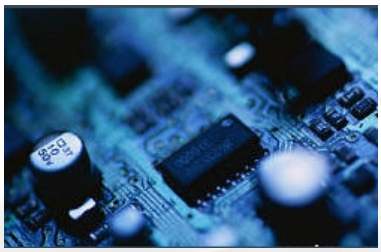
Cours Atmel AT90Sxxxx

NAVIGATION

Index

Les documentations présentées ici traiteront principalement du modèle **AT90S8535**. Ce choix délibéré est expliquer par le fait que ce microcontrôleur est un bon représentant de cette famille, il possède de très bonnes ressources matériels à savoir : 8K de mémoire programme Flash, 512 Octets de mémoire EEprom et RAM, 32 IN/OUT parallèles, 2 interruptions externes, 3 Timers, un CAN 8 voies de résolutions 10 Bits...



Bien entendue, la famille Atmel étant basée sur la même architecture interne et jeux d'instructions, les divergences entre un modèle ou un autre se matérialiseront le plus souvent par la différence de taille mémoire Flash, Ram et EEprom, le retrait de certaines fonctions matérielles tel que le Timer, le convertisseur analogique... Vous pouvez ainsi apprendre à programmer sur la base d'un 90S8535 et par la suite, après de minces modifications, intégrer votre programme sur un composant de gamme inférieure.

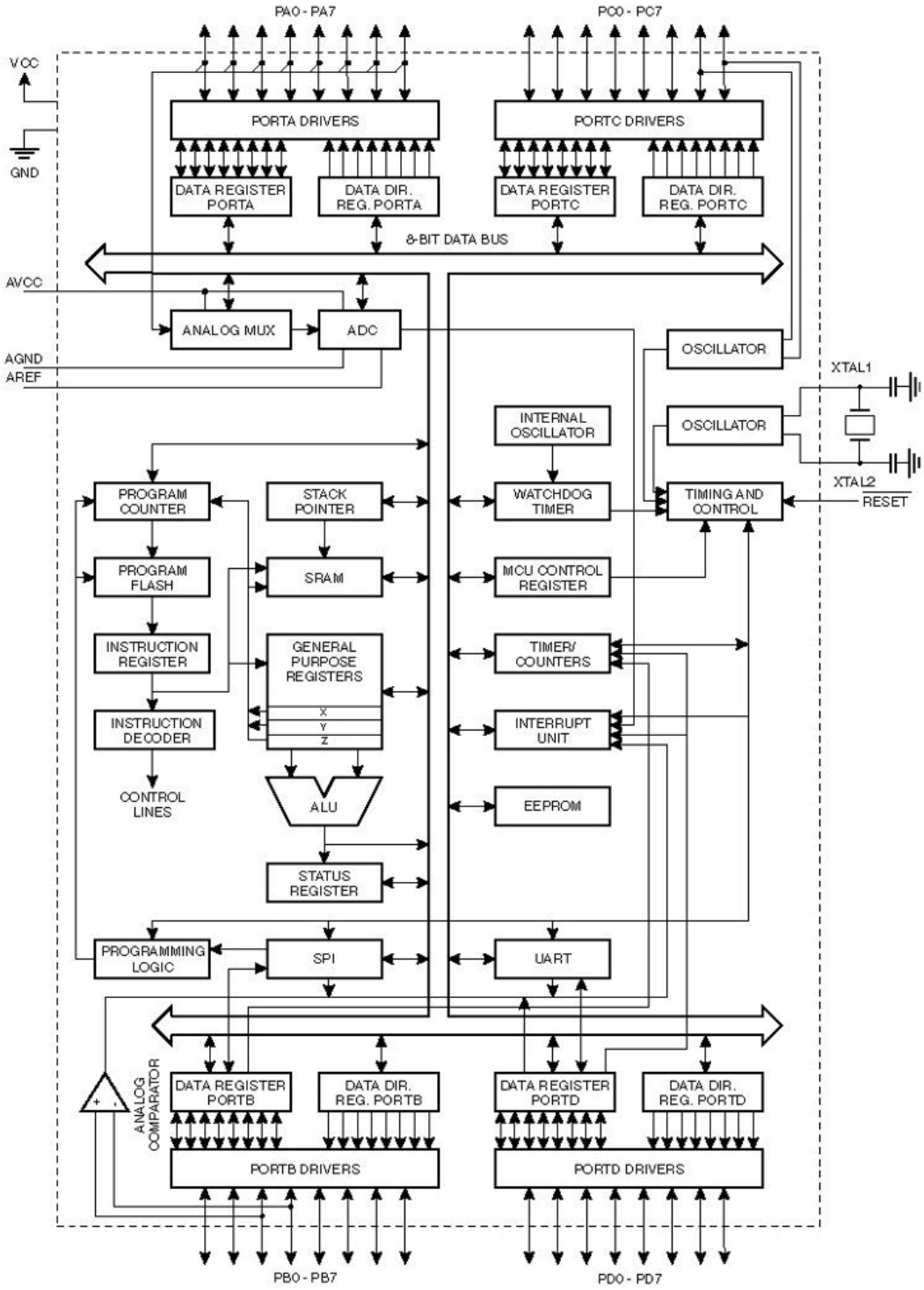
N"hésiter pas à poser vos questions sur le Forum.

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture Interne

Shématique



interne :

On remarque :

- L'unité centrale (ALU)
- Le Bus de donné 8 Bit
- La mémoire Programme Flash
- Le Bus de donné 8 Bit

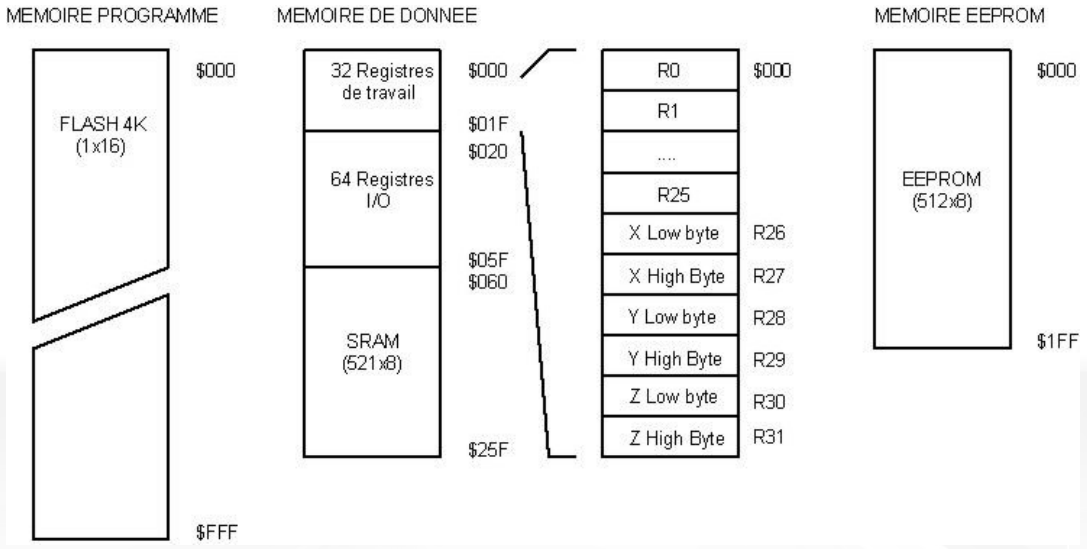
- La mémoire vive Sram (static random access memory)
- Une EEPROM (memoire morte programmable électriquement)
- 1 Convertisseur Analogique/Numérique de 10 Bits (avec selection de la voie d'entrée)
- 4 Ports numérique bi-directionnel avec partage de fonctions (In/Out/Ana,Rx,Tx...)
- 1 Comparateur analogique (sortie sur PB2,PB3)
- 1 SPI (port série synchrone)
- 1 UART (port série asynchrone)
- 3 Compteurs/Timers l

ainsi que VCC et GND, broches d'alimentation du microcontroleur, XTAL1 et XTAL2, pour la partie oscillations
AVCC et AGND, broches d'alimentation du CAN (Convertisseur Analogique/Numérique)
AREF, réference de tension pour le CAN
RESET, Broche de remise à zero du noyau, actif sur front descendant
NMI, entrée d'interruption externe non masquable

Le cœur du noyau est basé sur une architecture RISC (Circuit à jeu d'instructions réduit), qui compte environ 120 instructions. Ce nombre d'instructions est tout de même assez élevé pour une telle architecture, mais la dénomination de RISC prend en compte d'autres paramètres qui font que ces microcontrôleurs sont ainsi classifiés. L'horloge interne n'est pas divisé comme sur les classiques 80C51, ce qui donne une vitesse de traitement rapide, par exemple pour un Quartz de 10 Mhz, 10 Millions d'instructions par secondes (MIPS) sont effectués, la majorité des instructions étant réalisés en 1 ou 2 cycles, les microcontrôleurs AVR sont donc très rapide !

La programmation de la mémoire s'effectue In circuit, pas de programmateur d'eprom ni d'émulateur, Atmel garantie un durée de vie de 1000 cycles d'écritures, une aubaine pour le développement micro.

La cartographie mémoire représentée ci-dessous, est celle d'un AT90S8535. On remarque une fusion entre les registres de travail et la mémoire Ram. Bien sur, la taille des différentes mémoires varie avec le microcontrôleur sélectionné.



Cours Atmel AT90Sxxxx

NAVIGATION

Les registres system

Le registre d' état SREG :

Le registre **SREG** sert principalement pour les opérations arithmétiques. Il est le reflet des états des manipulations mathématiques et permet d'effectuer des tests afin d'effectuer des éventuels branchements au programme. Si vous comprenez mal sont rôle, les exemples de programmations vous démontrerons tout sont intérêt.

Adresse	7	6	5	4	3	2	1	0
$3F	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**

- **I** : Global Interrupt enable, sert à activer(1) ou interdire(0) toutes les sources d' interruptions. Si ce bit n' est pas activer alors que vous
avez programmer des interruptions, elles ne seront pas prise en compte.

- **T** : Copy Storage, ce bit joue un rôle de tampon lors de manipulation de bits avec les instructions BLD et BST

- **H** : Half Cary, signale qu 'une demie-retenue à été réalisé lors de l' emploi d' une instructions arithmétique.

- **S** : Sign bit, bit de signe résultant d' un OU exclusif avec le bit **N** ou **V**.

- **V** : Overflow bit, dépassement de capacité de calcul.

- **N** : Negative bit, il signale que le résultat de la dernière manipulation arithmétique est négative.

- **Z** : Zéro bit, le résultat de la dernière manipulation arithmétique est égale à zéro.

- **C**: Cary bit, l' opération arithmétique à donné lieu à une retenue ou à un dépassement.

Le registre MCUSR :

Ce registre est une extension de **SREG**, il permet de déterminer la source d' un RESET

Adresse	7	6	5	4	3	2	1	0
$34	X	X	X	X	X	X	**EXTRF**	**PORF**

Sur le AT90S8535, les bits 7 à 2 sont inutilisés. Le décodage peut se résumer ainsi :

Source du Reset	EXTRF	PORF
WatchDog	0	0
Mise sous tension	0	1
Reset Externe	1	0
Mise sous tension	1	1

Le registre MCUCR :

C' est un registre de contrôle en rapport avec la gestion de l' unité centrale.

Adresse	7	6	5	4	3	2	1	0
$35	X	X	**SE**	**SM**	**ISC11**	**ISC10**	**ISC01**	**ISC00**

- **SE** : Sleep Enable : mise en sommeil de l'unité centrale.

- **SM** : Sleep Mode : Choix du mode sommeil (arret ou ralentie). Les modes de sommeil ne seront pas étudiés

- **ISC11, ISC10** : Interrupt Sense Control, définition de prise en compte de l'interruption externe INT1 *.

- **ISC01, ISC00** : Interrupt Sense Control, définition de prise en compte de l'interruption externe INT0 *.

* Tableau récapitulatif des modes de programmation INT0 et INT1 :

ISC01	ISC00	Mode de déclenchement	ISC11	ISC10
0	0	Sur niveau Bas	0	0
0	1	sur changement de niveau (0/1 ou 1/0)	0	1
1	0	Sur Front descendant	1	0
1	1	Sur front Montant	1	1

Le Pointeur de pile :

Le pointeur de pile est en fait constitué de 2 registres : **SPL** et **SPH**. La pile sert d'endroit de stockage du compteur ordinal, lequel permet de conserver par exemple, l'adresse de retour du programme suite à un appel de procédure (RCALL), ou bien d'une interruption.
Plus l'espace de la pile est grand, plus le programme pourra faire appel à d'autre sous-programme imbriqués, et interruptions en nombres importantes.
A la suite d'un Reset le pointeur de pile contient l'adresse 60$, adresse qu'il faut immédiatement changé afin de ne pas interférer avec les variables des registres du système. On y inscrira le plus tot possible après un reset la valeur la plus élevé de la RAM.

Cartographie des registres :

Catégorie	*Désignation*	*Registre*	*Adresse*
Registre généraux de travail (voir architecture)	R0...R25	**R0...R25**	$00...$19
	Poids faible registre X	**R26**	$1A
	Poids fort registre X	**R27**	$1B
	Poids faible registre Y	**R28**	$1C
	Poids fort registre Y	**R29**	$1D
	Poids faible registre Z	**R30**	$1E
	Poids fort registre Z	**R31**	$1F
Convertisseur analogique	Poids Faible Résultat	**ADCL**	$04
	Poids fort Résultat	**ADCH**	$05
	Contrôle et statut du CAN	**ADCSR**	$06
	Multiplexeur, sélection de la voie à échantillonner	**ADMUX**	$07
Comparateur analogique	Contrôle et statut du Comparateur	**ACSR**	$08
UART	Vitesse de communications (Bauds)	**UBRR**	$09
	Contrôle	**UCR**	$0A

	Statut	**USR**	$0B
	I/O données	**UDR**	$0C
SPI	Contrôle	**SPCR**	$0D
	Statut	**SPSR**	$0E
	I/O données	**SPDR**	$0F
Port parallèle D	Adresse des broches (Pin)	**PIND**	$10
	Direction des broches	**DDRD**	$11
	données des broches	**PORTD**	$12
Port parallèle C	Adresse des broches (Pin)	**PINC**	$13
	Direction des broches	**DDRC**	$14
	données des broches	**PORTC**	$15
Port parallèle B	Adresse des broches (Pin)	**PINB**	$16
	Direction des broches	**DDRB**	$17
	données des broches	**PORTB**	$18
Port parallèle A	Adresse des broches (Pin)	**PINA**	$19
	Direction des broches	**DDRA**	$1A
	données des broches	**PORTA**	$1B
EEPROM	Contrôle	**EECR**	$1C
	données	**EEDR**	$1D
	Poids faible adresse	**EEARL**	$1E
	Poids Fort adresse	**EEARH**	$1F
Watchdog	WDTCR	**WDTCR**	$21
Timer / compteur 2 (8 bits)	Statut du mode asynchrone	**ASSR**	$22
	Comparaison	**OCR2**	$23
	Compteur	**TCNT2**	$24
	Contrôle	**TCCR2**	$25
Timer / compteur 1 (16 bits)	Entrée poids faible de capture	**ICR1L**	$26
	Entrée poids fort de capture	**ICR1H**	$27
	Timer/compteur B sortie comparaison poids faible	**OCR1BL**	$28
	Timer/compteur B sortie comparaison poids fort	**OCR1BH**	$29
	Timer/compteur A sortie comparaison poids faible	**OCR1AL**	$2A
	Timer/compteur A sortie comparaison poids fort	**OCR1AH**	$2B
	Timer/compteur poids Faible	**TCNT1L**	$2C
	Timer/compteur poids Fort	**TCNT1H**	$2D

	Contrôle Timer/compteur poids Faible	**TCCR1B**	$2E
	Contrôle Timer/compteur poids Fort	**TCCR1A**	$2F
Timer / compteur 0 (8 bits)	Timer/compteur 0	**TCNT0**	$32
	Contrôle Timer/compteur 0	**TCCR0**	$33
MCU	Statuts général MCU	**MCUSR**	$34
	Contrôle Général MCU	**MCUCR**	$35
Interruptions	Drapeau d' interruption Compteur/Timer	**TIFR**	$38
	Masque d' interruption Compteur/Timer	**TIMSK**	$39
	Drapeau d' interruption Généraux	**GIFR**	$3A
	Masque d' interruption Généraux	**GIMSK**	$3B
Pointeur de pile	Pointeur de pile poids faible	**SPL**	$3D
	Pointeur de pile poids fort	**SPH**	$3E
Registre SREG	Statuts Registre	**SREG**	$3F

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale

Les registres

Les interruptions

Le Watchdog

Les entrées / sorties

Le comparateur Analogique

Le convertisseur Analogique

Les timers

L'UART

La SPI

La Ram et l'EEprom

Les soft de programmation

Les interruptions

Présentation :

Pour que le microcontrôleur ne passe pas la majeurer partie de son temps à attendre une condition que vous auriez programmé, les constructeurs ont mis au point les interruptions.Iil s'agit d' avertir le microcontrôleur qu'une condition est remplie et qu' il faut immédiatement la traiter.

