

Laboratoire no 2 – Partie 1b

Création d'un BSP pour uC/OS-III et d'une application de base avec Xilinx SDK 2018.3

Dans la partie 1a, vous avez créé le matériel qui sera utilisé dans la partie 1b puis dans la partie 2. Dans cette partie 1b, vous allez apprendre comment Xilinx fait pour créer la partie logicielle qui roule sur le matériel. Dans cette deuxième partie, vous allez réaliser le bloc du milieu de la figure 3 (voir énoncé de laboratoire). Ce bloc est illustré à la figure 1 ci-bas. L'outil permettant de créer ce logiciel (embarqué) se nomme SDK (Software Development Kit). Dans un premier temps vous allez créer avec SDK le BSP (Board Support Package) qui inclura aussi le noyau de uC/OS-III (ce noyau est celui que vous avez utilisé au laboratoire no 1)¹. Afin de vérifier votre port uC/OS-III, vous ferez un premier test avec Hello Word. Le deuxième test sera la deuxième partie du lab 2. La figure 1 résume le travail à réaliser.

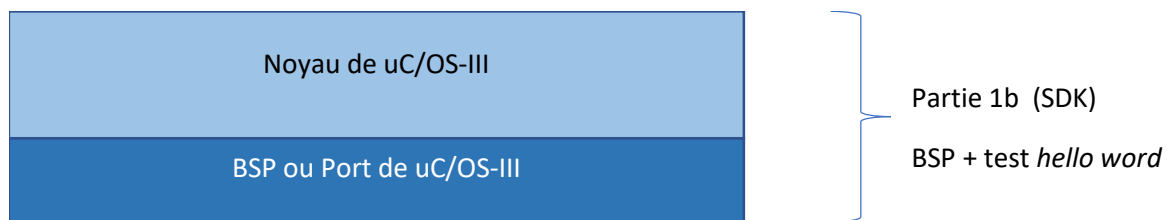


Figure 1. À concevoir dans la partie 1b

L'utilisation de OS-III sur la Zedboard, bien qu'assez simple, est très sensible aux erreurs. De plus, la documentation de Micrium s'avère non exhaustive sur la configuration et l'utilisation des drivers. Nous allons donc démêler tout cela dans la suite de ce document.

¹ Dans le laboratoire no 1 le BSP était émulé par Windows (p. ex. minuterie et changement de contexte), alors qu'ici ce sera un vrai BSP s'exécutant sur une vraie plate-forme embarquée (Zedboard).

Creation d'un projet

Étape 1 : Copier **UC/OS-III v1_44** dans C:/TEMP (ou dans votre compte). Si vous le déposez dans TEMP il faudra le remettre là chaque session de travail.

Étape 2 : Dans Xilinx SDK, allez dans Window->Preferences->Xilinx SDK -> Repositories, puis ajoutez le répertoire ucos aux répertoires spécifiés à l'étape 1, comme à la figure 2.

