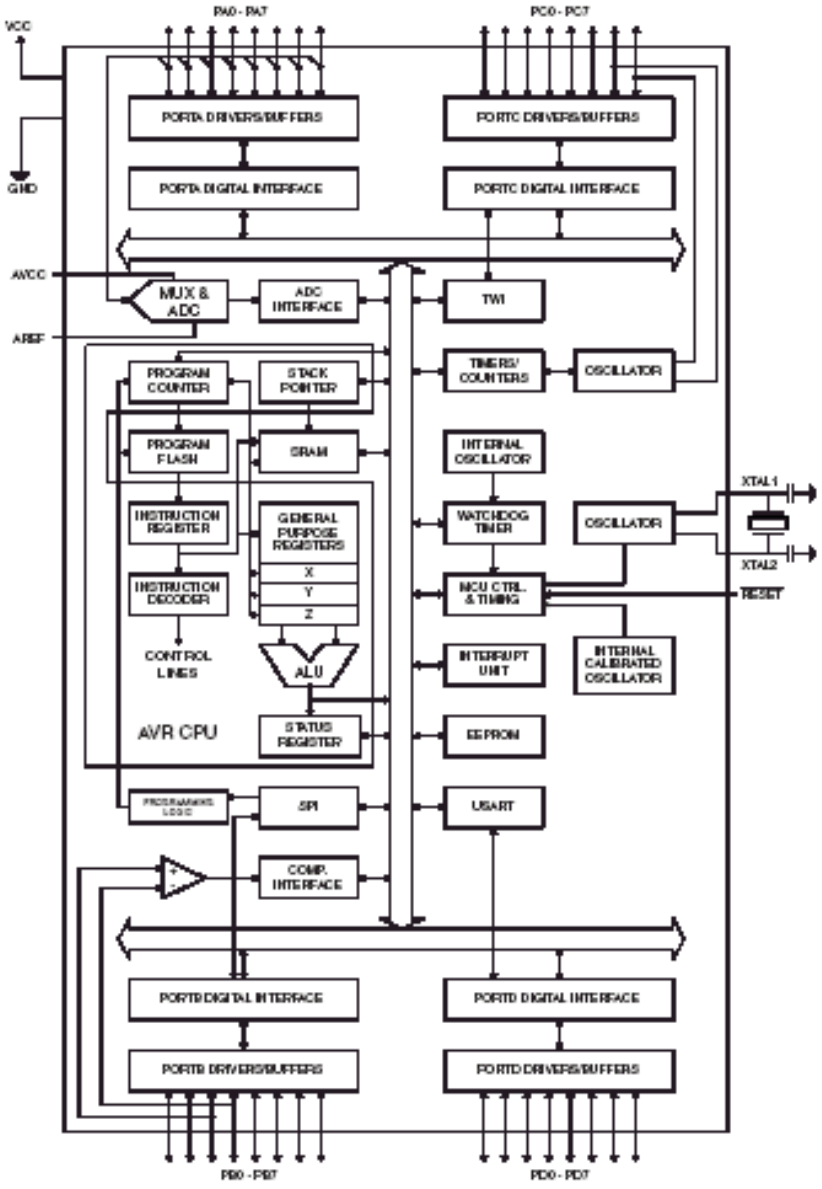
Le microcontrôleur Atmel Atmega16



Jean Louis BOIZARD
Maître de conférences Université Toulouse II

Version 3.0

Le microcontrôleur Atmel Atmega16

Avant-propos

Le présent document a pour objectif principal de démystifier le fonctionnement des microcontrôleurs.

L'évolution considérable des outils de développement fait qu'il est tout à fait possible de construire aujourd'hui des applications à base de ces composants sans en avoir a priori une connaissance intime.

Afin d'aborder le contenu de ce cours dans les meilleures conditions, le lecteur aura avantage à revoir/assimiler les points suivants :

- Numération binaire et hexadécimale
- logique combinatoire et tables de vérité
- logique séquentielle (essentiellement les bascules de type D ou flip flop et les compteurs)

Remarque :
Une partie de la documentation sur les entrées sorties du microcontrôleur Atmega16 est reprise de la documentation de J.M. SEGURET, enseignant au Lycée Tehnique Maurice Genevoix d'INGRE dans le LOIRET.

1) LE MICROPROCESSEUR

1.1) Historique

Le premier microprocesseur (Intel 4004) a été inventé en 1971. C'était un processeur ayant un bus de données de 4 bits. Sa fréquence de travail était de 108 Khz et il comportait 2300 transistors. Avec l'évolution de la technologie et l'arrivée des circuits sub-microniques (tracés inférieurs au micron), les performances des circuits électroniques ont décuplé tant par leur vitesse de traitement que par leur niveau d'intégration. Le tableau ci-dessous donne un aperçu de cette évolution (source INTEL).

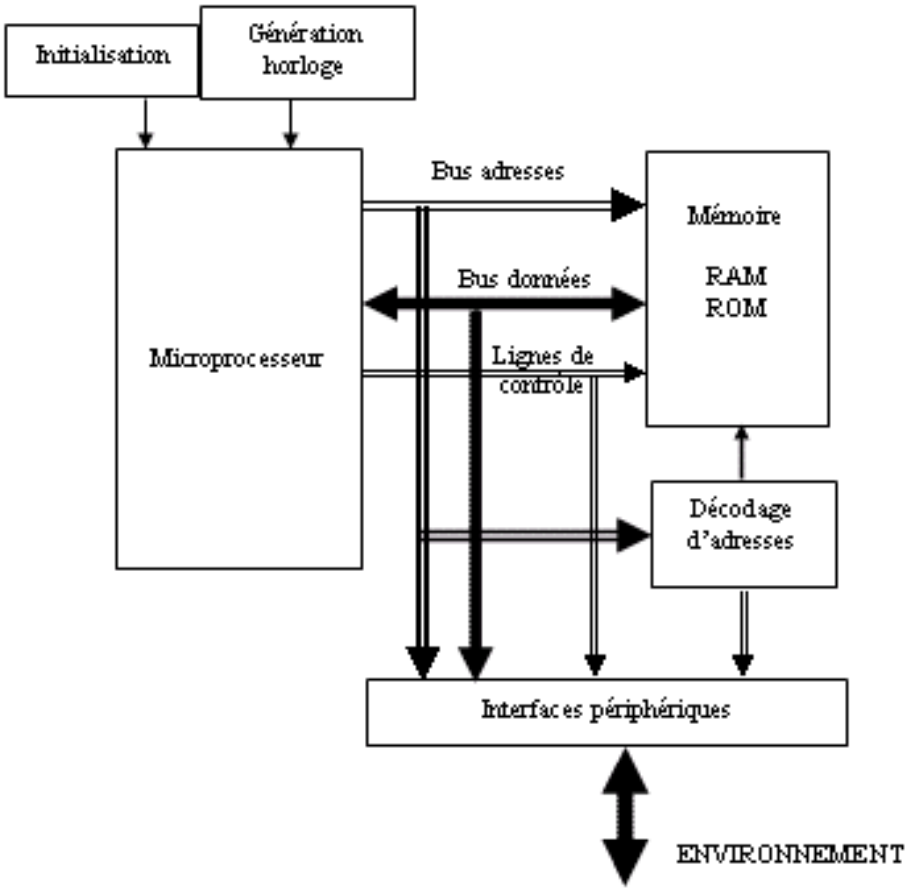
année	référence	Bus données	fréquence	Nbre de transistors
1971	4004	4	108 Khz	2300
1974	8080	8	2 Mhz	4500
1978	8086	16	10Mhz	29 000
1985	386	32	40Mhz	275 000
1993	Pentium	32	75Mhz	3.5 M
1995	Pentium Pro	32	150Mhz	5.5 M
1997	Pentium II	32	233Mhz	7.5 M
1999	Celeron	32	850Mhz	9.5 M
2000	Pentium IV	64	1.5 Ghz	55 M
2004	Pentium IV série 600	64	3.73 Ghz	169 M

1.2) Domaines d'utilisation

De part leur facilité de mise en œuvre et leur coût de revient très bas les microprocesseurs (microcontrôleurs) sont utilisés dans de nombreux domaines :

- Automobile (ex : Volvo S80 équipée de 11 µC/µP)
- Aéronautique (ex : calculateur de vol Airbus A320)
- Electroménager (four électrique, micro-onde, lave-linge, …)
- Machines outils
- Mesure et régulation (oscilloscopes numériques, multimètres, PID, …)
- Stations de travail (PC, station SUN, …)
- Distributeurs de boisson
- …

1.3) ARCHITECTURE D'UN SYSTEME A µP (CONFIGURATION MINIMUM)



L'architecture matérielle minimum d'un système à microprocesseur est représentée sur la figure précédente. Celle-ci comporte :

- le microprocesseur
- un circuit d'initialisation
- un générateur d'horloge
- une mémoire à lecture seule (ROM)
- une mémoire à lecture/écriture (RAM)
- un dispositif de décodage d'adresses
- des interfaces de périphériques.
- des bus de communication

1.3.1) le microprocesseur

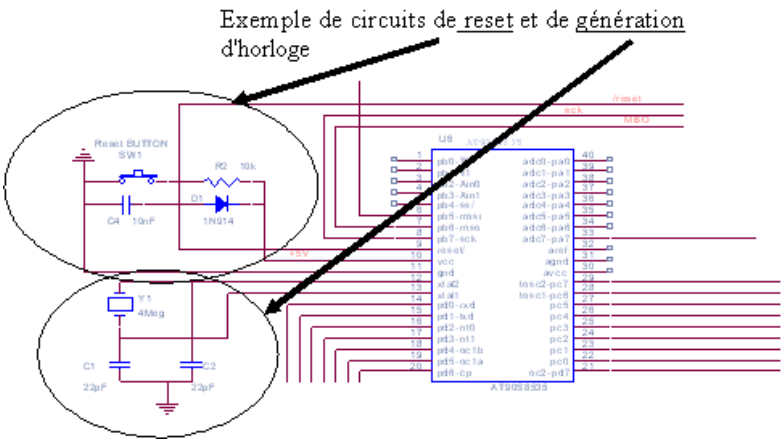
C'est un **automate séquentiel** dont le rôle est la **lecture, le décodage** puis **l'exécution** des instructions présentes en mémoire (ROM ou RAM) et qui constituent le **programme.** Son fonctionnement sera vu au chapitre 1.3.

1.3.2) le circuit d'initialisation (reset)

A la mise sous tension, il empêche le démarrage du microprocesseur tant que les alimentations en énergie ne sont pas stabilisées . En général le circuit de « reset » bloque le microprocesseur quelques millisecondes à partir de sa mise sous tension (cette durée dépend en fait du type de µprocesseur et est donnée par la documentation du constructeur).
En fonctionnement normal, il permet de réinitialiser le système.
Généralement le signal d'initialisation (reset) est obtenu à partir d'un circuit électrique mettant en œuvre un réseau RC. Le choix des valeurs de R et C permet de déterminer la durée à l'état actif du signal. On trouve également dans le commerce beaucoup de circuits spécialisés assurant cette fonction (ex : Motorola MC 34064, Maxim Max-6475, …).

1.3.3) le générateur d'horloge

C'est lui qui assure le cadencement de l'exécution des instructions. Plus la fréquence d'horloge est élevée, plus les instructions seront exécutées rapidement. La puissance d'un microprocesseur s'exprime en MIPS (Millions d'instructions par seconde) mais cette grandeur est très insuffisante pour estimer les performances d'un système.
Nota : dans les systèmes récents dont la fréquence de travail est très élevée, l'horloge est obtenue en multipliant la fréquence de l'horloge de base (issue d'un quartz). Cette opération est réalisée par une boucle à verrouillage de phase implantée sur la puce du processeur.



1.3.4) la mémoire ROM

C'est une mémoire à **lecture seule** (Read Only Memory). Dans les microsystèmes, c'est elle qui en général contient le programme à exécuter. Dans les systèmes plus complexes (stations de travail, PC, ..) le programme, stocké en mémoires de masse (disque dur, CD-rom, …), est chargé dans la mémoire vive (RAM) avant d'être exécuté.

Fonctionnellement la mémoire ROM a pour rôle de fournir sur son « buffer » de sortie un mot binaire correspondant au contenu de la case pointée par l'adresse d'entrée. Une mémoire est spécifiée par sa capacité et son organisation.

Exemple :

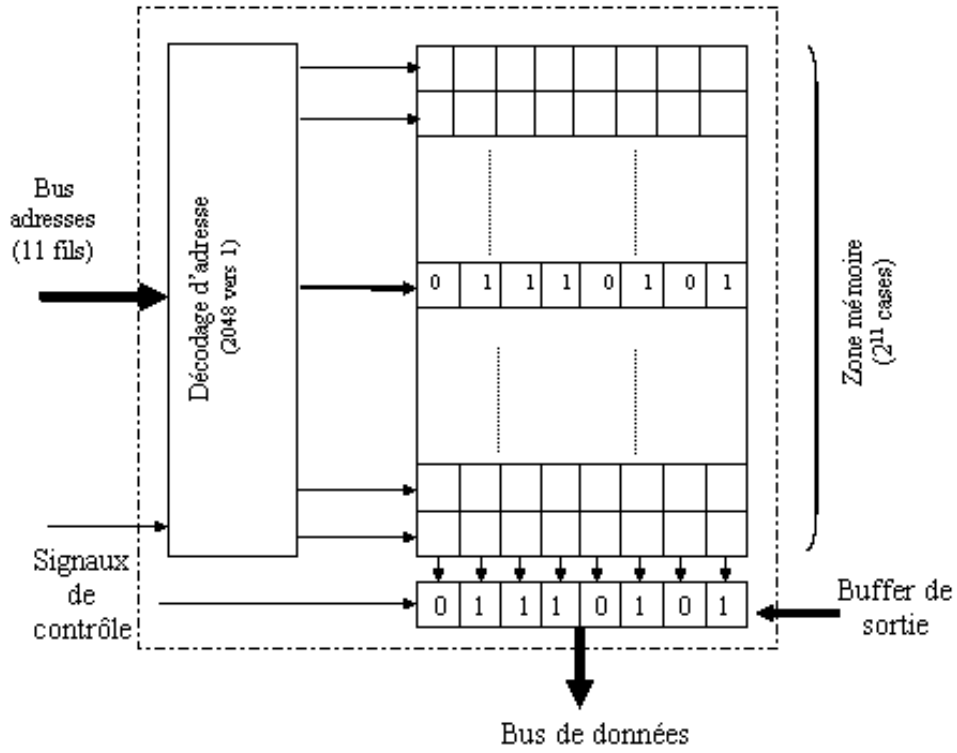
- une mémoire de 32K * 8 signifie que celle-ci contient 32K mots de 8 bits. Pour adresser toutes les cases de cette mémoire, le bus d'adresse devra avoir une largeur de 15 bits (2^{15} combinaisons soit 32768).
-

Nota : 1Koctets = 1024 octets

Remarque :

La mémoire ROM conserve son contenu lorsque son alimentation est coupée.

Ex : ARCHITECTURE INTERNE d'une MEMOIRE ROM de 2Ko



1.3.5) La mémoire RAM

La mémoire RAM (Random Access Memory) est une mémoire à accès aléatoire qui peut être **lue** ou **écrite** indéfiniment. Dans les microsystèmes, elle est souvent réservée pour stocker des variables, des résultats intermédiaires et sert également de « pile » pour le processeur. Comparativement à la ROM, le buffer de sortie est **bidirectionnel** et c'est le signal de lecture/écriture issu des lignes de contrôle qui fixe le sens de circulation des données.

Remarque :

La mémoire RAM perd son contenu lorsque son alimentation est coupée.

1.3.6) Les bus de communications

- le bus d'adresses

Ce sont des lignes physiques (équipotentielles) qui véhiculent les adresses générées par le microprocesseur. La taille du bus fixe sa capacité d'adressage. Dans les systèmes monoprocesseurs et selon leurs types, le bus d'adresses est souvent unidirectionnel.

Exemples :
Motorola 6800 : bus adresses 16 bits => 65536 adresses possibles (2^{16} combinaisons).
Motorola 68000 : bus adresses 20 bits => 1 Million adresses possibles (2^{20} combinaisons).
Intel 80386 : bus adresses 32 bits => 4 G adresses possibles (2^{32} combinaisons).

