ATMEGA 328P Interruptions

Table des matières

Interruptions

1- Résumé.

J'avais besoin de générer 4 interruptions a partir de 4 boutons indépendants.

Après avoir lu que l'on pouvait générer une interruption à partir de toutes les broches d'Entrée/Sortie de l'atmega328p j'ai regardé du coté de la fonction arduino « attachInterupt() », a priori elle ne gère pas ce cas, en tout cas je n'ai pas vu comment faire. Je me suis alors tourné vers la « bible », la datasheet atmell.

Au départ ce document est donc un document a usage personnel, il est donc adapté à moi et au micro ATMega328p. Ce n'est pas un cours. Je rends ce document public bien qu'il puisse être confus sur certains points au cas où il pourrait intéresser d'autres personnes.

Faisant partie d'une génération qui a appris l'anglais en étudiant Shakespeare dans le texte → résultats catastrophiques, j'ai pris le parti pour éviter toutes incompréhensions dues à une lecture mal maîtrisée d'écrire une traduction *libre* du chapitre de la datasheet de l'atmega328p traitant des interruptions.

Le document de départ étant la datasheet Atmell les broches ne sont pas repérées par leur dénomination arduino mais par celle d'Atmell.

De même il est fait largement appel aux écritures directes dans les registres sans passer par les fonctions arduino (pinMode, digitalWrite etc).

Schéma du document :

1. Identification des broches de l'ATMEGA328p.
   Identification de l'interruption associée à chaque broche.

2. Traduction « libre » de la partie de la spécification technique de l'ATMEGA328 qui traite des interruptions et des registres associés.

3. Exemples trouvés sur le forum arduino

4. Références

2- Relation entre broches et les interruptions pour ATMega 328P

Référence BROCHE		Interruption	Vecteur d'interruption
Arduino	ATMega		
8	PB0	PCINT0	PCIE0
9	PB1	PCINT1	
10	PB2	PCINT2	
11	PB3	PCINT3	
12	PB4	PCINT4	
13	PB5	PCINT5	
--Xtal--	PB6	PCINT6	
--Xtal--	PB7	PCINT7	
A0	PC0	PCINT8	PCIE1
A1	PC1	PCINT9	
A2	PC2	PCINT10	
A3	PC3	PCINT11	
A4	PC4	PCINT12	
A5	PC5	PCINT13	
Reset	PC6	PCINT14	
0	PD0	PCINT16	PCIE2 INT0 → PD2 INT1 → PD3
1	PD1	PCINT17	
2	PD2	PCINT18	
3	PD3	PCINT19	
4	PD4	PCINT20	
5	PD5	PCINT21	
6	PD6	PCINT22	
7	PD7	PCINT23	

Remarque très importante :
Seuls les vecteurs d'interruption INT0 et INT1 sont indépendants.

INT0 ne s'applique qu'à la broche PD2.
INT1 ne s'applique qu'à la broche PD3.

(Les broches PD2 et PD3 peuvent aussi être gérées à partir du vecteur PCIE2.)

Ces deux vecteurs d'interruption disposent d'un choix multiple pour le déclenchement : niveau bas, basculement de niveau, front montant ou descendant.

Les 3 vecteurs d'interruption PCIE0, PCIE1 et PCIE2 s'appliquent à un port complet, l'identification de la broche à l'origine de l'interruption est à la charge du programmeur.
Ces 3 vecteurs d'interruption ne disposent que du choix basculement de niveau pour le déclenchement de l'interruption. Avec un bouton ils se déclenchent à l'enclenchement et au relâchement. C'est au programmeur à différencier ces deux états.

3- Traduction « libre » de la partie de la Spec Atmel Atmega 328P traitant des interruptions.

3.1- Liste et priorité des interruptions pour l'atmega 328P

Plus l'adresse d'un vecteur est basse plus il est prioritaire. La priorité absolue est celle du vecteur RESET (adresse 0x0000).