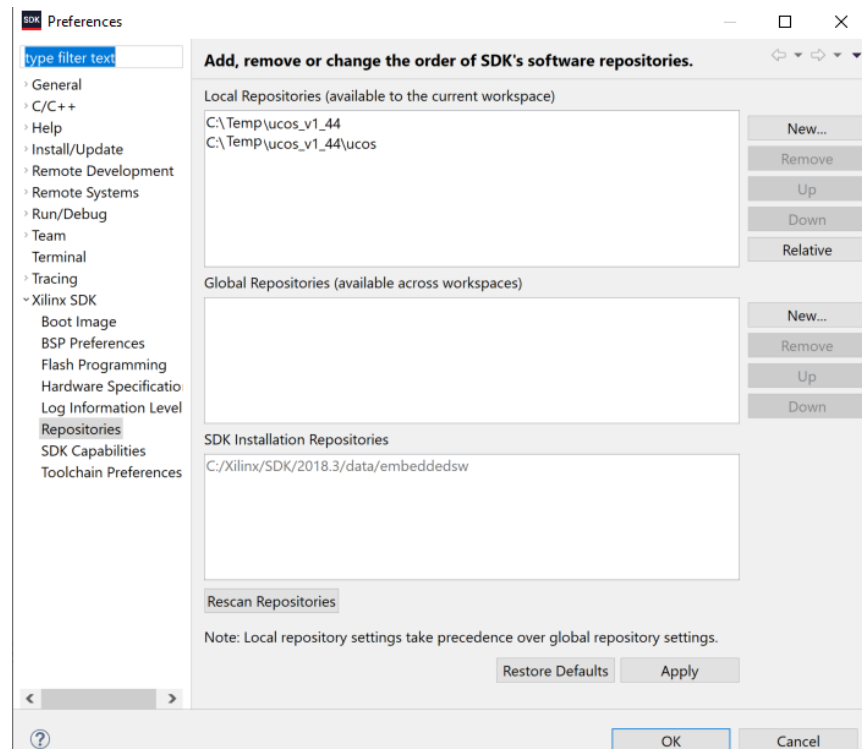


Figure 2 : Mettre OS-III en répertoire local.

Étape 3 : Créer le BSP, File->new->Board Support Package. Dans un premier temps, vous remarquerez (Figure 3, page suivante) que la plate-forme matérielle conçue dans la partie 1a devrait apparaître (voir Target Hardware *design_1_wrapper_hw_platform_0*). Sélectionnez ucos comme OS, vous devriez voir alors apparaître un nom de projet dans la fenêtre *Project name* : *ucos_bsp_0*. Faites ensuite OK.

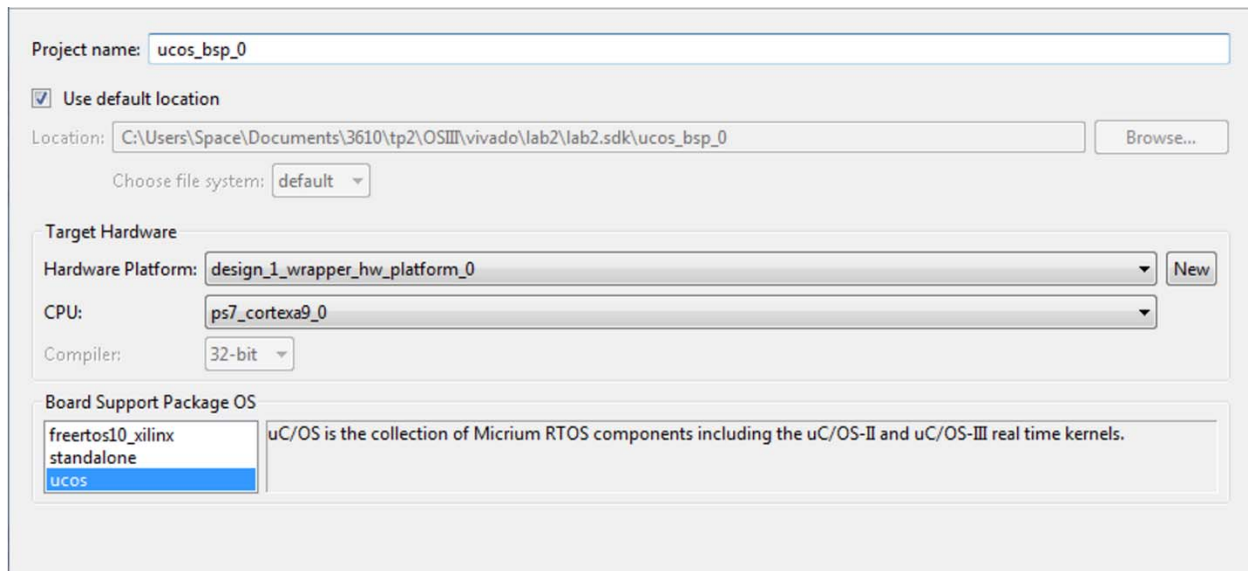


Figure 3 : création d'un BSP utilisant ucos en tant qu'OS

Après quelques instants, vous devriez voir apparaître à droite de l'écran (fenêtre Project Explorer) en dessous de *Hardware design_1_wrapper_hw_platform_0* votre BSP (*uC_bsp_0*). Vous devriez aussi voir apparaître la fenêtre *Board Support Package Settings* sur laquelle nous allons spécifier quelques paramètres (Étape 4). Si parfois cette fenêtre ne s'ouvre pas, alors placez-vous au-dessus *uC_bsp_0* et avec le bouton de droite cliquez sur Board Support Package Settings. La fenêtre devrait alors apparaître.

