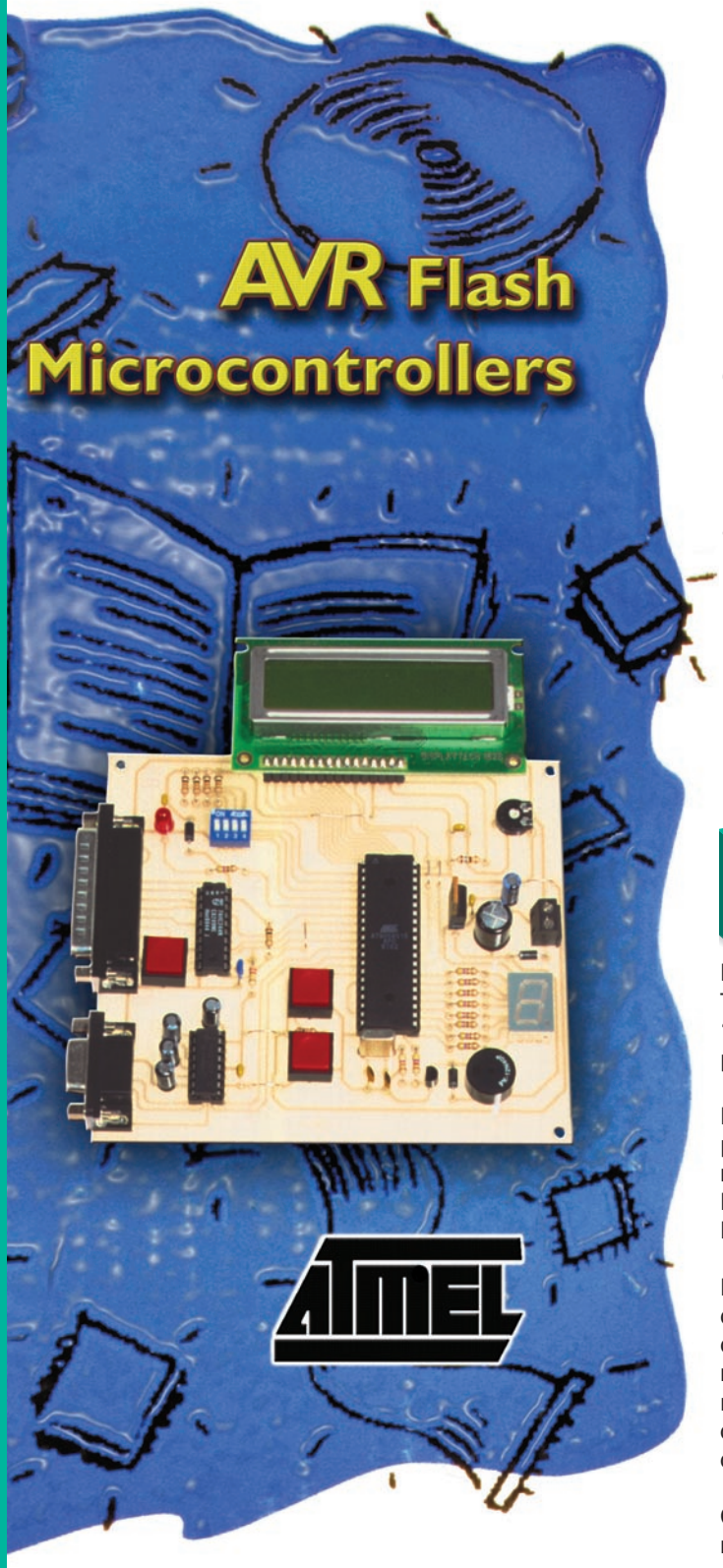
# Les microcontrôleurs Flash ATMEL AVR

## Leçon 1




**Le but de ce cours est de vous présenter les microcontrôleurs Flash de la famille ATMEL AVR. En utilisant une carte de test simple et complète pour la programmation "in-system", vous apprendrez à utiliser des périphériques comme les afficheurs à 7 segments, les boutons poussoirs, les lignes sérielles, les buzzers et les afficheurs LCD.**

**Les listings de démonstration que nous allons illustrer au fur et à mesure des leçons seront d'abord rédigés en langage Assembleur puis en Basic, plus simple et plus intuitif.**

Depuis le numéro 1 de la revue, nous vous avons proposé chaque mois un cours sur les microcontrôleurs. D'abord avec le 16F84, puis avec le 16F876, tous deux fabriqués par Microchip.

Nous allons maintenant vous parler des ATMEL AVR. Tous les microcontrôleurs cités appartiennent à la même "classe" même s'ils sont profondément différents entre eux pour leurs prestations.

La progression utilisée pour présenter les différents dispositifs n'est absolument pas due au hasard : ceux qui nous suivent depuis le début se seront rendu compte que la séquence des cours publiés est liée au mûrissement du lecteur dans ce domaine.

Nous pouvons dire que nous avons utilisé (même si c'est de façon impropre) les premières publications pour essayer de faire comprendre la structure de base qu'ont en commun tous les microcontrôleurs. Ensuite, nous avons repris rapidement quelques concepts de base pour focaliser, au contraire, notre attention sur les prestations et les ressources.

Cette introduction pour vous expliquer qu'à notre avis le moment est venu d'apprendre à utiliser une des deux

| TYPE | BROCHES | I/O | FLASH | VCC MIN | EEPROM | SRAM | SPI | CLOCK |
|---|---|---|---|---|---|---|---|---|
| ATTINY10 | 8 | 6 | 1 KB | 2,7 V | NO | NO | NO | 0-6 MHZ |
| ATTINY11 | 8 | 6 | 1 KB | 4 V | NO | NO | NO | 0-6 MHZ |
| ATTINY12 | 8 | 6 | 1 KB | 4 V | 64 B | NO | NO | 0-8 MHZ |
| ATTINY22 | 8 | 5 | 2 KB | 4 V | 128 B | 128 B | NO | 0-8 MHZ |
| ATTINY28 | 28 | 11 | 2 KB | 2,7 V | NO | NO | NO | 0-4 MHZ |
| AT90S1200 | 20 | 15 | 1 KB | 2,7 V | 64 B | NO | NO | 0-12 MHZ |
| AT90S2313 | 20 | 15 | 2 KB | 2,7 V | 128 B | 128 B | NO | 0-10 MHZ |
| AT90S2323 | 8 | 3 | 2 KB | 4 V | 128 B | 128 B | NO | 0-10 MHZ |
| AT90S2333 | 28 | 20 | 2 KB | 4 V | 128 B | 128 B | 1 | 0-8 MHZ |
| AT90S2343 | 8 | 5 | 2 KB | 4 V | 128 B | 128 B | NO | 0-10 MHZ |
| AT90S4414 | 40 | 32 | 4 KB | 2,7 V | 256 B | 256 B | 1 | 0-8 MHZ |
| AT90S4433 | 28 | 20 | 4 KB | 4 V | 256 B | 128 B | 1 | 0-8 MHZ |
| AT90S4434 | 40 | 32 | 4 KB | 4 V | 256 B | 256 B | 1 | 0-8 MHZ |
| AT90S8515 | 40 | 32 | 8 KB | 2,7 V | 512 B | 512 B | 1 | 0-8 MHZ |
| AT90C8534 | 48 | 15 | 8 KB | 3,3 V | 512 B | 256 B | NO | 0-15 MHZ |
| AT90S8535 | 40 | 32 | 8 KB | 4 V | 512 B | 512 B | 1 | 0-8 MHZ |
| ATMEGA161 | 40 | 35 | 16 KB | 4 V | 512 B | 1 KB | 1 | 0-8 MHZ |
| ATMEGA603 | 64 | 48 | 64 KB | 4 V | 2 KB | 4 KB | 1 | 0-6 MHZ |
| ATMEGA103 | 64 | 48 | 128 KB | 4 V | 4 KB | 4 KB | 1 | 0-6 MHZ |

**Tableau 1 : Mémoire et prestations.**

familles de microcontrôleurs les plus diffusées dans le milieu industriel : il s'agit des INTEL 8051 (ou compatibles) et des AVR. Ces derniers sont l'objet de ce cours.

Pour ce faire nous allons, comme d'habitude, associer à cette publication sur papier une carte hardware, que nous appellerons ensuite "carte de test", avec laquelle il vous sera possible de tester et de vérifier les différents programmes de démonstration.

Nous donnerons également au lecteur la possibilité de choisir entre deux approches d'étude différentes : la première, plus rapide, est basée sur la seule carte test qui fait également office de programmateur, la deuxième, plus complète, a pour protagoniste le système de développement original ATMEL (le tout nouveau STK 500) auquel nous associerons la carte de test.

## L'architecture RISC

La technologie RISC consiste à déplacer les complexités majeures du hardware vers le software, ce qui est le contraire exact de la technologie CISC (Complex Instruction Set Computer).

Dans l'architecture CISC les concepteurs ont misé sur la réduction du nombre d'instructions nécessaires pour exécuter le programme, en concevant des instructions très puissantes, ce qui a l'inconvénient de devoir augmenter moyennement le nombre de cycles machine nécessaires pour compléter une instruction. Dans ce cas, la fréquence de travail du système est réduite car il faut introduire une phase d'interprétation du code machine à travers des microcodes.

Par contre, dans l'architecture RISC, on mise beaucoup sur la minimisation du nombre des cycles machine et l'on rend la majeure partie des instructions exécutables en un seul cycle d'horloge, ce qui permet d'augmenter la fréquence de travail du système. Ceci est possible en éliminant la phase d'interprétation grâce à la simplicité des instructions qui peuvent être décodées et exécutées directement par une simple unité de contrôle câblée.

La simplification des unités de contrôle des machines de type RISC est particulièrement avantageuse pour la réalisation de la CPU sur un seul chip VLSI. L'économie d'espace obtenue permet, à zone de silice égale, d'augmenter sensiblement le nombre de registres internes et/ou d'intégrer directement sur le chip la mémoire cache pour exploiter au maximum la vitesse du microprocesseur.

Malheureusement, la technologie RISC a aussi des défauts. En effet, le nombre réduit d'instructions fait que le software résultant, à fonctions à accomplir égales, occupe plus de mémoire que celui d'une machine CISC, aussi bien statiquement que dynamiquement.

Toutes les machines RISC utilisent la technique du "pipelining" pour augmenter leurs prestations en terme d'instructions exécutées dans l'unité de temps.

## La famille AVR à 8 bits

Vous trouverez dans les tableaux 1 et 2 la liste des principales ressources internes de la famille des microcontrôleurs AVR.

Ce qui différencie les divers dispositifs est le nombre d'instructions Assembleur disponibles, la quantité de mémoire SRAM présente et surtout le nombre de lignes d'entrées/sorties (E/S ou I/O pour Input/Output), ainsi que la présence ou l'absence de périphériques tels que le UART, le timer, le convertisseur A/D, etc.

Par exemple le AT90S1200 est un dispositif qui possède 89 instructions Assembleur, 1 kilobyte de mémoire programme, 15 lignes de I/O, 64 bytes de EEPROM et 32 registres d'utilisation générale.

Ce qui unie la famille entière est l'architecture avec laquelle ces microcontrôleurs sont réalisés, le jeu d'instruction et les différentes méthodes de placement de la mémoire et des registres.

| TYPE | INT. | EXT INT. | UART | 8-B TMR | 16-B TMR | PWM | A/D CH | B.OUT |
|---|---|---|---|---|---|---|---|---|
| ATTINY10 | 4 | 1 | NO | 1 | NO | NO | NO | NO |
| ATTINY11 | 4 | 1 | NO | 1 | NO | NO | NO | NO |
| ATTINY12 | 5 | 1 | NO | 1 | NO | NO | NO | YES |
| ATTINY22 | 2 | 1 | NO | 1 | NO | NO | NO | NO |
| ATTINY28 | 5 | 2 | NO | 1 | NO | NO | NO | NO |
| AT90S1200 | 3 | 1 | NO | 1 | NO | NO | NO | NO |
| AT90S2313 | 10 | 2 | 1 | 1 | 1 | 1 | NO | NO |
| AT90S2323 | 2 | 1 | NO | 1 | NO | NO | NO | NO |
| AT90S2333 | 14 | 2 | 1 | 1 | 1 | 2 | 6 | YES |
| AT90S2343 | 2 | 1 | NO | 1 | NO | NO | NO | NO |
| AT90S4414 | 11 | 2 | 1 | 1 | 1 | 2 | NO | NO |
| AT90S4433 | 14 | 2 | 1 | 1 | 1 | 2 | 6 | YES |
| AT90S4434 | 15 | 2 | 1 | 2 | 1 | 3 | 8 | NO |
| AT90S8515 | 11 | 2 | 1 | 1 | 1 | 2 | NO | NO |
| AT90C8534 | 7 | 2 | NO | 1 | 1 | NO | 6 | NO |
| AT90S8535 | 15 | 2 | 1 | 2 | 1 | 3 | 8 | NO |
| ATMEGA161 | 20 | 3 | 2 | 2 | 1 | 4 | 8 | YES |
| ATMEGA603 | 16 | 8 | 1 | 2 | 1 | 4 | 8 | NO |
| ATMEGA103 | 16 | 8 | 1 | 2 | 1 | 4 | 8 | NO |

**Tableau 2 : Ressources internes.**

L'architecture se base, en particulier, sur le concept d'accès rapide aux registres. Les registres, comme vous le savez, sont des zones de mémoire utilisées pour communiquer avec les périphériques disponibles à l'intérieur du microcontrôleur comme les compteurs, les timers, les convertisseurs A/D et les ports d'I/O.

Certains registres peuvent être utilisés comme des pointeurs à placement indirect à 16 bits pour communiquer avec de la mémoire : ces registres à 16 bits sont appelés registres X, Y, Z.

Une autre caractéristique commune est le mode par lequel le microcontrôleur exécute les instructions. On l'appelle instruction "pipelining" (chaîne de montage). Le "pipelining" consiste à exécuter une instruction et à aller chercher simultanément l'instruction suivante.

## Le ATMEL AT90S8515

Cette brève introduction vous permet de comprendre qu'en apprenant la structure de n'importe quel microcontrôleur AVR, vous serez automatiquement en mesure de travailler avec la famille entière.

C'est la raison pour laquelle nous avons décidé de baser tout le cours (et, par conséquent, la carte test et les différents listings démo) sur un seul modèle de microcontrôleur.

Notre choix s'est porté logiquement vers le type le plus courant, c'est-à-dire le AT90S8515 dont le schéma synoptique est donné en figure 1.

Ce microcontrôleur, contenu dans un boîtier à 40 pattes, fournit un jeu de 118 instructions Assembleur, 8 kbytes de mémoire programme, 512 bytes de EEPROM, 512 bytes de SRAM et 32 lignes de I/O.

Le dispositif exécute de puissantes instructions en un seul cycle d'horloge, il est capable de traiter 1 MIPS par MHz (ceci en théorie).

Parmi ses autres caractéristiques, signalons la présence de 32 registres pour des opérations de I/O et 32 registres d'utilisation générale, des interruptions internes ou externes, un UART programmable par interconnexions sérielles, un watchdog timer programmable avec oscillateur interne, un port sériel SPI, deux états à basse consommation que vous pouvez sélectionner via software et un timer/counter.

Les deux états de basse consommation sont appelés respectivement "idle mode" et "power down mode".

Le premier arrête le CPU mais permet à la SRAM, au timer/counter, au port sériel SPI et aux systèmes d'interruption de continuer à fonctionner, alors que dans le second mode, le contenu des registres est sauvé et l'oscillateur est "gelé" ; toutes les autres fonctions du chip sont désactivées jusqu'à ce que l'on intervienne avec une interruption externe ou en effectuant un RESET du CPU.

Comme nous venons de le dire, le microcontrôleur AT90S8515 est disponible en version 40 broches (voir figure 2).

Nous allons les décrire une par une.

## La description des broches

**Vcc** : Patte d'alimentation positive (broche 40).

**GND** : Masse d'alimentation (broche 20).

**Port A** (PA7...PA0) : C'est un port de I/O bidirectionnel. Toutes les pattes du port ont des résistances internes de pull-up. Le buffer de sortie est en mesure de fournir jusqu'à 20 mA de courant, suffisant pour piloter un afficheur à Led. Les pattes sont en haute impédances quand une condition de reset devient active, ou bien lorsque l'horloge n'est pas active. Ce port est utilisé comme multiplexer d'entrée/sortie pour les données et les adresses quand une SRAM externe est reliée (broches 32 à 39).

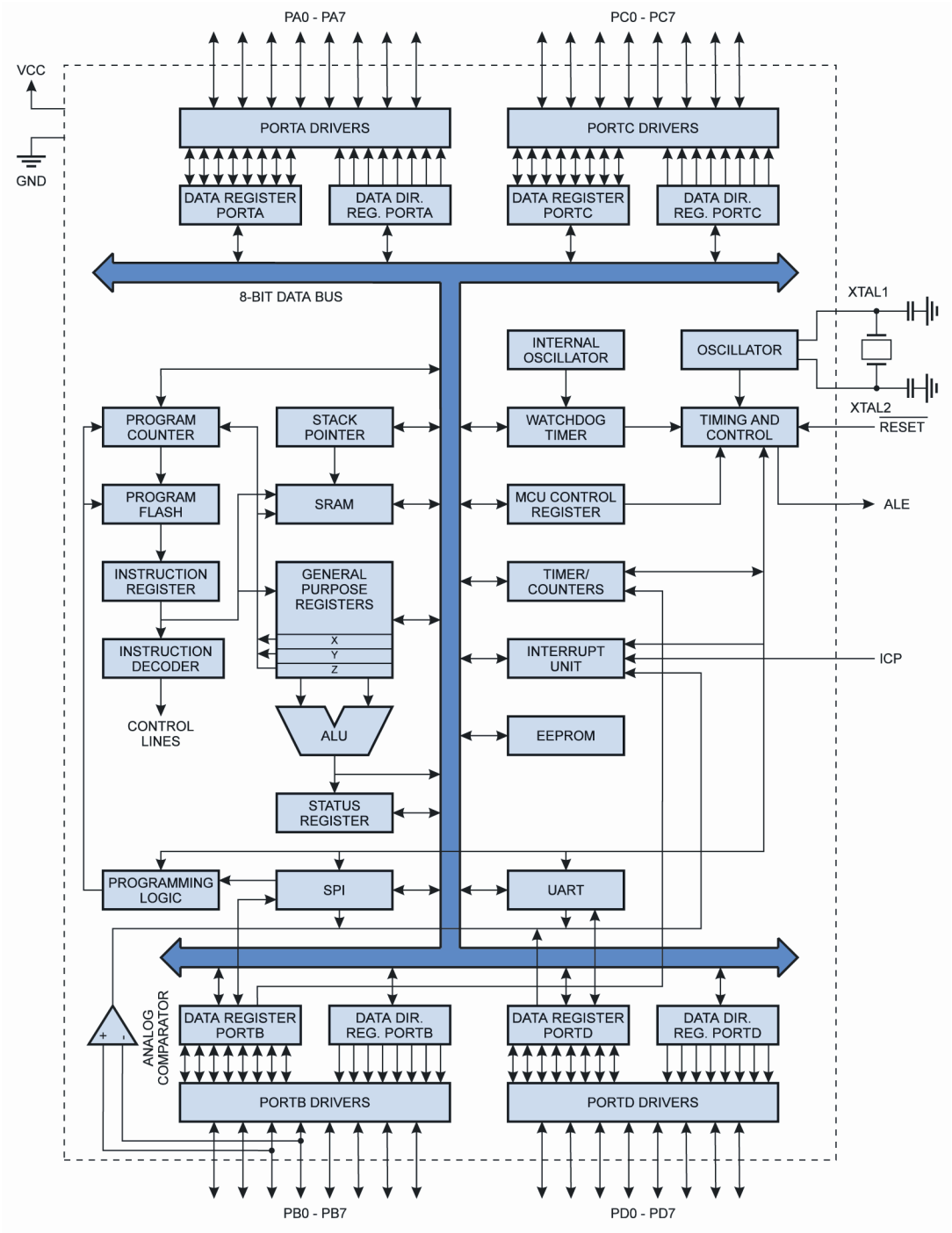



**Figure 1 : Schéma synoptique interne du microcontrôleur ATMEL AVR AT90S8515.**

**Port B** (PB7…PB0) : C'est un port de I/O bidirectionnel. Toutes les pattes du port ont des résistances internes de pull-up. Le buffer de sortie est en mesure de fournir jusqu'à 20 mA de courant. Les pattes du port sont en haute impédance quand une condition de RESET devient active, ou bien quand l'horloge n'est pas active (broches 1 à 8).

**Port C** (PC7…PC0) : C'est un port de I/O bidirectionnel. Toutes les pattes du port ont des résistances internes de pull-up. Le buffer de sortie est en mesure de fournir jusqu'à 20 mA de courant. Les broches sont en haute impédance quand une condition de RESET devient active, ou bien quand l'horloge n'est pas active. Lorsqu'on branche de la SRAM externe, ce port est utilisé comme bus adresses en sortie vers la SRAM (broches 21 à 28).

**Port D** (PD7…PD0) : C'est un port de I/O bidirectionnel. Toutes les pattes du port ont des résistances internes de pull-up. Le buffer de sortie est en mesure de fournir jusqu'à 20 mA de courant. Les broches sont en haute impédance quand une condition de RESET devient active, ou bien quand l'horloge n'est pas active (broches 10 à 17).

**RESET** (actif bas) : La broche de RESET est une entrée. Elle est activée par un niveau logique bas qui doit avoir une durée opportune. Habituel-

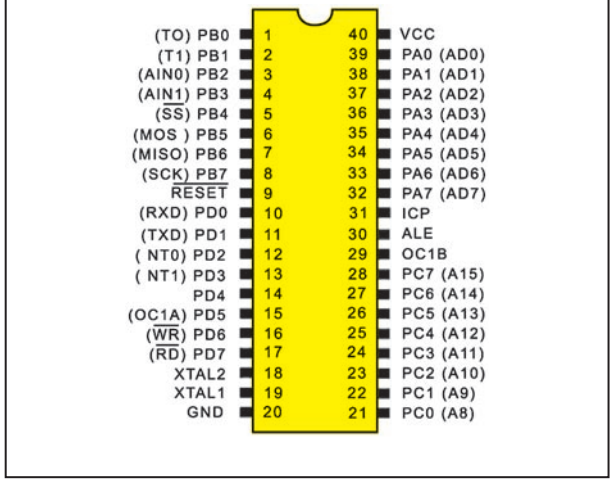



**Figure 2 : Brochage du AT90S8515.**

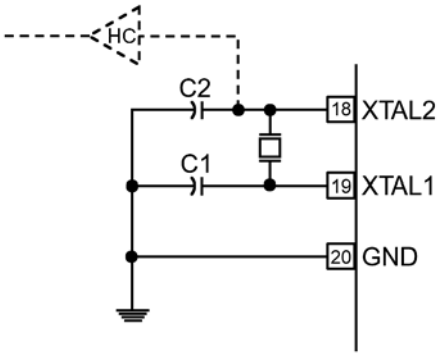



**Figure 3 : Raccordement du quartz et des condensateurs externes aux broches XTAL2 et XTAL1 du microcontrôleur.**

lement, le temps de RESET tourne autour de 50 ns. Des temps plus courts n'assurent pas la génération du RESET (broche 9).

**XTAL2 et XTAL1** : Ce sont les broches auxquelles est connecté le quartz de 4 à 8 MHz. En plus du quartz, deux condensateurs externes sont requis comme le montre le schéma de la figure 3 (broches 18 et 19).

**ICP** : C'est une broche d'entrée pour la fonction de "timer/counter input capture". Nous décrirons cette broche en détail dans la leçon consacrée au timer (broche 31).

**OC1B** : C'est une broche de sortie pour la fonction de "timer/counter1 compare B". Nous décrirons cette broche en détail dans la leçon consacrée au timer 1 (broche 29).

**ALE** : C'est l'abréviation de "Address Latch Enable" qui est utilisé lorsque de la mémoire externe est connectée au microcontrôleur. En fait, la broche génère une impulsion de référence qui est utilisée pour commencer une liaison entre un microcontrôleur et la mémoire (broche 30).

## La programmation "in-system"

Vous pouvez noter que le microcontrôleur dispose d'une grande quantité de mémoire programme. Dans notre cas, nous disposons de 8 kbytes de mémoire Flash.

Ce type de mémoire peut être programmé "in-system", c'est-à-dire que vous laissez le microcontrôleur sur le circuit sur lequel il doit travailler et qu'avec une connexion opportune au PC, vous le programmez selon vos propres exigences.

La programmation "in-system" évite l'inconvénient de devoir continuellement extraire le microcontrôleur de son support pour l'insérer dans le programmateur.
On évite également, de cette façon, de l'endommager, en tordant par exemple une patte ou en cassant carrément une pendant les manœuvres continues d'insertion et d'extraction du composant. Le seul inconvénient est la nécessité de devoir réaliser une liaison entre le circuit en conception et le PC.

Notre carte test a été prévue spécialement pour supporter la programmation "in-system".

Rendez-vous à notre prochaine leçon.

Nous commencerons à examiner la structure et le fonctionnement de la mémoire interne et des principaux registres.

◆ ***M. D.***

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 2

**Dans la première leçon de ce cours, nous avons présenté, dans leurs grandes lignes, les prestations et les ressources des dispositifs qui composent la famille des ATMEL AVR 8 bits, et nous avons décidé de fixer notre attention sur l'AT90S8515.**

Ce microcontrôleur dispose d'une grande capacité de mémoire programme : 8 kilobytes de mémoire Flash, en plus de laquelle nous retrouvons les 32 registres pour utilisation générale qui vont de l'adresse mémoire $0000 à $001F, les 64 registres d'I/O (on utilise indifféremment I/O = Input/Output ou E/S = Entrées/Sorties) qui vont de l'adresse $0020 à $005F, la mémoire SRAM interne (512 bytes) qui va de l'adresse $0060 à l'adresse $025F et, enfin, nous avons de l'espace pour placer une mémoire SRAM externe pour un total de 64 kilobytes (adresses de $0260 à $FFFF).

Observons le schéma synoptique interne de l'AT90S8515 (figure 1).

Au centre du schéma se trouve l'unité logique arithmétique l'ALU (Aritmetic Logic Unit) qui forme avec le bloc des registres le cœur du microcontrôleur.

Le cœur communique, au moyen d'un bus de données (Data Bus) de 8 bits, avec toutes les ressources implémentées : tout d'abord avec les registres de contrôle qui, vus apparemment comme une zone de mémoire particulière, forment en réalité des interfaces entre le programme et les périphériques implémentés.

Puis nous trouvons l'unité d'interruptions (Interrupt Unit) c'est-à-dire un dispositif qui s'occupe de gérer et de trier les différentes interruptions que les périphériques peuvent envoyer au CPU.

## Le concept d'interruption

Nous verrons plus tard que le CPU ne fait que lire et exécuter séquentiellement, rigoureusement l'une après l'autre, les instructions contenues dans la mémoire programme.

En pratique, le CPU lit le code opérateur "l'opcode" de la première instruction en mémoire, l'interprète en le transformant en une commande et l'exécute. Puis il répète le même processus sur l'opcode disponible dans le byte suivant de la mémoire programme et ainsi de suite.

En réalité, cette séquentialité est confiée au "Program Counter" qui, par définition, pointe l'adresse du byte de mémoire programme qui contient l'opcode de la prochaine instruction que le CPU doit exécuter.

Une machine à états ainsi élaborée ne permet cependant pas de gérer des événements en temps réel. C'est la raison pour laquelle on a inventé les interruptions.

Les périphériques internes peuvent, sur la base d'événements internes ou externes particuliers, générer une interruption du cycle normal du programme.

Dans la pratique, cela consiste à forcer le "Program Counter" à pointer l'adresse d'une zone de mémoire définie (vecteur d'interruption).

Donc, en activant, par exemple, l'interruption du périphérique UART, nous obtiendrons, en correspondance avec la fin de la réception des données, que le microcontrôleur aille exécuter l'instruction contenue à l'adresse $009 (vecteur d'interruption de la réception UART).

Nous pouvons alors insérer une série d'instructions consacrées à cet événement. Par exemple, nous pouvons lire la valeur que l'UART a reçue et l'écrire dans une variable.

