Architecture des Ordinateurs

Chapitre 6 : Traitement des interruptions

Jacques Supcik, Daniel Gachet, Luca Haab

2021-12-22

Table des matières

6 Traitement des interruptions

Les interruptions sont un aspect incontournable des systèmes à microprocesseurs. Elles permettent d'interrompre temporairement l'exécution d'un programme informatique pour traiter des événements prioritaires.

Les périphériques d'entrées/sorties les utilisent généralement pour signaler des événements asynchrones nécessitant un traitement en temps réel, telle la fin de période d'une horloge, la complétion d'une tâche. Elles servent également à économiser du temps CPU en évitant des boucles de scrutation (*Polling Loop*).

Des défaillances du code, des exceptions, peuvent également générer des interruptions temporaires du programme pour être traitées. Elles sont souvent dues à des dysfonctionnements inopinés du logiciel résultant de son exécution, par exemple des instructions erronées, des calculs arithmétiques incorrects ou des accès non autorisés à la mémoire.

Les systèmes d'exploitation modernes utilisent les interruptions logicielles pour réaliser les appels système. Ce type d'interruptions permet d'isoler les processus utilisateurs, des pilotes de périphériques et des fonctions du noyau.

6.1 Concept général

Le traitement des interruptions (*Interrupt Handling*) est un sujet indissociable de la programmation de systèmes embarqués et de systèmes sur puce (figure 6.1).

Une interruption (*Interrupt*) peut être vue comme un appel à une routine de traitement (ISR - *Interrupt Service Routine*). Déclenchée par un événement interne ou externe au processeur, elle suspend l'exécution du programme en cours, puis appelle une routine, laquelle traite l'événement pour finalement retourner au programme interrompu. Hormis un délai, le traitement correct d'une interruption ne doit pas altérer le comportement d'un programme en cours d'exécution.

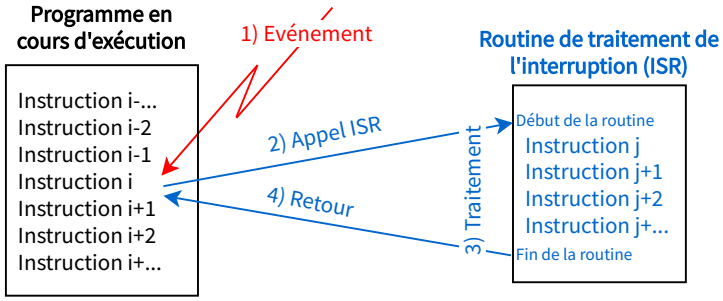

FIGURE 6.1 – Principe

6.1.1 Types d'événements

Les événements externes au CPU, levés par des périphériques, génèrent des interruptions matérielles. Ces interruptions sont dites asynchrones, car elles surviennent indépendamment de l'exécution du programme. Par contre, les événements internes, causés par des interruptions logicielles, des appels système, ou par des exceptions, des dysfonctionnements du programme lors de son exécution,sont dits synchrones. Elles sont dues à l'exécution du code et de ses instructions. Le signal "`reset`" est le dernier type d'événements. Il provoque le reset du processeur et de la réinitialisation de ses périphériques.

Interruptions matérielles

Les interruptions permettent de traiter de façon asynchrone des événements levés par le matériel (*Hardware Interrupt*). Cette capacité à traiter des événements spontanés procure une grande réactivité au système et à ses applications. Elles offrent un traitement rapide et approprié de l'événement (figure 6.2).

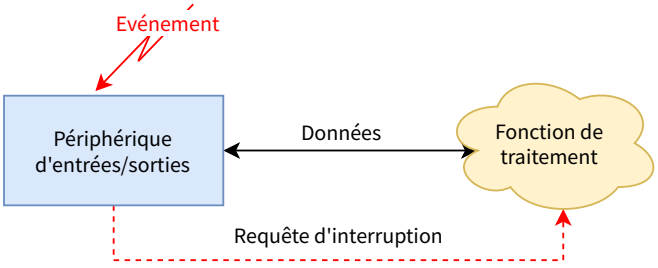

FIGURE 6.2 – Interruption matérielle

Il existe une multitude d'exemples pour illustrer leurs utilités. Pour n'en citer qu'un, prenons une interface de communication. Lors d'échanges d'information entre un pilote et un périphérique d'entrées/-

sorties gérant une interface de communication, interface ayant un comportement non prévisible/aléatoire, il est souhaitable que le périphérique puisse lui-même prendre l'initiative des échanges, en forçant le processeur à suspendre immédiatement l'exécution du programme en cours pour laisser le pilote traiter les requêtes (réceptions ou envois de données).

Des interruptions matérielles peuvent également être mises en œuvre pour faciliter le débogage d'applications logicielles lors de leur réalisation. Certains processeurs implémentent une infrastructure permettant de stopper l'exécution du programme en surveillant le bus d'adresses et/ou de données (*Hardware Breakpoint, Watchpoint*). Si l'adresse et/ou la donnée passant sur leur bus respectif correspond à un point d'arrêt, le processeur suspend l'exécution du programme et active la session de débogage.

Interruptions logicielles

Les interruptions logicielles (*Software Interrupt*) permettent à un processus en espace utilisateur de franchir la barrière de protection en quittant son mode non privilégié pour accéder au mode privilégié du µP et ainsi aux fonctions fournies par l'OS dans l'espace noyau (figure 6.3). Cette fonctionnalité est largement utilisée par les systèmes d'exploitation (GNU/Linux, Windows, MacOS, etc.) pour les appels système (*syscall*). Les processeurs ARM proposent l'instruction "SVC" (*Supervisor Call*) pour générer cette interruption logicielle.

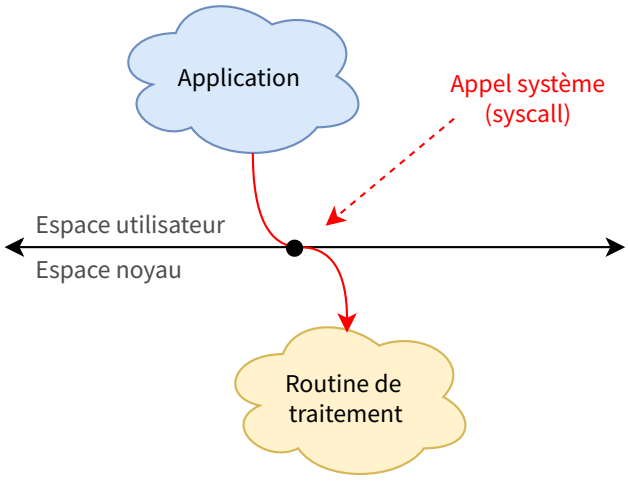

FIGURE 6.3 – Interruption logicielle

Les interruptions logicielles permettent également à des applications de communiquer avec des programmes (*firmware, silicon software*) contenus dans une mémoire non volatile du processeur ou de l'ordinateur. BIOS (*Basic Input/Output System*) des machines "Windows" est un exemple typique.

Elles permettent aussi à des outils de développement de poser des points d'arrêt (*Software Breakpoints*) facilitant le débogage d'applications. Les processeurs ARM proposent l'instruction "BKPT" (*Break Point*) pour générer cette interruption logicielle.

Exceptions

Les défaillances et dysfonctionnements dans les logiciels sont monnaie courante. Lors de l'exécution d'un programme, des erreurs peuvent survenir et perturber son bon déroulement. Les µP implémentent tout un arsenal de mécanismes de reconnaissance de ces exceptions et fournissent aux logiciels une série d'interruptions spécifiques permettant un traitement approprié (figure 6.4).

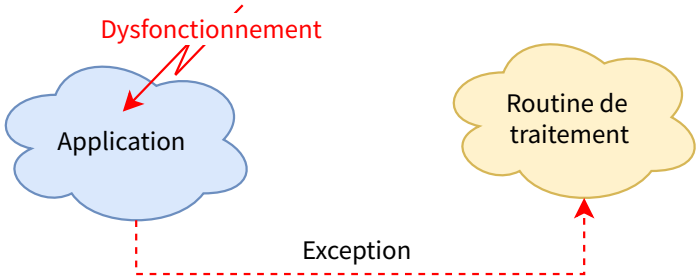

FIGURE 6.4 – Exceptions

Les **exceptions logicielles** les plus usuelles sont dues à des instructions illégales, des erreurs arithmétiques ou des violations de privilèges.

- Une instruction illégale (*Undefined Instruction*) est une instruction pas supportée et pas implémentée par le jeu d'instructions du processeur. La levée de cette exception permet au logiciel de réaliser par exemple des routines émulant le comportement de ces instructions non définies.
- Les unités de calculs (arithmétique et/ou à virgule flottante) sont conçues pour signaler des erreurs de calcul, telles les divisions par zéro. La levée d'une exception permet de détecter ce type d'erreurs et de réaliser un traitement approprié.
- Les processeurs implémentant plusieurs modes de fonctionnement avec différents niveaux de privilèges sont capables de détecter des violations de privilèges. Ces violations résultent d'accès à des ressources non autorisées, tel l'usage d'instructions nécessitant un niveau de privilèges supérieur ou des accès à des zones mémoires protégées.

Les **exceptions matérielles** sont dues généralement à des erreurs du logiciel, mais détectées par des unités externes à l'unité centrale de traitement du processeur (CPU). Les exceptions matérielles les plus courantes sont des erreurs d'accès sur le bus de données (*Bus Error*) ou sur le bus d'adresses (*Address Error*) ainsi que le "reset" du processeur.

— Des contrôleurs de périphériques ne répondant pas aux cycles d'accès dans les temps spécifiés soit par configuration, soit par le matériel sont en principe à l'origine des erreurs sur le bus de données et détectées par l'unité d'interface du bus processeur (BIU - *Bus Interface Unit*). La cause de ces erreurs est généralement la non-activation de l'horloge cadençant les contrôleurs.
— Deux causes principales sont à l'origine des erreurs sur le bus d'adresses, les accès non alignés et les accès non autorisés. La levée d'exceptions pour des accès non alignées dépend de l'implémentation de la BIU et de sa capacité à effectuer des accès multiples pour lire ou écrire des données non alignées. Les MMU/MPU (*Memory Management Unit / Memory Protection Unit*) fournissent les outils indispensables pour la protection des zones mémoires et l'isolation des processus. Elles sont conçues en autre pour détecter des violations d'accès à la mémoire, telle que des accès en lecture ou écriture non autorisés, l'exécution de code non autorisé, la protection de zones mémoire protégées pour un processus.

Reset du processeur

De multiples causes peuvent être à l'origine du reset du processeur, de sa réinitialisation. La première est bien naturellement la mise sous tension de la carte processeur. Cette mise sous tension est cruciale. Une fois stabilisé, le signal "`reset`" est levé. Détecté par le processeur et l'ensemble de ses unités et contrôleurs, il sert à leur réinitialisation. Les chiens de garde (*Watchdog*) sont la deuxième cause importante. Ils servent à surveiller des interblocages logiciels (*Deadlock*). En cas de blocage, les chiens de garde activent le signal "`reset`" et forcent la réinitialisation du système.

6.1.2 Séquence d'interruption

Lors de la levée d'une interruption suite à un événement interne ou externe, le processeur suspend l'exécution du programme en cours et effectue un traitement approprié à l'événement. Ce traitement se déroule en quatre étapes principales (figure 6.5).

