Polytechnique de Montréal - DGIGL

Laboratoire #1 : Routeur sur puce FPGA

INF3610 – Automne 2024

Séance no 2 : Programmation d'interruptions

1. Objectifs de la séance no 2 et suggestion de préparation

Nous avons vu dans l'énoncé du laboratoire de la séance (ou partie) no 1, les objectifs généraux du lab 1, la mise en contexte et le fonctionnement du routeur, auquel vous avez modifié la fonctionnalité pour finalement réaliser un test de performance.

L'**objectif** de cette 2e séance est de se concentrer sur la couche bas niveau (*firmware*) du routeur dans laquelle vous allez programmer les ISR pour le *Fit Timer*, le *GPIO* pour ajouter de nouvelles fonctionnalités à votre code de la partie 1.

Recommandation importante : le video du no 34 des exercices (sur Moodle) peu être une excellente préparation à ce laboratoire. Le video est 3 parties :

1) Il présente d'abord la solution au no 34 à l'aide de la figure 34.1 des exercices du chap. 2 qui contient une interruption (0 è 6.00 sec du video),
2) Il présente ensuite à nouveau 1 mais plutôt que suivre le tableau 34.1 il le fait en exécutant pas à pas le code disponible sur Moodle (Ex_no34) (de 6.00 à 17 :47)
3) Il se concentre sur les étapes 4 à 6 de la figure 34.1 i.e. le moment ou on appuis sur le bouton pressoir de la carte et le moment ou l'interruption débute (fonction *gpio_isr*). Autrement dit j'explique le passage dans le noyau de uC/OS-III et les appels aux fonctions (drivers) de la page 58 du bloc 5 vu en classe.

Remarques :

i) Le video utilisait une carte Cora Z7 mais le comportement est identique avec la Pynq Z2 quand on appuie sur le bouton pressoir (p.e. BP0).

ii) Dans Ex_no34 l'ISR *gpio_isr* fait un rendez-vous unilatéral avec une tâche uC via un sémaphore *Sem*. Dans le laboratoire que vous devez compléter, on aura des rendez-vous similaire entre par exemple *fit_timerisr0* et *TaskStop* mais via *OSFlagPost* et *OSFlagPend* respectivement.

2. Vue d'ensemble du fonctionnement via ses principaux états

Imaginez un instant que votre application de routeur est embarquée dans un système et qu'on souhaite faire des tests sur les performances d'utiliser ou pas, une section critique (p.e. entre le traitement des packets de type VIDEO et AUTRE), mais sans avoir à regénérer un nouveau *bitstream* et donc en utilisant uniquement les périphériques de la carte.

La figure 1 illustre le fonctionnement du système qu'on souhaite réaliser dans cette partie 2 sous la forme d'une machine à états. **Pour une introduction au bouton pressoir (BP) et switch (SW) lire la Section 3.**

Pour obtenir ce fonctionnement, vous devrez plus loin (Section 4) modifier les **3 tâches *TaskReset*, *TaskStop*** et **TaskComputing** et ajouter une nouvelle tâche **TaskClearFifo** ainsi que **3 ISRs** (*gpio_isr1, gpio_isr2* et *fit_timer~~2~~ isr0*).

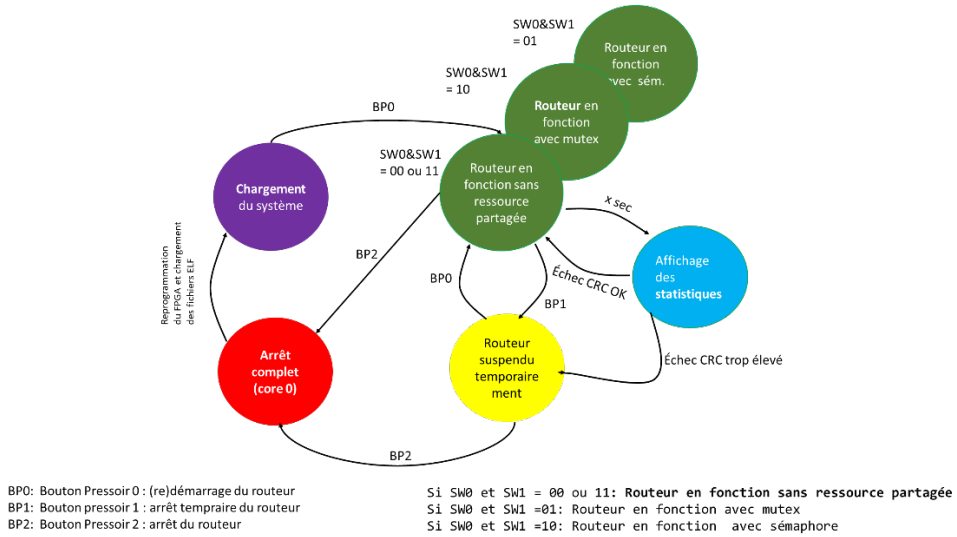

Figure 1. États du routeur pour la partie 2 – N.B. les 2 états verts *Routeur en fonction avec mutex* et *Routeur en fonction avec sémaphores* ont les mêmes arcs allant ou venant vers les états jaunes, rouges et bleus que *Routeur en fonction sans ressources partagées*

Dans ce qui suit nous donnons une explication générale du système et reviendrons sur les détails à la section 5 lors des 4 manipulations.