Liste des priorités :

Numéro Vecteur	Adresse du Programme (2)	Source	Définition de l'interruption
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
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

- Quand le Fuse BOOTRSF est programmé, au moment du Reset le micro saute à l'adresse du bootloader.
- Quand IVSEL du registre MCUCR est positionné à 1 les vecteurs d'interruption sautent au début de la zone de mémoire flash du bootloader. L'adresse effective des vecteurs d'interruption est celle indiquée dans le tableau augmentée de l'adresse de départ de la section du bootloader.

3.2- Généralités sur les interruptions

Les interruptions « Externes » sont déclenchées par les pins INT0, INT1 ou une quelconque des pins PCINT 0 à 23.

Les interruptions restent fonctionnelles même si les pins INT0, INT1 ou PCINT 0 à 23 sont configurées en sortie.
L'interruption PCI2 se déclenche si une quelconque des pins PCINT23-16 change d 'état.
L'interruption PCI1 se déclenche si une quelconque des pins PCINT14-8 change d 'état.
L'interruption PCI0 se déclenche si une quelconque des pins PCINT7-0 change d 'état.

Les registres PCMSK2, PCMSK1 et PCMSK0 contrôlent quelle broche est à l'origine du changement d'état.

Les changements d'état sur les broches PCINT0-23 sont détectés de manière asynchrone.
Ceci implique que ces interruptions sont capable de réveiller le micro qui aurait été mis en veille.

3.3- Interruptions INT0 et INT1

Les interruptions INT0 & INT1 disposent chacune de leur propre vecteur d'interruption et peuvent être déclenchées sur un front montant, descendant, un niveau bas ou un basculement de niveau.

La configuration s'effectue dans le registre EICRA.

Choix *niveau bas* : l'interruption dure aussi longtemps que le niveau est maintenu à l'état bas.

Choix *front* : La configuration sur un front nécessite la prise en compte de l'horloge « I/Oclock » (voir page 26 de la spec de l'atmega328).
Ce mode ne peut pas servir à réveiller un micro, car en mode veille, l'horloge est arrêtée sauf pour le mode « idle » ou l'horloge est maintenue.

Remarque :

Micro réveillé par une interruption sur le niveau :

Le niveau devra être maintenu suffisamment longtemps pour laisser au micro le temps d'intercepter l'interruption. Si ce n'est pas le cas le micro sortira quand même du mode veille mais les interruptions ne seront pas générée.

Pour le réglage du temps de réveil voir page 26 de la spécification technique de l'ATMega328p le réglage des « fuses ».

3.4- Description des registres.

3.4.1- Registres généraux.

La gestion des interruptions peut nécessiter de connaître deux registres généraux

MCUCR (MCU Control Register)

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

Valeur initiale	0	0	0	0	0	0	0	0

IVSEL Interrupt Vector Select

- = 0 → les vecteurs d'interruptions sont placés en début de mémoire flash.
- = 1 → les vecteurs d'interruptions sont placés en fin de mémoire flash.

IVCE Interrupt Vector Change Enable

- = 0 → interdit la modification de la valeur de IVSEL
- = 1 → autorise le changement de valeur de IVSEL

SREG Avr Status Register

Bit	7	6	5	4	3	2g	1	0
0x3F(0x5F)	**I**	**T**	**H**	**S**	**V**	**N**	**Z**	**C**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

I → Global Interrupt Enable

- I = 0 → interdit **toutes** les interruptions de manière globale.
- I = 1 → autorise les interruptions de manière globale.
- Chaque interruption doit être individuellement autorisée et configurée dans son registre spécialisé.

Dés qu'une interruption de déclenche elle met automatiquement le bit **I** à 0 bloquant ainsi la possibilité d'avoir plusieurs interruptions simultanées. L'instruction RETI repositionne le bit **I** à 1 permettant ainsi la prise en compte des interruptions immédiatement suivantes.
Le bit **I** peut être directement positionné à 1 avec sei() et à 0 avec cli()
reti() = return from interrupt

Les bits suivant sont cités pour mémoire, ils ne concernent pas les interruptions.
T → Copy Storage.
H → Half Carry Flag
S → Sign Bit, S = N ⊕ V
V → Two's Complement Overflow Flag
N → Negative Flag
Z → Zero Flag
C → Carry Fla

Remarque : Dans tout ce qui suit il sera considéré que le bit I de SREG est positionné à 1 → sei()

3.4.2- Présentation synthétique des registres spécifiques aux interruptions.