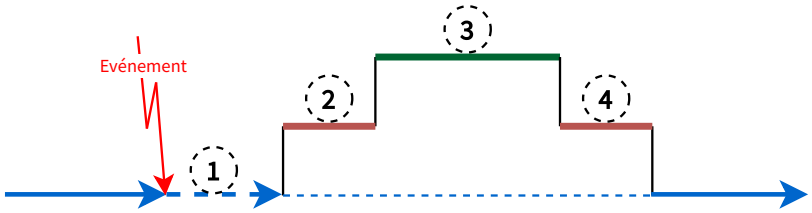

FIGURE 6.5 – Séquence d'interruption

1. Attente d'autorisation pour le traitement des interruptions
2. Sauvegarde du contexte du programme en cours

3. Exécution de la routine de traitement
4. Restauration du contexte et retour au programme suspendu

La première étape n'est présente que lors de requêtes d'interruptions levées par des événements asynchrones. Lors de tels événements, le CPU ne peut effectuer le traitement que si le traitement des interruptions est autorisé. La première cause de désactivation est le traitement d'une interruption. Celui-ci bloque en effet le traitement d'autres interruptions tant que le traitement de l'interruption active n'est pas terminé. La désactivation des interruptions est également un mécanisme indispensable à la conception d'applications logicielles de bas niveau, proche du CPU et de ses périphériques. Il permet en effet de protéger certaines sections critiques et éviter ainsi des conditions de concurrence (*Race Conditions*) entre différentes routines de traitement. Un des exemples typiques sont des données partagées et accédées par une routine de traitement d'interruption (ISR) et des routines exécutées par une tâche (*Thread*). Lors d'interruptions synchrones, cette étape n'existe pas, car la cause de l'événement est l'exécution de l'instruction. Dans de tels cas, le traitement de l'événement s'effectue immédiatement.

La deuxième étape commence aussitôt le traitement des interruptions autorisé. Le CPU suspend immédiatement le programme en cours, effectue une commutation de contexte avant de pouvoir appeler la routine de traitement d'interruption. Dans une première phase, le CPU sauve l'état courant du programme, l'adresse de retour vers le programme en cours d'exécution et les registres du processeur. Puis, il bloque les interruptions et change son mode d'opération pour un mode privilégié adapté au traitement de l'événement. Avant de chercher dans la table des vecteurs d'interruptions la routine de traitement à exécuter. Selon l'architecture du processeur, ces phases sont effectuées complètement par le CPU ou réparties entre CPU et logiciel.

En troisième étape, l'identité de la source de l'événement déterminée, son traitement peut s'effectuer.

En quatrième étape, une fois l'exécution de la routine de traitement terminée, il ne reste qu'à restaurer le contenu des registres du CPU pour rétablir le contexte du programme suspendu afin qu'il puisse poursuivre son exécution où il a été interrompu. Selon l'architecture du processeur, le CPU effectue directement toutes les opérations nécessaires lors du retour de la routine de traitement. Avec d'autres architectures, cette tâche est déléguée au logiciel.

6.1.3 Table des vecteurs d'interruptions

La table des vecteurs d'interruptions sert d'interface entre le HW et le SW (figure 6.6). Le logiciel l'utilise pour configurer le CPU afin qu'il puisse appeler la routine d'interruption appropriée à l'événement à traiter. Cette table est indispensable pour adapter le µP aux besoins spécifiques du système. Selon

les architectures, l'emplacement de cette table peut être fixe ou laissé au libre choix du logiciel et se trouver aussi bien en mémoire vive que morte.

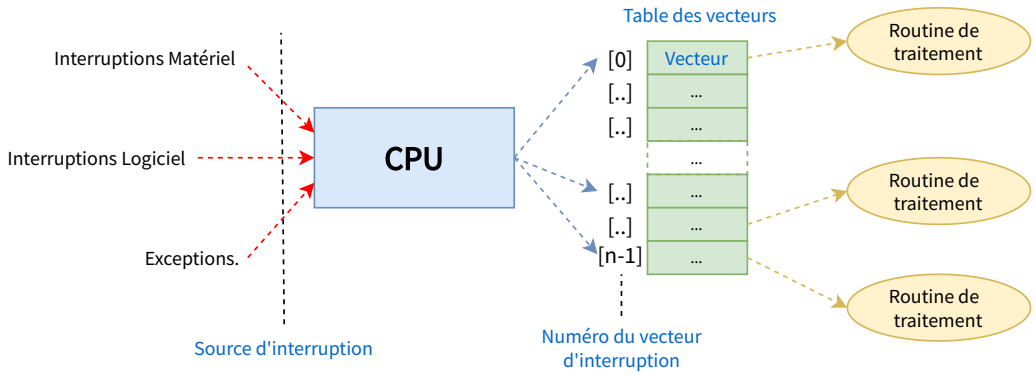

FIGURE 6.6 – Traitement des interruptions au niveau du µP

Lors de la levée d'une interruption, le CPU identifie la source d'interruption à laquelle un numéro fixe de vecteur d'interruption lui est associé. Avec ce numéro, le CPU accède à la table des vecteurs d'interruption afin d'obtenir le vecteur d'interruption et appeler la routine de traitement. Ce vecteur correspond, selon l'architecture du µP, soit à une instruction, soit à l'adresse de la routine de traitement.

Quelques définitions utiles

- La **source d'interruption** (*Interrupt Source*) est le signal ou l'événement capable de lever une interruption ou de générer une exception
- La **table des vecteurs d'interruptions** (*Interrupt Vector Table*) contient les instructions ou les adresses des routines d'interruptions permettant de traiter les événements
- Le **vecteur d'interruption** (*Interrupt Vector*), contenu de la table des vecteurs, sur les processeurs ARM il correspond soit à une instruction, soit à l'adresse de la routine de traitement
- Le **numéro du vecteur d'interruption** (*Interrupt Vector Number*) est l'index dans la table des vecteurs d'interruptions

6.1.4 Commutation de contexte

L'action de sauver les registres du µP pour appeler la routine de traitement lorsqu'une interruption est levée, ou de les restaurer à la fin du traitement de l'interruption, s'appelle la **commutation de**

contexte (*Context Switching*). Ces deux commutations de contexte s'effectuent lors de la deuxième et quatrième étape de la séquence de traitement (figure 6.7).

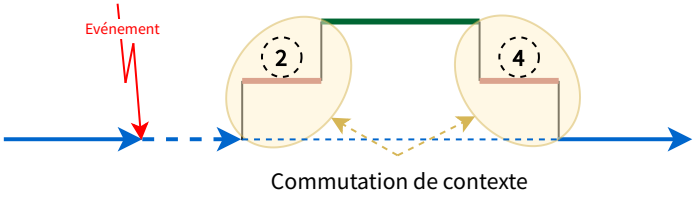

FIGURE 6.7 – Séquence de traitement

Le temps qui s'écoule entre la levée de l'interruption et l'exécution de la première instruction de la routine de traitement d'interruption est appelé **latence d'interruption** (*Interrupt Latency*) (figure 6.8). Elle est due à un facteur principal, le temps de désactivation des interruptions, temps durant lequel le µP n'est pas autorisé à traiter des événements externes. Dans un système fréquemment interrompu, la valeur de cette latence influence considérablement le comportement du système et peut dégrader fortement ses performances. Afin de réduire la latence, les routines de traitement ne doivent effectuer que le strict minimum d'opérations afin de satisfaire le µP et ses contrôleurs et déléguer le maximum du traitement dans des tâches (*Thread*).

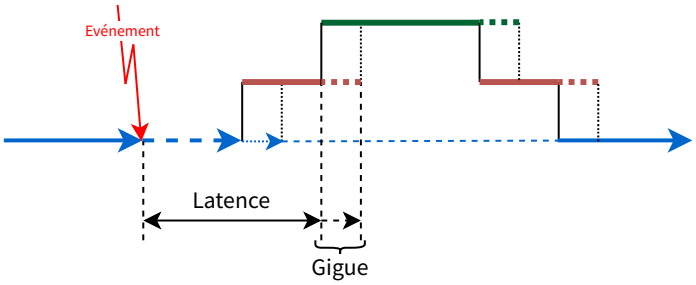

FIGURE 6.8 – Latence et gigue d'interruption

La variation de la latence d'interruption est appelée **gigue d'interruption** (*Interrupt Jitter*). Son amplitude est principalement due aux variations de la durée de désactivation des interruptions. La gigue peut poser des problèmes sérieux dans des systèmes temps réel ayant des contraintes de temps exigeantes.

6.1.5 Interruptions imbriquées

Les systèmes embarqués et les systèmes sur puce (Soc) sont généralement confrontés à devoir piloter simultanément un grand nombre de périphériques différents et gérer une multitude de sources d'interruptions. Si les systèmes d'exploitation riche ne permettent en principe qu'un traitement séquen-

tiel des interruptions, les OS temps réel (RTOS) proposent généralement un support logiciel pour un traitement imbriqué des interruptions (*Nested Interrupt Handling*) pour les processeurs disposant de cette capacité de traitement.

Dans un système de traitement d'interruptions imbriquées, chaque source d'interruptions reçoit un niveau de priorité. Ce niveau de priorité est généralement configurable et peut être unique ou partagé entre plusieurs sources d'interruptions. Lors du traitement d'une interruption (1), si une source avec un niveau de priorité supérieure lève une interruption, le CPU suspend la routine en cours de traitement pour traiter la source plus prioritaire (2) (figure 6.9).

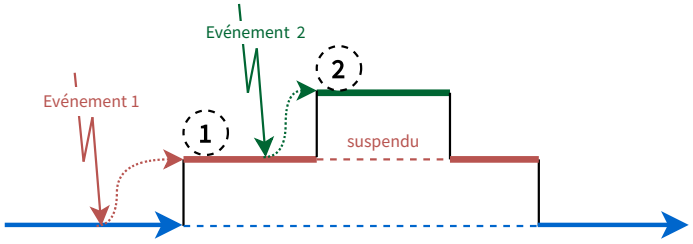

FIGURE 6.9 – Traitement imbriqué pour une source prioritaire

Par contre lors du traitement d'une interruption, si une source avec un niveau de priorité inférieure lève une interruption, le CPU termine d'abord le traitement de la routine en cours (1) avant de traiter la nouvelle source (2) (figure 6.10).

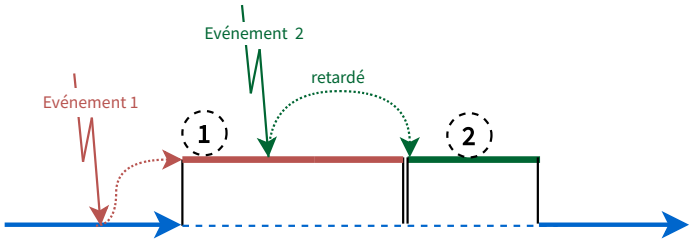

FIGURE 6.10 – Traitement retardé pour une source secondaire

Ce mécanisme est très intéressant, car il permet de réduire le temps de latence pour les sources d'interruptions prioritaires. Par contre, il peut rallonger le temps de traitement des sources secondaires.

Lors de la conception des routines de traitement d'interruptions, il est important d'effectuer le minimum d'opérations, ceci afin de garantir de bon temps de réaction du système. Si certains événements demandent de longs traitements, il est plus judicieux de les déléguer à des tâches d'arrière-plan (*Background Task*).

6.1.6 Gestion de la levée des interruptions

La conception d'applications mettant en œuvre des tâches de fond (*Background Tasks*) coopérant avec des tâches événementielles (*Event-Driven Tasks*) requiert un soin tout particulier. Dans de tels programmes, il est courant d'être confronté à des situations de concurrence lors d'accès à des ressources partagées entre les différentes tâches (*Race Condition*).