2.1) Initialisation et démarrage du système et reprise de service (sera assuré par l'*ISR gpio_isr 0* et *TaskReset*)

Au départ, vous devrez concevoir une sous-routine d'interruption ISR (nommé *gpio_isr0*) liée au bouton pressoir (en anglais *press button*) qui lorsque le bouton est activé (pressée) va démarrer le système[1] avec BP0 (au départ l'état vert à la figure 1 dans lequel le routeur est en fonction sans aucune

[1] Inspirez de la tâche StartUpTask de la partie 1…

ressource partagée[2]). Ce démarrage se fera une première fois (juste après l'état *Chargement du système*), mais aussi plus loin quand la tâche *TaskStop* va suspendre le système temporairement (état *Système suspendu*) avec BP1 ou suite à trop grand nombre de mauvais CRC. Que ce soit pour démarrer ou redémarrer après une suspension le système (routeur), vous utiliserez le bouton 0 (BP0).

Finalement, à partir de l'état *Initialisation et démarrage* du système vous pourrez également appuyer sur le bouton 2 (BP2) afin de mettre fin à l'exécution du système. Dans ce dernier cas, on passe alors à l'état *Arrêt complet*. Le contrôle devra alors revenir à *StartupTask* et se suspendre et afficher.

2.2) Passage à l'état suspendu temporaire

Deux possibilités peuvent vous amener à l'état suspendu temporaire :

2.2.1) Vérification du nombre de paquets rejetés à cause d'un trop grand nombre de mauvais CRC (sera assuré par *fittimer_isr1 et TaskStop*)

Cet état permettra de faire une vérification du système et suspendre ce dernier en bloquant les tâches *TaskGenerate*, *TaskComputing*, *TaskFowarding*, *TaskOutputPort* et *TaskStats* après qu'un certain nombre de paquets rejetés ont été rencontrés à cause d'un mauvais CRC (variables *nbPacketMauvaisCRC*). Plus précisément, ces interruptions, via le ISR de *fit_timer1*, que vous programmerez pour émettre des interruptions à toutes les x secondes (x donné à la section 4), vont à chaque fois réaliser un rendez-vous bilatéral avec *TaskStop.* Cette dernière va alors afficher les statistiques (*TaskStats*) et regarder si le *nbPacketCRCRejete* a dépassé un certain nombre de paquets (p.e. 2000). Dans l'affirmative, *TaskStop* va stopper les tâches du système (pour ainsi bloquer à nouveau *TaskGenerate*, *TaskComputing*, *TaskFowarding*, *TaskOutputPort* et *TaskStats*). On tombe alors dans l'état *Système suspendu* (en jaune sur Fig. 1) de la figure 2. Sinon (cas avec moins de 2000 paquets rejetés), on revient à l'état *Initialisation et démarrage du système*.

2.2.2) Arrêt temporaire du système demandé par via les boutons pressoir BP1 (sera assuré par *gpio_isr 0 et TaskStop*)

Votre sous-routine d'interruption ISR *gpio_isr0* liée au bouton pressoir devra aussi supporter la fonctionnalité suivante : tel que mentionné en 2.1, lorsque le bouton BP1 est activé (pressée), il devra réaliser un rendez-vous bilatéral avec la tâche *TaskStop*, qui comme pour la section 2.2.1, va stopper les tâches du système (pour ainsi bloquer à nouveau *TaskGenerate*, *TaskComputing*, *TaskFowarding*, *TaskOutputPort* et *TaskStats*). On tombe alors dans l'état *Système suspendu* (en jaune sur Fig. 1) de la figure 2.

[2] On parle ici de la ressource partagée entre le traitement du packet VIDEO et et packet AUTRE.

Finalement, quand on passe à cet état suspendu (que ce soit selon 2.2.1 ou 2.2.2), 2 possibilités pourront par la suite survenir:

1) vous appuyez sur le bouton 0 (BP0) afin de repartir le système et vous retournez alors à l'état *Initialisation et démarrage du système* (section 2.1), ou
2) vous pourriez également appuyer sur le bouton 2 (BP2) afin de mettre fin à l'exécution du système. Dans ce dernier cas, on passe alors à l'état *Arrêt complet*. Le contrôle devra alors revenir à *StartupTask* pour préparer la fin. Encore une fois, nous décrivons plus bas (section 5) ce qui sera exactement demandé pour cette fin.

Retenez bien que pour reconnaitre l'**état du système** à la figure 1, on utilisera les variable globale *routerIsOn* et *routerIsOnPause* :

- lorsque *routerIsOn* == 1 et *routerIsOnPause* = 0 indique qu'on est dans un fonctionnement normal (états vert et état bleu Fig. 1) ,
- Lorsque *routerIsOn* est à 0 le système est en arrêt complet (état rouge Fig. 1) et ce peu importe la valeur de *routerIsOnPause*,
- Pour reconnaitre l'état suspendu temporaire à cause de trop de paquets CRC rejetés ou lorsque BP1 est actif (bouton jaune sur Fig. 1) on aura *routerIsOnPause* = 1 avec *routerIsOn* = 1.

2.3) Passage entre les 3 états de fonctionnements (sera assuré par l'*ISR gpio_isr 1* et *TaskClearFifo*)

Tel qu'indiqué précédemment, on souhaite faire des tests sur les performances d'utiliser ou pas, une section critique (p.e. entre le traitement des packets de type VIDEO et AUTRE), mais sans avoir à regénérer un nouveau *bitstream* et donc en utilisant uniquement les switch de la carte pour nous permettrons de passer d'un des 3 états verts à un autre (Fig. 1).

