Intervenants:
florent.bernard@univ-st-etienne.fr,
laetitia.belguermi@univ-st-etienne.fr,
jorge.hoyos.ramirez@univ-st-etienne.fr,
mathieu.descos@univ-st-etienne.fr,
alexandre.ortega@univ-st-etienne.fr

Cahier TP Info2

Vous avez besoin des documents suivants :
— Schéma de la carte de développement
— DataSheet du microcontrôleur MC9S08QG8
— Ce cahier de TP au format électronique
Ces documents sont accessibles sur *Moodle - Info2 - Documents de TP*.

Le logo signifie que l'information cherchée se trouve dans la DataSheet.

Le logo signifie qu'il faut faire particulièrement attention à ce qui est demandé !

Un compte rendu d'avancement est à rendre à la fin de chaque séance par mail à votre enseignant de TP.

Le compte rendu doit contenir les réponses aux questions posées dans le fascicule.

Ces questions sont précédées de la mention (CR) **:** alors que les manipulations à faire en TP (et qui ne font pas l'objet d'un compte rendu) sont précédées de la mention **Manip :**.
Très important : un **NOUVEAU** compte rendu est à rendre à chaque nouvelle séance et doit contenir uniquement les réponses aux questions traitées pendant la séance.
Ne surtout pas reprendre un ancien compte rendu et le compléter.

Pour certaines questions, vous devez compléter les réponses directement sur votre cahier dans les encadrés correspondants [ ].
Ces réponses ne font pas l'objet d'un CR mais sont utiles pour avancer sur les **Manip**
Enfin, certaines manipulations sont à faire viser par un enseignant qui doit signer et noter le numéro de séance et le temps mis depuis le début de la séance.

Elles sont repérées par un encadré spécifique :

Séance/heure	:
Signature	:

Merci à tous les étudiants ayant contribué, par leurs questions, leurs compte-rendus de qualité ainsi que leur implication, à la création et à l'amélioration de ce document et merci aux collègues intervenants pour leurs remarques constructives et pertinentes.

Table des matières

1 La carte de développement **5**
1.1 Compréhension de la carte de développement 5
1.1.1 Carte légendée 5
1.1.2 Microcontrôleur légendé 6
1.1.3 Pattes reliées aux LEDs 8
1.1.4 Pattes reliées aux autres composants 8
1.1.5 Caractéristiques de chaque élément 9
1.2 Fonctionnement de la carte de développement 11
1.3 L'élément LP2950 - 3v 11
1.4 Connexion au PC 11
1.5 Lecture de la DataSheet 12
1.5.1 Caractéristiques 13
1.5.2 Configuration des ports d'entrées/sorties 13

2 CodeWarrior en mode Full Chip Simulation **15**
2.1 Prise en main de CodeWarrior 16
2.1.1 Lancer code-warrior 16
2.1.2 Créer un nouveau projet 16
2.1.3 Exploration des fichiers présents 19
2.1.4 Générer l'exécutable 24

3 Allumage de LEDs **27**
3.1 "Allumage" de LED en Full Chip Simulation 28
3.2 Simple allumage 34
3.3 Allumage avec utilisation du bouton poussoir 34
3.4 Allumage avec utilisation des interrupteurs 35

4 Clignotement des LEDs **37**
4.1 Taille des types de données 38
4.2 Compréhension d'un code proposé 39
4.3 Création d'une procédure de temporisation 40

5 Interruption sur le timer **43**
5.1 Utilisation d'une interruption sur le timer 44
5.1.1 Introduction au mécanisme d'interruption 44
5.2 Exercices de synthèse 48

6 [Optionnel] Modulation de Largeur d'Impulsion 53
6.1 PWM : premières notions (à préparer) 53
6.2 Pulse-Width Modulator 55
6.3 Contrôler l'intensité de la LED : utilisation d'un potentiomètre et d'un CAN 56

A Pense-bête 61
A.1 Définitions relatives à l'IDE 61
A.2 Outils 62
A.3 Errors et Warnings 64

Chapitre 1

La carte de développement

Objectif

L'objectif est de prendre connaissance de la carte de développement que nous allons utiliser et sur laquelle est placé le microcontrôleur.
Vous devez lire cette partie qui présente les différents composants présents sur cette carte et leurs rôles et y revenir en détail lorsque vous voudrez des informations plus détaillées.

1.1 Compréhension de la carte de développement

Le schéma original de la carte de développement se trouve ici.
Celle de la carte légendée est présentée sur les figures 1.1 (photo) et 1.2 (schématique).

1.1.1 Carte légendée

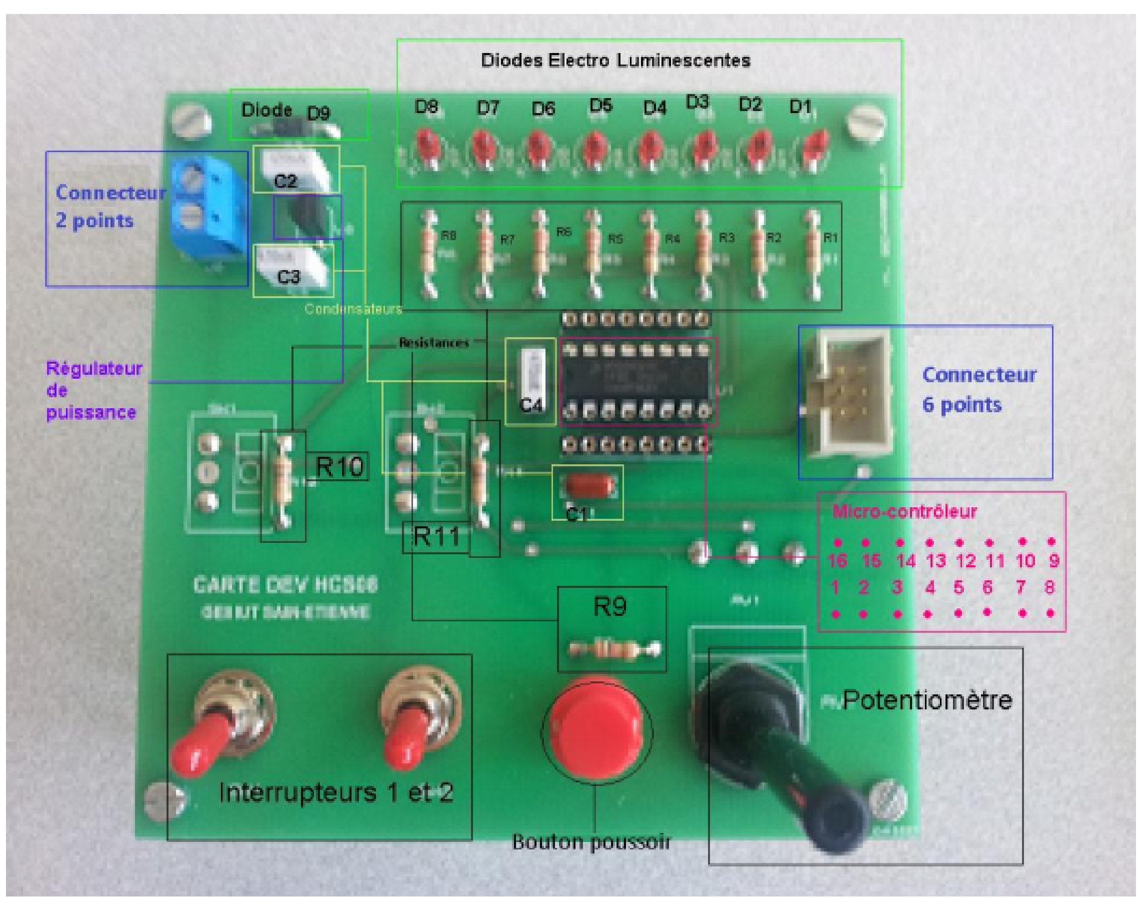
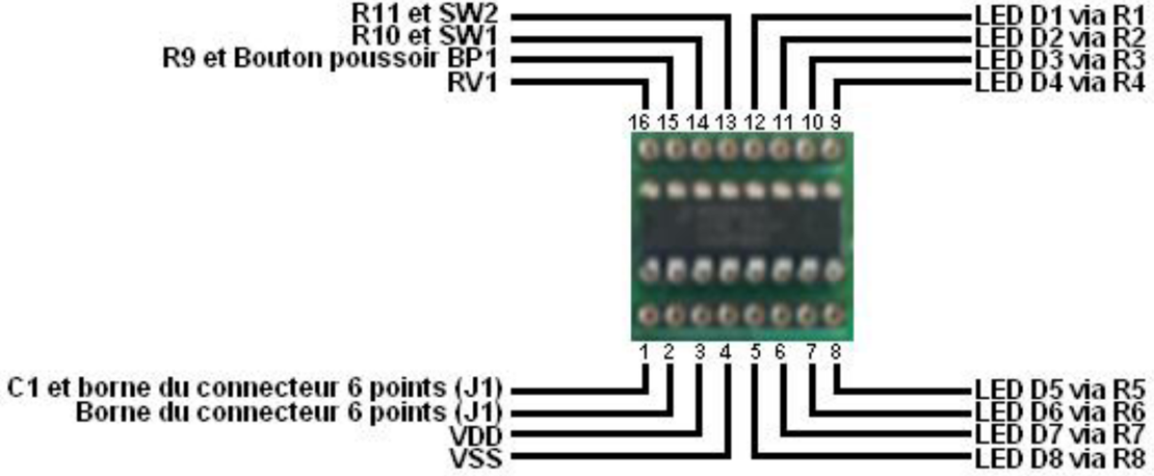


Figure 1.1 – Photographie de la carte avec repérage des différents éléments la constituant.

1.1.2 Microcontrôleur légendé



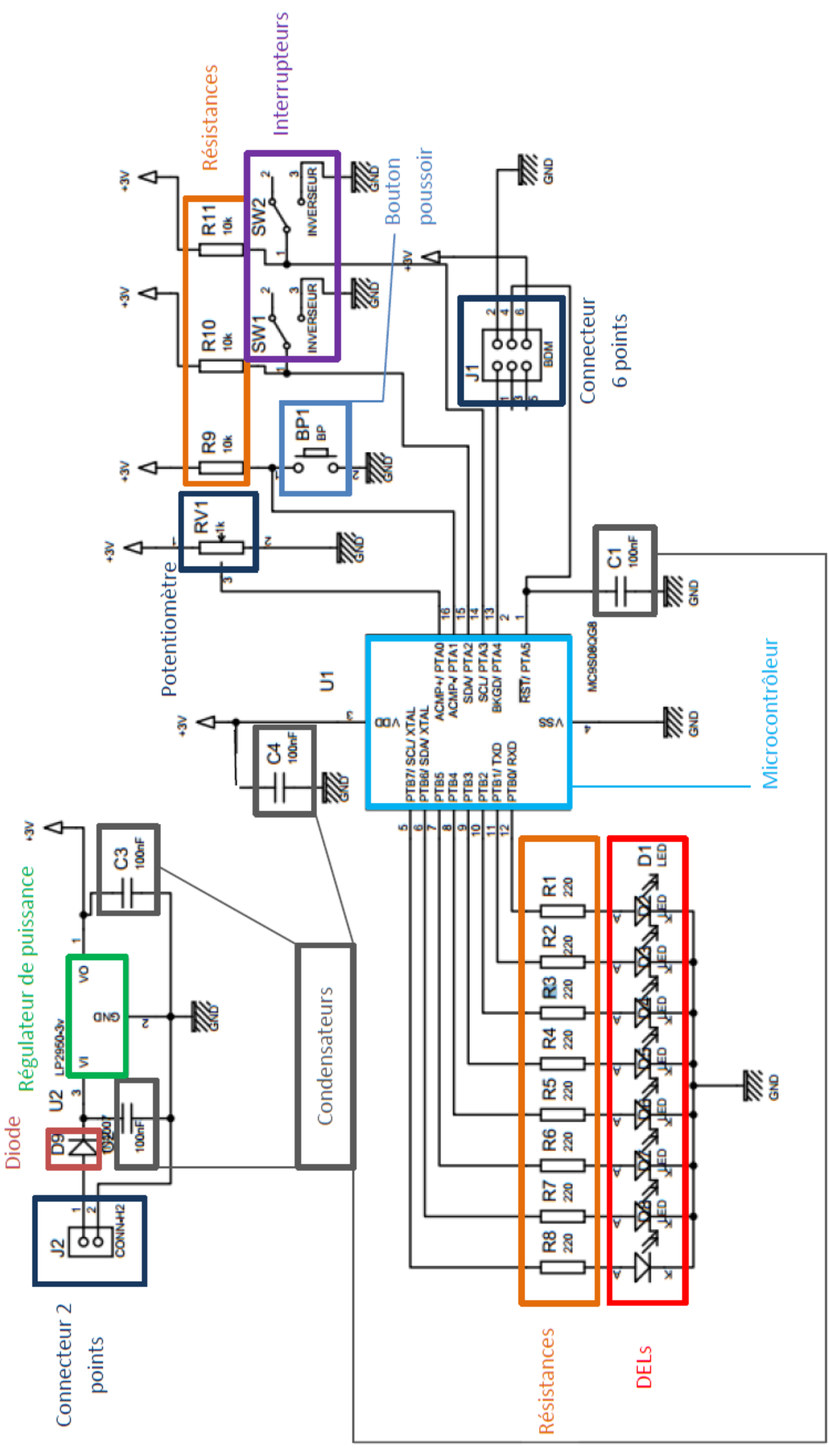
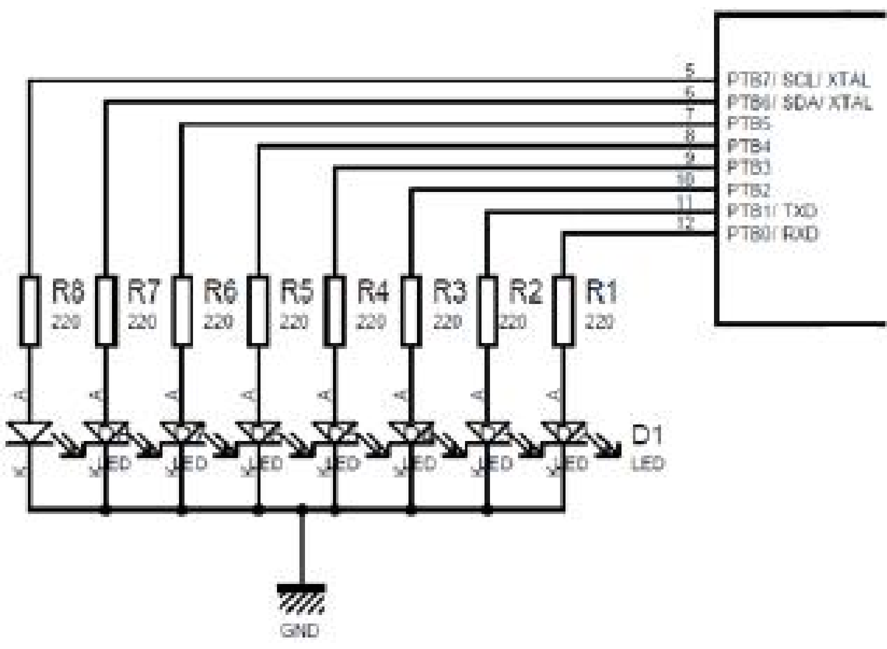


Figure 1.2 – Schématique légendée de la carte de développement

1.1.3 Pattes reliées aux LEDs



1.1.4 Pattes reliées aux autres composants

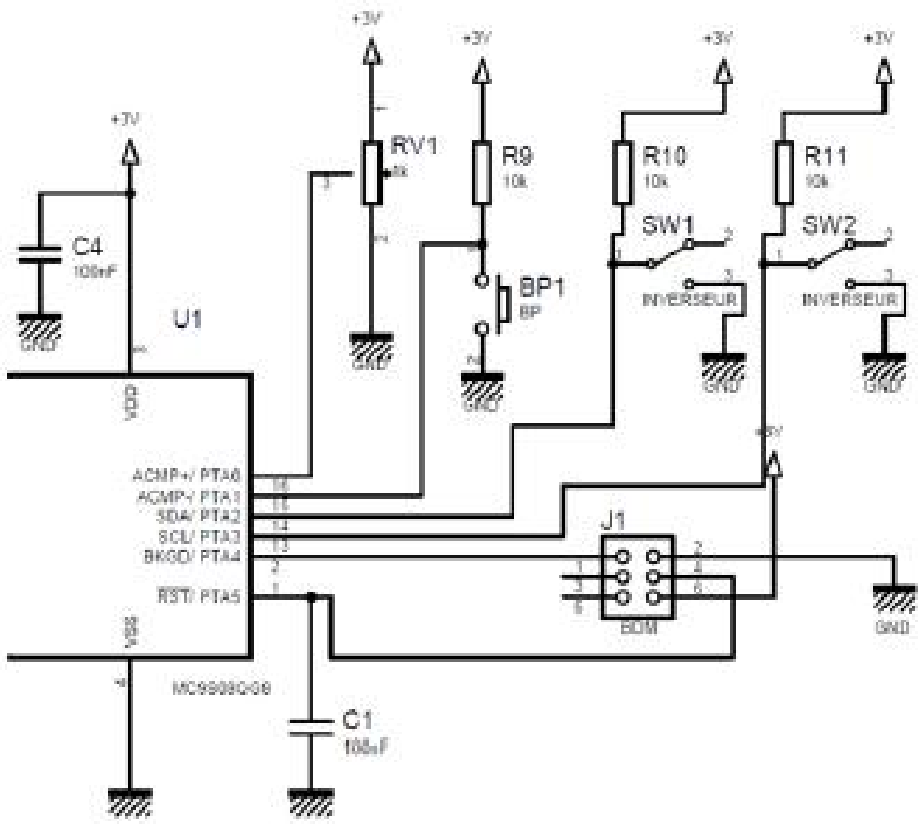
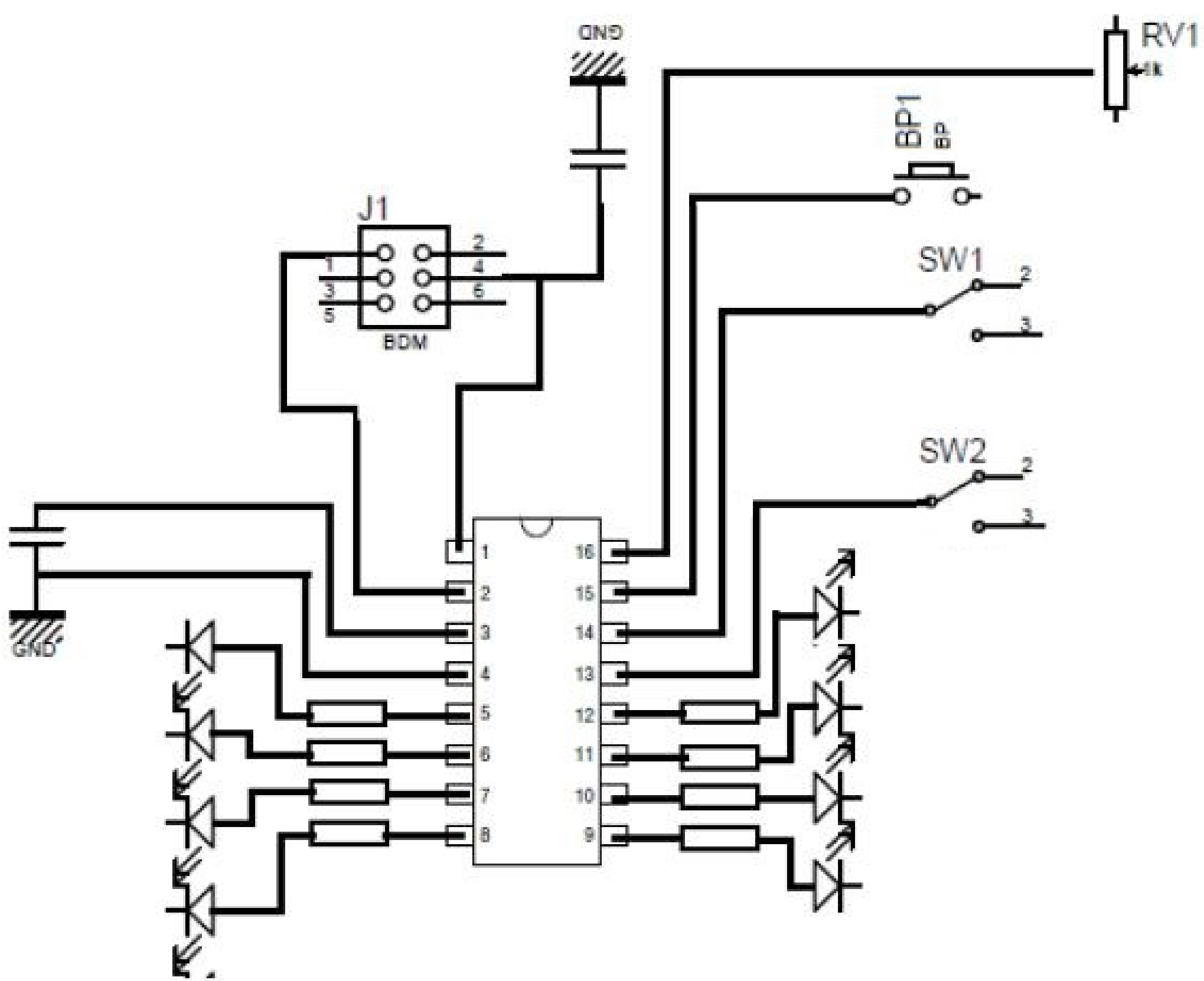


Schéma de la carte réordonné



1.1.5 Caractéristiques de chaque élément

Les Diodes ElectroLuminescentes

	Caractéristiques	Rôle	Intérêt
D1-D8	Diodes ElectroLuminescentes rouges	Ce sont les interfaces de sortie du microcontrôleur	Elles permettent de visualiser l'état (0 ou 1) de chaque sortie du microcontrôleur. Les DEL s'allument à l'état logique 1.
D9	Diode 1N4007	Fusible/Régulation de la tension	

Remarque : Cette configuration est valable sur CETTE carte.

