Informatique Embarquée

Informatique Embarquée

Volume horaire : 40,5h TP

Objectifs du module (P.P.N):

- Comprendre l'architecture d'un système à microcontrôleur.
- Maîtriser l'utilisation des périphériques d'un microcontrôleur.
- Savoir modéliser une application embarquée.
- Comprendre les mécanismes d'interruption.

Informatique Embarquée

Compétences visées (P.P.N):

- Développer une application en langage évolué pour une cible à microcontrôleur,
- Gérer les périphériques d'entrées / sorties pour s'interfacer avec un environnement,
- Mettre en œuvre le mécanisme de fonctionnement en régime d'interruption de programme,
- Utiliser un outil de développement croisé *(PC de développement et cible)*

Pré-requis (P.P.N): Modules M 1103 (Info1 : Informatique) et M 1102 (SIN1 : Système d'information numérique).

Informatique Embarquée

Mots clés (P.P.N):

- Microcontrôleur
- Périphériques
- Architecture matérielle
- Registres
- Interruptions
- Développement croisé

Microcontrôleur

Microcontrôleur

Qu'est ce qu'un microcontrôleur ?

Un **microcontrôleur** (**µC** en notation abrégée), ou **MCU** (**M**icro **C**ontroler **U**nit) est un circuit intégré qui rassemble dans un même boîtier, tous les éléments constitutifs d'un calculateur :

- une (UC) **Unité Centrale** ou *CPU (**C**entral **P**rocessing **U**nit)*),
- de la **mémoire** (*RAM, EEROM, Flash, ...*)
- des unités **périphériques** et des interfaces d'entrées-sorties.

Microcontrôleur

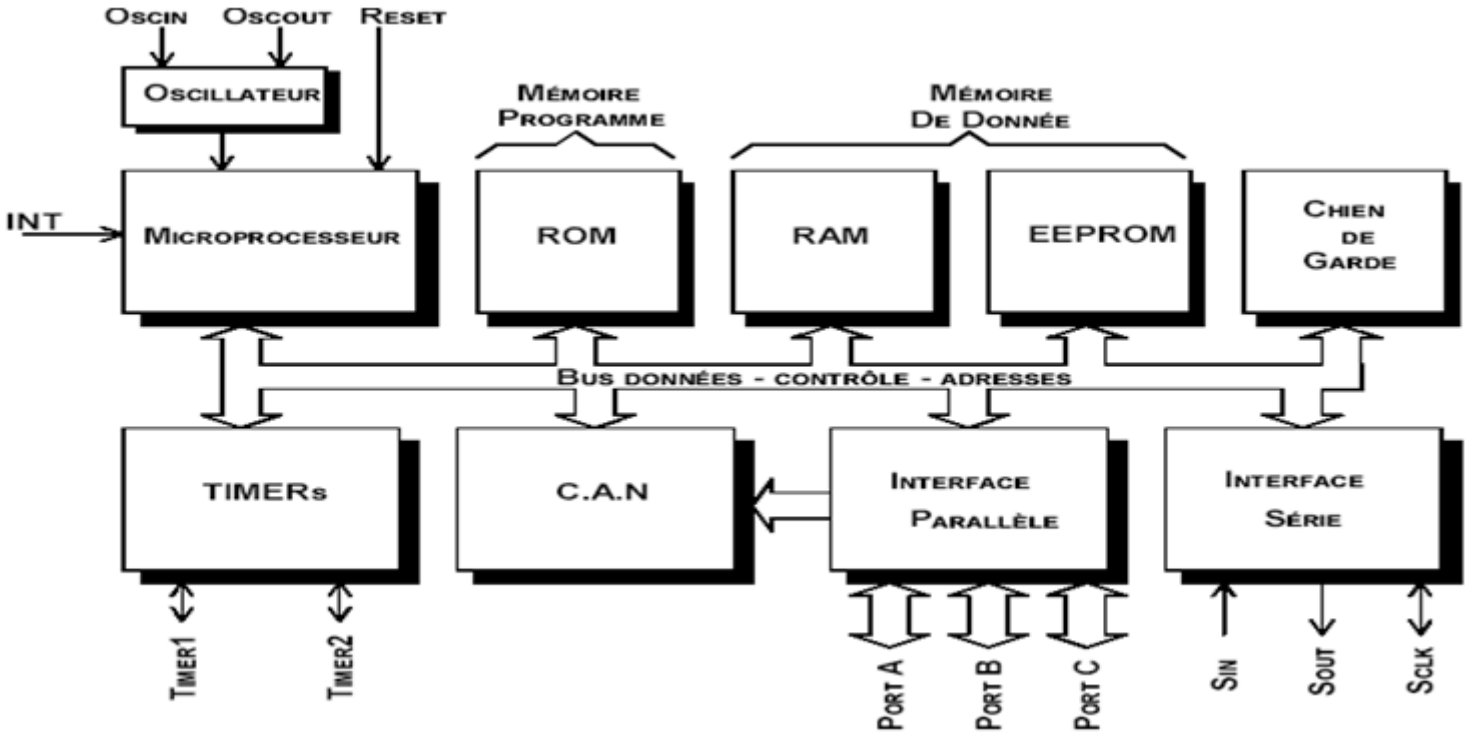
Qu'est ce qu'un microcontrôleur ?

Les microcontrôleurs se caractérisent par
- un haut degré d'intégration,
- une faible consommation électrique,
- une vitesse de fonctionnement relativement faible (quelques mégahertz)
- un coût réduit par rapport aux microprocesseurs utilisés dans les ordinateurs.

Ils sont utilisés dans des systèmes embarqués que l'on trouve dans différents domaines, comme
- les transports : l'avionique, le ferroviaire, l'automobile ...
- les télécommunications,
- la robotique,
- l'électroménager, ...etc...

Microcontrôleur

Structure interne d'un µC :



Microcontrôleur

L'unité centrale (UC) ou microprocesseur :

Comme son nom l'indique l'Unité Centrale (**UC**) est l'élément central d'un système informatique qui exécute les instructions d'un programme pour traiter des données.

Elle est constituée de différents éléments :

- une unité arithmétique et logique (**ALU** : Arithmetic and Logic Unit)
- un registre d'instruction
- un décodeur d'instruction
- un registre d'état
- des registres de stockage
- un séquenceur ...

Microcontrôleur

L'unité centrale (UC) ou microprocesseur :

Elle a besoin d'éléments externes pour fonctionner :

- **une horloge** (*en général à quartz*) pour la cadencer
- **de la mémoire** pour stocker :
  - les données manipulées par le programme (*mémoire vive volatile: RAM*)
  - les instructions d'un programme (*mémoire morte non volatile: ROM)*
- **de périphériques** pour interagir avec le monde extérieur (*interface série et parallèle, Convertisseur Analogique Numérique, Timers, ...etc...*)

Microcontrôleur

L'unité centrale (UC) ou microprocesseur :

UC, mémoires et périphériques sont reliés entre eux par trois bus :

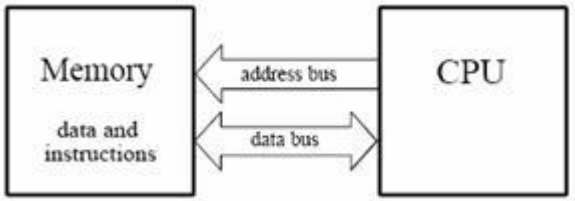
- un **bus d'adresse,** qui permet à l'UC de sélectionner la mémoire ou le périphérique auquel elle veut accéder
- un **bus de données,** qui permet le transfert des informations entre les différents éléments
- un **bus de contrôle,** pour contrôler le sens des transactions sur le bus de données

Microcontrôleur

L'architecture de Von Neuman et d'Harvard :

Il existe deux types fondamentaux de structure interne d'un µC :

- **L'architecture de « Von Neuman » :**



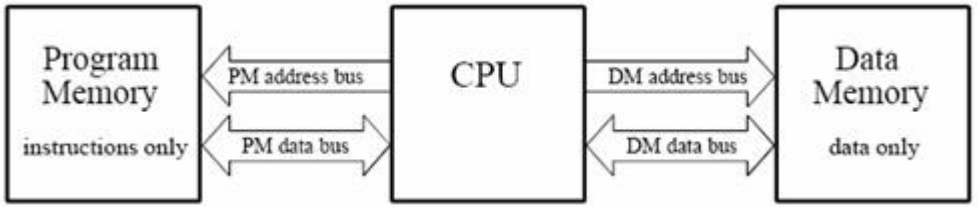
Les instructions du **programme** et les **données** qu'il manipule, sont stockées **dans la même zone mémoire**, accessible avec **un seul bus d'adresse et un seul bus de données.**
Les informations qui circulent sur le bus de données (data bus) peuvent être suivant le séquencement des opérations, soit des instructions, soit des données.

