Lab 2 - Compléments

Note : Assurez-vous d'utiliser les versions des logiciel Vivado et Xilinx SDK indiqués dans le tutoriel sur Moodle (Vivado 2017.2 et Xilinx SDK 2017.2).

1. À propos des interruptions

Lors de ce laboratoire, vous avez à composer avec des interruptions matérielles. Quelques éclaircissements sur ces interruptions semblent nécessaires.

1.1 Le contrôleur d'interruptions

Pour ce laboratoire nous n'utilisons pas directement le contrôleur d'interruption déjà présent sur la carte (GIC pour *Generic Interrupt Controller*, qui est présent sur une grande portion des implémentations de processeur ARM). Le détail de cette implémentation est présent dans le code, mais vous n'avez pas à vous en soucier. Le contrôleur que nous utilisons est un contrôleur matériel, *AXI_intc* qui est lui-même connecté à un IRQ du GIC laissé libre à cette fin (*Core0_nIRQ* dans Vivado). **Autrement dit, c'est le GIC qui interrompt le processeur, mais c'est le *AXI_intc* qui va gérer l'interruption.** C'est donc à ce contrôleur que vous serez confrontés lorsque vous devrez connecter vos *handlers* et activer vos interruptions, mais aussi lorsque vous devrez signaler que vos interruptions ont bien été servies. Ceci permet de multiplexer les 3 interruptions provenant du FPGA (sections 5.1.2 et 5.1.3) en un seul signal d'interruption du GIC. Vous pouvez donc vous baser sur le code connectant l'interruption GIC du AXI_intc pour connecter vos interruptions au AXI_intc, mais assurez-vous de garder en tête que vous devrez utiliser l'API du AXI_intc (définit dans le fichier *xintc.h*) plutôt que celui du GIC.

1.2 L'interruption générée par GPIO (switches)

Le driver XGpio de Xilinx doit être utilisée afin d'initialiser le GPIO et activer les interruptions. Notez qu'il vous faudra peut-être configurer le driver pour qu'il génère des interruptions lorsque l'état des switches est modifié. De même, dans votre *handler* d'interruption, il faudra l'aviser que l'interruption a été servie.

1.3 Les interruptions générées par minuterie (timer)

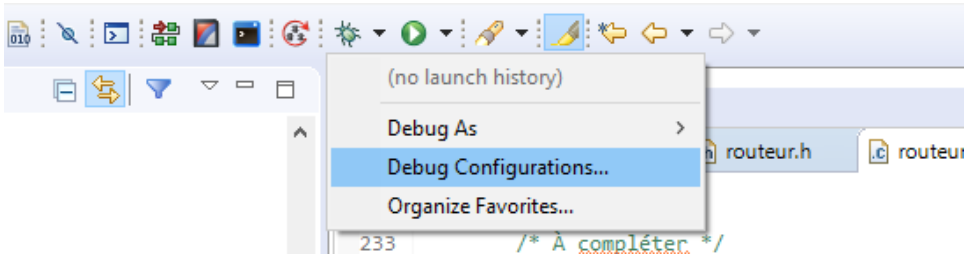
Ces interruptions sont moins compliquées à servir car elles ne font que générer une interruption à intervalle fixe. Aucune modification logicielle n'a donc besoin d'être effectuée (outre l'activation de l'interruption et la connexion de son *handler*).

