Table des matières

Chapitre 1 Introduction aux systèmes embarqués et microcontrôleurs

1.1 Introduction aux systèmes embarqués

Grâce aux avancées fulgurantes de la microélectronique, la puissance de calcul des microprocesseurs continue d'augmenter tandis que leur taille, leur coût et leur consommation d'énergie diminuent. Cela, a permis d'intégrer sur un même circuit intégré un processeur et les périphériques nécessaires à son fonctionnement, créant ainsi les microcontrôleurs utilisés dans les systèmes embarqués. Ces derniers ont envahi le quotidien de l'homme au point que leur utilisation est devenue indispensable dans bien de domaines.

1.1.1 Définition d'un système embarqué

Un système embarqué (embedded system en anglais) désigne tout système électronique programmable conçu pour exécuter des tâches spécifiques. Il est dit embarqué (embedded), car il est souvent intégré (enfoui) dans un système plus large, comme les caméras, les voitures (système de gestion du moteur, système de verrouillage centralisé, gestion des airbags, ABS…), les avions, les équipements médicaux, les appareils ménagers, les imprimantes, les distributeurs automatiques de billets, les téléphones portables, et bien d'autres encore.

Figure 1.1 Systèmes embarqués dans un véhicule.

1.1.2 Domaines d'applications des systèmes embarqués

Les systèmes embarqués sont présents dans de nombreux produits que nous utilisons quotidiennement, tels que les téléphones portables, les voitures, les avions, les équipements médicaux, les jouets pour enfants, les appareils électroménagers, les systèmes de sécurité, les robots, etc. Ils sont souvent cachés à l'intérieur de ces produits et sont utilisés pour effectuer une tâche spécifique ou pour contrôler un système complexe. Les systèmes embarqués sont également présents dans des secteurs tels que l'aérospatiale, la défense, les communications, l'automatisation industrielle et les technologies de l'information.

1.1.3 Types des systèmes embarqués

1.1.3.1 Classification des systèmes embarqués en fonction de leur complexité

Les systèmes embarqués peuvent être classés en trois catégories selon leur complexité et les tâches qu'ils effectuent : les systèmes embarqués à petite échelle, à moyenne échelle et sophistiqués.

- **Les systèmes embarqués à petite échelle** sont des systèmes simples équipés de microcontrôleurs de 8 ou 16 bits et peuvent être alimentés par des

batteries. Des exemples de ces systèmes sont des montres numériques, calculatrices simples ou télécommandes pour télévision.
- **Les systèmes embarqués à moyenne échelle** sont plus complexes que les systèmes à petite échelle et sont conçus pour effectuer des tâches plus avancées. Ils sont souvent équipés de microcontrôleurs ou microprocesseurs dédié de 16 ou 32 bits. Des exemples de systèmes embarqués à moyenne échelle sont des appareils photo numériques, des imprimantes ou lecteurs DVD.
- **Les systèmes embarqués sophistiqués** sont hautement complexes et sont conçus pour effectuer des tâches complexes et interagir avec d'autres systèmes. Ils peuvent inclure plusieurs microcontrôleurs ou microprocesseurs et nécessitent souvent des algorithmes logiciels avancés pour le contrôle et la communication. Des exemples de systèmes embarqués sophistiqués comprennent des équipements médicaux, des systèmes de contrôle industriels et des systèmes automobiles.

1.1.3.2 Classification des systèmes embarqués selon le concept temps réel

D'un point de vue temps réel, on peut distinguer trois types de systèmes embarqués :

- **Les systèmes temps réel "mous" :** ce sont des systèmes où la satisfaction des contraintes de temps réel n'est pas critique. Dans ces systèmes, une réponse tardive de la part du système n'est pas catastrophique. Par exemple, dans un système de contrôle de la température, un retard de quelques secondes dans la réponse ne pose pas de problème majeur.
- **Les systèmes temps réel "fermes" :** ce sont des systèmes où la satisfaction des contraintes de temps réel est critique, mais une infraction à ces contraintes n'entraîne pas de conséquences catastrophiques. Par exemple, dans un système de contrôle de la circulation, un retard de quelques secondes peut causer des perturbations, mais cela ne met pas en danger la vie des personnes.
- **Les systèmes temps réel "durs" :** ce sont des systèmes où la satisfaction des contraintes de temps réel est critique et une infraction à ces contraintes peut entraîner des conséquences catastrophiques. Par exemple, dans un système de contrôle d'un avion, un retard dans la réponse peut causer un accident. Ces systèmes nécessitent une attention particulière pour garantir la sécurité et la fiabilité du système.

1.1.4 Systèmes embarqués vs ordinateurs

Contrairement à un ordinateur de bureau ou portable, un système embarqué est conçu pour une tâche spécifique et il est souvent personnalisé pour une application particulière. Il est parfois confondu avec les termes "ordinateur embarqué" ou "ordinateur de bord", bien que ces termes ne soient pas tout à fait exacts. Le tableau suivant souligne quelques aspects de différence entre les systèmes embarqués et les ordinateurs.

Système embarqué	Ordinateur
Conçu pour des tâches spécifiques.	Polyvalent.
Basé sur microcontrôleur, microprocesseur à faible puissance ou DSP[1].	Basé sur microprocesseur
Faible consommation d'énergie.	Relativement, forte consommation d'énergie.
S'il utilise un système d'exploitation ce dernier est temps réel.	Utilise des systèmes d'exploitation non temps réel.
Mémoire centrale de faible taille (généralement quelques Ko)	Mémoire centrale de grande taille (en Go).
Son fonctionnement ne nécessite pas l'utilisation de mémoires secondaires. Cependant, dans certaines applications ils peuvent exploiter des flash disque, des disque dur…etc.	Utilise des mémoires secondaires de masse (disque dur interne, disque dur externe, …).
Possédant de simples interfaces homme/machine.	Possédant clavier souri et écran.
Moins encombrant, et donc peut être enfoui dans des système compacte	Encombrant.

Tableau 1.1 Brève comparaison entre systèmes embarqués et ordinateurs.

1.1.5 Architecture typique des systèmes embarqués

Contrairement aux ordinateurs à usage général, il n'existe pas une architecture commune à tous les systèmes embarqués. En réalité, chaque fabricant et chaque type de système ont leur propre architecture (par exemple, on ne s'attendrait pas à trouver la même architecture pour un système embarqué dans une machine à laver que dans un téléphone portable). Cependant, étant donné que la plupart des systèmes embarqués ont été conçus pour commander et contrôler d'autres systèmes plus larges (tels que mécaniques et électriques), ils ont généralement des capteurs en entrée et des actionneurs en sortie, contrairement aux ordinateurs qui ont des entrées et des sorties plus variées.

La figure ci-dessus représente une architecture commune à la plupart des système embarqués.

[1] Digital Signal Processor : processeur de traitement numérique des signaux.

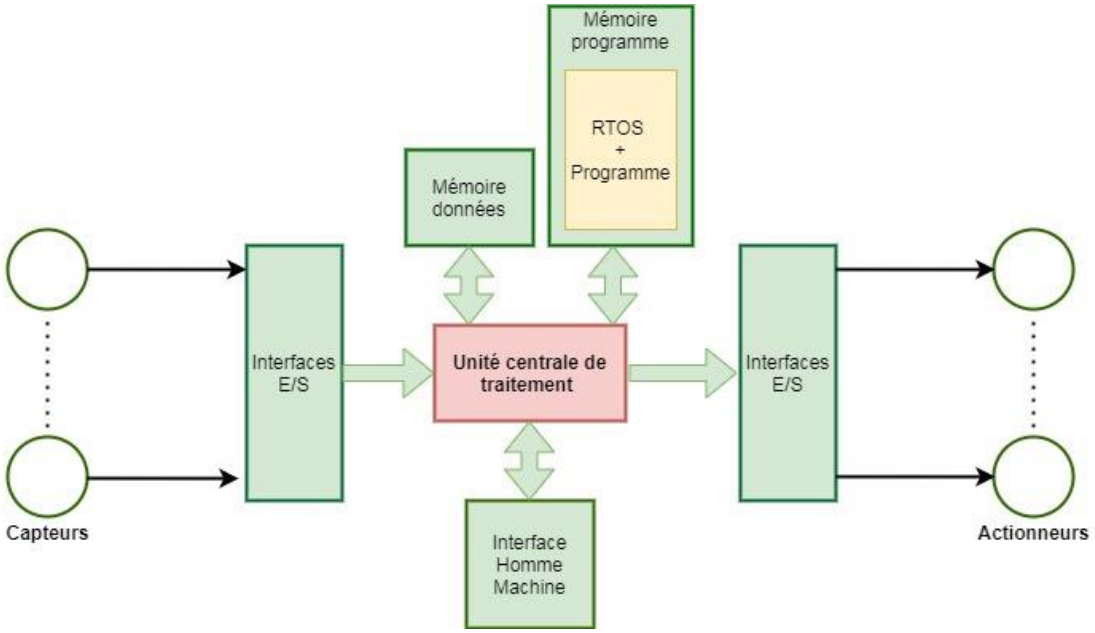


Figure 1.2 Architecture simplifiée des systèmes embarqués.

Un système embarqué se compose principalement de deux parties : le matériel et le logiciel embarqué. La partie matérielle peut varier en fonction de l'application spécifique du système embarqué. Généralement elle comprend les composants suivants[2]:

Microcontrôleur ou un microprocesseur : C'est la pièce maitresse du système embarqué, qui assure la gestion de toutes les opérations du système.

Mémoire : un système embarqué contient une mémoire programme (du type ROM, EPROM, EEPROM ou flash) pour stocker le code exécutable, une mémoire de données (du type RAM) pour stocker les données temporaires nécessaires au fonctionnement du système, et une mémoire souvent de type EEPROM pour stocker les paramètres de configuration du système.

Bus : il assure la communication entre les différents composants du système embarqué.

Interfaces entrées/sorties (E/S) : elles permettent de connecter le système à des périphériques externes, tels que des capteurs, des actionneurs, des écrans, des claviers, etc.

[2] Les composants matériels mentionnés peuvent ne pas être exhaustifs ou ne pas être tous présents dans un même système embarqué.

Alimentation électrique : elle fournit l'énergie nécessaire au fonctionnement du système embarqué.

La partie logiciel embarqué peut comprendre une ou plusieurs applications ainsi qu'un système d'exploitation embarqué et des pilotes de périphériques.

1.2 Composants de base des systèmes embarqués

1.2.1 Unité centrale de traitement

L'unité centrale de traitement (CPU : Central Processing Unit) ou processeur a pour fonction principale d'exécuter des instructions stockées dans la mémoire code, stocker les résultats de ces instructions dans la mémoire de données, ainsi qu'interagir avec les autres composants du système embarqué Toutes les activités de la CPU sont cadencées par une horloge qui régule toutes ses activités.

1.2.1.1 Unités fondamentales

La CPU est composée de deux unité fondamentales :

Unité de commande : qui se de chercher l'instruction dans la mémoire programme, la décoder et de chercher les données associées dans la mémoire de données. Elle envoie ensuite les ordres correspondants à l'unité de traitement pour effectuer les opérations arithmétiques ou logiques indiquées par l'instruction. Enfin, elle stocke éventuellement le résultat dans la mémoire de données avant de passer à l'instruction suivante dans la mémoire programme.

Unité de traitement (unité arithmétique et logique) : effectue les opérations arithmétiques et logiques ordonnées par l'unité de commande.

La CPU utilise manipule et stocke temporairement les données durant le traitement en utilisant des registres de travail. En plus, des registres indicateurs (ou registres d'état) sont employés pour signaler divers états d'opération tels que les débordements ou les résultats nuls, ainsi que pour enregistrer les événements tels que les interruptions et leurs types.

1.2.1.2Architecture de von Neumann et architecture d'Harvard

L'architecture de von Neumann et l'architecture d'Harvard sont deux types de modèles de traitement de l'information utilisées dans les ordinateurs et les systèmes embarqués.

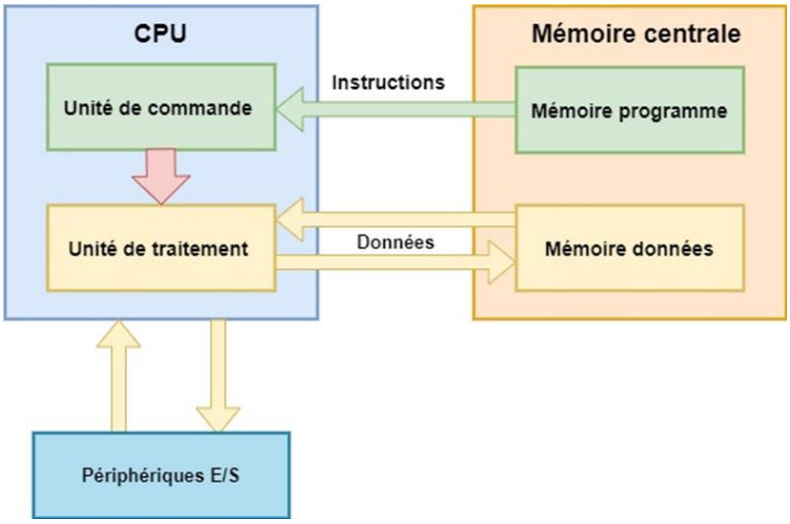


Figure 1.2 Structure et fonctionnement de la CPU.

a) Architecture de von Neumann

Dans ce modèle, les données et les instructions sont stockées dans une seule mémoire, du type RAM, et sont accessibles via un bus de données commun. Cela signifie que les instructions et les données doivent être chargées séquentiellement dans la mémoire et que le processeur doit accéder à la même mémoire pour exécuter les instructions et récupérer les données. Cette architecture est utilisée dans les ordinateurs.

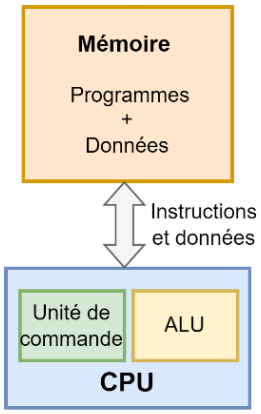


Figure 1.3 Architecture de von Neumann.

b) Architecture d'Harvard

Cette architecture utilise deux mémoires distinctes pour stocker les instructions et les données ; une mémoire morte pour les instructions (ROM, EPROM, EEPROM ou flash) et une mémoire vive pour les données (RAM). Cela signifie que les instructions et les données peuvent être chargées simultanément, ce qui peut améliorer les performances. Les deux mémoires sont reliées au processeur via des bus de données distincts. Cette architecture est souvent utilisée dans les microcontrôleurs et les systèmes embarqués où la vitesse de traitement est critique.

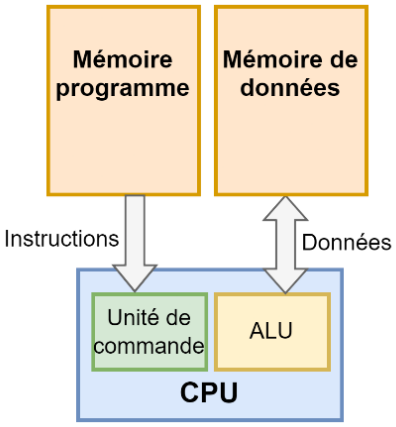


Figure 1.4 Architecture d'Harvard.

1.2.1.3 Architectures RISC et CISC

Les architectures CISC et RISC sont deux approches différentes pour la conception de CPU.