Pratiquement chaque périphérique interne dispose d' une source d' interruption.

Fonctionnement :

Raisonnons pratique : imaginons un projet qui aurait pour but d' allumer l' ampoule d' une cage d' escalier pendant x secondes à l' appui d' un bouton poussoir (En réalité nous n' utiliserions pas un microcontrôleur pour effectuer cette tache !) :

Sans les interruptions : nous aurions programmer ainsi :

- attende d' un appui sur le bouton poussoir
- allumage de la lampe
- boucle de temporisation logicielle
- Extinction de la lampe

C' est tout à fait imaginable, mais que diriez vous si en plus votre projet se voyait ajouter quelques fonctions supplémentaires comme la surveillance d' une zone à l' aide d' un capteur infrarouge, la gestion du digicode...
Ça commence à se compliquer. Laissons de coté ces fonctions et intéressons nous aux interruptions :

Avec les interruptions :

- L' appui sur le bouton poussoir est détecté par l' entrée INT0 (cela évite une boucle de conditions)
- l' allumage de la lampe est effectué
- on met en marche un temporisateur qui demande une interruption toutes les secondes
- une valeur de X fois une secondes est chargée dans une registre quelconque
- a chaque interruptions du timer on décrémente le contenue du registre
- si la valeur du registre = 0 : on arrête le timer et la lampe est désactivée

Voila, votre programme principale n'a pas eu à s' occuper de la gestion de la lampe, ce qui vous laisse des ressources logiciels et matérielles pour effectuer bien d' autres taches

Conseils :

Pensez d' abord à lever le drapeau d' autorisation d' interruption général dans **SREG** (instruction SEI).
Donner au périphérique concerné la possibilité d' interrompre en réglant les bits de leurs registres.
Ensuite chaque interruption va provoquer un saut de programme à une adresse bien précise, c' est ici que vous écrirez un saut vers la procédure choisi, le retour de l' interruption sera provoqué par l' instruction RETI.
Sauvegarder impérativement vos registres de travail, surtout **SREG** : en effet dans le programme de l' interruption, la valeur de ce registre à de très grande chance de changer et au retour de l' interruption, imaginez que votre programme principal effectuais un test sur ce même registre...

Pour sauvegarder un registre, il faut le mettre sur la pile avec l' instruction **PUSH**, puis le replacer à l' aide de **POP**

A propos de la pile, il faut changer immédiatement son pointeur en début de programme, car sa configuration d' origine pointe sur $00, en clair vous ne pouvez pas faire de sous-programme !

Si une seconde interruption intervient pendant le traitement de la première, le programme la traitera aussitôt fini la première.

La priorité des interruptions va par ordre croissant (voir tableau) ex : INT1 prioritaire sur ADC

Cartographie des vecteurs d' interruptions : (90S8535)

Adresse	Nom	Description
$00	**RESET**	RESET

$01	**INT0**	Entrée d' interruption broche INT0
$02	**INT1**	Entrée d' interruption broche INT0
$03	**TIM2_COMP**	Comparaison réussie Timer 2
$04	**TIM2_OVF**	Débordement Timer 2
$05	**TIM1_CAPT**	Capture terminé Timer 1
$06	**TIM1_COMPA**	Comparaison réussie Timer 1A
$07	**TIM1_COMPB**	Comparaison réussie Timer 1B
$08	**TIM1_OVF**	Débordement Timer 1
$09	**TIM0_OVF**	Débordement Timer 0
$0A	**SPI_STC**	Transmissions SPI terminée
$0B	**UART_RXC**	Réception d' un caractère par l'UART
$0C	**UART_DRE**	Registre tampon UART vide
$0D	**UART_TXC**	Émission d' un caractère par l'UART terminée
$0E	**ADC**	Conversion A/D terminée
$0F	**EE_RDY**	EEprom Prête
$10	**ANA_COMP**	Comparaison analogique effectuée

Listing :

Un exemple utilisant les interruptions est publié dans la section comparateur analogique

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale

Les registres

Les interruptions

Le Watchdog

Les entrées / sorties

Le comparateur Analogique

Le convertisseur Analogique

Les timers

L'UART

La SPI

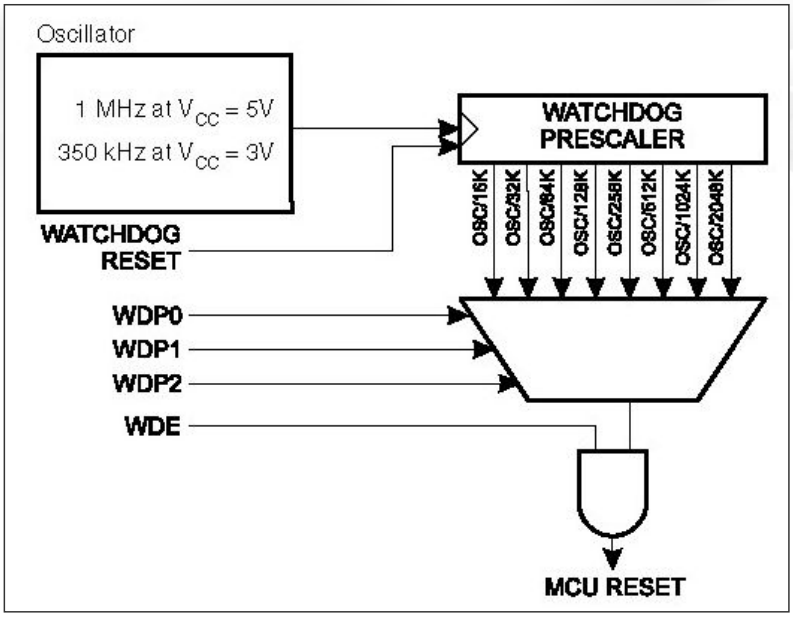
La Ram et l'EEprom

Les soft de programmation

Le Watchdog

Fonctionnement:

Le chien de garde ou Watchdog, est un compteur qui permet de palier au plantage microcontrôleur. Ce plantage peut être d'ordre logiciel (oublie de retour de routines, mis en boucle du programme sur une situation non prévue ou tout simplement erreur de structuration du logiciel), soit matériel (parasites, chute de tensions) ; dans les deux cas, le blocage du programme peut avoir des conséquences catastrophiques : imaginez que le processus demande un arrêt total d'une installations, et que celui-ci reste bloqué !
Le watchdog est un compteur, dont la source d'horloge est indépendante, la fréquence de fonctionnement est de 1MHz pour une tension d'alimentation de 5V et 350KHz pour 3V.
Le comptage part de 0, tout débordement du compteur (255) générer un Reset.
A vous de prévoir périodiquement une remise à zéro du compteur par l'instruction assembleur '**WDR**'



Le registre WDCTR :

Registre de contrôle du Watchdog

Adresse	7	6	5	4	3	2	1	0
$41	x	x	x	**WDTOE**	**WDE**	**WDP2**	**WDP1**	**WDP0**

- **WDTOE** : Bit de sécurité d'arrêt du chien de garde. Une sécurité logicielle permet de ne pas l'arrêter accidentellement. il faut respecter la procédure suivante : WDTOE = 1, puis aussitôt, WDE = 0. Si l'opération n'est pas effectuée durant les 4 prochains cycles d'horloge, ce bit repasse automatiquement à 0.

- **WDE** : Marche/Arrêt Watchdog. La désactivation de ce bit n'est possible qu'avec la procédure précédente.

- **WDP2,1,0** :Sélection du facteur de pré division du signal d'horloge, en fait vous définissez l'intervalle maximum de RAZ *.

* L'oscillateur interne étant entièrement dépendant du quartz, le temps maximum entre deux RAZ varie en fonction de la tension d'alimentation :

WDP2	WDP1	WDP0	Temps Max entre 2 RAZ, VCC= 3V	Temps Max entre 2 RAZ, VCC= 5V
0	0	0	47 ms	15 ms
0	0	1	94 ms	30 ms
0	1	0	190 ms	60 ms
0	1	1	380 ms	120 ms
1	0	0	750 ms	240 ms

1	0	1	1500 ms	490 ms
1	1	0	3000 ms	970 ms
1	1	1	6000 ms	1900 ms

Application :

La mise en marche du watchdog doit se faire le plus tôt possible, ensuite placez l'instruction **WDR** dans votre programme de manière a assurer un déroulement correcte du programme
Un petit conseil : n'oublier pas les temporisations, boucles d'attente de conditions…

Vous pouvez à tout moment déterminer la source d' un Reset grâce au registre **MCUSR**

Mise en marche du WD	Arrêt du WD

WDR ;mis à 0 compteur
LDI R16,$0F ;WDE = 1 avec base de temps Max
OUT WDTCR,R16
	LDI R16,$1E ;WDTOE et WDE = 1
	OUT WDTCR,R16
	LDI R16,$10 ;WDTOE =1 et WDE = 0
	OUT WDTCR,R16

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale

Les registres

Les interruptions

Le Watchdog

Les entrées / sorties

Le comparateur Analogique

Le convertisseur Analogique

Les timers

L'UART

La SPI

La Ram et l'EEprom

Les soft de programmation

Les entrées / sorties

Présentation :

Afin de communiquer avec le monde extérieur, les microcontrôleurs Atmel sont pourvues de ports d'entrées/sorties multidirectionnels : en effet, une broche spécifique peut être configurée soit en entrée ou sortie TOR, ou encore en entrée analogique sans parler des fonctions spéciales (RX,TX,INT0…)

Les registres :

Pour contrôler le mode de fonctionnement de chaque broche, 3 registres sont mis à notre dispositions :

- **DDR**x : Registre de direction
- **PORT**x : registre de donnés
- **PIN**x : registre d'état

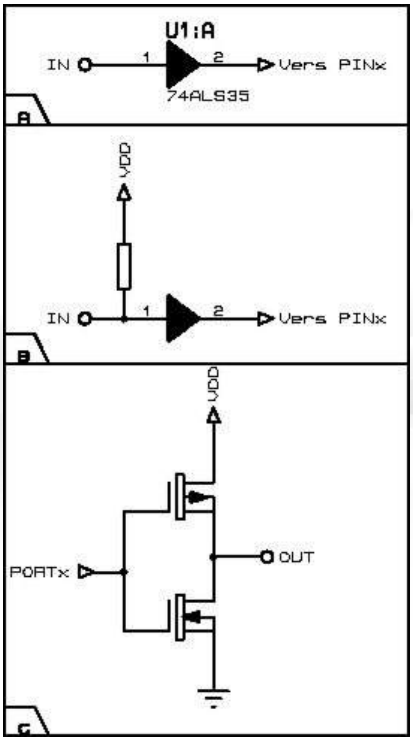
X Représente le nom de port (A,B,C,D)

Chacun de ces registres et configurable bit à bit, c'est à dire que sur l'on peu utiliser sur le même port des fonctions différentes.

Pour simplifier les explications des différents registres, le tableau suivant résume la configuration :

DDRx	PORTx	PINx	Configuration
0	0	In	Entrée sans résistance de rappel (a)
0	1	In	Entrée avec résistance de rappel (b)
1	0	X	Sortie à Zéro (c)
1	1	X	Sortie à 1 (c)

Représentation simplifié des configurations possibles :



Précision concernant la sortie push-pull :

La sortie push pull permet de délivrer du courant ou d'en écouler par la masse.

A titre d'informations, chaque broche peut fournir/écouler 20 ma, à utiliser en accord avec les 2 règles suivantes :

- Chaque port de 8 bits est limité à un courant total de **100 ma**
- Le microcontrôleur lui même peut supporter au maximum **300 ma**

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale

Les registres

Les interruptions

Le Watchdog

Les entrées / sorties

Le comparateur Analogique

Le convertisseur Analogique

Les timers

L'UART

La SPI

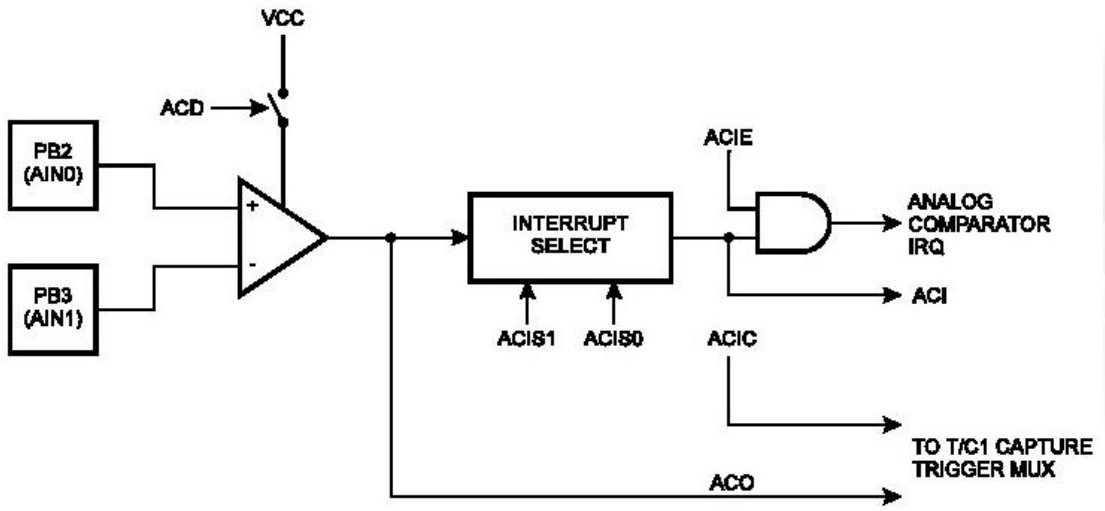
La Ram et l'EEprom

Les soft de programmation

Le comparateur analogique

Présentation :

Inspiré par son cousin analogique, celui-ci est entièrement configurable.



Le registre ACSR :

Adresse	7	6	5	4	3	2	1	0
$08	**ACD**		**ACO**	**ACI**	**ACIE**	**ACIC**	**ACIS1**	**ACIS0**

- **ACD** : Analog Comparator Disable : Bit d'arret du comparateur analogique; en cas d' inutilisation du comparateur, mettre ce bit à 1 afin de réduire la consommation du microcontrôleur. Le comparateur peut être arrêté ou mis en marche à tout moment.

