



Découvrez notre Chaîne YouTube "[Ingénierie et Projets](#)"

Découvrez notre Chaîne Secondaire "[Information Neuronale et l'Ingénierie du Cerveau](#)"

Objectifs

1. Comprendre la notion des interruptions
2. Différence entre pooling et interruption
3. Connaître et apprendre à configurer les registres d'interruptions
4. Savoir programmer son routine d'interruption
5. Savoir la procédures d'utilisation d'une interruption externe

Définition

Une interruption comme définition est un événement qui permet d'interrompre le programme principal (la fonction `loop()`) pour exécuter la fonction (ou routine) d'interruption. Lorsqu'une interruption est désactiver (ou masquer), la fonction `loop()` qui sera exécuter par défaut d'une façon infinie. Dans le cas échéant, la routine d'interruption se déclenche dès l'apparition d'un évènement.

Notion du pooling (sans interruption) –

Exemple

La technique de pooling consiste à l'utilisation normale de la boucle `loop()` sans faire appel aux interruptions. On scrute les entrées d'une façon itérative puis on déclenche les sorties



(actions) en **fonctions** des entrées. Le temps de mise à jour des sorties dépend étroitement du moment de l'apparition de l'évènement (exemples: surintensité dans un moteur détecté par un capteur, défaut dans un détecteur, emballement de la température d'une batterie, robot à proximité du mure, fin de course, etc.) et l'emplacement de l'instruction en **cours** dans le programme. Dans ce cas, il faut agir immédiatement. Une latence peut entraîner des conséquences graves. Si par exemple l'évènement est apparu au début du programme, mais le traitement du défaut se trouve à la fin de la boucle loop dans un programme constitué de 200 lignes d'une durée totale d'une seconde (le temps d'exécution de la fonction loop()). Dans ces conditions, la gestion de l'évènement ne sera effectuée qu'après une secondes !

```
loop()
{
Lecture des capteurs;      10 ms
Calcule;                   20 ms
Inst 3;                     20 ms
...                         50 ms
if (V>= Vmax) Arret Moteur;
Inst n;                     30 ms
}
```

Notion de l'interruption – Exemple

Interrompe la boucle loop() et exécution immédiate à l'ordre du microseconde de la routine d'interruption.

```
loop()
{
Inst 1;   10 ms
Inst 2;   1 ms
Inst 3;   20 ms
...       30 ms
Int n;    50 ms
}

IntISR(V>=Vmax)
{
```



```
Arret Moteur;  
}
```

Qu'est-ce qu'un évènement ?

Un évènement peut être un signal électrique Tout Ou Rien (TOR) issu d'un détecteur qui se trouve à l'extérieur de la [carte Arduino](#) dites évènement matériel. Il peut être aussi [logiciel](#) générer un périphérique propre à la carte Arduino ([Microcontrôleur](#)). Un évènement matériel peut être déclenché par :

- Un front montant (passage du 0 à 1 d'un pin digital) ou bien un front descendant (passage du 1 à 0 d'un pin digital). Dans ce cas, la routine d'interruption par transition du signal électrique entrant. Autrement dit, une seule exécution par cycle. Lorsqu'on utilise un bouton poussoir, le programme s'exécute d'interruption se déclenche une seule fois dès qu'on appuie sur le bouton (transition $0 \Rightarrow 1$ ou $1 \Rightarrow 0$). Il faut relâcher le bouton et appuie de nouveau pour recommencer le cycle.
- Niveau logique 1 (pin à 5V par exemple) ou niveau logique 0 (pin à la masse). Dans ce cas, la routine d'interruption s'exécute en boucle tant que le pin est à 1 ou à 0 contrairement à l'utilisation des fronts.

Un évènement logiciel un signal généré par un périphérique propre au microcontrôleur (Exemple : débordement d'un TIMER (Overflow), réception par liaison UART, liaison I2C, fin de conversion du convertisseur A/N, etc.). En résumé, 99% des périphériques du μC dispose d'une ou plusieurs interruptions. La carte Arduino Mega dispose de 57 sources d'interruptions (voir la page 101-102 du datasheet).

