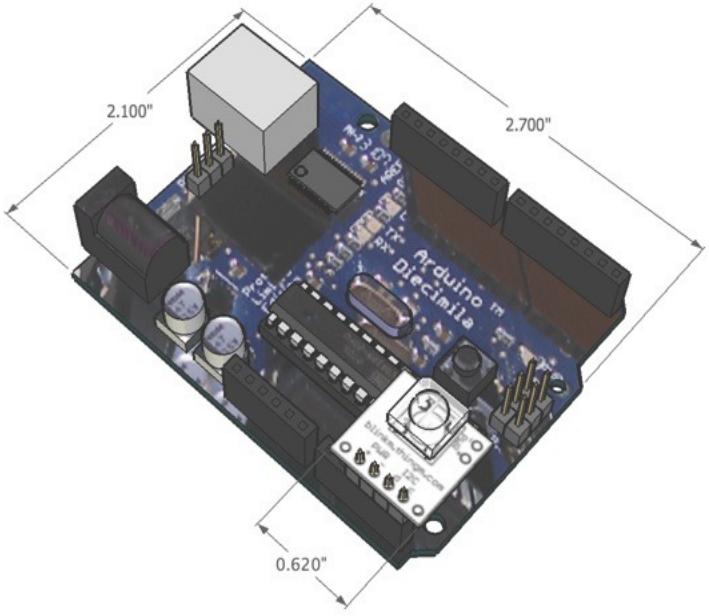
Microcontrôleurs EI3 AGI
2016-2017 ISTIA

Carte ARDUINO UNO
Microcontrôleur ATMega328



B. Cottenceau B311 ISTIA
bertrand.cottenceau@univ-angers.fr

Table des matières

1 Introduction

Le modèle UNO de la société ARDUINO est une carte électronique dont le coeur est un microcontrôleur ATMEL de référence ATMega328. L'ATMega328 est un microcontrôleur 8bits de la famille AVR dont la programmation peut être réalisée en langage C/C++.

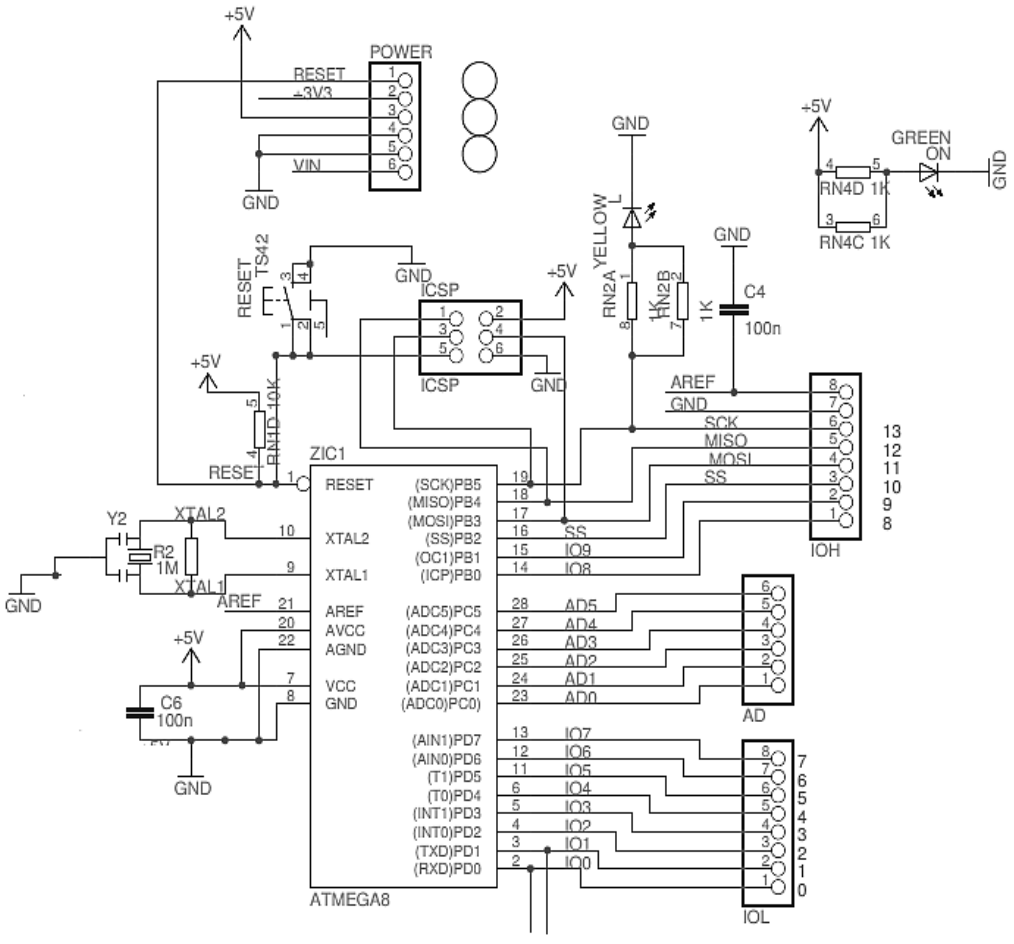
L'intérêt principal des cartes ARDUINO (d'autres modèles existent : Mega, Nano...) est leur facilité de mise en oeuvre. Un environnement de développement (IDE), s'appuyant sur des outils open-source, est fourni. En outre, charger le programme compilé dans la mémoire du microcontrôleur se fait très simplement (via par port USB) dans cet IDE. Enfin, beaucoup de bibliothèques de fonctions sont également fournies pour l'exploitation des entrées-sorties courantes : E/S TOR, gestion des convertisseurs ADC, génération de signaux PWM, exploitation de bus TWI/I^2C, exploitation de servomoteurs, d'afficheurs LCD ...

L'objectif du cours Microcontrôleurs n'est pas simplement de savoir utiliser la carte Arduino UNO. C'est surtout l'occasion d'aborder des problèmes de programmation de bas niveau (la valeur binaire des variables manipulées importe alors beaucoup) et d'apprendre à utiliser le langage C pour cette programmation bas niveau, notamment en sachant gérer des registres/variables "au niveau du bit". Donc quand on se complique la tâche, alors qu'une fonction Arduino existe, dites-vous que c'est voulu.

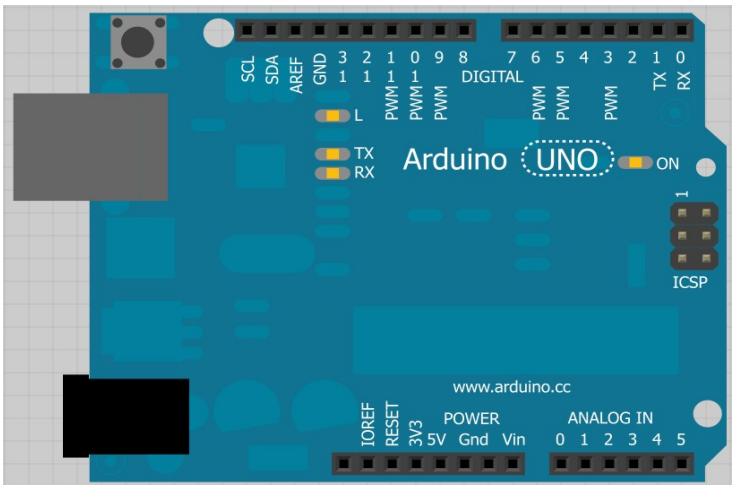
L'objectif de ce document est de mettre en évidence certaines informations techniques concernant l'exploitation des périphériques intégrés, en particulier lorsqu'on n'utilise pas les fonctions "clé en main" d'ARDUINO, dans l'objectif de comprendre comment ça marche !

2 Schéma simplifié de la carte Arduino UNO

Les broches du microcontrôleur sont reliées à des connecteurs selon le schéma ci-dessous.



