École Polytechnique de Montréal

Département de Génie Informatique

INF3610

Automne 2013

Laboratoire #2

Routeur sur puce FPGA

---

1. Objectif

L'objectif principal de ce laboratoire est de concevoir une application temps réel pour un système embarqué en ayant recours au RTOS µC et de l'exécuter sur un processeur µBlaze implanté sur une puce FPGA.

Plus précisément, les objectifs spécifiques du laboratoire sont:

- Approfondir ses connaissances de µC.
- S'initier aux environnements de développement de SoC, tel Xilinx EDK et ISE.
- Étudier les spécificités de la programmation pour SoC.
- Se familiariser avec une plateforme reconfigurable.
- Utiliser et comprendre un mécanisme d'interface entre modules logiciels et matériels.

2. Mise en contexte

La *figure 1* illustre un réseau téléinformatique permettant l'échange de paquets d'une source à une destination. Dépendamment de la destination, les paquets transitent à travers un ou plusieurs routeurs. Par exemple, pour aller de la ***source 0*** à la ***destination 1***, les paquets vont passer par le routeur 1, le routeur 2 et le routeur 4 alors que pour aller de la ***source 0*** à la ***destination 2***, les paquets vont passer par le routeur 1, le routeur 3 et le routeur 5. Le routeur 1 devra donc regarder l'adresse de destination de chaque paquet pour décider si ce dernier doit transiger vers le routeur 2 ou le routeur 3. La fonction principale d'un routeur est donc de prendre un paquet et de le renvoyer au bon endroit en fonction de la destination finale. Il devra du même coup rejeter les paquets qui ne sont pas dans son espace

d'adressage. Une deuxième fonction est de vérifier les erreurs de transmission à travers un calcul de type CRC (*checksum*). Finalement, un routeur peut aussi supporter une qualité de service (*QoS*) en triant les paquets selon qu'il s'agit par exemple d'un paquet audio, vidéo ou encore contenant des données quelconques.

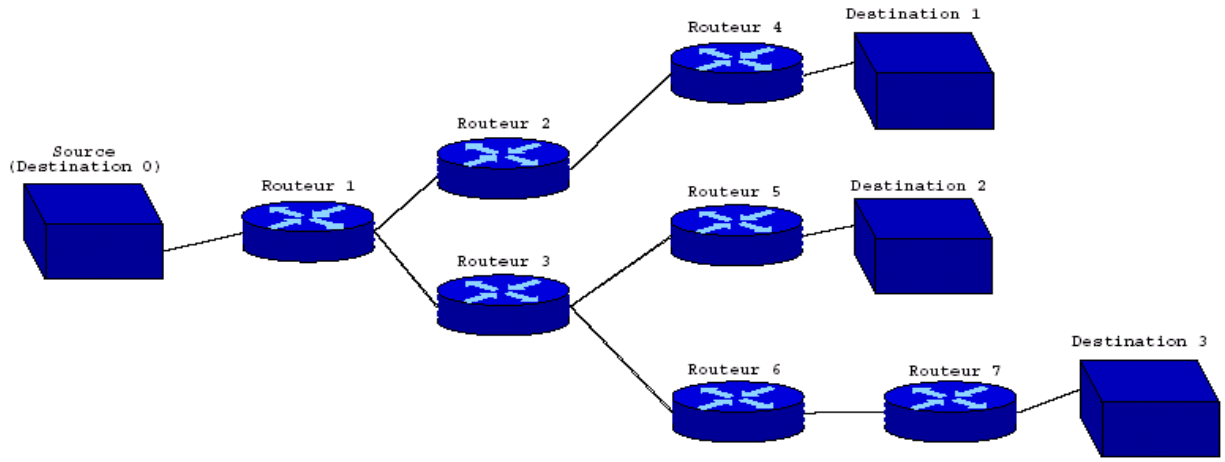


Figure 1 Réseau téléinformatique

Évidemment, un routeur peut supporter bien d'autres fonctions. Toutefois, dans ce laboratoire, nous nous concentrerons sur les trois énumérées au paragraphe précédent. Le format des paquets sur le réseau est le suivant :

4 octets	***4 octets***	***4 octets***	***4 octets***	***48 octets***
Source	Destination	Type	CRC	Data

Ces paquets sont de taille fixe (***64 octets***) et possèdent cinq champs, soit une source indiquant la provenance du paquet, une destination, un type pour la qualité de service, un code CRC pour la détection d'erreur et finalement les données transportées. La définition de cette structure vous sera fournie.