2 Documentation du BSP Xilinx

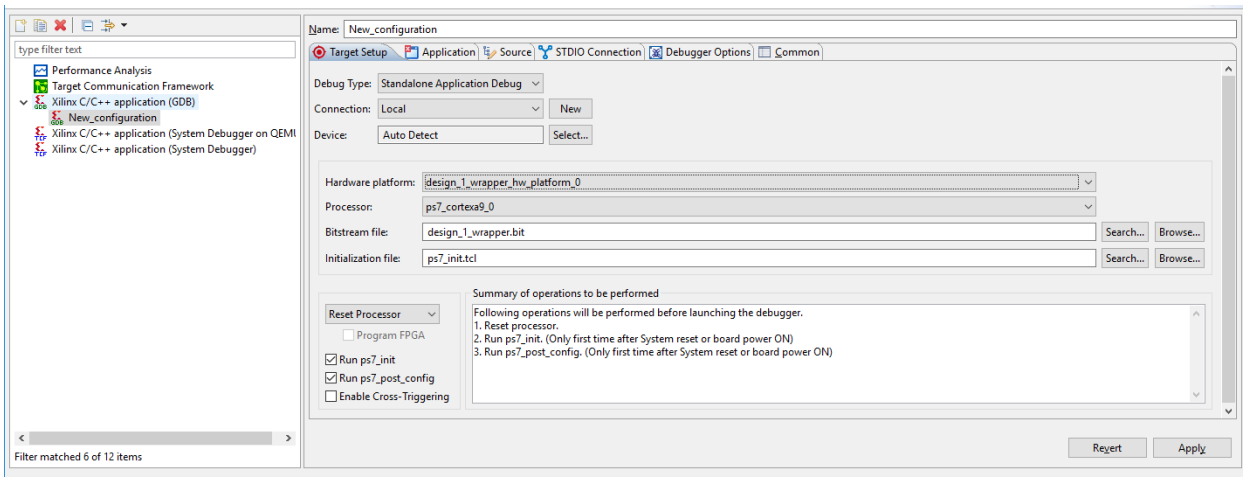
Le BSP de Xilinx, qui fournit les drivers XGpio[1], XIntc, etc. est documenté dans le code de celui-ci, dont les headers sont disponibles dans le projet *standalone_bsp_0/ps_cortexa9_1/include/{xgpio, xintc, etc.}.h.* Ceux-ci donnent un aperçu général des fonctions de cette partie spécifique du BSP, et une documentation plus spécifique est disponible dans l'implémentation, accessible par la sélection de la fonction dans son header et du raccourci F3.

3 Configurations de débogage

Afin de faire rouler votre code sur la carte et de pouvoir débogger sur votre PC, créer une nouvelle "Debuf Configurations" de type *Xilinx C/C++ application (GDB)* comme suit:

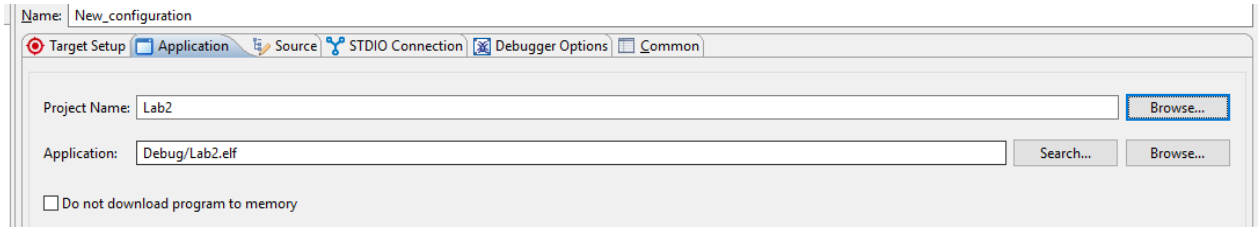


[1] Faites bien attention de ne pas confondre le driver XGpio, qui gère les broches GPIO provenant de la partie FPGA de la puce (ce qui est le cas des switches sur le Zedboard) et le driver XGpio_PS, qui gère les broches GPIO contrôlés par la partie processeur ARM de la puce (non utilisés ici).

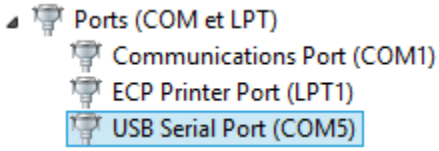


Vous devrez probablement re-sélectionner *design_1_wrapper_hw_platform_0* (ou le nom de votre plateforme si vous n'avez pas gardé les noms par défaut) afin de populer les champs *Bitstram file* et *Initialization file* automatiquement.

