Conseils d'un gars de Soft aux gars de FPGA

Par Martin Dubois, ing.

mdubois@kms-quebec.com

Présentation

En tant que consultant en génie informatique spécialisé en logiciel embarqué et en pilotes de périphériques, je travaille souvent avec vous, chers concepteurs électroniques de différentes entreprises. Je vous reformule donc régulièrement les mêmes conseils concernant la manière d'un peu faciliter la vie des développeurs logiciels sans vraiment compliquer la vôtre.

J'ai donc pensé faire un résumé de ces conseils.

Table des matières

1 Introduction

Pour plusieurs lecteurs, les conseils présentés dans ce document seront que de simples rappels! Mais soyons francs, un rappel ne fait jamais de mal. De plus, ceux-ci pourront garder le document (ou le lien) à portée de main pour le faire lire au petit nouveau et éviter d'avoir à corriger une erreur de débutant ou de devoir expliquer une énième fois la même chose.

2 En général

2.1 N'oubliez pas que les échelles de temps ne sont pas les mêmes

Vous, concepteur électronique, travaillez constamment dans un monde de ns (« nanoseconde ») et de ps (« picoseconde ») et vous en venez à oublier que ces unités représentent de très courts délais et parfois il vous arrive même de penser que les processeurs ou microcontrôleurs peuvent réagir dans des temps aussi exprimés avec ces mêmes unités.

J'ai une mauvaise nouvelle, ce n'est pas le cas.

Pour les petits microcontrôleurs n'utilisant pas de système d'exploitation (ou utilisant des ***RTOS*** simples) et ayant une charge de travail bien contrôlé, le temps de réponse aux interruptions se compte en us. Pour jouer sécuritairement, il est mieux de prévoir quelques dizaines de us.

Pour les processeurs modernes comportant un grand nombre de registres, un profond pipeline et un MMU, le temps de réponse aux interruptions est au mieux plusieurs dizaines de us. Et à moins de les utiliser dans un environnement extrêmement contrôlé avec un ***RTOS*** (« ***R****eal* ***T****ime* ***O****perating* ***S****ystem* »), il faut prévoir un pire cas de temps de réponse aux interruptions de l'ordre de la ms.

2.2 Associez un espace d'adressage assez grand aux *FIFO*s

Les interconnexions entre périphériques sont de plus en plus basées sur des technologies tirant un grand avantage de transfert de données en rafale (« burst »). C'est entre autres le cas de ***PCIe*** (« ***P****eripheral* ***C****omponent* ***I****nterconnect* ***e****xpress* ») et de plusieurs types d'interconnexions utilisé à l'intérieur des ***FPGA*** (« ***F****ield-****P****rogrammable* ***G****ate* ***A****rray* »).

Ici, le problème vient du fait que dans la majorité des cas, ces rafales travaillent sur des adresses consécutives et il est impossible de transférer toutes les données d'une rafale à une seule adresse ou d'une seule adresse.

Il devient alors très inefficace de lire ou écrire des ***FIFO***s (« ***F****irst* ***I****n,* ***F****irst* ***O****ut* ») au travers de ces interconnexions.

Une solution simple est de prévoir, non pas une adresse unique pour l'entrée ou la sortie du ***FIFO***, mais un espace d'adressage. Du côté matériel, il suffit d'ignorer un certain nombre de bits de poids faible de l'adresse lors du décodage de l'adresse permettant d'accéder au ***FIFO***. Ce nombre de bits à ignorer dépend de la longueur maximum des transferts qui seront effectués.

2.3 Permettez de masquer les interruptions

Lorsque nous développons le logiciel embarqué ou les pilotes de périphérique, il est important que le processeur reste le moins longtemps possible dans un état ou les interruptions sont désactivées. Ainsi, il est moins risqué que le code gérant un périphérique empêche le code gérant un autre périphérique de servir une interruption à temps.

Un des trucs souvent utilisés est d'effectuer le traitement urgent d'une interruption rapidement et de reporter le traitement moins urgent après la réactivation des interruptions. Le traitement des interruptions sous Linux ou Windows est entièrement basé sur ce concept.

Cette manière de faire cause parfois un petit problème. Il est possible que le processeur reçoive une seconde interruption d'un même périphérique avant même que le traitement moins urgent de la première interruption ne soit complété. Parfois ce n'est pas un problème, mais ce n'est malheureusement pas toujours le cas.

C'est ici que nous demandons l'aide de notre ami concepteur électronique. S'il est possible de désactiver et réactiver les interruptions efficacement, le problème se résout rapidement et simplement en désactivant les interruptions lors de la première partie du traitement de l'interruption et en les réactivant à la fin de la seconde partie.

