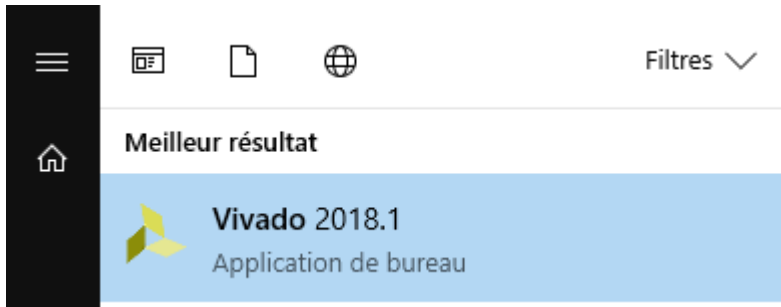


CRÉATION D'UN PROJET SUR ZEDBOARD AVEC VIVADO & XILINX SDK

CRÉATION DE LA PLATEFORME MATÉRIELLE SUR VIVADO

1. Ouvrez Vivado 2018.1

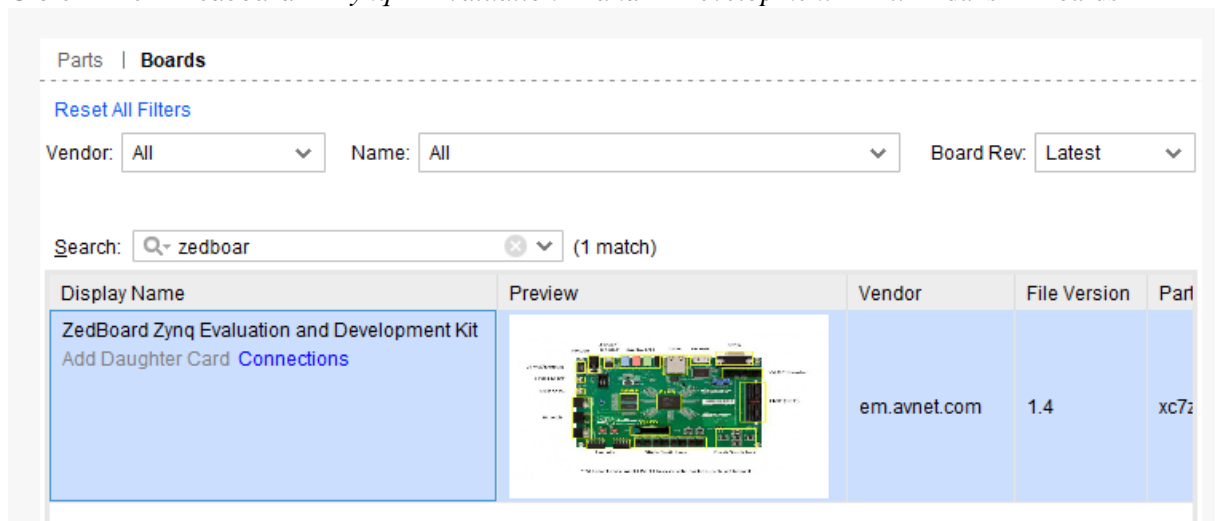


2. Créez un nouveau projet, nommé lab2, dans C:/Temp/3610/matricule1_matricule2¹

A screenshot of the 'New Project' wizard in Vivado. The 'Project name' field is set to 'Lab2'. The 'Project location' field is set to 'C:/TEMP/3610/1234567_9876543'. The checkbox 'Create project subdirectory' is checked. Below the fields, it states 'Project will be created at: C:/TEMP/3610/1234567_9876543/Lab2'.