Ces opérations sont effectuées par une sous-routine appelée "routine de réponse à l'interruption". La routine terminera avec une instruction qui fera l'opération opposée par rapport à l'interruption : c'est-à-dire qui forcera le "Program Counter" à pointer l'adresse contenant l'instruction suivant

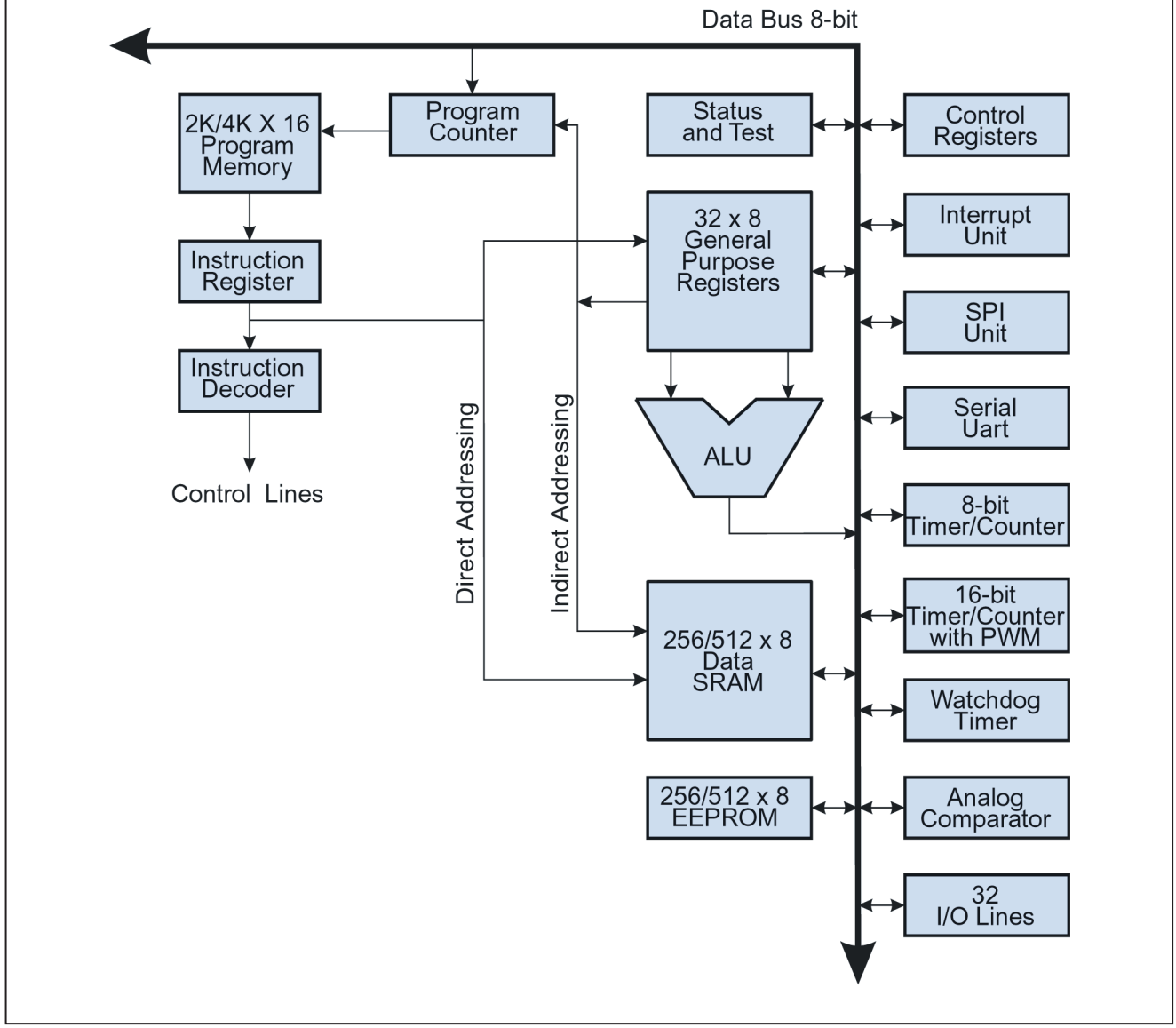



**Figure 1 : Schéma synoptique du microcontrôleur AT90S8515.**

la dernière exécutée avant l'interruption. L'AT90S8515 dispose de 13 vecteurs d'interruption.

Revenons maintenant au schéma synoptique interne.

Nous voyons que les autres périphériques disponibles sont la "SPI Unit", interface sérielle synchrone à 3 fils en mesure de travailler en mode "Master" (maître) ou "Slave" (esclave), le "Serial UART" (Serial Universal Asynchronous Receiver and Transmitter), un "Timer/Counter" 8 bits et un "Timer/Counter" 16 bits, un "Watchdog Timer", un comparateur analogique et 32 lignes d'entrée/sortie avec lesquelles le dispositif échange les données en niveau TTL avec le monde extérieur.

Pour communiquer avec les différents périphériques énumérés ci-dessus, une série de registres, définis selon le tableau de la figure 3, appelés "I/O Register" sont disponibles.

Dans ce tableau, vous trouverez la liste des adresses mémoire où ils sont disponibles et les sigles mnémotechniques qui les identifient.

## Le plan mémoire

Comme tous les microcontrôleurs, l'AT90S8515 dispose aussi de deux zones de mémoire spécifique internes : la "Program Memory" et la "Data Memory".

La "Program Memory" ou mémoire programme, contient le programme c'est-à-dire l'opcode des instructions que le CPU devra exécuter l'une après l'autre quand le microcontrôleur sera alimenté. La mémoire programme est de type Flash et peut être écrite et effacée plus de 1 000 fois. Sa capacité est de 4 kilobytes x 16 et ses adresses vont de $000 à $FFF.

La "Data Memory" peut être décomposée en deux parties significatives : l'une contient les données et l'autre est destinée aux registres.

Nous verrons que le déroulement d'un programme requiert l'utilisation des constantes mais aussi des variables.

Le terme "variable" sert à indiquer tous les paramètres numériques qui peuvent varier pendant l'exécution d'un programme.

Notre microcontrôleur dispose de 512 octets internes de mémoire (de

l'adresse $0060 à $025F) dans lesquelles il est possible de mémoriser des variables.

Observons le plan mémoire de données : nous pouvons noter que les adresses $0260 à $FFFF sont prévues pour une RAM externe.

Ce qui veut dire que la structure hardware et les ressources software du AT90S8515 permettent de brancher une mémoire externe SRAM de 64 kilobytes (maximum).

Les adresses $0000 à $005F contiennent, quant à elles, les registres qui, comme nous l'avons dit précédemment, sont des positions utilisées pour communiquer avec les périphériques ou utiles au travail de l'ALU.

Les registres dont les adresses vont de $0000 à $0001F sont appelés "General Purpose Working Register".

Ce sont les registres de travail, ceux qui sont utilisés pour faire des opérations mathématiques ou pour diriger le programme vers des adresses déterminées.

Nous étudierons ces registres lors de l'analyse du jeu d'instructions.

Nous vous rappelons, de toute façon, que sur les 32 registres d'utilisation générale, 6 peuvent être utilisés comme pointeurs d'adressage indirect à 16 bits pour travailler avec la mémoire.

Ces registres de 16 bits sont appelés registres X, Y et Z.

A chaque registre est associée une adresse unique, ce qui a permis de faire le plan mémoire de l'adresse $00 à l'adresse $1F en distinguant les 32 adresses à utiliser.

L'ALU à hautes prestations de l'AVR communique avec les 32 registres d'utilisation générale, et est en mesure d'exécuter, en un seul cycle d'horloge, des opérations entre les deux registres.

Les adresses de $0020 à $005F contiennent, par contre, les registres d'I/O (voir le tableau de la figure 3).

Il s'agit de 64 adresses à travers lesquelles il est possible de donner des ordres, ou encore d'envoyer des commandes aux différents périphériques et de les recevoir.

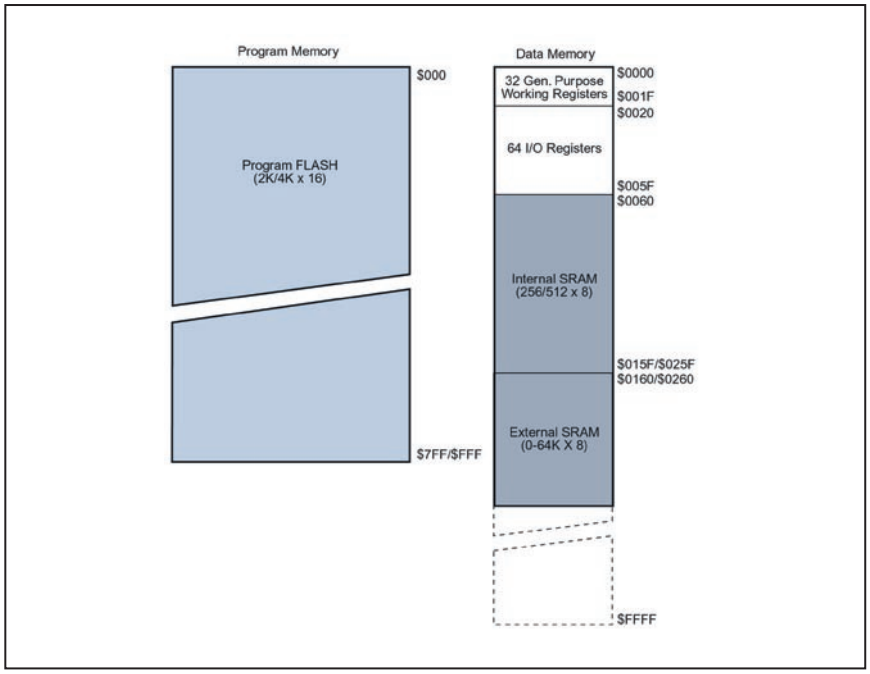



**Figure 2a : L'illustration montre que les adresses qui vont de $0000 à $001F servent aux registres d'utilisation générale, alors que les adresses $0020 à $0005F contiennent les registres d'I/O.**

**De la position $0060 jusqu'à la $025F nous trouvons la SRAM interne du microcontrôleur, alors que de l'adresse $0260 jusqu'à $FFFF nous avons l'espace utilisable pour ajouter au microcontrôleur de la mémoire SRAM externe.**

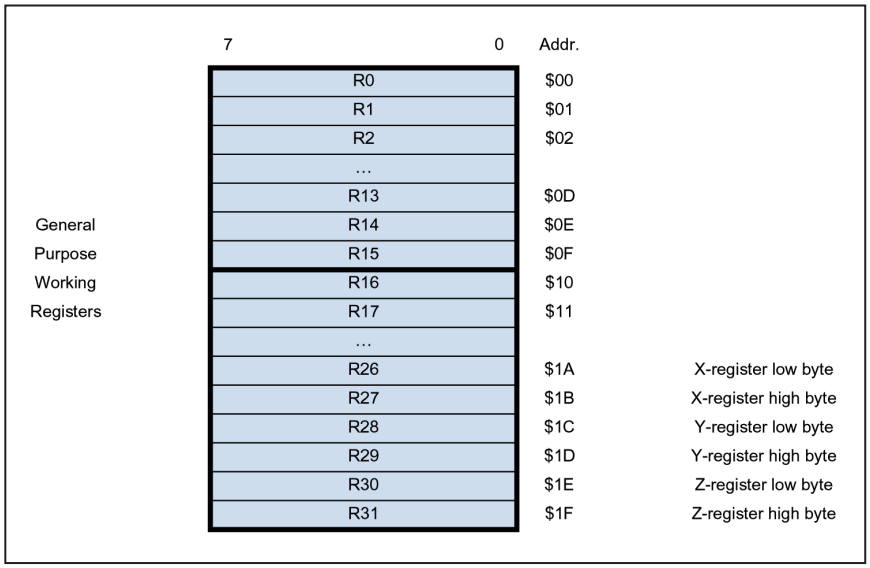



**Figure 2b : A chaque registre est associée une adresse unique. On a ainsi fait le plan mémoire de l'adresse $00 à l'adresse $1F en distinguant 32 positions à utiliser comme espace de données. Les registres qui vont de l'adresse $1A à l'adresse $1F peuvent être associés deux par deux pour obtenir des registres de 16 bits, nommés registres X, Y et Z.**

Nous décrirons mieux chaque registre d'I/O lors de l'analyse du périphérique auquel ils sont destinés.

Limitons-nous, pour l'instant, à dire que les adresse d'I/O, ou encore la partie de mémoire qui contient les registres d'I/O, sont accessibles à travers les instructions de "IN" et de "OUT" qui transfèrent les données entre les 32 registres d'utilisation générale et l'espace d'I/O.

## La mémoire EEPROM

La SRAM interne n'est pas la seule zone possible pour la mémorisation

| Address Hex | Name | Function |
|---|---|---|
| $5F | SREG | Status Register |
| $5E | SPH | Stack Pointer High |
| $5D | SPL | Stack Pointer Low |
| $5B | GIMSK | General Interrupt Mask register |
| $5A | GIFR | General Interrupt Flag Register |
| $59 | TIMSK | Timer/Counter Interrupt Mask register |
| $58 | TIFR | Timer/Counter Interrupt Flag register |
| $55 | MCUCR | MCU general Control Register |
| $53 | TCCR0 | Timer/Counter0 Control Register |
| $52 | TCNT0 | Timer/Counter0 (8-bit) |
| $4F | TCCR1A | Timer/Counter1 Control Register A |
| $4E | TCCR1B | Timer/Counter1 Control Register B |
| $4D | TCNT1H | Timer/Counter1 High Byte |
| $4C | TCNT1L | Timer/Counter1 Low Byte |
| $4B | OCR1AH | Timer/Counter1 Output Compare Register A High Byte |
| $4A | OCR1AL | Timer/Counter1 Output Compare Register A Low Byte |
| $49 | OCR1BH | Timer/Counter1 Output Compare Register B High Byte |
| $48 | OCR1BL | Timer/Counter1 Output Compare Register B Low Byte |
| $45 | ICR1H | T/C 1 Input Capture Register High Byte |
| $44 | ICR1L | T/C 1 Input Capture Register Low Byte |
| $41 | WDTCR | Watchdog Timer Control Register |
| $3E | EEARH | EEPROM Address Register High Byte (AT90S8515) |
| $3E | EEARL | EEPROM Address Register Low Byte |
| $3D | EEDR | EEPROM Data Register |
| $3C | EECR | EEPROM Control Register |
| $3B | PORTA | Data Register, Port A |
| $3A | DDRA | Data Direction Register, Port A |
| $39 | PINA | Input Pins, Port A |
| $38 | PORTB | Data Register, Port B |
| $37 | DDRB | Data Direction Register, Port B |
| $36 | PINB | Input Pins, Port B |
| $35 | PORTC | Data Register, Port C |
| $34 | DDRC | Data Direction Register, Port C |
| $33 | PINC | Input Pins, Port C |
| $32 | PORTD | Data Register, Port D |
| $31 | DDRD | Data Direction Register, Port D |
| $30 | PIND | Input Pins, Port D |
| $2F | SPDR | SPI I/O Data Register |
| $2E | SPSR | SPI Status Register |
| $2D | SPCR | SPI Control Register |
| $2C | UDR | UART I/O Data Register |
| $2B | USR | UART Status Register |
| $2A | UCR | UART Control Register |
| $29 | UBRR | UART Baud Rate Register |
| $28 | ACSR | Analog Comparator Control and Status Register |

Note: Reserved and unused locations are not shown in the table

**Figure 3 : Pour communiquer avec les différents périphériques, une série de registres, appelés “I/O Register”, sont disponibles. Dans ce tableau son listées les adresses mémoire où ils sont disponibles et les sigles mnémotechniques qui les identifient.**

des données. En effet 512 autres octets, dans lesquels vous pouvez écrire ou lire des données, sont implémentés.

Il s'agit de la mémoire EEPROM. Cette zone de mémoire peut être considérée comme une RAM à la différence près que les données insérées sont conservées, même en l'absence d'alimentation.

Pour écrire ou lire des données en EEPROM, il est cependant nécessaire d'utiliser trois registres spécifiques.

Cette mémoire permet un maximum de 100 000 cycles d'écriture/lecture.

## Le registre d'état

Ce registre sert à contrôler si des événements particuliers, dus à l'exécution de certaines instructions, comme les instructions logiques ou mathématiques, se vérifient. Chaque bit de ce registre a une fonction particulière.

**Bit 7 – I – Global Interrupt Enable**
Ce bit est mis à la valeur logique haute (c'est-à-dire 1) pour activer l'utilisation des interruptions. Ce bit est mis à la valeur logique basse (c'est-à-dire 0) après une demande d'interruption. Il est remis à 1 par l'instruction RETI au retour d'une routine d'interruption.

**Bit 6 – T – Bit Copy Storage**
Les instructions de copie des bits (BLD, bit lu et BST, bit emmagasiné) utilisent le bit T comme source et destination dans les opérations effectuées sur chaque bit d'un registre. Un bit d'un registre peut être copié dans le bit T par l'instruction BST alors que le bit T peut être copié dans un autre registre à travers l'instruction BLD.

**Bit 5 – H – Half Carry Flag**
Ce bit indique qu'une opération arithmétique a généré une retenue ou un dépassement.

**Bit 4 – S – Sign Bit**
Le bit S est donné par un OR exclusif entre le flag négatif N et celui complémenté à 2 du flag V. Il indique le signe de la donnée après avoir exécuté une opération arithmétique.

**Bit 3 – V – Overflow flag**
Ce bit contient le résultat d'overflow et est en complément à deux. Traduit littéralement, overflow veut dire débordement. C'est une condition dans laquelle une opération arithmétique fournit un résultat de grandeur supérieure au maximum qu'un registre ou une position de mémoire peut contenir.

**Bit 2 – N – Negative flag**

**Bit 1 – Z – Zero flag**

**Bit 0 – C – Carry flag**

Ces trois bits indiquent, respectivement, au terme d'une opération mathématique ou logique, si le résultat est négatif, si le résultat est égal à 0, si l'opération a donné lieu, en plus du résultat, à une retenue. Le registre d'état (Status Register) n'est pas automatiquement sauvegardé lorsque l'on appelle une routine d'interruption.

◆ *M. D.*

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 3

**Le microcontrôleur AT90S8515 dispose de quatre ports d'I/O (Input/Output) de 8 bits appelés port A, port B, port C et port D. Ces ports d'entrées/sorties permettent au microcontrôleur de communiquer avec le monde extérieur. Par exemple, si vous voulez connecter un convertisseur A/D (Analogique/Digital), il faudra que le micro dispose de quelques broches afin que vous puissiez effectuer le branchement au dispositif à commander.**

Chacun de ces ports d'I/O est constitué de 8 bits chacun et certains sont "programmables". Ce terme signifie que des broches particulières du port peuvent être utilisées dans des buts spécifiques.

Le port B, par exemple, dispose d'une interface sérielle (SPI) à quatre fils.

Si le microcontrôleur est programmé de façon correcte, la logique interne s'occupera d'utiliser les broches en question du port d'I/O pour exécuter une interconnexion sérielle à quatre fils avec un autre périphérique qui utilise le même protocole de communication.

Les quatre bits restants du port d'I/O peuvent être utilisés dans n'importe quel autre but.

Chaque port d'I/O est piloté à travers l'utilisation de trois registres qui sont appelés "Data Register", "Data Direction Register" et "Input Pins Address".

Pour utiliser le port A, comme pour les autres, trois adresses de mémoire d'I/O sont assignées, une pour chaque registre. On donne un nom mnémotechnique à chacune de ces adresses.

Le registre des données est appelé "PORTA" (adresse hexadécimale 3BH), le registre qui indique la direction de la donnée, In (entrée) ou Out (sortie), est appelé "DDRA" (adresse 3AH), et le registre "Input Pins Address" est appelé "PINA" (adresse 39H).

Ces noms ("PORTA", "DDRA" et "PINA") sont utilisés lorsque l'on programme en assembleur. En effet, si l'on veut écrire dans le registre "DDRA", il suffira d'utiliser les instructions :

```
LDI r16, 0xff
OUT DDRA, r16
```

Elles donnent au registre "r16" la valeur hexadécimale "FF" et le transfèrent dans le registre "DDRA".

De cette façon, le port A a été sélectionné comme port de sortie.

En résumé, le registre "PORTA" sert à envoyer en sortie des données, le registre "PINA" sert à les acquérir alors que le registre "DDRA" sert à indiquer la direction de la donnée, ou mieux, la direction que peut prendre chaque bit du port.

Vous trouverez le schéma électrique d'une broche du port A en figure 1.

En ce qui concerne le port B, le nombre de registres est le même mais leur emplacement à l'intérieur de la mémoire d'I/O change (voir tableau 1).

Les registres se programment en assembleur de la même façon que le port A à la différence près que certains bits peuvent être utilisés dans des buts particuliers.

Le tableau 2 montre bien les fonctions que l'on peut assigner au port B dont nous avons parlé précédemment.

| | PORT | DDR | PIN |
|---|---|---|---|
| **PORTA** | $H3B | $H3A | $H39 |
| **PORTB** | $H38 | $H37 | $H36 |
| **PORTC** | $H35 | $H34 | $H33 |
| **PORTD** | $H32 | $H31 | $H30 |

**Tableau 1 : Adresses hexadécimales des ports d'I/O.**

Notez que, dans ce cas, le schéma de chaque bit du port est différent, ce qui veut dire que le schéma de circuit pour la broche PB0 du port B sera différent du schéma de circuit de la broche PB1 et ainsi de suite en raison, justement, des caractéristiques différentes de chaque broche.

Pour se rendre compte de cela, il suffit de consulter la note technique.

| Port Pin | Alternate Functions |
|---|---|
| PB0 | T0 (Timer/Counter 0 external input) |
| PB1 | T1 (Timer/Counter 1 external input) |
| PB2 | AIN0 (Analog comparator pos input) |
| PB3 | AIN1 (Analog comparator neg input) |
| PB4 | /SS (SPI Slave Select input) |
| PB5 | MOSI (SPI Bus Master Out/Slave In) |
| PB6 | MISO (SPI Bus Master In/Slave Out) |
| PB7 | SCK (SPI Bus Serial Clock) |

**Tableau 2 : Port B.**

Le port C est identique au port A, alors que le port D a, lui aussi, des broches programmables dans des buts particuliers liés à l'utilisation de l'UART, de la mémoire et des interruptions externes (tableau 3).

## Les compteurs

A l'intérieur du microcontrôleur AT90S8515 se trouvent deux compteurs intégrés, l'un de 8 bits et l'autre de 16 bits. En réalité, la logique permet de les configurer soit en compteur soit en temporisateur.

Cela veut dire que, si le composant est configuré comme compteur, il est alors en mesure d'accepter des impulsions externes et d'interrompre le programme principal après un certain nombre d'impulsions établies par le programmeur (en pratique, s'il y a eu une demande d'interruption du compteur).

Ou bien, il peut être utilisé comme temporisateur et est donc en mesure de compter des impulsions d'horloge du système et de donner un signal d'interruption après un nombre prédéterminé d'impulsions d'horloge.

Chaque compteur peut bénéficier d'une logique interne qui sert de "PRESCALER", c'est-à-dire de diviseur de fréquence.

Le circuit se présente comme le montre le schéma synoptique des compteurs 8 et 16 bits de la figure 2. Dans le circuit, on remarque le "prescaler" et deux multiplexeurs nécessaires pour apporter le signal, soit au compteur 8 bits (TCK0), soit au compteur 16 bits (TCK1).

Les deux multiplexeurs sont programmables en utilisant les trois bits CS00, CS01 et CS02. Le tableau 4 montre la correspondance entre les valeurs logiques des trois bits et le type de signal appliqué au compteur. On peut sélectionner l'horloge interne en exécutant ou non le prescaler, ou bien un signal externe présent sur la broche "T0".

Sur le signal "T0", on peut choisir d'être actif sur le front montant ou bien sur le front descendant du signal.

Le tableau 4 est valable pour le compteur 8 bits. Pour le compteur 16 bits, c'est la même chose, seu-

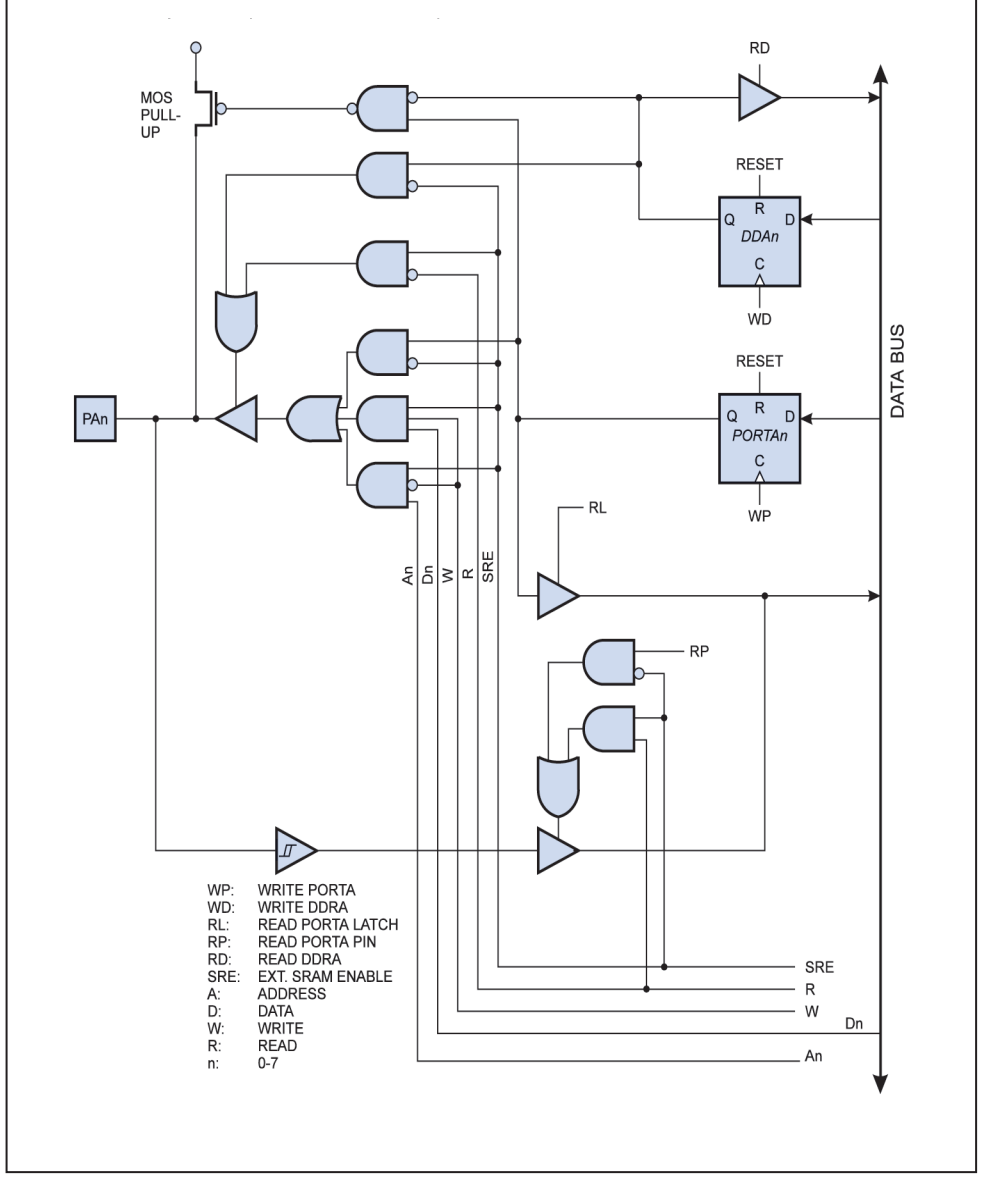



**Figure 1 : Schéma électrique d'une broche du port A.**

les changent les broches de sélection (CS00, CS01 et CS02 deviennent CS10, CS11 et CS12). Ces bits de sélection sont mémorisés par le programmeur à l'intérieur d'un registre 8 bits dont on utilise seulement les trois bits de poids faibles. Ce registre s'appelle "TCCR0".

Vous noterez que le compteur de 8 bits est un "up-counter" et que la valeur de comptage est mémorisée dans le registre "TCNT0".

L'analyse du schéma du circuit du compteur (figure 3) montre les deux registres de 8 bits "TIMSK" et "TIFR". Ces registres servent à gérer les événements d'interruption du compteur.