- [Datasheet Arduino Mega \(Microcontrôleur ATmega2560\)](#)
- [Datasheet Arduino Uno \(Microcontrôleur ATmega328\)](#)



Les sources d'interruptions de la carte Arduino Mega (57)



Vecteur Source

1	RESET
2	INT0
3	INT1
4	INT2
5	INT3
6	INT4
7	INT5
8	INT6
9	INT7
10	PCINT0
11	PCINT1
12	PCINT2
13	WDT
14	TIMER2 COMPA
15	TIMER2 COMPB
16	TIMER2 OVF
17	TIMER1 CAPT
18	TIMER1 COMPA
19	TIMER1 COMPB
20	TIMER1 COMPC
21	TIMER1 OVF
22	TIMER0 COMPA
23	TIMER0 COMPB
24	TIMER0 OVF
25	SPI, STC
26	USART0 RX
27	USART0 UDRE
28	USART0 TX
29	ANALOG COMP
30	ADC
31	EE READY
32	TIMER3 CAPT
33	TIMER3 COMPA
34	TIMER3 COMPB
35	TIMER3 COMPC
36	TIMER3 OVF
37	USART1 RX
38	USART1 UDRE
39	USART1 TX
40	TWI
41	SPM READY
42	TIMER4 CAPT
43	TIMER4 COMPA
44	TIMER4 COMPB
45	TIMER4 COMPC
46	TIMER4 OVF
47	TIMER5 CAPT
48	TIMER5 COMPA
49	TIMER5 COMPB
50	TIMER5 COMPC
51	TIMER5 OVF
52	USART2 RX
53	USART2 UDRE
54	USART2 TX
55	USART3 RX
56	USART3 UDRE
57	USART3 TX

Définition

External Pin, Power-on Reset, etc.
External Interrupt Request 0
External Interrupt Request 1
External Interrupt Request 2
External Interrupt Request 3
External Interrupt Request 4
External Interrupt Request 5
External Interrupt Request 6
External Interrupt Request 7
Pin Change Interrupt Request 0
Pin Change Interrupt Request 1
Pin Change Interrupt Request 2
Watchdog Time-out Interrupt
Timer/Counter2 Compare Match A
Timer/Counter2 Compare Match B
Timer/Counter2 Overflow
Timer/Counter1 Capture Event
Timer/Counter1 Compare Match A
Timer/Counter1 Compare Match B
Timer/Counter1 Compare Match C
Timer/Counter1 Overflow
Timer/Counter0 Compare Match A
Timer/Counter0 Compare match B
Timer/Counter0 Overflow
SPI Serial Transfer Complete
USART0 Rx Complete
USART0 Data Register Empty
USART0 Tx Complete
Analog Comparator
ADC Conversion Complete
EEPROM Ready
Timer/Counter3 Capture Event
Timer/Counter3 Compare Match A
Timer/Counter3 Compare Match B
Timer/Counter3 Compare Match C
Timer/Counter3 Overflow
USART1 Rx Complete
USART1 Data Register Empty
USART1 Tx Complete
2-wire Serial Interface
Store Program Memory Ready
Timer/Counter4 Capture Event
Timer/Counter4 Compare Match A
Timer/Counter4 Compare Match B
Timer/Counter4 Compare Match C
Timer/Counter4 Overflow
Timer/Counter5 Capture Event
Timer/Counter5 Compare Match A
Timer/Counter5 Compare Match B
Timer/Counter5 Compare Match C
Timer/Counter5 Overflow
USART2 Rx Complete
USART2 Data Register Empty
USART2 Tx Complete
USART3 Rx Complete
USART3 Data Register Empty
USART3 Tx Complete