3. Créez un projet RTL, en cochant *Do not specify sources at this time*

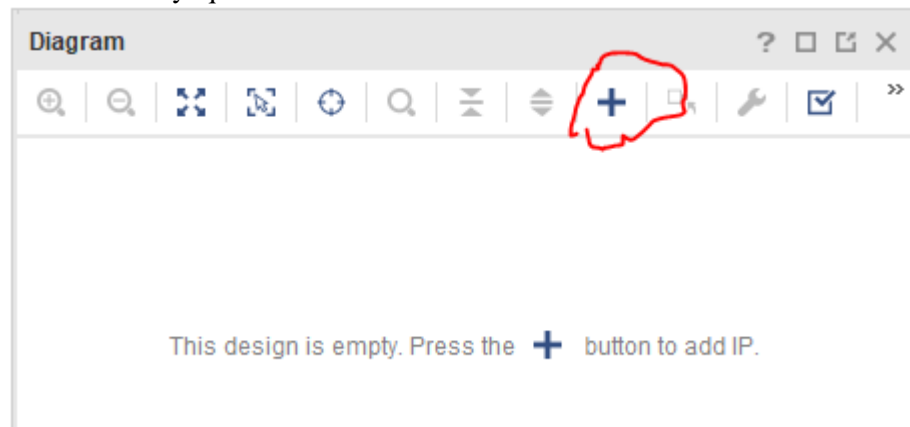
4. Ciblez le Zedboard Zynq Evaluation and Development Kit dans Boards



5. Cliquez sur *Next*, puis *Finish*.

¹ Il est primordial de ne pas créer ce projet sur le réseau de Poly, aka sur votre bureau, dans *Mes documents*, ou n'importe où ailleurs dans X:\, afin de sauver votre temps et celui de votre chargé(e) de laboratoire, qui devra se déplacer lorsque vous aurez des problèmes pour vous dire exactement ce qui est écrit ici.

6. Dans le menu à gauche, cliquez sur *Create Block Design*, puis OK.
7. Le bouton + dans la nouvelle fenêtre *Diagram* vous permet maintenant d'ajouter ou d'activer des blocs au Zynq.

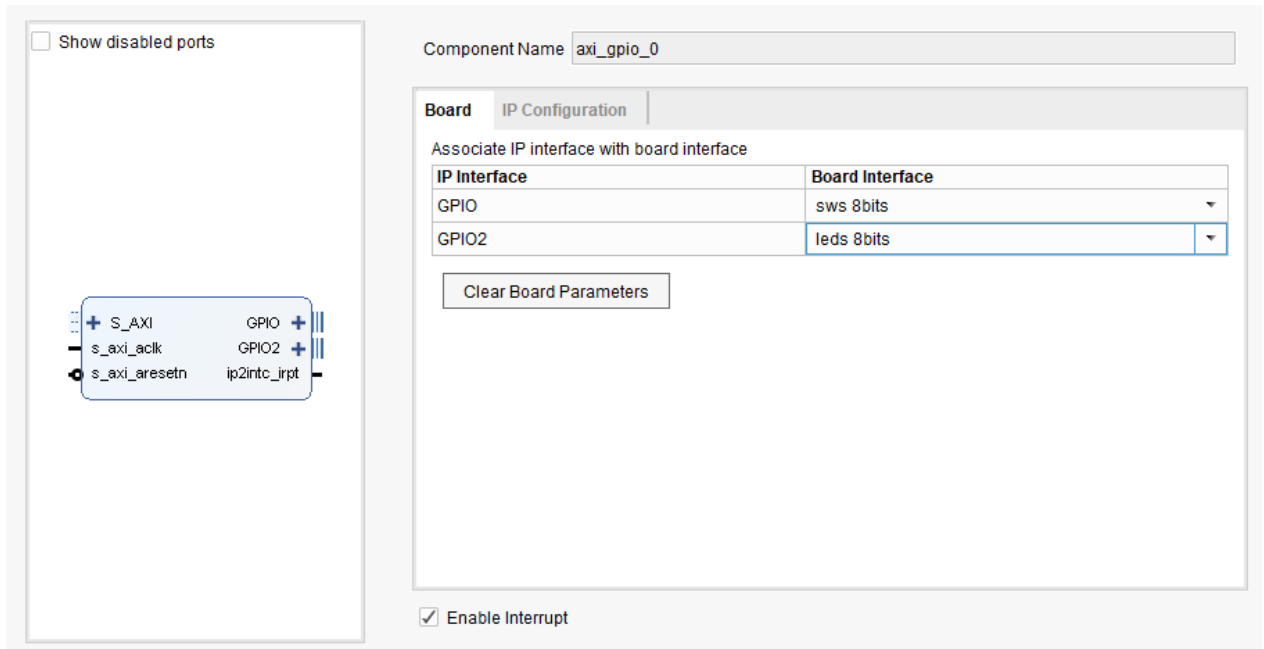


8. Ajoutez une instance de *ZYNQ7 Processing System*. Ceci permet d'activer le processeur ARM à deux cœurs² présents sur le Zedboard.
9. Cliquez sur *Run block automation* pour automatiser la connexion à la mémoire externe (les paramètres par défaut sont bons).
10. Double-cliquez sur l'instance créée, allez dans *Interrupts*, puis cochez *Fabric Interrupts* et *Core0_nIRQ* pour permettre aux blocs du FPGA de générer des interruptions sur le processeur.