Pour protéger de telles ressources, il est souvent nécessaire d'interdire la levée des interruptions le temps du traitement. Dans une telle situation, il est impératif de garantir à une tâche de fond qu'elle ne soit pas interrompue par une tâche événementielle concurrente, une interruption, lorsqu'elle accède aux données. La portion de code impliquée dans ce traitement est nommée en programmation concurrente section critique (*Critical Sections*).

Selon la nature de la ressource à protéger, les interruptions sont bloquées à des niveaux différents. Si la ressource est globale, tel le compteur d'un sémaphore sur un µP mono-coeur, la protection s'effectue généralement au niveau du CPU en inhibant toutes les interruptions. Par contre, s'il s'agit d'une ressource proche d'un périphérique, il est usuel de masquer les interruptions au niveau du contrôleur du périphérique. Le CPU ainsi que les contrôleurs placés dans la chaîne de traitement des interruptions disposent de registres spéciaux permettant au logiciel de gérer la levée des interruptions asynchrones.

6.2 Interruptions matérielles

Tout changement d'état d'un périphérique se reflète dans ses registres d'état (STAT - *Status Register*). Pour détecter ces changements, le logiciel peut scruter ces registres via le bus µP. Cette technique fonctionne bien pour des changements périodiques ou excessivement rapides, mais requiert beaucoup de temps CPU s'ils sont sporadiques. Pour soulager le CPU et éviter ainsi la scrutation, une ligne de requêtes d'interruptions (IRQ-Line - *Interrupt Request Line*) permet au périphérique de signaler tous ses changements au CPU (figure 6.11).

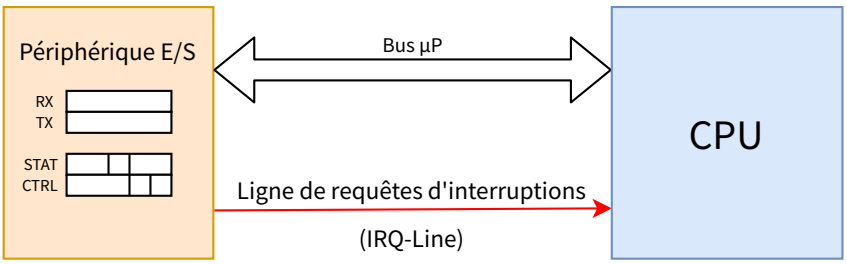

FIGURE 6.11 – Ligne de requêtes d'interruptions

En activant la IRQ-Line, le CPU suspend l'exécution du programme en cours pour exécuter une routine de traitement des événements du périphérique (*Interrupt Service Routine* ou *Interrupt Handler*).

Ce mécanisme fonctionne parfaitement pour un périphérique, mais qu'en est-il pour plusieurs périphériques ? La solution se trouve dans les réponses à ces trois questions :

— Comment connecter les périphériques au CPU ?
— Comment identifier le périphérique ayant levé une interruption ?
— Comment arbitrer des requêtes simultanées provenant de plusieurs périphériques ?

Trois techniques permettent de répondre à ces questions :

— Scrutation logicielle
— Priorité d'interruption
— Interruption vectorisée

6.2.1 Scrutation logicielle

La première solution consiste à connecter tous les périphériques en parallèle sur une même ligne de requêtes d'interruptions (figure 6.12). Cette ligne forme ainsi un "OU" câblé.

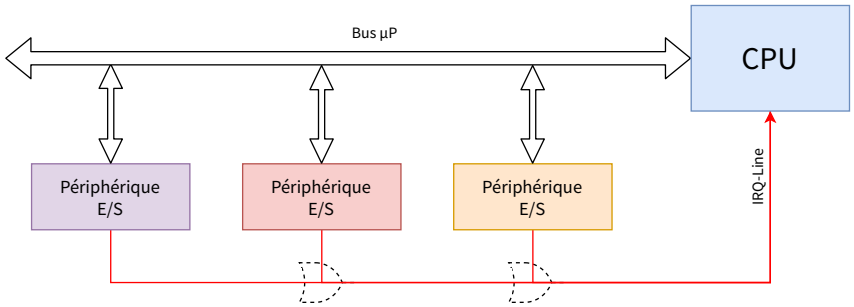

FIGURE 6.12 – Connexion de périphériques pour la scrutation logicielle

La gestion des périphériques s'effectue grâce à leurs registres de contrôle et d'état. Ils permettent de bloquer et d'autoriser la levée d'interruptions ainsi que d'identifier le périphérique ayant levé l'interruption.

Si une interruption apparaît, le CPU suspend le programme en cours et exécute la routine de traitement d'interruptions. Cette routine interroge successivement tous les périphériques connectés sur la ligne d'interruption afin de déterminer le ou les périphériques ayant activé le signal. Cette technique se nomme scrutation logicielle (*Interrupt Polling*), dont voici deux implémentations principales :

— Priorité fixe
les périphériques sont scrutés dans un ordre précis et déterminé à l'avance (*Fixed Priority*). Le 1er périphérique rencontré ayant levé une interruption est servi, puis le 2e et ainsi de suite.

— Tourniquet
les périphériques sont ordonnés comme dans la première méthode, mais la recherche débute depuis le dernier périphérique servi (*Round-Robin*). La recherche recommence au début, lorsque la fin de la liste est atteinte.

La scrutation logicielle est simple à mettre en œuvre, mais peut prendre énormément de temps selon le nombre de périphériques connectés au CPU.

Cette technique est par contre couramment utilisée par des périphériques capables de lever de différentes interruptions pour signaler divers types d'événements, telles la réception et la transmission de données. Dans l'exemple ci-dessous (figure 6.13), le périphérique lève une interruption si la condition suivante est vraie :

$$(\text{TIE} \wedge \text{TXE}) \vee (\text{RIE} \wedge \text{RXE})$$

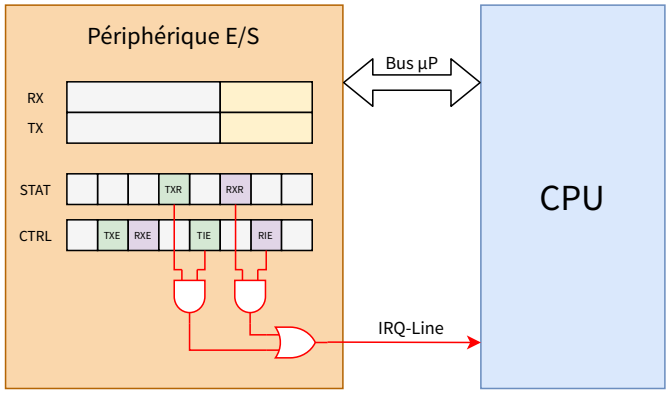

FIGURE 6.13 – Logique d'interruption d'un périphérique pour la scrutation logicielle

La routine de traitement du pilote logiciel (*io_handler*) distingue entre une interruption due à l'émission (TX) et/ou à la réception (RX) en testant les fanions des registres de contrôle et d'état.

```
1 void io_handler() {
2     if ((io->ctrl & CTRL_TIE) && (io->stat & STAT_TXR))
3         // interruption levée en émission
4
5     if ((io->ctrl & CTRL_RIE) && (io->stat & STAT_RXR))
6         // interruption levée en réception
7 }
```

6.2.2 Priorité d'interruption

Avec la deuxième solution, chaque périphérique dispose d'une ligne de requêtes d'interruptions dédiée connectée directement au CPU (figure 6.14).

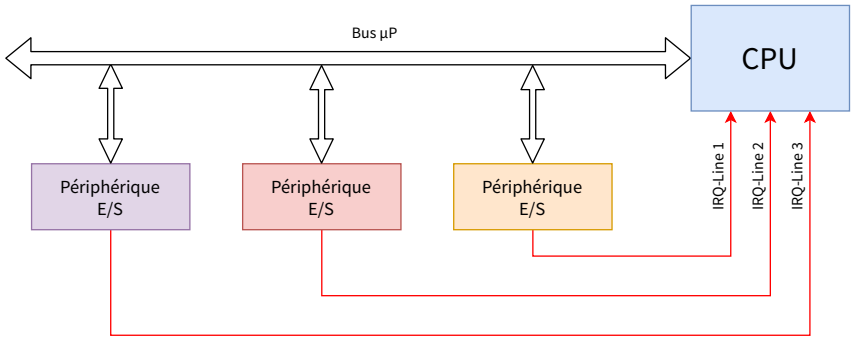

FIGURE 6.14 – Connexion de périphériques pour la priorité d'interruption

Le CPU affecte à chacune de ces lignes de requêtes d'interruptions une priorité matérielle fixe et unique, par exemple aux lignes IRQ-Line1, IRQ-Line2 et IRQ-Line3, le CPU accorde les priorités 1, 2 et 3. Lorsqu'une ou plusieurs requêtes d'interruptions sont levées, le CPU examine les lignes de requêtes d'interruptions et détermine la ligne active la plus prioritaire afin de servir le périphérique correspondant.

Pour chaque priorité d'interruptions (*Interrupt Priority*), le CPU dispose d'une entrée dans sa table des vecteurs d'interruptions. Ceci permet aux pilotes logiciels d'attacher une routine de traitement propre à chaque périphérique.

Cette technique améliore grandement le traitement des interruptions, car elle élimine la scrutation logicielle des périphériques. Elle permet aussi un traitement imbriqué des interruptions et réduit la latence pour les événements les plus importants et les plus prioritaires. Cependant, elle se butte au nombre de lignes de requêtes d'interruptions que les CPU disposent, souvent qu'une ou deux lignes.

6.2.3 Interruption vectorisée

La troisième solution consiste à générer des interruptions vectorisées (*Vectored Interrupt*). Celle-ci se caractérise par la mise en œuvre d'un contrôleur d'interruptions permettant de connecter une multitude de périphériques, chacun avec une ou plusieurs lignes de requêtes d'interruptions dédiées (figure 6.15). Il associe à chaque ligne un numéro de vecteur d'interruptions unique (*Unique Interrupt Vector Number*).

Lors de la levée d'une interruption, le contrôleur dispose d'une logique interne pour prioriser et identifier la source d'interruption. Une fois la source identifiée, le contrôleur informe le CPU. Là, il existe deux réalisations possibles d'interface entre le contrôleur et le CPU.

Avec la première interface, le contrôleur d'interruption informe le CPU en activant une ligne de re-

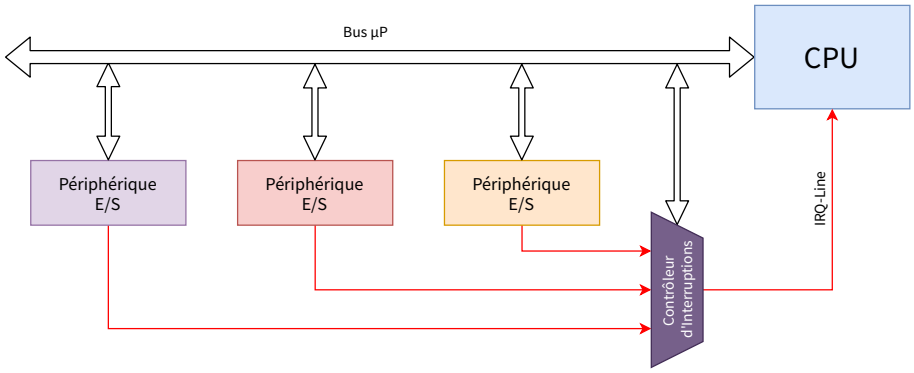

FIGURE 6.15 – Connexion de périphériques pour des interruptions vectorisées

quêtes d'interruptions (IRQ-Line), pour laquelle, il n'existe qu'une seule entrée dans la table des vecteurs d'interruptions. Lorsque l'interruption est levée, la routine de traitement attachée dans la table des vecteurs acquiert l'identité de la source d'interruption, le numéro de vecteur, en lisant un registre du contrôleur d'interruptions. La source connue, il appelle ensuite la routine de traitement correspondant au périphérique. Avec ce type d'interface, le logiciel effectue l'essentiel de l'identification de la source.

