INF3610 - TP2

Présenté à

Jeff Falcon, chargé de laboratoire

Par

Michael Chidiac, 1466616

Olivier Moussavou Cotte, 1538493

École Polytechnique de Montréal

30 octobre 2013

Barème correction

EXÉCUTION

☐ Fonctionnement sur carte FPGA	/4

TOTAL EXÉCUTION /4

CODE SOURCE

☐ Contenu des tâches	/3
☐ Mécanismes de communication et de synchronisation	/1
☐ Respect de l'énoncé, commentaire et clarté du code	/1
☐ Fonctionnement des fit_timer	/1

TOTAL RAPPORT /6

RAPPORT

☐ Présentation générale, introduction, conclusion	/1
☐ Fonctionnement du système	/2

QUESTIONS

☐ Question 1	/1
☐ Question 2	/1
☐ Question 3	/1
☐ Question 4	/2
☐ Question 5	/1
☐ Question 6	/1

TOTAL RAPPORT /10

NOTE GLOBALE **/20**

INTRODUCTION

Afin de nous familiariser avec les systèmes a temps réel, ce second laboratoire vise à nous faire développer un système embarqué sur un processeur µBlaze sur les cartes FPGA du laboratoire. Nous allons devoir utiliser le RTOS µC.

Nous allons concevoir un système qui simule l'échange de paquets informatiques à travers plusieurs routeurs. Ces paquets vont devoir se rendre à la bonne destination indiquée dans leur structure en passant par les bons routeurs sur le chemin. Chaque routeur doit regarder la destination finale du paquet et déterminer le prochain routeur où envoyer le paquet. Si un routeur reçoit un paquet qui n'est pas dans sa plage d'adresse à traiter, le paquet est alors rejeté. Le routeur doit aussi vérifier la validité du paquet grâce à un calcul CRC. La troisième et dernière fonction de nos routeurs sera de trier les paquets selon leur type, c'est-à-dire des paquets audio, vidéo ou autre.

FONCTIONNEMENT DU SYSTÈME

Toutes les tâches sont initialiser et créer dans le main qui appelle une fonction _CreateTask(). Nous avons aussi la création de trois Semaphore : SemPrint, SemStop et SemVerification; cinq queues: ptrFifoIn, ptrFifoVideo, ptrFifoAudio, ptrFifoOtherwise et ptrFifoAuxilary; et finalement 4 mailbox: Int1Mbox, Int2Mbox, Int3Mbox, Int4Mbox. La tâche main initie le BSP et commence le multitasking.

Chaque paquet est généré à l'intérieur d'une tâche nommée TaskInjectPacket. Cette tâche crée des structures de type Paquets définis ainsi :

```
typedef struct {
        unsigned int src;
        unsigned int dst;
        unsigned int type;
   unsigned int crc;
        unsigned int data[12];
}Packet;
```

Chacun des unsigned int de la structure représente les 4 octets définis dans la structure d'un vrai paquet. Seul le unsigned data représente 48 octets de données aléatoirement généré par la tâche. Le CRC est calculé en dernier par la tâche.

Ils sont ensuite introduits dans la FifoIn. Si la fifo est pleine, le paquet est rejeté. Un OSTimeDly(50) est ensuite effectué afin de ne produire que 2 paquets par seconde.

Le paquet est alors recueillis par de la FifoIn par la tâche TaskComputing. Sa première fonction est de rejeter les paquets qui ne sont pas dans son espace d'adresse en vérifiant les bordures REJECT_LOW et REJECT_HIGH de 1 à 4.

Si la source se retrouve bel et bien dans l'espace d'adressage, on vérifie la validité CRC. Pour ce faire, nous sauvegardons la valeur du CRC du paquet dans une variable temporaire puis nous

réinitialisons la variable CRC du paquet a 0. Nous passons ensuite le paquet complet à la fonction computeCRC qui nous retourne le bon CRC. Nous comparons cette valeur de retour avec celle transmise avec le paquet. Si les valeurs sont égales, nous continuions le traitement, autrement nous rejetons le paquet. Il ne faut pas oublier de remettre la valeur du CRC dans le paquet.

TaskComputing met alors le paquet dans l'une des fifos selon le type du paquet, c'est-à-dire High, Medium ou Low. Les trois fifos respectives sont FifoVideo, FifoAudio et FifoOtherwise.

Le paquet TaskForwarding se reveille ensuite afin de lire à l'interieur des trois fifo, et ceci en ordre de priorité. Si le message reçu n'est pas vide, nous procédons à la vérification de l'intervalle de destination, comme dans la tâche TaskComputing. Nous déposons par la suite le paquet dans la mailbox correspondant à sa destination. Si la boite est pleine alors le paquet est rejeté. Autrement, nous incrémentons la valeur à être affiché sur les LEDs de la planche FPGA.

TaskPrint à ensuite la tâche d'aller lire dans les mailbox et d'imprimer à l'écran l'information sur les paquets.

