



Architecture des systèmes embarqués et IOT

Chapitre 2: Arduino & microcontrôleur atmega328p

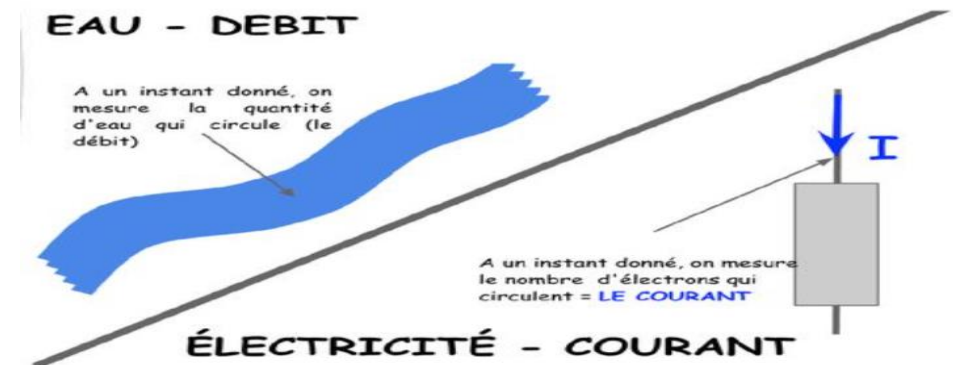
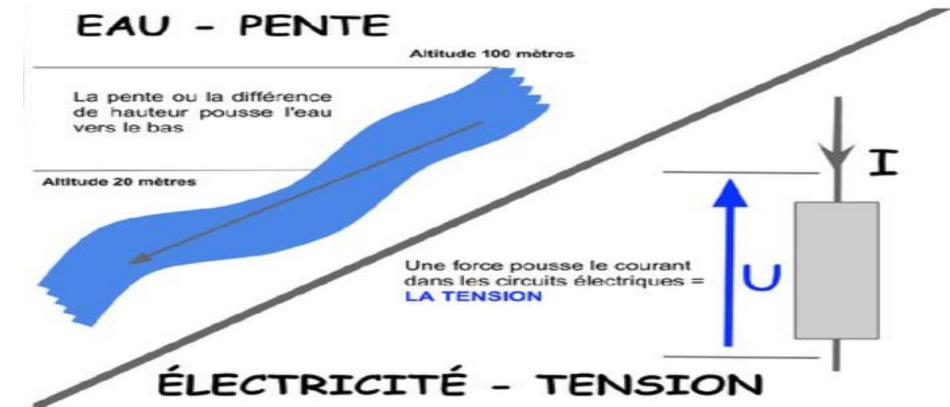
Objectifs

I

- Introduction aux systèmes embarqués et microcontrôleurs
- Architecture d'un microcontrôleur
- Programmation
- Interruptions
- Périphériques internes

Notions de Base

- **La tension électrique**
 - La tension électrique est la circulation du champs électrique le long d'un circuit électrique (volts)
- **Le courant électrique**
 - Un courant électrique est un déplacement des électrons, au sein d'un matériaux conducteur.
 - Ces déplacement sont imposés par l'action de la force électromagnétique (mesurée en Ampère)



Notions de Base

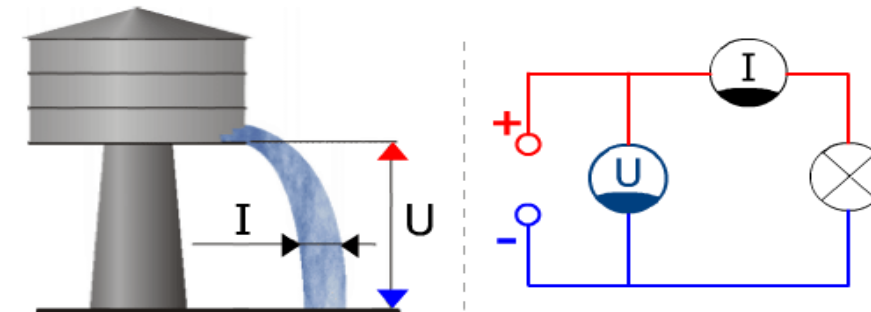
- La résistance
 - On appelle la résistance la propriété à retarder le passage du courant électrique dans un conducteur



- La puissance
 - La puissance est la quantité d'énergie par unité de temps.
 - La puissance correspond donc à un débit

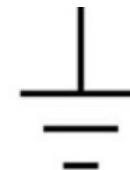
- VCC

- Vcc est la valeur de tension du circuit électronique

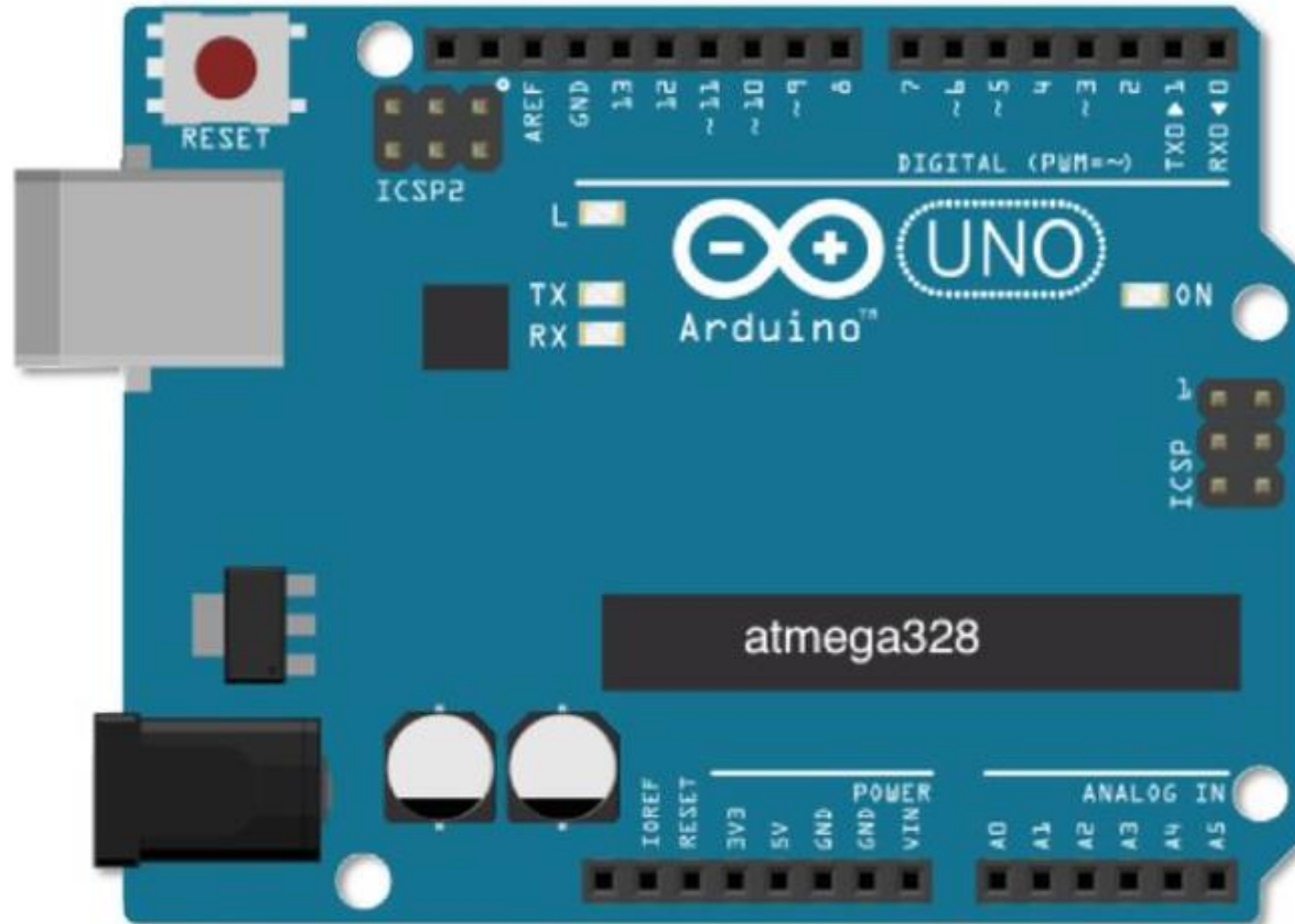


- Masse

- La masse c'est le référentiel. En électronique considère la masse d'un montage comme étant le zéro volt 0v

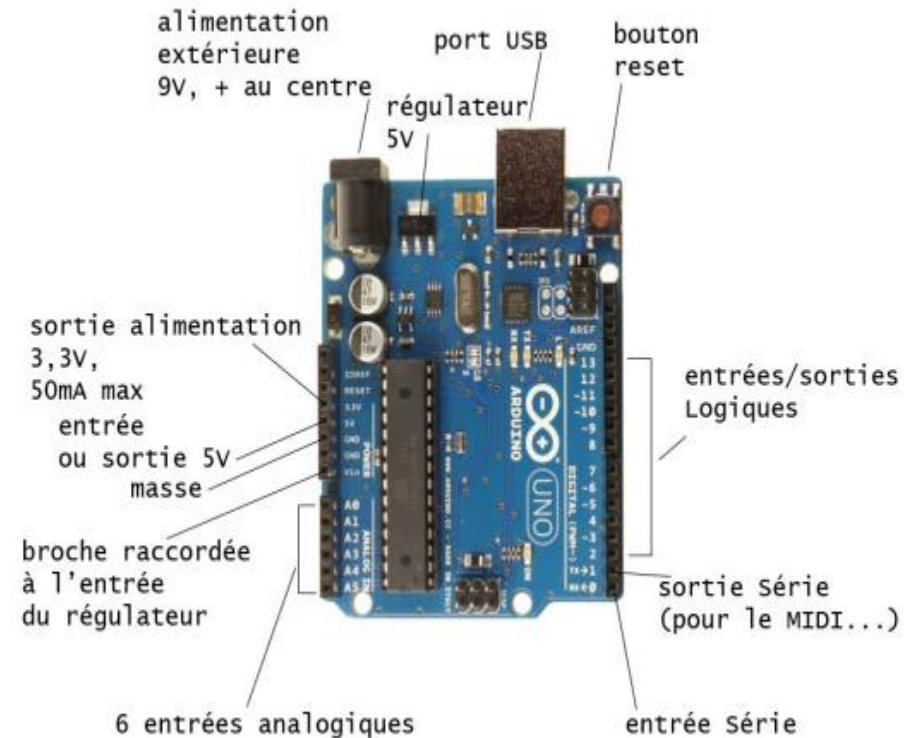


Arduino UNO



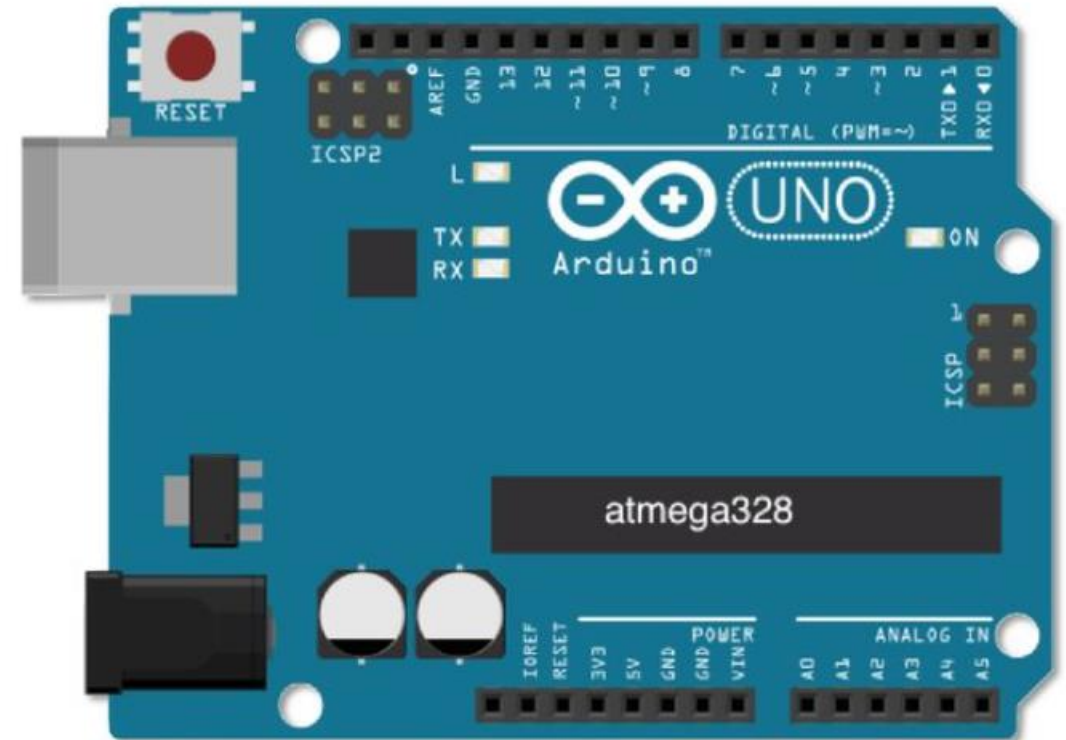
Carte Arduino

Arduino est une plate-forme de prototypage d'objets interactifs à usage créatif constituée d'une carte électronique et d'un environnement de programmation. Sans tout connaître ni tout comprendre de l'électronique, cet environnement matériel et logiciel permet à l'utilisateur de formuler ses projets par l'expérimentation directe avec l'aide de nombreuses ressources disponibles en ligne. Pont tendu entre le monde réel et le monde numérique, Arduino permet d'étendre les capacités de relations humain/machine ou environnement/machine. Arduino est un projet en source ouverte : la communauté importante d'utilisateurs et de concepteurs permet à chacun de trouver les réponses à ses questions.



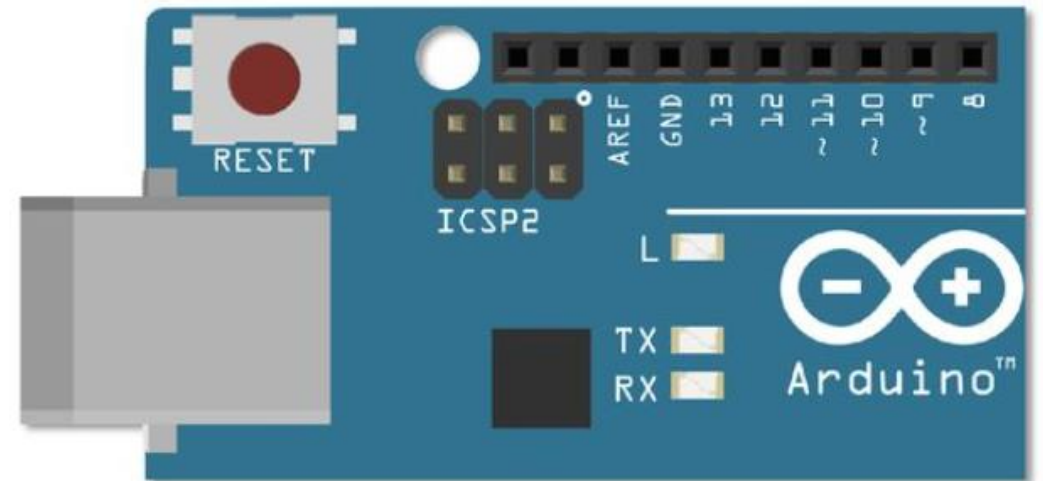
Principe de fonctionnement de la carte Arduino

- Le principe de » fonctionnement est simple :
 1. On réalise le programme sur un ordinateur
 2. On connecte l'Arduino à l'ordinateur via USB
 3. On envoie le code binaire vers le microcontrôleur de l'Arduino
 4. L'Arduino exécute le code en boucle



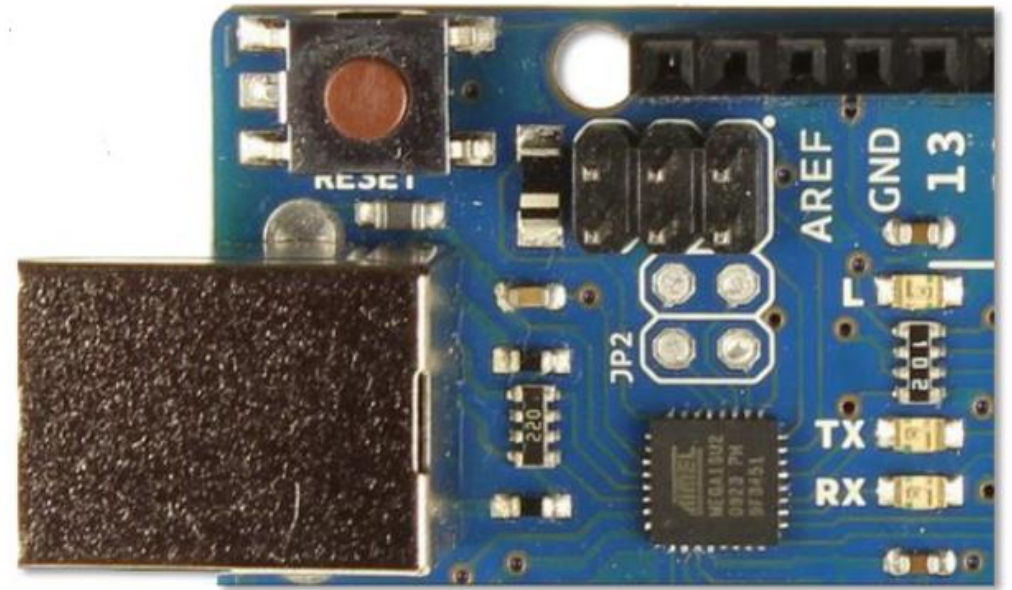
USB

- Cette partie de la carte est composée de 4 circuit
 1. Circuit Reset
 2. Convertisseur USB – UART
 3. Circuit de programmation du microcontrôleur atmega16u2
 4. Circuit de signalisation