Vue du dessus, la carte fournit les informations suivantes:



En recoupant avec le schéma précédent, on a les équivalences suivantes :

(connecteur) Numéros 0 à 7	↔	Broches PD0 à PD7 (microcontrôleur)
(connecteur) Numéros 8 à 13	↔	Broches PB0 à PB5 (microcontrôleur)
(connecteur) ANALOG IN 0 à 5	↔	Broches PC0 à PC5 (microcontrôleur)

ATTENTION : avec les fonctions Arduino (pinMode, digitalRead, digitalWrite ...), les signaux sont repérés selon la numérotation des connecteurs (partie gauche). En revanche, lorsque l'on programme en bas niveau, on utilise le nom des registres/des broches du microcontrôleur (partie droite).

digitalWrite(10,HIGH); //Arduino	↔	met la sortie PB2 du microC. à l'état HAUT
analogRead(1); //Arduino	↔	lit l'entrée analogique sur PC1

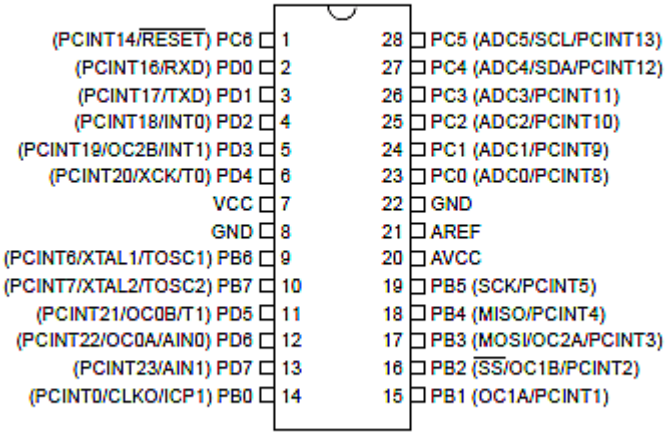
3 Microcontrôleur ATMEL ATMega328

Le microcontrôleur de la carte Arduino UNO est un **ATMega328**. C'est un microcontrôleur ATMEL de la famille AVR 8bits. Les principales caractéristiques sont :

FLASH = mémoire programme de 32Ko
SRAM = données (volatiles) 2Ko
EEPROM = données (non volatiles) 1Ko

Digital I/O (entrées-sorties Tout Ou Rien) = 3 ports PortB, PortC, PortD
(soit 23 broches en tout I/O)

Timers/Counters : Timer0 et Timer2 (comptage 8 bits), Timer1 (comptage 16bits)
Chaque timer peut être utilisé pour générer deux signaux PWM. (6 broches OCxA/OCxB)



Plusieurs broches multi-fonctions : certaines broches peuvent avoir plusieurs fonctions différentes, choisies par programmation. Elles ont alors plusieurs noms sur le brochage (voir ci-avant)

Par exemple, les broches PB1, PB2, PB3, PD3, PD5, PD6 peuvent servir de sortie PWM (Pulse Width Modulation), c'est-à-dire des sorties qui joueront le rôle de sorties analogiques. Elles correspondent aux broches des connecteurs 3,5,6,9,10 et 11. Cet autre rôle possible est lié aux timers et ces broches sont alors appelées OCxA ou OcxB dans la documentation. Ce sont les mêmes broches, mais pour une autre fonction. Si vous regardez à nouveau le brochage, vous constaterez que toutes les broches sont multi-fonctions.

PWM = 6 broches **OC0A(PD6)**, **OC0B(PD5)**, **0C1A(PB1)**, **OC1B(PB2)**, **OC2A(PB3)**, **OC2B(PD3)**

Les broches du PORTC peuvent être converties par un convertisseur Analog toDigital.

Analog to Digital Converter (résolution 10bits) = 6 entrées multiplexées **ADC0(PC0)** à **ADC5(PC5)**

Gestion bus I2C (TWI Two Wire Interface) = le bus est exploité via les broches **SDA(PC5)/SCL(PC4).**

Port série (USART) = émission/réception série via les broches **TXD(PD1)/RXD(PD0)**

Comparateur Analogique = broches AIN0(PD6) et AIN1 (PD7) peut déclencher interruption

Watchdog Timer programmable.

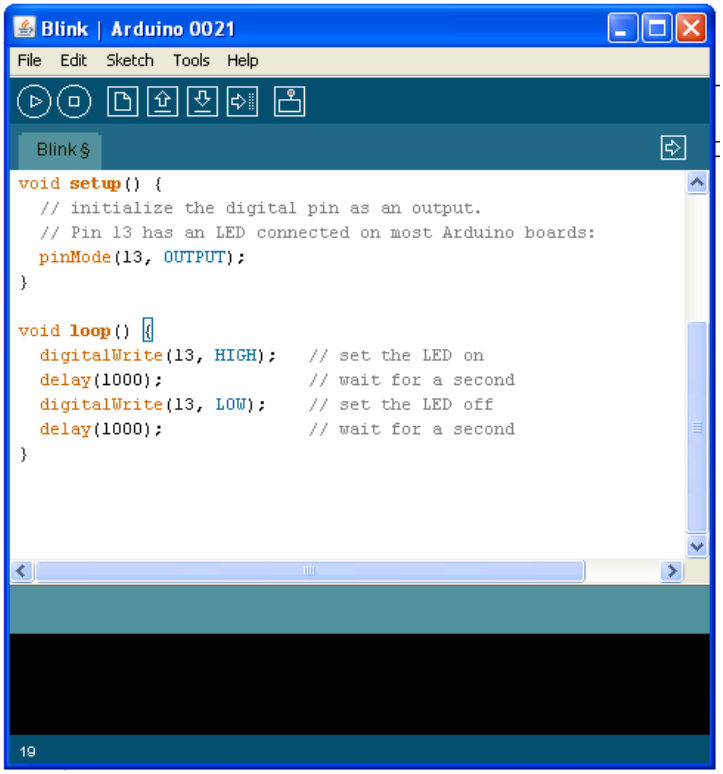
Gestion d'interruptions (24 sources possibles (cf interrupt vectors)**) :** en résumé

- Interruptions liées aux entrées **INT0 (PD2) et INT1 (PD3)**
- Interruptions sur changement d'état des broches **PCINT0** à **PCINT23**
- Interruptions liées aux Timers 0, 1 et 2 (plusieurs causes configurables)
- Interruption liée au comparateur analogique
- Interruption de fin de conversion **ADC**
- Interruptions du port série **USART**
- Interruption du bus **TWI** (**I2C**)

4 Le programmation avec l'IDE Arduino

4.1 Caractéristiques du développement ARDUINO

ARDUINO fournit un environnement de développement (IDE) avec un éditeur de source, les opérations de compilation et de chargement dans la mémoire du microcontrôleur étant ramenées à des clicks sur des boutons dans l'IDE(très simple).



La communication entre le PC et la carte se fait via le port USB, moyennant installation d'un driver adapté (fourni par ARDUINO).

Structure d'un projet ARDUINO

L'outil impose de structurer l'application de façon spécifique. Le compilateur utilisé est AVR GCC (compilateur C/C++ pour processeur AVR).

Le programme principal (fonction main) est imposé, non modifiable, et décrit ci-dessous.