Retenez donc qu'on aura donc 3 variables pour représentés les 3 états de fonctionnement (en vert sur Fig. 1) et donc la configuration de *TaskComputing* :

- No_CS 00 ou 11 // pas de sections critiques
- CS_Mutex 01 // utilisation du mutex
- CS_Semaphore 10 // utilisation du sémaphore

Cet état courant sera préservé dans *Status_TaskComputing* ainsi que l'état précédent *Prev_Status_TaskComputing* (avant dernière valeur de la position des switch). Ces états nous seront

utiles pour ré-initialiser les fifos et les pires temps d'exécution suite au passage d'un des 3 états vers un autre.

En résumé, à chaque fois que l'on modifiera la configuration de la switch, l'interruption *gpio_isr 1* devra mettre à jour le nouvel état et le précédent, puis donnez rendez-vous à *TaskClearFifo* pour qu'elle effectue correctement le changement. Il faudra aussi évidemment modifier *TaskComputing* pour effectuer la bonne configuration (avec ou pas de sections critiques et dans l'affirmative mutex ou sémaphore).

3. À propos des boutons pressoirs

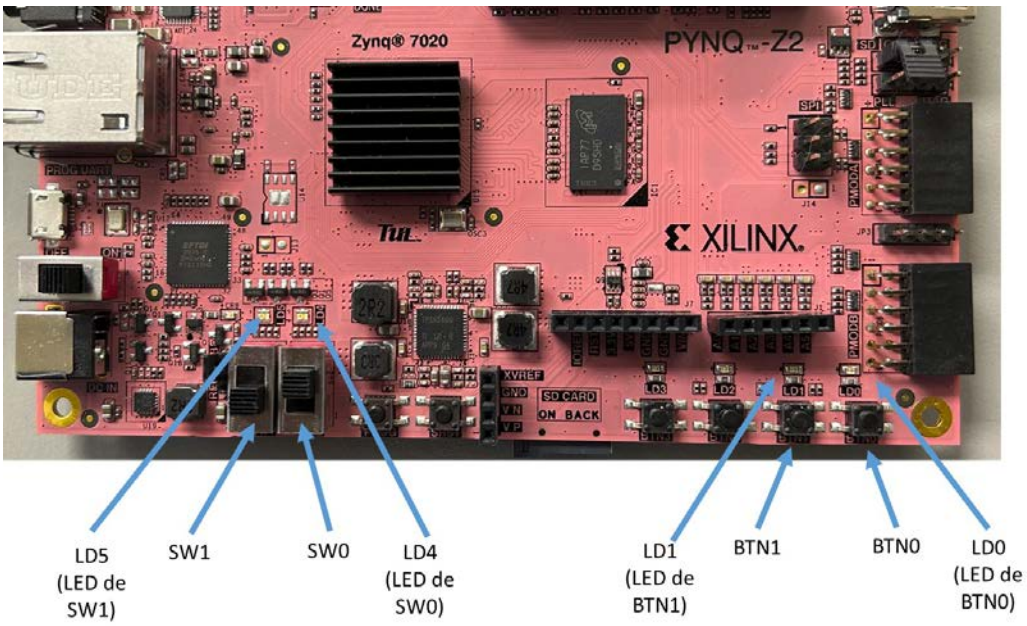

Figure 2 Emplacements des boutons et des switchs sur votre carte

Vous devez donc réserver 3 boutons (BTN0, BTN1 et BTN2) pour déterminer le fonctionnement. Vous utiliserez aussi les *switchs* (SW0 et SW1).

La figure 2 illustre ces boutons sur la PYNQ-Z2.

À chaque fois que le système (re)démarre, le LED correspondant c.-à-d. *LED0* devra s'afficher pour indiquer que le système (re)-démarre. Même chose avec le LED1 quand on décide de terminer.

Tel qu'illustré sur la figure 3, on peut lire le numéro du bouton sur lequel on a appuyé via *XGpio_DiscreteRead* (en spécifiant le canal 1) alors qu'on peut allumer le LED correspondant à un bouton par *Xgpio_DiscreteWrite* (en spécifiant le canal 2). Plus précisément :

button_data = Xgpio_DiscreteRead(&gpButton, 1); //get button data from

Xgpio_DiscreteWrite(&gpButton, 2, button_data); //write switch data to the corresponding LEDs

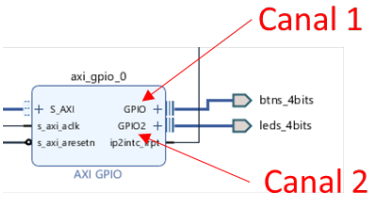

Figure 3 Un canal qui s'occupe de lire le bouton et un canal qui s'occupe d'allumer le LED correspondant

De manière similaire, on peut lire le numéro de la switch sur lequel on a appuyé via *XGpio_DiscreteRead* alors qu'on peut allumer le LED correspondant à une switch par *Xgpio_DiscreteWrite*. Plus précisément :

```
button_data = Xgpio_DiscreteRead(&gpSwitch, 1);  //get button data
Xgpio_DiscreteWrite(&gpSwitch, 2, button_data); //write switch data to the corresponding LEDs
```

4 Préparation de votre design Vivado et fichier routeur.h et routeur.c

4.1 Modifiez votre schéma Vivado

Tel que montrez à la figure 4 (page suivante), ajoutez dans votre schéma courant 1 gpio pour la switch et 1 fittimer pour une période de 13.000005 secondes (valeur de x sur Fig. 1). Puis, générez le *bitstream*, faites le *export* et *lauch SDK*. Votre lab 1 partie 1 va alors être mise à jour.