Microcontrôleur

L'architecture de « Von Neuman » et de « Harvard » :

Il existe deux types fondamentaux de structure interne d'un µC :

- **L'architecture de « Harvard » :**



Cette structure se distingue de l'architecture Von Neuman uniquement par le fait que les **mémoires programme et données sont accessibles via des bus séparés**.
Cette organisation permet de transférer simultanément vers le CPU une instruction et une donnée *(pipeline)*.
Le CPU peut donc pendant qu'il exécute une instruction, aller chercher l'instruction suivante, ce qui améliore les performances du système.

Microcontrôleur

Les mémoires:

- Les mémoires sont des composants essentiels qui servent à stocker des informations.

- Chaque information est un mot binaire stocké dans une case, dont le format correspond souvent à la taille du bus de données du µC (*8 bits, 16 bits, ...*).

- Chaque case mémoire a une adresse précise qui permet de retrouver l'information stockée.

- Le nombre de cases adressables, correspondant à la capacité de stockage de la mémoire, dépend de la taille du bus d'adresse (2^n *cases pour un bus d'adresse de n bits*).

Microcontrôleur

Les mémoires: on distingue deux types de mémoire, les ROMs et les RAMs:

Les mémoires regroupées sous le sigle ROM:

- Ce sont des mémoires **non volatiles**, dont le contenu est conservé, même après leur mise hors tension.
- Elles servent au **stockage des programmes** et aux données qui ne doivent pas être perdues lorsque le microcontrôleur n'est plus alimenté.
- Elles portent également le nom de **mémoires mortes**, car une fois programmées leur contenu n'évolue plus (ou très peu).
- Leur accès ne se fait donc qu'en lecture, d'où leur nom: **Read Only Memory** (*ROM ou Mémoire à lecture seule*).

Microcontrôleur

Les mémoires: on distingue deux types de mémoire, les ROMs et les RAMs:

Les mémoires regroupées sous le sigle RAM:

- Ce sont des mémoires **volatiles,** dont le contenu est perdu à chaque mise hors tension du µC.
- Elles servent au stockage des données temporaires manipulées par les programmes.
- Elles portent également le nom de **mémoires vives**, car leur contenu évolue en permanence lors de l'exécution d'un programme.
- Leur accès peut se faire en lecture ou en écriture, suivant que l'on veut lire ou stocker une donnée.
- Deux accès consécutifs pouvant se faire vers des cases dont les adresses sont aléatoires, il leur a été donné le nom de : **Random Access Memory** (*RAM ou Mémoire à accès aléatoire*)

Microcontrôleur

<u>Les mémoires ROM</u> (Read Only Mémory)

Il existe plusieurs types de mémoires ROM :

- les **ROMs** (Read Only Memory) : mémoires anciennes qui étaient programmées à la fabrication.
- les **PROMs** (Programmable ROM) : mémoires anciennes programmables une fois à l'aide d'un graveur.
- les **EPROMs** (Erasable PROM) : possibilité de les effacer en les exposant à un rayonnement ulta-violet (*10 à 20 minutes*) et de les reprogrammer à l'aide d'un graveur.
- les **EEPROMs** (Electrically EPROM) : possibilité de les effacer électriquement in-situ sans les retirer de leur support.
- les **Flash** ce sont des EEPROMs avec des opérations d'effacement / écriture rapides.

Microcontrôleur

<u>Les mémoires RAM</u> (Random Access Mémory)

Il existe deux types de mémoires RAM :

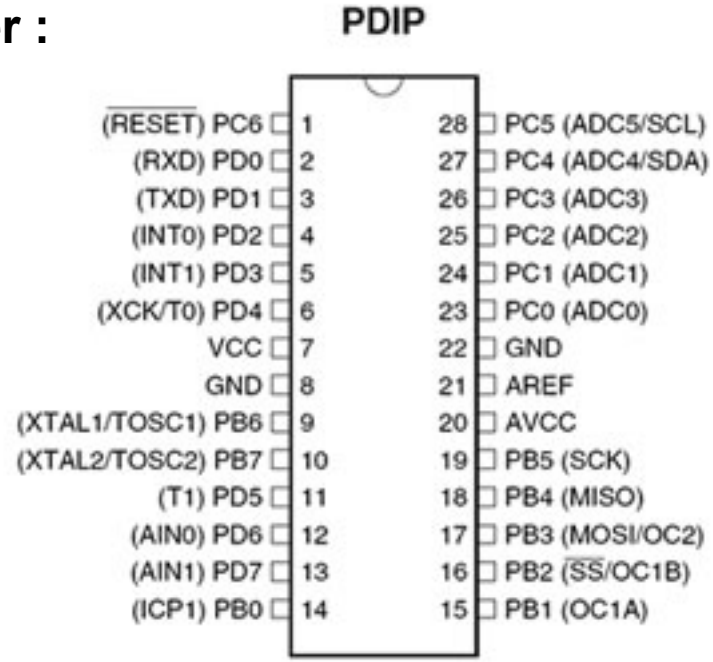
- les **SRAMs** (Static RAM) elles sont rapides et le point mémoire est une bascule.
  C'est le type de mémoire utilisé dans un µC pour la sauvegarde des données manipulées par les programmes .

- les **DRAMs** (Dynamic RAM) elles sont rapides et le point mémoire est un condensateur de faible valeur.
  <u>Avantage</u> : leur capacité de stockage est très élevée (*un condensateur prenant moins de place qu'une bascule*)
  <u>Inconvénients</u> : les condensateurs se déchargeant très rapidement (*quelques ms*) on doit les recharger en permanent à l'aide d'un dispositif de rafraîchissement.

Microcontrôleur

Le microcontrôleur Atmega328P de la société ATMEL :
C'est un microcontrôleur 8bits de la famille AVR utilisé sur les cartes Arduino UNO

Boitier :



Tension d'alimentation :
Vcc : de 1.8v à 5.5v

Vitesse :
0 – 4MHz @ 1.8 - 5.5V,
0 – 10MHz @ 2.7 - 5.5V,
0 – 20MHz @ 4.5 - 5.5V.

Microcontrôleur

Le microcontrôleur Atmega328P de la société ATMEL :

Ses 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 (TOR)) : 3 ports **PortB, PortC, PortD** (soit 23 broches)

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

Microcontrôleur

Le microcontrôleur Atmega328P de la société ATMEL :

Plusieurs broches multi-fonctions : certaines broches peuvent avoir plusieurs fonctions différentes selon la programmation.