En ce qui concerne le compteur 8 bits, on utilise le bit 1 du registre "TIMSK". Lorsque celui-ci est au niveau logique haut, avec le bit 1 du "Status Register", l'interruption de débordement (overflow) du compteur a été activée et, par conséquent, s'il y a débordement de la part du compteur, la routine correspondante (qui se trouvera à l'adresse 007H) sera exécutée.

| Port Pin | Alternate Functions |
|---|---|
| PD0 | RXD (UART Input line) |
| PD1 | TXD (UART Output line) |
| PD2 | INT0 (External interrupt 0 input) |
| PD3 | INT1 (External interrupt 1 input) |
| PD5 | OC1A (Timer/Counter1 Out compareA) |
| PD6 | /WR (Write strobe to external memory) |
| PD7 | /RD (Read strobe to external memory) |

**Tableau 3 : Port D.**

| CS02 | CS01 | CS00 | Description |
|---|---|---|---|
| 0 | 0 | 0 | Stop the timer/counter0 |
| 0 | 0 | 1 | CK |
| 0 | 1 | 0 | CK/8 |
| 0 | 1 | 1 | CK/64 |
| 1 | 0 | 0 | CK/256 |
| 1 | 0 | 1 | CK/1024 |
| 1 | 1 | 0 | External Pin T0, falling edge |
| 1 | 1 | 1 | External Pin T0, rising edge |

**Tableau 4 : Configuration du prédiviseur avec le compteur 8 bits.**

Du registre "TIFR", on utilise le bit 1. Celui-ci va au niveau logique haut quand un overflow se produit et est remis à 0 par le hardware dès que la routine correspondant à la demande d'interruption a été exécutée.

Le compteur 16 bits est, par contre, plus complexe que le précédent et il permet par conséquent de réaliser plus de fonctions.

L'une d'entre elles est la modulation PWM (Pulse Width Modulation), et peut être réalisée à 8, 9 ou à 10 bits. Comme le compteur 16 bits est de type "up/down" et peut donc aussi bien compter que décompter, vous pouvez générer un

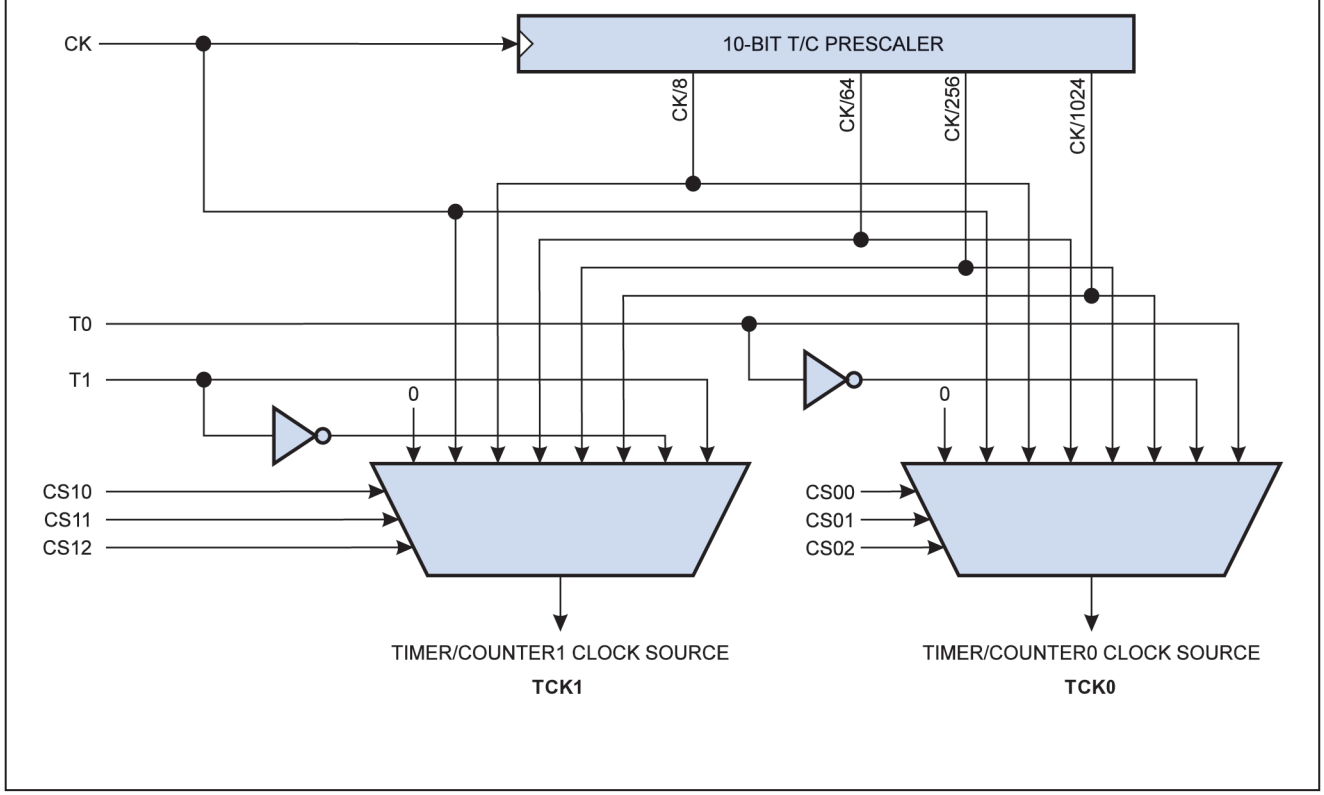



**Figure 2 : Schéma synoptique des compteurs 8 et 16 bits.**

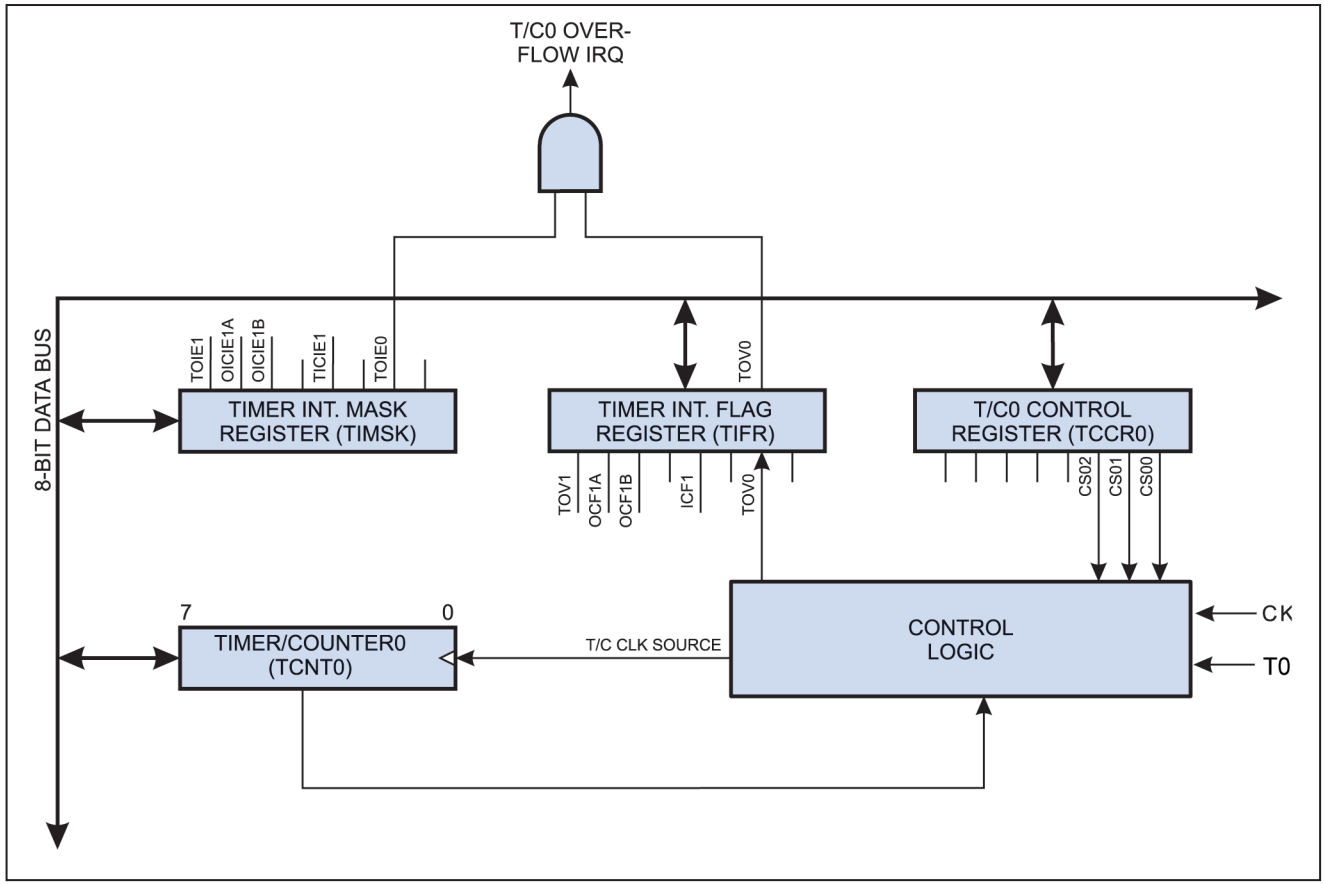



**Figure 3 : Schéma du circuit du compteur 8 bits.**

signal triangulaire (voir la figure 4), obtenu en faisant monter le compteur jusqu'à sa valeur maximale, et toujours à intervalles de temps constants, le faire décompter jusqu'à ce qu'il arrive à zéro pour, ensuite, recommencer l'incrémentation.

Cette caractéristique est utilisée pour générer la modulation PWM, qui consiste à changer le rapport cyclique (Duty Cycle) d'un signal carré sur la base d'un signal modulant représenté par la valeur de comparaison (Compare Value). Pour en comprendre le fonctionnement, revoyez la figure 4. A chaque fois que le signal triangulaire se trouve sous la valeur de référence (la ligne en pointillés) un changement de front est généré sur le signal "OC1X". La forme d'onde triangulaire représente donc la marche dans le temps

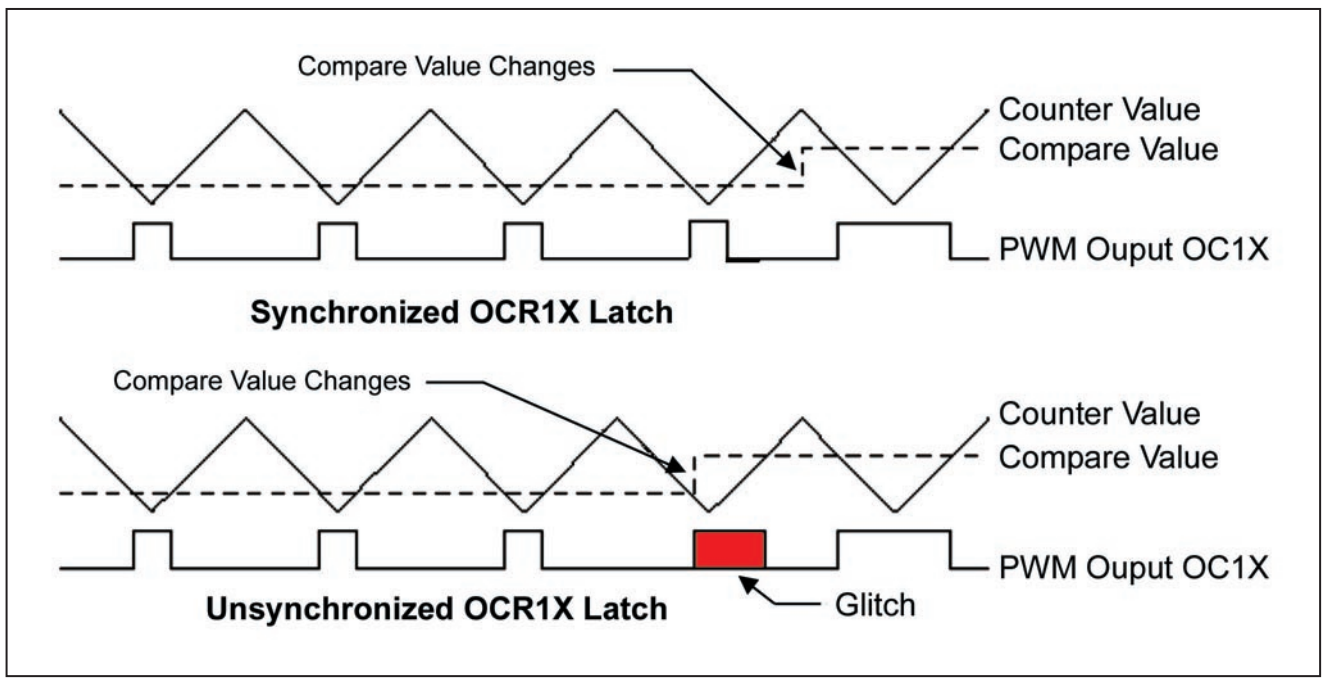



**Figure 4 : Modulation PWM réalisée avec le compteur 16 bits.**

| WDP2 | WDP1 | WDP0 | WDT Oscillator cycle | Typ. Time-out (Vcc=3V) | Typ. Time-out (Vcc=5V) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 16 k | 47 ms | 15 ms |
| 0 | 0 | 1 | 32 k | 94 ms | 30 ms |
| 0 | 1 | 0 | 64 k | 0,19 s | 60 ms |
| 0 | 1 | 1 | 128 k | 0,38 s | 0,12 s |
| 1 | 0 | 0 | 256 k | 0,75 s | 0,24 s |
| 1 | 0 | 1 | 512 k | 1,5 s | 0,49 s |
| 1 | 1 | 0 | 1 024 k | 3,0 s | 0,97 s |
| 1 | 1 | 1 | 2 048 k | 6,0 s | 1,9 s |

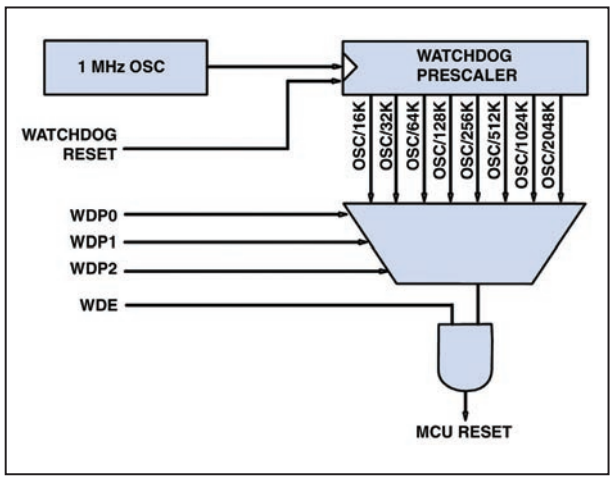



**Figure 5 : Schéma du circuit du watchdog et tableau du prescaler correspondant.**

du compteur. Il est donc clair qu'en changeant la valeur du seuil, vous pourrez changer la durée de l'impulsion en sortie et, donc, en continuant à faire varier le seuil, vous générez une forme d'onde PWM.

Le seuil doit être modifié avant qu'il ne rencontre le signal triangulaire (voir la figure 4) sous peine de générer un "Glitch" qui représente un dérangement non désiré.

## Le watchdog

Le watchdog est un temporisateur particulier qui est utilisé dans les systèmes à microprocesseur comme sécurité pour éviter que le programme n'aille dans une impasse et donc que le système ne se bloque dans une situation non prévue par le programmeur.

En pratique, le watchdog intervient et effectue le reset du microcontrôleur si celui-ci n'est pas effectué par l'instruction "WDR" (WatchDog Reset) dans le temps établi par les broches 0, 1 et 2 du registre "WDTCR".

Le watchdog des microcontrôleurs AVR est temporisé par une horloge interne à 1 MHz, ce qui nous permet de comprendre qu'il peut fonctionner également en l'absence de l'horloge du système car il est indépendant de celui-ci.

Le dispositif est programmé à travers le registre "WDTCR" grâce à l'utilisation des cinq premiers bits. Essayons maintenant d'en comprendre le fonctionnement en les analysant de façon détaillée.

Les bits 0, 1, et 2, comme nous l'avons déjà remarqué, servent à établir le temps qui doit s'écouler avant que le watchdog effectue le reset du micro.

Ce temps dépend également de l'alimentation du micro et peut varier d'environ 15 ms (WDP0 = 0, WDP1 = 0, WDP2 = 0 et alimentation Vcc = 5 V) jusqu'à environ 6 s (WDP0 = 1, WDP1 = 1, WDP2 = 1 et alimentation Vcc = 3 V).

Le bit 4 (WDTOE = Watch Dog Turn Off Enable) et le bit 5 (WDE) servent à désactiver la fonction de watchdog.

Etant donné qu'il s'agit d'un système de sécurité, il serait trop risqué d'utiliser un seul bit de validation/inhibition vu que l'on ne peut pas savoir comment se comporte un programme en cas de dysfonctionnement. Donc, pour éviter des inhibitions involontaires, il est nécessaire de suivre une séquence précise de désactivation du watchdog. Il faut d'abord mettre au 1 logique aussi bien "WDTOE" que "WDE" et, ensuite, pour les quatre cycles d'horloge suivants, effectuer le reset de "WDE". De cette façon le watchdog est désactivé.

Le schéma du Watchdog (figure 5) met en évidence l'oscillateur indépendant de 1 MHz, un "prescaler" et un multiplexeur. Les trois bits de contrôle vont justement agir sur le multiplexeur pour sélectionner les temporisations pour le reset.

◆ ***M. D.***

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 4

**Les microcontrôleurs sont des dispositifs séquentiels. Pour pouvoir effectuer les opérations pour lesquels ils sont programmés, ils ont besoin d'une horloge. Cette horloge est communément appelée "clock".**

Les microcontrôleurs peuvent travailler sur des fronts montants ou des fronts descendants.

Dans le cas de L'ATMEL AT90S8515, il suffit, pour générer le signal d'horloge, de relier un quartz et deux condensateurs aux broches d'entrée XTAL1 et XTAL2 (voir figure 1).

Habituellement, on utilise deux condensateurs de 22 pF (voir C1 et C2). Les quartz que l'on peut utiliser pour les applications vont d'un minimum de 1 MHz à un maximum de 8 MHz. Les broches XTAL1 et XTAL2 sont respectivement l'entrée et la sortie d'un amplificateur inverseur qui est utilisé comme oscillateur.

Il est également possible de piloter le dispositif avec une horloge extérieure. Dans ce cas, il est nécessaire de déconnecter XTAL2 et de connecter XTAL1 à l'oscillateur externe.

## Interfaçage d'une mémoire SRAM externe

Vous pourriez avoir besoin de disposer d'une grande quantité de SRAM, il sera alors nécessaire d'ajouter un boîtier de mémoire externe.

Le microcontrôleur AT90S8515 prévoit cette possibilité par l'utilisation du Port A et du Port C.

Le Port A est utilisé aussi bien comme bus de données 8 bits que comme poids faible du bus d'adresses 16 bits. Le Port C est utilisé comme poids fort pour compléter le bus d'adresse 16 bits. Un "latch" est relié au Port A et est activé par la broche "ALE" du microcontrôleur (voir figure 2).

Il faut, en outre, également relier les broches "RD" (PD7) et "WR" (PD6) à la mémoire externe pour effectuer les opérations de lecture et d'écriture en mémoire. Vous observerez que ces deux signaux sont actifs à l'état bas.

Afin d'activer la mémoire externe, vous devez mettre à "1" le bit de poids fort du registre "MCUCR".

Ce bit est appelé "SRE".

Quand celui-ci est au niveau logique haut, alors la mémoire externe est active et donc toutes les broches du microcontrôleur consacrées à l'utilisation de la SRAM seront opérationnelles.

Pour comprendre comment le microcontrôleur interagit avec la mémoire, nous allons analyser son diagramme temporel (voir figures 3a et 3b). Le diagramme permet de compren-

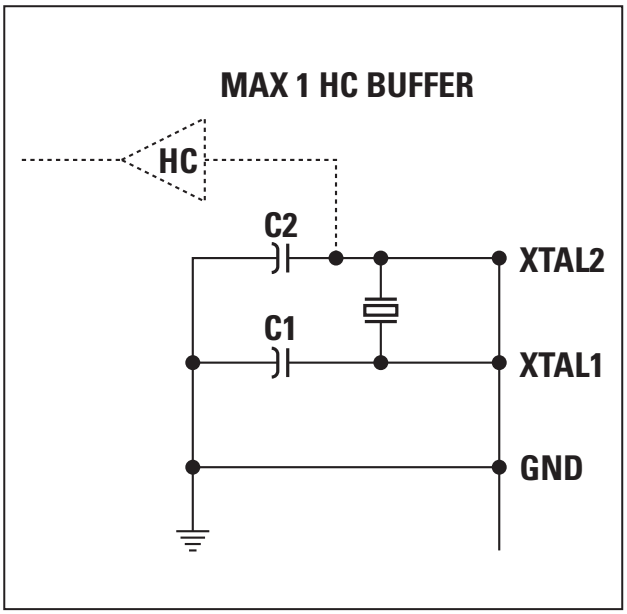



**Figure 1 : Connexions du quartz pour l'horloge.**

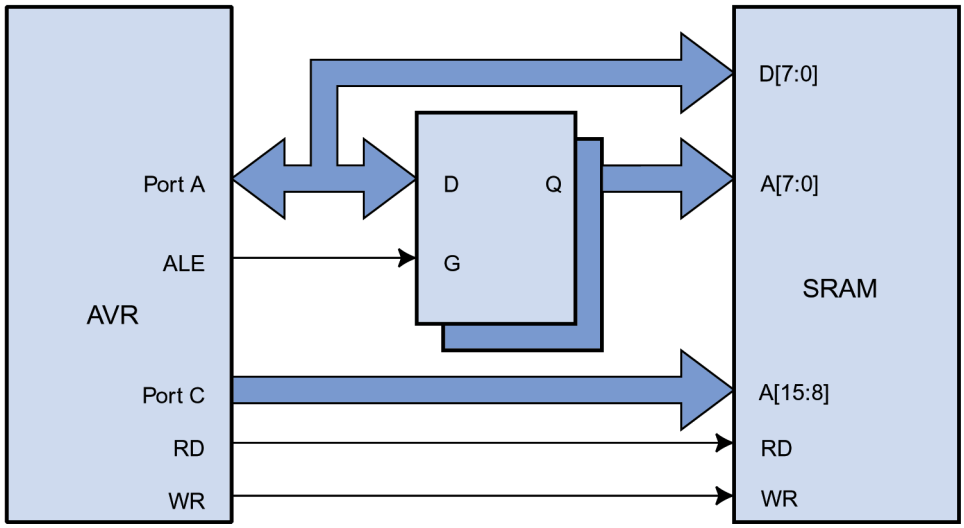



**Figure 2 : Voici comment interfacer une mémoire SRAM externe.**

dre que les signaux sont échantillonnés sur le front montant de l'horloge.

En plus de la "clock" qui rythme les différentes opérations, il est nécessaire de prendre en considération les broches "ALE", "WR" et "RD" qui différencient respectivement les opérations de placement, d'écriture et de lecture.

Lorsque le signal "ALE" passe d'une valeur logique haute à une valeur logique basse, l'adresse pointée dans la mémoire SRAM est valide.

Après quoi il faudra s'attendre ou à une opération de lecture en mémoire ou à une opération d'écriture dans celle-ci.

Si vous voulez faire une opération d'écriture en mémoire, vous devrez porter au niveau logique bas votre signal "WR" et, donc, vous écrirez sur le front montant de l'horloge la donnée en mémoire (voir le diagramme de la figure 3a). Si, au contraire, vous voulez lire, après avoir pointé, vous porterez le signal de "RD" à niveau logique bas et vous irez lire votre donnée sur le front montant.

Outre cette façon de faire, il en existe également une autre : celle avec la condition de "Wait State", qui introduit entre une opération de lecture ou d'écriture une pause qui dure un cycle d'horloge.

Le diagramme temporel correspondant (figure 3b) montre qu'il y a, après l'opération de lecture ou d'écriture, une période d'attente qui corres-

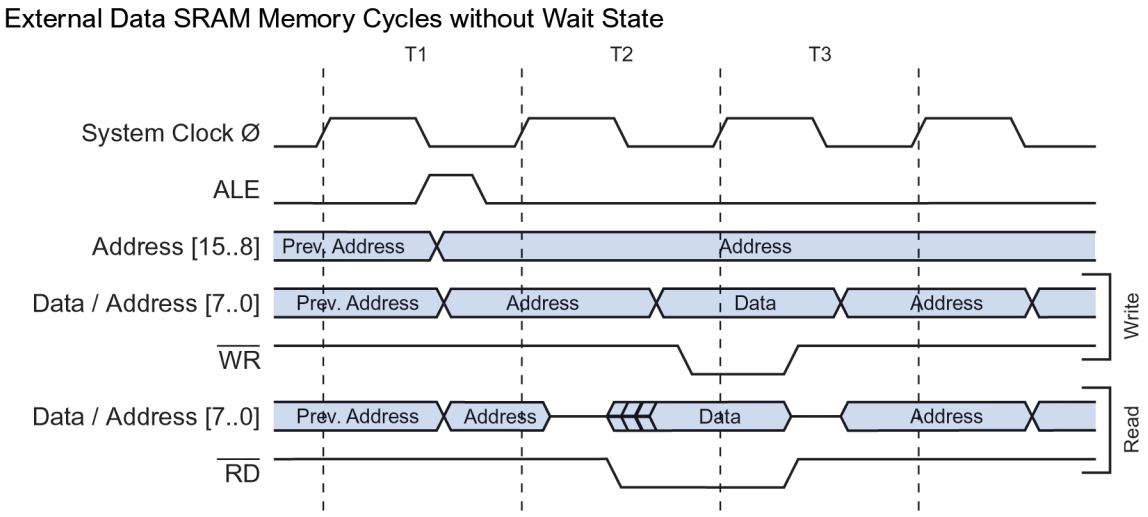



**Figure 3a : Cycle de lecture/écriture dans une SRAM externe sans «Wait State».**

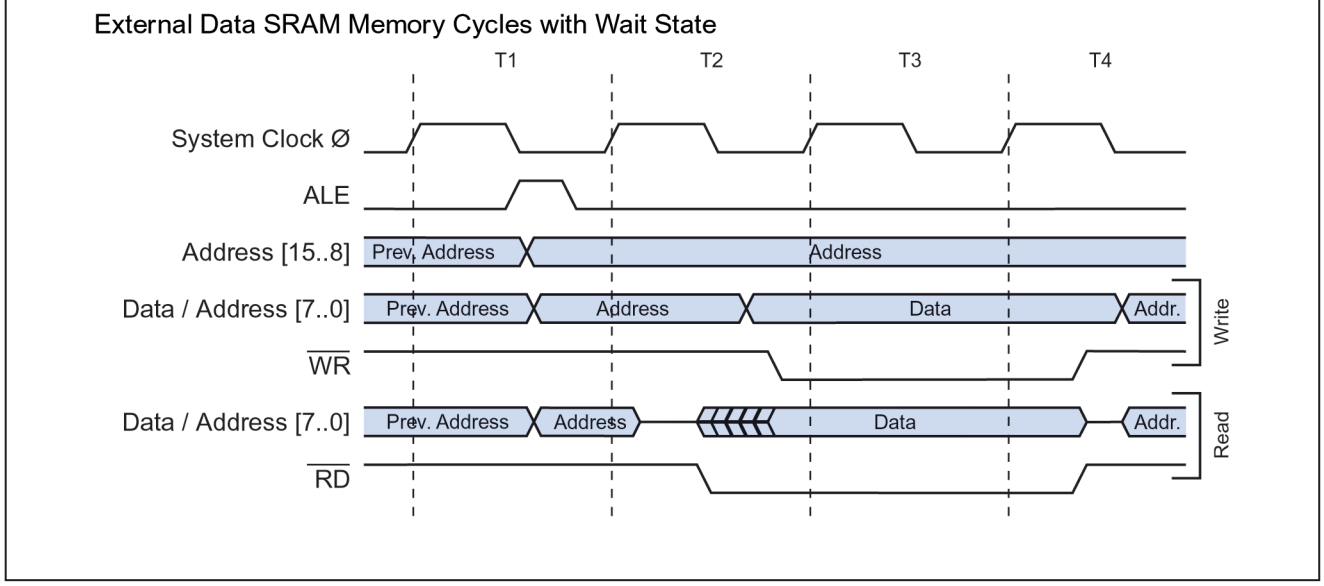



**Figure 3b : Cycle de lecture/écriture dans une SRAM externe avec «Wait State».**

pond à un cycle d'horloge. Ceci pourrait être nécessaire pour éviter de présenter des données à la mémoire lorsque celle-ci n'est pas encore prête à les recevoir parce qu'elle n'a pas encore fini d'exécuter la demande précédente par exemple.

## Comparateur analogique

A l'intérieur de l'AT90S8515 se trouve un comparateur analogique.