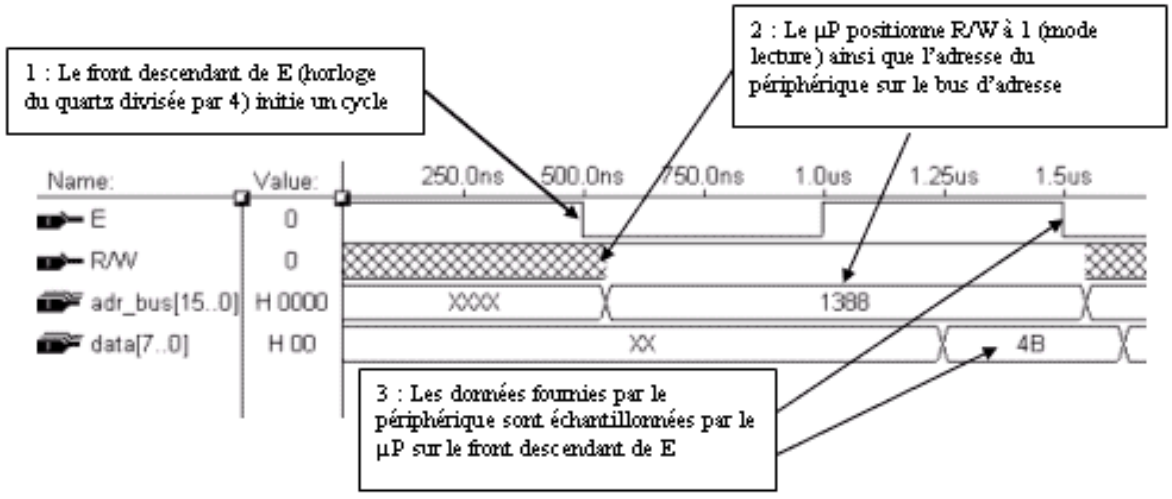
- le bus de données

Ce sont des lignes physiques (équipotentielles) qui véhiculent les données. Ce bus est bidirectionnel (les données peuvent entrer ou sortir du µP suivant qu'il est en mode lecture ou écriture). La taille de ce bus correspond à la taille du bus de données interne au µP.

Exemples :
Motorola 6800 : bus de données 8 bits.
Motorola 68000 : bus de données 16 bits.
Motorola DSP 56000 : bus de données 24 bits.
Motorola 68020 : bus de données 32 bits.

- Le bus de contrôle

Il est de taille variable suivant la complexité du processeur. C'est lui qui synchronise les échanges entre le processeur et ses périphériques. Le chronogramme ci-dessous illustre un cycle de lecture d'un périphérique du microprocesseur Motorola 6809.



1.3.7) Les interfaces de périphériques

Ce sont souvent des circuits intégrés spécialisés directement interfaçables avec le microprocesseur et qui assurent les liens avec les unités matérielles du système.

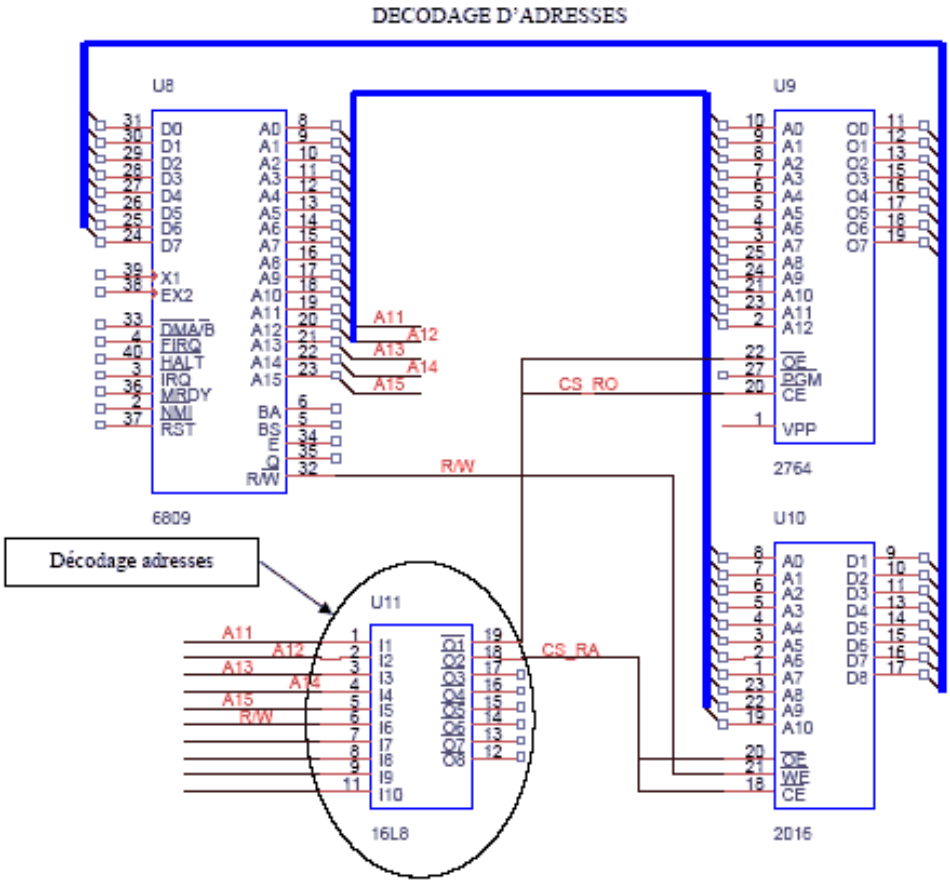
Exemples :

- interface IDE pour lecteur de disquettes
- interface SCSI pour disque dur
- entrées/sorties numériques et analogiques
- interfaces vidéo, claviers, bus USB
- etc…

Chaque interface dispose d'un champ adresse qui lui est propre. Ce champ adresse est fourni par **le bloc de décodage d'adresse**.

1.3.8) <u>Le décodage d'adresse</u>

Il a pour but d'**assigner** à chaque bloc mémoire et aux différents périphériques qui composent le système un emplacement et un espace dans le champ adressable par le microprocesseur.



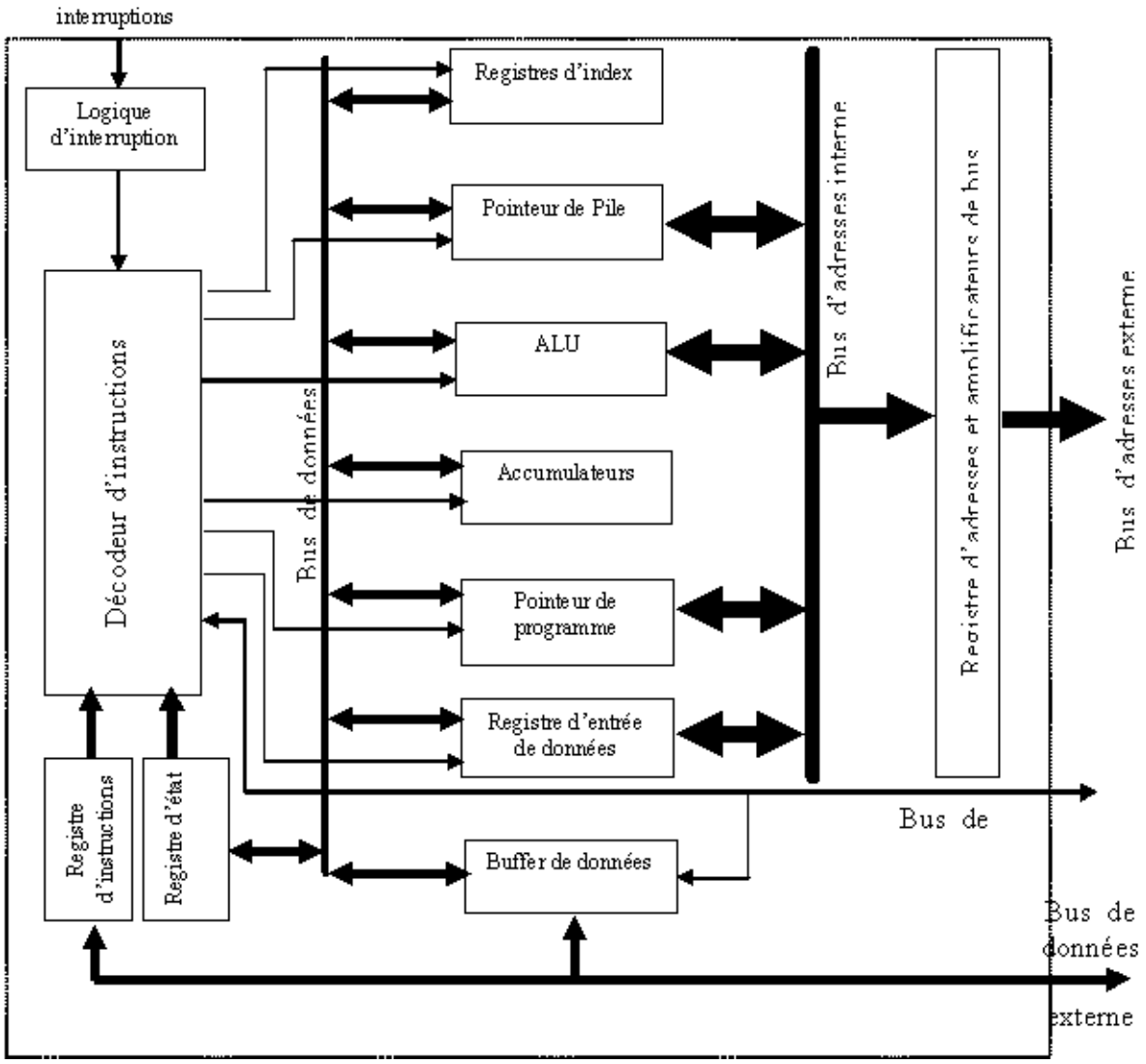
<u>Exercice :</u>

Déterminer les équations de CS_RAM et CS_ROM pour que les mémoires RAM et ROM débutent respectivement aux adresses :

- $(A000)_H$ pour la ROM (circuit U9)
- $(5000)_H$ pour la RAM (circuit U10)

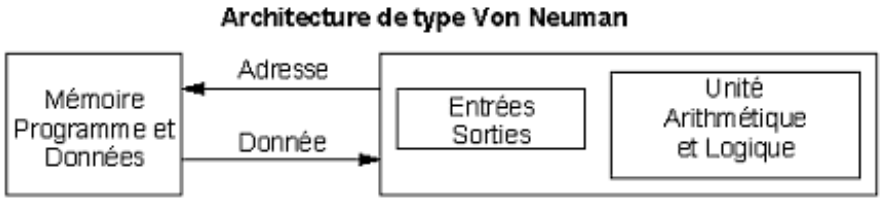
Quel est l'espace occupé par chaque mémoire. Donner l'adresse de fin correspondante. (rep : BFFF pour la ROM et 57FF pour la RAM).

1.4) ARCHITECTURE INTERNE D'UN MICROPROCESSEUR



L'architecture interne du microprocesseur ci-dessus est une architecture de type « Von Neuman [(1)] » (**un seul bus de données**). Elle se compose essentiellement :

- D'une unité de décodage d'instructions
- D'un ensemble de registres
- D'amplificateurs de lignes (buffers de données et d'adresses)
- D'une unité arithmétique et logique.



(1) Von Neuman : Mathématicien américain (Budapest 1903-Washington 1957)

1.5) Architectures RISC et CISC

Les premières générations de microprocesseurs étaient à architecture CISC (Complex Instructions Set Computer) ce qui nécessitait plusieurs cycles d'horloge de base (fournie par le quartz) pour exécuter une instruction. Au fil du temps les concepteurs de circuits se sont aperçus que 80% des traitements effectués faisaient appel à 20% des instructions du µP. L'idée est donc venue de développer des microprocesseurs possédant un jeu d'instructions réduit (RISC : Reduced Instruction Set Computer) mais exécutable dans un temps très court (souvent en un seul cycle d'horloge). Ces microprocesseurs sont souvent à architecture de « **Harvard** » et disposent d'un mode de fonctionnement en « **pipeline** ». Aujourd'hui les deux types de processeurs cohabitent et leur utilisation est fonction des applications visées (station de travail, serveur, système temps réel …).
Notons toutefois que si les architectures RISC sont plus simples à réaliser sur le plan matériel, les programmes assembleurs et les compilateurs associés sont plus compliqués à réaliser.

Exemple :

1) instruction d'addition de 2 registres (micro RISC Atmel Atmega16)

   ADD Rd, Rr Rd <= Rd + Rr nécessite un cycle d'horloge (125ns pour une horloge à 8Mhz).

2) instruction d'addition de 2 registres (micro CISC Motorola 68HC11)

   ABA A <= A + B nécessite deux cycles d'horloge (250ns pour une horloge à 8Mhz).

	RISC	CISC
Avantages	Temps d'exécution court, hardware plus simple, coût puce électronique réduit	Instructions puissantes, programme plus simple à écrire, moins de lignes de programme, compilateur simple à concevoir
Inconvénients	Programme plus complexe à écrire, plus de lignes, compilateur difficile à concevoir	Temps exécution long, hardware plus compliqué et plus coûteux

1.6) Les microcontrôleurs

L'apparition des microcontrôleurs résulte de l'évolution des niveaux d'intégration des transistors (des portes) dans les circuits intégrés. Il est devenu possible de rajouter sur la puce de silicium dédiée au processeur un ensemble de ressources telles que :

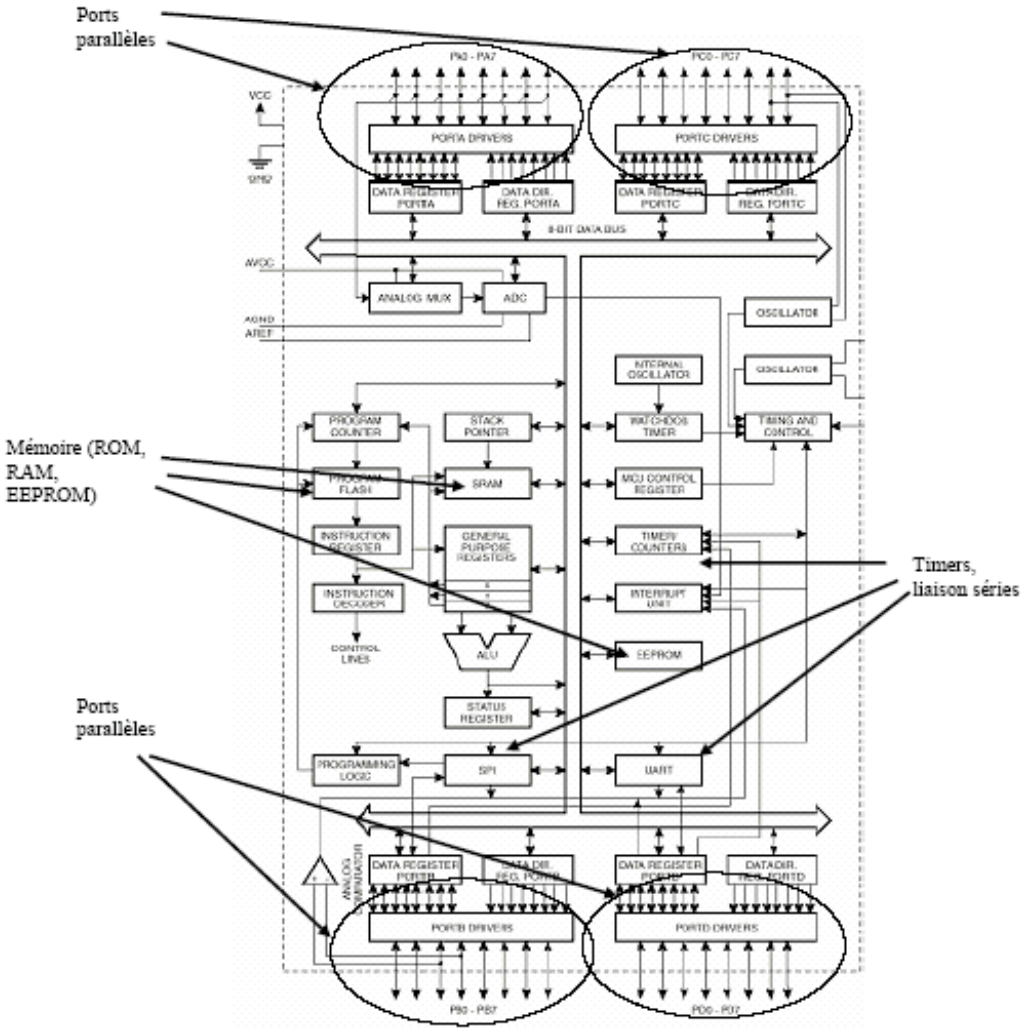
- de la mémoire ROM, EPROM, EEPROM, FlashEPROM
- de la mémoire SRAM

- du décodage d'adresse
- des périphériques (entrées-sorties numériques, liaison série RS232C, timers…)

Les microcontrôleurs récents intégrent des périphériques plus complexes tels que:

- des convertisseurs N/A et A/N 10 bits
- des interfaces bus I2C, bus CAN, VAN, USB …
- des interfaces type réseaux éthernet…
- des contrôleurs de mémoire dynamique…

Exemple d'architecture interne de microcontrôleur (Atmel Atmega16) :



2) LES METHODES ET OUTILS DE DEVELOPPEMENT

Compte tenu de la complexité croissante des microcontrôleurs , le matériel courant de laboratoire n'est plus adapté pour la mise en oeuvre de tels composants. Un certain nombre d'outils plus ou moins performants s'avèrent indispensables pour effectuer la mise en œuvre du circuit voire tester le bon fonctionnement de la carte électronique réalisée. Parmi ces outils on peut citer :

- les langages de programmation
- les simulateurs logiciels

- les émulateurs « in circuit »
- les émulateurs « temps réel » à mémoire trace
- les émulateurs « low cost »

2.1) Les langages de programmation

Les µC sont des composants qui exécutent un programme (ces programmes sont constitués de suites d'instructions). Se pose alors le problème du choix du langage de programmation : langage de bas niveau (assembleur) ou langage de haut niveau (C, Basic, Pascal, JAVA,…).