Un fifo auxiliaire est disponible pour les paquets qui sont rejeté dans la tâche TaskComputing. Cette fifo est vidée par la tâche TaskVerification. Cette tâche vérifie si il est possible de vidé cette la file auxiliaire et d'insérer les paquets dans une des trois files prioritaires. Si les fifos High, Medium et Low sont pleines le paquet est rejeté. Elle est réveillée grâce à une interruption faisant appel à fitTimer2Handler qui débloque le sémaphore SemVerification après laquelle TaskVerification attend.

La fonction fitTimerHandler est pour sa part appelé par une interruption. Elle sert a vérifié si le nombre de paquets rejeté est supérieur à 15 et si elle l'est, débloque le sémaphore SemStop qui lui-même débloque la tâche TaskStop. Cette dernière fait appelle à une fonction nommée BSP_IntDisAll() qui éteint les interruptions ainsi qu'une fonction _DeleteTask() qui détruit toutes les tâches en cours et ferme ainsi le routeur.

JUSTIFICATION

Dans la tâche TaskFowarding, nous avons décidé d'utiliser des OSQAccept plutôt que des accès bloquant pour être en mesure choisir le premier paquet près dans une file. Dans le cas où la file est vide, OSQAccept retourne le pointeur nul, alors avec un enchaînement if-else if, la première file à posséder des paquets est vidée et un pointeur valide est retourne. Pour se faire nous avons dû mettre à 1 dans le BSP OS_Q_EN.

Nous avons aussi débloqué OSQQuery pour vérifier l'état des files à l'exucition. On était conscient que lorsque la tâche TaskVerification est exécutée, il était possible que la file auxiliaire contienne plusieurs messages. On peut donc questionner la file sur le nombre de message et ensuite itérer pour les rajouter dans les files prioritaire sans bloquer la tâche TaskVerification. Pour se faire nous avons dû mettre à 1 dans le BSP OS_Q_EN && OS_QUERY_EN.

PRIORITÉS DES TÂCHES

Nécessairement la tâche TaskStop doit être la plus prioritaire. Dès que le fitTimerHandler incrémente le sémaphore de cette tâche, elle doit devenir la tâche la plus prioritaire à être ordonnancé d'où la nécessité d'avoir la plus grande priorité parmi toutes les tâches du système. Donc dès que le sémaphore SemStop = 1, TaskStop est exécutée et le routeur s'arrête peu importe les tâches roulant.

Deuxièmement, la tâche TaskVerification doit suivre TaskStop. Vu qu'elle aussi est réveillée périodiquement par l'incrémentation d'un sémaphore, elle doit être plus prioritaire que toutes les autres tâches roulant sur le système pour qu'une fois réveille elle soit automatiquement ordonnancée. Vu que TaskStop ne roule qu'une fois lorsque le système s'arrête, TaskVerification peut être moins prioritaire que cette dernière tout en étant plus prioritaire que toutes les autres tâches.

La tâche TaskPrint dépend uniquement de la tâche TaskForward, mais elle doit avoir préemption sur TaskInjectPacket et TaskComputing. Cette tâche est aussi synchronisée par un sémaphore binaire par la tâche TaskForward. Dès que l'on incrémente son sémaphore, cela signifie qu'un message dans la mailbox est disponible. Elle doit donc avoir préemption sur toutes les autres tâches non synchronisé par les sémaphores lorsqu'un message est disponible pour ensuite se rendormir et attendre que TaskForward traite un autre paquet.

La tâche TaskInjectPacket est plus prioritaire que TaskComputing puisque l'on a besoin qu'elle créer des paquets et les post dans la fifo pour que TaskComputing les traitent. TaskInjectPacket rend le contrôle à TaskComputing lorsqu'elle s'endort. SIion avait inversé les priorités de ces deux tâches, TaskComputing aurait passé son temps à attendre TaskInjectPacket.

L'exécution de la tâche TaskForward dépend de TaskComputing. En effet, TaskComputing met à jour une file auxiliaire qui contient les paquets rejetés plus tôt, car les files prioritaires étaient pleines. Lorsque TaskComputing finit de traiter les paquets poster par TaskInjectPacket, la tâche la moins prioritaire TaskForward se réveille et met à jour les mailbox pour les envoyer vers l'interface d'où la priorité la plus faible.

RÉPONSES AUX QUESTIONS

QUESTION 1

Un Board Support Package (BSP) est une collection de bibliothèques logicielles et de pilotes qui formeront la couche logicielle la plus basse d'un système matériel-logiciel. Pour que le logiciel soit en mesure de rouler sur une plate-forme matérielle donnée, on doit se lier avec un BSP, car pour faire communiquer le matériel avec le logiciel, on a besoin des pilotes des périphériques et de quelques librairies pour réaliser des programmes utiles.

Les pilotes et les libraires dépendent de la plateforme sur laquelle on les utilise puisqu'il communique avec ses périphériques. Si on change un périphérique, il faudrait nécessairement réécrire les pilotes et possiblement modifier les libraires pour quelles soit toujours en mesure de communiquer avec le nouveau périphérique.

