Chapitre 5

Les Interruptions

5.1. Principe Fondamental : Du Polling à l'Interruption

L'interruption constitue un mécanisme fondamental qui révolutionne la manière dont un microcontrôleur interagit avec son environnement. Pour en saisir toute l'importance, il est essentiel de comprendre d'abord l'approche alternative qu'elle remplace : l'attente active, également appelée "polling". Dans un système basé sur le polling, le processeur consacre une partie significative de son temps à interroger périodiquement l'état d'un périphérique pour détecter un événement, par exemple la pression sur un bouton ou la réception d'une donnée série. Cette méthode, bien que simple à mettre en œuvre, présente une inefficacité fondamentale : le processeur reste bloqué dans des boucles de test, consommant de l'énergie et des cycles d'horloge sans exécuter de traitement utile, ce qui dégrade considérablement la réactivité globale du système. L'interruption, en opposition radicale à cette logique, libère le processeur de cette charge en lui permettant de vaquer à ses tâches principales jusqu'à ce qu'un événement spécifique se produise. Lorsque cet événement survient, le périphérique concerné envoie un signal au cœur du processeur, interrompant temporairement l'exécution en cours pour forcer la prise en charge immédiate de l'événement. Ce mécanisme permet ainsi de passer d'une architecture où le processeur "attend" les événements à une architecture où le processeur "est averti" de leur survenue, garantissant une latence minimale et une efficacité énergétique et temporelle optimale.

5.2. Architecture Matérielle des Interruptions : Vectorisation, Flags et Priorités

La mise en œuvre efficace des interruptions repose sur une architecture matérielle sophistiquée intégrée au microcontrôleur. Le premier concept clé est celui de la vectorisation des interruptions. Chaque source d'interruption (broche externe, timer, périphérique de

communication) se voit attribuer un vecteur unique, c'est-à-dire une adresse mémoire spécifique où est stockée l'adresse de la routine de traitement associée. Lorsqu'une interruption se produit, le processeur consulte automatiquement ce vecteur pour se rendre directement à la routine correspondante, sans avoir à déterminer lui-même la source de l'événement, ce qui accélère considérablement le temps de réponse. La gestion de cet événement s'articule autour de flags d'interruption, qui sont des bits situés dans des registres spécialisés. Ces flags sont positionnés par le matériel lorsque l'événement se produit, signalant ainsi que la demande d'interruption est en attente, et doivent être effacés par le logiciel à la fin du traitement pour éviter une reprise intempestive. Par ailleurs, la hiérarchie des priorités introduit une dimension essentielle pour les systèmes complexes. Lorsque plusieurs interruptions surviennent simultanément ou qu'une interruption se produit alors qu'une autre est en cours de traitement, le système doit décider laquelle sera servie en premier. Le mécanisme de priorité, configurable dans la plupart des microcontrôleurs modernes, attribue un niveau de priorité à chaque source d'interruption, permettant aux événements critiques (comme une alarme de sécurité) de préempter des traitements moins urgents (comme la gestion d'une communication série), garantissant ainsi que les contraintes temporelles les plus strictes sont respectées en toutes circonstances.

5.3. Configuration Pratique des Sources d'Interruption

La mise en œuvre concrète des interruptions nécessite une configuration minutieuse des registres du microcontrôleur, qu'il s'agisse d'interruptions externes ou internes. Les interruptions externes concernent des broches d'entrée-sortie spécifiques, souvent désignées comme INT0, INT1, ou regroupées dans un vecteur de changement d'état sur un port entier (Pin Change Interrupt). Leur configuration implique la définition de la sensibilité de déclenchement : on peut programmer la broche pour générer une interruption sur un front montant, un front descendant, un niveau bas, ou, pour certaines architectures, sur les deux flancs. Cette souplesse permet d'adapter le comportement à la nature du capteur ou du signal externe connecté. Par ailleurs, les interruptions internes émanent des périphériques intégrés au

microcontrôleur et sont tout aussi cruciales. Les timers, par exemple, peuvent générer des interruptions lors d'un débordement (overflow) ou lorsqu'ils atteignent une valeur de comparaison (match), fournissant ainsi une base de temps précise et exempte de gigue pour l'ordonnancement de tâches périodiques. Les périphériques de communication série (UART, SPI, I2C) génèrent quant à eux des interruptions pour signaler des événements tels que la fin d'une transmission, la réception d'une donnée dans le buffer, ou la détection d'une erreur de communication. La maîtrise de la configuration de ces différentes sources, via les registres de masquage (qui autorisent ou non chaque interruption) et les registres de contrôle des périphériques, constitue une compétence fondamentale pour le développeur de systèmes embarqués.

5.4. Bonnes Pratiques de Programmation et Routines de Service d'Interruption (ISR)

Au-delà de la simple configuration matérielle, la fiabilité et la réactivité d'un système basé sur les interruptions dépendent largement de la qualité logicielle des routines de service d'interruption (ISR – Interrupt Service Routine). La règle d'or, absolue et incontournable, est de rendre les ISR les plus courtes et les plus rapides possible. Une ISR qui s'éternise bloque l'exécution du programme principal et retarde la prise en charge d'autres interruptions, ce qui peut entraîner une perte d'événements, un dépassement de délais critiques, et une dégradation catastrophique de la réactivité globale du système. Pour respecter ce principe, l'ISR doit se limiter à des opérations élémentaires : sauvegarder l'état du système si nécessaire, lire ou écrire dans des registres ou des buffers matériels pour acquitter l'événement, et éventuellement signaler l'occurrence de l'événement au programme principal. Ce signalement s'effectue idéalement en levant un simple drapeau (flag) ou en déposant une donnée dans une file de messages (queue) gérée par un système d'exploitation temps réel (RTOS). Le traitement lourd et potentiellement long de la donnée ou de l'événement doit être déporté dans le programme principal (la "tâche de fond" ou une tâche dédiée du RTOS), où les interruptions sont réactivées et où l'on peut utiliser sans risque des fonctions non réentrantes ou des opérations prenant du temps. Par ailleurs, il est impératif de veiller à la réentrance du

code : une ISR ne doit pas utiliser de variables globales sans protection adéquate (comme l'utilisation de mots-clés volatile et la désactivation temporaire des interruptions lors de l'accès depuis le programme principal), sous peine de rencontrer des bugs subtils et intermittents liés à des accès concurrents. Enfin, une attention particulière doit être portée à la gestion des flags d'interruption : oublier de les acquitter avant la fin de l'ISR conduit invariablement à une reprise immédiate et infinie de la même interruption, bloquant définitivement le système.