.h* contient les prototypes des fonctions et quelques constantes.
- *lib_UE441_lcd.c* contient les définitions des fonctions.

Vous pouvez donc ajouter ces fichiers à votre projet comme des fichiers d'une bibliothèque personnelle que vous constitueriez pour ce projet uniquement.

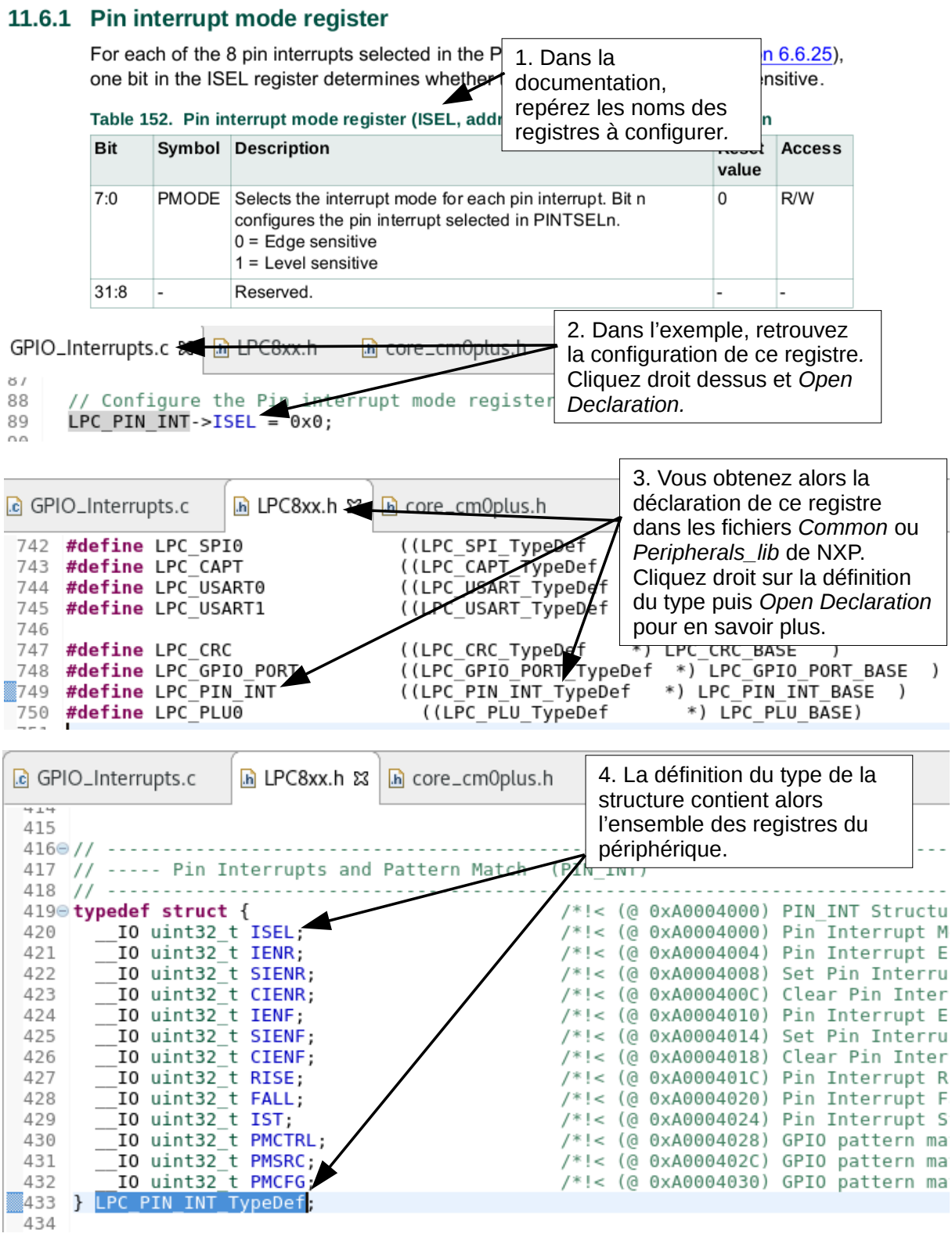
Téléchargez ces fichiers dans un dossier dont vous connaissez le chemin.
Dans votre projet, cliquez droit sur *src > Import...* puis dans la fenêtre d'importation choisissez *General > File System* et indiquez le dossier où se trouvent vos fichiers.

