LE MICROCONTROLEUR

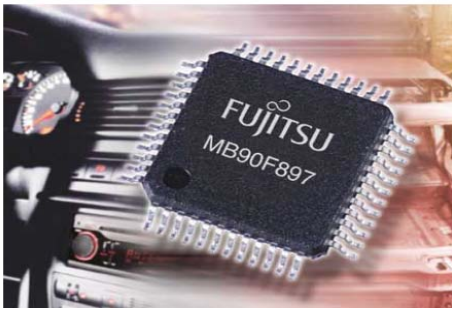
68HC9S12E128

ÉCOLE NATIONALE D'INGÉNIEURS DE TARBES
ENIT
concevons l'avenir

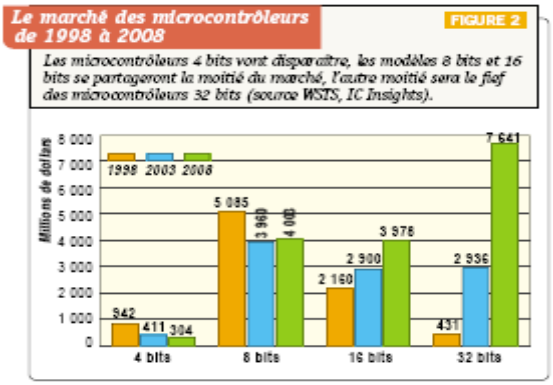
COURS ET EXERCICES

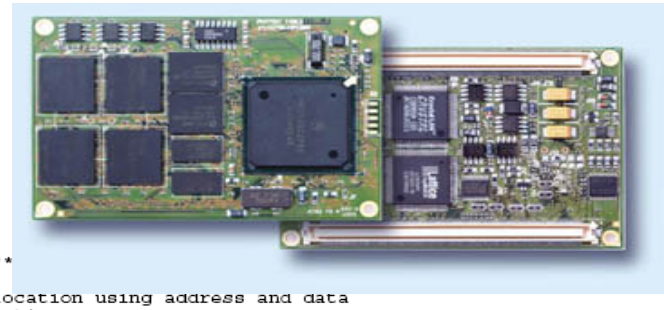
Formation de 4ème année – Semestre S7 et S7*


```
/*****************************************************************************
 *                    Freescale
 *
 *    DESCRIPTION:    S12 single array Flash routines
 *    SOURCE:         flash.c
 *    COPYRIGHT:      © 04/2004  Made in the USA
 *    AUTHOR:         rat579
 *    REV. HISTORY:   (none)
 *
 *****************************************************************************/
#include "projectglobals.h"
#include "flash.h"

extern DoOnStack(unsigned int *far address);

//*************************************************
//* Function Name: Flash_Init
//* Description  : Initialize Flash NVM for HCS12 by pr
//*                FCLKDIV based on passed oscillator f
//*                uprotect the array, and finally ensu
//*                ACCERR are cleared by writing to the
//*
                                                ********
```




```
sure we can program/erase */
```

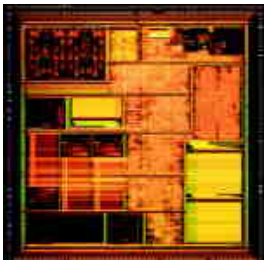




```
//*************************************************
//* Function Name: Flash_Write_Word
//* Description  : Program a given Flash location using address and data
//*                passed from calling function.
//*
//*****************************************************************************
signed char Flash_Write_Word(unsigned int *far far_address,unsigned int data)
```

DOSSIER

Les microcontrôleurs 16 et 32 bits



Un marché en pleine croissance

Le secteur des microcontrôleurs en général se porte très bien. Ainsi la société d'études In-Stat prévoit une croissance moyenne annuelle en volume de 8,74 % entre 2002 et 2007. Cette année-là, 7,4 milliards d'unités sont attendus contre 5,5 milliards en 2003. Dans son rapport d'analyse du marché, In-Stat précise que cette croissance est en partie due à toutes les ressources comme les protocoles de communication, le multimédia, les convertisseurs A/N et N/A et les interfaces de bus standard maintenant intégrées sur les puces. Un constat qui est confirmé par la société IC Insights montrant que le segment à croissance la plus rapide de tout le marché du contrôle embarqué est celui des 32 bits. Représentant aujourd'hui 31 % des 12,28 milliards de dollars du chiffre d'affaires global, ils devraient atteindre les 48 % des 15,926 milliards de dollars prévus en 2008 (figure 2). A cette date, les versions 16 bits vont rester stables avec 25 %, les composants 8 bits continueront leur lente descente avec 25 % et les modèles 4 bits auront pratiquement disparu (2 %).

Le vaste domaine du contrôle peut être découpé en trois catégories d'applications. En premier lieu, le contrôle de processus et de commandes dans l'industrie assuré par des microcontrôleurs 8 bits, de plus en plus par des 16 bits et même par des 32 bits. Pour citer quelques secteurs visés parmi une multitude : les automatismes (automates et robots), les moteurs électriques, les onduleurs et autres alimentations stabilisées, mais aussi les climatiseurs ou encore toute la panoplie de l'électroménager. Aujourd'hui une machine à laver et un aspirateur ne se contentent plus d'un processeur 8 bits ; ils nécessitent un 16 bits.

La deuxième catégorie concerne le contrôle de communications et flot de données, notamment multimédias, confié suivant les contraintes et la puissance requise à des modèles 16 ou 32 bits. A ce sujet, les microcontrôleurs 32 bits se taillent la part du lion dans tous les systèmes de communication sans fil, les modems xDSL et les applications graphiques. Ainsi, la prochaine génération de téléphones portables n'intégrera plus seulement un, mais deux microcontrôleurs 32 bits, à côté de l'inévitable DSP. Le deuxième sera entièrement destiné au traitement des données multimédias.

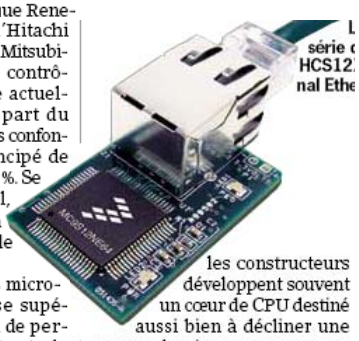
Dans cette catégorie d'applications, les composants 16 bits ont un domaine de prédilection avec les pilotes de disques durs, mais ils assurent également la gestion d'interfaces Ethernet ou Internet. Par exemple, le microcontrôleur MC9S12NE64 de Freescale est un véritable « terminal Ethernet » en une seule puce. Il intègre une pile de communications, des mémoires flash et Ram, un module MAC (media access controller) et un émetteur/récepteur PHY dans un boîtier unique. *« Il rend la connectivité Ethernet accessible à des systèmes bas coût pour lesquels cette possibilité n'était même pas envisageable auparavant »*, souligne Daniel Hoste, directeur général de la division produits 8/16 bits de Freescale.

Enfin, la troisième catégorie a trait au contrôle dans l'automobile qui, lui aussi suivant les exigences en performances, fera appel à un modèle 16 bits ou 32 bits. L'industrie automobile est particulièrement consommatrice de microcontrôleurs. Ils interviennent soit dans le fonctionnement même du véhicule (et sa sécurité), pour la gestion de l'allumage ou de l'injection, des freins ABS ou au niveau du tableau de bord, soit pour le confort dans l'habitacle avec la climatisation, les rétroviseurs électriques, le système de navigation, les modules de commande des portes, l'autoradio voire l'informatique de loisir.

Un dynamisme efficace

Cette explosion des applications de contrôle embarqué sert les microcontrôleurs 16 et 32 bits. Le microcontrôleur n'est plus le parent pauvre du microprocesseur. Signe révélateur : beaucoup de sociétés de renom dans les processeurs ont rejoint les acteurs classiques du domaine. Indiquons également que Renesas, né de la fusion d'Hitachi (modèles 8 et 32 bits) et Mitsubishi (ancien leader des contrôleurs 16 bits), domine actuellement avec 24 % de part du marché, toutes catégories confondues. Freescale (émancipé de Motorola) le suit avec 17 %. Se succèdent ensuite Intel, Nec, Toshiba et Infineon dont les poids varient de 15 à 8 %.

La migration vers des microcontrôleurs de classe supérieure est une question de performances, de coût et de consommation. C'est pourquoi



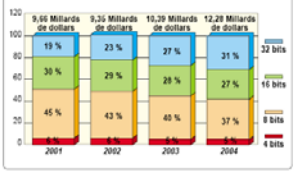
les constructeurs développent souvent un cœur de CPU destiné aussi bien à décliner une gamme de microprocesseurs que de microcontrôleurs ; tout est dans les attributs dont on l'entoure. Citons, entre autres, les unités de traitement PowerPC de Freescale, SuperH de Renesas, FR de Fujitsu ou V850 de Nec. Le cœur le plus prisé aujourd'hui est l'Arm7. Pour preuve, au moins neuf fournisseurs le mettent en œuvre dans le tableau des microcontrôleurs 32 bits qui suit ! La société Arm ne boude pas son succès puisqu'elle introduit une nouvelle version destinée à concurrencer les 8 bits (*Electronique* n°153, p.10).

Si les lettres de noblesse acquises par les microcontrôleurs sont évidentes avec la qualité des cœurs qu'ils embarquent, ce n'est cependant pas le seul exemple. L'adoption de la flash en tant que mémoire programme en est une autre manifestation. Comme nous le verrons dans l'article qui suit, cette petite révolution a grandement modifié et facilité l'utilisation des microcontrôleurs.

HÉLÈNE TRÉZÉGUET

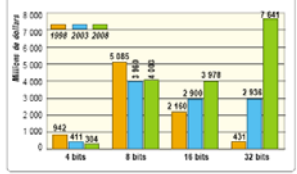
Répartition du marché par type de microcontrôleurs — FIGURE 1

Chaque année, les microcontrôleurs 32 bits gagnent une part plus importante du marché mondial, les autres catégories du secteur diminuent régulièrement leur chiffre d'affaires (source IC Insights).



Le marché des microcontrôleurs de 1998 à 2008 — FIGURE 2

Les microcontrôleurs 4 bits vont disparaître, les modèles 8 bits et 16 bits se partageront la moitié du marché, l'autre moitié sera le fief des microcontrôleurs 32 bits (source WSTS, IC Insights).



Pré requis

L’étude des circuits électriques ***S2-électricité***
Les bases de la Programmation en C
Le cours d’électronique ***S4-Les composants électroniques***
La logique combinatoire
Le codage en décimal, binaire (0b), hexadécimal (0x ou ....h)
L'algorithmique

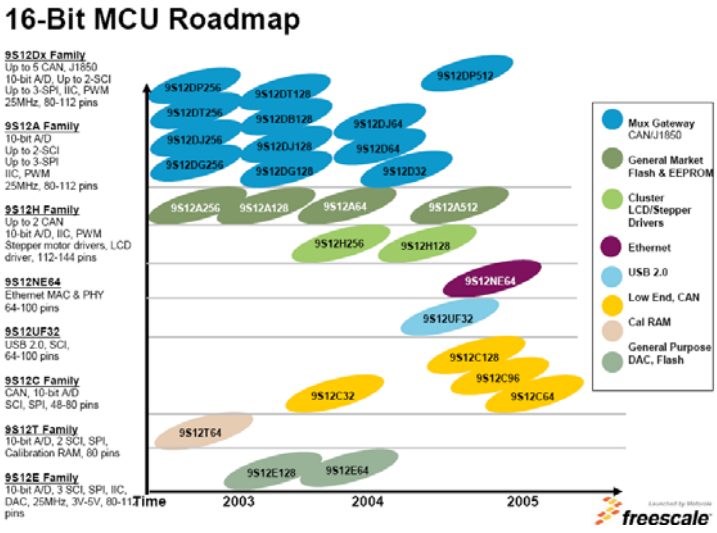
SOMMAIRE

Introduction

Comme le montre le dossier précédent, le microprocesseur et plus particulièrement le microcontrôleur est un outil indispensable pour la conception d'un produit industriel. On le retrouvera comme le pilote de tous les systèmes de commande et de puissance (électriques, mécaniques, physiques…).

Notre cour s'articule autour du tout récent microcontrôleur 16 bits de Motorola, le 68HC12, qui remplace très avantageusement le 68HC11, 8 bits en fin de cycle de vie. Le choix du type de microprocesseur est difficile, car de très nombreux constructeurs proposent des dispositifs similaires. Toutefois, ce produit possède un environnement de programmation très élaboré et « relativement souple » pour l'utilisateur.

Le 68HC12 se décline en un nombre de versions très important, qui diffèrent par la nature ou la taille des constituants qui entoure le cœur du microcontrôleur. Le MC9S12e128 nous est imposé par le vendeur de notre environnement de travail. Ce dernier est très performant, possède une taille de mémoire programme importante et les interfaces les plus courants (TIMER, CAD, PWM, I2C, SPI, SCI…).



L'objet de ce cours est de vous faire comprendre à quoi sert un microcontrôleur, comment il fonctionne et de vous familiariser avec sa programmation interne. Nous espérons que vous comprendrez qu'il s'agit d'un composant électronique qui utilise les techniques numériques, qui sert principalement à commander des organes extérieurs par ses sorties ou à acquérir des informations sur ses entrées. L'outil informatique et les auxiliaires de connexion ne sont là que pour aider l'utilisateur à configurer et à rentrer le programme dans le composant.

Le cours se veut de difficultés progressives, mais ne peut se permettre d'être répétitif d'une séance à l'autre. L'étudiant devra donc en début de chaque séance avoir parfaitement acquis le contenu de la séance précédente. L'école met à la disposition de chaque étudiant un outil performant et coûteux pour que votre apprentissage soit efficace, et les séances sont construites pour que vous puissiez bénéficier d'un temps largement suffisant pour expérimenter tous les points présentés en cours.

Chapitre 1

Présentation du MC9S12e128

Ce chapitre présente l'architecture interne du composant pour identifier les différents organes qui seront étudiés dans les chapitres suivants.
De même, une présentation de l'environnement matériel de programmation est indispensable pour la suite.
Le contenu du chapitre fera l'objet d'une première séance de TP, qui conduira l'élève à manipuler autour du µP pour comprendre le fonctionnement et corriger un premier programme.

Objectifs :

- Connaître les éléments essentiels du cœur du microcontrôleur
- Connaître la structure d'échange des données internes et externes
- Savoir ce que sont les ports E/S, leur dénomination, les registres
- Connaître la manière de configurer une entrée ou une sortie à l'aide des registres de direction
- Savoir expliquer la différence entre RAM, ROM, EEPROM, Flash
- Savoir ce qu'est le plan mémoire
- Savoir manipuler l'environnement informatique du µP pour pouvoir ouvrir un programme, le modifier à l'aide de l'éditer, le compiler pour le faire fonctionner

Vocabulaire :

Unité Centrale	Microprocesseur
Mémoire vive RAM	Microcontrôleur
Mémoire de programme ROM	

1.1/ A quoi sert un microcontrôleur ?

1.1.1/ Qu'est ce que c'est ?

Un microcontrôleur est un composant :

- électronique, à base de semi-conducteurs
- qui utilise le codage binaire des informations
- programmable qui exécute séquentiellement des opérations élémentaires

Les opérations élémentaires qu'il pourra traiter sont :

- les opérations arithmétiques courantes : addition, soustractions, divisions, multiplications
- des opérations logiques : et, ou, ou exclusif, inversion…
- des opérations de contrôle de bits : forçage à «1» ou «0» d'un bit d'un octet
- des opérations de test : =, >, <, >=, <=…
- la génération de signaux carrés,
- des mesures d'informations logiques sur ses entrées,
- des mesures d'informations analogiques issues de capteurs.

1.1.2/ A quoi ça sert ?

A contrôler des automatismes très divers :

- machine à laver
- photocopieuses
- guichet bancaire automatique
- …

A contrôler des processus industriels :

- acquisition d'informations issues de capteurs,
- élaboration de lois de commande,
- commande d'actionneur
- conversion de signaux

A surveiller le fonctionnement de dispositifs complexes :

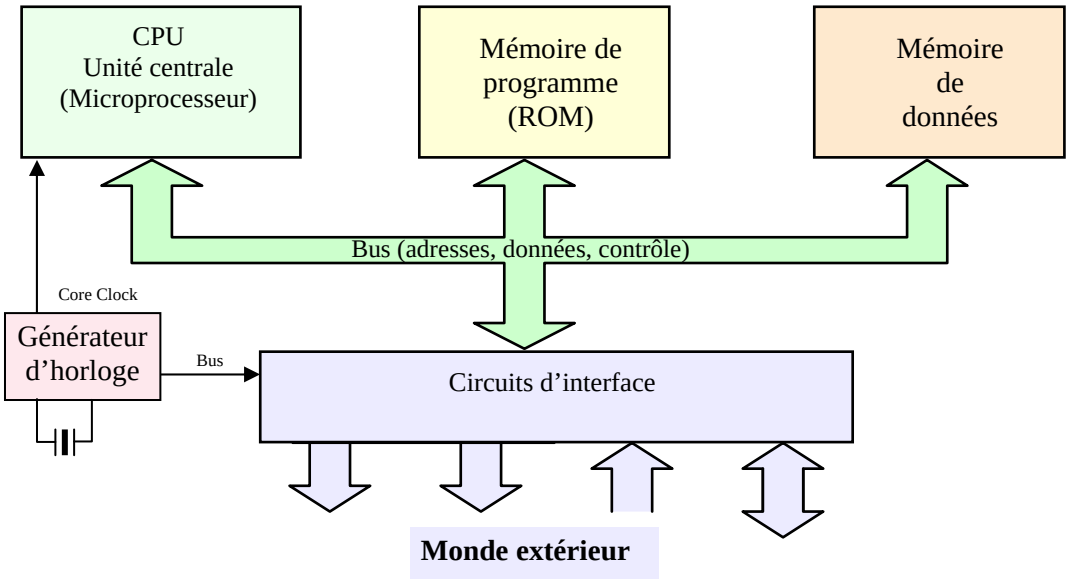
- supervision dans les appareils,
- alarme,
- détection
- …

1.2/ Structure interne du 68HC9s12e128

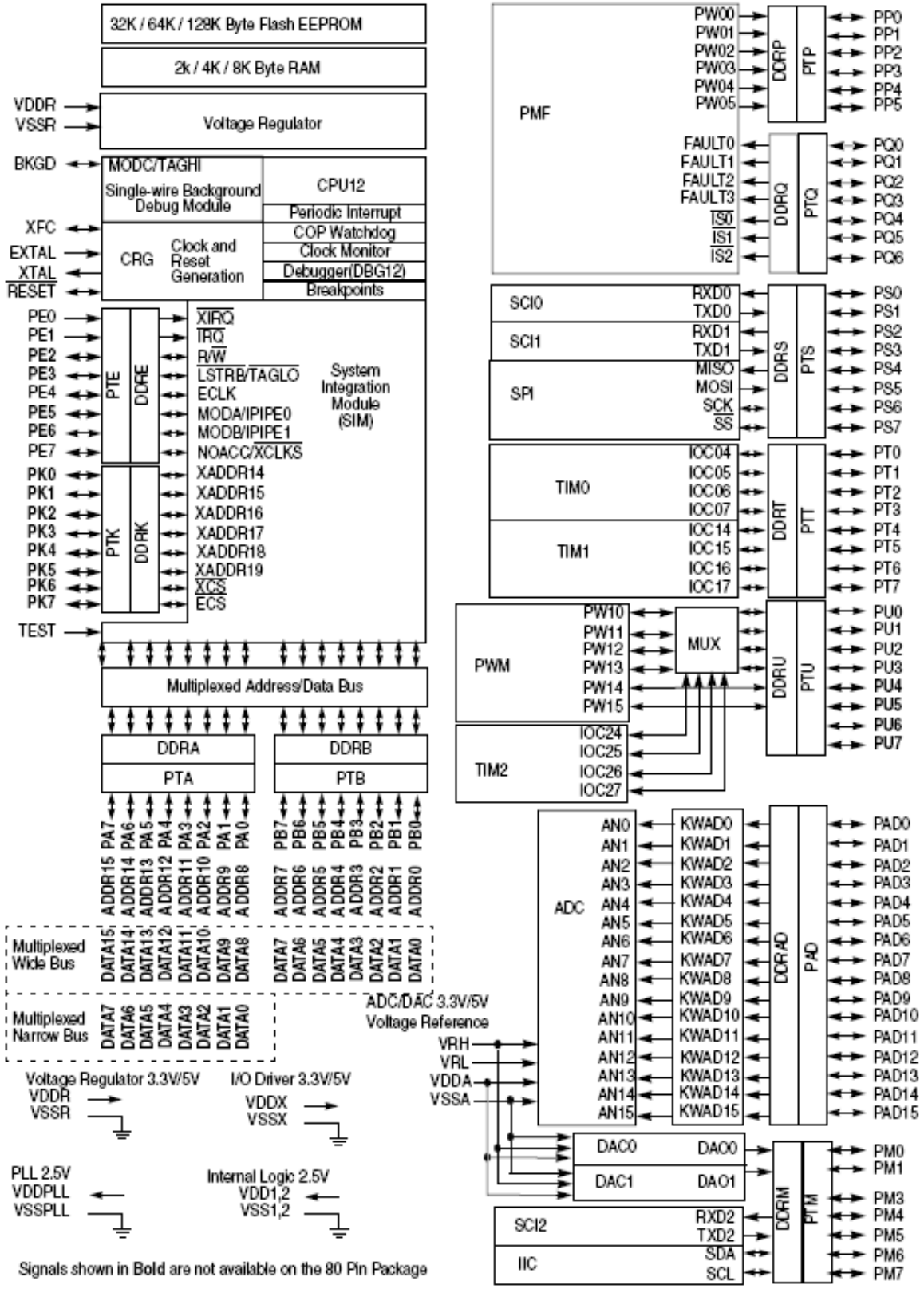
1.2.1/ Structure type d'un microcontrôleur

L'architecture interne du microcontrôleur est celle d'un microprocesseur avec ses interfaces. Le microprocesseur est le moteur (unité centrale) du microcontrôleur qui comporte en plus, tous les interfaces, et ceci dans un même boîtier. Il existe plusieurs versions d'un même microcontrôleur, car les utilisateurs n'ont pas tous besoin des mêmes interfaces. Par exemple, certains vont souhaiter avoir plus de convertisseurs analogiques digitaux alors que d'autres souhaitent étendre l'espace mémoire interne…
La figure suivante nous présente les éléments incontournables inclus dans tout microcontrôleur :

- une unité centrale, qui contrôle le fonctionnement et qui calcul,
- une mémoire, dite morte, du type ROM (read only memory) pour conserver le programme, ensemble des instructions qui seront exécutées par l'unité centrale, même si le microprocesseur perd son alimentation électrique,
- une mémoire de données intermédiaires, variables sous la forme d'une RAM (random access memory), dont le contenu est effacé si le microprocesseur perd son alimentation électrique,
- des « ports » pour communiquer avec l'extérieur, qui regroupés sous la forme d'entrées ou de sortie ou les deux (E/S),
- les informations sont échangées entre ces différents modules par l'intermédiaire de « bus », bus de données (16bits), bus d'adresses (16bits), bus de contrôle
- un générateur d'horloge qui rythme le déroulement des opérations élémentaires, qui sert aussi dans l'échange des données sur le bus



1.2.2/ Structure du 68HC9S12e128



Dénomination 68HC12…

Les références du 68HC12 sont assez compliquées. En fait, le terme 68HC12 (ou M68HC12 ou HC12) désigne une famille. Dans la réalité, les 68HC12 qui ont une FLASH intégrée deviennenr des 68HC912. MOTOROLA a, depuis peu, supprimé le "68" et la référence devient alors HC912. Il existe les déclinaisons suivantes de cette famille : HC912B32, HC912BC32, HC912D60A, HC912DG128A. Dans la suite de ce document, nous continuerons parfois à utiliser l'ancienne référence 68HC12 ou HC12.

Puissance du HC12…

Le 68HC12 dispose d'un jeu d'instructions mathématiques (16x16, 32/16, EMACS, FDIV...) qui simplifie les calculs. Le 68HC12 dispose également de nouvelles instructions sur la logique floue (fuzzy logic). Il s'agit d'une nouvelle approche dont le principe est expliqué brièvement ci-après. Parfois, les variables ne sont pas 0 ou 1 mais sont comprises entre 0 et 1. Une variable peut devenir de moins en moins vraie tandis qu'une autre adjacente de plus en plus vraie. Ces variables suivent certaines règles définies par le programmeur. Cette méthode est empruntée aux systèmes experts sans pour autant en garder la complexité. Par exemple, dans un sèche-linge contrôlé par un microcontrôleur, le linge n'est jamais complètement humide (niveau 0) ou sec à 100% (niveau 1) mais plus ou moins sec. Le problème devient complexe lorsque plusieurs règles analogiques interviennent, par exemple la température du linge. Dans des cas comme celui-ci, la machine s'arrête à de l'intersection de deux droites (intersection de l'hypoténuse de deux triangles rectangles mis en opposition, l'un pour l'humidité, l'autre pour la température). Le 68HC12 permet de programmer ces conditions sous la forme de "règles d'inférence" qui suivent les algorithmes sur la logique floue.

Le 68HC12 offre quatre canaux PWM (pulse width modulator) d'une grande utilité pour la commande de moteurs. Ces sorties, associées à un CI du genre L293 ou L297, permettent de réaliser rapidement des variateurs de vitesse et asservissements divers. Le 68HC12 réalise automatique toute cette gestion de rapport cyclique qui, avec les microcontrôleurs du genre 68HC11, s'effectuait par timer sous interruptions de programmes

La grande nouveauté du 68HC12 est incontestablement le bus CAN V2.0 qui, en quelques années, est devenu le standard du réseau industriel. Ce bus, mis au point initialement par BOSCH pour les automobiles, s'est très vite généralisé, notamment en raison de sa fiabilité. Il en résulte que le bus CAN se trouve aujourd'hui dans de nombreux domaines autres que celui de l'automobile. Ce bus bifilaire, de conception moderne, extrêmement fiable et dont la vitesse est de 1 Mbps, remplace les autres bus bifilaires lorsqu'une mise en réseau à la fois performante et fiable est requise. Aujourd'hui, il existe une telle demande au niveau de l'industrie pour le bus CAN que cette technique est devenue incontournable dans l'enseignement technique.

Le 68HC12 est particulièrement bien conçu pour la CEM. En milieu scolaire ce point n'est pas important, mais dans l'industrie, il est tout à fait capital. En effet, une société ne peut exporter du matériel s'il n'est pas conforme aux normes CE dont la CEM. En milieu industriel, il est donc capital de porter une attention

particulière à la CEM, d'autant qu'une carte conçue dans cette optique est beaucoup plus fiable qu'une carte traditionnelle, car les signaux sont plus propres.

Du 68HC 11 vers HC12 (pour les spécialistes)

La principale différence entre le 68HC11 et le 68HC12 est la présence d'une mémoire FLASH dans ce dernier microcontrôleur. Cette mémoire est importante et permet de programmer directement en langage C ou C++.
Le 68HC12 accepte les syntaxes du 68HC11. Le passage du HC11 au HC12 s'effectuera généralement sans grandes difficultés, car il suffit de réassembler les anciens programmes 68HC11.
Tous les microcontrôleurs se disent supporter les langages de haut niveau (HLL) mais il s'agit souvent plus d'un argument commercial qu'une réalité. Le support d'HLL nécessite la réentrance et une gestion correcte de la pile. Le 68HC12, a été conçu dans cette optique, ce qui n'est pas le cas des autres microcontrôleurs bas de gamme comme le 68HC11. En effet, lors de l'appel d'une fonction HLL, les paramètres sont stockés dans la pile S. La fonction appelée doit pouvoir les récupérer facilement. Pour cela, il lui faut des modes d'adressage adéquats, et en particulier l'indexé par S avec offset, avec pré/post incrémentation ou décrémentation (ex.: nn,S++). De même, la création d'espaces dynamiques pour les variables nécessite la fameuse LEAS xx,S. Même problème pour la réentrance. Avec les microcontrôleurs non adaptés aux HLL, le compilateur "se débrouille" mais au prix d'une "source-assembleur" extrêmement lourde puisqu'il faut 5 à 10 instructions là où le 68HC12, 6809 ou 68000 n'en nécessite qu'une seule.
Enfin, le 68HC12 emprunte au 68HC11 de nombreux modules (multiples timers, CAD...) avec des performances supérieures. Par exemple, les convertisseurs A/D sont sur 10 bits au lieu de 8, et les timers nettement plus sophistiqués.