Étape 4 : Ici, vous deviez cocher les lignes correspondant à *ucos_common*, *ucos_iii* et *ucos_standalone*, comme présenté dans la figure 4.

N.B. Notez que d'autres options existent (que nous n'utiliserons pas au lab 2), mais il est bon de savoir qu'elles existent (par exemple, la gestion de fichiers, la pile TCP/IP, la pile USB, etc.).

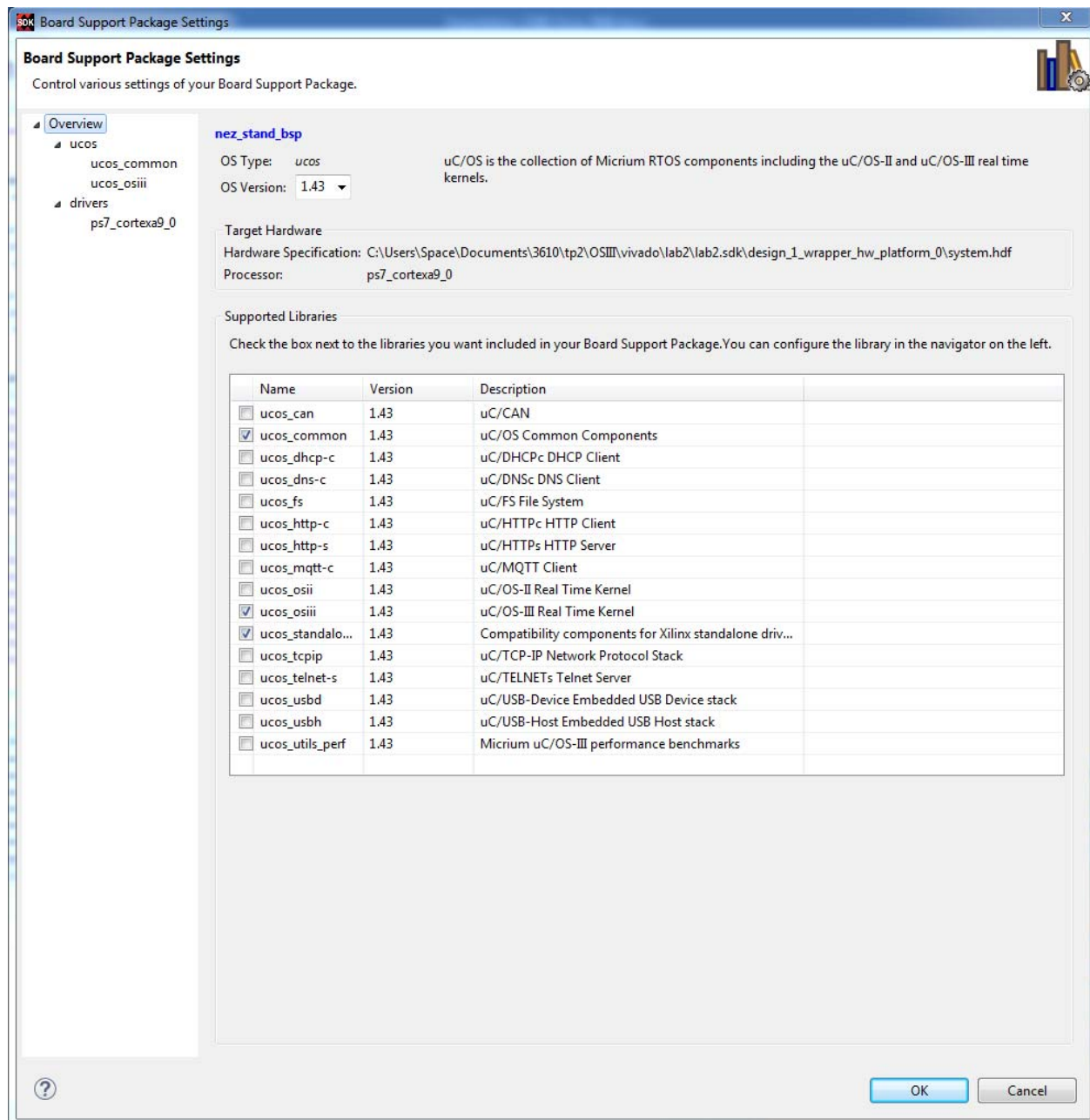


Figure 4 : Choix des librairies que le projet devra utiliser

Étape 5 : Dans l'onglet driver (cliquez sur driver), il est très important de modifier l'option suivante :