5. Configurer et utiliser un périphérique du microcontrôleur

La configuration d'un périphérique de microcontrôleur s'appuie sur la lecture du manuel de ce périphérique (LPC804 User Manual), la lecture des exemples fournis par NXP.
Par exemple, pour configurer les interruptions sur front d'une entrée sortie,

- Lisez les chapitres *Chapter 10: LPC804 General Purpose I/O (GPIO)* et *Chapter 11: LPC804 Pin interrupts/pattern match engine.*
- Ouvrez le projet d'exemple *GPIO_Interrupts* pour retrouver la dénomination exacte des registres utilisés.

- Cliquez droit puis *Open Declaration* pour voir où sont déclarés les registres où les fonctions :



On peut faire de même pour les fonctions utilisées par l'exemple :

The NVIC registers are located on the Arm private peripheral bus.

Table 39. Register overview: NVIC (base address 0xE000 E000)

Name	Access	Address offset	Description	Reset value	Reference
ISER0	RW	0x100	Interrupt Set Enable R… interrupts and reading … peripheral functions.	0	Table 40
-	-	0x104	Reserved.	-	-
ICER0	RW	0x180	Interrupt Clear Enable Register 0. This register allows disabling interrupts and reading back the interrupt enables for specific peripheral functions.	0	Table 41

1. Dans la documentation, repérez les noms des registres à configurer.

2. Dans le programme d'exemple, retrouvez la configuration des interruptions. Cliquez droit puis *Open Declaration* pour en savoir plus.

GPIO_Interrupts.c | LPC8xx.h | core_cm0plus.h

```
94   // Configure the IENF (pin interrupt enable
95   LPC_PIN_INT->IENF = 0x3;
96
97   // Clear any pending or left-over interrupt flags
98   LPC_PIN_INT->IST = 0xFF;
99
100  // Enable pin interrupts 0 - 1 in the NVIC (see core_cm0plus.h)
101  NVIC_EnableIRQ(PININT0_IRQn);
102  NVIC_EnableIRQ(PININT1_IRQn);
```

3. Vous trouvez alors des détails sur la fonction, propre au coeur Cortex M0+, et aux registres qu'elle utilise.

GPIO_Interrupts.c | LPC8xx.h | core_cm0plus.h

```
605 /** \brief  Enable External Interrupt
606
607     The function enables a device-specific interrupt in the NVIC interrupt controller.
608
609     \param [in]      IRQn  External interrupt number. Value cannot be negative.
610  */
611 __STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
612 {
613   NVIC->ISER[0] = (1 << ((uint32_t)(IRQn) & 0x1F));
614 }
```

GPIO_Interrupts.c | LPC8xx.h | core_cm0plus.h

```
276 typedef struct
277 {
278   __IO uint32_t ISER[1];                 /*!< Offs
279        uint32_t RESERVED0[31];
280   __IO uint32_t ICER[1];                 /*!< Offs
281        uint32_t RSERVED1[31];
282   __IO uint32_t ISPR[1];                 /*!< Offs
283        uint32_t RESERVED2[31];
284   __IO uint32_t ICPR[1];                 /*!< Offs
285        uint32_t RESERVED3[31];
286        uint32_t RESERVED4[64];
287   __IO uint32_t IP[8];                   /*!< Offs
288 } NVIC_Type;
```

Table 40. Interrupt Set Enable Register 0 register (ISER0, address 0xE000 E100)

Bit	Symbol	Description
0	ISE_SPI0	Interrupt enable.
1	-	Res
2	DAC0	Inte
3	ISE_UART0	Interrupt enable.
4	ISE_UART1	Interrupt enable.
23	ISE_CT32b0	Interrupt enable.
24	ISE_PININT0	Interrupt enable.
25	ISE_PININT1	Interrupt enable.