Les CPU CISC (Complex Instruction Set Computer) peuvent effectuer des tâches complexes en une seule instruction. Ils sont caractérisés par un nombre important d'instructions complexes, chacune pouvant prendre plusieurs cycles d'horloge pour s'exécuter. Les instructions CISC peuvent avoir une longueur variable allant de 1 à 15 octets. Malgré leur complexité, les CPU CISC ont un nombre limité de registres. Les processeurs CISC sont généralement utilisés dans les ordinateurs de bureau et les serveurs.

Les CPU RISC (Reduced Instruction Set Computer) sont caractérisés par un nombre réduit d'instructions simples qui peuvent être exécutées en une seule période d'horloge. Toutes les instructions ont la même longueur, ce qui simplifie la conception du processeur. Les CPU RISC utilisent généralement un grand nombre de registres, ce qui leur permet d'effectuer des opérations plus rapidement et efficacement que les CPU CISC. Les architectures RISC sont particulièrement adaptées aux systèmes embarqués, car elles sont moins coûteuses à fabriquer et consomment moins d'énergie.

Avec les progrès technologiques, la distinction entre ces deux architectures s'estompe et les CPU modernes utilisent souvent une combinaison des deux approches.

1.2.1.4 Pipelining

Les processeurs modernes sont basés sur une architecture dite « pipeline », qui consiste à diviser l'exécution des instructions en plusieurs étapes. Le pipeline est inspiré des chaines de montage, il est constitué de plusieurs « étages », chacun d'entre eux réalise une étape du processus d'exécution d'une instruction.

Le but du pipeline est d'optimiser le temps d'exécution des instructions en permettant à plusieurs instructions d'être en cours de traitement simultanément (exécution parallèle des instruction).

Les étapes classiques du pipeline sont :

1- Récupération (Fetch) : Le processeur récupère l'instruction suivante depuis la mémoire.
2- Décodage (Decode) : Le processeur décrypte l'instruction pour savoir quelle opération doit être effectuée.
3- Exécution (Execute) : Le processeur effectue l'opération spécifiée par l'instruction.
4- Accès à la mémoire (Memory access) : Si l'instruction nécessite un accès à la mémoire, le processeur effectue cette opération à cette étape.
5- Écriture (Write back) : Le résultat est écrit dans le registre approprié.

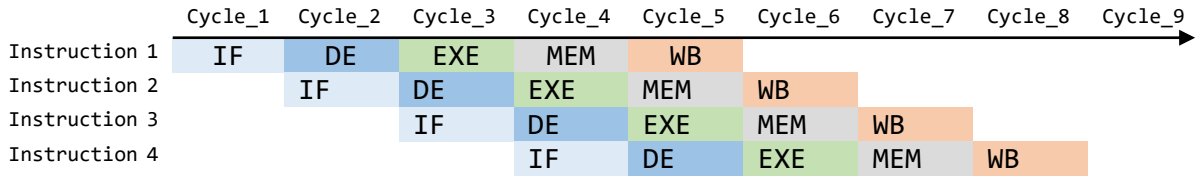


Figure 1.5 Séquence d'exécution des instructions sur une CPU avec un pipeline à 5 étages (IF :Instruction Fetch, DE : Decode, EXE : Execute, MEM : Memory Access et WB : Write Bach).

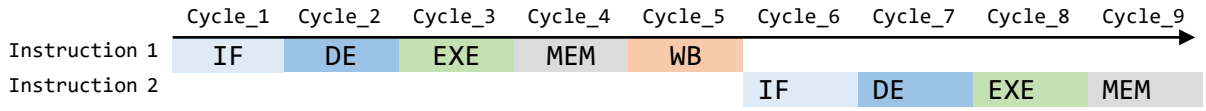


Figure 1.6 Séquence d'exécution des instructions sur une CPU sans pipeline.

La figure 1.5 montre que, pendant qu'une instruction est exécutée à l'étape 3, une autre instruction peut être décodée à l'étape 2 et une troisième peut être l'étape 1. Il est donc évident que lorsqu'une instruction dépende du résultat de l'instruction précédente, c'est-à-dire ne peut être exécutée qu'après l'obtention du résultat de l'instruction précédente, le pipeline perd son avantage. Une autre limite qui peut perturber l'exécution parallèle sont les instructions de saut et de branchement.

1.2.2 Bus système

Le bus système représente le système nerveux des ordinateurs et des systèmes embarqués, car c'est un ensemble de fils conducteurs qui transportent des signaux électriques qui permettre la communication et la coordination entre les différents composants du système.

Les principaux bus du système sont les suivants :

Bus de données : bidirectionnel, il assure le transport des données entre les différents composants du système, tels que le processeur, la mémoire et les périphériques d'entrée/sortie (E/S). Le bus de données est généralement de 8, 16, 32 ou 64 bits de largeur, ce qui signifie qu'il peut transporter 8, 16, 32 ou 64 bits de données à la fois.

Bus d'adresse : unidirectionnel, la CPU utilise ce bus pour indiquer à la mémoire ou aux périphériques E/S l'adresse à laquelle il doit lire ou écrire des données.

Bus de contrôle : ce bus transporte les signaux de contrôle qui régissent les opérations du système, tels que les signaux d'horloge qui synchronisent les différentes activités, les signaux de lecture/écriture qui indiquent si une opération est une lecture ou une écriture, et les signaux de validation qui confirment que les données ont été correctement écrites ou lues.

1.2.3 Mémoires

1.2.3.1 Types de mémoires dans les systèmes embarqués

Il existe différents types de mémoires utilisées dans les systèmes embarqués, chacune ayant des caractéristiques et des avantages en termes de coût, de capacité, de vitesse, de fiabilité et de consommation d'énergie.

a) Mémoire vive (RAM : Random Access Memory)

Elle permet de stocker temporairement les données utilisées par la CPU. La RAM est volatile, ce qui signifie que les données sont perdues lorsque l'alimentation est coupée. Elle peut être de différents types, tels que SDRAM, DDR, DDR2, DDR3, DDR4, etc., avec des vitesses et des capacités variables. Dans une architecture de von Neumann la RAM est utilisée pour stocker les données et les instructions. Il existe principalement deux types de mémoire RAM :

DRAM (Dynamic Random Access Memory) : c’est la RAM utilisée dans la plupart des ordinateurs personnels et des systèmes embarqués, à cause de son prix relativement plus bas. Les cellules de mémoire de la DRAM utilisent un condensateur et un transistor pour stocker chaque bit de données. Les données doivent être rafraîchies périodiquement pour éviter que les charges stockées ne se dissipent.

SRAM (Static Random Access Memory) : n’a pas besoin de rafraîchissement périodique et plus rapide que la DRAM. Cependant, elle est plus coûteuse et nécessite plus de puissance. Elle est principalement utilisée avec de petites tailles dans les microcontrôleurs les applications nécessitant une vitesse de traitement élevée.

Il existe également d'autres types de mémoire RAM, tels que la FCRAM (Fast Cycle RAM) et la QDR (Quad Data Rate) SRAM, qui sont utilisées dans des applications spécialisées nécessitant une vitesse de traitement encore plus élevée.

b) Mémoire morte (ROM: Read Only Memory)

La ROM est utilisée pour stocker des instructions et des données qui ne changent pas pendant le fonctionnement du système. La ROM est non volatile, ce qui signifie que les données sont conservées même lorsque l'alimentation est coupée. Il existe plusieurs types de mémoires ROM, chacun ayant des caractéristiques et des utilisations différentes.

ROM masquée (Mask ROM) : La ROM masquée est préprogrammée en usine et ne peut pas être modifiée une fois qu'elle a été fabriquée. Elle est utilisée pour stocker des données qui ne changeront jamais, comme le code de démarrage d'un ordinateur.

PROM (Programmable Read-Only Memory) : La PROM est vierge lorsqu'elle est fabriquée, mais peut être programmée une seule fois par l'utilisateur en utilisant un dispositif de programmation. Une fois programmée, les données ne peuvent pas être modifiées.

EPROM (Erasable Programmable Read-Only Memory) : L'EPROM peut être programmée et effacée plusieurs fois à l'aide d'un dispositif spécial appelé programmeur d'EPROM. Pour effacer l'EPROM, il doit être exposé à une forte lumière ultraviolette pendant environ 20 minutes.

EEPROM (Electrically Erasable Programmable Read-Only Memory) : L'EEPROM peut être programmée et effacée à l'aide d'un signal électrique plutôt qu'une lumière ultraviolette. L'EEPROM est souvent utilisée pour stocker des données qui doivent être mises à jour régulièrement, comme le mot de passe ou les paramètres de configuration d'un système embarqué.

Flash Memory : La mémoire flash est une version plus avancée de l'EEPROM, qui est moins chère et peut stocker des données plus rapidement, mais elle peut nécessiter plus d'énergie pour être programmée et effacée. Elle est mieux adaptée pour le stockage à grande échelle des programmes et des données en blocs. La mémoire flash est couramment utilisée dans les clés USB, les cartes mémoire, les disques SSD et les mémoires programmes des systèmes embarqués.

1.2.3.2 Fonctionnement des mémoires

On peut assimiler une mémoire à un tableau de cellules dites « case mémoire », chacune contenant un nombre fixe de bits qui est toujours une puissance de 2. Une case mémoire est identifiée par une adresse unique qui permet à la CPU de lire ou d'écrire dans cette case. Le nombre de cases

mémoire correspond à la taille de la mémoire, en d'autres termes sa « capacité de stockage » qui peut être mesurée en bits, octets, ko, Mo, Go ou To (téraoctets).

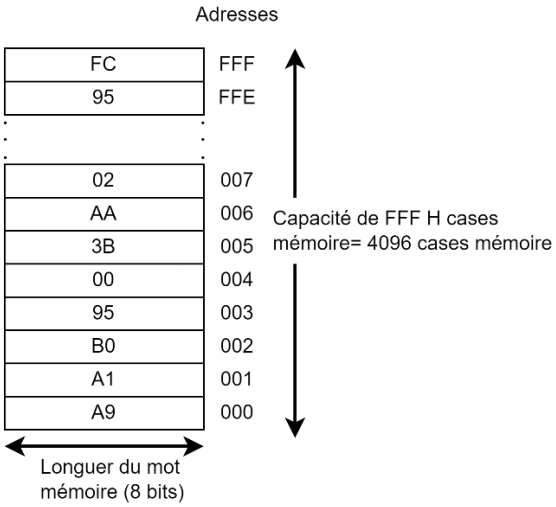


Figure 1.7 Cases mémoire et adresses.

Avec une adresse de n bits (bus d'adresse de n bits) il est possible d'adresser au plus 2^n cases mémoire. Chaque case est remplie par un mot de données. Le nombre de fils d'adresses d'un boîtier mémoire définit donc le nombre de cases mémoire que comprend le boîtier. Le nombre de fils de données définit la taille des données que l'on peut sauvegarder dans chaque case mémoire.

Exemple

Un microprocesseur dispose d'un bus d'adresses de 16 bits (deux octets) et d'un bus de données de 8 bits (soit un octet).

1- Quelle est la taille maximale de mémoire que ce microprocesseur peut adresser ?
2- Si ce microprocesseur utilisait un bus de données de 16 bits, quelle serait la nouvelle taille maximale de mémoire qu'il pourrait adresser ?

Solution

1- Puisque le microprocesseur a un bus d'adresses de 16 bits, il peut adresser jusqu'à 2^{16} cases mémoire. D'autre part, la taille des cases mémoire est de 1 octet, car le bus de données contient 8 bits. Par conséquent, le microprocesseur peut adresser jusqu'à 2^{16} octets=64 ko.
2- Si la taille du bus de données est 16 bits, La taille maximale de la mémoire que le microprocesseur pourrait adresser serait multipliée par deux, soit 128 ko, car la taille des cases mémoire serait de 2 octets.

Un boîtier mémoire comprend également des entrées de commande qui permettent de définir le type d'action que l'on effectue avec la mémoire (lecture/écriture pour une RAM ($R/\overline{W}$)) et de mettre les entrées/sorties du boîtier en haute impédance ($\overline{CS}$ (*Chip Select*) ou $\overline{CE}$ (*Chip Enable*)).

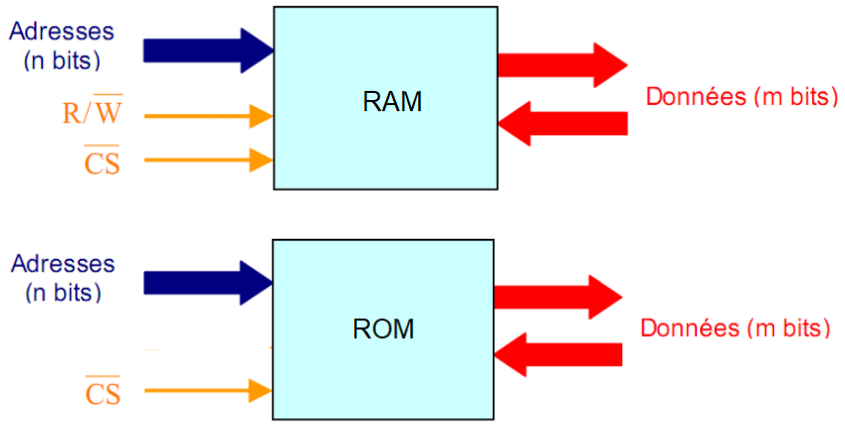


Figure 1.8 Représentation des bus de données, d'adresse et de commande des RAM et des ROM.

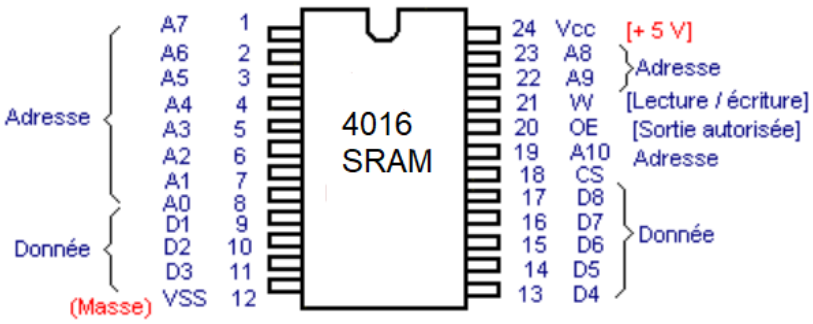


Figure 1.9 Exemple de brochage d'un boitier mémoire

1.2.3.3 Chronogrammes de lecture/écriture en mémoire

Une opération de lecture ou d'écriture suit le cycle :

1. sélection de l'adresse
2. choix de l'opération à effectuer ($R/\overline{W}$)
3. sélection de la mémoire Chip Select ($\overline{CS}$ = 0)
4. lecture ou écriture des données.

Il est à noter que ces étapes peuvent changées légèrement selon la technologie employée.