PWM : 6 broches ***OC0A(PD6),OC0B(PD5),OC1A(PB1),OC1B(PB2),OC2A(PB3),OC2B(PD3)***

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

Bus I2C (TWI) : les données du bus transitent via les broches ***SDA(PC4) / SCL(PC5)***

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

Comparateur Analogique : broches ***AIN0(PD6) et AIN1(PD7)***

Microcontrôleur

Le microcontrôleur Atmega328P de la société ATMEL :

Gestion des interruptions (plusieurs sources possibles) :

- 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 (débordements)
- interruption liée au comparateur analogique
- interruption de fin de conversion ADC
- interruptions du port série USART
- interruption du bus I2C

Microcontrôleur

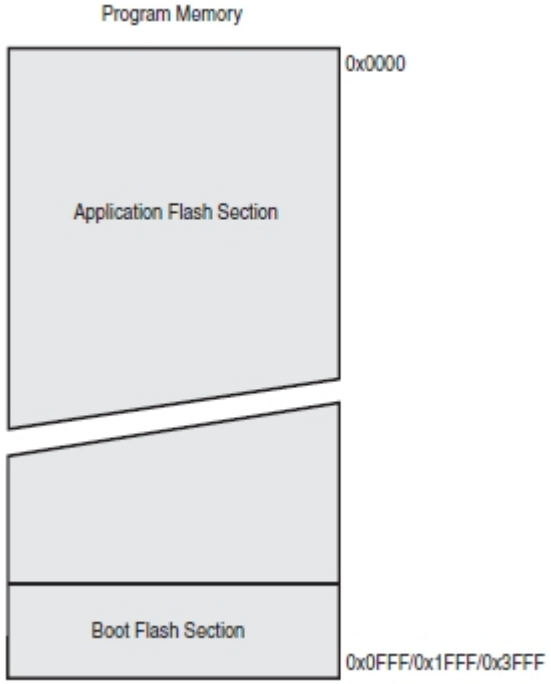
Plan mémoire Flash de l'Atmega328P :

Capacité de stockage des instructions :

$@_{fin} - @_{début} + 1 = 0x3FFF - 0x0000 + 1$
$= 0x4000$

$4 \times 16^3 = 2^2 \times (2^4)^3 = 2^2 \times 2^{12} = 2^{14}$
$= 2^4 \times 2^{10} =$ **16 Kinstructions**

Une instruction = 16 bits
Donc mémoire de **32 Koctets** *(=Kbytes)*



Microcontrôleur

Plan mémoire SRAM de l'Atmega328P :

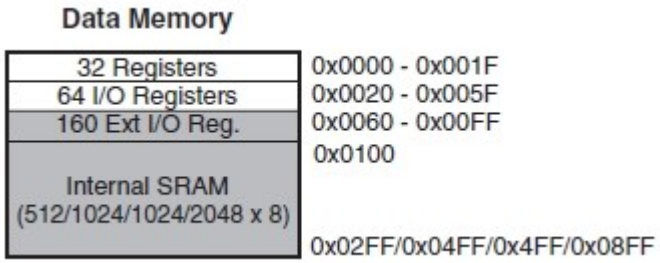
Capacité de stockage des données:

$@_{fin} - @_{début} + 1 = 0x8FF – 0x100 + 1$

$= 0x800$

$8 \times 16^2 = 2^3 \times (2^4)^2 = 2^3 \times 2^8 = 2^{11}$

$= 2 \times 2^{10} =$ **2 Koctets**



Microcontrôleur

Architecture AVR de l'Atmega328P :

AVR est le terme utilisé par Atmel pour désigner le cœur du processeur et la famille de microcontrôleurs qui le mettent en œuvre.

Sa fonction principale est d'assurer la bonne exécution des programmes.

Il doit donc être en mesure d'accéder aux mémoires, d'effectuer des calculs, de contrôler les périphériques et les interruptions.

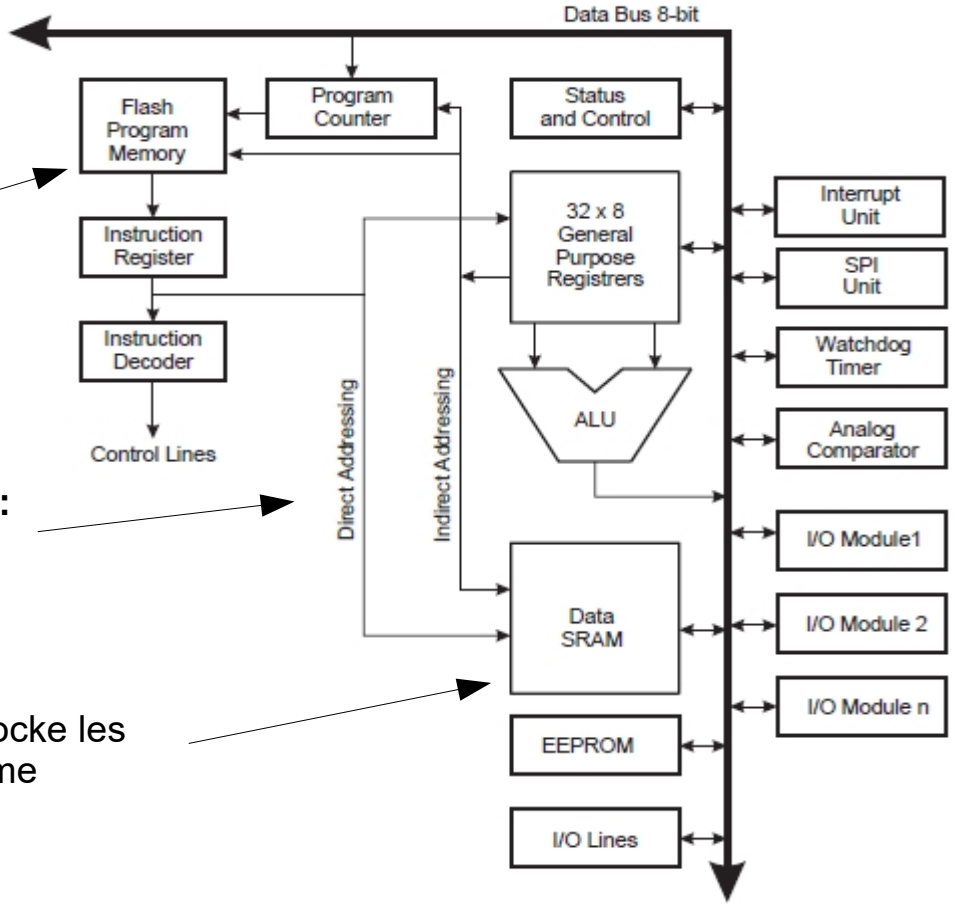
Microcontrôleur

Architecture AVR de l'Atmega328P :

Mémoire Flash : stocke les instructions du programme

ALU (Arithmetic and Logic Unit) : effectue les calculs

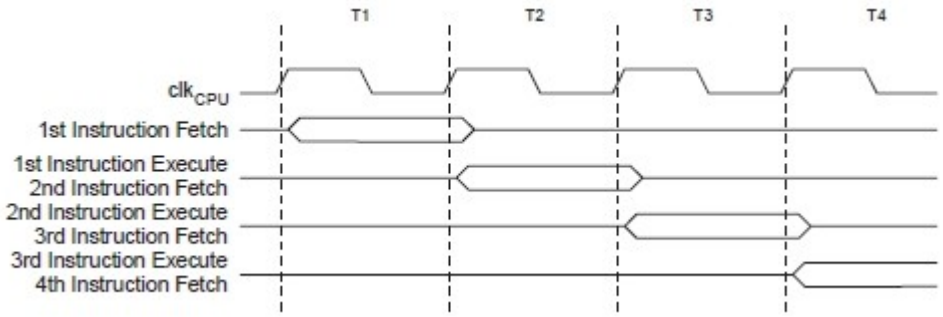
Mémoire SRAM : stocke les données du programme