1. La documentation vous informe sur les interruptions possibles pour le LPC804.

GPIO_Interrupts.c | LPC8xx.h | core_cm0plus.h | MCUXpresso_cr_startup.c

```
35
36 //
37 // Pin Interrupt 1 ISR
38 //
39 void PININT1_IRQHandler(void)
40 {
41   if (LPC_PIN_INT->RISE & (1<<1)) {
42     LED_On(LED_GREEN);              // Rising edge on PIN INT1, LED on
43     LPC_PIN_INT->RISE = 1<<1;       // Clear the interrupt flag
44     return;
45   }
46   if (LPC_PIN_INT->FALL & (1<<1)) {
47     LED_Off(LED_GREEN);             // Falling edge on PIN INT1, LED off
48     LPC_PIN_INT->FALL = 1<<1;       // Clear the interrupt flag
49   return;
50   }
51 }
```

2. Les fonctions d'interruptions sont des fonctions particulières, dont le nom et l'adresse sont déjà définies. Cliquez droit puis *Open Declaration* pour en savoir plus.

GPIO_Interrupts.c | LPC8xx.h | core_cm0plus.h | MCUXpresso_cr_startup.c

```
109 void ADC_OVR_IRQHandler(void)    ALIAS(IntDefaultHandler);
110 void CTIMER0_IRQHandler(void)    ALIAS(IntDefaultHandler);
111 void PININT0_IRQHandler(void)    ALIAS(IntDefaultHandler);
112 void PININT1_IRQHandler(void)    ALIAS(IntDefaultHandler);
113 void PININT2_IRQHandler(void)    ALIAS(Int
114 void PININT3_IRQHandler(void)    ALIAS(Int
115 void PININT4_IRQHandler(void)    ALIAS(Int
116 void PININT5_IRQHandler(void)    ALIAS(Int
117 void PININT6_IRQHandler(void)    ALIAS(IntDefaultHandler);
118 void PININT7_IRQHandler(void)    ALIAS(IntDefaultHandler);
```

3. Le code de démarrage du projet vous affiche alors la liste des fonctions d'interruptions déclarées.

Vous trouvez également les informations sur les fonctions d'interruptions :

6. Raccourcis bien utiles

Pour mettre ou ne plus mettre en commentaire un tronçon de code sélectionné : Shift+Ctrl+C ou *Source* > *Toggle Comment*.

Pour trouver la déclaration d'une variable ou la définition d'une fonction : sélectionnez cette variable puis cliquez droit dessus puis *Open Declaration*.

7. Résolution des problèmes

Win32 error 0

```
C:\nxp\MCUXpressoIDE_10.2.1_795\ide\msys\bin\make.exe: *** Couldn't reserve space for cygwin's heap, Win32 error 0
```

Il faut changer le fichier \ide\msys\bin\msys-1.0.dll dans le dossier du logiciel par le fichier msys-1.0-alternate.dll que l'on trouve dans le même dossier. On renomme le premier msys-1.0.old et on donne le nom msys-1.0.dll au second.

Mémoire Flash non accessible

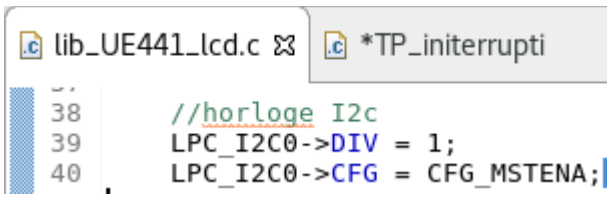
Votre programme a du invalider l'autorisation d'écrire sur la mémoire Flash (mémoire programme). Il faut donc autoriser matériellement cet accès : appuyer sur Reset et ISP (BP2 sur l'extension de la carte LPC804) et relâcher le reset d'abord, puis ISP. La carte est de nouveau programmable.