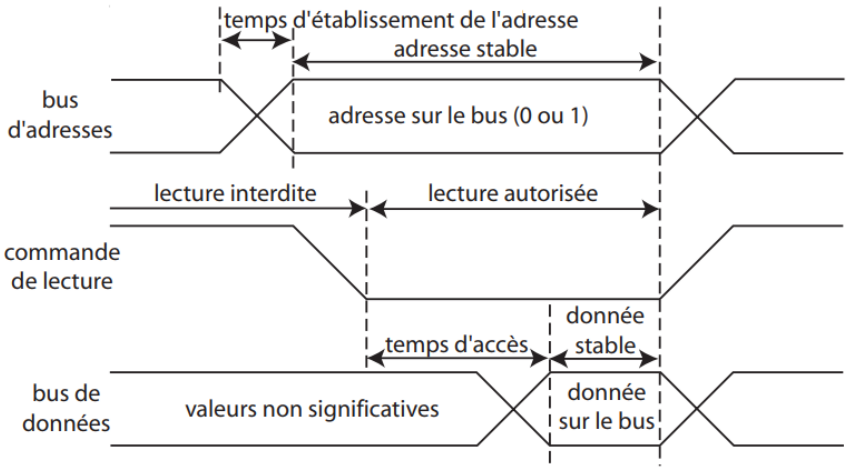


Figure 1.10 *Chronogramme de lecture d'une RAM.*

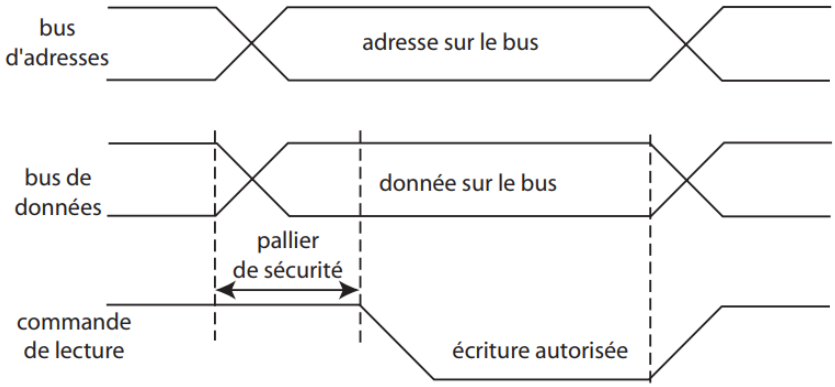


Figure 1.11 *Chronogramme de lecture d'une RAM.*

1.2.3.4 Assemblage de mémoires

Il est possible d'assembler plusieurs boîtiers mémoire en parallèle pour augmenter la longueur du mot mémoire (longueur de case mémoire équivalente) ou en série pour augmenter la capacité de stockage.

a) Assemblage en parallèle

L'assemblage en parallèle consiste à connecter les boîtiers mémoire de manière à ce qu'ils partagent les mêmes bus d'adresse et de contrôle. Cependant, les bus de données des boîtiers mémoire sont juxtaposés pour former le bus de données de la mémoire équivalente. Dans cette configuration, les mémoires sont généralement activées simultanément et ont la même plage d'adresse.

La figure 1.12 illustre l'association de deux mémoires en parallèle. On remarque que :

- Mémoire 1, Mémoire 2, et la mémoire globale possèdent un bus d'adresse de taille n.

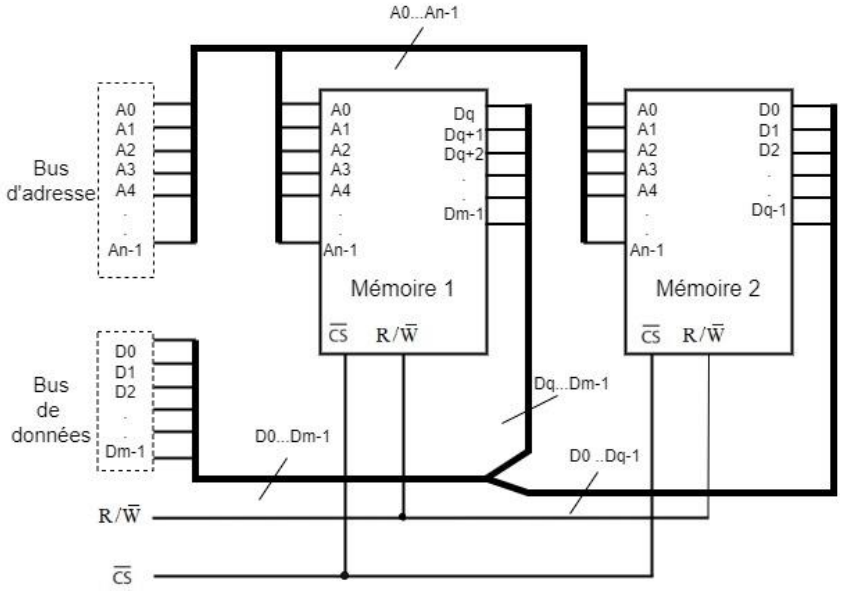


Figure 1.12 Exemple d'association de deux mémoires en parallèle.

- Mémoire 1 dispose d'un bus de données de taille (m-q), tandis que Mémoire 2 a un bus de données de taille q. La taille du bus de données de la mémoire globale est m=(m-q)-q.

b) Assemblage série

L'assemblage en série implique que les boîtiers mémoire partagent le même bus de données et le même bus de contrôle, à l'exception de la ligne CS (Chip Select). Ainsi, les boîtiers mémoire ne doivent jamais être actifs en même temps pour éviter les conflits au niveau du bus de données. Chaque boîtier mémoire doit avoir une adresse unique pour être sélectionné correctement. Pour cela, les lignes de poids faible du bus d'adresse sont partagées par tous les boîtiers mémoire, tandis que des lignes de poids fort sont utilisées avec CS pour les activer.

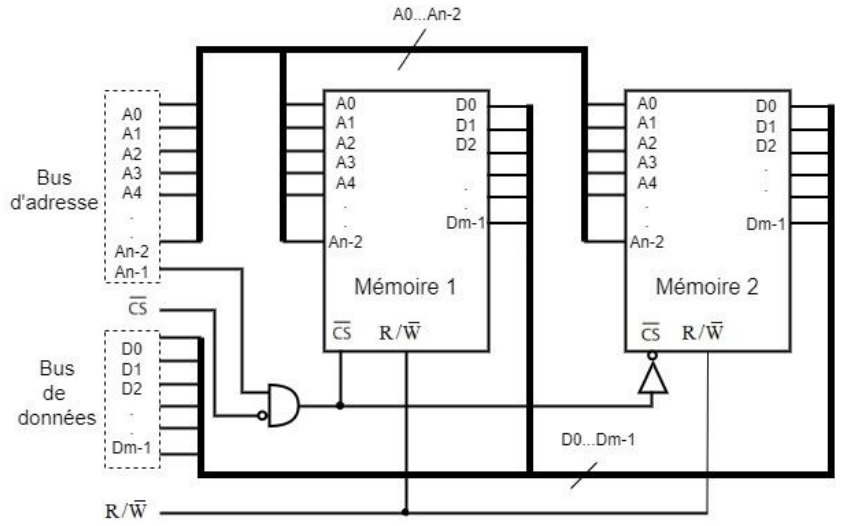


Figure 1.13 Exemple d'association de deux mémoires en série.

La figure 1.13 montre un exemple d'association de deux mémoires en série. On remarque que :

- La taille de la mémoire globale (2^n mots mémoire) est la somme des tailles de Mémoire 1 et Mémoire 2 (2^{n-1} mots mémoire chacune).
- La ligne d'adresse An-1 est utilisée avec pour sélectionner le boitier mémoire à activer.
- Les plages d'adresse en binaire des deux mémoires sont comme suit :

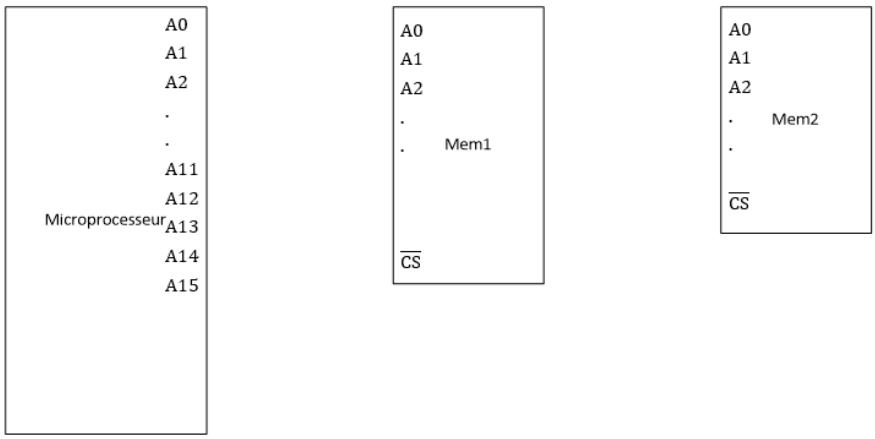
Mémoire	Début						Fin					
	An-1	An-2	…A3	A2	A1	A0	An-1	An-2	…A3	A2	A1	A0
Mémoire 1	0	0	0	0	0	0	0	1	1	1	1	1
Mémoire 2	1	0	0	0	0	0	1	1	1	1	1	1

Exercice

Un microprocesseur avec un bus d'adresses de deux octets (16 bits) et un bus de données d'un octet utilise deux boitiers mémoires (Mem1 et Mem2) en série implantées comme suit :

- Mem1 de 16 Ko implantée à partir de l'adresse 0000 H,
- Mem2 de 8 Ko implantée juste après la dernière case de Mem1.

1) Déterminer les plages d'adresses (adresses du début et de fin) des boitiers Mem1 et Mem2.
2) Effectuer le câblage nécessaire pour avoir ces plages d'adresse.



1.3 Généralités sur les microcontrôleurs

1.3.1 Définition

Les microcontrôleurs sont des circuits intégrés programmables qui intègrent sur une même puce un processeur, de la mémoire vive (RAM), de la mémoire morte (ROM ou Flash), des interfaces d'entrée/sortie (GPIO), des interfaces de communication série (SPI, I2C, UART, USB, etc.), des convertisseurs analogique/numérique (CAN) et d'autres périphériques.

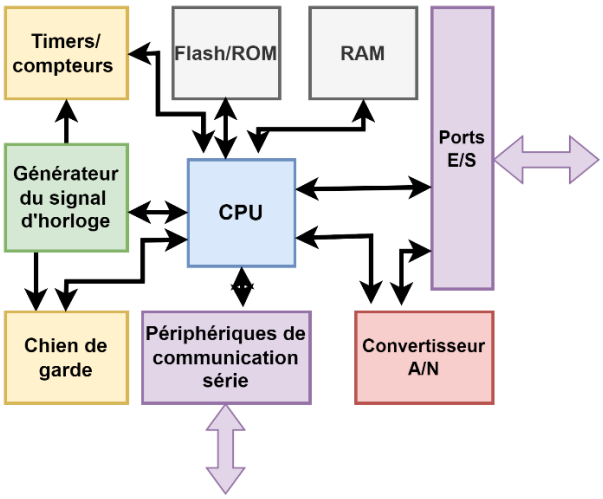


Figure 1.14 Composants typiques des microcontrôleurs.

Les microcontrôleurs sont largement utilisés dans les systèmes embarqués, à cause de leur faible coût, leur petite taille et leur faible consommation d'énergie.

1.3.2 Microcontrôleur vs microprocesseur

Les microcontrôleurs et les microprocesseurs ont des fonctions similaires, mais ils sont utilisés dans des contextes différents.

Microprocesseurs	Microcontrôleurs
Utilisés dans des ordinateurs, des serveurs et d'autres systèmes informatiques	Conçus pour être utilisés dans des systèmes embarqués
Utilisent des mémoires externes de grandes taille	Conçus pour fonctionner avec des ressources de mémoire interne limitées
Ne contiennent pas de ports d'E/S intégrés	Contiennent de ports d'E/S intégrés
Basés sur l'architecture de von Neumann	Basés sur l'architecture d'Harvard
Possèdent un nombre limité de registres	Contiennent un grand nombre de registres
Ils sont chers	Ils sont bon marché
Usage général dans la conception et l'exploitation.	Orientés à des applications spécifiques.
Fréquence d'horloge en GHz	Fréquence d'horloge en MHz
Relativement gourmands en énergie	Consomment moins d'énergie
Unité dépendante	Unité autonome

1.3.3 Aperçu sur l'évolution des microcontrôleurs

Le 8051 est considéré comme le premier microcontrôleur 8 bits et a été introduit par Intel en 1980. Cette puce révolutionnaire contient un

processeur, de la mémoire, des entrées/sorties, un convertisseur A/N, un convertisseur N/A, des timers, une interface de communication série, le tout intégré sur une seule puce. Le 8051 utilise l'architecture d'Harvard et son CPU utilise l'architecture CISC. Cependant, les microcontrôleurs contemporains utilisent principalement l'architecture RISC qui offre une approche rentable et des performances améliorées.

Les fabricants de microcontrôleurs ont constamment amélioré leurs produits pour répondre aux besoins des développeurs de systèmes embarqués. Aujourd'hui, les microcontrôleurs modernes intègrent plusieurs fonctions périphériques améliorées, telles que la modulation de largeur d'impulsion (PWM) et les interfaces de communication série (SPI, TW, CAN, etc.). En outre, ils sont conçus pour une variété d'applications, allant des dispositifs médicaux aux voitures connectées.

Les fabricants de microcontrôleurs les plus populaires comprennent Microchip Technology, Texas Instruments, Freescale et Atmel (qui a été acheté par Microchip en 2016). Microchip fabrique un large éventail de modèles de microcontrôleurs PIC (PICmicro) pour répondre aux besoins des développeurs de systèmes embarqués. Les modèles de microcontrôleurs PIC incluent le PIC16, le PIC18, le PIC24, le dsPIC et le PIC32, chacun ayant des spécificités qui les distinguent des autres et conçus pour répondre à des besoins différents.

Atmel, désormais une filiale de Microchip Technology, a également fabriqué plusieurs modèles de microcontrôleurs au fil des ans, notamment les AVR (ATmega, ATtiny et ATxmega) et les SAM (qui sont basés sur l'architecture ARM), entre autres. Chacune de ces familles de microcontrôleurs offre des fonctionnalités spécifiques qui la distinguent des autres et est conçue pour répondre à des besoins différents des développeurs de systèmes embarqués.

Enfin, ARM (Advanced RISC Machine) est une entreprise britannique qui conçoit des cœurs de processeurs et qui accorde des licences pour ses conceptions à des fabricants de puces. Les cœurs de processeurs 32 et 64 bits d'ARM sont largement utilisés par les fabricants de microcontrôleurs pour fabriquer leurs produits. Les conceptions d'ARM sont très populaires et sont particulièrement utilisées dans les appareils portables tels que les smartphones, les tablettes et les lecteurs multimédias en raison de leur faible consommation d'énergie et de leurs performances élevées.

Chapitre 2 Microcontrôleur ATmega328

2.1 Introduction

L'ATmega328 est un microcontrôleur 8 bits de la famille AVR développé par la compagnie ATMEL. AVR[3] est le terme utilisé pour désigner une architecture Harvard modifiée 8 bits RISC mise au point en 1996. L'ATmega328 possède plusieurs fonctionnalités qui en font actuellement l'un des microcontrôleurs les plus populaires au monde.

2.1.1 Caractéristiques générales de l'ATmega328

2.1.1.1 Architecture RISC

Les microcontrôleurs AVR exécutent les instructions en un cycle d'horloge. Ils utilisent un pipeline RISC de trois étapes :

1- Instruction Fetch : lors du premier cycle d'horloge, l'instruction est récupérée de la mémoire vers l'unité de prédécodage et le compteur ordinal est incrémenté pour pointer vers l'instruction suivante.

[3] ATMEL n'a jamais levé l'ambiguïté sur la signification de l'acronyme AVR, mais il est trivialement interprété « Advanced Virtual RISC ».

2- Decode : au cours du deuxième cycle d'horloge, l'instruction est décodée au sein de l'unité de prédécodage pour déterminer quelle opération doit être effectuée.
3- Execute : lors du troisième cycle d'horloge, l'opération identifiée dans l'étape de décodage est effectuée et les résultats sont stockés dans le registre de destination.