- **ACO** : Analog Comparator Output :Résultat direct de la sortie du comparateur.
si Vain0 > Vain1 alors **ACO** = 1

- **ACI** : Analog Comparator Interrupt Flag : Bit de demande d'interruption quand la conditions sélectionnés est vérifiés (ACIS1 et ASCI0). Ce bit retourne à 0 automatiquement après traitement de l'interruption (si **ACIE** =1)

- **ACIE** : Analog Comparator Interrupt enable : Bit de validation de l'interruption **ANA_COMP**

- **ACIC**: Analog Comparator Input Capture Enable : la mise à 1 de ce bit connecte la sortie du comparateur à l'entrée de capture du timer 1.

- **ACIS1 & ACSI0** : Séléction du mode d'activation de la sortie ACO :

Mode d'activation de la Sortie ACO	ACIS1	ACSI0
Changement d'etat 1-->0 ou 0-->1	0	0
Front descendant	1	0
Front montant	1	1

Application :

Ici nous déciderons de nous servir du comparateur analogique pour enclencher une alarme haute température.

La tension issue de la sonde de température varie de 0 à 5 Volt et correspond à une gamme de 0 – 100 °C.
Le signal sera câblé sur la broche AIN0 (PB2)
Le point de consigne de l'alarme est définie par un potentiomètre câblé sur AIN1(PB3)

Nous aurons donc l'équation suivante :

Si Ain0 > AIN1 alors Alarme = 1 (**si** température **>** point de consigne **alors** alarme **=** 1)
Si AIN0<AIN1 alors Alarme = 0

Paramétrage du comparateur :

L'écriture du programme se résume ainsi :

1 - SEI = 1 (autorisation d'interruption général)
2 - ACD = 0 (mise en marche du convertisseur)
3 - ACIS1 = 0 et ACIS0 = 0 (Déclanchement de l'interruption par changement d'etat à la sortie de comparaison)
4 - ACIE = 1 (autorisation du comparateur à générer un interruption)

Programmation de la routine d'interruption :

A chaque changement de front à la sortie du comparateur, une interruption sera généré (ACI) et il restera à déterminer soit la mise en marche de l'alarme ou son arrêt, grâce à l'état du bit ACO

Attention : ce montage ne propose pas d'hystérésis

Listing :

Cours Atmel AT90Sxxxx

NAVIGATION

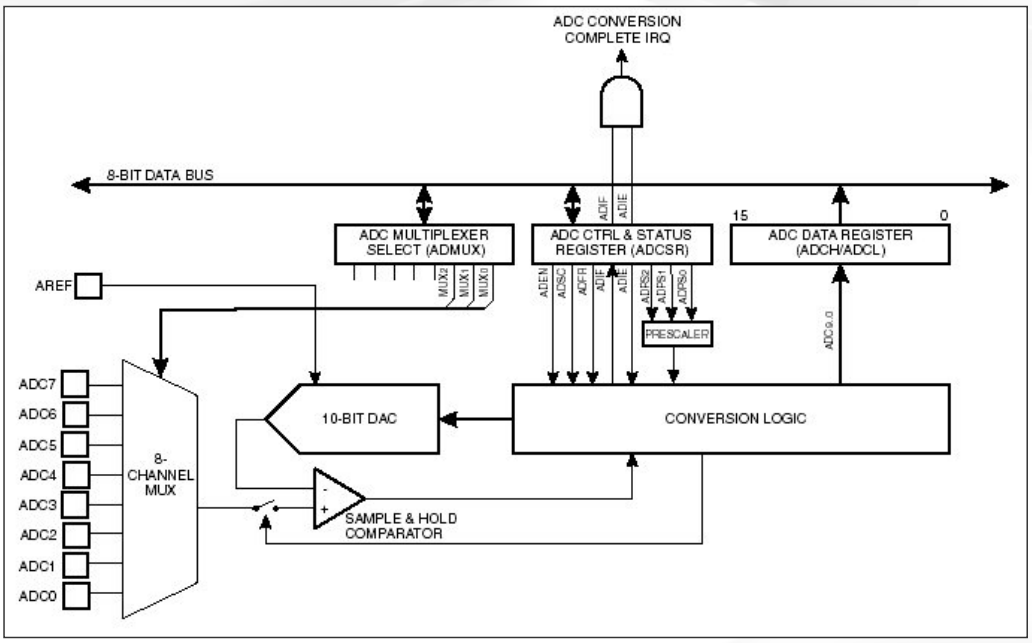
Le convertisseur analogique / digital

Présentation :

Le convertisseur analogique / numérique (CAN) intégré dans le AT90S8535 se dote de bonnes caractéristiques, en effet, une résolution de 10 bits n'est pas monnaie courante sur genre de petits microcontrôleurs.

En voici le résumé des caractéristiques :

- Résolution : 10 bits
- Nombres de voies : 8
- Non linéarité : inférieur à 1/2 LSB
- Erreur à zéro volt : 1 LSB
- Temps de conversion : réglable de 65 à 260 uS (plus le temps est long, plus le résultat et précis), soit près de 15000 échantillons/secondes
- Tension de référence externe (conversion de 0 à Vref analogique, le maximum étant VCC)
- Acquisitions simples ou continues



Rappel : Le convertisseur étant chargé de convertir une tension analogique en résultat numérique codé sur 10 bits, nous pouvons écrire l'équation suivante :

Résultat numérique = (Tension d'entrée / tension de référence Avref) X 1024

par exemple, pour avoir le résultat d'une tension d'entrée de 2,5 Volts, avec une tension de référence de 5 Volts (Avref) nous aurons :

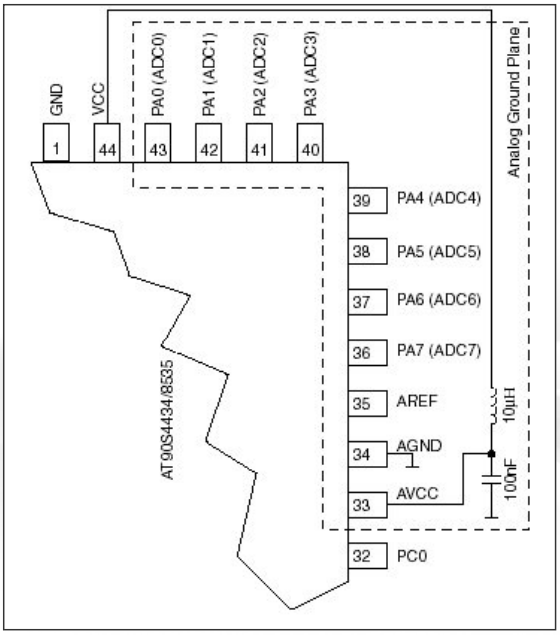
RN = (2,5/5) x 1024 = 512

Le temps de conversion est égale à 13 x Horloge système x facteur de pré-division

Attenuation du bruit de conversion:

Afin de réduire au maximum les erreurs de conversions dues à la logique interne du controlleur, plusieurs solutions peuvent êtres mis en oeuvre afin d'éviter ce désagrément :

- mettre en sommeil l'unité centrale avant le lancement d'une conversion
- découpler soigneusement l'alimentation en respectant le schéma présenté ci-après
- respecter les règles élémentaires du routage de circuit imprimé en analogiques (connexions courtes, plan de masse...)
- dans tout les cas, effectuer toujours un traitement numérique des résultats (amortis,moyenne...)



Les registres :

Ils sont aux nombres de quatre :

ADMUX : Selection de la voie de conversion (sur le port A, configuration des pins en entrées sans résistance de rappel)

Adresse	7	6	5	4	3	2	1	0
$07						**MUX2**	**MUX1**	**MUX0**

Tableau de multiplexage :

MUX2	MUX1	MUX0	Selection de la voie
0	0	0	0 (PA0)
0	0	1	1 (PA1)
0	1	0	2 (PA2)
0	1	1	3 (PA3)
1	0	0	4 (PA4)
1	0	1	5 (PA5)
1	1	0	6 (PA6)
1	1	1	7 (PA7)

ADCSR : Registre de controle et statut

Adresse	7	6	5	4	3	2	1	0
$06	**ADEN**	**ADSC**	**ADFR**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**

- **ADEN** : Mise en marche du convertisseur.

- **ADSC** : Lancement de la conversion de la voie selectionnée (retourne à 0 en fin de conversion).

- **ADFR** : La mis à 1 de ce bit permet de mettre en le convertisseur en mode acquisiton continue apres validation de ADSC.

- **ADIF** : passe à 1 une fois la conversion terminée et déclanche l'interruption si ADIE =1. Ce bit repasse automatiquement à 0 lors du traitement de la routine d'interruption.

- **ADIE** : Validation de l'interruption **ADC** , déclanché lors du passage à 1 de ADIF.

- **ADSP2...ADSP0** : Séléction du facteur de pré-division de l'horloge interne du convertisseur :

ADSP2	ADSP1	ADSP0	Facteur de division
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Généralement, une fréquence de 100 Khz permet d'exploiter de facon optimale le convertisseur.

ADCL et ADCH : Registres de résultats de la conversion analogique / digital

Adresse	7	6	5	4	3	2	1	0
$05							**AC9**	**ADC8**

Adresse	7	6	5	4	3	2	1	0
$04	**ADC7**	**ADC6**	**ADC5**	**ADC4**	**ADC3**	**ADC2**	**ADC1**	**ADC0**

Application :

Paramétrage du convertisseur (application basique) ***:***

1 - Mise en marche du convertisseur (ADEN = 1)
2 - Sélection du facteur de pré-division (ADPS2...ADPS0)
3 - Sélection de l'entrée à convertir (MUX2...MUX0)
4 - Enclanchement d'une acquisition (ADSC = 1)
5 - Attente fin de conversion (ADSC = 0)
6 - Lecture de ADCL et ADCH

Exemple :

Acquisition de la voie 0 et affichage du résultat sur un barre-graphe de LEDs cablé sur PB0...PB7 pour le mot de poids faible et PC0 PC1 pour les bits de poids forts.
Attention de ne pas cabler toutes les LEDs directement sur les ports du microcontrôleur : rappelez vous l'intensité maximum supportée par celui-ci (voir section IN/OUT)

Listing :

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale
Les registres
Les interruptions
Le Watchdog
Les entrées / sorties
Le comparateur Analogique
Le convertisseur Analogique
Les timers
L'UART
La SPI
La Ram et l'EEprom
Les soft de programmation

Les Timers

Présentation :

Un Timer est une circuiterie logique qui permet d'effectuer du comptage de temps, d'évenements, de base de temps pour la génération de signaux...
Les Timers représentes sans doute la partie la plus difficile à maitriser, certainement due aux nombres de registres, c'est pourquoi il faut vraiment prendre le temps de bien s'imprégner du fonctionnement général et de la signification de chacun des ces registres.

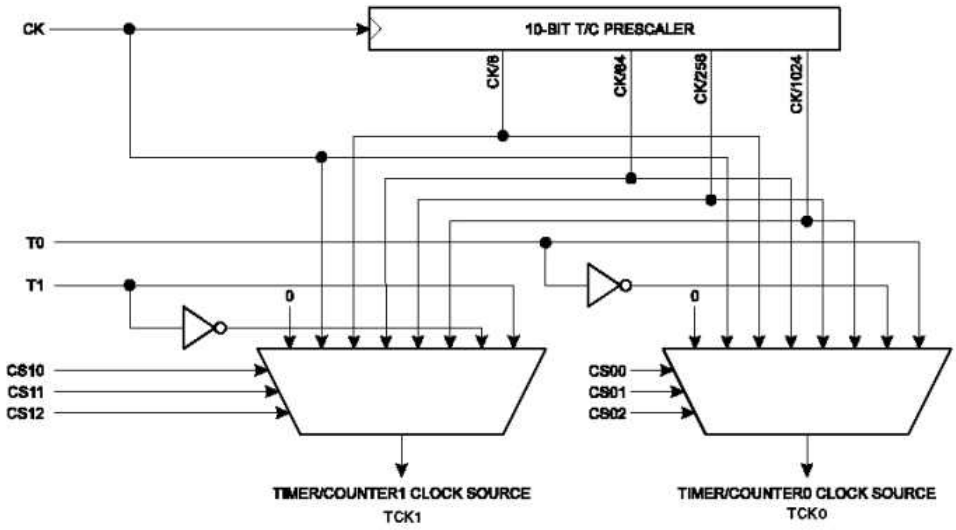
Le mode de fonctionnement en PWM sera présenté en fin de chapitre dans le but de ne pas compliquer la compréhension d'utilisation des Timers.

Le AT90S8535 est doté de 3 Timers :

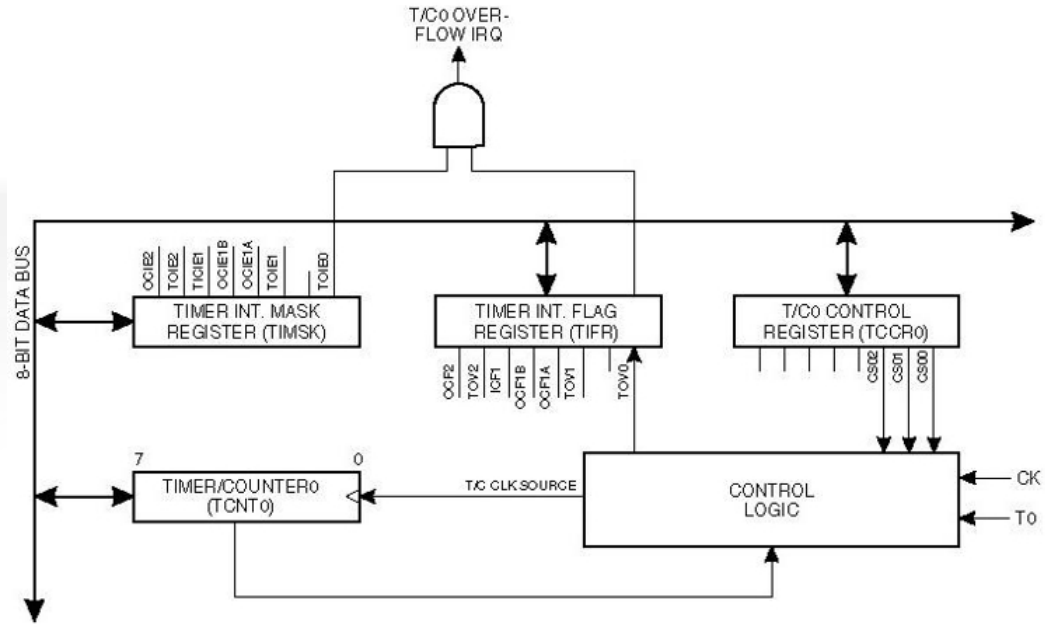
TIMER 0

C'est le plus simple des timers, il est composé d'un compteur 8 Bits.
On constate que son architecture est mise en commun avec le TIMER 1, et oui, généralement le timer 0 et 1 sont présents sur la famille AT90S85XX, le Timer 2 lui étant indépendant et optionnel

Architecture générale du Timer 0 et 1



ARCHITECTURE DU TIMER 0



Les registres :

TIFR : Registre d'etat partagé avec le TIMER 1 et 2

Adresse	7	6	5	4	3	2	1	0
$38	OCF2	TOV2	ICF1	OCF1A	OCF1B	**TOV1**		**TOV0**

- **TOV0** : Bit d'indication de débordement du Timer 0 (255 --->0). il est remis à 0 automatiquement lors du traitement de l'interruption correspondante, ou par écriture d'un 1 logique à la meme adresse.