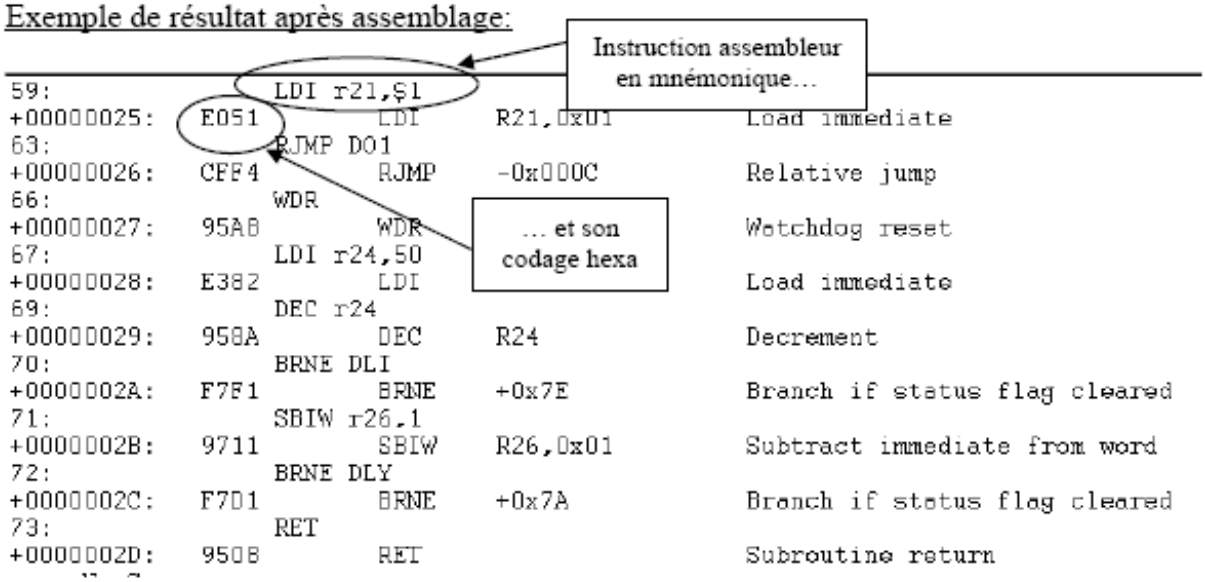
2.1.1) L'assembleur

Le langage de bas niveau (assembleur) est un langage qui **est directement compréhensible et exécutable** par le processeur. Pour des commodités d'utilisation, le programmeur manipule des instructions représentées par des **mnémoniques**. A chaque mnémonique correspond un code hexadécimal. C'est le rôle du **programme assembleur** de transformer le programme écrit en mnémonique en programme hexadécimal (ou binaire) directement exécutable.

Exemple de programme assembleur source (micro Atmel Atmega16) :

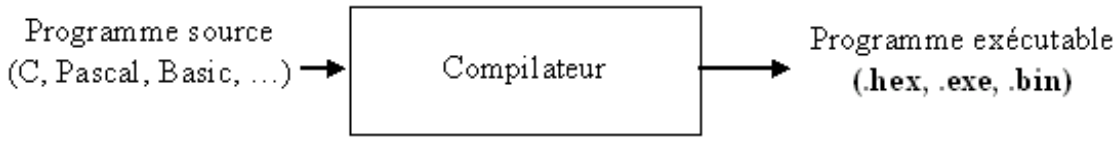
```
debut:      clr temp          ;temp=0
            clr th            ;th=0
            clr NBR           ;NBR=0
            clr cycle         ;cycle=0

            ldi cycle,50      ;mesure sur 50 périodes
att1:       in  temp,signal   ;test si le signal est au nvl 1
            andi temp,1
            cpi temp,0
            breq att1         ; si le nvl est 0 alors attente
incr1:      inc th            ; sinon on incrémente th
            in temp,signal    ;tant que le signal est au nvl 1, on incrémente th.
             andi temp,1
            cpi temp,1
            breq incr1
            …
```

Exemple de résultat après assemblage:



2.1.2) Les langages de haut niveau

Ils permettent de s'affranchir de la connaissance du langage assembleur qui est spécifique à une famille de µC. Ils ne nécessitent pas la connaissance approfondie de l'architecture interne du microcontrôleur utilisé (registres, ressources diverses…). C'est le **compilateur** qui se charge de **traduire** les instructions de haut niveau en instructions assembleur puis en code machine. La connaissance intime du processeur est laissée à ceux qui développent les compilateurs.



Nota:
Lorsque le compilateur est exécuté sur une machine hôte dont le processeur est différent de la machine cible, on parle de cross-compilateur (compilateur croisé).

2.1.3) Avantages et inconvénients des langages assembleurs et des langages évolués

ASSEMBLEUR	LANGAGE DE HAUT NIVEAU
Inconvénients: - souvent spécifique au µC utilisé - nécessite la connaissance de l'architecture interne - maintenabilité difficile - évolutivité difficile - peu de pérennité des investissements - nécessite une formation spécifique Avantages: - code généré très compact - vitesse d'exécution du programme	Inconvénients: - code généré moins compact (dépend des performances du compilateur) - vitesse d'exécution plus lente Avantages: - langage commun indépendant du µP/µC - ne nécessite pas une connaissance forte de l'architecture du processeur - programmes plus facilement maintenables et évolutifs - meilleure pérennité des investissements (si changement de processeur) - apprentissage du langage aisé

CONCLUSION:

A l'exception des cas où une rapidité d'exécution ou compacité du code est recherchée, les développements se font autant que possible en langage de haut niveau (C, C^{++} principalement, éventuellement JAVA avec l'apparition récente de quelques compilateurs). Pratiquement on effectue un mixage des deux méthodes: modules assembleurs pour les parties requérant de la vitesse d'exécution et langage de haut niveau pour le reste du programme.
Il est à noter toutefois que pour avoir un bon niveau d'expertise des systèmes temps réel la connaissance de l'assembleur peut s'avérer incontournable.

2.2) Les simulateurs logiciels

Ce sont des programmes informatiques exécutés sur des stations de travail ou PC de bureau et qui permettent de simuler les instructions du programme et l'évolution de l'ensemble des ressources internes du composant.
Ces simulateurs ne permettent pas de faire de la simulation « temps réel » mais s'avèrent suffisants pour beaucoup d'applications. Il est possible toutefois de prendre en compte des événements extérieurs au composant en créant des fichiers de stimulis mais cette méthode reste fastidieuse.
Une fois le programme mis au point (simulé et testé), celui-ci est téléchargé dans l'application par le port série ou le port // du PC. D'autres types de liaisons peuvent être utilisées pour cette opération notamment la liaison « JTAG » qui se répend de plus en plus ou la liaison « BDM » propre à la société Freescale (ex Motorola).

2.3) Les émulateurs « in circuit »

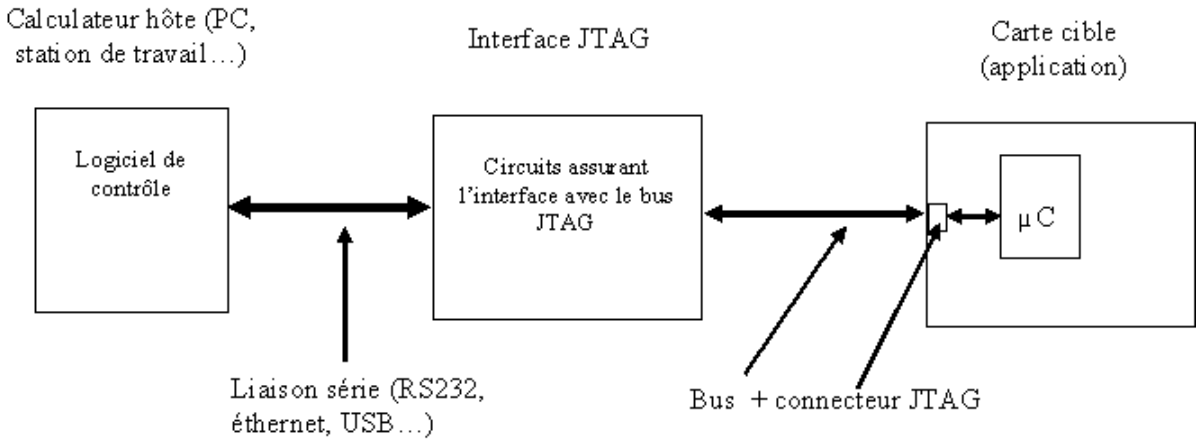
Ces émulateurs ont fait leur apparition assez récemment. Ils sont la conséquence de l'évolution rapide des niveaux d'intégration dans les circuits électroniques et du fait que les ressources des microcontrôleurs ne sont plus accessibles de l'extérieur. En effet, la partie matérielle nécessaire à l'émulation des microcontrôleurs devenant peu importante en regard des ressources intégrées dans ces composants, il était intéressant que celle-ci soit également intégrée sur la même puce de silicium.

Le principe d'un émulateur « in circuit » est le suivant :

Le programme d'application est **téléchargé dans la mémoire du composant** (souvent de la FlashEprom) par une liaison spécialisée (JTAG, BDM ou autre). Si l'utilisateur a mis un point d'arrêt dans son programme**, l'adresse** de l'instruction qui suit le point d'arrêt est **stockée** dans un registre spécifique du composant puis **comparée** en temps réel à la valeur courante du compteur de programme. Lorsqu'il y a **égalité** entre les deux valeurs, le fonctionnement du microcontrôleur est stoppé et l'état des ressources internes est renvoyé au calculateur hôte par la liaison spécialisée pour affichage. Le programme s'exécute par conséquent sur **la machine cible** (et non sur la machine hôte comme dans le cas précédent) à la **vitesse réelle**.

Ce type d 'émulateur est un bon compromis entre les performances et son prix de revient. Il n'est pas disponible cependant sur les microcontrôleurs d'entrée de gamme.
Il est à noter également que toute modification du programme applicatif entraine un effacement et une reprogrammation de la mémoire Flash du composant, laquelle n'accepte pas indéfiniment des cycles d'effacement et de reprogrammation (environ 10 000 cycles pour le circuit Atmel Atmega16).

INTERCONNEXION d'un EMULATEUR JTAG
(Le µC est <u>présent</u> sur la carte cible)



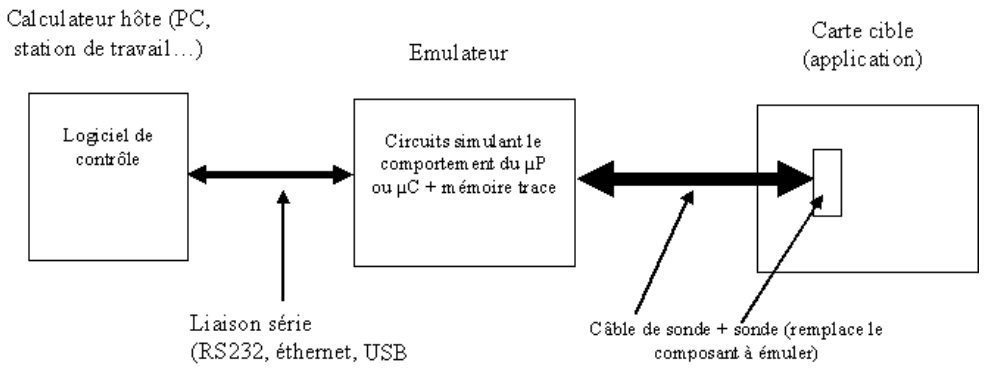
2.4) Les émulateurs « temps réel » à mémoire trace

A l'inverse des simulateurs logiciels, les émulateurs « temps réel » à mémoire-trace sont des simulateurs **comportementaux matériels**. Il s'agit d'un ensemble de circuits électroniques (une version « éclatée » du µC) ayant le même comportement que le circuit à émuler auxquels on a associé de la **mémoire-trace** (RAM).

<u>Mise en œuvre</u> :

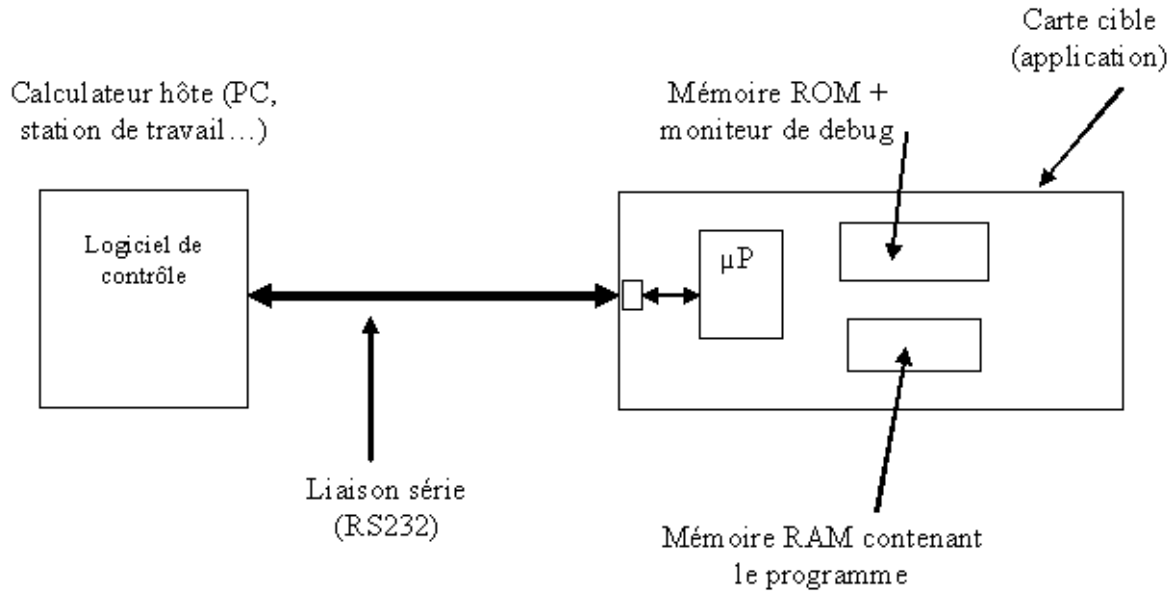
A partir d'un PC hôte, le programme est téléchargé dans l'émulateur. Une sonde **remplace** le composant à émuler sur la carte cible. Le programme est exécuté en temps réel par l'émulateur et la mémoire-trace « espionne » et enregistre l'activité des différents circuits. Lorsque le processeur rencontre un point d'arrêt dans le programme, le calculateur hôte reprend la main et a accès, en analysant le contenu de la mémoire-trace, à **l'historique** de l'exécution du programme et de l'évolution des ressources.
C'est l'émulateur le plus performant et le plus coûteux que l'on puisse rencontrer actuellement sur le marché. Il est surtout utilisé pour la mise au point matérielle de la carte cible et la mise au point de programmes complexes faisant appel à des entrées-sorties.

INTERCONNEXION d'un EMULATEUR CLASSIQUE
(Le µC est **<u>remplacé</u>** par une sonde)



2.5) Les émulateurs « low cost »

On les nomme ainsi car ils font appel à peu de matériel. L'architecture d'un tel émulateur est représentée ci-dessous.



Cet émulateur impose lors de la conception de la carte cible d'intégrer une mémoire ROM qui contiendra un programme appelé « moniteur de debug » et une RAM qui contiendra le programme applicatif à tester.