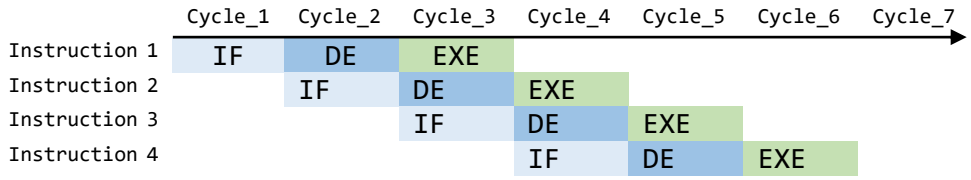


Figure 2.1 Séquence d'exécution des instructions sur une CPU AVR (IF : Instruction Fetch, DE : Decode, EXE : Execute).

Le pipeline AVR RISC utilise également des instructions à taille fixe de 16 bits. Les instructions sont divisées en deux parties de 8 bits : l'opcode (code opération) et l'opérande (données d'instruction).

2.1.1.2 Fréquence d'horloge

L'ATmega328 dispose d'un oscillateur interne RC et peut être alimenté par un oscillateur externe pour améliorer la précision de l'horloge.

Oscillateur interne : il génère un signal d'horloge avec une fréquence nominale de 8 MHz et a une précision typique de ±10%, mais il peut être calibré pour une plus grande précision. Il dispose également d'un diviseur de fréquence qui peut être ajusté pour économiser de l'énergie.

Oscillateur externe : il peut être un cristal, un oscillateur à quartz ou un oscillateur à céramique. La fréquence de l'oscillateur externe doit être comprise entre 0,4 MHz et 16 MHz pour garantir un fonctionnement correct de l'ATmega328, mais elle peut aller jusqu'à 20 MHz.

2.1.1.3 Mémoire

Le microcontrôleur ATmega328 dispose d'une mémoire programme de type flash de 32 Ko, qui peut être programmée jusqu'à 10 000 fois, d'une mémoire vive (SRAM) de 2 Ko et d'une mémoire EEPROM de 1 Ko. La mémoire flash est utilisée pour stocker le code exécutable, la SRAM est utilisée pour stocker les données temporaires, tandis que l'EEPROM est utilisée pour stocker les données non-volatiles (mots de passe, paramètres du programme, etc.).

2.1.1.4 Compteurs/Timers

L'ATmega328 dispose de trois timers/compteurs. Deux d'entre eux sont des compteurs 8 bits, tandis que le troisième est un compteur 16 bits. Ces

compteurs peuvent être utilisés pour mesurer le temps, générer des signaux PWM (Pulse width Modulation), etc.

2.1.1.5 Entrées/sorties

Il a 23 E/S programmables organisée en 3 ports. Ces E/S peuvent être configurées pour fournir des entrées ou des sorties numériques, ce qui permet de communiquer avec des périphériques externes tels que des LED, des capteurs, des moteurs, etc. Parmi ces E/S il y a :

- 6 E/S (8 dans certains modèle) qui peuvent servir comme des entrées analogiques ;
- 5 E/S capables des générer des signaux à modulation de largeur d'impulsion (MLI ou PWM en anglais pour Pulse Width Modulation).

2.1.1.6 Communication série

Il a 3 interfaces de communication série :

- SPI (Serial Peripheral Interface) qui est un bus série synchrone full-duplex opérant en mode maitre/esclave ;
- un USART (Universal Synchronous/Asynchronous Receiver Transmitter);
- une interface TW (Two-wire Serial Interface).

2.1.1.7 Interruptions

Le microcontrôleur ATmega328 dispose de différentes sources d'interruption, qui permettent de répondre rapidement à des événements spécifiques. Les interruptions peuvent être déclenchées par des signaux externes ou des événements internes, tels que des changements d'état sur des broches d'entrée/sortie, des erreurs de communication, des erreurs de calcul, etc.

2.1.1.8 Comparateur

L'ATmega328 possède un comparateur analogique qui permet de comparer deux tensions analogiques et de générer une sortie numérique en fonction de la différence entre ces tensions. Le comparateur peut être configuré pour utiliser l'une des deux entrées analogiques disponibles, ainsi que pour sélectionner un niveau de référence pour la comparaison. Il peut également être configuré pour déclencher une interruption lorsqu'une comparaison est effectuée.

2.2 Description physique de l'ATmega328

2.2.1 Boitiers

Le microcontrôleur ATmega328 se présente sous différentes formes de circuits intégrés : boitier DIP[4] de 28 pattes (ou broches), boitier QFP[5] (carré) de 32 pattes et boitier MLF[6] (plat sans broches) de 28 ou 32 connexions.

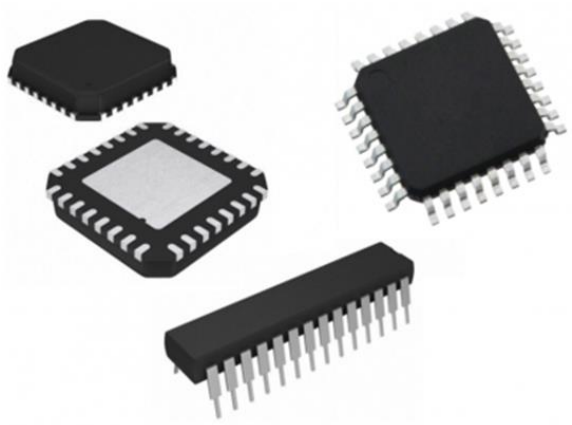


Figure 2.2 Boitiers de l'ATmega328 (de gauche à droite de haut en bas : MLF de 32 connexions, QFP de32 pates, MLF de 32 connexions vue d'en bas et DIP de 28 pattes).

2.2.2 Brochage de l'ATmega328

La figure 2.2 montre la configuration des entrées/sorties de l'ATmega328 en boitiers DIP et QFP. La plupart des pattes ont plusieurs noms car elles sont multifonctions.

a) Alimentations

VCC (patte 7) : Alimentation du circuit intégré ; tension de fonctionnement de l'ATmega328p est de 1.8V à 5.5V.

GND (pattes 8 et 22) : Masse.

AVCC (patte 20) : Alimentation des convertisseurs A/N internes du microcontrôleur. Elle devrait être connectée extérieurement à VCC même si les convertisseurs A/N ne sont pas utilisés. Cependant, dans le cas d'emploi des convertisseur A/N (utilisation des entrées analogiques) elle devrait être connectée à VCC à travers un filtre passe-bas.

[4] Dual Inline Package.
[5] Quad Flat Package.
[6] Micro-LeadFrame.

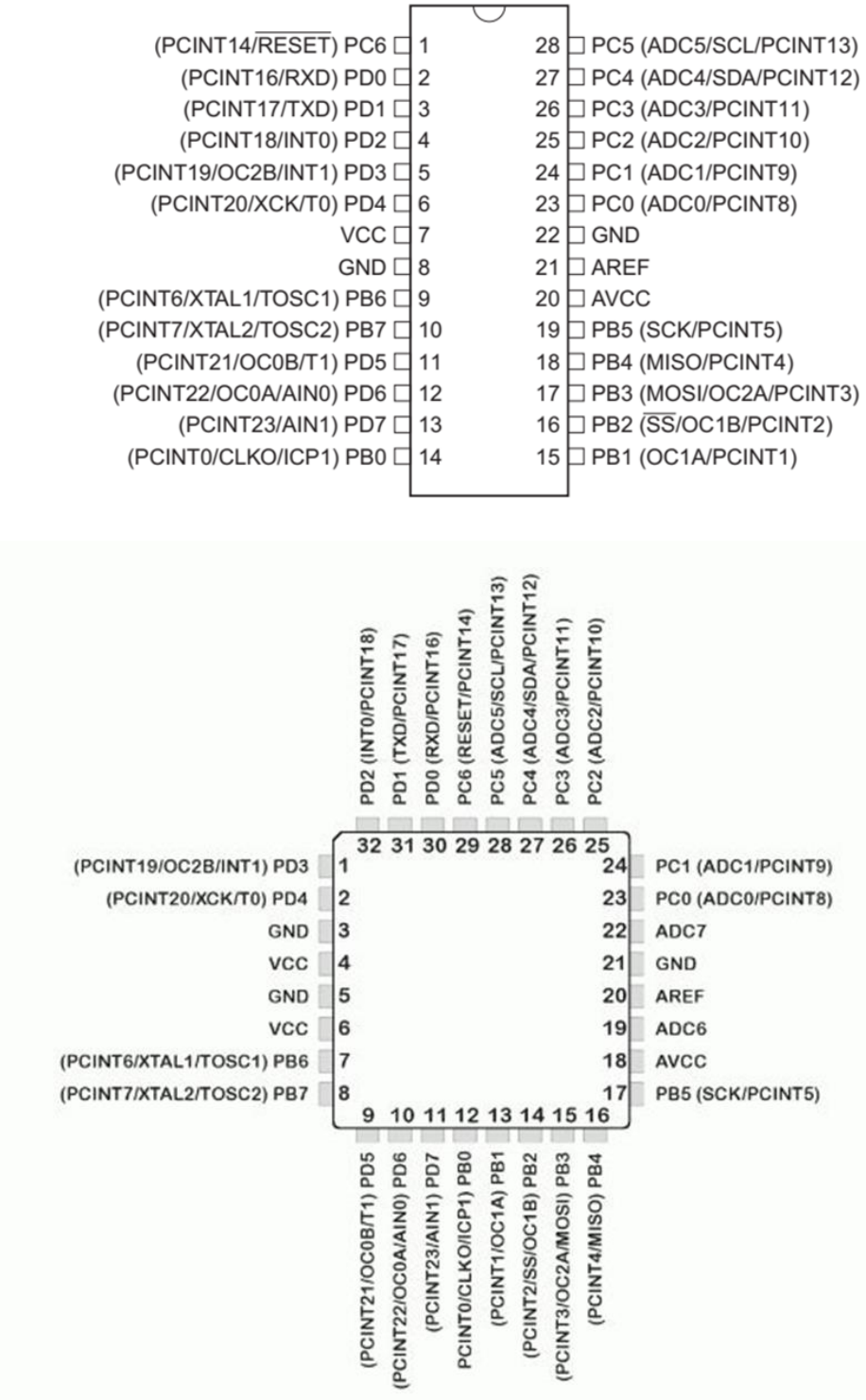


Figure 2.3 Brochage typique de l'ATmega328 en boitiers DIP et QFP 32 pates.

AREF (patte 21) : C'est l'entrée dont la tension est prise comme référence par le convertisseur A/N, c-à-d toutes valeurs à une entrée analogique qui dépasse la tension appliquée à AREF sera convertie à une valeur numérique proche de 3FF H.

b) Ports entrées/sorties

Port B (PB (0 :7)) : Port E/S numériques bidirectionnel à 8 bits avec des résistances internes de tirage pull-up qui peuvent être choisies différemment pour chaque bit par le programme.

Port C (PC (0 :5)) : Port E/S bidirectionnel à 6 bits avec des résistances internes de tirage. Chaque bit de ce port peut servir comme une entrée analogique convertie en valeur numérique sur 10 bits (peut prendre 1024 valeurs).

Port D (PD (0 :7)) : Port E/S numériques bidirectionnel à 8 bits possédant les mêmes caractéristiques du port B mais bien entendu avec des fonctions secondaires différentes.

PC6/RESET (patte 1) : C'est une E/S utilisée pour réinitialiser le microcontrôleur. Le maintien d'un niveau logique bas sur cette E/S pour une durée minimum de 50 ns provoque un reset du système.

XTAL1 et XTAL2 : Respectivement, entrée et sortie pour générer le signal d'horloge, un quartz est placé entre ces deux broches avec deux condensateurs (de 22pF).

N.B.
L'ATmega328 en boitiers MLF et QFP avec 32 pins possède deux entrées analogiques de plus qu'aux boitiers avec 28 pins, à savoir ADC6 et ADC7

2.3 Organisation interne de l'ATmega328

2.3.1 Unité centrale de traitement (CPU)

La CPU de l'ATmega328 est constituée de :

- L'unité arithmétique et logique (ALU) chargée d'exécuter les opérations mathématiques et logiques.

- 32 registres de travail de 8 bits chacun (général purpose registers) nommés de R0 à R31.

- Compteur ordinal ou pointeur d'instruction (PC : Program Counter) de 14 bits qui contient l'adresse mémoire de l'instruction à exécuter.

- Décodeur d'instruction (Instruction Decoder) qui a pour rôle de décoder les instructions.

- Un registre d'état de 8 bits nommé SREG (State REGister)

- Un pointeur de pile.

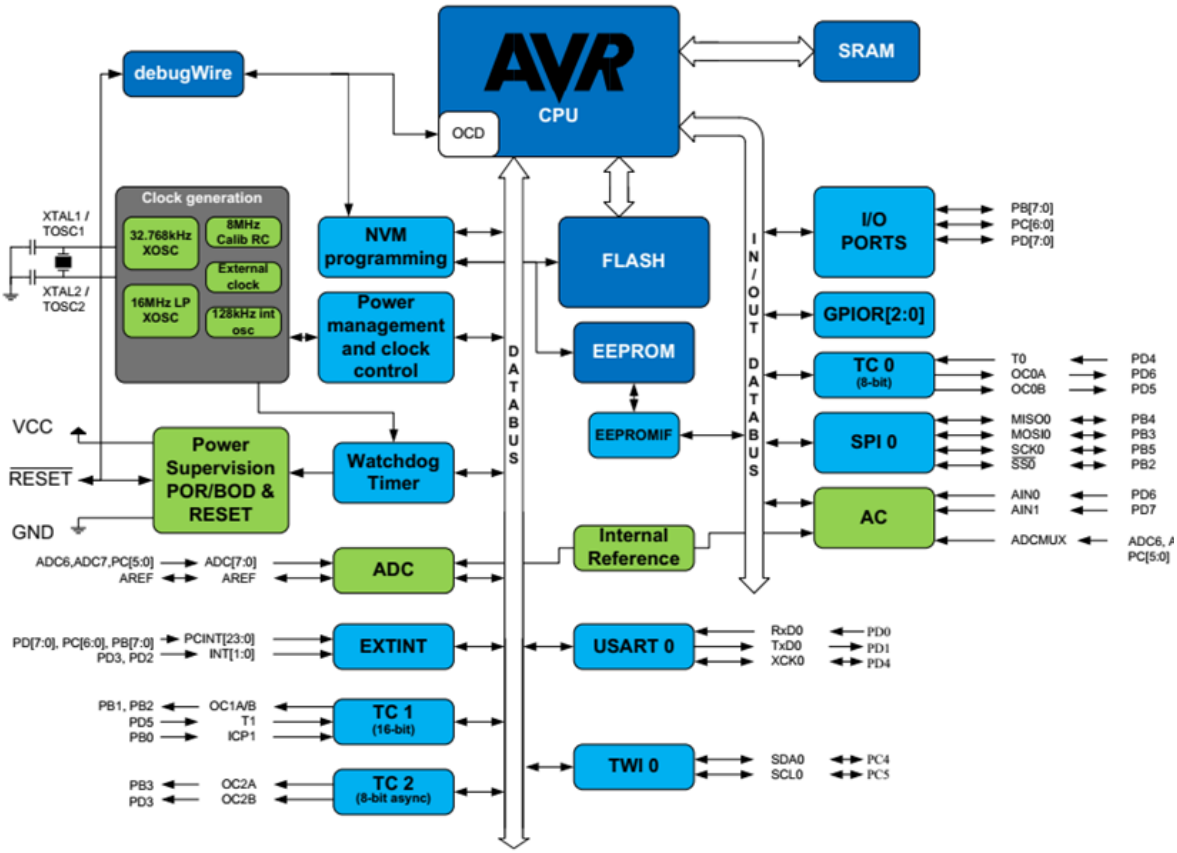


Figure 2.4 Organisation interne de l'ATmega328.