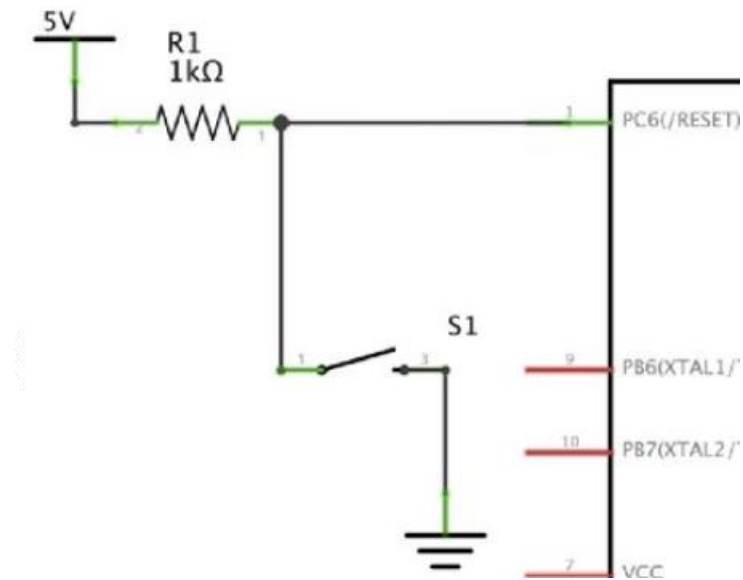
FTDI

- La carte Arduino se diffère des autres cartes de développement car elle utilise le circuit FTDI USB-vers série. A la place, elle utilise un Atmega8U2 ou Atmega16U2 programmer en convertisseur USB vers série



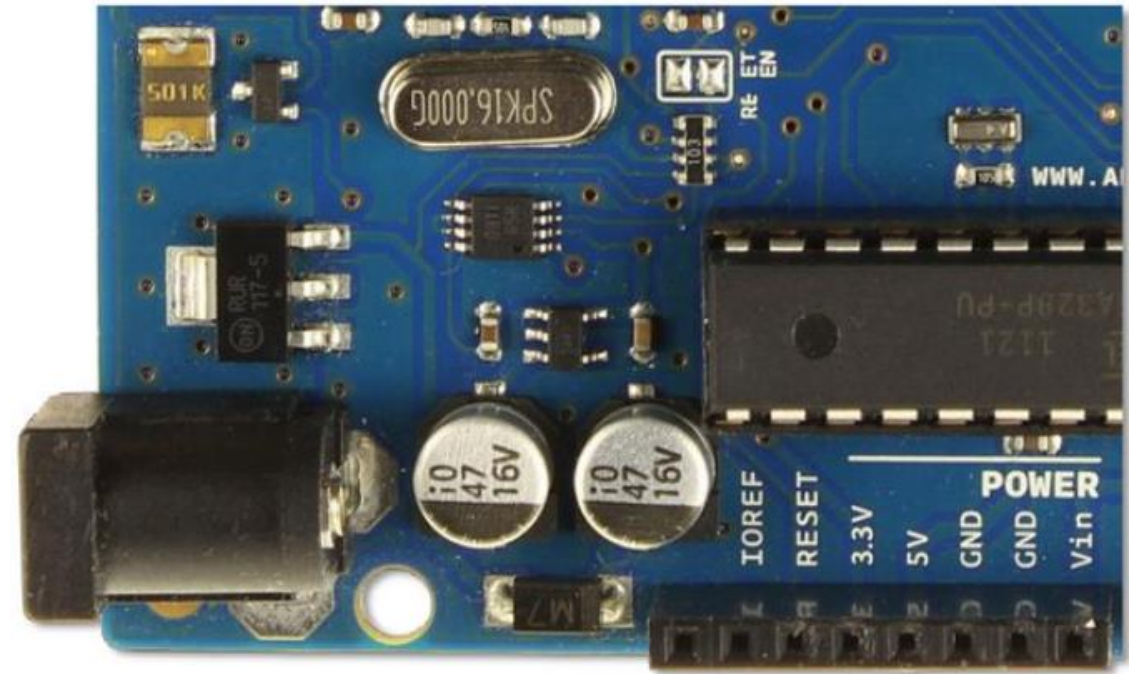
RESET

- Le pin 1 doit être connecté au vcc à travers une résistance de 10K



POWER

- Elle est composée de deux circuits :
 - Un circuit de régulation :
 - NCP1117 (5v)
 - LP2985 (3,3v)
 - Les pins d'alimentation



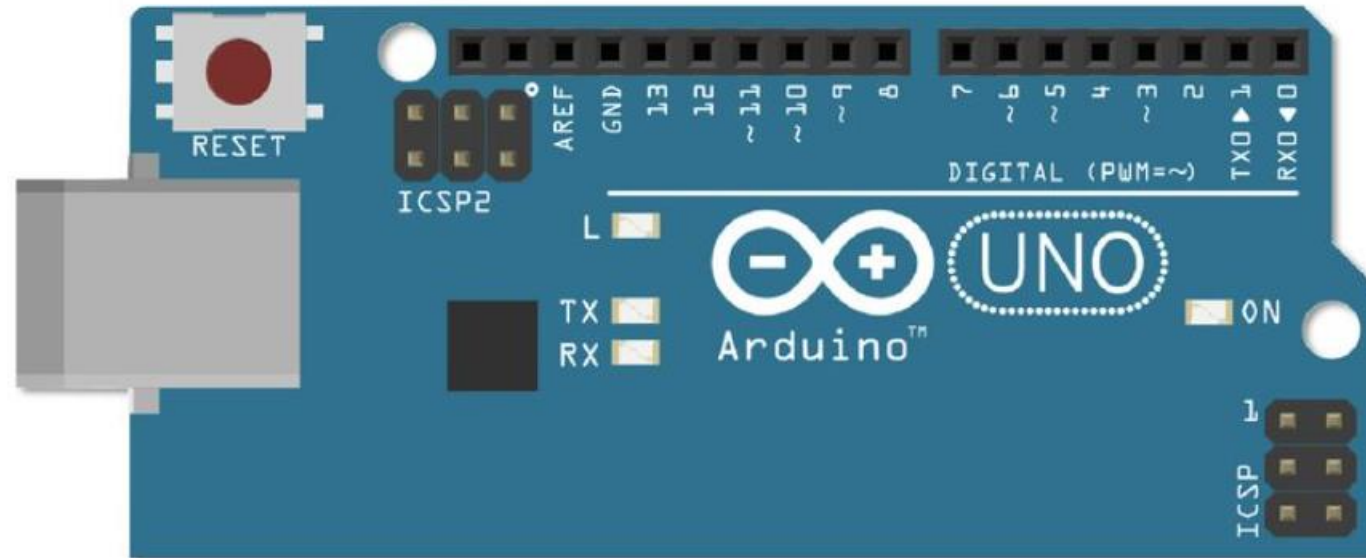
ISCP

- L'atmega est connectée par son port SPI à un connecteur de six pins dédiés à la programmation du microcontrôleur



GPIO

- La carte Arduino dispose de 14 broche numérique E/S
 - 6 peuvent être utilisé en sorties PWM
 - Le pin 2 est un pin d'interruption
 - Les pin 0 et 1 sont reliés au périphérique UART



Logiciel Arduino

- Le logiciel de programmation IDE est une application JAVA, open source et compatible avec Windows, Linux et MAC OS,
- Il est composé d'un éditeur de code et d'un compilateur qui transforme le code source écrit dans le langage compréhensible par l'humain en langage machine de programmation utilisé est le C++ compilé avec GCC.




C'est quoi un compilateur

- Le compilateur est la partie du logiciel Arduino qui transforme votre code C/C++ en un code machine (binaire), il correspond aux instructions réelles pour le microcontrôleur Atmega de la carte Arduino .



C'est quoi un Bootloader

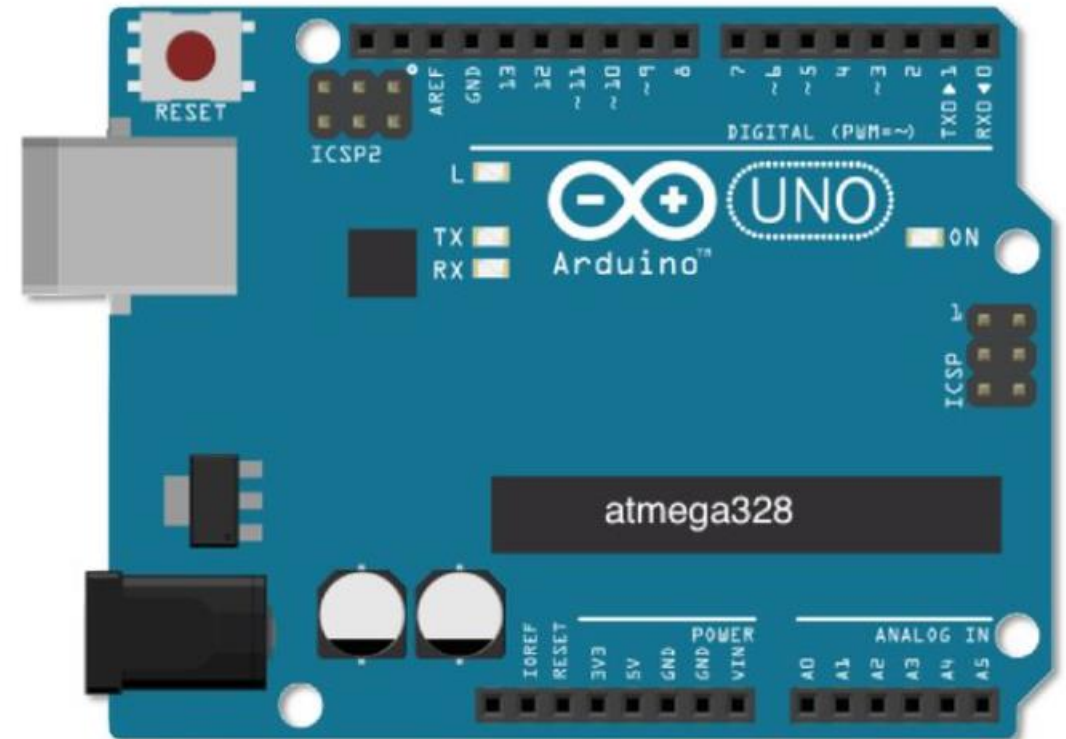
- Les microcontrôleurs sont habituellement programmés par un programmeur sauf si vous avez une partie du firmware dans votre microcontrôleur qui permet d'installer un nouveau firmware en utilisant un programmeur externe.
-  un Bootloader
- Le Bootloader est un genre de bios pour uContrôleur. Il gère les paramètres bas-niveau de la puce, comme sa fréquence, son voltage, etc
- Chaque uContrôleur à sa propre version

Microcontrôleurs de la famille AVR d'Atmel

I

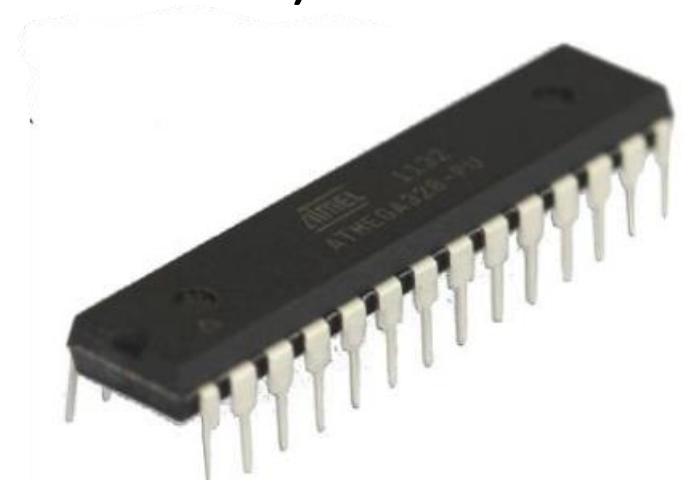
Caractéristiques généralesI

- Technologie *CMOS* haute vitesse : rapidité + faible consommation
- Architecture *RISC* et *Harvard*
 - Puissance en Mips = fréquence de l'horloge
- Alimentation entre 1,8 et 5,5V
- 3 mémoires indépendantes :
 - Mémoire flash intégrée (10000 cycles écriture)
 - Mémoire EEPROM pour le stockage de données semi-permanentes
 - (100000 cycles écriture)
 - Mémoire SRAM rapide



Atmega328p

- L'Atmega 328p est uContrôleur single-chip crée par Atmel, 8bits basé sur l'architecture RISC de la famille AVR
 - Technologie picoPower
 - Fréquence de traitement de 8 à 20Mhz (up to 20 MIPS)
 - Flash 32 Ko, EEPROM 1024o
 - 131 instructions
 - 32 registres d'usage générale
 - 23 GPIO



