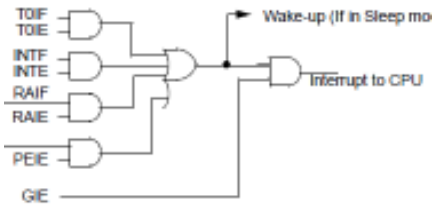
Interruptions sous Arduino

Les interruptions sont déclenchées par des signaux internes ou externes qui interrompent l'exécution du programme, appellent la routine d'interruption qui retourne au programme interrompu à la fin de la tâche d'interruption. Une demande d'interruption fait penser aux boites-aux lettres américaines: un événement active un flag (sémaphore). Si le processeur a été activé pour être dérangé par ce flag, il interrompt le programme principal, et exécute ce qui a été prévu avant de retourner au programme principal. Evidemment, il faut abaisser le flag pour ne pas rappeler immédiatement la routine d'interruption. Cette notion de flag peut aussi jouer entre les interruptions et le programme principal.
Plutôt que de bloquer le programme en attente d'une pression sur une touche, on peut écrire une routine qui, par interruption toutes les 10ms, surveille si la touche est pressée et active un flag. Le programme peut attendre sur ce flag (on n'a alors rien gagné) ou à la fin d'une tâche, voir si la touche a été pressée (éventuellement longtemps avant) et décider pour l'évolution du programme. Là aussi, le flag est remis à zéro quand l'action est faite.
Dans un processeurs, il y a beaucoup de sources d'interruptions possibles, qui activent chacune leur ''interrupt flag'' (IF), mais elles ne peuvent déclencher l'interruption que si c'est autorisé (interrupt enable IE). Un bit général autorise toutes les interruptions (GIE). La routine d'interruption sauve les registres, le programmeur doit gérer les variables qui communiquent entre le programme principal et les variables.



Arduino propose des librairies qui utilisent les interruptions et les lancent sans que l'on s'en rende compte. Le PWM (analogWrite) continue une fois demandé et ralenti l'exécution du programme. C'est facile d'écrire ses propres interruptions, dans la mesure ou elles n'interfèrent pas avec des interruptions d'Arduino ou d'une librairie importée.

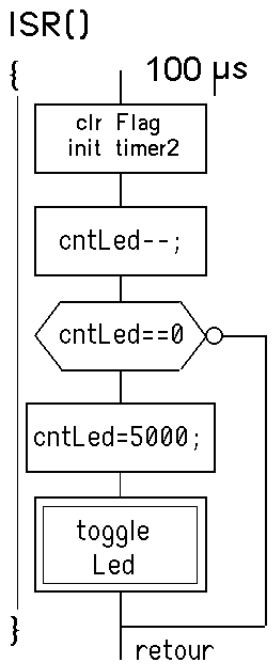
Interruptions par le Timer2
Il y a plusieurs "timers" dans un processeur. Ce sont des compteurs que l'on initialise à une certaine valeur par programmation, qui décomptent à une vitesse fixée par programmation, et activent un flag en arrivant à zéro. C'est la minuterie de ménage, sauf que la sonnerie doit être quittancée, comme le pour le réveil-matin, et on la réarme pour ne pas oublier de remettre une couche de peinture.
La lecture de http://www.uchobby.com/index.php/2007/11/24/arduino-interrupts/ encourage à utiliser le Timer2 et se priver du PWM sur les pins 3 et 11. Utilisons-le pour génèrer une interruption toutes les 200 microsecondes. Toutes les 200 microsecondes on pourra aller s'occuper du ménage. Il n'y aura peut-être rien à faire, mais on sait jamais.
Toutes les 200 microsecondes on va couper la parole au processeur, qui aura le hocquet. C'est gênant? En 200 microsecondes, le C exécute environ 200 à 500 instructions C (le processeur en fait 5 fois plus). La routine d'interruption en fait 10 à 20 au minimum, plus ce qu'il faut faire, toujours rapidement, le gros du travail se fait dans le programme principal. Donc on ralentit ce programme principal de 20%, ce qui est largement compensé par une programmation multi-tâche plus efficace.

Le Timer2, c'est plusieurs registres/compteurs 8 bits aux noms comprimés, avec des bits de modes dont l'effet remplis des pages de documentation. L'initialisation du timer et la mise en route de l'interruption se fait dans le setup. La routine d'interruption est dans un cadre (une fonction comme le setup) préparé par Arduino, pour être compatible avec les interruptions lancées par Arduino).
On doit s'occuper du timer en ensuite, on décide que faire si souvent:
- toutes les 200 us surveiller l'encodeur d'un moteur pour avoir une vitesse assez précise
- toutes les 2ms faire du PFM pour les moteurs et Leds en intensité variable
- toutes les 20ms surveiller des poussoirs, des fin de course, et mettre à jour des compteurs de temps et timouts
- toutes les secondes mesurer des températures et mettre à jour des compteurs de temps de longue durée.
Comme premier exemple, clignotons la Led13 toutes les secondes, mais par interruption.