TIMSK : Registre de validation des interruptions partagé avec le TIMER 1 et 2

Adresse	7	6	5	4	3	2	1	0
$39	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1		**TOIE0**

- **TOIE0** : Validation de l'interruption **TIM0_OVF** au débordement du Timer 0.

TCNT0 : Compteur du Timer 0 accessible en lecture / écriture

Adresse	7	6	5	4	3	2	1	0
$32	**MSB**							**LSB**

TCCR0 : Registre de controle du Timer 0 : séléction du facteur de pré-division de l'horloge de comptage ou incrémentation par signaux externe pin T0

Adresse	7	6	5	4	3	2	1	0
$33						**CS2**	**CS1**	**CS2**

Tableau de multiplexage :

MUX2	MUX1	MUX0	Source
0	0	0	Rien (stop le compteur)
0	0	1	Horloge systeme / 1
0	1	0	Horloge systeme / 8
0	1	1	Horloge systeme / 64

1	0	0	Horloge systeme / 256
1	0	1	Horloge systeme / 1024
1	1	0	Pin T0, active sur front descendant
1	1	1	Pin T0, active sur front montant

Application :

Paramétrage du TIMER 0 :

1 - Si interruption penser à mettre l'instruction SEI
2 - Dans le meme cas, mettre à 1 le drapeau TOIE0 du registre TIMSK
3 - Inscription de la valeur du compteur avec TCNT0
4 - Sélection de la source d'horloge du compteur dans TCCR0, qui à ce meme met le compteur en fonctionnement
5 - Execution de la routine d'interruption et rechargement du compteur

Exemple :

Déclanchement d'une interruption toute les 3,2 ms, signalé par un créneau sur la broche PB0

Premier élement à prendre en compte : la fréquence de résonance du Quartz; Dans le cas présent 8 Mhz, ce qui donne une période d'horloge de 125 nano-secondes (T=1/F --> T = 1 / 8 000 000). Vient à cela s'ajouter le choix de la pré-division (CS02...CS00) et ensuite la valeur du compteur (TCNT0)
On atribura les valeurs suivantes : Division d'horloge de 256 et valeur du compteur : 100
Ce qui finalement donne : 0,000000125 x 256 x 100 = 3,2 ms

Mais **attention**, on souhaite faire débordé le compteur pour 100 pas d'incrémentation, la valeur chargée réellement correspondras à 256-100 soit 156, et oui notre compteur ne décrémente pas de 255 à 0, il incrémente !

Listing :

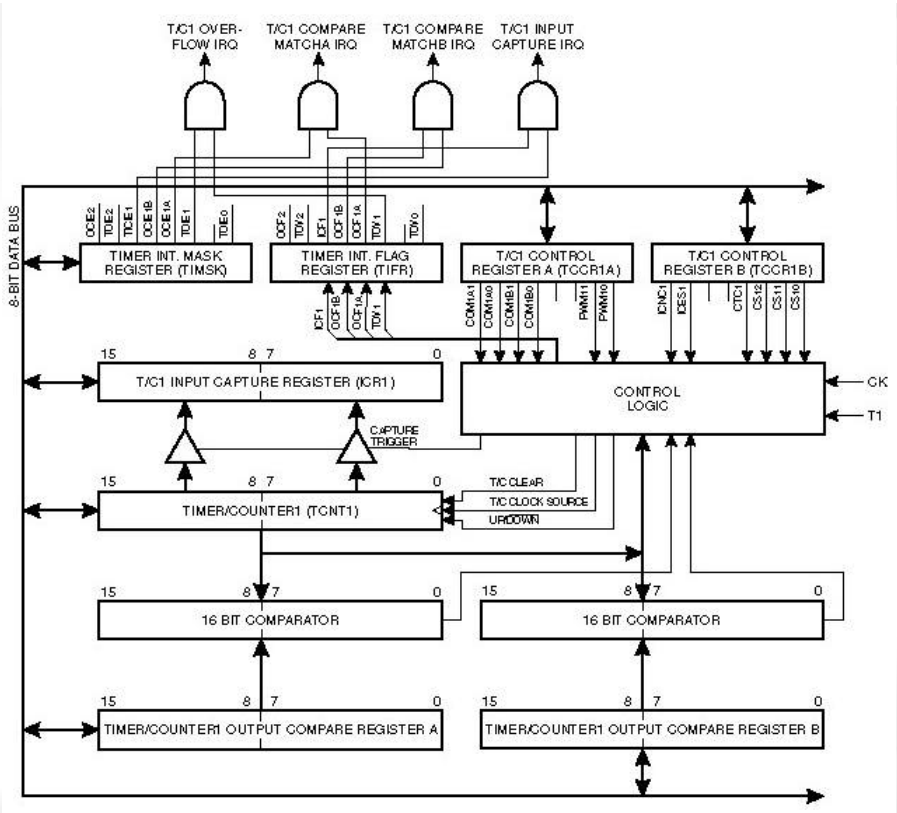
TIMER 1

Présentation :

Le TIMER 1 est beaucoup plus évolué que le TIMER 0, pour cause il est constitué de :

- Un compteur de 16 Bits
- 2 registres de comparaison liées sur 2 sorties (OC1A = PD5 et OC1B = PD4)
- Génération de PWM
- Un registre de capture avec une entrée externe (ICP = PD6)

-
ARCHITECTURE DU TIMER 1



Les registres :

Revoila nos deux registres partagé :

TIFR : Registre d'etat

Adresse	7	6	5	4	3	2	1	0
$38	OCF2	TOV2	**ICF1**	**OCF1A**	**OCF1B**	**TOV1**		TOV0

- **ICF1** : Passe à 1 lors de la copie du registre **TCNT1** vers **ICR1**, déclanché par un changement de front sur l'entrée ICP (PD6). A noter que cette entrée dispose d'un **réducteur de bruit**, efficace contre tout parasite !

- **OCF1A** : comparaison réussie de **TCNT1 = OCR1A**.

- **OCF1B** : comparaison réussie de **TCNT1 = OCR1B**.

- TOV1 : débordement du registre de comptage **TCNT1**.

Tous ces drapeaux sont remis à 0 lors du traitement de l'interruption correspondante, ou en écrivant un 1.

TIMSK : Registre de validation des interruptions

Adresse	7	6	5	4	3	2	1	0
$39	OCIE2	TOIE2	**TICIE1**	**OCIE1A**	**OCIE1B**	**TOIE1**		TOIE0

- TICIE1 : Valide l'interruption **TIM1_CAPT** lors de la détection d'un front actif sur l'entrée **ICP**.

- **OCIE1A** : Valide l'interruption **TIM1_COMPA** si comparaison réussie de **TCNT1 = OCR1A**.

- OCIE1A : Valide l'interruption **TIM1_COMPB** si comparaison réussie de **TCNT1 = OCR1B**.

- TOIE1 : Validation de l'interruption **TIM1_OVF** au débordement du Timer 1, **TCNT1**.

TCCR1A : Registre de controle du timer 1

Adresse	7	6	5	4	3	2	1	0
$2F	**COM1A1**	**COM1A0**	**COM1B1**	**COM1B0**			**PWM11**	**PWM10**

- COM1A1 & COM1A0 : Définissent l'action à réaliser sur la sortie **OC1A** quand la comparaison de **OCR1A** et **TCNT1** est vérifié.

- COM1B1 & COM1B0 : Définissent l'action à réaliser sur la sortie **OC1B** quand la comparaison de **OCR1B** et **TCNT1** est vérifié.

- PWM11 & PWM10 : Selection du mode de fonctionnement PWM.

Tableaux correspondants :

COM1A1	COM1A0	Action sur OC1x	COM1B1	COM1B0
0	0	Timer 1 déconnecté PWM non connecté	0	0
0	1	Changement d'etat PWM non connecté	0	1
1	0	Mise à 0 PWM normale	1	0
1	1	Mise à 1 PWM inversé	1	1

PWM11	PWM10	Mode PWM
0	0	Arreté
0	1	PWM 8 Bits (Fck/510)
1	0	PWM 9 Bits (Fck/1022)
1	1	PWM 10 Bits(Fck/2046)

TCCR1B : Registre de controle du timer 1

Adresse	7	6	5	4	3	2	1	0
$2E	**ICNC1**	**ICNS1**			**CTC1**	**CS12**	**CS11**	**CS10**

- ICNC1 : Mise en service du réducteur de bruit de l'entrée **ICP**

- ICNS1 : Sélection du type de front de l'entrée **ICP** : 0 = Front descendant, 1 = Front Montant. A cet instant le contenu du registre du compteur **TCNT1** est transféré dans **IRC1**.

- CTC1 : Mode de fonctionnement du compteur **TCNT1** : 0 = comptage en continu, débordement et retour à 0 , 1 = comptage j'usqu'a ce que **TCNT1 = OCR1A**, puis le cycle recommence.

- CS12...CS10 : Sélection de la source d'horloge du compteur **TCNT1**

Tableau de multiplexage :

CS12	CS11	CS10	Source
0	0	0	Rien (stop le compteur)
0	0	1	Horloge systeme / 1
0	1	0	Horloge systeme /8
0	1	1	Horloge systeme / 64
1	0	0	Horloge systeme / 256
1	0	1	Horloge systeme / 1024
1	1	0	Pin T1, active sur front descendant
1	1	1	Pin T1, active sur front montant

ICR1 : Registre de capture du timer 1en lecture seule

Adresse	7	6	5	4	3	2	1	0
$27	**MSB**							

Adresse	7	6	5	4	3	2	1	0

$26								**LSB**

TCNT1 : Registre du compteur Timer 1

Adresse	7	6	5	4	3	2	1	0
$2D	**MSB**							

Adresse	7	6	5	4	3	2	1	0
$2C								**LSB**

OCR1A : Registre de comparaison A

Adresse	7	6	5	4	3	2	1	0
$2B	**MSB**							

Adresse	7	6	5	4	3	2	1	0
$2A								**LSB**

OCR1B : Registre de comparaison B

Adresse	7	6	5	4	3	2	1	0
$29	**MSB**							

Adresse	7	6	5	4	3	2	1	0
$228								**LSB**

Application :

Les différentes utilisations du timer sont tellement nombreuse que je ne peut tout développer. néanmoins je reste à votre disposition pour toutes demandes d'aide.

Exemple :

Ici nous allons décider de faire clignoter une Led branché sur PB0, et ce de manière cyclique toute les secondes.
Pour cela nous utiliserons le compteur **TCNT1** avec le comparateur **OCR1A** ; dès que les contenus seront identiques, le drapeau d'interruption sera levée et l'interruption correspondante traitée.

on choisira une valeur arbitraire de division d'horloge par 1024, ce qui nous donne une incrémentation de :

$$\frac{1}{fquartz} * 1024 = \frac{1}{8000000} * 1024 = 128\mu S$$

pour faire une secondes reste à calculer le coéfficient multiplicateur :

$$\frac{1}{0.000128} = 7812{,}5$$

Le compte n'etant pas juste, reste à choisir 7812 ou 7813 !

Dans l'ordre on ecrit :

1 -Validation de l'interruption générale **SEI**, puis mis à 1 du drapeau **OCIE1A** (interruption quand **TCNT1** =**OCR1A**)
2 - Chargement de la valeur de comparaison dans **OCR1A** (7813)
3 - Remise à 0 du compteur **TCNT1** , bien que pas indispensable, ceci permet de fonctionner correctement au tout début du comptage
4 - Selection du facteur de division de l'horloge, et par conséquent mis en marche du timer

AVERTISSEMENT : il semble que pour entrer la valeur dans OCR1A, il soit préférable de rentrer d'abord le poids fort, puis le poids faible, sous peine de ne voir prendre en compte que le poids faible !

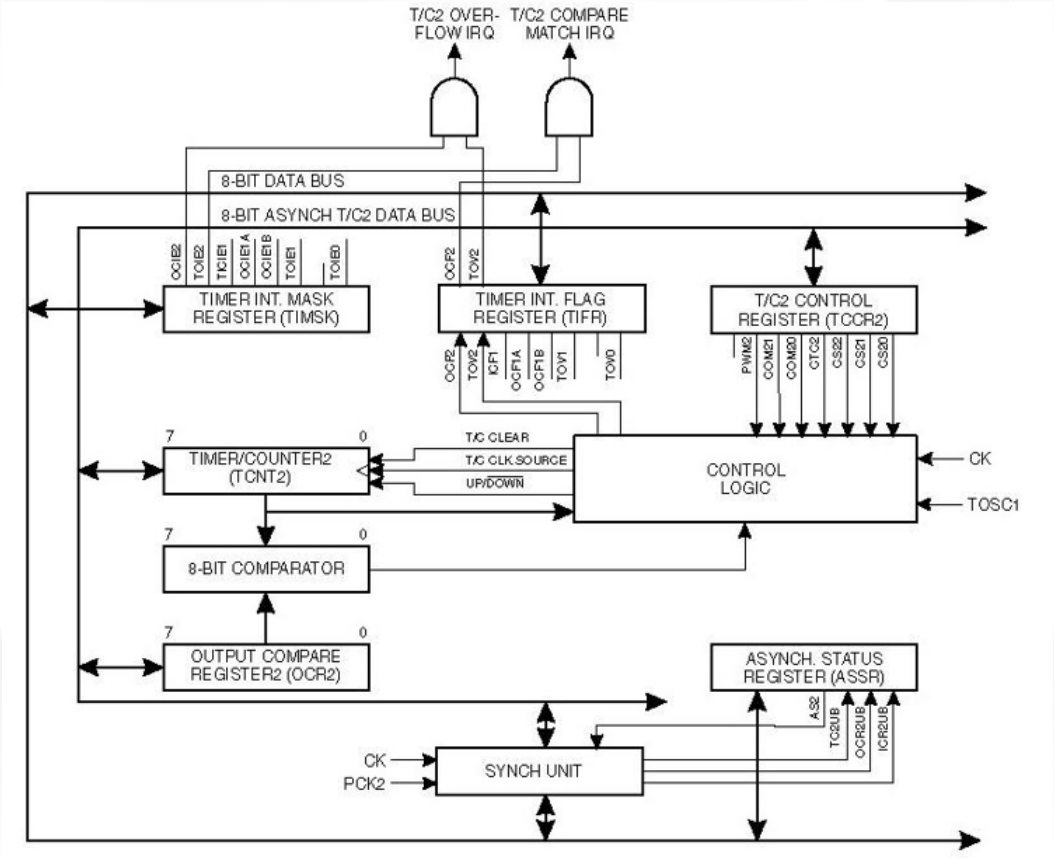
Listing :

TIMER 2

Présentation :

Le TIMER 2 comporte un compteur 8 Bit et comme le timer 1 un registre de comparaison qui agit sur OC2(PD7) et aussi un générateur de PWM

Architecture du Timer 2



Les registres :

Toujours les deux registres partagé :

TIFR : Registre d'etat

Adresse	7	6	5	4	3	2	1	0
$38	**OCF2**	**TOV2**	ICF1	OCF1A	OCF1B	TOV1		TOV0

- **OCF2** : Bit d'indication de débordement du compteur 2 **(TCNT2**).

- **TOV2** : Comparaison réussie de **TCNT2 = OCR2**.

drapeau remis à 0 lors du traitement de l'interruption correspondante, ou en écrivant un 1.

TIMSK : Registre de validation des interruptions

Adresse	7	6	5	4	3	2	1	0
$39	**OCIE2**	**TOIE2**	TICIE1	OCIE1A	OCIE1B	TOIE1		TOIE0

- OCIE2 : Valide l'interruption **TIM2_COMP** si comparaison réussie de **TCNT2 = OCR2**.