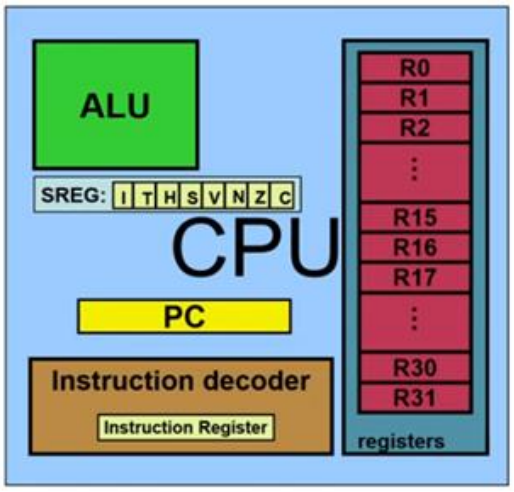


Figure 2.5 CPU de l'ATmega328.

2.3.1.1 Unité Arithmétique et Logique (ALU : Arithmetic and Logic Unit)

Elle permet d'effectuer non seulement des opérations arithmétiques et logiques sur deux opérandes de 8 bits pour obtenir un résultat sur 8 bits, mais aussi des opérations sur un seul bit.

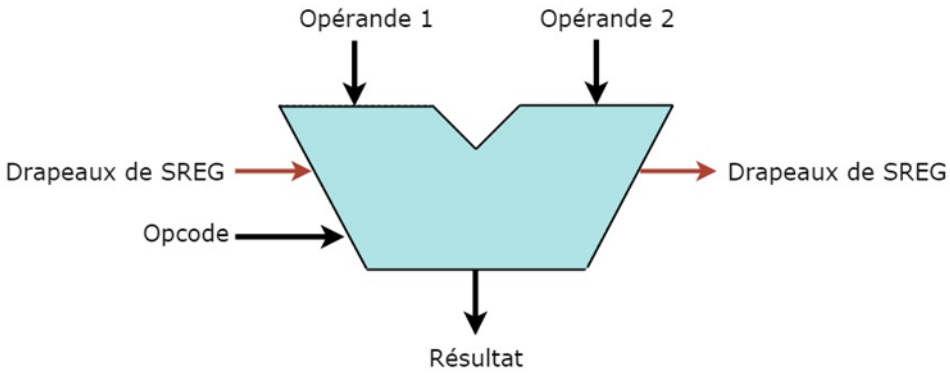


Figure 2.6 Unité arithmétique et logique de l'ATmega328.

2.3.1.2 Registre d'état SREG

Le registre SREG est constitué de 8 bits dont chacun sert comme un drapeau pour indiquer des informations additionnelles concernant la dernière instruction exécutée comme par exemple la retenue, résultat nul, résultat négatif…etc.

Le rôle des drapeaux est de permettre à la CPU de prendre des décisions lors de l'exécution de certaines instructions conditionnelles. Le contenu de SREG est mis ajour après l'exécution de chaque instruction.

I	Global Interrupt Enable
T	Copy Storage
H	Half Carry Flag
S	Sign Flag
V	2's Complementary Overflow Flag
N	Negative Flag
Z	Zero Flag
C	Carry Flag

Figure 2.7 Registre d'état SREG.

Le tableau suivant décrit le rôle de chacun des drapeaux de SREG.

Bit	Description
I	**Global Interrupt Enable :** doit être mis à 1 pour que les interruptions peuvent prendre lieu. Ce bit peut être paramétrer indépendamment des bits d'activation individuelle des interruptions
T	**Copy Storage :** ce bit est utilisé comme source ou destination dans les opérations bit à bit
H	**Half Carry Flag :** bit indicateur de demi-retenue qui indique qu'une retenue a été générer sur les quatre bits de poids faible du résultat d'une opération arithmétique. Il est principalement utilisé dans l'arithmétique BCD.
S	**Sign Flag :** ce bit est le résultat d'un ou exclusif entre les drapeaux N et V.
V	**2's Complementary Overflow Flag :** est mis à 1 lorsque le bit de poids fort change suite à une opération d'addition de deux nombres du même signe ou une opération de soustraction de deux nombres de signes opposés.
N	**Negative Flag :** indique un résultat négatif dans les opérations arithmétiques, logiques ou bit à bit.
Z	**Zero Flag :** mis à 1 lorsque le résultat d'une opération est nul.
C	**Carry Flag :** indique qu'une opération d'addition ou de soustraction a généré une retenue.

Tableau 2.1 Description des drapeaux de SREG.

2.3.1.3 Registres à usage général (GPR : Général Purpose Registers)

Les 32 registres à usage général (également appelé fichier de registres et registres à accès rapide) sont les registres de travail de 8 bits chacun nécessaires à de nombreuses instructions. Toutes les instructions arithmétiques et logiques passent par ces registres. Ils occupent la plage d'adresse de 0000→001F.

Les registres R26 à R31 peuvent être utilisés en pairs pour former 3 registres de 16 bits nommés registre X, registre Y et registre Z comme illustré dans la figure 2.7 et qui sont utilisés comme des pointeurs dans l'adressage indirect.

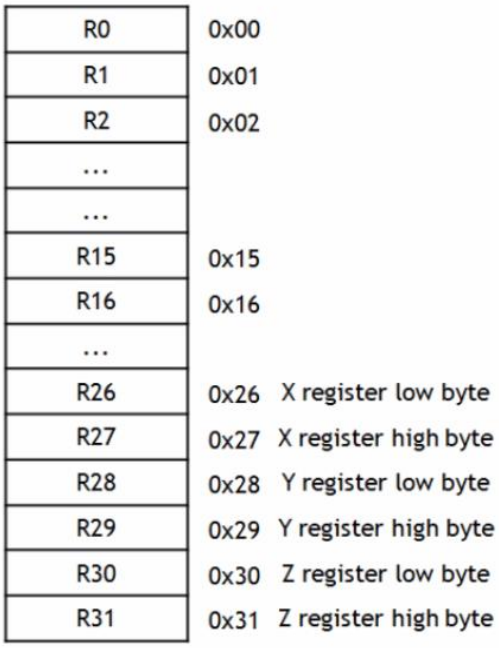


Figure 2.8 Registres à usage général (GPR).

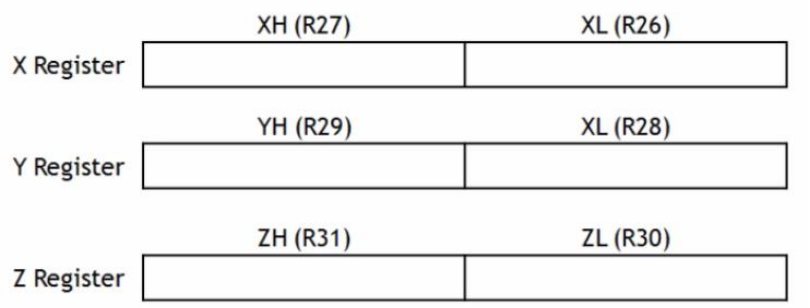


Figure 2.9 Registres X,Y et Z.

2.3.2 Mémoire

L'ATmega328 possède :

- Une mémoire programme de type flash de 32 Ko.
- Une mémoire de données de 2.25 Ko organisée en 3 zones.
- Une EEPROM de 1 Ko.

2.3.2.1 Mémoire programme (flash)

La mémoire flash de l'AVR dispose d'une capacité de 32 Ko et peut être reprogrammée environ 10 000 fois, ce qui correspond à une endurance de 10 k cycles d'effacement/écriture. Étant donné que les instructions de l'AVR sont codées sur 2 ou 4 octets, la mémoire flash est organisée en cases de 16 bits et le bus de données qui la relie à la CPU est également de 16 bits. Par

conséquent, la taille de la mémoire flash est de 16 k cases mémoire, ce qui explique la taille du compteur ordinal PC qui est de 14 bits.

La mémoire flash est divisée en deux zones distinctes : la zone du bootloader et la zone du programme d'application.

Le bootloader est un programme qui permet la programmation du microcontrôleur via une interface de communication série, telle qu'un port USB, depuis un ordinateur. Sans le bootloader, il serait nécessaire d'utiliser un programmateur de microcontrôleur pour écrire les instructions dans la mémoire flash du microcontrôleur.

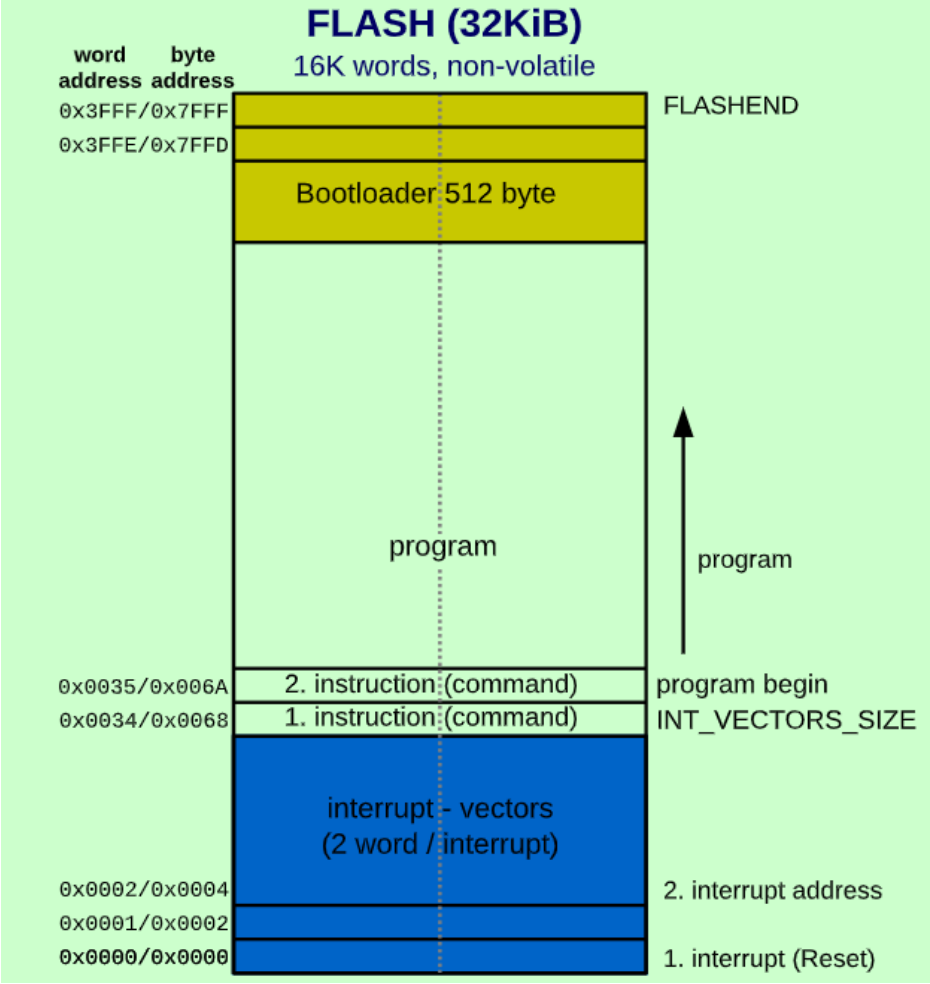


Figure 2.10 Mémoire programme de l'ATmega328.

2.3.2.2 Mémoire de données

La mémoire de données est constituée de 6 zones différentes :

- La première zone est constituée des registres à usage général (GPR) permettant les opérations de l'unité ALU et dont la plage des adresses est de 0000H→001FH.

- Après les GPR, il y a une zone de 64 octets (0020H→005FH) de registres spéciaux (SPR : Spacial function Registers) appelés aussi registres E/S (I/O registers). Ce sont principalement des registres de contrôle et de configuration, mais il y a aussi le registre d'état et le pointeur de pile. Ces registres sont à accès direct et 32 parmi eux peuvent être accédés par bit.

- L'espace mémoire 0060H→00FFH contient 160 registres supplémentaires SPR (extended SPR). Il est a noté que l'accès à ces registres est plus lent que les SPR.

- La mémoire données (SRAM) réelle (sans fonction particulière et qui est destinée au travail avec les données utilisateurs) a la plage d'adresse 0100H→08FFH. Elle contient aussi la pile.

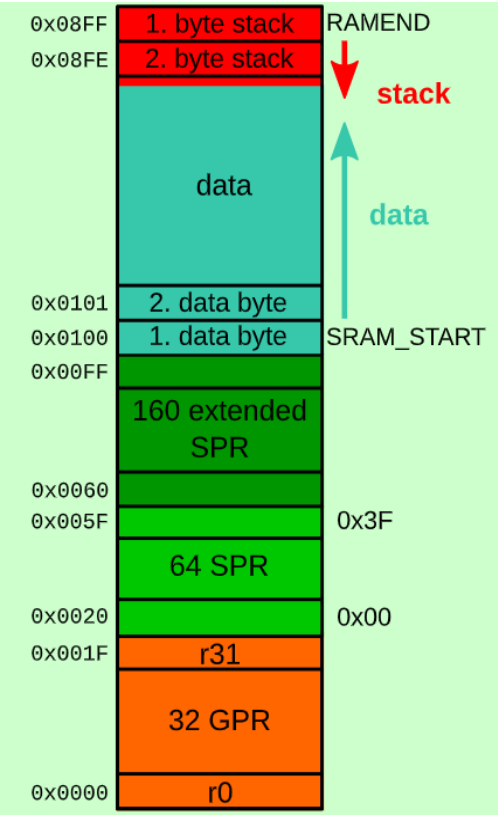


Figure 2.11 Mémoire de données.

2.3.2.3 Mémoire EEPROM

De taille de 1 Ko (plage de 000H→3FFH) organisée en octet et d'une endurance qui dépasse 100000 cycles d'effacement/écriture. Elle est utilisée pour

stocker les données importantes qui doivent être gardées même après la mise du système en dehors d'alimentation. Cette mémoire ne constitue pas une suite du SRAM et par conséquent sa plage des adresses est 0x0000→0x03FF.

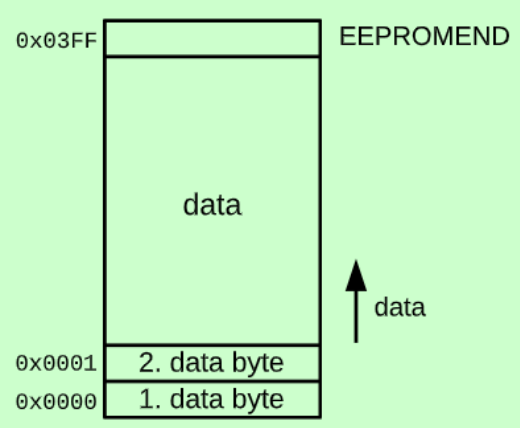


Figure 2.12 EEPROM.

2.3.3 Watchdog Timer (chien de garde ou timer de surveillance)

Le « Watchdog timer» est un circuit temporisateur mis en place pour redémarrer le microcontrôleur si ce dernier se bloque à cause d'une chute de tension, parasites, erreur de structuration, etc. Il est basé sur le principe que chaque étape du traitement doit s'exécuter en un temps maximal.

Le Watchdog Timer de l'ATmega328 peut être configuré pour déclencher une interruption ou une réinitialisation matérielle du microcontrôleur en cas de dépassement d'une certaine durée de temps.

Chapitre 3 Les entrées/sorties de l'ATmega328

3.1 Introduction

Les entrées/sorties (E/S) des microcontrôleurs sont employées pour échanger des données, soit en parallèle, soit par bit avec différents dispositifs, tels que des claviers, des afficheurs, des capteurs, des actionneurs, etc.