Les seules parties que l'on développe spécifiquement sont :

- **la fonction setup() :** doit contenir les initialisations (times, interrupts...)
- **la fonction loop()** : fonction répétée indéfiniment

```
// PROGRAMME ARDUINO IMPOSE
#include <WProgram.h>

int main(void)
{
     init();   // initialisations ARDUINO pour fonctions
               // utiles : delays, PWM ...

     setup();
     for (;;)  loop();   // répète indéfiniment loop()
     return 0;
}
```

UN PROGRAMME ARDUINO = une fonction setup() + une fonction loop()

4.2 Langage C pour ARDUINO UNO

Variables Types/Taille :

boolean : true/false (8 bits)
char = entier 8 bits signé [-128,+127] (stocke aussi des codes ASCII)
byte / **unsigned char** : entiers 8 bits non signés [0,255]
Note: des constantes sont définies et permettent d'écrire `byte b = B10010;`
int : entiers 16 bits signés [-32768,+32767]
word / **unsigned int** : entier 16 bits non signé [0,65535]
long : entiers 32 bits signé
unsigned long : entiers 32 bits non signés
float /double : flottant format IEEE 32 bits (float et double = même type ici)

```
//Exemple Blink
void setup()
{
   pinMode(13, OUTPUT);  //broche PB5 en sortie
}

void loop()
{
  digitalWrite(13,HIGH);  // set PB5
  delay(200);      // délai 200 ms
  digitalWrite(13,LOW);      // clear PB5
  delay(1000);     // délai 1 s
}
```

L'exemple le plus simple, fourni par ARDUINO, consiste à faire clignoter la LED (sur la carte UNO) connectée à la broche PB5 du microcontrôleur, broche n° 13 sur les connecteurs de la carte.

La fonction setup() configure la broche PB5 (connexion n°13 sur la carte) en sortie, à l'aide de la fonction Arduino pinMode(). La fonction loop() décrit ensuite ce qui sera répété indéfiniment : mettre PB5 à 1 pendant 200ms puis mettre PB5 à 0 pendant 1s, et ainsi de suite.

Quelques fonctions courantes fournies par ARDUINO

Constantes : **HIGH, LOW, INPUT, OUTPUT**

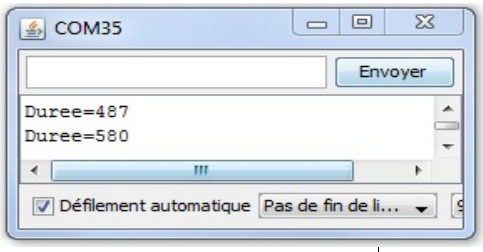
Digital I/O (gestion entrées/sorties binaires)
pinMode(pin,mode) : pin = numéro, mode = INPUT/OUTPUT/INPUT_PULLUP
digitalWrite(pin,value) : pin = numéro, value= HIGH/LOW
int digitalRead(pin) : pin = numéro sur connecteur, retourne la valeur

Temporisations
delay(unsigned long ms) : delai de ms milli secondes avec ms sur 32 bits
delayMicroseconds(unsigned int ms) : delai de ms microsecondes avec ms sur 16 bits
unsigned long micros() : nombre de micro secondes depuis démarrage. Revient à zéro tous les 70mn environ.
unsigned long millis() : nombre de milli secondes depuis démarrage. Revient à zéro tous les 50 jours environ.

Liaison Série : ces fonctions permettent de lire/écrire sur la liaison série du microcontrôleur et donc de communiquer avec le PC auquel il est relié (via cordon USB).

Serial.begin(9600); // configure la liaison à 9600 bits/s
Serial.println(v,f); // envoie v au format f (DEC, HEX, BIN)

L'exemple ci-dessous utilise exclusivement des fonctions Arduino pour la gestion du temps et des entrées-sorties.




```
//Exemple :
// indique la durée (millis) où le signal
// sur PC5 vaut 0. Cette durée est envoyée
// au PC via le cordon USB et peut être
// affichée dans le Moniteur Série

void setup()
{
   Serial.begin(9600);         //UART microC à 9600 bauds
   pinMode(A5,INPUT_PULLUP);  // PC5 en entrée (avec pull-up)
   pinMode(13,OUTPUT);         // PB5 (LED jaune) en sortie
   digitalWrite(13,LOW);       // PB5 à 0 (LED éteinte)
}

void loop()
{
  unsigned long t1,t2,duree; // variables locales
  while(digitalRead(A5)==1);
  // PC5 passe de 1->0  (front descendant)
  t1=millis();
  digitalWrite(13,HIGH);    // allume LED
  while(digitalRead(A5)==0);
  // PC5 passe de 0->1  (front montant)
  t2=millis();
  digitalWrite(13,LOW);     // éteint LED
  duree = t2-t1;
  Serial.print("Duree=");
  Serial.println(duree,DEC);  // envoie la durée
}
```

Exploitation des registres de l'ATMega328 dans les programmes

Lorsque l'on souhaite contrôler les périphériques Arduino au plus bas niveau, c'est-à-dire sans utiliser les - pourtant bien sympathiques - fonctions Arduino, on doit lire/écrire dans des registres internes du microcontrôleur. Ceux-ci sont détaillés, pour certains périphériques, dans la suite de ce document. A noter que la documentation complète de ce microcontrôleur fait plusieurs centaines de pages.

Par exemple, faire clignoter la LED de la carte Arduino (PB5), sans utiliser les fonctions ARDUINO, nécessite l'accès aux registres de configuration des ports E/S (voir section 5.2). Pour la broche PB5, les registres impliqués sont DDRB, PORTB et PINB. Dans le programme C, les registres sont référencés par leur nom en MAJUSCULE.

Exemple de gestion bas-niveau de la broche PB5.

```
// Exemple BLINK SANS FONCTION ARDUINO

//    LED <- PB5 : LED allumée quand PB5=1

void setup() {
      // configurer broche PB5 en sortie (voir section 5.2)
  DDRB |= 0x20;   // DDRB.5 <- 1
      // ou DDRB|=B100000;  // parce que B100000 vaut 0x20
      // ou DDRB|=32;       // parce que 0x20 vaut 32
  PORTB &= 0xDF;  // PORTB.5 <- 0
      // ou DDRB&= ~0x20;  // 0xDF est le complément de 0x20
}

void loop() {
  PORTB |= 0x20;  // PORTB.5 <- 1
  delay(200); // 200 ms
  PORTB &= 0xDF;; // PORTB.5 <- 0
  delay(1000); // 1s
}
```

Remarques :

Les exemples ATMega328 en langage C trouvés sur internet utilisent des "styles" d'écriture parfois déroutants. Notamment s'agissant de la gestion des ports I/O, on doit souvent faire des opérations logiques (&,|,~,^) pour mettre des bits à 0, 1 ou inverser des bits, c'est-à-dire pour modifier ou lire un ou plusieurs bits d'un registre 8 bits.

Exemple déroutant:

```
PORTB &= ~(1<<PORTB5);  // Mettre le bit 5 de PORTB à 0
```

Explication : PORTB5 est une contante qui vaut 5.

`1<<PORTB5` signifie "décaler la valeur (0000 0001)b de 5 bits vers la gauche"

Donc, `(1<<PORTB5)` représente (0010 0000)b

Ensuite `~(1<<PORTB5)` représente l'inverse bit à bit soit (1101 1111)b

Enfin, `PORTB &= ~(1<<PORTB5)` signifie `PORTB = PORTB & ~(1<<PORTB5)`

L'opération ET logique laisse inchangés tous les bits de PORTB **sauf le bit 5** qui est mis à 0 (en raison du bit à 0 de la valeur binaire (1101 1111)b = ~(1<<PORTB5)).

En résumé, les expressions suivantes réalisent le même traitement

```
PORTB &= ~(1<<PORTB5);
PORTB &= ~0x20;            // parce que (20)h=(100000)b
PORTB &= 0xDF;             // parce que (DF)h est le complément de (20)h
PORTB &= ~B00100000;       // parce que  B00100000 (constante) vaut 0x20
PORTB &= B11011111;        // idem B11011111 (constante)
```

5 Structure interne de l'ATMega328 (extraits de documentations ATMEL)

L'utilisation des périphériques intégrés (Entrées Sorties TOR, Timers, ...) repose sur l'exploitation (lecture/écriture) de registres internes. Ces registres, essentiellement 8 bits, sont décrits par un nom en MAJUSCULE dans les programmes en C. Cette section fournit quelques détails importants sur les registres internes du microcontrôleur ATMega328 impliqués dans l'exploitation des périphériques. Certaines parties sont des extraits (en langue anglaise) de la documentation Atmel.