- TOIE2 : Validation de l'interruption **TIM2_OVF** au débordement du Timer 2, **TCNT**2.

TCCR2 : Registre de controle du Timer 2

Adresse	7	6	5	4	3	2	1	0
$25		**PWM2**	**COM21**	**COM20**	**CTC2**	**CS22**	**CS21**	**CS20**

- PWM2 : Valide le mode de fonctionnement du Timer en PWM.

- COM20 & COM21 : Définissent l'action à réaliser sur la sortie **OC2** quand la comparaison de **OCR2** et **TCNT2** est vérifié.

- CTC2 : Mode de fonctionnement du compteur **TCNT2** : 0 = comptage en continu, débordement et retour à 0 , 1 = comptage j'usqu'a ce que **TCNT**2 = OCR2, puis le cycle recommence.

- CS22...CS20 : Sélection de la source d'horloge du compteur **TCNT2**

Tableaux correspondants :

COM21	COM20	Action sur OC2
0	0	Timer 2 déconnecté PWM non connecté
0	1	Changement d'etat PWM non connecté
1	0	Mise à 0 PWM normale
1	1	Mise à 1 PWM inversé

CS22	CS21	CS20	Source
0	0	0	Rien (stop le compteur)
0	0	1	Horloge systeme / 1
0	1	0	Horloge systeme /8
0	1	1	Horloge systeme / 32
1	0	0	Horloge systeme / 64
1	0	1	Horloge systeme / 128
1	1	0	Horloge systeme / 256
1	1	1	Horloge systeme / 1024

ASSR : Registre de controle et statut du Timer 2

Adresse	7	6	5	4	3	2	1	0
$22					**AS2**	**TCN2UB**	**OCR2UB**	**TCR2UB**

- AS2 : Sélection de la source d'horloge du compteur timer 2 : 0 = Horloge interne ; 1 = Horloge externe connectée entre les broches PC6 et PC7 , à l'aide d'un quartz .

Le reste des registres ne seront pas décrits car ils servent uniquement pour le mode asynchrone, qui ne sera pas développé ici.
Ce mode à été créer dans le but de mettre en oeuvre une horloge temps réel.

TCNT2 : Registre du timer 2

Adresse	7	6	5	4	3	2	1	0
$24	**MSB**							**LSB**

OCR2: Registre de compaison du Timer 2

Adresse	7	6	5	4	3	2	1	0
$23	**MSB**							**LSB**

Mode PWM du Timmer 1

Présentation :

Le mode **PWM** (pusle wave modulator, en francais modulation en largeur d'impulsion) sert à générer un train d'onde de fréquence constante, mais de rapport cyclique variable. (a noter que l'on puisse très bien faire les 2 en meme temps).

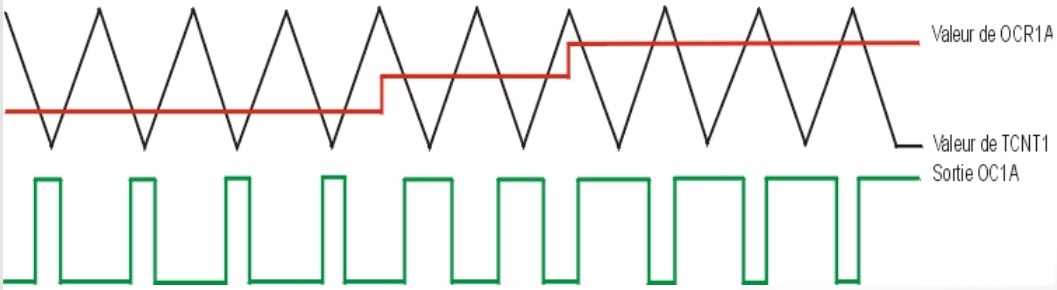
Les applications pratiques vont de la régulation de vitesse pour les moteurs à courant continu, à la variation de l'intensité lumineuse en passant par tout ce qui vous passe par la tete !

Fonctionnement :

En mode **PWM**, le timer et configuré pour sortir le signal sur une broche de sortie (OC1A par exemple).
Le registre **TCCR1A** est principalement utilisé.
Le compteur **TCNT1** compte de 0 à sa valeur maximale, puis décompte et ainsi de suite...
La valeur du compteur est définit par sa résolution de 8, 9 , et 10 bits ce qui correspond aux valeurs respective 255, 511, et 1023.
Dès que **TCNT1** est égale au contenu de la valeur de comparaison, la sortie correspondante est activée.

On peut donc conclure que la Valeur de **TCNT1 fixe la fréquence** et **OCRx le rapport cyclique**.

Ci dessous le chronogramme illustré du mode de fonctionnement avec OC1A :



Application :

Paramétrage :

1 - Choisir la résolution du compteur **TCNT1** avec les drapeaux **PWM11** et **PWM10** du registre **TCCR1A**.
2 - Chargement de la valeur de comparaison dans **OCR1x.**
3 - Sélectionner la polarité su signal à l'aide de **COM1X1** et **COM1X0**.
4 - Selection du facteur de division de l'horloge si necessaire.

Exemple :

Nous allons produire un signal **PWM** sur la sortie **OC1A** (PD5) de fréquence égale à 15 686 Hz, et de rapport cyclique variable à l'aide d'un potentiomètre branché sur **AIN0** (PA0).

Tout d'abord on configure le mode **PWM** normal, **COM1A1** = 0 et **COM1A0** = 0
Reste à définir la résolution du PWM, ce qui revient aussi à appliquer un facteur de pré-division car n'oubliez pas que TCNT1 compte j'usqu'a sa valeur maximum, puis décrémente et ainsi de suite. La valeur de la résolution sera fixée à 8 bits.

on peut donc calculer la fréquence centrale : $F = \frac{1}{Fquartz} * 510 = 63.75\mu s$

soit $f = \frac{1}{T} = \frac{1}{0.00006375} = 15686Khz$

510 correspond au facteur de pré-division par defaut pour le mode 8 bits (voir registre **TCCR1A**).

Ensuite on charge la valeur 128 dans le registres **OCR1A**, ce qui correspond à un rapport cyclique de 50 %

Et voila, il reste à faire varier la valeur **OCR1A** qui sera issue de la valeur du signal analogique échantillonné sur PA0.

Listing :

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale

Les registres

Les interruptions

Le Watchdog

Les entrées / sorties

Le comparateur Analogique

Le convertisseur Analogique

Les timers

L'UART

La SPI

La Ram et l'EEprom

Les soft de programmation

L'UART

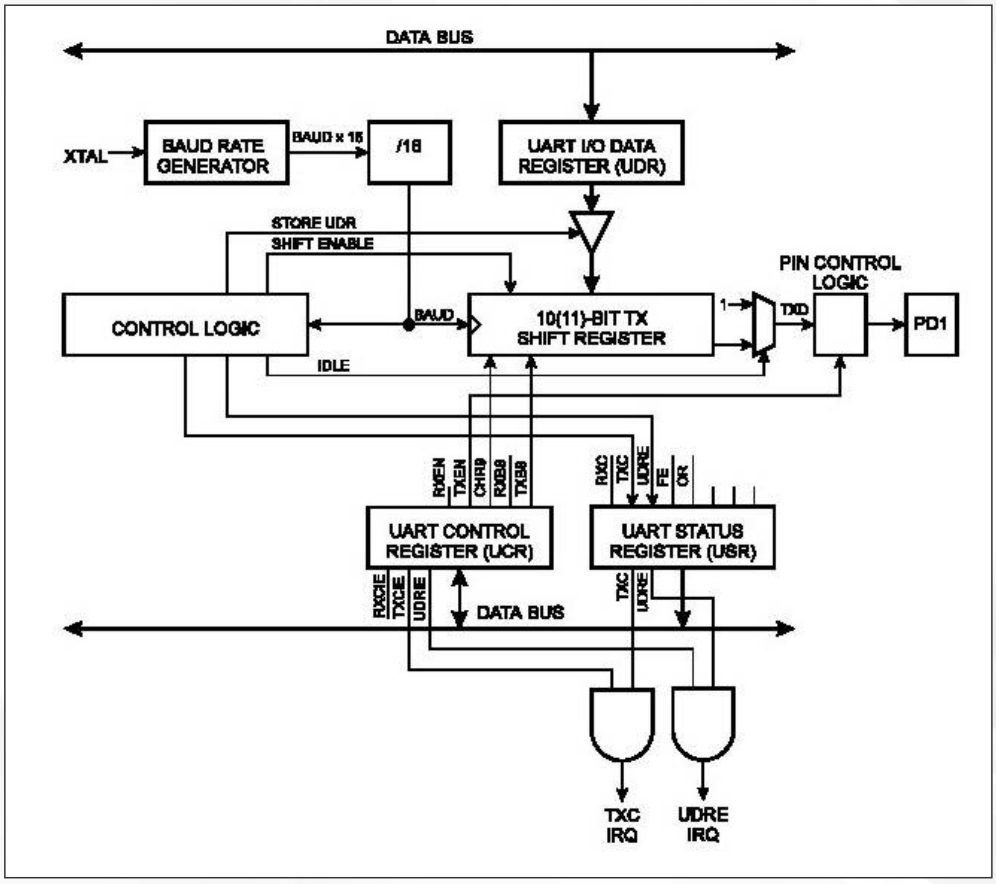
Présentation :

L'UART est l'abréviation de **U**niversal **A**synchronous **R**eceiver and **T**ransmitter , en francais on peut traduire et simplifier par Liaison Série Asynchrone.
Ce périphérique est plutot bien fournie sur l'ensemble de la famille Atmel 90S, pour preuve le résumé des caractéristiques :

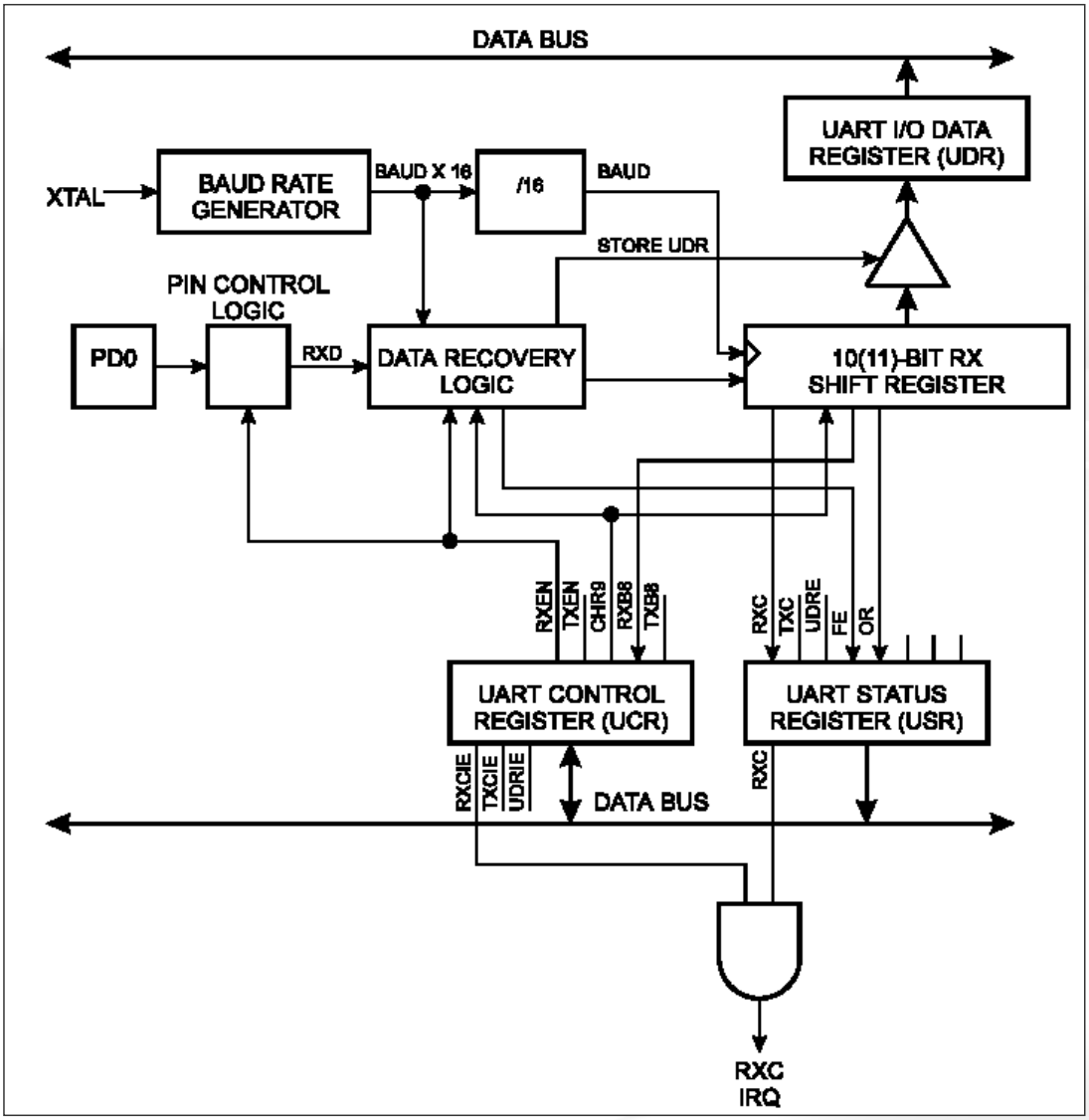
- Générateur interne de fréquence de cadencement
- Echanges de données sur 8 ou 9 bits
- Communication en Full-duplex (peut émettre et recevoir en meme temps)
- Protection des débordements
- Détection de faux départs de transmission
- Filtrage de l'entrée
- Plusieurs interruptions programmables sur le mode émission et réception

L'application principale de ce périphérique réside en la **communication entre le microcontroleur et un ordinateur** via le port série RS232.

Schématique de l'UART pour la partie transmission :



Schématique de l'UART pour la partie émission :



Les registres :

Ils sont aux nombres de quatre :

UDR : Registre de données de l'interface UART.

En fait ce registre est assez particulier, la gestion est transparente pour l'utilisateur, mais sachez que lors d'une opération d'écriture de données, ce qui lance le processus d'émission, on peut lire aussitot ce registre qui contiendra le tampon de réception.

Adresse	7	6	5	4	3	2	1	0
$0C	**MSB**							**LSB**

USR : Registre d'état de l'interface UART.

Adresse	7	6	5	4	3	2	1	0
$0B	**RXC**	**TXC**	**UDRE**	**FE**	**OR**			

- **RXC** : Passe à 1 lorsque le tampon de réception contient une donnée. Ce bit est automatiquement remis à 0 lors de la lecture du tampon correspondant, soit **UDR**.

- **TXC** : Passe à 1 lorsque la donnée contenue dans **UDR** à été transmise. Ce bit est automatiquement remis à 0 lors du traitement de l'interruption correspondante, ou alors il faut écrire un 1 par programme.

- **UDRE** : Passe à un lorsque la donnée écrite dans **UDR** est passée en phase d'émission. ce drapeau signale donc le déplacement du contenu de **UDR** vers le tampon d'émission. Repasse à 0 lors d'une nouvelle écriture de données dans **UDR**.

- **FE** : La levée de ce drapeau à lieu lors d'un defaut de reception, en particulier quand le bit de stop n'a pas été réalisé correctement. la remise à 0 de ce bit se fait automatiquement lors de la prochaine reception correcte.

