École Polytechnique de Montréal

Laboratoire #2 : Routeur sur puce FPGA

INF3610 – Automne 2017

Frédéric Fortier
Eva Terriault

Département de Génie Informatique

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 ARM implanté sur une puce FPGA.

Plus précisément, les objectifs spécifiques du laboratoire sont :

- Approfondir ses connaissances de µC et du temps réel
- S'initier aux environnements de développement embarqués, tels Xilinx SDK
- Étudier les spécificités de la programmation pour SoC.
- 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. Finalement, un routeur peut aussi supporter une qualité de service (***QoS***) en triant et priorisant les paquets selon qu'il s'agit par exemple d'un paquet audio, vidéo ou encore contenant des données quelconques.

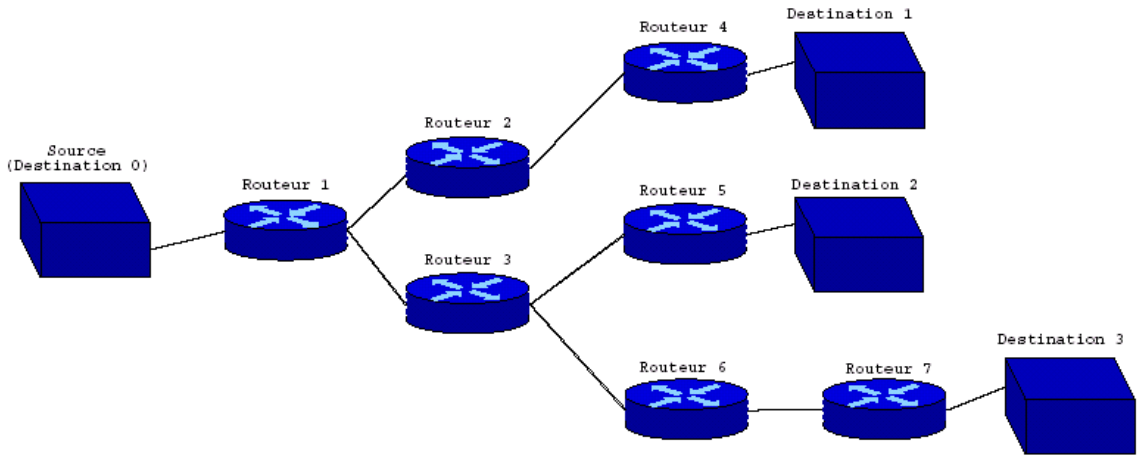

Figure 1 Exemple de Réseau téléinformatique

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	**Dest**	**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[1] et finalement les **données** transportées. La définition de cette structure vous sera fournie.

3. Plateforme matérielle

La flexibilité des puces multiprocesseurs configurables Zynq utilisées en laboratoire nous permet de les utiliser pour faire du multitâche synchrone ou asynchrone : tel que montré à la ***figure 2***, les deux processeurs de la puce se partagent les ressources du système et pourraient exécuter des systèmes d'exploitation différents, tels que Linux ou µC-OS (sur chacun un cœur différent), ou Linux sur les deux cœurs, par exemple. Le premier genre de configuration dit AMP (Asymmetric Multi-Processing) permet d'avoir un système d'exploitation plus général et offrant plus de fonctionnalités pour exécuter les tâches non critiques tout en gardant un processeur pouvant exécuter des tâches en temps réel pour les tâches devant garantir un temps de réaction maximal. Par contre, dans le deuxième genre de configuration, dit SMP (Symmetric Multi-Processing), un OS partagé contrôle tous les processeurs et permet l'exécution de n'importe quel tâche ou application sur n'importe lequel d'entre eux, facilitant la programmation du système au détriment de la flexibilité offerte par l'AMP. Nous verrons en classe plus en détails l'architecture de la Figure 2 et ces deux types de configuration.

Dans ce laboratoire, nous n'utiliserons qu'un seul processeur exécutant µC-OS.