Son fonctionnement est très simple (voir figure 4).

Lorsque la tension sur la broche non inverseuse (positive) du comparateur est plus grande que la tension sur la broche inverseuse (négative), cela veut dire qu'il y a un niveau logique haut en sortie.

Le comparateur est programmé à travers un registre 8 bits appelé "ACSR" (registre d'état et de contrôle du comparateur analogique).

**Voyons en détail la fonction de chaque bit**

Bit 7 "ACD" : Ce bit est le poids fort du registre en question. Quand il est au niveau logique haut, alors le comparateur est désactivé. Ce bit peut être remis à "1" à chaque fois que le comparateur ne doit pas être utilisé, ce qui réduit la consommation d'énergie.

Bit 5 "ACO" : Contient la sortie logique du comparateur.

Bit 4 "ACI" : Ce bit va au niveau logique haut quand le comparateur génère une demande d'interruption. Pour faire en sorte que la routine correspondante soit exécutée, il est également nécessaire de mettre à 1 le bit ACIE ainsi que le bit 1 du registre d'état SREG.

Bit 3 "ACIE" : Quand il est à niveau logique haut (avec le bit 1 du SREG), il est alors possible de faire une demande d'interruption à travers l'utilisation du comparateur.

Bit 2 "ACIC" : Ce bit, s'il est au niveau logique haut, active le déclenchement de la broche "ICP" du timer 16 bits.

Bit 1 "ACIS1" et bit 0 "ACIS0" : Ces bits servent à sélectionner le mode de déclenchement (front montant, descendant ou sur niveau).

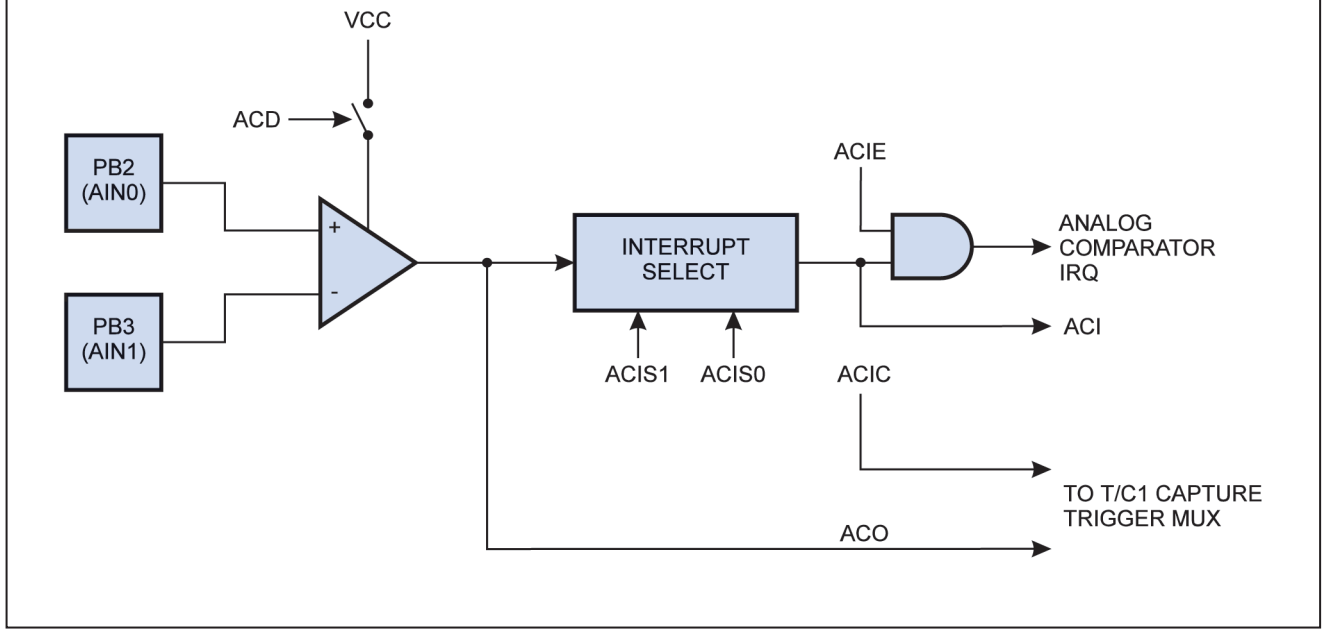



**Figure 4 : Fonctionnement du comparateur analogique de l'AT90S8515.**

## Mémoire EEPROM

Au début du cours, nous avons fait allusion à la présence d'une mémoire EEPROM interne. Nous allons maintenant vous la présenter de façon détaillée.

L'espace de mémoire EEPROM est de 512 bytes et donc, un seul registre 8 bits ne suffit pas pour pouvoir l'adresser.

Ce travail est donc effectué par le registre "EEAR". Plus précisément, ce registre est formé par "EEARH" poids fort de l'adresse, et par "EEARL" poids faible de l'adresse.



Vous verrez que pour adresser 512 bytes de mémoire, 9 bits suffisent, et donc "EEARL" sera utilisé entièrement, alors que seul le bit de poids faible de "EEARH" sera utilisé.

En plus des registres d'adresse, vous aurez besoin d'un registre de données et de contrôle, appelés respectivement "EEDR" et "EECR".

Dans le registre "EEDR" sera contenue la valeur de la donnée à écrire dans la mémoire EEPROM à l'adresse pointée par le registre "EEAR", ou bien la donnée lue par la mémoire à l'adresse pointée par le registre lui-même.

Le registre de contrôle de la mémoire EEPROM est utilisé en partie seulement et plus précisément ses trois bits de poids faible.

Pour pouvoir écrire en EEPROM, il est nécessaire que le bit "EEMWE" soit au niveau logique haut ainsi que le signal d'horloge généré par le bit "EEWE".

Si "EEWE" est au niveau logique bas, on ne pourra alors pas écrire en mémoire.

Le signal "EEWE" active l'écriture en mémoire.

Lorsque l'adresse et la donnée ont été correctement sélectionnées, vous devez porter au niveau haut le signal "EEWE" pour écrire dans la mémoire.

**Voici la procédure correcte à suivre pour écrire dans l'EEPROM**

- Contrôler et attendre jusqu'à ce que le bit "EEWE" soit à "0".
- Ecrire la nouvelle adresse de l'EEPROM dans les registres "EEARL" et "EEARL".
- Ecrire la nouvelle donnée à mettre dans l'EEPROM dans le registre "EECR".
- Mettre à l'état haut le bit "EEMWE" du registre "EECR".
- Mettre à l'état haut le bit "EEWE" du registre "EECR" avant que quatre cycles d'horloge ne soient passés. Autrement, "EEMWE" sera mis à "0" par le microcontrôleur lui-même.

Le bit "EERE" est le signal d'horloge pour la lecture dans la mémoire.

Lorsque l'adresse correcte a été établie, il est nécessaire de mettre à "1" logique ce bit pour activer la lecture de la mémoire.

Dès que "EERE" sera à l'état bas, la donnée sera présente dans le registre "EEDR". Lorsque "EERE" est remis à "1", le CPU attend deux cycles d'horloge.

***A suivre...***

◆ ***M. D.***

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 5

**Dans la première leçon nous avons introduit le concept de programmation “in-system”. Pour effectuer ce type de programmation on utilise une interface série à trois fils laquelle, connectée au programmateur correspondant, permet de charger le programme dans la mémoire Flash du microcontrôleur.**

En dehors de cette fonction, l’interface série peut être utilisée pour dialoguer avec d’autres dispositifs ou bien pour faire interagir deux microcontrôleurs de la famille AVR. On en déduira facilement que l’interface SPI (“Serial Peripheric Interface”) peut être utilisée pour se connecter à un grand nombre de dispositifs électroniques utilisant cette même technologie.

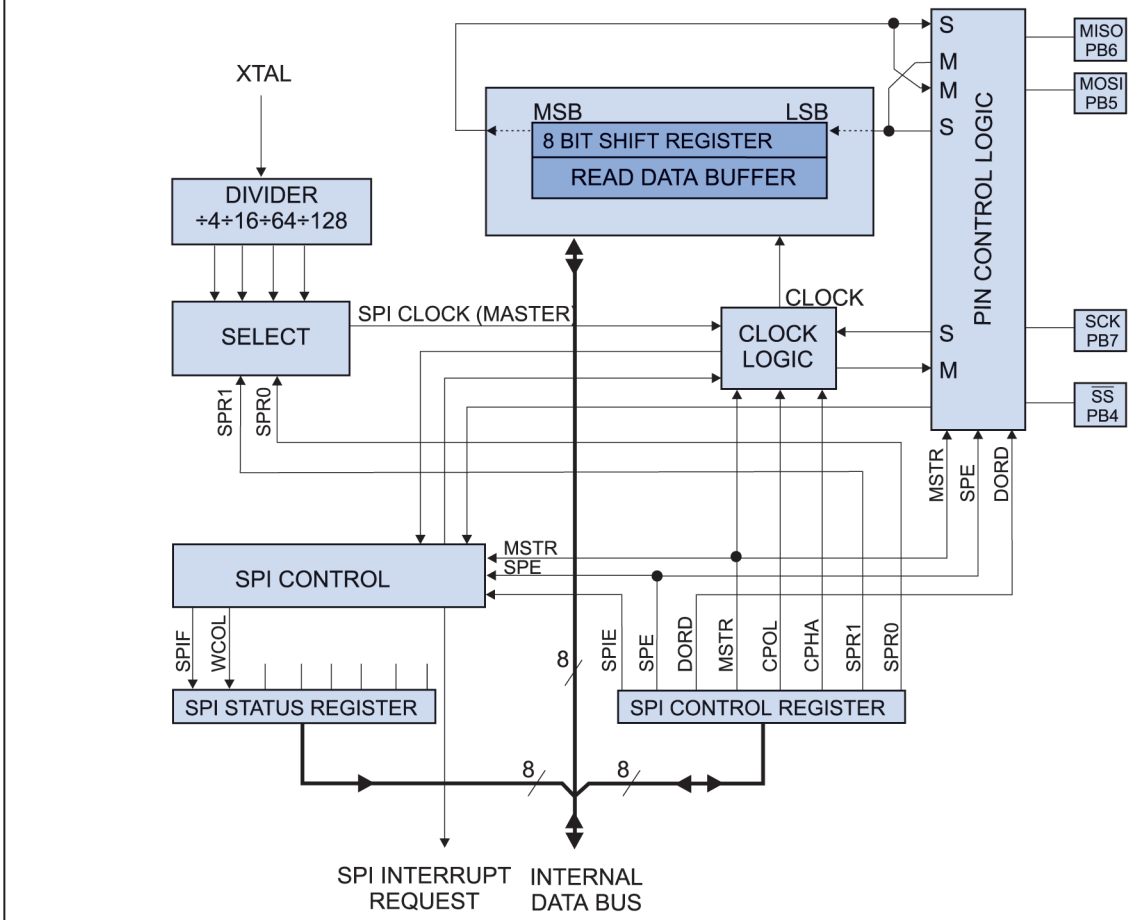



**Figure 1 : Représentation, de manière schématique, de l’interface série présente à l’intérieur du microcontrôleur.**

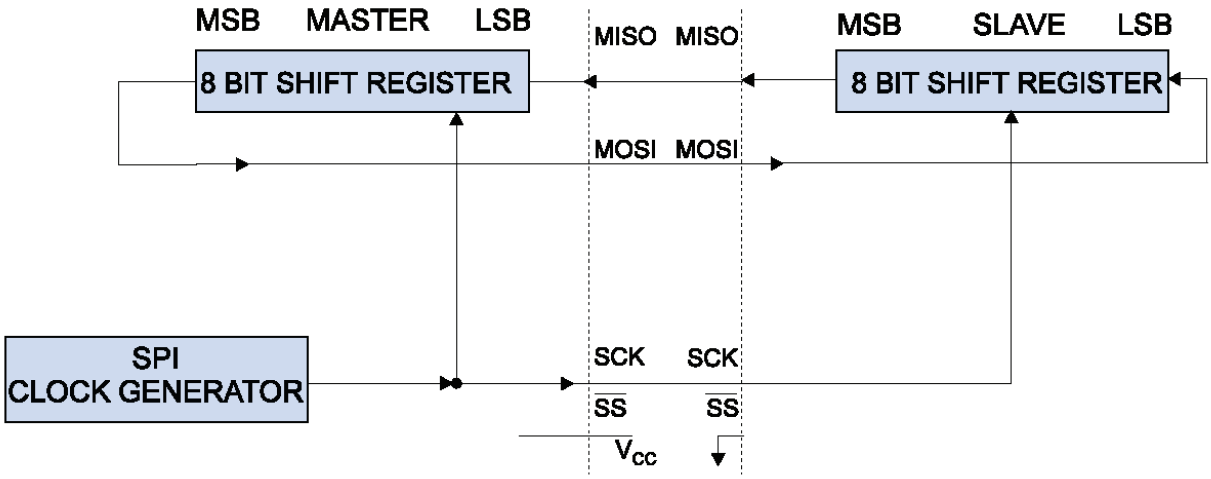



**Figure 2 : Interconnexion entre deux CPU AVR par l'intermédiaire du port série.**

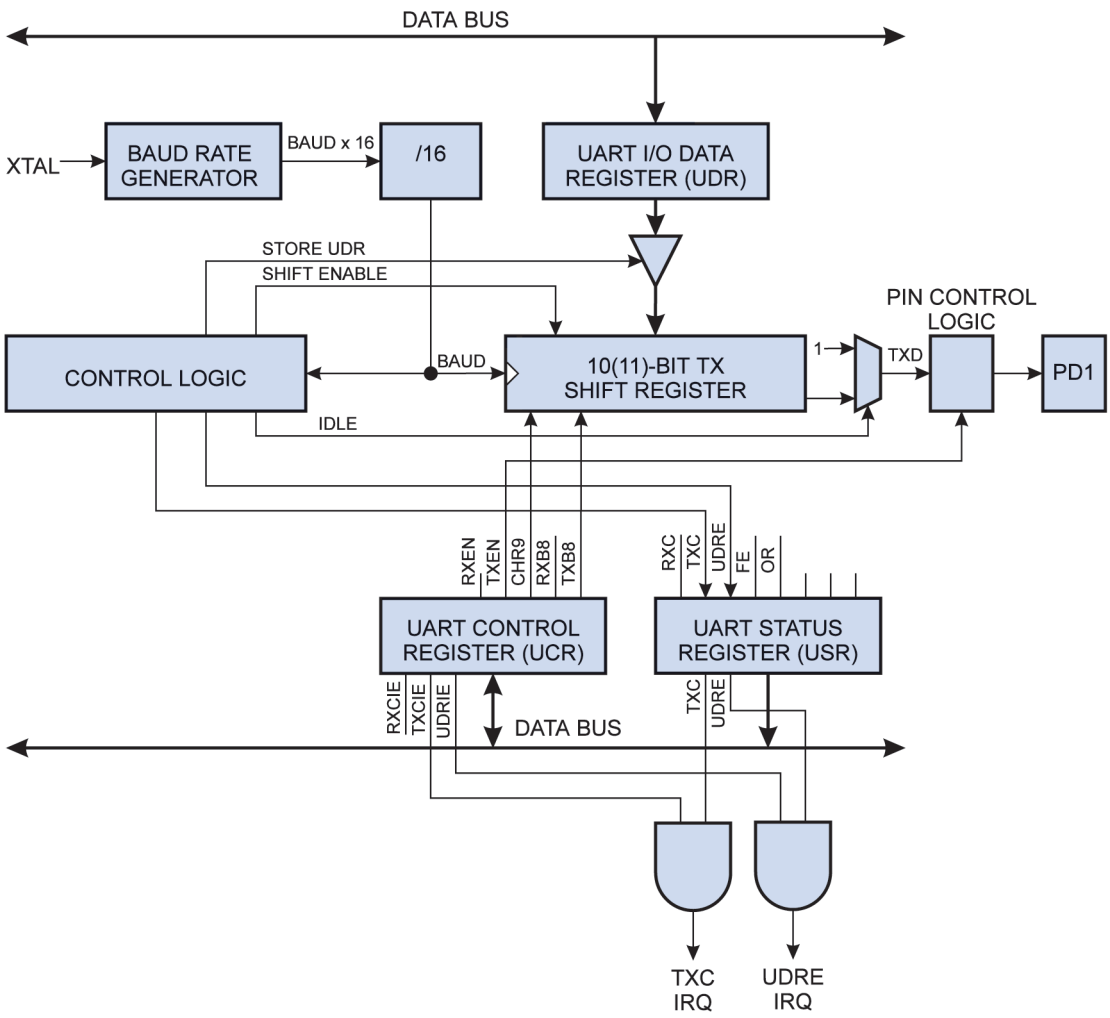



**Figure 3 : Organigramme de la section d'émission de l'UART.**

A l'intérieur du microcontrôleur AT90S8515 est intégrée une interface série synchrone à haute vitesse, présentant les caractéristiques suivantes :

- Transfert de données bidirectionnel sur deux lignes de communication distinctes ;

- Possibilité de décider quel dispositif est le Maître et lequel est l'Esclave ;

- Possibilité de transférer d'abord le bit le moins significatif de la donnée ou bien de transférer d'abord le bit le plus significatif ;

- Possibilité de programmer 4 vitesses de fonctionnement différentes de l'interface série ;

- "Flag" indiquant l'achèvement d'une émission ;

- "Flag" indiquant si une collision entre données s'est produite.

  Une collision se produit quand deux ou plusieurs dispositifs tentent d'accéder en même temps au même bus.

L'interface série contient un prédiviseur ("prescaler") programmable permettant d'obtenir une horloge ("clock") adaptée au fonctionnement de cette interface série.

Par exemple, avec une fréquence de travail pour la CPU de 8 MHz, l'interface série travaillera avec une fréquence maximale de 2 MHz jusqu'à un minimum de 62,5 kHz selon la programmation du prédiviseur.

En outre, un registre de contrôle appelé "SPI Controller Register" nécessaire pour régler tous les paramètres de fonctionnement de l'interface série et un registre d'état appelé "SPI Status Register" utilisé pour relever l'état des deux seuls "Flags" existants, sont présents.

Pour finir, nous avons un registre de données à 8 bits appelé simplement "SPI Data Register".

Celui-ci est un registre habilité tant en écriture qu'en lecture et il est utilisé pour transférer les données d'un registre interne à la CPU vers le registre de sortie de l'interface série ("SPI Shift Register") et vice-versa. Voir figure 1.

Comme on l'a vu précédemment, il est possible de relier, à travers l'interface série, deux microcontrôleurs.

Le dessin de la figure 2 montre ce type de connexion : une des deux CPU sera utilisée comme Maître et l'autre comme Esclave.

Le dispositif employé comme Maître aura sa broche d'horloge ("clock") SCK configurée comme sortie alors que le dispositif utilisé comme Esclave utilisera la même broche comme entrée.

En écrivant dans le registre des données ("SPI Data Register") du dispositif Maître, on fait démarrer la procédure de transfert de la donnée de la CPU Maître vers la CPU Esclave.

En premier le système active l'horloge de l'interface série et en même temps il commence à déplacer en sortie la donnée à émettre.

Une fois transférés les 8 bits, le dispositif Maître désactivera l'horloge, non sans avoir réglé le "Flag" de fin d'émission.

Les deux registres à 8 bits des dispositifs Maître et Esclave peuvent donc être considérés comme un seul "Shift Register" circulaire à 16 bits. Cela signifie que quand une donnée à 8 bits est déplacée du Maître vers l'Esclave, en même temps une donnée est déplacée de l'Esclave vers le Maître.

Ce qui signifie qu'en un cycle d'émission, les données présentes dans les deux registres sont interchangées.

## L'UART

A l'intérieur du AT90S8515 une seconde interface série, nommée UART ("Universal Asynchronous Receiver and Transmitter"), trouve sa place : elle est très semblable à celle qui se trouve sur les PC et qui permet d'effectuer des liaisons asynchrones avec n'importe quel périphérique externe (il n'est pas possible de programmer la mémoire du microcontrôleur par l'intermédiaire de l'UART).

L'UART présente dans le microcontrôleur a les caractéristiques suivantes :

- Différentes vitesses de transfert des données grâce au "Baud Generator" - programmable ;

- Haute vitesse de transfert des données, même à basse fréquence de fonctionnement de la CPU ;

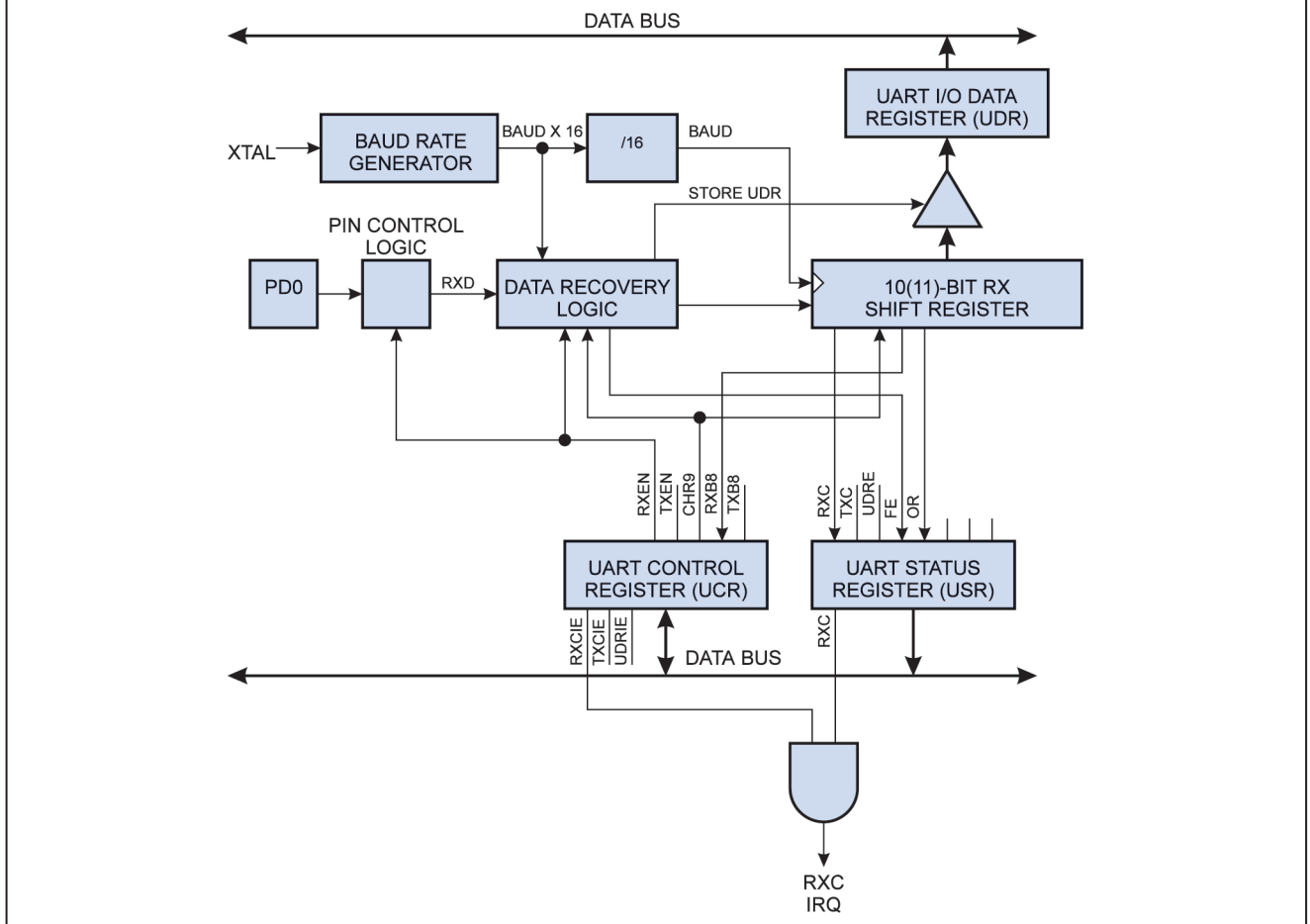



**Figure 4 : Organigramme de la section de réception de l'UART.**

- Possibilité d'envoyer des données à 8 ou 9 bits ;
- Système de filtrage du bruit ;
- Système de reconnaissance "faux bit de start" ;
- Système de reconnaissance de l'erreur de trame ("Framing Error") ;
- Présence de trois interruptions indiquant l'émission achevée, le registre d'émission des données vide et la réception achevée.

Les dessins de la figure 3 montrent, de manière schématique, comment sont réalisées les sections d'émission et réception de l'interface UART.

Dans les deux cas nous pouvons remarquer un bloc fonctionnel s'occupant de paramétrer la vitesse de communication : ce bloc est le "Baud Rate Generator".

Il existe ensuite un registre d'état et un registre de contrôle.

Pour finir, remarquons la présence d'un registre de données à 11 bits dans lequel sont chargés le bit de "Start", la donnée à 8 ou 9 bits et le bit de "Stop".

Tous ces registres, sauf le "Status Register", peuvent être aussi bien lus qu'écrits.

Observons en détail les registres d'état et de contrôle : le registre d'état de l'UART s'appelle USR.

Pour ce registre, les 4 bits les plus significatifs sont seuls utilisés.

Nous les décrivons ci-dessous :

- **Bit 7 RXC** : Ce bit est au niveau logique haut (1) quand la donnée reçue de l'UART a été transférée dans le registre UDR. RXC passe au niveau logique bas (0) quand on lit le registre UDR.
- **Bit 6 TXC** : Ce bit est au niveau logique haut (1) quand la donnée reçue plus son bit de "stop" ont été émis par l'UART. Ce bit est utilisé pour les communications "half-duplex".
- **Bit 5 UDRE** : Ce bit indique quand l'UART est prête à recevoir une nouvelle donnée à émettre.
- **Bit 4 FE** : Ce bit est au niveau logique haut (1) quand se produit une erreur de trame. Par exemple, si le bit de "stop" est au niveau logique bas (0), ceci est considéré comme une erreur car le bit de "stop" ne peut être qu'au niveau logique haut (1).
- **Bit 3 OR** : Ce bit indique une situation "d'Overrun".

Un autre registre présent dans l'UART est le registre de contrôle appelé UCR.

