Master Informatique

M1 Spécialité SESI

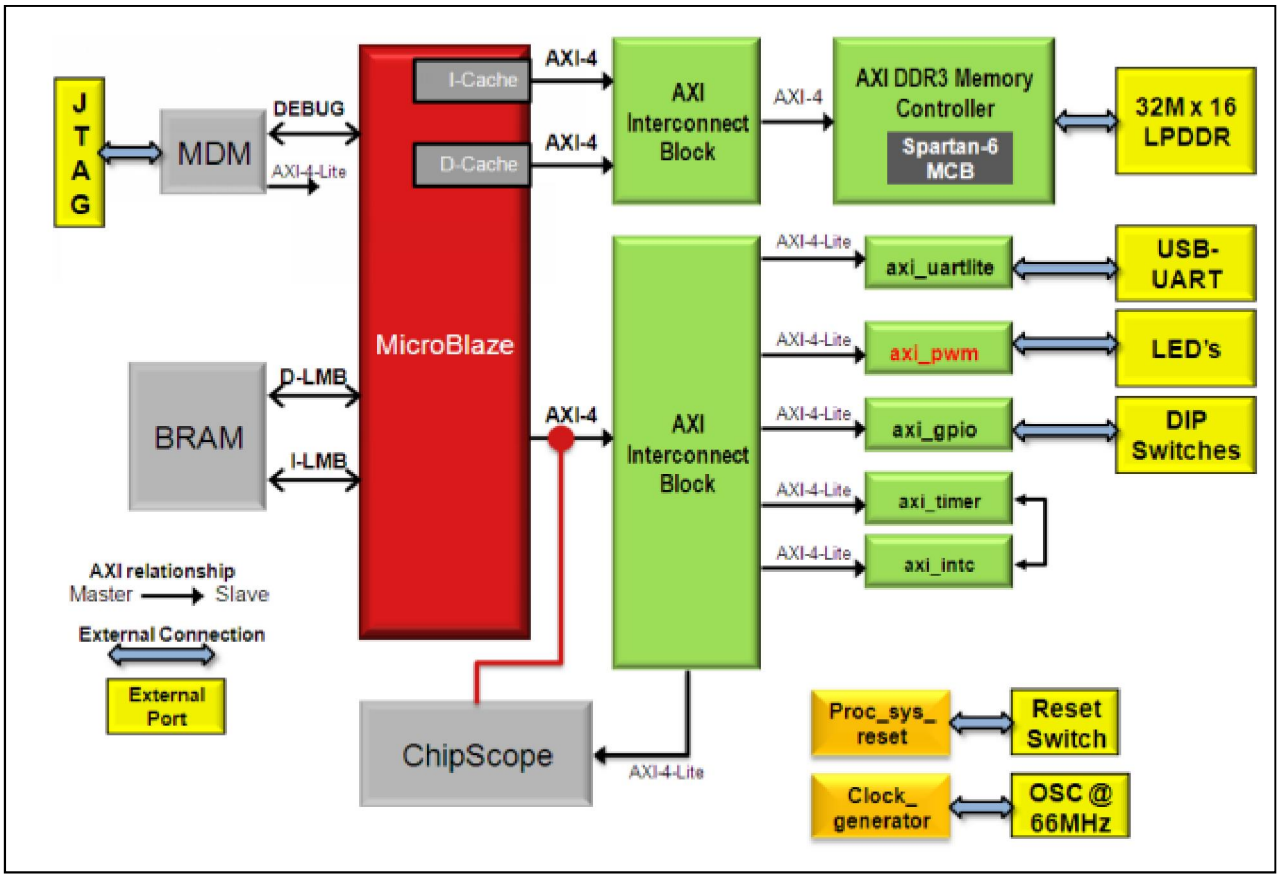
UE 4I108 - J.Denoulet

TP – FPGA1

SYSTEMES PROGRAMMABLES

2EME PARTIE – CODESIGN MATERIEL/LOGICIEL

L'objectif de ce TP est d'implémenter puis de programmer un système mixte matériel/logiciel sur le FPGA. Ce système sera basé sur le processeur embarqué Microblaze de Xilinx.