1.2.3/ Les circuits d'interface (E/S)

Les circuits d'interfaces avec le monde extérieur du MC9s12e128, sont articulés autour de « ports d'interface » :

- les ports « parallèle », regroupés en 6, 7, 8, 16 fils qui permettent d'échanger des données binaires, des octets. Certaines imprimantes communiquent encore sous forme parallèle (liaison Centronics) avec le PC,
- les ports « série », qui échangent les données sous la forme d'une trame. La liaison RS232 (COM1, 2) du PC est une liaison série, de même que la liaison USB.

Des Convertisseurs internes du type Analogique Digital (CAN ou ADC) sont reliés à des pattes du portAD pour permettre la conversion des signaux extérieurs.
Des TIMERS complètent l'ensemble et nous offrirons la possibilité de générer ou mesurer des temps à une fraction de µs prés.
L'horloge du système (osc) définit le temps d'exécution des instructions du microprocesseur (core clock), la résolution des timers et des fonctions spécialisées (bus clock).
La fréquence de la maquette est réglée à 16MHz, et donc core clock= 16MHz, bus clock=osc/2 soit 8MHz.

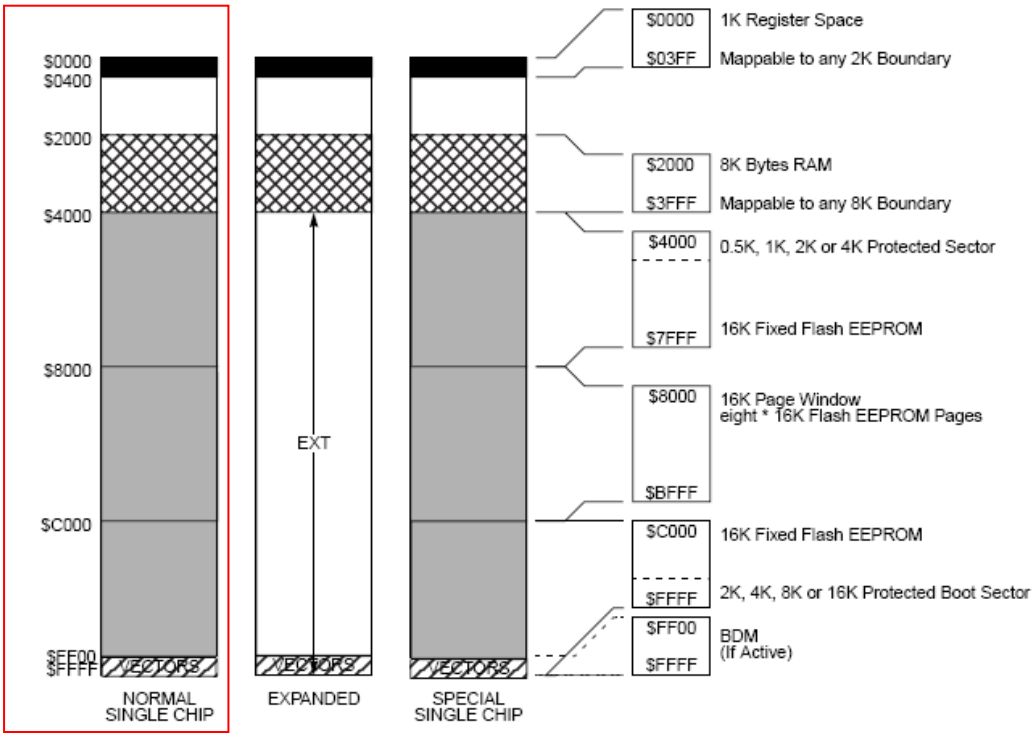
1.2.4/ Organisation mémoire ou espace adressable

L'adressage est, comme tous les autres microcontrôleurs de la gamme, sur 64 kots. Cependant, il est prévu une pagination qui permet de dépasser cette barrière de 64 kots.

Le processeur interne du 68HC12 est en 16 bits réels et les octets pairs/impairs des mémoires sont alignés automatiquement.

Au niveau extérieur, le 68HC12 peut se mettre en mode étendu ou en mode single. Il existe 3 façons d'organiser la mémoire (le plan mémoire), normal ou single chip (monochip), étendu 8 bits (expanded) ou étendu 16 bits (mode spécial).

Nous utiliserons exclusivement le mode normal, car l'espace sera largement suffisant pour nos programmes.



Address	Module	Size
0x0000–0x0017	CORE (Ports A, B, E, Modes, Inits, Test)	24
0x0018	Reserved	1
0x0019	Voltage Regulator (VREG)	1
0x001A–0x001B	Device ID register (PARTID)	2
0x001C–0x001F	CORE (MEMSIZ, IRQ, HPRIO)	4
0x0020–0x002F	CORE (DBG)	16
0x0030–0x0033	CORE (PPAGE, Port K)	4
0x0034–0x003F	Clock and Reset Generator (PLL, RTI, COP)	12
0x0040–0x006F	Standard Timer 16-bit 4 channels (TIM0)	48
0x0070–0x007F	Reserved	16
0x0080–0x00AF	Analog to Digital Converter 10-bit 16 channels (ATD)	48
0x00B0–0x00C7	Reserved	24
0x00C8–0x00CF	Serial Communications Interface 0 (SCI0)	8
0x00D0–0x00D7	Serial Communications Interface 1 (SCI1)	8
0x00D8–0x00DF	Serial Peripheral Interface (SPI)	8
0x00E0–0x00E7	Inter IC Bus	8
0x00E8–0x00EF	Serial Communications Interface 2 (SCI2)	8
0x00F0–0x00F3	Digital to Analog Converter 8-bit 1-channel (DAC0)	4
0x00F4–0x00F7	Digital to Analog Converter 8-bit 1-channel (DAC1)	4
0x00F8–0x00FF	Reserved	8
0x0100- 0x010F	Flash Control Register	16
0x0110–0x013F	Reserved	48
0x0140–0x016F	Standard Timer 16-bit 4 channels (TIM1)	48
0x0170–0x017F	Reserved	16
0x0180–0x01AF	Standard Timer 16-bit 4 channels (TIM2)	48
0x01B0–0x01DF	Reserved	48
0x01E0–0x01FF	Pulse Width Modulator 8-bit 6 channels (PWM)	32
0x0200–0x023F	Pulse Width Modulator with Fault 15-bit 6 channels (PMF)	64
0x0240–0x027F	Port Integration Module (PIM)	64
0x0280–0x03FF	Reserved	384

Chapitre 2

Présentation de l'environnement de travail

Le microcontrôleur est un composant qui permet de gérer un nombre de dispositifs pratiquement illimité; on peut le rencontrer dans tous les domaines aussi bien professionnels que grands publics. Le revers de la médaille est son extrême complexité, qui réclame un outil d'aide à sa programmation ou "outil de développement". Cet outil utilise des moyens, informatiques de base (utilisation d'un PC), et spécialisés sous la forme d'une carte électronique conçue par la société SOFTEC, pour le monde industriel, qui s'insère entre la maquette (port BDM) et le PC par un port USB.

Objectifs :

- Connaître les éléments matériels essentiels du système de développement.
- Savoir démarrer le logiciel CodeWarrior et reconnaître les dossiers essentiels où se trouvent les fichiers sources, compilés.
- Connaître les différences entre un fichier source et fichier compilé
- Savoir lancer, l'éditeur des fichiers sources, la compilation des fichiers sources, la commande make, et le debugger.
- Savoir placer un break point, lancer un programme en mode pas à pas, visualiser les registres et les variables.

Vocabulaire :

Programme
Langage machine – langage assembleur – langage évolué – langage C
Compilateur – Compilation
Edition des liens
Debuggeur - Debugage
Maquette de développement

2.1/ Outil logiciel

L'outil de développement doit principalement aider le concepteur à rentrer un programme dans la mémoire morte du microcontrôleur.
Le programme qui est entré dans la mémoire du composant est formé d'une suite de 0 et de 1, *le langage machine*, qui est incompréhensible. L'opération qui transforme un programme en langage compréhensible par le concepteur en cette suite de 0 et de 1, s'appelle une opération de « compilation » qui est entièrement automatisée.
Le programme compréhensible est écrit par l'utilisateur par le biais d'un éditeur de texte, sous la forme d'une suite d'instructions. Le langage utilisé pour la réalisation du « code » (programme) produira une suite d'instructions plus ou moins « évoluées » (riches), anciennement le « langage assembleur » représentait la référence en la matière, depuis quelques années le « langage C » est devenu la nouvelle règle. Toutefois, le concepteur n'emprunte à ce langage au plus quelques dizaines d'instructions qui suffisent très largement pour réaliser le code.
Une fois le code écrit, puis rentré dans la mémoire du microcontrôleur, il ne reste plus qu'à le tester. Malheureusement, dés que le programme dépasse une dizaine d'instructions, la probabilité d'un *fonctionnement au premier coût* est quasi nulle, et le plus souvent le concepteur ne voit rien évoluer autour du microcontrôleur pour le renseigner sur le motif du refus de fonctionnement. L'opération de dépannage demande alors un aller-retour entre phases d'écriture, phases de tests, phase de compréhension. Ces phases de mise au point s'appellent l'opération de « debuggage ».
Des entreprises ont développé des outils plus ou moins performants, pour aider le concepteur dans les opérations d'écriture du programme, de compilation, de debuggage. Un outil sera d'autant plus performant qu'il évitera à une trop longue phase de développement.

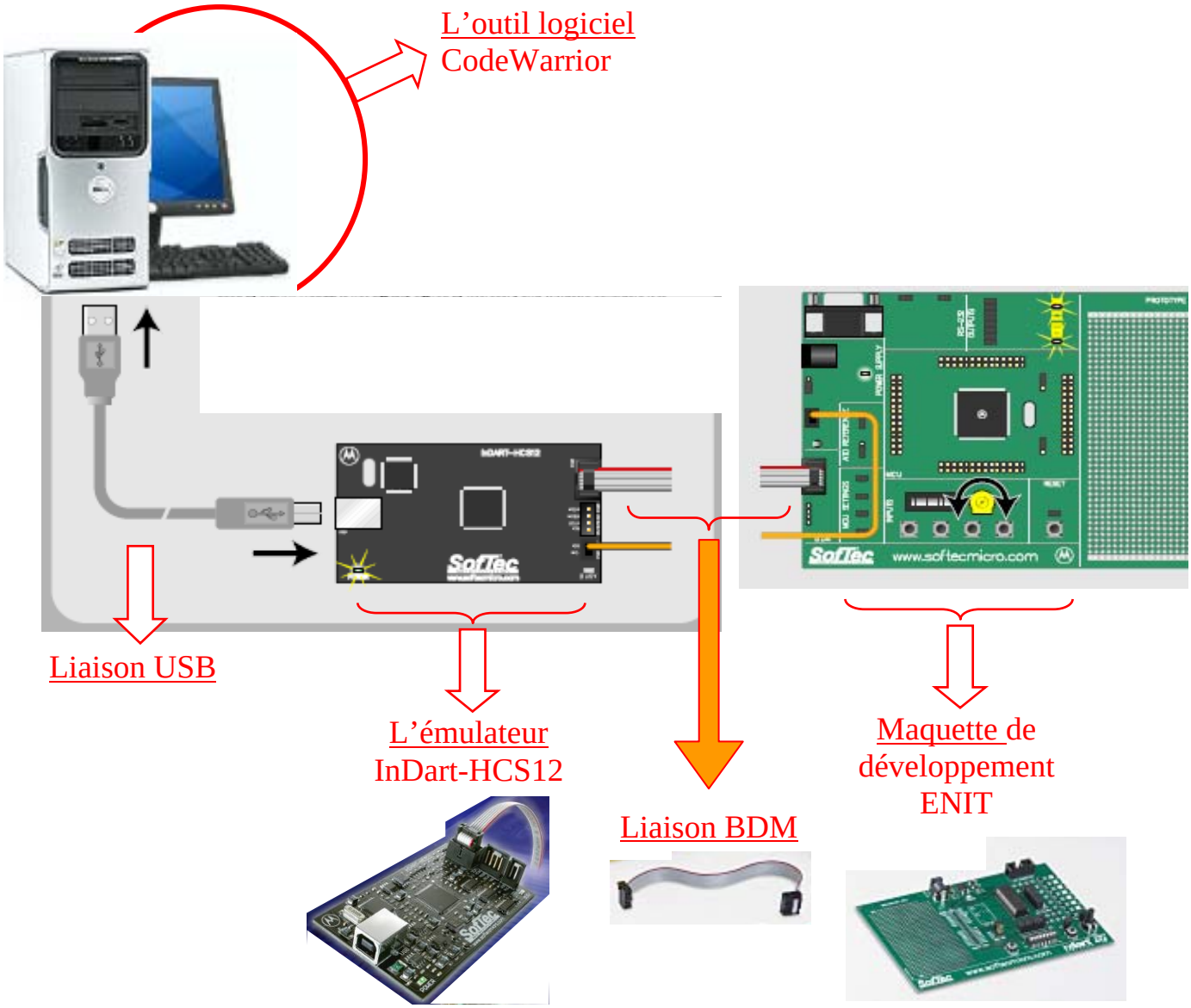
En résumé un bon outil de développement comprend :

- Un éditeur pour écrire le code en langage C.
- Un compilateur de langage C pour convertir le langage C en langage machine.
- Un éditeur de liens, qui génère le code exécutable.
- Un outil pour transporter le code dans la mémoire du µC.
- Un Débuggeur en C pour facilité la phase de mise au point.

L'outil logiciel sur lequel s'appuie cet enseignement est le logiciel CodeWarrior V3.1 de Metrowerk qui offre, l'éditeur, le compilateur, le debuggeur. Cet outil est couplé au logiciel « inDart-HCS12 V1.05 » de la société Softec qui prend en charge le transport du code vers la mémoire via un outil matériel.

2.2/ Outil matériel

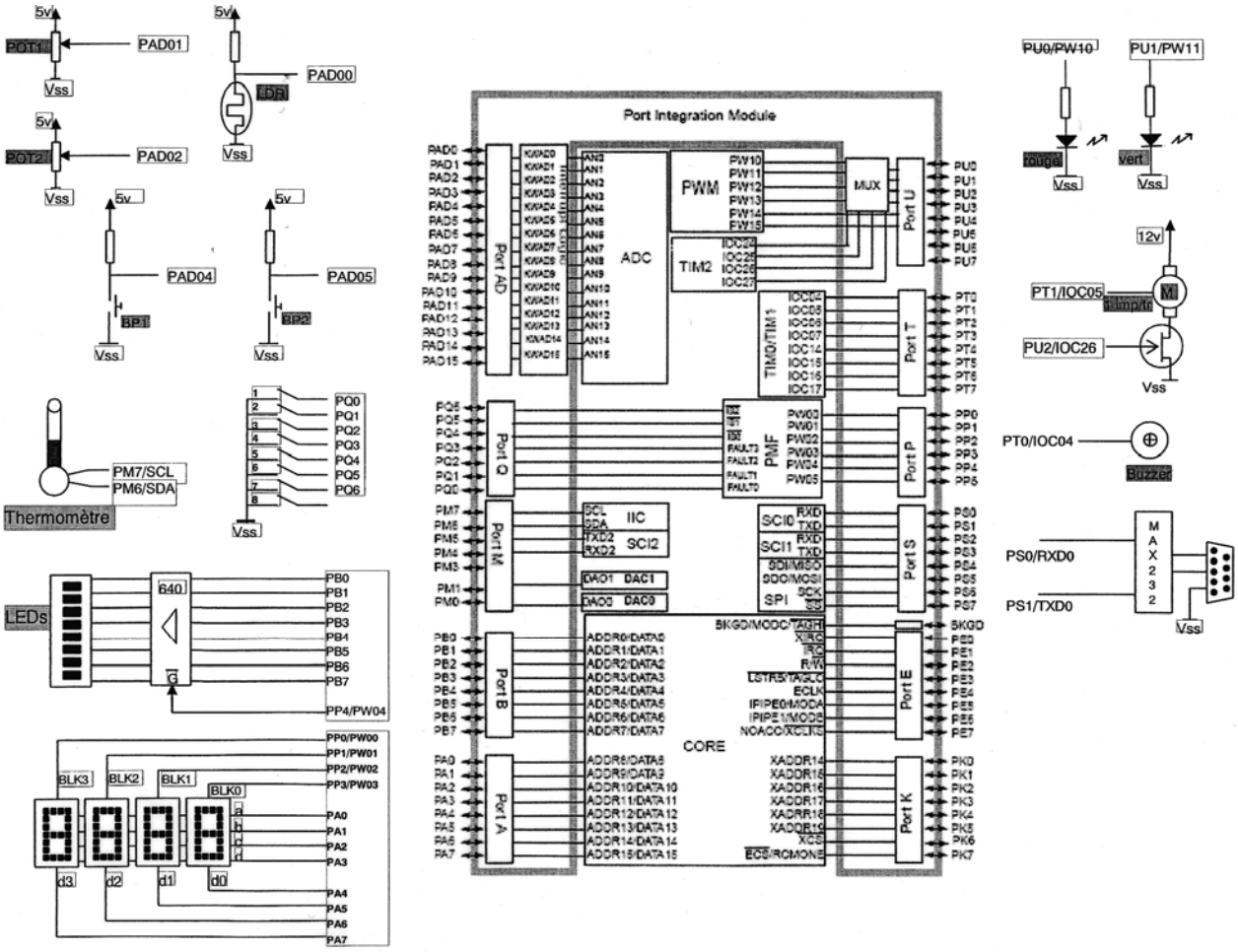
CodeWarrior est installé sur un PC classique. L'écriture du code, l'édition de liens et l'opération de compilation se font dans le PC.
Pour rentrer la programme dans la mémoire du microcontrôleur, CodeWarrior fait appel à inDart-HCS12, qui transporte le langage machine du bus USB de sortie du PC vers le port BDM d'entrée du 68HC12. Une carte électronique « l'émulateur » est insérée entre ces deux ports pour, d'une part réaliser la compatibilité de l'échange, et d'autre part offrir aussi la possibilité de réaliser l'opération de débogage dans les meilleures conditions.



2.3/ Maquette de développement ENIT

La maquette de développement a été conçue à l'ENIT par les professeurs, pour offrir toute l'attitude dans les exemples de projets réalisés en cours.
Cette maquette comporte :

- Des boutons-poussoirs pour tester les opérations de détection d'entrée
- Des Led's pour visualiser des sorties logiques et pour tester les fonctionnalités des timers
- Des potentiomètres, pour tester la conversion analogique digitale
- Une LDR, pour mesurer sous forme analogique l'éclairement
- Un moteur à courant continu, pour utiliser timer ou générateur PWM
- Un affichage 7 segments pour visualiser des nombres à 4 chiffres
- Une barre de Led's pour visualiser la sortie d'un port
- Un buzzer, pour générer des sons et utiliser les timers
- Un capteur de température pour utiliser le port I2C
- Une liaison RS232 pour tester les SCI



2.4/ Principes de développement

2.4.1/ Projet

Un projet comprend des phases incontournables et la réalisation de la phase suivante nécessite la fin de la réalisation de la phase précédente, généralement validée par une procédure de validation.
Les phases incontournables sont :

- La définition du besoin
- Le cahier des charges
- La définition du produit
- La mise au point
- Les reprises de définition

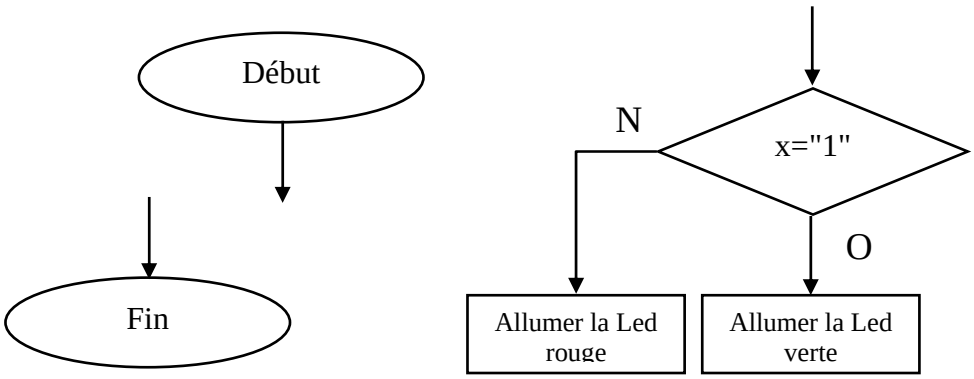
Qui peuvent aussi se traduire, à notre échelle, par :

- Que veut-on faire
- Comment le traduire
- On réalise les algorithmes, on écrit les codes
- On teste grâce à l'émulateur
- On modifie algorithme si nécessaire et code

Les phases qui nous intéressent dans cet apprentissage sont la réalisation des algorithmes et l'écriture du code associé.

2.4.2/ Algorithme

Un algorithme est un arbre, ou un ensemble d'arbres, construit pour traduire ce que l'on veut faire à travers la réalisation d'un ensemble de tâches élémentaires séquencées. On lit cet algorithme de haut en bas, il comporte toujours un début et généralement une fin. Les tâches élémentaires que nous utiliserons seront soient des actions à réaliser, par exemple allumer la Led verte, soient des branchements conditionnels, par exemple est-ce que x="1" si "oui" faire … et si "non" faire …



Un algorithme traduit un cahier des charges. Chaque étudiant peut trouver sa méthode pour réaliser le cahier des charges, et donc chaque étudiant écrira son propre algorithme.
D'autre part, l'algorithme doit pouvoir être transcrit en code quel que soit le type de microcontrôleur : 68HC11, 6809, 68000, 68HC12…

L'algorithme ne doit pas comporter de référence propre au matériel utilisé, par exemple il serait inconvenant d'écrire, forcer le bit n°4 du port B à 1 pour allumer la Led verte.

2.4.3/ Code

L'écriture du code se fait en langage C. Les instructions les plus couramment utilisées pour les projets à base de microcontrôleur sont les opérations de logique binaire, le forçage de bits, les quatre opérations, les tests conditionnels, les boucles, les définitions de variables. Dans les annexes sont regroupés les principaux codes qui seront utilisés dans ce cours.

Une « feuille programme » utilise une structure qui est toujours la même.

Partie secondaire écrite une fois puis recopiée de programmes en programmes.

- Appels de routines associées au fonctionnement du microcontrôleur 68HC12 (voir § 2.5)

Partie accessoire

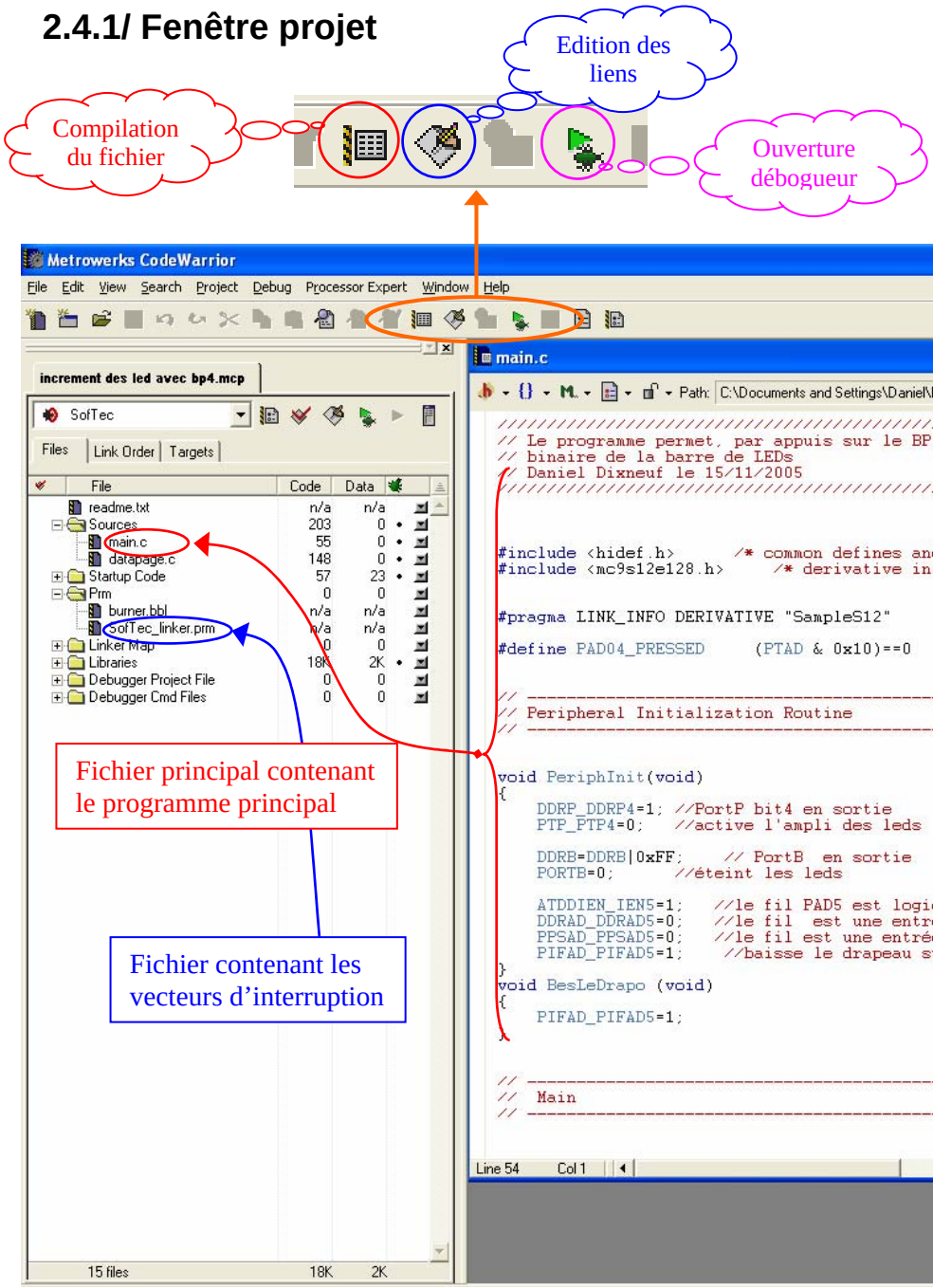
- Appels de routines déjà écrites par l'utilisateur

Partie principale écrite par le concepteur, qui constitue le cœur du programme et la retranscription des algorithmes.