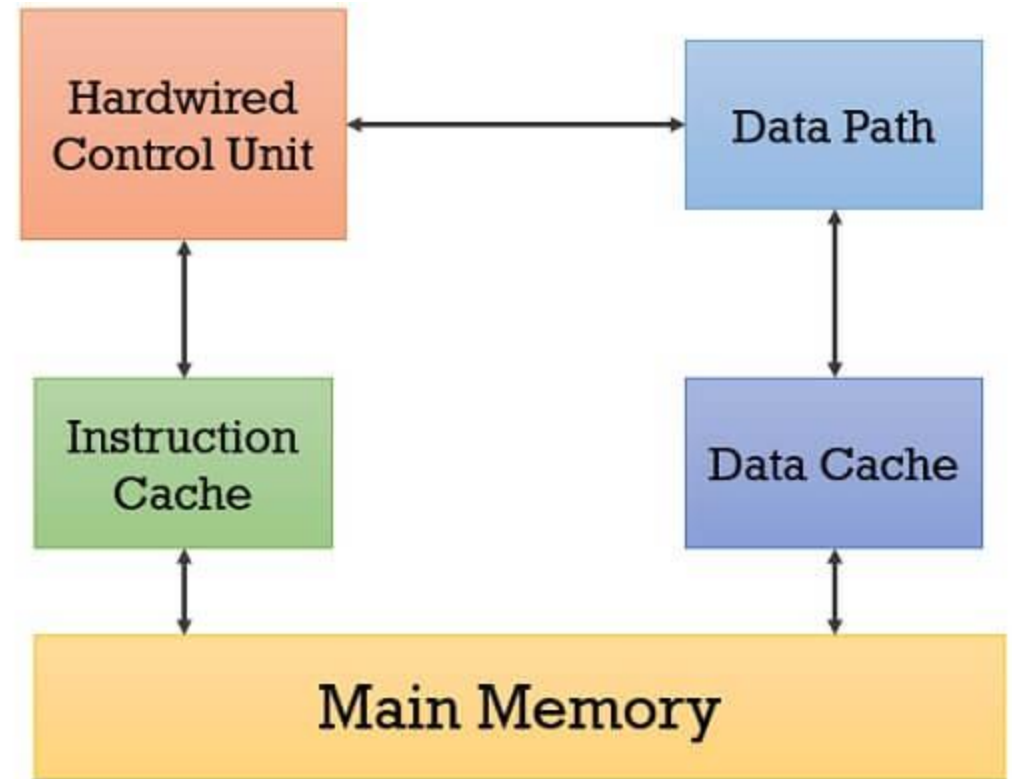
Atmega328p

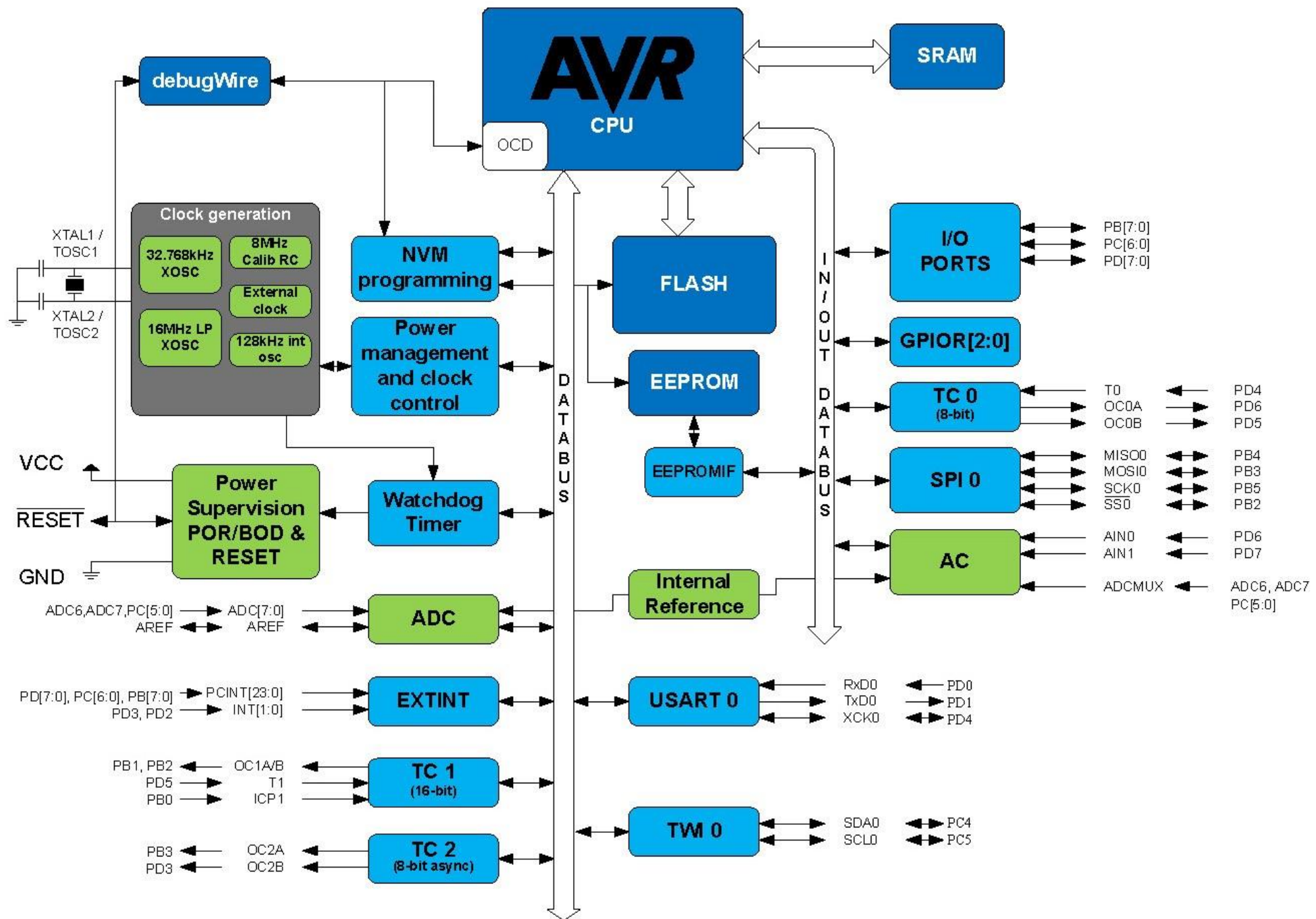
- Reset à la mise sous tension
- Oscillateur étalonner en interne
- Six mode de veille
- Interruption interne et externe
- ADC 10 bits
- Deux timers 8bits et un timer 16 bits
- 6 sorties PWM
- 3 interfaces de communications:
 - ❖ 1 SPI,
 - ❖ 1 I2C ,
 - ❖ USART.



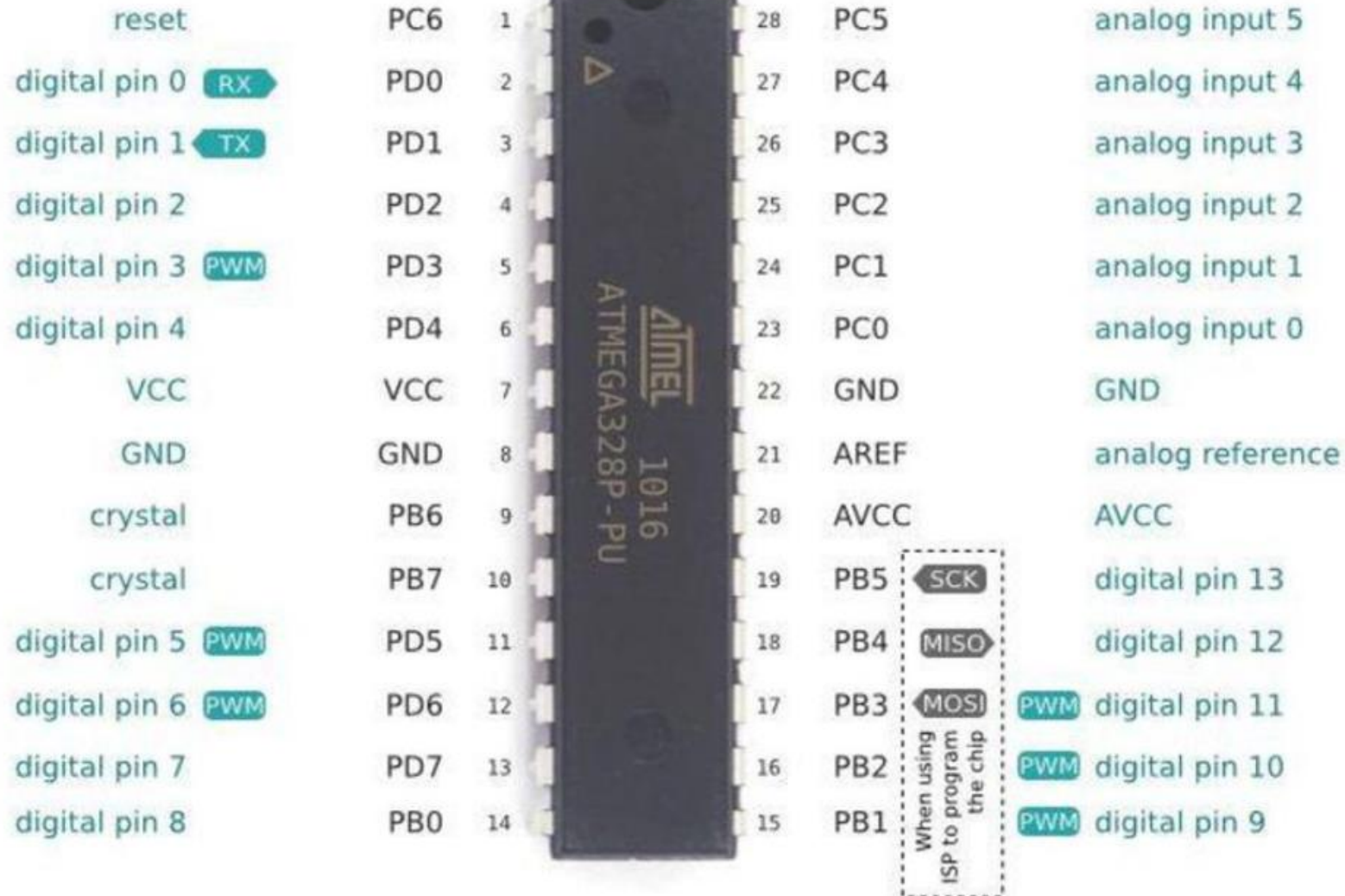
RISC

- L'atmega328p est un circuit basé sur l'architecture RISC.
- RISC est l'abréviation de Reduced instruction set computer en français processeur à jeu d'instructions réduit





Pinout



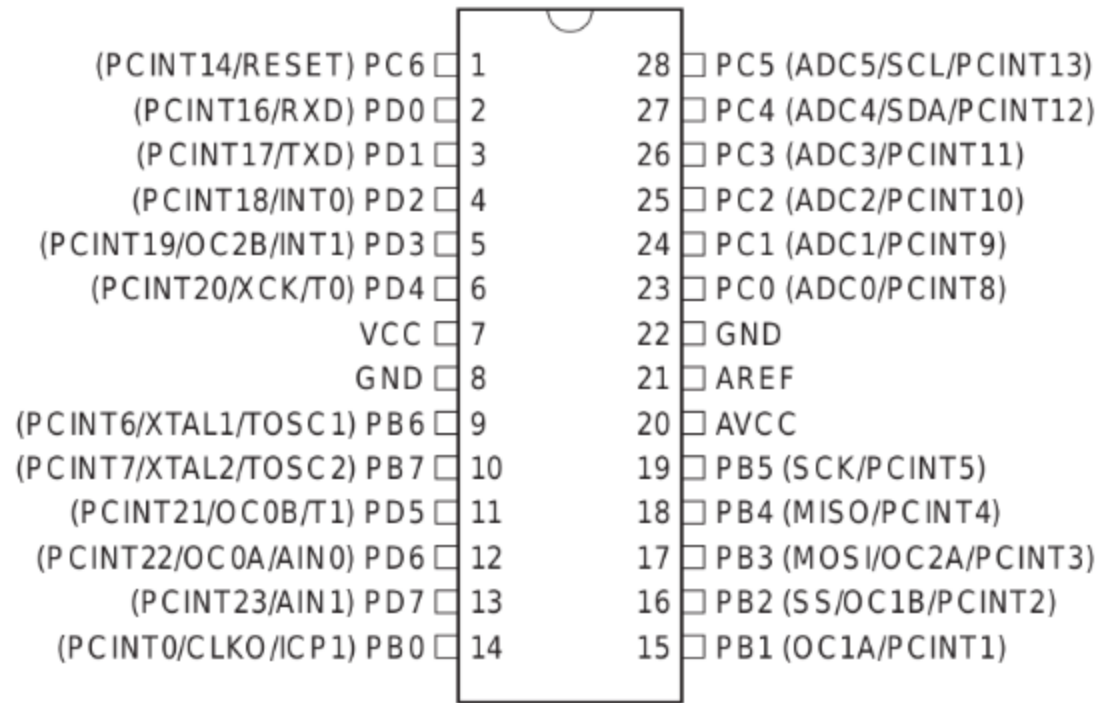
Caractéristiques générales

Microcontrôleur de l'*Arduino UNO*

Caractéristiques générales de l'Atmega328p

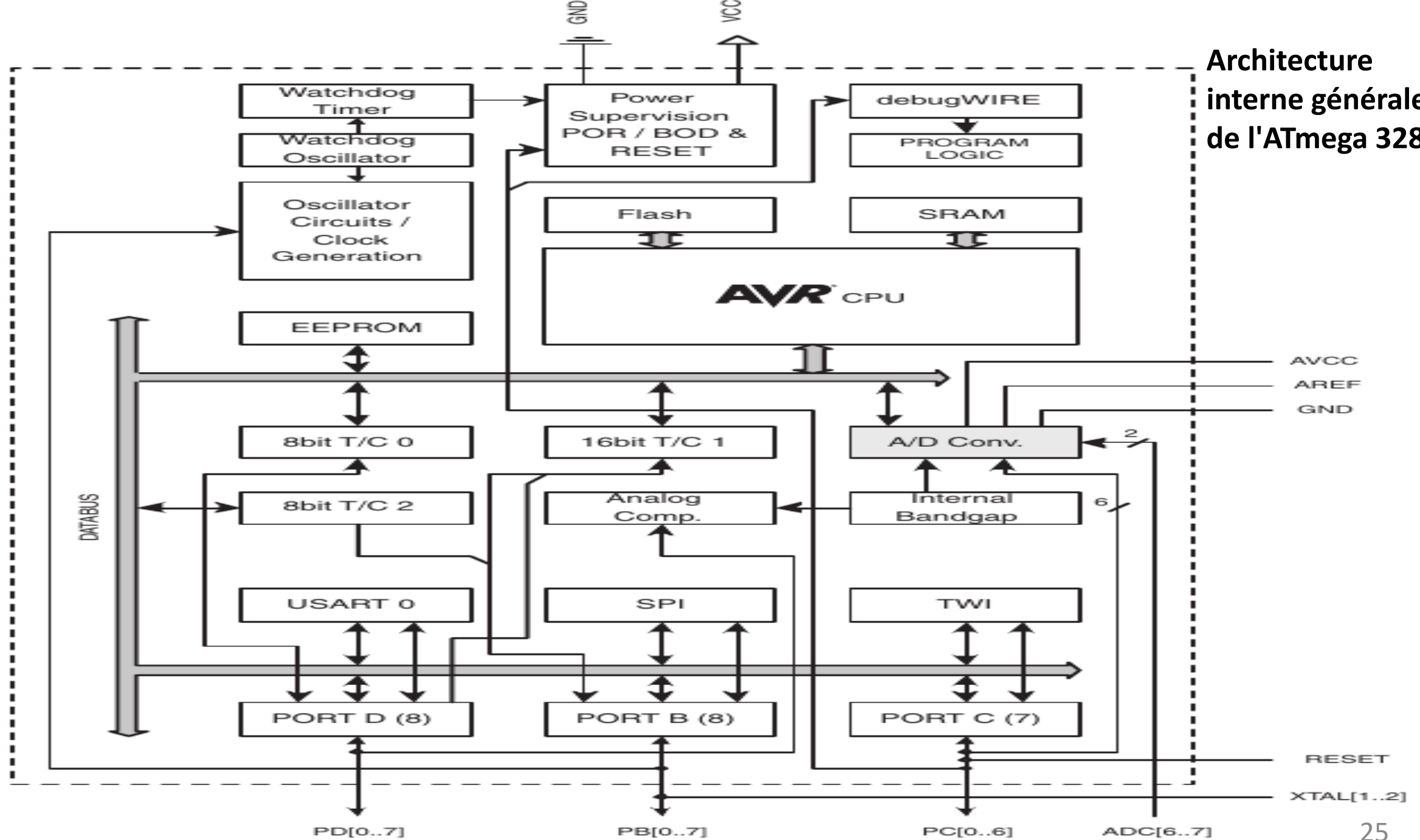
- Nombre de broches : 28 (sur le modèle utilisé en TP)
- Mémoire Flash : 32 ko (programmable par interface série)
- Mémoire Données EEPROM : 1 ko
- Mémoire RAM : 2 ko
- 32 registres de travail d'accès rapide pour l'ALU
- Ports parallèles : 3, avec 23 broches E/S
- Fréquence d'horloge : 16 Mhz (maxi tolérée = 20 Mhz)
 - donc : 16 cycles d'horloge par micro-seconde
- Périphériques internes
 - 6 convertisseur Analogique/Numérique 10 bits, comparateur analogique
 - 1 timer 16 bits (T1), 2 timers 8 bits (T0, T2)
 - 6 canaux PWM, 1 chien de garde (*watchdog*)
 - SPI, USART, TWI (=I2C)
- 26 interruptions
- 5 modes d'économie d'énergie

Brochage de l'ATmega 328p



- VCC (1), GND (2)
- E/S (=GPIO) : PORTB (8), PORTC (7), PORTD(8)
- AVCC (1) : alimentation CAN
- AREF (1) : entrée comparateur analogique

Architecture interne générale de l'ATmega 328



ATmega328 Pins

Pin Number	Pin Name	Pin Number	Pin Name
1	PC6	15	PB1
2	PD0	16	PB2
3	PD1	17	PB3
4	PD2	18	PB4
5	PD3	19	PB5
6	PD4	20	AVCC
7	Vcc	21	AREF
8	GND	22	GND
9	PB6	23	PC0
10	PB7	24	PC1
11	PD5	25	PC2
12	PD6	26	PC3
13	PD7	27	PC4
14	PB0	28	PC5

ATmega328 Pinout

Arduino Pins

RESET

Digital pin 0 (RX)

Digital pin 1 (TX)

Digital pin 2

Digital pin 3 (PWM)

Digital pin 4

Voltage (VCC)

Ground

Crystal

Crystal

Digital pin 5

Digital pin 6

Digital pin 7

Digital pin 8

Pin # 1: PC6

Pin # 2: PD0

Pin # 3: PD1

Pin # 4: PD2

Pin # 5: PD3

Pin # 6: PD4

Pin # 7: VCC

Pin # 8: GND

Pin # 9: PB6

Pin # 10: PB7

Pin # 11: PD5

Pin # 12: PD6

Pin # 13: PD7

Pin # 14: PB0

ATmega328

Pin # 28: PC5

Pin # 27: PC4

Pin # 26: PC3

Pin # 25: PC2

Pin # 24: PC1

Pin # 23: PC0

Pin # 22: GND

Pin # 21: Aref

Pin # 20: AVCC

Pin # 19: PB5

Pin # 18: PB4

Pin # 17: PB3

Pin # 16: PB2

Pin # 15: PB1

Arduino Pins

Analog Input 5

Analog Input 4

Analog Input 3

Analog Input 2

Analog Input 1

Analog Input 0

Ground (GND)

Analog Reference

Voltage (VCC)