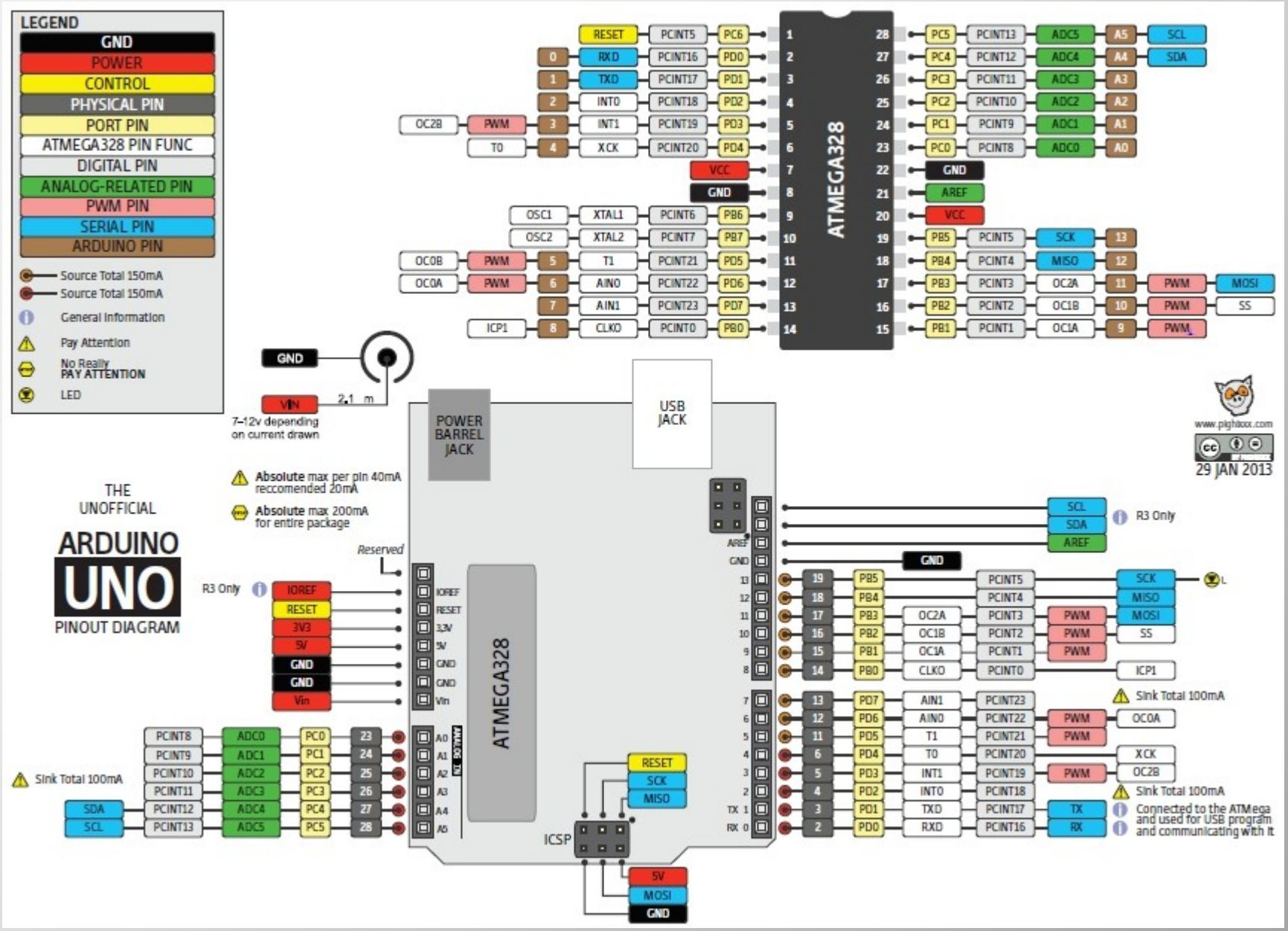
Microcontrôleur

Timing d'exécution d'une instruction *(pipeline)* :

Afin d'optimiser les performances, l'AVR utilise une architecture de Harvard avec des bus séparés pour le programme et les données.
Les instructions contenues dans la mémoire programme sont exécutées avec un seul niveau de pipeline, qui permet au processeur d'aller chercher une nouvelle instruction sans attendre que la précédente soit terminée.
Ce concept permet d'exécuter les instructions en un cycle d'horloge .



Les entrées/sorties


LEGEND
GND
POWER
CONTROL
PHYSICAL PIN
PORT PIN
ATMEGA328 PIN FUNC
DIGITAL PIN
ANALOG-RELATED PIN
PWM PIN
SERIAL PIN
ARDUINO PIN
Source Total 150mA
Source Total 150mA
General Information
Pay Attention
No Really PAY ATTENTION
LED
ATMEGA328
RESET PCINT5 PC6 1
RXD PCINT16 PD0 2
TXD PCINT17 PD1 3
INT0 PCINT18 PD2 4
OC2B PWM INT1 PCINT19 PD3 5
T0 XCK PCINT20 PD4 6
VCC 7
GND 8
OSC1 XTAL1 PCINT6 PB6 9
OSC2 XTAL2 PCINT7 PB7 10
OC0B PWM T1 PCINT21 PD5 11
OC0A PWM AIN0 PCINT22 PD6 12
AIN1 PCINT23 PD7 13
ICP1 CLKO PCINT0 PB0 14
28 PC5 PCINT13 ADC5 A5 SCL
27 PC4 PCINT12 ADC4 A4 SDA
26 PC3 PCINT11 ADC3 A3
25 PC2 PCINT10 ADC2 A2
24 PC1 PCINT9 ADC1 A1
23 PC0 PCINT8 ADC0 A0
22 GND
21 AREF
20 VCC
19 PB5 PCINT5 SCK 13
18 PB4 PCINT4 MISO 12
17 PB3 PCINT3 OC2A 11 PWM MOSI
16 PB2 PCINT2 OC1B 10 PWM SS
15 PB1 PCINT1 OC1A 9 PWM
GND
VIN
2.1 m
7-12v depending on current drawn
POWER BARREL JACK
USB JACK
www.pighixxx.com
29 JAN 2013
THE UNOFFICIAL
ARDUINO UNO
PINOUT DIAGRAM
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
Reserved
R3 Only
IOREF
RESET
3V3
5V
GND
GND
Vin
ATMEGA328
A0 A1 A2 A3 A4 A5
ANALOG IN
PCINT8 ADC0 PC0 23
PCINT9 ADC1 PC1 24
PCINT10 ADC2 PC2 25
PCINT11 ADC3 PC3 26
PCINT12 ADC4 PC4 27
PCINT13 ADC5 PC5 28
SDA
SCL
Sink Total 100mA
ICSP
RESET
SCK
MISO
5V
MOSI
GND
AREF
GND
SCL
SDA
AREF
R3 Only
GND
13 PB5 PCINT5 SCK
12 PB4 PCINT4 MISO
11 PB3 OC2A PCINT3 PWM MOSI
10 PB2 OC1B PCINT2 PWM SS
9 PB1 OC1A PCINT1 PWM
8 PB0 CLKO PCINT0 ICP1
7 PD7 AIN1 PCINT23
6 PD6 AIN0 PCINT22 PWM OC0A
5 PD5 T1 PCINT21 PWM
4 PD4 T0 PCINT20 XCK
3 PD3 INT1 PCINT19 PWM OC2B
2 PD2 INT0 PCINT18
TX 1 PD1 TXD PCINT17 TX
RX 0 PD0 RXD PCINT16 RX
Sink Total 100mA
Sink Total 100mA
Connected to the ATMega and used for USB program and communicating with it