Interrupt Port	ID	Description
<input checked="" type="checkbox"/> Fabric Interrupts		Enable PL Interrupts to PS and vice versa
PL-PS Interrupt Ports		
<input type="checkbox"/> IRQ_F2P[15:0]	[91:84], [6...	Enables 16-bit shared interrupt port from the PL. MSB is assigned th...
<input type="checkbox"/> Core0_nFIQ	28	Enables fast private interrupt signal for CPU0 from the PL
<input checked="" type="checkbox"/> Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the PL
<input type="checkbox"/> Core1_nFIQ	28	Enables fast private interrupt signal for CPU1 from the PL
<input type="checkbox"/> Core1_nIRQ	31	Enables private interrupt signal for CPU1 from the PL
PS-PL Interrupt Ports		

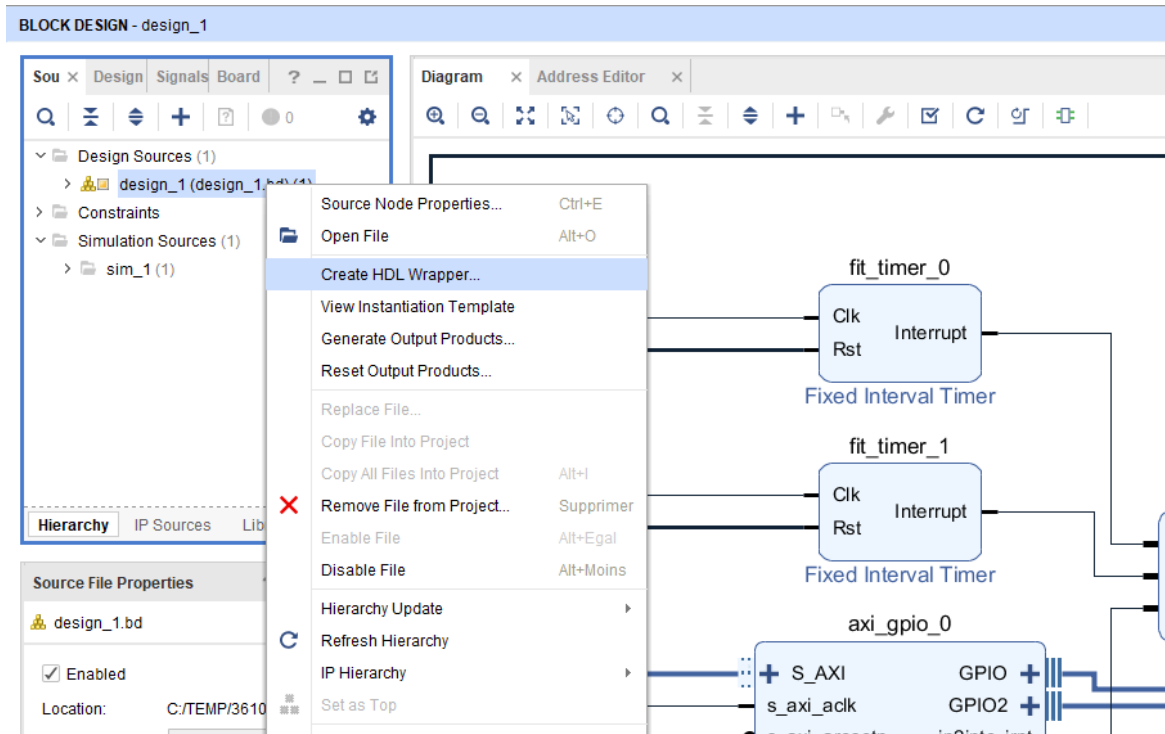
11. Créez 2 instances de *Fixed Interval Timer* (FIT), en cliquant sur *Run block automation* après chaque ajout.
12. Les FIT sont des timers très simples, qui ne font que lever un signal sur leur sortie *Interrupt* après un compte préprogrammé de cycles, en boucle. Modifiez leur configuration pour que le timer #0 lève son interruption à toutes les secondes et que le timer #1 lève son interruption aux 3 secondes (sachant qu'à l'étape précédente, vous avez connecté le bloc à une horloge de 100 MHz).
13. Ajoutez un bloc *AXI GPIO*. Modifiez sa configuration pour que l'interface *GPIO* soit connectée sur les switches du Zedboard (*sws 8bits*) et l'interface *GPIO2* soit connectée aux LEDs du Zedboard. Cochez aussi l'option *Enable interrupt* puis automatisez la génération de toutes les connexions.

