POLYTECHNIQUE MONTRÉAL

UNIVERSITÉ D'INGÉNIERIE

INF3610

Système embarqué

Lab 1 Partie 3

Travail présenté à :

[redacted]

Soumis par :

[redacted]

18 novembre 2024

Question 1

Lequel des cas s'applique à votre implémentation et quel est son avantage par rapport à l'autre cas.

Avantage de Cas 1 par rapport à Cas 2 :
L'avantage de ce cas est que la génération de la rafale se fait en premier, avant que TaskGenerate indique qu'elle est prête à être consommée par TaskQueueing. Cela signifie que TaskQueueing reçoit immédiatement la rafale une fois qu'il détecte le signal req, ce qui réduit le délai d'attente pour TaskQueueing. Cela optimise le débit de traitement, car TaskGenerate termine la génération de données avant d'indiquer la disponibilité, ce qui évite d'occuper inutilement TaskQueueing en attente d'une rafale incomplète.

En comparaison, Cas 2 retarde le signal req jusqu'à ce que TaskGenerate soit prêt à générer la rafale, ce qui peut allonger le temps d'attente côté TaskQueueing car celui-ci doit attendre plus longtemps avant de pouvoir traiter une rafale complète. Le flux de données est donc plus fluide avec Cas 1, car TaskQueueing peut commencer le traitement dès que les données sont prêtes.

Question 2

Impact du déplacement de TaskGenerate sur core1 et de l'augmentation de delai_pour_vider_les_fifos_msec

Lorsqu'on déplace TaskGenerate sur core1, cela entraîne effectivement une séparation des tâches, ce qui devrait libérer core0 pour permettre à TaskQueueing et aux autres tâches de s'exécuter plus rapidement. Cependant, dans un système multiprocesseur, la gestion du cache peut devenir un facteur critique. Désactiver la cache peut ralentir l'application d'un facteur 10

Effet de la désactivation du cache

Si le cache est désactivé pour éviter des incohérences de données entre les deux cœurs (ce qui est courant lorsqu'on utilise de la mémoire partagée sans un protocole de cohérence de cache), cela a les impacts suivants :

Accès plus lent à la mémoire : Avec le cache désactivé, chaque accès à la mémoire partagée entre core0 et core1 est directement effectué dans la mémoire DDR, ce qui est beaucoup plus lent que des accès en cache. Cela signifie que chaque opération de lecture et d'écriture dans la mémoire partagée (par exemple, pour le transfert des paquets) prend plus de temps, augmentant ainsi la latence des échanges entre TaskGenerate et TaskQueueing.

Augmentation de la latence pour TaskQueueing : En raison de ces accès directs à la DDR, TaskQueueing met plus de temps à traiter les paquets qu'il consomme, car chaque lecture (pour récupérer les paquets générés) est plus lente. Cela réduit donc la fréquence de traitement des paquets, et, par conséquent, TaskGenerate se retrouve à produire plus de paquets que

TaskQueueing peut consommer, d'où la nécessité d'augmenter delai_pour_vider_les_fifos_msec pour éviter que la file d'attente ne se remplisse trop rapidement.

Surcharge des files d'attente : Avec une vitesse de traitement plus lente du côté de TaskQueueing, les files de traitement (vidéo, audio, etc.) se remplissent plus rapidement, et le système risque d'atteindre la limite de capacité des files plus fréquemment, ce qui entraîne des rejets de paquets. La désactivation du cache limite donc le débit maximal du système.

En résumé
La désactivation du cache pour garantir la cohérence des données entre core0 et core1 a pour effet de ralentir les accès à la mémoire partagée, ce qui augmente la latence de traitement de TaskQueueing. Cela compense en partie les gains potentiels de la répartition des tâches sur deux cœurs. C'est pourquoi il devient nécessaire d'augmenter delai_pour_vider_les_fifos_msec pour donner à TaskQueueing suffisamment de temps pour traiter les paquets générés sans saturer les files d'attente.

En conclusion, ce ralentissement lié à l'absence de cache est la raison principale pour laquelle les performances de TaskQueueing ne s'améliorent pas proportionnellement à l'ajout d'un deuxième cœur, malgré le parallélisme. Voir le mot plus bas après la question 3)

Question 3

À propos de la minuterie watchdog

a) Expliquez comment vous avez implémenté votre watchdog. Plus précisément, expliquez les paramètres de OSTmrCreate, justifiez le mode de fonctionnement du watchdog et décrire comment la fonction de callback réalise le travail de stopper TaskGenerate.