Les entrées/sorties

Attention à bien faire la différence entre :

- le numéro de la broche du connecteur de la carte Arduino :

. en marron (ARDUINO PIN) sur le Pinout Diagram

. ce sont ces numéros de broche qu'il faut passer en argument aux fonctions des librairies Arduino : *pinMode(**broche**, mode), digitalRead(**broche**), digitalWrite(**broche**, valeur), ...*

- le numéro de la broche du microcontrôleur ATMEGA328 :

. en gris (PHYSICAL PIN) sur le Pinout Diagram

. ce numéro ne sera jamais utilisé dans les programmes

- le rang d'un bit sur les ports d' Entrée/Sortie :

. en jaune (PORT PIN) sur le Pinout Diagram

. utilisé quand on programme au niveau des registres internes, ex : **DDRB |= (1<<DDB4);**

. **exemple :** le bit 2 du PortB (PB2) correspond à la broche physique n°16 de l'ATMEGA328
et à la broche 10 de la carte ARDUINO
DDRB = DDRB | (1<<2) ; est équivalent à : pinMode(10,OUTPUT) ;

Le Pinout Diagram de la carte Arduino UNO vous permet d'établir la correspondance entre les différentes possibilités de repérer une même entrée/sortie.

Les entrées/sorties

Les ports d'entrée/sortie de l'Atmega328P :

Les entrées-sorties de l'ATmega328P sont réparties dans trois groupes de broches appelés **ports** qui permettent la communication avec l'extérieur :

- le **port B** regroupe les broches notées PBx,
- le **port C** les broches PCx et
- le **port D** les broches PDx.

Ces ports sont multidirectionnels et configurables broche à broche soit en entrée, soit en sortie.

Chaque port est configuré/exploité de 2 façons globalement :

- **fonctions de haut niveau** : pinMode, digitalWrite,digitalRead
- **les registres** : DDRx, PINx , PORTx (avec x=A,B,C ou D)

Les entrées/sorties

Fonctions d'E/S numérique :

pinMode (pin,mode**) :** configure une broche en entrée ou en sortie

- pin : numéro de la broche sur la carte Arduino
- mode : INPUT, OUTPUT ou INPUT_PULLUP

  Le mode INPUT_PULLUP, évite l'utilisation de résistances externes de pullup au profit des résistances internes disponibles sur chaque entrée.

digitalWrite (pin,value**) :** permet d'établir un niveau logique sur une broche préalablement configurée en sortie.

- pin : numéro de la broche sur la carte Arduino
- value : HIGH ou LOW

digitalRead (pin**) :** lit l'état logique d'une broche d'entrée référencée par son numéro.

Les entrées/sorties

Structure d'un programme :

La structure de base du langage de programmation Arduino est assez simple et comprend au moins deux parties *(ou fonctions)* : la fonction **« setup() »** et la fonction **« loop() »**

Ces deux fonctions contiennent des blocs d'instructions, et sont impérativement requises pour que le programme fonctionne.

```
void setup()
{  //blocs d'instructions;
}
```

La fonction **setup()** est la première fonction exécutée dans le programme :
- elle n'est **exécutée qu'une seule fois**.
- elle sert à configurer et initialiser les ressources utilisées.

```
void loop()
{  //blocs d'instructions;
}
```

La fonction **loop()** suit la fonction setup() et comprend le **code à exécuter en continu**.
C'est le **programme.**

Les entrées/sorties

Programmation sous formes de tâches :

PAS DE delay

```
#define led 5
#define intervalClignoteTask1 500


//global var
bool stateLed = true;
unsigned long currentTime;
unsigned long previousTimeClignoteTask1=0;


// the setup function runs once when you press reset or power the board

void setup() {
    pinMode(led, OUTPUT);
    digitalWrite(led, stateLed); // updating physical Led pin with logical state
}

// the loop function runs over and over again forever
void loop() {

    currentTime=millis();
   // TASK1:
    if (( currentTime-previousTimeClignoteTask1)> intervalClignoteTask1){
         previousTimeClignoteTask1=currentTime
         stateLed=!stateLed;
         digitalWrite(led, stateLed); // updating physical Led pin with logical state
    }
}
```

Les entrées/sorties

Lire des entrées : gestion sur niveau

```
#define bp 8
#define led3 9
#define intervalClignoteTask1 30

//global var
bool stateLed3 = true;
bool stateBP;
unsigned long currentTime = 0;
unsigned long previousTimeClignoteTask1 = 0;


// the setup function runs once when you press reset or power the board
void setup() {
    pinMode(led3, OUTPUT);
    pinMode(bp, INPUT);
}

// the loop function runs over and over again forever
void loop() {
    currentTime = millis();

    //TASK1: gestion sur niveau
    if ((currentTime - previousTimeClignoteTask1) > intervalClignoteTask1) {
        previousTimeClignoteTask1 = currentTime;
        stateBP = digitalRead(bp);
        if (stateBP == true){
                stateLed3 = 1;
        }
        else{
                stateLed3 = 0;
        }
        digitalWrite(led3, stateLed3); // updating physical Led pin with logi
    }
}
```

Les entrées/sorties

Lire des entrées : gestion sur front

```
void loop() {
    currentTime = millis();

    //TASK1: permute etat led à chaque appui
    if ((currentTime - previousTimeClignoteTask1) > intervalClignoteTask1) {
        previousTimeClignoteTask1=currentTime;
        stateBP=digitalRead(bp);

        if (stateBP == true)&&(previousStateBP == false)  { // BP active bas=front montant =relachement
            stateLed3=!stateLed3;
        }
        digitalWrite(led3, stateLed3);
        previousStateBP=stateBP;

    }
```

Les entrées/sorties

Les ports d'entrée/sortie de l'Atmega328P :

Les entrées-sorties de l'ATmega328P sont réparties dans trois groupes de broches appelés **ports** qui permettent la communication avec l'extérieur :

- le **port B** regroupe les broches notées PBx,
- le **port C** les broches PCx et
- le **port D** les broches PDx.

Ces ports sont multidirectionnels et configurables broche à broche soit en entrée, soit en sortie.

Chaque port est configuré/exploité grâce à trois registres :

- **DDRx** : détermine la direction de chaque broche du port (1:sortie, 0:entrée)
- **PORTx** : utilisé pour l'écriture de valeurs en sortie
- **PINx** : utilisé pour la lecture des valeurs en entrée

Les entrées/sorties

Les ports d'entrée/sortie de l'Atmega328P :

A fin d'éviter le flottement des valeurs en entrée d'un port, chaque broche de port E/S a une résistance de pull-up interne qui peut être activée ou pas.
Le bit **PUD** du registre **MCUCR** permet la désactivation des résistances de pull-up.

MCUCR – MCU Control Register

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

Notes: 1. BODS and BODSE only available for picoPower devices ATmega48PA/88PA/168PA/328P

- **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.