Je conseille donc toujours de toujours permettre d'activer et désactiver les sources d'interruption individuellement.

2.4 Utilisez « write 1 to clear » pour les bits d'interruptions

Dans bien des périphériques, il existe un registre avec plusieurs bits indiquant laquelle ou lesquelles des différentes sources d'interruptions en attente d'être servie. Pour ce type de registre, il est très important de toujours utiliser une écriture d'un 1 pour effacer un 1 indiquant une interruption à servir. De cette manière, il est certain que le logiciel n'effacera pas malencontreusement un bit indiquant une nouvelle interruption au moment où il effacera un bit indiquant une interruption qu'il vient de traiter.

2.5 Pensez aux pages

Au moment de placer les registres dans l'espace d'adressage, il faut se souvenir que les processeurs gèrent la protection de la mémoire par pages (habituellement de 4 Kio). Il peut donc être pratique pour la partie logicielle que les registres associés à une même fonctionnalité ou à des fonctionnalités similaires soient regroupés dans une ou des pages et surtout que les registres qui sont associés à des fonctionnalités différentes ne se retrouvent pas dans une même page.

Par exemple, imaginons un périphérique qui comporte des registres de statistiques générales, un ***FIFO*** permettant de recevoir des données et un ensemble de registres de configurations. Les registres de statistique ne sont jamais écrits et il n'y a probablement pas d'impact réel pour la sécurité du système s'ils sont lus directement par l'application. Si ceux-ci sont seuls dans une page de mémoire, le pilote peut simplement donner la permission aux applications de lire cette page et l'application pourra alors les consulter très efficacement.

C'est la même chose pour le ***FIFO*** de réception. S'il est seul dans sa page (ou ses pages), le pilote pourra donner le droit de lecture à une application en particulier et celle-ci pourra obtenir les données d'une manière très efficace.

Au contraire, si les registres de configuration sont éparpillés dans les mêmes pages que les registres de statistique ou le **FIFO**, il faudra que le pilote se charge de chacun des accès au registre de statistique et au FIFO pour éviter qu'une application malveillante ne lise ou écrive les registres de configuration. Cela rendra le pilote beaucoup moins efficace.

3 Concernant les DMA

3.1 Pensez aux pages

Souvent, quand l'électronique utilise un contrôleur ***DMA*** (« ***D****irect* ***M****emory* ***A****ccess* ») c'est pour transférer des données directement d'un espace mémoire d'une application vers un périphérique ou l'inverse.

Le problème vient ici du ***MMU*** (« ***M****emory* ***M****anagement* ***U****nit* ») des processeurs modernes. Celui-ci permet à une application de voir comme continues un ensemble de pages qui physiquement ne sont pas continues.

Il est donc important que le contrôleur ***DMA*** soit capable d'exécuter non pas un seul transfert, mais une liste de transfert. De cette manière, le logiciel peut programmer le contrôleur ***DMA*** pour effectuer une première partie du transfert vers une première page, une seconde partie vers une seconde page…

4 Dans le cas de PCIe

4.1 Utilisez les interruptions MSI

PICe (et ***PCI-X*** avant lui) définit le concept d'interruption passé par message. Ce concept libère les concepteurs électroniques des contraintes imposées par un nombre limité de lignes d'interruption. Il permet aussi, pour un même périphérique, de rapporter plusieurs types d'interruption différents.

Mais plus important, il permet aux pilotes de périphérique de ne plus avoir à partager les lignes d'interruption entre eux. Le partage d'interruption fonctionne habituellement bien. Cependant, surtout lors des phases de déverminage, il peut causer de petits ennuis aux développeurs logiciels.

Je me souviens d'une fois où le périphérique pour lequel je développais un pilote était incapable de transmettre une interruption vers le processeur, mais comme la ligne d'interruption était partagée entre ce périphérique et un autre périphérique produisant un grand nombre d'interruptions, je ne m'en suis simplement pas rendu compte jusqu'au jour où quelqu'un d'autre essaya la carte dans un autre ordinateur.

Donc, S.V.P. quand c'est possible, utiliser les ***MSI*** (« ***M****essage* ***S****ignaled* ***I****nterrupt* »).

4.2 Minimisez la taille des *BARs*

Ce conseil est vraiment important.

Dans un système, chacune des nombreuses cartes ***PCIe*** rapporte au système d'exploitation le nombre et la taille des espaces mémoire nécessaire pour son fonctionnement. Ensuite, le système attribue une adresse à chacun des espaces mémoire en tenant compte de sévère contrainte d'alignement des adresses. L'utilisation de plusieurs niveaux de pont (« bridge » ou « switch ») ajouter à ces contraintes d'alignement rend le gaspillage d'espace d'adressage gigantesque.