EICRA	External Interrupt control register A	Bit 7..4 Bit 3..2 Bit 1..0	Reserved bits ISC11, ISC10 : Interrupt Sense Control 1 ISC01, ISC00 : Interrupt Sense Control 0
EIMSK	External Interrupt Mask Register	Bit 7..2 Bit 1 Bit 0	Reserved bits INT1 External Interupt Request Enable INT0 External Interupt Request Enable
EIFR	External Interrupt Flag Register	Bit 7..2 Bit 1 Bit 0	Reserved bits INTF1 : External Interrupt Flag 1 INTF1 : External Interrupt Flag 1
PCICR	Pin Change Interrupt Control Register	Bit 7..3 Bit 2 Bit 1	Reserved bits PCIE2 : Pin Change Interrupt Enable 2 PCIE1 : Pin Change Interrupt Enable 1

		Bit 0	PCIE0 : Pin Change Interrupt Enable 0
PCIFR	Pin Change Interrupt Flag Register	Bit 7..3 Bit 2 Bit 1 Bit 0	Reserved bits PCIF2 : Pin Change Interrupt Flag 2 PCIF1 : Pin Change Interrupt Flag 1 PCIF0 : Pin Change Interrupt Flag 0
PCMSK1	Pin Change Mask Register 1	Bit 7..0	PCINT23 : Pin Change Enable Mask 23..16
PCMSK2	Pin Change Mask Register 2	Bit 7 Bit 6..0	Reserved bit PCINT14..8 Pin Change Enable Mask 14..8
PCMSK3	Pin Change Mask Register 3	Bit 7..0	PCINT7..0 : Pin Change Enable Mask 7..0

Nota : les bits dont la valeur est repérée par un tiret « **-** » sont des bits réservés, leur valeur doit toujours être égale à 0.

3.4.3- EICRA External Interrupt control register A

Bit	7	6	5	4	3	2	1	0
(0x69)	-	-	-	-	ICS11	ICS10	ICS01	ICS00
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

Bits 3-2 et 1-0 : ISC = Interrupt Sense Control

Pour que l'interruption externe soit active il faut que le masque EIMSK soit également configuré.

Configuration des bits :

ISC11 (*ISC01*)	ISC10 (*ISC00*)	Description
0	0	Déclenchement sur un niveau bas
0	1	Déclenchement sur changement de niveau
1	0	Déclenchement sur front descendant
1	1	Déclenchement sur front montant

Tableau 1

Si les modes de déclenchement sur un front ou un changement d'état sont sélectionné l'impulsion devra durer plus qu'une période d'horloge pour pouvoir être prise en compte.

3.4.4- EIMSK External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0
0x1D(0x3D)	-	-	-	-	-	-	INT1	INT0
Read/Write	R	R	R	R	R	R	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

INT1 (INT0) **:**

Conditions **:**

- INT1(ou INT0) est mis à 1.
- Le mode de déclenchement est défini dans EICRA.

Actions :
Une interruption correspondante au External Interrupt Request 1 est exécuté à partir du vecteur d'interruption INT1 (ou INT0)

3.4.5- EIFR External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0
0x1C(0x3D)	-	-	-	-	-	-	INTF1	INTF0
Read/Write	R	R	R	R	R	R	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

INTF1 (INTF0)
Conditions :

- EIMSK bit INT1(ou INT0) = 1
- le choix du mode de déclenchement est fait sur un front ou un changement de niveau

Action :

- Quand une demande d'interruption apparaît sur la broche sélectionnée le micro se branche sur le vecteur d'interruption correspondant.
- Le drapeau est remis à 0 à la fin de la routine d'interruption.

Remarque : le drapeau est toujours à 1 si le mode niveau bas est sélectionné.

3.4.6- PCCIR Pin Change Interrupt Control Register

Bit	7	6	5	4	3	2	1	0
0x68	-	-	-	-	-	PCIE2	PCIE1	PCIE0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

PCIEx Pin Change Enable x
PCIEx doit être mis à 1 pour permettre les interruptions.