Pour la documentation complète (442p) : chercher avec mots clés datasheet ATMega328

Notation : par la suite, pour un registre nommé **R**, la notation **R.n** désigne le bit de rang n du registre R. Attention, ce n'est qu'une notation. En revanche, le compilateur C ne peut pas exploiter cette notation.

Ex: PORTB.5 signifie "le bit numéro 5 du registre qui s'appelle PORTB"

```
PORTB.5=1;        // KO, pas de sens en langage C !
PORTB |= 0x20;    // OK: bit PORTB.5 à 1 sans changer les autres
```

5.1 Status Register (SREG)

Le registre **SREG** contient des indicateurs liés aux opérations et le bit d'autorisation générale des interruptions. Les bits de ce registre sont : **Z** (Zero), **C** (Carry), **S** (Sign) ...
Le bit d'activation général du système d'interruptions est le bit **I** (**SREG.7**)

SREG – AVR Status Register

The AVR Status Register – SREG – is defined as:

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	

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Note : en langage C, ce bit I est modifiable via les appels `sei()` (set IT) `cli()` (Clear IT)

5.2 Digital I/O Entrées Sorties Binaires/Tout Ou Rien (TOR)

Les microcontrôleurs disposent de broches d'entrée/sortie TOR, comme sur un automate programmable industriel. Pour placer l'état d'une sortie à 0 ou 1, ou lire l'état d'une entrée, il faut

exploiter des registres internes décrits ci-dessous.

Les entrées-sorties sont réparties dans 3 groupes de broches appelés *ports*. Le port B regroupe les broches notées PBx, le port C les broches PCx et le port D les broches PDx (voir brochage). Chaque port est exploité grâce à 3 registres.

Ex: `PORTB`, `DDRB` et `PINB` les registres pour la gestion des broches PB0 à PB7

PORTx = pour l'*ECRITURE* de valeurs en sortie
DDRx = détermine la *DIRECTION* de chaque broche du port (1-sortie 0-entrée)
PINx = pour la *LECTURE* de la valeur en entrée

DDRx = Direction du port x

Ex**:** si le bit **DDRB.6** vaut 1 alors la broche **PB6** est une sortie (TOR)

```
// Exemple config. ports
void setup()
{
  DDRB |= 0x40;   // DDRB.6 <- 1  <-> PB6 sortie
  DDRD &= ~0x08;  // DDRD.3 <- 0  <-> PD3 entrée
}
```

PORTx pour l'écriture des sorties TOR : si une broche est configurée en sortie (DDRx.n=1) alors l'écriture du bit PORTx.n permet de définir l'état de la sortie (0 ou 1).

```
// Exemple ecriture de sortie
void setup()
{
  DDRB |= 0x40;    // config PB6 en sortie
  PORTB &= ~0x40;  // PORTB.6 <-0 (écrire PB6 à 0)
}
```

PINx pour la lecture des entrées TOR : si une broche est configurée en entrée (DDRx.n=0) alors la lecture du bit PINx.n permet de connaître l'état de l'entrée.

```
// Exemple lecture d'entrée
void setup()
{
  DDRD &= ~0x08;  // config. PD3 entrée
}
void loop()
{
      if((PIND&0x08)!=0)  // si entrée PD3 à 1
      {
            ...
      }
}
```

Dans la documentation, les registres impliqués sont décrits ci-dessous (exemple pour les broches PB0 à PB7).

MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	BODS	BODSE	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See "Configuring the Pin" on page 76 for more details about this feature.

PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
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	

DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
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	

PINB – The Port B Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
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	

Gestion des résistances pull-up internes (important)

En technologie MOS, une entrée "en l'air" a un état indeterminé. Aussi, lorsqu'on veut exploiter des boutons poussoir/switchs, on les branche de façon à ramener l'entrée à 0 quand on ferme le contact. A l'inverse, lorsque le contact est ouvert, l'état de l'entrée doit être amené à 1 par des resistances de tirage à 1 (***pull-up*** en anglais). Ces résistances sont activées ou non par programmation:

PORTx.n=**1** ET **DDRx**.n=**0** ↔ **pull-interne de Pxn activée**
PORTx.n=0 OU **DDRx**.n=1 ↔ pull-interne de Pxn désactivée

```
// Config d'une entrée AVEC PULL-UP interne
void setup()
{
  DDRD  &= ~0x02;    // DDRD.1 <-0  (PD1 en entrée)
  PORTD |= 0x02;     // PORTD.1<-1  (pull-up sur PD1)
}
```

Détail : le bit **PUD** du registre MCUCR permet la désactivation de toutes les resistances de pull-up.

Si **PUD**=1 ALORS toutes les resistances de pull-up -interne de tous les ports désactivées

6 Sources d'interruption exploitables sur ATMega328 (carte Arduino UNO)

Interruption (IT) = suspension du programme en cours pour exécuter un traitement particulier. C'est différent d'une fonction : ce n'est pas appelé explicitement par le programme (on n'écrit pas l'appel de ce traitement dans notre source). C'est le processeur qui, suite à la détection d'une cause particulière, déclenche le traitement de l'interruption. La fonction associée à une interruption est appelée *fonction d'interruption* ou *routine d'interruption* (**Interrupt Service Routine ISR)**.

Les périphériques peuvent conduire à déclencher des interruptions. C'est ce qu'on va voir ici. Ce mécanisme permet d'assurer des temps de réponse très courts entre la cause de l'interruption et son traitement.

Important : les appels des ISR s'insèrent de façon *asychrone* (on ne sait pas à l'avance quand elles vont être appelées) dans l'exécution du programme. Par exemple, un programme Arduino est interrompu chaque milli seconde par une ISR liée au Timer 0. Cette routine ISR Timer 0 met à jour les variables de temps utilisées par les fonctions delay() millis() etc. Périodiquement, un programme arduino est donc suspendu (interrompu) pour l'ISR Timer 0, ce qui ralentit l'exécution (de 6%).

Ci-dessous, le vecteur d'interruptions, c-à-d toutes les sources (causes possibles) .

11.4 Interrupt Vectors in ATmega328P

Table 11-6. Reset and Interrupt Vectors in ATmega328P

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 COMPA	Timer/Counter2 Compare Match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x001A	TIMER1 OVF	Timer/Counter1 Overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x001E	TIMER0 COMPB	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

Table 11-1. Reset and Interrupt Vectors in ATmega48PA (Continued)

Vector No.	Program Address	Source	Interrupt Definition
24	0x017	ANALOG COMP	Analog Comparator
25	0x018	TWI	2-wire Serial Interface
26	0x019	SPM READY	Store Program Memory Ready

6.1 Interruptions Externes (liées aux entrées PD2 et PD3)

Il s'agit d'interruptions où les *causes* sont liées à des niveaux ou à des changements d'états des broches PD2 (INT0) ou PD3 (INT1) du microcontrôleur. Notez bien le nom INT0/INT1 représentant cette fonction alternative des broches PD2/PD3. Pour ce rôle d'interruption externe, les broches doivent être configurées en entrée (**cf. 5.2 DIGITAL I/O**).