L'espace global d'adressage physique est divisé entre plusieurs, dont les cartes ***PCIe***, les autres périphériques, la mémoire centrale… Aussi, les processeurs 64 bits n'ont pas 64 bits d'adressage réel. Souvent, ils sont limités à 48 bits ou même moins.

Il est donc important de ne pas utiliser d'espace mémoire plus grand que nécessaire. Il est aussi important de cesser d'utiliser des espaces mémoire avec des adresses de 32 bits. Celle-ci ajoute des contraintes supplémentaires au système lors de l'attribution des adresses.

Croyez-moi, il n'est vraiment pas rare qu'un nouveau périphérique ***PCIe*** soit inutilisable ou seulement utilisable dans certains ordinateurs en raison de de demande trop grande en espace d'adressage.

4.3 Permettez les « *Burst* »

L'arrivée du bus ***PCIe*** permet de libérer le monde informatique des limitations de vitesse de transfert associé au PCI « parallèle ». Cependant, un bénéfice a souvent un coup et c'est le cas ici. ***PCIe*** augmente les taux de transfert au détriment de la latence initiale. La lecture d'un simple mot de 32 bits sur une connexion ***PCIe*** x8 prend plusieurs us. Bref, si les transferts en rafale (« *burst* ») ne sont pas utilisés, le bus ***PCIe*** est significativement plus lent que son prédécesseur.

Pour les concepteurs matériels, il y a plusieurs moyens de permettre l'utilisation de transfert en rafale.

Le premier est de concevoir des cartes maîtres (« *master* »). Celle-ci initie les transferts et peuvent donc contrôler complètement l'utilisation des rafales.

Le second est de permettre l'utilisation de la cache par le processeur. Ainsi, le processeur peu utilisé des rafales de la longueur d'une ligne de cache (habituellement 64 octets) pour lire ou écrire les données. Naturellement ce n'est pas toujours possible, car l'utilisation de la cache change l'ordre d'accès aux registres ce qui est catastrophique quand les accès à certains registres servent à lancer des opérations dans le matériel.

Pour utiliser cette seconde méthode, il faut donc utiliser un ***BAR*** (« ***B****ase* ***A****ddress* ***R****egister* ») pour les registres de configuration, de statuts et de contrôle et un autre ***BAR*** pour les grands espaces de données ou la cache peut être utilisée sans risque.

4.4 N'oubliez pas que les adresses internes ne correspondent pas aux adresses *PCIe*

Lorsque vous concevez une carte ***PCIe***, vous avez souvent un ensemble d'interconnexion interne et un bloc qui permet de faire le lien entre ces interconnexions internes et le bus ***PCIe***.

Il est important de ce souvenir que les adresses internes ne correspondent pas aux adresses ***PCIe***.

Quand la carte ***PCIe*** joue le rôle d'esclave, les ***BAR***s définissent comment les adresses ***PCIe*** sont converties en adresse interne. Cette conversion est définie par le standard PCI et est généralement très bien comprise par les concepteurs électroniques.

Les choses se gâtent un peu quand la carte devient le maître du lien ***PCIe***. Les adresses internes doivent presque toujours être converties en adresse ***PCIe***. Souvent, des adresses internet de 32 bits doivent être converties en adresse 64 bits. Ce qui complique les choses ici, c'est que cette conversion n'est pas couverte par le standard ***PCI*** et chaque manufacturier (ou programmeur d'IP) à sa propre manière de faire. Parfois, cette étape de conversion est incluse dans le pont ***PCIe*** et parfois elle fait partie du moteur ***DMA***.

Ici, je ne peux vous donner de conseil générique. Je ne peux que vous inviter à lire attentivement la documentation à ce sujet dans votre cas spécifique.

5 Références

Plusieurs des conseils tournent autour de l'utilisation de la mémoire virtuelle et le traitement des interruptions par les processeurs. Le livre « What Makes IT Page? : The Windows 7 (x64) Virtual Memory Manager » de « Enrico Martignetti » aborde ces sujets dans la perspective spécifique du système d'exploitation Windows 7. Il inclut une première section expliquant la gestion des interruptions, les niveaux d'exécution et la mémoire virtuelle au niveau matériel. Cette section est très bien écrite, courte et facile à lire. Toute personne développant des périphériques utilisés à partir d'un système d'exploitation (que ce soit Linux ou Windows) devrait lire au moins cette section de ce livre.