Lors de la mise sous tension de la carte cible, le microprocesseur exécute le programme correspondant au moniteur de debug. Celui-ci permet alors de :

- gérer la liaison série avec le PC ou la station de travail
- exécuter les commandes issues de la station (téléchargement du programme applicatif dans la RAM, mise en place de points d'arrêt, …)
- retourner l'état des registres internes du µP vers le calculateur hôte
- …

Les avantages :

- Le programme applicatif est exécuté par le µP de la carte cible à **vitesse réelle**.
- Le programme peut être modifié et téléchargé indéfiniment.
- Pas besoin d'émulateur matériel.
- Possibilité de mettre des points d'arrêt
- Faible coût

Les inconvénients :

- Carte cible plus imposante que ne le nécessite l'application.
- Ajout d'une mémoire RAM + ROM de debug si émulation d'un µP
- Liaison série (RS232) non disponible pour l'application
- Le plan mémoire n'est pas celui de la version définitive.
- Très peu pratique voire impossible d'émuler des microcontrôleurs

3) Le microcontrôleur Atmega16

AVERTISSEMENT ! Ce chapitre ne présente que quelques ressources du microcontrôleur. Pour une information plus complète se référer au document constructeur : *http://www.atmel.com/dyn/products/datasheets.asp?family_id=607*

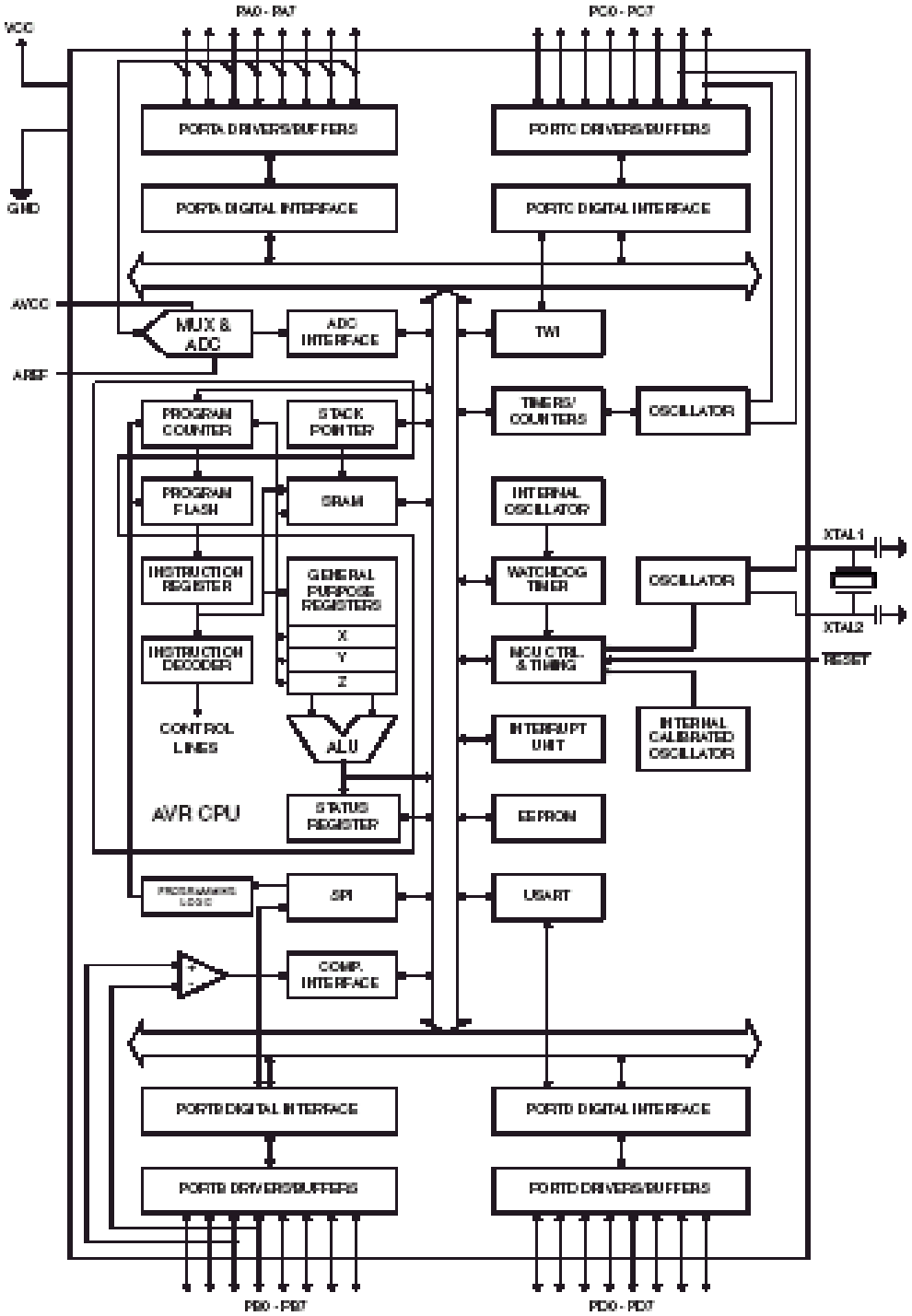
3.1) Présentation

Le circuit Atmega16 est un microcontrôleur **8 bits** à architecture **RISC** produit par la société ATMEL. Il intègre un certain nombre de ressources et de périphériques lui permettant de couvrir un large éventail d'applications.

Caractéristiques principales :

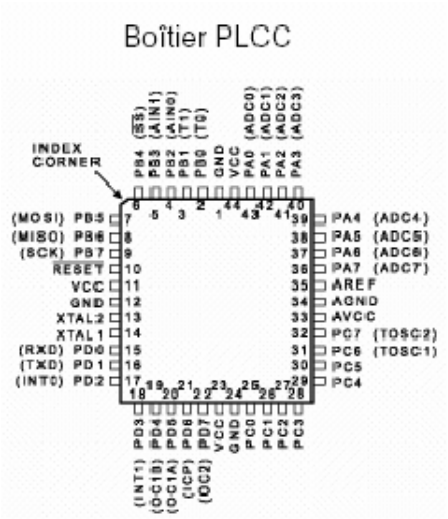
- CPU 8 bits capable d'exécuter la plupart des instructions en 1 cycle d'horloge
- Mémoire EEPROM de type flash de 16koctets programmable in situ (pour le programme applicatif)
- EEPROM de 512 octets à usage général
- RAM statique de 1K octets (stockage des variables, pile…)
- Un convertisseur analogique Numérique 10 bits à 8 entrées multiplexées
- Un comparateur de tensions analogiques
- Liaisons séries synchrone (SPI) et asynchrone (SCI)
- 2 timers 8 bits (dont un utilisable en horloge temps réel moyennant un oscillateur externe)
- 1 timer 16 bits
- 4 ports d'entrées/sorties parallèles 8 bits
- 2 entrées d'interruptions externes et une entrée de reset
- un bus I^2C, un port JTAG pour la programmation et l'émulation
- …

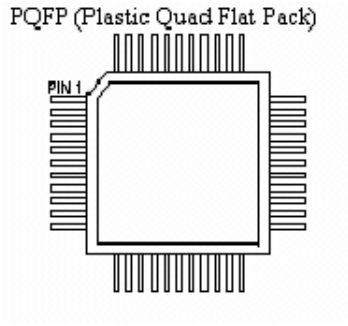
Architecture interne :

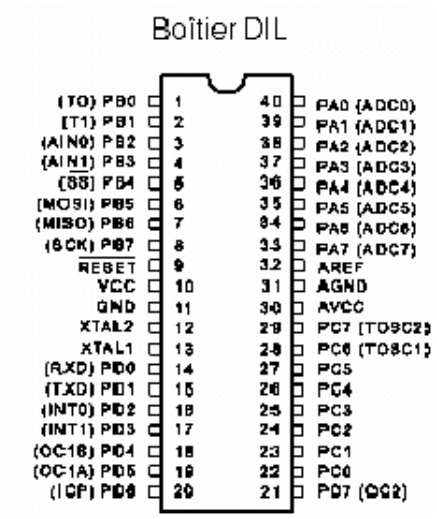


Encapsulation :

Le microcontrôleur est disponible dans plusieurs versions de boîtiers. Le boîtier de type MLF, le plus petit et non représenté ici, présente un encombrement de 7mm x 7mm.





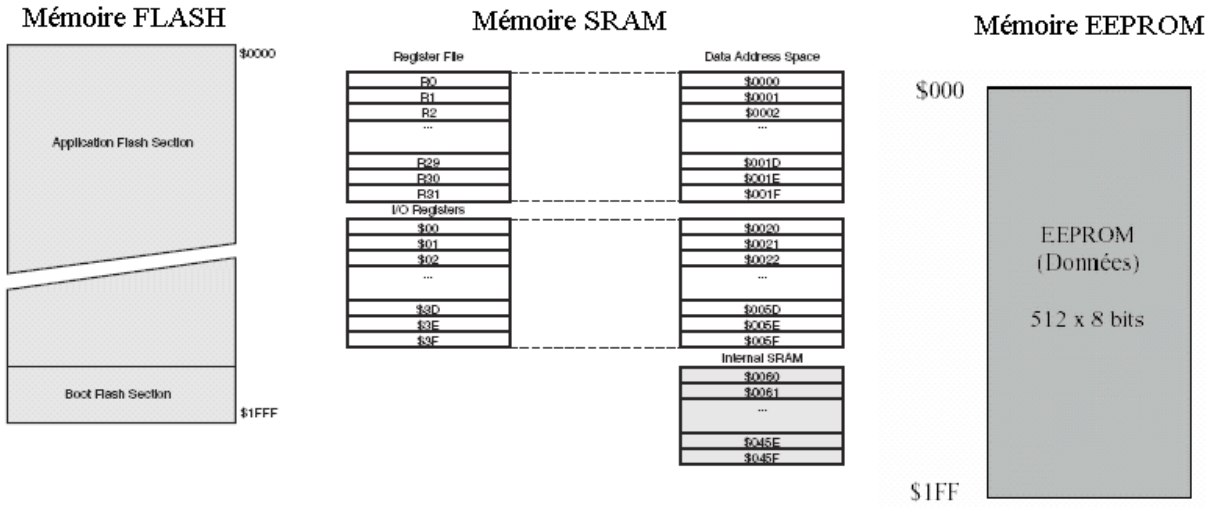


Mise en œuvre :

Dans sa configuration minimale le microcontrôleur ne nécessite qu'une alimentation, un circuit de « reset » et une horloge (horloge externe ou oscillateur à quartz). La plage d'alimentation va de 2.7V à 6V suivant les performances recherchées (économie d'énergie ou vitesse). La vitesse d'horloge peut aller jusqu'à 16 MHZ suivant la version du circuit (Atmega16 16PI).

3.2 Organisation de la mémoire

Les mémoires de données sont organisées en mots de 8 bits alors que la mémoire programme est organisée en mots de 16 bits. L'espace « données » est complétement séparé de l'espace « programme » (architecture de HARVARD), ce qui permet, grâce au fonctionnement en mode « pipe-line », une amélioration de la vitesse d'exécution des instructions.

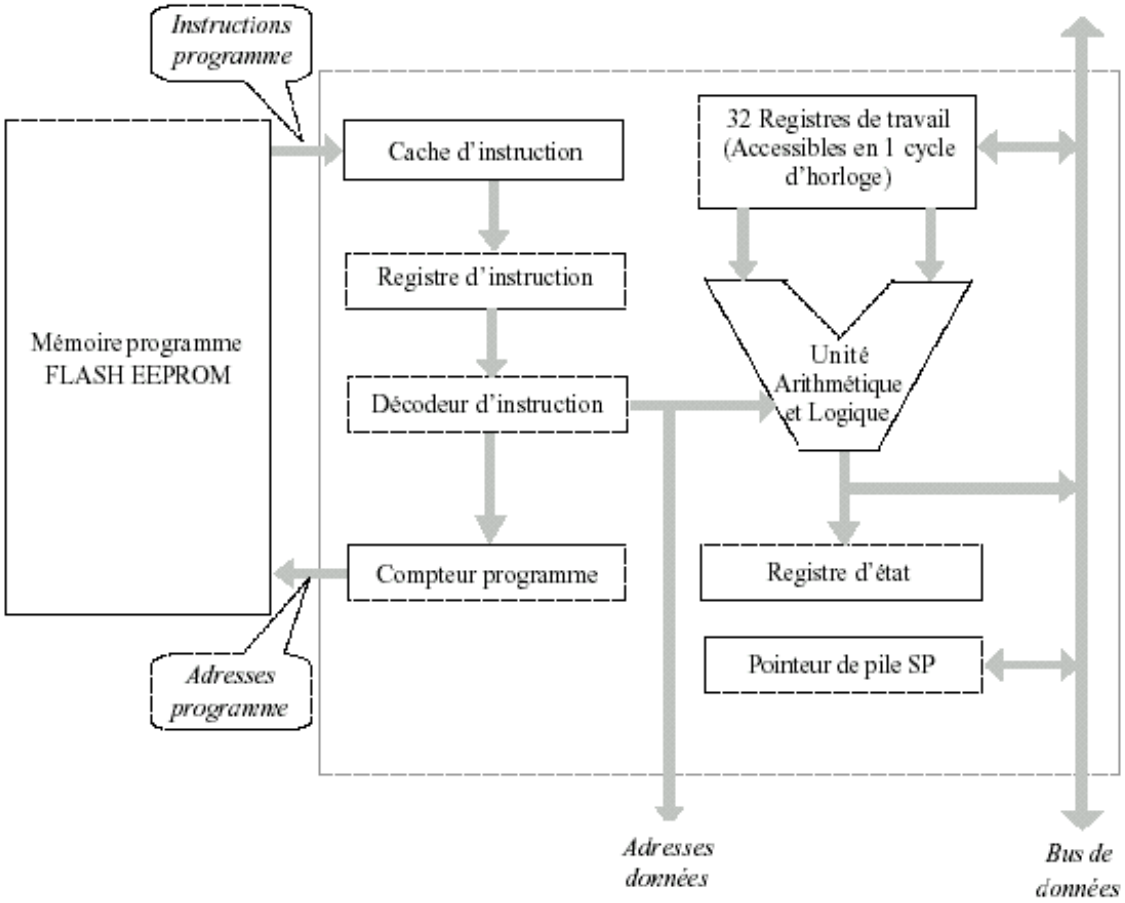


3.3 Architecture de la partie « microprocesseur »

Celle-ci est constituée de :

- un ensemble de 32 registres de travail de 8 bits

- une unité arithmétique et logique (ALU)
- un compteur programme de 13 bits (soit 8K mots adressables)
- un registre d'état
- un pointeur de pile
- un cache, un registre et un décodeur d'instruction



3.3.1) Les registres de travail

Ils sont au nombre de 32 (notés R0 à R31) et peuvent s'apparenter à des registres contenant chacun 8 bascules D et possédant une commande de sortie 3 états (haute impédance) .

Les registres :

- R26 et R27 concaténés constituent le registre d'index X
- R28 et R29 concaténés constituent le registre d'index Y
- R30 et R31 concaténés constituent le registre d'index Z

Les registres R26, R28, R30 constituent les poids faibles des index X, Y, Z.

3.3.2) L'unité arithmétique et logique

Elle effectue les opérations logiques et arithmétiques de base.

5.3.3) Le compteur de programme

C'est un compteur de 13 bits à chargement parallèle (prépositionnable). Il adresse la mémoire de programme organisée en 8192 mots de 16 bits (les instructions étant codées sur 16 bits). A chaque top d'une horloge interne (dont la fréquence est proportionnelle à celle du Quartz) le

compteur s'incrémente et vient pointer, s'il n'y a pas eu d'instruction de saut, l'adresse suivante de la mémoire de programme permettant ainsi le chargement et l'exécution de l'instruction qui s'y trouve. Les sauts de programme (ou sauts d'adresses) qui sont à l'initiative de l'instruction qui vient d'être exécutée sont rendus possibles grâce au chargement parallèle du compteur.

3.3.4) Le registre d'état SREG (Status Register)

Il s'agit d'un registre de 8 bits constitué par des bascules D qui peuvent être écrites ou lues séparément. Ce registre contient des « drapeaux » qui se positionnent ou non en fonction du résultat de l'instruction qui vient d'être exécutée (ils sont actifs à 1). La rubrique « Flags » du tableau des instructions indique quels sont les drapeaux affectés en fonction des instructions exécutées.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
I	T	H	S	V	N	Z	C

I : si à 1 => autorise les interruptions
T : Bit de stokage pour la copie d'un bit
H : Drapeau de signalisation de semi-retenue
S : S = N ⊕ V (Drapeau de signalisation de signe arithmétique)
V : Drapeau de signalisation de double complément de dépassement
N : à 1 si l'instruction exécutée génère un résultat négatif
Z : à 1 si l'instruction exécutée génère un résultat nul
C : à 1 si l'instruction exécutée a provoqué une retenue

Remarque : Lorsqu'une instruction est susceptible de modifier l'état d'un bit du registre SREG, celui-ci est remis à 0 avant l'exécution de l'instruction.

3.3.5) Le pointeur de pile SP (Stack Pointer)

C'est un registre de 16 bits (SPH et SPL) dont seulement les 11 bits de poids faibles sont utilisés (puisque la zone RAM va de $60 à $45F). Il pointe la prochaine adresse libre dans la pile (SRAM interne dont l'adresse finale est $45F). Son rôle est:

- d'adresser la pile pour la **sauvegarde** de l'adresse de retour du programme lors d'un appel à un sous-programme ou lors d'une interruption. (instruction CALL ou ICALL par exemple).
- d'adresser la pile pour la **restitution** de l'adresse de retour du programme lors du retour d' un sous-programme ou du retour d'une interruption. (instruction RET ou RETI).

Le pointeur de pile doit toujours être initialisé avec l'adresse haute de la RAM dès le début du programme. Dans le cas du Atmega16 cette adresse est $45F.

3.4 Reset, interruptions et modes de veille

3.4.1 Le Reset

Il a pour but la réinitialisation complète du microcontrôleur. Celle-ci se produit :

- à la mise sous tension du circuit tant que les alimentations ne sont pas stabilisées
- à partir d'un circuit « chien de garde »

- ou par action d'un opérateur (bouton poussoir).

Souvent la génération du **Reset** à la mise sous tension est confiée à des circuits intégrés spécialisés (ex : Motorola MC34064) et parfois à un simple réseau RC. Par construction (câblage interne) **le passage à 0 de la broche Reset force le compteur de programme à pointer l'adresse 0.** Le signal de Reset doit être maintenu actif pendant une durée T après la mise sous tension (cette durée est fournie par le constructeur du composant et est de 50ns minimum s'agissant du Atmega16).