- Pour *axi_intc*, choisissez le driver intc, qui correspond au driver écrit par Xilinx (Figure 5)

Étape 6 : Dans l'onglet ucos_standalone (cliquez sur ucos_standalone), il est très important de modifier l'option suivante si vous voulez avoir un output de votre application lorsqu'elle s'exécutera sur la carte:

- Pour *output*, choisissez *ps7_uart* (Figure 6)

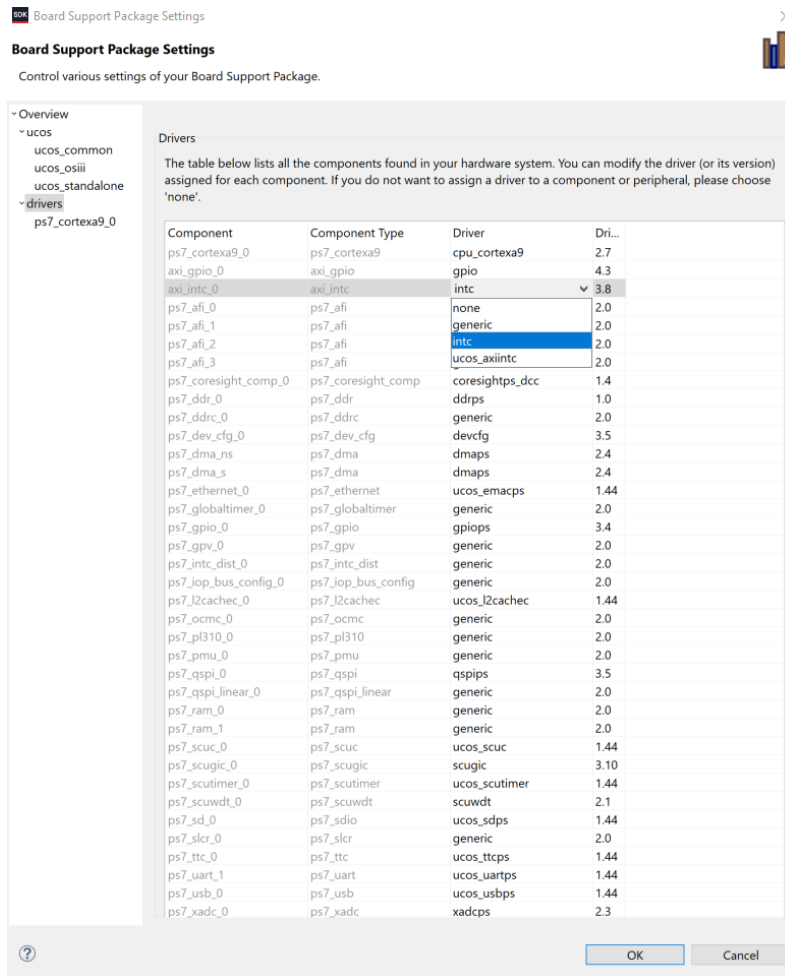


Figure 5 : Choix des drivers à utiliser dans le projet

Étape 7 : Vous pouvez à présent créer votre projet sur lequel vous roulez le programme Hello World. Faites :

file -> new -> Application Project

Pour la boîte déroulante permettant de choisir l'OS Platform, sélectionnez ucos. Choisissez également le BSP que vous venez de créer (*uC_bsp_0*) en cliquant sur Use existing (section Board Support Package). Tout ça est illustré à la figure 7a.

Finalement cliquez sur Next ce qui vous amènera à la figure 7b, et choisissez alors micrium uC/OS-III Hello World (**attention de ne pas choisir uC/OS-II**), puis cliquez sur finish.