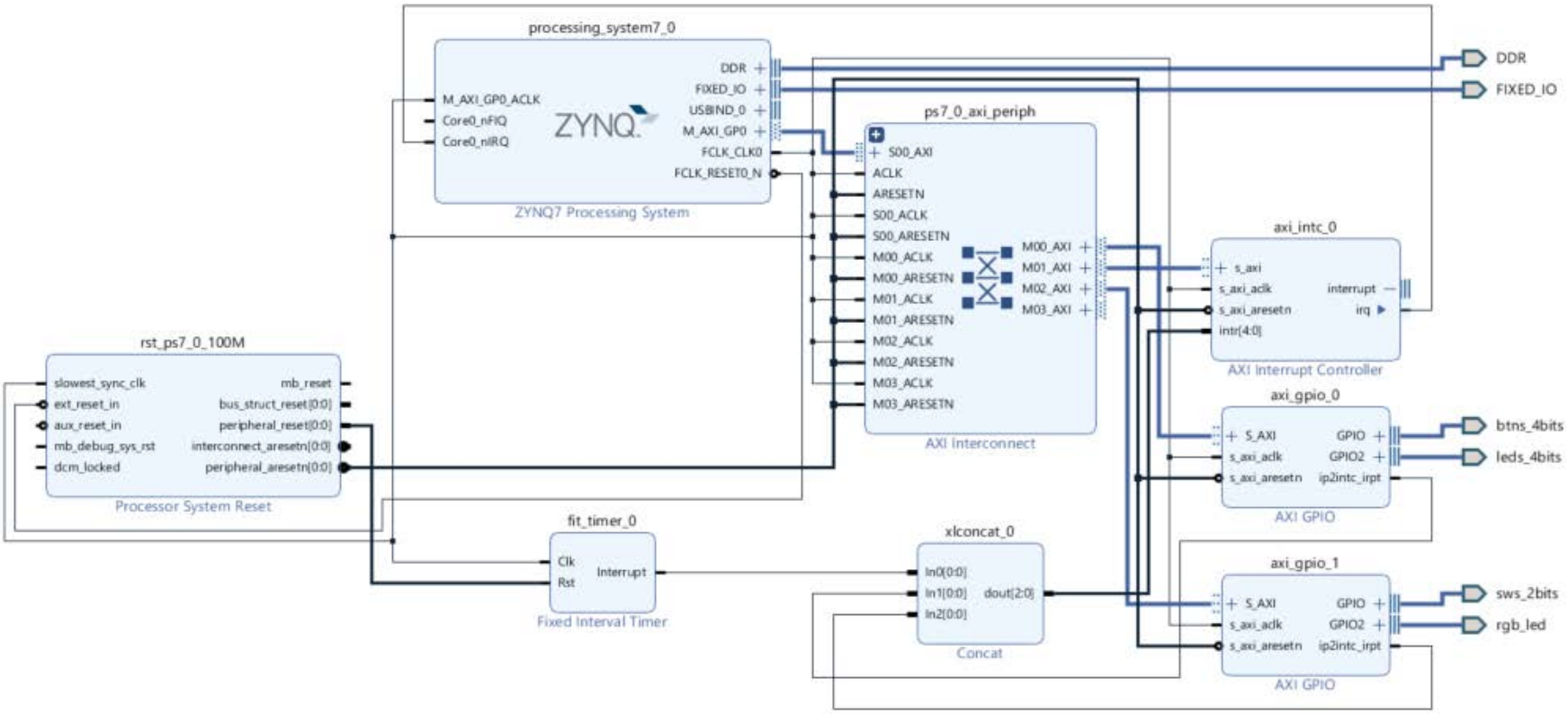

Figure 4. Mise à jour du schéma Vivado

4.2 Réorganisation des .h et ajouts de *interrupts.h* et *interrupts.c pour compiler sans erreur*

4.2.1 Fonctions pilotes (drivers) fournies

Les 15 fonctions suivantes seront données dans un code de départ sous le nom d'interruptions.c et interruptions.h que vous déposerez avec votre routeur.c et routeur.h. Prenez le temps de comprendre leurs rôles.

```
61 /* ************************************************
62  *              INTERRUPTION PROTOTYPES
63  ************************************************/
64
65 void fit_timer_isr0(void *p_int_arg, CPU_INT32U source_cpu);
66 void gpio_isr0(void *p_int_arg, CPU_INT32U source_cpu);
67 void gpio_isr1(void *p_int_arg, CPU_INT32U source_cpu);
68
69 void initialize_gpio0();
70 void initialize_gpio1();
71
72 int initialize_axi_intc();
73
74 int connect_fit_timer_irq0();
75
76 int connect_gpio_irq0();
77 int connect_gpio_irq1();
78
79 void connect_axi();
80
81 void cleanup();
82
83 void disconnect_intc_irq();
84
85 void disconnect_fit_timer_irq0();
```

Vous devrez compléter les 3 premières fonctions dans routeur.c juste avant TaskGenerate (ça va faciliter le travail pour la correction du chargé !). Également, vous trouverez de la documentation sur les fonctions des lignes 69 è 85 en cliquant sous BSP Documentation (p. ex. , dans *gpio_v4_3* et *intc_v3_8*) juste en dessous de *ucos_bsp_0* (fenêtre de gauche dans SDK).