L'ATmega328 dispose de 23 broches E/S organisés en trois groupes appelés "ports". Le port B comprend 8 bits, le port C en comprend 7 et le port D en comprend 8. Les fonctions principales de ces ports sont les suivantes :

- **E/S numériques** : ils permettent la communication numérique avec des appareils et des composants externes.
- **Entrées analogiques** : les 6 broches PC0 à PC5 (broches du port C) peuvent être utilisées comme entrées analogiques, ce qui permet de lire des signaux analogiques provenant de capteurs ou d'autres sources.
- **Sorties PWM** : les 6 broches PD3, PD5, PD6, PB1, PB2 et PB3 peuvent être configurées pour effectuer des sorties PWM (Pulse Wide Modulation, modulation de largeur d'impulsion), ce qui permet de contrôler avec précision des moteurs, des LED et d'autres dispositifs.

- **Entrées des interruptions :** les 23 broches E/S peuvent être également configurées comme entrées d'interruptions pour réagir rapidement à des événements spécifiques.

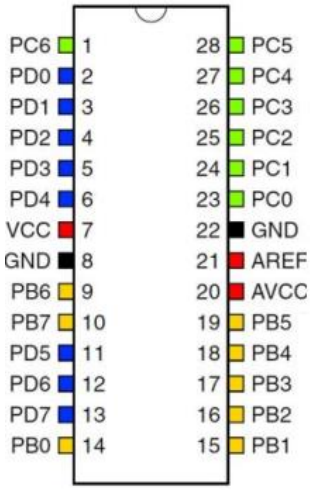


Figure 3.1 Port E/S de l'ATmega328.

3.2 Programmation des entrées/sorties numériques de l'ATmega328

3.2.1 Broches E/S et résistances de tirage

Une entrée/sortie peut être programmée individuellement ou avec tout l'ensemble de son port pour fonctionner soit comme entrée ou comme sortie. Si une broche est programmée pour fonctionner comme entrée est laissée flottante, c.à.d. n'est pas associée à une tension particulière, alors sa valeur ne peut être précisée et peut changer aléatoirement.

Afin de bien expliquer le concept des résistances de tirage et des résistances de rappel, on considère le schéma de la figure ci-dessous.

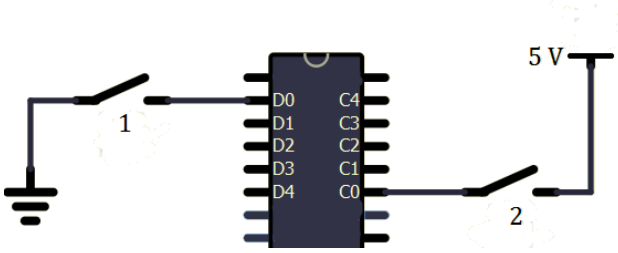


Figure 3.2 Branchement des entrées sans résistances de tirage et de rappel.

Evidemment les broches PD0 et PC0 sont des entrées, chacune est utilisée pour détecter la fermeture de l'interrupteur correspondant : la fermeture de l'interrupteur 1 doit normalement provoquer le passage de PD0 de 1 à 0 et la fermeture de l'interrupteur 2 doit faire passer PC0 de 0 à 1. Cependant,

pendant que les interrupteurs sont ouverts les tensions des entrées PD0 et PC0 ne sont pas assurées et font l'objet de plusieurs phénomènes (ne peuvent être identifiées si elles correspondent aux niveaux 1 ou 0).

Il est donc vivement recommandé de tirer PD0 vers Vcc par une résistance de tirage (pull-up) et de rappelé PC0 à GND par une résistance de rappel (pull-down) comme le montre la figure 3.2.

L'ATmega328 possède des résistances de tirage internes qui peuvent être sélectionnées en paramétrant des registres des ports E/S qu'on va étudier dans la section suivante.

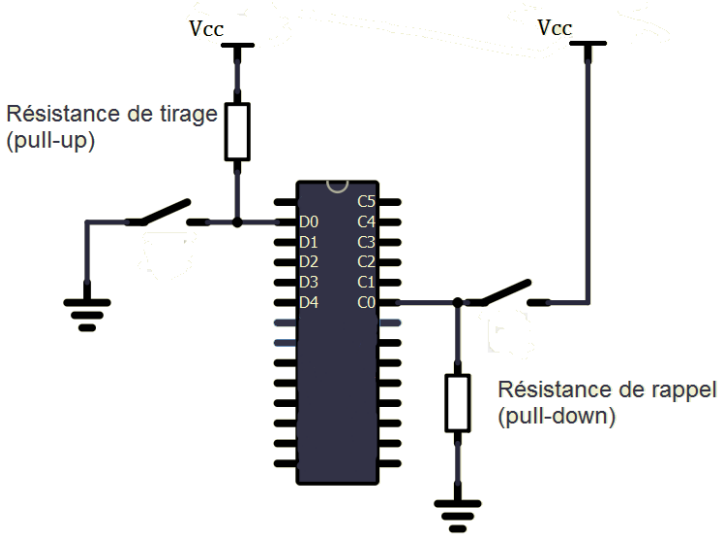


Figure 3.3 Branchement des résistances de tirage et de rappel.

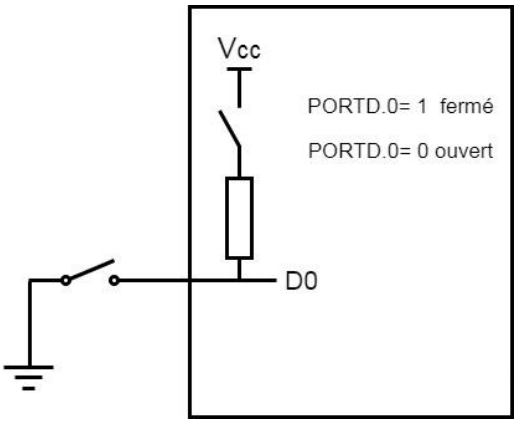


Figure 3.4 utilisation d'une résistance de tirage interne.

3.2.2 Registres des ports E/S

Pour chacune des broches des ports E/S, on peut choisir les quatre états : entrée à haute impédance[7], entrée avec résistance pull-up, sortie à l'état *0* et sortie à l'état *1*. Ces combinaisons peuvent être choisis en configurant les registres de gestion des E/S.

A chacun des port B, C et D sont associé trois registres de gestion des E/S numériques :

- **DDR** (***D**ata **D**irection **R**egister,* registre de direction) : DDRB, DDRC et DDRD sont les registres qui précisent pour chaque broche des 3 ports si elle est une entrée ou une sortie. Une broche est mise en sortie si le bit du registre DDR qui lui correspond est mis à 1 et elle et mise en entrée si ce bit est à *0*.
- **PORT** (Port driver register, registres de sortie PORTB, PORTC et PORTD) : Lorsqu'une broche est mise en sortie par DDR, ce registre précise son état de sortie, *0* ou *1*. Pour les broches mises en entrées, il active les résistances pull-up correspondantes s'il est mis à 1.
- **PIN** (port pins register): Les registrent PINB, PINC et PIND indiquent l'état de chacune des broches des trois ports. Il n'est possible de changer le contenu de ces registres par le programme.

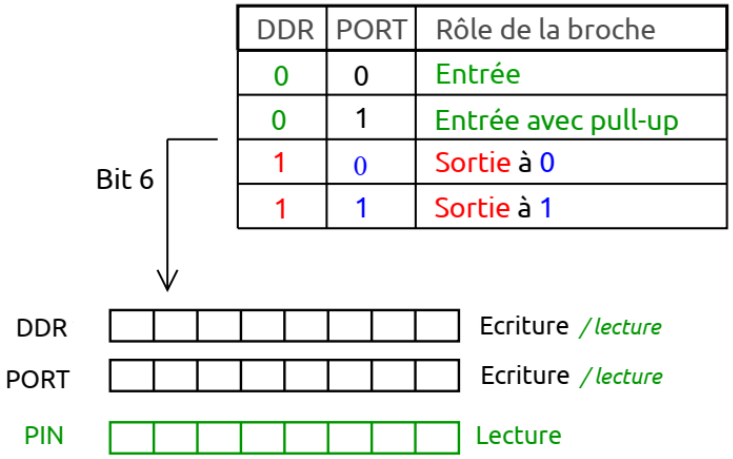


Figure 3.5 Fonction des E/S numériques selon DDR, PORT et PIN.

[7] Une entrée à haute impédance est une entrée qui ne consomme que très peu de courant de la source qui y est connectée.

3.2.3 Lecture et écriture d'un octet sur un port

Après la mise sous tension du microcontrôleur ou un *RESET*, tous les bits des registres DDR prennent la valeur *0*, c.à.d. les bits de tous les port E/S sont par défaut des entrées. Afin de convertir des broches en sortie, il faudra écrire des *1* dans les bits des registres DDR correspondants. L'exemple suivant montre le code C pour écrire un octet sur le port B.

Exemple 1

```
DDRB=0xff;         // met tous le port B en sortie.
DDRB=0b01000001;  // met les broches PB0 et PB6 en sortie et reste en entrées.
DDRD=0b11110011 ;// met PD2 et PD3 en entrée et le reste en sortie.
PORTD=0b11111011 ;//place une résistance de tirage à PD3 (entrée) et place
//tous les sorties à 1.
```

Généralement le choix du mode des E/S se fait souvent au début du programme. Cependant, il est possible de changer le mode d'une broche d'une entrée à une sortie ou l'inverse à n'importe quelle partie du programme.

Il est possible de lire la valeur d'un port indépendamment du mode assigné à ses broches (entrée ou sortie), en lisant le port PIN correspondant.

Exemple 2

```
DDRB=0xff;          // met tous le port B en sortie.
PORTB=0x45 ;       // permet de placer 45H dans PORTB. En d'autres termes, écrit 45H dans le port B.
Variable1=PINB;     //met le contenu du port B dans Variable1 (Variable1=45H).
```

3.2.4 Lecture et écriture d'un bit

Si ***CodeVision AVR*** accepte la syntaxe « DDRB.x, PORTD.x, PINC.x » pour désigner individuellement les bits des registres E/S, d'autres compilateurs (comme celui utilisé dans ***ATMEL Studio***) n'accepte pas cette syntaxe et ne permet pas de traiter séparément les bits des registres E/S. Pour surmonter cette difficulté, on doit utiliser les trois opérateurs logiques suivants opérant au niveau du bit:

- **Le *OU* logique :** L'opérateur se note *|*. Il prend comme opérandes deux valeurs binaires. L'opération OU s'effectue entre chacun des bits de même rang des deux opérateurs.
- **Le *ET* logique :** L'opérateur se note *&*. Il s'applique également entre deux valeurs binaires bit par bit.
- **L'inversion logique :** L'opérateur se note *~*. Il s'applique à une valeur binaire bit par bit (chaque bit est inversé : *0* devient *1* et *1* devient *0)*.

Il ne faut pas confondre les opérateurs *|*, *&* et *~* avec les opérateurs *||* (opérateur OU), *&&* (opérateur ET) et *!* (opérateur d'inversion), qui agissent

non pas sur chacun des bits des opérandes, mais sur des opérandes considérés comme une seule valeur booléenne : fausse (si la valeur est nulle) ou vraie (si la valeur est non nulle).

3.2.4.1 Mise à 1 de bits

La mise à *1* d'un ou de plusieurs bits dans un registre s'effectue avec l'opérateur OU.

Exemple 3

PORTB = 0b01000100; // Met les bits 2 et 6 du registre PORTB à 1 et les autres bits à 0.

PORTB |= 0b01000100; //Met les bits 2 et 6 du registre PORTB à 1 et laisse les autres bits inchangés.

PORTB|=(1<<2 | 1<<6) ; // Met les bits 2 et 6 du registre PORTB à 1 et laisse les autres bits inchangés.

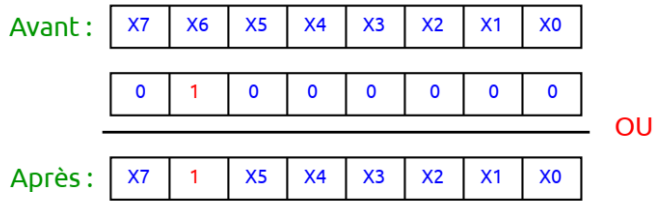


Figure 3.5 Mise à 1 d'un bit dans un registre.

3.2.4.2 Mise à 0 de bits

C'est l'opérateur ET qui permet la mise à *0* d'un ou de plusieurs bits dans un registre.

Exemple 4

PORTB &= 0b10111011; // Met les bits 2 et 6 du registre PORTB à 0 et les autres bits à 1.

PORTB &= 0b10111011; // Met les deux bits 2 et 6 à 0 en laissant les autres inchangés.

PORTB &=~(1<<2 | 1<<6) ; // Met les deux bits 2 et 6 à 0 en laissant les autres inchangés.

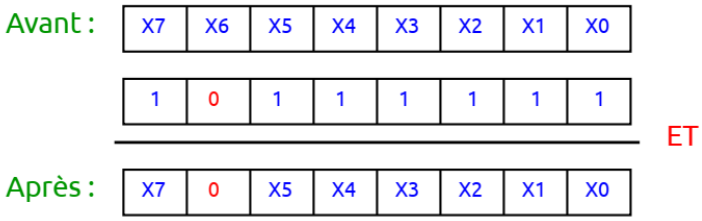


Figure 3.6 Mise à 0 d'un bit dans un registre.

3.2.4.3 Test de bits

Le test d'un bit peut se faire avec l'opérateur ET.

Exemple 5

```
PORTB & 0b00000100; // rend une valeur nulle si le bit 3 est à 0 et une
//valeur non nulle si le bit est à 1.

PORTB & (1<<2); // Effectue la même opération que l'instruction précédente.
```

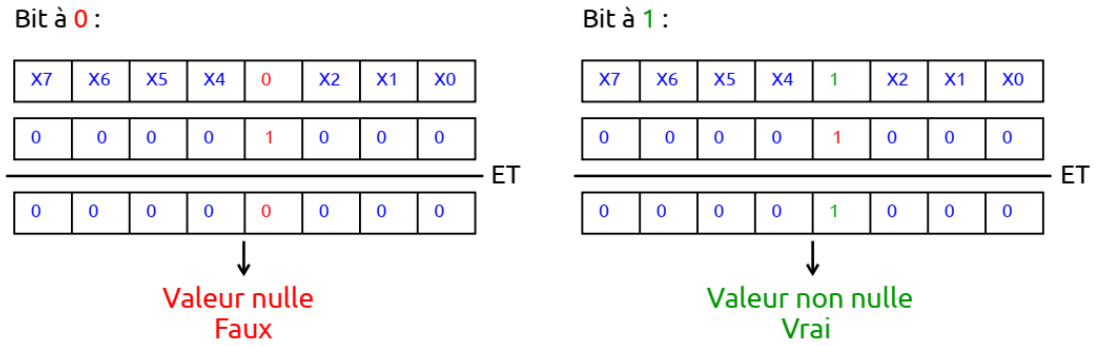
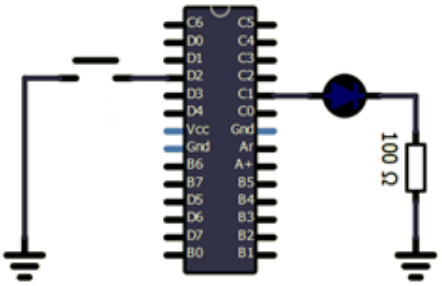


Figure 3.7 Test d'un bit.