Les capacités

	Valeur	Rôle	Intérêt
C1	100 nF	Permettent de lisser la tension sur le circuit	Évitent les changements brusques de tension.
C2-C4	470 nF		

Les résistances

	Couleurs	Valeur	Rôle	Intérêt
R1-R8	Rouge Rouge Marron	220 Ω	Limitent le courant	Permettent de limiter le courant arrivant sur les LED afin qu'elle ne grillent pas.
	Or	Tolérance 5%		
R9-R11	Marron Noir Rouge	10 $k\Omega$		Limitent l'intensité du courant arrivant sur le microcontrôleur lorsque les interrupteurs et le bouton-poussoir sont à 1. (Le circuit est fermé).
	Or	Tolérance 5%		

Pour rappel voici le code couleur des résistances :

	Couleur	1er chiffre	2nd chiffre	Multiplicateur	Tolérance
	Noir	0	0	0	20 %
	Brun	1	1	10	1 %
	Rouge	2	2	100	2 %
	Orange	3	3	1 000	
	Jaune	4	4	10 000	
	Vert	5	5	100 000	
	Bleu	6	6	1 000 000	
	Violet	7	7		
	Gris	8	8	0,01	
	Blanc	9	9	0,1	
	Argent			0,01	10 %
	Or			0,1	5 %

Les autres composants

	Caractéristiques	Rôle	Intérêt
Connecteurs	2 points	Permet de brancher une alimentation externe.	Lorsque qu'on n'est pas en phase de développement (non connecté à l'ordinateur), permet à la carte d'être en autonomie.
	6 points	Sert de liaison avec l'ordinateur afin de programmer le microcontrôleur.	
Bouton poussoir	État stable par défaut (niveau logique 1).	Ce sont les interfaces d'entrée du microcontrôleur.	Permettent de contrôler et modifier l'état des entrées du microcontrôleur.
Interrupteur `SW1-SW2`	Possède deux états stables.		
Régulateur de puissance	Transistor LP2950	Limite la puissance de l'alimentation secondaire. Limite donc la tension	La carte doit être alimentée sous une certaine tension à ne pas dépasser.
Potentiomètre		Résistance variable	C'est une entrée du microcontrôleur qui peut agir par exemple sur l'intensité de la luminosité des DEL.

1.2 Fonctionnement de la carte de développement

Cette carte de développement est composée d'un microcontrôleur qui dispose d'une mémoire RAM de 512 Bytes et d'une mémoire FLASH de 8192 Bytes, d'entrées (les interrupteurs SW1, SW2, le bouton poussoir BP ainsi que le potentiomètre) et de sorties (les 8 LEDs numérotées de D1 à D8).
Elle peut être alimentée de deux façons différentes :

- par le connecteur 6 points qui sert aussi à programmer le microcontrôleur grâce à un ordinateur.
- une alimentation secondaire (LP2950-3v).

Le microcontrôleur possède de la mémoire FLASH non volatile, ce qui permet un fonctionnement autonome du microcontrôleur grâce à l'alimentation externe.

1.3 L'élément LP2950 - 3v

L'élément LP2950-3v est représenté séparément sur le schéma car il sert à réguler la tension dans le cas d'une alimentation secondaire externe. Elle est utile lorsque le microcontrôleur est déjà programmé et que l'on veut exécuter un programme présent dans la mémoire FLASH sans être connecté avec l'ordinateur.

1.4 Connexion au PC

La carte est reliée au PC telle que présentée dans la figure 1.3.
La carte intermédiaire (USBDM), carte de programmation/débogage (voir figure 1.4) permet de programmer certains types de microcontrôleurs de chez freescale.Elle permet en particulier :

1. l'alimentation de la carte de développement,
2. l'envoi du programme en langage machine du PC (résultat de la compilation) vers la mémoire FLASH du microcontrôleur.

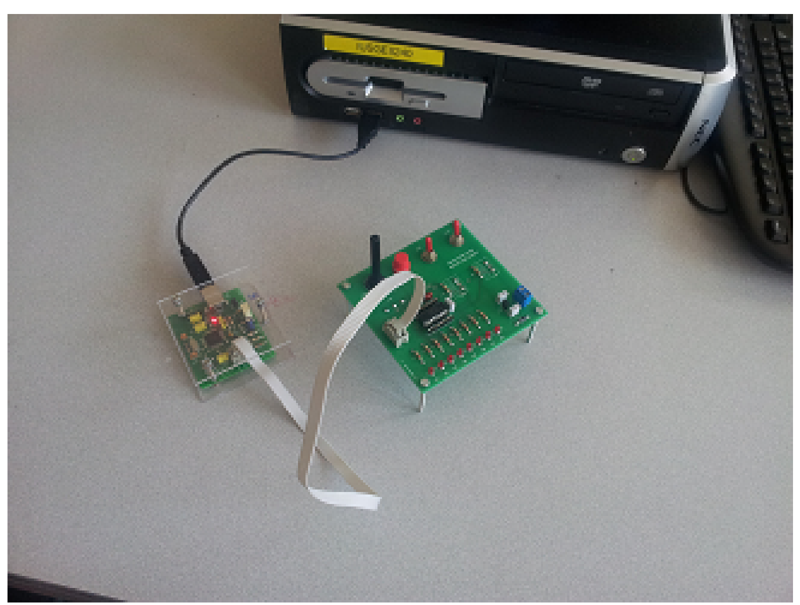


Figure 1.3 – Connexion de la carte de développement au PC via la liaison USBDM

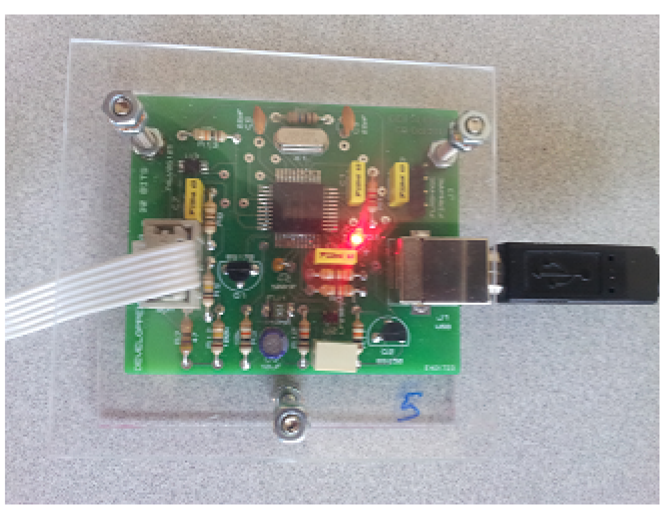


Figure 1.4 – Carte de programmation/débogage USBDM

Si la carte est bien reconnue par l'ordinateur, le voyant clignote deux fois (une fois rapidement, une fois lentement).

1.5 Lecture de la DataSheet

Ouvrir la documentation technique du microcontrôleur `MC9S08QG8.pdf` [1]
La documentation technique est volumineuse. Aussi, utilisez et abusez de la recherche numérique dans un document PDF (touche CTRL-F).

1. Le lien est cliquable dans la version électronique du document. Sinon cette documentation se trouve sur Moodle, rubrique Info2, Documents de TP.

1.5.1 Caractéristiques

Ce microcontrôleur possède deux types de mémoire :

— une mémoire ROM de type FLASH;

— une mémoire RAM.

Pour fonctionner, un microcontrôleur doit avoir un microprocesseur, une horloge, de la mémoire (mémoire vive RAM et mémoire morte ROM) et des périphériques d'entrée / sortie.
Le microprocesseur fonctionne a une fréquence d'environ 20 MHz. Compléter, directement sur le document les champs suivants.

Le bus d'adresse est de [ ] bits. (page 87 Chapitre 7)
Notre microcontrôleur possède un espace adressable de [ ] mots de [ ] bits.

On peut vérifier cette valeur grâce à la "Memory Map". (page 39)
Ce microcontrôleur possède 2 ports de 8 bits pour communiquer avec l'extérieur : le port A et le port B. Ces ports (ou une partie seulement) peuvent être configurés en entrée ou en sortie.
Les différentes mémoires et le microprocesseur sont directement reliés **à l'intérieur** du microcontrôleur grâce aux bus d'adresses, de données et de contrôle.
Les périphériques d'entrées / sorties sont reliés au microcontrôleur grâce aux pattes du composant (pattes 5 à 12 pour le port B, les autres pour le port A).

1.5.2 Configuration des ports d'entrées/sorties

1. On doit configurer les pins (ou broches) du port B comme pins de sorties pour pouvoir éclairer les LEDs.
2. On doit configurer les pins du port A comme pins d'entrées pour pouvoir recevoir les informations venant des différents interrupteurs.

Exercices Chapitre 1

En analysant la documentation technique, répondez aux questions suivantes :

1. Le Port B (comme le Port A) sont constitués de deux registres de 8 bits : PTBD et PTBDD.
   (**CR1**) **:** Expliquez quelle est la différence entre le registre PTBD et le registre PTBDD [2] ?
2. A quelles adresses se trouvent respectivement PTBD[ ] et PTBDD[ ]?
3. A quelles adresses se trouvent respectivement PTAD[ ] et PTADD[ ]?
4. Entre quelles plages d'adresses se trouvent la RAM ? [ ]
   La mémoire FLASH ?[ ]
   (**CR2**) **:** Vérifiez que les capacités annoncées sont correctes en utilisant les valeurs des plages d'adresses pour la RAM et la FLASH.
5. Consultez les schématiques de la carte au début de ce cahier de TP pour repérer à quels composants sont reliés les ports A et B du microcontrôleur sur la carte.
   (**CR3**) **:** expliquez pourquoi sur cette carte, le port B doit être configuré en sortie et le port A en entrée ?
6. Donnez la valeur des registres permettant de configurer le Port A en entrée [ ]
   et le Port B en sortie [ ].

2. Faites une recherche dans le document sur PTBD et PTBDD pour obtenir leurs descriptions.

7. A l'issue de ces questions, vous devriez avoir compris comment on gère une patte d'un port du microcontrôleur. Cette question ne fait pas l'objet d'un compte rendu.
   Grace au schéma de la figure 1.5, vérifiez (en mettant au crayon de papier des valeurs dans les bascules de PTAD0 et PTADD0) que vous avez bien compris le rôle de chacun des registres PTADD et PTAD lors d'une communication du microcontrôleur avec l'exterieur (en entrée ou en sortie). Ce schéma est dupliqué 8 fois pour chaque patte d'un Port du microcontrôleur (A ou B).

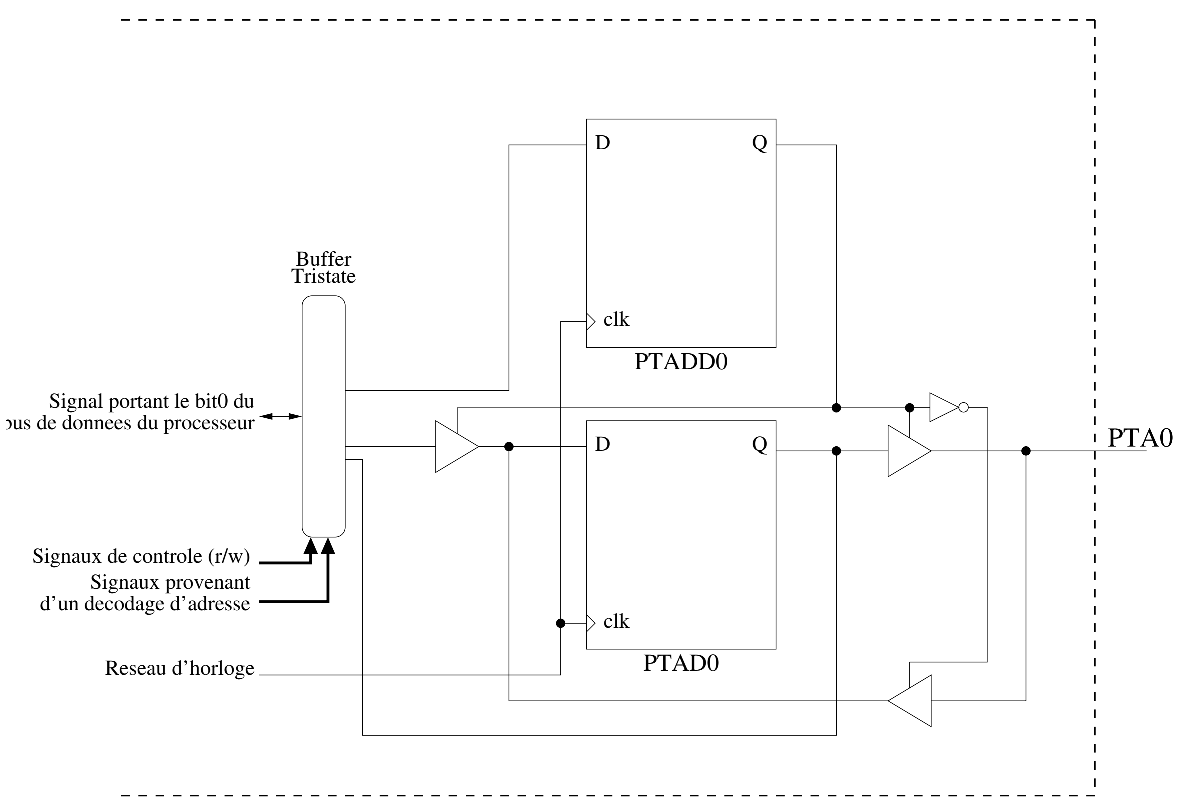


Figure 1.5 – Gestion d'une patte d'entrée sortie du microcontrôleur (ici PTA0)

Chapitre 2

CodeWarrior en mode Full Chip Simulation

Objectif

L'objectif de ce chapitre est de prendre en main l'IDE CodeWarrior en mode Full Chip Simulation [1] . Il vous permettra de créer votre premier projet sous CodeWarrior permettant de programmer le microcontrôleur, puis de le faire exécuter par le microprocesseur (exécution simple ou débogage), en mode simulation ou en mode exécution réelle.

CodeWarrior (64 bits) peut être récupéré via un dépôt mis à disposition sur Moodle, il faudra suivre ce lien [2].

Dans le précédent chapitre vous avez découvert la carte et fait le lien avec la DataSheet du microcontrôleur.

Dans ce chapitre, vous allez découvrir le logiciel permettant de générer du code pour le processeur du microcontrôleur et de l'envoyer dans la mémoire FLASH du microcontrôleur via la carte USBDM. Il permet aussi de faire le lien entre les déclarations spécifiques du microcontrôleur et la DataSheet.
Ce chapitre est principalement un tutoriel d'utilisation de CodeWarrior, à resuivre tant que la création de projet et l'utilisation de CodeWarrior n'est pas acquise.

Dans le prochain chapitre, vous ferez le lien entre ce logiciel et la carte de développement en faisant exécuter du code par le microcontrôleur que vous aurez programmé.

---

1. D'après :
   — `http://www.ief.u-psud.fr/~jok/iut/HC12/CW_HC12.pdf`
   — `http://meteosat.pessac.free.fr/IMA/ressources/Doc_C/Doc_HC12/TP_ii1_S1_GE1.pdf`

2. https://mood.univ-st-etienne.fr/mod/url/view.php?id=10520

2.1 Prise en main de CodeWarrior

Objectifs

— Créer un projet CodeWarrior,
— Editer les sources,
— Compiler les sources,
— Corriger les erreurs,
— Charger le programme,
— Utiliser le débogueur,
— Faire exécuter le programme en mode simulation.
— Tester les boutons.

Prérequis (cours Info1 et Info2)

— Identifier les éléments du langage C,
— Appeler une fonction avec des paramètres,
— Lire et écrire sur un port I/O avec des macros prédéfinies,
— Manipuler les bits d'un mot machine avec des masques.

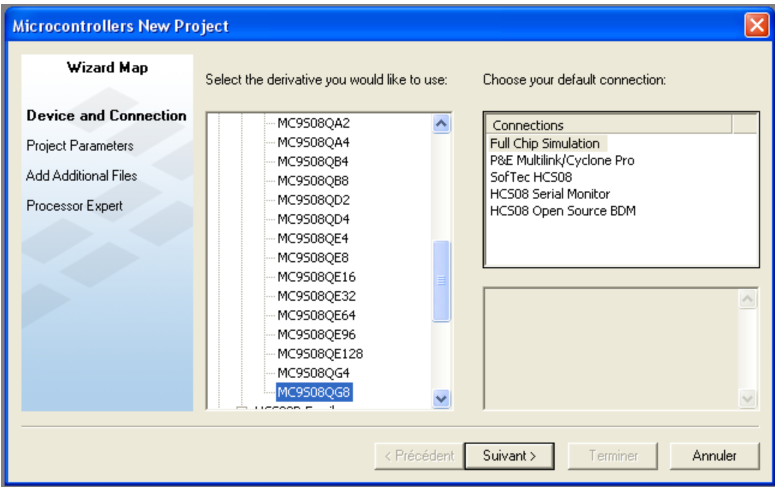
Nous allons créer notre premier projet grâce à l'IDE CodeWarrior (**Manip**).