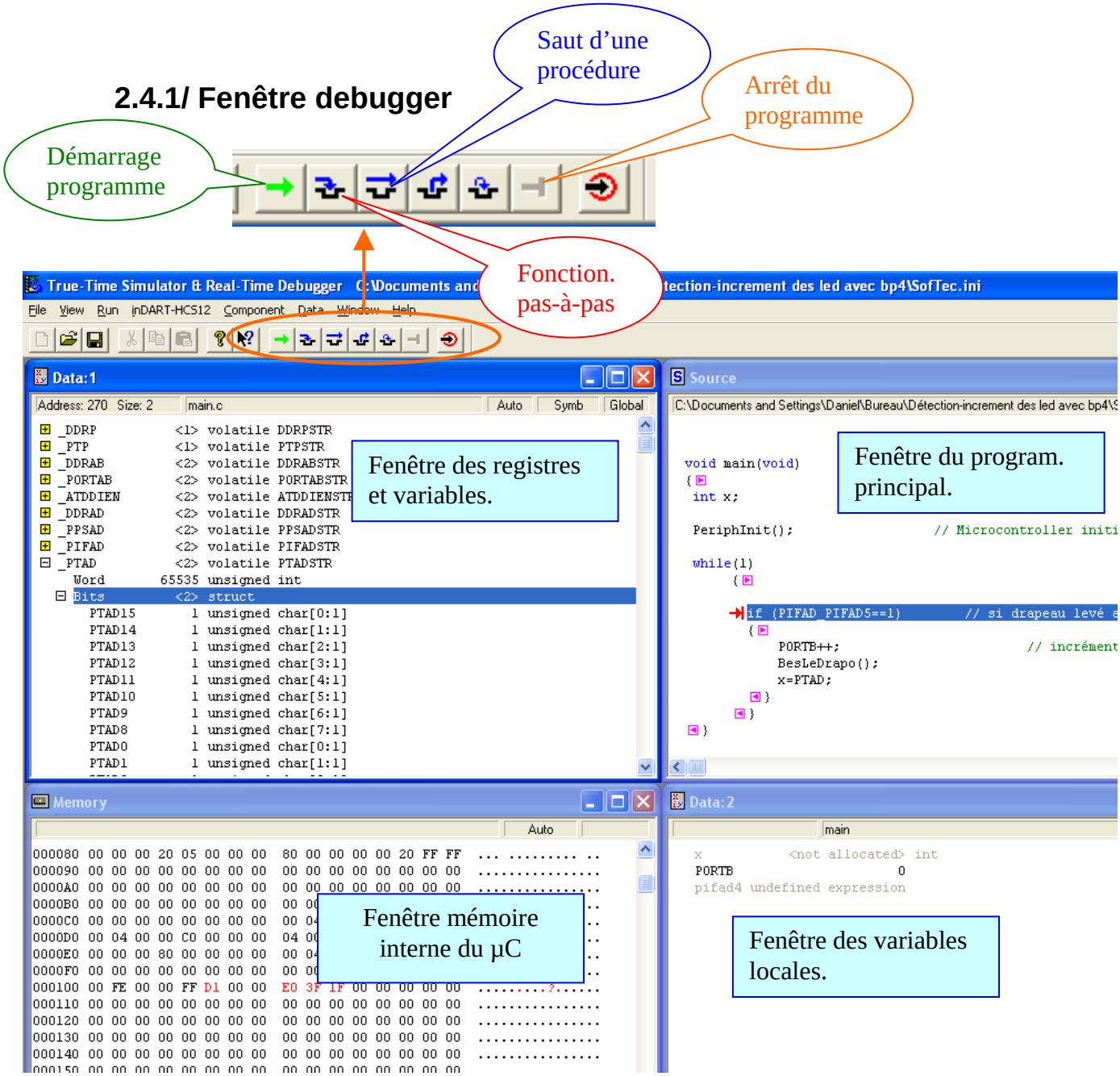
- Déclarations de variables globales
- Les fonctions et procédures, utilisées dans le programme principal
- Le programme principal
- La fin du programme

2.4/ Principes d'utilisation de CodeWarrior

2.4.1/ Fenêtre projet



2.4.1/ Fenêtre debugger



2.5/ Routines associées au 68HC11

Ces deux directives doivent être écrites comme en-tête du programme.

```
#include <hidef.h>              // Common defines and macros
#include <mc9s12e128.h>         // Derivative information
```

Chapitre 3

Les ports et les registres des entrées - sorties

Les entrées sorties, E/S, sont les fils physiques sur le boîtier du microcontrôleur qui permettent la communication avec l'extérieur. Une communication qui s'effectue sur des fils d'entrées pour que la puce puisse acquérir de l'information qu'elle devra traiter, ou sur des fils de sortie pour qu'une fois l'information traiter, soient délivrés les résultats du traitement.

Objectifs :

- Connaître l'organisation des fils sous forme de ports.
- Connaître les différents types d'entrées et de sorties, logiques ou analogiques.
- Connaître les principes de configuration d'un fil E/S par le biais des registres de configuration.
- Connaître le principe de fonctionnement d'une détection sur un fil d'entrée, par la technique de scrutation ou du levé de drapeau.

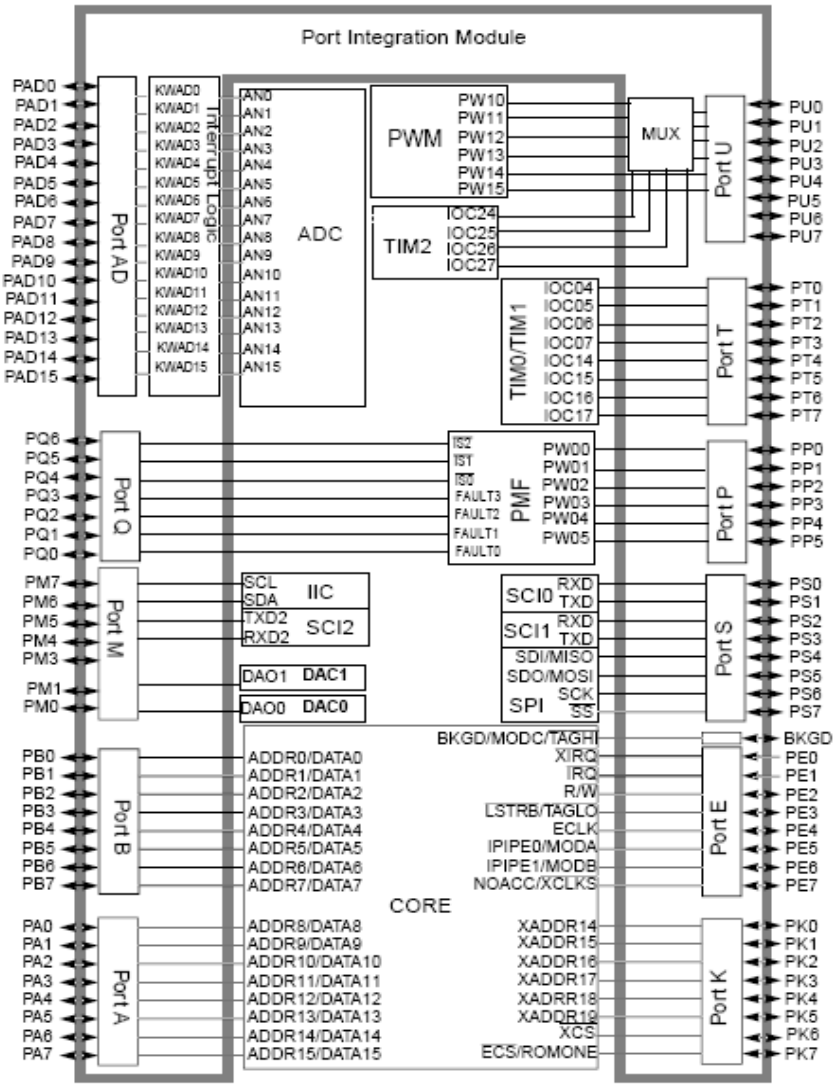
Vocabulaire :

Périphérique
Entrée –sortie – E/S
Bit - octet
Ports – fils – pattes
Registres de données
Registres de contrôle
Résistances de Pull Up et Pull Down

3.1/ Désignation des fils du composant

3.1.1/ Ports parallèles d'entrées et de sorties "ports E/S"

On regroupe des fils d'entrées ou de sorties en blocs (port) pour pouvoir les identifier plus facilement dans les registres de contrôle.
On trouve les ports A, B, T, U, S, M, P, Q, AD, E, K qui comportent en général 8 bits sauf pour le port AD (16bits) M (7 bits) P (6 bits) et Q (7bits).



Les E/S sur ces ports peuvent être de niveaux logiques ou pour certains analogiques. Presque tous ces fils ont des fonctions multiples, soit comme des E/S à niveau logique ou analogique, soit en entrée logique, soit comme une sortie ou entrée de communication série… Tout ceci doit être configuré pendant une phase dite "d'initialisation du microcontrôleur" en positionnant la valeur de certains bits dans des "registres de contrôle" associés aux ports.

Nom courant	Nom générique	Fonctions	fonctions
Port A	PORTA	8 E/S	GPIO, data, addr.
Port B	PORTB	8 E/S	GPIO
Port T	PTT	8 E/S	GPIO, IOC0, IOC1
Port U	PTU	8 E/S	GPIO, PW1, IOC2
Port S	PTS	8 E/S	GPIO, SPI, SCI0, SCI1,
Port M	PTM	7 E/S	GPIO, SCI2, I2C, DAO1, DAO2
Port P	PTP	6 E/S	GPIO, PW0
Port Q	PTU	7 E/S	GPIO, Fault
Port AD	PTAD	16 E/S	GPIO, DAC, KWAD
Port E	PE	8 E/S	GPIO, Contrôle, Intrerrupts
Port K	PK	8 E/S	GPIO, Contrôle

GPIO : general purpose input output
DAC : analogic to digital converter inputs
Data : utilisé en données lorsque adressage étendu
Addr. : utilisé en complément d'adresse lorsque adressage étendu
PW : pulse wave modulation
SPI : serial peripheral interface
SCI : serial communication asynchrone
DAO : digital to analogic converter output
Fault : entrée défaut
KWAD : keyboard wake up interrupts
Inrerrupts : entrées d'interruptions prioritaires
Contrôle: entrées/sorties de contrôle

3.1.2/ Les registres des "ports E/S"

Ces registres permettent de définir la configuration de chacun des fils d'entrées ou de sorties. En effet, suivant l'application chaque fil peut être utilisé de façon très différente; il convient donc de configurer au démarrage du programme, dans la phase d'initialisation, le rôle de chacun d'eux.

On peut regrouper les registres sous la forme :

- De registres I/O dans lesquels le concepteur vient lire ou écrire par programme la valeur des bits reliés aux fils d'E/S du port du microcontrôleur.
- De registres de direction qui placent les fils comme des sorties ou des entrées.
- de registres de recopie (dit d'entrée) qui lie la valeur sur les fils des pattes du microcontrôleur, pour vérifier qu'il n'y a pas de court-circuit ou de surcharge.

- De registres de résistances additionnelles de Pull up ou Pull down, qui sont des résistances qui viennent tirer les fils vers le +5V ou la masse plutôt que de le laisser en l'air.
- De registres qui permettent de réduire la consommation en courant des fils.
- De registres de sélection de polarité, soit front montant soit descendant, de détection de changement d'état sur les fils.
- De registres de validation locale de déclenchement d'interruptions sur changement d'état des fils.
- De registres contenant les drapeaux affectés aux interruptions.
- De registres particuliers, de choix de fonction MODRR, par exemple entre Timer et PWM ou de collecteur ouvert WOM, qui permettent des associations sous forme de ou entre des fils initialisés comme des sorties

Le tableau suivant fait le bilan des registres principaux attribués à chaque port :

Nom Registre du port	Registre de direction	Registre d'entré	Registre de Pull	Registre réduction puissance	Registre particulier	Registre de polarité	Registre interruption locale	Registre des drapeaux
PORTA	DDRAA							
PORTB	DDRAB							
PTT	DDRT	PTIT	PERT, PPST	RDRT				
PTU	DDRU	PTIU	PERU, PPSU	RDRU	MODRR			
PTS	DDRS	PTIS	PERS, PPSS	RDRS	WOMS			
PTM	DDRM	PTIM	PERM, PPSM	RDRM	WOMM			
PTP	DDRP	PTIP	PERP, PPSP	RDRP				
PTQ	DDRQ	PTIQ	PERQ, PPSQ	RDRQ				
PTAD	DDRAD	PTIAD	PERAD	RDRAD	ATDDIEN	PPSAD	PIEAD	PIFAD
PORTE	DDRE		PUCR		PEAR, MODE			
PORTK	DDRK		PUCR		MODE			

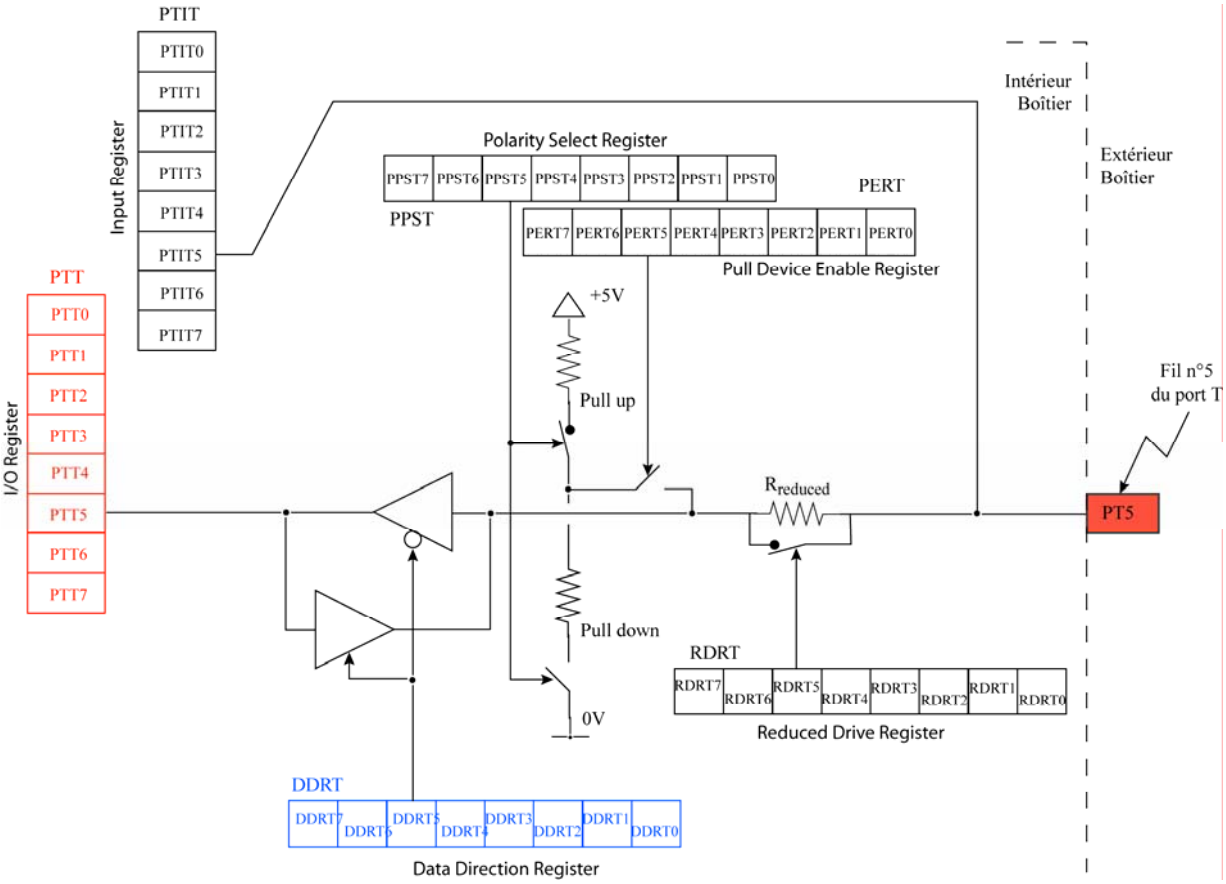
WOM : configuration des sorties en Push pull ou collecteur ouvert
ATDDIEN : configuration d'entrées logiques ou analogiques
MODE : configuration des modes single chip ou extended…

3.1.3/ Schéma équivalent d'un fil du boîtier du MC9S12

A/ Ports T, U, S, M, P, Q

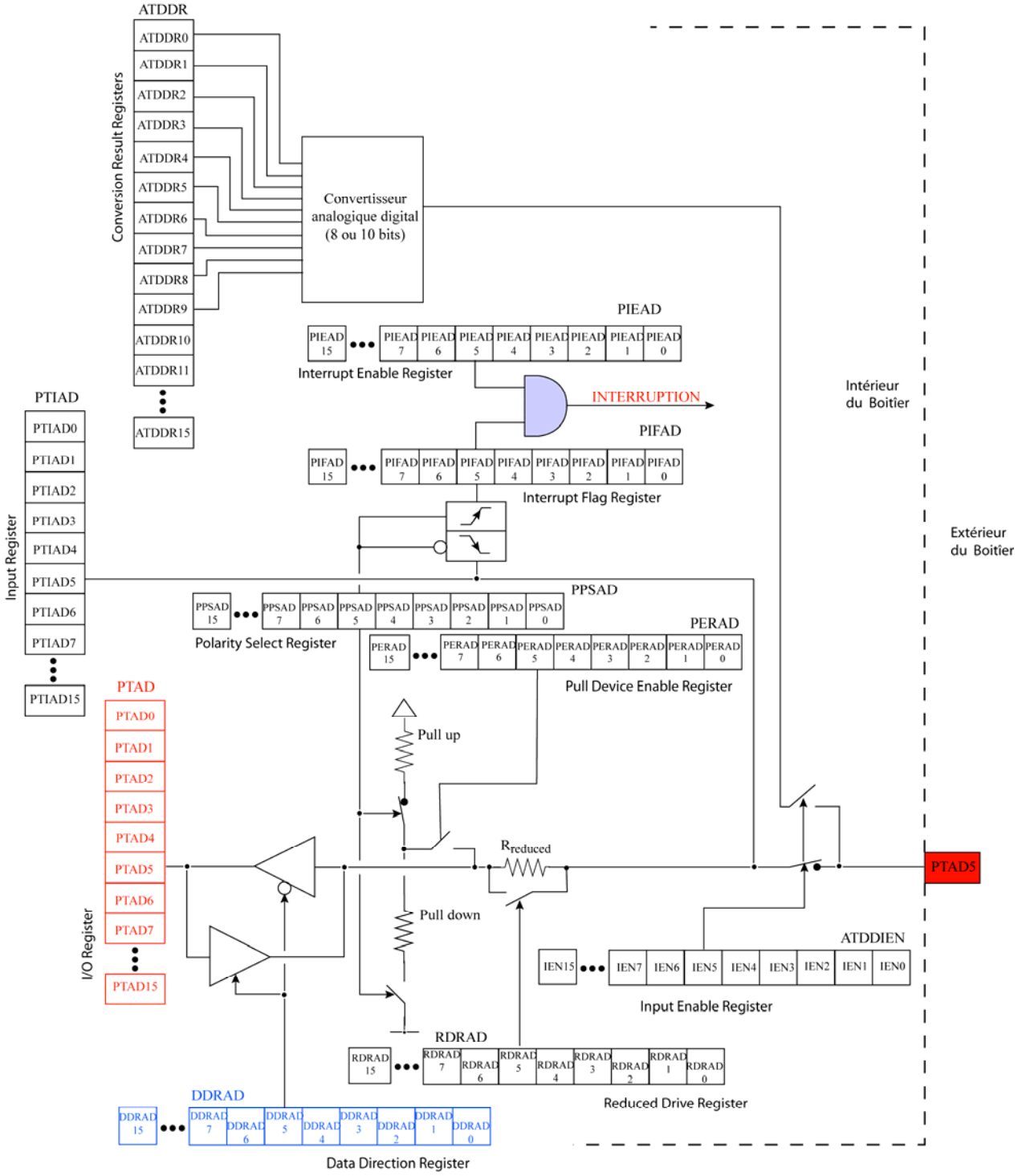
Pour mieux comprendre les relations qui existent entre registre du port, registre d'entrée, registres de configuration et fils des E/S, nous avons réalisé un schéma équivalent d'un fil du boîtier du 68HC12. Il s'agit du fil n°6 du port T, correspondant au bit 5 du port T (le fil n°1 est le bit 0 du port). Les registres les plus importants sont en couleur. En bleu, le registre DDRT de direction pour configurer le fil PT5 soit en entrée soit en sortie. En rouge, le registre PTT dans lequel le programme vient écrire pour positionner le fil, soit à "1" soit à "0", si PT5 est en sortie, ou si PT5 est en entrée, ce registre vient lire le niveau logique du fil PT5.

On comprend le rôle des registres de résistances additionnelles PPST et PERT, qui par défaut ne positionne aucune résistance. De même, le registre de réduction de consommation RDRT sort par défaut la pleine puissance dont il est capable.



B/ Ports AD

Le port AD, 16 bits, est particulier car il permet, en plus des fonctions classiques des autres ports, d'assurer la détection de changement d'état sur ses fils. Les registres associés sont donc différents.



3.1.3/ Définition symbolique des registres

Dans la documentation constructeur, les registres sont décrits en faisant apparaître, la correspondance entre bits et fils du boîtier, mais aussi en indiquant des informations supplémentaires telles que la valeur des bits du registre par défaut (valeur au reset) les actions après lecture ou écriture et l'adresse-mémoire.

Address Offset: $__02

	Bit 7	6	5	4	3	2	1	Bit 0
Read: Write:	DDRT7	DDRT6	DDRT5	DDRT4	DDRT3	DDRT2	DDRT1	DDRT0
Reset:	0	0	0	0	0	0	0	0

Read:Anytime.

Write:Anytime

3.2/ Méthodologie de configuration des fils d'un port d'E/S

3.2.1/ Les registres importants

Parmi tous les registres affectés à un port, certains sont très importants :

- Le registre de lecture et d'écriture, PTx, qui donne la valeur logique du bit associé au fil. Si le fil est une entrée, le registre doit être utilisé en lecture, sinon le programme doit écrire dans le registre pour forcer le fil de sortie. Si par erreur, le programme vient écrire dans le registre alors que le fil est une entrée, alors rien ne se passe, l'écriture est simplement ignorée.
- Le registre de direction, DDRx, qui permet de configurer le fil comme une entrée, de l'extérieur du boîtier vers l'intérieur, ou une sortie.
- Le registre d'entrée qui permet de lire l'état logique réel du fil programmé en sortie. Son utilisation est justifiée que s'il s'agit d'un fil de sortie et que l'on a des craintes de court-circuit ou de surcharge vis-à-vis du matériel électronique connecté sur le fil.

Pour un processus de détection de changement d'état des fils ou d'interruption, les registres suivants deviendront eux aussi très importants :

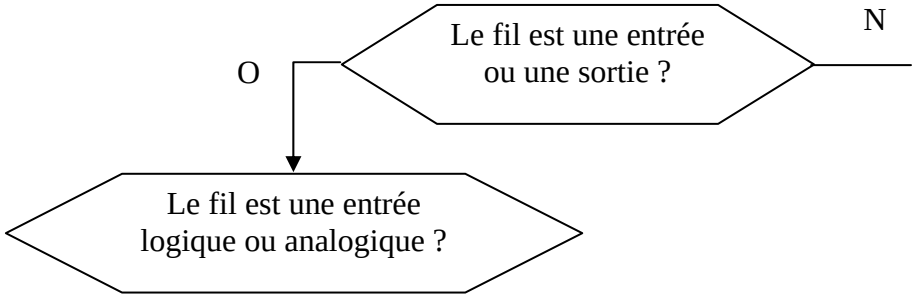
- Le registre de polarité, PPSAD, qui indique si le changement d'état est déclenché sur front montant, front descendant ou les deux.
- Le registre des drapeaux, PIFAD, qui contient les indicateurs de changement d'état, dans lequel une écriture est obligatoire pour mettre le bit du drapeau à "0". Ceci sera plus détaillé dans le chapitre concernant les interruption et changement d'état.
- Le registre des interrupteurs locaux, PIEAD, qui autorise l'interruption du fil.

Dans un processus d'acquisition analogique, les registres suivants sont utilisés :

- Le registre 16 bits du résultat de la conversion, ATDDRx.
- Les registres d'indication de fin de séquence de mesure et de fin de conversion, ATDCTL0 et ADSTAT1,2.
- Le registre d'activation du convertisseur analogique-digital, ATDCTL2.
- Le registre du nombre de conversion dans une séquence, ATDCTL3
- Le registre de configuration du résultat, 8bits ou 16 bits, ATDCTL4.
- Le registre du mode de conversion, conversion une seule fois en boucle…une ou plusieurs entrées, résultat justifié à droite ou à gauche dans le registre de résultat, ATDCTL5

3.2.2/ Le processus de configuration

Dans le cas général, le processus de configuration des fils d'E/S suit un algorithme simple, à travers les questions suivantes.



La réponse à chacune de ces questions impose le positionnement des bits dans les registres correspondants.

La suite du cours nous demandera de compléter le processus de questionnement conduisant à l'initialisation du microcontrôleur.

3.2.3/ Exemple de code d'initialisation

```
void PeriphInit(void)
{
   DDRP_DDRP4=1; //PortP bit4 en sortie
   PTP_PTP4=0;   //active l'ampli des leds

   DDRB=0xFF;    // PortB  en sortie
   PORTB=0;      //éteint les leds

    DDRAD_DDRAD4=0;   //le fil PAD04 est une entrée
    ATDDIEN_IEN4=1 ;     //le fil PAD04 est une entrée logique
}
```

Chapitre 4

Processus de détection de changement d'état et d'interruption

Lorsqu'un fil est configuré comme une entrée, le programme peut venir aléatoirement lire son état dans le registre du port correspondant. S'il s'agit d'une entrée qui varie rarement ou qui ne demande pas un traitement immédiat, par exemple la mise en marche du ventilateur de l'air conditionné dans une voiture, une séquence lecture aléatoire est suffisante. Par contre, si l'entrée demande une action immédiate de la part du microcontrôleur, par exemple l'arrêt d'urgence sur une machine outil, il devient indispensable d'indiquer au microcontrôleur que l'entrée à changer d'état. En plus d'indiquer le changement, il peut devenir impératif de déclencher immédiatement une suite d'actions en parallèle du fonctionnement du programme principal.

Objectifs :

- Connaître le principe de changement d'état sur un fil d'entrée par levé de son drapeau.
- Connaître le processus de réinitialisation des drapeaux.
- Connaître le principe de configuration du fil E/S avec changement d'état soit sur front montant ou descendant, soit sur les deux.
- Connaître le principe du dispositif d'interruption.
- Savoir configurer une entrée pour déclencher une interruption.
- Savoir écrire un programme d'interruption.
- Savoir utiliser les masques d'interruption.

Vocabulaire :

Détection de changement d'état
Processus de scrutation
Processus d'interruption
Drapeau
Masque général – Masque local
Vecteur d'interruption

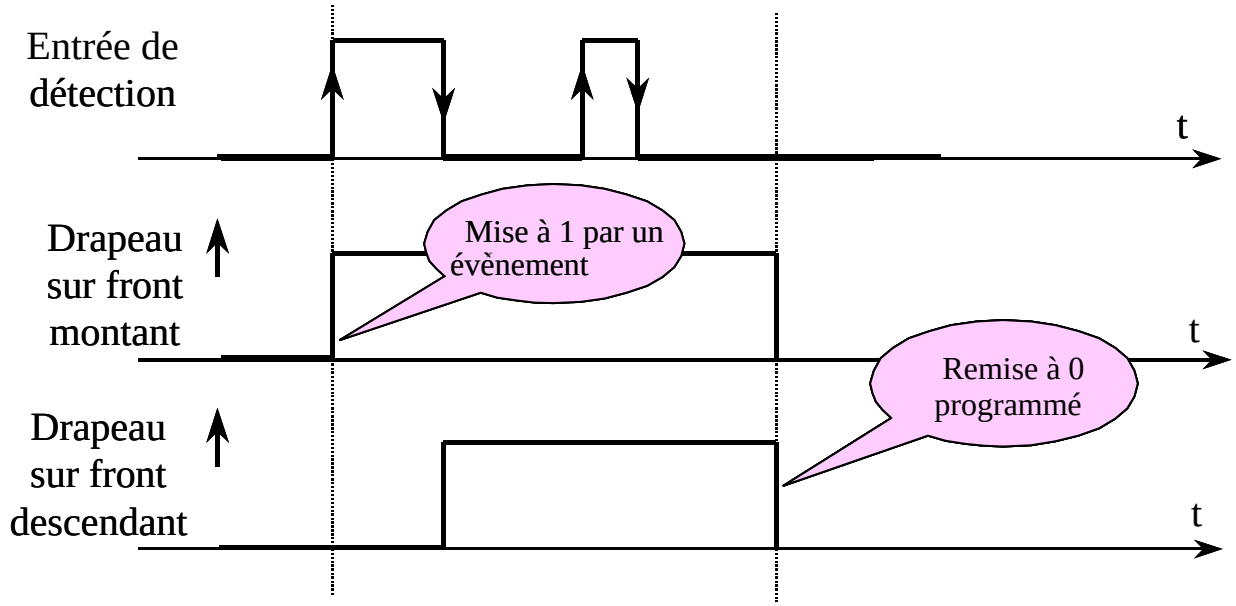
4.1/ Détection de "changement d'état"

Un changement d'état sur un fil d'un port peut être détecté par logiciel ou par un dispositif matériel interne au microcontrôleur.