Avec la deuxième interface, le contrôleur d'interruption informe toujours le CPU en activant une ligne de requêtes d'interruptions (IRQ-Line). Par contre, lorsque l'interruption est levée, c'est le CPU qui acquiert l'identité de la source d'interruption, le numéro de vecteur. Il utilise ensuite ce numéro de vecteur pour appeler la routine de traitement correspondant au périphérique, laquelle a été précédemment attachée directement dans la table des vecteurs. Avec cette interface, l'essentiel de l'identification de la source d'interruption étant réalisée par le contrôleur et le CPU, le logiciel n'a plus qu'à effectuer le traitement approprié à l'événement.

Les interruptions vectorisées sont spécialement bien adaptées à un traitement imbriqué des interruptions.

6.3 Profil A

Les processeurs du profil A implémentent une gestion plutôt basique du traitement des interruptions au niveau du CPU. Le CPU effectue seulement les opérations impératives, sauvegarde de l'adresse de retour et du registre d'état ainsi que l'appel de la routine de traitement de bas niveau. Il laisse la plus grande partie des opérations à effectuer pour le traitement des interruptions et exceptions au logiciel. Ce choix offre une plus grande liberté pour la réalisation des logiciels et plus particulièrement des systèmes d'exploitation riches.

Les exemples de ce chapitre se basent sur le processeur ARM Cortex-A8 de l'architecture ARMv7-A et plus spécialement le µP de TI AM3358.

6.3.1 Sources d'interruptions

Les µP du profil A [1] connaissent 7 sources d'interruptions (figure 6.16) :

- Les exceptions (Reset, Undefined Instructions, Data Abort, Prefetch Abort)
- Les interruptions matérielles (IRQ - *Interrupt Request* et FIQ - *Fast Interrupt Request*)
- Les interruptions logicielles ou systèmes (SVC - *Supervisor Call*, SMC - *Secure Monitor Call*, etc.)

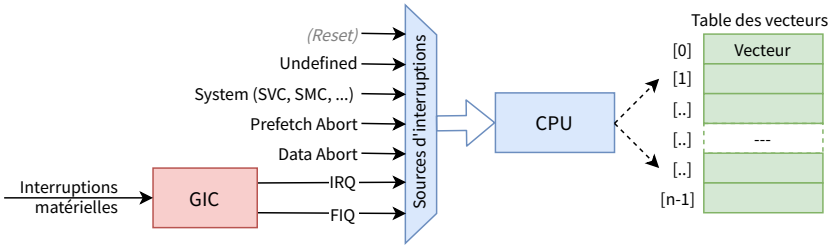

FIGURE 6.16 – Profil A - Sources d'interruptions

De ces 7 sources, l'exception "Reset" est traitée généralement par un micro-code brûlé dans une ROM du processeur, rendant impossible l'exécution de traitements spécifiques lors de la levée de cette exception. Ce n'est que lors du lancement de l'application logicielle que ceux-ci peuvent être exécutés. Ne disposant que de deux lignes de requêtes d'interruptions matérielles, le CPU utilise un contrôleur d'interruptions (GIC - *Generic Interrupt Controller*) pour gérer les requêtes des périphériques internes et externes et les relayer vers le CPU. Le µP TI AM3358 [2], un µP ARM Cortex-A8 [3], implémente quant à lui un "INTC" comme contrôleur d'interruptions.

6.3.2 Table des vecteurs d'interruptions

L'implémentation de la table des vecteurs est assez simple si l'on utilise le langage assembleur. Il suffit de définir une table avec l'instruction pour appeler les routines de traitement de bas niveau (les *handlers*). On prendra soin à n'utiliser que l'instruction "`b`" et non pas "`bl`". Il est en effet essentiel de ne pas "écraser" le contenu du registre "`LR`" contenant l'adresse de retour vers le programme suspendu.

```
1     .text
```

1. ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition
2. AM335x Sitara(TM) Processors Technical Reference Manual
3. Cortex-A8 Technical Reference Manual

```
2       .align 5
3   vector_table:
4   1:  b   1b      // reset
5       b   undef_handler
6       b   svc_handler
7       b   prefetch_handler
8       b   data_handler
9   1:  b   1b      // reserved
10      b   irq_handler
11      b   fiq_handler
```

La source "`reset`" étant traité par le µP et la source "`reserved`" n'étant pas jamais levée, il n'est pas nécessaire d'appeler une routine de traitement. Cependant, pour des raisons de fiabilité et de robustesse du code, il faut éviter d'exécuter une autre routine si le µP appelle quand même l'une de ces sources. Pour cela l'instruction "`1: b 1b`", permet de forcer le CPU à boucler sur elle-même.

Les µP ARM Cortex-A8 permettent de placer la table des vecteurs d'interruptions librement en mémoire. La seule contrainte est que celle-ci doit être alignée sur 32 bytes (2^5). Le co-processeur P15 dispose d'un registre pour spécifier l'adresse de base de cette table. Les deux instructions ci-dessous permettent de charger le registre avec cette adresse.

```
1  ldr   r0, =vector_table
2  mcr   p15, #0, r0, c12, c0, #0
```

Il est impératif d'effectuer ce placement avant d'autoriser la levée d'interruptions.

6.3.3 Traitement de l'interruption par le CPU

Durant la deuxième étape du traitement des interruptions par le µP, le CPU sauve un état minimal afin de pouvoir appeler une routine de traitement des interruptions de bas niveau pour permettre le retour au programme en cours avant la levée de l'interruption.

Sauvegarde de l'état minimal du CPU

Lorsqu'une interruption est levée et que son traitement est autorisé, le CPU effectue les opérations suivantes :

- Suspend l'exécution du programme en cours
- Détermine la source d'interruption
- Fixe le nouveau mode du processeur
- Sauve le compteur ordinal (PC), l'adresse de retour et le registre d'état (CPSR)
- Désactive la levée des interruptions matérielles

Pour des raisons de performances, le CPU sauve son état minimal dans des registres supplémentaires (*Banked Registers*). Ces registres sont déterminés par le nouveau mode du CPU (figure 6.17). L'adresse de retour est sauvée dans le registre LR_<mode> et le CPSR dans le registre SPSR_<mode>.

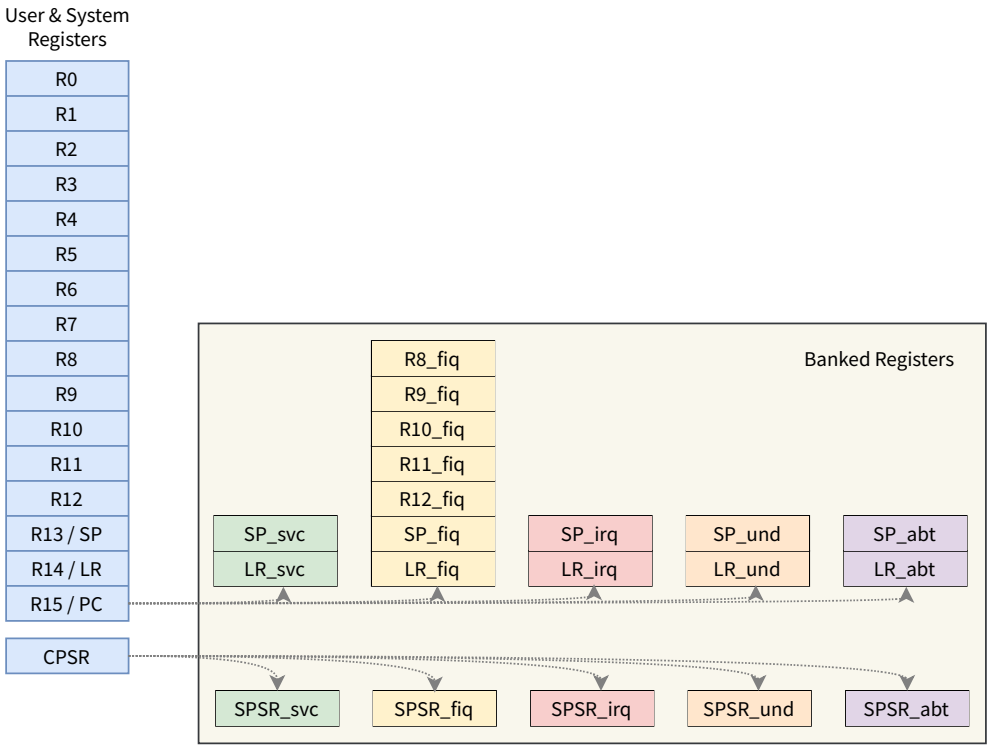

FIGURE 6.17 – Profil A - Sauvegarde de l'état minimal du CPU

Le nouveau mode du CPU se reflète dans le champ de bits M[4 :0] du registre CPSR (figure 6.18). La désactivation des interruptions matérielles, également reflétées dans le registre CPSR, dépend de la source d'interruption. La levée d'interruptions IRQ est toujours désactivée (bit I = 1), ceci indépendamment de la source d'interruption (interruption matérielle, interruption logicielle ou exception). Par contre, les interruptions FIQ ne sont désactivées (bit F = 1) que lors de la levée d'une interruption matérielle FIQ.

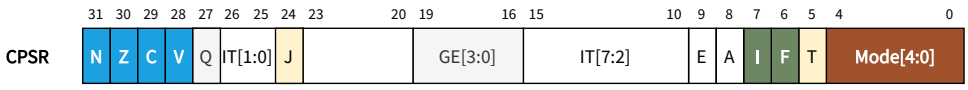

FIGURE 6.18 – Profil A - Registre d'état

Appel de la routine de traitement de bas niveau

La sauvegarde de l'état minimal du CPU terminée, le CPU appelle la routine de traitement correspondant à la source d'interruption en exécutant l'instruction contenue dans la table des vecteurs. Cette opération s'effectue en chargeant le registre "PC" avec l'adresse de la table des vecteurs corrigée avec l'offset correspondant au numéro du vecteur d'interruption.

```
1 PC = &vector_table[vector_number];
```

Sortie de la routine de traitement d'interruption

Pour sortir de la routine de traitement et poursuivre l'exécution du programme, le CPU restaure le contenu du registre "CPSR" et charge l'adresse de retour contenue dans le registre "LR" dans le registre "PC". L'instruction ci-dessous en principe suffit :

```
1 movs    pc, lr
```

Source	Offset	Instruction de retour
Reset	n/a	n/a
Undefined	0	movs pc, lr
SVC	0	movs pc, lr
Prefetch Abort	4	subs pc, lr, #4
Data Abort	4	subs pc, lr, #4
IRQ	4	subs pc, lr, #4
FIQ	4	subs pc, lr, #4

FIGURE 6.19 – Profil A - Correction de l'adresse de retour

Cependant, selon la source d'interruption, l'adresse contenue dans le registre "LR" doit être préalablement corrigée (figure 6.19). Cette offset est une relique des premières implémentations des µP ARM. La troisième colonne de la table ci-dessus propose une instruction de retour avec cet ajustement.