- PCIE2 gère les interruptions PCINT23 à 16 (PORT D)
- PCIE1 gère les interruptions PCINT14 à 8 (PORT C)
- PCIE0 gère les interruptions PCINT7 à 0 (PORT B)

N'importe quel changement de niveau sur les broches gérées provoquera une interruption.
Les broches prévues pour générer une interruption doivent être configurées individuellement avec le registre PCMSKx.
L'identification de la broche du port à l'origine de l'interruption est à la charge du programmeur

Remarque :

Contrairement aux interruptions int0 et int1 il n'est pas possible de choisir le mode de déclenchement, le seul mode disponible est sur basculement de niveau.

3.4.7- PCIFR Pin Change Flag Register

Bit	7	6	5	4	3	2	1	0
0x1B(0x3B)	-	-	-	-	-	PCIF2	PCIF1	PCIF0
Read/Write	R	R	R	R	R	R/W	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

PCIFx Pin Change Interrupt Flag x

Quand un changement se produit sur une des broches

- PCIE2 gère les interruptions 23 à 16
- PCIE1 gère les interruptions 14 à 8
- PCIE0 gère les interruptions 7 à 0

- PCIFx est mis à 1, si PCIEx de PCICR =1 le micro se branche sur le vecteur d'interruption corespondant.

Le drapeau est remis à 0 a la sortie de la routine d'interruption.

3.4.8- PCMSKx Pin change Mask Register x

PCMSK2 Pin change Mask Register 2

Bit	7	6	5	4	3	2	1	0
0x6D	PCINT23	PCINT22	PCINT21	PCINT20	PCINT19	PCINT18	PCINT17	PCINT16
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

PCMSK1 Pin change Mask Register 1

Bit	7	6	5	4	3	2	1	0
0x6D	-	PCINT14	PCINT13	PCINT12	PCINT11	PCINT10	PCINT9	PCINT8
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

PCMSK0 Pin change Mask Register 0

Bit	7	6	5	4	3	2	1	0
0x6D	PCINT7	PCINT6	PCINT5	PCINT4	PCINT3	PCINT2	PCINT1	PCINT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Valeur initiale	0	0	0	0	0	0	0	0

Chaque PCINTx sélectionne si l'interruption sur changement d'état est en service (=1) ou non (=0) .

4- Forum arduino Généralité sur le traitement des interruptions.

4.1- Quelques points de base :

1. Quand un ISR (Interrupt Service Routine) est appelée le micro-controleur désactive automatiquement les interruptions le temps de l'exécution de l'ISR.
2. Les variables partagées entre l'ISR et le reste du programme doivent être déclarées « volatile » `volatile unsigned nom_de_la_variable;`
3. Les autres variables internes à l'ISR n'ont pas besoin d'être déclarée « volatile ». En cas de doute les déclarer «volatile» par sécurité.
4. Avec un micro 8bits seules les variables de la taille d'un octet peuvent être adressée atomiquement, les variables de plus d'un octet doivent être protégées en désactivant les interruptions pendant les temps de lecture ou d'écriture.

Mode opératoire :

- sauvegarder le registre SREG
- désactiver les interruptions
- réaliser l'accès
- restaurer le registre SREG
- réactiver les interruptions

Exemple :

```
unsigned long CalledFromLoopToGetPulseCounts(void)
{
  unsigned long c;
  {
    uint8_t SaveSREG = SREG;                  // save interrupt flag
    cli();                                    // disable interrupts
    c = PulseCounts;                          // access the shared data
    SREG = SaveSREG;                          // restore the interrupt flag
  }
  return( c );
}
```

La grande majorité des ISR écrite pour l'arduino ne sont pas ré-entrante. Le mécanisme est le suivant :

- Un événement extérieur se produit et déclenche une interruption.
- Le micro stoppe son action en cours, désactive les interruptions et appelle l'ISR affectée au vecteur d'interruption.
- Une fois que l'ISR a terminé son travail le micro rétabli les interruptions et reprend ce qu'il était en train de faire avant l'interruption.