Les trois bits les plus significatifs de ce registre servent à produire les 3 interruptions relatives au périphérique UART. Les 5 autres bits, en revanche, sont des paramétrages spécifiques de l'UART :

| Baud Rate | 1 MHz | %Error | 1.8432 MHz | %Error | 2 MHz | %Error | 2.4576 MHz | %Error |
|---|---|---|---|---|---|---|---|---|
| 2400 | UBRR= 25 | 0.2 | UBRR= 47 | 0.0 | UBRR= 51 | 0.2 | UBRR= 63 | 0.0 |
| 4800 | UBRR= 12 | 0.2 | UBRR= 23 | 0.0 | UBRR= 25 | 0.2 | UBRR= 31 | 0.0 |
| 9600 | UBRR= 6 | 7.5 | UBRR= 11 | 0.0 | UBRR= 12 | 0.2 | UBRR= 15 | 0.0 |
| 14400 | UBRR= 3 | 7.8 | UBRR= 7 | 0.0 | UBRR= 8 | 3.7 | UBRR= 10 | 3.1 |
| 19200 | UBRR= 2 | 7.8 | UBRR= 5 | 0.0 | UBRR= 6 | 7.5 | UBRR= 7 | 0.0 |
| 28800 | UBRR= 1 | 7.8 | UBRR= 3 | 0.0 | UBRR= 3 | 7.8 | UBRR= 4 | 6.3 |
| 38400 | UBRR= 1 | 22.9 | UBRR= 2 | 0.0 | UBRR= 2 | 7.8 | UBRR= 3 | 0.0 |
| 57600 | UBRR= 0 | 7.8 | UBRR= 1 | 0.0 | UBRR= 1 | 7.8 | UBRR= 2 | 12.5 |
| 76800 | UBRR= 0 | 22.9 | UBRR= 1 | 33.3 | UBRR= 1 | 22.9 | UBRR= 1 | 0.0 |
| 115200 | UBRR= 0 | 84.3 | UBRR= 0 | 0.0 | UBRR= 0 | 7.8 | UBRR= 0 | 25.0 |

| Baud Rate | 3.2768 MHz | %Error | 3.6864 MHz | %Error | 4 MHz | %Error | 4.608 MHz | %Error |
|---|---|---|---|---|---|---|---|---|
| 2400 | UBRR= 84 | 0.4 | UBRR= 95 | 0.0 | UBRR= 103 | 0.2 | UBRR= 119 | 0.0 |
| 4800 | UBRR= 42 | 0.8 | UBRR= 47 | 0.0 | UBRR= 51 | 0.2 | UBRR= 59 | 0.0 |
| 9600 | UBRR= 20 | 1.6 | UBRR= 23 | 0.0 | UBRR= 25 | 0.2 | UBRR= 29 | 0.0 |
| 14400 | UBRR= 13 | 1.6 | UBRR= 15 | 0.0 | UBRR= 16 | 2.1 | UBRR= 19 | 0.0 |
| 19200 | UBRR= 10 | 3.1 | UBRR= 11 | 0.0 | UBRR= 12 | 0.2 | UBRR= 14 | 0.0 |
| 28800 | UBRR= 6 | 1.6 | UBRR= 7 | 0.0 | UBRR= 8 | 3.7 | UBRR= 9 | 0.0 |
| 38400 | UBRR= 4 | 6.3 | UBRR= 5 | 0.0 | UBRR= 6 | 7.5 | UBRR= 7 | 6.7 |
| 57600 | UBRR= 3 | 12.5 | UBRR= 3 | 0.0 | UBRR= 3 | 7.8 | UBRR= 4 | 0.0 |
| 76800 | UBRR= 2 | 12.5 | UBRR= 2 | 0.0 | UBRR= 2 | 7.8 | UBRR= 3 | 6.7 |
| 115200 | UBRR= 1 | 12.5 | UBRR= 1 | 0.0 | UBRR= 1 | 7.8 | UBRR= 2 | 20.0 |

| Baud Rate | 7.3728 MHz | %Error | 8 MHz | %Error | 9.216 MHz | %Error | 11.059 MHz | %Error |
|---|---|---|---|---|---|---|---|---|
| 2400 | UBRR= 191 | 0.0 | UBRR= 207 | 0.2 | UBRR= 239 | 0.0 | UBRR= 287 | - |
| 4800 | UBRR= 95 | 0.0 | UBRR= 103 | 0.2 | UBRR= 119 | 0.0 | UBRR= 143 | 0.0 |
| 9600 | UBRR= 47 | 0.0 | UBRR= 51 | 0.2 | UBRR= 59 | 0.0 | UBRR= 71 | 0.0 |
| 14400 | UBRR= 31 | 0.0 | UBRR= 34 | 0.8 | UBRR= 39 | 0.0 | UBRR= 47 | 0.0 |
| 19200 | UBRR= 23 | 0.0 | UBRR= 25 | 0.2 | UBRR= 29 | 0.0 | UBRR= 35 | 0.0 |
| 28800 | UBRR= 15 | 0.0 | UBRR= 16 | 2.1 | UBRR= 19 | 0.0 | UBRR= 23 | 0.0 |
| 38400 | UBRR= 11 | 0.0 | UBRR= 12 | 0.2 | UBRR= 14 | 0.0 | UBRR= 17 | 0.0 |
| 57600 | UBRR= 7 | 0.0 | UBRR= 8 | 3.7 | UBRR= 9 | 0.0 | UBRR= 11 | 0.0 |
| 76800 | UBRR= 5 | 0.0 | UBRR= 6 | 7.5 | UBRR= 7 | 6.7 | UBRR= 8 | 0.0 |
| 115200 | UBRR= 3 | 0.0 | UBRR= 3 | 7.8 | UBRR= 4 | 0.0 | UBRR= 5 | 0.0 |

**Figure 5 : Tableau de paramétrage du “Baud Rate” de l’UART présente dans les microcontrôleurs ATMEL.**

- **Bit 4 RXEN** : Ce bit, s’il est au niveau logique haut (1), active l’UART pour la réception ;

- **Bit 3 TXEN** : Ce bit, s’il est au niveau logique haut (1), active l’UART pour l’émission ;

- **Bit 2 CHR9** : Si ce bit est au niveau logique haut (1), cela signifie que la donnée émise est à 9 bits. A celui-ci sont ajoutés le bit de “stop” et celui de “start” ;

- **Bit 1 RXB8** : Sert à mémoriser le bit le plus significatif de la donnée reçue quand la longueur de la donnée est réglée à 9 bits ;

- **Bit 0 TXB8** : Sert à mémoriser le bit le plus significatif de la donnée émise quand la longueur de la donnée est réglée à 9 bits.

Outre ces deux registres, il en existe un pour programmer le Baud Rate du périphérique.

La valeur à insérer dans ce registre à 8 bits est à trouver dans la figure 5 qui tient compte du Baud Rate à produire (compris entre un minimum de 2 400 bps et un maximum de 115 200 bps) et de la fréquence d’horloge utilisée ; en outre, le pourcentage d’erreur possible est indiqué.

◆ ***M. D.***

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 6

**Après avoir vu, au cours des leçons précédentes, les ressources internes des microcontrôleurs Atmel, nous allons commencer à analyser les instructions principales du AT90S8515.**

**Chaque microcontrôleur possède un "set" des instructions Assembleur nécessaires pour le programmer, de manière à rendre disponibles ses circuits internes pour le développement des opérations requises par le système dans lequel il sera ensuite inséré.**

Un programme écrit en Assembleur est constitué d'une série de déclarations, ou "statements", dont chacune peut représenter une série d'informations ; il y a ensuite les étiquettes, ou "labels", le code d'opération, représentant les instructions que le microcontrôleur est capable d'exécuter, les opérateurs, c'est-à-dire les éléments (registres ou localisation ["location"] des mémoires) sur lesquels les instructions doivent agir.

Nous pouvons en outre trouver des commentaires, c'est-à-dire des indications nécessaires au programmeur pour comprendre à quoi sert la portion de code commentée.

L'AT90S8515 possède un "set" étendu des instructions Assembleur, composé de 118 instructions.

Celles-ci peuvent être subdivisées en quatre catégories distinctes :

- Instructions arithmétiques logiques.

- Instructions de saut.

- Instructions de transfert de données.

- Instructions sur les Bits et de Test sur les Bits.

Toutes les instructions ne seront pas décrites car certaines se ressemblent beaucoup ; souvent les seules différences sont relatives aux registres auxquels elles sont appliquées, ou bien aux bits du registre qui sont gérés.

## Instructions arithmético-logiques

Nous pouvons donc commencer à décrire les instructions arithmético-logiques présentes dans le "set" des instructions pour microcontrôleurs AT90S8515 :

**ADD Somme sans report**

Description : cette instruction permet d'effectuer la somme de deux registres sans tenir compte du Flag de report C.

**Syntaxe :** **ADD Rd,Rr**

*Note : Le résultat se situe à l'intérieur du registre Rd.*

Exemple : Add r1,r2

Résultat : exécute la somme r2 avec r1 et pose le résultat en r1.

| | | |
|---|---|---|
| **Rd** | **=** | **Registre destination** |
| **Rr** | **=** | **Registre source** |
| **R** | **=** | **Résultat après que l'instruction ait été exécutée** |
| **K** | **=** | **Constante** |
| **k** | **=** | **Adresse** |
| **b** | **=** | **Bit d'un registre** |
| **s** | **=** | **Bit présent dans le Status Register** |
| **X,Y,Z** | **=** | **Registres à adressage indirect** |
| **A** | **=** | **Adresse d'une localisation ("location") de I/O** |
| **q** | **=** | **Déplacement par adressage direct** |

**Figure 1 : Légende.**

**ADC Somme avec report**

Description : additionne le contenu des deux registres en tenant compte du Flag de report C.

**Syntaxe : ADC Rd,Rr**

*Note : Le résultat est situé dans le registre Rd.*

```
Exemple :   Add   r2,r0   ; somme occasionnant
                          ; un report
            Adc   r3,r1
```

Résultat : Additionne les registres r3 et r1 en tenant compte du report de l'opération précédente.

**SUB Soustraction sans report**

Description : cette instruction effectue la soustraction entre deux registres.

**Syntaxe : SUB Rd,Rr**

*Note : Le résultat est situé dans le registre Rd.*

```
Exemple :   Sub   r13,r12
```

Résultat : soustrait le contenu de r12 du contenu de registre r13.

**SBC Soustraction avec report**

Description : cette instruction fait la soustraction entre deux registres et tient compte de l'éventuel Flag de report C.

**Syntaxe : SBC Rd,Rr**

*Note : Le résultat est situé dans le registre Rd.*

```
Exemple :   Sub   r2,r0   ; soustraction occasionnant
                          ; un report
            Sbc   r3,r1
```

Résultat : exécute une soustraction en tenant compte du report occasionné précédemment.

**SBCI Soustraction immédiate avec report**

Description : cette instruction soustrait une constante du contenu du registre en tenant compte du report. Le résultat est situé dans le registre Rd.

**Syntaxe : SBCI Rd,K**

*Note : Le résultat est situé dans le registre Rd.*

```
Exemple :   Subi  r16,$23  ; soustraction
                           ; occasionnant un report
            Sbci  r17,$4F
```

Résultat : soustrait la constante du registre r17 en tenant compte du précédent report.

**AND Logique**

Description : exécute le AND logique entre les registres Rd et Rr.

**Syntaxe : AND Rd,Rr**

*Note : Le résultat est situé dans le registre Rd.*

```
Exemple :   And   r2,r3
```

Résultat : exécute le AND bit par bit et place le résultat en r2.

**OR Logique**

Description : cette instruction exécute le OR logique entre le contenu des deux registres Rd et Rr.

**Syntaxe : OR Rd,Rr**

*Note : Le résultat est situé dans le registre Rd.*

```
Exemple :   Or    r15,r16
```

Résultat : exécute le OR bit par bit entre les deux registres.

**COM Complément à un**

Description : cette instruction exécute le complément à un du registre Rd.

**Syntaxe : COM Rd**

*Note : cette instruction change le registre d'état, en particulier elle règle le Flag C.*

```
Exemple :   Com   r4
```

Résultat : exécute le complément à un du registre r4 et règle le Flag C du registre d'état.

**SBR Règle un bit dans un registre**

Description : cette instruction règle le bit spécifié dans le registre Rd.

**Syntaxe : SBR Rd,K**

*Note : cette instruction influence le Registre d'Etat SREG.*

```
Exemple :   Sbr   r16,3
```

Résultat : cette instruction règle les bits 0 et 1 du registre r16.

| | | |
|---|---|---|
| **ADD** | **Rd, Rr** | **Somme sans report** |
| **ADC** | **Rd, Rr** | **Somme avec report** |
| **SUB** | **Rd, Rr** | **Soustraction sans report** |
| **SBC** | **Rd, Rr** | **Soustraction avec report** |
| **SBCI** | **Rd, Rr** | **Soustraction immédiate avec report** |
| **AND** | **Rd, Rr** | **AND Logique** |
| **OR** | **Rd, Rr** | **OR Logique** |
| **COM** | **Rd** | **Complément à 1** |
| **SBR** | **Rd, K** | **Règle un bit d'un registre** |
| **CBR** | **Rd, K** | **Réinitialise un bit d'un registre** |
| **MUL** | **Rd, Rr** | **Multiplication non signée** |

**Figure 2 : Instructions arithmético-logiques.**

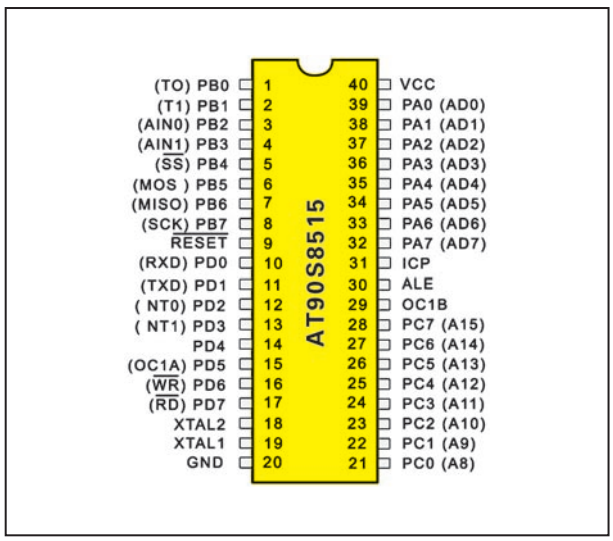



**Figure 3 : Brochage du microcontrôleur AT90S8515, vu de dessus.**

### CBR Réinitialise un bit dans un registre

Description : cette instruction réinitialise un bit à l'intérieur du registre Rd.

**Syntaxe : CBR Rd,K**

*Note : cette instruction influence le Registre d'Etat SREG.*

Exemple : Cbr r16,$F0

Résultat : cette instruction réinitialise les quatre bits les plus significatifs du registre r16.

### MUL Multiplication non signée

Description : cette instruction exécute une multiplication entre deux registres à 8 bits et met le résultat dans un registre à 16 bits sans tenir compte du signe.

**Syntaxe : MUL Rd,Rr**

*Note : les registres Rd et Rr contiennent des données non signées. Le résultat de la multiplication est situé pour moitié dans le registre R0 et pour moitié dans le registre R1. Si le multiplicande ou le multiplicateur sont mis dans les registres R0 ou R1, pendant l'opération, leur contenu est effacé par réécriture du résultat.*

Exemple : Mul r5,r4
Movw r4,r0

Résultat : exécute la multiplication non signée entre le contenu du registre r5 et r4 et ensuite copie le résultat de la multiplication, se trouvant dans les registres r0 et r1, dans les registres r4 et r5.

Et c'est avec cette instruction que nous concluons ce panorama des instructions arithmétiques et logiques et que nous vous donnons rendez-vous au mois prochain pour une nouvelle leçon dans laquelle nous analyserons les instructions de saut.

◆ *M. D.*

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 7

**Dans la leçon précédente nous avons introduit une série de fonctions dédiées aux ATMEL AVR et relatives aux opérations arithmétiques et logiques. Aujourd'hui nous poursuivons avec une partie fondamentale de n'importe quel langage de programmation à haut ou à bas niveau : la gestion des sauts.**

Dans l'exécution d'un programme, en effet, il est nécessaire, pour pouvoir effectuer des opérations diverses en fonction des cas se présentant, d'exécuter des parties du programme dans un autre ordre que séquentiel. Par exemple, si nous devons visualiser sur un afficheur une écriture identifiant une touche pressée, il est nécessaire de pouvoir sauter, en fonction de la pression de la touche, à la partie de la source visualisant l'écriture correspondante. Voyons donc comment gérer ces situations en Assembleur.

**JMP : Saut**

Description : Exécute un saut sans condition à une adresse de mémoire indiquée par une étiquette.

**Syntaxe : JMP k**

*Note : Cette instruction n'influence pas le registre d'état SREG.*

```
Exemple :       Mov     r1,r0
                Jmp     pluto
                .....
                .....
          pluto : Mov     rr3,r2
                ....
```

Résultat : Copie le contenu du registre r0 dans le registre r1 et saute sans condition à la routine pluto.

**CALL : Appel d'une "subroutine"**

Description : Appelle un sous-programme ("subroutine") à exécuter avant de revenir au flux normal du programme. Cette instruction est très utile dans le cas où l'on doit utiliser une série d'instructions toujours les mêmes dans de nombreuses parties du programme. En effet, on fait une seule copie de ces instructions, formant justement une "subroutine", lesquelles seront appelées le cas échéant.

Quand on utilise un "Call", dans le "Stack Pointer" (pointeur de mémoire spéciale), est sauvegardée l'adresse de mémoire de l'instruction suivant l'appel du "Call" : ainsi, quand la "subroutine" finit d'exécuter ces instructions, dans le "Program Counter" (compteur de programme) est chargée l'adresse de mémoire précédemment sauvegardée dans le "Stack Pointer" et le programme continue son flux d'instructions exactement là où il l'avait interrompu.

**Syntaxe : CALL k**

*Note : Le registre d'état SREG n'est pas influencé par l'instruction "Call".*

```
Exemple :       Mov     r16,r0
                Call    pippo
                ....
          Pippo : Cpi     r16,$42
                ....
                Ret
```

| | | |
|---|---|---|
| **Rd** | **=** | **Registre destination** |
| **Rr** | **=** | **Registre source** |
| **R** | **=** | **Résultat après que l'instruction ait été exécutée** |
| **K** | **=** | **Constante** |
| **k** | **=** | **Adresse** |
| **b** | **=** | **Bit d'un registre** |
| **s** | **=** | **Bit présent dans le Status Register** |
| **X,Y,Z** | **=** | **Registres à adressage indirect** |
| **A** | **=** | **Adresse d'une localisation ("location") de I/O** |
| **q** | **=** | **Déplacement par adressage direct** |

**Figure 1 : Légende.**

Résultat : A chaque appel (“Call”) le programme exécute les instructions présentes sous l’étiquette “Pippo” jusqu’à l’instruction “RET”. Celle-ci trouvée, le contrôle revient à l’instruction suivant le “Call”.

**RET : Revenir d’une “subroutine”**

Description : Permet de revenir d’une instruction “Call”. L’adresse de retour est lue dans le “Stack Pointer”. Ce dernier utilise un schéma de pré-augmentation pendant une instruction de Ret. Le registre d’état SREG n’est pas influencé par cette instruction.

**Syntaxe :** **RET**

**RETI : Revenir d’un appel d’interrupt**

Desription : Sert à retourner au programme principal après un appel d’interrupt. Dans ce cas aussi, l’adresse de retour est lue dans le “Stack Pointer”. Le registre d’état n’est pas automatiquement sauvegardé quand arrive un appel d’interrupt ; en outre, quand on revient d’un appel d’interrupt, on doit restaurer la valeur précédente du registre d’état SREG.

**Syntaxe :** **RETI**

**CP : Instruction de comparaison**

Description : Sert à faire la comparaison entre le contenu du registre Repère détrompeur et le contenu du registre Rr. Aucune donnée contenue dans les registres n’est modifiée. Tous les sauts conditionnels disponibles sont utilisés après cette instruction.

**Syntaxe :** **CP** **Rd,Rr**

Exemple :
```
Cp    r4,r19
Brne  pippo
```

Résultat : Effectue la comparaison entre le contenu du registre r4 et le contenu du registre r19 et saute à l’étiquette “pippo” si le contenu des deux registres est différent (BRNE = Branch if Not Equal).

## Les autres intructions de saut

Outre les instructions que nous venons de décrire, d’autres instructions de saut, reportées dans le tableau de la figure 2, sont disponibles. Celles-ci vérifient l’état de l’un des huit bits se trouvant dans le “Status Register” (registre d’état) et agissent en conséquence.

## Les instructions pour le transfert des données et la manipulation des bits des registres

**MOV : Copie un registre**

Description : Sert à faire une copie d’un registre. Le registre source é Rr tandis que le registre destination est Rd. Dans Rd se trouvera une copie de Rr.

**Syntaxe :** **MOV** **Rd,Rr**

**LDI : Chargement immédiat**

Description : Permet de charger une constante à 8 bits à l’intérieur des registres du 16 au 31.

**Syntaxe :** **LDI** **Rd,K**

Exemple :
```
Clr   r31
Ldi   r30,$F0
```

Résultat : Est remise à zéro la partie la plus significative du registre Z et réglée la partie la moins significative avec la valeur $F0.

| Mnemonic | Operands | Description | Operation | Flags | # Clocks |
|---|---|---|---|---|---|
| SBRC | Rr, b | Skip if Bit in Register Cleared | if (Rr(b) = 0) PC ← PC + 2 or 3 | None | 1/2/3 |
| SBRS | Rr, b | Skip if Bit in Register is Set | if (Rr(b) = 1) PC ← PC + 2 or 3 | None | 1/2/3 |
| SBIC | P, b | Skip if Bit in I/O Register Cleared | if (P(b) = 0) PC ← PC + 2 or 3 | None | 1/2/3 |
| SBIS | P, b | Skip if Bit in I/O Register is Set | if (P(b) = 1) PC ← PC + 2 or 3 | None | 1/2/3 |
| BRBS | s, k | Branch if Status Flag Set | if (SREG(s) = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRBC | s, k | Branch if Status Flag Cleared | if (SREG(s) = 0) then PC ← PC + k + 1 | None | 1/2 |
| BREQ | k | Branch if Equal | if (Z = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRNE | k | Branch if Not Equal | if (Z = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRCS | k | Branch if Carry Set | if (C = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRCC | k | Branch if Carry Cleared | if (C = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRSH | k | Branch if Same or Higher | if (C = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRLO | k | Branch if Lower | if (C = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRMI | k | Branch if Minus | if (N = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRPL | k | Branch if Plus | if (N = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRGE | k | Branch if Greater or Equal, Signed | if (N ⊕ V = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRLT | k | Branch if Less Than Zero, Signed | if (N ⊕ V = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRHS | k | Branch if Half-carry Flag Set | if (H = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRHC | k | Branch if Half-carry Flag Cleared | if (H = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRTS | k | Branch if T-flag Set | if (T = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRTC | k | Branch if T-flag Cleared | if (T = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRVS | k | Branch if Overflow Flag is Set | if (V = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRVC | k | Branch if Overflow Flag is Cleared | if (V = 0) then PC ← PC + k + 1 | None | 1/2 |
| BRIE | k | Branch if Interrupt Enabled | if (I = 1) then PC ← PC + k + 1 | None | 1/2 |
| BRID | k | Branch if Interrupt Disabled | if (I = 0) then PC ← PC + k + 1 | None | 1/2 |

**Figure 2 : Les autres instructions de saut.**

**LD : Chargement indirect de l'espace de données vers un registre utilisant l'indice Z**

Description : Lit un octet indirectement, avec ou sans espacement, de l'espace données et le met dans un registre. Les localisations de mémoire concernant les données sont pointées dans le registre Z (16 bits). L'accès à la mémoire est limité à 64 octets. Le registre pointeur Z peut subir une post-augmentation ou une pre-diminution ou bien demeurer inchangé. Grâce à ces caractéristiques, le registre Z peut être aussi utilisé comme "Stack Pointer" et ceci vaut également pour les registres X et Y. Pour les dispositifs ayant seulement 256 octets d'espace données, on utilise seulement la partie la moins significative du registre Z.

```
Syntaxe :   LD    Rd,Z
            LD    Rd,Z+     ; Post-augmentation
            LD    Rd,-Z     ; Pre-diminution

Exemple :   Clr   r31
            Ldi   r30,$60
            Ld    r0,Z+
            Ld    r1,Z
```

Résultat : La partie la plus significative du registre Z est remise à zéro et la partie la moins significative réglée avec $60 ; la valeur contenue dans la localisation de mémoire $60 est chargée dans le registre r0 ; Z est augmenté de un et la valeur contenue dans la localisation de mémoire $61 est chargée dans le registre r1.

**IN : Charge une donnée des I/O dans un registre**

Description : Cette instruction charge une donnée de l'espace I/O (Porte A,B,C,D, Timers, registres de configuration) et la met dans le registre Rd.

```
Syntaxe :   IN    Rd,A

Exemple :   In    r25,$16
            Cpi   r25,4
            Breq  Exit
```

Résultat : Une donnée du port B est chargée dans le registre Rd qui est comparé à la constante 4. Saute à l'étiquette Exit si le contenu de r25 = 4.

**PUSH**

Description : Sauvegarde le contenu du registre Rr dans le "Stack Pointer". Celui-ci est post-diminué de 1. Cette instruction est utilisée quand on fait appel à des "subroutines".

```
Syntaxe :   PUSH  Rr
```

**POP**

Description : Charge dans le registre Rd la valeur prélevée dans le "Stack Pointer". Celui-ci est pré-augmenté de 1 avant d'exécuter la POP.

```
Syntaxe :   POP   Rd
```

**LSL : Shift logique à gauche**

Description : Déplace tous les bits du registre Rd à gauche. Le bit 0 est remis à zéro tandis que le bit 7 est chargé dans le Flag C du registre SREG. Cette instruction, pratiquement, exécute la multiplication par deux de la donnée signée et non signée.

```
Syntaxe :   LSL   Rd
```

**LSR : Shift logique à droite**

Description : Déplace tous les bits du registre Rd à droite. Le bit 7 est mis à zéro tandis que le bit 0 est chargé dans le Flag C. Cette instruction exécute une division par deux de la donnée non signée.

```
Syntaxe :   LSR   Rd
```

**ASR : Shift à droite arithmétique**

Description : Déplace tous les bits de Rd à droite d'une position. Le bit 7 est maintenu constant tandis que le bit 0 est chargé dans le Flag C du registre SREG. Cette instruction divise une donnée par deux.

```
Syntaxe :   ASR   Rd
```

**SEI : Set Interrupt Globaux**

Description : Habilite les interrupts réglant le Flag I du registre SREG.

```
Syntaxe :   SEI
```

**CLI : Déshabilite Interrupts Globaux**

Description : Déshabilite les interrupts. En effet, il remet à zéro le Flag I du "Status Register" SREG.

```
Syntaxe :   CLI
```

**Sleep**

Description : Met en "sleep" le microcontrôleur et agit sur le registre MCUCR. En particulier sur le bit 4 dit de "Sleep Mode" (SM).

```
Syntaxe :   SLEEP

Exemple :   mov   r0,r11
            Ldi   r16,(1<<SE)      Out
MCUCR,r16
            Sleep
```

Résultat : Habilite le mode "SLEEP" et met MCU dans ce mode.

Donnons nous rendez-vous à la prochaine leçon pour continuer dans la découverte des instructions de ce fabuleux microcontrôleur qu'est le AT90S8515 ATMEL.

◆ ***M. D.***

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 8

## Une carte de test pour microcontrôleur AT90S8515

**Pour essayer les programmes que nous allons écrire pour l'ATMEL AT90S8515, nous avons conçu une carte de test très simple par son circuit mais suffisante pour soulever les problèmes rencontrés dans la programmation du microcontrôleur et y apporter des solutions.**

Les programmes réalisés pour piloter les dispositifs présents sur la carte de test ont été d'abord écrits en langage Assembleur puis, dans un second temps, en langage Basic, beaucoup plus facile à comprendre et plus facile à manier si les programmes sont particulièrement complexes.

Cette carte de test prévoit la programmation "in-circuit", par conséquent il n'est pas nécessaire de retirer le microcontrôleur de la platine chaque fois que nous devons le programmer.

Le logiciel utilisé pour interfacer le PC à la carte de test est le ATMEL AVR ISP que l'on peut télécharger gratuitement sur le site de la marque.

### Le schéma électrique

Si nous regardons le schéma électrique de la figure 1, nous voyons :

- 1 afficheur 7 segments à cathodes communes (DISPLAY 1),
- 1 afficheur LCD de 2 lignes de 16 caractères (DISPLAY 2),
- 1 buzzer (BZ1),
- 1 interface sérielle (U4) pour relier la carte de test à n'importe quel périphérique sériel,
- 1 interface de programmation "in-circuit" (U3) à relier au port parallèle du PC,
- 1 étage de paramétrage constitué de quatre micro-interrupteurs (DS1).
- 3 poussoirs (P1, P2, P3) dont un (P1) servant à remettre à zéro ou réinitialiser le système, ("to reset" en anglais),
- 1 LED (LD1).

### Les différents composants

- DISPLAY 1 – Commençons par décrire l'interfaçage à l'afficheur 7 segments. Celui utilisé est de type cathodes communes. Il est connecté aux ports C (PC0 à PC7) du microcontrôleur U2, à travers des résistances de limitation du courant consommé par ses LED. Il est possible d'écrire un programme allumant séquentiellement les segments, ou alors, visualisant des suites de nombres. Si l'on veut quelque chose de plus interactif, en se servant des poussoirs P2 et P3, toujours en écrivant un programme adapté, il est possible, par exemple, d'exécuter des comptages en avant ou en arrière selon la pression sur les touches de contrôle.

- DISPLAY 2 – L'afficheur LCD, de type 2 lignes de 16 caractères, relié en partie au port A (bus de données) et en partie au port D (contrôles), est bien plus commode. Il permet de réaliser beaucoup plus d'applications, surtout si le logiciel est écrit en Basic. Pour sa gestion, il est nécessaire d'utiliser les signaux de contrôle : RS, E et R/W.

- RS sert à distinguer l'envoi d'une instruction de celui d'une donnée ;
- E est "Enable" (habilitation) : actif au niveau logique haut.
- R/W "Read/Write" (lire/écrire) est relié directement à la masse car nous voulons toujours écrire et non pas lire la mémoire de l'afficheur LCD.