[1] Pour plus d'informations : http://www.faqs.org/rfcs/rfc1071.html

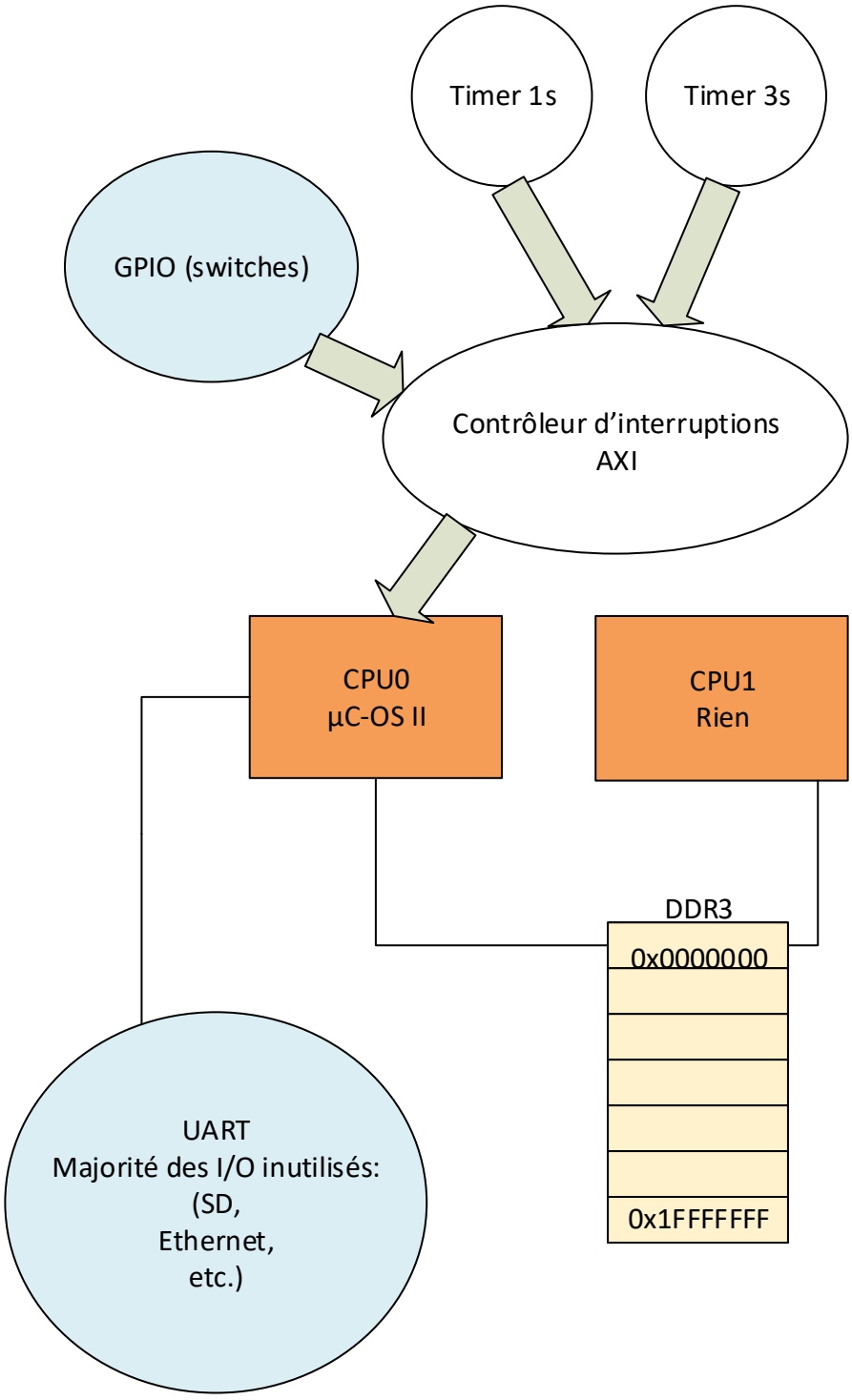

Figure 2 Configuration matérielle

4. Description des tâches

La ***figure 3*** illustre le flot des paquets à l'intérieur du routeur que vous devez implémenter sur le CPU µC-OS de la carte.