² Notez que bien que le processeur ait 2 cœurs, un seul (le cœur #0) sera utilisé plus loin pour faire fonctionner μ C.

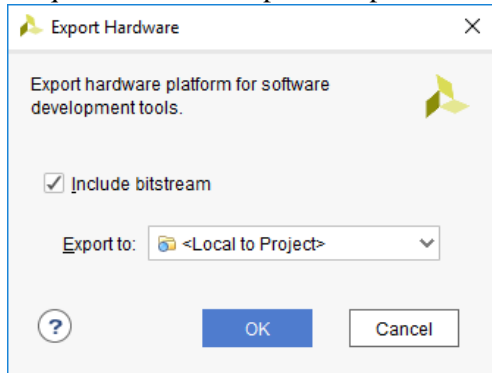


14. Ajoutez un bloc *AXI Interrupt Controller*. Tel qu'expliqué dans l'énoncé du laboratoire, ce contrôleur permettra de faire le multiplexage des 3 signaux d'interruption des blocs préalablement générés dans la seule interruption privée au cœur utilisé du processeur ARM. Connectez sa sortie *irq* (disponible en cliquant sur le + du signal *interrupt*) à l'entrée *Core0_nIRQ* du Zynq. Puis, automatisez la génération des connexions.
15. Ajoutez une instance de *Concat*³ et modifiez là pour qu'elle ait 3 entrées. Connectez les sorties d'interruption des blocs FIT et GPIO aux entrées de cette instance, et sa sortie au port *intr* du contrôleur d'interruption AXI.
16. Votre diagramme de blocs devrait maintenant ressembler au diagramme en Annexe I.
17. Dans le panneau *Sources*, faites un clic droit sur votre design et cliquez sur *Create HDL Wrapper*, puis OK.

³ Notez que *Concat* n'est qu'un moyen visuel de connecter plusieurs sorties indépendantes à un port d'entrée à plusieurs bits.



18. Cliquez sur *Generate Bitstream* dans le panneau de gauche, en bas. Cliquez sur Yes lorsque Vivado vous informe qu'il doit préalablement exécuter la synthèse du système. Démarrez autant de *jobs* en parallèle que possible. Le tout prendra quelques minutes (le statut sera affiché en haut à droite). Si vous tenez à être efficace, vous pouvez commencer à répondre aux questions théoriques à la fin de l'énoncé pendant ce temps.
19. Fermez la fenêtre apparaissant à la fin de la génération correcte du bitstream.
20. Cliquez sur File->Export->Export Hardware et assurez-vous de cocher *Include bitstream*.



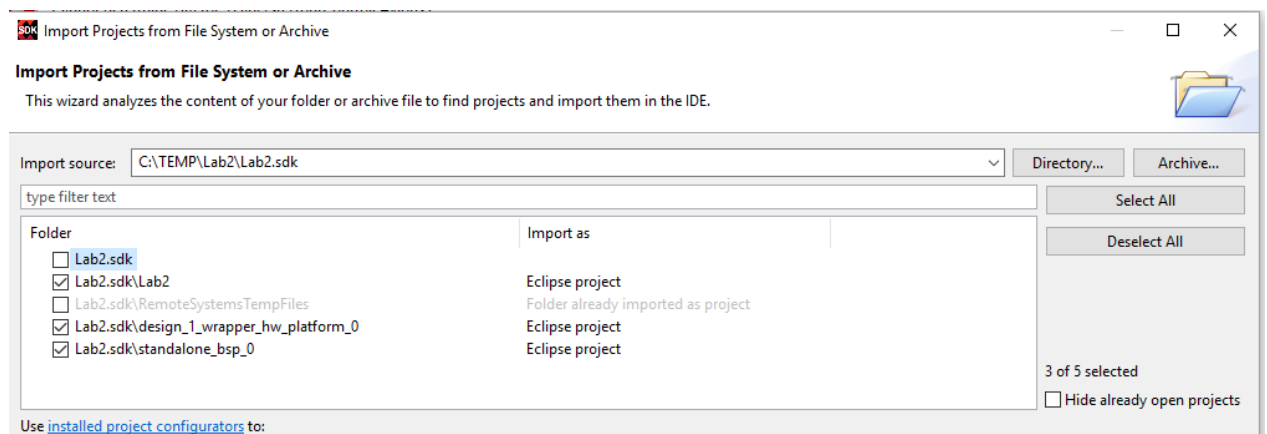
21. *Optionel* : Vous pouvez ouvrir Xilinx SDK à partir de Vivado afin que votre workspace soit bien initialisé et que les fichiers nécessaires soient importés : File-> Launch SDK.
22. Vivado peut maintenant être fermé.

