Les interruptions

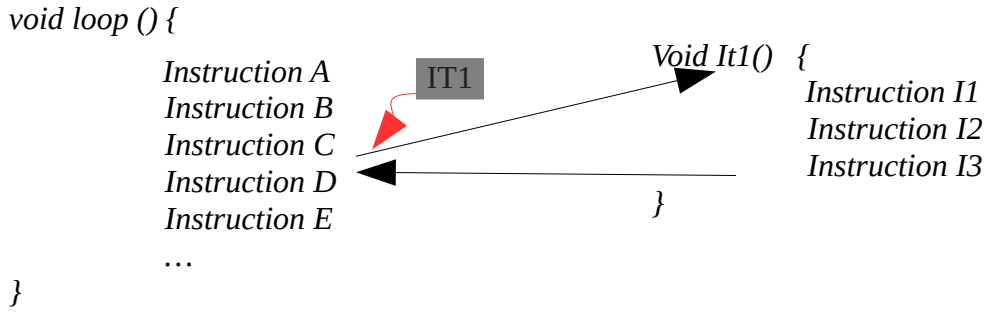
1/ Généralités

1.1/ Qu'est ce qu'une interruption ?

Une interruption (IT) **est un événement** qui, si il est autorisé, **interrompt le programme en cours** pour exécuter une routine (fonction) d'interruption qui lui est associée.

Fonction principale : boucle infinie

Routine d'IT associée à It1



1.2/ Qu'est ce qu'un événement

Un événement peut être matériel ou logiciel. Dans le cas d'un événement matériel ce peut être :

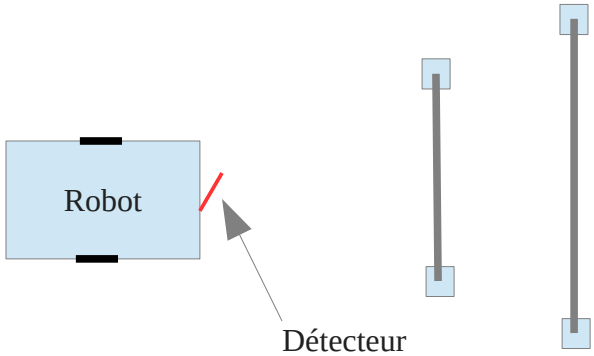
- un front significatif sur une broche d'entrée
- un signal généré par un périphérique (Overflow TIMER, Fin de conversion analogique numérique, Réception d'un caractère sur liaison série ...)

1.3/ Interruption VS pooling, qui est le gagnant ?

La technique du pooling, ou scrutation, consiste à tester l'état d'une entrée dans une boucle infinie. Voyons la mise en œuvre de ces deux différentes techniques sur un exemple concret !

Mise en situation :

Lors du concours de robotique de GEII, le robot finit sa course en frappant une barre en pin en équilibre mais sans faire chuter la deuxième qui se trouve quelques centimètres plus loin. Par conséquent, dès que le robot se rend compte qu'il a percuté la barre il doit stopper ses moteurs le plus rapidement possible !



Le détecteur peut être assimilé à un contact NO qui se ferme en cas de contact avec un objet.

Pooling

```
void setup() {
….
}


void loop () {
//1
Acquerir_Capteurs() ;          // 60ms
Traiter_Donnees() ;            // 30 ms
//2
Commander_Moteurs() ;          // 5ms

if (contact == true)  {
   Stopper_Moteurs() ;         // 5ms
   while(1) ;
}
}                                  // Fin boucle infinie
```

Dans le cas du pooling le temps de réaction du robot en cas de choc avec la barre en pin dépend d'où on se situe dans le programme au moment où l'impact a lieu.
Si l'impact a lieu en 2 : Les moteurs seront stoppés environ 5 + 5 ms après le choc, soit environ 10ms

Si l'impact a lieu en 1 : Les moteurs seront stoppés environ 60 + 30 + 10 + 5 ms après le choc soit presque 100ms

Vu la vitesse du robot, en 50ms il a parcouru plus de distances que l'écart entre les deux barres.