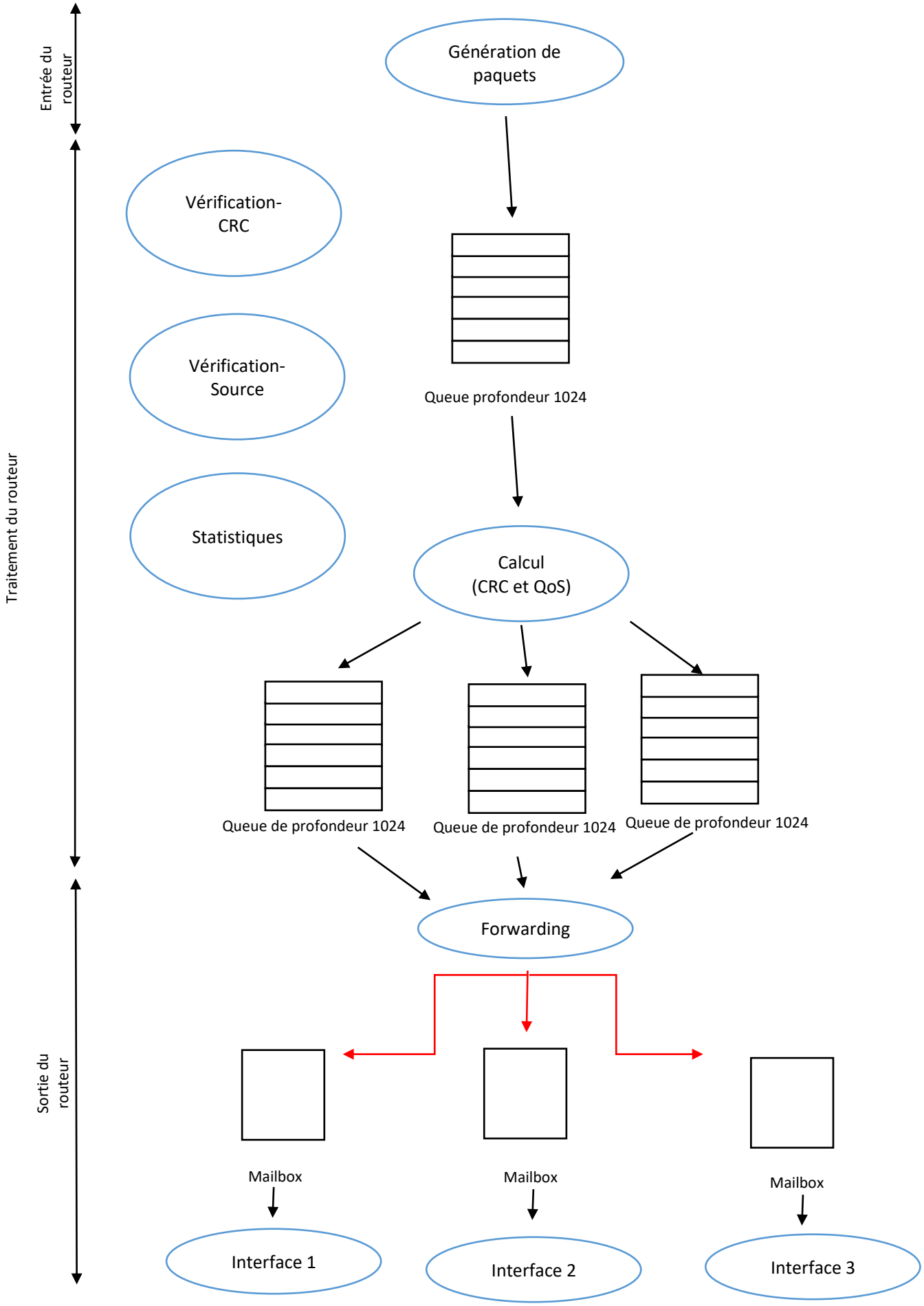

Figure 3 Flot de données dans le routeur

Voici une brève description des différentes tâches :