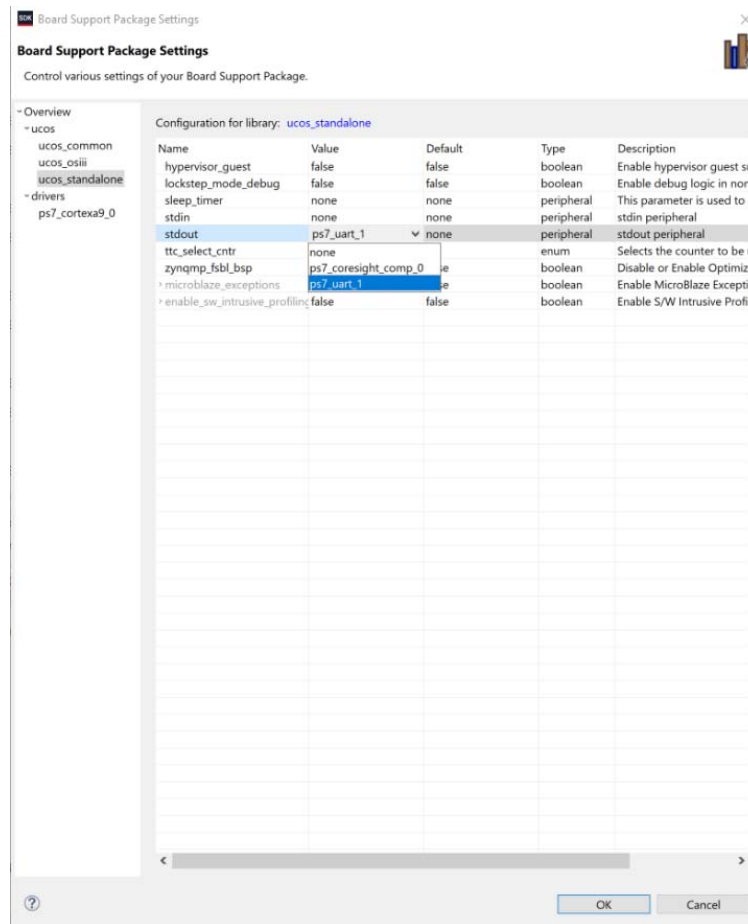


Figure 6 : Choix des drivers à utiliser dans le projet

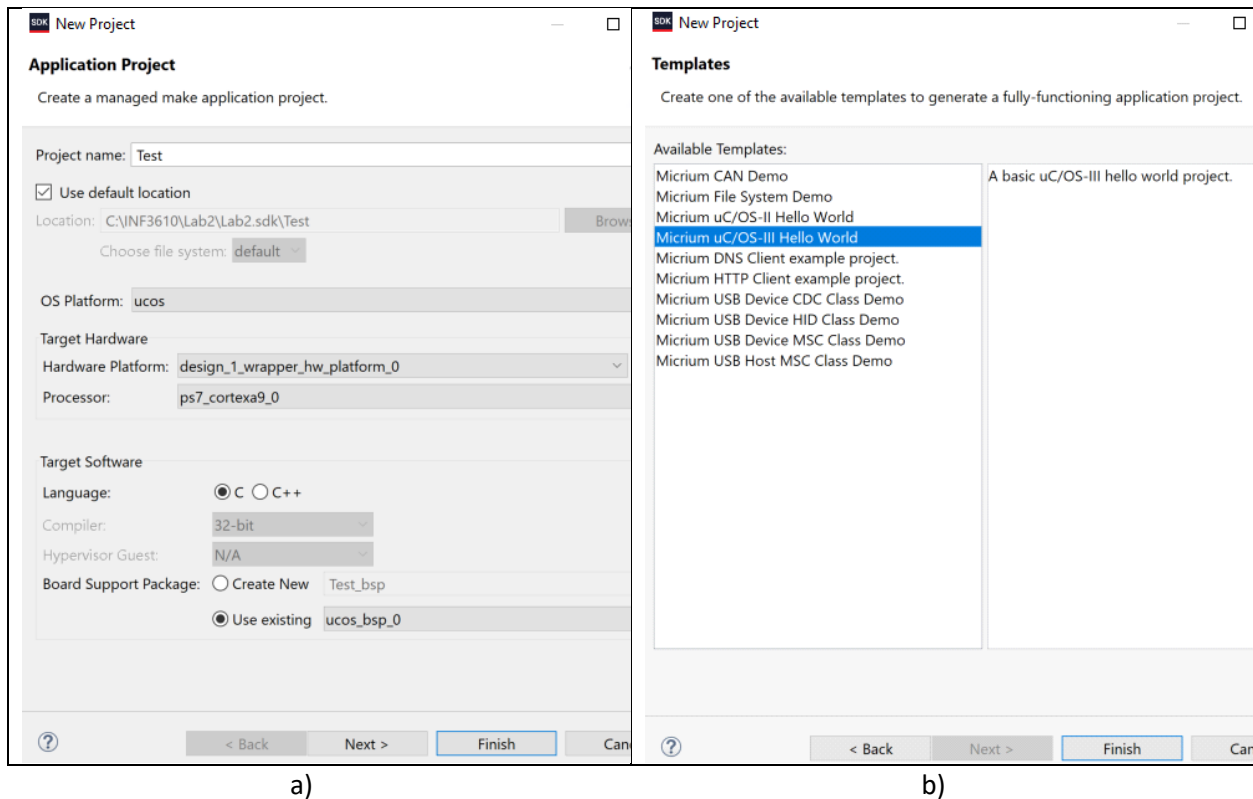

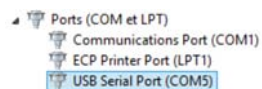


Figure 7 : Création du projet en 2 étapes

Félicitation, vous venez de créer votre premier BSP en utilisant OS-III !!!!!

Étape 8. Il reste à exécuter sur la carte le programme Hello World. Vous pouvez tester que tout fonctionne en exécutant les étapes suivantes :

- 1) Programmez d'abord le bitstream² (voir icône sur la figure 8) afin de l'envoyer sur la carte. Vous verrez alors apparaître la fenêtre de la figure 9 dans laquelle vous retrouverez votre plate-forme et son bitstream généré dans la partie 1a. Cliquez sur Program.