CRÉATION DU BSP ET DÉBUT DU DÉVELOPPEMENT SUR XILINX SDK

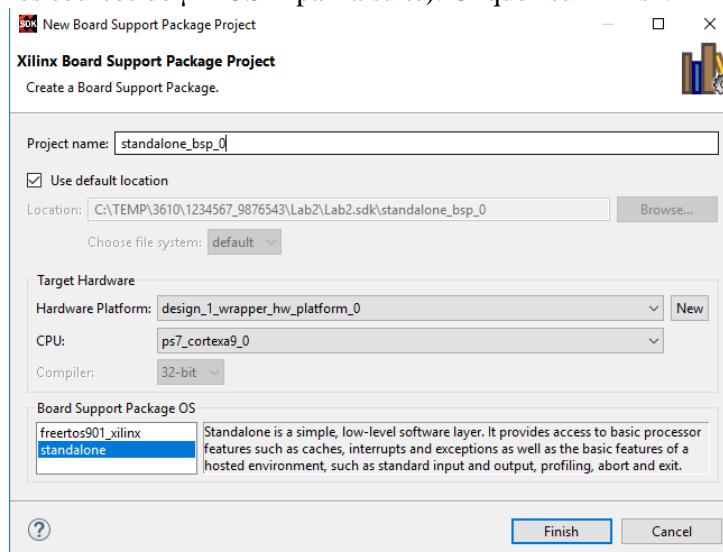
Si vous ne l'avez pas déjà fait, ouvrez le *Xilinx SDK* (Menu Démarrer->Xilinx SDK 2018.1) et faites pointer la workspace vers C:\TEMP\3610\matricule1_matricule2\Lab2\Lab2.sdk

À ce stade-ci, le *Xilinx SDK* un IDE basé sur la plateforme *Eclipse* devrait être ouvert, et une plateforme matérielle générée (*design_1_wrapper_hw_platform_0* si vous avez gardé les noms par défaut). Cette plateforme matérielle contient essentiellement les informations de configuration matérielle faite dans Vivado, l'adresse des périphériques, etc. On veut maintenant créer un *Board Support Package* (BSP) contenant les drivers pour ces périphériques et par la suite créer notre projet se basant sur le BSP (lui-même basé sur la plateforme matérielle importée de Vivado).

Note : Si vous ouvrez le SDK et aucun projet n'est affiché dans l'explorateur de projet (par exemple, si vous ne l'avez pas ouvert depuis Vivado), cliquer sur File->Open Projects from File System..., puis cliquez sur *Directory...* et allez chercher votre dossier Lab2.sdk. Ensuite, sélectionner les dossiers nécessaires à votre implémentation, typiquement le *hardware platform*, le BSP, et votre *Application project* (si vous les avez déjà créés).