- **OR** : Ce bit passe à un 1 lors du phénomène suivant : une donnée prècedement recue est disponible dans le registre UDR, cependant une autre donnée arrive dans le tampon de reception alors que **UDR** n'a toujours pas été lu. On peut donc dire qu'il s'agit d'un drapeau d'avertissement avant la future perte de donnée si rien est fait. La remise à 0 de ce bit est effectué automatiquement lors de la lecture du registre **UDR**.

UCR : Registre de Controle de l'interface UART.

Adresse	7	6	5	4	3	2	1	0
$0A	**RXCIE**	**TXCIE**	**UDRIE**	**RXEN**	**TXEN**	**CHR9**	**RXB8**	**TXB8**

- **RXCE** : Valide l'interruption **UART_RXC** lorsque le bit **RXC** du registre **USR** passe à 1.

- **TXCIE** : Valide l'interruption **UART_TXC** lorsque le bit **TXC** du registre **USR** passe à 1.

- **UDRIE** : Valide l'interruption **UART_DRE** lorsque le bit **UDRE** du registre **USR** passe à 1.

- **RXEN** : La mise à 1 de ce bit valide la partie reception de l'UART. La broche PD0 (RXD) est alors basculé en entrée.

- **TXEN** : La mise à 1 de ce bit valide la partie émission de l'UART. La broche PD1 (TXD) est alors basculé en sortie.

- **CHR9** : Séléction du format de réception/émission des données : 1 = 9 bits et 0 = 8 bits.

- **RXB8** : Valeur recue du 9 ème bits, uniquement en cas de mode de transmission sur 9 bits.

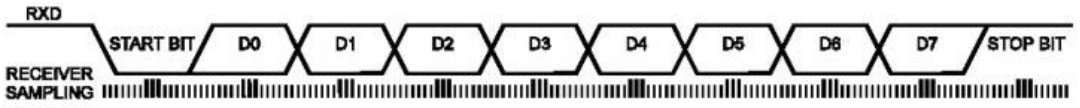
- **RXB8** : Valeur du 9 ème bits à émettre, uniquement en cas de mode de transmission sur 9 bits.

UBRR : Registre de selection de la vitesse de cadencement de l'UART.

Adresse	7	6	5	4	3	2	1	0
$0B	**UBRR7**	**...**	**...**	**...**	**...**	**...**	...	**UBRR0**

- UBRR7...**UBRR0** : Valeur permettant de sélectionner la fréquence de cadencement de l'UART.

Il est à signaler avant tout que la fréquence de cadencement ne doit pas forcément correspondre exactement aux normes en vigueur, tout système à une tolérance concernant les fréquences d'émissions et réceptions.
D'ailleurs le microcontroleurs lui meme effectue un triple échantillonage du signal a un instant de demi flanc, ce qui confirme que le système est insensible aux fréquences qui dérivent autours de la normes, et permet de limiter les erreurs dues aux parasites.



La valeur du registre UBRR peut etre calculée suivant la formule suivante :

$$UBRR = \frac{Frequence.Quartz}{16 * Vitesse.souhaitee.en.baud} - 1$$

Tableau contenant les valeurs de UBRR en fonctions de la valeur du Quartz :

On remarquera que les valeurs en Gras indique la compatibilité de la combinaison. Les valeurs sont bien sur données en décimal.

Vitesse (Bauds)	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=	d	25

Vitesse (Bauds)	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=	14	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

Vitesse (Bauds)	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**

Application :

Paramétrage de l'UART :

Voici les principaux mode de fonctionnement :

Initialisation	Emission	Reception
Mise en marche et validation de L'Uart avec **UCR** Configuration de la vitesse de L'Uart avec **UBRR**	RAZ de TXC dans **USR** Emission de la donnée contenue dans **UDR** Attente de TXC = 1 pour fin d'emission	Attente de RXC = 1 pour fin de réception lecture de la donnée contenue dans **UDR**

Exemple :

Nous allons tout simplement envoyé le mot "Ok" sur le PC.
Le driver de bus utilisé sera un MAX232
Le programme ' Hyperterminal ' fourni avec windows fera très bien l'affaire pour ce test ; Quelques reglages seront necessaires :

Une fois connecter la fenetre d'hyperterminal se remplira de "Ok" à n'en plus finir

Le programme en assembleur ne demande pas d'explications particulières.

Listing :

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale

Les registres

Les interruptions

Le Watchdog

Les entrées / sorties

Le comparateur Analogique

Le convertisseur Analogique

Les timers

L'UART

La SPI

La Ram et l'EEprom

Les soft de programmation

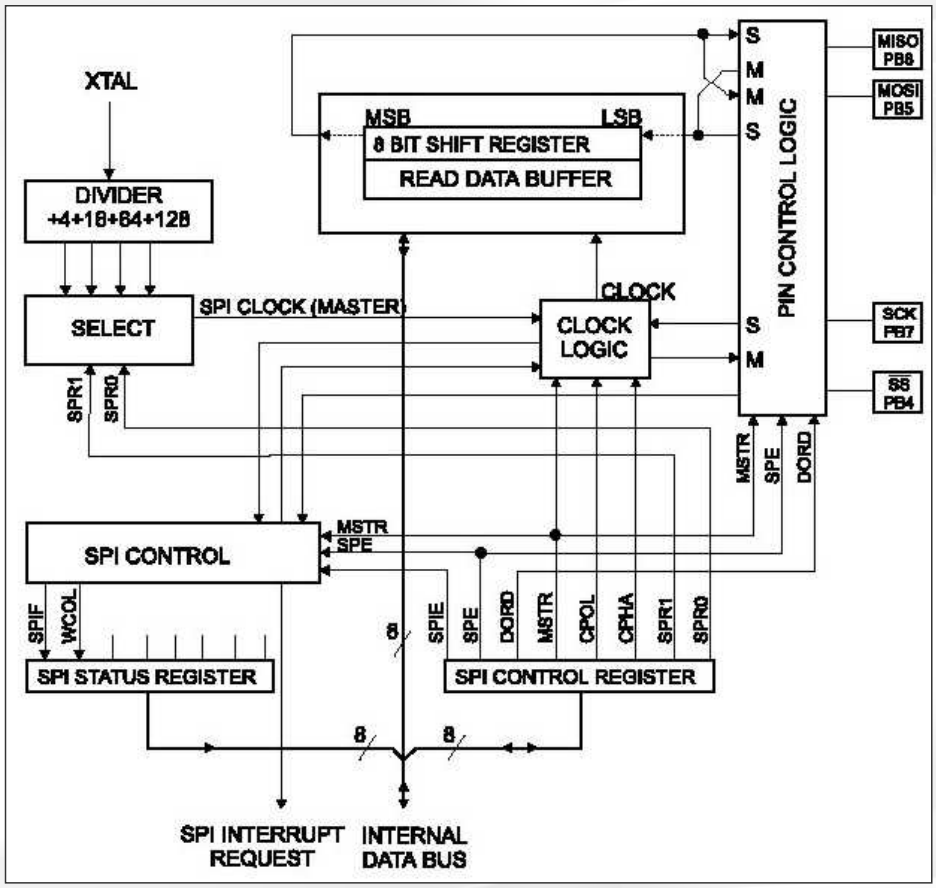
La SPI

Présentation :

LA SPI est l'abréviation **S**erial **P**eripheral **I**nterfacel **A**synchronous , soit Liaison Série Asynchrone.
Contrairement à l'UART, et comme son nom l'indique, ce type de périphérique génère les signaux d'horloge, arbitré par le maitre.
Ce type de liaison nécessite donc de définir un maitre et un esclave. Sur les microcontroleurs ATMEL, ce choix est fonction de l'état de la broche " /SS".

Les applications sont multiples, comme la gestion d'une communication I2C, ou encore l'interfacage avec nombres de périphériques spécialisés comme les capteurs de température Dallas, les EEPROM séries.

Schématique de la SPI :



Description des Broches :

La SPI comporte quatre broches :

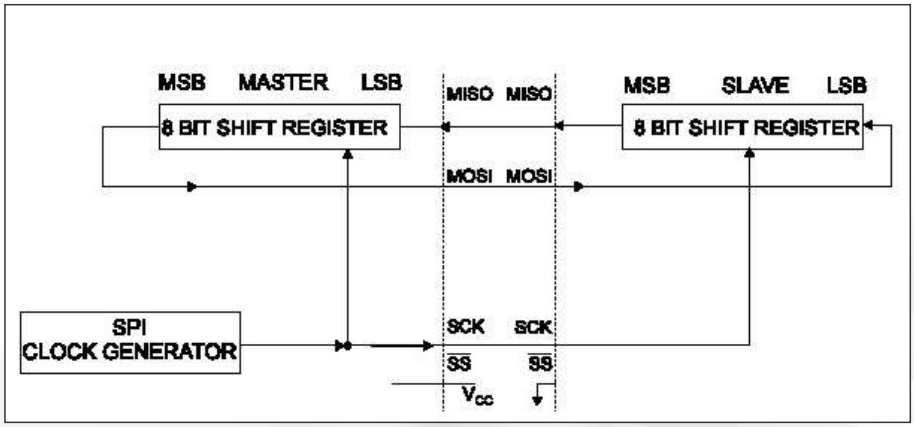
MOSI : Master **O**ut **SI**ave**,** sortie des données série pour le maître est entrée pour l'esclavage.

MISO : **M**aster **I**n **S**lave **O**ut, entrée des données série sur le maître sorties sur l'esclave.

SCK : **S**erial **C**loc**K**, broche d'horloge pour synchroniser les circuits connectés sur le Bus. Ces signaux d'horloge sont générés par le maître.

SS: broche de sélection du mode de fonctionnement du circuit. En mode maître elle doit être au niveau Haut.

Exemple de raccordement entre microcontrôleurs :



Les registres :

SPCR : Registre de contrôle de l'interface SPI.

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

- **SPIE** : Mis en marche de l'interface SPI.

- **SPE** : Validation d'interruption SPI lors du passage à 1du bit **SPIF** du registre.

- **DORD** : Choix de transmission de l'ordre des bits : Si à 1, me bit de poids faible et émis en premier, à l'inverse pour 0.

- **MSTR** : Mode de fonctionnement de l'interface : Si à 1, mode maître, à l'inverse pour 0.

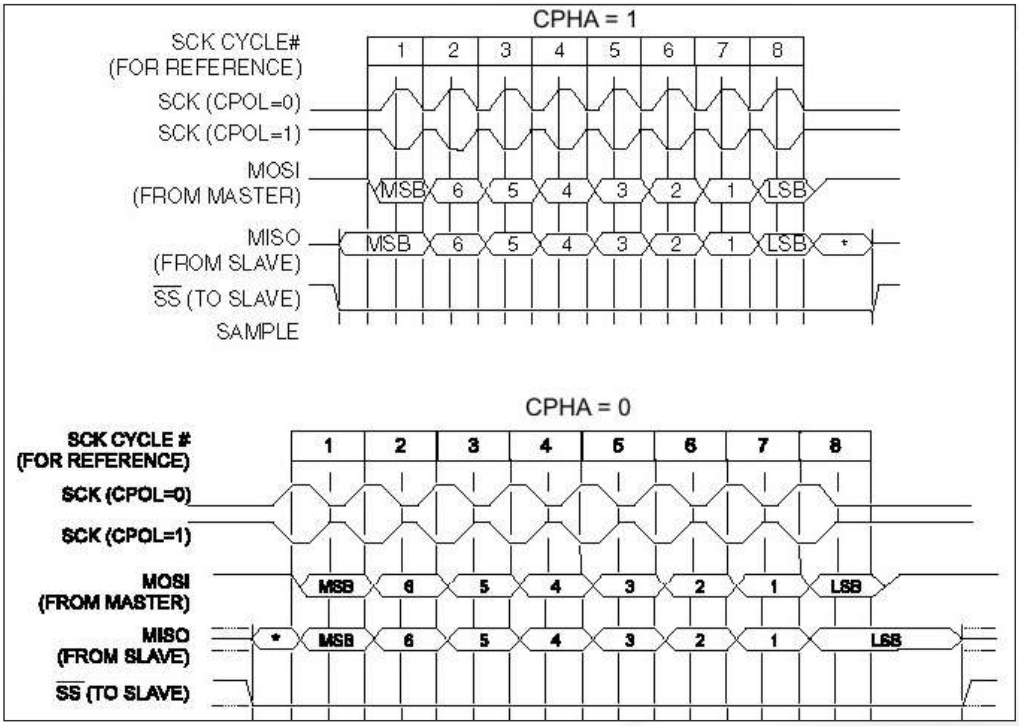
- **CPOL** : Sélection de la polarité de l'horloge : Si à 1, repos de l'horloge au niveau haut, à l'inverse pour 0.

- **CPHA**: Sélection de la phase des données par rapport à l'horloge : si à 1,les données changent de front quand le signal d'horloge est au niveau de repos, à l'inverse pour 0.

- **SPR1,SPR2** : Sélection de la fréquence de transmission, définit par le maitre.

SPR1	SPR2	Facteur de pré-division
0	0	Fck / 4
0	0	Fck / 16
1	0	Fck / 64
1	1	Fck / 128

Récapitulatif des mode de fonctionnement de la SPI :



SPSR : Registre de Statut.

Adresse	7	6	5	4	3	2	1	0
$0E	**SPIF**	**WCOL**						

- **SPIF** : Indication de fin de transmission. de l'octet.

- **WCOL** : Bit d'Indication de collision de données, provoqué par l'ecriture dans le tampon **SPDR** alors que le transfert en cours n'est pas terminé.

La remise à 0 de ce bit est effectué en lisant en premier lieu le registre **SPSR** quand ce bit est à 1, puis le registre **SPDR**.

SPDR : Registre de Données.

Adresse	7	6	5	4	3	2	1	0
$0E	**RXCIE**	**TXCIE**	**UDRIE**	**RXEN**	**TXEN**	**CHR9**	**RXB8**	**TXB8**

En écrivant dans ce registre, le début de la transmission est lancé. Il contient aussi le résultat de la dernière reception.

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale

Les registres

Les interruptions

Le Watchdog

Les entrées / sorties

Le comparateur Analogique

Le convertisseur Analogique

Les timers

L'UART

La SPI

La Ram et l'EEprom

Les soft de programmation

La Ram et L'EEprom

Présentation :

L'EEprom est une mémoire programmable est effaçable électriquement. La particularité de cette mémoire et de pouvoir garder les informations stockées même hors tension.
On imagine immédiatement les applications réalisables avec un tel élément, comme la sauvegarde d'un point de consigne température sur un régulateur, et tous les éléments qui permettent une reprise du programme après une coupure de tension.

Les registres :

EEARH et EEARL : Registres d'adressage de l'EEPROM

Adresse	7	6	5	4	3	2	1	0
$1F	**EEARH7**							**EEARH0**

Adresse	7	6	5	4	3	2	1	0
$1E	**EEARL7**							**EEARL0**

A noté que le registre **EEARH** est seulement présent sur les microcontrôleurs dotés de plus de 256 Octets de mémoire.

EEDR : Registre de données de l'EEPROM

Adresse	7	6	5	4	3	2	1	0
$1D	**ADEN**	**ADSC**	**ADFR**	**ADIF**	**ADIE**	**ADPS2**	**ADPS1**	**ADPS0**

Ce registre contient les données de l'EEPROM après une lecture, ou bien les données à écrires

EECR : Registres de contrôle de L'EEPROM

Adresse	7	6	5	4	3	2	1	0
$1C					**EERIE**	**EEMWE**	**EEWE**	**EERE**

- **EERIE** : Validation de l'interruption **EE_RDY**, se produisant quand une opération de lecture ou écriture est terminée.