Procédure d'utilisation d'une interruption

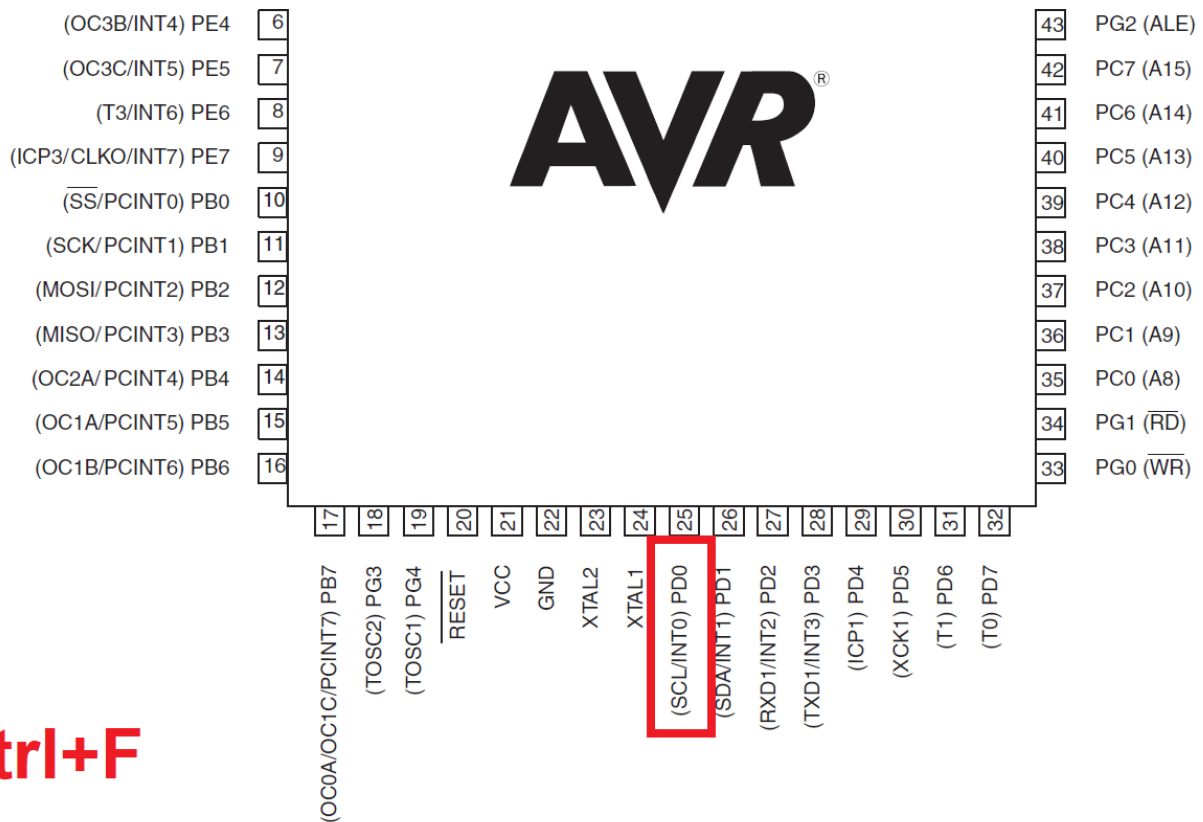
Dans cet exemple on se focalise sur l'interruption externe INT0. La démarche est identique pour les autres types interruptions.

1. Définit le type de l'interruption

- **Pin externe : INT0**
- Périphériques (TIMERS, ADC, UART, etc.)

2. Chercher le pin correspond à l'interruption en fonction du type de la carte Arduino

Faire correspondre l'interruption INT0, INT1, etc. avec le pin physique est une étape importante. L'emplacement du pin dépend du type du microcontrôleur. Il faut se référencer au datasheet pour ne pas se tromper de l'emplacement du pin. Ici, on va utiliser l'interruption INT0 en utilisant la carte Arduino Mega. Ci-dessous l'extrait de l'emplacement du pin. Utiliser « Ctrl+F » avec le mot clé « INT0 » pour retrouver des informations relatives à l'interruptions INT0 dans le datasheet.



Ctrl+F

3. Câbler votre montage et configurer le pin dédié en entrée

Configure le pin dédié en entrée en utilisant la fonction pinMode() ou bien le registre DDRx en positionnant le bit concerné à « 0 » : Exemple DDRD=0x00 (8 bits du port D en entrée). Ci-dessous la mapping des pins de la carte Arduino Mega2560:



Pin Arduino

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

Port Mapping

PORTE 0
PORTE 1
PORTE 4
PORTE 5
PORTG 5
PORTE 3
PORTH 3
PORTH 4
PORTH 5
PORTH 6
PORTB 4
PORTB 5
PORTB 6
PORTB 7
PORTJ 1
PORTJ 0
PORTH 1
PORTH 0
PORTD 3
PORTD 2
PORTD 1
PORTD 0
PORTA 0
PORTA 1
PORTA 2
PORTA 3
PORTA 4
PORTA 5
PORTA 6
PORTA 7
PORTB 7
PORTB 6
PORTB 5
PORTB 4
PORTB 3
PORTB 2
PORTB 1
PORTB 0
PORTD 7
PORTG 2
PORTG 1
PORTG 0
PORTL 7
PORTL 6
PORTL 5
PORTL 4
PORTL 3
PORTL 2
PORTL 1
PORTL 0
PORTB 3
PORTB 2
PORTB 1
PORTB 0