4.2.2 Changements dans routeur.h et routeur.c

- Créer d'abord une nouvelle application nommée lab1_part2 (avec le hello world uC/OS-III)
- Détruire le fichier *app.c*
- Copier les fichiers *routeur.c* et *routeur.h* de la partie 1
- Importer les fichiers *interruptions.c* et *interruptions.h* du code de départ
- Faites les 9 étapes suivantes :
    1) Dans *routeur.h*, ajouter les 8 define suivants (expliqués dans les section 2 et 5 de l'énoncé) juste après ceux liés aux tâches dans le code de départ:

```
95 #define TASK_RESET_RDY          0x20     // RDV entre gpio_isr0 et TaskReset
96 #define TASK_SHUTDOWN           0x40     // RDV entre gpio_isr0 et StartUp
97 #define TASK_STOP_RDY           0x80     // RDV entre fittimer0 et TaskStop pour
98 #define TASK_STATS_PRINT        0x100    // RDV entre TaskStop et TaskStats
99 #define TASK_CLEAR_FIFO_RDY     0x200    // RDV entre gpio_isr1 et TaskClearFifo
100
101 #define CS_Mutex                SWITCH0
102 #define CS_Semaphore            SWITCH1
103 #define No_CS                   0
```

Faites simplement un copie/coller avec le fichier *masque_lies_aux_ISRs.c* du code de départ.

Aussi, si vous avez un #define FlagTaskComputing 0x8000 vous pouvez le détruire.

2) Couper le bloc suivant dans *routeur.h* :

```
30 /*DECLARATION DES COMPTEURS POUR STATISTIQUE
31 int nbPacketCrees = 0;
32 int nbPacketTraites = 0;
33 int nbPacketRejetes = 0;
34 int nbPacketRejetesTotal = 0;
35 int nbPacketFIFOpleine = 0;
36 int nbPacketFIFOpleineTotal = 0;
37 int nbPacketMauvaiseSource = 0;
38 int nbPacketMauvaiseSourceTotal = 0;
39 int nbPacketMauvaisCRC =0;
40 int nbPacketMauvaisCRCTotal =0;
41 int nbPacketMauvaisePriorite =0;
42 int nbPacketMauvaisePrioriteTotal = 0;
43
44 int delai_pour_vider_les_fifos_sec = 0;
45 int delai_pour_vider_les_fifos_msec = 128;
46 int print_paquets_rejetes = 0;
47 int limite_de_paquets= 500000;
```

puis transférer le dans *routeur.c* juste avant la macro de *safeprint*

N.B. Pour simuler plus rapidement vous pouvez diminuer delai_pour_vider_les_fifos_msec à la valeur de trouver à la section 5.2.3.1.

3) Ajouter à ce bloc l'initialisation les 4 variables suivantes qui indiquera si le système est en fonction ou pas, s'il est temporairement suspendu et donne le statut des tâches périodiques TaskComputing (référé à la section 2 de l'énoncé pour explication)

```
49 int routerIsOn = 0;
50 int routerIsOnPause = 0;
51 int Status_TaskComputing = 0;
52 int Prev_Status_TaskComputing = 0;
```

4) Détruire les lignes suivantes de routeur.h :

```
11 #include <os.h>
12 #include <stdlib.h>
13 #include <inttypes.h>
14 #include <stdbool.h>
```

5) Couper le bloc suivant dans routeur.c :

```
19 #include  <cpu.h>
20 #include  <lib_mem.h>
21
22 #include <os.h>
23 #include <stdlib.h>
24 #include <inttypes.h>
25 #include <stdbool.h>
26
27 #include <app_cfg.h>
28 #include <cpu.h>
29 #include <ucos_bsp.h>
30 #include <ucos_int.h>
31 #include <xparameters.h>
32 #include <KAL/kal.h>
33
34 #include <xil_printf.h>
35
36 #include  <stdio.h>
37 #include  <ucos_bsp.h>
38
39 #include <Source/os.h>
40 #include <os_cfg_app.h>
41
42 #include <xgpio.h>
43 #include <xintc.h>
44 #include <xil_exception.h>
```

Puis transférer le au tout début de *routeur.h* c'est-à-dire juste après:

```
2* * router.h
7
8 #ifndef SRC_ROUTEUR_H_EXT_
9 #define SRC_ROUTEUR_H_EXT_
```

6) Dans *routeur.c* ajouter un #include "interrupts.h"
7) Finalement, dans router.c créez les fonctions ISRs *gpio_isr0*, *fit_timer_isr0* et *gpio_isr1* (juste avant TaskGenerate). Puis dans chacun de ces ISRs, et pour des raisons de débogages uniquement[3], mettez pour le moment uniquement un *xil_printf*:

```
xil_printf("---------------gpio_isr0---------------\n");    // dans l'ISR gpio_isr0
xil_printf("---------------gpio_isr1---------------\n");    // dans l'ISR gpio_isr1
xil_printf("---------------fit_timer_isr0---------------\n");  // dans l'ISR fit__isr0
```

Puis ajouter XGpio_InterruptClear(&gpButton, XGPIO_IR_MASK); et XGpio_InterruptClear(&gpSwitch, XGPIO_IR_MASK); dans gpio_isr0 et gpio_isr1 resp. Sinon ça va partir en boucle car l'interruption n'est jamais remise à 0.

8) Dans la fonction startup, vous devrez appeler 4 fonctions de drivers afin d'initialiser les différents périphériques juste après UCOS_IntInit();:

[3] Une fois ce débogage terminer on doit enlever ces xil_printf car il amène une composante non déterminisme non souhaité dans un ISR.

```
1078    UCOS_Print("Programme initialise - \r\n");
1079
1080    initialize_gpio0();
1081    initialize_gpio1();
1082    initialize_axi_intc();
1083    connect_axi();
```

À partir de là ça devrait compiler sans erreur et vous devriez pouvoir exécuter votre routeur avec la fonctionnalité de la partie 1. *TaskStats* sera encore ordonnancer par *OSTimeDlyHMSM(0, 0, 30, 0, OS_OPT_TIME_HMSM_STRICT, &err);* et vous verrez aussi afficher le print de *fit_timer_isr0* et à la fréquence de 13.00005 sec le print de *fit_timer_isr1*. De même, quand vous appuyez sur un des boutons pressoir vous verrez afficher le print de *gpio_isr0* et finalement quand vous bougerez une switch, le print de *gpio_isr1*.