6.3.4 Traitement de l'interruption par le logiciel

Durant la deuxième phase du traitement par le logiciel, la routine de traitement sauve le contexte du programme interrompu, les registres pas sauvés par le CPU, avant d'appeler la routine de traitement appropriée à la source d'interruption. Le traitement terminé, elle restaure le contexte et retourne au programme suspendu.

Le peusdo-code ci-dessous indique les opérations à effectuer.

```
1 sub    lr, #OFFSET          // only if necessary
2 stmfd  sp!, {r0-r3,r12,lr}  // save the context
3 mov    r0, #VECTOR_NR       // indicate vector number (source)
4 bl     interrupt_handler    // process interrupt
5 ldmfd  sp!, {r0-r3,r12,pc}^ // restore the context
```

Une technique courante avant la sauvegarde du contexte du programme en cours consiste à corriger l'adresse de retour contenue dans le registre "LR" (1re instruction) à l'entrée de la routine d'interruption et de la sauver avec les autres registres (2e instruction). La valeur de la constante "OFFSET" est spécifique à la source d'interruption.

Sur ce type de µP, processeur d'application, l'appel des routines de traitement des interruptions s'effectue généralement par "callback" en réalisant le concept d'écouteur (*Listener*). Alors pour des raisons de confort de réalisation, juste les instructions n'existant pas en langage de programmation évolué sont effectuées en assembleur. Le reste du traitement est délégué à la fonction générique "interrupt_handler" (4e instruction). Cette routine sert d'intermédiaire à l'appel de la routine de traitement spécifique à la source de l'interruption au niveau applicatif. Elle reçoit en paramètre l'indication sur la source d'interruption, le numéro de vecteur "VECTOR_NR" (3e instruction).

La restauration du contexte du CPU à la sortie de la routine de traitement d'interruption s'effectue en une seule opération (5e instruction). L'utilisation du "^" avec le registre "PC" indique au CPU qu'il doit restaurer le "CPSR" depuis le "SPSR".

Utilisation d'une macro

L'utilisation d'une macro permet d'adapter simplement ce code aux différentes sources d'interruptions.

```
1  .macro ll_interrupt_handler offset, vector_nr
2     nop
3     .if \offset != 0             // adjust return address
4       sub  lr, #\offset          // only if necessary
5     .endif
6     stmfd  sp!, {r0-r3,r12,lr}   // save the context
7     mov    r0, #\vector_nr       // indicate vector number (source)
8     bl     interrupt_handler     // process interrupt
9     ldmfd  sp!, {r0-r3,r12,pc}^  // restore the context
10 .endm
```

Avec cette macro, les routines de traitement de bas niveau appelées depuis la table des vecteurs se définissent comme suit :

```
1     .text
```

```
2     .align 2
3  undef_handler:    ll_interrupt_handler 0, INT_UNDEF
4  svc_handler:      ll_interrupt_handler 0, INT_SWI
5  prefetch_handler: ll_interrupt_handler 4, INT_PREFETCH
6  data_handler:     ll_interrupt_handler 4, INT_DATA
7  irq_handler:      ll_interrupt_handler 4, INT_IRQ
8  fiq_handler:      ll_interrupt_handler 4, INT_FIQ
```

La valeur du numéro de vecteur passé en 2e paramètre doit bien évidemment correspondre celle utilisée par la fonction "`interrupt_handler`" réalisée en langage évolué.

Pointeurs de piles

Chaque mode du processeur possède sa propre pile (figure 6.18). Cette pile, accessible par le registre "`SP`" du mode correspondant doit être initialisée impérativement avant la levée d'interruption ou d'une exception. Cette opération passe par la création d'une pile en réservant une zone mémoire dans la mémoire vive et ensuite par l'assignation du registre "`SP`" avec l'adresse la plus haute de la pile.

Le code ci-dessous donne un exemple d'implémentation pour l'un des modes du processeur, le mode IRQ.

```
1     .bss
2     .align 4
3  irq_s: .space 0x2000            // reserve stack of the irq mode
4  IRQ_STACK_TOP:                  // points to the top of the stack
5
6     .text
7     .align 2
8     msr   cpsr_c, #0xd2          // switch to irq mode
9     ldr   sp, =IRQ_STACK_TOP     // init SP with top stack address
```

Dans l'exemple ci-dessus, la réservation de la pile est statique. Il est bien évidemment possible de la réserver dynamiquement.

6.3.5 Principe pour le traitement des interruptions matérielles

La signalisation d'événements par des périphériques d'entrées/sorties passe par plusieurs multiplexeurs avant d'arriver sur l'une des lignes d'interruptions du CPU. Ces multiplexeurs permettent de connecter une grande quantité de périphériques au CPU. Ils sont également en charge d'arbitrer, de prioriser et d'identifier efficacement les périphériques ayant levé une interruption. Comme présenté sur la figure ci-dessous (figure 6.20), les µP Cortex-A8 implémentent deux multiplexeurs, un contrôleur d'interruption (INTC - *Interrupt Controller*) et une unité de gestion des broches d'entrées/sorties (GPIO - *General Purpose Input Output*).

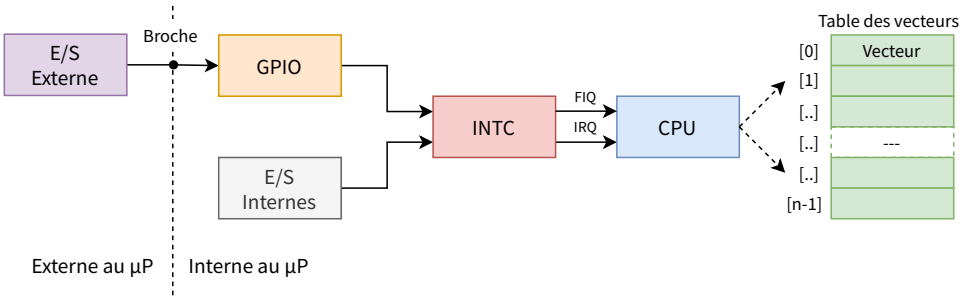

FIGURE 6.20 – Profil A - Connexion des périphériques au CPU

L'INTC est connecté directement au CPU. Sa fonction est de multiplexer toutes les lignes d'interruptions des périphériques internes du µP vers le système d'interruptions du CPU. L'INTC permet d'arbitrer et de prioriser les différentes requêtes. Pour faciliter le traitement des requêtes par le logiciel, il livre un numéro de vecteur d'interruption permettant d'identifier très efficacement la source d'interruption. Ce numéro est unique à chaque périphérique. L'INTC permet de connecter jusqu'à 128 lignes de requêtes d'interruptions.

Le GPIO, hormis sa fonction de contrôleur d'entrées/sorties numériques, permet, par ses broches, de connecter au système d'interruptions du CPU des périphériques externes d'entrées/sorties. Le GPIO n'offre aucun support pour identifier la source d'interruption. Il ne livre dans un registre qu'un set de bits indiquant quelles broches ont levé une interruption. C'est ensuite au logiciel de scruter chaque bit pour identifier la broche à servir et à quittancer après traitement.

Pour trouver la source lors d'une interruption, le CPU appelle juste la routine de traitement d'interruption de bas niveau (*interrupt handler*) contenue dans sa table des vecteurs et correspondant à la source de l'interruption. Il délègue le reste du traitement au logiciel (figure 6.21).

La routine de traitement d'interruption (*intc handler*) traite les interruptions matérielles (IRQ et/ou FIQ) levées par le contrôleur INTC. Pour identifier le périphérique interne du µP ayant levé l'interruption, elle demande au contrôleur INTC de lui livrer le vecteur d'interruption pour appeler la routine de traitement correspondant au périphérique. Si le vecteur correspond à un contrôleur GPIO, la routine de traitement du contrôleur (*gpio handler*) demande alors au contrôleur GPIO de lui livrer la broche à l'origine de l'interruption, pour appeler la routine de traitement au niveau de l'application (*application e/s externe*) et effectuer le traitement souhaité.

6.3.6 Gestion de la levée d'interruptions

La gestion de la levée des interruptions au niveau du CPU par le logiciel passe par la manipulation des fanions "`I`" et "`F`" du registre "`CPSR`". Le fanion "`I`" gère la levée des interruptions provenant de la ligne d'interruptions IRQ, tandis que le fanion "`F`" celle des FIQ.

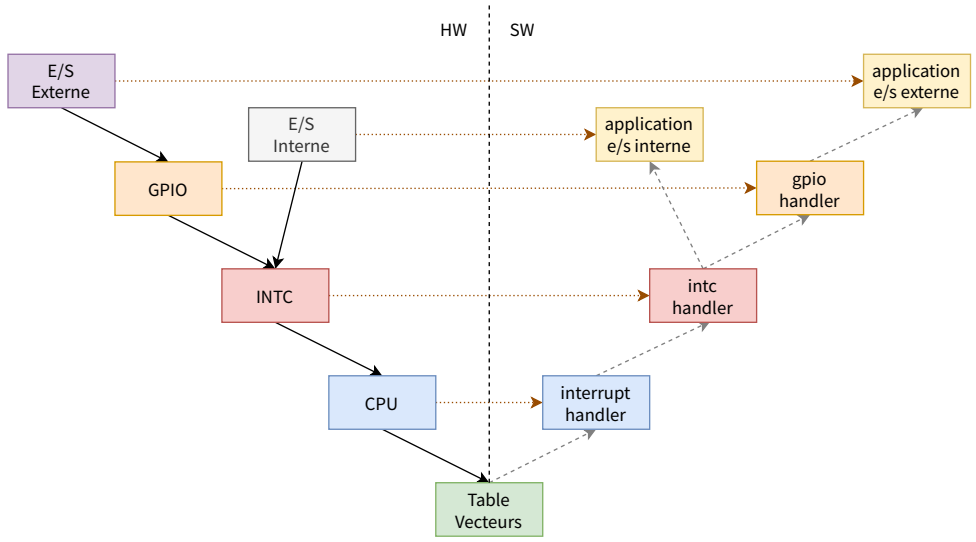

FIGURE 6.21 – Profil A - Pyramide de traitement des interruptions

Pour bloquer la levée d'interruptions, les fanions doivent être mis à 1

```
1 mrs   r0, cpsr
2 orr   r0, #0xe0
3 msr   cpsr, r0
```

par contre, s'ils sont mis à zéro, la levée d'interruptions est autorisée.

```
1 mrs   r0, cpsr
2 bic   r0, #0xe0
3 msr   cpsr, r0
```

Le CPU n'autorise la manipulation de ces fanions que si le programme s'exécute en mode privilégié.

Pour rappel, lorsqu'une interruption matérielle est levée, le CPU change de mode et sauve son état dans les registres "LR" et "SPSR", puis il bloque la levée de futures interruptions en manipulant les fanions "I" et "F"

- Si une IRQ est levée, alors "I" est mis à 1 et "F" reste inchangé
- Si une FIQ est levée, alors "I" et "F" sont mis à 1

6.3.7 Priorité et préemption

La notion de priorité des interruptions et exceptions prend tout son sens lorsque celles-ci surviennent simultanément. Les µP du profil A les classent sur 7 niveaux (figure 6.22). Le niveau 1 est le plus prio-

ritaire et le niveau 7 le moins.