Le développement sera réalisé grâce aux outils suivants :

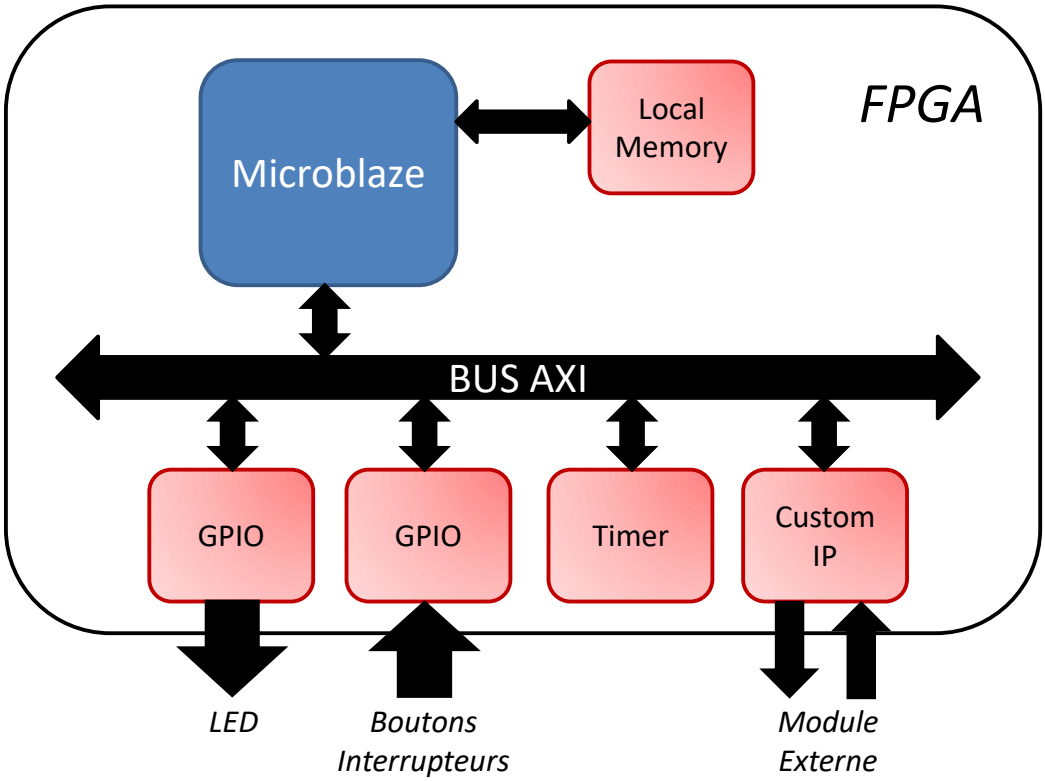
- Le module ***IP Integrator*** de ***Vivado*** pour la génération de la plate-forme matérielle
- ***SDK (Software Development Kit)*** pour le développement et l'exécution de l'application logicielle

1) Présentation

- Le ***Microblaze*** est un cœur de processeur ***softcore***, c'est-à-dire un module se présentant sous la forme d'une description matérielle (en VHDL dans notre cas), qui peut être implémenté sur une cible de type FPGA, et surtout associé à des périphériques standard (Timer, port GPIO, etc…) ou "custom" (comme le module de commande des trains électriques que vous développerez par la suite).
  - ➢ On peut ainsi concevoir un système de type microcontrôleur adapté aux besoins de l'application, combinant les avantages du logiciel (rapidité de développement) et du matériel (performances, flexibilité).

- Le ***Microblaze***, ainsi que la plupart des périphériques standards sont déjà disponibles dans ***Vivado*** sous la forme de modules paramétrables que l'on peut ajouter ou enlever au système. On appelle également ces modules ***IP*** (pour ***Intellectual Property***).
  - ➢ Réaliser un système ***Microblaze*** revient à appliquer une ***méthodologie de conception à base d'IP***.