Dans ce qui suit, on va y aller étape par étape. On va d'abord mettre en service le bouton pressoir pour démarrer le système c'est-à-dire le compléter le code de ISRs *gpio_isr0*.

5. Quatre manipulations à compléter

5.1 Manip 1 : ISR *gpio_isr0* et *tâche StartUpTask*

Une fois la manip 1 compléter mettez **en commentaire le code de TaskReset car il ne servira plus**.

Le démarrage se fera directement à partir du ISR quand BP0 du bouton pressoir est activé.

Pour assurer le démarrage de votre système, voici le pseudo-code que vous devez compléter :

ISR gpio_isr0 (à mettre juste avant TaskGenerate dans routeur.c)

```
void gpio_isr0(void *p_int_arg, CPU_INT32U source_cpu) {

   CPU_TS ts;
   OS_ERR err;
   OS_FLAGS  flags;
   int button_data = 0;

   Lecture du bouton et affichage du led correspondant avec button_data
   (N.B. Inspirez-vous du no 34 (fig. 5 plus bas) et du vidéo pour le
         DiscreteRead et DiscreteWrite, mais aussi pour comprendre le
         fonctionnement des drivers pour ISR avec Xilinx sur Pynq Z2)
        Référez aussi à interrupts.h

   xil_printf("---------------gpio_isr0---------------\n");

   On regarde quel bouton a été activé (BP0 ou BP1)[4] en le mettant dans
   button_data

   Si button_data == BP0:
       1) Configurer le système en mode fonctionnement normal (en vert sur
          Fig. 1). Aussi, par défaut on part avec No_CS (pas de sections
          critiques).
       2) On fait fait un rendez-vous unilatéral avec OSFlagPost pour
          démarrer une première fois (ou redémarrer le système) i.e. les
          taches TaskGenerate, TaskComputing, TaskFowarding, TaskOutputPort
          et TaskStats avec le masque TASKS_ROUTER

    Sinon si button_data == BP1:
       1) Configurer le système en mode suspendu  (en jaune sur Fig. 1)
       2) On fait un rendez-vous unilatéral avec OSFlagPost  pour
          stopper le système temporairement i.e. les tâches TaskGenerate,
          TaskComputing, TaskFowarding, TaskOutputPort et TaskStats avec le
          masque TASKS_ROUTER

        Sinon si button_data == BP2:
        1) on fait un rendez unilatéral avec OSFlagPost pour débloquer
           StartupTask avec le masque TASK_SHUTDOWN qui va alors s'occuper
           d'arrêter le système complètement

    On met à 0 les interruptions du GPIO avec l'aide du masque XGPIO_IR_MASK
}
```

[4] Si parfois le bouton est à 0, consultez la section 6

StartupTask (dernière ligne), remplacez OSTaskSuspend((OS_TCB *)0,&err) par :

```
UCOS_Print("Router initialized - Ready to launch - Hit push button\r\n");

Mettre ici le OSFlagPend() qui bloque sur TASK_SHUTDOWN

// Dans la prochain et dernière partie 3du lab 1  nous procéderons à la
destruction des tâches ici

UCOS_Print("Prepare to shutdown System - \r\n");

while (1) {       // indique que le système est en arrêt permanent
            TurnLEDButton(0b1111);   // mettre 4 bits
            OSTimeDlyHMSM(0, 0, 1, 0, OS_OPT_TIME_HMSM_STRICT, &err);
            TurnLEDButton(0b0000);
            OSTimeDlyHMSM(0, 0, 1, 0, OS_OPT_TIME_HMSM_STRICT, &err);
}
```

5.2 Manip 2 : Création du ISR fit_timer_isr0 ainsi que l'adaptation de la tâches TaskStats et TaskStop.

Nous allons maintenant mettre en service la minuterie matérielle qui remplacera la minuterie logicielle *OSTimeDlyHMSM(0, 0, 30, 0, OS_OPT_TIME_HMSM_STRICT, &err);* dans TaskStats **(donc à partir de maintenant mettre cette ligne en commentaire)**. Autrement dit, la minuterie *fit_timer_isr0* se synchronisera avec *TaskStop* qui elle affichera les statistiques en se synchronisant avec *TaskStats* et suspendra temporairement le système si le nombre de paquets rejetés depuis le dernier affichage (variable *nbPacketCRCRejete*) est trop élevé.

Regardons d'abord la création de la minuterie matérielle, qui est très simple :

ISR fit_timer_isr0 (à mettre juste après ISR gpio_isr0 dans routeur.c)

```
void fit_timer_isr0(void *p_int_arg, CPU_INT32U source_cpu); {

      OS_ERR perr;
      CPU_TS ts;
      OS_FLAGS  flags;

xil_printf("------------------ FIT TIMER 0 -------------------\n");

      On vérifie  si on est déjà le système n'est pas en pause (suspendu
temporairement),
      si oui mettre le code du RDV unilatéral pour démarrer TaskStop
      avec OSFlagPost et le masque TASK_STOP_RDY
      sinon on fait rien…
}
```

Puis vous devrez mettre à jour TaskStop (souvenez-vous que TaskReset ne sert plus). Voici le pseudo-code pour vous inspirer :