- 2) 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.



- 3) 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.

² 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.

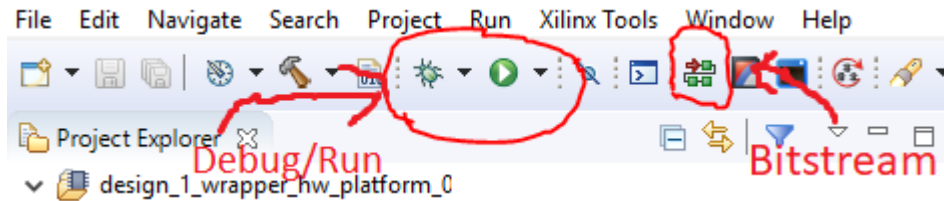


Figure 8

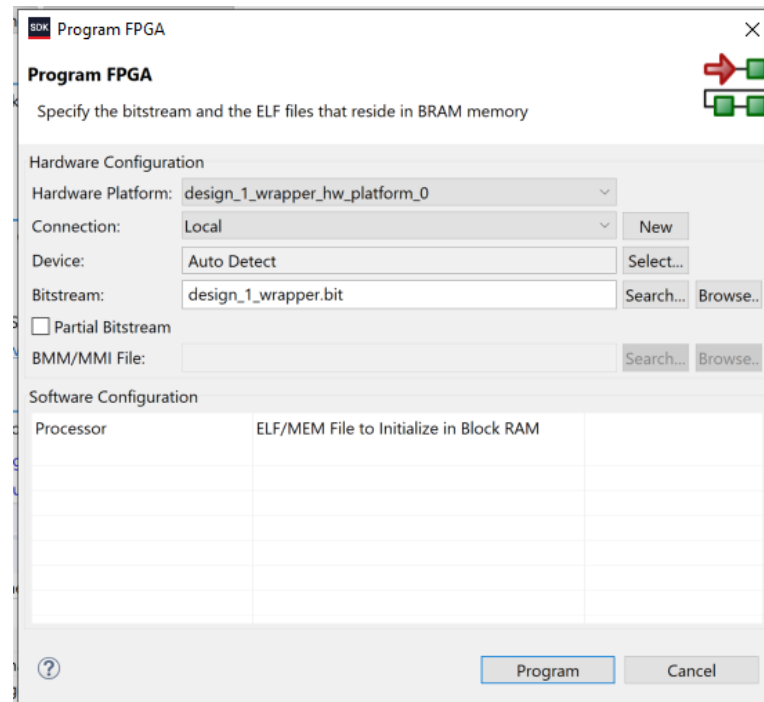


Figure 9

Étape 9. Finalement préparer le code de la partie 2 du lab. Pour cela, 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 app.c (hello world), qui n'est plus nécessaire. Finalement, dans le SDK, faites un clic droit sur le projet Lab2 -> Refresh.