Par logiciel, une boucle attend le changement d'état en venant lire régulièrement le registre associé au fil. Cette opération est dite opération de "scrutation". Cette technique mobilise toute l'attention du microcontrôleur pour un évènement qui n'aura peut-être pas lieu! On pourrait utiliser le temps de façon plus rationnelle en ne venant lire l'état du fil moins souvent, mais alors on doit admettre premièrement que le changement d'état reste suffisamment longtemps pour que le programme ne rate pas l'événement, et deuxièmement supposé que le temps perdu ne soit pas important pour les actions à mener.

Dans le microcontrôleur est implanté un dispositif matériel qui avertit qu'un changement d'état à eu lieu sur un fil. La partie matérielle donne l'information de détection en "levant un drapeau" c'est-à-dire en forçant à "1" un bit du registre des drapeaux associé au port du microcontrôleur.
Le programme n'a plus alors à se soucier de scruter l'entrée dans la crainte de perdre l'information. Le drapeau indique que le changement d'état a eu lieu, il mémorise cette information même si le fil d'entrée revient à son niveau initial. La réinitialisation du drapeau à l'état "0" se fait par programme, en écrivant un "1" à l'emplacement du bit forcé dans le registre des drapeaux.

Remarque : tous les fils d'entrée ne sont pas équipés du dispositif de déclenchement matériel.



Il convient de remarquer que si le programme ne lit pas suffisamment rapidement l'information de changement d'état ou qu'il n'a pas réinitialisé le drapeau alors plusieurs changements d'état ne seront pas détectés.

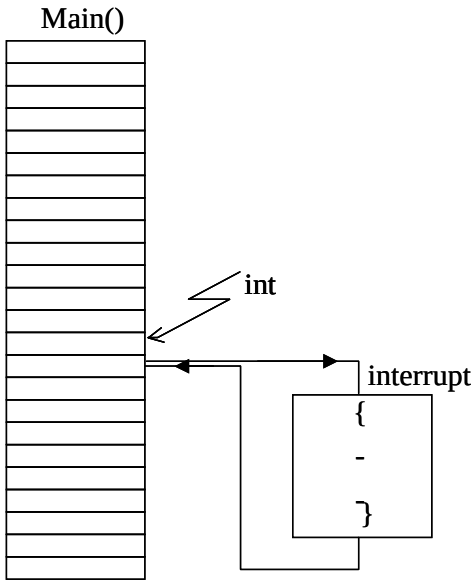
4.2/ Processus d'interruption

4.2.1/ Déclenchement d'une interruption

Le paragraphe précédent montrait comment était détecté un changement d'état sur un fil d'entrée. Pour prendre en compte le drapeau levé, le programme devait régulièrement venir lire le registre des drapeaux pour déclencher l'action nécessaire.

Le retard temporel induit par la scrutation des drapeaux peut avoir de fâcheuses conséquences. Par exemple si l'on souhaitait mesurer précisément l'intervalle de temps qui sépare deux changements d'état sur un fil, cette technique ne serait pas adaptée.

Pour ceci, on ajoute au processus de détection, le processus d'interruption qui, dès qu'un drapeau se lève, déroute l'exécution du programme principal pour exécuter une suite d'actions prioritaires. Une fois ces actions prioritaires réalisées, le programme principal reprend son exécution à l'endroit où l'interruption a eu lieu.



Cette reprise nécessite que tout le contexte du programme principal (valeur des variables, valeurs des registres…) soit mémorisé. Heureusement, l'opération de sauvegarde du contexte est automatique, et est l'opération prioritaire réalisée avant l'exécution du programme d'interruption.

Remarque : le programme d'interruption possède un organigramme propre et indépendant du programme principal. Le concepteur choisit le nom du programme d'interruption, mais ce dernier est déclaré avec comme une fonction en commençant par "interrupt void xxxx()".

Fonction d 'interruption

```
interrupt void PTAD_ISR(void)
{
 //réarmer l'indicateur
PIFAD=pifad6;
/*traitement de l'interruption*/
}
```

4.2.2/ Les vecteurs d'interruption

On associe à chaque dispositif interrupteur ou à chaque groupe interrupteur , un vecteur d'interruption. Ce vecteur comporte l'adresse de branchement du programme d'interruption.

A/ Les vecteurs d'interruption du MC9S12

N° Vecteur	Adresse Vecteur	Source d' Interruption	Masque Global	Bit local de Validation
0	0xFFFE, 0xFFFF	Reset	-	-
12	0xFFF6, 0xFFF7	IOC04	I	TIE0_C4I
13	0xFFF4, 0xFFF5	IOC05	I	TIE0_C5I
14	0xFFF2, 0xFFF3	IOC06	I	TIE0_C6I
15	0xFFF0, 0xFFF1	IOC07	I	TIE0_C7I
20	0xFFD6, 0xFFD7	SCI0	I	SCI0CR2_(TIE, RIE)
23	0xFFD0, 0xFFD1	ATD	I	ATDCTL2_ASCIE
24	0xFFCE, 0xFFCF	PORTADx	I	PIEADx
36	0xFFB6, 0xFFB7	IOC14	I	TIE1_C4I
37	0xFFB4, 0xFFB5	IOC15	I	TIE1_C5I
38	0xFFB2, 0xFFB3	IOC16	I	TIE1_C6I
39	0xFFB0, 0xFFB1	IOC17	I	TIE1_C7I
44	0xFFA6, 0xFFA7	IOC24	I	TIE2_C4I
45	0xFFA4, 0xFFA5	IOC25	I	TIE2_C5I
46	0xFFA2, 0xFFA3	IOC26	I	TIE2_C6I
47	0xFFA0, 0xFFA1	IOC27	I	TIE2_C7I

Remarque : le port AD ne comporte, pour toutes ses entées, qu'un seul vecteur d'interruption, le vecteur n°24

Il existe en plus des vecteurs d'interruptions déclenchés par les entrées classiques, des vecteurs particuliers déclenchés soit par des pattes prioritaires présentes sur tous les microcontrôleurs (RESET, IRQ), soit par logiciel (SWI), soit par erreur d'exécution des instructions…

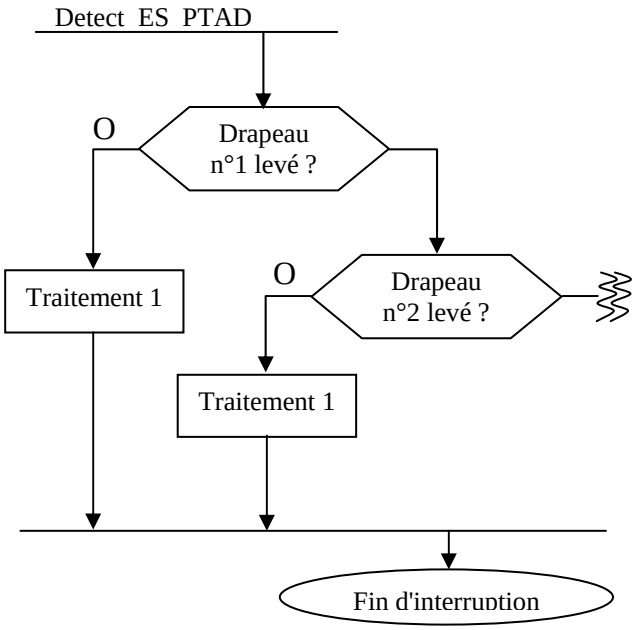
Vector Address	Source
$FFFE–$FFFF	System reset
$FFFC–$FFFD	Clock monitor reset
$FFFA–$FFFB	COP reset
$FFF8–$FFF9	Unimplemented opcode trap
$FFF6–$FFF7	Software interrupt instruction (SWI)
$FFF4–$FFF5	$\overline{\text{XIRQ}}$ signal
$FFF2–$FFF3	$\overline{\text{IRQ}}$ signal
$FF00–$FFF1	Device-specific interrupt sources (HCS12)
$FFC0–$FFF1	Device-specific interrupt sources (M68HC12)

B/ Détection dans un groupe d'interruption

Un groupe d'interruption ne possède qu'un vecteur d'interruption pour plusieurs entrées différentes.

Il convient au début du programme d'interruption de détecter qui est l'auteur de l'interruption.
Cette détection se fait par scrutation des drapeaux.

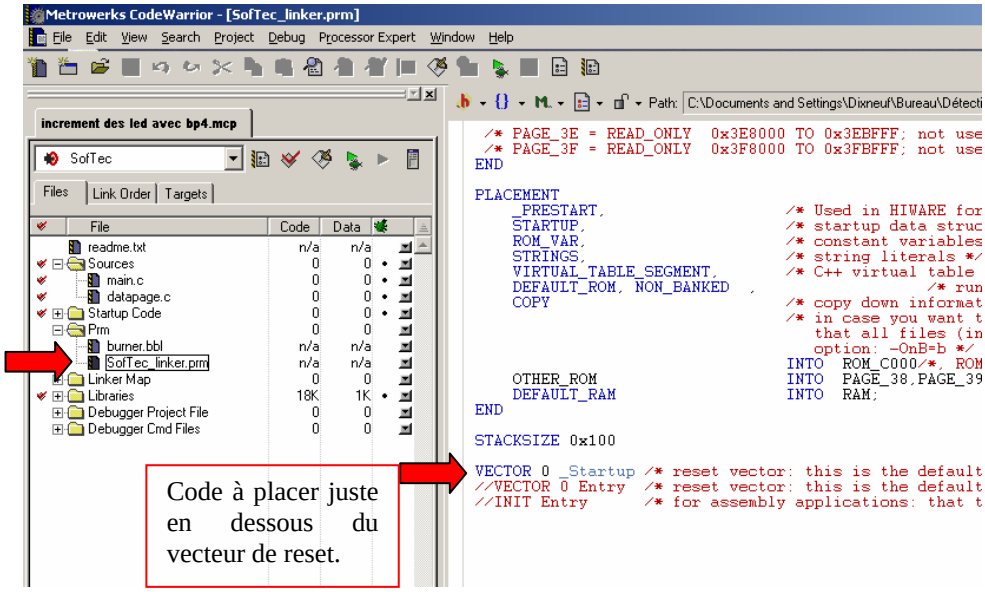
O



C/ Affectation du vecteur d'interruption

Le vecteur d'interruption doit connaître l'adresse du programme d'interruption. CodeWarrior nous aide dans cette opération, il suffit d'indiquer le nom du programme d'interruption correspondant, comme suit :

VECTOR 24 Detect_ES_PTAD /* vecteur de détection …*/



4.2.2/ Les masques d'interruption

Un masque d'interruption est une opération qui empêche l'exécution d'une interruption. Il existe plusieurs niveaux de masque, un masque général et des masques locaux.
Le masque général est utilisé rarement au cours du programme principal sauf tout au début lors de la procédure d'initialisation des ports. En effet, pourquoi déclencher une interruption alors que le microcontrôleur n'est pas encore apte à fonctionner en toute sécurité sur la carte électronique. Le masque général est validé lorsque l'instruction suivante apparaît : "DisableInterrupts();"

Les masques locaux d'interruption inhibent ou activent une interruption et/ou induisent une forme de priorité entre les interruptions. Par exemple, on peut lors du traitement d'une interruption interdire l'arrivée d'une autre interruption.
Les masques locaux sont déclarés dans le registre des masques PIEAD.

Remarque : si une interruption a été masquée, la détection sur le fil d'entrée a bien levé le drapeau. Quand le masque disparaît, l'exécution de l'interruption aura normalement lieu, sauf si entre-temps le drapeau a été volontairement baissé.

4.3/ Configuration d'une entrée du port AD

4.3.1/ Configuration en vue d'un changement d'état

A/ Configuration

Pour qu'un changement d'état sur une entrée du port AD puisse être détecté, il convient :

- Que l'entrée soit considérée comme une entrée, registre DDRAD
- Comme une entrée logique, registre ATDDIEN
- Que le front de détection soit explicité, registre PPSAD
- Que le drapeau de cette entrée soit initialement baissé, registre PIFAD

Par exemple : pour que le fil n°4 soit détecté sur front montant, il faut que : DDRAD5=0, IEN5=0, PPSAD5=1 et que PIFAD5=0.

B/ Code associé

```
  DDRAD_DDRAD5=0;              // PAD05 est une entrée
  ATDDIEN_IEN5=1;              // active PAD05 en tant qu'entrée logique
…PERAD_PERAD5=0;
  PPSAD_PPSAD5=1;          // détection sur front montant
  PIFAD=0x0010;   // baisse le drapeau en écrivant un 1 dans le bit5
```

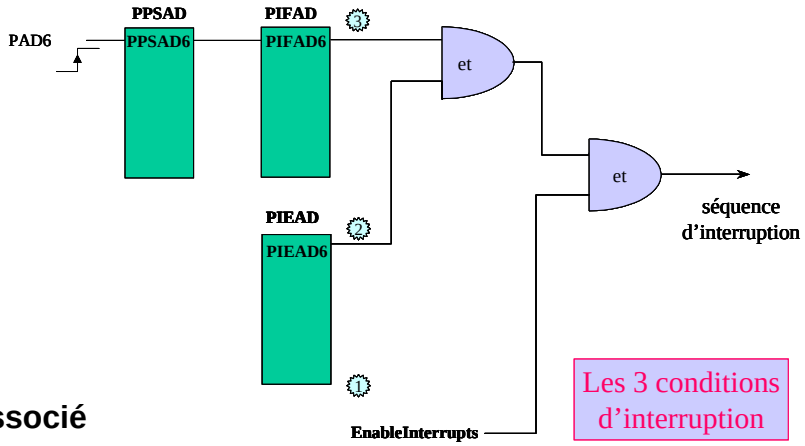
4.3.2/ Configuration en vue d'une interruption

A/ Configuration

Pour qu'une entrée puisse entraîner une interruption, il faut :

- Que le dispositif de détection du fil soit activé.
- Que le masque général soit inactif.
- Que le masque local soit inactif.
- Que le drapeau du fil d'entrée soit baissé.
- Que le vecteur d'interruption soit déclaré avec l'adresse de début de l'interruption, sinon le programme va conduire au « plantage » du µC.

Pour exemple voici la description schématique du processus de déclenchement d'une interruption provoquée par un front montant sur la patte PAD6 du port AD.

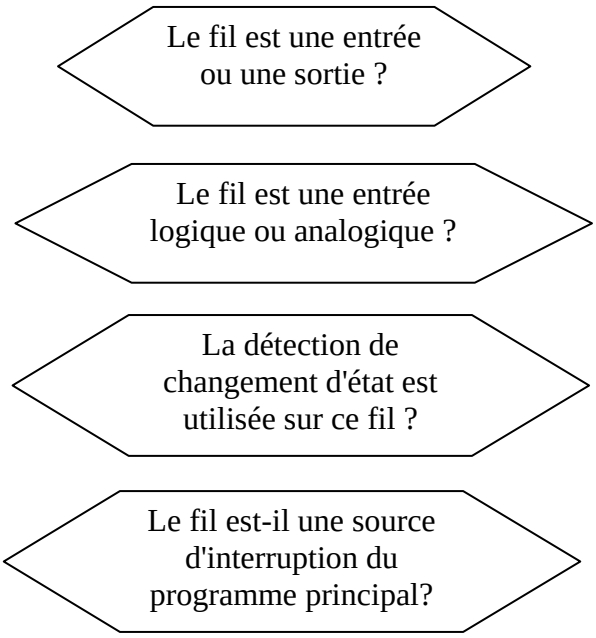


B/ Code associé

```
void PeriphInit(void)
{
    ///init détection sur PAD05 avec IT////
   /////// pas nécessaire si des résistances de pull-up sont installées en externe///////////////
   PERAD_PERAD5=1;      // active la resistance de Pull-up ou Pull-down sur PAD05
   PPSAD_PPSAD5=0;      // sélectionne pull-up et front descendant
   ////////////////////////////////////////////////////////////////////////////////////////////////////////////////
   DDRAD_DDRAD5=0;                // PAD05 est une entrée
   ATDDIEN_IEN5=1;   // active PAD05 en tant qu'entrée logique
   PIFAD=pifad5 ;          //réarme le drapeau
   PIEAD_PIEAD5=1;     //autorise les ITs sur PAD05
}
interrupt void PTADIF_ISR(void)
{
   PIFAD=pifad5 ;          //réarme le drapeau
   …
}
void main(void)
{

   PeriphInit();              // Microcontroller initialization
   EnableInterrupts;        // supprime le masque général d'interruptions
   ….
}
```

4.3.3/ Le processus de configuration

Nous avions au chapitre précédent développé un processus de configuration des fils d'E/S à travers des questions. Il convient de le compléter, car le dispositif de changement d'état et d'interruption nécessite une initialisation préalable.



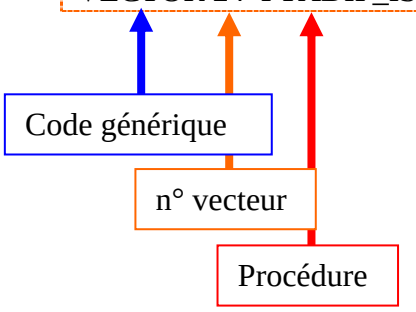
La réponse à chacune de ces questions impose le positionnement des bits dans les registres correspondants.

La suite du cours nous demandera de compléter le processus de questionnement conduisant à l'initialisation du microcontrôleur.

4.3.4/ Exemple de positionnement d'un vecteur d'interruption

Exemple d'attribution des vecteurs dans la feuille **.PRM dans le dossier PRM de CodeWarrior.

```
VECTOR 0 _Startup /* reset vector: this is the default, for a C application. */
VECTOR 12 TIM0C4_ISR
VECTOR 24  PTADIF_ISR
```



Chapitre 5

Contrôle du temps par le TIMER

Le temps, à l'intérieur du microcontrôleur, est rythmé par le circuit d'horloge interne dont la fréquence de fonctionnement dépend du quartz qui est connecté sur les fils extérieurs du boîtier (EXTAL, XTAL). La fréquence très précise du quartz est de 16 MHz pour le MC9S12e128, qui est ensuite divisée par deux (8MHz) pour produire le cycle interne de cadencement des instructions à 125ns, et fournir la fréquence au bus interne.
La mesure du temps ou la réalisation de temps très longs nous oblige à subdiviser encore la fréquence d'horloge. Un composant structurel du microcontrôleur, le TIMER, fonctionne en parallèle du circuit d'horloge et facile la gestion des temps. En plus de la génération de temps très précis, le programme peu utilisé des fonctionnalités d'entrées et de sorties directement liées au dispositif d'interruption couplé au Timer. Le Timer est un composant interne très utilisé dans le développement des programmes à base de microcontrôleur.

Objectifs :

- Connaître l'organisation interne des timers.
- Savoir utiliser les prédiviseurs pour calibrer les temps maximums.
- Savoir déclencher une interruption par débordement.
- Savoir produire, à l'aide d'interruptions, des signaux de sortie déclenchés par un temps précis (IOCx)
- Savoir acquérir la date d'un événement produit sur une entrée (IOCx), et produire une interruption à cette date.

Vocabulaire :

Timer
Prédiviseur
Débordement
Entrée de capture IOC
Sortie de comparaison IOC

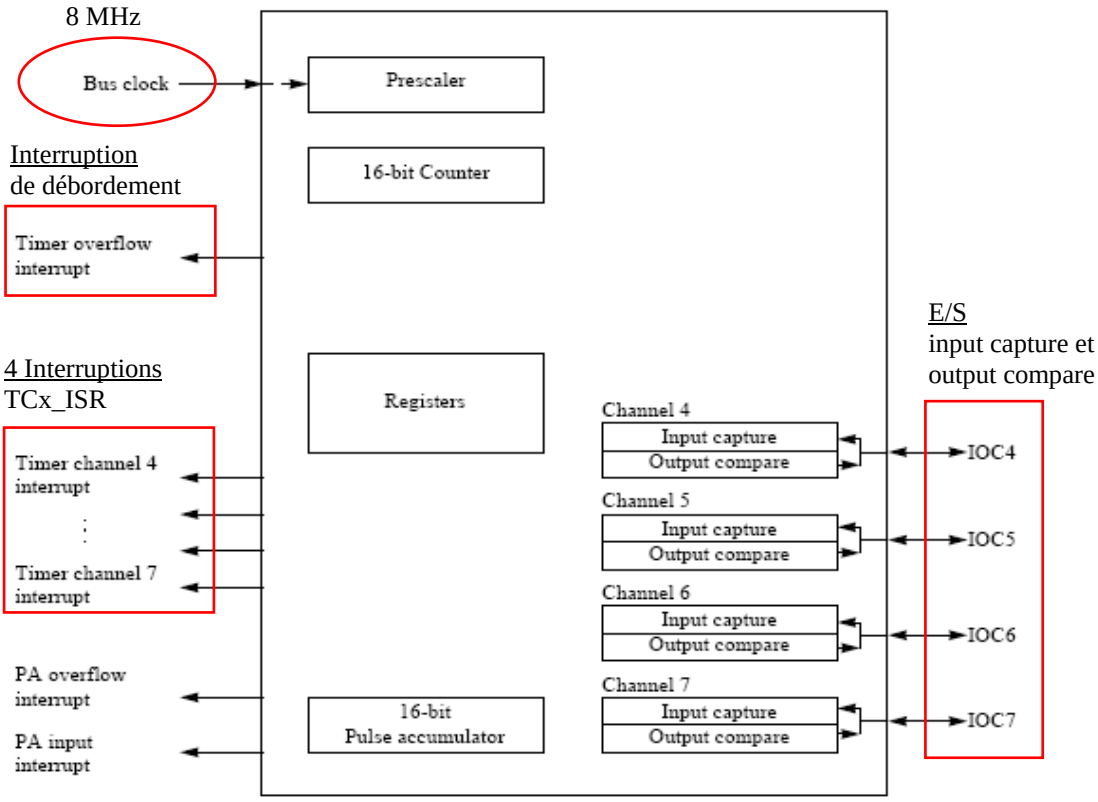
5.1/ Structure des TIMERs

Le MC9S12e128 possède trois timers indépendants, TIM0, TIM1, TIM2. Chaque timer possède un compteur de 16 bits (de 0 à 65535), un ensemble de registres, des entrées et des sorties utilisées au travers des fonctions de capture et de comparaison. La notion d'interruption vue dans le chapitre précédent sera largement utilisée dans le fonctionnement du timer.

5.1.1/ Organisation interne

La figure suivante présente l'organisation interne pour chacun des timers :

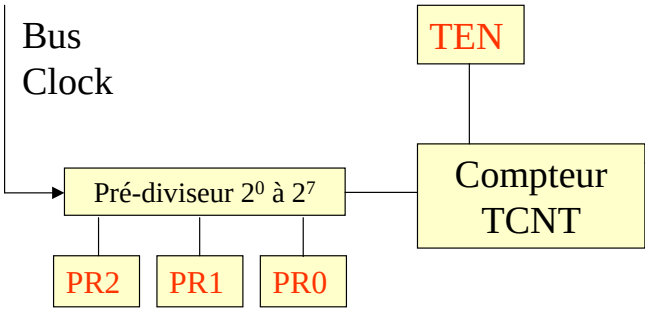
- Un diviseur de fréquence, Prescaler, alimenté par la fréquence fixe du bus (8MHz), qui fournit la fréquence de base au compteur 16 bits.
- Un compteur 16 bits qui s'incrémente, de 0 à 65535, à chaque coup d'horloge. Le comptage terminé, il recommence à la valeur 0 et produit un signal dit de "débordement".
- Un ensemble de registres de configuration, de fonctionnement
- Quatre voies d'entrées de capture, pour dater des évènements entrants.
- Quatre voies de sortie pour produire des signaux à des dates précises.
- Quatre sources d'interruption, chacune associée à un vecteur d'interruption différent, produites à des dates précises.
- Une source d'interruption à partir de l'événement overflow, produit à chaque débordement du compteur.
- Un accumulateur d'impulsions et ses deux voies d'interruption d'entrée et de sortie.



5.1.2/ Initialisation d'un TIMER

Chacun des trois timers est par défaut à l'arrêt, aucun comptage ne s'effectue, pour des raisons d'économie d'énergie. Le bit TEN (Timer Enable) est chargé de la mise en route du timer, dont le comptage commence par défaut à 0.
Le registre TCNT, 16 bits, s'incrémente à chaque coup d'horloge du prédiviseur. TCNT est un registre qui fournit par lecture la valeur du compteur, mais dans lequel il est possible d'écrire, par exemple pour initialiser la valeur de début de comptage.
Le prédiviseur possède 3 bits (8 valeurs) de configuration, PR0 à PR2, qui conditionnent la fréquence d'horloge qui alimente le compteur TCNT.

Pré-diviseur			Bus Clock = 8MHz		
PR2	PR1	PR0	Division	Résolution	Maximum
0	0	0	BC/1	125 ns	8,192 ms
0	0	1	BC/2	250 ns	16,384 ms
0	1	0	BC/4	500 ns	32,768 ms
0	1	1	BC/8	1 µs	65,536 ms
1	0	0	BC/16	2 µs	131,072 ms
1	0	1	BC/32	4 µs	262,144 ms
1	1	0	BC/64	8 µs	524,288 ms
1	1	1	BC/128	16 µs	1,049 s



5.1.3/ Registres associés au Timer

Register offset:$_04-$_05

TCNT

	Bit 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Bit 0
R W	tcnt 15	tcnt 14	tcnt 13	tcnt 12	tcnt 11	tcnt 10	tcnt 9	tcnt 8	tcnt 7	tcnt 6	tcnt 5	tcnt 4	tcnt 3	tcnt 2	tcnt 1	tcnt t 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Register offset:$_06

TSCR1

	Bit 7	6	5	4	3	2	1	Bit 0
R W		TSWAI	TSFRZ	TFFCA	0	0	0	0
RESET:	0	0	0	0	0	0	0	0

= Reserved

Register offset:$_0D

TSCR2

	Bit 7	6	5	4	3	2	1	Bit 0
R W	TOI	0	0	0	TCRE			
RESET:	0	0	0	0	0	0	0	0

= Reserved

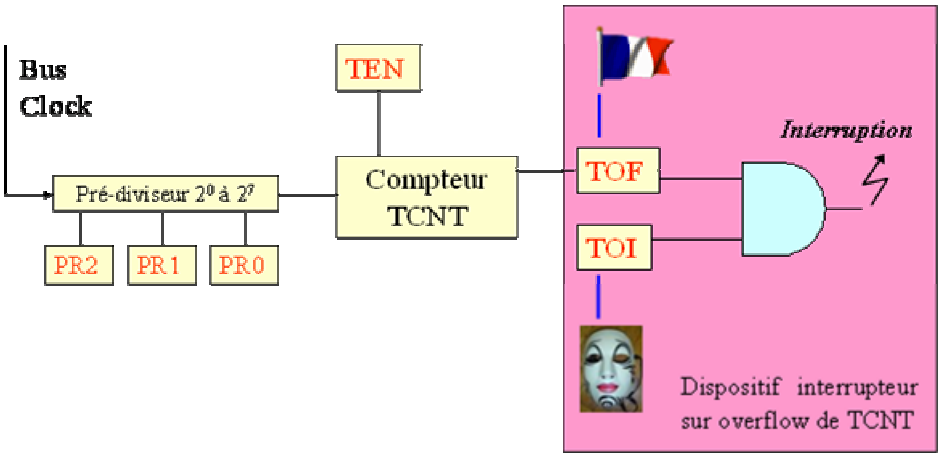
5.1.3/ Code d'initialisation

```
TIM0_TSCR2_PR0=1; //prédiviseur=1, résolution 250ns
TIM0_TSCR1_TEN=1; //active le timer 0
```

5.2/ Débordement (overflow) du Timer

5.2.1/ Principe de débordement

Lorsque le contenu de TCNT atteint la valeur maximale, 0xFFFF soit 65535, alors un drapeau TOF se lève. Parallèlement une procédure d'interruption "TOV_ISR" est déclenchée si celle-ci a été préalablement initialisée.