2.1.1 Lancer code-warrior

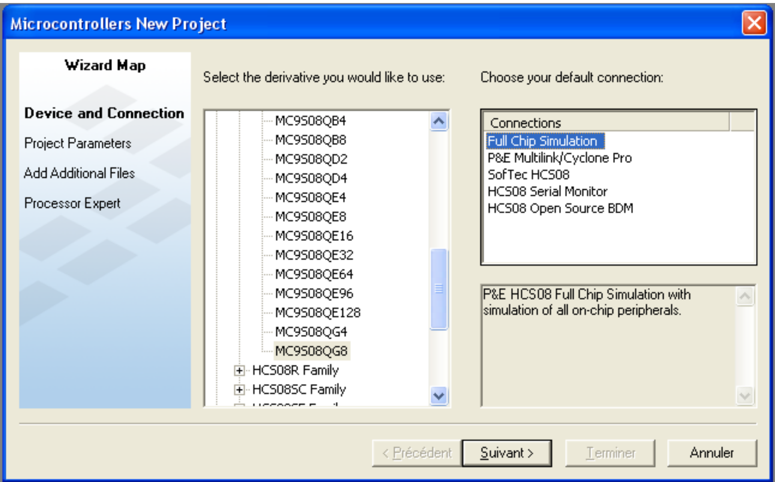
`Menu Demarrer->Programmes->Freescale->Codewarrior MOT V1.2->CodeWarrior`

2.1.2 Créer un nouveau projet

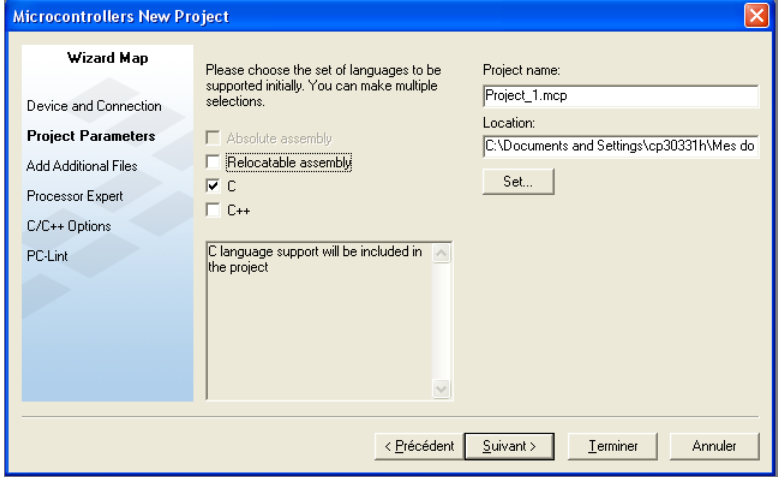
Cliquer sur `Create New Project`.
Choisissez la famille `HCS08Q` puis le microcontrôleur `MC9S08QG8`



Pour **ce TP**, on choisira le mode `Full Chip Simulation`. Pour les suivants, on utilisera en plus la connection `HCS08 Open Source BDM`

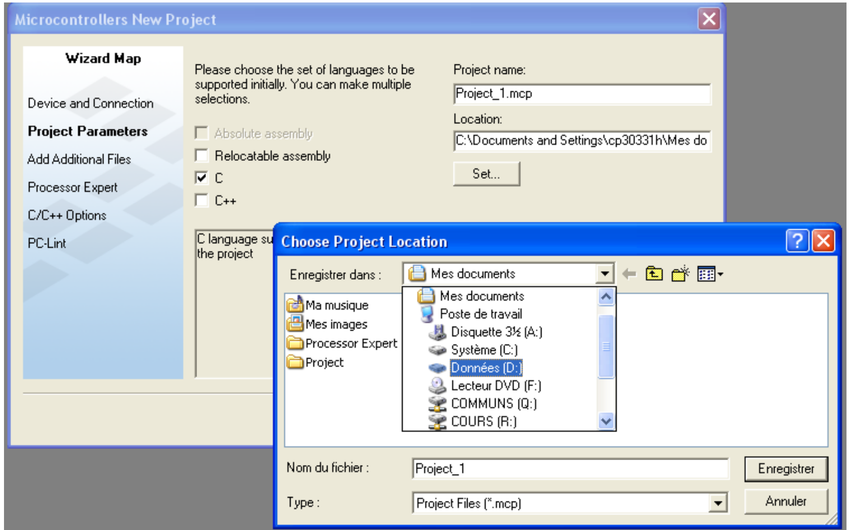


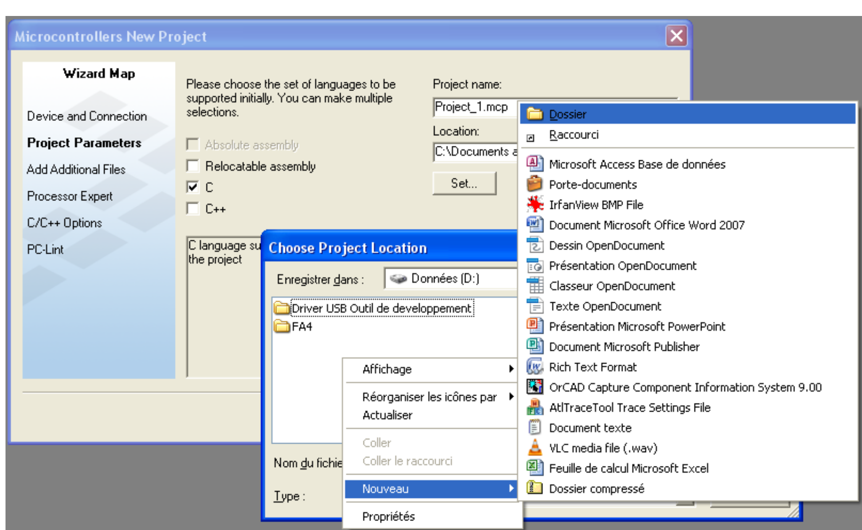
Cliquer sur `Suivant` (deux fois).

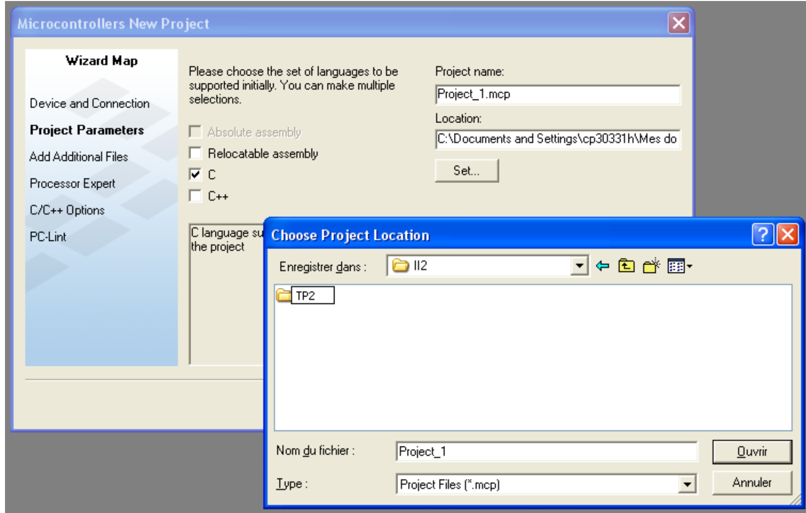


Nommer votre projet dans le champ `project name`.

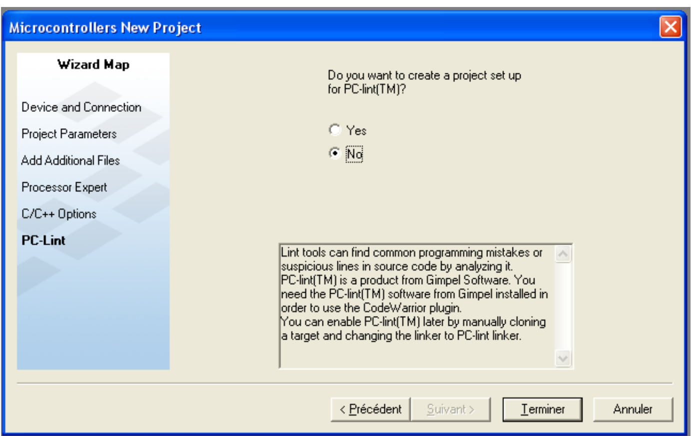
Choisir (ou éventuellement créer) un répertoire pour enregistrer le projet, de préférence sur le disque local `D:` ou encore sur `C:\temp`.







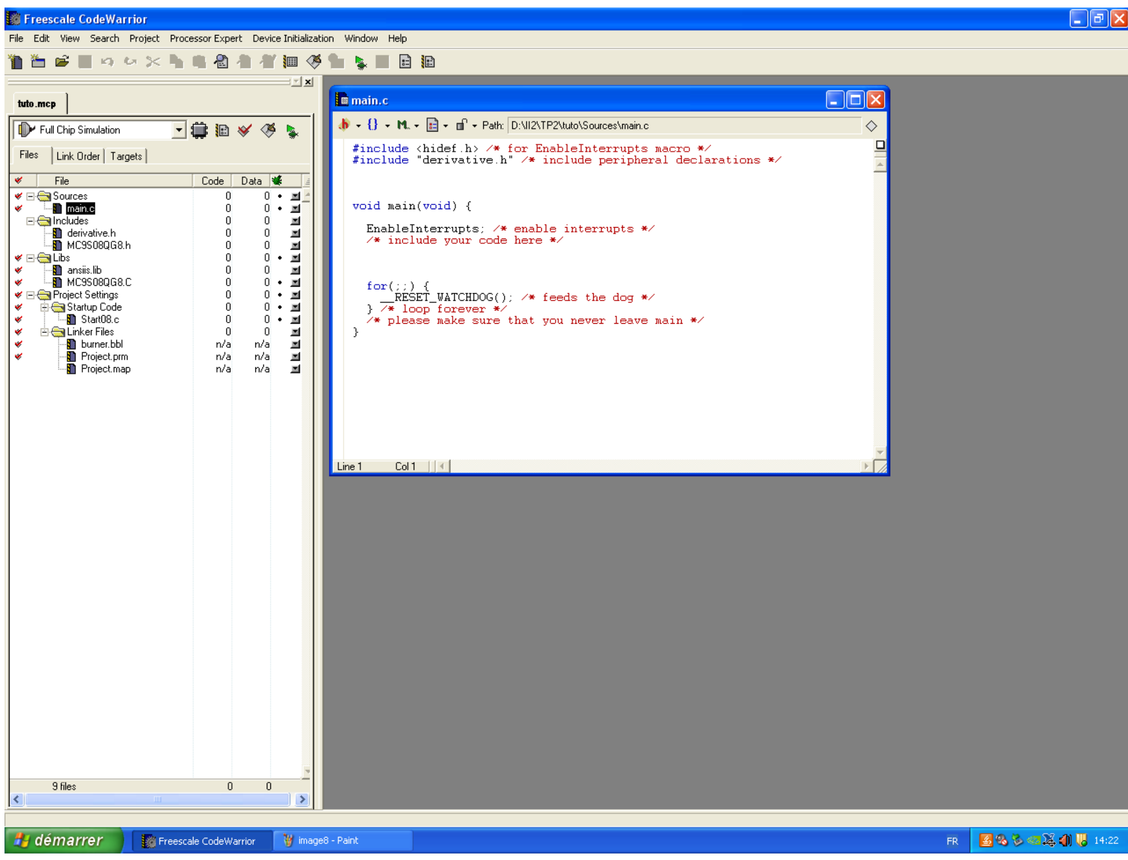
Puis cliquer sur `Suivant`.
Passer les pages suivantes et cliquer sur `Terminer`.



2.1.3 Exploration des fichiers présents

Vous observez dans l'arboresence l'existence de plusieurs fichiers :
Vérifiez que vous êtes bien en `Full Chip Simulation`.
Nous allons nous intéresser aux fichiers suivants :

- `main.c`
- `derivative.h`
- `MC9S08QG8.h`
- `MC9S08QG8.C`



Le fichier `main.c`

Dans la fenêtre de projet sélectionner l'onglet File, développer les groupes (en forme de dossier) et y repérer le fichier `main.c`.

L'ouvrir en double cliquant dessus.

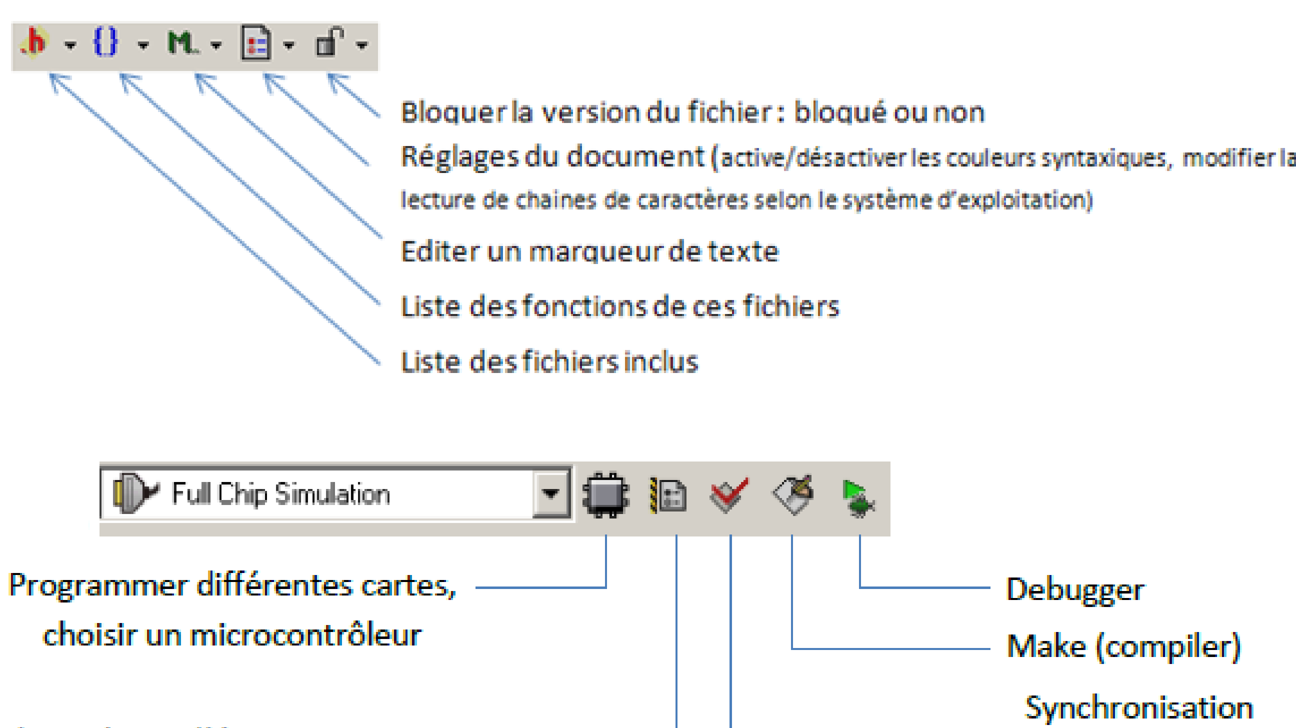
Vous pouvez observer que l'Environnement de Développement Intégré (IDE) fournit une coloration syntaxique des éléments du langage C.
La coloration syntaxique facilite la lecture d'un fichier de code qui permet d'identifier rapidement les éléments.

1. La couleur bleue permet de mettre en évidence les "mots-clés".

2. La couleur rouge est réservée pour les commentaires.

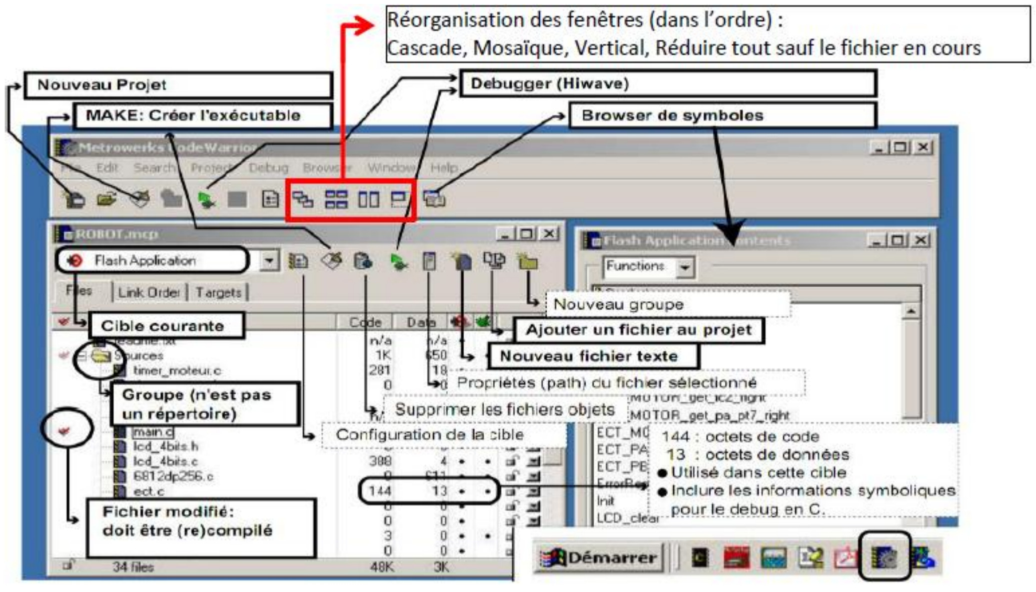
3. Le reste du texte est en noir.

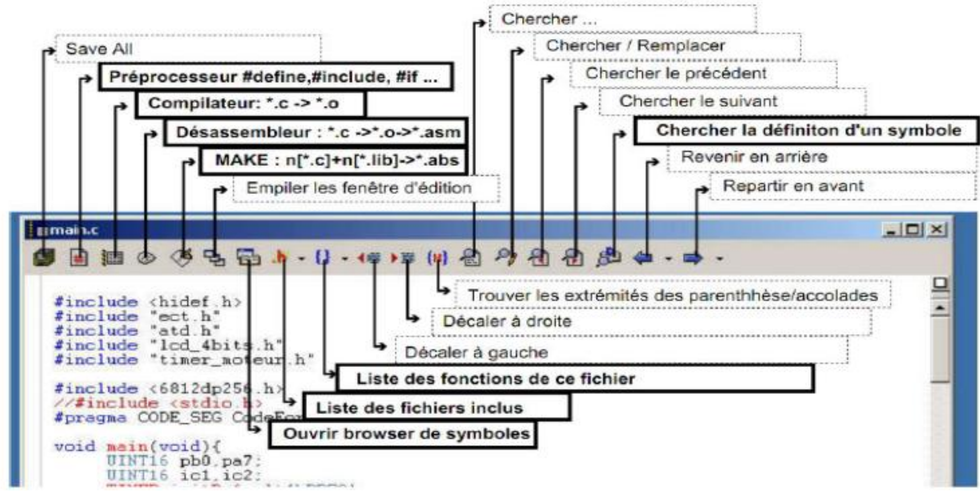
Nous avons les différents outils de l'éditeur de texte suivant :

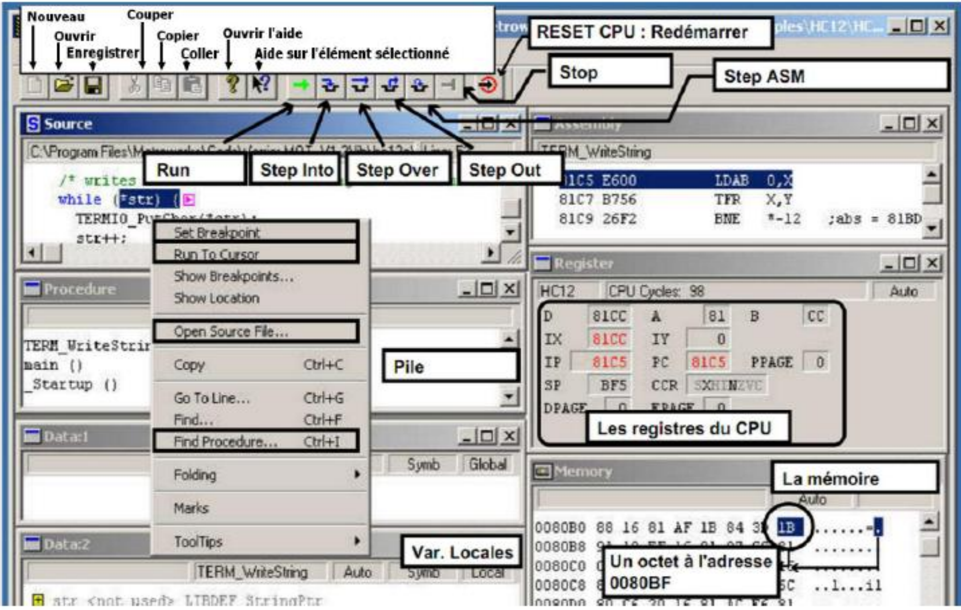


et n'oubliez pas que vous pouvez annuler plusieurs opérations grâce à la touche [CTRL-Z].

Résumé graphiques des différentes icônes de CodeWarrior




Save All
Préprocesseur #define,#include, #if ...
Compilateur: *.c -> *.o
Désassembleur : *.c ->*.o->*.asm
MAKE : n[*.c]+n[*.lib]->*.abs
Empiler les fenêtre d'édition
Chercher ...
Chercher / Remplacer
Chercher le précédent
Chercher le suivant
Chercher la définiton d'un symbole
Revenir en arrière
Repartir en avant
Trouver les extrémités des parenthhèse/accolades
Décaler à droite
Décaler à gauche
Liste des fonctions de ce fichier
Liste des fichiers inclus
Ouvrir browser de symboles


Nouveau
Ouvrir
Enregistrer
Couper
Copier
Coller
Ouvrir l'aide
Aide sur l'élément sélectionné
RESET CPU : Redémarrer
Stop
Step ASM
Run
Step Into
Step Over
Step Out
Set Breakpoint
Run To Cursor
Show Breakpoints...
Show Location
Open Source File...
Copy Ctrl+C
Go To Line... Ctrl+G
Find... Ctrl+F
Find Procedure... Ctrl+I
Folding
Marks
ToolTips
Pile
Les registres du CPU
La mémoire
Un octet à l'adresse 0080BF
Var. Locales

Exercices Chapitre 2

Quelques questions sur le fichier `main.c`
Un microcontrôleur fonctionne en permanence tant qu'il est alimenté. Il passe la plus grande partie de son temps à attendre des interactions avec son environnement (capteurs, pérpiphériques d'entrées, ...) afin de déclancher des actions. Il y a donc une boucle principale dont le rôle est d'attendre en permanence une interaction avec l'environnement du microcontrôleur. Cette boucle principale est une boucle infinie (c'est la boucle `for(;;)` que vous voyez apparaître dans le programme principal. Boucle infinie qu'on aurait pu traduire également par `while(1)`.)