Problème de lien avec la bibliothèque i2c

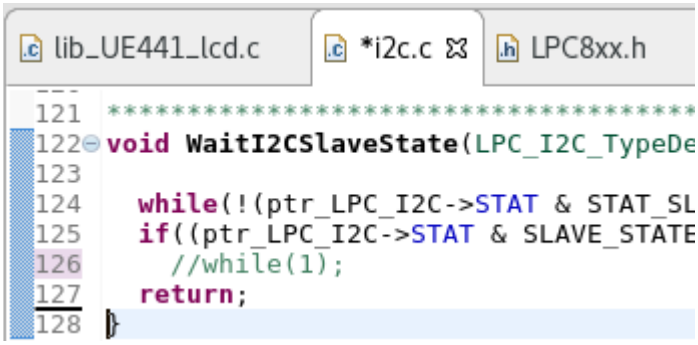
Le fichier i2c.h fourni par nxp appelle le fichier Lpc8xx.h. Il faut modifier ce nom de fichier par lpc8xx.h sans majuscule, Linux étant sensible à la casse dans les noms de fichier.

Problème d'initialisation de l'afficheur lcd

L'afficheur lcd fonctionne à 400 kHz. Il est donc bon de régler l'horloge du bus i2c à 350 kHz :



Le test sur la bibliothèque i2c de NXP est bloquant. Au premier appel, l'afficheur i2c peut bloquer. Il suffit de rendre la fonction non bloquante, en mettant en commentaire le blocage while(1) :



Registres non déclarés dans LPC8xx.h

Dans LPC8xx.h, les registres MSR (Shadow registers) ne sont pas déclarés. Vous pouvez modifier le fichier en rajoutant une ligne :

```
//------------------- Standard Counter/Timer (CTIMER) --
typedef struct {
  __IO uint32_t IR;                 // 0x00
  __IO uint32_t TCR;                // 0x04
  __IO uint32_t TC;                 // 0x08
  __IO uint32_t PR;                 // 0x0C
  __IO uint32_t PC;                 // 0x10
  __IO uint32_t MCR;                // 0x14
  __IO uint32_t MR[4];              // 0x18 - 0x24
  __IO uint32_t CCR;                // 0x28
  __IO uint32_t CR[4];              // 0x2C - 0x38
  __IO uint32_t EMR;                // 0x3C
  __I  uint32_t RESERVED0[12];      // 0x40 - 0x6C
  __IO uint32_t CTCR;               // 0x70
  __IO uint32_t PWMC;               // 0x74
  __IO uint32_t MSR[4];     // 0x78 -0x84 -> oublié !!
} LPC_TIMER_TypeDef;
```

De même dans Ctimer.h, les bits MR3RL validant l'utilisation des registres Shadow ne sont pas définis. Vous pouvez les rajouter.

```
// Match Control Register (MCR) shifters
#define MR0I   0
#define MR0R   1
#define MR0S   2
#define MR1I   3
#define MR1R   4
#define MR1S   5
#define MR2I   6
#define MR2R   7
#define MR2S   8
#define MR3I   9
#define MR3R   10
#define MR3S   11
#define MR0RL  24
#define MR1RL  25
#define MR2RL  26
#define MR3RL  27
```

Merci de signaler les problèmes : anthony.juton@ens-paris-saclay.fr

8. Bibliographie

Article Technique de l'ingénieur : Microcontrôleurs : principes et aspects temps réel
Auteur(s) : Roger D. HERSCH Date de publication : 10 déc. 2001

OpenClassRoom : Développez en C pour l'embarqué