- A l'exception de quelques composants comme la mémoire ou le gestionnaire d'horloge et de reset, le ***Microblaze*** communique avec ses périphériques grâce à un bus d'interconnexion au protocole ***AXI***.
  - A l'image d'un microcontrôleur, chaque périphérique dispose de registres de configuration,
    - Ces registres permettent de commander les actions du périphérique, ou surveiller son état.
    - A chaque registre est associée une adresse dans l'espace d'adressage du ***Microblaze***.
  - ➢ le ***Microblaze*** interagit donc avec ses périphériques en lisant ou en écrivant des données aux adresses de leurs registres.

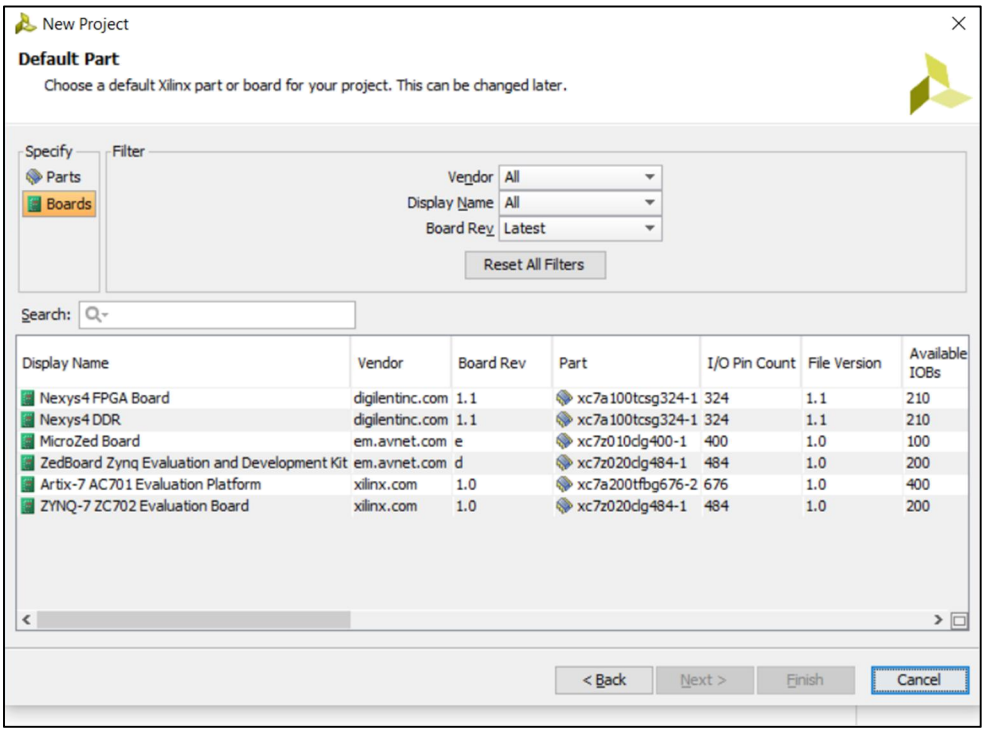


- Le développement d'un système ***Microblaze*** s'effectue en 4 étapes
  - Spécification de la plate-forme matérielle
  - Implémentation de la plate-forme matérielle
  - Création d'un projet pour la partie logicielle
  - Développement de l'application logicielle

2) Spécification de la plate-forme matérielle

- Avant de lancer les outils Xilinx, aller dans le répertoire ***I:\denouletj\4I108-FPGA1\Board Parts***. Copier les répertoires ***Nexys4*** et ***Nexys4_DDR*** à l'endroit suivant :
  C:\Xilinx\Vivado\2014.2\data\boards\board_parts
  - Ces répertoires contiennent les propriétés de nos cartes de développement. Ils permettent de simplifier la définition de la plate-forme matérielle implémentée dans le FPGA.

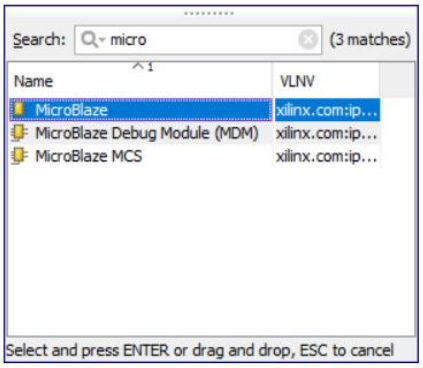
- Ouvrir ***Vivado*** et créer un nouveau projet.
  - Dans la fenêtre de choix du FPGA, cliquer sur ***Boards*** et choisir votre carte (***Nexys4*** ou ***Nexys4 DDR***).



