Chapitre 25 - Les interruptions externes

Qu'est-ce qu'une interruption ?

C'est ce qui arrive dans la vie de tous les jours. Imaginez que vous soyez au téléphone et que l'on sonne à votre porte. Vous demandez à votre interlocuteur de patienter, vous posez votre téléphone puis vous allez ouvrir. C'est le facteur qui vous apporte un colis. Une fois celui-ci réceptionné et la porte refermée, vous retournez prendre votre téléphone pour continuer votre conversation. L'interruption (sonnerie) vous a fait vous détourner de votre programme principal (conversation téléphonique) pour aller réaliser un sous-programme (ouvrir la porte) puis vous avez repris votre programme principal là où vous l'aviez laissé (si vous n'avez pas perdu le fil de votre conversation !).

Pourquoi interruption externe ?

Le microcontrôleur peut gérer un événement extérieur comme quelque chose de prioritaire et pour cela exécuter un sous-programme. On parle d'interruption externe car c'est un événement extérieur au microcontrôleur qui crée l'interruption (par exemple l'appui d'un poussoir). Le microcontrôleur est informé de cet événement (car le poussoir est connecté à une de ses broches) ; il va immédiatement arrêter son programme principal pour aller exécuter un sous-programme de gestion de l'événement, et lorsque ce sous-programme est entièrement exécuté, le microcontrôleur reprend l'exécution du programme principal, là où il s'était arrêté.

Quel est l'intérêt ?

Entre chaque interruption, le microcontrôleur peut faire autre chose, sans être obligé de surveiller les événements extérieurs. Prenons l'exemple de notre poussoir : le programme principal peut surveiller périodiquement le poussoir, tout en faisant autre chose entre chaque surveillance. Mais si le poussoir est appuyé (très brièvement) entre deux surveillances, le risque est grand de louper l'événement, malgré le fait que les microcontrôleurs travaillent très rapidement. Avec une interruption, l'appui sur le poussoir « prévient » le microcontrôleur qu'il doit immédiatement réaliser la tâche que vous avez définie pour l'appui du poussoir.

Concrètement avec Arduino Uno ?

Arduino Uno est conçu autour du microcontrôleur ATMega328, capable de gérer beaucoup d'interruptions internes ou externes (une interruption interne vient du microcontrôleur lui-même, mais ce n'est pas le sujet de ce chapitre). Le compilateur du langage Arduino (qui transforme notre programme en code machine) n'en fait pas autant, mais a prévu toutefois de gérer deux interruptions externes (c'est mieux que rien !). L'événement extérieur doit provoquer un changement d'état (changement de tension) sur les broches numériques 2 ou 3. Par exemple, un poussoir monté sur la broche 2 ou la broche 3, peut provoquer un changement d'état sur la broche (passage de 0 à 5 V, ou l'inverse). La broche 2 (ou la broche 3) est donc une entrée et doit être déclarée comme telle par pinMode (2, INPUT). Mais cela ne suffit pas ; il faut également faire savoir au microcontrôleur, que la broche est utilisée pour créer une interruption, quel sous-programme réaliser alors, et quel type de signal doit créer l'interruption (car on a le choix, comme nous allons le voir).

Quatre fonctions pour nos interruptions externes

attachInterrupt (entrée, ISR, mode)
entrée = 0 si vous utilisez la broche 2, et 1 si vous utilisez la broche 3
ISR (Interrupt Service Routine) est le nom de la fonction à exécuter lorsqu'il y a interruption
mode détermine le type de changement sur la broche, devant déclencher l'interruption, et est égal à :

- LOW si le niveau logique bas (0 V) doit déclencher l'interruption
- CHANGE pour n'importe quel changement de niveau logique (0 à 5 V ou 5 V à 0)
- RISING passage du niveau bas (0 V) à haut (5 V) encore appelé front montant
- FALLING passage du niveau haut (5 V) à bas (0 V) encore appelé front descendant

Le contraire de cette fonction est **detachInterrupt**(entrée) qui permet d'arrêter la génération d'interruption sur une entrée préalablement affectée par attachInterupt (entrée vaut 0 ou 1 selon que la broche utilisée est 2 ou 3).