À propos de l'utilisation des drivers

Dans ce tp, on utilisera un certain nombre de driver afin de communiquer avec la carte. Voici les principaux :

- ucos_bsp : Permet de configurer le bsp
- ucos_int : permet de configurer les interruptions. Utilise le Gic.
- xgpiops : Permet d'obtenir les informations sur les switches

- xintc : Permet de configurer les interruptions appartenant au axi_intc dans votre design vivado.

Dans les figures ci-dessous, vous verrez comment utiliser les différents drivers. Également référez-vous au github de Xilinx. Par exemple, vous trouverez de l'information sur le xintc à :

https://github.com/Xilinx/embeddedsw/blob/master/XilinxProcessorIPLib/drivers/intc/examples/xintc_example.c

Prenez le temps de lire et de regarder les exemples, elles seront une bonne source d'inspiration. Ça peut vous paraître moins pédagogique comme approche, mais c'est la vraie vie...

```
OS_ERR os_err;
UCOS_LowLevelInit();

CPU_Init();
Mem_Init();
OSInit(&os_err);

OSTaskCreate(&StartupTaskTCB,
             "Main Task",
             MainTask,
             (void *)0,
             UCOS_START_TASK_PRIO,
             &StartupTaskStk[0],
             0,
             UCOS_START_TASK_STACK_SIZE,
             0,
             0,
             DEF_NULL,
             (OS_OPT_TASK_STK_CHK | OS_OPT_TASK_STK_CLR),
             &os_err);

init_interrupt();
connect_interrupt();

OSStart(&os_err);

return 0;
```

Figure 12 : Initialisation et création des tâches.

```

int initialize_axi_intc() {
    int status;

    status = XIntc_Initialize(&axi_intc, XPAR_AXI_INTC_0_DEVICE_ID);
    if (status != XST_SUCCESS)
        return XST_FAILURE;

    //handle_axi = AXIIntCInit(XPAR_AXI_INTC_0_DEVICE_ID);
    return XST_SUCCESS;
}

void initialize_gpio()
{
    if (XST_DEVICE_NOT_FOUND == XGpio_Initialize(&gpSwitch, GPIO_SW_DEVICE_ID))
        UCOS_Print("Erreur init gpio\n");
    XGpio_InterruptGlobalEnable(&gpSwitch);
    XGpio_InterruptEnable(&gpSwitch, XGPIO_IR_MASK);
}

void init_interrupt() {
    KAL_ERR kal_err;
    CPU_INT32U tick_rate;
    UCOS_IntInit(); /* Initialize interrupt controller. */

    tick_rate = OS_CFG_TICK_RATE_HZ;
    UCOS_TmrTickInit(tick_rate); /* Configure and enable OS tick interrupt. */
    KAL_Init(DEF_NULL, &kal_err);
    initialize_gpio(); // Initialize gpio
    initialize_axi_intc(); // Initialise le axi_intc
    UCOS_StdInOutInit();
}

```

Figure 13 : Initialisation du axi_intc et du gpio

```

void connect_interrupt() {
    CPU_BOOLEAN succes = UCOS_IntVectSet (PL_INTC_IRQ_ID,
    1,
    0,
    (UCOS_INT_FNCT_PTR)XIntc_DeviceInterruptHandler,
    (void*) (uint32_t)axi_intc.CfgPtr->DeviceId);

    if (succes != DEF_OK)
        UCOS_Print ("connect axi : FAIL \n");
    succes = UCOS_IntSrcEn(PL_INTC_IRQ_ID);
    if (succes != DEF_OK)
        UCOS_Print ("enable axi : FAIL \n");

    connect_gpio_irq();
    connect_fit_timer_1s_irq();
    connect_fit_timer_3s_irq();
    XIntc_Start(&axi_intc, XIN_REAL_MODE);
}

```

Figure 14 : Connexion du axiintc avec le gic