5.2.1/ Les registres

Register offset:S_0F

	Bit 7	6	5	4	3	2	1	Bit 0
TFLG2 R		0	0	0	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Register offset:S_0D

	Bit 7	6	5	4	3	2	1	Bit 0
TSCR2 R		0	0	0	TCRE	PR2	PR1	PR0
W								
RESET:	0	0	0	0	0	0	0	0

= Reserved

5.1.3/ Code d'initialisation

On rappelle que la procédure de réarmement d'un drapeau consiste à écrire un "1" en lieu et place du drapeau.

```
# define  tof  0b10000000
   TIM0_TSCR2_PR0=1; //prédiviseur=1, résolution 250ns
   TIM0_TSCR1_TEN=1; //active le timer 0
   TIM0_TFLG2=tof; //réarme le drapeau pour le débordement
   TIM0_TSCR2_TOI=1 ; //autorise l'interruption de débordement
```

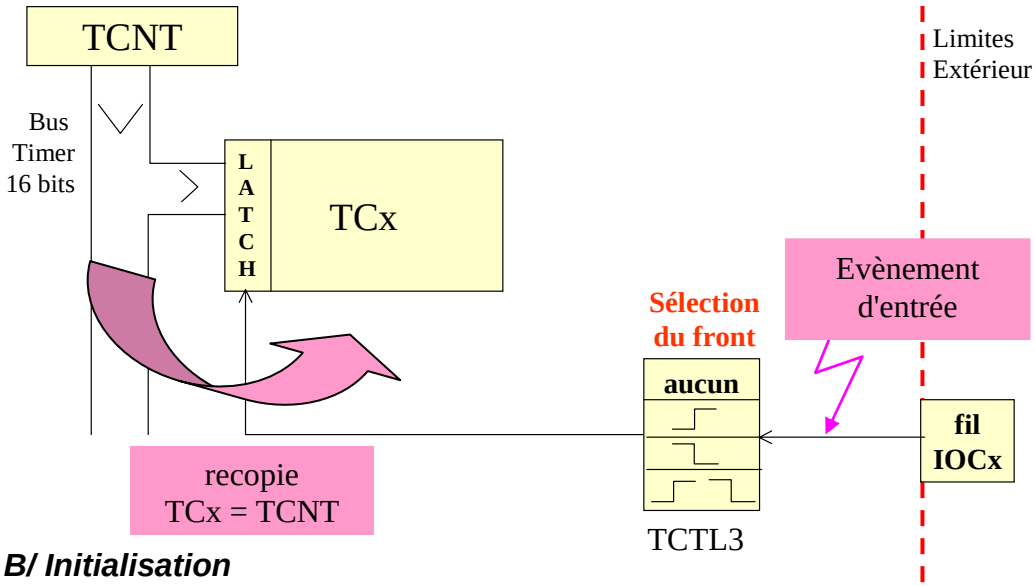
5.3/ Entrées de capture (Input Capture) IOCx

Chacun des trois timers comporte quatre E/S qui peuvent servir aussi bien comme entrée de capture que comme sortie de comparaison.

5.3.1/ Principe des 4 entrées de capture par timer

A/ Principe

Lorsqu'un changement d'état est annoncé sur une entrée de capture IOCx (IOC4 à IOC7 pour TIM0) alors à cet instant, si le dispositif est correctement initialisé, le contenu du registre TCNT du compteur est recopié dans le registre TCx (TC4 à TC7 pour TIM0), lui aussi de 16 bits, datant ainsi l'arrivée de l'événement.
Le programme peut alors utiliser cette information, par exemple en calculant le temps séparant deux événements sur une entrée de capture ou entre deux entrées de capture.



B/ Initialisation

Les quatre E/S IOCx doivent être initialisées comme des entrées par le bit IOSx (IOS4 à IOS7 pour TIM0) dans le registre TIOS. IOCx est une entrée si IOSx est à "0", sinon IOCx devient une sortie de comparaison.

L'événement déclencheur de la recopie TCx=TCNT à quatre positions possibles, entrée inactive, front montant, front descendant, changement d'état. L'un de ces quatre états est initialisé par les deux bits, EDGxA et EDGxB, dans le registre TCTL3.

EDGxB	EDGxA	Configuration
0	0	Capture désactivée
0	1	Capture sur front montant
1	0	Capture sur front descendant
1	1	Capture sur n'importe quel front

C/ Les registres

TIOS

	Bit 7	6	5	4	3	2	1	Bit 0
R	IOS7	IOS6	IOS5	IOS4	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Reserved

TCTL3

	Bit 7	6	5	4	3	2	1	Bit 0
R	EDG7B	EDG7A	EDG6B	EDG6A	EDG5B	EDG5A	EDG4B	EDG4A
W								
RESET	0	0	0	0	0	0	0	0

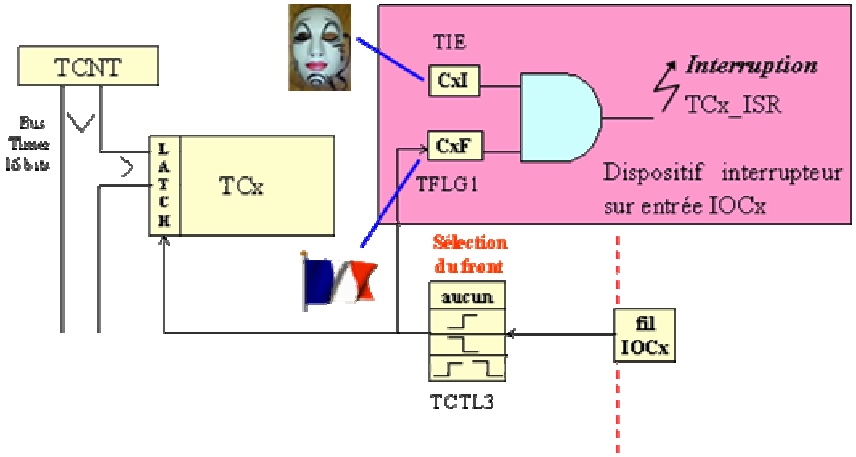
D/ Code d'initialisation

```
/////TIM2 canal 6 est utilisé comme entrée de capture//////
TIM2_TSCR1_TEN=1;   //active le timer 2
TIM2_TIOS_IOS6=0;   //active l'entrée comme entrée de capture
TIM2_TCTL3_EDG6A=0;  //front descendant sur TC6
TIM2_TCTL3_EDG6B=1 ;
```

5.3.2/ Interruption générée par une entrée de capture

A/ Principe de déclenchement de TCx_ISR

L'événement sur l'entrée IOCx produit une recopie de TCNT dans TCx, mais peut aussi produire une interruption TCx_ISR, si le dispositif interrupteur est initialisé. L'événement crée la levée d'un drapeau CxF dans le registre TFLG1. Le masque local d'interruption CxI dans le registre TIE autorise le déclenchement de l'interruption.



B/ Les registres et vecteurs d'interruption

TIE

	Bit 7	6	5	4	3	2	1	Bit 0
R	C7I	C6I	C5I	C4I	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Reserved

TFLG1

	Bit 7	6	5	4	3	2	1	Bit 0
R	C7F	C6F	C5F	C4F	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Reserved

Les v

C/ Code d'initialisation

On rappelle que la procédure de réarmement d'un drapeau consiste à écrire un "1" en lieu et place du drapeau.

```
TIM0_TFLG1=0x40;    //réarme le drapeau entrée IOC6
TIM0_TIE_C6I=1 ; //autorise l'interruption
```

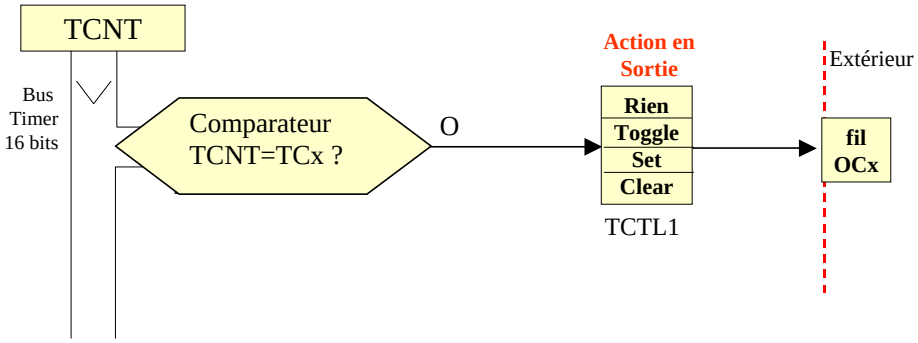
5.4/ Sortie de comparaison (Output compare) IOCx

Chacun des trois timers comporte quatre E/S qui peuvent servir aussi bien comme entrée de capture que comme sortie de comparaison. Le choix de la direction se fait par la valeur du bit IOSx ("0"= entrée, "1"=sortie) dans le registre TIOS comme indiqué dans le paragraphe précédent "Entrées de capture".

5.4.1/ Principe des 4 sorties de comparaison par timer

A/ Principe

Un dispositif interne au microcontrôleur compare en permanence le contenu TCNT avec le contenu des registres TCx (TC4 à TC7 pour TIM0). Lorsqu'il il y a égalité entre TCNT et TCx, un signal peut être transmis en sortie du microcontrôleur à travers la sortie IOCx.



B/ Initialisation

Le signal transmis peut être un changement d'état (un "0" suit un "1" et vice-versa, c'est la fonction Toggle), une mise à "1" (fonction Set), une mise à "0" (fonction Clear). La combinaison des bits OMx et OLx dans le registre TCTL1, attribue la fonction désirée.

OMx	OLx	Configuration
0	0	Pas d'action sur la sortie
0	1	Inversion de la sortie (toggle)
1	0	Mise à 0 de la sortie (clear)
1	1	Mise à 1 de la sortie (set)

C/ Les registres

TIOS

	Bit 7	6	5	4	3	2	1	Bit 0
R	IOS7	IOS6	IOS5	IOS4	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Reserved

TCTL1

	Bit 7	6	5	4	3	2	1	Bit 0
R	OM7	OL7	OM6	OL6	OM5	OL5	OM4	OL4
W								
RESET	0	0	0	0	0	0	0	0

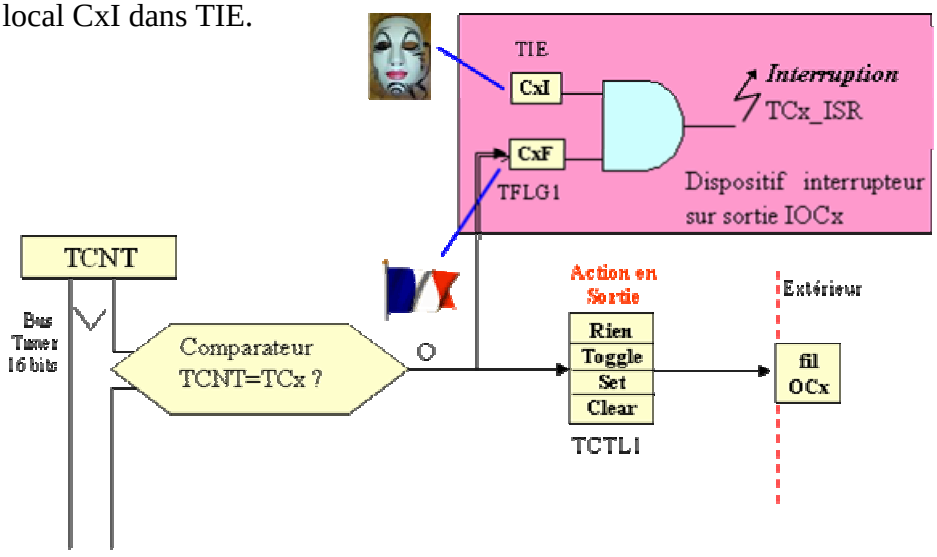
D/ Code d'initialisation

```
TIM0_TIOS_IOS4=1; //active la fonction sortie de comparaison voie 4
TIM2_TCTL1_OM4=0;  //fonction de bascule sur IOC4
TIM2_TCTL1_OL4=1 ;
```

5.4.2/ Interruption générée par comparaison

A/ Principe de déclenchement de TCx_ISR

Parallèlement à la génération du signal IOCx après égalité entre les registres TCNT et TCx, un dispositif interrupteur permet le déclenchement de l'interruption TCx_ISR. Ce dispositif comprend le drapeau CxF dans le registre TFLG1, le masque local CxI dans TIE.



B/ Les registres et vecteurs d'interruption

TIE

	Bit 7	6	5	4	3	2	1	Bit 0
R	C7I	C6I	C5I	C4I	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Reserved

TFLG1

	Bit 7	6	5	4	3	2	1	Bit 0
R	C7F	C6F	C5F	C4F	0	0	0	0
W								
RESET:	0	0	0	0	0	0	0	0

= Reserved

Les vecteurs sont donnés dans la table à la fin du chapitre.

D/ Code d'initialisation

On rappelle que la procédure de réarmement d'un drapeau consiste à écrire un "1" en lieu et place du drapeau.

```
TIM0_TIOS_IOS4=1; //active la fonction sortie de comparaison TC4
TIM2_TCTL3_OM4=0; TIM2_TCTL1_OL4=1 ;
TIM0_TFLG1=0x10;  //réarme le drapeau
TIM0_TIE_C4I=1;     //autorise l'interruption sur la voie 4
```

5.4/ Vecteurs d'interruptions des Timers

N° Vecteur	Adresse Vecteur	Source d' Interruption	Masque Global	Bit local de Validation
0	0xFFFE, 0xFFFF	Reset	-	-
12	0xFFF6, 0xFFF7	IOC04	I	TIE0_C4I
13	0xFFF4, 0xFFF5	IOC05	I	TIE0_C5I
14	0xFFF2, 0xFFF3	IOC06	I	TIE0_C6I
15	0xFFF0, 0xFFF1	IOC07	I	TIE0_C7I
20	0xFFD6, 0xFFD7	SCI0	I	SCI0CR2_(TIE, RIE)
23	0xFFD0, 0xFFD1	ATD	I	ATDCTL2_ASCIE
24	0xFFCE, 0xFFCF	PORTADx	I	PIEADx
36	0xFFB6, 0xFFB7	IOC14	I	TIE1_C4I
37	0xFFB4, 0xFFB5	IOC15	I	TIE1_C5I
38	0xFFB2, 0xFFB3	IOC16	I	TIE1_C6I
39	0xFFB0, 0xFFB1	IOC17	I	TIE1_C7I
44	0xFFA6, 0xFFA7	IOC24	I	TIE2_C4I
45	0xFFA4, 0xFFA5	IOC25	I	TIE2_C5I
46	0xFFA2, 0xFFA3	IOC26	I	TIE2_C6I
47	0xFFA0, 0xFFA1	IOC27	I	TIE2_C7I

Chapitre 6

Convertisseur Analogique Digital

Les signaux traités par le microcontrôleur sont en général des signaux numériques. Toutefois, pour éteindre l'universalité du microcontrôleur et en faire un composant généralisé par la réalisation des fonctions de l'électronique, les concepteurs ont ajouté l'acquisition des signaux analogiques. Un signal analogique dont la tension ne doit pas dépasser les niveaux des tensions d'alimentation (0V, 5V) peut donc être appliqué sur une des pattes AN0 à AN15 du boîtier. Ce signal peut provenir d'un capteur, de température, de vitesse, d'un potentiomètre, d'un AOP...il sera alors converti à l'intérieur du microcontrôleur en un signal numérique, qui pourra alors être traité avec toute la puissance du 68HC12.

Objectifs :

- Savoir comment s'effectue une conversion analogique digitale.
- Connaître l'organisation interne des convertisseurs.
- Connaître les différents registres de configuration.
- Connaître les notions de séquence et de conversion.
- Savoir initialiser le processus de conversion.
- Savoir utiliser les interruptions de fin de séquence et de conversion.

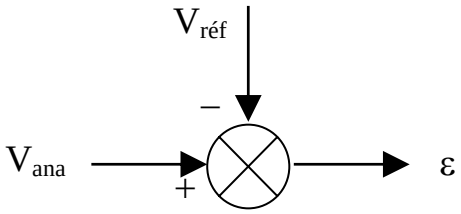
Vocabulaire :

Conversion analogique digitale – CAD – CAN - ATD
Conversion digitale analogique
Séquence – conversion
Approximations successives
Résolution 8 et 10 bits
Temps et fréquence de conversion
Multiplexeur analogique - voie
Conversion multiple MULT
Conversion continue SCAN

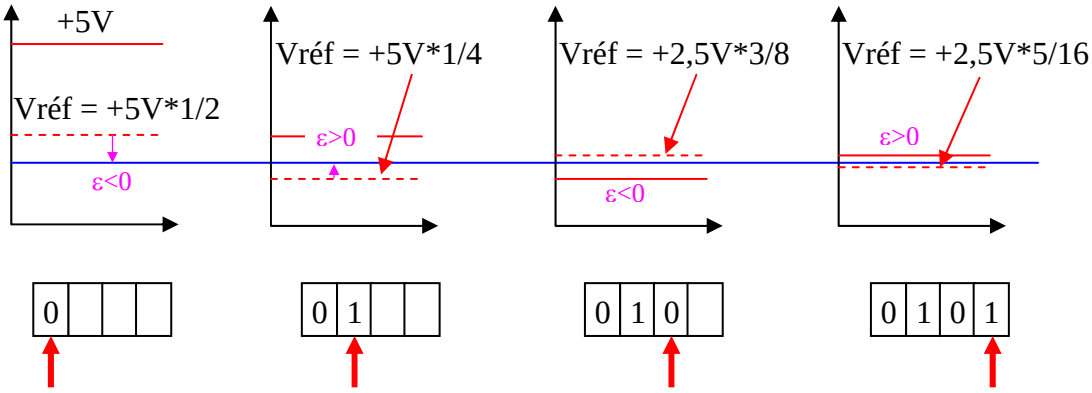
6.1/ Conversion par approximations successives

6.1.1/ Principe

Soit V_{ana} le signal analogique dont on recherche le mot numérique équivalent. On sait que Vana est compris entre 0V et +5V.
Soit $V_{réf}$ le signal de comparaison.
Pour savoir si V_{ana} est supérieure à la moitié de +5V, soit 2,5V, on réalise le montage de la figure ci-contre en imposant $V_{réf}$ égale à 2,5V.

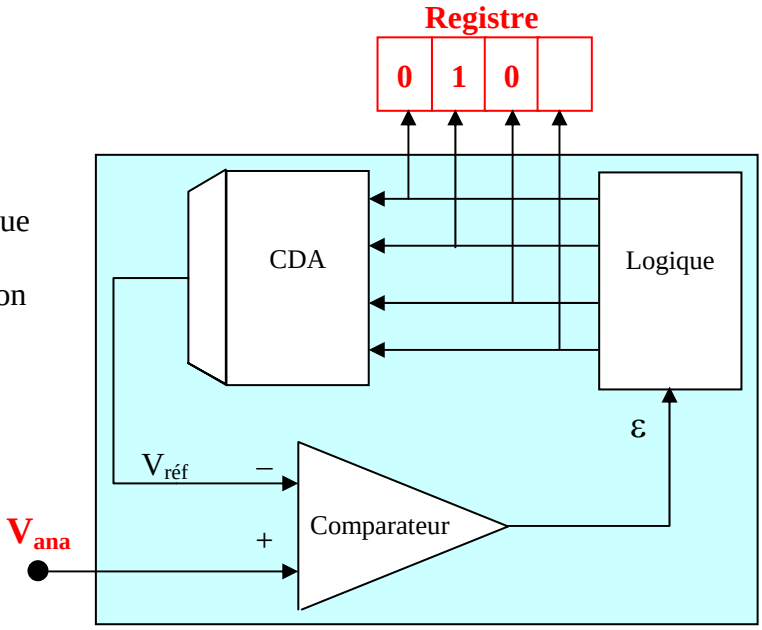


Le signe de $\varepsilon = V_{ana} - V_{réf}$ répond alors à la question précédente : si ε>0 alors V_{ana} est > 2,5V sinon V_{ana}<2,5V. Au signe de ε on affecte un bit égal à « 1 » si ε>0, sinon le bit est à « 0 ». Ce bit étant positionné, on va ensuite rechercher la valeur du bit suivant, de poids plus faible, en réitérant l'opération. La figure suivante montre le processus des itérations pour la recherche d'une conversion à quatre bits significatifs.



Le schéma-bloc d'un CAD comporte :

- un comparateur
- un convertisseur numérique analogique
- un bloc logique de décision
- un registre de sortie

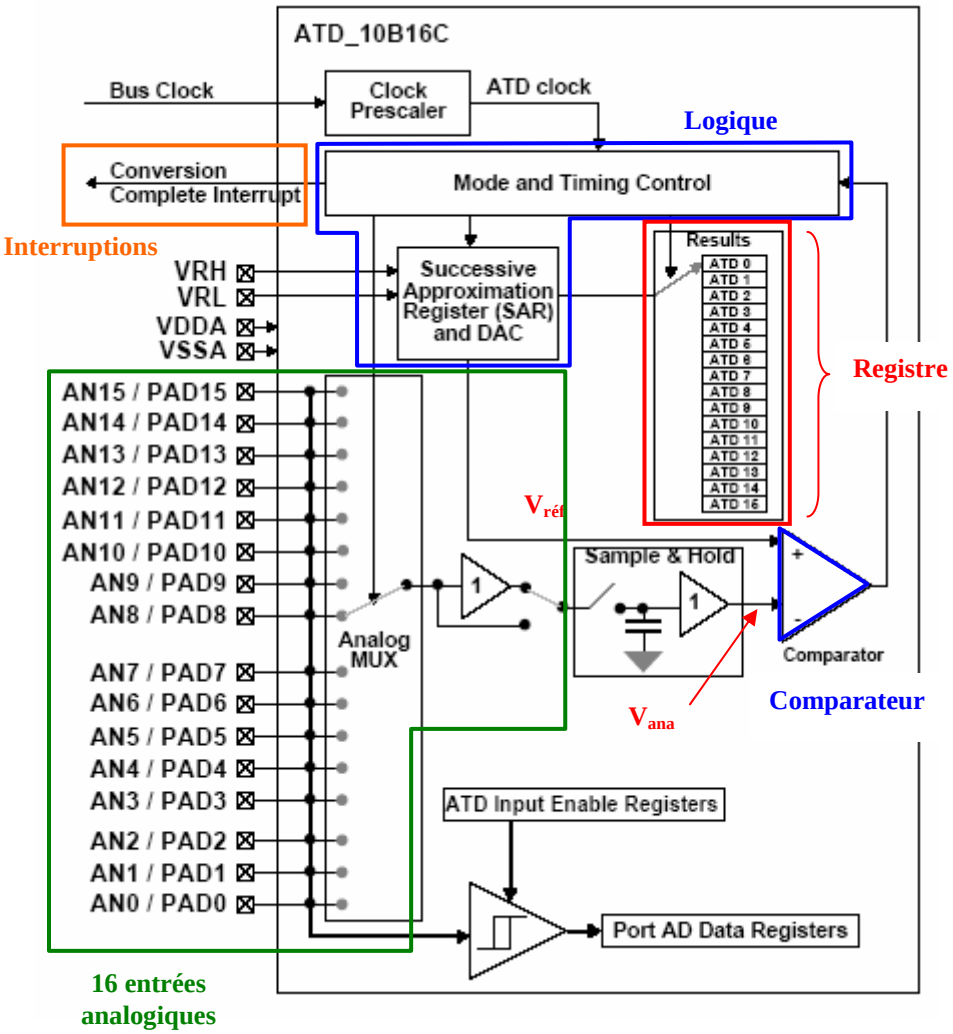


6.1.2/ Le CAD 16 bits du HC12

Le schéma-bloc du convertisseur interne du microcontrôleur fait apparaître, en plus du schéma-bloc du paragraphe précédent, la possibilité de traiter 16 entrées analogiques, la génération d'interruptions et une division programmable de la fréquence du bus pour contrôler le temps de conversion.

Les 16 entrées sont traitées, via le multiplexeur, les unes après les autres.

Les entrées supplémentaires, VRH et VRL, permettent de définir extérieurement les valeurs analogiques, haute et basse, qui correspondent aux valeurs numériques haute et basse. En général, VRH=+5V et VRB=0V.
VDDA et VSSA sont des entrées d'alimentations séparées pour le dispositif de conversion, ce qui permet d'effectuer un filtrage pour éviter les parasites sur la conversion. En général, VDDA est relié au +5V et VSSA au 0V. Une sortie DAO non représentée sur le schéma bloc est ressortie sur le boîtier est représente la valeur moitié entre VDDA et VSSA.



6.1.3/ Registres d'E/S sur le port AD

IENx= « 0 » : entrée analogique.
IENx= « 1 » : E/S logique.

ATDDIEN0

$_0C	BIT	7	6	5	4	3	2	1	0
	R W	IEN15	IEN14	IEN13	IEN12	IEN11	IEN10	IEN9	IEN8
	RESET:	0	0	0	0	0	0	0	0

ATDDIEN1

$_0D	BIT	7	6	5	4	3	2	1	0
	R W	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IEN0
	RESET:	0	0	0	0	0	0	0	0

6.2/ Séquences et conversions

Définitions :

- L'échantillonnage est l'opération qui consiste à mémoriser dans le microcontrôleur, le signal analogique provenant d'une voie (entrée analogique).
- Une « conversion » correspond à la transformation du signal analogique mémorisé dans le microcontrôleur, en un mot numérique mémorisé dans le registre de résultat.
- Une « séquence de conversions » est une suite de conversions de la même voie ou d'un groupe de voies.

6.2.1/ Résolution, temps de conversion

Registre ATDCTL4

ATDCTL4

$_04	BIT	7	6	5	4	3	2	1	0
	R W	SRES8	SMP1	SMP0	PRS4	PRS3	PRS2	PRS1	PRS0
	RESET:	0	0	0	0	0	1	0	1

La résolution d'une conversion sur le microcontrôleur peut prendre par programmation du bit « SRES8 » deux valeurs, 8 bits (SRES8 = « 1 ») et 10 bits (SRES8 = « 0 »).
Une résolution 8bits (non signé) correspond à 256 valeurs, d'où une résolution analogique de : $\frac{+5V}{256} = 19,5mV$ avec VRH=+5V et VRL=0V. De même, une résolution 16 bits correspond à une résolution analogique de 4,9mV.
La résolution 8 bits est en général très suffisante. La résolution 10 bits nécessite des précautions de mises en œuvre pour être sûr que la précision peut être atteinte, par exemple filtrer l'alimentation pour éviter de voir une même valeur analogique convertie en un ensemble de mots numériques.

Le temps d'échantillonnage est configurable par les deux bits SMP0 et SMP1, et dure entre 2 à 16 périodes de la fréquence de conversion.

SMP1	SMP0	Length of 2nd phase of sample time
0	0	2 A/D conversion clock periods
0	1	4 A/D conversion clock periods
1	0	8 A/D conversion clock periods
1	1	16 A/D conversion clock periods

La fréquence des conversions dépend de la configuration de cinq bits, PRS0 à PRS4. Le tableau suivant donne le temps que dure une conversion 8 bits et 10 bits.