- **EEMWE** : Autorisation d'écriture dans la mémoire (voir procédure).

- **EEWE** : Seconde autorisation d'écriture dans la mémoire (voir procédure).

- **EEMWE** : demande de lecture de la mémoire EEPROM.

Application :

Procedure de lecture / écriture dans l'EEPROM :

bien que l'utilisation de l'EEPROM à l'aide des registres reste simple, une procédure particulière est effectuée avant toute demande d'écriture dans cette mémoire, dans le but de sécuriser au maximum le bon fonctionnement de celle-ci :

Procédure d'écriture :

- Test du bit : **EEWE** = 0 afin de savoir si une procédure d'écriture est en cours.
- Ecrire l'adresse où l'on souhaite écrire la donnée dans les registres **EEARH et EEARL**.
- Ecrire la donnée dans le registre EEDR
- Mettre à 1 le bit **EEMWE**
- Mettre à 1le bit **EEWE** dans un délai inférieur à quatre cycles d'horloge sous peine d'annuler l'écriture.

Procédure de lecture :

- Test du bit : **EEWE** = 0 afin de savoir si une procédure d'écriture est en cours.
- Ecrire l'adresse où l'on souhaite accéder à la donnée dans les registres **EEARH et EEARL**.
- Mettre à 1 le bit **EERE**
- Lecture de la donnée dans le registre **EEDR**.

Exemple :

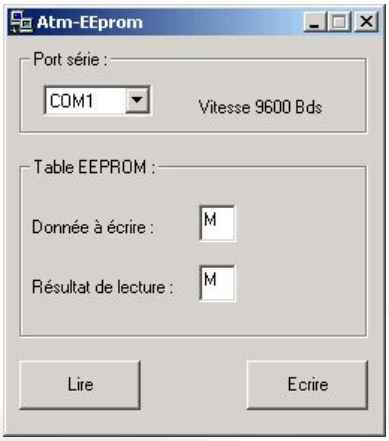
le petit projet présenté ici, nous permettra de lire ou d'écrire un caractère numérique ou alphanumérique à l'adresse $01 de l'eeprom à l'aide d'un petit programme (section téléchargement)

On s'aperçoit à l'aide du listing ci-après du fonctionnement du programme :

Dans un premier temps le microcontrôleur se met obligatoirement en réception et attend qu'un caractère qui désignera un ordre de lecture ou d'écriture de la part du programme informatique.

- La réception du caractère alphanumérique "E" indique un ordre d'écriture dans l'eeprom, caractère suivi de la donnée à inscrire.Le microcontrôleur saute alors au sous programme ecr01 et inscrit la donnée dans la mémoire EEPROM.- La demande de lecture d'un caractère est signalée au microcontrôleur par l'envoi du caractère "L" sur le port série. Celui-ci effectue un saut à la routine lir01 et envoie immédiatement après le caractère lu dans la mémoire.

Le programme informatique par lui-même de demande pas d'explication supplémentaire étant donné sa simplicité.
Il est juste et nécessaire de sélectionner le numéro de port série.



Après coupure electrique du montage, on peut vérifier à nouveau que le caractère inscrit en mémoire soit toujours présent.

Listing :

Cours Atmel AT90Sxxxx

NAVIGATION

Architecture générale

Les registres

Les interruptions

Le Watchdog

Les entrées / sorties

Le comparateur Analogique

Le convertisseur Analogique

Les timers

L'UART

La SPI

La Ram et l'EEprom

Les soft de programmation

La programmation software

Il est temps maintenant d'aborder la présentation des logiciels permettant de programmer nos microcontrolleurs.
Je ferais une présentation sommaire des logiciels les plus connus et gratuits, tous disponible en section téléchargement.

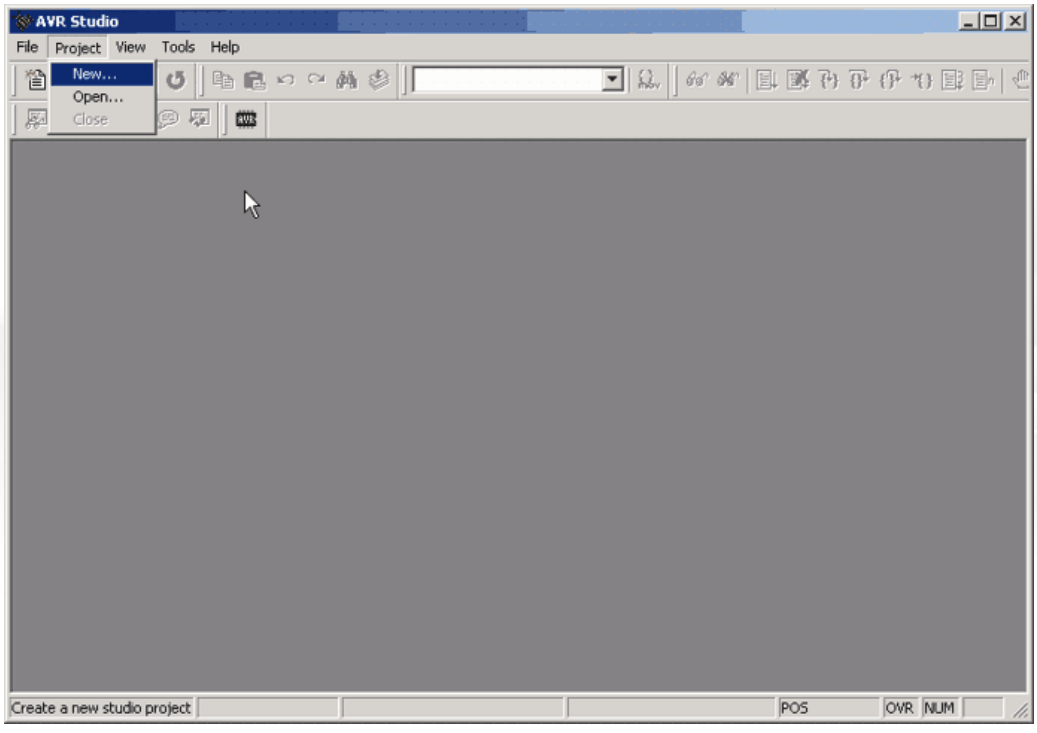
AVR STUDIO

PONY PROG

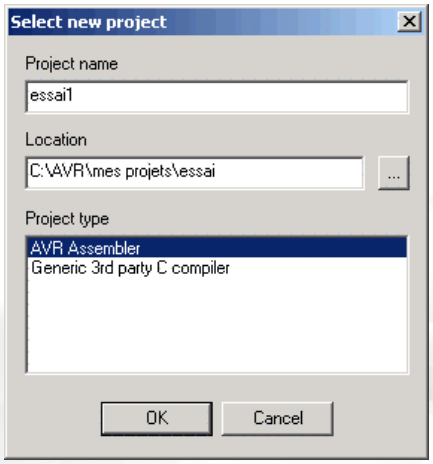
Voici l'un des plus célèbre logiciel de programmation. En effet AVR STUDIO est un éditeur, assembleur, simulateur et programmateur/débogueurs (pour les platines de développements STK 501,500...).
Il est évident que le simulateur n'est pas aussi perfectionné que les logiciels professionnels, mais il permet de débusquer de bonnes erreurs de programmation.
Pour ma part j'utilise la version 3.53 qui me semble la plus pratique.

Dans un premier temps il faut créer un repertoire de travail dans lequel nous déciderons d'inclure les fichiers d'un nouveau projet, remarquez au passage l'appellation "projet", qui determine un ensemble de fichiers (le résultat de la compilation, la configuration des outils de simulations...).
Donc on peut créer un repertoire comme : C:\AVR\mes projets\essai\

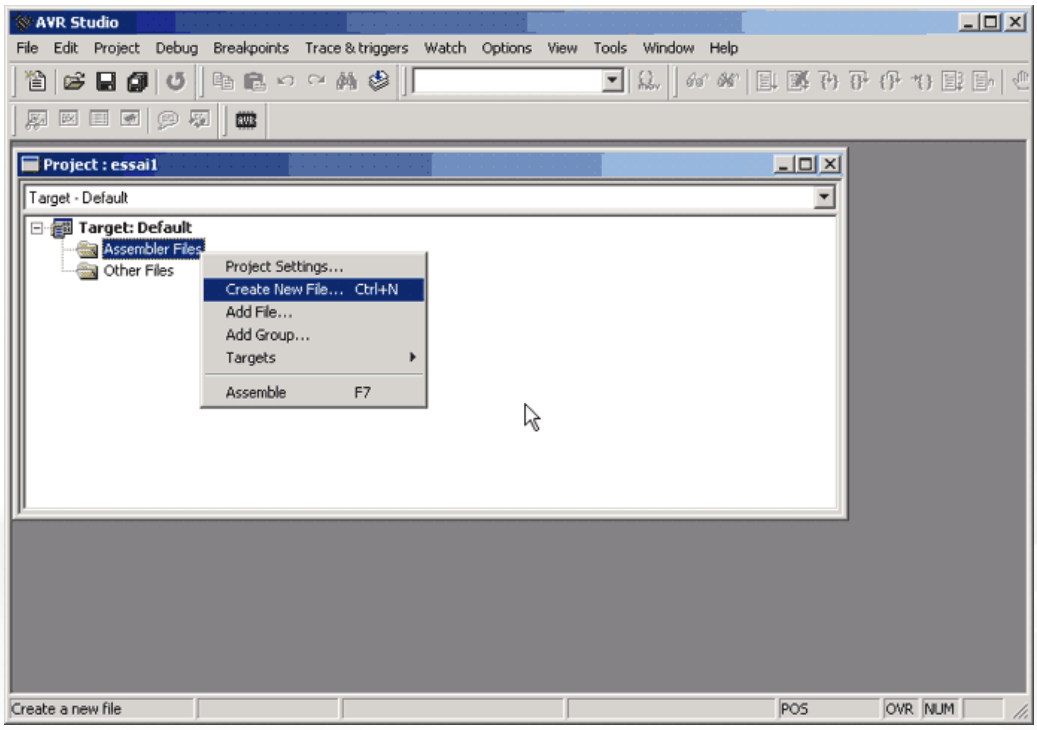
On lance AVR STUDIO et on choisi dans l'onglet PROJECT > New



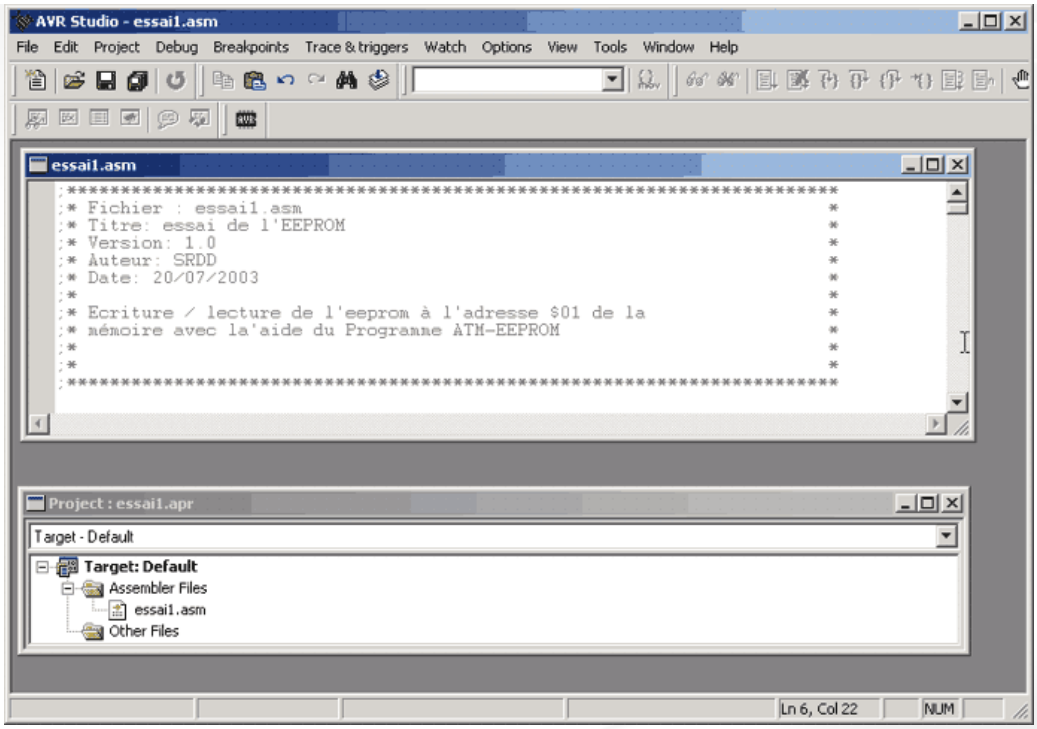
Ensuite s'ouvre une nouvelle fenêtre : entrer le nom du fichiers projet, sa destination et sélectionner le type : ici AVR assembler



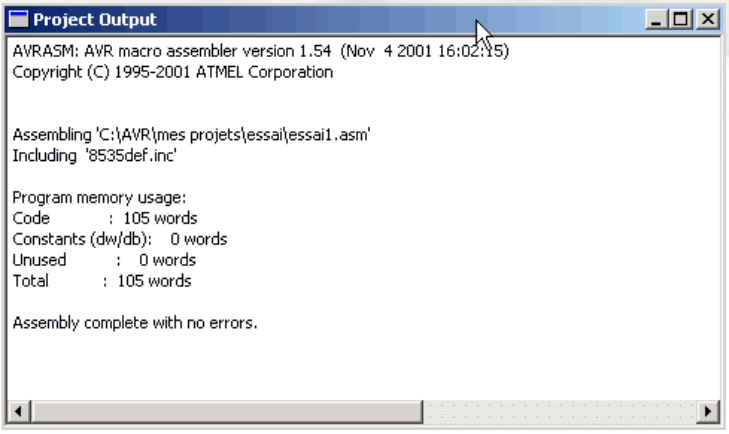
En validant, on aperçoit une fenêtre client :
cliquer à droite sur Assembleur Files, une multitude de solutions s'offre à vous : on peut créer un nouveau fichier ASM vide, ou en sélectionner un déjà existant (fichiers de programmations type), dans ce cas pensez à le déplacer dans le repertoire de travail.
Si un nouveau fichier est créé, AVR studio en demande le nom, terminez l'extension du fichier par .asm



On retrouve la fenêtre suivante :
Dans essai1.asm, j'ai tout simplement effectué un copier collé pour récupérer la partie des déclarations en assembleur.