Broches INT0 (PD2)/INT1(PD3) : configurables pour déclencher les interruptions (n° 2 et 3 dans le vecteur ou INT0_vect/INT1_vect). Les causes possibles (choisies par programmation) sont

- détection d'un niveau 0 sur l'entrée (low level),
- front négatif ou positif (falling/rising edge).
- changement d'état quelconque (any logical change)

Choix de la cause d'interruption : ***ce qui est écrit dans le registre EICRA***
les bits EICRA.1-EICRA.0 pour INT0 (voir table 12.2)
les bits EICRA.3-EICRA.2 pour INT1 (même table pour ISC11-ISC10)

EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

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

Table 12-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Activation des interruptions INT0/INT1 = bit SREG.7 à 1 et mise à 1 de EIMSK.0/EIMSK.1

EIMSK – External Interrupt Mask Register

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

```
// Config INT0 et INT1 pour IT externes
void setup()
{
  cli();            // aucune IT possible (SREG.7<-0)
  EICRA &= 0xF0;    // reset bits EICRA.3-EICRA.0
  EICRA |=0x09;     // ISC11=1 ISC10=0  (IT sur front ↓ sur INT1)
                    // ISC01=0 ISC00=1 (IT sur front ↑ ou ↓ INT0)
  EIMSK |=0x03;     // IT INT0 et INT1 activées
  sei();            // IT possibles SREG.7<-1
}
```

Détail : Flags internes = lorsqu'une cause d'IT est détectée, un flag interne de EIFR est positionné

EIFR – External Interrupt Flag Register

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

Nom des sources d'interruption dans un programme Arduino (pré-définis)

```
#define INT0_vect          _VECTOR(1)   /* External Interrupt Request 0 */
#define INT1_vect          _VECTOR(2)   /* External Interrupt Request 1 */
#define PCINT0_vect        _VECTOR(3)   /* Pin Change Interrupt Request 0 */
#define PCINT1_vect        _VECTOR(4)   /* Pin Change Interrupt Request 1 */
#define PCINT2_vect        _VECTOR(5)   /* Pin Change Interrupt Request 2 */
#define WDT_vect           _VECTOR(6)   /* Watchdog Time-out Interrupt */
#define TIMER2_COMPA_vect  _VECTOR(7)   /* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect  _VECTOR(8)   /* Timer/Counter2 Compare Match A */
#define TIMER2_OVF_vect    _VECTOR(9)   /* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect   _VECTOR(10)  /* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect  _VECTOR(11)  /* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect  _VECTOR(12)  /* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect    _VECTOR(13)  /* Timer/Counter1 Overflow */
#define TIMER0_COMPA_vect  _VECTOR(14)  /* TimerCounter0 Compare Match A */
#define TIMER0_COMPB_vect  _VECTOR(15)  /* TimerCounter0 Compare Match B */
#define TIMER0_OVF_vect    _VECTOR(16)  /* Timer/Couner0 Overflow */
#define SPI_STC_vect       _VECTOR(17)  /* SPI Serial Transfer Complete */
#define USART_RX_vect      _VECTOR(18)  /* USART Rx Complete */
#define USART_UDRE_vect    _VECTOR(19)  /* USART, Data Register Empty */
#define USART_TX_vect      _VECTOR(20)  /* USART Tx Complete */
#define ADC_vect           _VECTOR(21)  /* ADC Conversion Complete */
#define EE_READY_vect      _VECTOR(22)  /* EEPROM Ready */
#define ANALOG_COMP_vect   _VECTOR(23)  /* Analog Comparator */
#define TWI_vect           _VECTOR(24)  /* Two-wire Serial Interface */
```

Exemple : interruptions externes (programme complet config. + ISR)

`ISR(nom) { ... }` = **fonction d'IT associée à une cause d'IT, `nom` doit être pris dans la table ci-dessus.**

```
// Exemple : exploitation IT Externe 0
ISR(INT0_vect)  // ISR INT0 = traitement si interruption INT0
{
      PORTB ^= 0x20;  // bascule PORTB.5
}
void setup(){  // configuration pour exploiter IT INT0
      cli();
      Serial.begin(9600);
      DDRB |= 0x20;          // PB5 en sortie
      PORTB &= ~0x20;        // PORTB.5 <-0 (sortie PB5 à 0)
      DDRD &= ~0x04;         // PD2 en entrée
      PORTD |= 0x04;         // PORTD.2=1 <-> activer pull-up
      EICRA = 0x02;     // IT si front ↓sur INT0 (table 12-2)
      EIMSK |= 1;       // source INT0 activée
      sei();            // active les IT en général
}
int cpt=0;  // variable globale (hors fonction)
void loop() { // ce traitement peut être interrompu pour INT0
      Serial.println(cpt,DEC);
      cpt++;
      delay(1000);
}
```

Pour le programme précédent, on branche un bouton poussoir entre PD2(/INT0) et la masse GND. Quand on enfonce le bouton poussoir, on amène le niveau de l'entrée PD2 à 0. On utilise ici la résistance de pull-up interne pour retirer le niveau à 1 quand le bouton est relâché.

Principe : chaque fois que le bouton poussoir fait passer le niveau de 1 à 0 sur l'entrée INT0(PD2), la fonction d''interruption associée à INT0 est exécutée. Cette action a pour effet d'inverser l'état de la LED et de revenir au programme principal. Il est important de comprendre que l'interruption ne dure que quelques microsecondes. En dehors de ces interruptions, le programme principal (fonction loop()) envoie chaque seconde la valeur de la variable cpt.

Remarque : cette façon de découpler le traitement du bouton poussoir de celui du programme principal s'apparente à de la parallélisation de tâches.

[Point technique] Mettre à 0 flag dans registre EIFR : les bits du registre EIFR indiquent (flags) qu'une demande d'interruption INT0/INT1 est en attente : un flag à 1 dans ce registre signifie que la cause d'IT a été détectée mais que la routine ISR n'est pas encore exécutée. Si l'on souhaite annuler une demande d'IT (avant son exécution), on doit remettre ces flags à 0. Bizarrerie : pour annuler une demande (clear flag), il faut écrire 1 (et pas 0) dans le registre EIFR pour le flag concerné

```
EIFR|=1;  // annuler demande IT INT0
EIFR|=2;  // annuler demande IT INT1
```

6.2 Interruptions "Pin Change" (possible sur toute entrée TOR)

Les interruptions *Pin Change* (PCINT0_vect, PCINT1_vect, PCINT2_vect) permettent d'associer un traitement (ISR) à une cause qui est le changement d'état d'une entrée. Chaque front, montant ou descendant, conduit à une interruption. Ce mécanisme peut être exploité pour toute entrée binaire. En revanche, la même ISR est utilisée pour un ensemble de broches. Il est donc parfois difficile de déceler quelle entrée précise est à l'origine de l'interruption, ceci en comparaison de INT0/INT1 où l'on sait précisément quelle est la cause de l'interruption.

Broches PCINT0 à PCINT23 (nom alternatif pour PBx, PCx et PDx) : configurables pour déclencher des interruptions (n° 4, 5 et 6 ou `PCINT0_vect`, `PCINT1_vect`, `PCINT2_vect`) suite à des changements d'état ("Pin Change") des broches (configurées en entrée DDRx.n=1). Les broches sont séparées en 3 sous-groupes, il y a une source d'interruption par sous-groupe, et pour chaque broche on peut activer ou non le système "Pin Change Interrupt"

Pin Change 3 groupes de broches (liés aux ports B, C et D)

PCINT0 – PCINT7 <-> broches PB0 à PB7 groupe lié à IT `PCINT0_vect`
PCINT8 – PCINT15 <-> broches PC0 à PC7 groupe lié à IT `PCINT1_vect`
PCINT16 – PCINT23<-> broches PD0 à PD7 groupe lié à IT `PCINT2_vect`

Les registres **`PCMSK0`**, **`PCMSK1`** et **`PCMSK2`** contrôlent, pour chacun de ces groupes (donc pour chaque port B, C D), quelle(s) broche(s) peut(vent) conduire (ou non) à une interruption de type "pin change".