3.4.2) Les interruptions

Il existe deux types d'interruptions :
- les interruptions externes (broches INT0 et INT1 et INT2 du circuit)
- les interruptions internes provoquées par les périphériques intégrés.

Le rôle des interruptions est de provoquer une suspension de l'exécution du programme en cours pour traiter un programme devenu de priorité supérieure.
Les interruptions peuvent être masquées par le programmeur.
Comme pour le **Reset**, l'apparition d'une interruption force le compteur de programme à une adresse déterminée fixée par le câblage interne du composant.

Le tableau suivant recense les différentes adresses d'interruptions du circuit Atmega16 :

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	Timer/Counter0 Overflow
11	$014	SPI, STC	Serial Transfer Complete
12	$016	USART, RXC	USART, Rx Complete
13	$018	USART, UDRE	USART Data Register Empty
14	$01A	USART, TXC	USART, Tx Complete
15	$01C	ADC	ADC Conversion Complete
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Analog Comparator
18	$022	TWI	Two-wire Serial Interface
19	$024	INT2	External Interrupt Request 2
20	$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	$028	SPM_RDY	Store Program Memory Ready

Mécanismes de traitement d'une interruption :
Lors d'une interruption le processeur exécute la séquence suivante :

- le bit I du registre d'état (SREG) est mis à 0 (inhibition des autres sources d'interruption)
- l'exécution du programme en cours est interrompue
- le contenu du PC est sauvegardé dans la pile
- le PC est chargé avec la valeur de l'adresse d'interruption
- l'instruction de saut à l'adresse de traitement de l'interruption est exécutée
- le programme de traitement d'IT est exécuté
- à la fin du programme, l'instruction **RETI** restaure le contenu du PC
- le bit I du registre d'état (SREG) est mis à 1 (validation des autres sources d'interruption).

Remarque : lors d'une interruption, **le processeur ne sauvegarde pas le contexte** (état des registres de travail et registre d'état SREG). Il est donc important de sauvegarder au moins le registre d'état en début du sous-programme d'interruption et de le restituer à la fin en utilisant les instructions **PUSH** et **POP**.

3.4.3) Les modes veille

Ce sont des modes qui permettent d'économiser de l'énergie (utilisés dans des applications portables).

3.5) Le jeu d'instructions

l'Atmega16 dispose d'un jeu de 131 instructions codées en majorité sur 16 bits, ce qui explique l'organisation de la mémoire programme en mots de 16 bits. La plupart de ces instructions sont exécutées en un cycle d'horloge.

Nota :

Ayant opté pour une programmation du circuit en langage C, ces instructions seront peu abordées dans le cadre de ce cours.

3.7) Les périphériques

3.7.1 Les ports d'interfaces parallèles

L'Atmega16 dispose de 4 ports d'interfaces de 8 bits chacun accessibles au travers de 3 registres de 8 bits. Cependant certaines broches de ces ports peuvent être partagées avec d'autres ressources internes. **C'est la phase d'initialisation qui permet de déterminer quelles seront les ressources mises en œuvre**.

Chaque port dispose de :

- un registre de direction **DDRx** qui spécifie si le port est utilisé en entrée ou en sortie (un 1 dans un de ses bits signifie que la broche correspondante est en sortie).
- Un registre de données entrantes **PINx** : l'état des bits de ce registre correspond aux niveaux logiques sur les broches programmées en entrée.
- Un registre de données sortantes ou d'options **PORTx** : en sortie il fixe le niveau logique sur la broche correspondante. Si la broche a été configurée en entrée, le port permet de valider ou non la résistance de pull-up interne.

Les registres PINx permettent au microcontrôleur de lire l'état logique des broches. Les registres PORTx permettent au microcontrôleur de forcer à un niveau logique les broches des ports (commande d'actionneurs « tout ou rien »).

Description des registres :

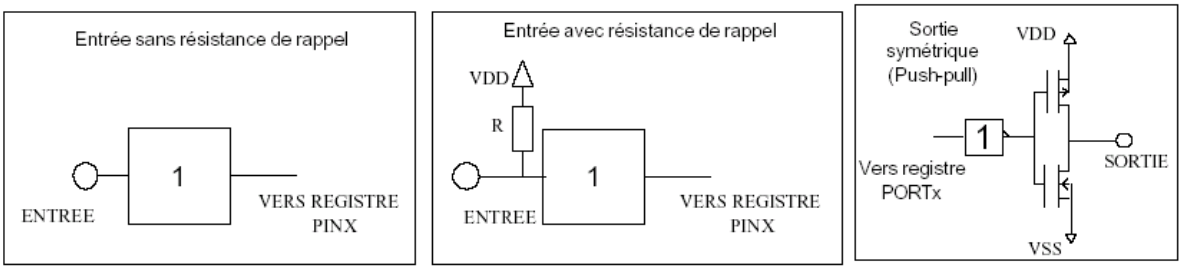
	Registre de Direction et adresse	Registre de données entrantes et adresse	Registre de données sortantes et adresse	Nbre de bits
Port A	DDRA $1A ($3A)	PINA $19 ($39)	PORTA $1B ($3B)	8
Port B	DDRB $17 ($37)	PINB $16 ($36)	PORTB $18 ($38)	8
Port C	DDRC $14 ($34)	PINC $13 ($33)	PORTC $15 ($35)	8
Port D	DDRD $11 ($31)	PIND $10 ($30)	PORTD $12 ($32)	8

Remarque : La 1ère adresse correspond à l'adresse à utiliser avec les instructions OUT et IN, la 2ème adresse donnée entre parenthèses correspond à l'adresse réelle du registre. Cette 2ème adresse est celle à utiliser lors des accès au registre par les instructions d'accès en RAM (STS par exemple).

Configuration des broches :

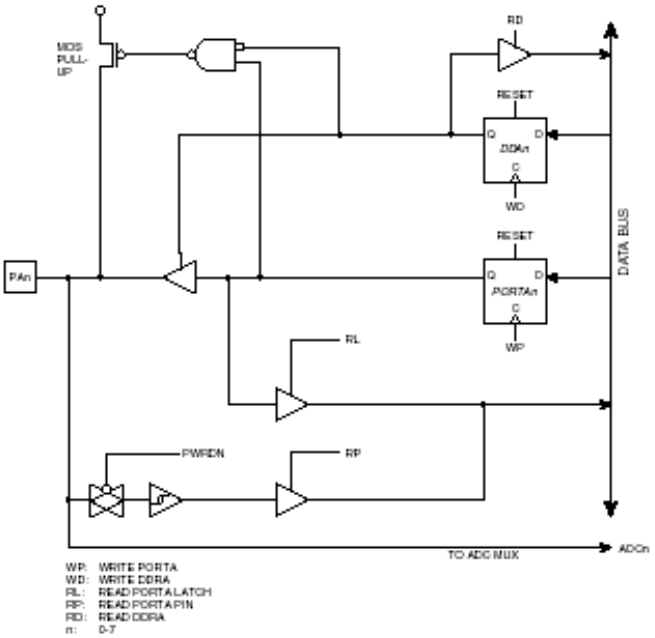
Bit du DDRx	Bit du PORTx	Bit du PINx	Direction	Configuration
0	0	X_{IN}	Entrée	Sans Pull-Up. (X_{IN} est l'image du niveau présent sur la broche)
0	1	X_{IN}	Entrée	Avec Pull-Up. (X_{IN} est l'image du niveau présent sur la broche)
1	X	X	Sortie	Sortie symétrique Push-Pull. X est le niveau délivré par la broche.

La lecture du registre PINx donne l'état des broches du PORTx orientées en entrée. L'écriture dans le registre PORTx fixe l'état des broches configurées en sortie. (PINx n'est accessible qu'en lecture)



La mise en œuvre des ports parallèles nécessite en premier **la programmation du registre DDRx** pour définir si les broches du circuit sont en entrée ou en sortie.

Schéma électrique associé à une broche du port A



Exercice :
A partir du programme ci-dessous, vérifier sur le schéma électrique l'adéquation entre les valeurs des registres et les fonctions réalisées.

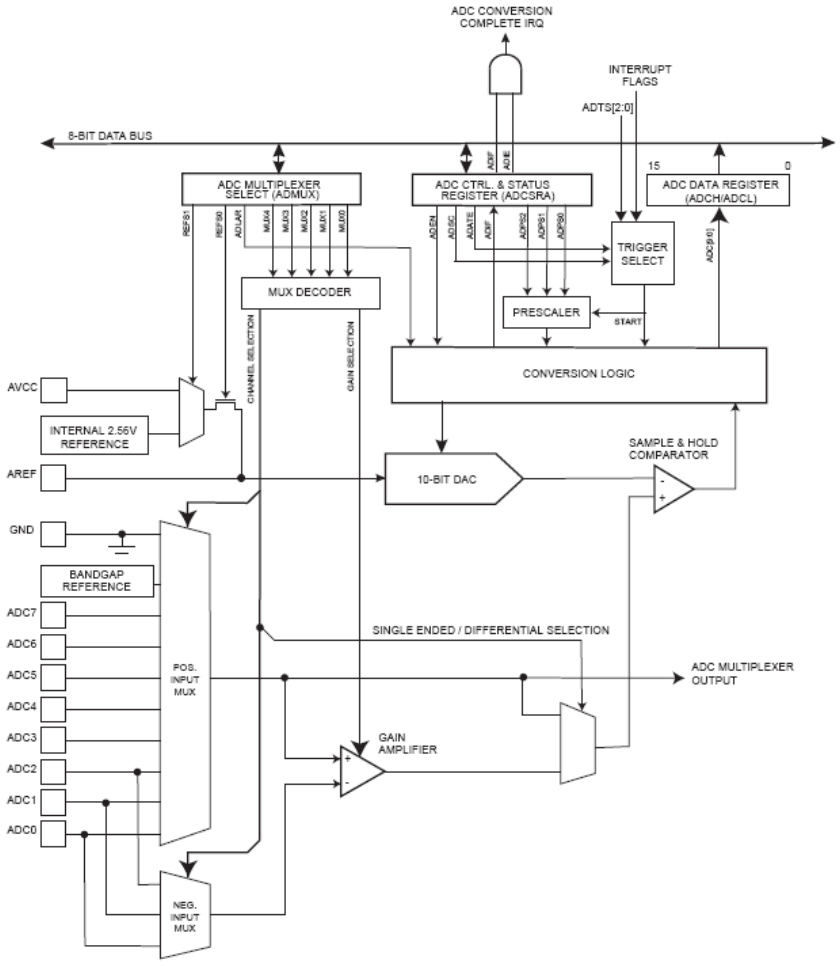
Exemple de programme d'initialisation des ports B et D:
Programme en langage C mettant en œuvre le compilateur AVR CodeVision : attention respecter les majuscules !

```
#include <mega16.h>          // (fichier de définition des mots clés)
DDRB=0xFF ;                  // portB en sortie
DDRD=0 ;                     // portD en entrée
PORTD=0xFF ;                 // pullup actifs
```

3.7.3) Le convertisseur analogique-numérique

L'Atmega16 intègre un Convertisseur Analogique Numérique possédant les caractéristiques suivantes :
- Résolution : 10 bits.
- Conversion par approximations successives.
- 8 entrées unipolaires (ADC0..ADC7) par un multiplexeur analogique 8 voies, 7 différentielles, 2 avec gain programmable.

Architecture :



L'expression de la valeur numérique N délivrée par le CAN en fonction de la tension à convertir VE est donnée par la relation suivante :

$$N = \frac{V_E}{V_{AREF}} \times 1024$$

Avec V_{AREF} = Tension présente sur l'entrée de référence AREF

Le temps de conversion TCONV du CAN est proportionnel à la valeur de l'horloge de conversion. L'horloge de conversion doit être comprise entre 50KHz et 200KHz (typiquement 100KHz) pour obtenir des conversions 10 bits précises.

$$T_{CONV} = 13 \times CK_{CNV}$$

On obtient donc un temps de conversion compris entre 65µS et 260µS

REMARQUES :

- La 1ère conversion qui suit la validation de la fonction CAN est plus longue, en effet le CAN procède à certains ajustements internes. Cette 1ère conversion nécessite 25 cycles d'horloge CAN.
- Si on se limite à une résolution de 8 bits (en ignorant les 2 bits de poids faibles) la fréquence de l'horloge de conversion peut être choisie jusqu'à 1MHz. (Voire 2MHz si résolution utile 6 bits)

Registres du CAN :

Le CAN est doté de 3 registres : **ADMUX, ADCSR, ADC**

Le registre ADMUX :

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Les Bits REFS1 et REFS0 :

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Le bit ADLAR : permet de justifier à gauche le résultat de la conversion si positionné à 1.

Les bits MUX4 à MUX0 :

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0			
00001	ADC1			
00010	ADC2			
00011	ADC3	N/A		
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000		ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010	N/A	ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

Le registre ADCSRA :

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADEN : Ce bit met en fonction le CAN quand il est positionné à 1. Si ce bit est à 0 le CAN est désactivé, cela permet de réduire la consommation.

- ADSC : La mise à 1 de ce bit lance la conversion de la voie sélectionnée. (Ce bit retourne à 0 automatiquement en fin de conversion).

- ADATE : Ce bit lorsqu'il est à 1 permet de déclencher la conversion sur l'apparition d'un signal de trig. La source de ce signal est déterminée par les bits ADTS dans le registre SFIOR.

- ADIF : Ce bit passe à 1 quand la conversion en cours est terminée et que le résultat est disponible. Ce bit est remis automatiquement à 0, lorsque l'interruption associée au CAN est générée. Si la génération de l'interruption du CAN n'est pas validée, l'utilisateur doit remettre ce bit à 0 (en y écrivant un 1) avant de relancer une conversion.

- ADIE : Ce bit valide la génération lorsqu'il est à 1, de l'interruption CAN. L'interruption CAN est générée lorsque ADIE et ADSC = 1 (Fin de conversion).

- Les bits ADPS2..ADPS0 permettent de choisir le facteur de pré division de l'horloge système CK afin de générer la fréquence d'horloge de conversion désirée.

ADPS2	ADPS1	ADPS0	Facteur de pré division
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADC : Registre de conversion : Le résultat de la conversion sur 10 bits est placé dans les 2 registres suivants ADCH et ADCL :

ADLAR = 0

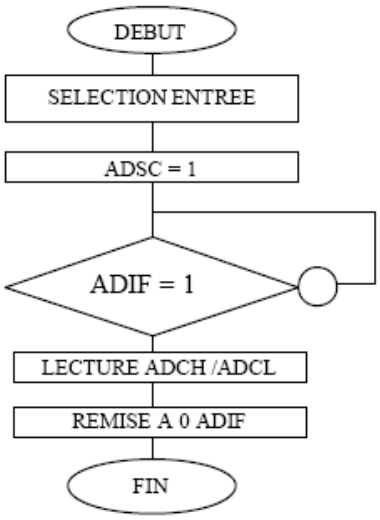
Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Exemple d'utilisation du CAN :

Nota : Avant d'exécuter les opérations de l'organigramme ci-contre, il convient de mettre en fonction le CAN en positionnant le bit ADEN à 1. ***De plus la broche du PORT A correspondant à l'entrée ADCx à convertir doit être configurée en entrée.***

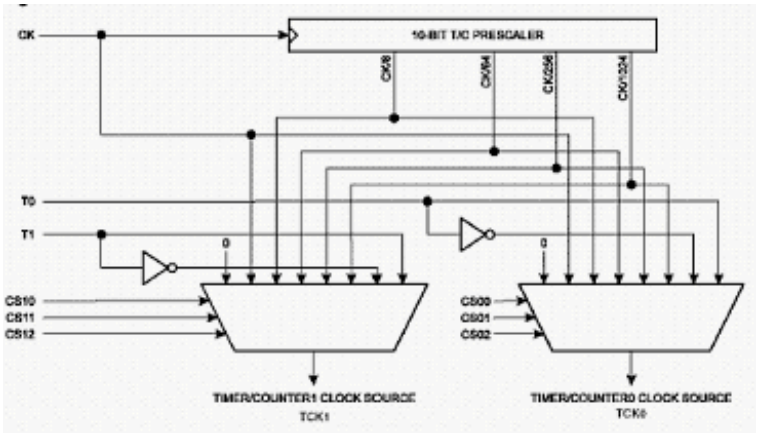


Exemple de programme en C d'utilisation du CAN :