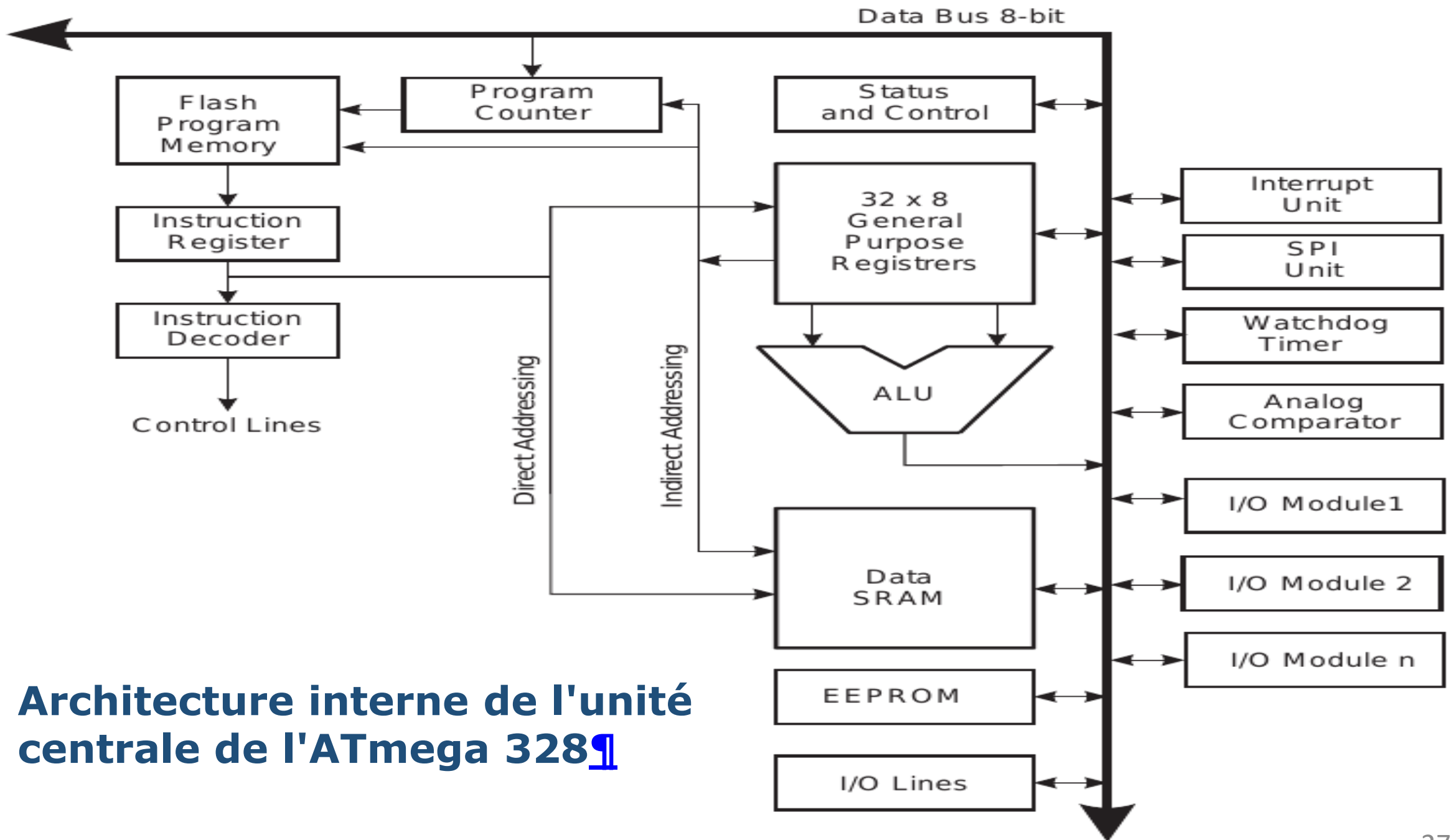
Digital Pin 13

Digital Pin 12

Digital Pin 11 (PWM)

Digital Pin 10 (PWM)

Digital Pin 9 (PWM)



Architecture performante

I Architecture Harvard : 3 accès mémoires indépendants

- Mémoire Programme (Flash)
 - liée directement à l'ALU et au bloc de 32 registres de travail internes
 - propre bus de données
- Autres mémoires + autres ressources
 - partagent un même bus de données
- Le processeur dispose d'un *pipeline* à 1 niveau
 - l'instruction N s'exécute,
 - **pendant** que l'instruction $N+1$ est récupérée en mémoire
- Donc : exécution d'1 instruction en 1 cycle d'horloge
 - lecture de 2 opérandes dans les registres de travail
 - exécution du calcul
 - stockage du résultat dans les registres de travail

Mémoires

Program Memory

16-bits

Flash
2 Ko x 16

Data Memory

8-bits

32 General
Purpose
Registers

64 I/O
Registers

160 I/O Ext
Registers

SRAM
2048 x 8

EEPROM

8-bits

EEPROM
1 Ko x 8

Quelques registres essentiels de l'ATmega328

- 32 registres 8 bits, R0 à R31
- certains sont utilisables en 16 bits (pour les adressages indirects)
- placés en début de la RAM de donnée

General
Purpose
Working
Registers

7	0	Addr.
R 0		0x00
R 1		0x01
R 2		0x02
...		
R 13		0x0D
R 14		0x0E
R 15		0x0F
R 16		0x10
R 17		0x11
...		
R 26		0x1A
R 27		0x1B
R 28		0x1C
R 29		0x1D
R 30		0x1E
R 31		0x1F

X-register Low Byte
X-register High Byte
Y-register Low Byte
Y-register High Byte
Z-register Low Byte
Z-register High Byte

Registre d'état SREG

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Constitués de 8 bits, ayant chacun un rôle propre :

- **I** : autorisation globale des interruptions
- **T** : bit source ou destinataire pour les instructions de copie de bit seul
- **H** : demi-retenue (*Half-carry*) - c.-à-d. du premier quartet - sur opérations arithmétiques
- **S** : bit de signe
- **V** : bit de débordement (*oVerflow*)
- **N** : bit de résultat négatif
- **Z** : bit de zéro, positionné à 1 lorsque le résultat de certaines opération est nul
- **C** : bit de retenue (*Carry*)

Attention

Ce registre n'est pas sauvegardé sur la pile sur interruption ou appel de fonction.

Pile et son registre « pointeur de pile »

- La pile est une portion de la SRAM, localisée par un registre de pointeur de pile
- Elle est manipulable via les instructions **PUSH** et **POP**
- Elle sert en particulier à sauvegarder le compteur ordinal **PC** (*Program Counter*), qui désigne l'adresse de l'instruction à exécuter
 - lors d'un appel de sous-programme
 - ou lors d'une interruption
- Sa position en mémoire est définie par le registre « pointeur de pile » :

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
0x3D (0x5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	
	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	RAMEND	

Registres de contrôle des ports

- Chaque port *x* (pour *B*, *C* ou *D*) est contrôlé par 3 registres
 - **DDRx** : le registre de direction (sens de transfert) du port
 - **PORTx** : le registre de donnée du port
 - **PINx** : le registre d'entrée du port
- Chacun de leur bit contrôle une ligne particulière du port
 - exemple : le port *B* regroupe les broches numériques 8 à 13
 - le bit 0 de ses registres est donc associé à la broche numérique 8

PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0									
0x0B (0x2B)	<table><tr><td>PORTD7</td><td>PORTD6</td><td>PORTD5</td><td>PORTD4</td><td>PORTD3</td><td>PORTD2</td><td>PORTD1</td><td>PORTD0</td></tr></table>								PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0										
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W									
Initial Value	0	0	0	0	0	0	0	0									

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Fonction des registres

- Seuls **PORTx** et **DDRx** se comportent vraiment comme des variables mémoires
 - un composant mémoire leur ait associé, qui joue le rôle de verrou (= *latch*)
 - les écritures dans ces registres sont donc persistentes
- **PINx** n'est pas un composant mémoire
 - il est directement connecté aux lignes du port x, et traduit donc leur tension
- **DDRx** : détermine la direction de chaque broche du port x
 - 0 pour une **E**ntrée, 1 pour une **S**ortie (« 0/1 » pour « E/S »)
- **PORTx**
 - pour les broches en sortie : 0 pour tension basse, 1 pour tension haute
 - pour les broches en entrée : 1 pour activer la résistance « pull-up », 0 sinon
- **PINx**
 - indique la tension des broches : 0 pour tension basse, 1 pour tension haute

Autorisations, priorités et mécanisme de demande d'interruption

- Les interruptions sont désactivables
 - globalement
 - ou individuellement, par source d'interruption
- La prise en compte de l'interruption peut être retardée si son niveau de priorité est insuffisant
 - les niveaux de priorités sont fixes, et fonctions de la source d'interruption
 - exemple : *reset* = interruption de plus forte priorité
- Les sources d'interruptions (périphériques internes, entrées externes) font leur demande d'interruption en positionnant à 1 un bit dédié (*interrupt flag*), stocké dans l'un de leurs registres
 - ainsi, une demande d'interruption non traitée faute de priorité suffisante ou faute d'autorisation, n'est pas oubliée pour autant : le drapeau reste levé
- Sur l'*ATmega328*, il faut de 4 à 6 cycles d'horloge avant que le microcontrôleur exécute la routine d'interruption

Déroulement des interruptions

- Sur une interruption, voici ce qui se produit
 - l'instruction en cours s'achève
 - le masque d'interruption **I** est automatiquement mis à 1 pour empêcher d'autres interruptions
 - le compteur ordinal **PC** est sauvegardé sur la pile (au contraire de **SREG**)
 - **PC** est ensuite chargé avec le vecteur d'interruption (qui effectue normalement un saut vers le sous-programme d'interruption)
- En fin d'exécution de la routine d'interruption, c'est l'inverse
 - **PC** est chargé avec le dessus de la pile
 - **I** est remis à 0
 - le programme reprend alors là où il avait été interrompu

Liste des interruptions de l'ATmega328

- 26 sources d'interruptions
- leurs vecteurs sont constitués de 2 mots d'instructions
- ils sont ordonnés par priorité décroissante
 - vecteur 0 = *reset : le plus prioritaire

Liste des interruptions de l'ATmega328

VectorNo.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMP A	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMP B	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMP A	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMP B	Timer/Counter1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMP A	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMP B	Timer/Counter0 Compare Match B
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART, RX	USART Rx Complete
20	0x0026	USART, UDRE	USART, Data Register Empty
21	0x0028	USART, TX	USART, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator
25	0x0030	TWI	2-wire Serial Interface
26	0x0032	SPM READY	Store Program Memory Ready

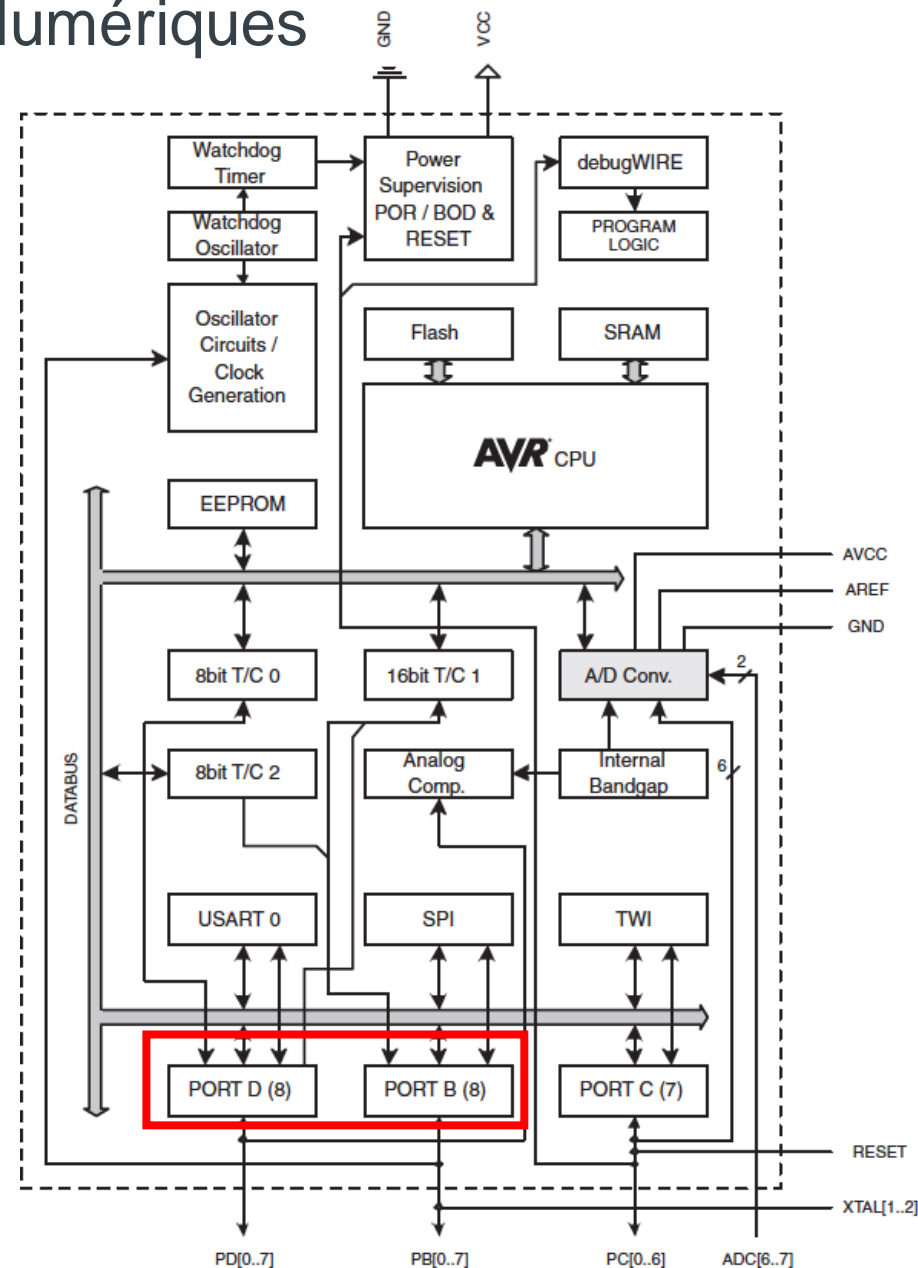
Périphériques internes

Caractéristiques générales

- L'*Arduino UNO* dispose de 23 broches d'E/S, organisées en 3 ports parallèles numériques
 - ce sont les ports *B*, *C* et *D*
 - port *B* : broches numériques 8 à 13
 - port *C* : broches analogiques (0 à 5)
 - port *D* : broches numériques 0 à 7
- Ils sont tous bidirectionnels
- le port *C* peut aussi être utilisé comme « port d'entrée **analogique** » (CAN)
- Leurs lignes peuvent être configurées/utilisées individuellement

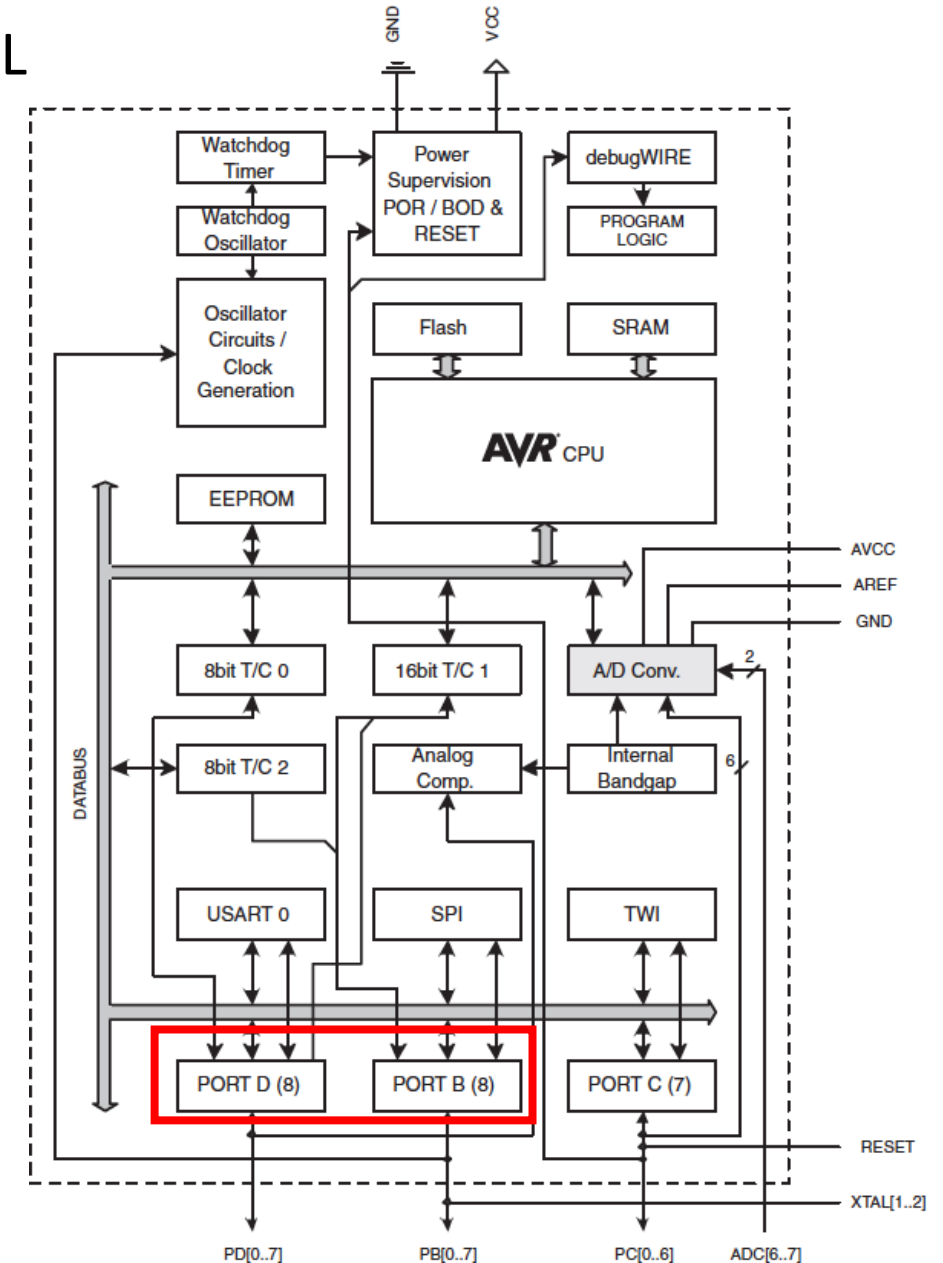
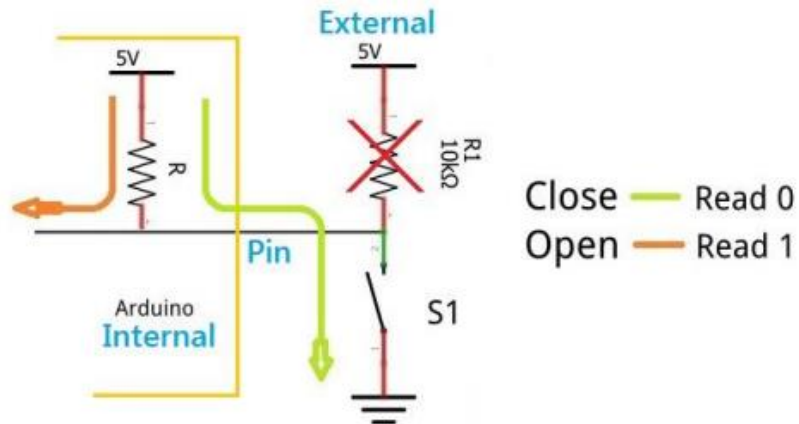
Architecture microcontrôleur ATmega328: E/S Numériques