Activation des interruptions PCINT0 à PCINT23 si bit SREG.7=1 et mise à 1 de PCIEx

Registre PCICR : activation des IT Pin Change pour un groupe

PCICR.0 : activation des IT Pin Change pour les broches du port B (PB0 à PB7)
PCICR.1 : activation des IT Pin Change pour les broches du port C (PC0 à PC6)
PCICR.2 : activation des IT Pin Change pour les broches du port D (PD0 à PD7)

PCICR – Pin Change Interrupt Control Register

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

Activation à l'intérieur d'un groupe : le registre **PCMSKx** détermine quelles broches du groupe sont prises en compte pour l'interruption "pin change"

PCMSK0 – Pin Change Mask Register 0

Bit	7	6	5	4	3	2	1	0	
(0x6B)	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0	PCMSK0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PCMSK1 – Pin Change Mask Register 1

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

PCMSK2 – Pin Change Mask Register 2

Bit	7	6	5	4	3	2	1	0	
(0x6D)	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16	PCMSK2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Exemple : mise en oeuvre IT Pin Change sur Port B et Port D

```
// Config pour IT Pin Change
void setup()
{
  cli();            // aucune IT possible (SREG.7<-0)
  PCICR |= 0x05;    // Pin Change actif sur port D et port B
  DDRB&=~0x03;      // PB1 et PB0 en entrée
  DDRD&=~0x0A;      // PD7 et PD5 en entrée
  PCMSK0=0x03;      // Pin Change pour broches PB0 et PB1
  PCMSK2=0xA0;      // Pin Change pour broches PD7 et PD5
  sei();            // IT possibles SREG.7<-1
}

ISR(PCINT0_vect){   ...   }  // appelée si changement sur PB1 ou PB0

ISR(PCINT2_vect){   ...   }  // appelée si changement sur PD7 ou PD5
```

Flags internes pour les IT "Pin Change"

PCIFR – Pin Change Interrupt Flag Register

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

Autre exemple "PIN CHANGE"

```
//IT Pin Change sur PC4,PC5,PD2 et PD3

void setup()
{
  Serial.begin(9600);
  cli();            // aucune IT possible (SREG.7<-0)
  PCICR |= 0x06;    // Pin Change actif sur port D et port C
  PCMSK2=0x0C;      // Pin Change pour broches PD3 et PD2
  PCMSK1=0x30;      // Pin Change pour broches PC5 et PC4
  DDRD&=~0x0C;      // PD2 et PD3 en entrée
  PORTD|=0x0C;      // pull-up sur PD2 PD3
  DDRC&=~0x30;      // PC4 et PC5 en entrée
  PORTC|=0x30;      // pull-up sur PC4 et PC5
  sei();            // IT possibles SREG.7<-1
}

volatile int cpt1=0;
volatile int cpt2=0;

ISR(PCINT1_vect)       // IT changement sur PC4 ou PC5
{
    cpt1++;
}

ISR(PCINT2_vect)       // IT changement sur PD2 ou PD3
{
    cpt2++;
}

void loop()
{
    delay(2000);
    Serial.print("cpt1=");
    Serial.println(cpt1,DEC);
    Serial.print("cpt2=");
    Serial.println(cpt2,DEC);
}
```

Remarque : sur AVR, une fonction d'interruption ne peut pas elle-même être interrompue.

6.3 Interruptions Timers

Les timers intégrés peuvent déclencher des interruptions. On propose une section complète sur la configuration des timers intégrés et l'exploitation des interruptions associées.

7. Timers/Counters de ATMega328

Le microcontrôleur ATMega328 dispose de plusieurs modules de temporisation/comptage internes (Timers), fonctionnant pour certains avec des registres de comptage sur 8 bits, et pour d'autres sur 16 bits. Dans tous les cas, chaque événement de comptage conduit à une modification du registre de comptage (+1). L'événement de comptage peut être un "tick" de l'horloge du microcontrôleur, ce qui revient à mesurer l'écoulement du temps. L'événement de comptage peut aussi être un front sur une broche d'entrée du microcontrôleur (les broches T0 et T1 peuvent servir d'entrée de comptage).

Fonction Temporisateur : lorsque l'on compte des "ticks" de l'horloge qui cadence le microcontrôleur, on mesure du temps. Les modules Timers/Counters permettent de compter les ticks du signal d'horologe, ou un signal de fréquence plus faible obtenu par un diviseur appelé **prescaler**. C'est la fonction temporisateur qui va être détaillée par la suite.

Note : sur la carte Arduino UNO, l'horloge est à **16MHz**, soit 16 000 000 de cycles horloge par seconde, ou 16 cycles horloge par micro seconde. Ce sont ces cycles là qui sont comptés en fonction temporisateur. Il faut compter 16000000 cycles pour faire une seconde.

Fonction Compteur : lorsque l'on compte des fronts sur une entrée de comptage (broches T0 ou T1), on utilise alors la fonction "compteur" du module (non étudié ici).

Le choix entre fonction de temporisation (avec prédiviseur de fréquence ou non) et fonction de comptage se fait par paramétrage de registres dédiés à la gestion des modules Timers/Counters.

Génération de signaux périodiques : les modules Timers/Counters sont assez complexes et chacun de ces modules peut générer deux signaux PWM (Pulse Width Modulation) dont le rapport cyclique est facilement modifiable. Dans ce cas, utiliser la fonction Arduino analogWrite() qui génère un signal PWM. Ce signal PWM n'est géré que sur les sorties liées à des Timers intégrés c'est-à-dire PD6,PD5,PD3, PB1,PB2 et PB3.

Arduino analogWrite( ) = génère un signal PWM

Remarque : les timers sont des périphériques intégrés assez complexes (environ 70 pages du datasheet ATMega). Seule une vision simplifiée est fournie ici.

7.1 Timer/Counter 0 (comptage 8 bits)

C'est un module Timer/Counter avec registre de comptage 8 bits. En utilisant l'IDE Arduino, le timer 0 (et l'interruption associée) est implicitement utilisé par les fonctions de gestion du temps (delay(),millis() ...). Ce module Timer/Counter n'est donc pas utilisable directement avec la carte ARDUINO Uno. Sauf si l'on accepte de se passer des fonctions de gestion de temps Arduino.

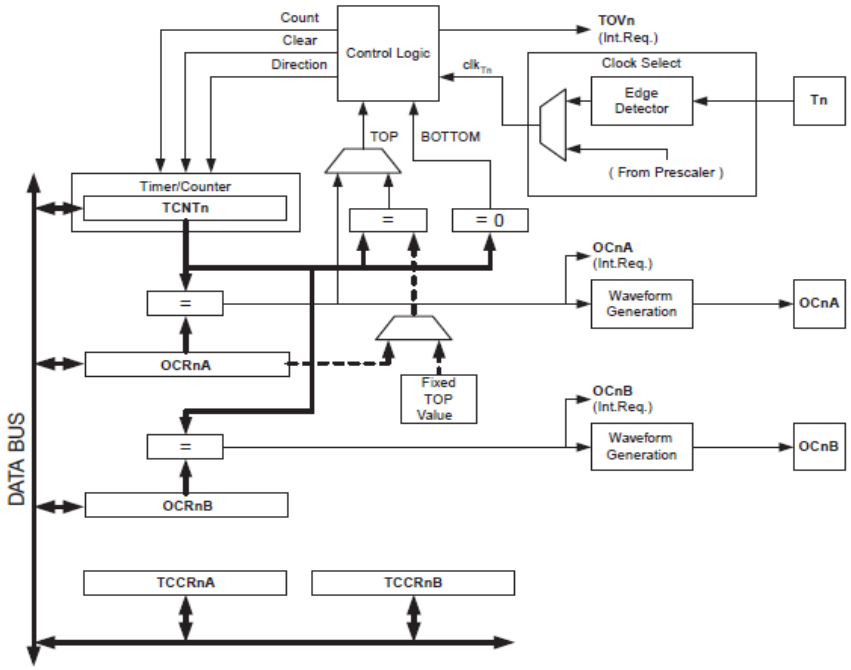
7.2 Timer/Counter 2 (comptage 8 bits)

C'est un module Timer/Counter avec registre de comptage 8 bits.