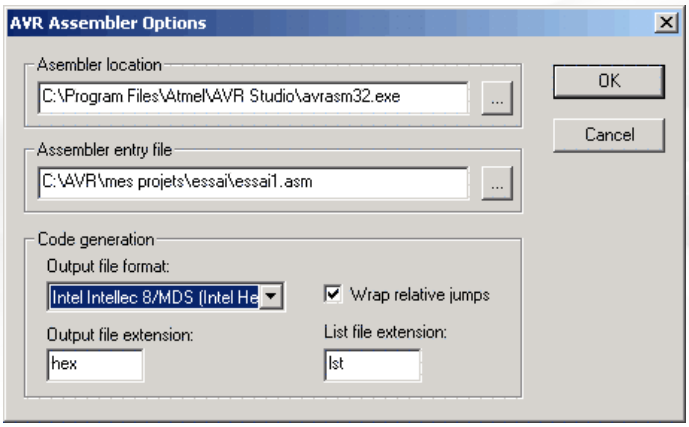


On peut maintenant écrire son programme, et le compiler en appuyant sur la touche F7 ou encore dans le menu Project > Assemble.
Si tout se passe bien et dans le meilleur des mondes, nous devrions avoir le message suivant :



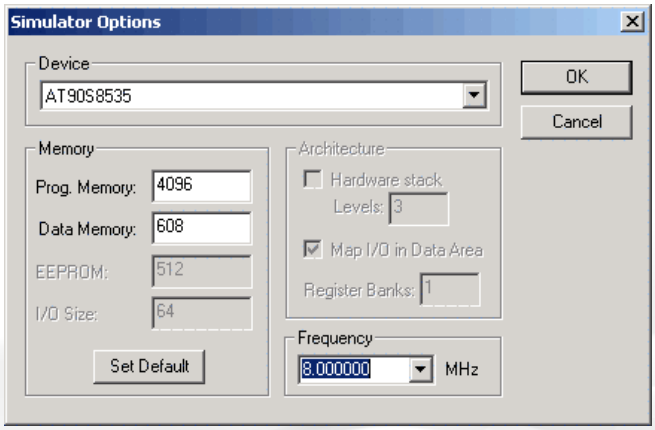
Dans le cas contraire le numéro d'erreurs s'affiche, ainsi qu'une descriptions et le numéro de ligne à laquelle la faute est détectée.
N'oubliez pas d'inclure dans le repertoire de travail le fichiers de définition concerné, ici : 8535def.inc

Parfait ! , mais le fichiers assembleur généré à pour extension *.lst ce qui nous intéresse pas, il faut donc régler l'assembleur pour qu'il généré un fichiers *.hex, pour ce faire cliquer : Project > Project Settings et choisissez "**Intel Intellec 8/MDS**" , vient s'afficher par défaut l'extension hex, parfait.



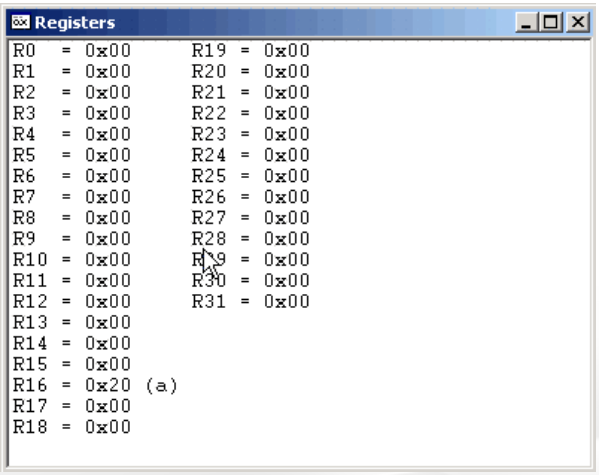
Une fois le programme ne comportant aucune erreur, passons a la simulations ; Appuyons sur ' Ctrl F7' , ou encore Project > Build and Run. Une nouvelle fenêtre s'affiche nous demandant plusieurs renseignements : Le type de microcontroleur et la fréquence de fonctionnement. Pour ce qui est de la taille mémoire et la Data memory, ne touchez à

rien.

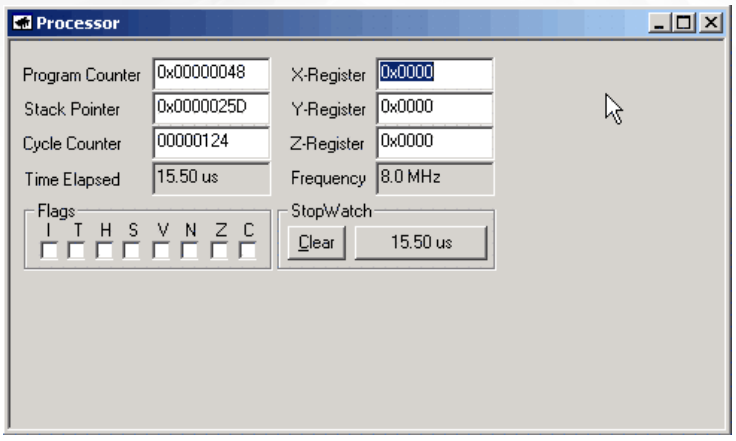


Maintenant nous pouvons simuler notre programme de plusieurs manières, pas à pas (F11),automatique... Regardez pour cela le menu Debug.
Il nous faut ajouter maintenant les paramètres à surveiller avec View , plusieurs outils sont disponibles, voici les principaux :

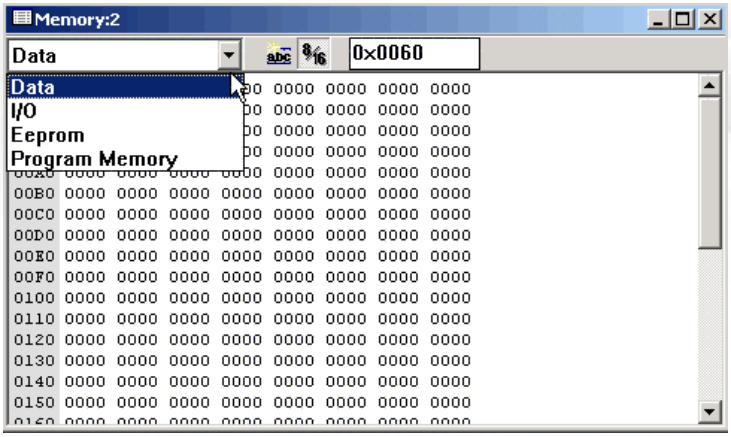
Registers :
Il s'agit de la valeur hexadecimal des registres principaux (R0 à R31). On peut modifier la valeur d'un registre en cliquant dessus, ou bien encore de le nommer : il est beaucoup plus pratique de scruter les informations par leurs nom plutôt que par leurs numéros de registre; Pour se faire : clique droit sur le registre, puis Add Comment pour rentrer le nom du registre. Dans mes programmes je me sers souvent du registre R16 en tant qu'accumulateur 'a', je l'ai donc noté ainsi. (les veilles manières avec le 8051 ne se perdent pas !)



Processors :
Ici on trouve le contenu de plusieurs registres 'système ainsi que le temps écoulé depuis le lancement de la simulations.'



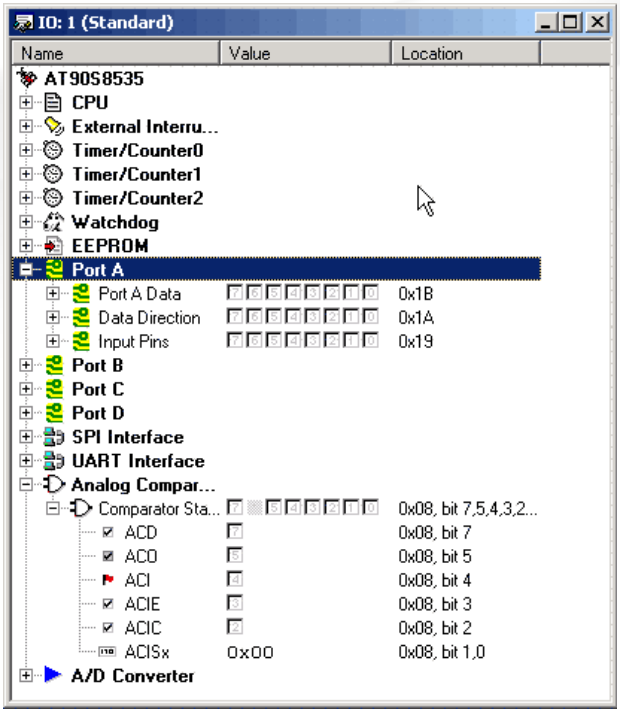
New Memory view :
Permet de visualiser les valeurs contenues dans la Sram, EEprom...



ATTENTION :

AVR STUDIO ne charge pas automatiquement le fichier EEprom : si vous avez créé des déclaration de segment avec . db ou autre dans l'espace EEprom, il faut charger manuellement le fichier nomfich.epp à l'aide de la procédure suivante : Dans le menu DEBUG, choisir UP/Dowload Memories, sélectionner le type de mémoire; Ici EEprom, parcourir pour charger le fichier .epp, et terminer le processus avec le bouton load and program.

New Memory view :
Tous les registres matériels sont représentés ici. On peut développer chacun d'entre eux et modifier leurs valeurs, bien pratique pour simuler une fin de conversion, ou l'arrivé d'un caractère sur le port série



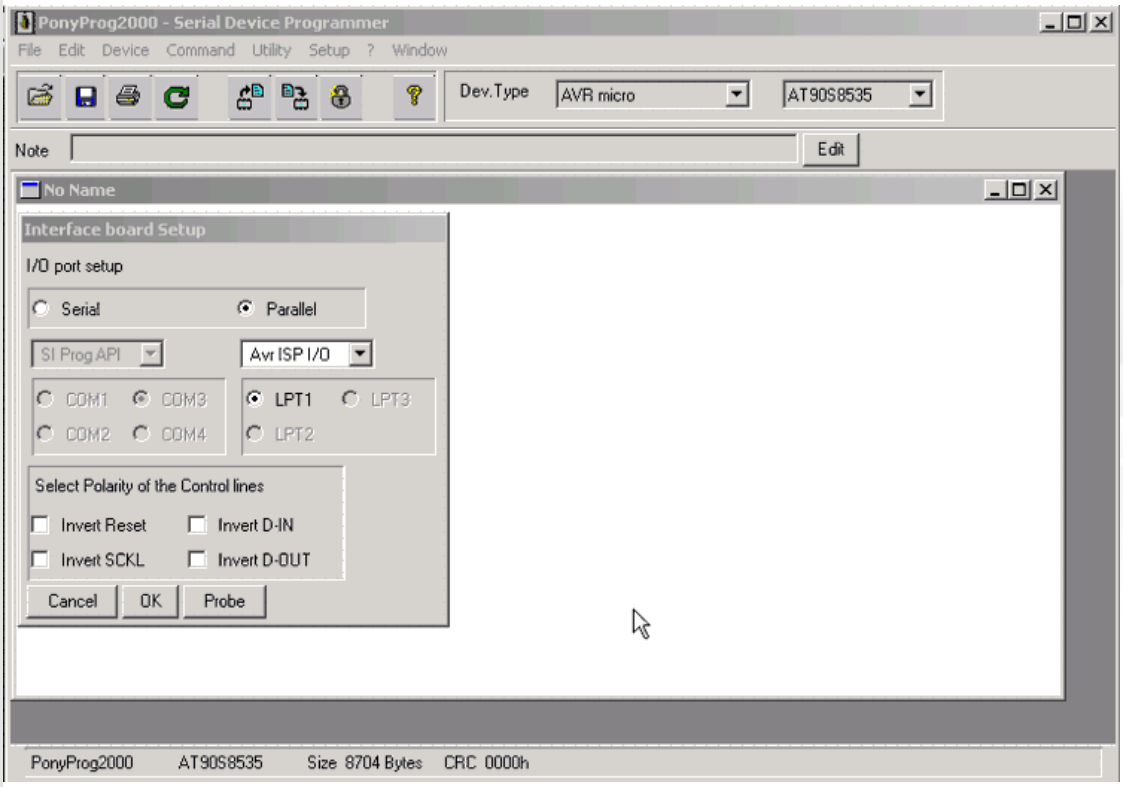
Voila, je pense qu'avec cette présentation de prise en main rapide d'AVR studio vous êtes maintenant capable de l'utiliser sans aucune difficultés...

Pony Prog est un logiciel de programmation In-situ dont l'utilisation ne pose vraiment aucune difficulté.
Capable de programmer les microcontroleurs Atmel, il est aussi possible de réaliser différentes interfaces, dont les plans sont joint avec le logiciel, afin de programmer une multitude de composant comme des EEPROM séries, les PICs....

La seule chose importante à effectuée au premier lancement du logiciel est la calibration de l'interface, mécanisme

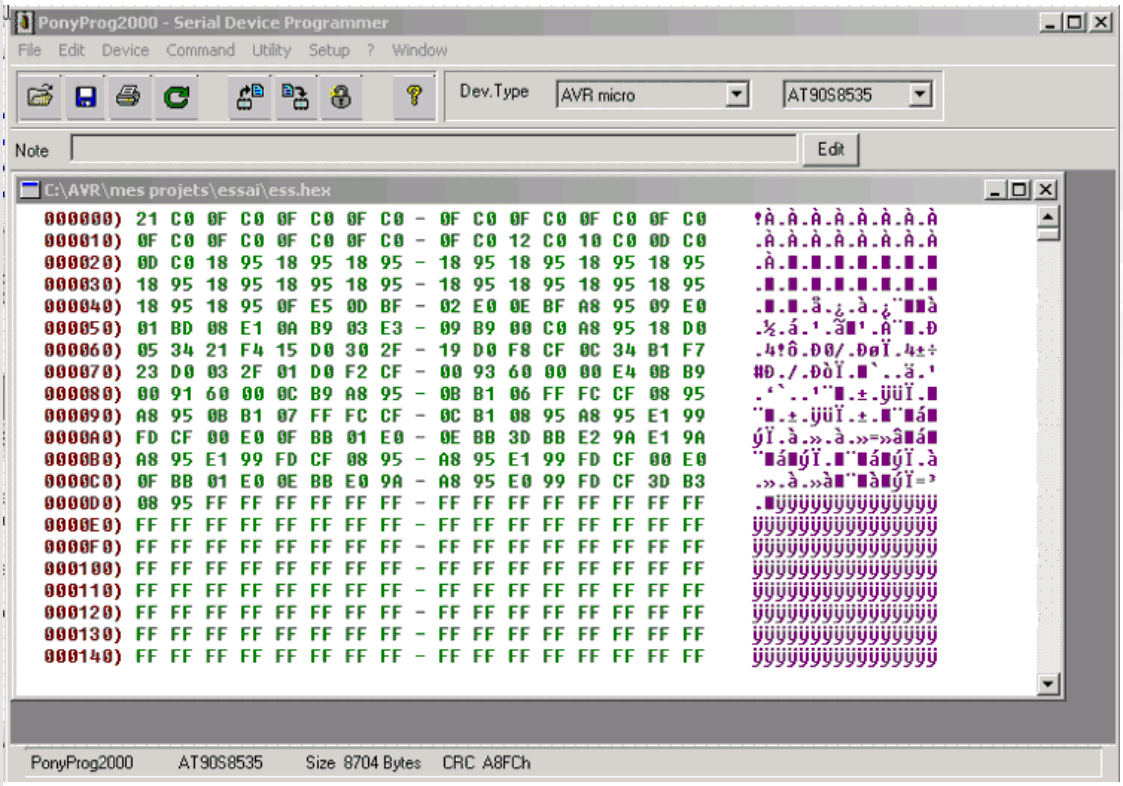
totalement automatique mais nécessaire, Stop, avant toute chose je vous conseille de baisser le volume de vos enceintes, a vous de voir... , vous saurez pourquoi !!!

On lance PONY Prog et on choisi dans l'onglet SETUP > Interface Setup



Choisissez le type de conection (dongle sur port parallèle), le numéro de port ainsi que le type de protocole, ici : AVR ISP I/O.
Valider le tout, ré-ouvrer le menu SETUP, et selectionné Calibration.

Reste plus qu'a programmer le microcontroleur.
Ouvrer le menu OPEN, puis Open Programm (flash file), parcourir pour trouver le fichier .hex.
Maintenant on peut visualiser notre programme :



Ensuite dans le menu Command, on execute Write All, ce qui programme la flash du microcontroleur.

A noter la possibilité de programmer la memoire EEprom, ou encore les "fusible", ou bien d'autres fonctions que vous découvrirez, mais ne posant aucun soucis majeur de compréhension.