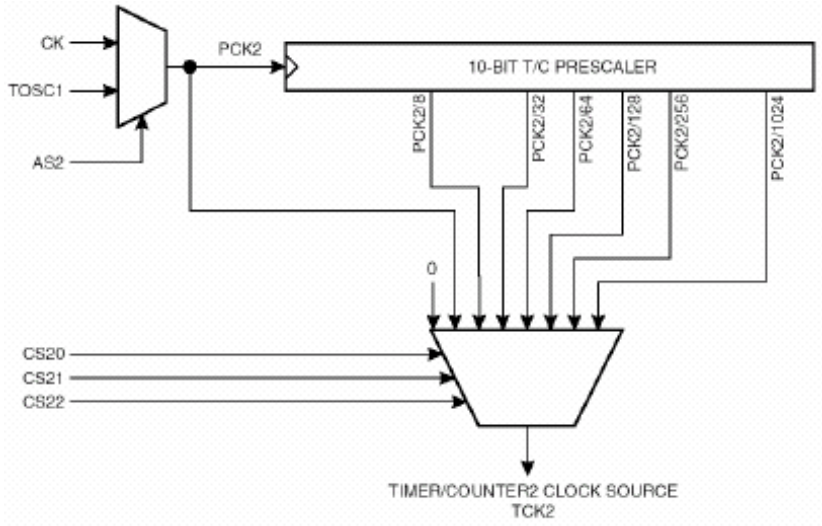
```
#include <mega16.h>
#include <delay.h>
#define ADC_VREF_TYPE 0x20

unsigned char read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;          // programmation voie utilisée
ADCSRA|=0x40;                           // Start the AD conversion
while ((ADCSRA & 0x10)==0);             // Wait for the AD conversion to complete
ADCSRA|=0x10;                           // RAZ bit ADIF du CAN
return ADCH;
}
 char a;
void main(void)
{
PORTA=0x00;
DDRA=0x00;
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x85;

while (1)
     {
      a=read_adc(0) ;                   //conversion voie 0
      PORTB=~a;                         //affichage sur le portb
      delay_ms(100) ;
     };
}
```

3.7.4) Les compteurs

L'Atmega16 dispose de deux compteurs 8 bits et d'un compteur 16 bits. Ils peuvent être utilisés pour générer des temporisations précises, compter des évènements, mesurer des périodes ou fréquences de signaux, générer des signaux PWM pour le contrôle en vitesse de moteurs à courant continu par exemple …
Les compteurs 0 et 1 évoluent sous contrôle d'une horloge externe ou interne. Dans ce dernier cas l'horloge est un dérivé de l'horloge du quartz comme représenté sur la figure ci-dessous :



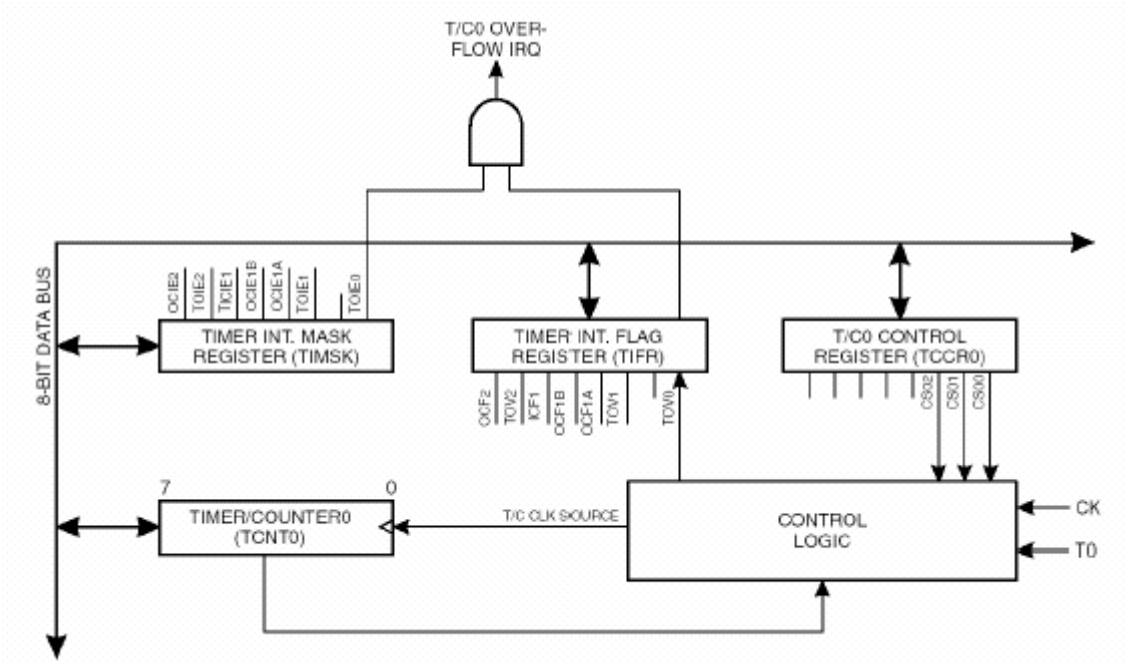
Pour le compteur 2 la génération de l'horloge correspond au synoptique ci-dessous :



La chaîne de pré-division contenue dans le microcontrôleur permet de fournir 3 horloges :

TCK0 : Compteur libre du timer 0 : Choix par les bits CS02, CS01, CS00 du registre TCCR0.
TCK1 : Compteur libre du timer 1 : Choix par les bits CS12, CS11, CS10 du registre TCCR1B.
TCK2 : Compteur libre du timer 2 : Choix par les bits CS22, CS21, CS20 du registre TCCR2.

3.7.4.1) Architecture du compteur 0 :



Le compteur 0 est contrôlé par certains bits de 5 registres : TCNT0, TCCR0, OCR0, TIMSK, TIFR.

Le registre **TCNT0** :

TCNT0 : Compteur du timer 0 : Ce registre est accessible en lecture et en écriture.

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$32 ($52)	MSB							LSB

Le registre **TCCR0** :

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Les bits CS02, CS01, CS00 permettent de sélectionner la source d'horloge du compteur TCNT0 ou de le stopper :

CS02	CS01	CS00	Source d'horloge du compteur
0	0	0	Aucune : Le compteur est stoppé
0	0	1	Horloge système (CK)
0	1	0	Horloge système (CK) / 8
0	1	1	Horloge système (CK) / 64
1	0	0	Horloge système (CK) / 256
1	0	1	Horloge système (CK) / 1024
1	1	0	Broche T0, active sur front descendant
1	1	1	Broche T0, active sur front montant

Le registre **OCR0 :**

Il permet de stocker une valeur de comparaison. Lorsque celle-ci devient identique au contenu de TCNT0, il est possible de générer une interruption. Cette possibilité est très utile pour générer des temporisations précises.

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Le registre **TIFR :**

C'est un registre d'état des trois compteurs.
Le bit 1 OCF0 indique une comparaison réussie entre TCNT0 et OCR0.
Le bit 0 TOV0 indique un débordement du compteur (passage de 255 à 0).

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

La remise à 0 de ces 2 bits s'effectue par :
- *L'écriture par le programme d'un « 1 » dans le drapeau à remettre à 0.*
- *Le microcontrôleur lui même si l'interruption correspondante au drapeau à été générée.*

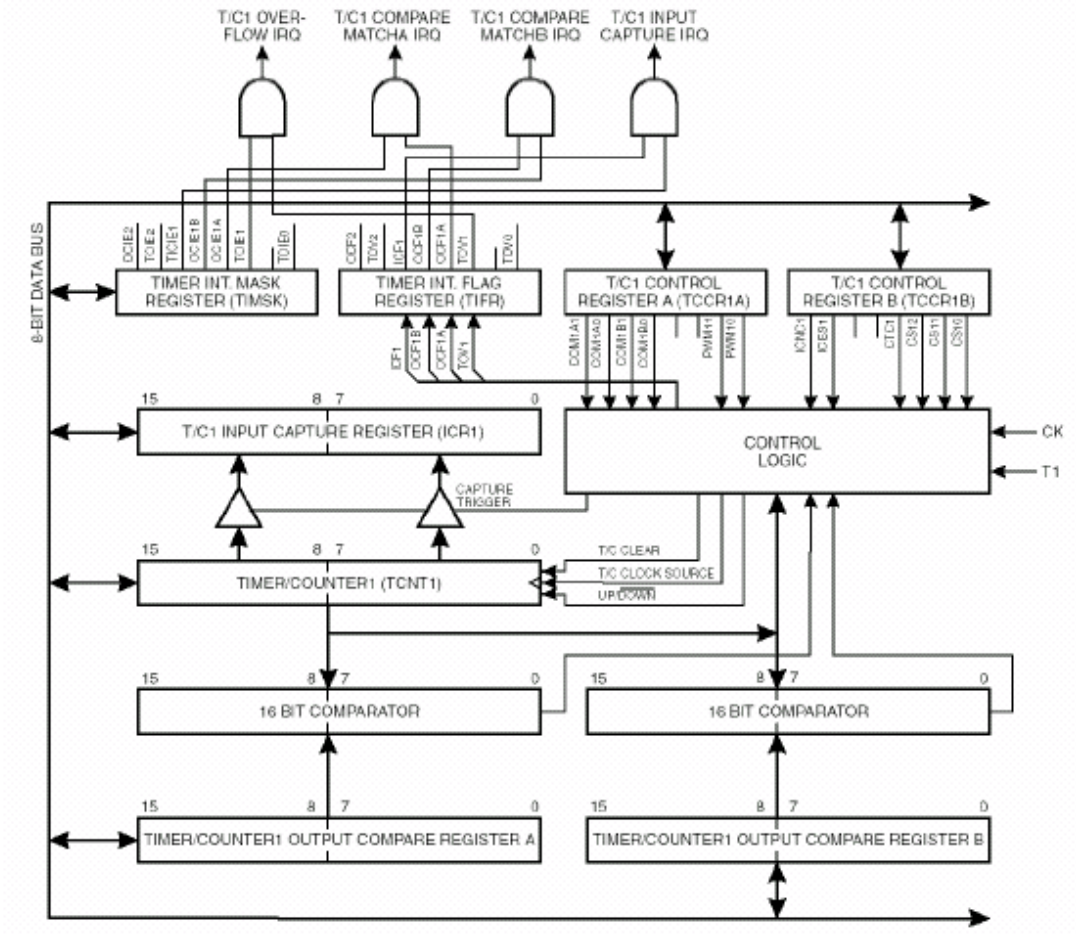
Le registre **TIMSK** :

C'est un registre de validation des interruptions des compteurs. Il permet de valider la génération d'interruptions par le compteur. (A condition que le bit I du registre SREG soit à 1).

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

3.7.4.2) Architecture du compteur 1 :

Synoptique :



Le Timer 1 comporte un compteur 16 bits , 2 registres de comparaison qui peuvent agir sur 2 sorties et un registre de capture qui fonctionne avec l'entrée ICP. Il est contrôlé par les registres TCNT1, TCCR1A, TCCR1B, OCR1A, OCR1B, TIFR, TIMSK et ICR1

<u>Fonctionnement en mode normal (mode PWM inactivé) :</u>
Le registre de contrôle **TCCR1A** :

TCCR1A : 1er registre de contrôle du timer 1.

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$2F ($4F)	COM1A1	COM1A0	COM1B1	COM1B0	-------	--------	PWM11	PWM10

COM1X1	COM1X0	Action programmée sur OC1X
0	0	Timer 1 déconnecté (pas d'action)
0	1	Complémentation du niveau
1	0	Mise à 0
1	1	Mise à 1

Les bits COMX1 et COMX0 permettent de définir l'action qui doit se produire sur la sortie OC1X lorsque les valeurs présentes dans le registre de comparaison OCR1X et le compteur TCNT1 sont égales. Voir table ci-contre :

« X » est à remplacer par A pour la sortie OC1A et par B pour la sortie OC1B.

Le registre **TCCR1B** :

TCCR1B : 2eme registre de contrôle du timer 1.

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$2E ($4E)	ICNC1	ICES1	-------	-------	CTC1	CS12	CS11	CS10

Les bits ICNC1 et ICES1 définissent le mode de fonctionnement de l'entrée de capture : ICP.
- ICNC1 : Ce bit lorsqu'il est positionné à 1, met en fonction le réducteur de bruit qui permet

d'éviter la prise en compte de parasites comme fronts actifs.

- ICES1 : Sélection du type de front actif. Si à 1 : front montant, Si à 0 : front descendant. Lorsqu'un front actif se produit sur l'entrée ICP, le contenu du compteur TCNT1 est transféré dans le registre de capture ICR1.

- CTC1 définit le mode de fonctionnement du compteur TCNT1 :
0 : Comptage en continu jusqu'à la capacité maximale ($0000 - $FFFF), puis le cycle recommence.
1 : Comptage en continu jusqu'à la valeur contenue dans le registre de comparaison OCR1A ($0000 - OCR1A), puis, après la comparaison valide (TCNT1 = OCR1A), le cycle recommence.
- CS12, CS11, CS10 : permettent de sélectionner la source d'horloge du compteur TCNT1 ou de le stopper. Voir table suivante :

CS12	CS11	CS10	Source d'horloge du compteur
0	0	0	Aucune : Le compteur est stoppé
0	0	1	Horloge système (CK)
0	1	0	Horloge système (CK) / 8
0	1	1	Horloge système (CK) / 64
1	0	0	Horloge système (CK) / 256
1	0	1	Horloge système (CK) / 1024
1	1	0	Broche T1, active sur front descendant
1	1	1	Broche T1, active sur front montant

Les registres de comparaison/capture et le compteur du timer 1 :

Le registre TCNT1 :

TCNT1 : Compteur libre du timer 1 (16 bits), accessible en lecture et en écriture

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
$2D ($4D)	MSB								TCNT1H
$2C ($4C)								LSB	TCNT1L

Les registres de comparaison OCR1A et OCR1B :

OCR1A et OCR1B : Registres de comparaison du timer 1 (16 bits), accessibles en lecture et en écriture

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
$2B ($4B)	MSB								OCR1AH
$2A ($4A)								LSB	OCR1AL
Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
$29 ($49)	MSB								OCR1BH
$28 ($48)								LSB	OCR1BL

Les registres de comparaison contiennent la valeur qui est comparée en permanence avec le contenu du compteur libre TCNT1. Lorsque la valeur contenue dans le registre OCR1x est égale à celle présente dans TCNT1 l'action programmée sur OC1x est exécutée et le bit OCF1x du registre TIFR est positionné à 1.

Le registre de capture ICR1 :

ICR1 : Registre de capture du timer 1 (16 bits), accessible en lecture

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
$27 ($47)	MSB								ICR1H
$26 ($46)								LSB	ICR1L

Lors d'un front actif sur ICP, le contenu du compteur libre **TCNT1** est recopié dans le registre ICR1 et le bit ICF1 du registre **TIFR** est positionné à 1.

Les registres **TIFR** et **TIMSK** d'état et de validation des interruptions :

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TOV1 : Ce bit passe à 1 lors du débordement du compteur du Timer1 , TCNT1.
- OCF1A : Ce bit passe à 1 lorsque TCNT1 = OCR1A (Comparaison A réussie).
- OCF1B : Ce bit passe à 1 lorsque TCNT1 = OCR1B (Comparaison B réussie).
- ICF1 : Ce bit passe à 1 lorsqu'un front actif se produit sur l'entrée ICP.

La remise à 0 de ces drapeaux s'effectue par :
- L'écriture par le programme d'un « 1 » dans le drapeau à remettre à 0.
- Le microcontrôleur lui même si l'interruption correspondante au drapeau à été générée.

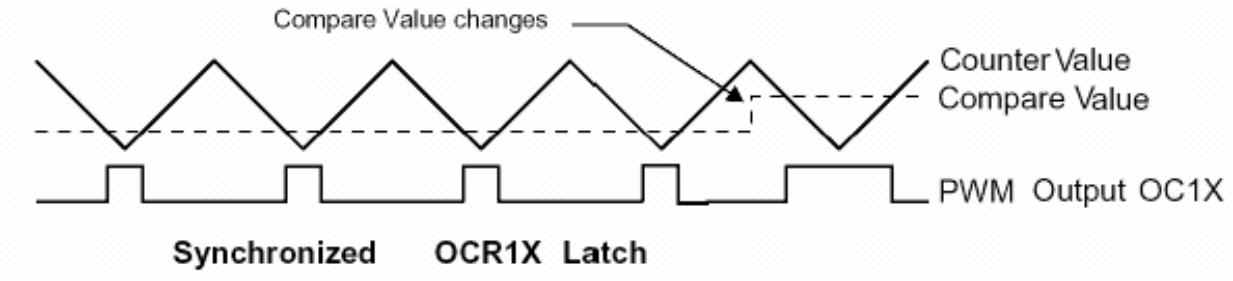
TIMSK :Registre de validation des interruptions des timers. Il permet de valider la génération d'interruptions par le timer .(A condition que le bit I du registre SREG soit à 1)

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- TOIE1 : Si à 1, l'interruption N°9 ($0008) est générée lors du débordement du compteur du Timer 1 : TCNT1.
- OCIE1B : Si à 1 , l'interruption N°8 ($0007) est générée, lors d'une comparaison réussie entre TCNT1 et OCR1B.
- OCIE1A : Si à 1 , l'interruption N°7 ($0006) est générée, lors d'une comparaison réussie entre TCNT1 et OCR1A.
- TICIE1 : Si à 1 , l'interruption N°6 ($0005) est générée, lors de la détection d'un front actif sur l'entrée ICP.