Génération des paquets

La tâche est fournie. Cette tâche s'occupe de générer des paquets aléatoires qui seront traités par le routeur. La tâche alterne entre deux modes, soit le mode génération et le mode attente. Chaque mode s'exécute en alternance pour une durée allant jusqu'à 0,5 sec.

Pendant le mode génération, les paquets sont générés avec des valeurs aléatoires pour la source, la destination, le type et les données. Chaque génération de paquet est entrecoupé d'un OSTimeDlyHMSM(0,0,0,2), afin de laisser les tâches à plus basse priorité de s'exécuter et ainsi vider les FIFOs.

Calcul

La tâche est à concevoir. Elle doit répondre à plusieurs critères :

- Dans un premier temps, elle doit valider la provenance des paquets. Ainsi, tous les paquets provenant d'une plage d'adresse à rejeter doivent être rejetés. Ces plages vous sont fournies (defines commençant par `REJECT_`).
- Une fois l'adresse vérifiée, il faut s'assurer que le paquet n'a pas été corrompu. Il faut donc tester le CRC. La fonction qui calcule le CRC vous est fournie.
- Ensuite, les paquets doivent être envoyés dans différentes files selon le type de paquets (vidéo, audio et autres resp. 0, 1 et 2). Si jamais cette file est pleine, vous devez détruire le paquet.

Afin de simuler un temps de traitement plus significatif, une attente active doit être effectuée à l'intérieur de la tâche de calcul. Pour ce faire, incrémenter un compteur jusqu'à une valeur d'environ 220000 pour chaque paquet traité.

Forwarding

La tâche est à concevoir et doit répondre à ces critères :

- La tâche doit lire dans les trois queues en respectant la priorité définie par cette qualité de service : les paquets vidéo sont plus prioritaires que les paquets audio qui sont eux-mêmes plus prioritaires que les autres paquets.
- La tâche doit ensuite lire l'adresse de destination du paquet dans une table de routage. Dans les routeurs complexes, cette table est souvent réalisée en matériel. Ici, nous simplifions cette étape : le premier quart des adresses correspond à un *broadcast* sur les 3 interfaces, le second quart correspond à l'interface 3, le troisième quart correspond à l'interface 2 et le dernier à l'interface 1.

- 0 <= **Destination** < 1073741823 -> interface 1
- 1073741824 <= **Destination** < 2147483647 -> interface 2
- 2147483648 <= **Destination** < 3221225472 -> interface 3
- 3221225473 <= **Destination** < 4294967295 -> BROADCAST

NB : Pour les paquets broadcastés, faites bien attention à allouer de l'espace pour les paquets nouvellement créés.

Interface (PRINT)

La tâche est à concevoir. Cette tâche peut représenter le périphérique d'arrivée ou un autre routeur. Ici, elle se contentera de lire les paquets et de les imprimer.

Vérification - source

La tâche est à concevoir. Cette tâche va se réveiller toute les secondes grâce à une synchronisation unilatérale avec l'interruption générée par le timer d'une seconde (*fit_timer_1s*). Elle a pour charge de vérifier le nombre de paquets rejetés pour cause de mauvaise source. Si le nombre est >= à 200, alors elle suspend toutes les tâches du pipeline du routeur (sauf la tâche statistique).