- ❑ › Nombre d'Entrées/Sorties numériques: 14 (marquées de 0 à 13)
- ❑ › Chaque E/S peut prendre soit:
 - Etat Haut \Rightarrow Valeur logique 1 \Rightarrow Tension 5V
 - Etat Bas \Rightarrow Valeur logique 0 \Rightarrow Tension 0V
- ❑ › Chaque pin peut être configuré comme:
 - Entrée avec la fonction: `pinMode(pin, INPUT)`
 - Sortie avec la fonction: `pinMode(pin, OUTPUT)`
- ❑ › On lit l'état d'un pin configuré en entrée par la fonction:
`value = digitalRead(pin)` qui retourne:
 - `value = LOW` (ou 0) si la tension imposée sur le pin est de 0V
 - `value = HIGH` (ou 1) si la tension imposée sur le pin est de 5V
- ❑ › On impose l'état d'un pin configuré en sortie par la fonction: `digitalWrite(pin, value)`, où `value` peut être:
 - `LOW` pour imposer la tension 0V sur le pin
 - `HIGH` pour imposer la tension 5V sur le pin



Architecture microcontrôleur ATmega328: Entrée PUL

- › Toutes les entrées numériques peuvent être configurées pour inclure des résistances de PULL-UP internes
- › Cette configuration permet d'éliminer la résistance de PULL-UP (ou PULL-DOWN externe) si un interrupteur ou un bouton poussoir est connecté à l'entrée
- › La résistance de PULL-UP interne est activée par la fonction:
`pinMode(pin, INPUT_PULLUP)`
- › Avec cette configuration, la fonction:
`value = digitalRead(pin)` retourne:
 - value = LOW (ou 0) si le bouton est appuyé (fermé)
 - value = HIGH (ou 1) si le bouton est relâché (ouvert)



Architecture microcontrôleur ATmega328: Sorties PWM

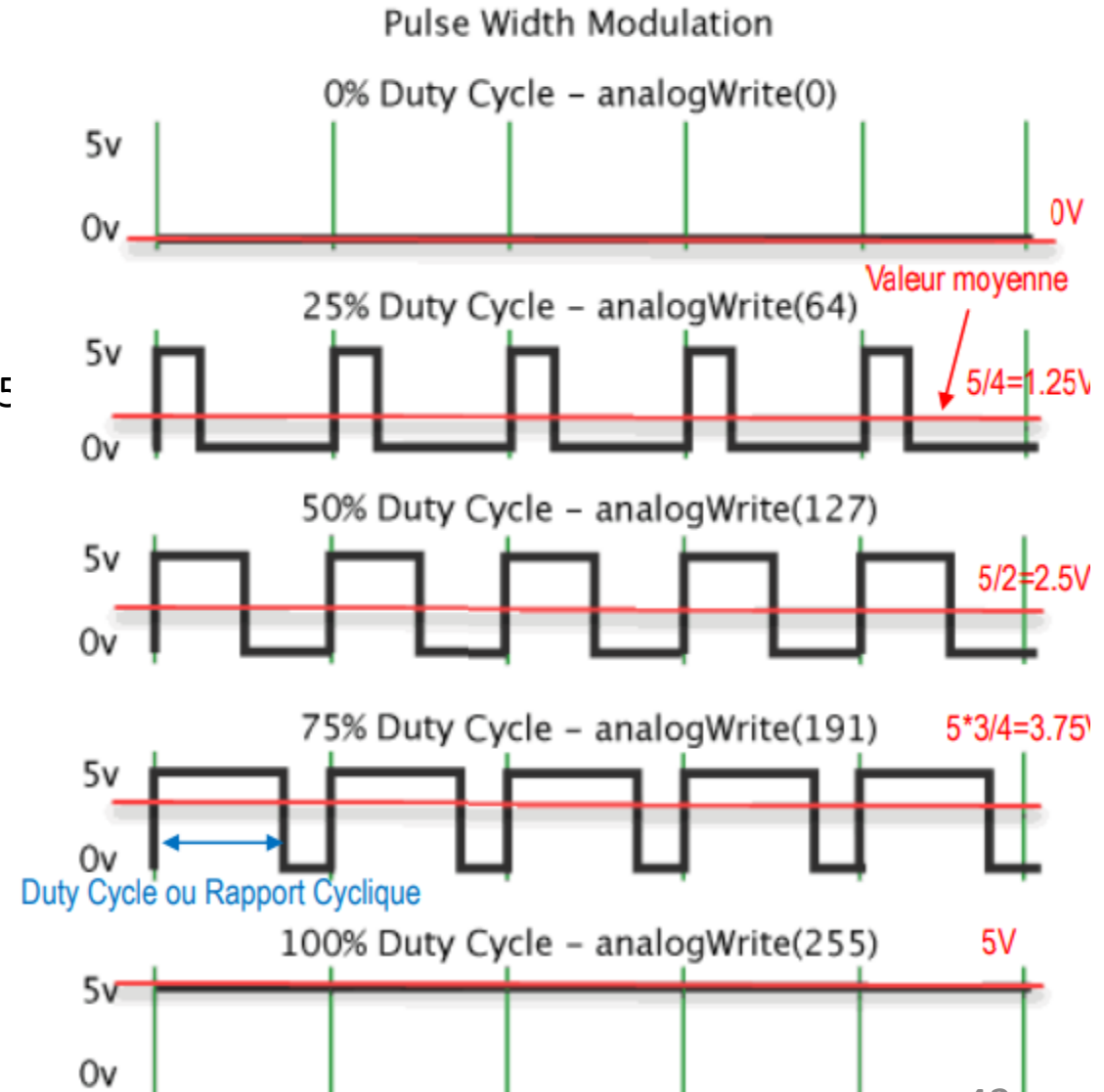
Les pins 3, 5, 6, 9, 10 et 11 (marqués par ~) permettent d'avoir une sortie en PWM (Pulse Width Modulation) qui est équivalente à une sortie analogique

› Le pin doit être configuré en sortie:

`pinMode(pin, OUTPUT)`

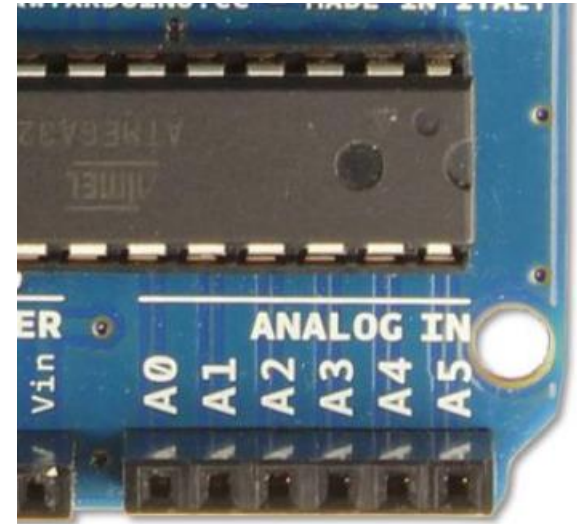
› La valeur analogique est écrite par la fonction

`analogWrite(pin, value)`, value est comprise entre 0 et 255 pour avoir des tensions entre 0V et 5V



ADC

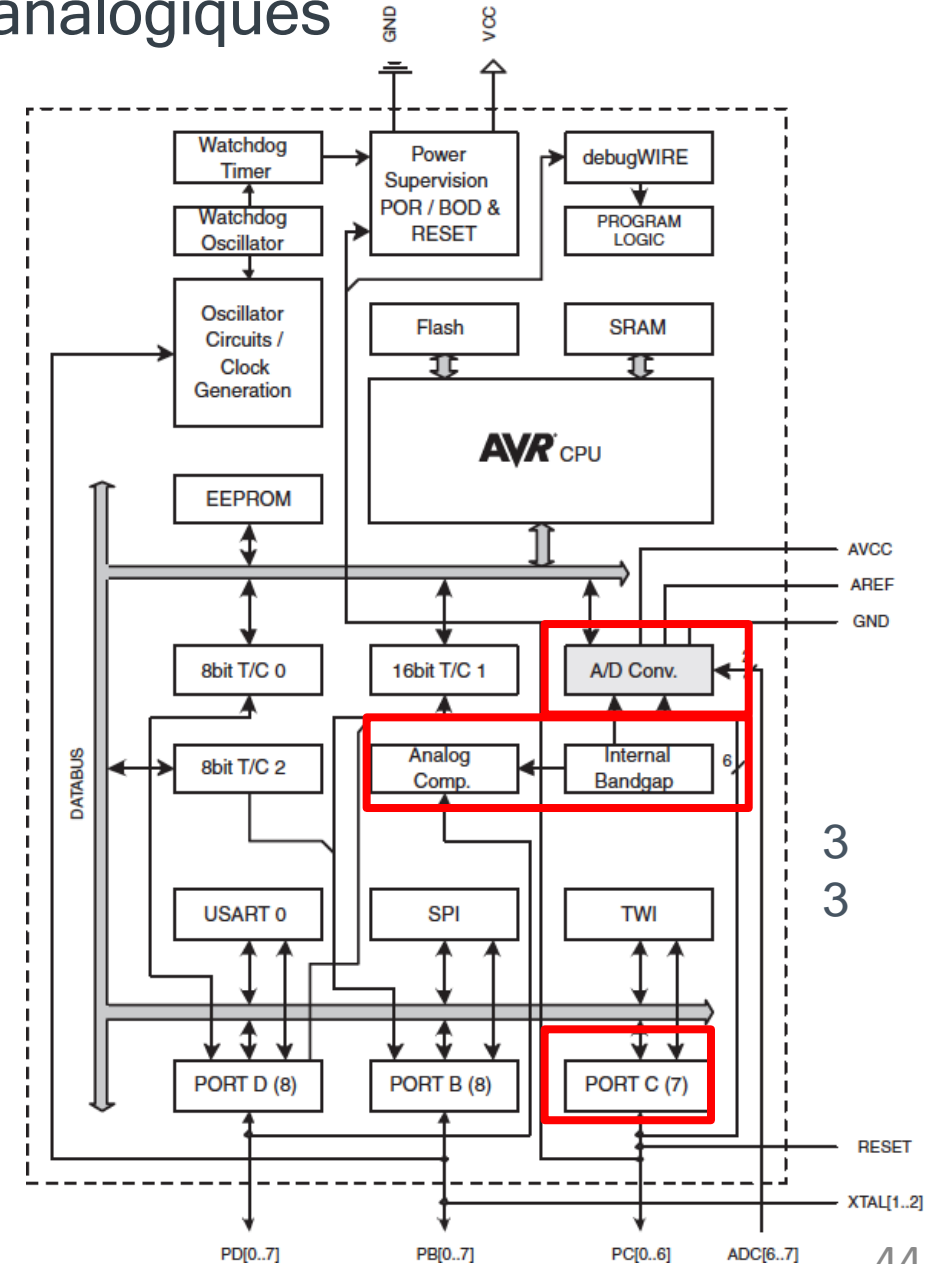
- La carte Arduino dispose de 6 pins ADC analogique 0-5
 - Chaque entrée est connectée à un convertisseur analogique numérique
 - Ces pin peuvent aussi être programmer comme sortie ou entrée numérique



Architecture microcontrôleur ATmega328: Entrées analogiques

- › Nombre d'Entrées analogiques: 6 (A0 à A5)
- › Ces entrées sont connectées à un convertisseur Analogique Numérique (ADC: Analog to Digital Converter) à 10 bits
- › Donc: chaque valeur analogique est convertit en une valeur numérique entre 0 et 1023
- › La fonction: `value = analogRead(pin)` permet de lire la valeur de la tension appliquée à une entrée analogique (pas besoin d'utiliser `pinMode(pin, INPUT)`)
- › La résolution de lecture est: $5V/1024 = 0.0049V$ (4.9 mV). La tension 5V est appelée tension de référence
- › La résolution peut être changée en utilisant la fonction: `analogReference(type)`, avec `type=`
 - DEFAULT: Tension de référence 5V
 - INTERNAL1V1 (ou INTERNAL): Tension de référence 1.1V
 - INTERNAL2V56: Tension de référence 2.56V
 - EXTERNAL: Tension de référence appliquée à l'entrée AREF (0 à 5V)

Remarque: A0 à A5 peuvent être utilisées comme E/S numériques (pins 14 à 19)



Timers

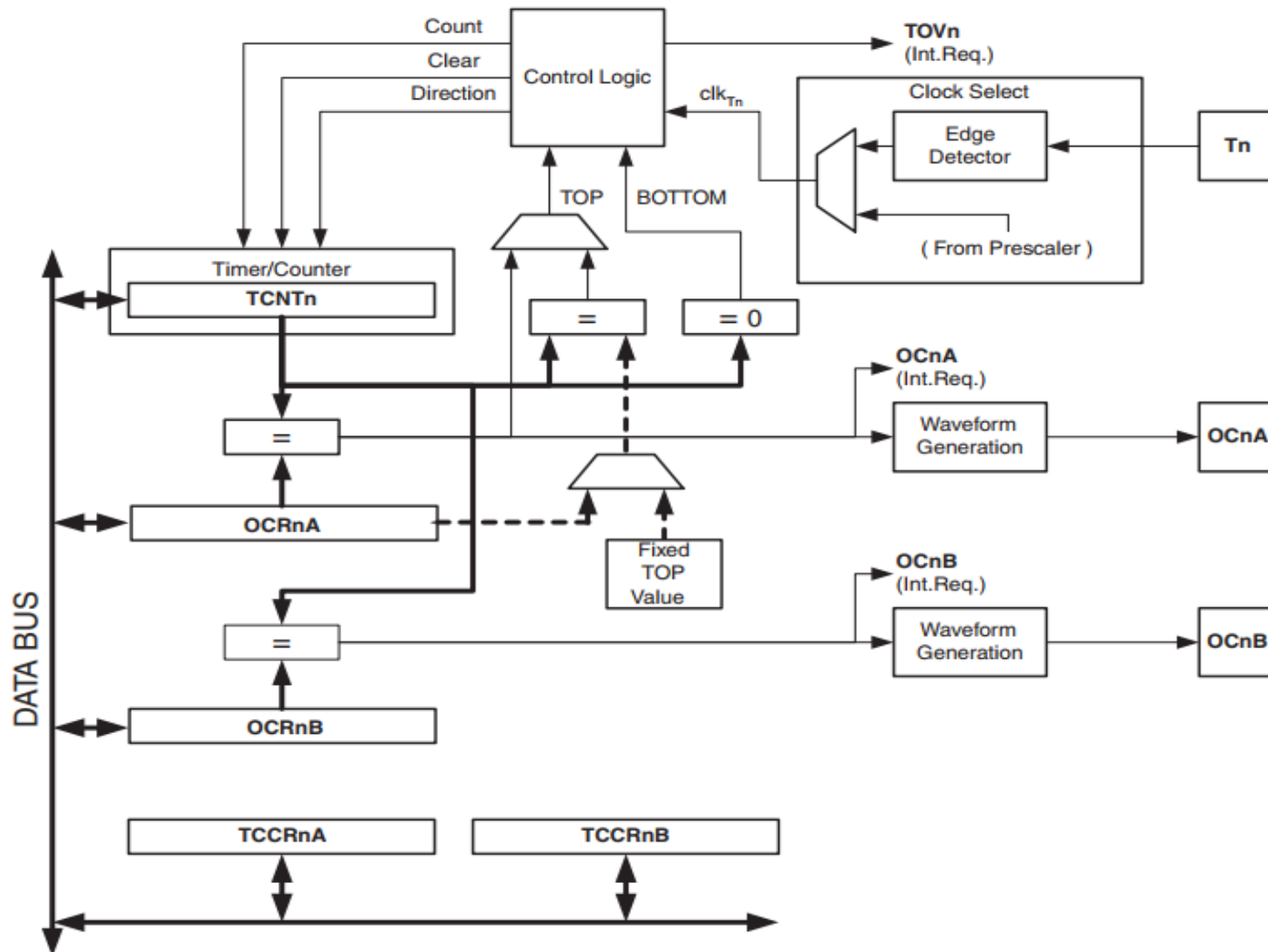
Définition : Timer

- C'est un périphérique matériel destiné à *compter* :
 - soit des *événements* extérieurs, potentiellement non-périodiques : c'est sa fonction de **compteur** ;
 - soit des *événements* internes périodiques (tops d'horloge) : c'est sa fonction de **temporisateur**, de mesure le *temps*.
- Tant qu'il est « activé », un timer *compte*. C'est sa fonction : il s'incrémente chaque fois qu'il détecte un nouvel événement.
En cas de débordement, il reprend à 0 (sauf mode de fonctionnement particulier), comme n'importe quelle variable C de type entier non-signé.
- Un timer sert généralement à :
 - mesurer des durées
 - déclencher périodiquement des routines d'interruptions
 - générer des signaux MLI