Les entrées/sorties

Le PORTB de l'Atmega328P :

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[(1)]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Les entrées/sorties

Synoptique d'un port d'entrée/sortie de l'Atmega328P :

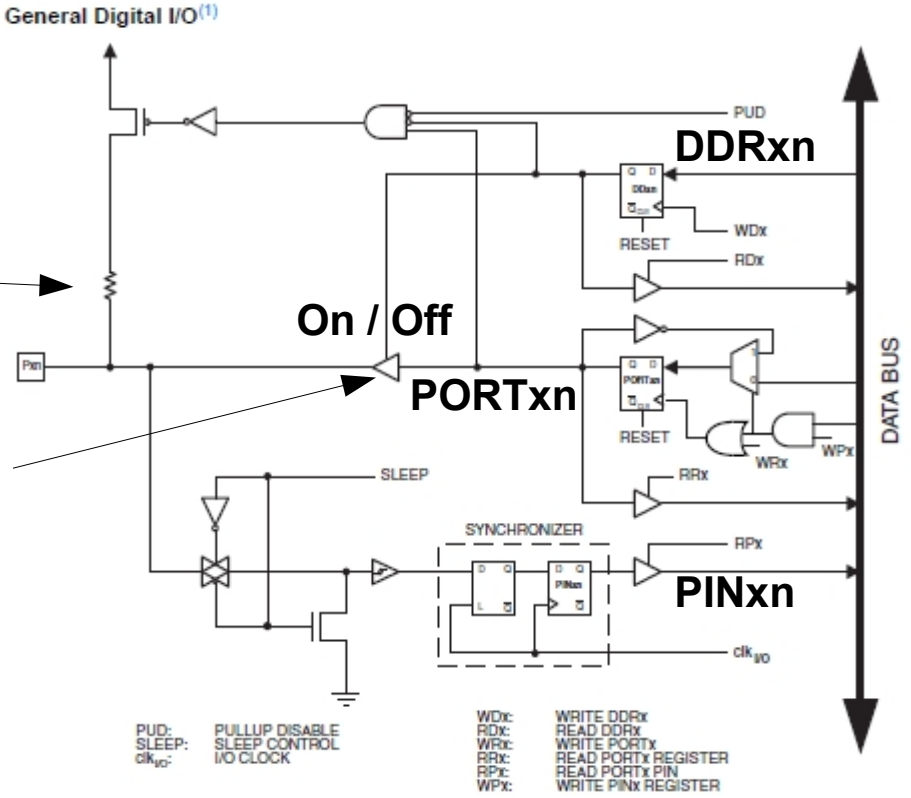
Le bit **PUD** du registre **MCUCR** permet de désactiver, ou pas, les résistances de pull-up **de tous les** ports.

Résistance de pull-up interne

L'état On/Off du buffer (*Tri-state*) dépend du bit **DDRxn :**

- si **DDRxn = 0** *(buffer Off)* broche configurée en **entrée.**
  L'état de l'entrée se lit dans le bit **PINxn**

- si **DDRxn = 1** *(buffer On)* broche configurée en **sortie**
  La sortie prend la valeur du bit **PORTxn**



Les entrées/sorties

Configuration et exploitation d'un port numérique *(TOR)* :

La configuration d'un port consiste à configurer le sens *(entrée ou sortie)* des broches utilisées, en positionnant les bits correspondants du registre DDRx.

L'état d'une entrée sera détecté par la lecture du bit correspondant à la broche, dans le registre PINx.

L'état d'une sortie sera établi par une écriture du bit correspondant à la broche, dans le registre PORTx.

La gestion par programme des bits d'un registre peut se faire :
- soit en utilisant des fonctions,
- soit en utilisant des instructions de manipulation de bits.

Les entrées/sorties

TRAVAIL PAR PORT Exemple1 : allumer plusieurs leds

```
void setup() {

     DDRD=0b10101000;
     PORTD=0b10100000;
}
```

Bénéfices des registres
1- méthode universelle
2- mise à jour simultanées de plusieurs sorties
3- code compact et rapide

TRAVAIL PAR PORT Exemple2 : travail par port et comptage

```
// the loop function runs over and over again forever
void loop() {
    currentTime = millis();

    //TASK1: clignote led 1
    if ((currentTime - previousTimeClignoteTask1) > intervalIncrementeTask1) {
        previousTimeClignoteTask1=currentTime;
        compteur++;
        PORTD=compteur;
    }

}
```

Les entrées/sorties

TRAVAIL PAR BIT :

Une opération « bit à bit » permet la manipulation des données directement au niveau des bits, selon les règles de l'algèbre de Boole. Elles sont utiles dès qu'il s'agit de manipuler les données à bas niveau.

Les opérations « bit à bit » courantes comprennent :

- des opérations logiques bit par bit ***(not, and, or, xor)*** et
- des opérations de décalage de bits ***(décalage à gauche ou à droite)***.

Les entrées/sorties

L'opérateur « bit à bit » NOT :

- il est représenté par le caractère tilde : **~**
- il s'applique à un seul opérande placé à sa droite
- il inverse chacun des bits de l'opérande *(0 devient 1 , et 1 devient 0)*

Exemple :

```
int a = 103;    // binary:  0000000001100111
int b = ~a;     // binary:  1111111110011000 = -104
```

Les entrées/sorties

L'opérateur « bit à bit » AND :

- il est représenté par le caractère : **&**
- il s'applique à deux opérandes placés de part et d'autre de l'opérateur
- il réalise l'opération « ET » entre chaque bit des deux opérandes

Exemple :

```
int a =  92;      // binary:     0000000001011100
int b = 101;      // binary:     0000000001100101
int c = a & b;    // result:     0000000001000100,  or 68 in decimal.
```

L'une des utilisations les plus courantes de l'opérateur « ET » est de sélectionner un (ou plusieurs) bit(s) d'une variable, par une opération de masquage :

```
int x    =  0b00101100 ;
int mask =  0b10000000 ; // sélection du bit 7 (b7)
int y    =  x & mask ;    // permet de déterminer la valeur du bit7 de x  (b7=0 si y=0, b7=1 sinon)
                          //  tous les bits autres que b7 ont été masqués.
```

Les entrées/sorties

L'opérateur « bit à bit » OR :

- il est représenté par le caractère : **|**
- il s'applique à deux opérandes placés de part et d'autre de l'opérateur
- il réalise l'opération « OU » entre chaque bit des deux opérandes

Exemple :

```
int a =  92;    // binary:    0000000001011100
int b = 101;    // binary:    0000000001100101
int c = a | b;  // result:     0000000001111101,    or 125 in decimal.
```

L'une des utilisations les plus courantes de l'opérateur **« OU »** est la **mise à 1** d'un (ou plusieurs) bit(s) d'une variable sans toucher aux autres, par une opération de masquage:

```
x  =  x  |  0x18 ;    // permet de mettre à 1 les bits 3 et 4 de la variable x
```

Les entrées/sorties

L'opérateur « bit à bit » XOR :

- il est représenté par le caractère : **^**
- il s'applique à deux opérandes placés de part et d'autre de l'opérateur
- il réalise l'opération « OU exclusif » entre chaque bit des deux opérandes

Exemple :

```
int x = 12;      // binary:  1100
int y = 10;      // binary:  1010
int z = x ^ y;   // binary:  0110,     or decimal 6.
```

L'une des utilisations les plus courantes de l'opérateur **« OU exclusif »** est le **changement d'état** d'un (ou plusieurs) bit(s) d'une variable sans toucher aux autres, par une opération de masquage:

```
x = x ^ 18 ;   // inverse les bits 1 et 4 de la variable x
```

Les entrées/sorties

Les opérateurs de décalage :

Il y a deux opérateurs de décalage :
- décalage à gauche représenté par les caractères : **<<**
- décalage à droite représenté par les caractères : **>>**

Ces opérateurs s'appliquent à deux opérandes
- celui de gauche est la variable dont les bits seront décalés
- celui de droite est la valeur du décalage

L'expression : *x << y → décale x de y bits à gauche*

Exemple :

```
int a = 5;          // binary: 0000000000000101
int b = a << 3;     // binary: 0000000000101000,   or 40 in decimal
int c = b >> 3;     // binary: 0000000000000101,   or back to 5 like we started with
```

Les entrées/sorties

Les opérateurs de décalage :

Lors d'un décalage à gauche « **x << n** » :

- les n bits de x les plus à gauche (MSB) qui vont sortir, seront définitivement perdus
- les n bits vacants à droite seront remplacés par des '0'.