REGISTRES ET FONCTIONNEMENT EN MODE PWM :

Présentation du mode PWM :
En mode PWM le Timer1 est configuré pour produire un signal PWM (Modulation de largeur d'impulsion) sur les sorties OC1A et OC1B. Dans ce mode certains bits du registre TCCR1A, voient leur rôle modifié. Ce mode permet de réaliser une PWM sur 8, 9 ou 10 bits, ce qui définit la capacité maximum du compteur TCNT1 (255, 511 ou 1023). Le compteur TCNT1 compte de 0 à sa valeur maximale puis décompte jusqu'à 0 et le cycle se renouvelle en permanence. L'état de la sortie OC1x dépend du résultat de la comparaison entre OCR1x et TCNT1.
Dans l'exemple suivant, on a :
- Si OCR1x > TCNT1 alors OC1x = 1
- Si OCR1x < TCNT1 alors OC1x = 0
La valeur placée dans OCR1x fixe donc le rapport cyclique du signal PWM généré :



Configuration en mode PWM :

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$2F ($4F)	COM1A1	COM1A0	COM1B1	COM1B0	-------	--------	PWM11	PWM10

Les bits PWM11 et PWM10 permettent de choisir le type de PWM (8, 9 ou 10 bits) ou le mode normal du Timer1

PWM11	PWM10	Mode de fonctionnement du timer1
0	0	Mode normal (Non PWM)
0	1	PWM 8 bits
1	0	PWM 9 bits
1	1	PWM 10 bits

On obtient alors les valeurs de comptage maximales indiquées dans la table ci-contre et les fréquences correspondantes du signal PWM généré. (Avec FTCK1 = Fréquence horlogetimer)

Résolution PWM	Valeur maxi compteur	Fréquence
8 bits	$00FF (255)	F_{TCK1}/510
9 bits	$01FF (511)	F_{TCK1}/1022
10 bits	$03FF (1023)	F_{TCK1}/2046

Les bits COM1x1 et COM1x0 permettent de sélectionner la polarité du signal PWM généré sur les sorties OC1x :

COM1X1	COM1X0	Type de PWM
0	0	PWM non active
0	1	PWM non active
1	0	PWM normale
1	1	PWM complémentée

Remarques :

- Les bits du registre TCCR1B conservent le rôle qu'ils ont dans le mode normal, sauf le bit CTC1 qui est ignoré en mode PWM.
- Les bits des registres TIFR et TIMSK fonctionnent de la même manière qu'en mode normal, on peut donc toujours détecter les comparaisons réussies et utiliser les interruptions générées.

Exemple de mise en œuvre du timer 1 :

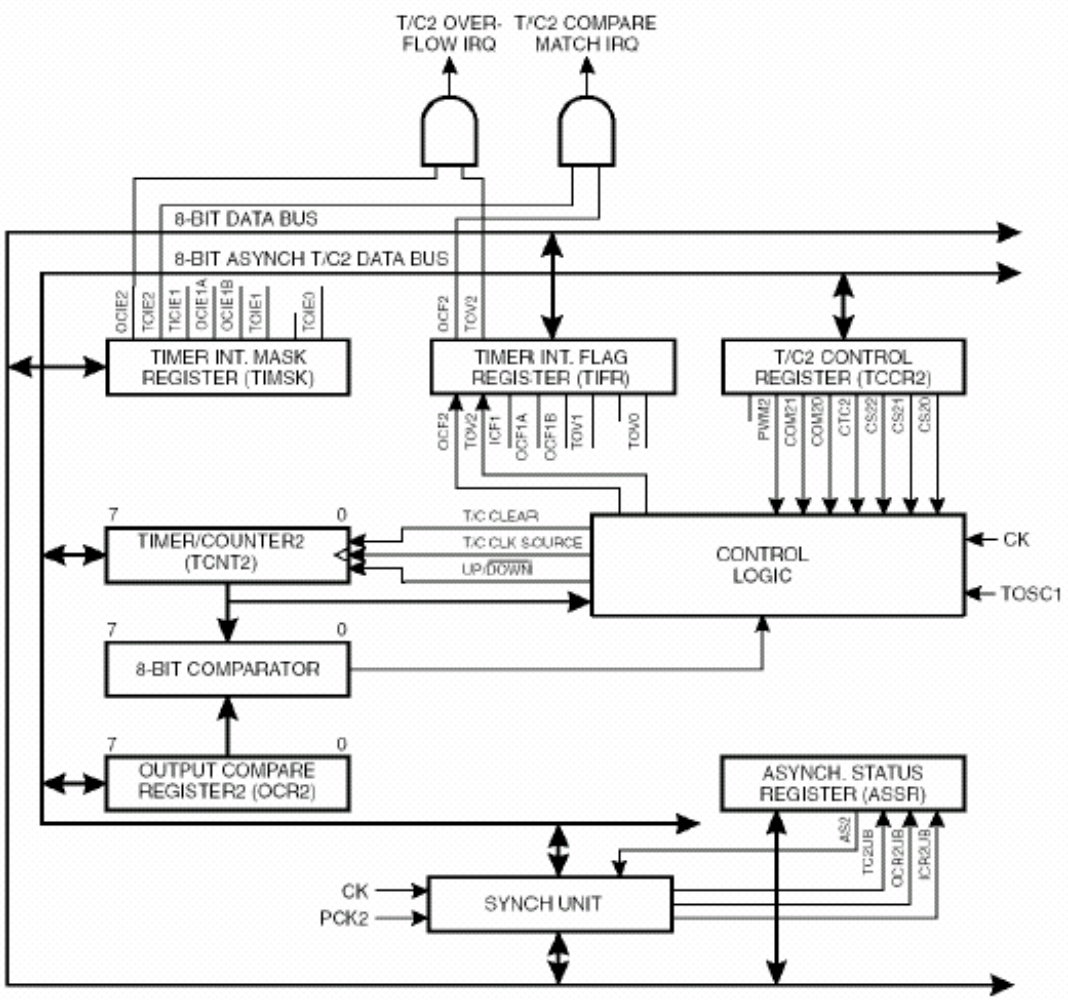
Le programme ci-dessous initialise le timer 1 pour générer des interruptions toutes les 10 ms.

Nota : lignes commençant par // correspondent à des commentaires

```
// Timer/Counter 1 initialization
// Clock source: System Clock (4 MHZ)
// Clock value: 62,500 kHz (division par 64)
// Mode: CTC top = OCR1A compte jusqu'à OCR1A puis revient à 0
// OC1A output: Discon.
// OC1B output: Discon.
```

// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On comparaison sur A activée
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x0B; // mode CTC et division par 64 de l'horloge => T=16µs
OCR1AH=0x02; // $271= $(625)_{10}$
OCR1AL=0x71; //625 x 16 µs= 10 ms
TIMSK=0x10; // valide l'interruption associée

Le Timer 2 :
Architecture :



Le Timer 2 comporte un compteur 8 bits et un registre de comparaison qui peut agir sur la sortie OC2.

REGISTRES ET FONCTIONNEMENT EN MODE NORMAL (NON PWM) :
Les registres de commande :
ASSR :Registre de contrôle et de statut du timer 2 en mode asynchrone.

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$22 ($42)	-------	-------	-------	-------	AS2	TCN2UB	OCR2UB	TCR2UB

Le bit AS2 permet de sélectionner la source d'horloge du compteur du Timer 2.

- Si AS2 = 0 : Horloge système CK.
- Si AS2 = 1, Les broches PC6 et PC7 forment un oscillateur et un quartz doit être connecté entre ces 2 broches. Le Timer 2 utilise alors le signal d'horloge fourni par cet oscillateur.

Si AS2 = 1, l'oscillateur externe est sélectionné et dans ce cas le fonctionnement du timer 2 n'est plus synchrone avec la CPU du microcontrôleur. L'écriture d'une donnée dans un registre du timer se fait en 2 cycles : Transfert dans un tampon puis transfert vers le registre visé. Bien que ces 2 opérations soient transparentes pour l'utilisateur (Il suffit d'écrire à l'adresse du registre visé), ces opérations prennent du temps.

On dispose donc de drapeaux d'état indiquant si l'opération d'écriture en cours est terminée :
- TCN2UB : Passe à 1 pendant la durée d'une écriture dans le compteur TCNT2.
- OCR2UB : Passe à 1 pendant la durée d'une écriture dans le compteur OCR2.
- TCRUB : Passe à 1 pendant la durée d'une écriture dans le compteur TCCR2.

Le registre de contrôle TCCR2 du Timer 2 :

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$25 ($45)	--------	PWM2	COM21	COM20	CTC2	CS22	CS21	CS20

COM21	COM20	Action programmée sur OC2
0	0	Timer 2 déconnecté (pas d'action)
0	1	Complémentation du niveau
1	0	Mise à 0
1	1	Mise à 1

Les bits COM21 et COM20 permettent de définir l'action qui doit se produire sur la sortie OC2 lorsque les valeurs présentes dans le registre de comparaison OCR2 et le compteur TCNT2 sont égales.

Le bit CTC2 définit le mode de fonctionnement du compteur TCNT2 :
- 0 : Comptage en continu jusqu'à la capacité maximale ($00 - $FF), puis le cycle recommence.
- 1 : Comptage en continu jusqu'à la valeur contenue dans le registre de comparaison OCR2 ($00 – OCR2), puis, après la comparaison valide (TCNT2 = OCR2), le cycle recommence.

CS02	CS21	CS20	Source d'horloge du compteur
0	0	0	Aucune : Le compteur est stoppé
0	0	1	Horloge timer 2 (PCK2)
0	1	0	Horloge timer 2 (PCK2) / 8
0	1	1	Horloge timer 2 (PCK2) / 32
1	0	0	Horloge timer 2 (PCK2) / 64
1	0	1	Horloge timer 2 (PCK2) / 128
1	1	0	Horloge timer 2 (PCK2) / 256
1	1	1	Horloge timer 2 (PCK2) / 1024

Les bits CS22, CS21, CS20 : permettent de sélectionner le facteur de division de l'horloge du compteur TCNT2 ou de le stopper :

Les registres de comparaison et de compteur du timer2 :

TCNT2 : Compteur du timer 2 : Ce registre est accessible en lecture et en écriture

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$24 ($44)	MSB							LSB

OCR2 : Registre de comparaison du timer 2, accessible en lecture et en écriture

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$24 ($44)	MSB							LSB

Le registre de comparaison contient la valeur qui est comparée en permanence avec le contenu du compteur libre TCNT2. Lorsque la valeur contenue dans le registre OCR2 est égale à celle présente dans TCNT2 l'action programmée sur OC2 est exécutée et le bit OCF2 du registre TIFR est positionné à 1.

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

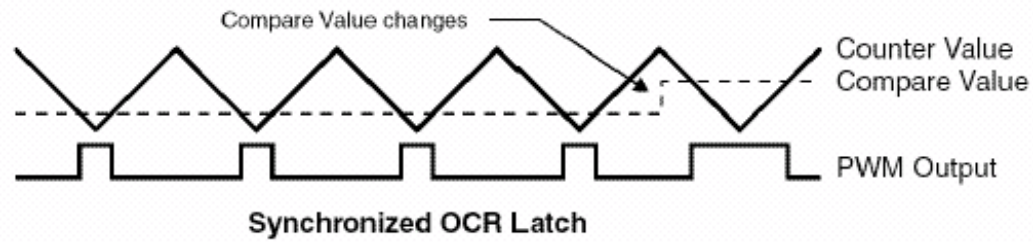
TOV2 : Ce bit passe à 1 lors du débordement du compteur du Timer2 , TCNT2.
- OCF2 : Ce bit passe à 1 lorsque TCNT2 = OCR2 (Comparaison réussie).
La remise à 0 de ces drapeaux s'effectue par :
- *L'écriture par le programme d'un « 1 » dans le drapeau à remettre à 0.*
- *Le microcontrôleur lui même si l'interruption correspondante au drapeau à été générée.*

TIMSK :Registre de validation des interruptions des timers, permet de valider la génération d'interruptions par le timer .(A condition que le bit I du registre SREG soit à 1).

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- TOIE2 : Si à 1, l'interruption N°5 ($0004) est générée lors du débordement du compteur du Timer 2 : TCNT2.
- OCIE2 : Si à 1 , l'interruption N°4 ($0003) est générée, lors d'une comparaison réussie entre TCNT2 et OCR2.

REGISTRES ET FONCTIONNEMENT EN MODE (PWM) :
Présentation du mode PWM :
Le fonctionnement du Timer2 en mode PWM est identique a celui du Timer1.
Le signal PWM est disponible sur la broche OC2.



Configuration du mode PWM :

7	6	5	4	3	2	1	0	
FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	TCCR2
W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

Mode	WGM21 (CTC2)	WGM20 (PWM2)	Timer/Counter Mode of Operation	TOP	Update of OCR2	TOV2 Flag Set on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR2	Immediate	MAX
3	1	1	Fast PWM	0xFF	BOTTOM	MAX

Note: 1. The CTC2 and PWM2 bit definition names are now obsolete. Use the WGM21:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

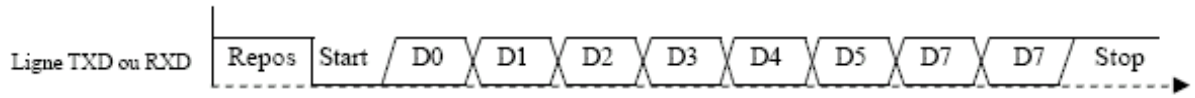
Les bits WGM20 et WGM21 permettent de choisir le mode PWM. Pour plus de précision se repporter à la documentation constructeur de l'Atmega16 (page 129).

3.7.5) l'UART (Unité Asynchrone de Réception Transmission)

C’est une interface série asynchrone. Elle permet de relier une unité centrale à des périphériques tels que :

- consoles de visualisation
- oscilloscopes ou autres appareils de mesure
- claviers
- souris
- modems
- …

ou à d’autres ordinateurs. Elle a l’avantage de relier deux systèmes sur une longueur de plusieurs dizaines de mètres mais elle a l’inconvénient d’être lente. Elle est de plus en plus remplacée par la liaison USB.
Elle est constituée d’une ligne d’émission (TXD) et d’une ligne de réception (RXD).
Si on souhaite se connecter à un ordinateur il faut prévoir une adaptation des niveaux entre celui-ci (Norme RS232) et le µC (Signaux compatibles TTL). Des circuits spécialisés permettent aisément de faire cette adaptation (MAX232 par ex.).
Dans le domaine industriel l’UART est aussi utilisée pour piloter des liaisons de type RS485 ou RS422.
En l’absence de transmission, les lignes TXD et RXD se trouvent au niveau logique haut.
Format de transmission :
- 1 bit de start : Front descendant puis niveau « 0 » pendant la durée de transmission d’un bit.
- 8 ou 9 bits de données : Bit de poids faible transmis en 1er.
- 1, 1.5 ou 2 bits de stop : niveau « 1 » pendant la durée de transmission de ces bits.



Description des registres :

UDR, UCSRA, UCSRB,UCSRC, UBRRH, UBRRL

UDR : Registre de données de l’UART

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0C ($2C)	D7	D6	D5	D4	D3	D2	D1	D0

Ce registre est en fait constitué de 2 registres distincts. Lorsque l’on écrit une donnée dans ce registre elle est stockée en fait dans le tampon d’émission avant l’envoi. Lorsqu’on lit dans ce registre on accède au tampon de réception qui contient la dernière donnée reçue. Cette disposition permet d’émettre une donnée sans altérer le contenu du tampon de réception.
L’écriture d’une donnée dans ce registre lance le processus d’émission de la donnée.

Registre UCSRA :

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

- RXC : Ce bit passe à « 1 » lorsqu’une donnée vient d’être reçue et qu’elle est disponible dans le registre de données UDR. La RAZ du bit RXC est effectuée par de la lecture de la donnée reçue, contenue dans le registre UDR.

- TXC : Ce bit passe à « 1 » lorsque la donnée, écrite dans le registre de données UDR, a été émise. La RAZ du bit TXC est effectuée par :
* L'écriture d'un « 1 » dans ce bit par le programme.
* L'appel à la routine d'interruption associée (Lorsque qu'elle est validée !)
- UDRE : Ce bit passe à « 1 » lorsque la donnée, écrite dans le registre de données UDR a été transférée dans le registre à décalage d'émission et que l'émission de la donnée à commencée. La RAZ du bit UDRE est effectuée par l'écriture d'une nouvelle donnée dans le registre de données UDR.
- FE : Ce bit passe à « 1 » lorsque le bit de stop de la donnée qui vient d'être reçue n'est pas à « 1 ». La RAZ du bit FE est réalisée lors de la réception correcte de la donnée suivante.
- DOR : Ce bit passe à « 1 » lors de la réception complète d'une donnée, alors que la précédente donnée reçue n'a pas encore été lue dans le registre de données UDR. Ce bit reste à « 1 » tant que la donnée précédemment reçue n'a pas été lue dans le registre de données UDR. La RAZ du bit OR est effectuée par la lecture du contenu du registre UDR.
- PE : à 1 si erreur de parité du caractère reçu alors que le contrôle parité a été activé.
- U2X : si à 1 double la vitesse de transmission.
- MPCM : si à 1 => fonctionnement en mode multi-processeurs.