Le microcontrôleur, dispose aussi d'un composant particulier qui s'appelle un *chien de garde* (WATCHDOG en anglais). Le rôle de ce composant est de réinitialiser le programme principal lorsqu'une itération de la boucle principale prend un temps anormalement long. Le Watchdog est un simple compteur matériel, indépendant du processeur, qui compte jusqu'à une valeur seuil et qui, lorsqu'il l'atteint, interrompt le programme principal est le réinitialise (ce qui empêche le microcontrôleur d'être dans un état indeterminé ou bloqué).
(CR4) : Que signifie `__RESET_WATCHDOG` ? Pourquoi est-ce présent **en début** de boucle `for(;;)` ?
(CR5) : Vous remarquerez qu'il n'y a pas de directive `#include<iostream>` (ni `#include<stdio.h>` pour `printf` et `scanf` en C). Pourquoi ?

Le fichier `derivative.h`

Manip : L'ouvrir en double cliquant dessus. On ne modifiera jamais ce fichier !

Ce fichier générique utilisé par CodeWarrior est fourni lors de la création du projet et est indispensable. Il permet d'inclure les déclarations spécifiques à notre microcontrôleur (choisi lors de la création du projet) qui sont regroupées dans le fichier `MC9S08QG8.h`.

Le fichier `MC9S08QG8.h`

Manip : L'ouvrir en double cliquant dessus. On ne modifiera jamais ce fichier !

Ce fichier indispensable traduit l'espace adressable du microprocesseur tel qu'il est présenté dans la documentation technique en un ensemble de variables représentées par leurs noms. Ceci a pour but de faciliter leur manipulation en C (et donc la manipulation de l'espace adressable) comme on le verra par la suite.

Dans la documentation technique du microcontrôleur figurent des informations sur son espace adressable. Ces informations sont traduites dans le fichier `MC9S08QG8.h`.
Quelques questions sur le fichier `MC9S08QG8.h`
(CR6) : Déclaration des variables représentant les ports du microcontrôleur :

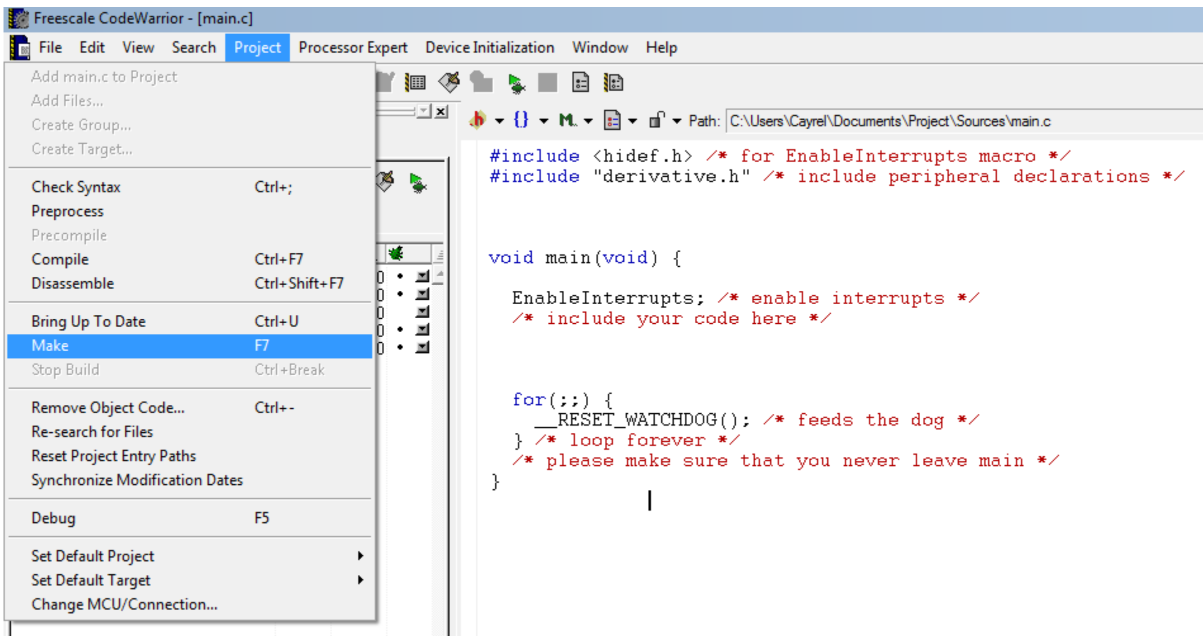
1. Quelle structure de données est utilisée pour représenter les registres des ports du microcontrôleur ?
2. Comment sont déclarés ces registres dans ce fichier ?
3. A quoi correspond le mot clé `extern` ? le mot clé `volatile` ?
4. Comment est fait le lien entre l'adresse des registres du Port B et les variables permettant de représenter son registre de données et son registre de direction ?
5. D'après la documentation seuls les 6 premiers bits du registre de données du port A sont accessibles. Comment ceci est traduit dans le fichier `MC9S08QG8.h` ?

(CR7) : Accès aux informations binaires des registres :

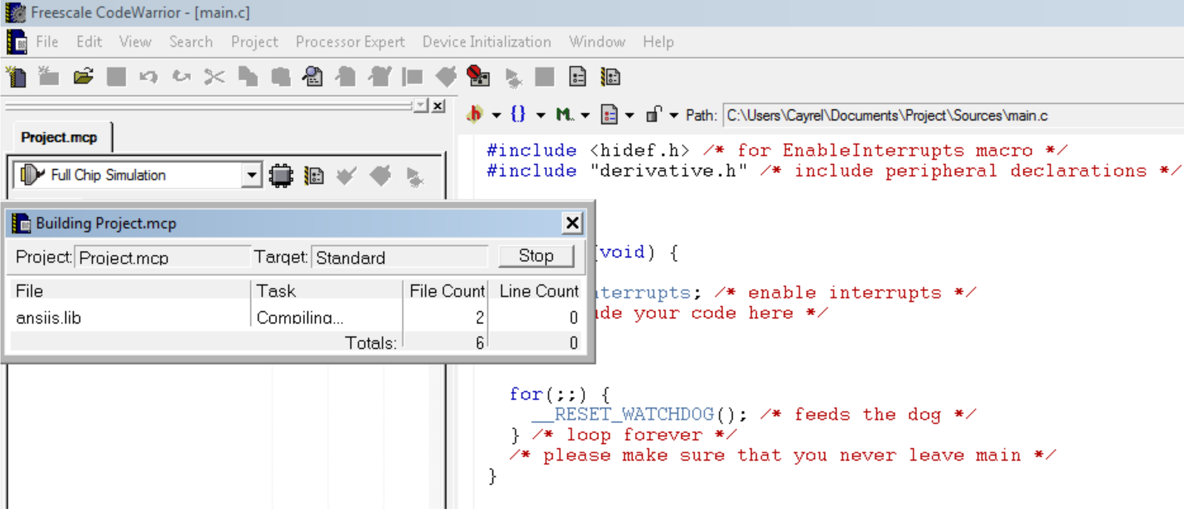
1. En utilisant des masques et des opérateurs bit à bit, quelle instruction en langage C devrions-nous écrire pour tester si le bit d'indice 4 du registre PTBD est à 1 ?
2. À quoi correspond `#define PTBD_PTBD0 _PTBD.Bits.PTBD0` ?
   On précise que les instructions du type `#define instruction_lisible instruction_complexe` sont exécutées avant la compilation et permettent de remplacer les instructions lisibles écrites dans le code et de les remplacer par les instructions réelles qui seront vraiment comprises et compilées par le compilateur. (C'est une sorte de rechercher/remplacer dans le texte avant la compilation). L'objectif est de simplifier l'écriture du code est d'améliorer sa lisibilité.
   Par exemple, on pourrait taper dans un code :
   `Si(a==3)` pour peu qu'on ait mis en en-tête du fichier la ligne `#define Si if`.
   Avant la compilation, à chaque fois que le précompilateur rencontre `Si` il sait qu'il doit le remplacer par `if` selon la correspondance décrite dans le `#define`.
3. Sans utiliser des masques et des opérateurs bit à bit, mais en utilisant les correspondances définies par le `#define`, que pouvez-vous écrire, en langage C, pour tester si le bit d'indice 4 du registre PTBD est à 1 ?

2.1.4 Générer l'exécutable

Afin de générer l'exécutable, il faut compiler chacun des fichiers sources (`.c`) afin de générer des fichiers objets (`.o`), puis lier (correspondant à l'édition de lien) chaque fichier objet pour fabriquer un seul exécutable.
Cette procédure de génération de la solution est automatisée et s'appelle en cliquant sur l'icône `make`.



Une fenêtre présente alors l'état d'avancement de la compilation du projet.
Manip : Faites-le.
Félicitations, vous venez de créer votre premier exécutable pour le microcontrôleur `MC9S08QG8`. En tant que tel il ne fait rien pour le moment, mais il est intéressant de constater que le langage machine généré par cette compilation l'est pour le processeur de la carte et pas le processeur du PC qui n'ont



pas la même architecture, ni le même jeu d'instructions. On parle de compilation croisée ou cross-compilation.

Pour vous en convaincre, essayer d'exécuter votre exécutable depuis le PC.

1. **Manip :** Localiser votre fichier exécutable : il se trouve dans votre répertoire de projet, sous le répertoire `bin` et devrait avoir le nom générique `Project.abs`.
2. **Manip :** Si vous l'ouvrez directement, windows vous demandera avec quel logiciel vous souhaitez l'ouvrir puisqu'il ne connaît pas l'extension. Renommez-le en `Project.exe` et exécutez le en constatant que le processeur de votre PC ne reconnaît pas les instructions codées dans ce fichier.

Séance/heure	:
Signature	:

Il faudra envoyer ce fichier dans la mémoire FLASH du microcontrôleur, cette étape sera effectuée grace à la carte USBDM.

Chapitre 3

Allumage de LEDs

Objectif

L'objectif de ce chapitre est d'utiliser CodeWarrior pour configurer la carte et allumer des LEDs avec ou sans l'utilisation des interrupteurs et du bouton-poussoir.
Dans le premier chapitre, vous avez découvert la carte et fait le lien avec la DataSheet du microcontrôleur. Dans le deuxième chapitre, vous avez découvert le logiciel permettant de placer du code dans le microcontrôleur et fait le lien entre les déclarations spécifiques du microcontrôleur et la DataSheet.
Dans ce chapitre, vous ferez le lien entre ce logiciel et la carte de développement en faisant exécuter du code par le microcontrôleur que vous aurez programmé [1].
Dans le prochain chapitre, vous ferez clignoter les LEDs de la carte de développement à différentes fréquences.

1. D'après :
   — `http://www.ief.u-psud.fr/~jok/iut/HC12/CW_HC12.pdf`
   — `http://meteosat.pessac.free.fr/IMA/ressources/Doc_C/Doc_HC12/TP_ii1_S1_GE1.pdf`

3.1 “Allumage” de LED en Full Chip Simulation

Créez un **nouveau projet TP3** en suivant la procédure décrite dans le TP2.
Ouvrez le fichier `main.c`. A ce stade, il ne comporte que les informations originelles.
L'objectif est d'“allumer” [2] la LED D1.
(**CR8**) **:** En vous rappelant à quels composants de la carte de développement est connecté le port B, donnez l'instruction en C permettant d'initialiser le PORT B dans la bonne direction
(**CR9**) **:** Allumage d'une led : On suppose que les leds D2, D4, D6, D8 sont allumées sur la carte.

1. Expliquer la différence entre ces deux instructions : `PTBD=1;` et `PTBD_PTBD0=1;`
2. Laquelle faut-il utiliser pour allumer la LED D1 et **ne pas agir sur les autres** ?
3. Donnez une instruction équivalente en langage C utilisant un opérateur bit à bit bien choisi et un masque adapté.

Placement des instructions :

1. Où faut-il placer l'instruction d'initialisation de la direction du Port B ? (avant le `for`, dans le `for` ou après le `for`).
2. Où peut-on placer l'instruction permettant d'allumer une LED ? (avant le `for`, dans le `for` ou après le `for`).
   Discutez des différentes possibilités. Quel est le placement le plus judicieux ?
3. Où faut-il placer l'instruction permettant de tester, durant le fonctionnement du microcontrôleur, si la LED D1 est allumée ?

Manip : Après analyse du bon placement de ces instructions, écrivez les dans le fichier `main.c` et compiler votre code (icône `make`).

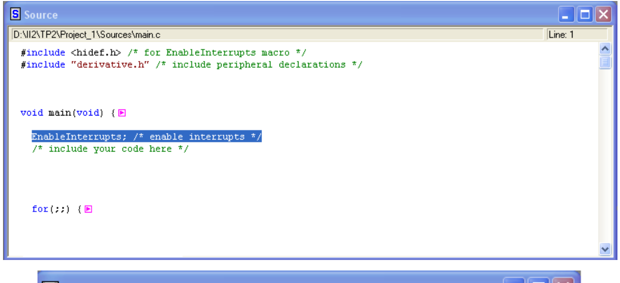
Séance/heure	:
Signature	:

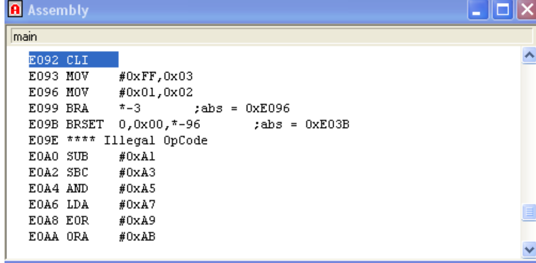
2. On n'allume pas vraiment la LED D1 mais on doit envoyer une information sur un port de sortie. L'objectif est de préciser quelle est cette information pour que la LED D1 puisse s'allumer si on la connecte au port de sortie correspondant.

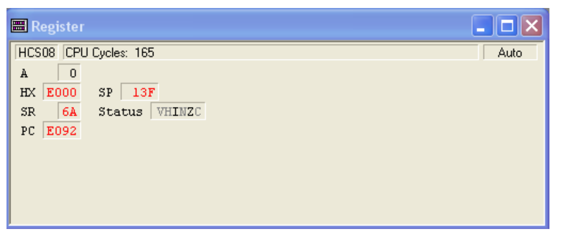
Lancer le débogueur et charger le programme en mémoire

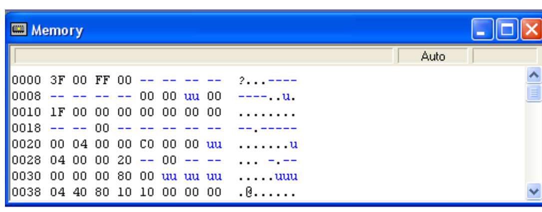
À partir de la fenêtre du projet : Project → Debug ou [F5] ou .

Dans cette première partie du TP, nous sommes en mode `Full Chip simulation`, il n'y a donc pas de communication entre l'IDE Code Warrior et le microcontrôleur. Tout se passe à l'intérieur du PC au niveau de l'IDE qui connait les caractéristiques du microcontrôleur.
On s'intéresse aux fenêtres `Source`, `Assembly`, `Register` et `Memory`.

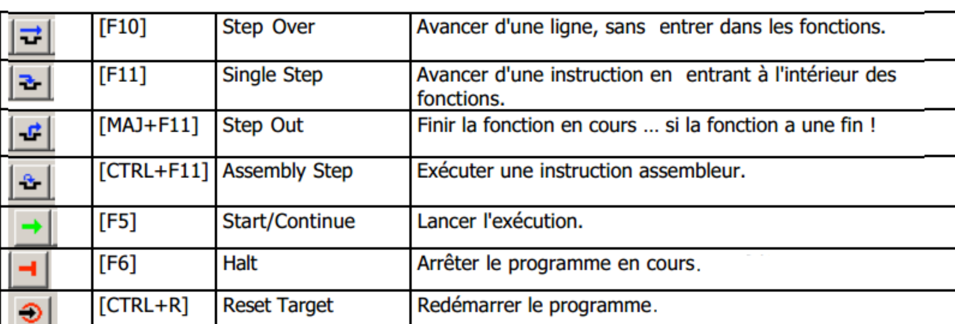








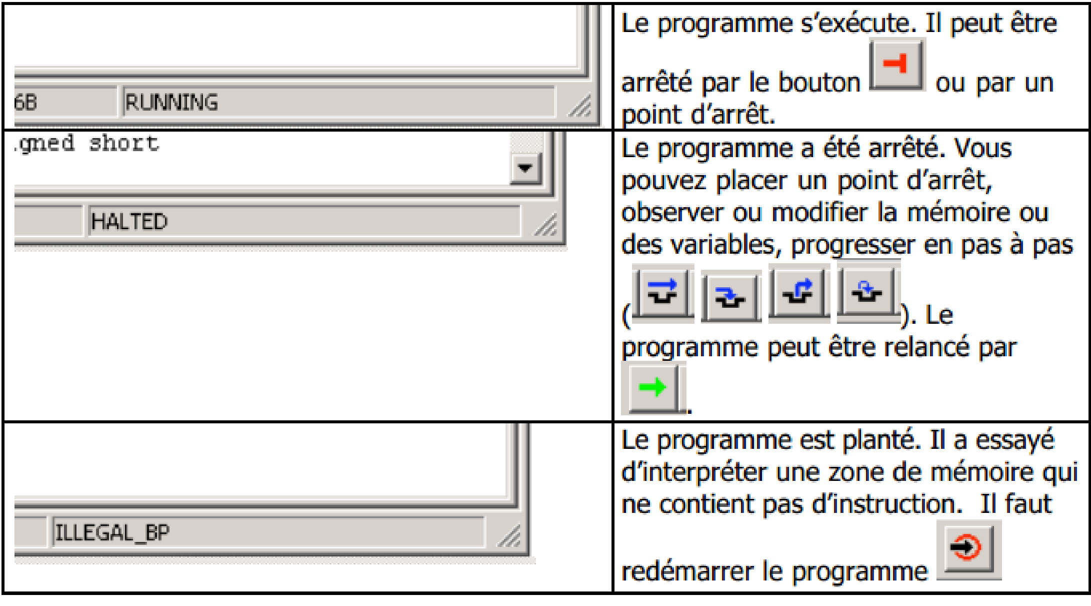
La case `Source`
Vous pouvez contrôler l'exécution du programme.
Vous pouvez avancer en pas à pas ou lancer/arrêter l'exécution du programme ou redémarrer la carte avec les icônes de la barre d'outils : .



	Raccourci	Nom	Description
	[F10]	Step Over	Avancer d'une ligne, sans entrer dans les fonctions.
	[F11]	Single Step	Avancer d'une instruction en entrant à l'intérieur des fonctions.
	[MAJ+F11]	Step Out	Finir la fonction en cours ... si la fonction a une fin !
	[CTRL+F11]	Assembly Step	Exécuter une instruction assembleur.
	[F5]	Start/Continue	Lancer l'exécution.
	[F6]	Halt	Arrêter le programme en cours.
	[CTRL+R]	Reset Target	Redémarrer le programme.

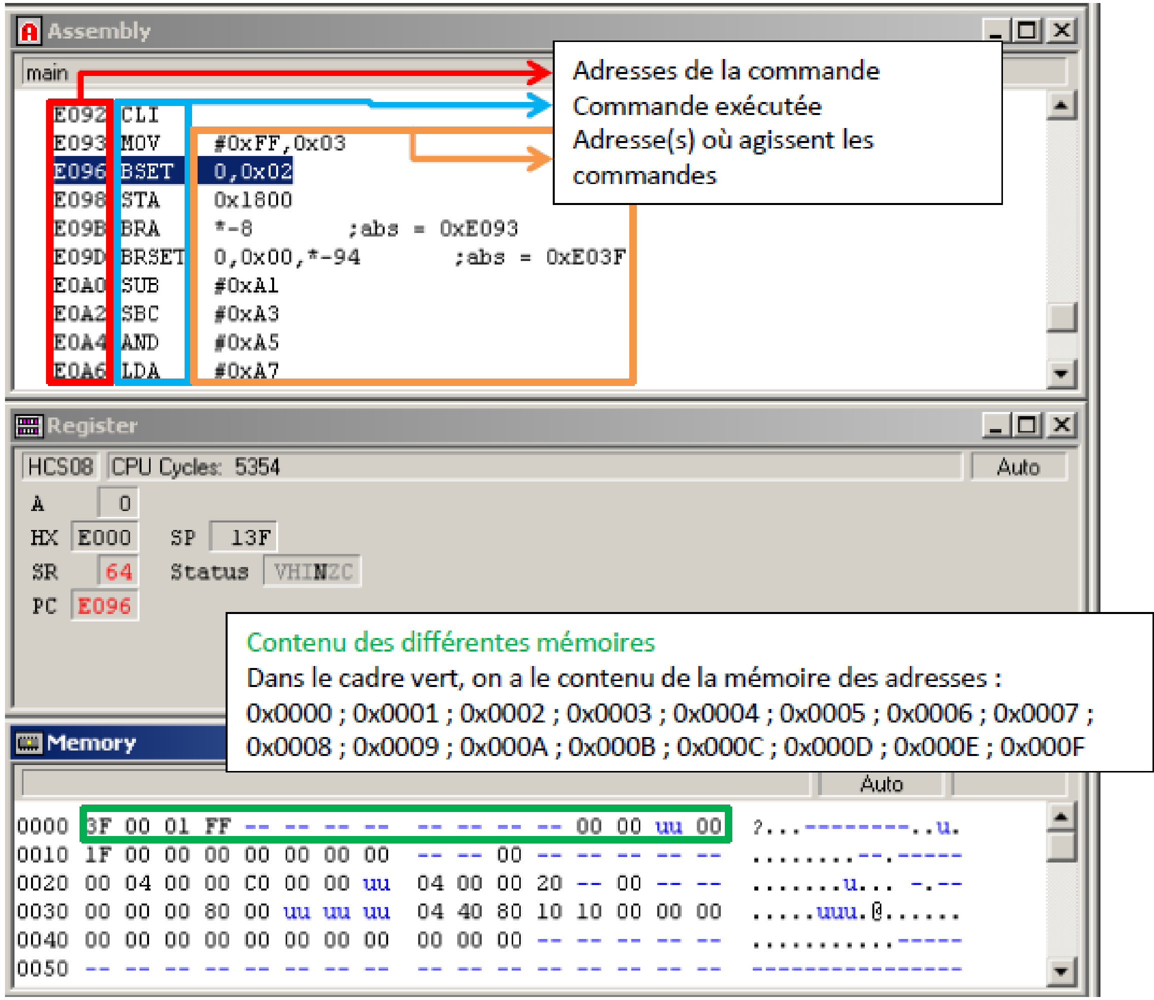
Table 3.1 – Signification des éléments permettant de dérouler ou réinitialiser le programme

Le coin inférieur droit vous indique dans quel état se trouve la carte.
Vous devez savoir à tout moment si votre programme est en cours d'exécution ou arrêté.