Composition d'un timer type

- des *broches* :
 - d'entrée : pour compter ses changements d'états ; ce sont les « événements externes » que le timer peut compter ;
 - de sortie : pour signaler que le compteur a atteint une valeur particulière ;
- un *sélecteur* : sélectionne l'entrée dont les changements d'états (*événements*) seront comptés, en général :
 - l'horloge interne
 - ou une broche externe ;
- un *prédiviseur* :
 - dans le cas où le compteur est connecté à l'horloge, il est possible de ralentir le comptage en ne comptant qu'*1 top d'horloge sur n* ;
 - *n* : c'est le prédiviseur. Il est généralement choisi parmi un ensemble de puissances de 2, par exemple : {1, 16, 64, 256} ;
- un *registre de comptage* : c'est lui qui stocke la valeur de comptage ;
- des *unités de comparaison*, chacune munie d'un *registre de comparaison* : ces registres contiennent une valeur destinée à être comparée à celle du compteur ; dans le but de changer l'état d'une broche de sortie de l'unité de comparaison, ou de déclencher une interruption ;
- des *registres de contrôle* : pour configurer son mode de fonctionnement.

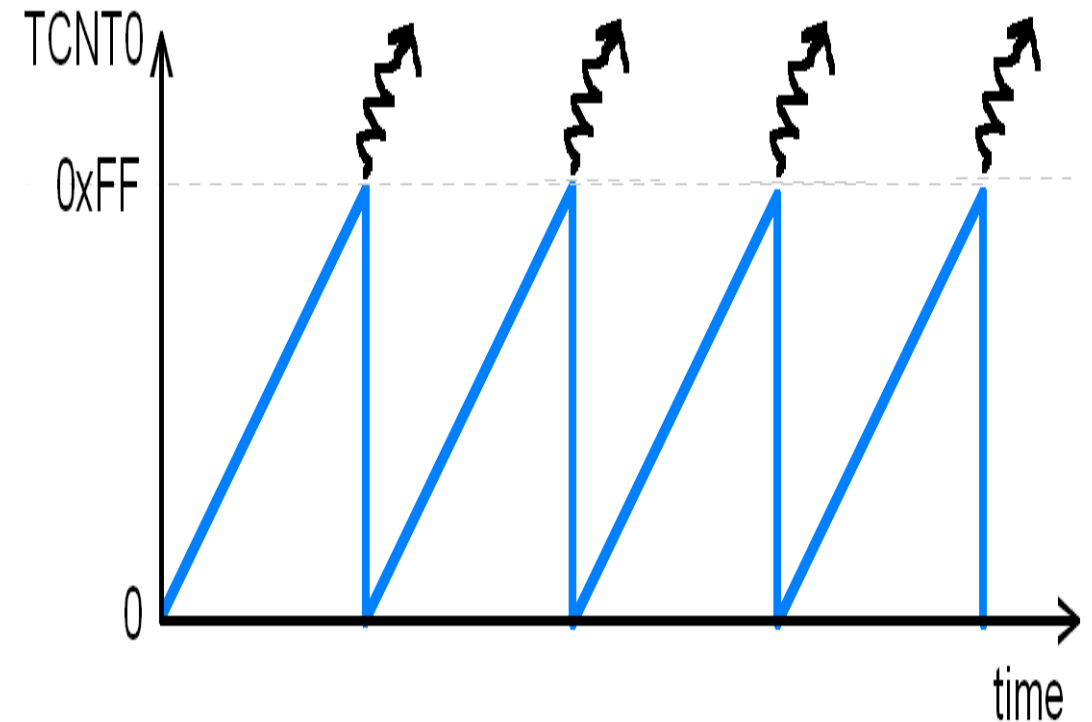
Schéma du timer 0 de l'ATmega328



Deux principaux modes de fonctionnement

Mode normal

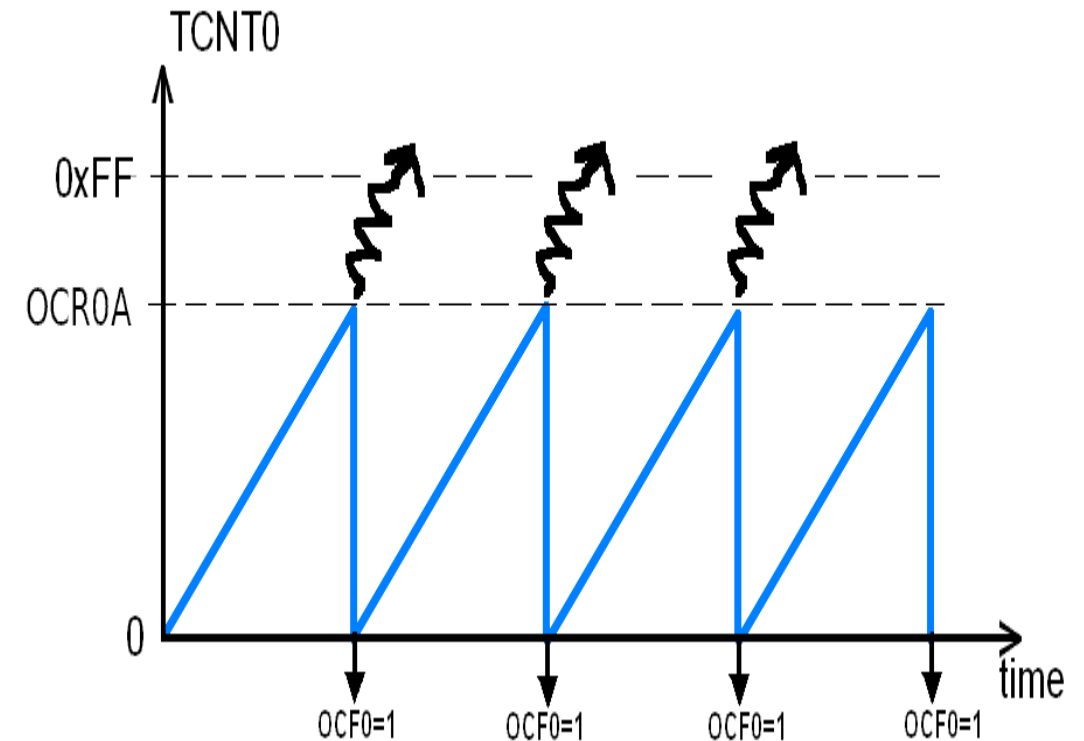
- le timer compte tout le temps, c'est-à-dire qu'il s'incrémente à chaque « évènement » détecté, qui peuvent être au choix :
 - les tops d'horloge prédivisés, en mode *temporisateur*,
 - les changements d'état de sa broche d'entrée, en mode *compteur* ;
 - il revient naturellement à 0 après un débordement (*overflow*).
 - il peut déclencher une *interruption de débordement* :
- Lorsque le timer compte les tops d'horloge prédivisés (fonction *temporisateur*), la durée (période) de ses cycles est définie par :



$$T_{\text{cycle_normal}} = \text{prédiviseur}/f_{\text{cpu}} * 256$$

Mode remise à zéro sur comparaison

- C'est le mode *CTC* : *Clear Timer on Compare match*
- Le domaine de comptage est raccourci : le timer évolue de 0 à la valeur du registre de comparaison A, puis revient à 0
- le test de comparaison peut aussi déclencher (outre le reset du compteur) :
 - une *interruption de comparaison A* (différente de celle de débordement ! et de celle de comparaison B)
 - un changement d'état de la broche de sortie OC0A

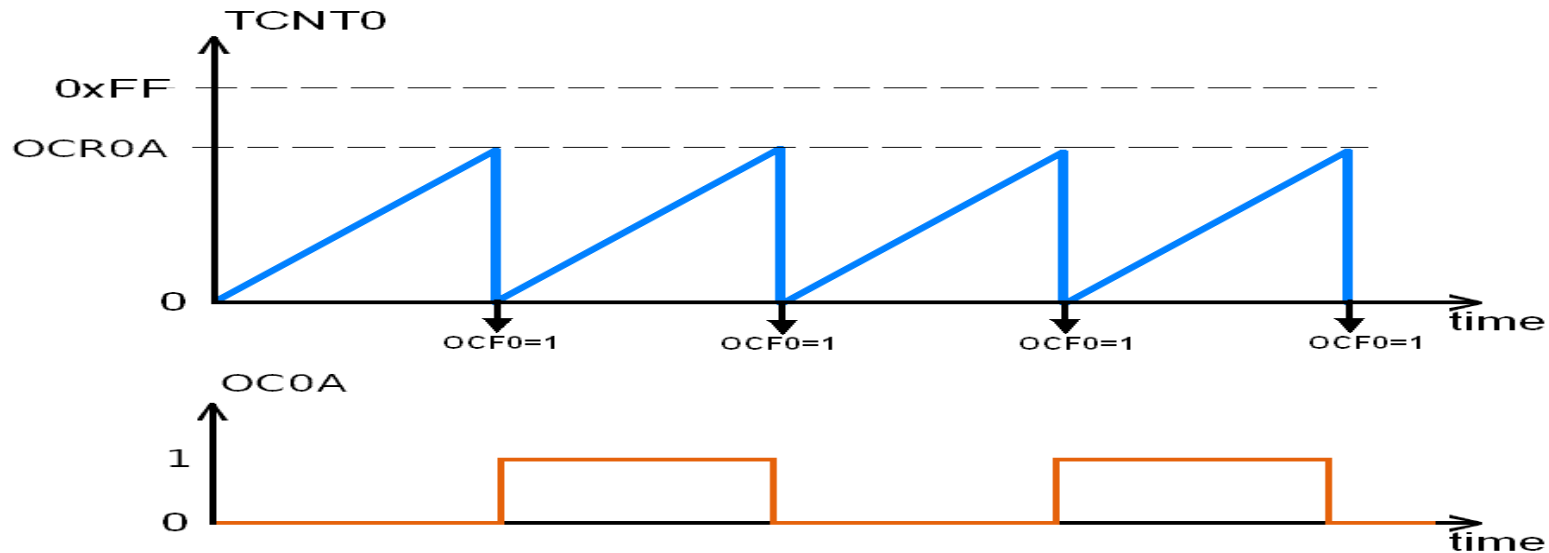


- Dans ce mode, + fonction *temporisateur*, la durée du cycle est :

$$T_{\text{cycle_CTC}} = \text{préd.}/f_{\text{cpu}} * (\text{OCR0A} + 1)$$

Génération de signal carré en mode CTC

- La broche de sortie OC0A (= broche du timer 0 associée à la comparaison A) peut s'inverser à chaque comparaison :



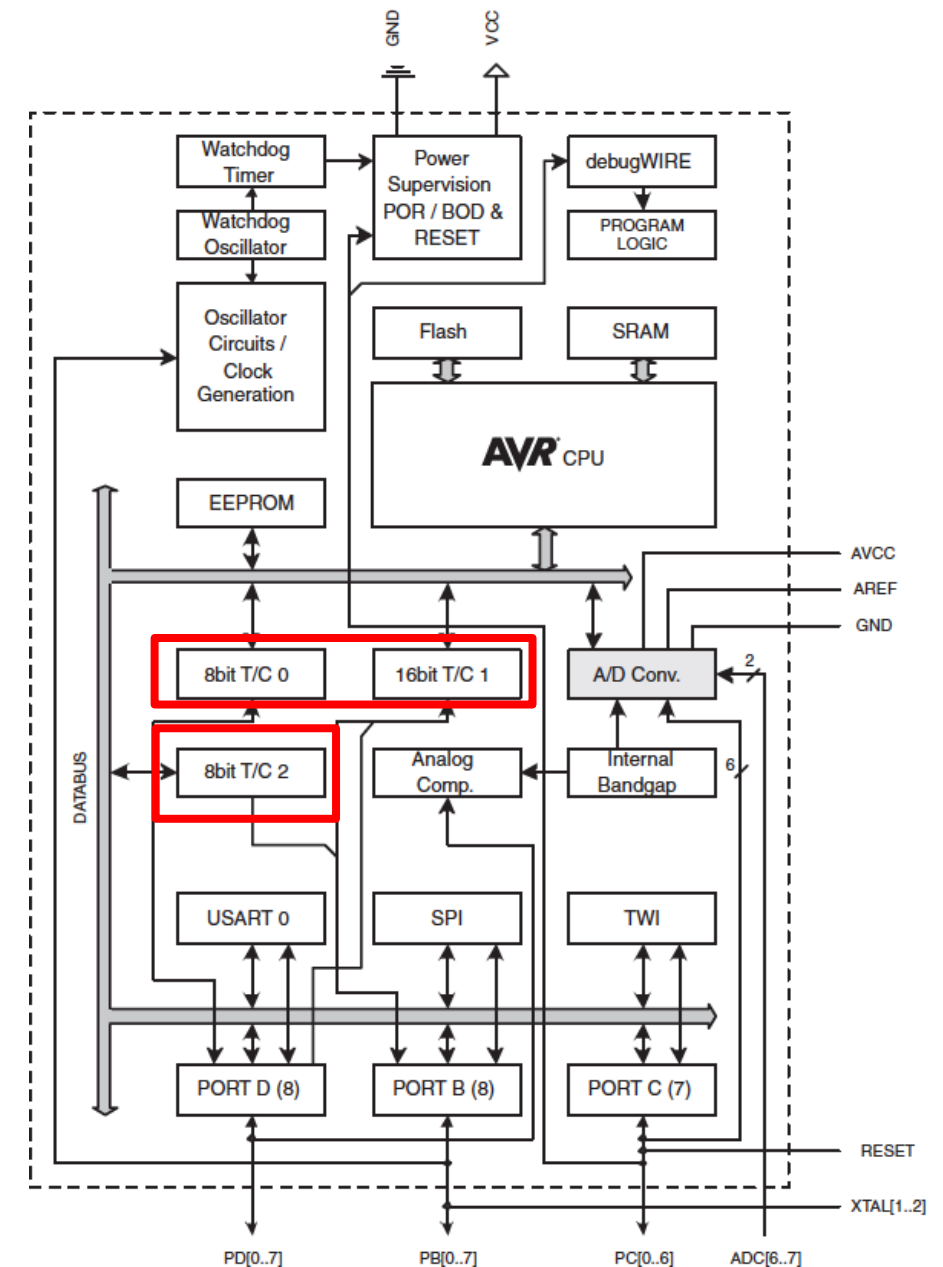
- Cela permet la génération d'un signal carré de période double de celle du cycle de comptage :

$$T_{\text{signal_sortie_CTC}} = 2 \cdot \text{préd.}/f_{\text{cpu}} \cdot (\text{OCR0A} + 1)$$

Architecture microcontrôleur ATmega328: Timers

Trois Timers peuvent être utilisés pour compter le temps:

- T0: Timer 8 bits (de 0 à 255)
- T1: Timer 16 bits (de 0 à 65535)
- T2: Timer 8 bits › Pour chaque Timer, la fréquence d'horloge et la valeur maximale de comptage peuvent être programmées
 - › Fonctions de temps qui utilisent les Timers :
 - **delay (ms)**: attendre un délais en ms
 - **delayMicroseconds (us)**: attendre un délais en us
 - **ms = millis()**: retourne le nombre de ms écoulés à partir du début d'exécution du programme
 - **us = micros()**: retourne le nombre de μ s écoulés à partir du début d'exécution du programme