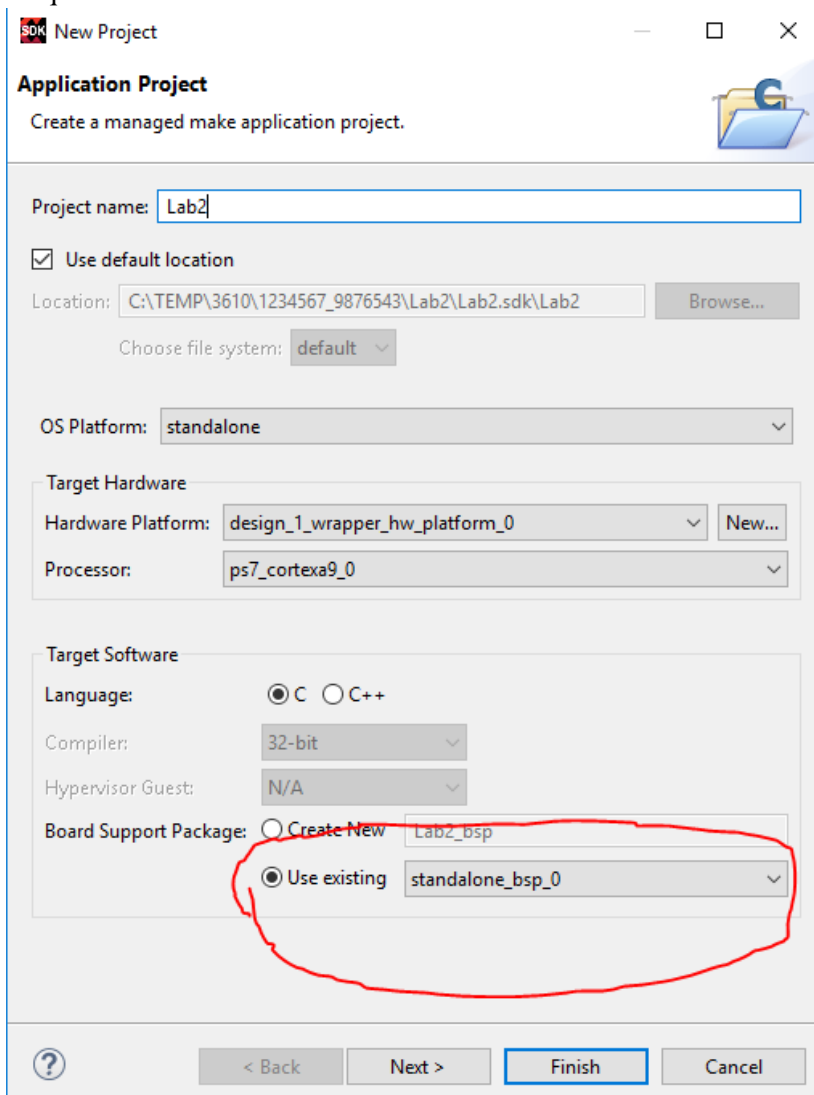


1. Cliquez sur File->New->Board Support Package.
2. Vérifiez que l'OS est bien à *standalone* (ceci est contre-intuitif, mais nous ajouterons manuellement les sources de μ C-OS II par la suite). Cliquez sur Finish.





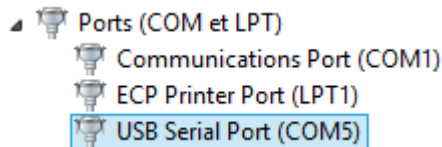
3. Dans la fenêtre *Board Support Package Settings* qui apparaît :
 - a. À l'onglet standalone, vérifiez que stdin/stdout sont sur *ps7_uart_1*
 - b. À l'onglet *ps7_cortexa9_0*, ajoutez les paramètres de compilateur *-O0 -g3 -DDEBUG* qui bien qu'optionnels, faciliteront grandement le débogage.
4. Remarquez, dans le BSP créé (*standalone_bsp_0*->*ps7_cortexa9_0*->*include*), les fichiers *xgpio.h*, *xintc.h* et *xparameters.h*. Les deux premiers sont les headers des drivers GPIO et du contrôleur d'interruption AXI et documentent entre autres toutes leurs fonctions, alors que *xparameters.h* contient des définitions vers les adresses et configurations des différents périphériques. Notez par exemple les numéros d'interruptions des FIT et GPIO.


```
#define XPAR_FIT_TIMER_0_INTERRUPT_MASK 0X000001
#define XPAR_AXI_INTC_0_FIT_TIMER_0_INTERRUPT_INTR 0
#define XPAR_FIT_TIMER_1_INTERRUPT_MASK 0X000002
#define XPAR_AXI_INTC_0_FIT_TIMER_1_INTERRUPT_INTR 1
#define XPAR_AXI_GPIO_0_IP2INTC_IRPT_MASK 0X000004
#define XPAR_AXI_INTC_0_AXI_GPIO_0_IP2INTC_IRPT_INTR 2
```
5. Faites File->New->Application project, appelez-le lab2 et utilisez le BSP créé précédemment. Cliquez ensuite sur Finish