Priorité	#	Description
la plus haute	1	Reset
	2	Data abort
	3	FIQ
	4	IRQ
	5	Imprecise abort
	6	Prefetch abort
la plus basse	7	Undefined instruction / SVC / BKPT

FIGURE 6.22 – Profil A - Priorités des interruptions

Si plusieurs événements sont levés exactement au même instant, le CPU les traite séquentiellement en commençant par le plus prioritaire. Lors de la levée simultanée d'interruptions matérielles par des périphériques, le contrôleur d'interruptions (INTC) se charge de les prioriser et de les relayer au CPU par les lignes de requêtes IRQ et FIQ.

L'infrastructure des µP de ce profil permet une gestion imbriquée des interruptions et exceptions. Cependant, il est peu courant que les systèmes d'exploitation riches en profitent. La préférence est souvent donnée à une gestion minimale dans les routines de traitement pour déléguer l'essentiel de la gestion de l'événement à des tâches de fond (*Thread*) et éviter ainsi un traitement préemptif.

6.3.8 Contrôleur d'interruptions

Le contrôleur d'interruptions INTC multiplexe 128 sources d'interruptions. Chacune de ces sources correspond à une ligne de requêtes d'interruptions d'un périphérique interne au processeur. Il dispose de registres pour prioriser (*Priority*) et bloquer (*Mask*) chaque ligne de requêtes d'interruptions, ainsi que pour simuler (*Software Interrupt*) par logiciel la levée d'une interruption (figure 6.23).

Lorsque des interruptions sont levées, le contrôleur sélectionne, grâce à son unité pour trier les priorités (*Priority Sorter*), la source (*Source*) la plus prioritaire et la sert en premier. Une fois servie, le logiciel quittance le traitement par le registre de contrôle (*Control*), permettant ainsi la levée de la prochaine source. Si aucune priorité n'a été attribuée par logiciel, le contrôleur utilise le numéro de la source.

Le contrôleur INTC dispose également d'une logique pour ne prendre en compte que les sources ayant un niveau de priorité suffisant pour être traitées. Si le niveau de priorité de la source d'interruption est inférieur ou égal à une valeur prédéfinie (*Threshold*), la requête sera bloquée jusqu'à ce que le threshold soit ajusté (figure 6.24).

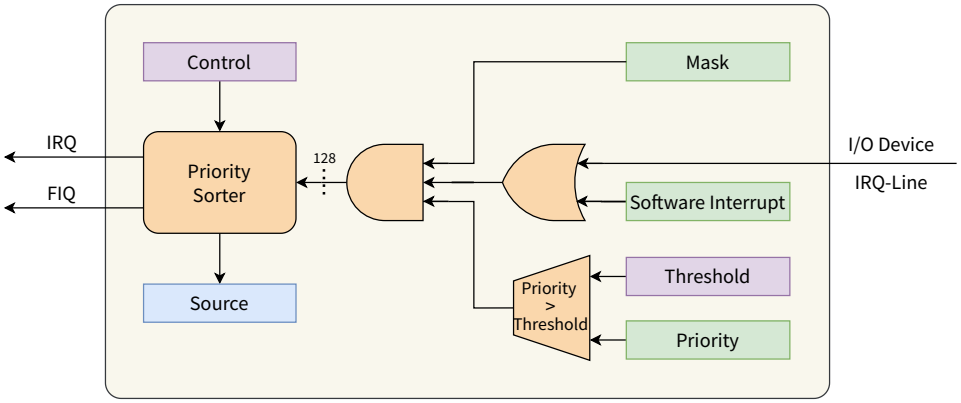

FIGURE 6.23 – Profil A - INTC

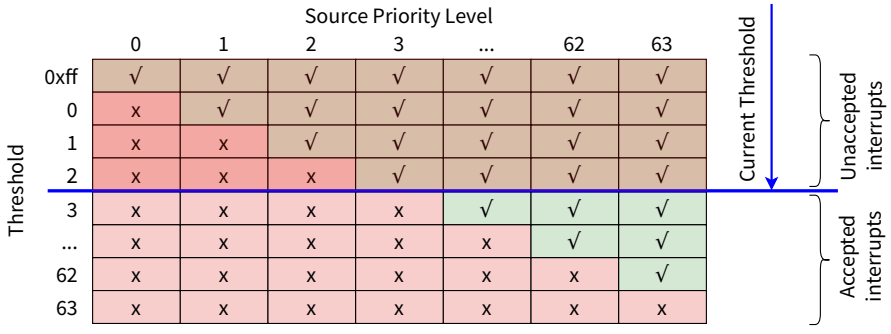

FIGURE 6.24 – Profil A - Niveaux de priorités de l'INTC

Bien que disponible, cette fonctionnalité permettant de réaliser un système de traitement des interruptions imbriquées n'est généralement pas utilisée par les systèmes d'exploitation riches. Ceux-ci préfèrent un traitement rapide en mode interruptif pour déléguer la plus grande portion du traitement dans des tâches de fond.

6.3.9 Unité de gestion des entrées/sorties

L'unité de gestion des entrées/sorties (GPIO - *General Purpose Input Output*) sert à piloter 32 broches (*Pin*) numériques (figure 6.25). Chaque broche peut se configurer aussi bien en entrée qu'en sortie (*Output Enable*).

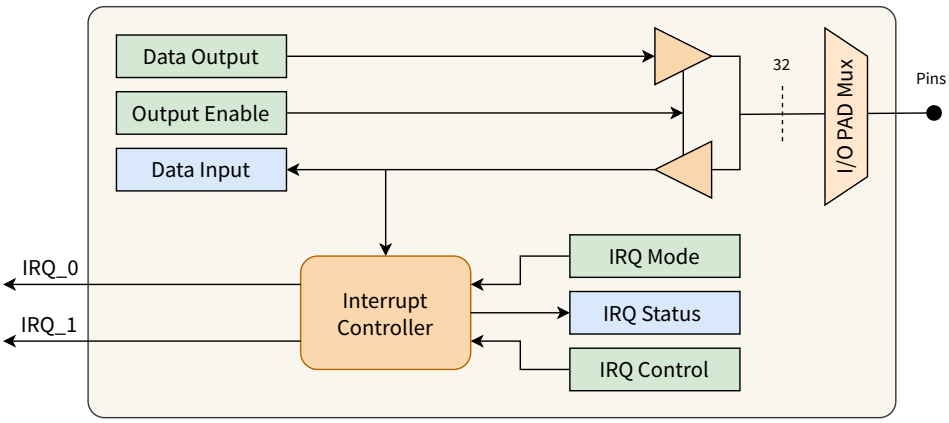

FIGURE 6.25 – Profil A - GPIO

Configurés en entrée, des périphériques externes au µP, tels des boutons-poussoirs, peuvent les utiliser comme ligne de requêtes d'interruptions. Deux lignes de requêtes connectent l'unité GPIO au contrôleur INTC, les lignes IRQ_0 et IRQ_1. Ces deux lignes permettent de prioriser certaines broches par rapport à d'autres.

Chaque broche peut de générer une interruption si elle détecte (figure 6.26)

- Niveau haut du signal (*High Level*)
- Niveau bas du signal (*Low Level*)
- Flanc descendant (*Falling Edge*)
- Flanc montant (*Rising Edge*)

Bien qu'il soit possible de générer une interruption selon le niveau du signal, on préfère généralement la détection sur les changements d'état, sur les flancs montants, descendants ou les deux.

Lors de la levée d'une interruption sur une broche, l'unité GPIO la propage vers le contrôleur INTC. Afin que cette interruption atteigne le CPU, il faut encore autoriser la levée d'interruptions du contrôleur INTC. Hormis la liste des broches ayant levé une interruption, l'unité GPIO ne fournit pas d'autres

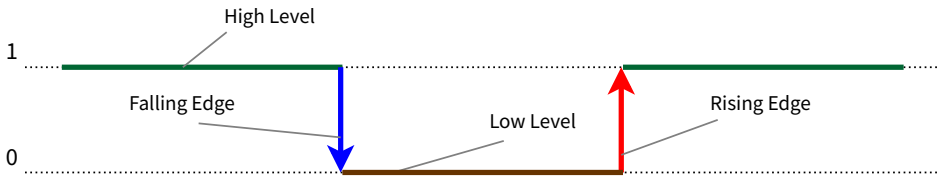

FIGURE 6.26 – Profil A - GPIO Système de détections

indications. C'est au logiciel d'itérer sur les différentes broches pour identifier celles ayant levé une interruption et la servir.

6.4 Profil M

Les processeurs du profil M implémentent une gestion des interruptions au niveau du CPU permettant un traitement logiciel réalisé en langage C/C++. Sur ces µP, il n'est pas nécessaire de réaliser des routines de bas niveau en assembleur. Le CPU se charge de sauvegarder le contexte du programme en cours avant d'appeler la routine de traitement de l'événement et de le restaurer après traitement. Ce choix simplifie énormément la réalisation des applications logicielles.

Les exemples de ce chapitre se basent sur le processeur ARM Cortex-M4F et plus spécialement le µC de ST STM32F412 et utilisent la bibliothèque "`libopencm3`" sous l'environnement "`platformio`".

6.4.1 Sources d'interruptions

Les µC du profil M connaissent 15 sources d'interruptions internes et de multiples sources d'interruptions externes (figure 6.27). Toutes ces sources d'interruptions sont connectées et gérées par le contrôleur d'interruptions (NVIC - *Nested Vectored Interrupt Controller*).

Sur les 15 sources internes, 10 sont actuellement allouées [4] :

— **Reset** : l'exception est levée lorsque le µC est mis sous tension ou lors d'un reset local du µC par logiciel. Elle génère la réinitialisation du CPU et de ses périphériques.
— **NMI** : l'interruption (*Non-Maskable Interrupt*) est l'interruption la plus élevée après le reset et ne peut pas être bloquée. Elle sert généralement à des composants matériels pour signaler des erreurs exigeant un traitement immédiat ne pouvant être retardé.

4. Armv7-M Architecture Reference Manual - chapitre B1.5

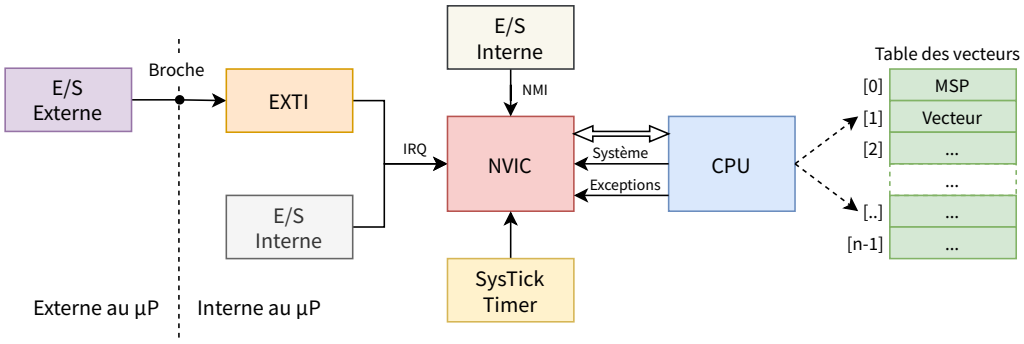

FIGURE 6.27 – Profil M - Système d'Interruptions

— **HardFault** : l'exception signale des fautes génériques ne trouvant aucune autre exception offrant un traitement permettant de récupérer la situation d'erreur. Elle est généralement le résultat d'un enchaînement de fautes, dues à un non-traitement d'autres fautes, telles `MemManage`, `BusFault` ou `UsageFault`.
— **MemManage** : l'exception signale une violation de la protection de la mémoire détectée par la MPU (*Memory Protection Unit*).
— **BusFault** : l'exception signale des erreurs lors de transfert de données ou d'instructions sur les bus système.
— **UsageFault** : l'exception signale des erreurs causées par l'exécution d'instructions, mais non liées à la mémoire, par exemple des instructions non définies, des accès non alignés, des divisions par zéro, des accès à des coprocesseurs non existants ou déclenchés, etc.
— **DebugMonitor** : l'exception signale des événements de debugging.
— **SVCall** : l'exception est levée suite l'appel de l'instruction "`SVC`". Les systèmes d'exploitation l'utilisent pour les appels système (*syscall*).
— **PendSV** : l'exception (*Pendable Service Call*) est levée par logiciel pour des appels système asynchrones.
— **SysTick** :l'interruption est levée par l'horloge interne au µC. Les systèmes d'exploitation l'utilisent pour générer l'horloge système.