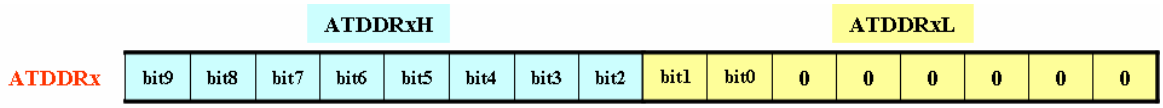
PRS4	PRS3	PRS2	PRS1	PRS0	Division fréq. bus	T conv. 8bits	T conv. 10 bits
0	0	0	0	0	2	65,5µs	262µs
0	0	0	0	1	4	131µs	524µs
0	0	0	1	0	6	196,5µs	786µs
-	-	-	-	-	-	-	-
1	1	1	1	0	62	2,03ms	8,12ms
1	1	1	1	1	64	2,1ms	8,4ms

6.2.2/ Formats du résultat, conversions multiples, continues

A) Les résultats

Les résultats, ou le résultat, de la conversion sont rangés dans des registres 16 bits. Il y a 16 voies analogiques, donc 16 registres 16 bits, de ATDDR0 à ATDDR15. Chaque registre 16 bits est la concaténation de 2 registres de 8bits, par exemple ATDDRx correspond à ATDDRxH concaténé avec ATDDRxL.

Le mot numérique de 8 ou 16 bits, est rangé dans le registre 16 bits, justifié à droite ou à gauche, suivant la valeur du bit DJM (« 0 » justifié à gauche) dans ATDCTL5.



Résultat justifié à gauche

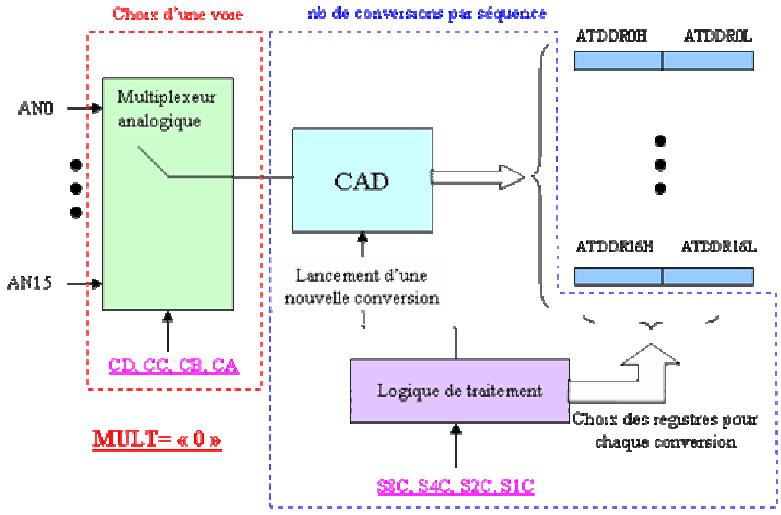
Résultat justifié à droite

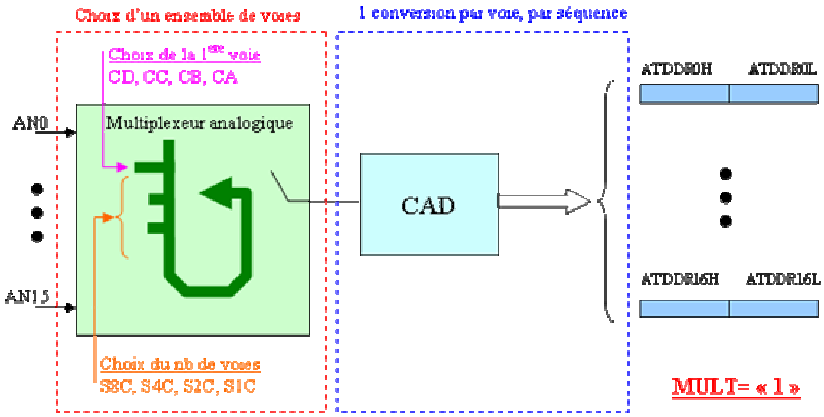
B/ La conversion simple ou multiples voies

Dans une séquence on peut, soit effectuer plusieurs conversions pour une même entrée pour par exemple en faire une moyenne, soit réaliser la conversion pour des voies différentes.

Lorsque le bit MULT du registre ATDCTL5 est à « 0 » alors la séquence effectue pour une même voie, un nombre de conversions défini par les bits S8C, S4C, S2C et S1C (de 1 à 16 conversions) dans ATDCTL3, dont le numéro est défini par les bits CD, CC, CB et CA (AN0 à AN16) dans ATDCTL5. Pour cette voie, à chaque conversion, un nouveau registre résultat est rempli.

Lorsque le bit MULT est à « 1 » alors la séquence effectue une conversion à partir de la voie, dont le numéro est défini par les bits CD, CC, CB et CA (ATDCTL5) et dont le nombre est défini par les bits S8C à S1C (ATDCTL3).
Les résultats sont rangés à partir du registre résultat n°0, correspondant à la voie de la première conversion dans la séquence et non pas forcément à AN0.





CD	CC	CB	CA	Voie analogique
0	0	0	0	AN0
0	0	0	1	AN1
-	-	-	-	-
1	1	1	1	AN15
S8C	S4C	S2C	S1C	nb conversions
0	0	0	0	16
0	0	0	1	1
-	-	-	-	-
1	1	1	1	15

C/ Conversion simple ou continue

Lors d'une séquence, un certain nombre de conversions sont réalisées. Une fois la séquence terminée, soit le processus s'arrête, soit une séquence est automatiquement relancée et les mesures s'effectuent en continu.
Le bit SCAN configuré à « 0 » arrête le processus à la fin de chaque séquence. Une nouvelle séquence redémarre à chaque écriture dans le registre ATDCTL5.

D/ Les registres ATDCTL2, ATDCTL3 et ATDCTL5

Le convertisseur peut être désactivé, pour économiser l'énergie, en configurant le bit ADPU dans ATDCTL2 (activé si ADPU= « 1 »).

ATDCTL2_ADPU=1;

ATDCTL2

BIT	7	6	5	4	3	2	1	0
R W	ADPU	AFFC	AWAI	ETRIGLE	ETRIGP	ETRIGE	ASCIE	ASCIF
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Les 4 bits s1c à s8c définissent le nombre de conversions par séquence (1 à 16).
Une séquence de mesure correspond à la mesure successive des entrées sélectionnées.

ATDCTL3

BIT	7	6	5	4	3	2	1	0
R W	0	S8C	S4C	S2C	S1C	FIFO	FRZ1	FRZ0
RESET:	0	0	1	0	0	0	0	0

djm = 1 → résultat justifié à droite djm = 0 → résultat justifié à gauche
scan =1 → mesures en continu scan =0 → mesure unique
mult =1 → mesure sur plusieurs entrées mult = 0 → mesure une seule entrée
cd,cc,cb,ca → choix de l'entrée à mesurer (AN0 à AN15)

ATDCTL5

BIT	7	6	5	4	3	2	1	0
R W	DJM	DSGN	SCAN	MULT	CD	CC	CB	CA
RESET:	0	0	0	0	0	0	0	0

Remarque : à chaque écriture dans ATDCTL5, une nouvelle séquence de mesures est lancée.

E/ Code d'initialisation du CAN

```
void PeriphInit(void)
{
  ATDCTL2_ADPU=1;        //activation du convertisseur A/D
  ATDCTL3_S2C=1;         //2 conversions par séquence de mesure
  ATDCTL4_SRES8=0;       //Résolution 10 bits
  ATDCTL5=0b10110001;    //mesures multiples, continu, justifié à droite
  ATDCT5_CB=1 ;          //3ème voie mesurée : AN2
}
```

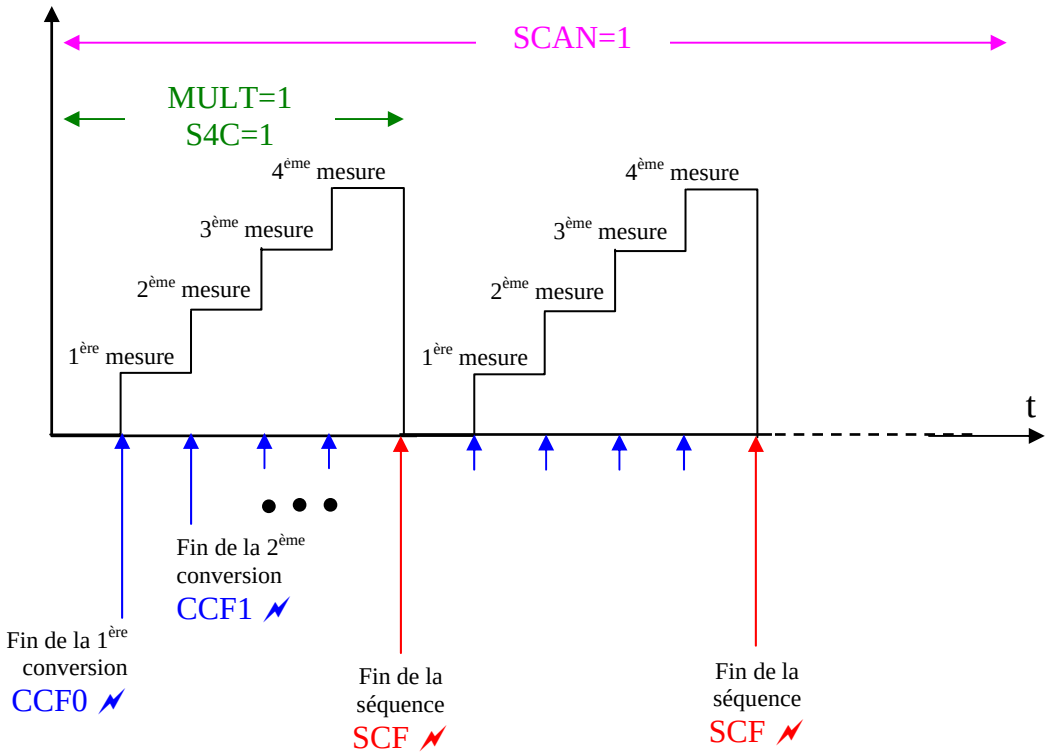
6.3/ Indicateurs de fin de séquence et de conversion

Comme tous les dispositifs qui fonctionnent en parallèle du programme utilisateur, l'ATD est doté d'un certain nombre de drapeaux qui vont permettre de gérer au mieux les résultats.
Un compteur de conversions renseigne l'utilisateur sur l'emplacement où sera rangée la conversion en cours.

6.3.1/ Drapeaux

Deux types de drapeaux :

- SCF dans ATDSTAT0, qui se lève dés que la séquence de mesures est terminée, il est réinitialisé en écrivant SCF= « 1 » dans ATDCTL5 ou en écrivant dans ATDCTL5 (lancement d'une nouvelle séquence de conversions).
- CCFx dans ATDSTAT1, levé à chaque fin de conversion correspondante. Le numéro du drapeau est aussi le numéro de la position de la conversion finie dans la séquence. CCFx est réinitialisé en écrivant dans ATDCTL5 (lancement d'une nouvelle séquence de conversions).



6.3.2/ Compteur

Un compteur formé par 4 bits, CC3 à CC0, dans le registre ATDSTAT0 donne le numéro du registre de résultat qui reçoit la conversion en cours d'exécution.
Par exemple si CC3=0, CC2=1, CC1=1, CC0=0 alors le résultat de la conversion en cours sera dans le registre 6. Le compteur est remis à zéro en début de chaque nouvelle séquence.

6.3.3/ Les registres

A/ Compteur et drapeau de fin de séquence

ATDSTAT0

BIT	7	6	5	4	3	2	1	0
R	SCF	0	ETORF	FIFOR	CC3	CC2	CC1	CC0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

B/ Drapeaux de fin de conversion

ATDSTAT1

BIT	7	6	5	4	3	2	1	0
R	CCF7	CCF6	CCF5	CCF4	CCF3	CCF2	CCF1	CCF0
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

ATDSTAT2

BIT	7	6	5	4	3	2	1	0
R	CCF15	CCF14	CCF13	CCF12	CCF11	CCF10	CCF9	CCF8
W								
RESET:	0	0	0	0	0	0	0	0

= Unimplemented or Reserved

Chapitre 7

Liaison numérique série asynchrone - SCI

Un fil de communication entre le microcontrôleur et l'extérieur n'avait pas jusqu'alors un rôle de « transmission d'informations série ». En effet, il était utilisé soit seul, pour transmettre une information ponctuelle, soit au sein d'un groupe pour transmettre une information parallèle, par exemple un octet. Pourtant, un simple fil peut transmettre ce même octet, bit après bit. L'avantage d'une telle transmission est la réduction du nombre de fils, par contre la vitesse de transmission décroît forcément. Le 68HC12 possède deux types de transmission série : synchrone (SPI) pour des liaisons courtes, ou asynchrone (SCI) qui n'utilise que 3 fils au minimum en « full duplex ».

Objectifs :

- Connaître les notions de base de la transmission série asynchrone.
- Connaître la différence entre transmissions « unidirectionnel », « half duplex » et « full duplex ».
- Savoir expliquer le rôle des bits de « start », de « stop » et de « parité ».
- Savoir modifier la vitesse de transmission et conjointement déterminer la durée d'une trame.
- Savoir configurer une des liaisons série du microcontrôleur 68HC12.

Vocabulaire :

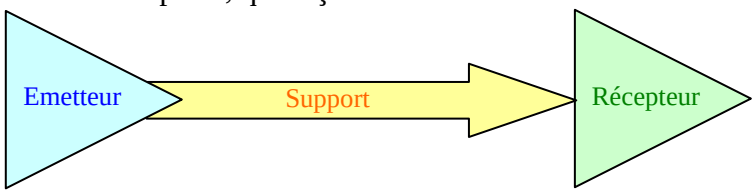
Emetteur - récepteur
Communication série – transmission série
Communication bidirectionnelle
Half duplex – full duplex
Trame
Codage ASCII
Transmission série synchrone et asynchrone
Vitesse de transmission – flux
Bit de start – bit de stop - parité

7.1/ Principe de la transmission série

7.1.1/ Notions de base de la transmission série

Une transmission série suppose de disposer d'au moins :

- un émetteur, qui envoie l'information.
- un support, qui conduit le signal.
- un récepteur, qui reçoit l'information.



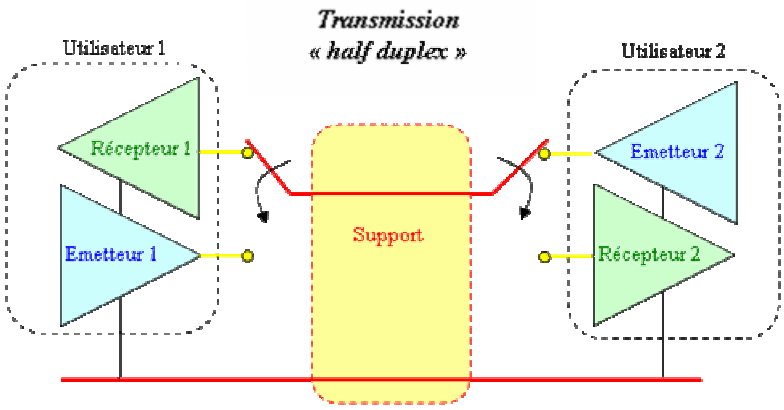
Une transmission comme celle représentée ci-dessus est dite « unidirectionnel » car le sens de la transmission est déterminé sans ambiguïté : l'émetteur envoie, le récepteur reçoit. Ce type de transmission se retrouve lorsqu'un ordinateur communique avec une imprimante.

Couramment les transmissions sont aujourd'hui « bidirectionnelles » : l'émetteur est aussi récepteur et vice-versa.

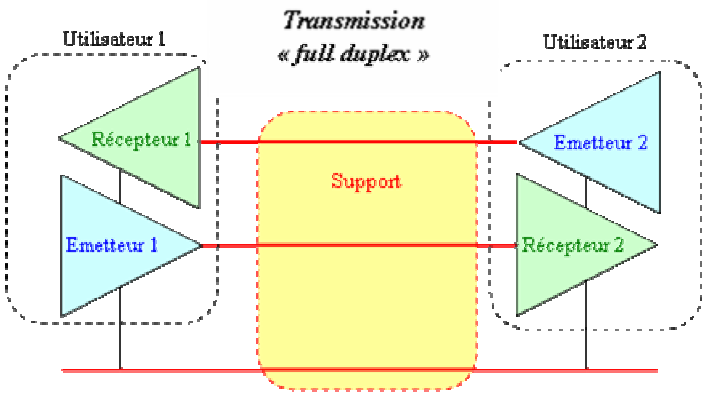
La réalisation d'une liaison bidirectionnelle demande que :

- Le support soit bidirectionnel. Dans notre cas le support est matérialisé par des fils électriques, qui ne posent pas de problème particulier.
- L'émetteur soit aussi récepteur. Deux cas sont possibles, soit la transmission a lieu tantôt dans un sens et tantôt dans l'autre sens, soit l'émission et la réception sont conjointes.

Dans le cas où la transmission se fait dans un sens puis dans l'autre, elle est « half duplex ». On retrouve la transmission supportée par des talkies-walkies, ou chacun des utilisateurs doit parler en appuyant sur un bouton (position émettrice) et terminer sa conversation par le mot « terminé » en relâchant ensuite le bouton (position réceptrice). Ce type de transmission est pas essence lente et peu propice à une interactivité importante. Toutefois, son avantage réside dans la simplicité du support qui peut se limiter dans notre cas à deux fils, le fil du signal et la masse.



Une transmission « full duplex » nécessite une voie d'émission et une voie de réception, qui dans notre cas se résume à trois fils, le fil du signal aller, le fil du signal retour et le fil de masse.



Les trois fils portent couramment les noms suivants :

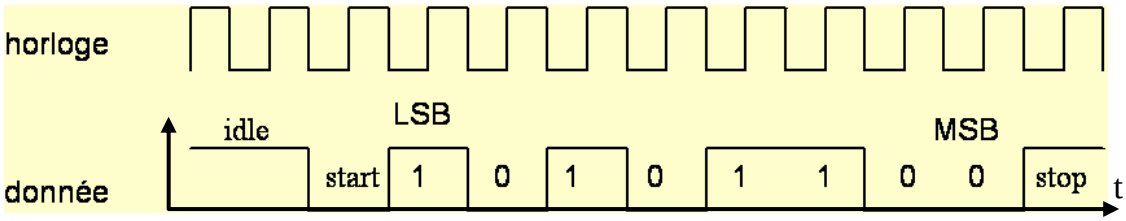
- TxD (Transmit Data) qui est la sortie de transmission.
- RxD (Receive Data) qui est l'entrée de transmission.
- GND (Ground) qui est la référence de masse.

Remarque : l'émetteur 1 est connecté au récepteur 2 et vice versa. Il convient donc de croiser les fils pour avoir une liaison entre un émetteur et un récepteur. Les câbles existent déjà croisés sous l'appellation « câble nul modem ».

7.1.2/ Transmission d'une TRAME normalisée

Sur la figure ci-dessous, sont représentés, un signal d'horloge pour cadencer le temps, un signal de donnée correspondant au transport d'un octet.

L'octet transmis a le code hexadécimal 35h ou le code binaire 00110101, qui est aussi le code ASCII du chiffre 5.



La durée d'un bit est celle de la période d'horloge de synchronisation.

Cette horloge est transmise via un fil supplémentaire lorsque la liaison série est dite « synchrone » (SPI).

Dans le cas d'une transmission asynchrone (SCI), notre cas d'étude, le signal d'horloge n'est pas fourni ; l'émetteur et le récepteur conviennent donc d'utiliser chacun de son côté un signal d'horloge non synchronisée, mais de même fréquence à 5% prés.

A/ Bit de START

Les horloges de l'émetteur et du récepteur, de fréquences proches, ne sont pas synchronisées. Un bit, appelé « bit de start » placé en début du signal de donnée, a alors pour rôle de donner le top de départ de la fréquence du récepteur, et dure une période de la fréquence de l'émetteur.

B/ L'octet de donnée

A la suite du *bit de start*, est envoyé le signal de donnée sous la forme d'un octet. Le premier bit envoyé est celui de poids faible (Low Significant Bit), pour finir la donnée avec le bit de poids fort (Most Significant Bit).

C/ Bit de STOP

La fin du message se termine par un bit, dit « bit de stop », voire parfois deux *bits de stop* lorsque le récepteur est très lent, et lui permettre ainsi de « digérer » le message avant l'envoi du prochain.

D/ Bit de PARITE

Si la transmission est bruitée, le signal reçu peut comporter des erreurs. Pour vérifier que le contenu du message ne comporte pas d'erreur, un bit supplémentaire est ajouté, dit « bit de parité ».

Dans le cas d'une parité paire, ce bit prend une valeur a « 0 » si le nombre de « 1 » de l'octet est pair et « 1 » dans le cas contraire. On inverse les valeurs lors d'une transmission avec une parité impaire.

Après contrôle de la parité, le récepteur peut demander à l'émetteur de lui renvoyer le dernier message.

E/ La TRAME

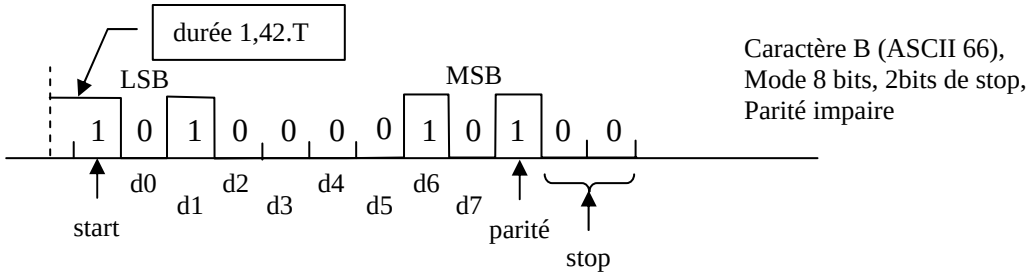
La trame représente donc le signal émis qui comporte :

- Un bit de start.
- L'octet de donnée avec parfois un bit de parité.
- Un ou deux bits de stop.

Une fois la trame transmise, la ligne passe à l'état libre ou « IDLE ».

Historiquement, la liaison série transmettait principalement des codes ASCII qui représentaient les symboles des machines à écrire ou des télétypes.

Le codage ASCII initial ne comportait que 128 caractères correspondant à 7 bits. L'octet de donnée d'une trame laissait alors un bit libre, le 8ème bit, qui servait alors de bit de parité. Le codage ASCII est aujourd'hui dit « étendu », et comporte jusqu'à 256 caractères.



F/ Vitesse de transmission

La vitesse de transmission s'exprime en BAUD, ou bits par seconde.

Une trame dure environ 10 périodes d'horloge, un bit de start, un octet de donnée, un bit de stop. Une transmission à 9600 BAUDS conduit aussi à une vitesse de transmission de 960 octets par seconde.

Les vitesses de transmission sont normalisées, ce qui permet de limiter le nombre d'horloges à reconstituer par le récepteur. Les valeurs normalisées des vitesses de transmission sont :

75, 150, 300, 600, 1200, 4800, 9600, 19200, 38400 bauds...

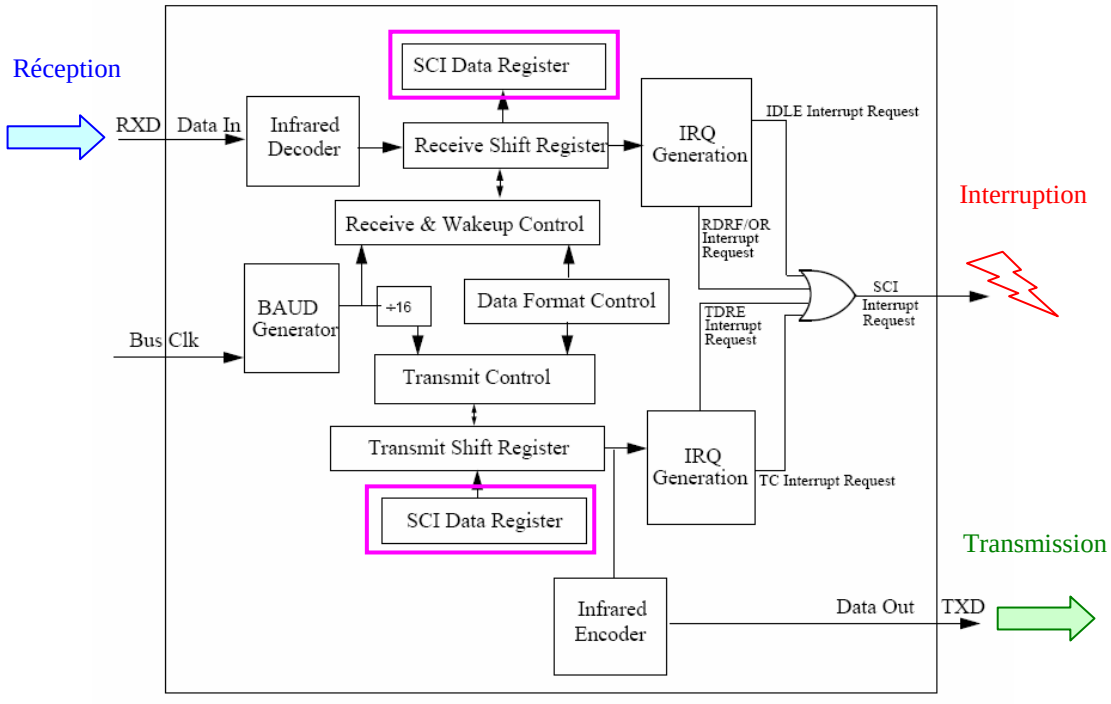
7.2/ La SCI du microcontrôleur

Le dispositif de communication série asynchrone (Serial Communication Interface) fonctionne en parallèle du cœur du microcontrôleur, tout comme les dispositifs précédents du TIMER et de l'ATD.

On s'attend donc à avoir un dispositif d'interruption capable de l'avertir d'un évènement sur son entrée SCI.

Le 68HC9S12e128 possède trois liaisons série indépendantes : SCI0, SCI1, SCI2.

7.2.1/ Principes



Le schéma fonctionnel fait apparaître :

- Un dispositif de réception, qui une fois la donnée extraite, la transfert dans le registre « SCI Data Register », et active une interruption.

– Un dispositif de transmission, qui transfert la donnée à envoyer, du même registre « SCI Data Register », vers le registre intermédiaire de transmission, et active une interruption.

Une fois le dispositif initialisé (vitesse, parité, stop...), l'utilisateur ne fait qu'écrire la donnée à transmettre dans le registre « SCI Data Register » ou « SCIDR », car l'empaquetage de celle-ci, qui conduit à la trame réellement transmise, est automatiquement réalisée par le dispositif SCI du microcontrôleur.

De même, l'utilisateur ne fait que lire le registre « SCI Data Register » lorsqu'une donnée est reçue. L'opération d'extraction de la donnée de la trame est automatiquement réalisée par le microcontrôleur.

Remarque : Les registres « Receive Shift Register » et « Transmit Shift Register » sont des registres à décalage qui comportent l'octet de donnée et tous les bits d'accompagnement de la trame.

7.2.2/ Format de transmission

Le format de la trame du 68HC12 peut contenir un bit de start, une donnée de 8 ou 9 bits, un bit de parité et un ou deux bits de stop.

Le format est défini à travers le registre de contrôle « SCISCR1 »

A/ Longueur de la donnée

♦ M définit la longueur de la donnée :

– M= « 0 » ⇒ 1bit de start, un octet de donnée, 1 bit de stop.
– M= « 1 » ⇒ 1bit de start, 9 bits de donnée, 1 bit de stop.

B/ Parité

♦ PE, PT définissent la parité. Un bit de parité est inséré comme bit de MSB si PE= « 1 ».