Vérification - CRC

La tâche est à concevoir. Cette tâche est réveillée de manière asynchrone grâce à une synchronisation unilatérale avec une interruption générée par le timer de 3 secondes (*fit_timer*_3s). Elle est identique à la tâche vérification – source, mais elle vérifie le nombre de paquets rejetés pour cause de mauvais CRC. Si le nombre est >= à 200, alors elle suspend toutes les tâches du pipeline du routeur (sauf la tâche statistique).

Statistiques

La tâche est à concevoir. Cette tâche est réveillée de manière asynchrone grâce à une synchronisation unilatérale avec avec l'interruption générée par la modification de l'état de n'importe laquelle des switches du Zedboard. Cette même interruption activera (et désactivera) le mode profilage (variable mode_profilage). L'utilisation attendue des switches sera alors d'en modifier une, attendre quelques secondes, puis d'en modifier une autre (ou la même). Ainsi, le mode profilage sera activé pendant quelques secondes.

Lorsque le mode profilage est activé, la tâche statistique doit calculer des statistiques par rapport aux différentes files utilisées dans le routeur. Plus précisément, il sera question de calculer l'utilisation moyenne de chaque file ainsi que la valeur maximale de leur taille durant la période de profilage. Vous devrez utiliser des fonctions de uC-OS afin d'obtenir des informations sur vos files.

À la fin de la période de profilage (soit quand nous quittons le mode), la tache va afficher les informations calculée (max et moyenne pour chaque file), ainsi que les informations des paquets traités au moment de son appel (nb paquets traités, nb paquets créés et nb de paquets rejetés pour mauvaise source).

Note 1 : Attention à protéger les *xil_printf()* avec des exclusions mutuelles car cette tâche peut se déclencher n'importe quand.

Note 2 : Un mutex doit aussi être utilisé si plus d'une tâche (incluant la tâche de génération de paquets fournie) fait appel à la fonction malloc et/ou free. En effet, l'implémentation de ces deux fonctions suppose une configuration *bare metal* et n'est donc pas *thread-safe*.

Note 3 : D'autres mutex peuvent être nécessaires pour la protection de diverses variables. À vous d'en juger de la pertinence.

4. Travail à effectuer – partie 1 : Implémentation du routeur

4.1 Matériel

Suivez le tutoriel *Création d'un projet Vivado.pdf* joint avec cet énoncé pour la création de la plateforme matérielle avec Vivado, et la création du BSP (*Board Support Package*) sur Xilinx SDK.

4.2 Logiciel

Notez que bien que cette partie décrive le travail logiciel à effectuer, la lecture de la section 5 de cet énoncé **avant** de commencer à l'implémenter est essentielle.

Assurez-vous d'avoir lu le documents *INF3610-Lab2-Complements.pdf* afin de commencer l'implémentation.

4.2.1 Les interruptions

Pour servir les interruptions générées par les minuteries (*fit_timer* de 1 sec et 3 sec) et les switches, vous devrez implémenter des fonctions *handler* [2] qui seront appelées à chaque fois qu'une interruption sera générée. Vous avez à modifier les fichiers suivants :

- Bsp_init.h :
    - Ajout de la déclaration de vos fonctions de connexion et de déconnexion d'interruption.

Remarquez également la déclaration des *handlers* des interruptions ayant la signature suivante:

```
void VotreInterruption(void* InstancePtr) ;
```

[2] En classe on utilise le terme *myISR*, c'est la partie propre à l'ISR. En d'autres termes, les *handlers* d'interruption peuvent être vus comme des *callbacks* exécutés lors d'une interruption.

- Bsp_init.c :
    - Implémentation de vos fonctions de connexion et déconnexion d'interruption ;
    - Appel des fonctions de connexion et déconnexion aux endroits pertinents ;
    - Implémentation de la fonction initialize_gpio(), en faisant les appels nécessaires pour activer l'interruption générée par les switches. Voir la section *Compléments* pour plus d'informations sur le driver GPIO.
- routeur.c :
    - Définition des fonctions *handlers* associées à vos interruptions des *fit_timer* et des switches. Ces interruptions génèrent des synchronisations unilatérales par sémaphore avec les tâches arrêt de service, statistiques et reprise du service.

