Les interruptions

Une interruption, comme son nom l'indique, consiste à interrompre momentanément le programme que l'Arduino exécute pour qu'il effectue un autre travail. Quand cet autre travail est terminé, l'Arduino retourne à l'exécution du programme et reprend à l'endroit exact où il l'avait laissé.

Les broches d'interruptions

Les interruptions forment une caractéristique indispensable des processeurs modernes, puisqu'elles lui permettent de réagir à des événements internes ou externes qui surviennent sans prévenir, en faisant rediriger le flux d'exécution vers un bloc de code de gestion d'interruption. Une fois ce bloc exécuté, le programme reprend où il en était au moment de l'interruption. Dans les processeurs AVR, vous pouvez activer ou inhiber la réponse à une interruption en modifiant des bits dans les registres de contrôle. La description qui suit est spécifique au modèle ATmega168. Pour les autres modèles, vous irez d'abord consulter les tableaux de l'Annexe B, puis la documentation officielle Atmel. Le modèle ATmega168 propose deux entrées pour interruptions externes, INT0 et INT1. Vous pouvez les configurer pour provoquer une interruption sur un front montant, un front descendant ou un état Bas. Le registre de contrôle nommé EICRA permet de contrôler le comportement exact. Les deux entrées INT0 et INT1 ont besoin d'une horloge d'entrées-sorties. Le mode interruption sur état Bas génère une interruption de façon répétée tant que l'entrée correspondante est maintenue à l'état Bas. Les broches d'entrées-sorties de l'ATmega168 peuvent également servir de sources d'interruption. Les interruptions sur changement de port portent le nom PCINT0 jusqu'à PCINT23, chacune étant associée à une des 24 broches d'entrées-sorties. Lorsqu'une interruption est autorisée, elle est déclenchée lorsque l'état de la broche change, même si cette broche est configurée comme sortie. C'est ce qui permet à une broche de générer une interruption sous le contrôle du logiciel lorsque le programme lui-même provoque le changement d'état de la broche (bien sûr, il faut que la détection d'interruption sur changement d'état du port soit active). Dès qu'une des broches entre PCINT0 et PCINT7 change d'état, elle déclenche une interruption PC0. Pour les broches PCINT8 à PCINT14, cela correspond à l'interruption PC1. Enfin, les broches de PCINT16 à PCINT23 correspondent à l'interruption PC2. La configuration des broches qui déclenchent les interruptions en cas de changement d'état est réalisée par les registres PCMSK0, PCMSK1 et PCMSK2. Quand une interruption qui n'est pas masquée survient, le processeur va chercher la prochaine instruction à exécuter à l'adresse trouvée dans une table de vecteursen mémoire ; cette adresse contient une instruction machine de saut RJMP qui pointe vers le bloc de code incarnant la réaction à l'interruption. Une fois ce bloc exécuté, l'exécution reprend dans le programme normal, juste après l'interruption. La Figure 2.10 montre comment la table des vecteurs d'interruption est exploitée pour rediriger le flot d'exécution vers le code de l'interruption puis revenir au programme principal.

Sur un Arduino Uno basé sur microcontrôleur AVR 328P

INT0 Interruption externe sur la broche 2

INT1 Interruption externe sur la broche 3

0 ou 1 sur un Arduino Uno, ce qui correspond respectivement aux broches 2 et 3.

Sur un Arduino Mega basé sur microcontrôleur AVR 2560

0 à 5 sur un Arduino Mega ce qui correspond, dans l'ordre, aux broches 21, 20, 19, 18, 2 et 3.

Sur un ESP32

L'ESP32 dispose de 26 broches numériques qui peuvent être utilisées pour déclencher l'exécution d'une fonction à l'aide d'une interruption externe

Interruptions sur ESP32

Les interruptions : Syntaxe

methodespourlesinterruptions

```
digitalPinToInterrupt (broche); // convertit un identifiant de broche
en identifiant d'interruption,
                                // à utiliser avec attachInterrupt() et
detachInterrupt() .

attachInterrupt (digitalPinToInterrupt (pin), ISR, mode); // conseillé

attachInterrupt (interruption, ISR, mode); // non recommandé

detachInterrupt (digitalPinToInterrupt (pin));

detachInterrupt (interruption);

noInterrupts (); // désactive les interruptions

interruptions (); // réactiver les interruptions après l'
noInterrupts() de noInterrupts() .
```