Une ISR ré-entrante fonctionne avec les interruptions toujours actives. Une ISR ré-entrante peut donc être interrompue par une autre interruption et peut être elle même appelée plusieurs fois. Ce type d'interruption est très compliqué à gérer correctement.

Utilité de toujours sauvegarder le registre SREG :
Même dans les cas où la sauvegarde n'est pas strictement obligatoire il est préférable de toujours sauvegarder SREG car il ne faut pas négliger le risque de modifications ultérieures qui, par effet de bord, pourrait rendre l'interruption ré-entrante provoquant alors un blocage du programme difficile à détecter.

4.2- Autre info du forum arduino

Affecter un rôle interruption à des pins.

Interruptions

On affecte le rôle dans le setup
Quand une interruption est détectée il faut lire les pins pour savoir laquelle à été activée.

```
int mPin1 = 5; // H-bridge leg 1
int mPin2 = 6; // H-bridge leg 2

void setup() {
   pinMode( mPin1, OUTPUT );
   pinMode( mPin2, OUTPUT );

   //make analog pins 5 and 6 into pin change interrupts
   PCICR |= (1 << PCIE1);
   PCMSK1 |= (1 << PCINT12);
   PCMSK1 |= (1 << PCINT13);
 }
```

Pour être complet quand une interruption est détecté il faut lire les pins pour savoir laquelle à été activée.

```
/*********************************************************************/
#include <avr/interrupt.h>

ISR (PCINT0_vect) {
  if (PINB & (1<<PB7)) {
    // your code here
  }
}
```

(the pins PCINT0..PCINT7 call lead to the PCINT0_vect where you must then read the port to find out which was the actual interrupt source)

```
/* How to set up additional interrupts in Arduino (ATMEL 328)
7/7/2010
Louis Van Blarigan
This code uses Pin Change interrupts to switch between two display states.
*/

/* Pin to interrupt map:
* D0-D7 = PCINT 16-23 = PCIE2 = pcmsk2
* D8-D13 = PCINT 0-5 = PCIE0 = pcmsk0
* A0-A5 (D14-D19) = PCINT 8-13 = PCIE1 = pcmsk1
*/

#include <SoftwareSerial.h>
#define interrupt_pin 7
#define interrupt_pin2 8

//(Include the needed libraries)
//Define Analog Pin (analog pins are 16-21)
//Define Analog Pin (analog pins are 16-21)

volatile int bump = 0;
SoftwareSerial mySerial = SoftwareSerial(13, 13); //Change LCD Tx and Rx Pins to pins
                                                    of our choosing

void setup()
{
 PCICR  |= (1<<PCIE2)     ;
 PCMSK2 |= (1<<PCINT23)   ;
 MCUCR   = (1<<ISC01) | (1<<ISC00);
 PCICR  |= (1<<PCIE0)      ;
 PCMSK0 |= (1<<PCINT0);

//LCD Configuration
 pinMode(13, OUTPUT);
 mySerial.begin(9600);
 mySerial.print("?f");
```

```
//Sends clear screen command to LCD
//Make pin an input
 pinMode(interrupt_pin, INPUT);
 pinMode(interrupt_pin2, INPUT);

 digitalWrite(interrupt_pin,HIGH) ; //Enable pullup resistor on Analog Pin
 digitalWrite(interrupt_pin2,HIGH); //Enable pullup resistor on Analog Pin

 bump = 0;
 interrupts();
}

void loop()
{
 if (bump == 0)
 {
  mySerial.print("?f");
  mySerial.print("LOW");
 }
 else
 {
  mySerial.print("?f");
  mySerial.print("HIGH");
 }
}


ISR(PCINT2_vect)
{
 bump = 1;
}
ISR(PCINT0_vect)
{
 bump = 0;
}
*-----------------------------------------------------------------------------*
http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1259918394/15


#include <avr/interrupt.h>
#include <avr/io.h>

void setup()
{
     pinMode(11, INPUT);
     Serial.begin(9600);

     PCICR |= (1 << PCIE0);
     PCMSK0 |= (1 << PCINT3);
}
volatile int print_flag = 0;

ISR(PCINT2_vect)
{print_flag = 3;}

ISR(PCINT0_vect)
{print_flag = 1;}

ISR(PCINT1_vect)
{print_flag = 2;}

void loop()
{
  if(print_flag == 1)
  {
    Serial.println("interrupt on pcint 0");
    print_flag = 0;
  }
```

```
  else if (print_flag == 2)
  {
    Serial.println("interrupt on pcint 1");
    print_flag = 0;
   }
   else if (print_flag == 3)
   {
     Serial.println("interrupt on pcint 2");
      print_flag = 0;
   }
}
```

5- Exemple personnel.