La structure générale du module Timer/Counter 0 est donnée dans le schéma ci-après. Le registre de comptage est TCNT2 (registre 8 bits).

Points importants (Timer 2) :

- détection et IT sur débordement (**`TIMER2_OVF_vect`**)
- entrée de comptage = signal d'horloge avec prédivision ou non
- possibilité de comparer TCNT2 à deux registres de comparaison OCR2A/OCR2B
- l'égalité TCTN2=OCR2A peut déclencher une IT (**`TIMER2_COMPA_vect`**)
- l'égalité TCTN2=OCR2B peut déclencher une IT (**`TIMER2_COMPB_vect`**)
- Les broches OC2A(PB3) et OC2B (PD3) peuvent être activées par le Timer/Counter 2 pour génération de signaux périodiques (PWM).



Registres du module Timer/Counter 2

TCNT2 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
(0xB2)	TCNT2[7:0]								TCNT2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

OCR2A – Output Compare Register A

Bit	7	6	5	4	3	2	1	0	
(0xB3)	OCR2A[7:0]								OCR2A
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	

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC2A pin.

OCR2B – Output Compare Register B

Bit	7	6	5	4	3	2	1	0	
(0xB4)	OCR2B[7:0]								OCR2B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	

TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R	R	R/W	R/W	

TIMSK2 – Timer/Counter2 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x70)	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	TIMSK2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	

TIFR2 – Timer/Counter2 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	–	–	–	–	–	OCF2B	OCF2A	TOV2	TIFR2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	

Modes de fonctionnement en mode temporisateur :

Normal : le registre TCNT2 est incrémenté de 1 à chaque événement de comptage. Le registre ne revient à 0 qu'après un débordement (passage de 0xFF à 0x00).

CTC (Clear Timer on Compare) : le registre TCNT2 s'incrémente à chaque événement de comptage mais est remis à 0 si TCNT2=OCR2A.

D'autres modes non détaillés, notamment pour la gestion des PWM.

Le choix du mode se fait via les bits WGM22:20 (bits TCR2A et TCR2B)

Table 17-8. Waveform Generation Mode Bit Description

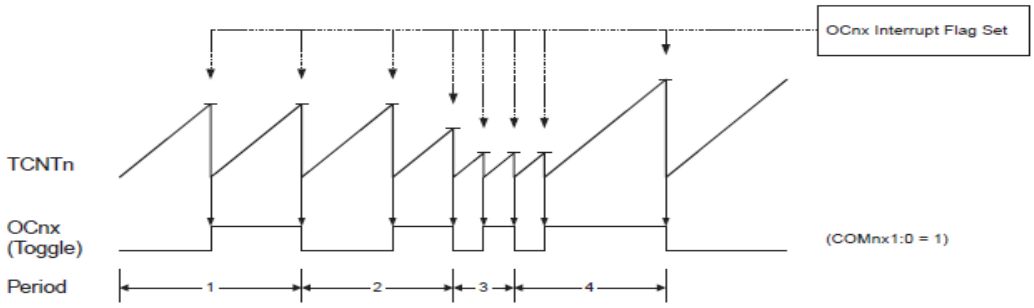
Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes: 1. MAX= 0xFF
2. BOTTOM= 0x00

Clear Timer on Compare Match (CTC) Mode

En mode CTC (WGM22:0 = 2), le registre OCR2A règle la résolution. Le compteur TCTN2 est remis à zéro après l'égalité (match) TCTN2=OCR2A. Le registre OCR2A definit la valeur maximale pour le compteur, et donc sa résolution.

Figure 17-5. CTC Mode, Timing Diagram



Prescaler : en fonction temporisateur, le registre de comptage TCNT2 est incrémenté en fonction des cycles horloge. L'incrément peut être à chaque cycle horloge (pas de prescaler) ou bien à une fréquence moindre. Rappelons que le cycle horologe est de 1/16 micro-secondes. On peut aussi mettre un diviseur de fréquence entre l'horloge (16Mhz) et l'incrément du registre de comptage. Les prescalers possibles pour le Timer2 sont ci-après.

Prescaler Timer 2 (clk = 16Mhz)

Table 17-9. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

7.3 Exemples Timer 2 avec Interruption

L'exemple suivant illustre l'utilisation du timer 2 et de l'interruption associée à son débordement.

Activation des interruptions Timer 0 (3 sources n°14, 15 et 16)

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter Interrupt Flag Register – TIFR0.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

Remarque : pour les Timers 1 et 2, les configurations sont similaires.

Exemple : on souhaite que chaque débordement du Timer 0 conduise à une interruption.

> **SREG.7=1** (bit général d'activation des IT, sans lui aucune IT)
> **TIMSK0.0 (TOIE0)=1** (interruption sur débordement du timer 0)

Pour le Timer 2, à chaque débordement de TCNT2 (passage de 0xFF à 0x00) une interruption TIMER2_OVF_vect peut être exploitée. Il faut donc activer ce mécanisme

```
TIMSK2=0x01;     // IT Timer2 Over Flow Active
```

Fournir la routine d'IT :

```
ISR(TIMER2_OVF_vect){ ... }     //ISR de l'IT débordement Timer2
```

et configurer le Timer 2.

```
// ARDUINO UNO - IT Timer 2 Overflow

volatile unsigned char cpt=0; // compteur d'IT

// Fonction de traitement IT n°10 = Timer 2 OverFlow
ISR(TIMER2_OVF_vect){
      cpt++;
      if(cpt==61){
            PORTB ^=0x20;
            cpt=0;
      }
}

void setup(){
      //Configuration PORTB.5
      DDRB |= 0x20;    // PB5 en sortie
      PORTB &= ~0x20; // PORTB.5 <-0
      cli();
      // Configuration Timer 2
      TCCR2A=0;        // Mode Normal (pas PWM)
      TCCR2B=0x07;     // Prescaler 1024 (Clock/1024)
      TIMSK2=0x01;     // IT Timer2 Over Flow Active
      sei();      // activation des IT (SREG.7=1)
}

void loop() {    /* aucun traitement*/       }
```

Paramètres Timer 2

Mode 0 (Normal) : WGM2=0 WGM1=0 WGM0=0 [TCCR2A=0]
Prescaler = 1024 : CS22=1 CS21=1 CS20=1 [TCCR2B=0x07=(111)b]

Interruptions

Masque d'IT : Interruption sur Overflow = TIMSK2.0 =1
Autorisation générale des IT : SREG.7=1 [activé par sei()]

Digital I/O = Broche PORTB.5 en sortie.

Principe : après la fonction setup(), le registre **TCNT2** (8bits) est incrémenté à chaque tick du signal périodique clock/1024. A chaque débordement du registre TCNT2, le débordement déclenche l'interruption n°10 appelée "Timer 2 Over Flow". Tous les 60 appels de cette fonction, la broche PB5 (LED) change d'état. La LED clignote donc.

Quelle fréquence de clignotement ?