Exemple 6

Ecrire le programme en langage C qui allume la LED pendant une durée de 1 seconde lorsque le bouton-poussoir est enfoncé.




```
#include <io.h>
#include <delay.h>

void main(void)
{
   DDRC.1 = 1;  // configure PC1 en sortie pour la LED
   DDRD.2 = 0; // configure PD2 en entrée pour le bouton poussoir
   PORTD.2 =1; // active une résistance de tirage interne à PD2

   while (1)    // boucle infinie
   {
      if (PINB.2 == 0) // Si le bouton-poussoir est enfoncé
      {
         PORTC.1 = 1; // Allume la LED
         delay_ms(1000); // Attend 1 seconde
         PORTB.4 = 0; // Éteint la LED
       }
   }
}
```

Pour les compilateurs qui n'accepte pas la syntaxe « DDRB.x, PORTD.x, PINC.x », les bits ne doivent pas traités séparément. Le programme précédent devient :

```
#include <io.h>
#include <delay.h>

void main(void)
{
   DDRC |=(1<< 1);       // configure PC1 en sortie pour la LED : DDRC=DDRC | (0b00000010)
   DDRD& =~(1<<2);       // configure PD2 en entrée pour le bouton poussoir : DDRD=DDRD & ~(0b00000100)
   PORTD|=(1<<2);        // active une résistance de tirage interne à PD2 : PORTD=PORTD | (0b00000100)

   while (1)    // boucle infinie
   {
      if ( !(PINB & (1<<2)))      // Si le bouton-poussoir est enfoncé : si ( !(PINB & 0b00000100))
```

```
    {
      PORTC|=(1<<1);      // Allume la LED :  PORTC= PORTC | (0b00000010)
      delay_ms(1000);       // Attend 1 seconde
      PORTB&=~(1<<1);             // Éteint la LED : PORTC= PORTC & ~(0b00000010)
    }
  }
}
```

3.3 Entrées analogiques

Les broches du port C, PC0 à PC5, peuvent être utilisées comme des entrées analogiques, mais seulement une broche à la fois. La tension mesurée en entrée est convertie en une valeur numérique sur 10 bits.

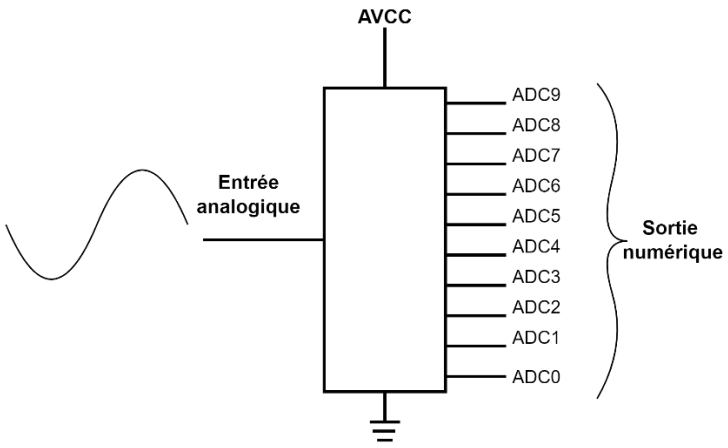


Figure 3.8 Convertisseur analogique-numérique de résolution de 10 bits.

La formule qui permet de calculer la valeur numérique ADC en sortie du convertisseur analogique-numérique du signal converti se calcul en utilisant la relation suivante :

$$ADC = \frac{V_e}{V_{ref}} 1023$$

Où V_e est la tension à l'entrée analogie et V_{ref} est la tension de référence. Le temps de conversion peut aller de 13 à 260 µs.

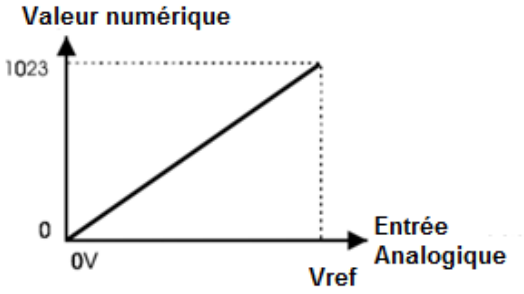


Figure 3.9 La relation linéaire entre la tension de l’entrée analogique du convertisseur et la valeur numérique générée.

3.3.1 Tension de référence

Les convertisseurs analogiques numériques ont besoin de leur fournir la plage des tensions avec laquelle ils doivent fonctionner. Cette plage spécifie la référence basse et la référence haute pour lesquelles le convertisseur attribuera, respectivement, la valeur numérique minimale et maximale. Si la tension de l'entrée analogique est égale à la référence basse, la sortie numérique est 0 et si elle est égale à la référence haute, elle produit le nombre numérique maximal (3FF pour l’ATmega328). Le convertisseur attribuera de manière linéaire les tensions d’entrée entre la valeur numérique basse et haute.

Pour l'ATmega328P, la référence basse est fixée à la masse, mais la référence haute peut être sélectionner des trois sources suivantes.

AREF : C’est la broche 21 sur le boitier DIP du microcontrôleur qui peut être utilisée pour fournir la tension de référence souhaitée, à condition qu'elle se situe dans la plage de 1,0V à VCC.

AVCC : C’est la broche 20 sur le boitier DIP du microcontrôleur. Son rôle est d’alimenter en énergie le circuit du convertisseur AN. Mais elle peut servir également comme tension de la référence haute. Généralement, on la connecte à VCC (alimentation du microcontrôleur) via un filtre passe-bas.

1.1V : L’ATmega328 dispose d'une tension de référence interne de 1,1V.

3.3.2 Fréquence de la conversion A/N

Le convertisseur A/N nécessite un signal d'horloge dont la fréquence doit être comprise entre 50 kHz et 200 kHz. Toutefois, l'horloge système du microcontrôleur peut être trop rapide pour le convertisseur. Pour remédier à ce problème, la puce intègre un "*prescaler*" réglable qui permet de diviser la fréquence de l'horloge système à une valeur acceptable. Le *prescaler* offre la possibilité de configurer une division par 2, 4, 8, 16, 32, 64 ou 128.

3.3.3 Registres

Registre ADMUX (Multiplexer Selection Register)

Le registre ADMUX contient trois champs de bits pour contrôler différents aspects de la conversion des données.

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

REFS1 et REFS2 : ces eux bits permettent de sélectionner la source de la tension de référence.

REFS1	REFS0	Source de la tension de référence
0	0	AREF
0	1	AVCC (AREF connectée à AVCC via un filtre passe-haut)
1	0	Réservé
1	1	Tension de référence interne de 1.1 v

ADLAR : affecte la présentation de la valeur convertie dans le registre de conversion. Il détermine la justification du résultat à gauche ou à droite.

MUXn : ces bits sont utilisés pour sélectionner l'entrée analogique à connecter au convertisseur A/N.

MUX3	MUX2	MUX1	MUX0	Entrée analogique
0	0	0	0	PC0
0	0	0	1	PC1
0	0	1	0	PC2
0	0	1	1	PC3
0	1	0	0	PC4
0	1	0	1	PC5
1	0	0	0	Capteur de température interne[8]

Registre ADCSRA (Control and Status Register A)

L'ATmega328P dispose de deux registres d'état et de contrôle du convertisseur A/N, ADCSRA et ADCSRB. Pour les opérations de base, seuls les bits du registre ADCSRA doivent être modifiés.

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

[8] L'ATmega328 dispose de jonction de diode interne il mesure la température interne du microcontrôleur en utilisant la variation de tension de la jonction de diode.

ADEN : Doit être à 1 pour autoriser la conversion A/N.

ADSC : Ce bit doit être mit à 1 pour démarrer la conversion A/N. Il demeurera dans cet état jusqu'à ce que la conversion soit achevée. En testant l'état de ce bit, le programmeur peut déterminer si la conversion est terminée ou non.

ADATE : Lié au registre ADCSRB.

ADIF : Ce bit est positionné à 1 quand la conversion est terminée.

ADIE : Si ce bit est positionné à 1 et que l'activation globale des interruptions est activée, alors à la fin de la conversion une interruption est déclenchée.

ADPSn : Ces bits sont utilisés pour déterminer le facteur de division du *prescaler* entre la fréquence d'horloge système et la fréquence d'horloge d'entrée du convertisseur A/N.

ADPS2	ADPS1	ADPS0	Facteur de 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

Registre de données ADCL et ADCH (Data Register high and low bytes)

Le résultat de la conversion est stocké dans un registre de données de deux octets (16 bits) que l'on appelle dans un programme ADC ou ADCW. Ce registre est formé par l'octet ADCH qui content les bits les plus significatifs et ADCL qui contient les bits les moins significatifs.

Si l'on souhaite utiliser le résultat complet de la conversion sur 10 bits, on laisse le bit ADLAR=0 (dans le registre ADMUX). Cela provoque le stockage des 8 bits de poids faible du résultat de la conversion dans ADCL et les deux bits de poids fort dans ADCH comme le montre la figure 3.11.

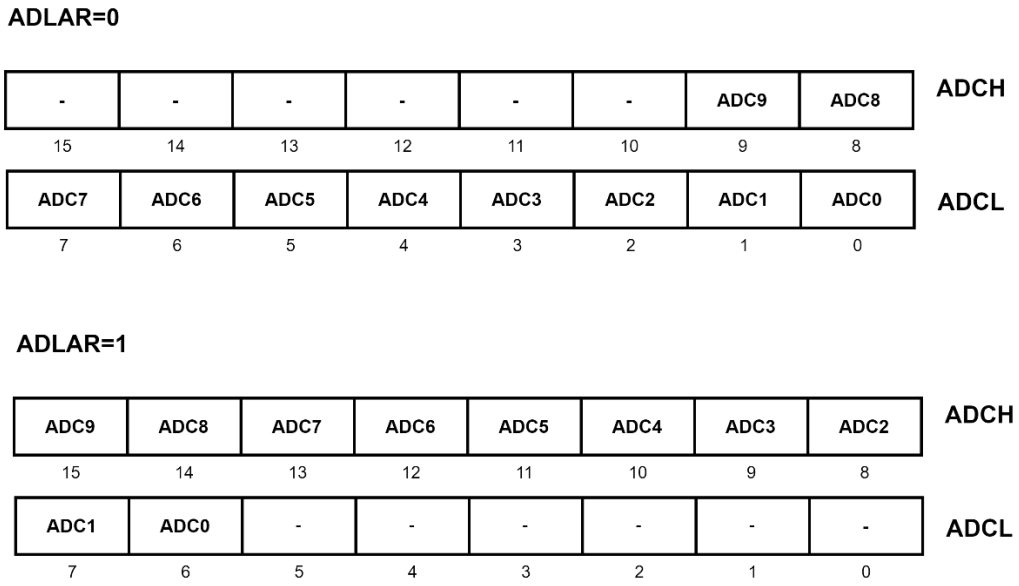


Figure 3.10 Disposition du résultat dans ADCL et ADCH pour ALDAR=0 et ALDAR=1.

Dans de nombreuse situation, seule une valeur de conversion sur huit bits est nécessaire. Dans ce cas, on met ALDAR=1. Le résultat de la conversion s'alignera à gauche, c-à-d les deux bits les moins significatifs se trouvent dans ADCL (bits 1 et 0) et les huit bits les plus significatifs se trouvent dans ADCH (voir la figure 3.11).

Exemple 7

L'entrée PC3 d'un ATmega328 est branchée à la source d'un signal analogique variant de 0 à 5V. On désire convertir ce signal en un signal numérique sur 10 bits avec une référence de tension haute de 5V.

1- Calculer la résolution de la conversion.
2- Quel est le nombre des niveaux de quantification (nombre des valeurs numériques possibles) ?
3- Si le signal d'entrée est de 2,23V, quelle sera la valeur numérique de la conversion ?
4- Si l'on augmente la référence de tension à 7V, quelle sera la nouvelle valeur numérique de la conversion correspondante à l'entrée 2.23V ?
5- Écrire un programme pour effectuer la conversion A/N avec un facteur de division de la fréquence d'horloge système de 128 et envoyer la valeur numérique convertie sur 8 bits au port D.

Solution

1- Résolution =(référence haute-référence basse)/$2^{\text{nbre de bit de la conversion}}$
=(5V-0V)/2^{10}=4.88x10^{-3}V
2- Nbre de quantification=2^{10}=1024
3- ADC==$\frac{V_e}{V_{ref}}$1023=2.23x1023/5=456=0x1C8=0b0111001000

4- ADC=2.23x1023/7=326=0b0101000110
5-

```
#include <io.h>
void main(void)
{
 // Configuration du port D en sortie.
 DDRD=0xFF;

 // Configuration des registres pour la conversion A/N.
 ADMUX=0b01100011; //Tension de Référence AVCC, ADLAR=1 et entrée
                   //analogique PC3

 ADCSRA|=0b10000111; // Activation de l'ADC et configuration du
                      // facteur de division à 128.
 while (1)
    {
     // Démarrer la conversion A/N
     ADCSRA|=(1<<6);
     // Attendre la fin de la conversion.
     while(ADCSRA &(1<<6));
     //Envoyer la valeur numérique sur le port D.
     PORTD=ADCH;
    }
 }
```

Chapitre 4 Interruptions de l'ATmega328

4.1 Bref rappel sur les interruptions

4.1.1 Définitions

Une interruption est un événement qui interrompt momentanément et quasi-instantanément l'exécution d'un programme afin d'exécuter un sous-programme, appelé *routine d'interruption* ou *ISR* (Interrupt Service Routine). Les interruptions permettent aux microcontrôleurs de réagir rapidement à des événements spécifiques, sans avoir à attendre que l'unité centrale ait terminé l'exécution de la tâche en cours.

Un événement déclencheur d'une interruption est appelés *source d'interruption* et l'adresse prédéfinie vers laquelle se déroute le compteur ordinal après le déclenchement d'une interruption est appelée *vecteur d'interruption*.

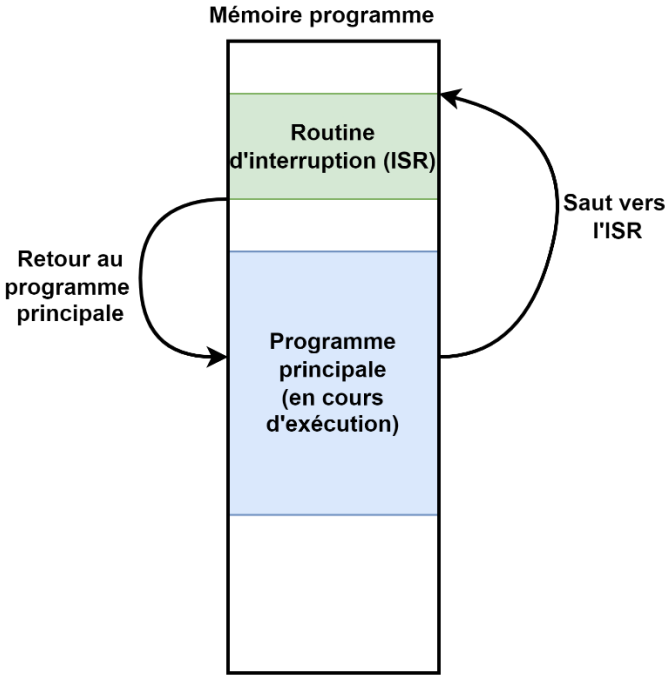


Figure 4.1 Déroulement d'une interruption d'un point de vue de l'exécution d'un programme.