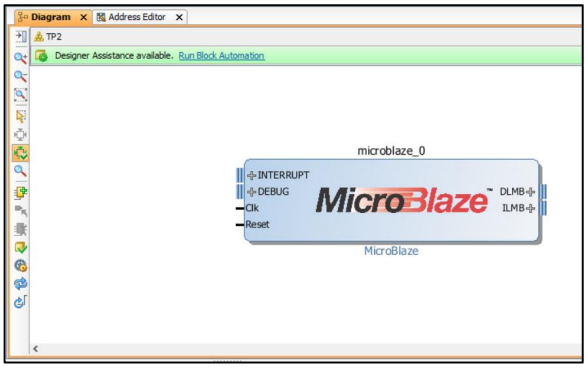
- Une fois le projet crée, dans le ***Flow Navigator***, cliquer sur ***Project Settings*** et choisir ***VHDL*** comme ***Target Language***.

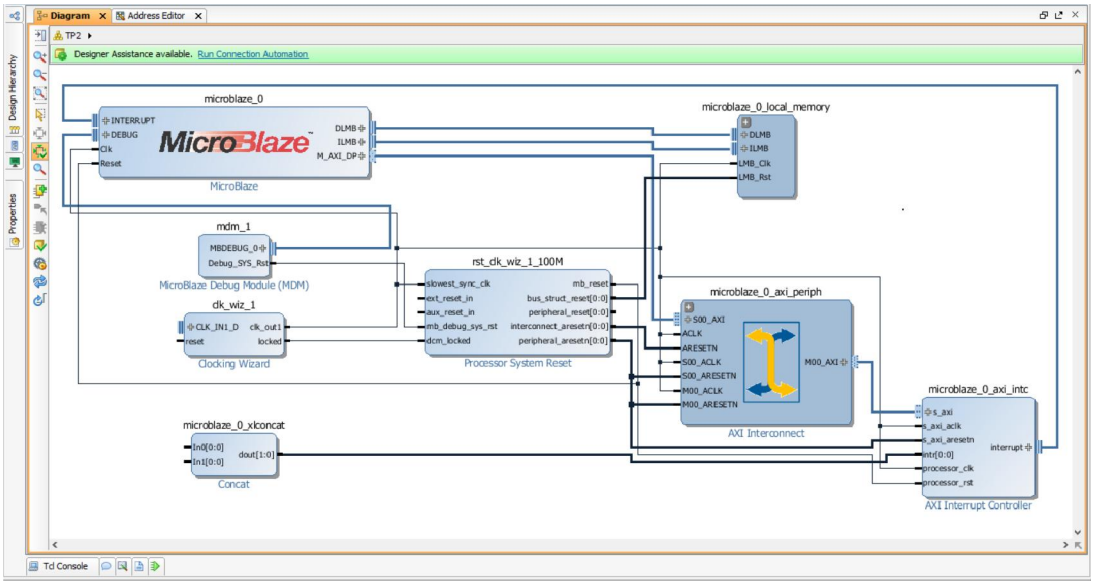
- Dans la partie ***IP Integrator*** du ***Flow Navigator***, cliquer sur ***Create Block Design***.
  - Donner le nom ***TP2*** à votre système.
  - Une fenêtre ***Diagram*** se crée à droite de l'écran. C'est dans cette fenêtre que nous allons spécifier de manière graphique les composants de notre système ***Microblaze***.

- Pour instancier un ***Microblaze***, cliquer sur ***Add IP*** dans le bandeau vert en haut de la fenêtre ***Diagram***.
  - Dans la fenêtre de recherche qui s'ouvre, taper les premières lettres de ***Microblaze*** et sélectionner ***Microblaze*** dans la liste filtrée des ***IP*** disponibles (voir ci-contre).