Parfois, il est nécessaire qu'une portion de code ne soit pas interrompue (code sensible au temps écoulé par exemple), dans ce cas, on utilise juste avant **noInterrupts**() (il n'y a rien dans la parenthèse). Pour ensuite permettre à nouveau les interruptions (une fois le code sensible exécuté), on utilise **interrupts**().

Dans le cas d'Arduino, les interruptions sont autorisées par défaut car utilisées dans des fonctions comme delay, millis ou bien Serial ; **il faut donc être extrêmement prudent lorsqu'on interdit les interruptions** !

Le sous-programme d'interruption (ISR)

Ce sous-programme se présente comme une fonction qui ne peut recevoir aucun paramètre et ne retourne aucun résultat. Si cette fonction modifie des variables du programme principal, ces variables doivent être déclarées comme « volatile » ; cela indique au compilateur de charger ces variables en mémoire RAM et non dans des registres du microcontrôleur afin qu'elles ne soient pas écrasées accidentellement. A l'intérieur de la fonction ISR, delay ne fonctionne pas et millis n'est plus incrémentée. De plus, des données éventuellement reçues par Serial peuvent être perdues.

On voit donc qu'il faut prendre beaucoup de précautions quand on veut utiliser les interruptions. Voici quelques conseils concernant l'ISR, prodigués par Nick Gammon sur son site référencé sur le site officiel d'Arduino (c'est dire qu'ils sont de bons conseils !) :
1 - l'ISR doit être courte
2 - ne pas utiliser delay
3 - ne pas faire de Serial prints
4 - déclarer comme volatile toutes variables partagées avec le programme principal
5 - ne pas tenter d'interdire les interruptions
Vous pouvez retrouver tous les conseils de Nick Gammon à cette adresse :
http://gammon.com.au/interrupts (il y a beaucoup de choses et c'est plutôt pour des « Arduineurs » confirmés).

Lorsqu'une interruption est en cours de traitement, les autres interruptions sont ignorées jusqu'à la fin du traitement de l'interruption en cours ; c'est pourquoi delay et millis (qui utilisent des interruptions) ne fonctionnent plus pendant une interruption en cours, par contre delayMicroseconds fonctionne comme prévu car n'ayant pas recours à un processus d'interruption.

Malgré toutes ces précautions à prendre, les interruptions offrent beaucoup de possibilités et il serait dommage de préférer s'en passer. Maintenant, c'est à vous de jouer, et si votre programme ne se comporte pas tout à fait comme vous l'espérez, sachez qu'il y a moyen de résoudre cela, surtout si vous avez bien respecté les 5 principes listés plus haut.

Arduino Mega2560 et Leonardo

Ces modules Arduino possèdent respectivement 6 et 4 broches permettant des interruptions externes. En consultant le site Arduino – Reference – attachInterrupt, vous trouverez la description des broches et leur déclaration dans la fonction attachInterrupt. Le reste est quasiment identique à l'Uno. Enfin, cette fonction est un peu différente pour les modules Due ; l'interruption peut être attachée à toutes broches disponibles, et il existe le mode HIGH (tout comme LOW) en plus.

À retenir sur les interruptions externes :

- Une interruption est un événement qui oblige le microcontrôleur à interrompre le déroulement de son programme principal pour aller exécuter un sous-programme.
- Les microcontrôleurs permettent de gérer des interruptions internes (provenant du microcontrôleur) ou externe (provenant de l'environnement extérieur) suivant de multiples façons.
- Une interruption externe est produite par le changement d'état d'une broche du microcontrôleur (passage de LOW à HIGH ou réciproquement).
- Le module Uno dispose de deux entrées pouvant produire une interruption externe (les entrées 2 et 3).
- Il existe quatre fonctions dans le langage d'Arduino qui permettent de gérer des interruptions externes sur les entrées 2 et 3.
- Le sous-programme à exécuter lorsqu'une interruption externe a lieu doit respecter certaines règles pour garantir le bon fonctionnement de l'ensemble du processus d'interruption.
- Lorsqu'une interruption est en cours de traitement, les autres interruptions sont ignorées jusqu'à la fin du traitement de l'interruption en cours.