3. Description des tâches

La figure 2 illustre le flot des paquets à travers le routeur à implémenter.

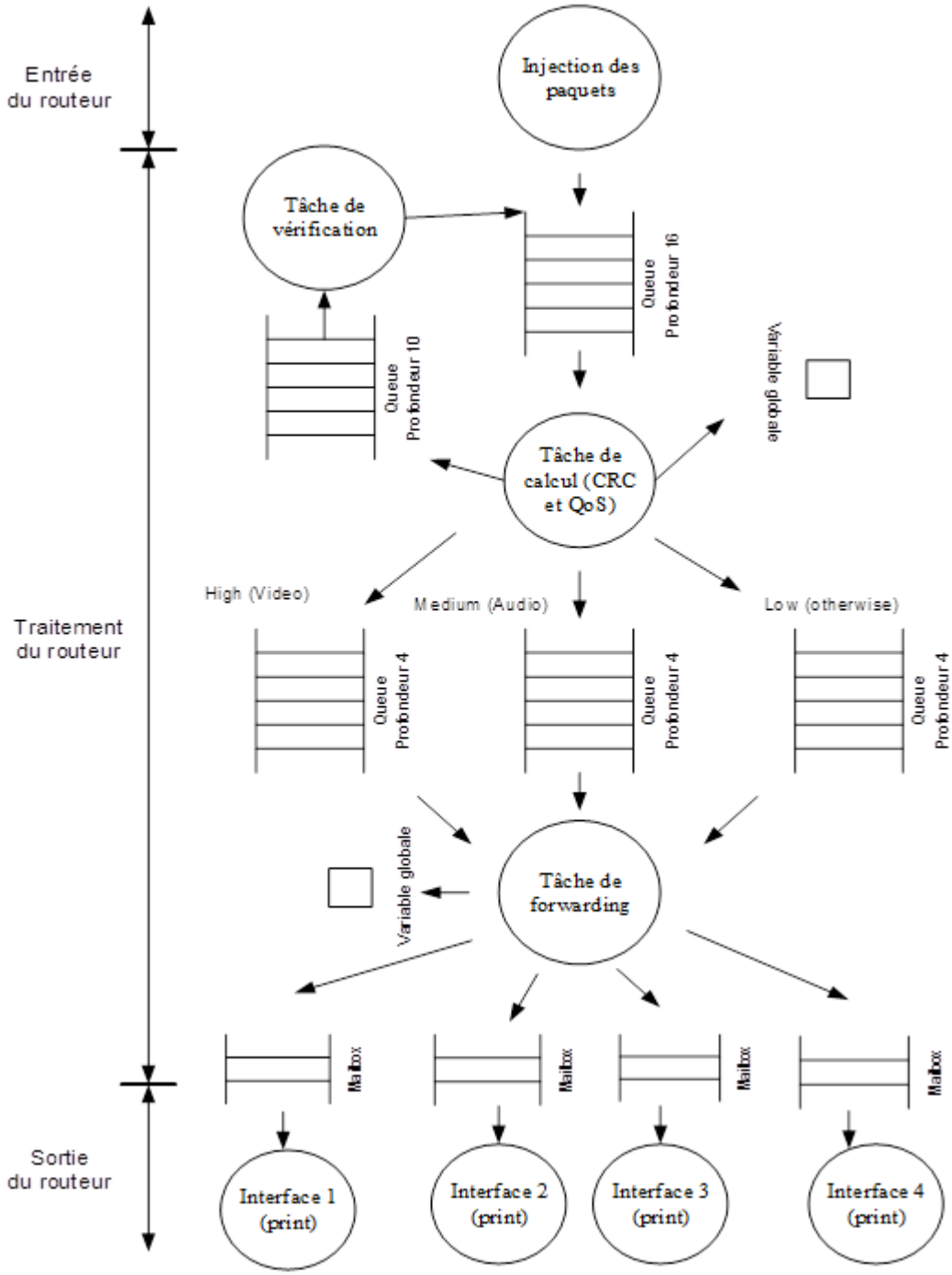


Figure 2 Flot de données dans le routeur

Voici une description des différentes tâches :

- **Tâche d'injection de paquets**

Cette tâche vous sera fournie. Pour simplifier les choses, un générateur de paquets placera des paquets dans une queue avec un débit fixe (soit 2 paquets par seconde). Toutefois, on peut imaginer cette tâche comme un ISR qui lit les paquets à une adresse de périphérique et les transferts dans une queue sur le processeur.