- Un module ***Microblaze*** est ajouté au ***Diagram***.
    - Cliquer sur ***Run Block Automation*** puis ***/microblaze_0*** pour configurer le processeur.
    - Modifier les paramètres suivants :
        - ***Local Memory*** : Passer de 8 à 32KB.
        - ***Interrupt Controller*** : Cocher la case.
    - Cliquer sur OK. ***Vivado*** ajoute alors automatiquement d'autres modules au ***Diagram***.





- Sont ajoutés
    - ***Clocking Wizard*** et ***Processor System Reset*** :
        - Gèrent le système d'horloge et de réinitialisation du processeur et de ses périphériques.
    - ***Microblaze Debug Module*** :
        - Permet d'utiliser le debugger lors de l'exécution des programmes.
    - ***Microblaze Local Memory*** :
        - Fournit une interface vers la mémoire du processeur.
    - ***AXI Interconnect***:
        - Correspond au bus d'interconnexion entre le ***Microblaze*** et ses périphériques
    - ***AXI Interrupt Controller :***
        - Comme son nom l'indique, c'est le contrôleur d'interruptions du système
        - Le module ***Concat*** permet de regrouper plusieurs signaux individuels sous la forme d'un bus.

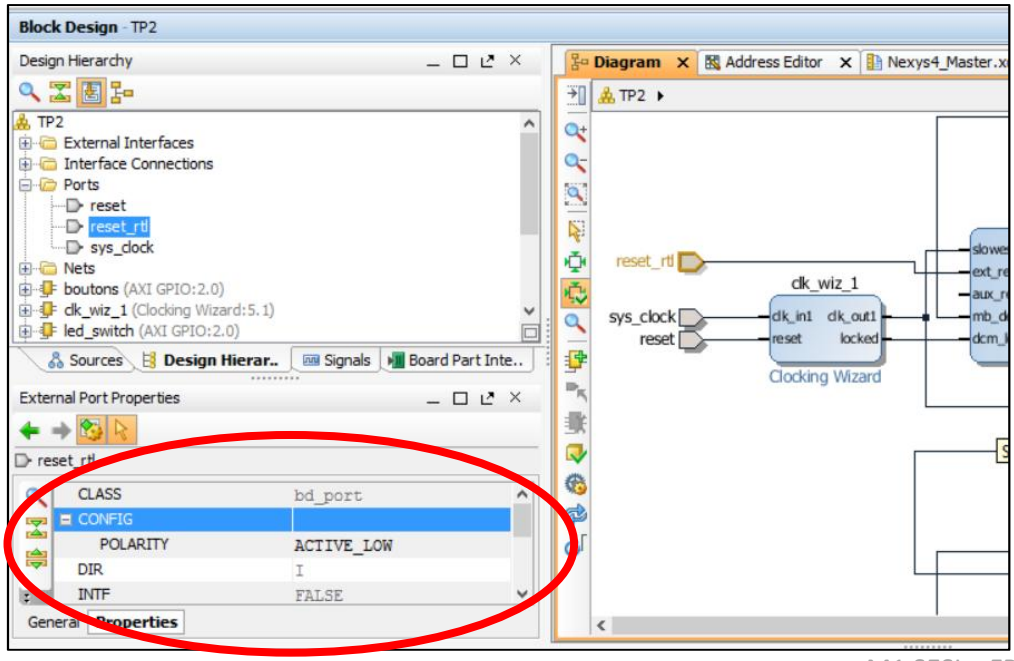
- Double-Clique à présent sur le module ***Clocking Wizard.***
    - Dans l'onglet ***Clocking Options***, dans la section ***Input Clock Information***, modifier le paramètre ***Source*** de la ***Primary Clock*** en sélectionnant ***Single Ended Clock Capable Pin***. Cliquer sur ***OK***

- Nous allons à présent ajouter deux interfaces ***GPIO*** pour être en mesure de piloter les ***LED***, ***interrupteurs*** et ***boutons*** de la carte ***Nexys4***. Cliquer avec le bouton droit sur le ***Diagram*** et choisir ***Add IP***
    - Taper ***GPIO*** dans la fenêtre de recherche et sélectionner le module ***AXI GPIO***.
    - Sélectionner le bloc ***GPIO*** ajouté. Cliquer sur le bouton droit et choisir ***Block Properties***.