Finalement, le BSP contient des aussi directives de compilation et les paramètres matériels qui sont utilisés pour configurer la plate-forme (l'adresse de base des périphériques, le mode dans lequel certain périphérique ont été synthétisé etc.). Ces paramètres dépendent eux aussi des périphériques et ne serait pas compatible si on utilisait une plate-forme différente (du genre une autre carte de développement).

QUESTION 2

Aucune section critique n'est présente dans le code. Il ne peut donc pas y avoir d'inversion de priorité, car aucune tâche n'attend les ressources possédées par une autre. Les tâches sont synchronisées unilatéralement par des sémaphores binaires ce qui permet de dicter à quel moment certaines tâches doivent s'exécuter.

QUESTION 3

OSTimeDly() suspend la tâche courant pour le nombre de tiques passé en paramètre. Le temps de délai dépend donc du nombre de tiques par seconde du système. On peut trouver cette valeur en examinant OS_TICKS_PER_SEC. Dans notre cas, OS_TICKS_PER_SEC = 100. Donc pour attendre une demi-seconde on doit effectivement attendre 50 tiques.

QUESTION 4

Non, car le tique de l'OS est généré par une partie matérielle indépendante de l'horloge du microprocesseur. Pour ce faire, l'OS possède une minuterie interne qui lui permet de générer des interruptions à intervalle régulière qui correspondent à son tique.

La cadence du CPU correspond à la vitesse d'horloge du microprocesseur. L'horloge du processeur coordonne l'ensemble des opérations du processeur de la mémoire et de ses périphériques en générant périodiquement un signal de référence.

L'horloge du CPU n'étant pas l'horloge du compteur interne de l'OS, les deux tiques sont différents. Par exemple, pour ce tp nous avons les valeurs suivantes : l'horloge du compteur est de 100Hz (100 tiques / secondes) et l'horloge du µBlaze est cadencée à 100 MHz.

QUESTION 5

On ne peut pas faire le travail de la tâche TaskVerification dans le fitTimerHandler2, car on effectue un appel bloquant sur une file. On ne peut pas faire d'appel bloquant dans un gestionnaire d'interruption, car les gestionnaires d'interruption ne sont pas des tâches, donc s'ils s'endorment, il n'y a rien de plus important à réveiller. On bloque donc l'exécution des autres tâches. Les ISRs doivent donc exécuter de façon atomique.

On pourrait utiliser les appels non-bloquants tel que OSQAccept(void*) dans l'ISR et réveiller la tâche TaskVerification lorsque la file serait prête à vider. Le programme serait toutefois plus complexe et les résultats seraient non-reproductibles à cause de l'ordonnancement (dans le cas où il y aurait des tâches plus prioritaires que la tâche TaskVerification).

L'avantage de réveiller la tâche TaskVerification à partir de l'ISR est que le système n'a pas besoin de faire d'attentes passives. L'OS ne passe pas son temps à attendre que la file se fait remplir par un autre ISR.

Le désavantage est qu'il faut synchroniser la tâche TaskVerification et l'ISR par un sémaphore binaire. Dans un tp comme celui-ci c'était trivial, mais à mesure que le nombre d'évènement augmente, la programmation devient plus complexe et plus sensible aux erreurs humaines.

Un compromis intéressant aurait été d'utiliser un appel bloquant avec timeout dans la tâche TaskVerification et retourner une erreur si la file était vide. Comme sa plus besoin du fitTimerHandler2 et aucune attente passive de la part de l'OS.

QUESTION 6

La tâche TaskInjectPacket envoie dans la file d'entrée deux paquets par seconde. De plus, on devait simuler une table de routage en ajoutant un délai d'une seconde dans la tâche TaskFowarding. Les paquets sont donc effectivement traités à un débit de 1 paquet/seconde. On ne pourra donc que traiter 1 paquet/seconde à cause de la tâche TaskFowarding.

Puisque l'on a accès 2 paquets/seconde dans la file d'entrée, on aurait besoin de deux tâches TaskFowarding au minimum pour maximiser le nombre de paquets traité.

CONCLUSION

Dans ce laboratoire, nous avons pu implémenter sur une carte de développement possédant une FPGA un système temps réel en utilisant un RTOS qui était MicroC-OS/II. Nous avons à la fois synthétisé le matériel avec XPS et implémenter du logiciel avec SDK. On du travailler avec des aspects plus techniques tel les pilotes des périphériques et la synchronisation de tâches dans un contexte temps réel.

Finalement, on comprend mieux les limitations des systèmes matériels à cause de la petite quantité de mémoire disponible, car il n'y avait pas beaucoup d'espace sur la BRAM. Aussi, les outils ne sont pas aussi performant que si l'on développait uniquement du matériel ou du logiciel (du genre EDK vs la suite ISE de XIlinx ou tout simplement Eclipse). Le débogage était ardue puisque gdb n'est pas complétement stable, ne donne pas nécessairement la bonne valeur des variables etc. Aussi déboguer des systèmes temps réels est difficile à cause de l'ordonnancement des tâches qui doit respecter des délais, les ressources partagées et les interruptions. Il n'est pas toujours possible de faire une exécution step-by-step dans ce contexte.