Donc cette solution ne fonctionne pas (enfin, pas toujours) !

Interruption

```
void setup() {
….
// La routine Stop() est associée à un front montant sur  le
//contact NO
attachInterrupt(0,Stop, RISING) ;
}


void loop () {

Acquerir_Capteurs() ;          // 60ms
Traiter_Donnees() ;            // 30 ms
Commander_Moteurs() ;          // 5ms
}                                  // Fin boucle infinie

void Stop() {  //routine d'interruption
   Stopper_Moteurs() ;         // 5ms
}
```

Avec l'interruption, le micro processeur est capable de détecter l'événement « front montant sur le contact » en quelques µs, et ce, quel que soit l'endroit ou il se trouve dans le programme !

Par conséquent suite au front montant quelques µs plus tard la fonction *Stopper_Moteurs* sera exécutée et 5ms après les moteurs seront arrêtés !

Le temps de prise en compte de l'événement est fixe, on parlera de déterminisme !

Cette solution fonctionne.

1.4/ Les différentes sources d'interruptions de l'Atmega328P

- 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

Toutes ces différentes sources d'interruptions sont masquables, c'est à dire qu'elle peuvent être désactivées suite à l'appel de la fonction *noInterrupts*(). Pour autoriser de nouveau les interruptions il faudra invoquer la fonction *interrupts*().

Exemple (sources arduino.cc) :

```
void setup() {}

void loop()
{
  noInterrupts();
  // critical, time-sensitive code here
  interrupts();
  // other code here
}
```

Remarque : On peut aussi indifféremment utiliser les fonctions *sei( )* et *cli( )* pour autoriser ou interdire les interruptions.

Liste des différentes interruptions de l'ATMEGA328P

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

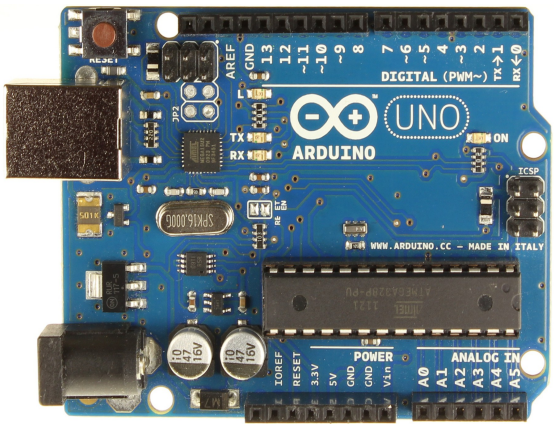
VectorNo.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-ou
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	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
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

1.5/ Les interruptions sur les entrées INT0 et INT1

Le Arduino Uno (ATMega328) peut gérer 2 **interruptions externes sur ses broches INT0** (broches 2) **et INT1** (broche 3)**.**

L'interruption peut être déclenchée si la broche est à l'état bas ou subit un front (montant, descendant ou change d'état).

Exercice : Câbler un BP sur **INT0** de telle sorte à obtenir un front montant lors de l'appui.



1.5.1/ Sans utiliser les registres internes du µc

Il est nécessaire de lier la source de l'interruption (INT0) avec la routine (ISR) qui sera exécutée lorsque cette interruption surviendra. En d'autres termes, comment faire pour que la fonction *myIsr()* s'exécute quand une IT est déclenchée sur INT0 suite à un front montant?

On utilisera la fonction *attachInterrupt(numéro, ISR, mode);*

- *numéro :* C'est le numéro de l'IT, 0 pour INT0 (sur la broche 2) et 1 pour INT1 (sur la broche3)
- *ISR* est la routine d'interruption qui sera appelée lorsque l'interruption surviendra. Cette routine d'interruption est une fonction qui ne renvoit rien et qui ne prend aucun argument, comme ceci : `void myIsr() { ... }`.
- *mode* indique ce qui va provoquer l'interruption. Les valeurs possibles pour *mode* sont :

  -LOW : l'interruption est déclenchée quand la broche concernée est à l'état LOW. Attention, vu qu'il s'agit d'un état et non d'un événement, l'interruption sera déclenchée infiniment tant que la broche est LOW. Ceci entraînant la non exécution du reste du programme donc de *loop()*.
  -CHANGE : l'interruption est déclenchée à chaque front (montant ou descendant) sur la broche concernée.
  -RISING : l'interruption est déclenchée suite à un front montant sur la broche concernée.
  -FALLING : l'interruption est déclenchée suite à un front descendant sur la broche concernée.