Exemple :

```
int a = 5;          // binary:   0000000000000101
int b = a << 14;    // binary:   0100000000000000  -  the first 1 in 101 was discarded
```

Les entrées/sorties

Les opérateurs de décalage :

Lors d'un décalage à droite « **x >> n** » :

- les n bits de x les plus à droite (LSB) qui vont sortir, seront définitivement perdus
- le remplacement des n bits vacants de gauche dépend du type de la variable x :

  - si x est non signé, des '0' sont rentrés
  - si x est signé, le bit de signe est propagé *(c-à-d : '0' si x>0 ou '1' si x<0)*

Exemple :

```
int x = -16;                 // binary: 1111111111110000
int y = x >> 3;              // binary: 1111111111111110
```

ou

```
int x = -16;                 // binary: 1111111111110000
int y = unsigned(x) >> 3;    // binary: 0001111111111110
```

Les entrées/sorties

Exemple: clignotement d'une led en utilisant les registres

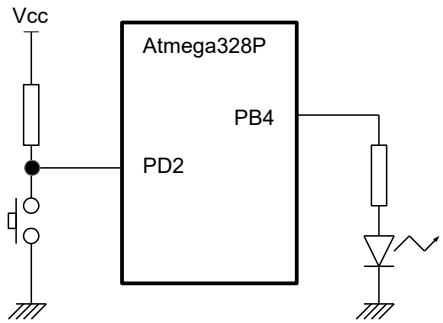
```
#define intervalClignoteTask1 500
unsigned long currentTime=0;
unsigned long previousTimeClignoteTask1 = 0;

void setup() {

    DDRD=DDRD|0b11010000;    // method2: initialize digital PD3 (alias pin 3 o
    PORTD=PORTD|0b1001000;
}

// the loop function runs over and over again forever
void loop() {
    currentTime = millis();

    //TASK1: clignote led 1
    if ((currentTime - previousTimeClignoteTask1) > intervalClignoteTask1) {
        previousTimeClignoteTask1=currentTime;
        PORTD=PORTD^0b01000000;
    }

}
```

Les entrées/sorties

Exemple: « commande d'une led par un bouton poussoir » en utilisant les opérations « bit à bit » sur les registres des ports d'E/S.

La Led est connectée à PB4 et le bouton poussoir à PD2, conformément au schéma :

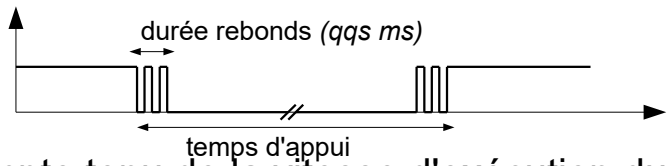
```
void setup() {

   DDRB |= 0x10;            // DDRB |= (1<<DDB4);
}

void loop() {

  if ( (PIND & 0x04) == 0x00 ) {
       PORTB |= 0x10;
   }
   else {
       PORTB &= 0xEF;      //PORTB &= ~(0x10);
   }
}
```



Les entrées/sorties

« Incrémenter un compteur à chaque action sur un bouton poussoir »

Problème : les boutons poussoirs génèrent des rebonds.



Compte tenu de la vitesse d'exécution du programme (quelques Mips), les rebonds sont interprétés comme des appuis successifs très rapides et provoquent des dysfonctionnements

Solution rapide : les boutons poussoirs sont gérés sous forme de tâche. La période d'échantillonage sera > Trebond

Antirebond sous forme d'une tâche

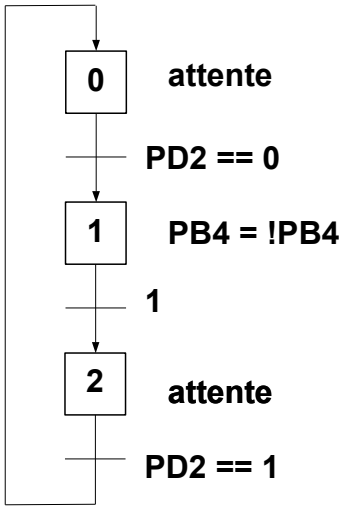
```
#define intervalBPtask 20

unsigned long currentTime;
unsigned long previousTimeBPTask = 0;

// the setup function runs once when you press reset or power the board

void setup() {
    DDRC = 0xFF;
    DDRB = DDRB & 0b11110111; //SWO PB3
}

unsigned int compt=0;
bool stateButton, previousStateButton;
// the loop function runs over and over again forever

void loop() {
    currentTime = millis();
    if ((currentTime - previousTimeBPTask) > intervalBPtask) {

        if ((PINB & 0b00001000) == 0X00) {
            stateButton = true;
        } else {
            stateButton = false;
        }

        if ((stateButton == true)&&(previousStateButton == false)) {
            compt++;
            PORTC=compt;
            Serial.print(res);
        }
        previousStateButton = stateButton;
        previousTimeBPTask = currentTime;
    }
}
```

Les entrées/sorties

Codage d'un graphe d'état *(ou d'un grafcet)* avec l'instruction « switch … case »

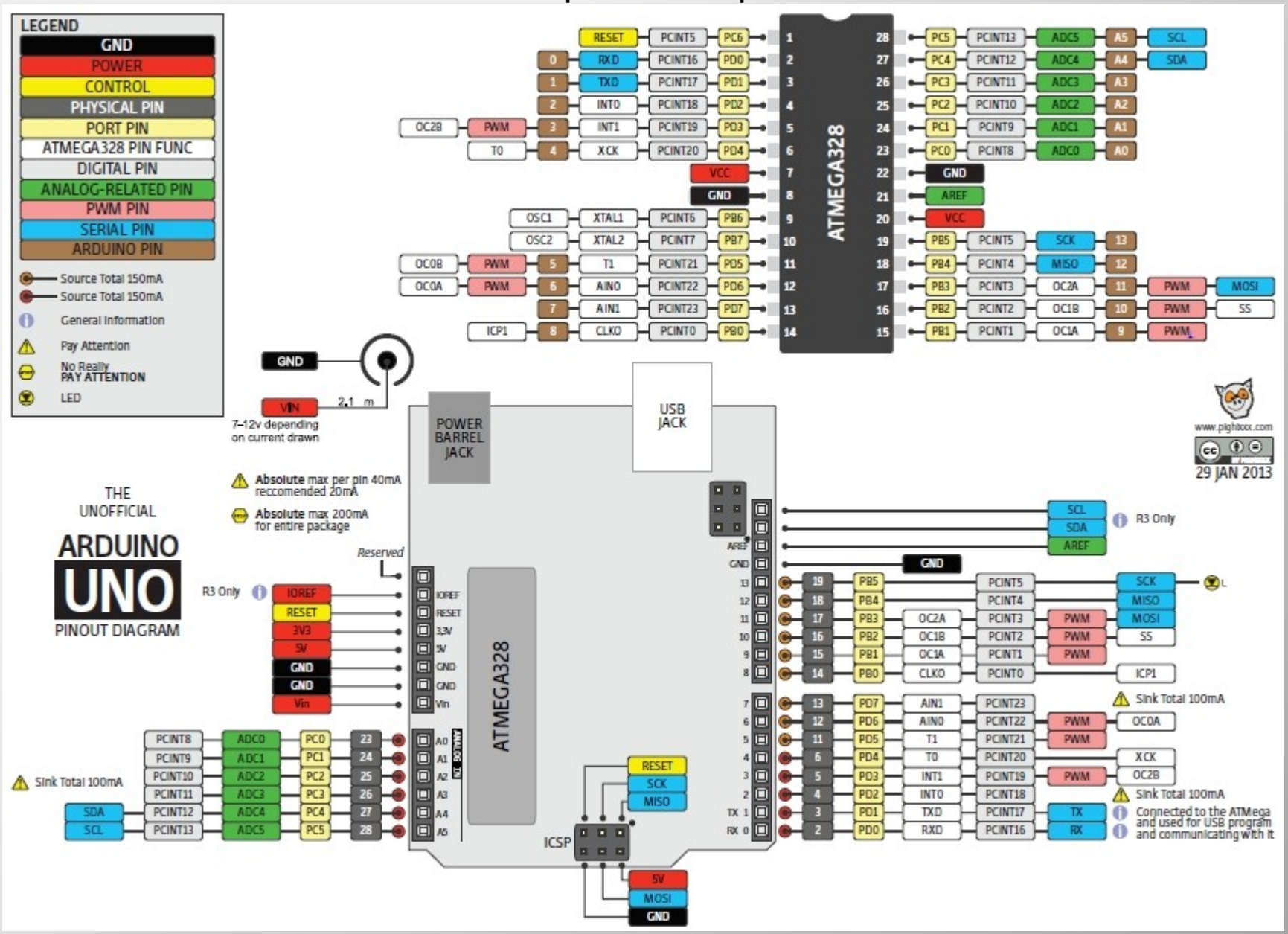
Exemple : « Changement d'état d'une led à chaque action sur un bouton poussoir »
(led sur PB4 et bouton poussoir sur PD2)

```
switch(etat) {
  case 0 :   // pas d'action pour l'étape 0
             if ( (PIND & 4) == 0 ) etat=1 ; // si appui poussoir
             break;
  case 1 :   PORTB ^= 0x10;     // led =!led ;
             etat=2 ;           // changement d'état
             break;
  case 2 :   // pas d'action pour l'étape 2
              if ( (PIND & 4) != 0 ) etat=0 ; // si poussoir relaché
             break;
}
```