4.1.2 Interruptions internes et externes

Les interruptions se produisent en réponse à deux types d'événements, internes et externes :

- **Interruptions internes :** Elles sont générées par les composants internes du microcontrôleur, et sont déclenchées par des événements tels que des erreurs d'exécution, le convertisseur analogique-numérique, les timers/compteurs, etc.

- **Interruptions externes :** Elles sont générées par des événements externes au microcontrôleur, et sont déclenchés par des événements sur les E/S tels qu'un changement sur une entrée, un front montant ou un front descendant.

4.2 Interruptions de l'ATmega328

4.2.1 Liste des interruptions

L'Atmega328 possède 26 sources d'interruption illustrées dans le tableau ci-dessous, ordonnés par priorité décroissante.

N°	ADRESSE	SOURCE	DESCRIPTION
1	0x0000	RESET	Reset Externe, Power-on reset, Brown-out reset and Watchdog reset.
2	0x0002	INT0	Demande d'interruption externe 0
3	0x0004	INT1	Demande d'interruption externe 1

4	0x0006	PCINT0	Demande d'interruption sur changement d'état d'une broche 0
5	0x0008	PCINT1	Demande d'interruption sur changement d'état d'une broche 1
6	0x000A	PCINT2	Demande d'interruption sur changement d'état d'une broche 2
7	0x000C	WDT	Dépassement durée du Watchdog
8	0x000E	TIMER2 COMPA	L'égalité du Timer/Compteur2 comparateur A
9	0x0010	TIMER2 COMPB	L'égalité du Timer/Compteur2 comparateur B
10	0x0012	TIMER2 OVF	Dépassement du Timer/Compteur2
11	0x0014	TIMER1 CAPT	Evénement Capture Timer/Compteur1
12	0x0016	TIMER1 COMPA	L'égalité du Timer/Compteur1 comparateur A
13	0x0018	TIMER1 COMPB	L'égalité du Timer/Compteur1 comparateur B
14	0x001A	TIMER1 OVF	Dépassement du Timer/Compteur1
15	0x001C	TIMER0 COMPA	L'égalité du Timer/Compteur0 comparateur A
16	0x001E	TIMER0 COMPB	L'égalité du Timer/Compteur0 comparateur B
17	0x0020	TIMER0 OVF	Dépassement du Timer/Compteur0
18	0x0022	SPI, STC	Fin d'envoi série SPI (*Serial transfert complete*)
19	0x0024	USART, RX	Fin de réception USART
20	0x0026	USART, UDRE	Registre de données USART vide
21	0x0028	USART, TX	Fin de transfert USART
22	0x002A	ADC	Fin de conversion numérique (ADC)
23	0x002C	EE READY	EEPROM prêt
24	0x002E	ANALOG COMP	Comparateur Analogique
25	0x0030	TWI	Interface série 2 fils
26	0x0032	SPM READY	Prêt pour le Stockage en mémoire Programme

Les interruptions sont ordonnées dans le tableau selon leurs priorités décroissantes. Par exemple, l'interruption de RESET a la priorité la plus élevée. Si plusieurs interruptions se produisent simultanément, le microcontrôleur traite en premier celle qui a la priorité la plus haute.

Les vecteurs d'interruption, dans la deuxième colonne, sont des adresses prédéfinies de la mémoire programme et ne peuvent pas être changés.

4.2.2 Activation globale des interruptions

Pour qu'un événement déclenche une interruption, il faut que cette dernière soit activée globalement et individuellement.

Le bit I du registre SREG[9] est utilisé pour activer ou désactiver globalement toutes les interruptions. Lorsque ce bit est positionné sur 1, le système d'interruptions est activé et les interruptions peuvent survenir. En revanche, si ce bit est positionné sur 0, toutes les interruptions sont désactivées.

[9] SREG et le registre d'état (voir chapitre 2 page 26-27).

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
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 instructions ci-dessous permettent d'activer et de désactiver le bit I.

```
sei();  // Activation globale des interruptions
cli();  // Désactivation globale des interruptions
```

4.2.3 Déroulement des interruptions

Le déroulement des interruptions de l'ATmega328 se fait selon les étapes suivantes :

1- L'instruction en cours s'achève avant que l'ATmega328 ne prenne en compte l'interruption.
2- Le bit drapeau I (Global Interrupt Enable) de SREG est automatiquement mis à 0 pour empêcher d'autres interruptions d'interrompre l'ISR en cours.
3- Le compteur ordinal PC est sauvegardé dans la pile. En revanche, le registre d'état SREG n'est pas sauvegardé.
4- PC est ensuite chargé avec le vecteur d'interruption correspondant à l'interruption en cours.
5- La routine d'interruption est exécutée jusqu'à sa fin.
6- Le PC est chargé avec sa valeur sauvegardée dans la pile, c'est-à-dire avec l'adresse de retour stockée lors de l'étape 3.
7- Le bit drapeau I est remis à 1 pour permettre la prise en compte d'autres interruptions.
8- Le programme reprend là où il avait été interrompu ; à l'instruction suivant celle qui été en cours d'exécution avant l'interruption.

Ces étapes sont répétées chaque fois qu'une interruption est déclenchée.

> N.B.
> Le registre d'état SREG (Status Register) de l'ATmega328 n'est pas automatiquement sauvegardé sur la pile lors d'une interruption ou d'un appel de sous-routine. Il est donc important de sauvegarder manuellement le contenu du registre SREG avant d'effectuer une opération dans l'ISR qui pourrait modifier son contenu.

4.3 Interruptions externes

Les interruptions externes (PCINT0:2 et INT0:1) de l'ATmega328 sont déclenchées par un changement d'état ou par front montant ou descendant sur les broches configurées en mode d'interruption externe.

Il est important de noter que les interruptions externes peuvent être actives même si les broches concernées sont configurées comme des sorties. Cela

signifie que les broches peuvent être utilisées à la fois comme des entrées pour la détection de l'interruption et comme des sorties pour contrôler des périphériques externes.

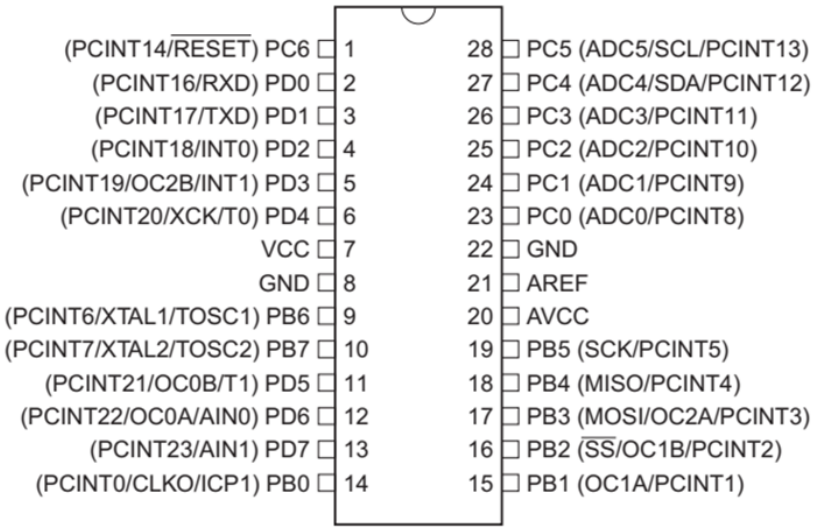


Figure 4.2 Toutes les broches E/S de l'ATmega328 peuvent être employées comme sources d'interruption externes.

4.3.1 INT0 et INT1

Les interruptions externes INT0 et INT1 sont déclenchées lorsque des signaux sont détectés sur les broches E/S numériques PD2 et PD3, respectivement (voir la figure 4.2). Ces interruptions peuvent être configurées pour être déclenchées sur un front montant, un front descendant ou sur les deux. Ces deux interruptions peuvent être utilisées pour détecter des événements externes, tels que des impulsions, des changements d'état ou des signaux d'horloge externes. En général, elles sont plus adaptées pour détecter des événements externes à haute priorité.

4.3.1.1 Registres

La configuration du registre EICRA permet de choisir les types des fronts de déclenchement des interruptions et le registre EIMSK permet de les activer individuellement. La validation globale des interruptions se fait à l'aide du drapeau I du registre SREG.

Registre EICRA (External Interrupt Control Register A)

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Ce registre contient les bits des modes de déclenchement des interruptions.

ISC00 et ISC01 : Ces bits permettent de sélectionner si l'interruption doit être déclenchée par un front montant, un front descendant, un niveau bas ou un niveau haut sur la broche INT0.

ISC10 et ISC11 : Ces bits contrôlent le mode de déclenchement de l'interruption externe INT1.

ISC00	ISC01	DESCRIPTION
0	0	Interruption déclenchée sur un niveau bas de la broche INT0.
0	1	Interruption déclenchée sur tout changement logique de la broche INT0.
1	0	Interruption déclenchée sur un front descendant de la broche INT0.
1	1	Interruption déclenchée sur un front montant de la broche INT0.

Les modes de déclenchement de INT1 par ISC10 et ISC11 sont similaires à ceux de INT0.

Registre EIMSK (External Interrupt Mask)

Il comporte deux bits d'activation des interruptions externes INT0 et INT1.

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Le bit INT0 : Lorsqu'il est à 1 et si le bit I du registre SREG est à un, l'interruption INT0 est active.

Le bit INT1 : Lorsqu'il est à un et si le bit I est à un, l'interruption INT1 est active.

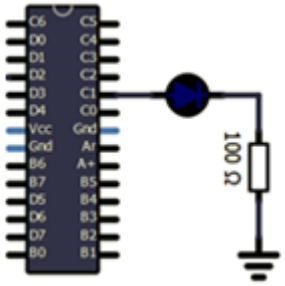
Le registre EIFR (External Interrupt Flag Register)

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Ce registre contient deux bits de drapeau ("flags") INTF0 et INTF1 qui indiquent quelles interruptions externes ont été déclenchées, c-à-d ces bits sont automatiquement activés juste après le déclenchement des interruptions correspondante. INTF0 et INTF1 doivent être remis à zéro ("cleared") par le programmeur pour indiquer que chaque interruption a été traitée.

Exemple 1

En utilisant l'interruption INT1, écrire un programme C (CodeVisionAVR) qui permet d'activer et d'éteindre la LED à chaque front montant détecté sur PD3.



Solution

```
#include <io.h>

// Déclaration de la fonction de l'interruption INT1 (ISR)
interrupt [3] void external_int1(void)
{
  PORTC.1 = !PORTC.1; // Inverse l'état de PC1
}

void main(void)
{
  DDRD.3 = 0; // Définit PD3 comme une entrée (instruction facultative)
  PORTD.3 = 1; // Active la résistance de pull-up pour PD3
  DDRC.1 = 1; // Définit PC1 comme une sortie

  // Activation globale des interruptions
  #asm("sei");

  // Configuration de l'interruption INT1
  EICRA = (1 << 1) | (0 << 2); // Déclenche l'interruption sur un front
                               //montant de INT1
  EIMSK = (1 << 2); // Active l'interruption INT1

  while (1)
  {
    // Boucle principale du programme
  }
}
```

Il faut noter que le bit I du registre SREG est mis à 1 après la mise sous tension du microcontrôleur ou après un reset.

4.3.2 Les interruptions externes PCINT0:2

Contrairement aux interruptions INT0:1 qui ne sont associées qu'aux deux entrées/sorties numériques PD2 et PD3, chaque interruption PCINT0:2 peut être assignée à une broche ou à un groupe de broches appartenant au même port. PCINT0 peut être assignée aux E/S du port B (les entrées d'interruption PCINT0 à PCINT7 sur la figure 4.2), PCINT1 peut être assignée aux entrées/sorties du port C (PCINT8 à PCINT14) et PCINT2 peut être assignée aux E/S du port D (PCINT16 à PCINT23).

Une fois activés les interruptions PCINT0:2 se déclenche pour tous changement d'état (que ce soit un front montant ou un front descendant) sur les E/S numériques correspondantes.

4.3.2.1 Registres

Registre PCICR (Pin Change Interrupt Control Register)

Bit	7	6	5	4	3	2	1	0	
(0x68)	–	–	–	–	–	PCIE2	PCIE1	PCIE0	PCICR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Le bit PCIE0 (Pin Change Interrupt Enable) : lorsque ce bit est à un et si le bit I du registre SREG est à un, tout changement sur l'une des broches PCINT7 à PCINT0 déclenchera une interruption.

Le bit PCIE1 (Pin Change Interrupt Enable) : lorsque ce bit est à un et si le bit I du registre SREG est à un, tout changement sur l'une des broches PCINT14 à PCINT8 déclenchera une interruption.

Le bit PCIE2 (Pin Change Interrupt Enable) : lorsque ce bit est à un et si le bit I du registre SREG est à un, tout changement sur l'une des broches PCINT23 à PCINT16 déclenchera une interruption.

Le registre PCMSK0 (Pin Change Mask Register 0)

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
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	

Lorsque l'un des bits PCINT0 à PCINT7 est mis à un et le bit PCIE0 du registre PCICR est positionné à un, tout changement sur la broche d'E/S correspondante provoquera une interruption. Si un des bits PCINT7 à PCINT0 est mis à zéro, aucune interruption sur changement d'état n'aura lieu.

Le registre PCMSK1(*Pin Change Mask Register* 1)

Bit	7	6	5	4	3	2	1	0	
(0x6C)	–	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8	PCMSK1
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Lorsque l'un des bits PCINT8 à PCINT14 est mis à un et le bit PCIE1 du registre PCICR est positionné à un, tout changement sur la broche d'E/S correspondante provoquera une interruption. Si un des bits PCINT14 à PCINT8 est mis à zéro, aucune interruption sur changement d'état n'aura lieu.

Le registre PCMSK2 (*Pin Change Mask Register* 2)

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
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	

Lorsque l'un des bits PCINT16 à PCINT23 est mis à un et le bit PCIE2 du registre PCICR est positionné à un, tout changement sur la broche d'E/S correspondante provoquera une interruption. Si un des bits PCINT23 à PCINT16 est mis à zéro, aucune interruption sur changement d'état n'aura lieu.

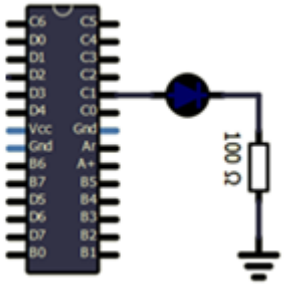
Le registre PCIFR (Pin Change Interrupt Flag Register)

Bit	7	6	5	4	3	2	1	0	
0x1B (0x3B)	–	–	–	–	–	PCIF2	PCIF1	PCIF0	PCIFR
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Lorsque l'une des trois interruptions est activée, le bit PCIFn ($0 < n < 2$) est positionné à un. Celui-ci peut être effacé à la fin de l'exécution de l'interruption, ou par logiciel en écrivant un niveau logique un.

Exemple 2

Ecrire un programme C (CodeVisionAVR) qui configure le port B en entrée avec résistances de tirage et en utilisant l'interruption PCINT0 active et éteint la LED à chaque front montant et front descendant reçu sur le port B.



Solution

```
#include <io.h>
// Déclaration de la fonction de l'interruption PCINT0
interrupt [4] void external_pcint0(void)
{
  PORTC.1 = !PORTC.1; // Inverse l'état de PC1
}

void main(void)
{
 DDRC.1 = 1; // Définit PC1 comme une sortie
 DDRB=0xFF;  // Configure le port B en sortie
 PORTB=0xFF;  // Active les résistances de tirage
 // Activation de PCINT0
 PCICR=(1<<0); // Activation locale de PCINT0
 PCMSK0=0xFF;  // active l'interruption sur les broches PB0 à PB7
while (1)
    {
  // Boucle principale du programme
     }
}
```