Le contrôleur NVIC est capable en principe de gérer jusqu'à 496 sources d'interruptions externes. Cependant le nombre exact de sources dépend de la réalisation spécifique du fabricant du µC.

6.4.2 Table des vecteurs d'interruptions

La conception de la table des vecteurs est très simple à utiliser avec le langage C/C++. Elle contient l'adresse initiale du pointeur de pile "`MSP`" (*Main Stack Pointer*) ainsi que les adresses des routines de traitement des interruptions. Comme le montre le pointeur de fonction "`vector_table_entry_t`",

ces routines sont des fonctions sans paramètre ni valeur de retour.

```
1  #define NVIC_IRQ_COUNT 96
2
3  typedef void (*vector_table_entry_t)(void);
4
5  typedef struct {                                  // vector number
6      unsigned int *initial_sp_value;               // MSP
7      vector_table_entry_t reset;                   //  1
8      vector_table_entry_t nmi;                     //  2
9      vector_table_entry_t hard_fault;              //  3
10     vector_table_entry_t memory_manage_fault;     //  4
11     vector_table_entry_t bus_fault;               //  5
12     vector_table_entry_t usage_fault;             //  6
13     vector_table_entry_t reserved_x001c[4];       // (7-10)
14     vector_table_entry_t sv_call;                 // 11
15     vector_table_entry_t debug_monitor;           // 12
16     vector_table_entry_t reserved_x0034;          //(13)
17     vector_table_entry_t pend_sv;                 // 14
18     vector_table_entry_t systick;                 // 15
19     vector_table_entry_t irq[NVIC_IRQ_COUNT];
20 } vector_table_t;
21
22 __attribute__ ((section(".vectors"))) vector_table_t vector_table={};
```

La table des vecteurs (`vector_table`) peut être placée librement dans la mémoire SRAM ou Flash (*Code*) du processeur à l'aide du registre "`VTOR`" contenu dans le bloc de contrôle du système (SCB - *System Control Block*) [5]. Par défaut et grâce à la section "`.vectors`", l'éditeur de liens (*Linker*) place la table des vecteurs dans les premiers blocs de la Flash à l'offset 0.

Avec la bibliothèque "`libopencm3`", cette table est initialisée dans le module "`vector.c`". Quant aux routines de traitement par défaut, elles sont définies dans le fichier "`nvic.h`" de la famille du µC. Pour attacher une routine spécifique à l'application (ISR - *Interrupt Service Routine*) au système de traitement des interruptions, il suffit de surcharger la routine par défaut en implémentant une routine correspondante. Celle-ci doit respecter le nom par défaut, par exemple pour le "timer 2" la routine doit impérativement être nommée "`void tim2_isr(void)`". Son numéro de vecteur est "`NVIC_TIM2_IRQ`".

Au démarrage, le processeur initialise le registre "`MSP`" avec la valeur de la première entrée de la table des vecteurs (*initial_sp_value*). Cette valeur initiale est générée par l'éditeur de lien et pointe sur le sommet de la SRAM (adresse la plus haute). Le processeur exécute ensuite la routine attachée à la source "`Reset`".

5. STM32 Cortex®-M4 MCUs and MPUs programming manual (PM0214) - chapitre 4.4

6.4.3 Traitement de l'interruption par le CPU

Lors de la levée d'une interruption autorisée, c'est-à-dire une interruption ayant une priorité suffisante, le CPU sauve le contexte du programme en cours d'exécution avant d'appeler la routine de traitement correspondante et de le restaurer une fois l'interruption servie. Cette technique décharge le logiciel des opérations à effectuer au niveau du CPU et lui permet ainsi de s'occuper directement et seulement du traitement de l'événement (figure 6.28).

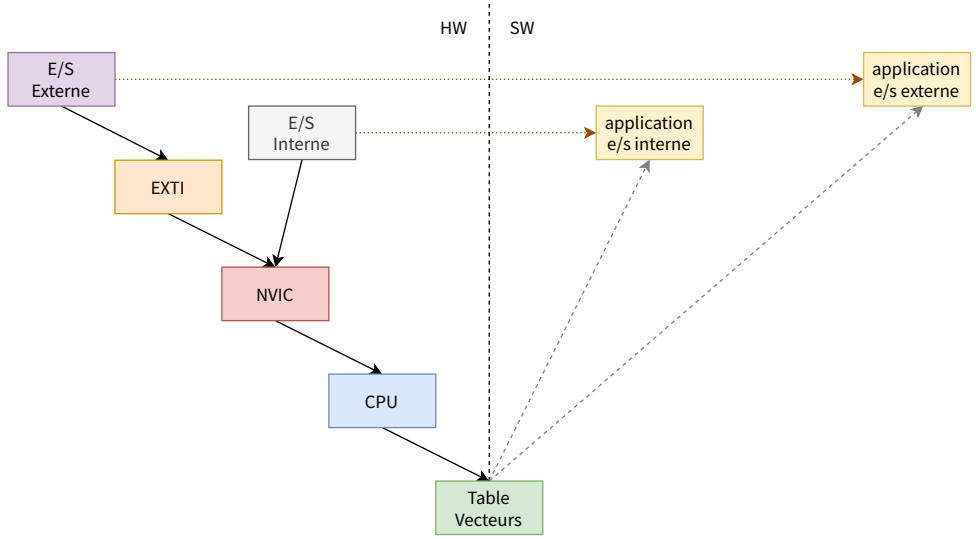

FIGURE 6.28 – Profil M - Traitement des interruptions

Pour sauver le contexte, le CPU bascule dans le mode "Thread". Il sauve ensuite les registres qui ne sont pas sauvés lors de l'appel de fonction (*Scratch Registers*), le registre de liens (LR - *Link Register*), l'adresse de retour (*Return Address*) ainsi que le registre de statut du programme (xPSR - *Program Status Register*) selon le standard AAPCS [6] (figure 6.29). Si nécessaire et afin de respecter la convention d'alignement de la pile sur 8 octets, le CPU réserve un mot supplémentaire sur la pile.

Si le µC dispose d'une unité de calcul à virgule flottante (FPU - *Floating Point Unit*), le CPU peut également sauver les registres de cette unité (figure 6.29). Cette sauvegarde ne s'effectue que si le CPU est configuré pour l'effectuer. Cette configuration s'effectue via le registre "FPCCR" contenu dans les registres de contrôle de la FPU [7].

Il charge ensuite dans le registre "LR" un code (figure 6.31) lui permettant de restaurer l'état du CPU après traitement.

6. Arm Architecture Procedure Calling Standard
7. STM32 Cortex®-M4 MCUs and MPUs programming manual (PM0214) - chapitre 4.6

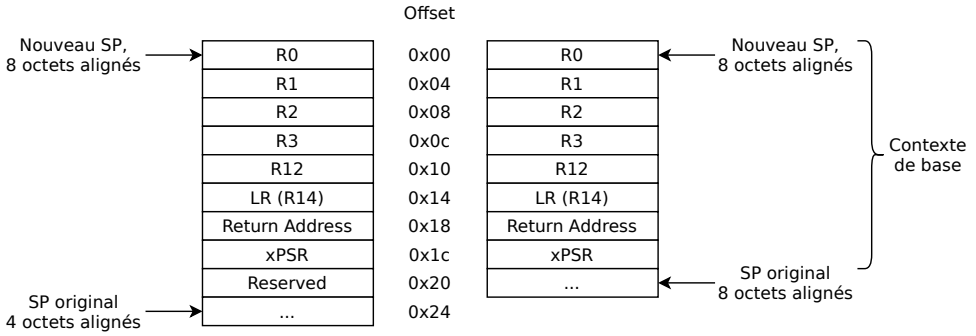

FIGURE 6.29 – Profil M - Contexte

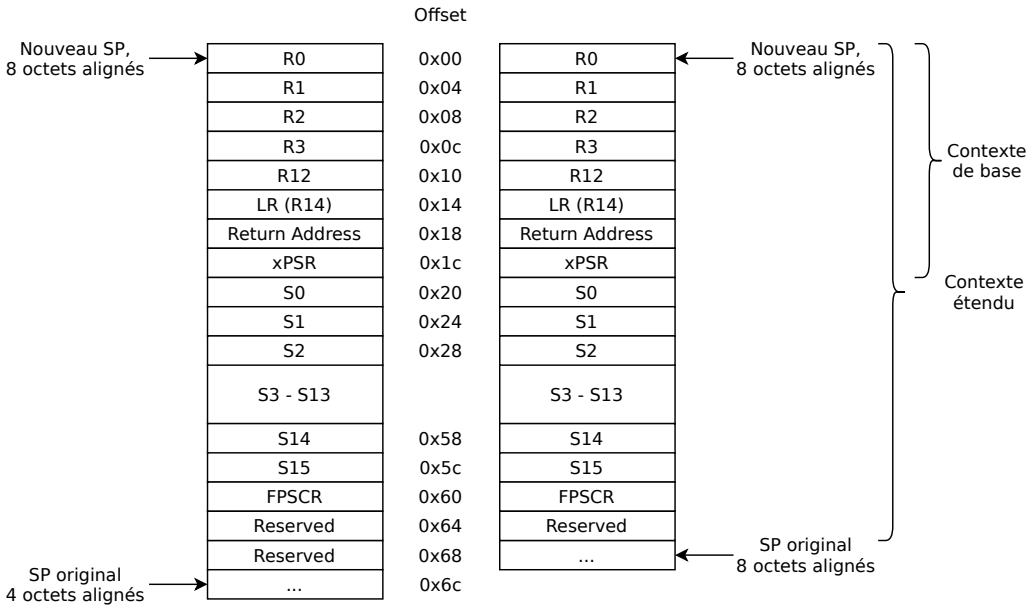

FIGURE 6.30 – Profil M - Contexte étendu

Code	Mode au retour	Pile au retour	Contexte
0xFFFF'FFE1	Handler	Main (MSP)	étendu
0xFFFF'FFE9	Thread	Main (MSP)	étendu
0xFFFF'FFED	Thread	Process (PSP)	étendu
0xFFFF'FFF1	Handler	Main (MSP)	de base
0xFFFF'FFF9	Thread	Main (MSP)	de base
0xFFFF'FFFD	Thread	Process (PSP)	de base

FIGURE 6.31 – Profil M - Codes pour la restauration du contexte

6.4.4 Priorité et préemption

La notion de priorité des interruptions et exceptions prend tout son sens lorsque celles-ci surviennent simultanément. Les µC du profil M utilisent un système de priorité des événements où plus la valeur de la priorité est faible, plus le niveau de priorité de l'événement est élevé (figure 6.32).