Paramètres	**remarques**
interruption	0 ou 1 pour Uno. Identifiant de l'interruption. Ne doit pas être confondu avec le numéro d'identification.
ISR	Routine de service d'interruption. C'est la méthode qui sera exécutée lorsque l'interruption se produira.Appel de la fonction
mode	Provoque le déclenchement de l'interruption, **FALLING:** Passage de l'état haut à l'état bas (détection d'un front descendent).** RASING:** Détection du front montant (Passage de l'état bas à l'état haut). **LOW:** Détection d'un passage à état bas de la broche. **CHANGE:** Lorsque la broche change d'état. Les cartes DUE permettent **HIGH**.

Remarques

Les routines de service d'interruption (ISR) **doivent être aussi courtes que possible**, car elles

mettent en pause l'exécution du programme principal et peuvent donc vider le code en fonction du temps. Généralement, cela signifie que dans l'ISR, vous définissez un drapeau et sortez, et dans la boucle du programme principal, vous vérifiez le drapeau et faites ce que ce drapeau est censé faire.

Vous ne pouvez pas utiliser delay() ou millis() dans un ISR car ces méthodes elles-mêmes reposent sur des interruptions.

Toute valeur modifiée à l'intérieur de la routine d'interruption devra être déclarée comme volatile, afin que le processeur aille chercher la valeur en mémoire et ne se fie pas à ce qui se trouve dans ses registres qui étaient gelés au moment de l'interruption.

Programmes d'exemples

Exemple 1 BP avec Rebonds

Interruption sur le bouton presse

Cet exemple utilise un bouton-poussoir (commutateur tactile) connecté à la broche numérique 2 et à la masse, en utilisant une résistance de rappel interne pour que la broche 2 soit haute lorsque le bouton n'est pas enfoncé.

interrupBP.ino

```
const int LED_PIN = 13;
const int INTERRUPT_PIN = 2;
volatile bool ledState = LOW;

void setup() {
    pinMode(LED_PIN, OUTPUT);
    pinMode(INTERRUPT_PIN, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), myISR,
FALLING); // trigger when button pressed, but not when released.
}

void loop() {
    digitalWrite(LED_PIN, ledState);
}

void myISR() {
    ledState = !ledState;
    // note: LOW == false == 0, HIGH == true == 1, so inverting the
boolean is the same as switching between LOW and HIGH.
}
```

Avec cet exemple simple, les boutons-poussoirs ont tendance à rebondir, ce qui signifie que le circuit s'ouvre et se ferme plus d'une fois avant de s'établir dans l'état final fermé ou ouvert.

Cet exemple ne prend pas cela en compte. Par conséquent, il suffit parfois d'appuyer sur le bouton

pour basculer le voyant plusieurs fois, au lieu d'une fois.

Exemple 2 sur les broches A1 A2 et A3

brochesA1A2A3eninterruption.ino

```
//ce code configure les broches A1,A2 et A3 en interruption.
//ont utilise la bibliotheque PinChangeInt.h
//ont peut egalement le faire avec les broches digitale.
#include <PinChangeInt.h>//appele de la fonction
const byte B1 = A1;
const byte B2 = A2;//Déclaration des boutons
const byte B3 = A3;

volatile int led1=2;
volatile int led2=3;// Déclaration des LED
volatile int led3=4;

volatile bool bascule = false;//Cette variable enregistre l'état d'un
bouton
volatile int i;// Cette variable prmet de savoir si un bouton à été
actionner

int f1(){
  bascule = !bascule;
  i = 1;
  if(bascule){
    digitalWrite(led1,HIGH);
    }
    else{
      digitalWrite(led1,LOW);
      }
  }

int f2(){
  bascule = !bascule;
  i = 1;
  if(bascule){
    digitalWrite(led2,HIGH);
    }
    else{
      digitalWrite(led2,LOW);
      }
  }

int f3(){
  bascule = !bascule;
  i = 1;
  if(bascule){
```

```
    digitalWrite(led3,HIGH);
    }
    else{
      digitalWrite(led3,LOW);
      }
  }

void setup() {// fonction d'initialisation des variables
  Serial.begin(9600);
  pinMode(B1,INPUT);
  pinMode(B2,INPUT);
  pinMode(B3,INPUT);

  PCintPort::attachInterrupt(B1,f1,FALLING);
  PCintPort::attachInterrupt(B2,f2,FALLING);
  PCintPort::attachInterrupt(B3,f3,FALLING);

}

void loop() {
  if(i==1){
  delay(5);
  i=0;
    }else{i=0}
}
```