Clock = signal périodique de 16MegaHz (16000000 de cycles par seconde)
Prescaler = 1024 -> la fréquence d'incrément de TCNT2 = (16/1024) MegaHz
Fréquence de débordement : TCNT2 ne déborde que tous les 256 incréments
c'est-à-dire à la fréquence de 16/(1024*256) MegaHZ ≈ 61 Hz
Il y a 1 interruption Timer2 Over Flow tous les 1/61 secondes.
Il faut 61 Interruptions pour que la LED change d'état

La LED change d'état (0->1 1->0) à intervalles de environ 1 seconde

Autre exemple : timer2 en mode CTC et interruptions

On utilise ici le mode CTC du Timer 2. A chaque fois que TCNT2=OCR2A, TCNT2 est remis à 0 et l'égalité déclenche une interruption.

```
// ARDUINO UNO - IT Timer 2
// Mode Clear Timer On Compare

volatile unsigned char cpt;

// Fonction de traitement IT n°10 = Timer 2 OverFlow
ISR(TIMER2_COMPA_vect){
      cpt++;
        if(cpt==40) PORTB|=0x20;
        if(cpt==50){
            PORTB &=~0x20;
            cpt=0;
         }
}

void setup(){
      //Configuration PB5
      DDRB |= 0x20;     // PB5 en sortie
      PORTB &= ~0x20;   // PORTB.5 <-0

      // Configuration Timer 2
      TCCR2A=0x02;     // Mode CTC (Clear Timer On Compare)
      OCR2A=156;       // Registre de comparaison A = 156
      TCCR2B=0x07;     // Prescaler 1024 (Clock/1024)
      TIMSK2=0x02;     // IT Timer2 Quand TCNT2=OCR2A
      sei();     // activation des IT (SREG.7=1)
}

void loop() {      /*aucun traitement*/  }
```

Quelle fréquence de clignotement ?

Clock = signal périodique de 16MegaHz (16 millions de ticks par seconde)
Prescaler = 1024 -> la fréquence d'incrément de TCNT2 = (16/1024) MegaHz
Fréquence de débordement :
TCNT2 ne déborde que tous les 156 incréments (valeur de OCR2A)
c'est-à-dire à la fréquence de 16/(1024*156) MegaHZ ≈ 100 Hz
Il y a 1 interruption Timer2 On Compare A environ tous les 1/100 secondes.

La LED s'allume 1/10 de seconde puis s'éteint 4/10 de seconde. Elle s'allume brièvement 2 fois par seconde.

Fonction de configuration du timer2 en mode CTC

La fonction SetTimer2CTC(CSB,periode) founie ci-dessous permet de configurer le Timer 2 en mode CTC avec une periode donnée (sur 8 bits) entre deux remises à 0.

Exemple :

SetTimer2CTC(2,50); // prescaler = /8 periode=50

Dans l'exemple ci-dessous :

prescaler /256 = TCNT2 incrémenté 62500 fois par seconde
periode 100 = l'interruption TIMER2_COMPA_vect a lieu 625 fois par seconde

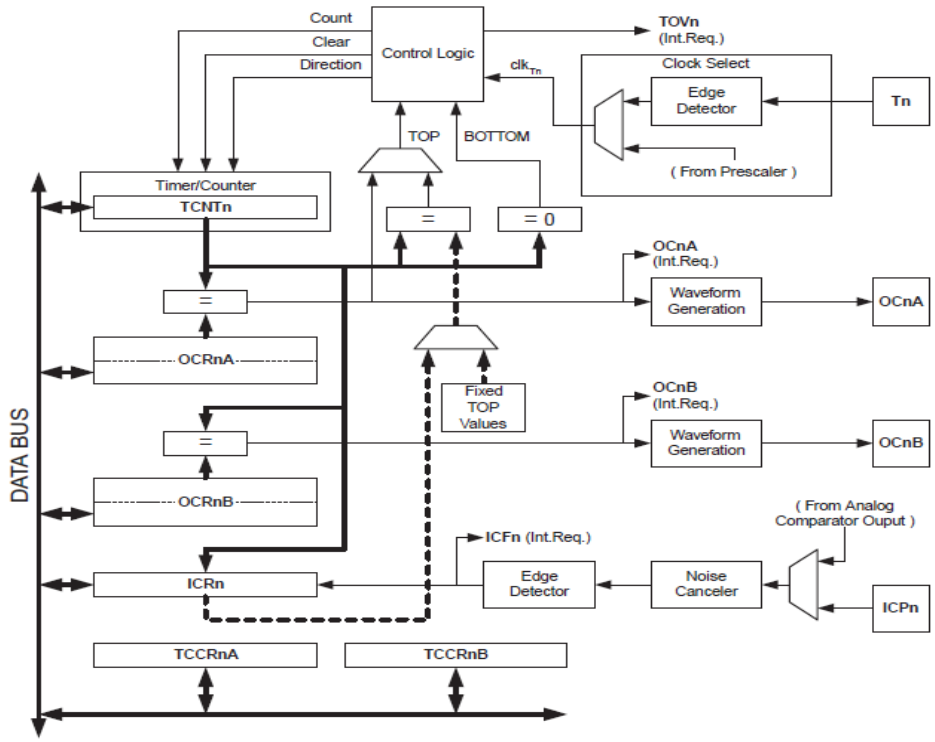
```
/* Config. Timer2 en Mode CTC
CSB(Clock Select)= 0(timer2 stop),1(=/1),2(=/8),3(=/32),
4(=/64),5(=/128),6 (=/256),7(=/1024)
periode = nbre incr. TCNT2 (8bits) entre 2 remises à 0
*/
void SetTimer2CTC(byte CSB,byte periode)
{
  TCCR2A=B010;    // Mode CTC (Clear Timer On Compare)
  OCR2A=periode; // Registre de comparaison OCR2A (8bits)
  TCCR2B&=0xF0;
  TCCR2B|=(CSB&0x07); // Choix prescaler
}

unsigned int cpt=0;

ISR(TIMER2_COMPA_vect)  // IT quand TCNT2==OCR2A
{
  cpt++;
}

void setup()
{
    Serial.begin(9600);
    cli();
    SetTimer2CTC(6,100);  // prescaler /256 periode 100
    TIMSK2|=0x02; //IT Timer2 quand TCNT2==OCR2A
    sei();
}

void loop()
{
   delay(1000);
   Serial.println(cpt,DEC);
}
```

7.4 Timer/Counter 1 (comptage 16 bits)

Le registre de comptage TCNT1, ainsi que les registres de comparaison OCR1A et OCR1B, sont cette fois-ci sur 16 bits.

Note: en langage d'assemblage, il faut deux accès 8 bits pour lire/écrire ces registres 16 bits. En langage C, on peut manipuler symboliquement des données 16 bits via les symboles TCNT1, OCR1A et OCR1B sans se soucier de la façon dont le code sera généré.

Figure 15-1. 16-bit Timer/Counter Block Diagram[1]



Registres du module Timer/Counter 1

TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

OCR1BH and OCR1BL – Output Compare Register 1 B

Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B[15:8]								OCR1BH
(0x8A)	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

Selon le mode choisi par les bits WGM10:3 on a les options suivantes (mode PWM Phase correct non décrit)

Table 15-4. Waveform Generation Mode Bit Description(1)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

Prescaler Timer 1 (réglage de la vitesse de débordement en fonction de l'horloge)

Table 15-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Fonction de configuration du timer 1 en mode CTC

Dans l'exemple ci-dessous

prescaler /256 = 62500 incréments de TCNT1 par secondes
periode 10 000 = 6,25 interruptions TIMER1_COMPA_vect par seconde

```
/* Timer1 en Mode CTC
CSB = 0 (timer1 stop), 1(/1),2(/8),3(/64),4(/256),5(/1024)
periode = nb incr. de TCNT1(16 bits) entre 2 remises à 0*/
void SetTimer1CTC(byte CSB,unsigned int periode)
{
      TCCR1A=0;
      OCR1A=periode; // Registre de comparaison OCR1A (16bits)
      TCCR1B=0x08;
      TCCR1B|=(CSB&0x07);
}

unsigned int cpt=0;

ISR(TIMER1_COMPA_vect)  // IT quand TCNT2==OCR2A
{
  cpt++;
}

void setup()
{
    Serial.begin(9600);
    cli();
    SetTimer1CTC(4,10000);  // prescaler /256 periode 10000
    TIMSK1|=0x02; //IT Timer1 quand TCNT1==OCR1A
    sei();
}

void loop()
{
   delay(1000);
   Serial.println(cpt,DEC);
}
```