Tâche TaskStop (à adapter)

```
void TaskStop(void* data){
      CPU_TS ts;
      OS_ERR err;
      while (true) {

      Mettre ici le code du RDV unilatéral avec fit_timer_isr0
      via OSFlagPend et le masque TASK_STOP_RDY
      (N.B. TaskStop doit s'exécuter une seule fois et attendre le prochain
      rendez-vous de fit_timer_isr0)

      Mettre ici le code du RDV unilatéral pour démarrer TaskStats avec
      OSFlagPost et le masque TASK_STATS_PRINT (pour explication de ce masque
      voir section suivante)

       Test sur nbPacketCRCRejete
         Si nbPacketCRCRejete a pas dépassé la valeur limite
         alors 3 choses:
            1) safeprintf("-------- Task stop suspend all tasks -------\n");
            2) mettre à jour l'état du système pour une suspension temporaire
            3) On fait fait un rendez-vous unilatéral avec
               OSFlagPost  pour stopper
               le système temporairement i.e. les tâches
               TaskGenerate, TaskComputing, TaskFowarding,
               TaskOutputPort et TaskStats avec le masque TASKS_ROUTER
}
```

Suggestion pour tester cette fonctionnalité : observer le nombre moyen de paquets rejetés pour cause de mauvais CRC en fonctionnement normal (soit p.e. 1000) puis augmenter dans TaskGenerate le pourcentage de paquets erronés (***if*** *(nbPacketCrees % 100 == 0) packet->crc++;*) pour dépasser la limite fixer (p.e. 2000). Une fois que votre test est fait et que la fonctionnalité est validée, ramener au pourcentage initiale le nombre de paquets erronés (1 sur 100%) et laisser la valeur limite à 2000 pour ne pas être dans la suite du développement constamment dans l'état suspendu.

Tâche TaskStats (à adapter)

Ici on indique seulement les modifs à faire :

1. Ajoutez à la suite de `OSFlagPend(&RouterStatus, TASK_STATS_RDY, 0, OS_OPT_PEND_FLAG_SET_ALL + OS_OPT_PEND_BLOCKING, &ts, &err);` un deuxième OSFlagPend qui lui va répondre au rendez-vous de TaskStop avec le masque le masque TASK_STATS_PRINT
   (N.B. TaskStats doit s'exécuter une seule fois et attendre le prochain rendez-vous de TaskStop quand le système fonctionne normalement)

2. Mettre en commentaire le code à la fin de TaskStats:

a. `if (nbPacketCrees > limite_de_paquets)  OSSemPost(&Sem, OS_OPT_POST_1 + OS_OPT_POST_NO_SCHED, &err);`

b. `OSTimeDlyHMSM(0, 0, 30, 0, OS_OPT_TIME_HMSM_STRICT, &err);`

5.3 Manip 3 : Création du ISR gpio_isr1 et de la tâche TaskClearFifo et mise è jour de TaskComputing

Création de ISRs gpio_isr1 (à mettre juste après ISR fit_timer_isr0 dans routeur.c)

```
void gpio_isr1(void *p_int_arg, CPU_INT32U source_cpu) {
      CPU_TS ts;
      OS_ERR err;
      int switch_data = 0;

      Lecture de la switch et affichage du led correspondant via switch_data

      Si le système est en fonction normal

            Prev_Status_TaskComputing = Status_TaskComputing

            Si switch_data == SWITCH0 // voir valeur dans interrupts.h
            Status_TaskComputing = CS_Mutex

            Sinon si  switch_data == SWITCH1
            Status_TaskComputing = CS_Semaphore

            Sinon si  switch_data == No_SWITCH ou SWITCH1and2
            Status_TaskComputing = No_CS

            Mettre le code du RDV unilatéral pour démarrer TASK_CLEAR_FIFO_RDY
            avec OSFlagPost et le masque TASK_CLEAR_FIFO

      On met à 0 les interruptions du GPIO avec l'aide du masque
      XGPIO_IR_MASK
}
```

Création de la tâche TaskClearFifo (à mettre juste après TaskGenerate dans routeur.c)

Vous devez d'abord ajoutez une toute nouvelle tâche dans le système, incluant la priorité et tout l'information du TCB. Inspirez-vous de TaskStop. Réfléchissez bien à la priorité (plus ou moins que celle de TaskStop ?).

```
void TaskClearFifo(void* data){
	CPU_TS ts;
	OS_ERR err;
	while (true) {

	Mettre ici le code du RDV unilatéral avec gpio_isr1
	via OSFlagPend et le masque TASK_CLEAR_FIFO_RDY
	(N.B. TaskClearFifo doit s'exécuter une seule fois et attendre le prochain
	rendez-vous de gpio_isr1)

	Si Status_TaskComputing a changé suite à l'interruption
		1) Vider les 4 fifos (1 Queuing + 3 TaskComputing)
		2) Mettre NbrEntriesMax de chaque fifo à 0
		3) Mettre max_delay_video,  max_delay_audio et max_delay_autre à 0
		4) Mettre ici le code du RDV unilatéral pour démarrer TaskStats avec
		   OSFlagPost et le masque TASK_STATS_PRINT // important pour
		   // s'assurer que les différentes mises à 0 ont été faites
}
```

En 1) de TaskClearFifo, il y a 2 façons de vider les fifos, je vous laisse y réfléchir…

Et finalement, vous devriez être en mesure de modifier rapidement TaskComputing pour se conformer à une des 3 configurations selon les valeurs de Status_TaskComputing.