Architecture microcontrôleur ATmega328: Interfaces Communication

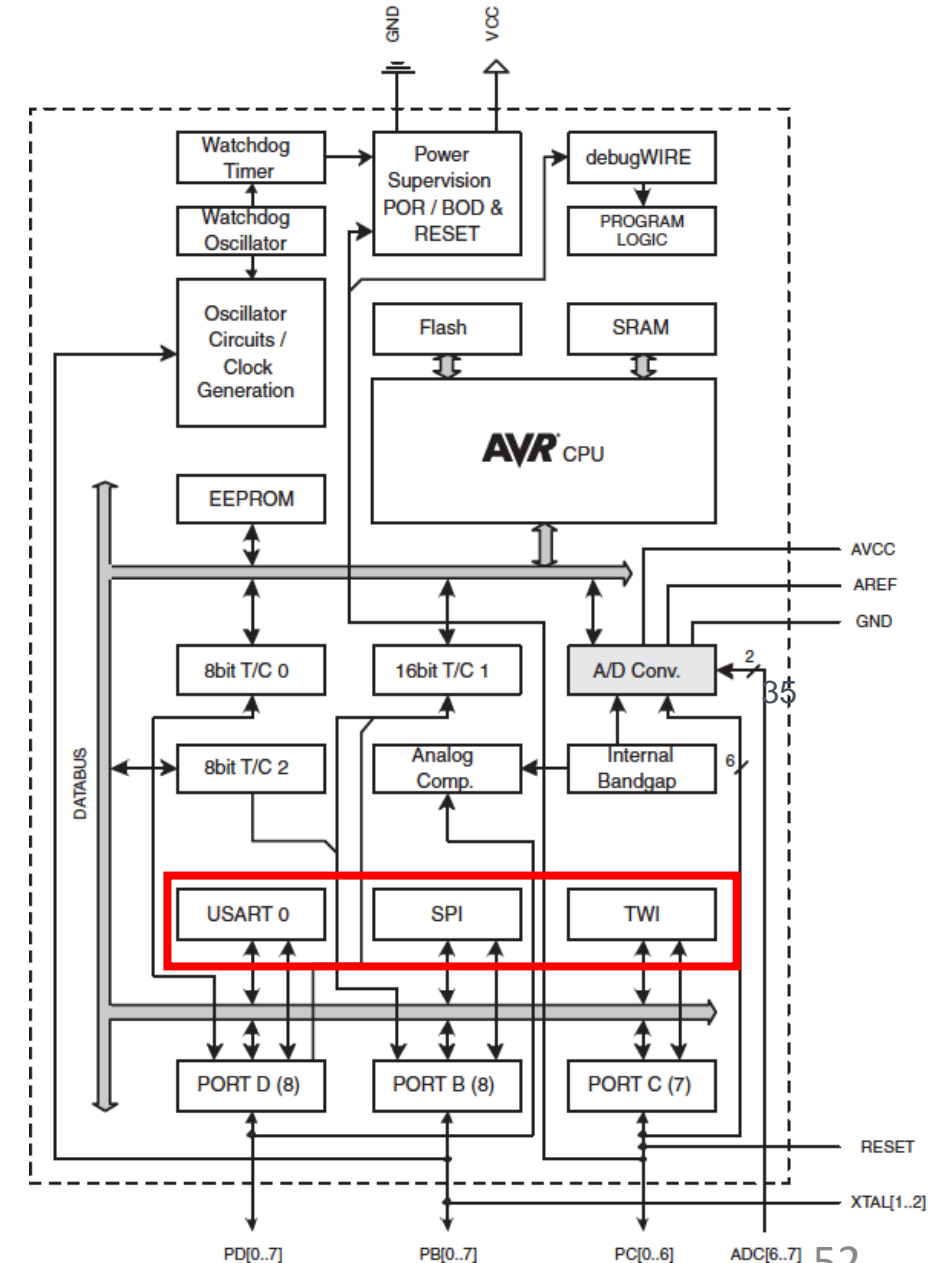
ATmega328 possède trois interfaces de communication:

- UART: Universal Asynchronous Receiver Transmitter
- I2C: Inter Integrated Circuit (ou TWI: Two Wire Interface)
- SPI: Serial Peripheral Interface

› Ces trois interfaces sont des interfaces séries. Les données sont donc envoyées en série (bit/bit)

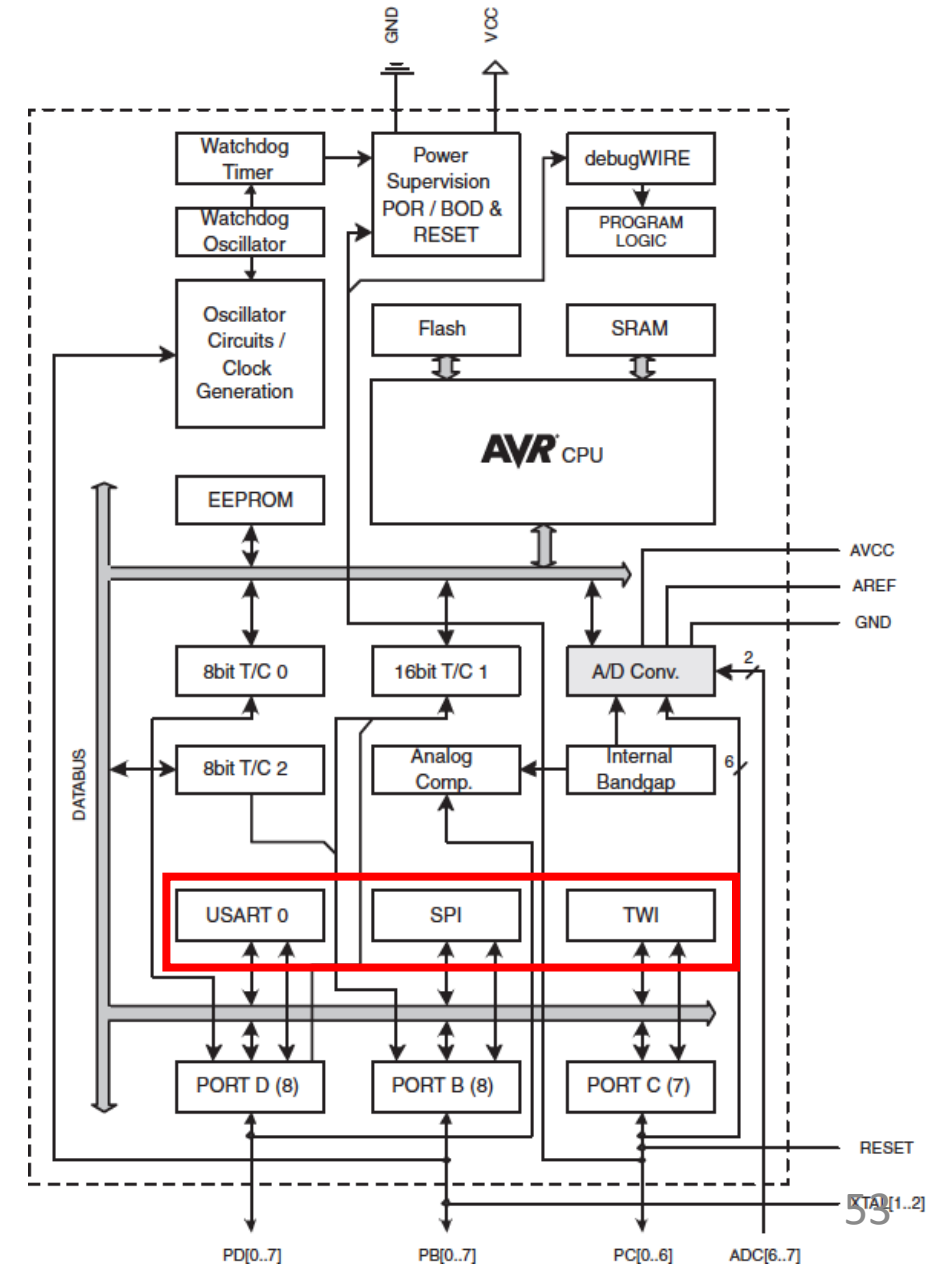
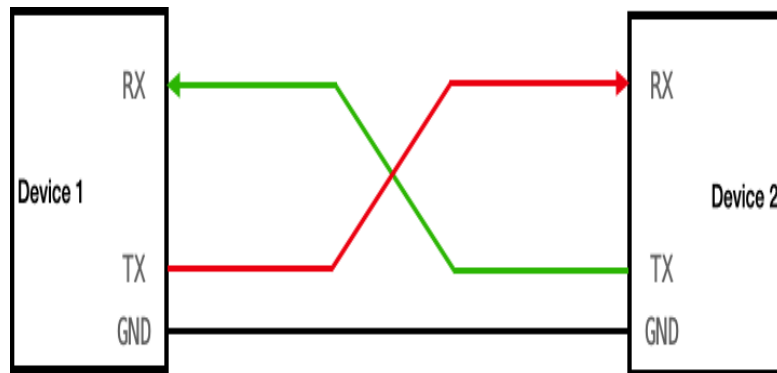
› Chaque interface possède ses propres caractéristiques et ses propres avantages

› Ces interfaces permettent la communication avec d'autres composants du système (capteurs, actionneurs, mémoire externe, etc.)



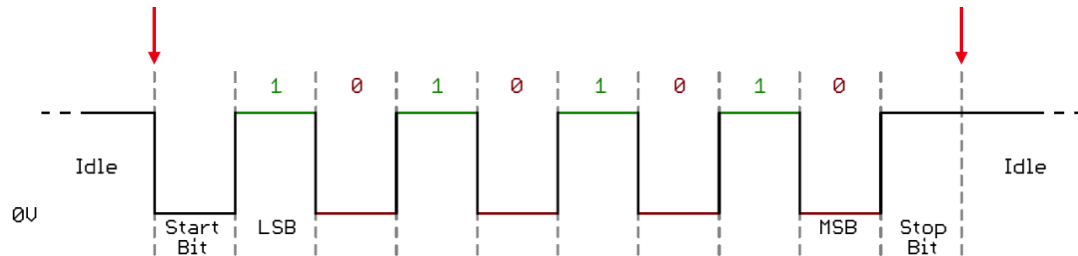
Architecture microcontrôleur ATmega328: Interface UART

- › L'UART permet la communication série asynchrone entre 2 systèmes en Full Duplex (émission et réception en même temps)
- › L'UART utilise 3 fils:
 - Tx: Transmission
 - Rx: Réception
 - GND: masse commune
- › Les lignes Tx et Rx doivent être croisés pour avoir la communication:



Architecture microcontrôleur ATmega328: Interface UART

› La transmission de données n'est pas synchronisée \Rightarrow l'émetteur doit envoyer 1 bit START pour signaler le début de la trame et 1 ou 2 bits STOP pour signaler la fin:

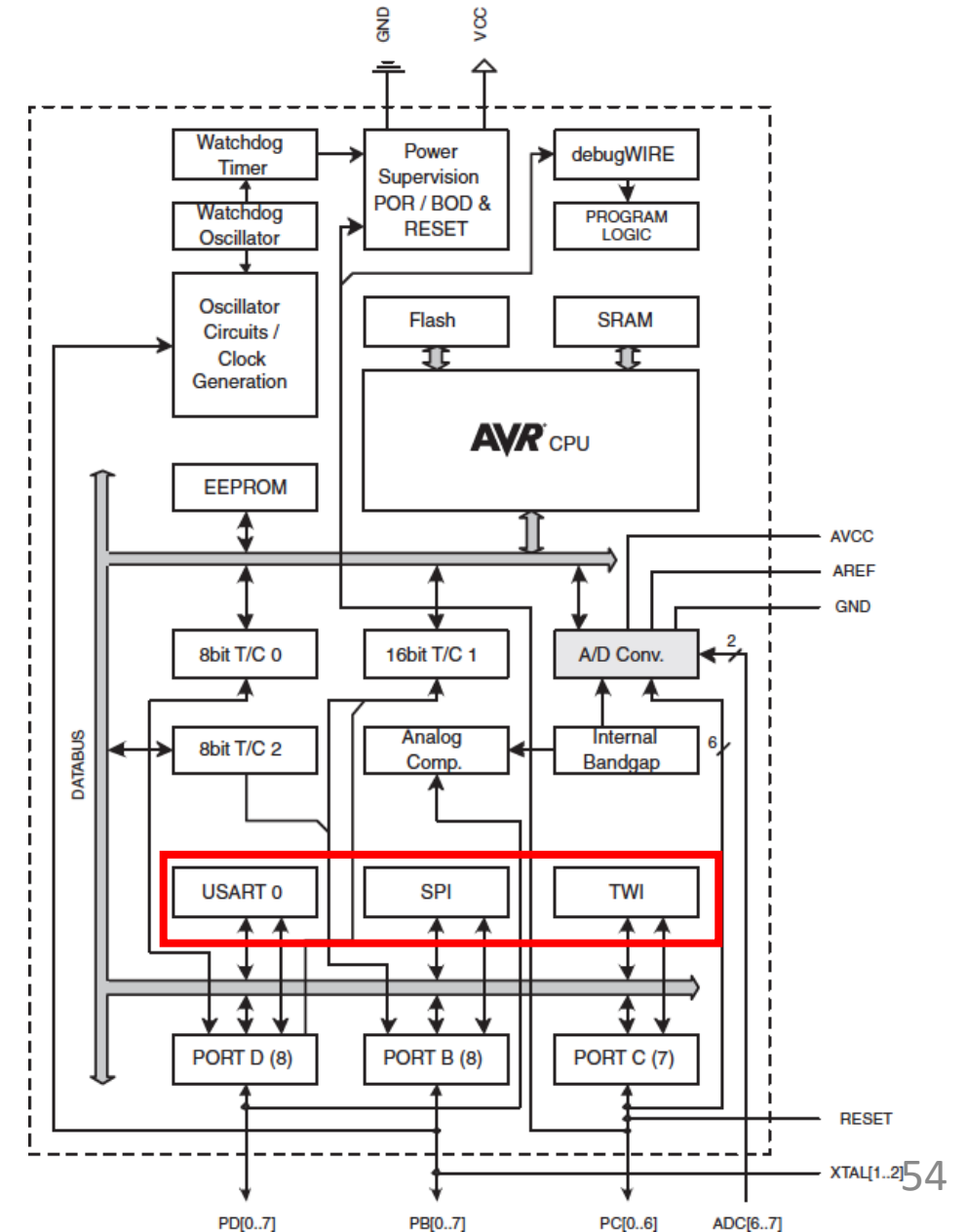


Le récepteur doit se synchroniser sur le bit START de l'émetteur pour recevoir correctement les données

› Les deux systèmes doivent utiliser la même vitesse de transmission exprimée en Baud (ou bits/s)

› Valeurs typiques de vitesse: 1200, 2400, 4800, 9600, 19200, 38400, 57600, ou 115200

› Un bit de Parité peut être ajouté avant les bits STOP pour s'assurer de la transmission correcte des données