État	Signification
6B RUNNING	Le programme s'exécute. Il peut être arrêté par le bouton ou par un point d'arrêt.
.gmed short HALTED	Le programme a été arrêté. Vous pouvez placer un point d'arrêt, observer ou modifier la mémoire ou des variables, progresser en pas à pas (). Le programme peut être relancé par .
ILLEGAL_BP	Le programme est planté. Il a essayé d'interpréter une zone de mémoire qui ne contient pas d'instruction. Il faut redémarrer le programme

Lors d'une exécution du programme en mode pas à pas, l'instruction qui va être exécutée est surlignée en bleu dans la fenêtre Assembly, on a un état des registres du processeur au moment où l'instruction s'apprête à être exécutée (fenêtre Register) et on a un état de l'espace adressable du processeur (fenêtre Memory).

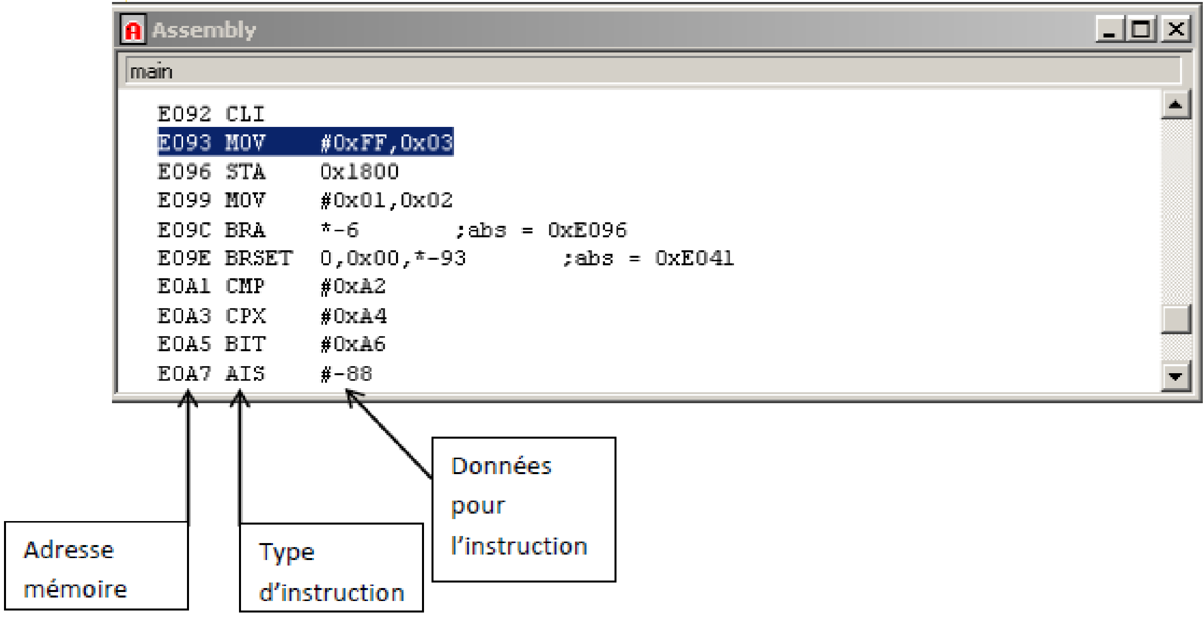


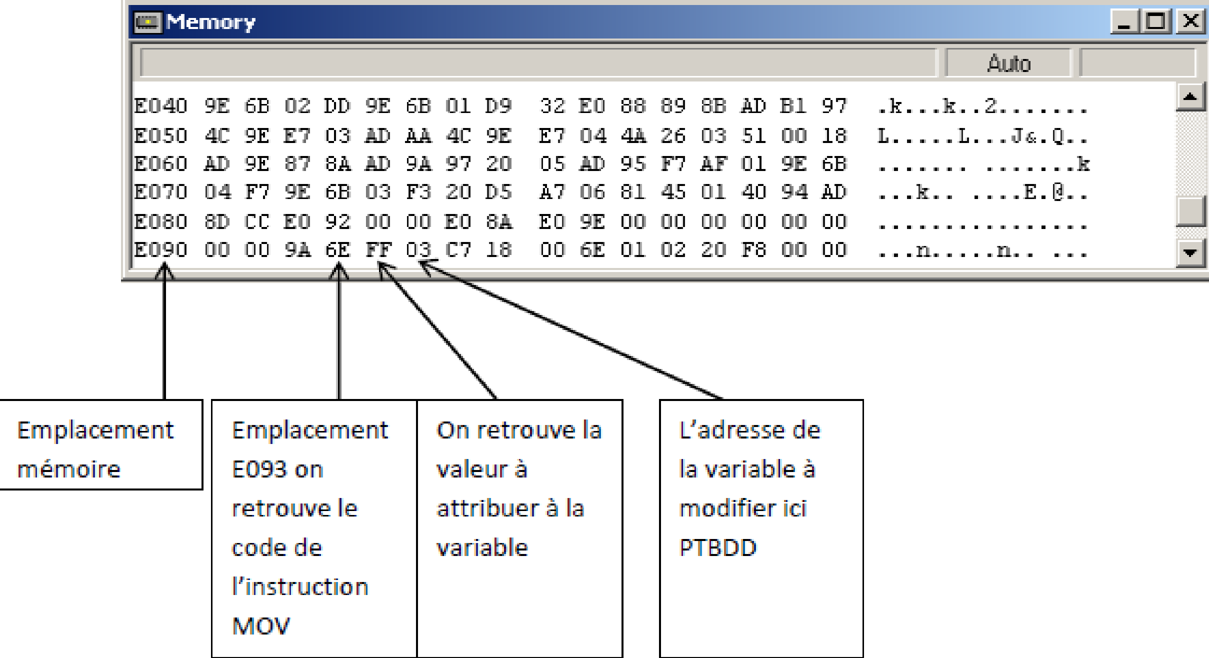
Voyons un peu plus en détail les informations contenues dans chacune de ces fenêtres.

La case `Assembly` (Les réponses sont données ci-après et ne font pas l'objet d'un CR)

1. Que signifient les 4 premiers chiffres ? Les caractères suivants ? Et les suivants ?
2. Pouvez-vous en déduire (à l'aide de la case `Memory` le code d'instruction de l'instruction `MOV`) ? Donner ce code.

Réponses :





Avec le bouton droit de la souris vous pouvez afficher (`Display`) plus d'informations.

La case `Register` Elle permet de connaître le nombre de cycles CPU nécessaires pour effectuer les opérations du code exécutable généré.
Manip : Vérifiez que le nombre de cycles change en exécutant le programme pas à pas (touche [F10] `step over`).

On peut également consulter le contenu des registres du processeur :

A : registre accumulateur

HX : registre d'index

SP : pointeur de pile

PC : program counter

Status : en particulier pour les bits indicateurs d'état (Z, N, C et V).

La case `Memory`
Cette fenêtre représente l'état de la mémoire (ie : l'espace adressable du processeur).
La colonne de gauche représente les adresses. Les autres colonnes représentent des données (16 octets par ligne). De ce fait, les adresses dans la colonne de gauche sont affichées uniquement de 16 en 16 (et pas de 1 en 1).
Par défaut, le contenu de la mémoire est présenté en hexadécimal, mais un menu contextuel (bouton droit) permet de choisir d'autres formats d'affichage.
Manip : Vérifiez dans cette fenêtre `Memory` que les cases mémoires correspondant aux ports que vous avez initialisés comportent les bonnes valeurs. Appelez l'enseignant pour qu'il atteste que la LED D1 s"'allume" en mode simulation.

Séance/heure	:
Signature	:

À l'aide de la documentation technique, faite une recherche de `Vreset` dans le pdf et noter la plage d'adresses occupée par `Vreset` ______ .
À l'aide de la fenêtre memory, rendez-vous à l'adresse occupée par `Vreset`. Quelle est sa valeur ? Notez là : ______ .

Manip : Faites un redémarrage du programme (voir table 3.1) et regardez dans la fenêtre `Assembly`.

(CR10) : À quelle adresse se trouve l'instruction par laquelle le programme démarre. Est-ce cohérent ?

Allumage de LEDs

3.2 Simple allumage

À partir du même projet que celui réalisé en section 3.1 et après avoir fait constater à l'enseignant que la LED D1 s'allume en mode simulation, passez en mode de programmation `HCS08 Open source BDM` à la place de `Full chip Simulation`. Envoyez votre programme dans la mémoire FLASH du microcontrôleur (icône)

Manip : Vérifiez que la LED D1 s'allume physiquement sur la carte de développement puis répondez aux questions suivantes (Si la validation est faite par l'enseignant et que vous n'avez rien modifié, ça devrait être le cas !!).

1. Complétez, directement sur le cahier et **avant toute manipulation**, le tableau suivant permettant d'indiquer quelles leds **devraient** s'allumer (mettre une croix dans les cases correspondantes) lorsque on écrit les instructions suivantes :

Instructions	D8	D7	D6	D5	D4	D3	D2	D1
`PTBD = 0b01100010;`		x	x				x	
`PTBD = 0xAC;`								
`PTBD = 123;`								
`PTBD = 010;`								
`PTBD = 300;`								
`PTBD = -1;`								

2. **Manip :** vérifiez sur la carte que vos suppositions sont correctes. Pour gagner du temps, vous pourrez taper toutes les instructions d'affectation à la suite et exécuter le programme en mode pas à pas pour voir les différents cas successivement à l'aide d'une exécution du programme (plutôt que de modifier le code, recompiler et reflasher la mémoire du microcontrôleur).
3. **(CR11) :** Dans le cas où il y aurait des différences entre vos suppositions (avant Manip) et l'information visualisée sur les leds lors de la Manip, expliquez ce que « comprend » le microprocesseur de l'information écrite en langage C.

L'objectif est à présent d'interagir avec les entrées du microcontrôleur pendant son fonctionnement. On va donc agir sur l'état des entrées (bouton poussoir, interrupteurs dans un premier temps) afin d'allumer ou éteindre les LED si l'appui sur un bouton poussoir ou l'actionnement d'un interrupteur est effectué.

3.3 Allumage avec utilisation du bouton poussoir

L'objectif de cette partie est d'allumer la led D1 si et seulement si le bouton poussoir est enfoncé.

1. En vous aidant des schémas de cablage figurant en début de cahier de TP, rappelez à quelle patte du microcontrôleur est relié le bouton poussoir [   ].
2. Rappelez ensuite à quel PORT et dans quel registre (en précisant l'indice de ce registre) est envoyée l'information (enfoncé/relaché) provenant du bouton poussoir [   ].
3. En vous aidant du schéma de la carte de développement, précisez la valeur du bit d'état lorsque le bouton poussoir est enfoncé [   ].

4. **Manip :** Dans le fichier `main.c`, Le registre de direction de ce PORT sera configurée globalement en entrée. Faites-le directement dans le code et testez l'allumage de la led D1 sur la carte si et seulement si le bouton poussoir est enfoncé.

Séance/heure	:
Signature	:

   Nous aimerions remplacer l'instruction `if(acces_au_port == valeur_a_tester)` que vous avez tapée précedemment, par une instruction plus simple permettant de considérer une instruction du type `if(BP)` où `BP` représentera la variable logique qui est VRAI lorsque le bouton poussoir est enfoncé et FAUX lorsque le bouton poussoir est relâché.

   Compléter la ligne suivante `#define  BP  ...à compléter...` afin de pouvoir utiliser par la suite `if(BP)`.
   `#define BP` [ ].
   Ces alias seront écrits dans un fichier d'en-tête qui les regroupera.
   Manip : Créez un fichier `mesraccourcis.h` dans lequel vous ajouterez la ligne complétée ci-dessus.
5. **Manip :** Inclure le fichier `mesraccourcis.h` dans votre `main.c`.
   Attention, vous incluerez ce fichier `.h` en utilisant des `""` puisque il se trouve dans le même répertoire que votre fichier `.c` l'incluant.
   On rappelle que les inclusions de fichiers avec des `<>` correspondent à l'inclusion de fichiers systèmes présents à l'installation de l'IDE et figurant dans un répertoire d'installation dont le chemin d'accès est « masqué » par les `<>`.
6. **Manip :** Effectuer le test précédent en utilisant `BP` au lieu de ce que vous utilisiez avant et constater que le résultat est le même (sinon modifier votre `#define` dans le fichier `mesraccourcis.h`).

Séance/heure	:
Signature	:

   Dans le futur, on pourra modifier ce fichier `mesraccourcis.h` mais aucun autre fichier `.h`

3.4 Allumage avec utilisation des interrupteurs

On souhaite s'inspirer de ce qui a été fait dans la section précédente et l'adapter aux interrupteurs `SW1` et `SW2`.
Mettre en commentaire, dans le fichier `main.c`, les lignes de code correspondantes aux instructions permettant de tester l'appui sur le bouton poussoir.

1. **Manip :** Effectuez un test sur la valeur du registre de données du port A associé à l'interrupteur `SW1` pour allumer la LED D1 uniquement lorsque l'interrupteur `SW1` est activé.
2. **Manip :** Modifiez le fichier `mesraccourcis.h` en ajoutant un `#define SW1` [ ] correspondant à l'interrupteur `SW1` considéré comme une variable logique VRAI lorsque actif et FAUX lorsque inactif [3].
3. **Manip :** Modifiez le fichier `mesraccourcis.h` en créant un `#define SW2` [ ] correspondant à l'interrupteur `SW2` considéré comme une variable logique.

3. Quand on regarde la carte de sorte à pouvoir lire à l'endroit les noms des différents composants, on considère que le SW est actif lorsqu'il est en position haute et inactif lorsqu'il est en position basse

4. **Manip :** Effectuez le test précédent en utilisant `SW1` au lieu de ce que vous utilisiez avant. En déduire l'état logique dans lequel le `SW1` (ou `SW2`) est activé.
   `SW1` (ou `SW2`) est actif lorsque la valeur envoyée au bit du registre est égale à .
   Nous vous rappelons ici les différentes conditions logiques en `C` ainsi que leur syntaxe.

<table>
<tr><th>Opérateur</th><th>Fonction</th></tr>
<tr><td>&</td><td>ET (AND)</td></tr>
<tr><td>|</td><td>OU (OR)</td></tr>
<tr><td>^</td><td>OU exclusif (XOR)</td></tr>
<tr><td>~</td><td>Inversion (NOT)</td></tr>
<tr><td>>></td><td>Décalage à droite (Right Shift)</td></tr>
<tr><td><<</td><td>Décalage à gauche (Left Shift)</td></tr>
<tr><td>&&</td><td>ET logique</td></tr>
<tr><td>||</td><td>OU logique</td></tr>
<tr><td>!</td><td>NON logique</td></tr>
</table>

5. **Manip :** Écrivez dans le `main.c` un code qui allume les LEDs (D_1, ..., D_8) d'indices pairs si et seulement si les interrupteurs `SW1` et `SW2` sont actionnés.
6. **(CR12) :** Mettez une capture d'écran de ce code dans votre CR.
   Pourquoi est-ce que `if(SW1 & SW2)` et `if(SW1 && SW2)` donnent le même résultat ici ? Est-ce toujours le cas[4] ?
   Même si elles donnent le même résultat ici, que doit-on utiliser au final entre `if(SW1 & SW2)` et `if(SW1 && SW2)` pour tester si les deux interrupteurs sont actifs simultanément ?
7. **Manip :** Écrivez dans le `main.c` un code qui allume les LEDs d'indices impairs si et seulement si l'interrupteur `SW1` est actionné en même tant que le bouton poussoir ou bien lorsque l'interrupteur `SW2` est actif mais pas le bouton poussoir.
   Faites vérifier par l'enseignant le bon fonctionnement de cette condition.

Séance/heure	:
Signature	:

4. Si votre réponse est oui, voyez ce qu'il se passe entre `if(0b01 & 0b10)` et `if(0b01 && 0b10)` puis expliquez.

Chapitre 4

Clignotement des LEDs

Objectif

L'objectif de ce chapitre est d'étudier les différentes méthodes permettant de régler la fréquence de clignotage des LEDs.

Dans le premier chapitre, vous avez découvert la carte et fait le lien avec la DataSheet du microcontrôleur.

Dans le deuxième chapitre, vous avez découvert le logiciel permettant de placer du code dans le microcontrôleur et fait le lien entre les déclarations spécifiques du microcontrôleur et la DataSheet.

Dans le troisième chapitre, vous avez pu programmer le microcontrôleur pour allumer des LEDs en utilisant les interrupteurs et le bouton poussoir sur la carte de développement.

Dans ce chapitre, vous allez aborder la notion de cycles machine cadencés par une horloge fonctionnant à une fréquence donnée et pourrez ainsi faire clignoter les LEDs à une fréquence demandée.

4.1 Taille des types de données

Sur ce microcontrôleur, pour savoir quelle taille occupe un type de donnée (`int`, `char`, `float`, `double`, etc...), on peut utiliser la fonction `sizeof` (il faut inclure le fichier `stdlib.h` en en-tête de fichier : `#include<stdlib.h>`) qui retourne le nombre d'octets occupé par un type donné.
Si ce nombre d'octets est affecté à `PTBD`, il sera possible de visualiser ce nombre sur les leds (en binaire) sur la carte (ou en mode `Full Chip Simulation` en exécutant le programme pas à pas, dans la fenêtre `Memory` en visualisant le contenu du registre PTBD) et de l'associer à un nombre décimal correspondant au nombre d'octets de chaque type.
Exemple : après avoir initialisé le PORT B en sortie, vous pouvez écrire : `PTBD=sizeof(char);` qui vous retourne 1 et l'affecte à PTBD. PTBD contenant 1, c'est la led D1 qui s'allumera.