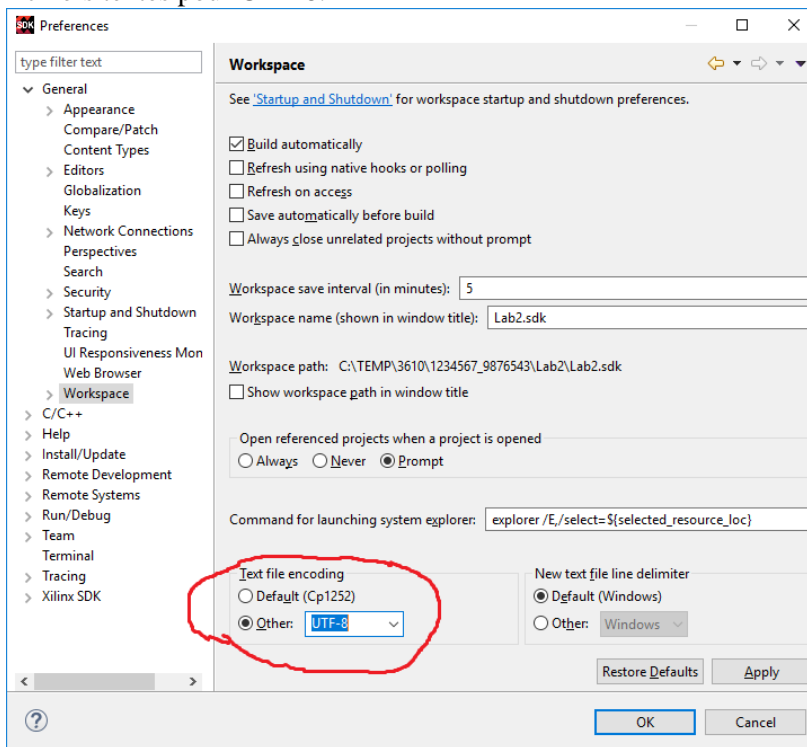


6. Un fichier helloworld.c devrait être créé dans ce nouveau projet.

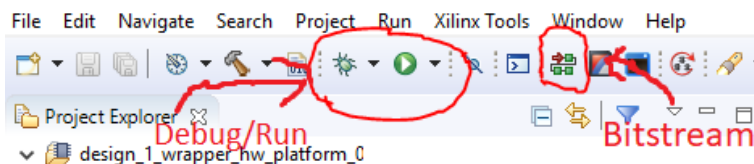
7. Cliquez sur le bouton *Program FPGA*  afin d'envoyer le bitstream⁴ sur la carte.
8. Dans l'onglet *SDK Terminal* en bas de la fenêtre, cliquer sur le bouton  afin de connecter un port serial pour permettre de visualiser la sortie de votre programme. Vous devrez aller sélectionner le port COM correspondant 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*. Conservez le Baud Rate à 115200.



9. Allez dans Windows->Preferences puis General->Workspace et modifiez l'encodage par défaut des fichiers textes pour UTF-8.



10. Finalement, exécutez votre programme en cliquant sur *Run* (ou *Debug*). Choisissez l'option *Launch on Hardware (GDB)*. Vous devriez voir afficher « Hello Word » dans l'onglet SDK Terminal.



⁴ Cette étape a besoin d'être refaite au *reset* du FPGA/à chaque nouvelle séance de lab., mais pas lorsque seul le logiciel est modifié.