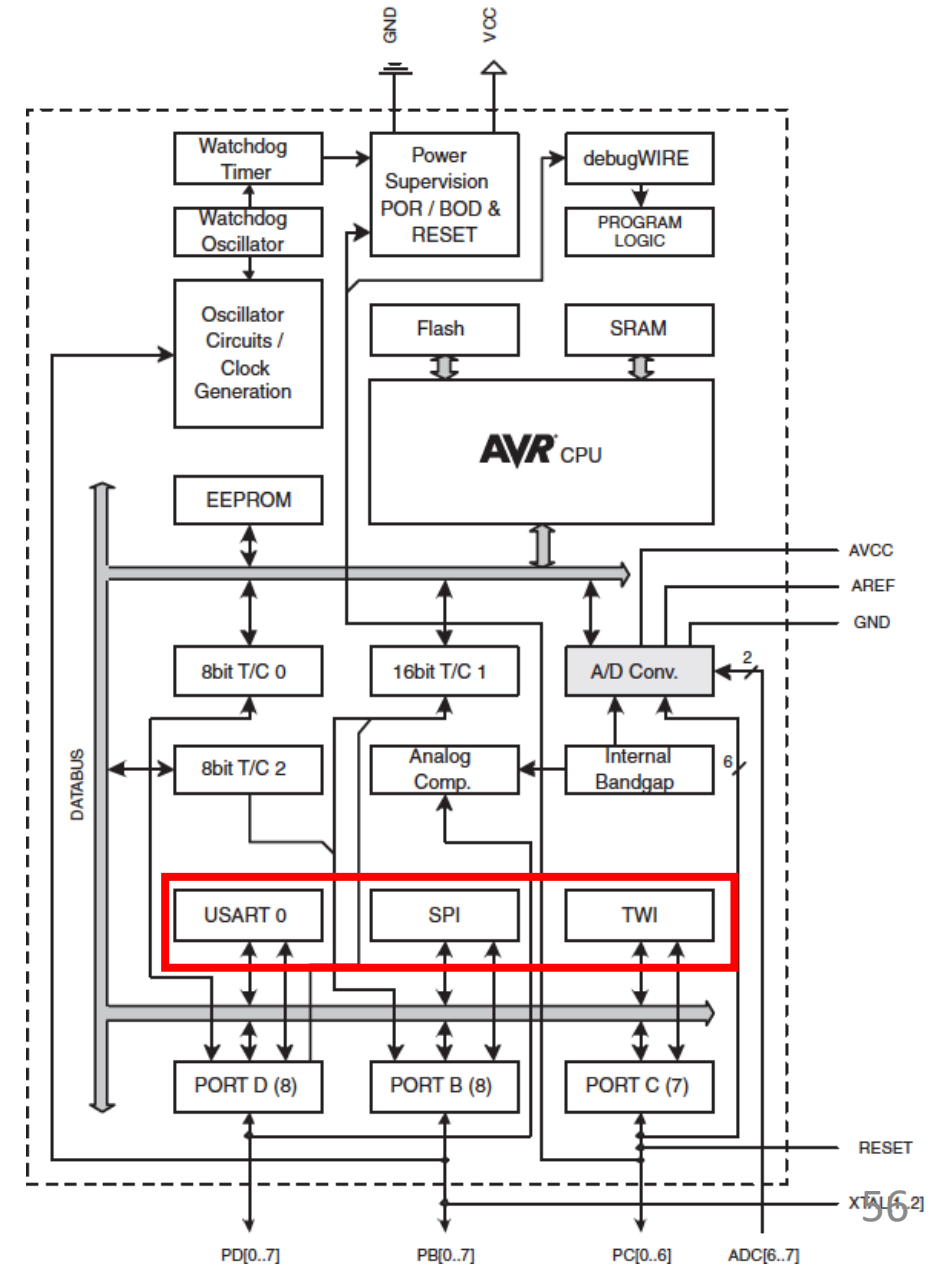
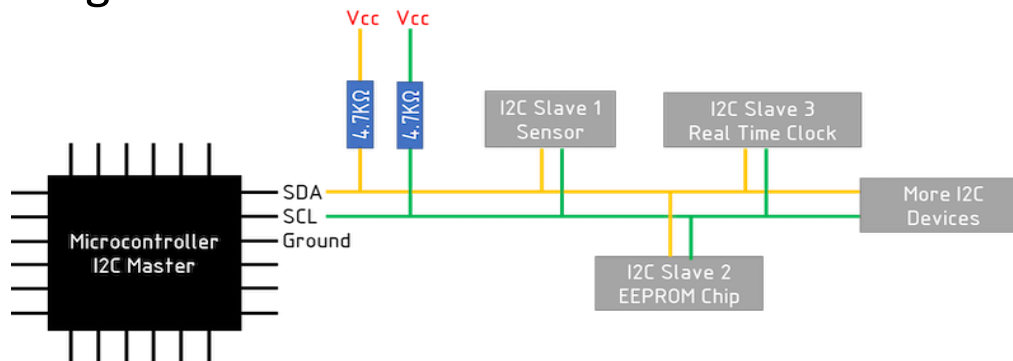
Architecture microcontrôleur ATmega328: Interface UART

- › Pour Arduino Uno: Rx=pin 0 et Tx=pin 1
- › Ces 2 pins sont aussi connectés au port USB pour pouvoir communiquer avec le PC à travers le moniteur série
- › Principales fonctions:
 - Initialisation du port série: `Serial.begin(speed)`
 - Ecriture d'une valeur (numérique ou chaîne de caractères) sur le port série: `Serial.print(value)` ou `Serial.println(value)`
 - Vérifier s'il y a des données reçues, et retourner le nombre d'octets disponibles: `nbBytes=Serial.available()`
 - Lecture d'un caractère du port série: `value=Serial.read()`
 - Lecture de plusieurs caractères: `Serial.readBytes(bytes, nbr)`
 - Lecture d'un entier du port série: `intValue=Serial.parseInt()`
 - Lecture d'un float: `floatValue=Serial.parseFloat()`
- › Exemple: lecture d'un entier à partir du port série fonctionnant avec une vitesse 115200 baud et affichage de la valeur entrée

```
void setup() {  
  Serial.begin(115200);  
}  
void loop() {  
  if (Serial.available() > 0) {  
    long val = Serial.parseInt();  
    Serial.print("Valeur=");  
    Serial.println(val);  
  }  
}
```

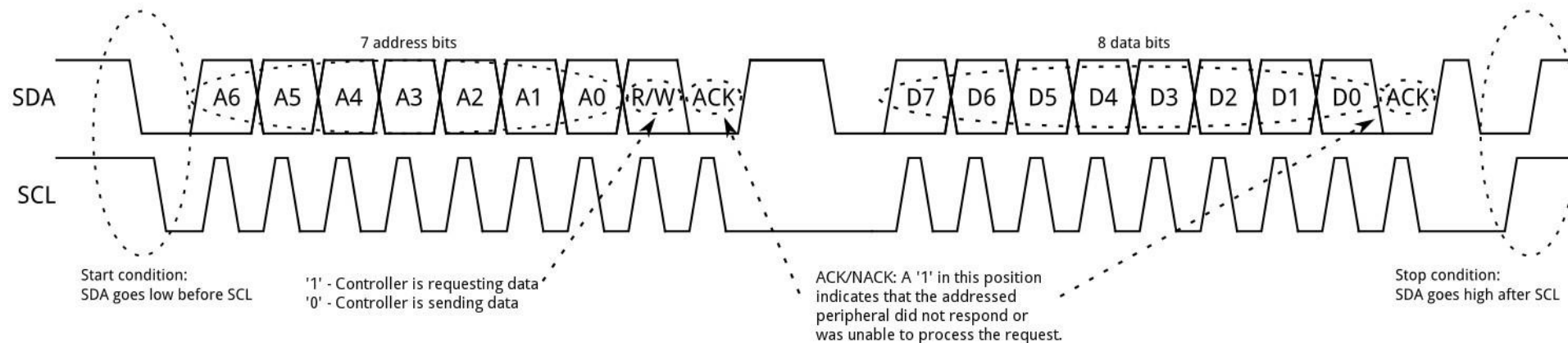
Architecture microcontrôleur ATmega328: Interface I2C

- › L'I2C permet la communication série synchrone entre 2 systèmes ou plus en Half Duplex (émission et réception ne se font pas en même temps)
- › L'I2C utilise 3 fils:
 - SCL: Serial Clock: horloge de synchronisation
 - SDA: Serial Data: ligne de transmission des données (bidirectionnelle)
 - GND: masse commune
- › La communication I2C se fait entre un Maître (Master) et un ou plusieurs Esclaves (Slaves) connectés en parallèle
Remarque: Il faut toujours utiliser des résistances de PullUp externes sur les lignes SCL et SDA



Architecture microcontrôleur ATmega328: Interface I2C

- › Le Maître génère le signal d'horloge (SCL)
- › Les données sont envoyées sous forme de Trames sur la ligne SDA du Maître vers l'Esclave ou inversement
- › La communication se fait toujours entre le Maître et un seul Esclave à la fois, sélectionné par son adresse unique sur 7 bits
- › La trame est constituée par: 1) condition Start; 2) Adresse Esclave; 3) Bits de contrôle (R/W) et acquittement (ACK/NACK); 4) les données (par ensembles de 8 bits); 5) Acquittement (ACK/NACK); 6) condition Stop
- › Deux vitesses sont possibles pour l'I2C sur Arduino Uno:
 - Standard Mode: 100 kbits/s
 - Fast Mode: 400 kbits/s



Architecture microcontrôleur ATmega328: Interface I2C

Pour Arduino Uno: SDA=pin A4, SCL=pin A5

› Bibliothèque à utiliser: `#include`

› Principales fonctions à utiliser:

- Initialisation du port I2C: `Wire.begin()`
- Envoi de la donnée data vers l'adresse address (Master → Slave):
`Wire.beginTransmission(address)` `Wire.write(data)`
`Wire.endTransmission()`
- Lecture d'un nombre nbrBytes d'octets de données (Slave → Master):
`Wire.requestFrom(address, nbrBytes)`
`value = Wire.read()`
- Lecture du nombre d'octets disponibles après appel de
`requestFrom: nbrBytes=Wire.available()`
- Changement de la vitesse de transmission (par défaut 100kb/s):
`Wire.setClock(clockFrequency)` avec `clockFrequency=100000` ou
`clockFrequency=400000`

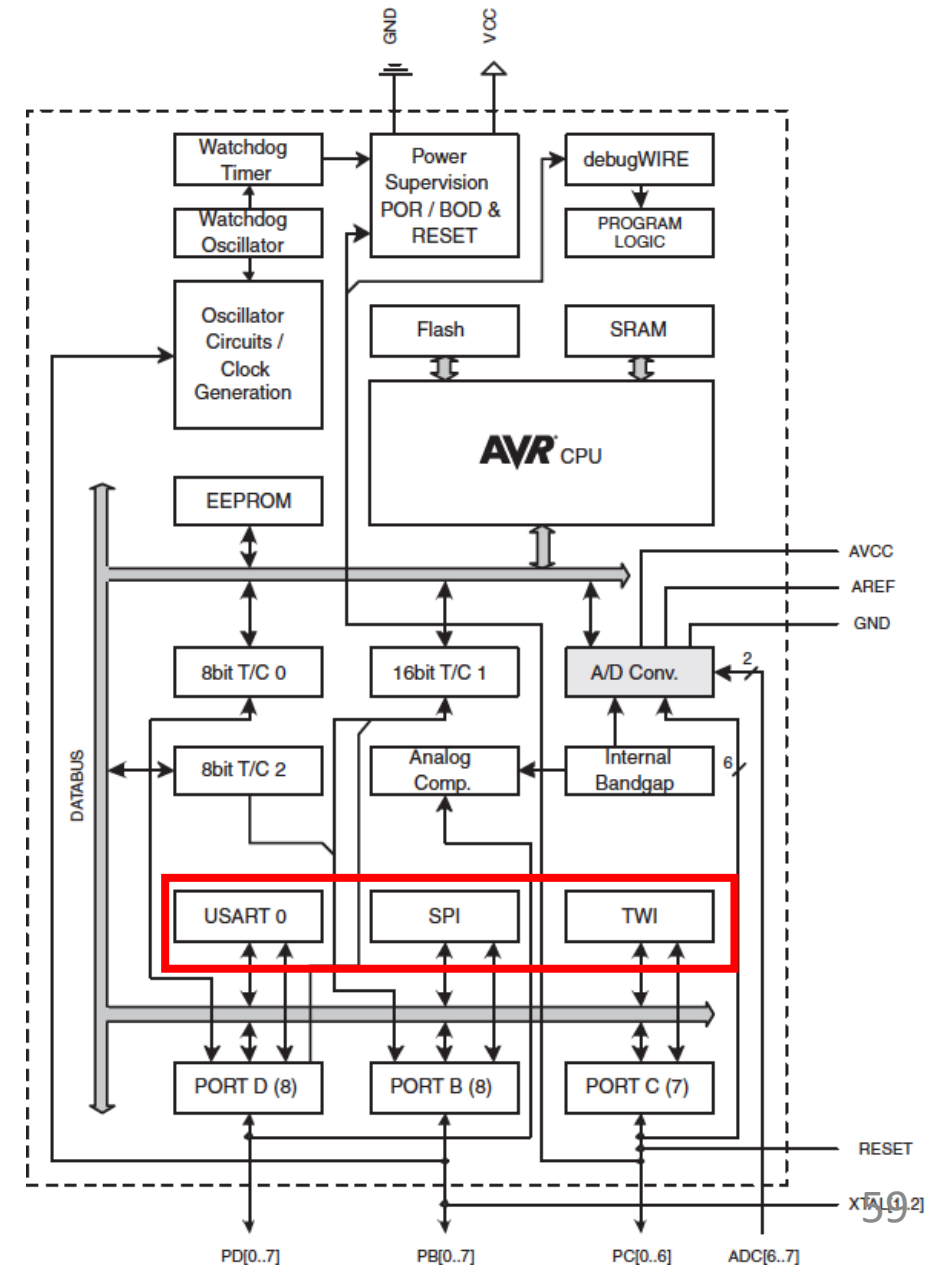
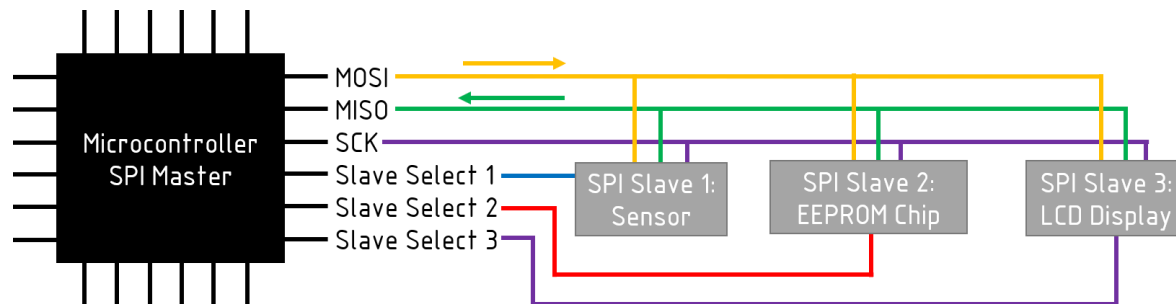
Exemple: Envoi de la valeur 7 au composant d'adresse 0x68 et réception de 12 octets puis affichage sur moniteur série:

```
#include <Wire.h> void
setup() {
Wire.begin();
Serial.begin(9600);
}
void loop() {
Wire.beginTransmission(0x68);
Wire.write(7);
Wire.endTransmission();
Wire.requestFrom(0x68, 12);
while(Wire.available()) {
char c = Wire.read();
Serial.println(c);
}
delay(500);
}
```

Architecture microcontrôleur ATmega328: Interface SPI

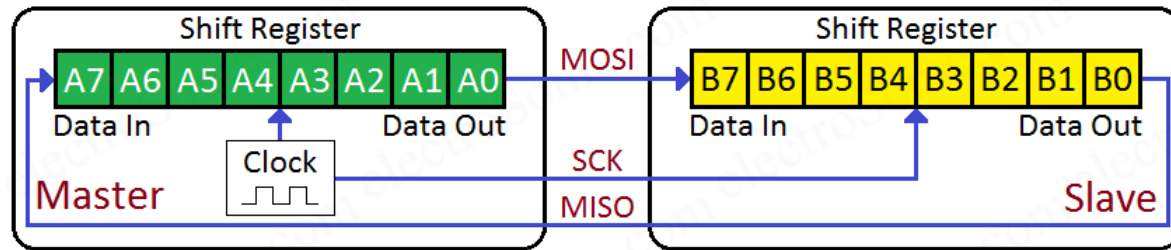
Le SPI permet la communication série synchrone entre 2 systèmes ou plus en Full Duplex (émission et réception en même temps)

- › La communication SPI se fait entre un Maître (Master) et un ou plusieurs Esclaves (Slaves) connectés en parallèle
- › Le SPI utilise 4 fils (en plus du GND):
 - SCLK: Serial Clock: horloge de synchronisation
 - MOSI: Master Out Slave In: transmission de données Master → Slave
 - MISO: Master In Slave Out: transmission de données Slave → Master
 - SS: Slave Select: ligne de sélection du Slave pour la communication
- › Le Master génère le signal d'horloge SCLK Remarque: Le SPI n'utilise pas d'adresse comme le I2C mais le Master met le pin SS du Slave sélectionné à 0



Architecture microcontrôleur ATmega328: Interface SPI

› Le SPI utilise des registres à décalage pour envoyer les données du Master vers le Slave en même temps que les données du Slave vers Master



- › La vitesse de transmission peut se faire à grande vitesse (qqe Mbits/s)
- › Pour Arduino Uno: SCLK=13; MISO=12; MOSI=11; SS=10 ›
- Bibliothèque à utiliser: `#include`
- › Principales fonctions:
 - Initialisation SPI: `SPI.begin()`
 - Transfert de données: `receivedVal = SPI.transfer(sentVal)`

Exemple: Envoi des valeurs 0 à 255 et affichage de la valeur reçus sur moniteur série:

```
#include <SPI.h> void setup()
{
  SPI.begin();
  Serial.begin(9600);
}

void loop() {
  digitalWrite(10, LOW); //SS
  LOW
  for(byte
  val=0;val<=255;val++) { byte
  ret=SPI.transfer(val);
  Serial.println(ret);
  delay(500);
}

  digitalWrite(10, HIGH); //SS
  HIGH
}
```

Résumé: Interface I2C vs SPI

I2C	SPI
Vitesse 100kb/s ou 400kb/s (peut atteindre 3.4Mb/s)	Vitesse généralement > 1Mb/s (peut atteindre 100Mb/s)
Protocole synchrone Half Duplex	Protocole synchrone Full Duplex
Utilise 2 fils uniquement: SCL et SDA	Utilise 4 fils: SCLK, MOSI, MISO, SS
Architecture Master/Slaves (peut supporter des Masters multiples)	Architecture Master/Slaves (ne supporte pas de Master multiples)
Ajout de circuits Slave au système est assez facile (sélection par adresse)	Ajout de circuits Slave au système est plus difficile (sélection par SS)
Acquittement (ACK) à chaque transfert	Pas d'acquittement
Données supplémentaires envoyées (Start, Stop, ACK, Adresse)	Pas de données supplémentaires envoyées