En dehors de ces trois broches, nous avons un bus de données à 8 bits, nécessaire à l'envoi des instructions comme des données mais, en cas de nécessité, on peut

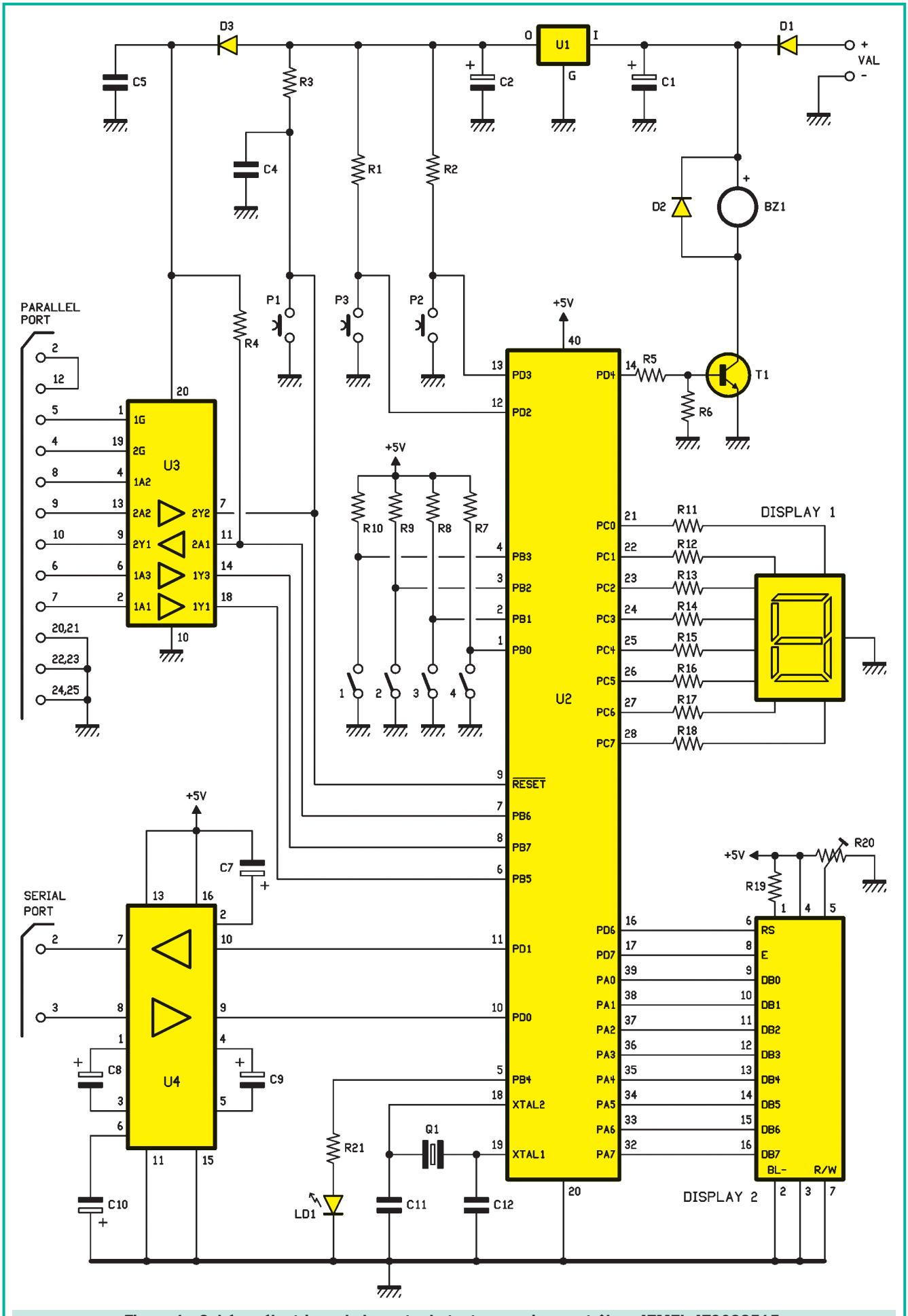



Figure 1 : Schéma électrique de la carte de test pour microcontrôleur ATMEL AT90S8515.

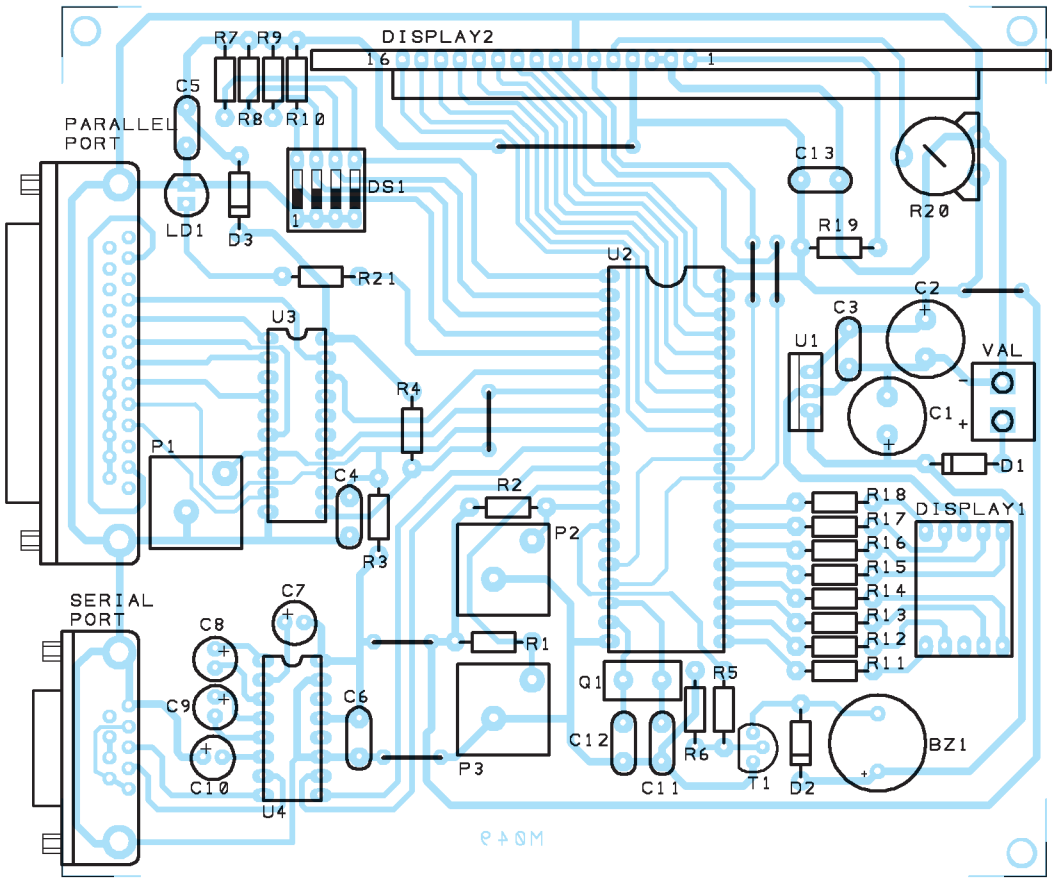



**Figure 2 : Schéma d'implantation des composants de la carte de test.**

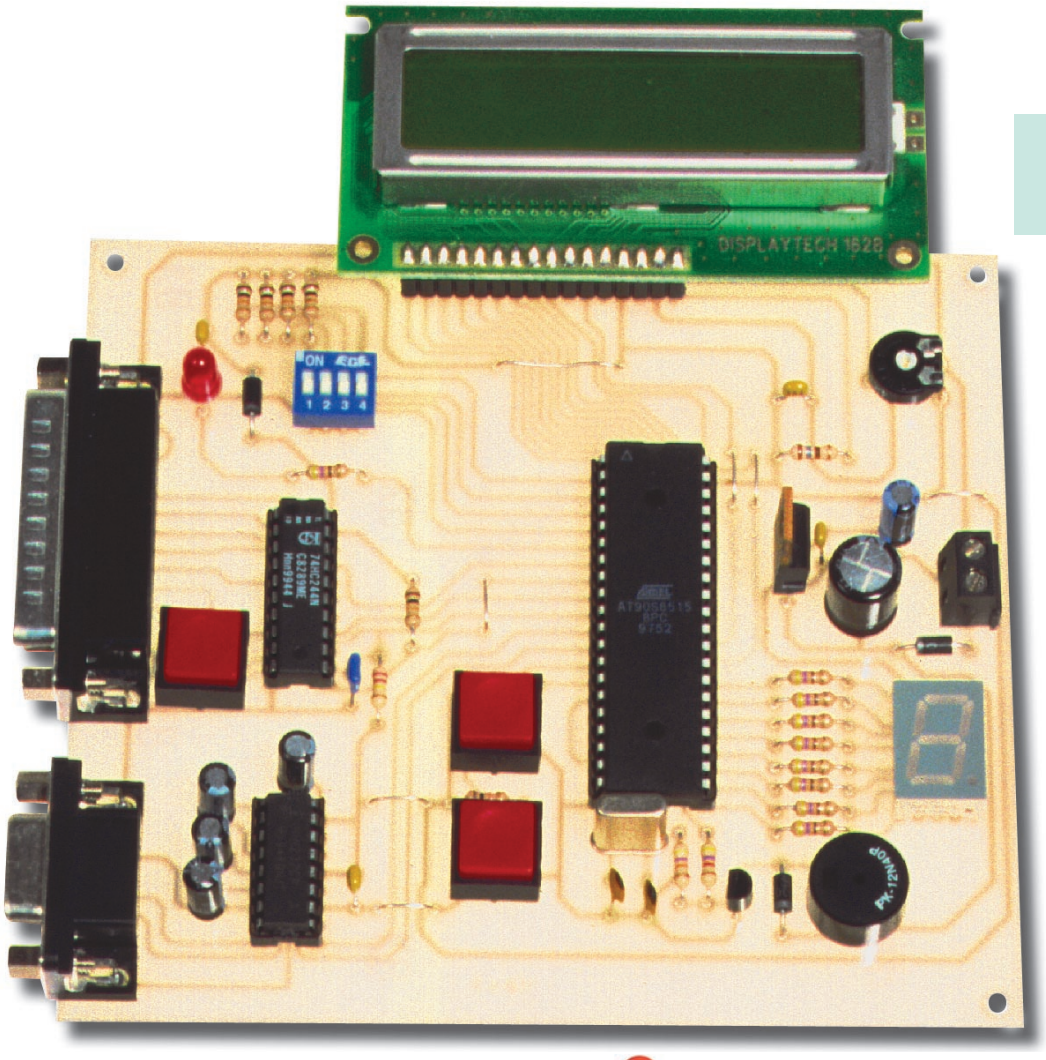



**Figure 3 : Photo d'un des prototypes de la carte de test avec l'afficheur en place.**

## Liste des composants

R1 = 10 kΩ
R2 = 10 kΩ
R3 = 220 kΩ
R4 = 10 kΩ
R5 = 4,7 kΩ
R6 = 47 kΩ
R7 = 10 kΩ
R8 = 10 kΩ
R9 = 10 kΩ
R10 = 10 kΩ
R11 = 470 Ω
R12 = 470 Ω
R13 = 470 Ω
R14 = 470 Ω
R15 = 470 Ω
R16 = 470 Ω
R17 = 470 Ω
R18 = 470 Ω
R19 = 180 Ω
R20 = 4,7 kΩ trimmer
R21 = 470 Ω
C1 = 470 µF 25 V
C2 = 100 µF 25 V
C3 = 100 nF multicouche
C4 = 100 nF multicouche
C5 = 100 nF multicouche
C6 = 100 nF multicouche
C7 = 1 µF 25 V
C8 = 1 µF 25 V
C9 = 1 µF 25 V
C10 = 1 µF 25 V
C11 = 22 pF céramique
C12 = 22 pF céramique
C13 = 100 nF multicouche
D1 = Diode 1N4007
D2 = Diode 1N4007
D3 = Diode 1N4007
LD1 = LED rouge 5 mm
U1 = Régulateur 7805
U2 = µC ATMEL AT90S8515
U3 = Intégré 74HC244
U4 = Intégré MAX232
Q1 = Quartz 4 MHz
T1 = NPN BC547
BZ1 = Buzzer sans élec.
P1 = Poussoir N.O.
P2 = Poussoir N.O.
P3 = Poussoir N.O.
DS1 = Dip switchs 4 micro-inter.
DISPLAY 1 = Afficheur 7 segments c.c.
DISPLAY 2 = Afficheur LCD CDL4162
2 lignes de 16 caractères

Divers :
1 Support 2 x 8 broches
1 Support 2 x 10 broches
1 Support 2 x 20 broches
1 Connecteur SUB-D 9 broches femelle à 90° pour c.i.
1 Connecteur SUB-D 25 broches mâle à 90° pour c.i.
1 Bornier 2 pôles pas de 5 mm
1 Connecteur sécable 16 broches femelle
1 Connecteur sécable 16 broches mâle
4 Entretoises adhésives ou
4 Pieds adhésifs caoutchouc
1 Câble parallèle 25 broches M/F
1 Câble série 9 broches M/F

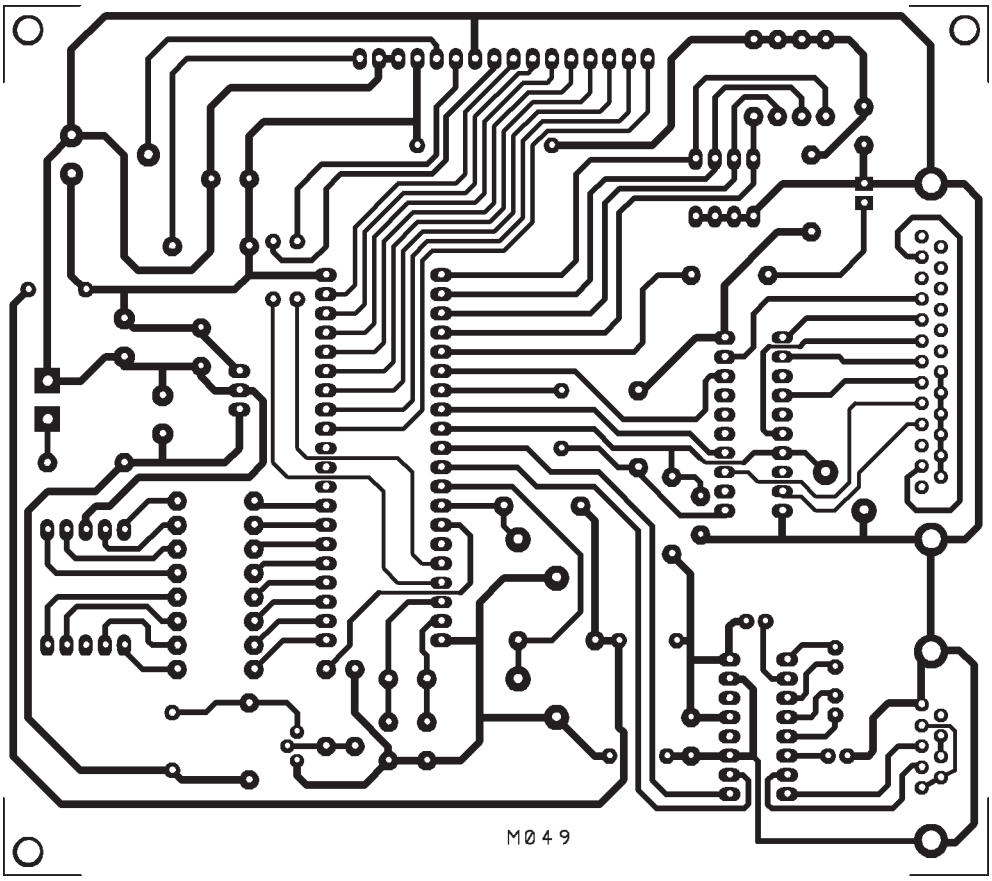
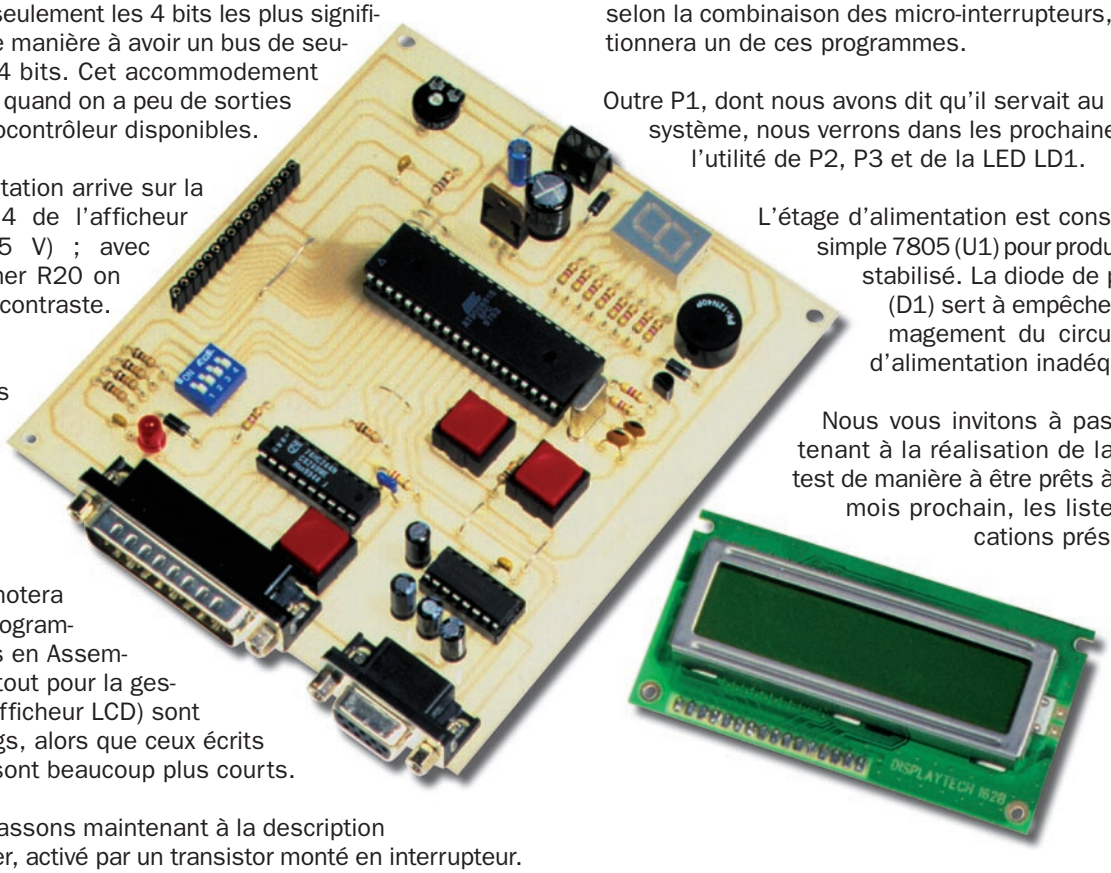



**Figure 4 : Dessin, à l'échelle 1, du circuit imprimé de la carte de test. Il pourra être réalisé par la méthode décrite dans le numéro 26 d'ELM.**

utiliser seulement les 4 bits les plus significatifs de manière à avoir un bus de seulement 4 bits. Cet accommodement s'utilise quand on a peu de sorties du microcontrôleur disponibles.

L'alimentation arrive sur la broche 4 de l'afficheur LCD (+5 V) ; avec le trimmer R20 on règle le contraste.

Dans les prochaines leçons, quand nous présenterons les listes d'applications, on notera que les programmes écrits en Assembleur (surtout pour la gestion de l'afficheur LCD) sont assez longs, alors que ceux écrits en Basic sont beaucoup plus courts.



- BZ1 – Passons maintenant à la description du buzzer, activé par un transistor monté en interrupteur. Le transistor est piloté par le port D4, quand il est au niveau logique haut (1) T1 est ON et, par conséquent, il ferme le circuit auquel est relié le buzzer émettant des sons.

- U4 – Outre le buzzer, nous avons évoqué une interface sérielle pour interfacer le microcontrôleur AT90S8515 avec le monde extérieur. La connexion se fait grâce au fameux MAX232, nécessaire à la conversion les seuils de valeurs logiques provenant, par exemple, du PC, à une valeur compatible avec les niveaux du AT90S8515. Le programme de gestion de l'interface sérielle, dans ce cas, sera réalisé seulement en Basic, étant donnée sa complexité. Les broches du microcontrôleur impliquées dans cette connexion sont la PD1 et la PD0. Le programme d'interface prévoit d'acquérir des données par le port sériel et de visualiser le tout sur l'afficheur LCD.

- U3 – Ce circuit intégré, un 74HC244, sert à programmer le microcontrôleur "in-system". Il s'agit d'un simple "buffer" à trois étages relié au port parallèle du PC. Rappelons que, lorsque ces circuits intégrés sont actifs, ils se comportent comme de simples "buffers" alors que, quand leurs broches de contrôle ne sont pas actives, leurs sorties sont à haute impédance. Pour la programmation on utilise les broches PB5, PB6 et PB7, correspondant respectivement à MOSI (IN), MISO (OUT) et SCK.

- DS1 – Notons enfin la présence de quatre micro-interrupteurs reliés au port B, précisément à PB0, PB1, PB2 et PB3. Ceux-ci peuvent servir à sélectionner 16 possibilités de modes de fonctionnement pour une hypothétique application. Dans notre cas, ils sont utilisés pour sélectionner une des applications écrites, pour montrer les fonctionnalités du circuit. En effet, les programmes écrits ont été regroupés en un seul macroprogramme et, selon la combinaison des micro-interrupteurs, on sélectionnera un de ces programmes.

Outre P1, dont nous avons dit qu'il servait au RESET du système, nous verrons dans les prochaines leçons, l'utilité de P2, P3 et de la LED LD1.

L'étage d'alimentation est constitué d'un simple 7805 (U1) pour produire le +5 V stabilisé. La diode de protection (D1) sert à empêcher l'endommagement du circuit en cas d'alimentation inadéquate.

Nous vous invitons à passer maintenant à la réalisation de la carte de test de manière à être prêts à suivre, le mois prochain, les listes d'applications présentées.




**Figure 5 : La carte de test complète avant la mise en place de son afficheur à cristaux liquides.**

## La réalisation

Les figures 2, 3 et 4 vous parleront bien plus qu'un long verbiage (j'aime cette expression, elle me fait toujours penser à ce fameux renard qui faisait la conversation à un stupide corbeau à propos d'un camembert !).

Il vous faut vous procurer ou réaliser le circuit imprimé de la figure 4. Une fois que vous l'avez gravé et percé, insérez et soudez tous les composants en partant des plus bas pour aller vers les plus hauts (c'est un principe immuable !). Comme toujours, veillez à la polarité des composants polarisés en vous référant à la figure 2. Une fois toutes les soudures vérifiées et un dernier coup d'œil de sécurité, vous êtes prêts pour la prochaine leçon.

***M. D.*** ***A suivre***

## Coût de la réalisation*

Tous les composants visibles figure 2, pour réaliser cette carte de test pour microcontrôleur ATMEL AT90S8515 EM.049, y compris le circuit imprimé percé et sérigraphié : 105,00 €.

Le circuit imprimé seul : 20,00 €.

Le kit de développement (Starter Kit) STK500 : 190,55 €.

*Les coûts sont indicatifs et n'ont pour but que de donner une échelle de valeur au lecteur. La revue ne fournit ni circuit ni composant. Voir les publicités des annonceurs.

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 9

**Bien que les langages de haut niveau (C, Basic, etc.) soient beaucoup plus simples et d'apprentissage immédiat (ou intuitifs), dans certains cas, il est nécessaire de dialoguer "à contact direct" avec le microcontrôleur. L'unique langage de programmation nous permettant d'agir directement sur le matériel du microcontrôleur (registres, mémoire, ports des I/O...) est l'Assembleur (ou langage machine) : il permet de tirer le maximum de profit du microcontrôleur du point de vue des possibilités opérationnelles comme de celui de la vitesse.**

Lorsqu'on travaille en Assembleur, les prestations et l'exhaustivité de l'exploitation de la machine s'améliorent mais, corollaire, la difficulté de rédaction du code augmente sensiblement. Nous verrons, dans cette leçon, comment réaliser un programme en Assembleur en analysant une liste (ou séquence de programme) de démonstration simple (voir figure 4) fort utile pour donner une explication élémentaire.

Il est avant tout nécessaire de disposer du logiciel adéquat: pour la programmation en langage Assembleur, sur plateforme Windows, on trouve sur le site ATMEL (www.atmel.com, voir "Sur l'Internet" dans ELM 36), le programme WAVRASM. Pour le télécharger, à partir de la page d'accueil du site ATMEL, sélectionner, en haut à gauche le mot "PRODUCTS", puis "AVR 8-BIT RISC", puis "SOFTWARE". Dans cette page est disponible l'outil ("tool") "ASMPACK.EXE" qu'on installera ensuite sur son PC. Vous pouvez simplifier en téléchargeant depuis le site de la revue (rubrique téléchargement et WAVRASM). Quand cela est fait, en sélectionnant dans "PROGRAMMES" le mot "AVR Assembleur", apparaît l'écran de la figure 1. Il faut alors décider si l'on écrit un nouveau programme ou si on en ouvre un déjà existant pour lui apporter des modifications.

Si l'on choisit un nouveau fichier ("file") on ouvrira simplement une fenêtre de texte vide (figure 2a). Dans le cas où l'on veut modifier ou repartir d'un fichier existant, apparaîtra à l'écran une fenêtre contenant la séquence de programme à modifier (figure 2b).

Un programme écrit en Assembleur est un fichier texte. Il peut prendre n'importe quel nom mais il doit être caractérisé par l'extension ASM. Un tel programme est constitué d'une série de phrases, définies comme "statements" (déclarations), dont chacune peut représenter une série d'informations.

Les divers types de déclarations comprennent:

- les étiquettes ("labels"): elles sont utilisées, en cas de sauts conditionnels ou inconditionnels, pour indiquer un point de la liste (séquence de programme); ou bien pour donner un nom facilement mémorisable à des variables ou des constantes;

- le code opérationnel: représente les instructions que le microcontrôleur est capable d'exécuter;

- les opérants: c'est-à-dire les éléments (registres et localisation de mémoire, "location" en anglais) vers et sur lesquels les instructions doivent aller et agir;

- les commentaires: c'est-à-dire des indications utiles au programmeur pour comprendre à quoi sert la séquence de code commentée;

- les directives: ce sont des instructions (commandes) adressées au compilateur ou bien au programme qui transformera notre fichier .ASM (fichier de texte source) en un fichier .HEX (fichier objet à "insérer" dans le mémoire programme du microcontrôleur).


**Figure 1 : Le programme sous Windows AVRASM (WAVRASM) peut être gratuitement chargé sur le site www.atmel.com : suivez les liens Microcontroller, AVR, Software (logiciel), Tools (outils). Bien entendu, vous pourrez également télécharger ce programme directement depuis la rubrique idoine de notre site.**

Les directives les plus importantes sont :

- CSEG : indique le début du code programme ;
- DEF : attribue un nom et un registre ;
- DSEG : indique le début d'une aire de données ;
- EQU : attribue une valeur à une étiquette ;
- ESEG : indique le début d'un espace de mémoire de type EEPROM ;
- INCLUDE : sert à inclure un fichier Assembleur à l'intérieur d'un fichier Assembleur.

Pour comprendre comment écrire en code Assembleur pour le microcontrôleur AT90S8515, et plus généralement pour n'importe quel type de microcontrôleur AVR, nous examinerons la séquence de programme de la figure 4. La première ligne du programme (.include "8515def.inc") sert à "informer" le compilateur sur le dispositif matériel utilisé, dans ce cas le AT90S8515. La séquence "8515def.inc" identifie un fichier dans lequel sont contenues toutes les adresses physiques des registres, des ports des I/O et des dispositifs matériels (comme le Watch Dog) présents à l'intérieur du AT90S8515.

Après avoir sélectionné le dispositif matériel, il faut définir les vecteurs d'interruption ("interrupt"), déjà décrits dans la seconde leçon du cours. Les vecteurs d'interruption pour le microcontrôleur AT90S8515 sont au nombre de 13 et sont disponibles dans les localisations ("location") de mémoire programme allant de $000 à $00C.