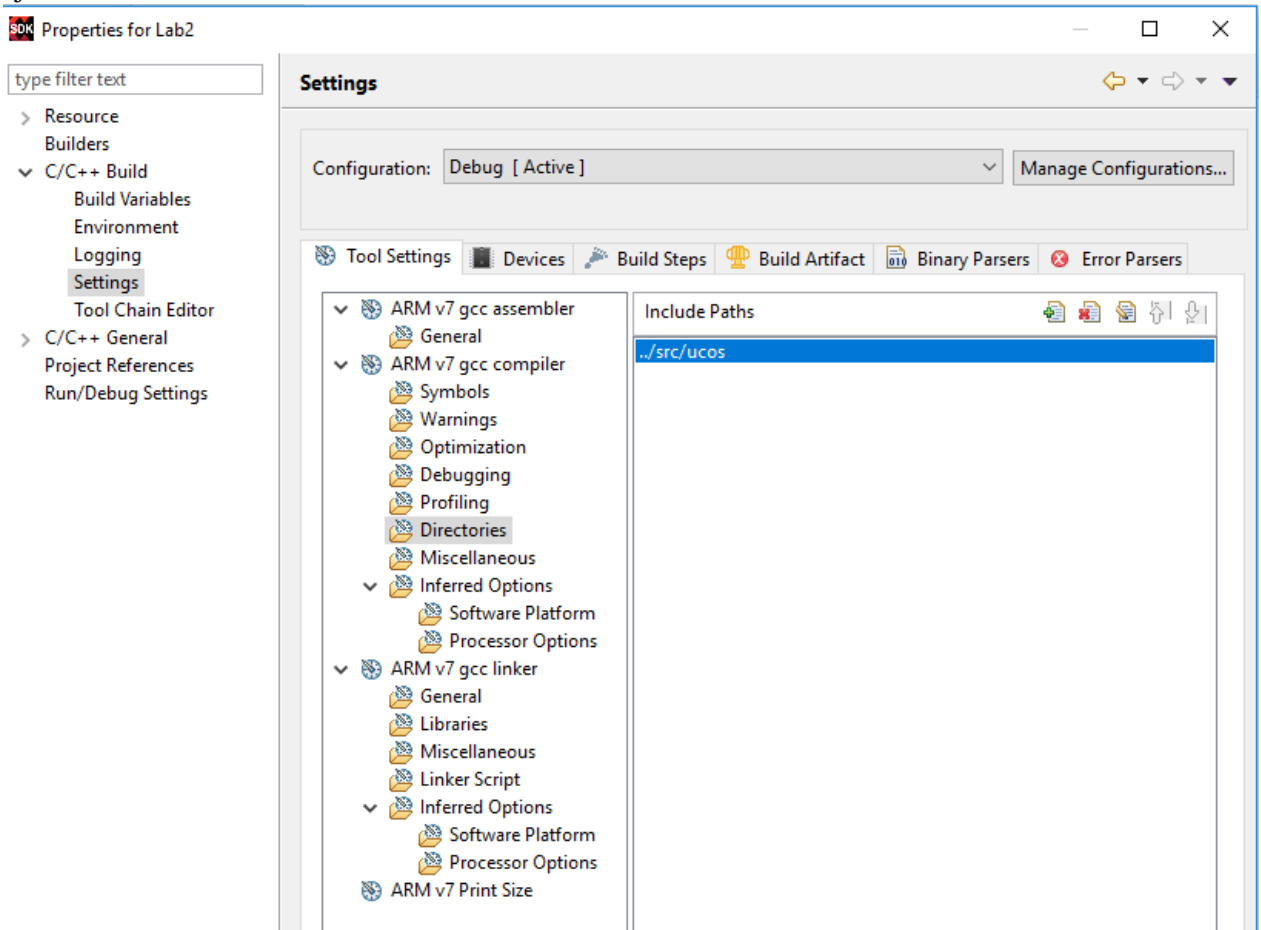
11. Copiez les fichiers fournis avec l'énoncé (le **contenu** du dossier src) dans le dossier des fichiers sources du projet Lab2 (C:\TEMP\3610\1234567_9876543\Lab2\Lab2.sdk\Lab2\src) et supprimez le fichier *helloworld.c*, qui n'est plus nécessaire.
12. Dans le SDK, faites un clic droit sur le projet *Lab2* -> Refresh.
13. Ouvrez le fichier *Iscrip.ld* et ajoutez 2-3 zéros aux champs *Stack* et *Heap Size*.

Stack and Heap Sizes

Stack Size

Heap Size

14. Faites un clic droit sur le projet->C/C++ *Build Settings*. Dans *ARM v7 gcc compiler->Directories*, ajoutez *../src/ucos*:



15. Vous devriez pouvoir exécuter le programme du labo 2. Retournez à l'énoncé du laboratoire pour continuer l'implémentation logicielle.

Diagramme final