- **Tâche de calcul**

Cette tâche à concevoir réalise différentes opérations :

- D'abord, elle doit valider l'adresse source des paquets en rejetant les paquets qui ne sont pas dans son espace d'adressage. Pour ce faire, tous les paquets ayant une adresse source contenue entre REJECT_LOW et REJECT_HIGH de chaque plage pour les interfaces devront être rejetés. Ces valeurs vous seront fournies.
- Par la suite, le code CRC doit être vérifié. Les paquets contenant des erreurs sont simplement rejetés. Le nombre de paquets rejetés doivent être gardé dans une variable globale. La fonction implémentant le calcul du CRC vous sera fournie. Pour plus de renseignements sur le sujet, on vous suggère de lire : http://www.faqs.org/rfcs/rfc1071.html.
- Finalement, selon le type de paquets (vidéo, audio et autres, respectivement 1, 2 et 3), les paquets valides seront envoyés dans trois queues différentes. Si les queues sont pleines vous devrez rajouter les paquets dans une queue qui sera lu par la tâche *verification*. (Si cette queue est pleine aussi, vous devez détruire le paquet)

- **Tâche *forwarding***

Cette tâche à concevoir devra prendre les paquets des 3 différentes queues et les aiguiller aux bonnes interfaces (selon l'adresse de destination). La qualité de service à implanter est la suivante : les paquets de haute priorité doivent d'abord être traités. Si aucun paquet de haute priorité n'est prêt, alors un paquet de moyenne priorité peut être envoyé. Sinon, un paquet de basse priorité pourra être envoyé.

Généralement, la correspondance entre les interfaces et l'espace d'adressage est contenue dans une table de routage. Dans les routeurs complexes, cette table est souvent partagée par plusieurs processeurs et est réalisée en matériel (coprocesseur). Ici la table de routage à implanter est toute simple et consiste simplement à envoyer le premier quart des paquets à l'interface un, le deuxième à l'interface deux et ainsi de suite. Plus précisément, la table est la suivante :

0	≤	**Destination**	≤ 1073741823	→ ***Interface 4***
1073741824	≤	**Destination**	≤ 2147483647	→ ***Interface 3***
2147483648	≤	**Destination**	≤ 3221225472	→ ***Interface 2***
3221225473	≤	**Destination**	≤ 4294967295	→ **Interface 1**

Toutefois, afin de simuler l'accès à cette table, un wait (***OSTimeDly***) d'une seconde doit être ajouté.

Comme l'indique la figure 2, la communication via la tâche *forwarding* et l'interface se fera par un mailbox. Finalement, pour des fins de statistique, la tâche *forwarding* met à jour une variable globale du nom de *nbPacket* indiquant le nombre total de paquets traités et écrire sur les *DELs* la valeur du nbPacket.

- **Tâche *interface***

Chaque tâche interface s'occupera d'acheminer le paquet au prochain routeur. Ici, ces tâches ne feront qu'imprimer le contenu des paquets pour des fins de déverminage.

- **Tâche *verification***

La tâche *verification* va être responsable de la bonne retransmission des paquets. La tâche doit remettre dans la queue les paquets qui vont avoir été mises de côtés par la tâche calcul dû à une queue trop pleine (video, audio ou otherwise). Cette tâche s'exécutera périodiquement à toutes les cinq secondes grâce à une synchronisation avec l'interruption du deuxième fit_timer.

- **Tâche *stop***

Cette tâche n'est pas représentée dans la figure 2. La tâche *stop* va être responsable d'arrêter le routeur (utilisez la fonction *OSTaskDel*()). Le routeur doit être arrêté lorsque plus que 15 paquets vont avoir été rejetés à cause d'un mauvais CRC. Cette tâche va être réveillée (Synchronisation unilatérale par sémaphore) par une interruption périodique responsable de faire la vérification de la quantité de paquets rejetés à cause d'un mauvais CRC.

4. Travail à effectuer

4.1. Matériel

Vous devez ajouter deux minuteries (*fit_timer)* dans la plateforme matérielle fournie. De plus, vous devez la connecter au contrôleur d'interruption et les configurer. La première doit générer une interruption par seconde, alors que la deuxième génère une interruption aux cinq secondes.