Exercice : Sur le précédent schéma, câbler une LED sur la broche 8 et faire le programme qui complémente son état à chaque front montant sur la broche 2. Que pensez vous de ce programme par rapport à la problématique soulevée avec le robot précédemment ?

1.5.2/ En utilisant les registres internes du µc

Le bit7 du registre SREG permet d'autoriser/interdire les interruptions.

Ce bit est mis à 1 par l'appel de la fonction sei() et mis à zéro par cli().On peut aussi, bien entendu,modifier sa valeur par une écriture directe dans le registre SREG !

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**

Ensuite il est nécessaire de valider l'interruption en mettant à 1 le bit INTi correspondant dans le registre EIMSK

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	

Pour choisir le mode de déclenchement de l'interruption on configure le registre EICRA (de la même façon pour INT0 et INT1)

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 13-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Enfin il est nécessaire de lier la routine d'IT avec la source d'IT, pour cela on utilisera la fonction ISR de la façon suivante :

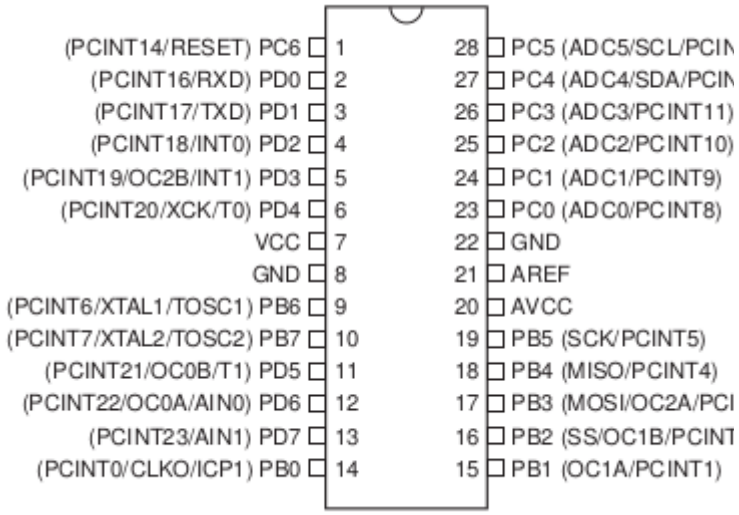
```
ISR(INT0_vect) {
… // Routine d'IT associée à l'IT INT0
}
```

1.6/ Interruptions sur changement d'état des broches : PCINT0 à PCINT23

Les broches de PCINT0 à PCINT 23 à condition quelles soient configurées en entrées peuvent provoquer des IT sur changement d'état.

Ces entrées sont séparées en trois groupes et chaque groupe provoque une IT différente.

Ce sont les IT n° 4,5 et 6 (PCINT0, PCINT1 et PCINT2)



Les entrées PCINT[23:16] sont associées à l'IT PCINT2

Les entrées PCINT[14:8] sont associées à l'IT PCINT1

Les entrées PCINT[7:0] sont associées à l'IT PCINT0

Chaque IT est masquable dans le registre PCICR

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	

Le bit PCIE0 autorise PCINT0, PCIE1 autorise PCINT1 et PCIE2 autorise PCINT2.

Dans chaque groupe, chaque entrée est autorisée individuellement :

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	

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	

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	

Là aussi il sera, bien entendu, nécessaire de lier la source d'IT avec la routine d'IT :

```
ISR(PCINT0_vect) {
  //Routine d'IT associée à PCINT0
}
```