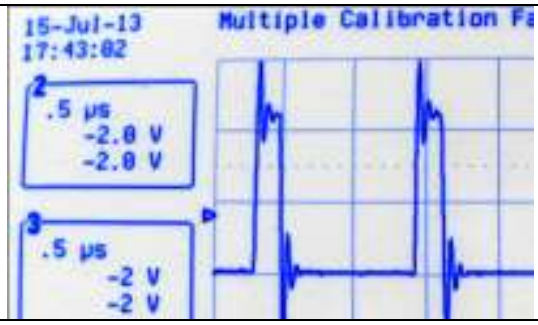
On remarque que dans le set-up il faut configurer des registres du timer. Cette configuration est très simple, car c'est le mode le plus simple du timer: il compte sur 8 bits à 16 MHz/8 . quand il arrive à 256 il y a interruption et pour avoir 100 us il faut initialiser le compteur à –200 (donc 56 théorique, 62 plus précis).
La tâche d'interruption est de compter par 5000 et changer l'état de la Led. Le demi-période est donc de 0.5 secondes.

Evidemment, cela n'a pas de sens de mettre un `delay();` dans une routine d'interruptions.

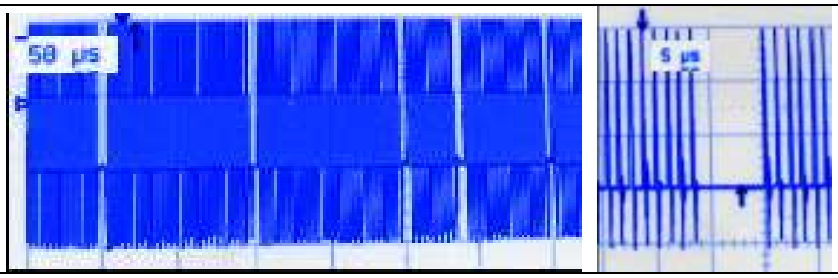
```
void setup()
{
  pinMode(Led, OUTPUT);
  cli();
    TCCR2A = 0; //default
    TCCR2B = 0b00000010; // clk/8
    TIMSK2 = 0b00000001; // TOIE2
   sei();
}

ISR (TIMER2_OVF_vect)
{
  // le flag OVF est mis à zéro par la fonction ISR
  TCNT2 = 62; // 256-200 --> 200X50ns = 100 us
   if (cntLed++ > 10000)
   {
    cntLed = 0;
    digitalWrite Led,!digitalRead(Led));
   }
}

void loop ()
{
  PORTC = ~PORTC ;  // oscillo
  PORTC = ~PORTC ;
}
```



L'observation à l'oscillo est intéressante.
La boucle comporte nos deux instructions qui inversent les bits du port plus des instructions fabriquées par arduino pour fermer la boucle.
Notre instruction dure 0.16 us et la boucle prend 1,12 us. Donc le saut prend 1.12-2*0.16 = 0.8 us, soit 5 instructions.



Ce qui nous intéresse c'est la durée de l'interruption.
On voit que l'oscillation du PORTC est interrompue toutes les 100 us, ce que l'on a programmé, pour une durée de 6-7 us.



Notre routine d'interruption n'a pas toujours la même durée, à cause du .if, mais le plus souvent ce n'est que 5-6 instructions, moins de 2 us. La routine d'interruption prend le reste, 4 à 5 us.
On voit aussi qu'une autre interruption existe. C'est le timer1 qui gère les fonctions millis et micros.

Plusieurs tâches dans la routine d'interruption

Ce mécanisme très simple se prête à la gestion des tâches d'importance différentes mentionnées plus haut. Il faut définir des compteurs qui vont passer à zéro toutes les 20ms et toutes les secondes, comme dans l'exemple ci-contre, et lorsque le compteur est réinitialisé, on exécute la tâche correspondante.

```
ISR (TIMER2_OVF_vect)    // tous les 200 us
{
  // le flag OVF est mis à zéro par la fonction ISR
  TCNT2 = 62; // 256-200 --> 200X50ns = 100 us
   #include "Encode.h"
  if (cnt20ms-- == 0)
  {
     cnt20ms = 100;
      #include "Pfm.h"
     if (cnt1s-- == 0)
     {
        #include "Timout1s.h"
     }
  }
}
```