- Renommer le nom du module en ***led_switch*** pour signifier que ce bloc ***GPIO*** permettra de gérer les ***LED*** et les ***interrupteurs*** de la carte. (Cela facilitera la programmation du système par la suite)
- Double cliquer sur le bloc ***GPIO*** et sélectionner l'onglet ***IP Configuration***
  - Dans la section ***GPIO***, sélectionner ***All Inputs*** et fixer la taille du port à ***4 bits***
    - Le port 1 du bloc GPIO sera associé aux 4 interrupteurs à droite de la carte.
  - Cocher la case ***Enable Dual Channel***, et dans la section ***GPIO2***, cocher ***All Outputs*** et fixer la taille à ***16 bits***.
    - Le port 2 du bloc GPIO sera donc associé aux 16 LED de la carte.
  - Cocher enfin la case ***Enable Interrupt*** puis valider.
- Créer ensuite un nouveau bloc ***GPIO***, appelé ***boutons*** possédant ***1 port d'entrée de 3 bits*** et pouvant également interrompre le processeur. Ce bloc sera associé aux boutons ***Left***, ***Center*** et ***Right*** de la carte.

- Dans le ruban vert du ***Diagram***, cliquer sur ***Run Connection Automation*** puis :
  - Choisir ***/clk_wiz_1/clk_in***. Vérifier que le parameter ***sys_clock*** est sélectionné et valider.
  - *Attention, les noms des ports/modules peuvent être légèrement différents !*
- Répéter ***Run Connection Automation*** plusieurs fois en sélectionnant successivement :
  - ***/clk_wiz_1/reset***. Vérifier que le parameter ***reset*** est sélectionné et valider.
  - ***/rst_clk_wiz_1_100M/ext_reset_n***. Sélectionner le paramètre ***Custom*** et valider.
  - ***/led_switch/S_AXI***. Vérifier que le paramètre ***Auto*** est sélectionné et valider.
  - ***/led_switch/GPIO***. Sélectionner le paramètre ***Custom*** est sélectionné et valider.
  - ***/led_switch/GPIO2***. Sélectionner le paramètre ***Custom*** est sélectionné et valider.
  - ***/boutons/S_AXI***. Vérifier que le paramètre ***Auto*** est sélectionné et valider.
  - ***/boutons/GPIO***. Sélectionner le paramètre ***Custom*** est sélectionné et valider.

- Vous pouvez voir que plusieurs connecteurs d'E/S ont été ajoutés au design.
  - Pour rendre le ***Diagram*** plus clair, vous pouvez cliquer avec le bouton droit sur la feuille et choisir ***Regenerate Layout***.
  - Cliquer sur le connecteur ***reset_rtl*** (celui qui est connecté au ***reset*** du module ***Processor System Reset***) et dans la fenêtre ***External Block Properties***, onglet ***Properties*** (voir figure ci-dessous), vérifier si le paramètre ***CONFIG/POLARITY*** est en ***ACTIVE_LOW*** (ou faire la modification sinon).



- Faire de même pour le connecteur ***reset*** du module ***Clocking Wizard*** en vérifiant que le paramètre ***POLARITY*** est cette fois égal à ***ACTIVE_HIGH***.