Registre UCSRB :

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 7 – RXCIE: RX Complete Interrupt Enable
Si à « 1 », l'interruption de vecteur N°12 (Adresse $000B) est générée lorsque le bit RXC du registre UCSRA passe à 1.
• Bit 6 – TXCIE: TX Complete Interrupt Enable
Si à « 1 », l'interruption de vecteur N°14 (Adresse $000D) est générée lorsque le bit TXC du registre UCSRA passe à 1.
• Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable
Si à « 1 », l'interruption de vecteur N°13 (Adresse $000C) est générée lorsque le bit UDRE du registre UCSRA passe à 1.
• Bit 4 – RXEN: Receiver Enable
Si à « 1 », valide la partie réception de l'UART : La broche PD0 (RXD) est configurée en entrée série et n'est plus utilisable comme broche de port. Toutefois la résistance de pull-up interne peut être activée en mettant le bit PORTD0 du registre PORTD à « 1 ».
• Bit 3 – TXEN: Transmitter Enable
Valide la partie émission de l'UART : La broche PD1 (TXD) est configurée en sortie, et n'est plus utilisable comme broche de port.
• Bit 2 – UCSZ2: Character Size
Sélection du format de transmission : Si à « 1 » les données sont émises par mot de 9 bits, dans le cas contraire les données sont émises par mots de 8 bits.
• Bit 1 – RXB8: Receive Data Bit 8 et • Bit 0 – TXB8: Transmit Data Bit 8
Dans le cas ou le format de transmission par mots de 9 bits est sélectionné ces 2 bits représentent respectivement le neuvième bit reçu (RXB8) où à envoyer(TXB8).

Registre UCSRC:

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

Les registres UCSRC et UBRRH utilisent la même adresse physique. La différenciation se fait par le bit URSEL. URSEL doit être à 1 pour accéder le registre UCSRC en écriture.

- UMSEL à 0 sélectionne le mode de transmission asynchrone ou synchrone si 1.
- UPM1:0: Parity Mode

Déterminent le mode de parité: si une erreur est détectée le bit PE est mis à 1.

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- USBS : nombre de bits de stop.

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- UCSZ1:0: taille du caractère.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- UCPOL : Polarité de l'horloge

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

Registre UBRR (16 bits) :

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- URSEL : bit de sélection de registre : permet de sélectionner UCSRC (si à 1) ou UBRRH (si à 0) .

- UBRR11 à UBRR0 : bits de sélection de la vitesse de transmission. Voir tableau.

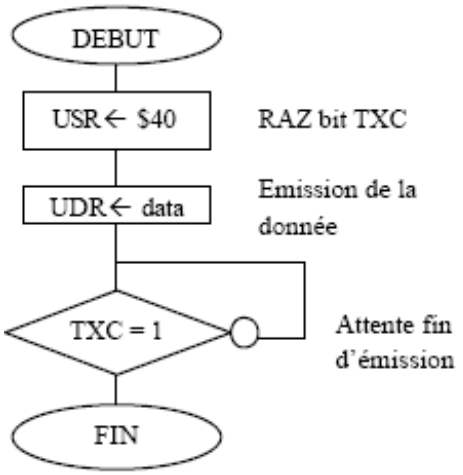
Baud Rate (bps)	f_{osc} = 1.0000 MHz				f_{osc} = 1.8432 MHz				f_{osc} = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	–	–	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	–	–	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	–	–	–	–	–	–	0	0.0%	–	–	–	–
250k	–	–	–	–	–	–	–	–	–	–	0	0.0%
Max (1)	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

<u>Exemple de programme de configuration de la liaison série asynchrone en émission et réception à 9600 bauds :</u>

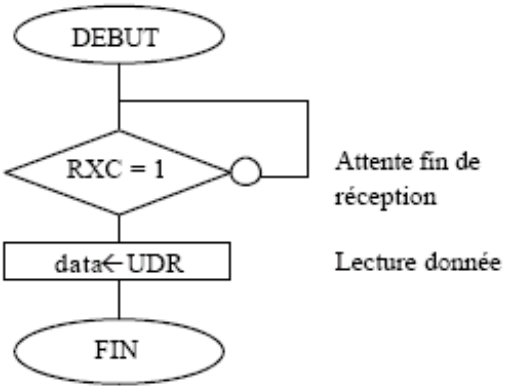
// USART initialization **Quartz = 4 MHZ**
// Communication Parameters: **8 Data, 1 Stop, No Parity**
// USART Receiver: **On**
// USART Transmitter: **On**
// USART Mode: **Asynchronous**
// USART Baud rate: **9600**
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x19;

Algorigrammes pour l'émission et la réception de caractères :

Emission de la donnée contenue dans « data »

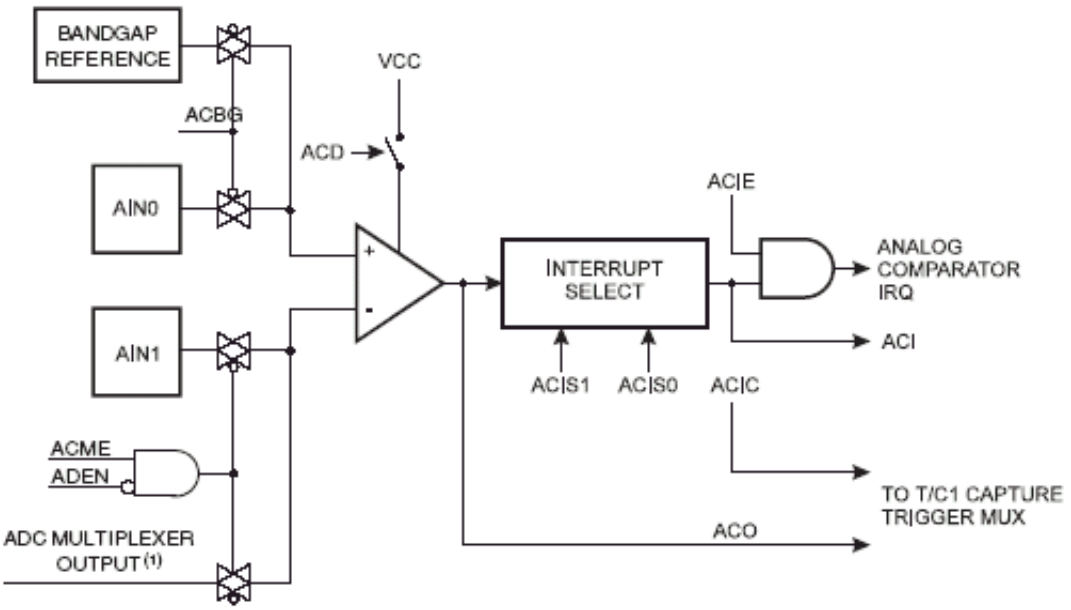


Réception d'une donnée et sauvegarde dans « data »



3.7.7) Le comparateur analogique

Architecture :



Principe :

Le système de comparaison analogique se compose essentiellement d'un comparateur analogique de tension et d'une logique de détection des changements d'état de la sortie du comparateur de tension. Le système de comparaison analogique peut être utilisé pour piloter la fonction d'entrée de capture du compteur1, en lieu et place de l'entrée ICP.

Registre ACSR :

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

• Bit 7 – ACD: Analog Comparator Disable
Un 1 coupe l'alimentation du comparateur.
• Bit 6 – ACBG: Analog Comparator Bandgap Select
Mis à 1 ce bit active une référence de tension interne qui se substitue à l'entrée positive du comparateur. Mis à 0 c'est l'entrée AIN0 qui est appliquée sur l'entrée + du comparateur.
• Bit 5 – ACO: Analog Comparator Output
Sortie directe du comparateur de tension :
- Si VAIN0 > VAIN1 alors ACO = « 1 »
- Si VAIN0 < VAIN1 alors ACO = « 0 »
• Bit 4 – ACI: Analog Comparator Interrupt Flag
Drapeau de détection de changement d'état de ACO. Lorsque le changement d'état à détecter (Sélectionné par ACIS1..0) se produit, ce bit passe à « 1 ».
La RAZ du bit ACI est effectuée par :
- L'écriture d'un « 1 » dans ce bit par le programme.
- L'appel à la routine d'interruption associée (Lorsque qu'elle est validée !)
• Bit 3 – ACIE: Analog Comparator Interrupt Enable
Validation de l'interruption comparaison analogique. Si ce bit est à « 1 » et que le bit ACI passe à « 1 », alors l'interruption de vecteur N° 17 (Adresse :$0010) est générée.
• Bit 2 – ACIC: Analog Comparator Input Capture Enable
Lorsque ce bit est à « 1 », la sortie directe ACO du comparateur de tension est reliée à la logique d'entrée de capture du timer1. Dans ce cas la sortie directe ACO du comparateur de tension remplace la broche ICP. On dispose alors sur ACO, de toutes les possibilités de détection de fronts que le timer1 possède.
• Bits 1, 0 – ACIS1, ACIS0: Analog Comparator Interrupt Mode Select
Sélection du type de changement d'état sur ACO à détecter.

ACIS1	ACIS0	Type de changement d'état sur ACO à détecter
0	0	Front montant et front descendant
0	1	Réservé : Ne pas utiliser
1	0	Front descendant
1	1	Front montant

Le registre SFIOR :

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

• Bit 3 – ACME: Analog Comparator Multiplexer Enable
Seul bit du registre utilisé: mis à 1 avec l'ADC inactif (ADEN et ADCSRA à 0), la sortie du multiplexeur de l'ADC est connectée à l'entrée – du comparateur. Mis à 1 c'est l'entrée AIN1 qui y est connectée. Cette fonction permet d'effectuer des comparaisons avec plusieurs niveaux de tensions analogiques.

ACME	ADEN	MUX2..0	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

Exemple d'initialisation du comparateur analogique:

```
// Analog Comparator initialization
// Analog Comparator: On
// The Analog Comparator's positive input is connected to the Bandgap Voltage Reference
// The Analog Comparator's negative input is connected to the ADC multiplexer channel 3
// Analog Comparator Input Capture by Timer/Counter 1: On
ACSR=0x44;
SFIOR=0x08;
ADMUX=0x03 ;
```

3.7.8) L'interface série synchrone :

La liaison série synchrone (SPI) de l'Atmega16 permet d'envoyer et de recevoir des données sous forme série à l'aide de 4 lignes :
- Une ligne d'entrée et une ligne de sortie de données série : MISO (PB6) et MOSI (PB5)
- Une ligne d'horloge : SCK (PB7)
- Une ligne de validation des transferts : /SS (PB4)

Une des particularités de cette interface est de pouvoir fonctionner dans 2 modes :
- Le mode maître : Dans ce cas l'interface génère l'horloge de synchronisation des données.
- Le mode esclave : Dans ce cas l'interface reçoit l'horloge de synchronisation des données.

Rôle des broches en fonction du mode de fonctionnement de la SPI :

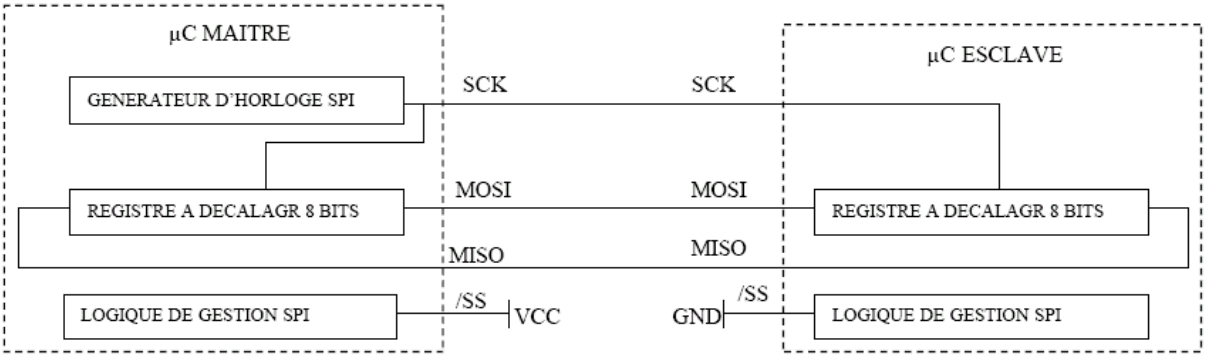
Broche	Mode Maître	Mode esclave
MISO	Entrée des données série	Sortie des données série
MOSI	Sortie des données série	Entrée des données série
SCK	Sortie horloge de transfert de données	Entrée horloge de transfert de données
/SS	Sortie d'E/S générale (PB4) ou entrée d'autorisation d'émission de données	Entrée de validation de l'interface SPI. (/SS = « 0 » durant les transferts de données)

Définition du sens des broches lors de la validation de la SPI :

Broche	Mode Maître	Mode esclave
MISO	Direction forcée : ENTREE	Direction définie par le bit 6 du registre DDRB
MOSI	Direction définie par le bit 5 du registre DDRB	Direction forcée : ENTREE
SCK	Direction définie par le bit 7 du registre DDRB	Direction forcée : ENTREE
/SS	Direction définie par le bit 4 du registre DDRB	Direction forcée : ENTREE

Remarque :

La validation de l'interface SPI, ne paramètre pas complètement la direction des 4 broches affectées à cette interface. L'utilisateur doit agir sur le registre de direction du PORT B (DDRB) afin, par exemple forcer MOSI (PB5) en sortie lors d'une utilisation de l'interface SPI en mode maître. Dans le cas contraire, aucune donnée série ne sera émise.



On constate que l'ensemble des 2 registres à décalage 8 bits forment un registre à décalage 16 bits unique partagé entre les 2 µC. L'émission d'un octet vers l'esclave par le maître provoque, la réception par le maître de la donnée contenue dans le registre de l'esclave.

En conséquence c'est toujours le maître qui décide des transfert :, il doit émettre une donnée pour en recevoir une autre.

REMARQUE :

En mode Maître *lorsque la broche /SS est configurée en* ***entrée,*** *elle doit être maintenue au niveau* ***haut pour autoriser les transferts****. Si elle est configurée en* ***sortie,*** *elle devient une broche de sortie d'intérêt général (PB4)* ***et est ignorée par l'interface SPI. En mode Esclave*** *cette broche est* ***obligatoirement une entrée****, et elle doit être au niveau logique* ***bas lors des transferts.***

Description des registres :

SPCR : Registre de contrôle de l'interface SPI :

Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0D ($2D)	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

- SPE : Validation de l'interface SPI lorsqu'il est positionné à « 1 ». Dans le cas contraire l'interface est désactivée.
- SPIE : Validation de l'interruption SPI. Si ce bit est à « 1 » et que le bit SPIF du registre SPSR passe à « 1 », alors l'interruption de vecteur N° 11 (Adresse :$000A) est générée.
- DORD : Ordre de transmission des bits.
  - * DORD = 1 : Le bit de poids faible est émis en 1er,
  - * DORD = 0 : Le bit de poids fort est émis en 1er.
- MSTR : Définit le mode de fonctionnement de l'interface :
  - * MSTR = 1 : Mode maître,
  - * MSTR = 0 : Mode esclave.
- CPOL : Sélection de la polarité de l'horloge :
  - * CPOL = 1 : Horloge inactive (Au repos) au niveau haut.
  - * CPOL = 0 : Horloge inactive (Au repos) au niveau bas.
- CPHA : Sélection de la phase des donnée par rapport à l'horloge :

* CPHA = 1 : Données positionnées lors des transitions vers le niveau actif présentes sur la ligne d'horloge.

* CPHA = 0 : Données positionnées lors des transitions vers le niveau inactif présentes sur la ligne d'horloge. (Voir documentation Atmel Atmega16 p. 141)

- SPR1..0 : Sélection de la fréquence de l'horloge de transmission (Utile en mode maître uniquement):

SPR1	SPR0	Fréquence de l'horloge de transmission
0	0	Horloge système / 4
0	1	Horloge système / 16
1	0	Horloge système / 64
1	1	Horloge système / 128

Registre SPSR :

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- SPIF : Fin de transmission. Ce bit passe à « 1 » lorsque le transfert d'un octet est terminé.
- WCOL : Collision en écriture : Ce bit passe à « 1 » lors d'une tentative d'écriture dans le registre de données SPDR alors qu'un transfert est en cours.

Ces bits sont remis à « 0 » par :

- L'exécution de la séquence suivante : Lecture dans SPSR puis lecture ou écriture dans SPDR.
- L'appel à la routine d'interruption associée (Lorsque qu'elle est validée !).
- SPI2X : mis à 1 il double la vitesse de transmission.

Le registre SPDR :

SPDR : Registre de données de l'interface SPI :

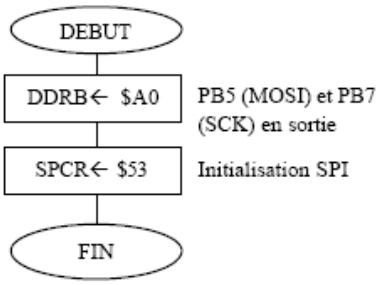
Adresse	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
$0E ($2E)	---------	---------	---------	---------	---------	---------	---------	---------

En mode maître, une écriture dans ce registre lance le transfert. Lorsque le transfert est terminé ce registre contient la donnée renvoyée par l'esclave.

En mode esclave, la donnée placée dans ce registre sera renvoyée vers le maître quand celui-ci exécutera un transfert. Lorsque le transfert sera terminé ce registre contiendra la donnée envoyée par le maître.

Exemple d'algorigrammes de mise en œuvre :

Initialisation



Emission de la donnée contenue dans « data » vers le registre à décalage