Cliquer ensuite sur l'onglet *Application,* puis *Browse...* pour choisir votre projet *Lab2.* Le champ *Application* devrait se remplir automatiquement avec votre .elf.



Cliquez ensuite sur l'onglet *STDIO Connection*. Vous devrez aller sélectionner le port COM correspondant à la connexion UART du Zedboard. Windows et déterminisme ne rimant pas ensemble, la planchette de développement peut être connectée sur n'importe quel port COM. Pour le savoir, ouvrez le *Gestionnaire de périphériques* de Windows et cherchez le port du *USB Serial Port*.



Sélectionner le BAUD Rate à 115200.

Cliquez ensuite sur *Apply*, puis, *Debug.*

4. Facultatif : affichage de la sortie du programme sur Putty

Un terminal vers l'UART du Zedboard doit être ouvert, ce qui peut être fait à l'aide de Putty, qui devrait être préinstallé mais peut aussi être téléchargé ici. Il suffit ensuite d'ouvrir ce programme, de choisir une connexion de type *Serial*, sur COMX (voir étape précédente) à 115200 bauds, de cocher « *Implicit CR in every LF* » dans l'onglet *Terminal*, puis de cliquer sur *Open*.

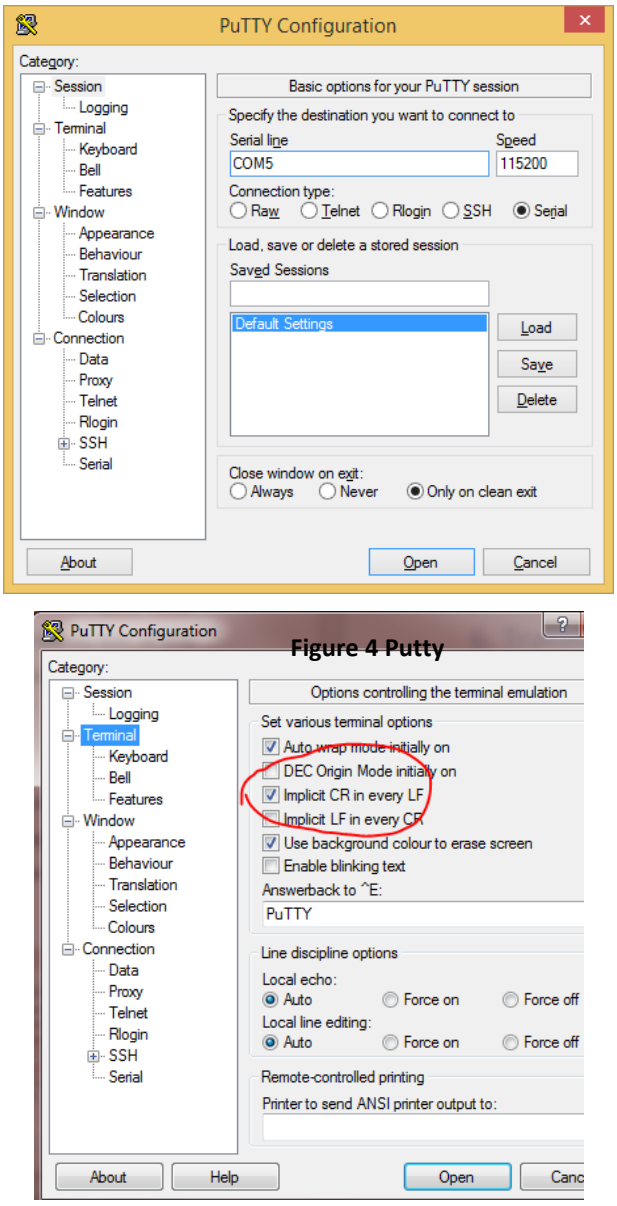


Figure 4 Putty