J'ai développé deux fichiers : interruptions.h et interruptions.c

la fonction teste_bouton teste quel bouton a été pressé et ne retient qu'un seul sens : variation du niveau haut vers le niveau bas, le relâchement du bouton est ignoré.

Les balises dans les commentaires sont conformes à l'utilisation de Doxigen.

Les boutons sont raccordés sur le port analogique, c'est du au hasard, c'était ce qui me simplifiait le câblage.

5.1- interruptions.h

```
/**
 * @file interruptions.h
 * @brief Défini et lance les interruptions sur le port C
 * @author 68tjs
 * @version 0,1
 * @date 11/02/20
 */
#ifndef INTERUPTIONS_H_INCLUDED
#define INTERUPTIONS_H_INCLUDED

/**
* @fn positionne_interruptions();
* @brief Positionne les interruptions sur les bits 0 à 5 du port C
* @brief autorise globalement les interruptions
*/
void positionne_interruptions();

/**
* @fn teste_boutonPortC();
* @brief Teste quel bouton à été pressé.
*/
int teste_boutonPortC();

#endif // INTERUPTIONS_H_INCLUDED
```

5.2- interruptions.c

```
/* ********************************************/
/*    Interruptions .c                        */
/*  Positionnement des registres              */
/*  Pour valider les interruption sur les broches*/
/*  PC2 (PCINT10) à PC5 (PCINT13)             */
/*                                            */
/*  Définition des routines d'interruption    */
/*  version 0.1    11/02/2012                 */
/*  68tjs                                     */
/* ********************************************/

#include "interruptions.h"
#include "/usr/lib/avr/include/stdlib.h"

/**
* @fn positionne_interruptions();
* @brief Positionne les registres pour les Interruptions suivantes :\n
PC2 -> PCINT10  ; PC3 -> PCINT11  ; PC4 -> PCINT12  ;  PC5 -> PCINT13
```

```
*/
void positionne_interruptions()
{
  PCICR |= (1<<PCIE1) ;  //Sélection du portC
  //Sélectionne les broches 2 à 5 du port C
  PCMSK1 |= (1<<PCINT10) | (1<<PCINT11) | (1<<PCINT12) | (1<<PCINT13) ;
  sei() ;  // Autorise globalement les interruptions
}

/**
* @fn ISR (PCINT1_vect)
* @brief Routine executée par l'interruption
*/
ISR(PCINT1_vect)
{
  uint8_t reg1_sauv;
  uint8_t bouton ;

  if (PINC != 0b00111100)
  {
  // code
  }
  SREG=reg1_sauv ;
}
int teste_boutonPortC()
{
  if ( ((1<<PC2) & ~PINC) == (1<<PC2))  //logique négative à cause des Pull-up actif quand = 0
  {                                     //dans PINC il faut inverser les 1 avec les 0
    // code
  }
  else if (((1<<PC3) & ~PINC) == (1<<PC3))
  {
     // code
  }
  else if ( ((1<<PC4) & ~PINC) == (1<<PC4) )
  {
     //code
  }
  else if ( ((1<<PC5) & ~PINC) == (1<<PC5))
  {
     //code
  }
  else
  {
     // code
  }
}
```

6- Références.

https://sites.google.com/site/qeewiki/books/avr-guide/external-interrupts-on-the-atmega328

Avr LibC

http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html

Forum arduino :

http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1261124850/0

Divers :
http://www.me.ucsb.edu/~me170c/Code/How_to_Enable_Interrupts_on_ANY_pin.pdf