4.2. Logiciel

Pour servir l'interruption générée par ces minuteries, vous devrez implémenter deux fonctions (*fonction handler*) qui seront appelées par une fonction de µC à chaque fois qu'une interruption sera générée. Vous devez modifier les fichiers suivants.

- **bsp.h** : déclaration des fonctions associées à l'interruption de la forme

  void VotreInterruption(void InstancePtr) ;*

- **bsp.c** : connexion de vos fonctions au contrôleur d'interruption et activation de l'interruption dans le contrôleur d'interruption
- **routeur.c** : définition de la fonction associée à l'interruption

Dans le fichier routeur.c, le *handler* du premier fit_timer, qui exécute une interruption par seconde, devra effectuer les opérations suivantes afin de traiter correctement l'interruption :

1. Lire le nombre de paquets qui a été rejeté à cause que le CRC était invalide et s'assurer que cette valeur est plus petite que 15.
2. Si cette quantité est plus grande ou égale à 15, la tâche logicielle *taskStop* doit être réveillée. (Synchronisation unilatérale par sémaphore). Si la quantité est plus petite que 15, vous ne faites aucun traitement particulier et l'interruption se poursuit simplement.

Toujours dans le fichier routeur.c, le handler du deuxième fit_timer, qui exécute une interruption toutes les cinq secondes, devra réveiller la tâche verification pour qu'elle exécute son code.

Finalement, vous devez implémenter la tâche de calcul (TaskComputing), la tâche *forwarding* (TaskForwarding), la tâche interface (TaskPrint), la tâche *verification* (TaskVerification) et la tâche *stop* (TaskStop). La tâche injection de paquet (TaskInjectPacket) et la fonction qui calcule le CRC vous seront fournies. La figure numéro 2 doit être **absolument** respectée lors de l'implémentation.

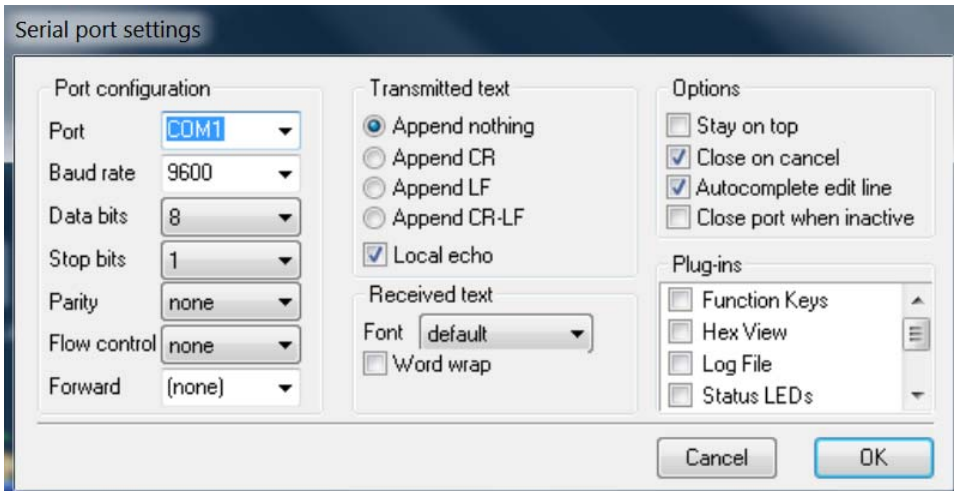
Les étapes à suivre sont :

1. Ouvrez le projet EDK qui vous a été donné
2. Ajoutez et configurez les *fit_timer*
3. Générez le *bitstream* et le téléchargez sur le FPGA
4. Allez dans *Project -> Export hardware design to SDK* et cliquez sur *Export & launch SDK*. Vous devez créer un répertoire de travail
5. Suivez le tutorial Tuto_edk_inf3990 à partir de **la page 10** pour créer votre projet. **Toutefois, vous devez choisir l'OS µC**.
6. Modifiez dans les settings du BSP les quatre configurations suivantes :
   - Task_Management : OS_TASK_DEL_EN à 1
   - Messages_Queues : OS_Q_ACCEPT_EN à 1
   - Miscellaneous : OS_MAX_EVENTS à 15 et OS_MAX_QS à 10
7. Copiez le code donné dans le répertoire « ***src*** **»** de votre projet
8. Faire les modifications nécessaires dans bsp.h et bsp.c
9. Implémentez les cinq tâches demandées dans routeur.c.

