# Introduction aux temporisateurs de surveillance (Watchdog Timers)

*Par Michael Barr*

## Pour les systèmes embarqués qui ne peuvent pas être surveillés en permanence par un humain, les temporisateurs de surveillance (watchdog timers) peuvent être la solution.

La plupart des systèmes embarqués doivent être autonomes. Il n'est généralement pas possible d'attendre qu'une personne les redémarre si le logiciel se bloque. Certains systèmes embarqués, comme les sondes spatiales, ne sont tout simplement pas accessibles à des opérateurs humains. Si leur logiciel se bloque, ces systèmes deviennent définitivement hors service. Dans d'autres cas, la rapidité avec laquelle un opérateur humain pourrait réinitialiser le système serait trop lente pour répondre aux exigences de disponibilité du produit.

Un temporisateur de surveillance (watchdog timer) est un composant matériel permettant de détecter automatiquement les anomalies logicielles et de réinitialiser le processeur si nécessaire. De manière générale, un watchdog repose sur un compteur qui décompte à partir d'une valeur initiale jusqu'à zéro. Le logiciel embarqué choisit la valeur initiale du compteur et le réinitialise périodiquement. Si le compteur atteint zéro avant que le logiciel ne l'ait réinitialisé, on considère que le logiciel est défaillant et le signal de réinitialisation (reset) du processeur est activé. Le processeur (et le logiciel embarqué qu'il exécute) redémarre alors comme si un opérateur humain avait coupé puis rétabli l'alimentation.

La figure 1 illustre un montage typique. Comme on peut le voir, le temporisateur de surveillance est ici une puce externe au processeur. Il pourrait toutefois tout aussi bien être intégré à la même puce que le processeur, ce qui est le cas dans de nombreux microcontrôleurs. Dans les deux cas, la sortie du temporisateur de surveillance est directement reliée au signal de réinitialisation du processeur.

*Figure 1 : Montage typique d'un watchdog*

*[Schéma] Temporisateur de surveillance (Watchdog Timer) → signaux "Reset" et "Restart" ↔ Processeur ; entrée "Clock" (horloge) vers le temporisateur de surveillance.*

## Donner un coup de pied au chien (« kicking the dog »)

Le fait de réinitialiser le compteur du temporisateur de surveillance est parfois appelé, en anglais, « kicking the dog » (donner un coup de pied au chien). La métaphore visuelle est celle d'un homme attaqué par un chien féroce : tant qu'il continue de repousser le chien à coups de pied, celui-ci ne peut jamais le mordre. Mais il doit le faire à intervalles réguliers pour éviter d'être mordu. De la même façon, le logiciel doit réinitialiser le watchdog à un rythme régulier, sous peine de provoquer un redémarrage du système.

Un exemple simple est présenté dans le Listing 1. On y trouve une unique boucle infinie qui pilote l'ensemble du comportement du système. Cette architecture logicielle est courante dans de nombreux systèmes embarqués utilisant des processeurs bas de gamme et fonctionnant à une seule fréquence de cycle. L'implémentation matérielle de ce watchdog permet de fixer la valeur du compteur via un registre mappé en mémoire.

### Listing 1 : Réinitialiser le watchdog

```
uint16 volatile * pWatchdog =
  (uint16 volatile *) 0xFF0000;

main(void)
{
  hwinit();

  for (;;)
  {
    *pWatchdog = 10000;
```

```
    read_sensors();
    control_motor();
    display_status();
  }
}
```

Supposons que la boucle doive s'exécuter au moins une fois toutes les cinq millisecondes (par exemple, parce que le moteur doit recevoir de nouveaux paramètres de commande au moins aussi souvent). Si le compteur du temporisateur de surveillance est initialisé à une valeur correspondant à cinq millisecondes écoulées, disons 10 000, et que le logiciel ne contient aucun bogue, le watchdog n'arrivera jamais à expiration : le logiciel réinitialisera toujours le compteur avant qu'il n'atteigne zéro.

## Anomalies logicielles

Un temporisateur de surveillance peut sortir un système de nombreuses situations dangereuses. Cependant, pour être efficace, la réinitialisation du watchdog doit être prise en compte dans la conception logicielle globale. Les concepteurs doivent savoir ce qui pourrait mal tourner dans leur logiciel et s'assurer que le watchdog sera capable de détecter ces problèmes s'ils surviennent.

Les systèmes se bloquent pour de nombreuses raisons. L'erreur de logique menant à l'exécution d'une boucle infinie en est l'exemple le plus simple. Supposons qu'une telle condition survienne dans l'appel à read_sensors() du Listing 1 : aucune autre partie du logiciel (à l'exception des routines d'interruption, si les interruptions sont encore actives) n'aurait alors de nouveau l'occasion de s'exécuter.