4.2.2 Tâches du routeur

Ensuite, vous devez implémenter les tâches décrites plus haut. La figure 3 **doit** être respectée lors de l'implémentation. La tâche de génération des paquets par phase vous est fournie. Notez la présence de la fonction *shouldSlowThingsDown* dans cette tâche, qui, si mise à *true*, ralentira significativement la génération de paquet pour vous aider à concevoir le système. Par contre, la correction sera effectuée avec cette variable à *false*.

Vous ne pouvez pas modifier la priorité des tâches. Cependant, si vous allouez des mutex, vous devrez définir leur priorité en conséquence.

En plus de répondre au cahier des charges des tâches citées plus haut, votre code doit gérer de manière sommaire les cas d'erreurs ainsi que protéger les variables partagées s'il y a besoin ainsi que, au besoin, les appels à malloc/free ou à xil_printf. Notez que cette protection doit aussi s'appliquer au code fourni avec cet énoncé (par exemple non exclusif à la tâche de génération de paquets).

Voici un déroulement proposé :

1. Ne créez que les tâches liées aux interruptions des *fit_timer* ainsi que les *handlers* associés
2. Connectez vos *handlers* aux interruptions comme décrit dans la section 4.2.1
3. Testez vos interruptions (Est-ce que les interruptions fonctionnent ? Est-ce que les tâches associées démarrent ?)
4. Codez les tâches du flot principal (*computing*, *forwarding* et *print*), en faisant bon usage des éléments de synchronisation et de communication. Utilisez des files de taille suffisamment grande (1024).
5. Testez le flot principal. Assurez-vous que les paquets sont reçus par la bonne interface et que la priorité des paquets est respectée.

6. Ajoutez le code des tâches restantes (arrêt et reprise du service)
7. Rajoutez les interruptions GPIO et le code de la tâche statistique.

4.3 Affichage de trace pour débogage sous ARM

Il est interdit d'utiliser le fameux **printf()**. En effet, le laboratoire porte sur la programmation d'un système embarqué. La mémoire sur FPGA étant limitée, il faut minimiser la taille du code. 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.

5. Travail à effectuer – partie 2

À venir.

6. Rapport

Vous avez à écrire un rapport contenant les réponses aux questions ainsi que, au besoin, les points de votre implémentation qui vous semblent importants. Concernant les explications sur votre implémentation, soyez brefs, il s'agit simplement pour nous de comprendre votre démarche si certains points vous paraissent obscurs. Vous terminerez votre rapport par une brève critique du laboratoire ainsi que le nombre d'heures passées sur celui-ci.

7. Procédure de remise

Vous devez remettre à la fin de ce laboratoire un fichier .zip contenant un répertoire avec le code et un fichier PDF avec le rapport. Veuillez mettre dans le nom du rapport et de l'archive le texte suivant : « Lab2_H18_matricule1_matricule2 ». Les fichiers source à remettre sont les suivants :

- routeur.c/h
- bsp_init.c/h

La plateforme matérielle que vous avez créée n'est pas nécessaire.

Barème	
Exécution du code	
Fonctionnalités	/10
Qualité du code	/4
Réponse aux questions (partie 2)	

À venir	/6
Le non-respect des consignes entraine des points négatifs (peut aussi invalider les points d'un exercice)	
TOTAL	/20

8. Pénalités

Pénalités	
Retard	-4 points par jour ouvrable, note 0 si 4 jours de retard
Fichiers mal présentés ou orthographiés	Jusqu'à -1 point
Source non compilable	Assurez-vous que votre projet compile, sinon vous obtiendrez la note 0 pour l'exécution
Remise	Fichiers mal nommés = -1 point