Tâche Grafcet

```
// the loop function runs over and over again forever
void loop() {
    currentTime = millis();
    //TASK1:
    if ((currentTime - previousTimeClignoteTask1) > intervalGrafcetTask1) {
        previousTimeClignoteTask1 = currentTime;
        grafcetCompteur();
    }
}

void grafcetCompteur(void) {
    stateBP = digitalRead(bp);
    switch (etatGrafcet) {
        case 0:
            if (stateBP == false) {
                etatGrafcet = 1;
            }
            break;

        case 1:
            compteur++;
            PORTB = compteur;
            etatGrafcet=2;
            break;
```


LEGEND
GND
POWER
CONTROL
PHYSICAL PIN
PORT PIN
ATMEGA328 PIN FUNC
DIGITAL PIN
ANALOG-RELATED PIN
PWM PIN
SERIAL PIN
ARDUINO PIN
Source Total 150mA
Source Total 150mA
General Information
Pay Attention
No Really PAY ATTENTION
LED
ATMEGA328
RESET PCINT5 PC6 1
RXD PCINT16 PD0 2
TXD PCINT17 PD1 3
INT0 PCINT18 PD2 4
OC2B PWM INT1 PCINT19 PD3 5
T0 XCK PCINT20 PD4 6
VCC 7
GND 8
OSC1 XTAL1 PCINT6 PB6 9
OSC2 XTAL2 PCINT7 PB7 10
OC0B PWM T1 PCINT21 PD5 11
OC0A PWM AIN0 PCINT22 PD6 12
AIN1 PCINT23 PD7 13
ICP1 CLKO PCINT0 PB0 14
PC5 PCINT13 ADC5 A5 SCL 28
PC4 PCINT12 ADC4 A4 SDA 27
PC3 PCINT11 ADC3 A3 26
PC2 PCINT10 ADC2 A2 25
PC1 PCINT9 ADC1 A1 24
PC0 PCINT8 ADC0 A0 23
GND 22
AREF 21
VCC 20
PB5 PCINT5 SCK 13 19
PB4 PCINT4 MISO 12 18
PB3 PCINT3 OC2A 11 PWM MOSI 17
PB2 PCINT2 OC1B 10 PWM SS 16
PB1 PCINT1 OC1A 9 PWM 15
GND
VIN
2.1 m
7-12v depending on current drawn
POWER BARREL JACK
USB JACK
www.pighixxx.com
29 JAN 2013
THE UNOFFICIAL
ARDUINO UNO
PINOUT DIAGRAM
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
Reserved
R3 Only
IOREF
RESET
3V3
5V
GND
GND
Vin
ATMEGA328
A0 A1 A2 A3 A4 A5
ANALOG IN
Sink Total 100mA
PCINT8 PCINT9 PCINT10 PCINT11 PCINT12 PCINT13
ADC0 ADC1 ADC2 ADC3 ADC4 ADC5
PC0 PC1 PC2 PC3 PC4 PC5
23 24 25 26 27 28
SDA SCL
ICSP
RESET SCK MISO
5V MOSI GND
AREF GND
SCL SDA AREF
R3 Only
PB5 PCINT5 SCK
PB4 PCINT4 MISO
PB3 PCINT3 OC2A PWM MOSI
PB2 PCINT2 OC1B PWM SS
PB1 PCINT1 OC1A PWM
PB0 PCINT0 CLKO ICP1
Sink Total 100mA
PD7 PCINT23 AIN1
PD6 PCINT22 AIN0 PWM OC0A
PD5 PCINT21 T1 PWM
PD4 PCINT20 T0 XCK
PD3 PCINT19 INT1 PWM OC2B
PD2 PCINT18 INT0
PD1 PCINT17 TXD TX
PD0 PCINT16 RXD RX
TX 1
RX 0
Sink Total 100mA
Connected to the ATMega and used for USB program and communicating with it

Antirebond sous forme d'une tâche

```
#define intervalBPtask 20

unsigned long currentTime;
unsigned long previousTimeBPTask = 0;

// the setup function runs once when you press reset or power the board

void setup() {
    DDRC = 0xFF;
    DDRB = DDRB & 0b11110111; //SWO PB3
}

unsigned int compt=0;
bool stateButton, previousStateButton;
// the loop function runs over and over again forever

void loop() {
    currentTime = millis();
    if ((currentTime - previousTimeBPTask) > intervalBPtask) {

        if ((PINB & 0b00001000) == 0X00) {
            stateButton = true;
        } else {
            stateButton = false;
        }

        if ((stateButton == true)&&(previousStateButton == false)) {
            compt++;
            PORTC=compt;
            Serial.print(res);
        }
        previousStateButton = stateButton;
        previousTimeBPTask = currentTime;
    }
}
```

Les entrées/sorties

Attention à bien faire la différence entre :

- le numéro de la broche du connecteur de la carte Arduino :

. en marron (ARDUINO PIN) sur le Pinout Diagram

. ce sont ces numéros de broche qu'il faut passer en argument aux fonctions des librairies Arduino : *pinMode(**broche**, mode), digitalRead(**broche**), digitalWrite(**broche**, valeur), ...*

- le numéro de la broche du microcontrôleur ATMEGA328 :

. en gris (PHYSICAL PIN) sur le Pinout Diagram

. ce numéro ne sera jamais utilisé dans les programmes

- le rang d'un bit sur les ports d' Entrée/Sortie :

. en jaune (PORT PIN) sur le Pinout Diagram

. utilisé quand on programme au niveau des registres internes, ex : **DDRB |= (1<<DDB4);**

. **exemple :** le bit 2 du PortB (PB2) correspond à la broche physique n°16 de l'ATMEGA328
et à la broche 10 de la carte ARDUINO
DDRB = DDRB | (1<<2) ; est équivalent à : pinMode(10,OUTPUT) ;

Le Pinout Diagram de la carte Arduino UNO vous permet d'établir la correspondance entre les différentes possibilités de repérer une même entrée/sortie.