Exemple 3 avec un bouton poussoir pour passage pietons

FeuxBP.ino

```
const int vert = 10; // Broche 10 pour le  voyant vert
const int orange = 11;// Broche 11 pour le  voyant orange
const int rouge = 12;// Broche 12 pour le  voyant rouge
const int BP1_Pietons = 2;// Broche 2 pour le  Bouton pietons
const int RougePietons = 9;// Broche 9 pour le  voyant Rouge Pietons
const int VertePietons = 8;// Broche 8 pour le  voyant Vert Pietons
const int delais1s = 1000;// Defini un delai de 1 s ==> 1000
millisecondes
const int delais3s = 3000;// Defini un delai de 3 s ==> 3000
millisecondes
const int delais5s = 5000;// Defini un delai de 5 s ==> 5000
millisecondes
volatile int etatBP1 = LOW;
```

```
void setup() {
  pinMode(vert, OUTPUT); // Definit les broches des Voyants en sortie
  pinMode(orange, OUTPUT);
  pinMode(rouge, OUTPUT);
  pinMode(RougePietons, OUTPUT);
  pinMode(VertePietons, OUTPUT);
  pinMode(BP1_Pietons, INPUT);// Definit la broche Bouton pietons en
entrée
  digitalWrite(vert, LOW);// Initialise tous les voyants eteint pour
demmarrer
  digitalWrite(orange, LOW);
  digitalWrite(rouge, LOW);
  digitalWrite(VertePietons, LOW);
  digitalWrite(RougePietons, LOW);
  attachInterrupt(0, memoireBP, CHANGE);

}

void memoireBP(){
etatBP1 = 1;

}



void loop() {
  digitalWrite(VertePietons, LOW);  // eteint le voyant vert pietons
  digitalWrite(RougePietons, HIGH); // allume le  voyant rouge  pietons
  digitalWrite(vert, HIGH);// allume le voyant vert voiture
  delay(delais3s);// pendant 3 secondes
  digitalWrite(vert, LOW);// eteint le  voyant vert voiture
 // if (digitalRead(BP1_Pietons) == 1 ){ // test si le bouton pietons
est appuyé pendant que  le voyant vert voiture est allumé
  if (etatBP1 == 1){
        digitalWrite(orange, HIGH); // si OUI fait la sequence pietons
--> orange voiture allumé
        delay(delais1s);// pendant 1 seconde
        digitalWrite(orange, LOW);//  voyant orange voiture eteint
        digitalWrite(rouge, HIGH);// voyant rouge voiture allumé
        digitalWrite(RougePietons,LOW);// voyant rouge pietons eteint
        digitalWrite(VertePietons,HIGH);// voyant vert pietons allumé
        delay(delais5s);  // pendant 5 secondes
        digitalWrite(VertePietons, LOW);  // eteint voyant vert pietons
        digitalWrite(RougePietons, HIGH);// allume  voyant rouge
pietons
        digitalWrite(rouge, LOW); // voyant rouge voiture eteint
        etatBP1 = 0;
        }  //  et l'on recommence la boucle au debut

  else {   // si NON = Bouton pietons non appuyé pendant le voyant vert
```

```
voiture allumé
    digitalWrite(orange, HIGH);// sequence normale des feux voiture -->
Orange voiture allumé
      delay(delais1s);// pendant  1 seconde
      digitalWrite(orange, LOW);//  voyant orange voiture eteint
      digitalWrite(rouge, HIGH);// voyant rouge voiture allumé
      delay(delais3s);// pendant 3 secondes
      digitalWrite(rouge, LOW); // voyant rouge voiture eteint
      }
    // et l'on recommence la boucle au debut
}
```

Liens Web avec les interruptions

Sur Locoduino et ILS

Autre Exemple

From:
https://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
https://chanterie37.fr/fablab37110/doku.php?id=start:arduino:interruptions&rev=1606934437

Last update: **2023/01/27 16:08**