Manip : Faites les étapes nécessaires [1] au niveau du code (fichier `main.c`) et complétez le tableau suivant directement sur le document :

Type	Led qui s'allume	Taille en octets	Taille en bits
`char`			
`short int`			
`long int`			
`int`			
`long long int`			
`float`			
`double`			

Table 4.1 – Taille des types de base sur le microcontrôleur MC9S08QG8

1. On vous conseille d'écrire toutes les instructions à la suite avant le `for(;;)` et de les exécuter en mode pas à pas détaillé pour voir les affichages successifs sans avoir à remodifier votre code et recompiler à chaque fois

4.2 Compréhension d'un code proposé

On vous propose le code suivant.

```
#include <hidef.h>
#include "derivative.h"

void main(void) {
  unsigned int i;
  /* Internal clock of 8MHz selected. */
  ICSC2 = 0x00;
  ICSC1 = 0x04;

  SOPT1_COPE=0; /* Disable Watchdog */

  /* PTB configured as an output */
  PTBDD=0xFF;

  for(;;) {
    i=0;
    while(i<65535)
    {
      i++;
    }//Boucle complète
    PTBD=~PTBD;
  } /* loop forever */
}
```

1. **Manip :** Recopiez puis exécutez ce code. Ne pas faire de copier/coller à partir du fichier pdf (problème d'encodage de caractères).
   Qu'observez vous sur la carte de développement ?
2. Quelle est la valeur de `i` en sortie de boucle.
3. **(CR13) :** En vous aidant du tableau 4.1, expliquez pourquoi on s'arrête à cette valeur. On appellera une telle boucle, une **boucle complète**.
4. **Manip :** Déclarez à présent la variable `i` comme un `unsigned char` (au lieu de `unsigned int`) et changer 65535 par la valeur maximale que peut prendre un `unsigned char` .
   Recompilez, programmez le microcontrôleur puis exécutez ce code.
   (CR14) : Que constatez-vous ? Expliquez ce phénomène.
5. **Manip :** Redéclarez `i` comme un `unsigned int` et rechanger la valeur maximale en 65535 dans le test de la boucle, recompilez (`make`) puis repassez en mode `Full Chip Simulation` et lancer le programme (`Debug`).
   Dans la fenêtre `Source`, utilisez des points d'arrêts (clic droit à l'endroit où vous voulez en ajouter un puis `set breakpoint`) avant et après la boucle `while` pour déterminer sa durée en nombre de cycles CPU.
6. **(CR15) :** La fréquence de l'horloge interne utilisée ici étant approximativement de 8MHz, donnez en exposant votre raisonnement, la durée approximative en ms d'une boucle complète.
7. **(CR16) :** On suppose que le nombre de cycles CPU est proportionnel à la taille de la boucle (ie : le nombre d'itérations).
   (a) donnez en justifiant la taille théorique de la boucle (la valeur de comparaison théorique dans le `while(i<valeur_theorique)`) pour avoir un clignotement des LEDs toutes les 500ms.

(b) Est-ce possible avec une boucle dont l'indice est un `unsigned char` ? un `unsigned int` ? un `unsigned long int` ? un `unsigned long long int` ? [2] ?

Manip : Essayez en déclarant `i` comme un et en mettant la valeur théorique du nombre d'itérations que vous avez trouvée.

CR17 : Que constatez-vous ? Est ce conforme à vos prédictions ? Essayez de donner une raison.

8. **CR18 :** A l'aide de la documentation du microcontrôleur, recherchez des informations sur les deux bits de poids fort du registre `ICSC2`.

9. **En veillant à bien redéclarer `i` comme un `unsigned int`** pour le compteur de boucle, proposez une modification de la valeur de `ICSC2` : ICSC2= ; ainsi que de la taille de la boucle de manière à obtenir un changement d'état sur les LEDs toutes les 500 ms.

   Manip : Testez votre modification sur la carte (modification du code et programmation du microcontrôleur).

   CR19 : Modification de `ICSC2` :

   (a) Quel est l'inconvénient principal pour le système d'une telle modification de `ICSC2` ?

   (b) Y a-t-il une seule valeur de `ICSC2` qui convient ? si non, quelle est la moins pire ?

4.3 Création d'une procédure de temporisation

Dans cette section, `i` est redéclaré comme un `unsigned int`.
De plus, pour des raisons évidentes (voir votre réponse CR19), on veut conserver la fréquence de 8MHz.

Pour compter un temps correspondant à 500ms, on peut alors utiliser plusieurs **boucles complètes** les unes à la suite des autres. Toutefois, le nombre de boucles complètes ne correspond pas forcément à exactement la durée de 500ms. On peut donc utiliser une dernière boucle qualifiée d'incomplète dont l'indice n'ira que jusqu'à une valeur donnée correspondant à la `taille` de la dernière boucle (l'indice de boucle ne prendrait alors que les valeurs entre 0 et `taille-1`).

Dans cette section, on vous demande de créer une procédure `tempo` afin d'assurer ce comptage et de créer une temporisation dont la durée sera paramétrable (500ms n'est qu'un exemple d'utilisation).
Cette procédure prend comme argument :

— le nombre de boucles complètes codé sur un octet ;
— la taille de la dernière boucle codée sur deux octets.

Le prototype (imposé) de la procédure `tempo` est donc le suivant :

```
void tempo(unsigned char, unsigned int);
```

1. **Manip :** Créez un nouveau projet ;
2. **Manip :** Recopiez le fichier `main.c` suivant :

2. Référez-vous au tableau 4.1

```
#include <hidef.h>
#include "derivative.h"

void main(void) {
  /* Internal clock of 8MHz selected. */
  ICSC2 = 0x00;
  ICSC1 = 0x04;

  SOPT1_COPE=0; /* Disable Watchdog */

  PTBDD=0xFF;
  /* include your code here */
  for(;;) {

  } /* loop forever */
}
```

3. **Manip :** Créez un fichier `tempo.h` dans lequel vous déclarerez la procédure `tempo` puis inclurez ce fichier dans le dossier `Includes` de l'arborescence de votre projet ;
4. **Manip :** Créez un fichier `tempo.c` dans lequel vous ajouterez en en-tête `#include "tempo.h"` et vous définirez la procédure `tempo` puis inclurez ce fichier dans le dossier `Sources` de l'arborescence de votre projet ;
5. **Manip :** Dans le fichier `main.c`, ajoutez `#include"tempo.h"` en en-tête.
   Puis, dans la boucle principale, appelez la procédure `tempo(1,0)` ;
6. **Manip :** Passez en mode `Full Chip Simulation` et appuyez sur la touche `debug`.
   Une nouvelle fenêtre s'ouvre.
   Cliquez droit dans la fenêtre `Source` (ouverture d'un menu contextuel) puis cliquez sur `Open Source File` et choisissez le fichier `tempo.c`.
   En utilisant des points d'arrêts au début et à la fin de la procédure `tempo( , )`, déterminez le nombre de cycles utilisés par la procédure `tempo(1,0)`.
7. **Manip :** Recommencez l'étape 5. en changeant les paramètres de la procédure `tempo( , )` afin de compléter le tableau suivant :
   (CR20) : A l'aide d'un fichier excel (ou libreoffice calc), tracer deux graphiques XY : le premier

nombre de boucles complètes (nb)	taille de la dernière boucle (tb)	nombre de cycles utilisés (nc)
1	0	
5	0	
10	0	
15	0	
0	5000	
0	10000	
0	20000	
0	40000	
0	65535	

donnant le nombre de cycles en fonction du nombre de boucles complètes $nc = f(nb)$ (partie supérieure du tableau) et le deuxième donnant le nombre de cycles en fonction du nombre d'itérations $nc = g(tb)$ (partie inférieure du tableau). Ces 2 courbes doivent apparaître dans votre CR. Que pouvez-vous en déduire ?

CR21 : Essayez d'expliquer pour quelle raison l'appel à `tempo(1,0)` et l'appel à `tempo(0,65535)` donnent un nombre de cycles assez différent (environ 15% d'écart) [3].

CR22 : Sachant que la fréquence utilisée est de 8MHz, donnez la durée d'un cycle, puis indiquez en justifiant vos choix, le nombre de boucles complètes ainsi que la taille de la dernière boucle pour que la durée de l'appel à la procédure `tempo` soit de 500ms.

9. **Manip :** Vérifiez que vos paramètres sont corrects en faisant clignoter les LEDs sur la carte. [4]
10. **Manip :** En utilisant la procédure `tempo` avec les bons paramètres, proposez un code permettant d'allumer une seule led, toutes les 500ms, en commençant par la led D1, puis la led D2, etc... jusqu'à la led D8, puis de recommencer par la led D1 et de continuer le cycle.
    Votre code doit passer à l'échelle, c'est à dire que si on disposait de 10000 leds, un changement minime dans le code permettrait de l'adapter à cette situation).

    Une et une seule led doit être allumée à un instant donné.

Faites-le vérifier par votre enseignant.

Séance/heure	:
Signature	:

Vous pouvez préparer les questions jusqu'à la CR28 du chapitre 5 en autonomie avec CW installé et en mode `Full Chip Simulation` pour gagner du temps en séance.

3. Vous pourrez par exemple mettre des points d'arrêt dans les boucles de la procédure `tempo` et voir, à l'aide de la notion de hierarchie mémoire vue en cours, la durée que prennent les itérations dans chacune des boucles.

4. Vous pouvez utilisez un chronomètre (oui oui vous pouvez sortir votre portable pour cette tâche et le ranger immédiatement après !) et comptez le nombre de clignotements.
Attention, la fréquence de 8MHz n'est pas très précise selon les microcontrôleurs. Donc c'est approximativement 500ms.

Chapitre 5

Interruption sur le timer

Objectif

L'objectif de ce TP est d'étudier les différentes méthodes permettant de régler la fréquence de clignotage des LEDs.

Dans le premier chapitre, vous avez découvert la carte et fait le lien avec la DataSheet du microcontrôleur.

Dans le deuxième chapitre, vous avez découvert le logiciel permettant de placer du code dans le microcontrôleur et fait le lien entre les déclarations spécifiques du microcontrôleur et la DataSheet.

Dans le troisième chapitre, vous avez pu programmer le microcontrôleur pour allumer des LEDs en utilisant les interrupteurs et le bouton poussoir sur la carte de développement.

Dans le quatrième chapitre, vous avez fait le lien entre l'horloge, le nombre de cycles pour effectuer une tâche et ainsi pu créer une temporisation.

Dans ce chapitre, vous verrez le principe de fonctionnement des interruptions et constaterez qu'il est préférable d'utiliser une interruption plutôt qu'une procédure de temporisation.

Vous pourrez également effectuer les exercices de synthèse suggérés en 5.2.

5.1 Utilisation d'une interruption sur le timer

5.1.1 Introduction au mécanisme d'interruption

Etude du mécanisme d'interruption basé sur le timer

Dans la section précédente, une procédure de temporisation a été créée.
Lorsqu'elle est appelée, cette procédure consomme du temps CPU simplement pour incrémenter une variable ce qui empêche le CPU, pourtant plus généraliste, de faire d'autres instructions.

Pour palier ce problème, on recourt au mécanisme d'interruption.

Une **interruption** est un événement (interne ou externe) capable d'interrompre le déroulement d'un programme pour exécuter immédiatement et automatiquement une routine de service en réponse à cet événement. Plus de détails sont donnés dans le chapitre 5 du fascicule de cours que nous vous invitons à lire pour plus de généralités.
Nous nous intéressons ici à un évènement particulier qui est l'interruption sur le timer. Le timer est simplement réalisé grâce à un compteur qui compte les fronts montants d'un signal d'horloge (éventuellement prédivisé avant d'arriver sur le port d'horloge du compteur). Ceci permet de mesurer un nombre de cycles (entre deux fronts montants) et donc de mesurer un temps de manière assez précise.

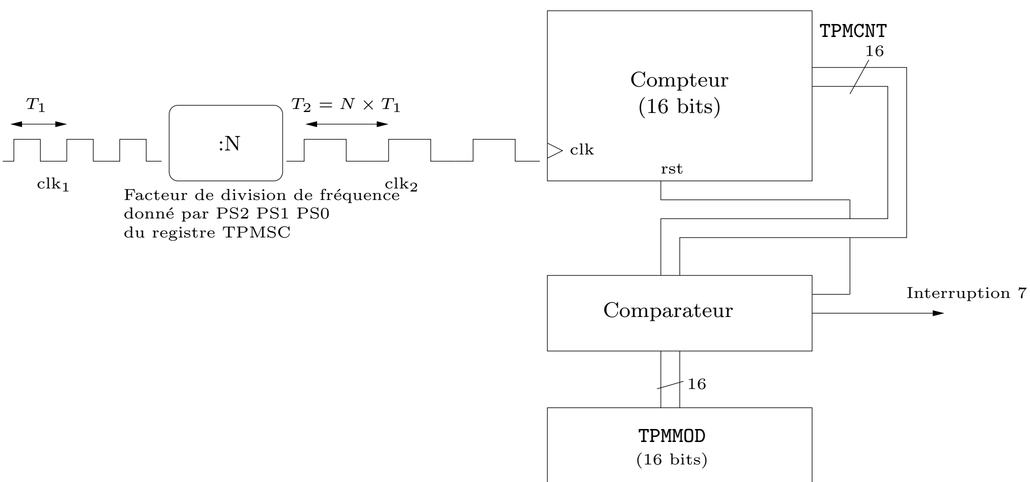


Figure 5.1 – Principe de fonctionnement d'une interruption sur le timer.

Le compteur est incrémenté à chaque front montant du signal d'horloge clk_2 **indépendamment** du travail du CPU.

La valeur de compteur, stockée dans le registre `TPMCNT`, obtenue en sortie du compteur est comparée à la valeur contenue dans le registre `TPMMOD` grâce au comparateur.

Si les deux valeurs sont identiques, le comparateur envoie deux informations :
- un signal reset au compteur qui permet de le remettre à 0 ;
- un signal d'interruption avec niveau de priorité 7 au processeur.

Sinon, le compteur continue à compter à chaque nouveau front montant du signal clk_2.

Du point de vue du processeur qui reçoit le signal d'interruption, il stoppe la tâche courante et sauvegarde le contexte (ie : l'adresse de l'instruction qu'il aurait dû exécuter avant l'interruption ainsi que les données de ces registres liées à l'instruction) dans la pile système.
La ressource processeur étant libérée, il est alors possible d'exécuter un ensemble d'instructions en réponse à l'événement ayant provoqué l'interruption.
Une fois toutes les instructions de l'interruption exécutées, le processeur récupère, sur la pile système, l'instruction qu'il était en train de traiter ainsi que les données associées avant l'interruption et reprend son fonctionnement "normal" jusqu'à la prochaine interruption.

Configuration de l'interruption basée sur le timer

Dans notre cas, le registre permettant de paramétrer l'horloge utilisée et d'autoriser l'interruption sur le timer est le registre `TPMSC` (Timer Pulse-Width Modulation Status and Control). Vous pouvez utiliser la documentation et faire une recherche sur `TPMSC` pour comprendre de quoi est constitué ce registre.

Si le bit `TOIE` de ce registre est à 1, les interruptions basées sur le dépassement de valeur du compteur 16 bits sont autorisées.

Le dépassement est renseigné par le bit (flag) `TOF` (Timer OverFlow) qui passe à 1 lorsque la valeur du compteur atteint celle située dans le registre `TPMMOD`.

Il est donc indispensable de remettre le bit `TOF` de ce registre à 0 **à la fin** de l'interruption pour que celle-ci puisse prendre fin.

La syntaxe qui permet d'écrire l'interruption liée au timer (avec priorité 7) est la suivante [1] :

```
void interrupt 7 Depassement_compteur(void){
/*On interrompt le processeur pour faire quelque chose*/
/*Ce quelque chose se traduit par des instructions à
traiter pendant l'interruption qu'il faut positionner ici*/


//Attention à bien remettre à 0 le bit TOF après avoir traité les instructions de l'interruption
TPMSC=TPMSC & 0b01111111;
}//fin de l'interruption
```

1. Le nom de la procédure (ici `Depassement_compteur`) n'a pas d'importance.
le mot clé `interrupt` indique qu'il s'agit d'une interruption et qu'elle peut être déclanchée à partir d'un événement extérieur et non pas par un appel fait par le développeur.
Le chiffre 7 indique la priorité de cette interruption. Pour information, la priorité 0 est réservée au reset.

Les interruptions en simulation (mode Full Chip Simulation)

1. **Manip :** Créez un nouveau projet et créez un fichier `interruption_timer.c` dans lequel vous définirez l'interruption précédente (telle quelle pour le moment, on interrompera le processeur pour ne rien faire).

   N'oubliez pas d'inclure le fichier `derivative.h` en en-tête de fichier.
2. **Manip :** Ajoutez ce fichier à votre dossier source dans votre arborescence de projet (Clic droit sur Sources, Add Files).
3. **Manip :** On vous donne le code suivant dans lequel XXX et Y seront à remplacer par des valeurs que nous définirons par la suite.

```
#include <hidef.h>
#include "derivative.h"

void main(void) {
  /*Autorisation des interruptions */
  EnableInterrupts;
  /* Selection horloge interne à 8MHz */
  ICSC2 = 0x00;
  ICSC1 = 0x04;
  /* Désactivation du Watchdog */
  SOPT1_COPE=0;

  /* Initialisation de TPMSC */
  TPMSC=0b01001XXX;
  /* Définition de la valeur max de comptage */
  TPMMOD=0xYYYY;

  PTBDD=0xFF;
  PTBD=0;
  PTADD=0;
  for(;;) {

  } /* loop forever */
}
```

   (a) **Manip :** Remplacez XXX par 000 et YYYY par 64 (64 en hexadécimal = 100 en décimal) et compilez votre code (touche Make), puis, en mode Full Chip Simulation, appuyez sur la touche Debug.

   (b) **Manip :** A l'aide de la documentation du micro-contrôleur (chapitre 4 : Memory Map), repérez dans la fenêtre Memory les emplacements de `TPMCNT` (16 bits), `TPMMOD` (16 bits), `TPMSC` (8 bits) ainsi que l'emplacement des deux registres de 8 bits : `ICSC1` et `ICSC2`.

   (c) **Manip :** En exécutant le programme en mode pas à pas (touche F11) :
   — vérifiez que les contenus de `TPMCNT` et `TPMSC` évoluent dans la fenêtre Memory.
   — relancez votre programme[2] (toujours en pas à pas) et complétez le tableau Tab.5.1. Avant chacune des instructions mentionnées, notez le nombre de cycles CPU consommés (fenêtre Register) et la valeur de `TPMCNT`.

   (d) Complétez la phrase suivante : `TPMCNT` s'incrémente tous les ______ cycles.

2. Fermez la fenêtre « True Time simulator & ... » et relancer le programme en appuyant sur la touche debug pour repartir d'un nombre de cycles CPU faibles

	XXX=000		XXX=001		XXX=010	
Avant l'instruction	nb_cycles	TPMCNT	nb_cycles	TPMCNT	nb_cycles	TPMCNT
`SOPT1_COPE=0;`						
`PTBDD=0xFF;`						
`PTBD=0;`						
`PTADD=0;`						
`void interrupt 7 ...{`						
`TPMSC=TPMSC & 0b01111111;`						

Table 5.1 – Lien entre nombre de cycles et valeur de TPMCNT

(e) **CR23 :** Utilisez le lien entre les valeurs de `TPMCNT` et le nombre de cycles pour indiquer si la valeur `TPMMOD` est atteinte par `TPMCNT`.

(f) **CR24 :** En utilisant les valeurs de `TPMCNT` observées durant l'interruption, est ce que le comptage se poursuit pendant l'interruption ou reprend seulement à la fin de l'interruption ?

(g) **CR25 :** Quel serait le risque à mettre une boucle (finie)[3] `for()` ou `while()` dans l'interruption ?

4. **Manip :** Reprenez les questions (a) à (d) avec XXX=001 puis avec XXX=010
   Pour la question (d), vous pourrez compléter :
   — pour XXX=001, `TPMCNT` s'incrémente tous les [ ] cycles.
   — pour XXX=010, `TPMCNT` s'incrémente tous les [ ] cycles.
5. **CR26 :** Recherchez les informations sur les 3 bits de poids faible de `TPMSC` dans la documentation et indiquez quel est leur rôle.
   En vous servant des résultats précédents, prévoyez le nombre de cycles CPU nécessaires pour que `TPMCNT`, en partant de 0, atteigne `TPMMOD` lorsque XXX=100.
6. **Manip :** Vérifiez-le. (vous pouvez par exemple mettre un point d'arrêt au début de la ligne
   `void interrupt 7 Depassement_compteur(void){`
   et exécuter votre programme normalement (touche F5), il s'arrêtera à chaque interruption, donc entre deux cycles complets de comptage).
7. **CR27 :** En déduire la formule générale donnant le nombre de cycles CPU entre deux interruptions en fonction de `TPMMOD` et des 3 bits de poids faibles de `TPMSC` : `PS2`, `PS1` et `PS0`.
8. **CR28 :** Nous souhaitons provoquer une interruption toutes les 500ms. Pour celà, il faut déterminer, en s'appuyant sur la figure 5.1, la configuration donnée des registres `ICSC1` et `ICSC2` déterminant la fréquence d'horloge et sur les résultats précédents :
   — le facteur de division de **fréquence** N codé en binaire sur les 3 bits `PS2`, `PS1` et `PS0` du registre `TPMSC` ;
   — la valeur du registre `TPMMOD`.

   (a) Rappelez pourquoi `TPMCNT`$< 2^{16}$.
   En sachant que $N = 2^{PS_2 PS_1 PS_0}$ et que le temps mis par le compteur pour parcourir toutes les valeurs de 0 à `TPMMOD` doit être égal à 500ms, trouvez la plus petite valeur de N acceptable puis la valeur des bits $PS_2 PS_1 PS_0$ qui permettent de configurer une partie du registre `TPMSC` dans le code fourni.

3. Comme la procédure tempo() par exemple.

(b) En déduire enfin la valeur de `TPMMOD` pour que le compteur soit remis à 0 toutes les 500ms. Cette valeur devra être utilisée dans le code fourni.