Une autre possibilité est qu'un nombre inhabituel d'interruptions survienne pendant un seul passage de la boucle. Tout temps supplémentaire passé dans les routines d'interruption est du temps qui n'est pas consacré à l'exécution de la boucle principale, ce qui pourrait entraîner un délai dangereux dans la transmission de nouvelles instructions au moteur.

Lorsque des noyaux multitâches sont utilisés, des interblocages (deadlocks) peuvent survenir. Par exemple, un groupe de tâches peut se retrouver bloqué, chacune attendant les autres ou un signal externe dont l'une d'elles a besoin, ce qui bloque indéfiniment l'ensemble des tâches.

Si de telles défaillances sont transitoires, le système peut fonctionner parfaitement pendant un certain temps après chaque réinitialisation provoquée par le watchdog. En revanche, une défaillance matérielle pourrait entraîner des redémarrages incessants. Pour cette raison, il peut être judicieux de compter le nombre de réinitialisations provoquées par le watchdog et de cesser les tentatives après un certain nombre d'échecs.

## Leçons de karaté

Une implémentation concrète de watchdog présente généralement une interface logicielle plus complexe que celle du Listing 1. Lorsque la séquence d'instructions nécessaire pour réinitialiser le watchdog est trop simple, il est possible qu'un logiciel défaillant l'exécute par accident. Imaginons un bogue qui écrit sans cesse la valeur 10 000 à chaque emplacement de la mémoire : ce code réinitialiserait alors régulièrement le compteur du watchdog, qui ne se déclencherait peut-être jamais. Pour éviter cela, de nombreuses implémentations de watchdog exigent une séquence complexe d'au moins deux écritures consécutives pour réinitialiser le temporisateur.

Si le watchdog est intégré à votre microcontrôleur, il se peut qu'il ne soit pas activé automatiquement à la réinitialisation de l'appareil : vous devez veiller à l'activer lors de l'initialisation matérielle. Pour se prémunir contre un bogue qui désactiverait accidentellement le watchdog, la conception matérielle rend généralement impossible sa désactivation une fois qu'il a été activé.

Si votre logiciel peut effectuer une boucle complète plus rapidement que la période du watchdog, la structure du Listing 1 peut vous convenir. Les choses se compliquent lorsqu'une partie du logiciel prend beaucoup de temps à s'exécuter — par exemple, une boucle qui attend qu'un élément chauffe jusqu'à une certaine température avant de

rendre la main. De nombreux watchdogs ont une période maximale d'environ deux secondes : si votre attente doit dépasser cette durée, il se peut que vous deviez réinitialiser le watchdog à l'intérieur même de la boucle d'attente. S'il existe de nombreux endroits de ce type dans votre logiciel, la gestion du watchdog peut devenir problématique.

L'initialisation du système est une partie du code qui prend souvent plus de temps que la période maximale du temporisateur de surveillance, par exemple à cause d'un test mémoire ou d'un transfert de données de la ROM vers la RAM. C'est pourquoi certains watchdogs peuvent attendre plus longtemps avant leur première réinitialisation qu'ils ne le font ensuite.

À mesure que des fils d'exécution supplémentaires sont ajoutés au logiciel (sous forme de routines d'interruption ou de tâches logicielles), il devient inefficace de n'avoir qu'un seul endroit dans le code où le watchdog est réinitialisé.

Le choix d'un intervalle de réinitialisation approprié est également une question importante, qui ne peut être traitée qu'au cas par cas, selon le système. Ces questions, ainsi que d'autres plus complexes, sont abordées dans les références indiquées à la fin de cet article.

## En résumé

Un temporisateur de surveillance est un outil précieux pour aider votre système à se remettre de défaillances transitoires. Comme les watchdogs sont désormais couramment intégrés aux microcontrôleurs modernes, cette technique est pour ainsi dire gratuite. Si vous travaillez sur un système critique, le bon sens — ou un organisme de réglementation — vous imposera d'utiliser un watchdog. C'est toujours une bonne idée de rendre vos systèmes plus autonomes.

## Références :

1. Murphy, Niall. « Watchdog Timers », Embedded Systems Programming, novembre 2000, p. 112.

2. Santic, John S. « Watchdog Timer Techniques », Embedded Systems Programming, avril 1995, p. 58.

Voir aussi l'article de Dale Lantrip et Larry Bruner, « General Purpose Watchdog Timer Component for a Multitasking System », Embedded Systems Programming, avril 1997.

---

*Remarques : 1. Les caractéristiques peuvent être modifiées sans préavis. 2. Service ODM/OEM disponible. 3. Pour télécharger le manuel d'utilisation et la fiche technique, consultez www.ibt.ca*