Mapping pour ATmega8 et ATmega168

- B (digital pin 8 to 13)
- C (analog input pins)
- D (digital pins 0 to 7)



4. Activation de l'interruption globale (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**

SREG: AVR Status Register – Activer/Désactiver l'interruption globale

Bit 7 – I: Global Interrupt Enable (0x80)

- `SREG|=0x80 // Activation (Positionner le bit 7 à "1")`
- `SREG&=0x7f // Désactivation (Forcer le bit 7 à "0")`

Ou bien

- `sei(): Set interrupt – Activation`
- `cli(): clear interrupt – Désactivation (masquer)`

5. Validation de l'interruption (EIMSK)

La validation d'une interruption spécifique est opérée dans le registre EIMSK et positionner le bit concerne à '1'. Le nombre des interruptions va dépendre du type de la carte (nano, mini, mega, etc.). Consulter le datasheet pour être sûr des interruptions disponibles. Dans le cas de la carte arduino Mega, il dispose de 8 interruptions INT0-INT7, deux pour Arduino uno (INT0 et INT1) seulement!



EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
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	

- **Bits 7:0 – INT7:0: External Interrupt Request 7 - 0 Enable**

EIMSK – External Interrupt Mask Register

Bit 7 6 5 4 3 2 1 0

0x1D INT7 INT6 INT5 INT4 INT3 INT2 INT1 **INT0**

- `EIMSK|=0x01; // Validation de l'interruption INT0`
- Ou bien `EIMSK =0x01;` (non recommandée)

6. Choix du mode de la détection de l'évènement

D'une autre façon, comment la routine d'interruption sera réveillée lorsqu'un évènement se présente dans le pin dédié (par front ou niveau logique). Les registres EICRA/ EICRB permettent de configurer le mode de déclenchement des interruptions INT0, INT1,..., INT7. Chaque registre permet de configurer 4 interruptions EICRA (INT0-INT3), EICRB (INT4, INT4).

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)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
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	

- **Bits 7:0 – ISC31, ISC30 – ISC00, ISC00: External Interrupt 3 - 0 Sense Control Bits**



EICRA/EICRB

Bit 7 6 5 4 3 2 1 0

ISC31 ISC30 ISC21 ISC20 ISC11 ISC10 ISC01 ISC00

- IS_{Cn}1 IS_{Cn}0
- 0 0 **niveau** bas '0' dans INT_n
- 0 1 **front montant** & **descendant** dans INT_n (changement d'états: passage du 0 à 1 ou bien 1 à 0)
- 1 0 **Front descendant** dans INT_n
- 1 1 **Front montant** dans INT_n

Dans notre exemple, on utilise l'interruption INTO alors les bits ISC01 et ISC00 qui sont concernés. Dans le cas de déclenchement par front montant, il faut positionner les deux bits à '1'. Autrement dit il faut charger le registre EICRA avec 0x03 de la façon suivante: $EICRA=0x03$ ou bien $EICRA|=0x03$;

7. Enfin, il faut programmer la fonction (routine) de l'interruption

Interrupt sub-routine (ISR) est la fonction par défaut utiliser pour programmes les interruptions. Elle ne prend aucun argument en entrée et elle ne retourne aucun résultat. En revanche, il faut bien préciser le vecteur d'interruption. Chaque interruption dispose d'un vecteur qui lui propre. La carte Arduino Mega dispose de 57 interruptions, chaque interruption est caractérisée par son vecteur et adresse. Consultez le datasheet du microcontrôleur pour connaître les vecteurs (voir la page 101 pour la carte Arduino Mega). Vous pouvez dupliquer la fonction autant de fois, il faut juste préciser le vecteur pour différencier entre les routines. Ci-dessous et un exemple d'utilisation de la fonction ISR().



Syntaxe de de définition de la routine d'interruption

```
ISR(Nom_vect)
{
    // Ici le code de votre interruption
}
```

Exemple

```
ISR(INT0_vect)
{
    // Programme INT0
}

ISR(INT1_vect)
{
    // Programme INT1
}

ISR(INT2_vect)
{
```



Arduino #35: les interruptions en 7 étapes

```
// Programme INT2  
  
}
```

Click to rate this post!

[Total: 6 Average: 3.5]

[Nous Soutenir](#) 