Mise en place de l'interruption pour faire clignoter les LED

1. Quelle instruction doit contenir votre interruption pour changer l'état des LEDs toutes les 500ms ? [ ]
2. **Manip :** Pour illustrer le fait que le processeur peut travailler indépendamment du compteur, proposez un code dans la boucle `for(;;)` du programme principal qui permet d'inverser l'état des LEDs `D5` à `D8` à chaque pression du bouton poussoir [4].
   Puis proposer une interruption sur le timer qui permette de faire clignoter les LEDs `D1` à `D4` toutes les 500ms.
3. **Manip :** Vérifiez qu'en ayant rajouté ces instructions dans la boucle `for(;;)`, le nombre de cycles entre deux interruptions ne change pas.
4. **Manip :** Proposez une interruption sur le timer dont le rôle est d'allumer successivement et cycliquement les leds `D1` à `D8`. A tout instant, une et une seule led doit être allumée (comme dans la question posée à la fin du chapitre 4 qui utilisait la procédure de temporisation).

   Encore une fois votre code doit passer à l'échelle
   (que se passerait-il s'il y avait 10000 leds au lieu de 8 ?)

Faites vérifier le code et le résultat par un enseignant :

Séance/heure	:
Signature	:

5.2 Exercices de synthèse

L'objectif est de réussir à traiter, de manière autonome, ces exercices classés par niveau de difficulté.

Comptage/Décomptage (*)

Créer un programme qui permet de compter à chaque commutation (changement d'état) de l'interrupteur `SW1`, décompter à partir de chaque commutation de l'interrupteur `SW2`.

Attention au phénomène de rebonds mécaniques.
La valeur de comptage sera visualisée en binaire sur les leds D8 à D1. Cette valeur sera remise à 0 lors d'un appui sur le bouton poussoir.

Il n'y a pas besoin d'utiliser le mécanisme d'interruption dans cet exercice. Faites vérifier le code et le résultat par un enseignant :

Séance/heure	:
Signature	:

Chronomètre (*)

En utilisant des interruptions, réalisez un chronomètre qui compte de seconde en seconde. La valeur de comptage sera visualisée en binaire sur les leds D8 à D1.

4. En séance, on peut constater qu'il y a un phénomène de rebonds sur le bouton poussoir. Il faut donc rajouter une temporisation tempo(1,0) par exemple au moment du test de l'appui ET du relâchement (le `else`)

- départ/arrêt indiqué par l'interrupteur `SW1`
- comptage/décomptage indiqué par l'interrupteur `SW2`
- remise à 0 à l'aide du bouton poussoir.

Faites vérifier le code et le résultat par un enseignant :

Séance/heure	:
Signature	:

Pour finir vous pouvez traiter l'un (ou les deux !) des deux exercices plus exigeants ci-dessous selon celui qui vous motive le plus.

Clignotants dynamiques (***)

Pour l'exercice suivant, vous devez fournir dans votre CR, les captures d'écran de :
- vos codes commentés (fichiers `main.c`, `fonctions.c`, `raccourcis.h`, etc...)
- les fenêtres Memory et Register commentées illustrant le bon fonctionnement de votre programme.

On souhaiterait utiliser le SW1 pour simuler une mise en route d'un clignotant dynamique (type Audi) gauche et SW2 pour simuler celle d'un clignotant droite.
Sur l'action de SW1 (resp. SW2), un cycle composé de 2 étapes démarre :

1. les leds doivent s'allumer successivement de la droite vers la gauche (resp. de la gauche vers la droite) jusqu'à atteindre un état où elles sont toutes allumées.
   L'allumage successif doit durer en totalité 400ms et les leds doivent rester allumer 100ms
2. les leds doivent ensuite s'éteindre successivement de la droite vers la gauche (resp. de la gauche vers la droite) jusqu'à atteindre un état où elles sont toutes éteintes.
   L'extinction successive doit durer en totalité 400ms et les leds doivent rester éteintes pendant 100ms.

Le cycle dure 1 seconde en totalité et se répète tant que SW1 (resp. SW2) est actif.
Si les deux interrupteurs sont actionnés en même temps (warning), les leds doivent clignoter toutes les 500ms.
Enfin lorsque les deux interrupteurs sont inactifs, les leds doivent rester éteinte.
Un appui sur le bouton poussoir doit permettre d'allumer toutes les leds en continu tant que le bouton poussoir est maintenu appuyé.
Cet appui doit être prioritaire sur toutes les autres actions.

Un jeu de reflexe (***)

L'objectif de cet exercice est de proposer un jeu dans lequel les leds s'allument successivement et cycliquement de `D1` à `D8` à une certaine fréquence f_{cligno}. Comme dans les chapitres précédents, une et une seule led doit être allumée à un instant donné. $\frac{1}{f_{cligno}}$ désigne donc la durée en secondes entre l'allumage de deux leds consécutives.
Les règles du jeu sont les suivantes :
- Lorsque la led `D7` est allumée, le joueur doit appuyer sur le bouton poussoir pour incrémenter son score.
- Lorsque la led `D7` est éteinte, le joueur doit relacher le bouton poussoir sinon son score est décrémenté. Il faut donc appuyer au bon moment et juste au bon moment !

Lorsque le score maximal est atteint (vous pouvez le fixer en dur dans le programme à la valeur 10 par exemple ou définir une constante non signée de 8 bits `SCORE_MAX` initialisée à la valeur de votre choix, mais impérativement inférieure à 63), les leds doivent s'allumer cycliquement comme précédemment

mais à une fréquence plus élevée pour indiquer que le jeu est terminé.

On disposera des périphériques d'entrée étudiés dans ce cahier de TP :
- Bouton poussoir BP
- Interrupteur SW1
- Interrupteur SW2 (optionnel)

Partie 1 : utilisation de BP et SW1 uniquement

On distinguera deux modes de fonctionnement possibles pendant le jeu :

1. Mode score : lorsque SW1 est inactif, le score doit être affiché, en binaire, sur les leds `D6` à `D1` (score maximum de 63 donc) et la led `D7` doit clignoter à la fréquence f_{cligno} indiquant la difficulté du jeu (plus la fréquence est élevée, plus il sera difficile d'appuyer au moment où la led `D7` est allumée.
   Dans ce mode, un appui sur BP déclenchera la réinitialisation du score à 0, permettant ainsi de commencer une nouvelle partie.
2. Mode jeu : lorsque SW1 est actif, la séquence de clignotement cyclique des leds reprend et le programme doit prendre en compte les appuis sur le bouton poussoir du joueur pour incrémenter ou décrémenter son score.

Le joueur, peut à tout moment, commuter SW1 pour consulter son score et reprendre ensuite le jeu. Le score doit donc être mémorisé pour ne pas être réinitialisé à chaque commutation du SW1.

Attention au phénomène de rebonds mécaniques lors de l'action de BP ou de SW1.
Les rebonds mécaniques sont supposés être masqués si entre deux appuis sur BP un temps de ≈ 100ms s'écoule. Vous pourrez utiliser la procédure `tempo` avec les bons paramètres pour représenter ce temps. La led `D7` ne peut donc pas clignoter à une fréquence plus grande que $\frac{1}{200e-3}$Hz= 5Hz, comme il y a 8 leds, l'allumage cyclique doit donc se faire à une fréquence maximale f_{cligno} de 40Hz.
Vous pouvez donc régler la fréquence f_{cligno}, c'est à dire paramétrer l'interruption devant gérer l'allumage cyclique des leds avec n'importe quelle valeur n'excédent pas 40Hz (ce qui est déjà beaucoup et représente un niveau de difficulté important ! En pratique on n'excédera pas 20Hz pour que le jeu reste jouable par un humain).

Partie 2 (optionnelle) : Réglage du mode de difficulté

En utilisant SW2, il est possible de définir un mode de saisie de difficulté.

1. Lorsque SW2 est inactif : on doit retrouver le fonctionnement précédent décrit en partie 1.
2. Lorsque SW2 est actif, on souhaite :
   - à chaque appui sur le bouton poussoir BP, changer cycliquement le niveau de difficulté (qui correspond à changer la fréquence f_{cligno} et donc à paramétrer la durée de l'interruption) :
     - Niveau 0 : $f_{cligno} = 1$Hz
     - Niveau 1 : $f_{cligno} = 2$Hz
     - Niveau 2 : $f_{cligno} = 4$Hz
     - Niveau 3 : $f_{cligno} = 6$Hz
     - Niveau 4 : $f_{cligno} = 10$Hz
     - Niveau 5 : $f_{cligno} = 16$Hz

     Un appui sur BP alors que le niveau de difficulté 5 est atteint doit ramener au niveau de difficulté 0.
   - Voir la led `D7` clignoter à la fréquence correspondant à la difficulté choisie.
   - Voir le niveau de difficulté (entre 0 et 5) s'afficher sur les leds `D3` à `D1` en binaire.

Lorsque l'un (ou les deux !) des deux exercices précédents est réalisé, appelez l'enseignant pour validation

Séance/heure	:
Signature	:

Chapitre 6

[Optionnel] Modulation de Largeur d'Impulsion

Objectif

L'objectif de ce TP est d'étudier la modulation de largeur d'impulsion pour augmenter ou diminuer l'intensité lumineuse d'une LED.

Dans le premier chapitre, vous avez découvert la carte et fait le lien avec la DataSheet du microcontrôleur.

Dans le deuxième chapitre, vous avez découvert le logiciel permettant de placer du code dans le microcontrôleur et fait le lien entre les déclarations spécifiques du microcontrôleur et la DataSheet.

Dans le troisième chapitre, vous avez pu programmer le microcontrôleur pour allumer des LEDs en utilisant les interrupteurs et le bouton poussoir sur la carte de développement.

Dans le quatrième chapitre, vous avez fait le lien entre l'horloge, le nombre de cycles pour effectuer une tâche et ainsi pu créer une temporisation.

Dans le cinquième chapitre, vous avez vu le principe de fonctionnement des interruptions et constaté qu'il est préférable d'utiliser une interruption plutôt qu'une procédure de temporisation.

Dans ce chapitre, vous allez travailler sur la notion d'intensité lumineuse des LEDs et voir qu'il est possible de la faire varier. Dans un premier temps, vous étudierez la notion de modulation de largeur d'impulsion et de rapport cyclique d'un signal. Vous verrez ensuite que le timer vu au TP5 permet de générer un tel signal dont on peut modifier le rapport cyclique en configurant les registres ad-hoc. Enfin, vous verrez qu'il est possible de faire varier dynamiquement l'intensité de la LED à l'aide d'un potentiomètre relié à un convertisseur analogique numérique.

6.1 PWM : premières notions (à préparer)

La Modulation de Largeur d'Impulsion (MLI ou en anglais PWM : Pulse Width Modulation) est une méthode permettant, à partir d'un signal logique (deux états seulement : 1 ou 0), de générer n'importe quelle valeur moyenne de ce signal comprise entre 0 et 1 sur une durée donnée. Cette valeur moyenne, également appelée le *rapport cyclique* du signal, permet de définir la durée à l'état haut et la durée à l'état bas pour une période T donnée.
On s'intéresse aux chronogrammes de la figure 6.1.

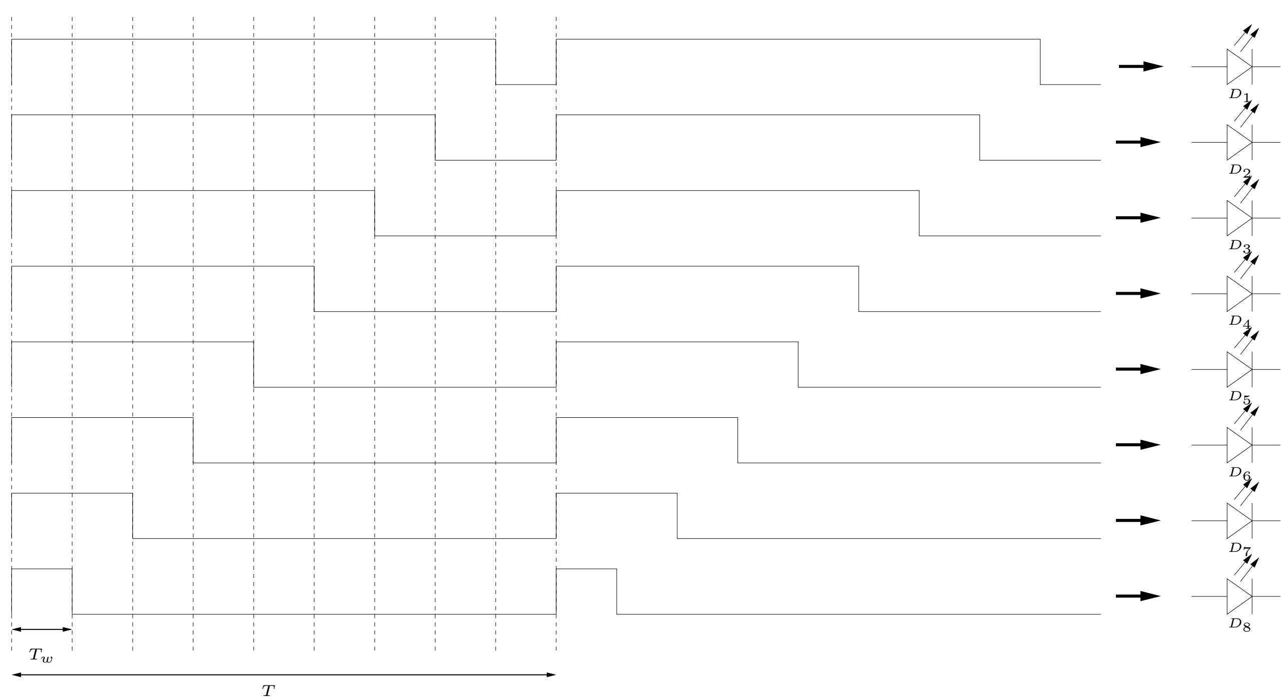


FIGURE 6.1 – Rapport cyclique et intensité lumineuse

1. Donner les rapports cycliques (ie : les valeurs moyennes) des signaux entrant sur chacune des LEDs D_1 à D_8 sur la période T.

   Rapport cyclique D1 :

   Rapport cyclique D2 :

   Rapport cyclique D3 :

   Rapport cyclique D4 :

   Rapport cyclique D5 :

   Rapport cyclique D6 :

   Rapport cyclique D7 :

   Rapport cyclique D8 :

2. **(CR29) :** Donnez la valeur de T_w en fonction de T pour obtenir les rapports cycliques 50%, 0%, 100%, 25% et 75% sur le signal arrivant sur la LED D_8. Prévoyez ce qui va se passer au niveau de l'éclairage de la LED D_8 dans chacun de ces cas.

3. **Manip :** Proposer un code en C[1] sous CodeWarrior permettant d'allumer les 8 LEDs de la carte de développement avec des intensités différentes tel que présenté sur la figure 6.1.
   - L'intensité maximale est elle atteinte ?
   - L'intensité minimale est-elle atteinte ?
   - Que faudrait-il modifier dans le code pour les atteindre ?
   - Effectuer ces modifications et tester.

1. Vous pouvez réutiliser la procédure `tempo( , )` pour représenter la durée T_w

4. **(CR30) :** Que se passe-t-il au niveau des leds si, pour un rapport cyclique donné, on choisit T_w grand ? T_w petit ?
5. **Manip :** Tester en modifiant les paramètres de la procédure `tempo( , )`.
6. Complétez avec `grande` ou `petite` :
   le rapport cyclique d'un signal de [   ] période permet d'influencer l'intensité lumineuse des LEDs.

6.2 Pulse-Width Modulator

Comme pour la gestion d'une temporisation à la fin du chapitre 4 qui peut se faire en utilisant une interruption sur le timer, il serait possible de remplacer l'appel aux procédures `tempo(,)` par de telles interruptions (**Exercice recommandé à titre personnel en vous rappelant l'intérêt d'utiliser une interruption plutôt qu'une procédure de temporisation**).

Il se trouve que le Timer/Pulse-Width Modulator (TPM) du microcontrôleur `MC9S08QG8`, vu au chapitre 5, peut être configuré pour générer un signal tel que ceux présentés dans la figure 6.1. C'est à dire pouvant paramétrer la valeur de T_w par rapport à T, autrement dit permettant de régler le rapport cyclique du signal.
Ce signal sort sur le canal 1 du TPM : `TPMCH1` qui est connecté à la broche `PTB5` du microcontrôleur comme le montre la figure 6.2

⚠ On pourra, en utilisant le TPM en mode PWM, agir uniquement sur l'intensité lumineuse de la LED D_6 connectée à `PTB5`.

Configuration des registres du TPM

Pour répondre à la 1ère question, il faudra effectuer une lecture attentive de la documentation (pp229-244).

1. **(CR31) :** Sachant qu'on veut configurer la `PWM` en mode `edge-aligned`, que l'horloge de référence doit être `BUSCLK` sans prédivision, qu'on ne veut pas utiliser d'interruption sur le timer, donner la configuration des registres `TPMSC` et `TPMC1SC`.
   Pourquoi ne s'intéresse-t-on qu'à `TPMC1SC` et pas aux autres registres `TPMCnSC` (pour `n` $\neq 1$) ?
2. **Manip :** Créer un nouveau projet `LED_intensity` dans lequel :
   (a) Vous créerez un fichier `fonctions.c` (et son fichier `fonctions.h` correspondant) contenant les définitions (resp. déclarations) des procédures suivantes :
      — `Init_uc` : permettant d'initialiser les ports d'entrées/sorties du microcontrôleur dans les bonnes directions, de choisir la fréquence d'horloge à 8MHz (cf : TPs précédents avec les registres `ICSC1` et `ICSC2`) et de désactiver le Watchdog ;
      — `Init_TPM` : permettant d'initialiser les registres du TPM selon les valeurs trouvées à la question CR31.
      — Après lecture de la documentation [2] (p241 : §16.4.2.3), indiquer quels sont les registres permettant de régler la durée de la période (notée T dans la section 6.1) et celle de la largeur du pulse (notée T_w dans la section 6.1).
   (b) Vous modifierez le contenu de la fonction `main` de votre fichier `main.c` de manière à :
      — Utiliser les fonctions d'initialisation `Init_uc` et `Init_TPM`.
      — Proposer, dans la boucle principale, des valeurs pour les registres représentant les périodes T et T_w afin d'obtenir les rapports cycliques suivants : 50%, 0%, 100%, 25% et 75%.

2. On rappelle que le signal modulé en largeur d'amplitude sort sur le canal 1 du TPM.

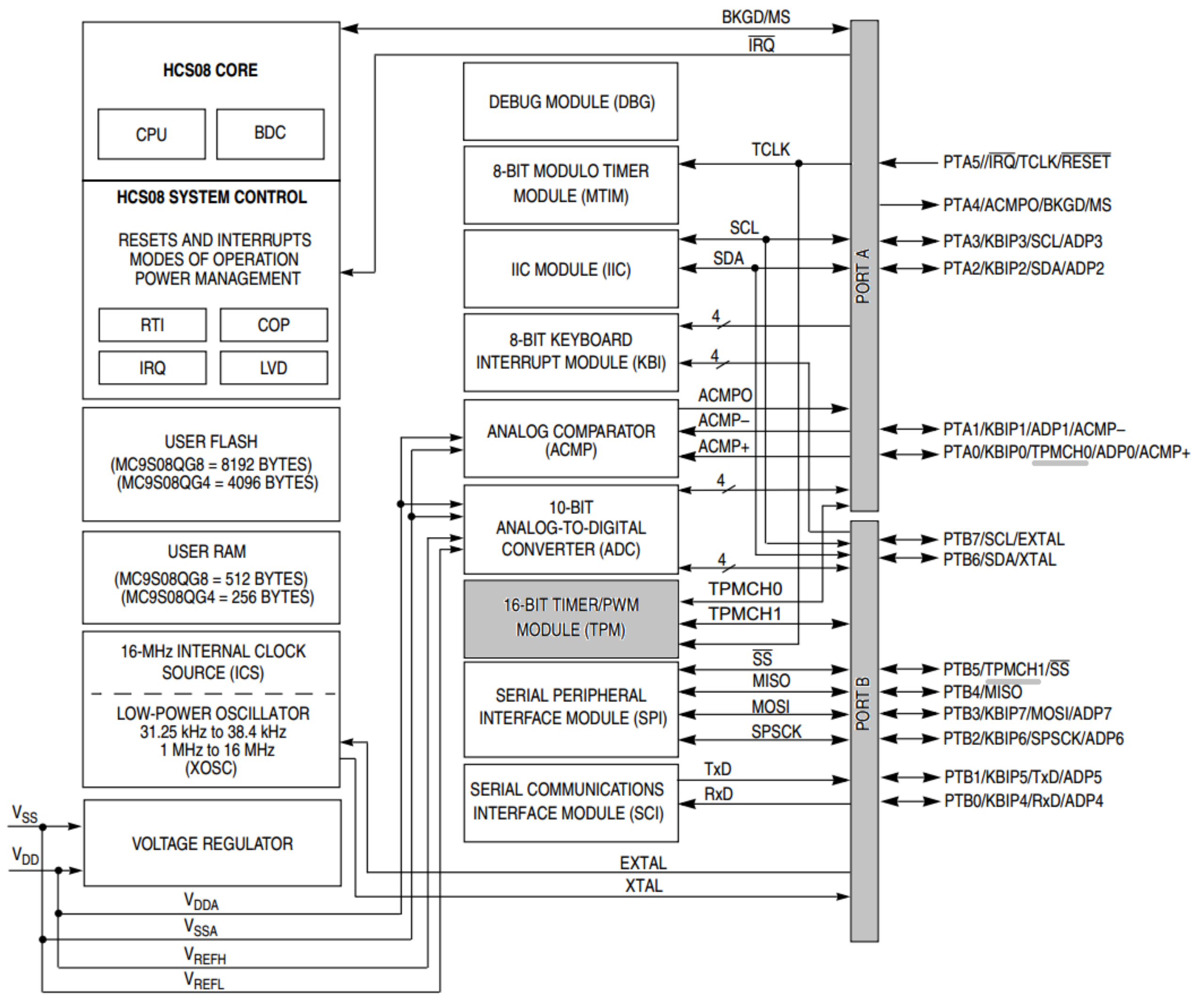


FIGURE 6.2 – MC9S08QG8 et TPM : schéma bloc

— Pour un rapport cyclique donné, modifier la valeur du registre représentant la période T du signal sortant sur la LED D_6
(CR32) : Est-ce que celà change quelque chose sur l'état de la LED ? Pourquoi ?