– PT= « 0 » ⇒ parité paire
– PT= « 1 » ⇒ parité impaire

Si la parité est paire (PT= « 0 »), alors un nombre de « 1 » paire place le bit de parité à « 0 » et un nombre impair, force le bit à « 1 ». Si la parité est impaire alors le bit de parité est égal à « 0 » si le nombre de « 1 » est impair.

SCICR1	7	6	5	4	3	2	1	0
Read Write	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
RESET:	0	0	0	0	0	0	0	0

7.2.3/ Vitesse de transmission

Le registre « SCIBD » se charge de définir la vitesse de transmission à travers la valeur des bits de SBR0 à SBR12.

La vitesse de transmission est calculée comme suit :

$$\text{Vitesse} = \frac{\text{SCI BUS Clock}}{(16 \times \text{mot } SBR)}$$

où, « mot *SBR* » est la valeur décimale des bits SBR0 à SBR12.

(Pour SCI BUS Clock=8MHz)

SBR [12-0]	Baud Rate réelle	Baud Rate théorique	Erreur
13	38461,5	38400	0,16%
26	19230,8	19200	1.6%
52	9615,4	9600	0,16%
104	4807,7	4800	0,16%
208	2403,8	2400	0,16%
417	1199,0	1200	0,08%
833	600,24	600	0,04%
1667	299,94	300	0,31%
3333	150,01	150	0,007%

Mot *SBR*

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SCIBD	IREN	TNP1	TNP0	SBR12	SBR11	SBR10	SBR9	SBR8	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

7.2.3/ Activation de la transmission SCI

Les bits « TE » et « RE » du registre « SCICR2 » activent les fonctions d'émission et de réception de la liaison série.

- Pour TE= « 1 » l'émetteur est prés à transmettre dès qu'une écriture est réalisée dans le registre de donnée « SCI Data Register ».
- Pour RE= « 1 » le récepteur est activé, et dés qu'une trame est reçu, fournit la donnée dans le registre « SCI Data Register ».

Remarque : TE et RE doivent être à « 1 » pour une liaison Full Duplex. Pour une liaison unidirectionnelle, seul un des bits est actif.

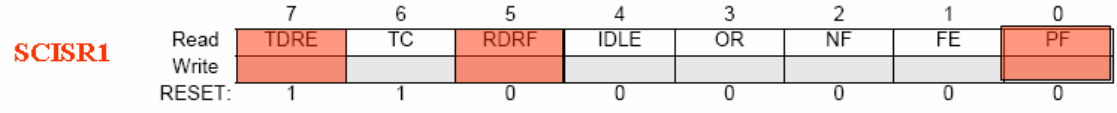
	7	6	5	4	3	2	1	0
SCICR2 Read	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Write								
RESET:	0	0	0	0	0	0	0	0

7.2.4/ Drapeaux d'état de la transmission

Deux bits « TDRE » (Transmission data register empty) et « RDRF » (receive register data full) indique l'état des registres de transmission et de réception. Un troisième bit, « PF », calcul la correspondance de parité.

- Si TDRE = « 1 » alors le registre de transmission est vide et est prés à recevoir une nouvelle donnée à transmettre. Ce bit est remis à « 0 » par lecture du registre « SCISR1 », suivi d'une écriture dans le registre de donnée « SCIDR ». L'écriture conduit à la transmission d'une donnée.

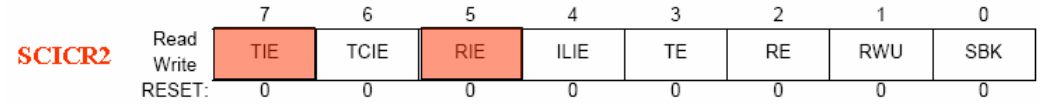
- Si RDRF = « 1 » alors le registre de réception est plein, une donnée vient d'être reçue. Ce bit est remis à « 0 » par écriture de « 80h » dans le registre SCISR1, suivi par une lecture dans le registre SCIDR. La lecture correspond à acquérir la donnée envoyée.
- Si PF= « 1 » alors il y a une erreur dans la parité du mot reçu. Ce bit est remis à « 0 » par lecture du registre « SCISR1 », suivi d'une écriture dans le registre de donnée « SCIDR ». L'écriture conduit à la transmission d'une donnée.



7.2.4/ Interruptions

Quatre bits dans le registre de contrôle « SCICR2 » sont des masques locaux « TIE », « TCIE », « RIE » et « ILIE », qui activent une interruption commune « SCI0CR2_(TIE,RIE) ». Le numéro du vecteur d'interruption associé à SCI0 est le 20. Les deux sources d'interruption les plus utilisées sont :

- TIE= « 1 », conduit à une interruption dés que le drapeau TDRE se lève (registre de transmission vide). Ce qui permet l'envoie de la donnée suivante sans perte de temps.
- RIE= « 1 », conduit à une interruption dés que le drapeau RDRF se lève (registre de réception plein). Ce qui permet de lire la donnée reçue sans la perdre. En effet, si une autre donnée arrive elle écrasera le contenu de SCIDR, le registre de donnée.



7.2.5/ Initialisation

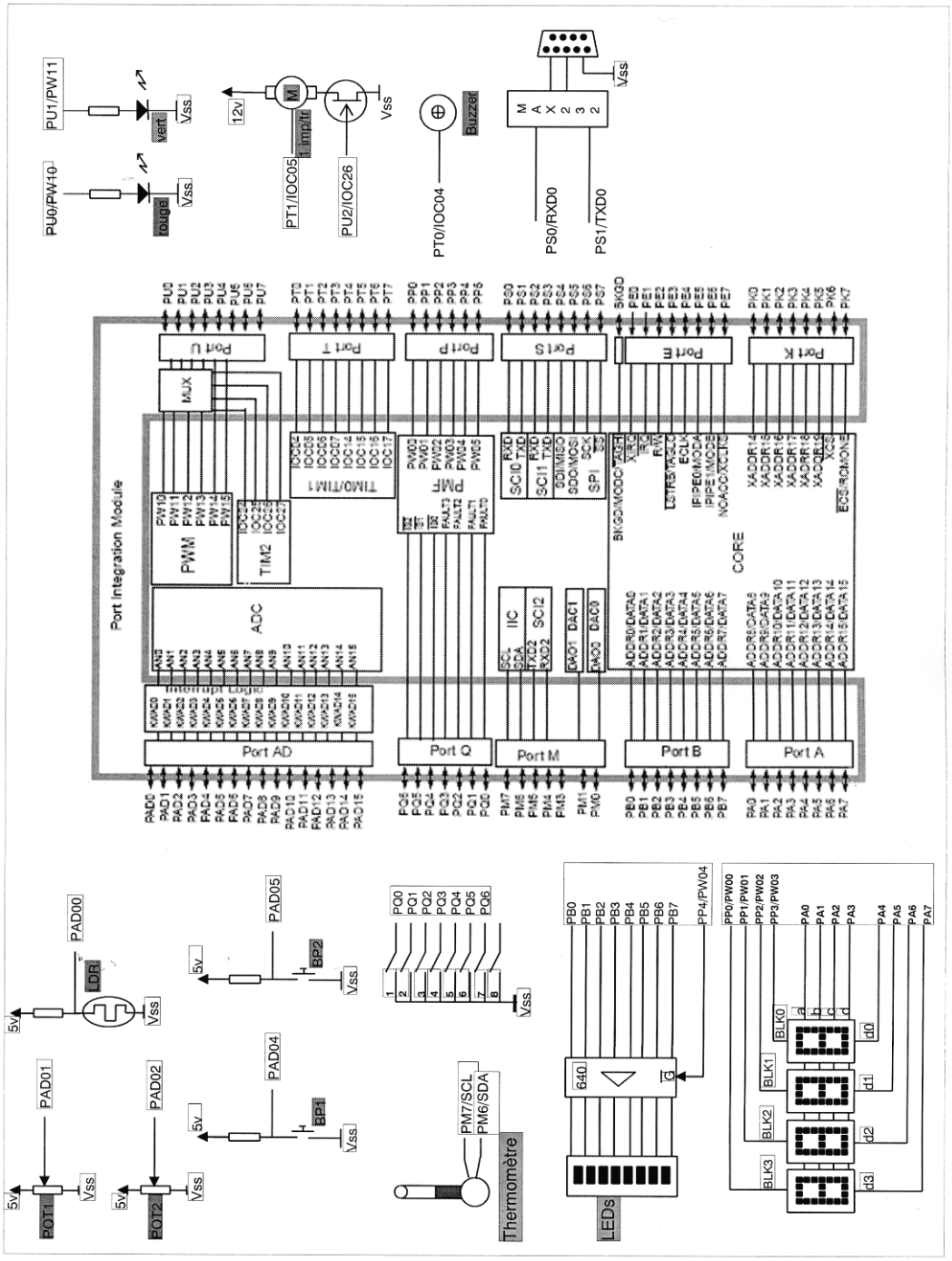
```
//initialisation de la liaison série SCI0
  SCI0BD=52;  //vitesse 9600 bds
  SCI0CR1_M=0;  //mode 8 bits
  SCI0CR2 |=te+re+rie;       //active emission et reception
```

ANNEXES

- Schémas électriques de la maquette ENIT
- Quelques éléments d'algorithmique
- Codage en C
- Directives de compilation usuelles et mots clés
- Le standard de communication série, RS232
- Utilisation de l'Hyper Terminal
- Codes ASCII
- Compléments CodeWarrior
- Programmes utilitaires

Schémas électriques de la maquette ENIT

Schémas fonctionnels



Schémas électrique

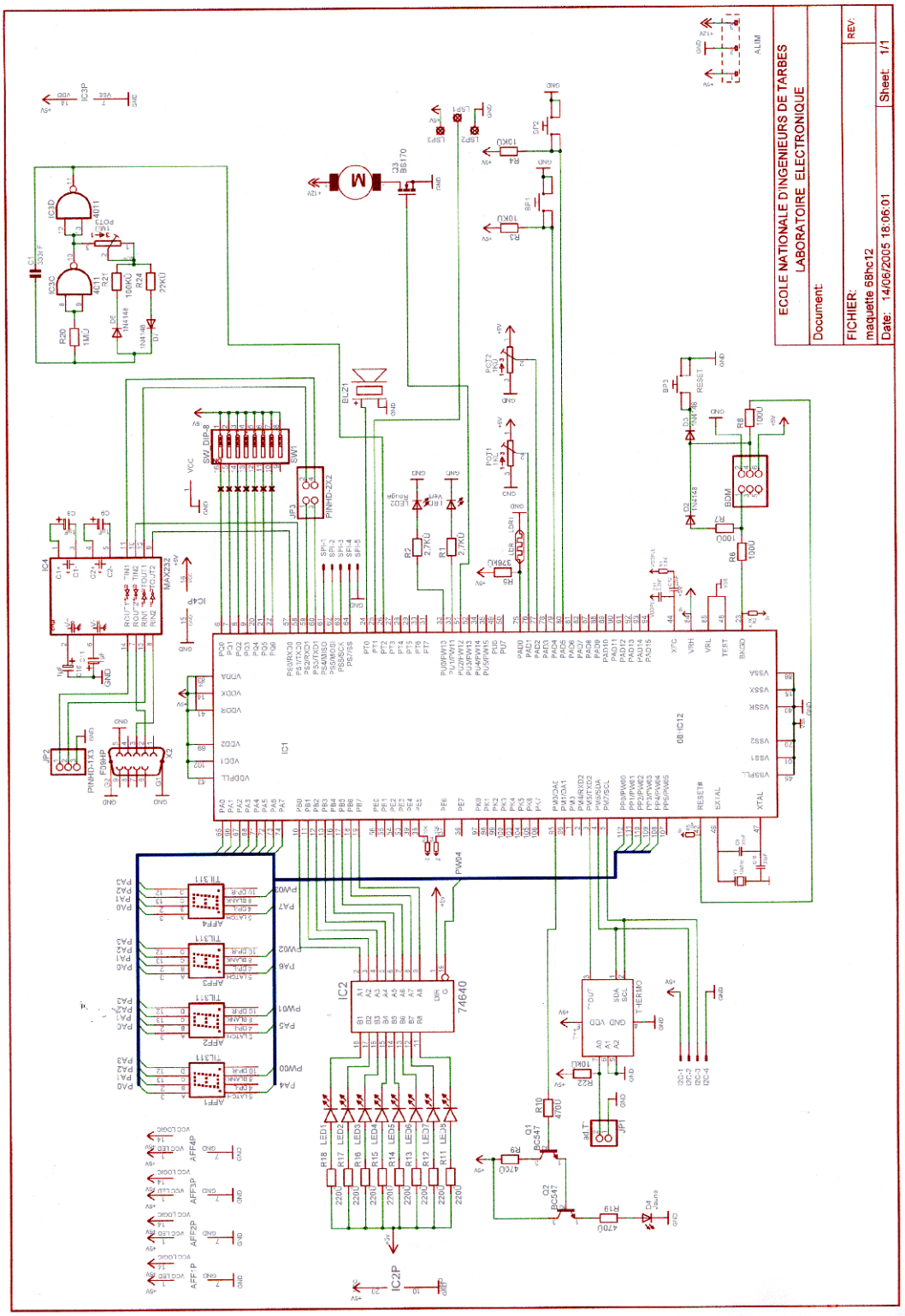
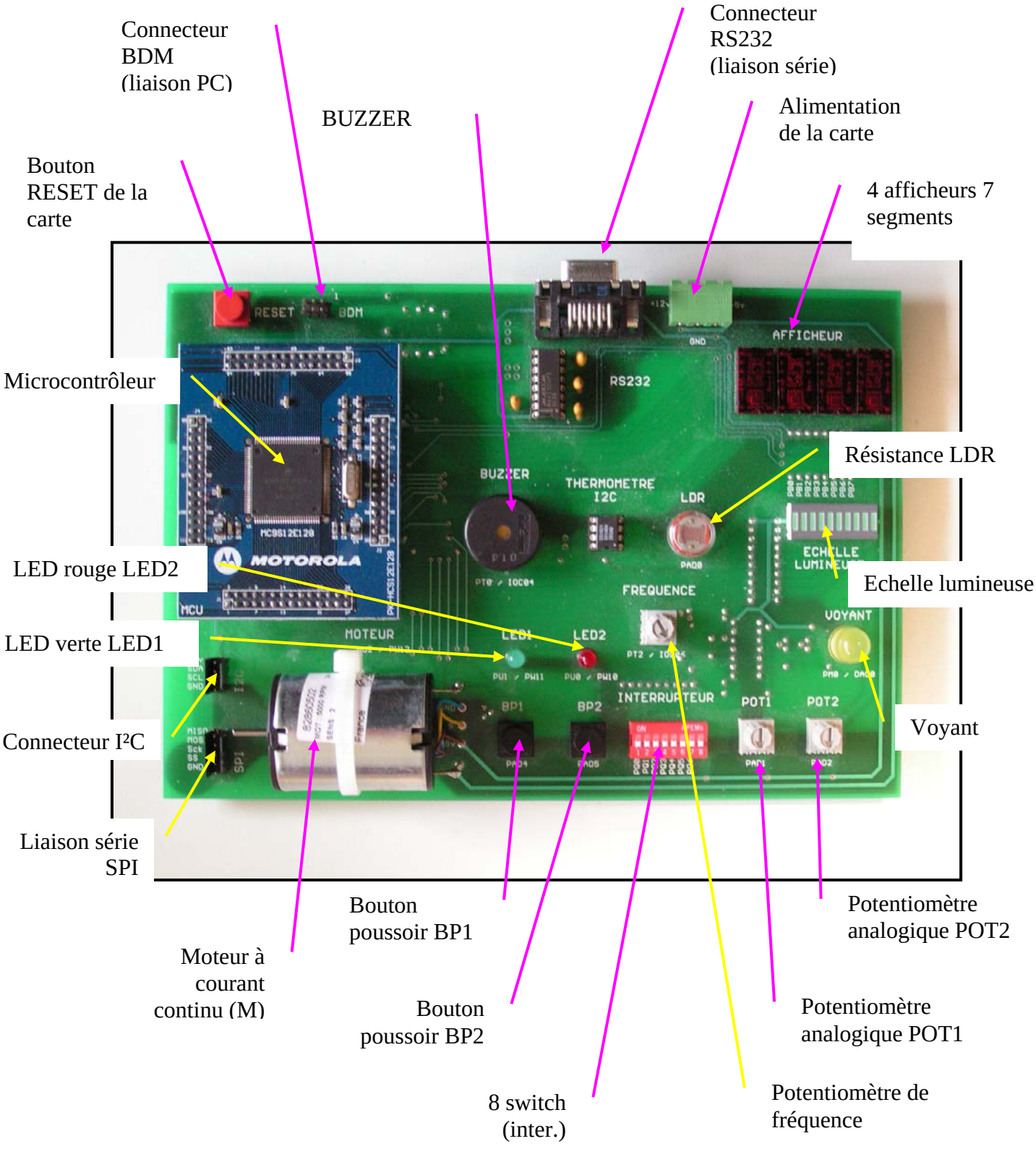
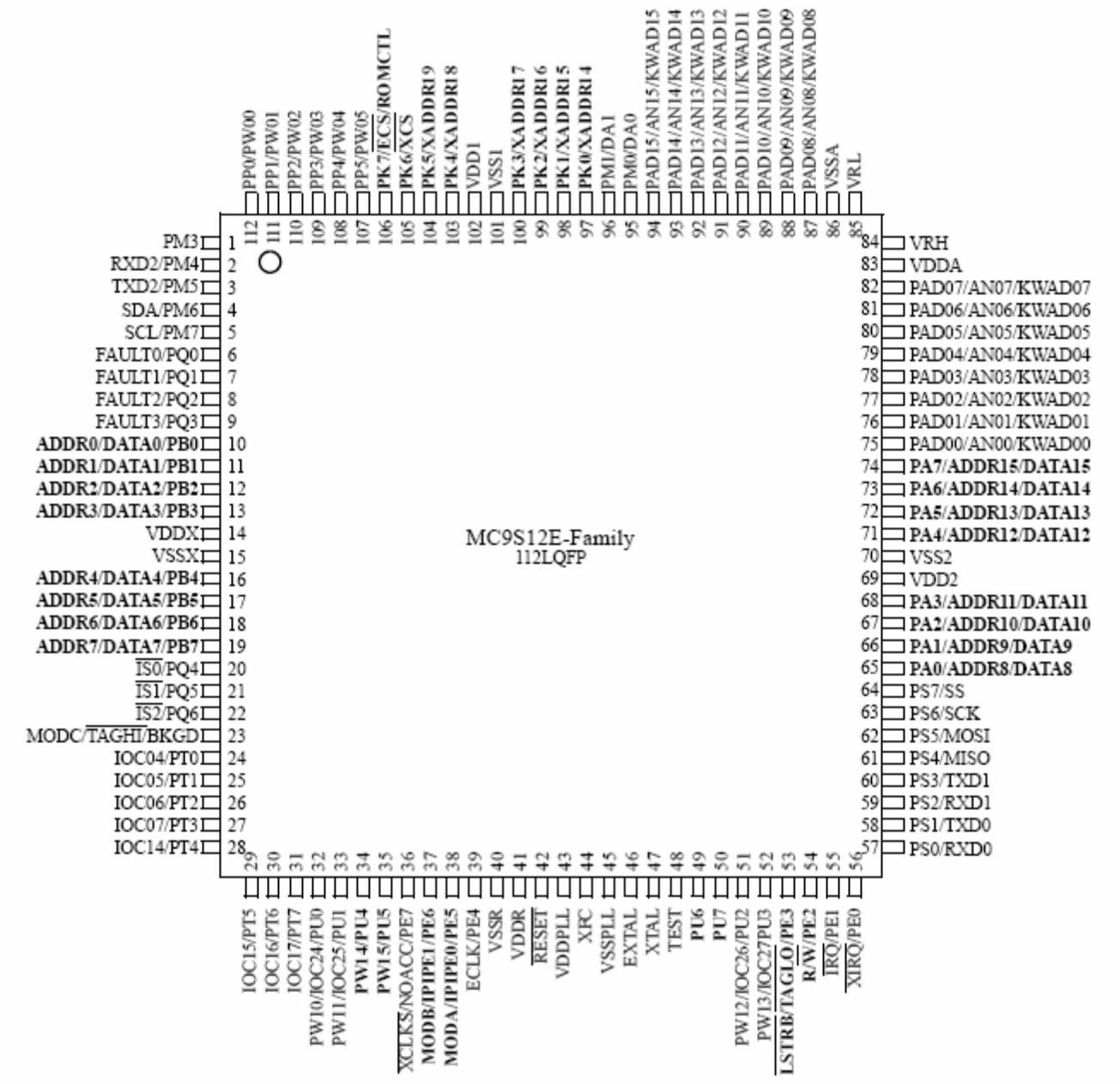


Schéma mécanique



Brochage du composant HC12S9E…



Quelques éléments d'algorithmique

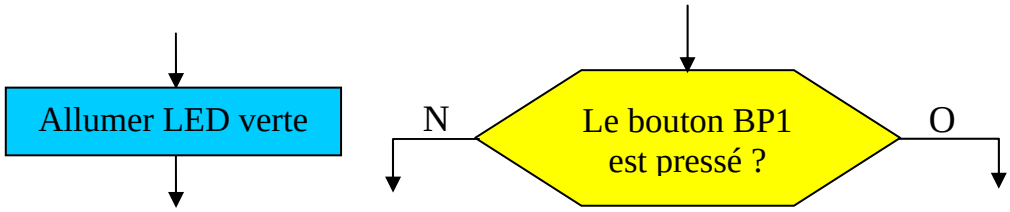
Objectifs

Un algorithme est un arbre d'actions et de choix, qui permet de décrire le fonctionnement séquentiel d'une machine automatique (le mot machine automatique est ici employé dans son sens large). Cet algorithme est un graphique qui utilise des symboles connus de tous.

Symboles

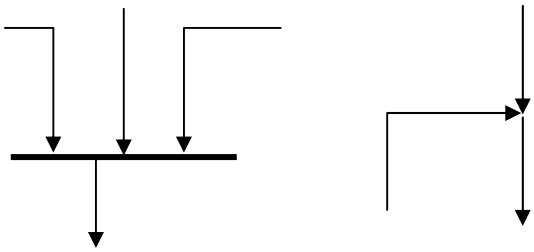
Ces symboles représentent soit une action à réaliser (ex. : « Allumer la LED verte ») soit pose une question dont la réponse (Oui ou Non) entraîne un aiguillage dans le parcourt de l'arbre (ex. : « Le bouton BP1 est pressé ? »). On passe automatiquement d'un symbole à un autre, dès que l'action d'un « symbole action » est terminée, ou dès que la réponse à un « symbole de questionnement » est trouvée.

Ces deux symboles (« action » et « questionnement ») indiquent par des flèches le sens d'arrivée sur le symbole et le sens de départ du symbole.

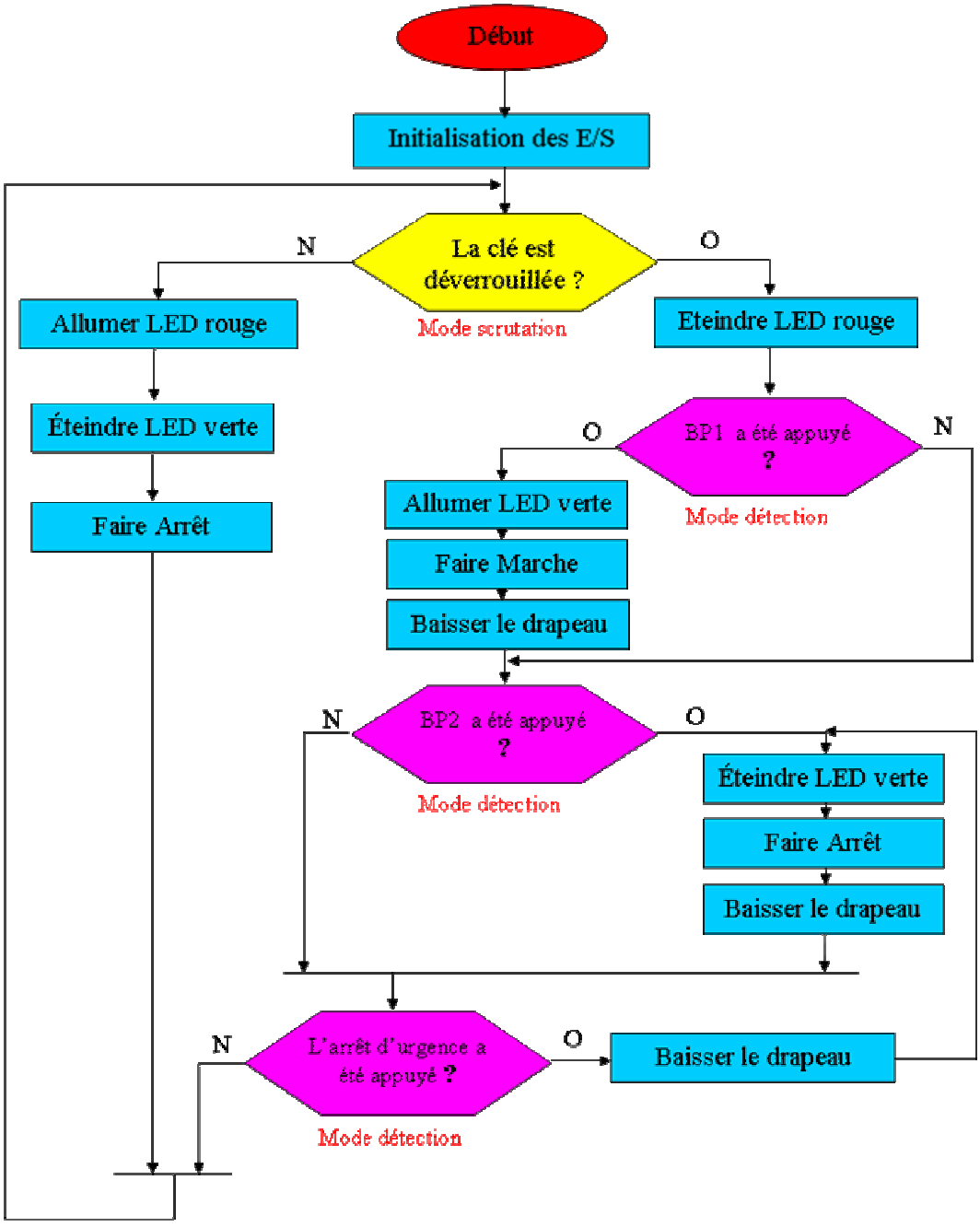


Deux autres symboles nous permettent d'indiquer d'où démarre l'algorithme et quand il se finit.

Il existe aussi des combinaisons de liaisons entre les symboles, qui simplifient en général la lecture de l'algorithme. ATTENTION les sens des flèches doivent être sans ambiguïté.