**Figure 2a : Fenêtre de texte vide.**

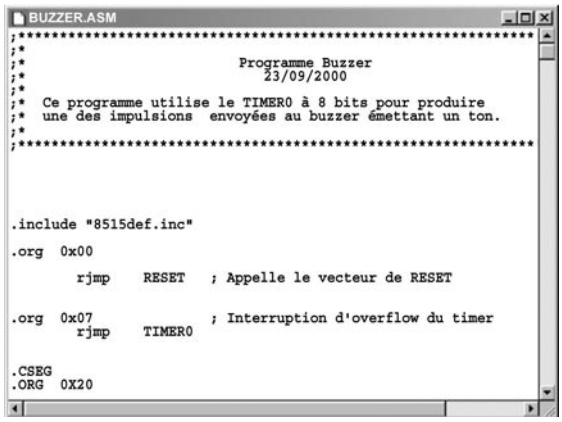



**Figure 2b : Séquence de programme à modifier.**

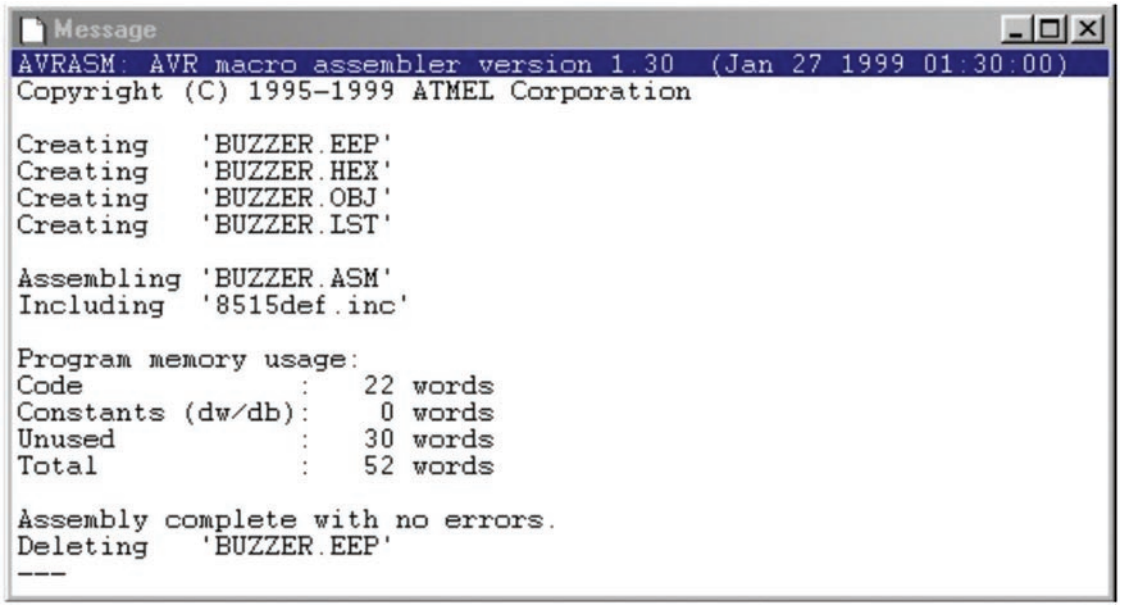



**Figure 3 : Programme WAVRASM sous Windows : il permet de programmer les AVR en Assembleur.**

```
.include  "8515def.inc"

.org 0x00

      rjmp  RESET   ; Appelle le vecteur
                    ; de RESET

.org 0x07           ; Interruption
                    ; d'overflow du timer

      rjmp  TIMER0

.CSEG
.ORG 0X20

.def  sortie=r18
.equ  sortie=0xff

TIMER0:

      ldi   r16,0xff
      out   DDRD,r16

      com   r18
      andi  r18,0b00010000
      out   PORTD,r18

      ldi   r17,0xB9
      out   TCNT0,r17
      reti

RESET:

      ldi   r16,high(RAMEND)
      out   SPH,r16
      ldi   r16,low(RAMEND)
      out   SPL,r16

      ......................
      ......................
      ......................
```

*La liste (séquence de programme) publiée ici ne représente qu'un simple programme de démonstration aidant à la compréhension des bases et de la structure d'un programme écrit en Assembleur pour le microcontrôleur ATMEL AVR AT90S8515.*

**Figure 4 : Programme de démonstration.**

En particulier, quand on alimente le circuit, ou bien après une impulsion de réinitialisation ("reset"), le microcontrôleur commence à travailler en exécutant l'instruction se trouvant à la localisation $000; dans cette localisation nous devrons insérer une instruction de saut inconditionnel au début du programme. Dans notre exemple, nous utilisons la directive .ORG 0x00 pour écrire dans la localisation $000 l'instruction rjmp RESET. On remarque que la localisation de mémoire vers laquelle on se dirige est définie par une étiquette, cela signifie que nous pouvons positionner la routine pour l'interruption de RESET là où nous voulons dans la mémoire, pourvu que ce soit après les 13 premières localisations de mémoire (de 0x00 à 0x0C) servant à identifier justement les 13 appels possibles d'interruption.

Dans notre exemple, nous supposons d'utiliser aussi l'interruption du TIMER: le vecteur d'interruption de ce dernier correspond à la localisation $007. Dans ce cas aussi, nous devrons forcer dans cette localisation (.org 0x07) une instruction de saut inconditionnel (rjmp TIMER0) à une routine de réponse à l'interruption. La routine de réponse sera précédée par l'étiquette utilisée dans l'instruction de saut (TIMER0:) et terminera avec l'instruction RETI.

La directive .CSEG suivies de la .ORG informent le compilateur du début du code proprement dit de notre programme et que ce code sera écrit dans la mémoire programme du microcontrôleur en partant de la localisation $020 (.ORG 0x20).

Il est normal de commencer avec la routine de réponse à l'interruption. Dans notre exemple, nous écrivons la routine de réponse au TIMER et nous la terminons avec l'instruction RETI. Comme nous n'avons pas inséré d'autre interruption, nous reportons ensuite l'étiquette par laquelle le microcontrôleur commence à traiter les instructions après un RESET ou quand il vient d'être alimenté. Pratiquement, après l'étiquette ("label") RESET: les instructions appartenant au programme principal ("main program") commencent.

Les premières instructions reportées servent à indiquer au compilateur la présence d'une SRAM interne au microcontrôleur de 512 octets ("Byte" en anglais). Outre celles décrites jusqu'à présent, il y a d'autres directives utiles (comme la DSEG et la ESEG) dans la rédaction d'un programme.

La DSEG informe le compilateur sur le début d'une aire de données. Par exemple:

```
.DSEG          ; Commence l'aire de données
var1: .BYTE 1  ; Réserve un byte
               ; à la variable var1
```

La ESEG définit le début de l'espace pour la mémoire EEPROM. Par exemple:

```
.ESEG
eevar: .DW 0x00ff
```

Quand tout le code a été écrit dans la fenêtre du programme WAVRASM, il est nécessaire de créer le fichier .HEX devant être chargé dans la mémoire du microcontrôleur. Pour ce faire, il suffit de compiler le tout en sélectionnant ASSEMBLE.

S'il y a des erreurs dans le programme source, le compilateur produira automatiquement un fichier de texte avec listage des problèmes rencontrés. Dans le cas contraire, un fichier de texte semblable à celui représenté sur la figure 3 sera produit: l'assembleur crée le fichier temporaire .EPP dont il tire trois fichiers distincts (le .HEX, le .OBJ et le .LST).

Le fichier avec extension .HEX sera chargé dans la mémoire Flash du microcontrôleur utilisant le programme ATMEL AVR ISP (lui aussi disponible sur le site ATMEL).

***M. D.***

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 10

**Le but de ce cours est de vous apprendre à programmer les microcontrôleurs Flash de la famille ATMEL AVR, en nous servant d'une carte de test simple mais complète, telle que celle que nous avons décrite dans la leçon 8. Avec son programmateur "in-circuit", nous apprendrons à utiliser des périphériques comme un afficheur à 7 segments, des poussoirs, des lignes sérielles, un buzzer et un afficheur LCD. Les listings de démo que nous illustrerons petit à petit, seront tout d'abord rédigés en classique langage Assembleur puis en Basic, plus simple et plus intuitif.**

Après avoir vu quelles sont les bases pour pouvoir écrire un programme en Assembleur pour les microcontrôleurs ATMEL AVR, voyons maintenant comment réaliser des programmes simples utilisant les potentialités de la platine de démonstration. Avant de commencer la description du fonctionnement du premier listing, nous faisons un pas en arrière et, pour éclairer un peu mieux la signification de l'instruction : .include "8515def.inc", nous donnons la totalité du fichier ("file") des définitions relatives au microcontrôleur ATMEL AVR AT90S8515 (figure 1).

## Le programme "Buzzer"

Ceci dit, passons à la description du premier programme présenté : "Buzzer", permettant de piloter la sortie PortD.4 de façon à faire émettre un son au buzzer connecté. Tout d'abord, il est nécessaire de paramétrer les vecteurs d'interruption ("interrupt"). Nous avons besoin, en particulier, du vecteur de "RESET" (présent dans toutes les applications) et du vecteur TIMER0.

Le premier vecteur se trouve à l'adresse 0x00 et le second à l'adresse 0x07 : à chacun correspond une instruction de saut inconditionné à la routine de gestion de l'interruption ("interrupt"). Tout d'abord (et cela arrive à chaque mise en marche du microcontrôleur) est exécuté le vecteur de RESET lequel, entre autres choses, s'occupe d'habiliter les 512 octets de SRAM disponibles. Cette procédure est exécutée à partir des quatre instructions se trouvant après l'étiquette RESET.

Quand cette opération a été exécutée, il est nécessaire d'habiliter les interruptions et cela se fait à partir de l'instruction "sei". Une fois les interruptions habilitées, il faut programmer le registre TIMSK et en particulier le bit TOEI0, s'occupant de l'habilitation du vecteur de dépassement de capacité ("overflow") du "Timer/Counter0" et de programmer le registre TCCR0 de manière à travailler sur les fronts de montée du signal. Enfin, une valeur de seuil pour le comptage est chargée dans le registre TCCR0.

Après cette phase de programmation, on entre dans une boucle infinie formée des deux dernières instructions, dans l'attente d'une interruption lançant l'exécution de la routine TIMER0. Dans la routine TIMER0, le port D est habilité comme sortie, on fait le complément à un du registre R18, on le multiplie bit par bit par une valeur constante et on l'envoie en sortie au PortD.4, pilotant ainsi le transistor T1.

Après quoi, il est nécessaire de remettre à jour le registre TCNT0, ce qui conclut la routine avec une instruction de retour par interruption. Ceci provoque l'émission, par le buzzer, d'un ton acoustique (figures 2 et 3).

```
;***************************************************************************************
;* A P P L I C A T I O N    N O T E    F O R    T H E    A V R    F A M I L Y
;*
;* Number: AVR000
;* File Name: "8515def.inc"
;* Title: Register/Bit Definitions for the AT90S8515
;* Date: 99.01.28
;* Version: 1.30
;* Support telephone   : +47 72 88 43 88 (ATMEL Norway)
;* Support fax : +47 72 88 43 99 (ATMEL Norway)
;* Support E-mail      : avr@atmel.com
;* Target MCU: AT90S8515
;*
;* DESCRIPTION
;* When including this file in the assembly program file, all I/O register
;* names and I/O register bit names appearing in the data book can be used.
;* In addition, the six registers forming the three data pointers X, Y and
;* Z have been assigned names XL - ZH. Highest RAM address for Internal
;* SRAM is also defined
;*
;* The Register names are represented by their hexadecimal address.
;*
;* The Register Bit names are represented by their bit number (0-7).
;*
;* Please observe the difference in using the bit names with instructions
;* such as "sbr"/"cbr" (set/clear bit in register) and "sbrs"/"sbrc"
;* (skip if bit in register set/cleared). The following example illustrates
;* this :
;*
;* inr16,PORTB ;read PORTB latch
;* sbrr16,(1<<PB6)+(1<<PB5)     ;set PB6 and PB5 (use masks, not bit#)
;* out PORTB,r16       ;output to PORTB
;*
;* in r16,TIFR  ;read the Timer Interrupt Flag Register
;* sbrcr16,TOV0;test the overflow flag (use bit#)
;* rjmpTOV0_is_set     ;jump if set
;* ...          ;otherwise do something else

;***************************************************************************************



;***** Specify Device
.device AT90S8515

;***** I/O Register Definitions
.equ      SREG  =$3f
.equ       SPH  =$3e
.equ       SPL  =$3d
.equ     GIMSK  =$3b
.equ      GIFR  =$3a
.equ     TIMSK  =$39
.equ      TIFR  =$38

.equ     MCUCR  =$35

.equ     TCCR0  =$33
.equ     TCNT0  =$32
.equ      OCR0  =$31

.equ    TCCR1A  =$2f
.equ    TCCR1B  =$2e
.equ    TCNT1H  =$2d
.equ    TCNT1L  =$2c

.equ    OCR1AH  =$2b
.equ    OCR1AL  =$2a
.equ    OCR1BH  =$29
.equ    OCR1BL  =$28
.equ     ICR1H  =$25
.equ     ICR1L  =$24

.equ     WDTCR  =$21
.equ     EEARH  =$1f
.equ     EEARL  =$1e


.equ      EEDR  =$1d
.equ      EECR  =$1c

.equ     PORTA  =$1b
.equ      DDRA  =$1a
.equ      PINA  =$19
.equ     PORTB  =$18
.equ      DDRB  =$17
.equ      PINB  =$16
.equ     PORTC  =$15
.equ      DDRC  =$14
.equ      PINC  =$13
.equ     PORTD  =$12
.equ      DDRD  =$11
.equ      PIND  =$10

.equ      SPDR  =$0f
.equ      SPSR  =$0e
.equ      SPCR  =$0d
.equ       UDR  =$0c
.equ       USR  =$0b
.equ       UCR  =$0a
.equ      UBRR  =$09
.equ      ACSR  =$08


;***** Bit Definitions
.equ      INT1  =7
.equ      INT0  =6

.equ     INTF1  =7
.equ     INTF0  =6

.equ     TOIE1  =7
.equ    OCIE1A  =6
.equ    OCIE1B  =5
.equ    TICIE1  =3
.equ     TOIE0  =1

.equ      TOV1  =7
.equ     OCF1A  =6
.equ     OCF1B  =5
.equ      ICF1  =3
.equ      TOV0  =1

.equ       SRE  =7
.equ       SRW  =6
.equ        SE  =5
.equ        SM  =4
.equ     ISC11  =3
.equ     ISC10  =2
.equ     ISC01  =1
.equ     ISC00  =0

.equ      CS02  =2
.equ      CS01  =1
.equ      CS00  =0

.equ    COM1A1  =7
.equ    COM1A0  =6
.equ    COM1B1  =5
.equ    COM1B0  =4
.equ     PWM11  =1
.equ     PWM10  =0

.equ     ICNC1  =7
.equ     ICES1  =6
.equ      CTC1  =3
.equ      CS12  =2
.equ      CS11  =1
.equ      CS10  =0
```

```
.equ      WDDE  =4
.equ       WDE  =3
.equ      WDP2  =2
.equ      WDP1  =1
.equ      WDP0  =0

.equ     EEMWE  =2
.equ      EEWE  =1

.equ      EERE  =0

.equ       PA7  =7
.equ       PA6  =6
.equ       PA5  =5
.equ       PA4  =4
.equ       PA3  =3
.equ       PA2  =2
.equ       PA1  =1
.equ       PA0  =0

.equ      DDA7  =7
.equ      DDA6  =6
.equ      DDA5  =5
.equ      DDA4  =4
.equ      DDA3  =3
.equ      DDA2  =2
.equ      DDA1  =1
.equ      DDA0  =0

.equ     PINA7  =7
.equ     PINA6  =6
.equ     PINA5  =5
.equ     PINA4  =4
.equ     PINA3  =3
.equ     PINA2  =2
.equ     PINA1  =1
.equ     PINA0  =0

.equ       PB7  =7
.equ       PB6  =6
.equ       PB5  =5
.equ       PB4  =4
.equ       PB3  =3
.equ       PB2  =2
.equ       PB1  =1
.equ       PB0  =0

.equ      DDB7  =7
.equ      DDB6  =6
.equ      DDB5  =5
.equ      DDB4  =4
.equ      DDB3  =3
.equ      DDB2  =2
.equ      DDB1  =1
.equ      DDB0  =0

.equ     PINB7  =7
.equ     PINB6  =6
.equ     PINB5  =5

.equ     PINB4  =4
.equ     PINB3  =3
.equ     PINB2  =2
.equ     PINB1  =1
.equ     PINB0  =0

.equ       PC7  =7
.equ       PC6  =6
.equ       PC5  =5
.equ       PC4  =4
.equ       PC3  =3
.equ       PC2  =2
.equ       PC1  =1
.equ       PC0  =0
```

```
.equ      DDC7  =7
.equ      DDC6  =6
.equ      DDC5  =5
.equ      DDC4  =4
.equ      DDC3  =3
.equ      DDC2  =2
.equ      DDC1  =1
.equ      DDC0  =0

.equ     PINC7  =7
.equ     PINC6  =6
.equ     PINC5  =5
.equ     PINC4  =4
.equ     PINC3  =3
.equ     PINC2  =2
.equ     PINC1  =1
.equ     PINC0  =0

.equ       PD7  =7
.equ       PD6  =6
.equ       PD5  =5
.equ       PD4  =4
.equ       PD3  =3
.equ       PD2  =2
.equ       PD1  =1
.equ       PD0  =0

.equ      DDD7  =7
.equ      DDD6  =6
.equ      DDD5  =5
.equ      DDD4  =4

.equ      DDD3  =3
.equ      DDD2  =2
.equ      DDD1  =1
.equ      DDD0  =0

.equ     PIND7  =7
.equ     PIND6  =6
.equ     PIND5  =5
.equ     PIND4  =4
.equ     PIND3  =3
.equ     PIND2  =2
.equ     PIND1  =1
.equ     PIND0  =0
```

```
.equ      SPIE  =7
.equ       SPE  =6
.equ      DORD  =5
.equ      MSTR  =4
.equ      CPOL  =3
.equ      CPHA  =2
.equ      SPR1  =1
.equ      SPR0  =0

.equ      SPIF  =7
.equ      WCOL  =6

.equ       RXC  =7
.equ       TXC  =6
.equ      UDRE  =5
.equ        FE  =4
.equ        OR  =3

.equ     RXCIE  =7
.equ     TXCIE  =6
.equ     UDRIE  =5
.equ      RXEN  =4
.equ      TXEN  =3
.equ      CHR9  =2
.equ      RXB8  =1
.equ      TXB8  =0

.equ       ACD  =7
.equ       ACO  =5
.equ       ACI  =4
.equ      ACIE  =3
.equ      ACIC  =2

.equ     ACIS1  =1
.equ     ACIS0  =0

.def        XL  =r26
.def        XH  =r27
.def        YL  =r28
.def        YH  =r29
.def        ZL  =r30
.def        ZH  =r31
```

```
.equ    RAMEND  =$25F  ;Last On-Chip SRAM Location
.equ   XRAMEND  =$FFFF
.equ     E2END  =$1FF
.equ  FLASHEND  =$FFF


.equ  INT0addr  =$001  ;External Interrupt0 Vector Address
.equ  INT1addr  =$002  ;External Interrupt1 Vector Address
.equ  ICP1addr  =$003  ;Input Capture1 Interrupt Vector Address
.equ  OC1Aaddr  =$004  ;Output Compare1A Interrupt Vector Address
.equ  OC1Baddr  =$005  ;Output Compare1B Interrupt Vector Address
.equ  OVF1addr  =$006  ;Overflow1 Interrupt Vector Address
.equ  OVF0addr  =$007  ;Overflow0 Interrupt Vector Address
.equ   SPIaddr  =$008  ;SPI Interrupt Vector Address
.equ  URXCaddr  =$009  ;UART Receive Complete Interrupt Vector Address
.equ  UDREaddr  =$00a  ;UART Data Register Empty Interrupt Vector Address
.equ  UTXCaddr  =$00b  ;UART Transmit Complete Interrupt Vector Address
.equ   ACIaddr  =$00c  ;Analog Comparator Interrupt Vector Address
```

*Dans la leçon précédente, nous avions insisté sur la nécessité d'insérer, dans tous les programmes, la définition du type de processeur utilisé. Dans cette leçon nous donnons le fichier complet définissant tous les paramètres relatifs au processeur que nous avons utilisé : le AT90S8515. Ce fichier est fourni avec le système de développement et il est réclamé par le programme de l'usager par l'intermédiaire de l'instruction : .include "8515def.inc".*

**Figure 1 : Note d'application pour la programmation des microcontrôleurs de la famille AVR.**

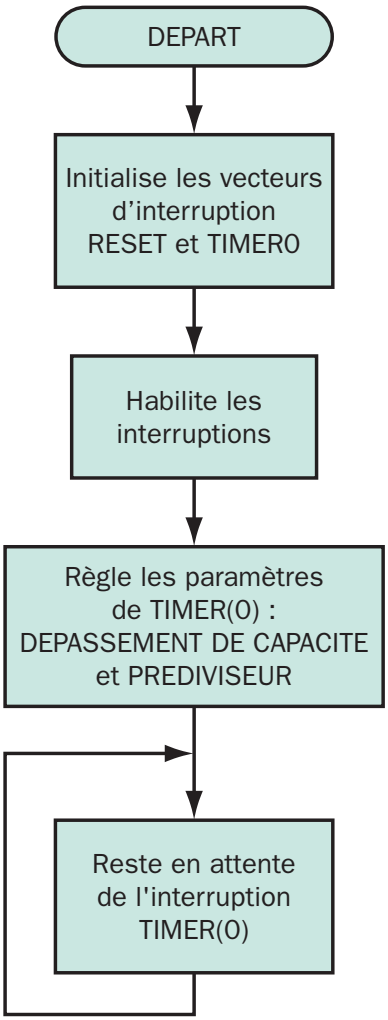



**Figure 2 : Organigramme du programme "Buzzer".**

## Le programme LED

Le deuxième programme que nous allons décrire s'occupe de faire clignoter la diode LED reliée au PortB.4. Comme dans le programme précédent, il est nécessaire d'initialiser le vecteur de Reset et par conséquent de paramétrer l'aire de mémoire SRAM de 512 octets. Une fois que le microcontrôleur est paramétré, on peut passer à l'exécution des instructions restantes.

Tout d'abord, il faut régler le port B comme sortie et pour ce faire il est nécessaire de charger la juste valeur dans le registre DDRB. Dans ce cas, nous nous limitons à habiliter comme sortie le seul Port B.4 et, par conséquent, nous devons charger en DDRB la valeur hexadécimale 0x10. Après avoir réglé le Port B, on peut allumer la LED en mettant au niveau logique haut (1) la broche PortB.4. Pour cela, il suffit de charger la valeur décimale 255 dans le registre PORTB. De cette façon la LED est allumée.

Maintenant, appelons une routine de retard de manière à maintenir la LED allumée pendant un certain temps. Une fois cette routine de retard terminée, nous mettrons au niveau logique bas (0) la broche PortB.4 et par conséquent

```
;*****************************************
;*  Programme Buzzer - Buzzer.asm
;*  27/05/2002
;*
;*  Ce programme utilise le timer0 à 8 bits
;*  pour produire une série d'impulsions
;*  envoyées au buzzer et émettant un ton.
;*
;*****************************************

.include "8515def.inc"

;Appelle le vecteur de RESET
.org 0x00
        rjmp    RESET

;Interruption de dépassement
;de capacité du timer
.org 0x07
        rjmp    TIMER0

.CSEG
.ORG 0X20
.def    sortie = r18
.equ    sortie = 0xff

TIMER0:
        ldi     r16,0xff
        out     DDRD,r16
        com     r18
        andi    r18, 0b00010000
out     PORTD, r18
        ldi     r17, 0xB9
        out     TCNT0, r17
        reti

RESET:
        ldi     r16,high(RAMEND)
        out     SPH,r16
        ldi     r16,low(RAMEND)
        out     SPL,r16

;Habilite le timer0 et les interruptions
        sei     ;Habilite les interruptions

;Ces instructions servent à habiliter
;l'interruption de dépassement de capacité
;du timer et pour régler le prédiviseur
;du timer0
        ldi     r17, 0b00000010
        out     TIMSK, r17
        ldi     r17, 0b00000011
        out     TCCR0, r17
        ldi     r17, 0xB9
        out     TCNT0, r17

forever:
        rjmp    forever
```

**Figure 3 : Programme "Buzzer".**

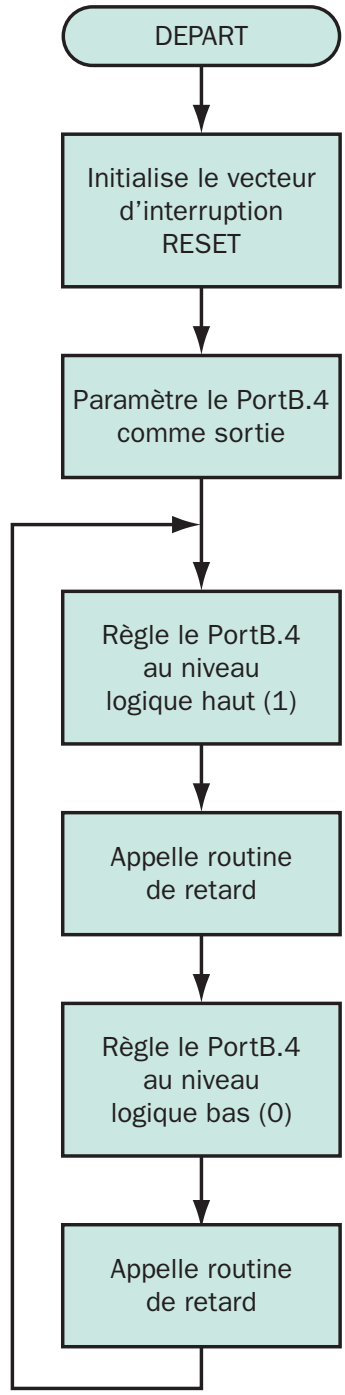



**Figure 4 : Organigramme du programme "LED clignotante".**

nous éteindrons la LED. La procédure est la même que la première fois, sauf que la valeur chargée dans le registre PORTB est zéro. Ces procédures sont répétées un nombre infini de fois. La routine de retard respecte l'organigramme visible figure 6.

Le fonctionnement est très simple : par commodité, on a donné un nom aux registres utilisés dans la routine, en particulier r18, r19, r20. A chacun est donnée une valeur décimale. Par conséquent, on commence à diminuer (décompter) une variable à chaque fois en faisant des tests de manière à détecter l'arrivée à zéro.

```
;****************************************
;*  Programme LED - LED.asm
;*  27/05/2002
;*
;*  Ce programme fait clignoter la LED
;*  relié au PortB.4
;*
;****************************************

.include "8515def.inc"

;Appelle le vecteur de RESET
.org 0x00
        rjmp    RESET

;Donne un nom à des registres
.def    premier         = r18
.def    second = r19
.def    troisième       = r20

RESET:
        ldi     r16,high(RAMEND)
        out     SPH,r16
        ldi     r16,low(RAMEND)
        out     SPL,r16

;Règle le portb.4 comme sortie
        ldi     r16,0x10
        out     DDRB,r16

;Envoie sur le Portb.4 la valeur logique
;haute (5 V), laquelle allume la LED connectée
        ldi     r16,255
        out     PORTB,r16
        rcall   Retard   ;Appelle retard

;Envoie sur le Portb.4 la valeur logique
;basse (0 V), laquelle éteint la LED reliée
        ldi     r16,0x00
        out     PORTB,r16
        rcall   Retard

Retard:
        ldi     premier,25
a:      ldi     second,255
b:      ldi     troisième,255
c:      dec     troisième
        brne    c
        dec     second
        brne    b
        dec     premier
        brne    a
        ret

forever:
        rjmp forever
```

**Figure 5 : Programme "LED clignotante".**

La routine en question exécute 25 * 255 * 255 itérations (en effet r18=25, r19=255, r20=255), c'est-à-dire 1625625 diminutions (comptages à rebours) avant d'être terminée. Ceci revient à dire que le temps nécessaire pour le déroulement de ces itérations est d'une demie seconde environ. Cela dépend, bien sûr, de la fréquence à laquelle on fait fonctionner le microcontrôleur. Dans notre cas, comme nous avons utilisé un quartz de 4 MHz, le temps nécessaire pour accomplir le cycle de retard est, justement, d'une demie seconde.

Il importe de noter que la routine de retard doit être appelée après chaque variation d'état de la LED et par conséquent après chaque allumage mais aussi après chaque extinction de la LED. Ainsi, la LED restera allumée pendant une demie seconde et éteinte également pendant la même durée.

## Conclusion et au revoir

Ces deux programmes très simples ayant été vus, nous vous renvoyons à prochaine leçon pour faire face à des situations plus complexes et fonctionnelles. Nous verrons comment gérer des périphériques de type afficheurs à 7 segments et l'afficheur LCD à deux lignes monté sur la platine de démonstration. A la prochaine!