Pour implémenter le watchdog, j'ai d'abord créé une variable ShutdownTmr de type OS_TMR. Ensuite, au début de TaskGenerate, j'ai initialisé cette minuterie avec la fonction OSTmrCreate. Voici une brève explication des paramètres de cette fonction :

- p_tmr et p_name servent à nommer la minuterie, ce qui permet de la différencier d'autres minuteries.
- Le paramètre dly définit le délai initial avant que la minuterie ne commence.
- Le paramètre period représente la période de la minuterie. Cela signifie que la minuterie se réinitialise et redémarre à intervalles réguliers définis par la valeur de period. Comme nous voulons que la minuterie s'exécute une seule fois, nous avons choisi l'option OS_OPT_TMR_ONE_SHOT, indiquant une minuterie qui ne s'écoule qu'une seule fois, et nous avons fixé period à 0 car nous ne voulons pas une minuterie par intervalles.
- Le paramètre p_callback correspond à la fonction appelée lorsque la minuterie arrive à expiration.
- Lorsque la minuterie arrive à 0 (c'est-à-dire qu'elle n'a pas reçu de REQ depuis plus de 30 secondes), la fonction de callback ShutdownTaskGenerateFct s'exécute. Nous avons donné une priorité très grande à la fonction de callback pour qu'elle soit supérieure à celle de TaskGenerate, qui est en attente active. Ainsi, sans cette priorité élevée, TaskGenerate resterait en exécution continue et la fonction de callback ne pourrait pas s'exécuter.

Dans la fonction de callback, endExecution est appelée (fournie dans le code initial), ce qui entraîne la suppression de la tâche TaskGenerate, suivie de la suppression de StartupTask via la fonction OSTaskDel. Pour le dly, vu que la tache OS_Timer_Task (tache s'occupant de la minuterie logicielle) est appelle avec une fréquence de 10hz (paramètres par défaut), alors nous allons choisir un dly de 300 ticks (10hz * 30sec).

b) J'ai dit en classe qu'une des limitations du watchdog est sa précision limitée à un multiple du tick d'hologe (tick rate). Selon vous, la solution offerte par Xilinx avec le composant AXI Timerbase Watchdog Timer de la librairie offrirait-elle un comportement similaire au watchdog utilisé à la section 3.3 mais avec plus de précision ? Justifiez.

Le AXI Timerbase Watchdog Timer de Xilinx, étant un compteur matériel, permet effectivement d'obtenir un comportement similaire à celui du watchdog logiciel tout en offrant une précision supérieure. Contrairement à un watchdog logiciel qui dépend du tick rate du système, le compteur AXI est Indépendant. Ainsi, même si le tick rate varie, la précision du watchdog AXI restera constante.

L'implémentation de ce mécanisme dans le système nécessiterait d'utiliser les registres du AXI Timer pour configurer le compteur, plutôt que d'appeler OSTmrStart comme dans le cas d'un watchdog logiciel. De plus, lorsque le AXI Timer arrive à expiration, il déclenche un ISR, qui remplit le même rôle que le callback dans un watchdog logiciel.

Cette solution offre une bien meilleure précision, car le AXI Timer fonctionne à une fréquence beaucoup plus élevée que le tick rate du système. Par exemple, si l'AXI est connecté à une horloge de 100 MHz, sa précision sera supérieure à celle d'un système avec un tick rate de 8000 Hz. (Le Axi ayant une horloge max entre 120Mhz et 220Mhz le modele de FPGA)

Suite du no 2:
Une ne solution aurait peut être d'utiliser Xil_DCacheInvalidateRange i.e. invalider seulement la région partagée à BASEADDR. J'ai essayé mais ça été été sans succès sur la performance. Comme je manquais de temps j'ai pas pu aller plus loin. C'est peut être parce que je ne spécifiait pas les bons ranges (doit peut être compatible avec les zones de cache). À investiguer.

Autre approche possible (à investiguer également) serait l'utilisation de composant FIFO de la librairie Vivado comme AXI Data FIFO ou encore AXI4-Stream Data FIFO plutôt que d'utiliser directement la DDR comme on l'a fait. Peut être que ces composant ont déjà des mécanismes pour éviter de désactiver toute la cache.