Exemple


Début
Initialisation des E/S
La clé est déverrouillée ?
N
O
Mode scrutation
Allumer LED rouge
Éteindre LED verte
Faire Arrêt
Eteindre LED rouge
BP1 a été appuyé ?
O
N
Mode détection
Allumer LED verte
Faire Marche
Baisser le drapeau
BP2 a été appuyé ?
N
O
Mode détection
Éteindre LED verte
Faire Arrêt
Baisser le drapeau
L'arrêt d'urgence a été appuyé ?
N
O
Mode détection
Baisser le drapeau

Codage en C

Définitions de variables

(Par défaut les variables sont signées)

Caractère (1 octet) : char x; (-128< x <127)

Entier (2 octets) : int x; ou short int x; (-32768< x <32767)

Entier long (4octets) : long x ; (-2 147 483 648< x <2 147 483 647

Réel (2 octets ou 4 octets) : float x; (±1.0, 0.1037 5, 1e2, chiffres après virgule)

(Pour des nombres non signés)

Caractère (1 octet) : unsigned char x; (0< x <255)

Entier (2 octets): unsigned int x; (0< x <65 535)

Entier long (4octets) : unsigned long x; (0< x < 4 294 967 296)

Opérateurs bit à bit

Symbole	Fonction	Exemple
« | »	OU	PTT |=x;
« & »	ET	x &= 0xF7;
« ^ »	OU excl.	y ^= PTA;
« ~ »	NOT	z = ~ 0b11110000;

Le forçage de bits

Symbole	Fonction	Exemple	Explication
« | »	OU	PTT |=0b00100000;	Forçage à 1 du bit 5 du port T
« & »	ET	x &= 0xF7;	Forçage à 0 du bit 3 de l'opérande x
« ^ »	OU excl.	PTA ^= 0b00000100;	Inversion du bit 2 du port A
« ~ »	NOT	y ~= 0b11110000;	y=0b00001111

Opérations de décallage

Symbole	Fonction	Exemple	Explication
« y >>= x »	x décalages à droite sur y	0b101100 >>= 2 ;	Résultat → 0b001011
« y <<= x »	x décalages à gauche sur y	0x13 <<= 1;	Résultat → 0x26

Opérations d'incrément et décrément

Symbole	Fonction	Exemple	Explication
« y++ »	Incrément de y aprés	I=10; x=I++;	Résultat → x=10 et i=11
« ++y »	x décalages à gauche sur y	I=10; x=++I;	Résultat → x=11 et i=11
« y-- »	Décrément de y aprés	I=10; x=I--;	Résultat → x=10 et i=9
« --y »	Décrément de y avant	I=10; x=--I;	Résultat → x=9 et i=9

Les 4 opérations

Symbole	Fonction	Exemple	Cas particulier
« + »	plus	X=X + y	x=x+y équ x += y
« – »	moins	Z=0x1F – 0x0F	x=x–y équ x –= y
« * »	mult.	y=y * 2	
« / »	div.	x=0b00111111 / 2	

Les boucles ou opérateurs d'itérations

for :

(pour <expr1>; jusqu'à <expr2> vraie; itération)

```
for (i=0; i<10; i++)
        Instruction 1;
for (i=10; i<0; i--) {
        Instruction 1;
        Instruction 2;
}
for (; ; )                      //boucle infinie
        Instruction 1;
```

while :

(Tant que <cond> faire…)

```
while (x>=0) {
        Instruction 1;
        Instruction 2;
    }
while (1) {                     //boucle infinie
        Instruction 1;
    }
```

do while :

(faire .... tant que <condition> est vraie)

```
do {
        Instruction 1;
        Instruction 2;
{ while (x>0);
```

Les opérateurs de comparaison

Inversion condition : « ! »
Egal, différent : « = », « != »
Supérieur, inférieur : « > », « < »
Sup. égal, inf. égal : « >= », « <= »

Les opérateurs logiques:

(Renvoi un « 1 » si la condition est vraie, sinon renvoi un « 0 »)

((cond1) ET (cond2)) : « ((...) && (...)) » (« 1 » si les 2 sont vraies)
((cond1) OU (cond2)) : « ((...) || (...)) » (« 1 » si une des 2 est vraie)
Inversion (cond) : « ! (...) » (« 0 » si cond vraie, sinon « 1 »)

Opérateur conditionnel :

Si ... alors ... sinon ...

```
if (condition)
        Instruction 1;
if (condition) {
        Instruction 1;
        Instruction 2;
    }
if (condition)
        Instruction A;
    else
        Instruction D;
if (condition) {
        bloc instructions;
    }
    else {
        bloc instructions;
    }
```

Si <variable égale à > (valeur1 alors...) (valeur2 alors ...)

```
switch (expression) {
    case <expression1> :
            Instruction 1;
    case <expression2> :
            Instruction 2;
    default :
            Instruction par défaut;
    }
```

?

(Renvoi « valeur si vrai » si la condition est vraie, sinon renvoi « valeur si faux »)

(condition) ? (valeur si vrai) : (valeur si faux) x>0 ? 0x10 : 0x11;

Directives de compilation usuelles et mots clés

Les directives de compilation indique, ou ordonne, au préprocesseur[1] de prendre en compte en considération des informations. Par exemple, au lieu d'utiliser « 0b00000001 » pour indiquer qu'il s'agit d'une opération qui utilisera le bit PAD0, il est plus clair d'écrire le mnémonique « pad0 » à la place du précédent. Pour ceci, il faut indiquer au préprocesseur de remplacer pad0 par 0b00000001 au moment de la compilation, on utilise pour ceci une directive de déclaration #include. Toutes les directive commencent par le caractère réservé « # », et ne se termine jamais par un « ; », qui reste la marque d'une instruction en C.

Directive de déclaration

```
#define   SW1   PTQ_BIT0
```

A chaque fois que le préprocesseur rencontre « SW1 », il doit le remplacer par « PTQ_BIT0 », c'est-à-dire « le bit 0 du port Q ». Après l'instruction undef, la déclaration est annulée.

```
#define   periode   8192
#define   delai   2500
    ……
# undef   delai
```

Directive d'insertion

#include <hidef.h>	introduit des macros complémentaires au langage C.
#include <mc9s12e128.h>	informe le préprocesseur de la version du µC utilisé sur la maquette ENIT

hidef.h et mc9s12e128.h sont des librairies incluses dans le projet dans la section « libraries » de metrowerks.

Mots clés

```
extern   void   init_affichage(void);
```

permet d'introduire dans le programme une procédure déjà écrire. Ceci évite de réécrire cette procédure et de cette façon allège l'écriture et la compréhension du programme. La procédure doit être ajoutée dans la liste des fichiers sources, comme un document « nom_procédure.c ».

[1] Le pré processeur, est un processus associé à la compilation, il intervient juste avant la compilation. Il met en ordre la feuille qui sera à compiler.

Le standard RS232

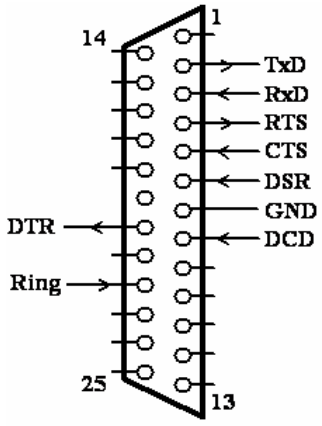
A partir des connaissances apportées par les paragraphes précédents du chapitre, on sait envoyer et recevoir des données à partir d'une des trois liaisons série du microcontrôleur. En résumé, il suffit après avoir réalisé le processus d'initialisation, de lire ou d'écrire dans le registre de donnée SCIDR.

Lorsque le flux de données est important, il est impératif d'utiliser les interruptions pour agir immédiatement et ne pas perdre de données.

Toutefois, la gestion asynchrone par les interruptions peut s'avérer insuffisante pour garantir une réception correcte des données lorsque le flux est important et aléatoire ou lorsque le milieu est bruité. On a alors recours à une procédure interactive entre l'émetteur et le récepteur. Dans la procédure de « la poignée de main », « HANDSHAKE » en anglais, des fils de liaison sont ajoutés entre l'émetteur et le récepteur pour contrôler les données transmises. Dans cette procédure, les fils servent de transport d'information du type question-réponse, par exemple un fil de l'émetteur passe à « 1 » pour indiquer qu'il est prêt à transmettre et attend que le récepteur place un « 1 » sur un autre fil pour lui indiquer qu'il est prêt à recevoir, et ce n'est qu'alors que la donnée est transmise...La procédure peut sembler lourde, mais elle est très efficace et offre une garantie dans la transmission des données.

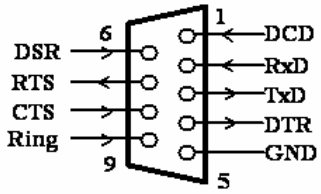
I/ Description matérielle

Des connecteurs standardisés au format 9 et 25 broches, présents d'ailleurs dans tous les ordinateurs (COM1 et COM2), permettent la transmission des données suivant le standard RS232.



Brochage SUB-D 25 broches

- Vue de l'extérieur de l'appareil -



Brochage SUB-D 9 broches

On reconnaît les fils TxD et RxD ainsi que GND de la liaison série classique. Les six autres sont des fil de contrôle. Tous ne sont pas systématiquement utilisés, car hérités de transmissions entre des matériels complexes comme les MODEM.

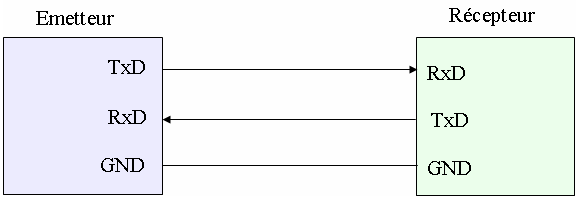
TxD	Transmit Data (sortie transmission)
RxD	Receive Data (entrée de réception)
DCD	Data Carrier Detect - entrée active à 0, informe l 'ordinateur qu'un correspondant a initié une liaison
DTR	Data Terminal Ready - sortie active a 0 utilisée par l 'ordinateur pour demander au récepteur s 'il est prêt à recevoir une donnée.
DSR	Data Set Ready (donnée prête ?) - entrée active à 0 par laquelle le récepteur informe qu'une donnée est prête à être envoyée.
RTS	Request To Send (demande pour émettre)- sortie active à 0 utilisée par l'ordinateur pour signaler qu'il veut transmettre une donnée.
CTS	Clear To Send (prêt à recevoir ?)- entrée active à 0 par laquelle le récepteur informe qu'il est prêt à recevoir des données.
RI	Ring Indicator - entrée active à 0 par laquelle l ordinateur est informé qu'un correspondant veut entrer en communication avec lui.
GND	Ground - la référence des potentiels, la masse.

II/ Protocole matériel

II.1/ Ligne Full Duplex simple

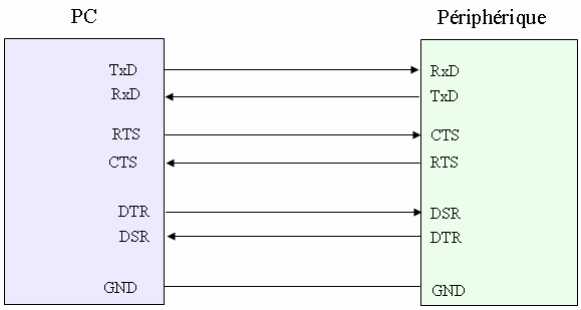
Une ligne full duplex simple nécessite, comme nous l'avions déjà vu, que trois fils.

Toutefois, quand un émetteur envoie une donnée, il n'est pas sûr que le récepteur soit à l'écoute, et donc la donnée peut très bien être ignorée ou écrasée par la suivante.



II.2/ Ligne Full Duplex complète

Ce type de transmission est nécessaire par exemple dans le branchement d'un PC avec un MODEM ou un périphérique intelligent (une imprimante...). Dans la configuration entre un PC et un périphérique, les connecteurs sont câblés par les constructeurs pour que les broches correspondent, le maître étant le PC.



II.3/ Mécanisme de la communication série

1- l'émetteur demande si le récepteur peut recevoir des données (DTR).
2- le récepteur répond qu'il peut recevoir des données (DSR).
3- l'émetteur informe le récepteur qu'il va émettre des données (RTS).
4- le récepteur répond qu'il est prêt (CTS).

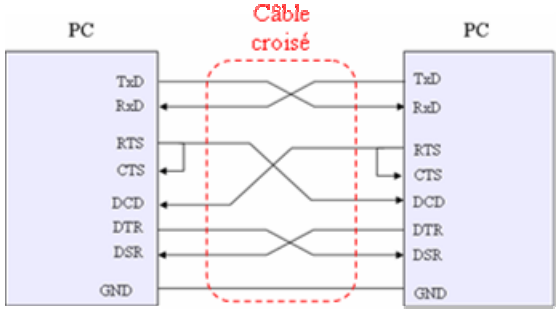
Une fois la donnée transmise tous les signaux reviennent successivement dans leur état initial.

5- l'émetteur signale la fin de la transmission (RTS).
6- le récepteur accuse réception (CTS).
7- l'émetteur libère la ligne de transmission (DTR).
8- le récepteur accuse réception (DSR).

II.4/ Communication entre deux PC

Dans la communication entre PC, par exemple pour échanger des fichiers, les connecteurs sont identiquement câblés, car il n'y a plus de notion de maître et d'esclave.

Pour que les broches correspondent, par exemple TxD (ligne d'émission) avec RxD (ligne de réception), il faut utiliser un câble croisé, dit « null modem ».



III/ Protocole logiciel

Un protocole logiciel gère l'envoi et la réception des données à travers la liaison série, mais cette fois le programme contrôle le flux des données. Il existe un ensemble de protocoles logiciels dont un, le protocole Xon/Xoff, très utilisé du fait de sa simplicité, qui n'utilise que les lignes TxD et RxD.

1- L'émetteur émet vers le récepteur les données les unes à la suite des autres.
2- Le récepteur reçoit les données les unes à la suite des autres jusqu'à ce qu'il soit saturé. Alors, il émet vers le récepteur le caractère spécial Xoff, code 19 réservé en ASCII.
3- A la réception du code 19, Xoff, l'émetteur arrête d'émettre, il attend.
4- Dès que le récepteur a « digéré » les données saturantes, il émet le caractère ASCII Xon, le code de valeur 17, pour indiquer à l'émetteur qu'il peut reprendre l'envoi des données suivantes.
5- L'émetteur recevant le code 17, reprend l'envoi des données suivantes.

Hyper Terminal

I/ Fonction

L'hyper Terminal est une application de Microsoft Windows, qui permet de visualiser dans une fenêtre des informations qui proviennent soit du clavier soit d'un port série (COM1,.., COM4) du PC.
A chaque fois que l'on frappe une touche du clavier, son code est transformé en code ASCII et la trame est envoyée sur le port série par la ligne TxD.
Les informations qui entrent sur le port série du PC, par la ligne RxD, sont véhiculées par des trames au format RS232. La donnée dans chacune des trames est en général d'un octet.
L'application, sous XP, se trouve en général grâce au chemin :

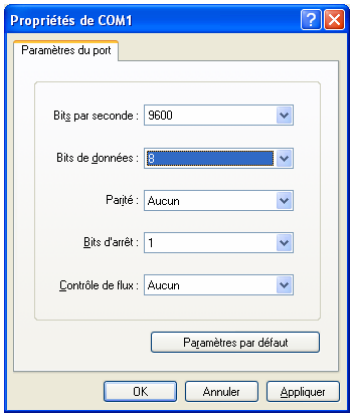
Tous les programmes/ Accessoire/ Communication/ Hyper Terminal

II/ Configuration de la connexion

A la première utilisation, l'application demande des paramètres de configuration.

- Un nom pour la feuille ouverte : EssaiHyper

- Un port série de connexion : Com1



- Des paramètres de format de la trame : 9600 bauds, un octet, pas de parité, 1 bit de stop, pas de contrôle du flux.

III/ Utilisation

Grâce à cette application, il est possible de communiquer avec un autre système, par exemple pour le surveiller ou lui donner des ordres par le clavier ou tester un système en boucle ouverte.

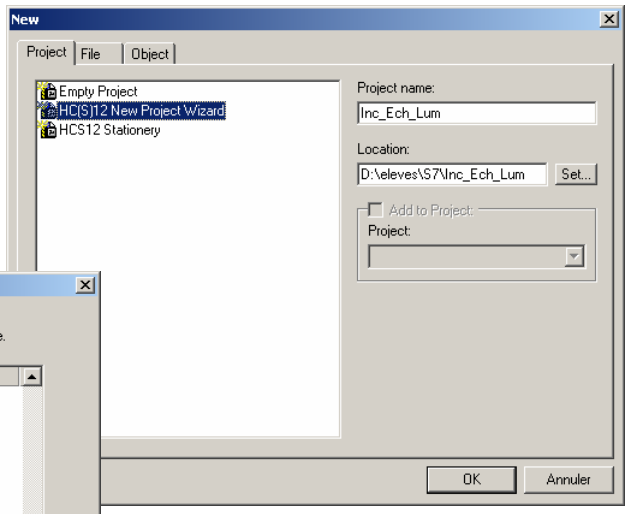
Remarque : pour voir les données la fenêtre doit être active sur le bureau, c'est à dire sélectionnée en premier plan.

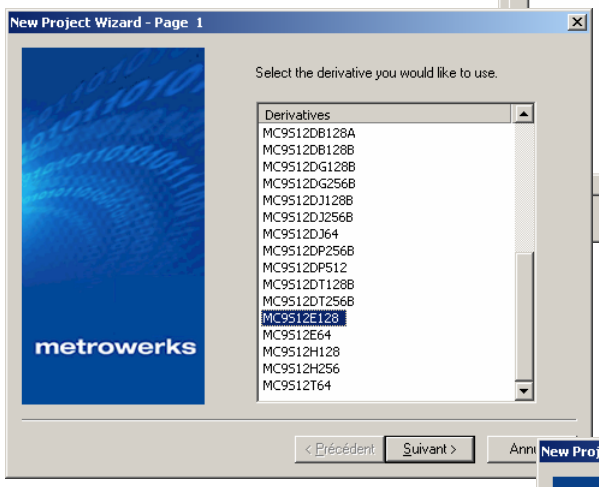
Compléments CodeWarrior

Créer un nouveau projet

La meilleure solution pour créer un nouveau projet est d'utiliser le générateur de projet, le *WIZARD*.

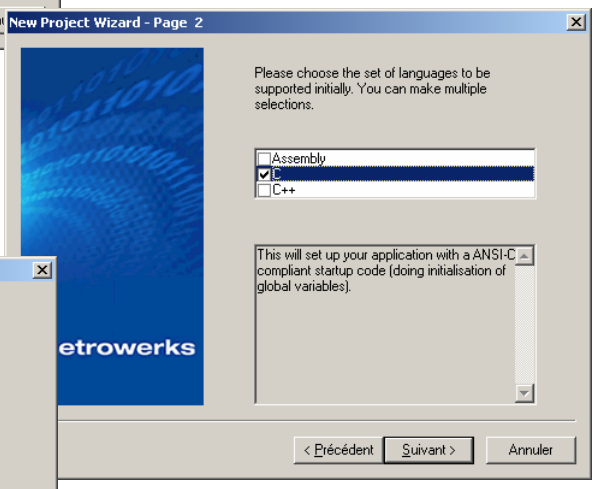
1) Lancer CodeWarrior, ouvrir les menus Fichier/new, cliquer sur « MC(S)12 New Project Wizard », et remplir les 2 champs « Project name » et « Location ».

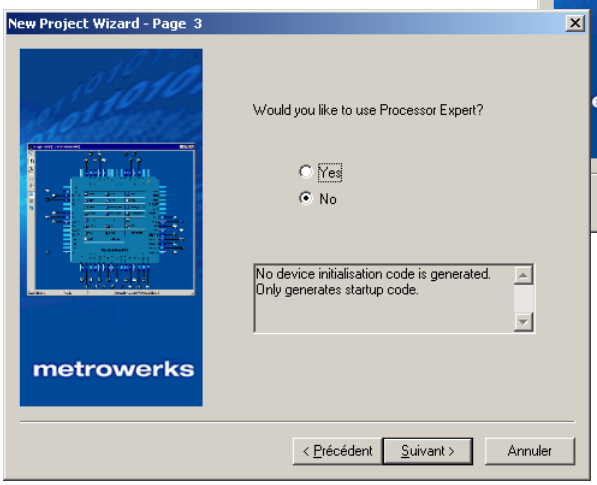




2) Sélectionner le µC adapté à nos applications : MC9S12E128

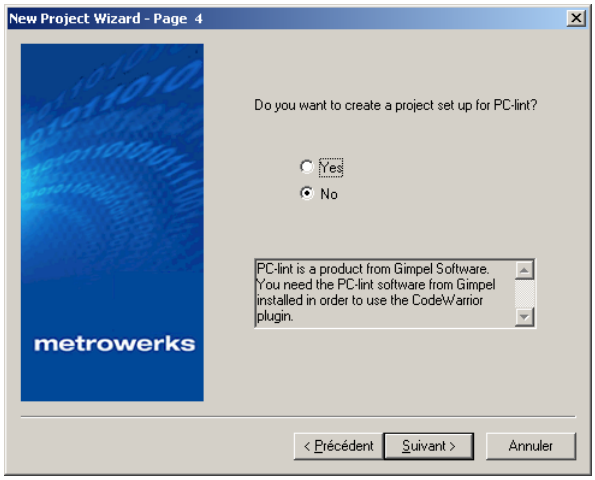
3) Cocher la fenêtre « C »

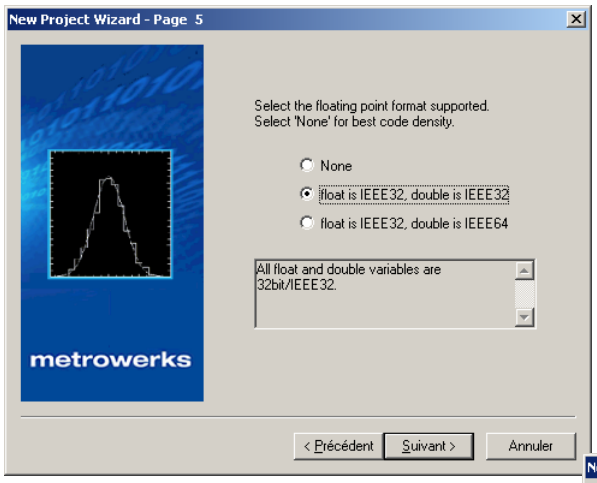




4) Pointer « No », les projets ne nécessite pas l'utilisation du processeur expert qui permet de configurer les E/S de l'applications à notre place.

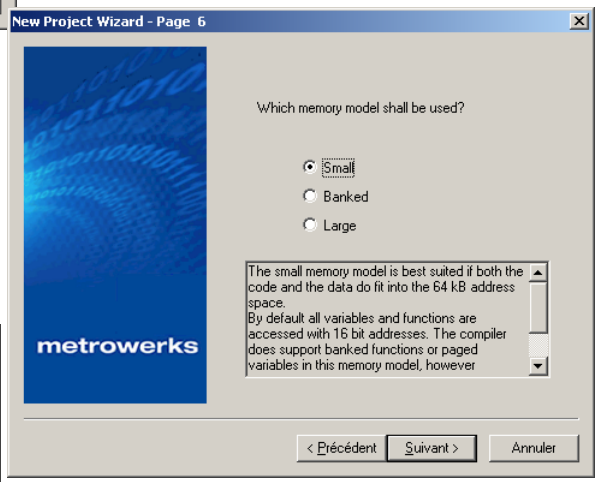
5) Pointer « No », pas d'utilisation de PClint.

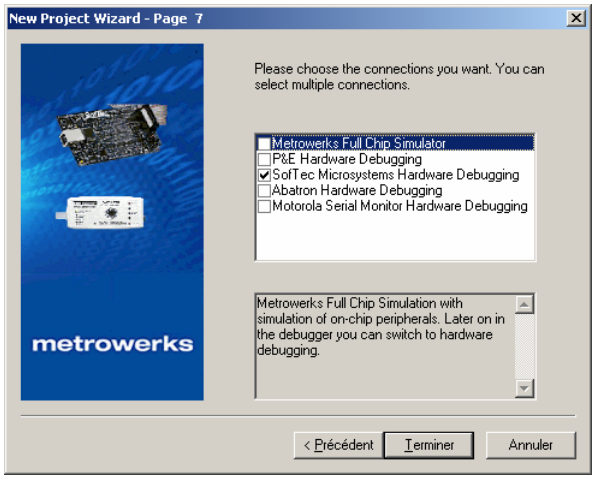




6) On utilise parfois des calculs à virgule flottante et donc on définit ces variables sur 32 bits (4 octets)

7) Pointer « Small », les programmes réalisés en séance étant courts, ils utilisent peu de mémoire.





8) On utilise que très rarement le simulateur, mais par contre tout le temps le debugger matériel, il faut donc cocher « Softec Microsystems Hardware Debugging », et décocher « Metrowerk Full Chip Simulator ».

Le projet est ainsi créé et possède à l'endroit voulu tous les répertoires et sous répertoires du projet.

Codes ASCII

Le morse, créé en 1844 par Samuel F.B. Morse, a été le premier codage à permettre une communication longue distance. L'invention du téléphone, en 1876 par Graham Bell ou plutôt Antonio Meucci en 1871, permit la transmission sur téléscripteurs, qui codaient et décodaient dans l'instant les caractères grâce au code Baudo qui possédait 32 caractères (codage sur 5 bits). C'est dans les années 1960 que le codage ASCII, *American Standard Code for Information Interchange*, est né. Il existe, les codes ASCII standards codés sous 7 bits (128 caractères) et les codes ASCII étendus codés sous 8 bits (256 caractères). L'ASCII étendu n'est pas unique, il dépend de la plateforme utilisée, le plus courant étant le code ANSI.

Les premiers symboles codés de l'ASCII, de 0 à 31, ne sont pas des caractères imprimables, car il s'agit de caractères de contrôle, par exemple de fin de ligne (LF) ou de retour chariot (CR)...de 65 à 90 il s'agit des caractères majuscules, et de 97 à 122 des minuscules (il suffit d'ajouter 32 aux codes des majuscules).

Remarque : le codage UNICODE, sur 16 bits, permet de représenter un caractère quelconque quel que soit le système d'exploitation. Il regroupe ainsi la quasi-totalité des alphabets existants et est compatible avec l'ASCII.

Codes ASCII standards 7 bits, 128 caractères

Car	Dec	Hex	Car	Dec	Hex
Nul	0	0	SP	32	20
SOH	1	1	!	33	21
STX	2	2	"	34	22
ETX	3	3	#	35	23
EOT	4	4	$	36	24
ENQ	5	5	%	37	25
ACK	6	6	&	38	26
BEL	7	7	'	39	27
BS	8	8	(	40	28
HT	9	9	)	41	29
LF	10	A	*	42	2A
VT	11	B	+	43	2B
FF	12	C	,	44	2C
CR	13	D	-	45	2D
SO	14	E	.	46	2E
SI	15	F	/	47	2F
DLE	16	10	0	48	30
DC1	17	11	1	49	31
DC2	18	12	2	50	32
DC3	19	13	3	51	33
DC4	20	14	4	52	34
NAK	21	15	5	53	35
SYN	22	16	6	54	36
ETB	23	17	7	55	37
CAN	24	18	8	56	38
EM	25	19	9	57	39
SUB	26	1A	:	58	3A
ESC	27	1B	;	59	3B
FS	28	1C	<	60	3C
GS	29	1D	=	61	3D
RS	30	1E	>	62	3E
US	31	1F	?	63	3F

@	64	40	`	96	60
A	65	41	a	97	61
B	66	42	b	98	62
C	67	43	c	99	63
D	68	44	d	100	64
E	69	45	e	101	65
F	70	46	f	102	66
G	71	47	g	103	67
H	72	48	h	104	68
I	73	49	i	105	69
J	74	4A	j	106	6A
K	75	4B	k	107	6B
L	76	4C	l	108	6C
M	77	4D	m	109	6D
N	78	4E	n	110	6E
O	79	4F	o	111	6F
P	80	50	p	112	70
Q	81	51	q	113	71
R	82	52	r	114	72
S	83	53	s	115	73
T	84	54	t	116	74
U	85	55	u	117	75
V	86	56	v	118	76
W	87	57	w	119	77
X	88	58	x	120	78
Y	89	59	y	121	79
Z	90	5A	z	122	7A
[	91	5B	{	123	7B
\	92	5C	|	124	7C
]	93	5D	}	125	7D
^	94	5E	~	126	7E
_	95	5F	Del	127	7F

Programmes utilitaires

I/ Conversion d'un entier en BCD

```
char d0,d1,d2,d3,d4;

void bin_to_bcd(int bin)
    {
    d4=(char)(bin/10000);
    bin=bin-(d4*10000);
    d3=(char)(bin/1000);
    bin=bin-(d3*1000);
    d2=(char)(bin/100);
    bin=bin-(d2*100);
    d1=(char)(bin/10);
    d0=bin-(d1*10);
    }
```

II/ Affichage LCD