3. Autres précisions

a. Affichage de trace pour débogage sous µBlaze

Il est interdit d'utiliser le fameux **printf()**. En effet, le laboratoire porte sur la programmation dans un système embarqué. La mémoire sur FPGA étant limitée, il faut une taille de code petite. La fonction **printf()** classique prend *55 Ko* en mémoire. À la place, vous utiliserez **xil_printf().** Cette fonction se comporte exactement comme **printf()** mais ne prend que **2 Ko** de mémoire. De plus, pour effectuer du débogage, vous devez ouvrir un terminal (Termine Terminal est recommandé) avec les options suivantes :



4. Questions

Question 1

Le fichier BSP (Board Support Package) est essentiel au port de µC vers d'autres plates-formes. Quelle est l'utilité de ce fichier? Qu'est-ce qu'il nous permet de faire?

Question 2

Comme vous le savez maintenant, l'inversion de priorité est un problème très connu dans le domaine des applications temps réel. Avez-vous fait face à ce problème lors de ce laboratoire? Expliquez.

Question 3

À la dernière ligne de la tâche d'injection de paquets, la fonction OSTimeDly() est appelée avec la valeur 50 en paramètre, en vue d'attendre une demi-seconde. Expliquez comment la fonction OSTimeDly() fonctionne et expliquez pourquoi 50 est la bonne valeur à passer.

Question 4

Est-ce que l'horloge qui cadence la vitesse du processeur est le même qui détermine le ***tick*** de l'OS? Expliquez.

Question 5

Quel est l'intérêt d'avoir TaskVerification, au lieu de mettre le code directement dans fitTimerHandler2? Quel(s) seraient les avantage(s) et/ou désavantages d'effectuer de cette façon?

Question 6

Avec les chiffres qui vous sont donnés, quel peut être au mieux le débit des paquets traités? Afin d'accélérer ce débit, on vous demande d'augmenter la quantité de tâches *Forwarding.* Combien de copies de cette tâche avez-vous besoin au minimum?

5. Rapport

Vous devez écrire un rapport de maximum **10 pages** qui décrira le travail effectué. Ne pas oublier de répondre aux questions dans votre rapport.

Le contenu typique du rapport est le suivant :

1. Page titre
2. Barème de correction
3. Introduction
4. Fonctionnement du système (justifiez les valeurs de priorités pour vos tâches, les lectures bloquantes ou non bloquantes, les settings de l'OS que vous avez modifié, entre outres.)
5. Réponses aux questions
6. Conclusion
   - Indiquez comment vous avez apprécié ce laboratoire.
   - Indiquez également le nombre d'heures que vous avez pris pour réaliser ce laboratoire.

6. Procédure de remise

Vous devez remettre **un** fichier **.zip** ou **.rar** clairement identifié contenant un répertoire avec le code et un fichier **.doc** ou **.docx** avec le rapport. (SVP ne mettez pas le rapport dans un fichier .pdf). Le nom des fichiers doit contenir les matricules des deux étudiants et se présentera sous la forme :

Lab2_INF3610_A13_Matricule1_Matricule2.zip et Lab2_INF3610_ A13_Matricule1_Matricule2.docx

Avant de compresser votre travail, assurez-vous de bien débarrasser les objets encombrants pour ne pas encombrer la boîte aux lettres du correcteur. Ceci implique que vous devez envoyer seulement les fichiers .c et .h que vous avez modifiés.

Date limite de remise : 30 octobre 2013 à 23h59

Tout ceci devra être remis à jeff.falcon@polymtl.ca avant la date de remise.

7. Pénalités

Des pénalités s'appliqueront si vous ne remettez pas un travail de qualité. Voici un résumé des pénalités possibles :

Pénalités	
Retard	-1.5 pts/jour ouvrable, plus de 4 jours entraînent la note 0
Fichiers mal présentés et Orthographe	Jusqu'à -1.0pt
Opérations manuelles non prévues	Aucune opération non prévue ne sera acceptée. Assurez-vous que vos projets compilent sinon vous obtiendrez la note 0 pour votre exécution.
Remise	Fichier .zip ou .docx mal nommé, -1pt

8. 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

PÉNALITÉ	

NOTE GLOBALE	**/20**