A partir de là, il ne vous reste qu'à mettre l'identification de la configuration lors de l'impression de TaskStats (ce qui va aussi aider à la correction…). Pour cela mettre le code suivant

```
864         if (Status_TaskComputing == CS_Mutex) {
865             xil_printf("Mode mutex");
866         }
867         else if (Status_TaskComputing == CS_Semaphore) {
868             xil_printf("Mode semaphore");
869         }
870         else
871             xil_printf("Pas de section critique");
872         xil_printf("\r\n");
```

juste avant la ligne :

```
875         xil_printf("1 - Nb de packets total crees : %d\n", nbPacketCrees);
```

5.4 Manip 4 : Passage d'une une interruption de type privée à interruption de type partagée

Maintenant que vous avez une bonne expérience avec les interruptions, dans cette manipulation vous devez modifier l'interruption privée en interruption partagée (tel que vu en classe avec le bloc 5). Vous devez faire ce changement dans Vivado (figure 5) et modifier interrupts.h pour retrouver le fonctionnement de la manipulation 3. Je vous laisse y réfléchir…

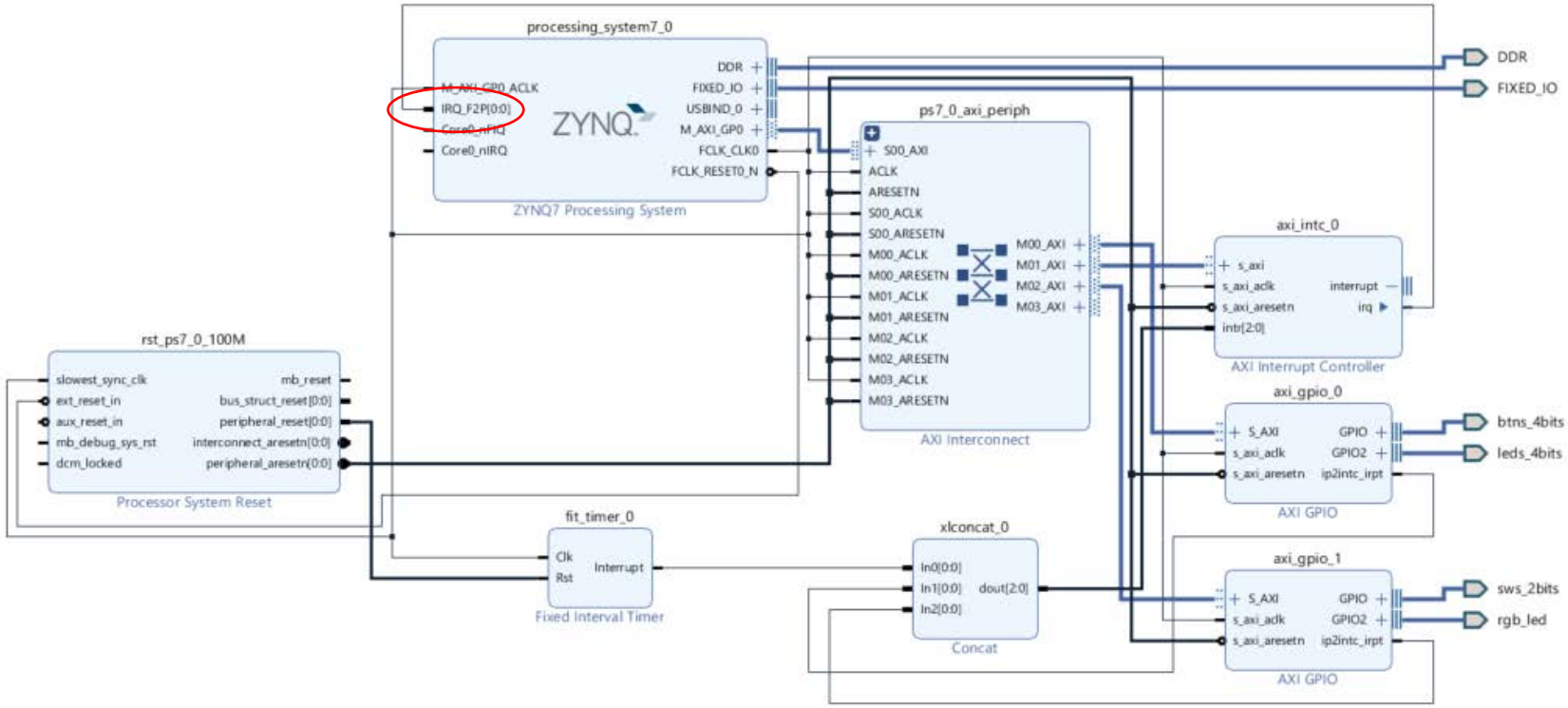

Figure 5. Notre système avec un mode interruption partagée

Questions pour le rapport

1) Il est fortement recommandé d'utiliser l'option OS_OPT_POST_NO_SCHED dans l'appel des fonctions gpio_isr0, gpio_isr1 et fit_timer_isr0. Expliquez la raison.

2) Décrivez le rôle des fonctions suivantes dans la tâche StartupTask :
   - initialize_gpio0();
   - initialize_gpio1();
   - initialize_axi_intc();
   - connect_axi();

3) Concernant la manipulation 4,
   a. Expliquez la différence entre une interruption privée FIQ et IRQ (celles que vous avez utilisée dans ce lab)
   b. Expliquez la différence entre une interruption IRQ privée et partagée
   c. Expliquez votre démarche pour la réalisation de la manipulation 4 (passage de l'interruption IRQ privée à partagée).

Barème de correction et date de remise

Barême Lab 1 Partie 1

Questions	**Notes**	**Commentaires**
Q1 sur .5 point		
Q2 sur 1 point		
Q3 sur 1 point		
Fonctionnement sur 5.5 points		
Total sur 8		

Dates de remise :

Gr. 2 : 25 octobre 2024 11h59 PM

Gr. 1 : 1 novembre 2024 11h59 PM

Gr. 3 : 4 novembre 2024 11h59 PM

Guy Bois
Responsable du cours