***A suivre*** ◆◆◆

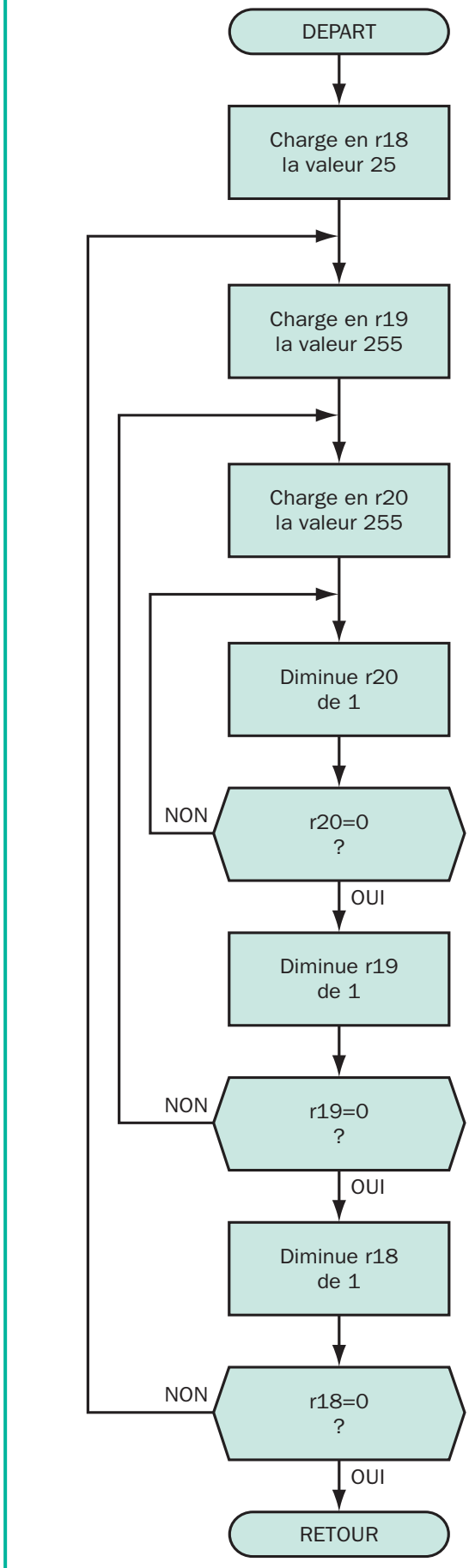



**Figure 4 : Organigramme du programme "LED clignotante".**

# Les microcontrôleurs Flash ATMEL AVR

## Leçon 11

**Comme prévu dans la leçon 10, nous allons maintenant analyser deux programmes, toujours en Assembleur, significativement plus complexes que ceux étudiés précédemment.**

**Le premier visualise sur afficheur à sept segments les nombres de 0 à 9 et les lettres de A à F, avec possibilité de faire défiler en avant ou en arrière la suite des nombres et des lettres en utilisant les poussoirs P2 et P3.**

**Le second programme s'occupe de la gestion d'un afficheur LCD à deux lignes de 16 caractères chacune.**

### Le programme afficheur à sept segments

Voyons tout de suite comment se comporte le premier programme en nous aidant de l'organigramme de la figure 1 et du listing de la figure 2. Comme d'habitude, le listing commence par la définition des vecteurs d'interruption ("interrupt") : dans ce cas, seul le vecteur de RESET est suffisant. La mémoire SRAM est initialisée et on paramètre la variable "UP" à zéro. A noter qu'au début du programme des noms ont été donnés à certains registres du microcontrôleur. Ainsi le programme devient plus lisible.

Après les paramètres initiaux, il faut régler le port C comme sortie et cela se fait sur le mode habituel, c'est-à-dire en chargeant dans le registre "DDRC" la valeur "0xFF" : les huit lignes du port sont paramétrées comme sorties. A la première mise sous tension, nous voulons visualiser sur l'afficheur la valeur zéro. Pour ce faire, il est nécessaire d'envoyer le code exact sur la sortie du port C. Dans ce cas, la valeur à charger dans le registre est "0x77". A partir de ce point, le microcontrôleur va tester la broche d'entrée du port D, en particulier Port D.3 et Port D.2, correspondant respectivement aux poussoirs P2 et P3. Le microcontrôleur lit la totalité du port D avec l'instruction "in" puis va vérifier si le bit 3 ou le bit 2 sont au niveau logique bas (0). Pour effectuer la vérification sur le bit 3, on utilise l'instruction "sbrs" laquelle vérifie si le bit à tester est au niveau logique haut (1) : si c'est le cas, il saute l'instruction suivante et teste ainsi l'autre bit correspondant à l'autre poussoir, soit le bit 2. Si le bit 3 est au niveau logique bas (0), la routine pour la touche P2 est appelée. L'instruction testant le bit 2 est "sbrc" laquelle vérifie si le bit 2 est au niveau logique bas (0) : auquel cas il saute l'instruction suivante et rappelle la routine pour la touche P3, sinon il saute l'étiquette une et entre ainsi dans une boucle infinie, interrompue seulement par une pression de P2 ou P3.

Nous pouvons maintenant décrire la sous-routine "P2". Comme on peut le deviner, cette routine correspond à la pression de la touche P2, lequel sert à augmenter la valeur visualisée sur l'afficheur à sept segments. De même, la routine "P3" se réfère à la touche P3 et permet de diminuer la valeur visualisée sur l'afficheur à sept segments. Tout d'abord, on charge la valeur 15 (0x0F) dans le registre "r18" et on exécute une comparaison avec "UP" : s'ils sont égaux, cela veut dire que le comptage est arrivé à 15 et, donc, on passe à la routine "égaux". Dans le cas contraire, on augmente "UP" et on va ensuite rappeler la routine de visualisation des caractères nommée "visualise".

La routine pour P3 est identique à celle que l'on vient de décrire, la seule différence étant que la variable "UP" est diminuée. La routine "égaux" met à zéro la variable "UP" alors que la routine "égaux2" met "UP" à 15. La routine "visualise" (qui, comme nous l'avons dit, s'occupe de gérer l'afficheur à sept segments) est très simple. En pratique, une comparaison est faite entre la variable "UP" et le registre "r18" qui peut contenir les

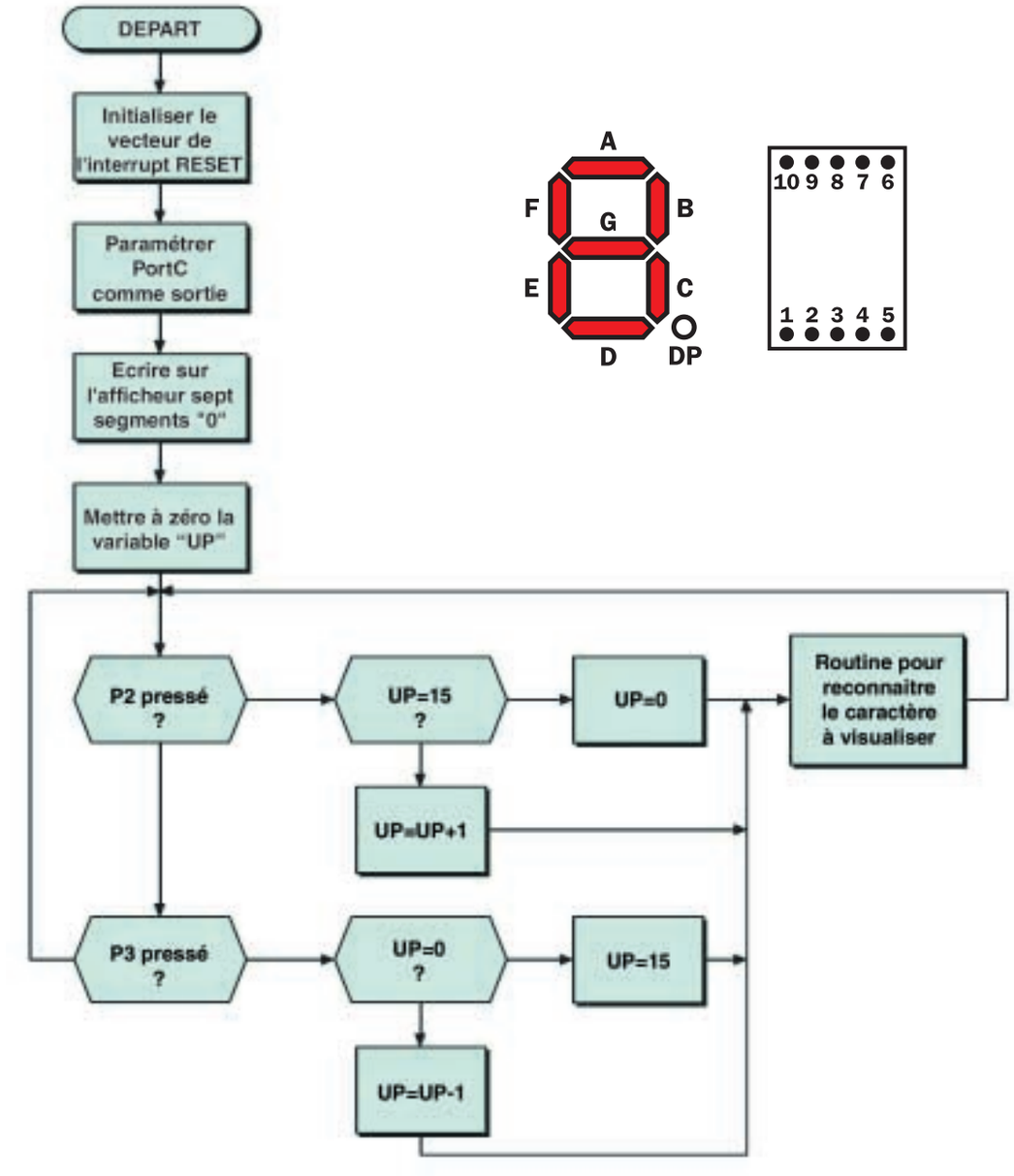



**Figure 1 : Organigramme de la gestion de l'afficheur à sept segments.**

valeurs de 0 à 9 ou les lettres de A à F. Quand elle trouve une correspondance entre les deux registres, elle visualise la valeur sur l'afficheur à sept segments. Pour chaque caractère visualisé, une routine de retard est rappelée, cette routine étant identique à celle utilisée dans le programme LED de la leçon 10.

## Le programme afficheur LCD

Le second listing, figure 4, s'occupe de gérer l'afficheur LCD dont la "demoboard" (platine d'essais) est également pourvue : la logique de contrôle d'un afficheur LCD dispose d'une mémoire de paramétrages des caractères nommée "CG RAM" et d'une mémoire de données nommée "DD RAM". Pour les afficheurs LCD à deux lignes, la mémoire "DD RAM" est située aux adresses allant de "00" à "0F" et de "40" à "4F", les caractères écrits dans ces 32 allocations de mémoire sont ceux effectivement visualisés sur l'afficheur LCD. Pour "écrire" des caractères sur l'afficheur LCD, il faut par conséquent écrire dans ces localisations les caractères l'un après l'autre, à chaque envoi d'un caractère le curseur se positionnant automatiquement sur la cellule suivante. Comme tous les programmes présentés jusqu'à maintenant, la section initiale prévoit la définition du vecteur de "RESET" et l'attribution d'un nom symbolique aux registres internes du microcontrôleur, de manière à avoir un code plus lisible. Dans notre application, nous associons des noms symboliques aux registres de contrôle de l'afficheur LCD. Pour informer l'afficheur LCD que les données arrivant sont des commandes, il faut

```
;*****************************************
;*  7SegUPDOWN.asm
;*  23/09/2001
;*
;*  Ce programme gère deux poussoirs
;*  pour piloter un afficheur à sept segments
;*
;*****************************************

.include "8515def.inc"

; Donne un nom aux registres
.def       UP         = r17
.def  premier  = r20
.def deuxième  = r21
.def troisème  = r22
.org     0x00
rjmp    RESET

RESET:
          ldi  r16,high(RAMEND)
          out  SPH,r16
          ldi  r16,low(RAMEND)
          out  SPL,r16

; Charge dans la variable "UP" la valeur zéro
héxadécimale
          ldi  UP,0x00

; Habilite le port  C comme sortie et envoie sur
l'afficheur à sept segments "0"
          ldi  r16,0xff
          out  DDRC,r16
          ldi  r16,0x77
          out  PORTC,r16

; Lis la porte D (broche  PD3 e PD2)
un :       in  r19,PIND

; Vérifie si le  Bit 3 du port D est réglé haut
         sbrs  r19,3
        rcall  P2 ; Appelle routine pour touche P2

; Vérifie si le Bit 2 du port D est réglé bas
         sbrc  r19,2
         rjmp  un
        rcall  P3; Appelle routine pour touche P3
         rjmp  un

; Routine pour  reconnaître si "UP" = 15
P2:
          ldi  "r18",0x0f
           cp  UP,"r18"
         breq  égaux1
          inc  UP
        rcall  visualise
          ret

égaux1:
          ldi  UP,0x00
        rcall  visualise
          ret

; Routine pour  reconnaître si "UP" = 0
P3:
          ldi  "r18",0x00
           cp  UP,"r18"
         breq  égaux2
          dec  UP
        rcall  visualise
          ret

égaux2:
          ldi  UP,0x0f
        rcall  visualise
          ret

; Routine de visualisation des caractères, selon
; la valeur de "UP" ira visualiser les nombres de
; 0 à 9 ou les lettres de A à F

visualise :
                ldi    "r18",0x00
                cpse   "r18", "UP"
                rjmp   a
                ldi    r16,0x77 ;0
                out    PORTC,r16
                rcall  retard
                ret
a:              ldi    "r18",0x01
                cpse   "r18", "UP"
                rjmp   b
                ldi    r16,0x14 ;1
                out    PORTC,r16
                rcall  retard
                ret
b:              ldi    "r18",0x02
                cpse   "r18", "UP"
                rjmp   c
                ldi    r16,0xB3 ;2
                out    PORTC,r16
                rcall  retard
                ret
c:              ldi    "r18",0x03
                cpse   "r18", "UP"
                rjmp   d
                ldi    r16,0xB6 ;3
                out    PORTC,r16
                rcall  retard
                ret
d:              ldi    "r18",0x04
                cpse   "r18", "UP"
                rjmp   e
                ldi    r16,0xD4 ;4
                out    PORTC,r16
                rcall  retard
                ret
e:              ldi    "r18",0x05
                cpse   "r18", "UP"
                rjmp   f
                ldi    r16,0xE6 ;5
                out    PORTC,r16
                rcall  retard
                ret
```

```
f:          ldi   “r18”,0x06
            cpse  “r18”, “UP”
            rjmp  g
            ldi   r16,0xC7 ;6
            out   PORTC,r16
            rcall retard
            ret
g:          ldi   “r18”,0x07
            cpse  “r18”, “UP”
            rjmp  h
            ldi   r16,0x34 ;7
            out   PORTC,r16
            rcall retard
            ret
h:          ldi   “r18”,0x08
            cpse  “r18”, “UP”
            rjmp  i
            ldi   r16,0xF7 ;8
            out   PORTC,r16
            rcall retard
            ret
i:          ldi   “r18”,0x09
            cpse  “r18”, “UP”
            rjmp  l
            ldi   r16,0xF4 ;9
            out   PORTC,r16
            rcall retard
            ret
l:          ldi   “r18”,0x0A
            cpse  “r18”, “UP”
            rjmp  m
            ldi   r16,0xF5 ;A
            out   PORTC,r16
            rcall retard
            ret
m:          ldi   “r18”,0x0B
            cpse  “r18”, “UP”
            rjmp  n
            ldi   r16,0xC7 ;B
            out   PORTC,r16
            rcall retard
            ret
n:          ldi   “r18”,0x0C
            cpse  “r18”, “UP”
            rjmp  o
            ldi   r16,0x63 ;C
            out   PORTC,r16
            rcall retard
            ret
o:          ldi   “r18”,0x0D
            cpse  “r18”, “UP”
            rjmp  p
            ldi   r16,0x97 ;D
            out   PORTC,r16
            rcall retard
            ret
p:          ldi   “r18”,0x0E
            cpse  “r18”, “UP”
            rjmp  q
            ldi   r16,0xE3 ;E
            out   PORTC,r16
            rcall retard
            ret
q:          ldi   r16,0xE1 ;F
            out   PORTC,r16
            rcall retard
            ret
; routine de retard d’une seconde environ
;retard :
            ldi   premier,25
aa:         ldi   deuxième,255
bb:         ldi   troisème,255
cc:         dec   3e
            brne  cc
            dec   deuxième
            brne  bb
            dec   premier
            brne  aa
            ret
forever:
            rjmp  forever
```

**Figure 2 : Programme afficheur à sept segments.**

porter au niveau logique bas (0) sa broche “RS” par l’intermédiaire de la commande “cbi” qui réinitialise le bit 6 du port D. Après quoi est envoyée la commande “Function Set” qui règle l’afficheur LCD avec un bus de données à 8 bits et une interface à deux lignes.

Une fois la commande envoyée, il est nécessaire de rappeler une routine de retard, car l’afficheur LCD est un dispositif “lent” par rapport au microcontrôleur et, par conséquent, il faut lui donner le temps nécessaire pour “acquérir” la commande qu’on lui a envoyée. “Function Set” est envoyée trois fois. Maintenant, il faut envoyer la commande “Display Control”, paramétrée, dans notre cas, pour allumer l’afficheur LCD et visualiser le curseur et la commande “Entry Mode” pour spécifier la direction de mouvement du curseur. Ensuite, il faut “effacer” l’afficheur LCD avec la commande “Display Clear” et, par la suite, envoyer la commande “Return Home”, déplaçant le curseur sur le premier caractère de la première ligne.

Enfin, il faut communiquer les adresses de la “CG RAM” et de la “DD RAM”. Quand la série des commandes est finie, la broche “RS” prend le niveau logique haut (1) par l’intermédiaire de l’instruction “sbi”. Maintenant on peut envoyer les caractères, cependant il faut d’abord transmettre l’adresse de mémoire pour la “DD RAM” de manière à sélectionner la première ligne. Pour ce faire, on met au niveau logique bas (0) la broche “RS” de l’afficheur LCD, on envoie sur le bus de données l’adresse de mémoire de la première ligne, on porte au niveau logique haut (1) “l’ENABLE” (E) de l’afficheur LCD que l’on remet aussitôt au niveau logique bas (0), on rappelle la routine de retard et enfin on remet au niveau logique haut (1) la broche “RS”. Maintenant, il est possible d’envoyer un à un tous les caractères de la première ligne. La logique d’envoi des caractères est le reflet de l’organigramme de la figure 3. Pratiquement, un compteur distinguant les numéros des caractères de la première ligne de l’afficheur LCD est mis à zéro. Quand le compteur est à zéro, nous écrivons le premier caractère de

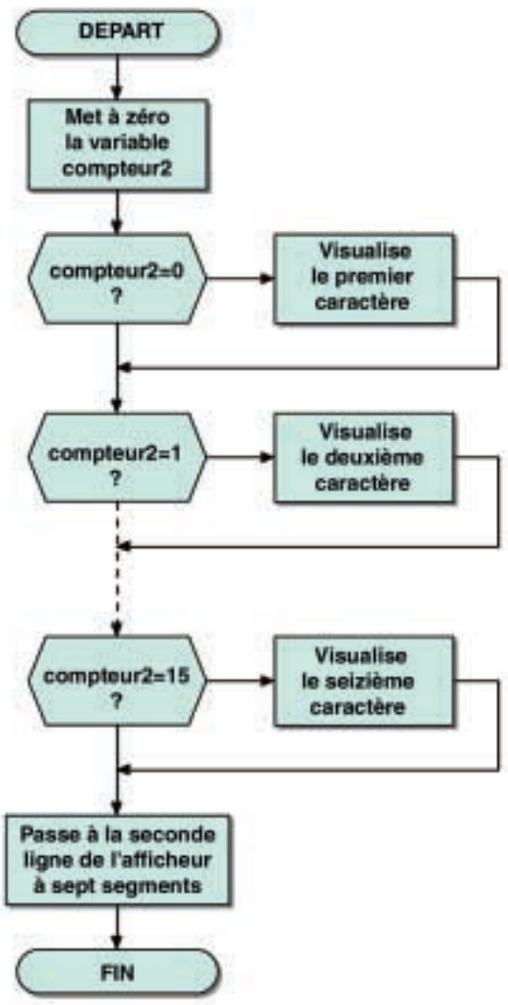



**Figure 3 :**
**Organigramme de gestion de l'afficheur LCD.**

la première ligne. Quand le compteur est à un, nous écrivons le deuxième caractère de la ligne et ainsi de suite.

Si nous regardons le code à partir de l'étiquette "données", nous remarquons qu'une comparaison entre les valeurs de la variable compteur2 et la constante zéro est exécutée d'abord. Si elles sont différentes, on saute à l'étiquette "a" où se trouve le deuxième caractère à écrire. Sinon, on envoie le premier caractère et il en va de même pour les 16 caractères de la première comme de la seconde ligne. Quand la première ligne est terminée, il faut dire à l'afficheur LCD de passer à la seconde : cette opération consiste à envoyer l'adresse de mémoire "DD RAM" de la seconde ligne en utilisant la méthode exposée ci-avant. Arrivés à ce point, les caractères que nous enverrons seront automatiquement visualisés sur la seconde ligne. Le code exposé est répété un nombre infini de fois, visualisant ainsi l'inscription "ELECTRONIQUE LOISIRS MAGAZINE" sur les deux lignes disponibles de l'afficheur LCD. ➡➡➡

```
;****************************************
;*   Dis2x16.asm
;*  23/09/2001
;*
;*  Ce programme gère un afficheur LCD
;*  2 lignes de 16 caractères
;*
;****************************************

.include "8515def.inc"
rjmp    RESET

; Donne des noms symboliques aux registres
.def    Compteur1               = r25
.def    Compteur2               = r26
.def    Donnée          = r27
.def    Return_Home     = r16
.def    Entry_Mode              = r17
.def    Display_Control = "r18"
.def    Shift                   = r19
.def    Function_Set    = r20
.def    Display_Clear   = r21
.def    CG_RAM          = r22
.def    DD_RAM_Riga1    = r23
.def    DD_RAM_Riga2    = r24

RESET:
        ldi     r16,HIGH(RAMEND)
        out     SPH,r16
        ldi     r16,LOW(RAMEND)
        out     SPL,r16

; Initialise Ports A et D comme sorties
        ldi     r16, 0xff
        out     DDRD, r16
        out     DDRA, r16

; Donne une valeur aux variables en jeu
        ldi     Function_Set, 0x38
        ldi     Display_Control, 0x0C
        ldi     Entry_Mode, 0x06
        ldi     Return_Home, 0x02
        ldi     Display_Clear, 0x01
        ldi     CG_RAM, 0x40
        ldi     DD_RAM_Ligne1, 0x80
        ldi     DD_RAM_Ligne2, 0xC0
        ldi     Compteur1, 0xFF

; Initialisation Afficheur
        cbi     PORTD,6
        rcall   Delay

; Envoie à l'afficheur la commande Function_Set
; 3 fois Comme indiquent les spécifications
; pour la programmation de l'afficheur
        out     PORTA,Function_Set
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        sbi     PORTD,7
```

```
        cbi     PORTD,7
        rcall   Delay

; Envoie la commande Display_Control,
; Entry_Mode,Display_Clear, Return_Home
; Pour ces commandes il faut voir la table de
; vérité de l'afficheur
            out PORTA, Display_Control
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        out     PORTA, Entry_Mode
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        out     PORTA, Display_Clear
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        out     PORTA, Return_Home
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay

; Envoie les adresses de la CG_RAM,
; DD_RAM_Ligne1 et Ligne2
        out     PORTA, CG_RAM
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        out     PORTA, DD_RAM_Ligne1
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        out     PORTA, DD_RAM_Ligne2
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay

; Remet au niveau logique haut la broche RS
; de l'afficheur
            sbi PORTD,6

; Envoie caractères
        cbi     PORTD, 6
        out     PORTA, DD_RAM_Ligne1
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        sbi     PORTD, 6
        ldi     Compteur2, 0x00

; Commence à envoyer les caractères un par un
; jusqu'à ce que s'affiche l'inscription
; ELECTRONIQUE LOISIRS MAGAZINE
données :       cpi     Compteur2, 0x00
        brne    a
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
a:      cpi     Compteur2, 0x01
        brne    b
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
b:      cpi     Compteur2, 0x02
        brne    c
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
c:      cpi     Compteur2, 0x03
        brne    d
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
d:      cpi     Compteur2, 0x04
        brne    e
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
e:      cpi     Compteur2, 0x05
        brne    f
        ldi     Donnée , 0x46   ; Lettre F
        rcall   Envoie
f:      cpi     Compteur2, 0x06
        brne    g
        ldi     Donnée , 0x55   ; Lettre U
        rcall   Envoie
g:      cpi     Compteur2, 0x07
        brne    h
        ldi     Donnée , 0x54   ; Lettre T
        rcall   Envoie
h:      cpi     Compteur2, 0x08
        brne    i
        ldi     Donnée , 0x55   ; Lettre U
        rcall   Envoie
i:      cpi     Compteur2, 0x09
        brne    l
        ldi     Donnée , 0x52   ; Lettre R
        rcall   Envoie
l:      cpi     Compteur2, 0x0A
        brne    m
        ldi     Donnée , 0x41   ; Lettre A
        rcall   Envoie
m:      cpi     Compteur2, 0x0B
        brne    n
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
n:      cpi     Compteur2, 0x0C
        brne    o
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
o:      cpi     Compteur2, 0x0D
        brne    p
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
p:      cpi     Compteur2, 0x0E
        brne    q
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
q:      cpi     Compteur2, 0x0F
        brne    aa
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
aa:
        ldi     Compteur2, 0x00
        cpi     Compteur2, 0x00
        brne    bb
        cbi     PORTD, 6
        out     PORTA, DD_RAM_Ligne2
        sbi     PORTD,7
```

```
        cbi     PORTD,7
        rcall   Delay
        sbi     PORTD, 6
        ldi     Donnée ,0x20    ; Lettre " "
        rcall   Envoie
bb:     cpi     Compteur2, 0x01
        brne    cc
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
cc:     cpi     Compteur2, 0x02
        brne    dd
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
dd:     cpi     Compteur2, 0x03
brne    ee
        ldi     Donnée , 0x45   ; Lettre E
        rcall   Envoie

ee:     cpi     Compteur2, 0x04
        brne    ff
        ldi     Donnée , 0x4C   ; Lettre L
        rcall   Envoie
ff:     cpi     Compteur2, 0x05
        brne    gg
        ldi     Donnée , 0x45   ; Lettre E
        rcall   Envoie
gg:     cpi     Compteur2, 0x06
        brne    hh
        ldi     Donnée , 0x54   ; Lettre T
        rcall   Envoie
hh:     cpi     Compteur2, 0x07
        brne    ii
        ldi     Donnée , 0x54   ; Lettre T
        rcall   Envoie
ii:     cpi     Compteur2, 0x08
        brne    ll
        ldi     Donnée , 0x52   ; Lettre R
        rcall   Envoie
ll:     cpi     Compteur2, 0x09
        brne    mm
        ldi     Donnée , 0x4F   ; Lettre O
        rcall   Envoie
mm:     cpi     Compteur2, 0x0A
        brne    nn
        ldi     Donnée , 0x4E   ; Lettre N
        rcall   Envoie
nn:     cpi     Compteur2, 0x0B
        brne    oo
        ldi     Donnée , 0x49   ; Lettre I
        rcall   Envoie
oo:     cpi     Compteur2, 0x0C
        brne    pp
        ldi     Donnée , 0x43   ; Lettre C
        rcall   Envoie
pp:     cpi     Compteur2, 0x0D
        brne    qq
        ldi     Donnée , 0x41   ; Lettre A
        rcall   Envoie
qq:     cpi     Compteur2, 0x0E
        brne    rr
        ldi     Donnée , 0x20   ; Lettre " "
        rcall   Envoie
rr:     ldi     Donnée , 0x20   ; Lettre " "
        out     PORTA, Donnée
        rcall   Envoie
        rjmp    données

; Routine de retard. Nécessaire car le
; microcontrôleur  est beaucoup plus rapide
; que l'afficheur
Delay:
        dec     Compteur1
        cpi     Compteur1,0x00
        brne    Delay
        ldi     Compteur1,0xff
        ret

; Routine pour envoyer les caractères à visualiser
Envoie :
        out     PORTA, Donnée
        sbi     PORTD,7
        cbi     PORTD,7
        rcall   Delay
        inc     Compteur2
        ret
forever:
        rjmp    forever
```

**Figure 4 : Programme afficheur LCD**

***À suivre*** ◆◆◆