N° de vecteur	N° IRQ	Priorité	Description
1	-	-3 (la plus haute)	Reset
2	-14	-2	NMI
3	-13	-1	Hard Fault
4	-12	0 (configurable)	Memory Management Fault
5	-11	0 (configurable)	Bus Fault
6	-10	0 (configurable)	Usage Fault
11	-5	0 (configurable)	SVCall
14	-2	0 (configurable)	PendSV
15	-1	0 (configurable)	SysTick
16 et supérieur	0 et supérieur	0 (configurable)	Interruptions externes (IRQ)

FIGURE 6.32 – Profil M - Priorités des interruptions

Les sources internes "`Reset`", "`NMI`" et "`HardFault`" s'exécutent avec des priorités fixes de -3, -2 et -1 respectivement. Le logiciel peut définir la priorité (valeur entre 0 et 255) pour toutes les autres sources. La priorité des sources internes, sources avec un numéro IRQ négatif, se configure via les registres "`SHPR1`", "`SHPR2`" et "`SHPR3`" contenu dans le "`SCB`" [8], tandis que les sources externes se configurent via les registres du contrôleur d'interruptions (NVIC - *Nested Vector Interrupt Controler*) [9]. Le niveau de priorité d'une source se laisse configurer avec la fonction "`nvic_set_priority`" du module "`nvic.h`".

Le µC traite les exceptions ou interruptions séquentiellement selon leur priorité en débutant par la plus prioritaire. Lorsque plusieurs événements se lèvent simultanément et ont la même priorité, celui ayant le numéro le plus bas est prioritaire. Seul un événement avec une priorité plus élevée peut le préempter. Afin d'affiner le contrôle des priorités et du système de préemption, chaque priorité se compose de deux champs, un groupe de priorité et une sous-priorité. Les bits de poids fort définissent le groupe et les bits de poids faible la sous-priorité. Le registre "`AIRCR`" contenu dans le "`SCB`" permet de définir le nombre de bits déterminant la taille du groupe de priorité. La fonction

8. STM32 Cortex®-M4 MCUs and MPUs programming manual (PM0214) - chapitre 4.4
9. STM32 Cortex®-M4 MCUs and MPUs programming manual (PM0214) - chapitre 4.3

"scb_set_priority_grouping" du module "scb.h" permet de configurer la taille du groupe de priorité.

Lors de la levée d'une interruption ou d'une exception, seul le groupe de priorité détermine la préemption d'un traitement en cours. Si de multiples interruptions arrivent dans le même groupe de priorité, alors la sous-priorité sert à la précédence du traitement.

6.4.5 Gestion de la levée des interruptions

Le logiciel peut gérer la levée des interruptions et d'exceptions au niveau du CPU sur trois niveaux différents :

— Au **premier niveau**, le registre spécial "BASEPRI" permet au logiciel de restreindre la levée d'interruptions aux sources ayant une priorité plus élevée qu'une certaine valeur.

```
1 movs r0, #0x80    // choix du niveau de priorité (p.ex. 0x80)
2 msr  basepri, r0  // assignation du niveau
```

— Au **deuxième niveau**, le registre spécial "PRIMASK" permet au logiciel de bloquer la levée de toutes les interruptions ayant un niveau de priorité inférieur ou égal à 0.

```
1 cpsid i   // désactivation
2 cpsie i   // autorisation
```

— Au **troisième niveau**, le registre spécial "FAULTMASK" permet au logiciel de désactiver la levée de toutes les interruptions ayant un niveau de priorité inférieur ou égal à -1.

```
1 cpsid f   // désactivation
2 cpsie f   // autorisation
```

Les fonctions "cm_enable_interrupts", "cm_disable_interrupts", "cm_enable_faults" et "cm_disable_faults" du module "cortex.h'" permettent de gérer l'activation et le blocage des interruptions au niveau du CPU.

6.4.6 Contrôleur d'interruptions

Le contrôleur d'interruptions NVIC est capable de multiplexer, selon les réalisations, jusqu'à 496 sources d'interruptions. Le registre "ICTR" contenu dans l'espace de contrôle du système (SCS - *System Control Space*) [10] indique le nombre spécifique à l'implémentation déployée sur le µC. Chacune de ces sources correspond à une ligne de requêtes d'interruptions d'un périphérique interne au processeur.

10. Armv7-M Architecture Reference Manual - chapitre B3.2

Pour gérer les requêtes de ces sources, le NVIC dispose de six sets de registres [11]. Le module "`nvic.h`" de la bibliothèque "`libopencm3`" offre des fonctions pour manipuler ces registres.

- "**ISER**" (*Interrupt Set-Enable Registers*), ce registre permet d'autoriser la levée d'interruptions pour les différentes lignes de requêtes (fonction "`nvic_enable_irq`").
- "**ICER**" (*Interrupt Clear-Enable Registers*), ce registre permet de bloquer la levée d'interruptions pour les différentes lignes de requêtes (fonction "`nvic_disable_irq`").
- "**ISPR**" (*Interrupt Set-Pending Registers*), ce registre indique si une ou plusieurs interruptions sont en attente de traitement. Il permet également de générer, de simuler, par logiciel, la levée d'interruptions pour les lignes de requêtes données (fonctions "`nvic_get_pending_irq`" et "`nvic_set_pending_irq`").
- "**ICPR**" (*Interrupt Clear-Pending Registers*), ce registre permet de quittancer une requête en attente pour les différentes lignes de requêtes (fonction "`nvic_clear_pending_irq`").
- "**IABR**" (*Interrupt Active Bit Registers*), ce registre indique si la requête d'une source d'interruption est en cours de traitement (fonction "`nvic_get_active_irq`").
- "**IPR**" (*Interrupt Priority Registers*), ce registre permet de configurer la priorité de la ligne de requêtes donnée lors de la levée d'interruptions (fonction "`nvic_set_priority`").

Le registre "`STIR`" contenu dans le SCS permet également de générer, de simuler, la levée d'interruption pour une source donnée (fonction "`nvic_generate_software_interrupt`"). La levée d'interruptions n'est possible que si les requêtes pour une ligne d'interruptions donnée ont préalablement été autorisées.

6.4.7 Unité de gestion des entrées/sorties

L'unité de gestion des entrées/sorties (GPIO - *General Purpose Input Output*) sert à piloter 16 broches (*Pin*) numériques. En utilisant les fonctions du module "`gpio.h`", chaque broche peut se configurer aussi bien en entrée qu'en sortie.

Configurés en entrée, des périphériques externes au µP, tels des boutons-poussoirs, peuvent les utiliser comme ligne de requêtes d'interruptions via le contrôleur EXTI (*External Interrupt/Event Controller*). Ce contrôleur multiplexe 8 GPIO (figure 6.33). La broche servant de ligne de requête est choisie par les registres "`syscfg_exticr1`" à "`syscfg_exticr4`" ou en utilisant la fonction "`exti_select_source`" du module "`exti.h`".

Chaque broche peut de générer une interruption si elle détecte (figure 6.34)

- Flanc descendant (*Falling Edge*)
- Flanc montant (*Rising Edge*)

La fonction "`exti_set_trigger`" du module "`exti.h`" sert à configurer le trigger.

11. Armv7-M Architecture Reference Manual - chapitre B3.4

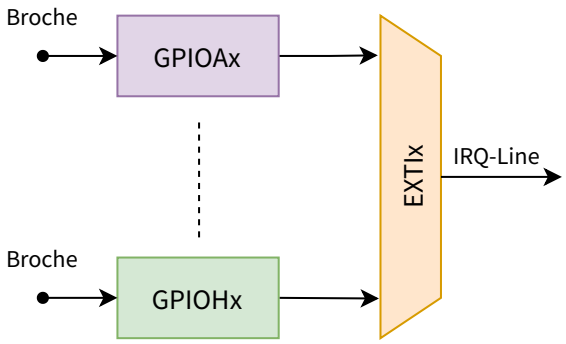

FIGURE 6.33 – Profil M - EXTI

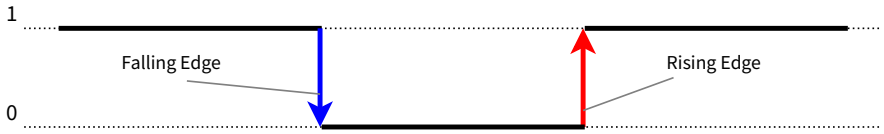

FIGURE 6.34 – Profil M - EXTI Système de détections

Lors de la levée d'une interruption sur une broche, l'unité EXTI la propage vers le contrôleur NVIC. Afin que cette interruption atteigne le CPU, il faut premièrement l'autoriser sur le contrôleur EXTI ainsi que sur le contrôleur NVIC. Les fonctions "`exti_enable_request`" et "`exti_disable_request`" permettent gérer la levée d'interruption sur le contrôleur EXTI.

La levée d'une interruption externe doit être quittancée sans quoi elle reste active. Ce quittancement doit s'effectuer durant le traitement de l'interruption avec la fonction "`exti_reset_request`" du module "`exti.h`".

6.5 Exercices

Quelques exercices pour assimiler la matière de ce chapitre.

6.5.1 Exercice 1 : Concept général

Décrivez succinctement le concept général d'une interruption, d'exception et de leur traitement.

6.5.2 Exercice 2 : Types d'événements

Citez les types d'événements pouvant survenir sur un système à µP.
Décrivez succinctement la différence entre des événements ou interruptions synchrones et asynchrones.

6.5.3 Exercice 3 : Séquence d'interruption

Citez les 4 étapes principales du traitement d'une interruption.

6.5.4 Exercice 4 : Table des vecteurs d'interruptions

Décrivez la fonction de la table des vecteurs d'interruptions.
Indiquez son contenu.

6.5.5 Exercice 5 : Commutation de contexte

Expliquez la commutation de contexte d'interruption.
Décrivez la latence d'interruptions.
Décrivez la gigue d'interruptions et donnez quelques exemples.

6.5.6 Exercice 6 : Interruptions imbriquées

Décrivez le principe d'interruptions imbriquées.

6.5.7 Exercice 7 : Section critique

Quel est le résultat de l'instruction ci-dessous si durant l'exécution de l'instruction une interruption matérielle est levée et appelle la fonction "`irq_handler`"?

```
1  int len = 0;
2
3  len += 2;   // <-- irq_handler() est appelée durant l'exécution
4
5  void irq_handler(void)
6  {
7    len += 4;
8  }
```

6.5.8 Exercice 8 : Interruptions matérielles

Citez les 3 techniques pour connecter des périphériques d'entrées/sorties à un processeur pour un traitement interruptif.

6.5.9 Exercice 9 : Génération d'exceptions

Imaginez des petits codes permettant de générer les exceptions suivantes :

- Une interruption logicielle
- Une instruction non définie
- Une exception "data abort"
- Une exception "prefetch abort"

Indiquez pour chacune de ces exceptions le numéro de vecteur pour un µC du profil M.

6.5.10 Exercice 10 : Gestion de la levée d'interruptions

Implémentez en assembleur les deux fonctions ci-dessous permettant de bloquer et d'autoriser les interruptions au niveau du CPU. Réalisez ces fonctions pour les µC du profil M.

```
1 void interrupt_enable(void);
2 void interrupt_disable(void);
```

6.5.11 Exercice 11 : Priorité d'interruptions

Démontez à l'aide de "timers" la priorité d'interruptions et la préemption.

6.5.12 Exercice 12 : Traitement d'un bouton-poussoir par interruption

Détectez les changements d'état d'un bouton-poussoir à l'aide d'interruptions.