3. **Manip :** Proposer à présent, dans la boucle principale du `main.c`, un code permettant de faire varier automatiquement l'intensité de la LED `D6`, de l'intensité la plus faible à l'intensité la plus forte puis de l'intensité la plus forte à l'intensité la plus faible et ainsi de suite.

Faites vérifier le code et son exécution par votre enseignant.

Séance/heure	:
Signature	:

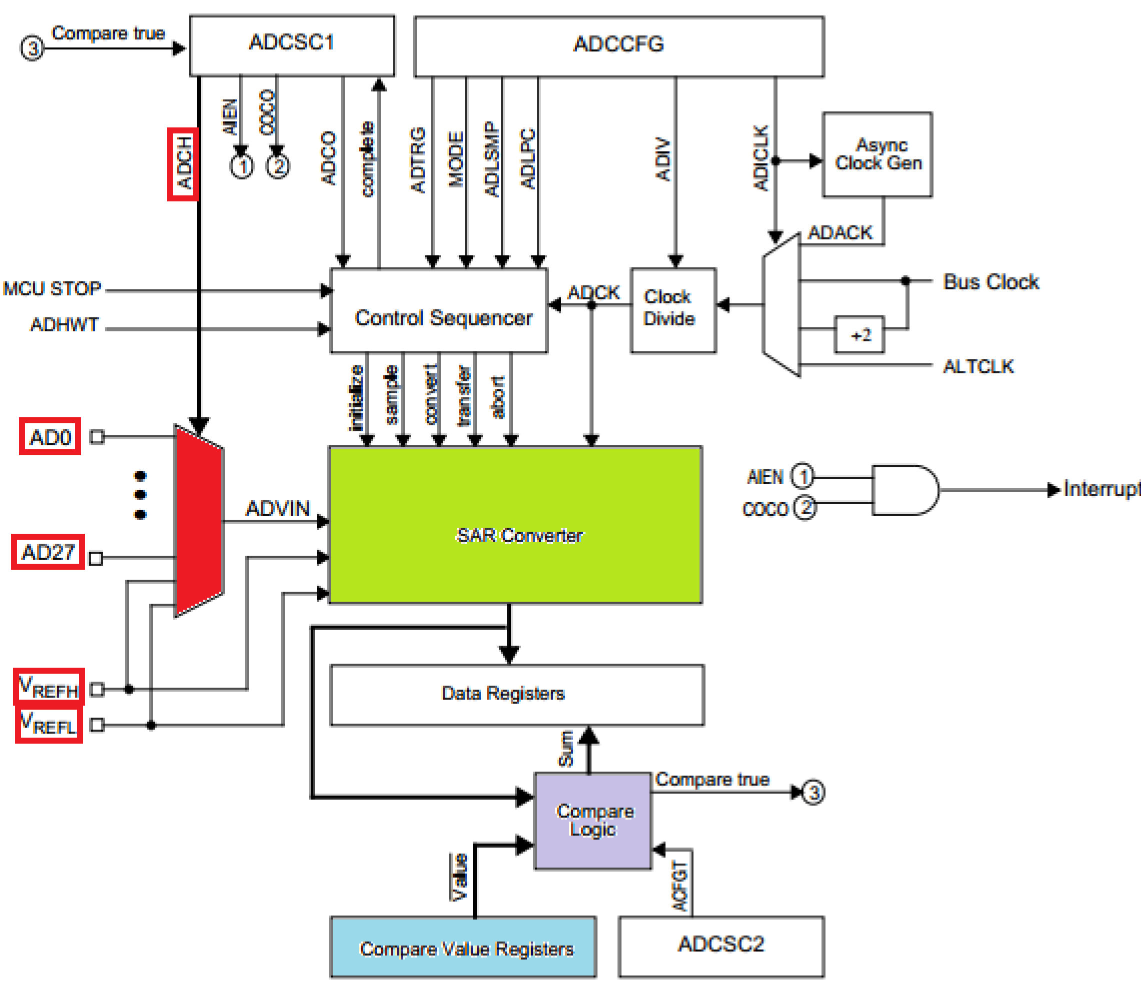
6.3 Contrôler l'intensité de la LED : utilisation d'un potentiomètre et d'un CAN

Dans la section précédente, nous devions changer la valeur du rapport cyclique "en dur" dans le programme (en modifiant la valeur des registres représentant les valeurs T et T_w), puis recompiler et reprogrammer le microcontrôleur pour observer un changement de l'intensité de la LED.

Dans cette section, nous allons voir qu'il est possible de modifier l'intensité de la LED dynamiquement en utilisant un potentiomètre relié à un Convertisseur Analogique-Numérique.

Introduction : informations générales

Un potentiomètre est un sytème contrôlant une résistance variable qui permet de générer des niveaux de tension variant entre 0 et Vcc volts. Ces niveaux de tension sont présents sur la patte du micro-contrôleur reliée au potentiomètre (`PTA0`).
Par rapport aux bouton poussoir et interrupteurs qui permettaient d'avoir uniquement deux niveaux logiques (0 volt ou Vcc volts), le potentiomètre permet de représenter plusieurs valeurs dans cette plage de tension. Ces différentes valeurs ne peuvent plus être représentées par un seul bit.
Il y a, en théorie, un nombre infini de valeur de tension entre 0 et Vcc volts. En pratique, celà va dépendre d'un composant dont le rôle est de convertir ces grandeurs analogiques en grandeurs numériques (seules informations intelligibles par un système informatique). Ce composant est appelé un *Convertisseur Analogique Numérique* (CAN) ou *Analog to Digital Converter* (ADC). Dans la suite, on utilisera la terminologie ADC pour parler du Convertisseur Analogique Numérique.
Pour simplifier la compréhension, nous nous intéresserons uniquement aux éléments colorés sur le dessin ci-dessous (tiré de la figure 9.2 de la datasheet).



Le bloc `SAR Converter` est le bloc chargé de la conversion analogique numérique. C'est un type de

convertisseur basé sur des approximations successives du niveau de tension analogique en entrée à l'aide d'un processus de dichotomie.
Ainsi, `ADVIN` représente le niveau de tension à convertir dans la plage limitée par `Vrefl` et `Vrefh` (typiquement : 0 et Vcc volts). On prendra Vcc= $3.3V$.
`ADVIN` est ainsi comparé au niveau de tension moyen entre `Vrefl` et `Vrefh` (ie : $\frac{V_{REFH}+V_{REFL}}{2}$). Si il est plus haut que le niveau moyen, on met le bit de poids fort du registre `ADCR` à 1 sinon on le met à 0 et on recommence le processus dans l'intervalle $[\frac{V_{REFH}+V_{REFL}}{2}, V_{REFH}]$ ou dans l'intervalle $[V_{REFL}, \frac{V_{REFH}+V_{REFL}}{2}]$ selon le résultat de la première comparaison.
Ce processus peut être répété en théorie à l'infini, mais en pratique, le registre `ADCR` contenant la valeur de la conversion a une taille limitée.

Analyse du principe de conversion analogique numérique et configuration des registres du ADC

1. **(CR33) :** Indiquer quelles sont les tailles possibles (en bits) de la valeur résultant de la conversion analogique numérique stockée dans le registre `ADCR`.
2. **(CR34) :** Trouver le registre permettant de configurer la conversion analogique numérique sur le nombre de bits le plus grand (meilleure précision) et donner sa configuration (on laissera les autres bits à leur valeur par défaut).
3. **(CR35) :** On suppose qu'il y a la valeur 2.783V sur `ADVIN`, donner le contenu du registre `ADCR` après conversion de cette valeur de tension selon le processus de dichotomie. De même (et c'est bien plus rapide) pour les valeurs de tension 0 et Vcc volts.
4. **(CR36) :** Quel est le bloc représenté en rouge [3] ? À quoi sert-il ?
5. **(CR37) :** Sur combien de bits devrait être codé `ADCH` ? pourquoi ? Vérifier le dans la datasheet.
6. **(CR38) :** Quelle est la voie qui doit être sélectionnée par ce bloc [4] ? En déduire la configuration du registre `ADCSC1` sachant qu'on souhaite faire la conversion en continu (une conversion suit automatiquement la précédente).
7. **(CR39) :** La patte d'entrée du microcontrôleur reliée au ADC doit être "marquée" comme n'étant pas une entrée/sortie. On doit donc désactiver le contrôle d'entrée/sortie au niveau de cette patte du microcontrôleur.
   (CR40) : Vérifier que le registre `APCTL1` permet de faire cette manipulation et proposer une valeur de configuration de ce registre.
   Les autres registres `APCTLx` (`x`$\geqslant$ 2) ont-ils besoin d'être configurés ? Pourquoi ?

Codage final : lien entre potentiomètre et LED

Après avoir répondu à ces questions préliminaires, vous êtes à présent en mesure de procéder au codage permettant de commander l'intensité de la LED `D6` en utilisant le potentiomètre.
Vous devez récupérer et compléter les fichiers contenant les fonctions précédentes (`Init_uc`, `Init_TPM`) :

1. **Manip :** Proposer (déclarer et définir) une fonction `Init_adc` permettant d'initialiser le convertisseur analogique numérique (configuration des registres du ADC vus aux questions 5) et 6)).
2. **(CR41) :** Rappeler quels sont les registres du `TPM` permettant de représenter les périodes T_w et T.

3. En cas de doute, il est possible de réouvrir votre cours de SIN1
4. Consulter le schéma de la carte de développement sur le cahier de TP ainsi que la datasheet

3. **CR42 :** Rappeler quel est le registre du `ADC` permettant de stocker la valeur (convertie en numérique) de la tension analogique en sortie du potentiomètre.
4. **CR43 :** Quelle est la valeur minimale (resp. maximale) de ce registre ? Dans quel cas sont-elles atteintes ?
5. **CR44 :** On souhaite que, lorsque la valeur minimale (resp. maximale) de ce registre est atteinte, la LED `D6` soit éteinte (resp. allumée). Quel doit être le rapport cyclique du signal `TPMCH1` dans chacun de ces cas ? Comment configurer `TPMMOD` pour obtenir ces rapports cycliques ?
6. **CR45 :** Déduire des questions précédentes, l'instruction de la boucle principale permettant de faire varier l'intensité lumineuse de la LED `D6` en fonction de la position du potentiomètre.
7. **Manip :** Procéder au codage final en reprenant toutes les informations précédentes afin de faire varier l'intensité lumineuse de la LED `D6` à l'aide du potentiomètre. Faites vérifier à l'enseignant.

Séance/heure	:
Signature	:

Annexe A

Pense-bête

D'après `http://www.ief.u-psud.fr/~jok/iut/HC12/CW_HC12.pdf`.

A.1 Définitions relatives à l'IDE

Projet

Un projet regroupe un ensemble de cibles ou d'autres projets. Il correspond logiquement à la programmation d'une application ou d'une bibliothèque.

Cible

Une cible est une configuration de projet destinée à fabriquer un fichier exécutable. Une cible se caractèrise par l'ensemble des fichiers sources et les options associées.

Groupe

Le groupe est une entité qui sert à organiser les fichiers sources de façon hiérarchique. Un groupe peut contenir des sous-groupes. Le groupe n'est pas un repertoire. Les sources d'un même groupe peuvent se trouver n'importe où dans la hiérarchie des fichiers.

Path

L'IDE cherche les fichiers sources dans un ensemble de repertoire précis : le chemin d'accès (access path). Ces chemins sont définis soit de façon absolue soit de façon relative. La définition des chemins doit être choisie de telle façon que l'IDE retrouve les fichiers sources même si le répertoire du projet est copié ou deplacé.

Stationery

Un stationery est un projet de référence déjà configuré. Il sert de base à la création de nouveaux projets. On peut l'assimiler à un modèle. Lors de la création d'un projet à partir d'un stationery, le repertoire correspondant est dupliqué en adaptant les noms au nouveau projet. Les sources qui n'y sont pas ne sont donc pas copiées, elles sont partagées.

A.2 Outils

Préprocesseur

Le préprocesseur effectue un prétraitement du TEXTE du fichier source et traite toutes les directives repérées par le symbole `#`, notamment les inclusions de fichiers entête, les définitions de constantes par des macros et les directives de compilation conditionnelles (`#if #endif`).

Bibliothèques

Les fichiers objets (c'est à dire déjà compilés) fréquemment utilisés peuvent être regroupés dans des bibliothèques (library).

Linker

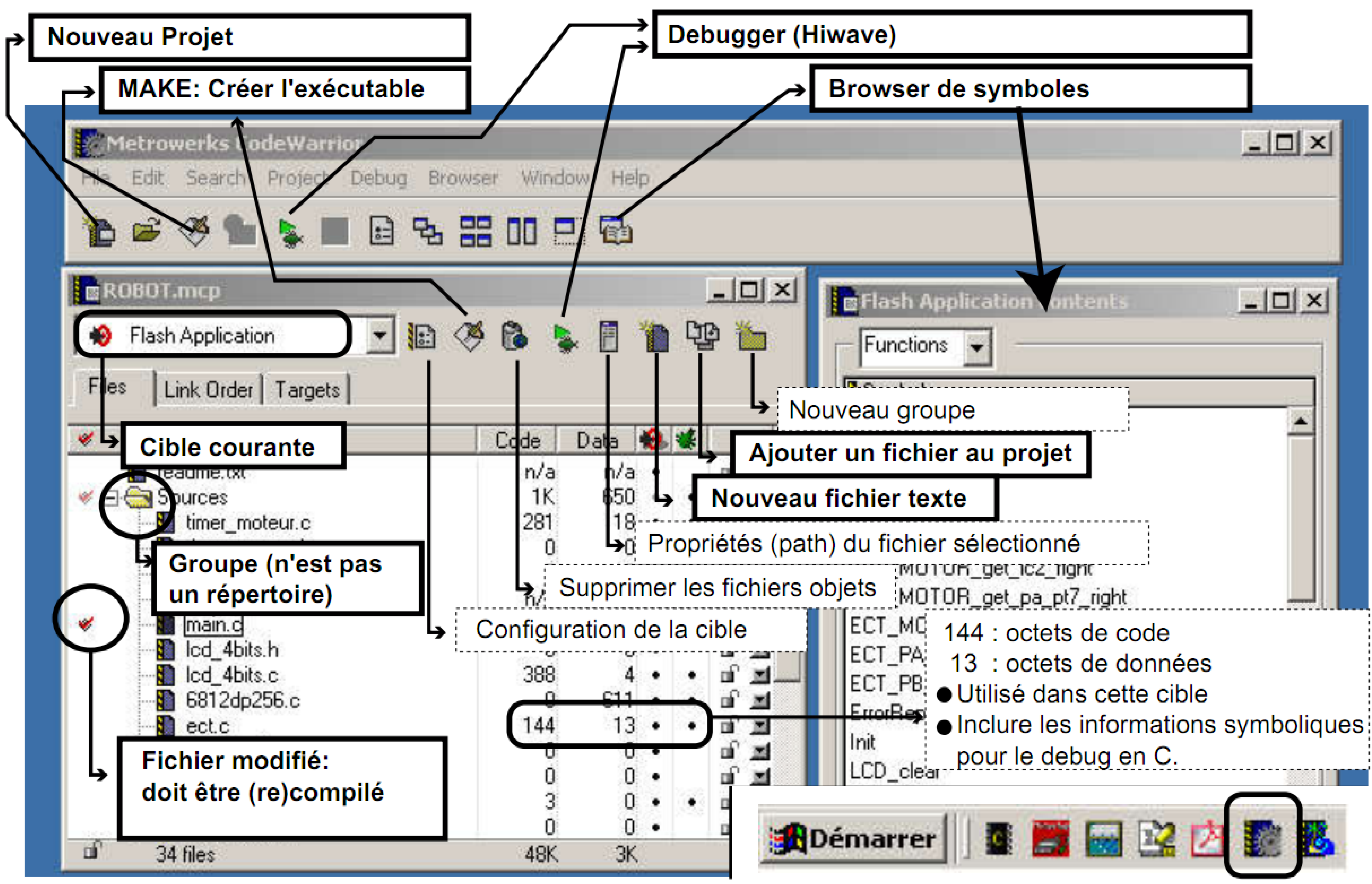
Le linker (éditeur de liens) fabrique l'exécutable à partir des fichiers objets et des fichiers d'entête. Il parcourt les fichiers objets puis les fichiers d'entête pour trouver les symboles référencés. Il calcule les adresses de tous les symboles.

Compilateur

Le compilateur est associé à un langage de programmation et à un processeur cible. Il analyse séparément chaque fichier source et génère pour chacun un fichier objet. Il détecte les erreurs de syntaxe et vérifie la cohérence de l'utilisation des opérateurs en fonction du type de données.

Symbole

Toutes les variables globales et toutes les fonctions (non `static`) n'auront une adresse définitive qu'à la fin de l'édition de liens. Avant cette étape, (notamment lors de la compilation) le compilateur les manipule comme des symboles. On peut donc appeler une fonction ou lire une variable en ne connaissant que son type et son nom.



Nouveau Projet
MAKE: Créer l'exécutable
Debugger (Hiwave)
Browser de symboles
Metrowerks CodeWarrior
File Edit Search Project Debug Browser Window Help
ROBOT.mcp
Flash Application
Files
Link Order
Targets
Flash Application Contents
Functions
Nouveau groupe
Ajouter un fichier au projet
Nouveau fichier texte
Propriétés (path) du fichier sélectionné
Supprimer les fichiers objets
Configuration de la cible
Cible courante
Groupe (n'est pas un répertoire)
Fichier modifié: doit être (re)compilé
Code
Data
timer_moteur.c
main.c
lcd_4bits.h
lcd_4bits.c
6812dp256.c
ect.c
34 files
48K
3K
MOTOR_get_pa_pt7_right
144 : octets de code
13 : octets de données
Utilisé dans cette cible
Inclure les informations symboliques pour le debug en C.
Démarrer

A.3 Errors et Warnings

Could not find or load X for target "Y" for project Z

Un fichier présent dans le projet n'existe pas ou le chemin (access path) ne pointe pas sur lui.

xx.h file not found

Le fichier inclu n'existe pas ou est mal orthographié.

Expected ;

- Vous avec oublié un point virgule en fin d'instruction.
- Le type d'une variable dans une déclaration est mal orthographié.

Implicit parameter declaration

Une fonction n'a pas été déclarée.

MaVariable not declared (or typename)

Une variable ou un type n'a pas été déclaré.

' }' missing

- Vous avez mélangé des déclarations aux instructions.
- Il manque une accolade fermante.
  L'indentation de votre code permet d'éviter ce genre d'erreur.

Wrong number of arguments

Vous avez appelé une fonction en lui passant trop ou trop peu de paramètres.
Vérifiez la déclaration de cette fonction.

Type mismatch (expected XX, given YY)

Vous avez appelé une fonction en lui passant un argument dont le type n'est pas bon.
Vérifiez la déclaration de cette fonction.

1er parametre : Old style declaration
Les suivants : Typedef name expected

Dans la déclaration d'une fonction, le nom du type d'un paramètre utilisé n'existe pas.

Link Error : Symbol XXin file YY.c.o is undefined

Une fonction ou une variable est déclarée mais elle n'existe pas. Elle n'a pas été définie.
Vérifiez la cohérence entre déclaration et définition.

,,,,