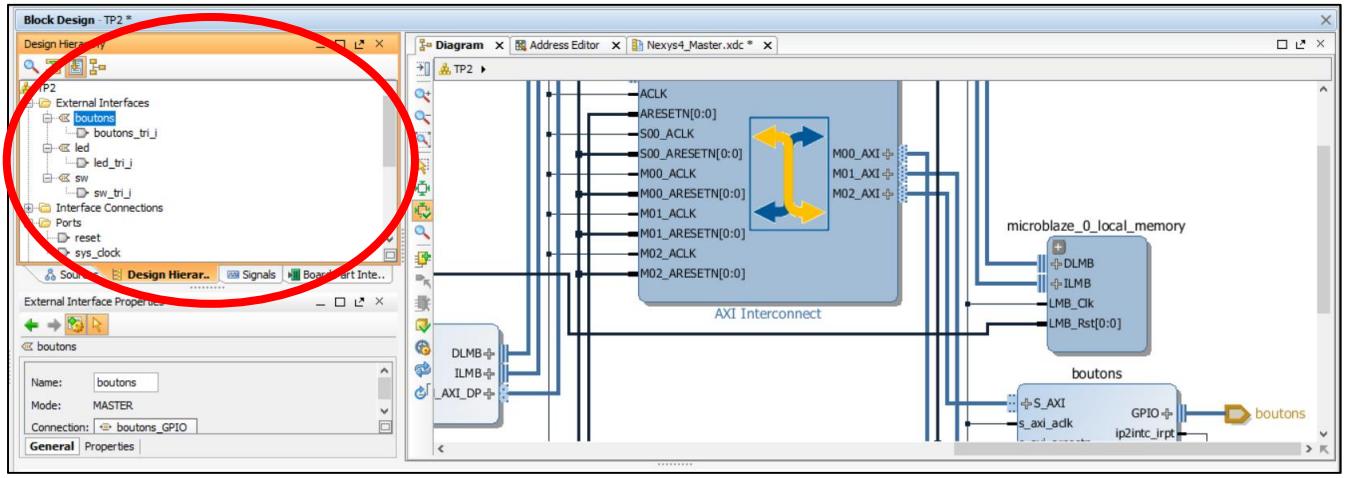
- Localiser sur le schéma le bloc ***GPIO led_switch***. Cliquer avec le bouton droit sur le connecteur d'E/S relié au port ***GPIO*** du module et choisir ***External Interface Properties***. Dans la fenêtre activée, changer le nom du connecteur en ***sw***.
  - Faire de même pour le connecteur relié au port ***GPIO2*** en le renommant ***led***.
  - Idem pour le connecteur relié au port ***GPIO*** du module ***boutons*** en le renommant ***boutons***.

- Il reste à présent à connecter les signaux d'interruption des blocs ***GPIO***.
  - Pour cela tracer manuellement un fil entre les ports ***ip2intc_irpt*** des blocs ***GPIO*** et les deux entrées du module ***Concat.***

- Après avoir sauvegardé le ***Block Design***, aller dans le menu ***Tools*** et choisir ***Validate Design***.
  - Ce processus va vérifier que tout est bien connecté.

3) Implémentation de la plate-forme matérielle

- Pour préparer l'implémentation, nous allons à présent affecter les broches du FPGA aux différents connecteurs d'E/S du ***Block Diagram***.
  - Ajouter le fichier ***XDC*** correspondant à votre carte (***Nexys4*** ou ***Nexys4-DDR***), qui se trouve dans ***I:\denouletj\4I108 – FPGA1*** (en n'oubliant pas de cocher la case ***Copy constraints files into project***)
  - Ouvrir le fichier ***XDC*** et décommenter les lignes correspondant à l'horloge et modifier le nom du port pour qu'il soit identique à celui du ***Diagram*** (a priori ***sys_clock***).
  - Décommenter les lignes correspondant au ***Switch 15*** (l'interrupteur situé à gauche de la carte) et modifier le nom du port pour que le ***reset*** soit associé à cet interrupteur.
  - Décommenter les lignes correspondant au ***Switch 14*** (l'interrupteur situé à droite du Switch15) et modifier le nom du port pour que le ***reset_rtl*** soit associé à cet interrupteur.
  - Dans la fenêtre encerclée sur la figure ci-dessous, cliquer sur l'onglet ***Design Hierarchy*** et dérouler tous les intitulés du dossier ***External Interfaces***.
    - Les noms des ports associés aux ***GPIO*** des boutons, LEDs et interrupteurs figurent au dernier niveau de hiérarchie (exemple pour les boutons: ***boutons_tri_i)***



- Retourner dans le fichier ***XDC***. Décommenter les lignes associées aux :
  - ***Boutons Left, Center et Right***
  - ***LED 15 à 0***
  - ***Interrupteurs 3 à 0***

  et modifier le nom des ports pour y écrire ceux indiqués dans le dossier ***External Interfaces.*** Associer le bouton ***Left*** au port ***boutons_tri_i[2]***, le bouton ***Center*** à ***boutons_tri_i[1]*** et le bouton ***Right*** à ***boutons_tri_i[0].***

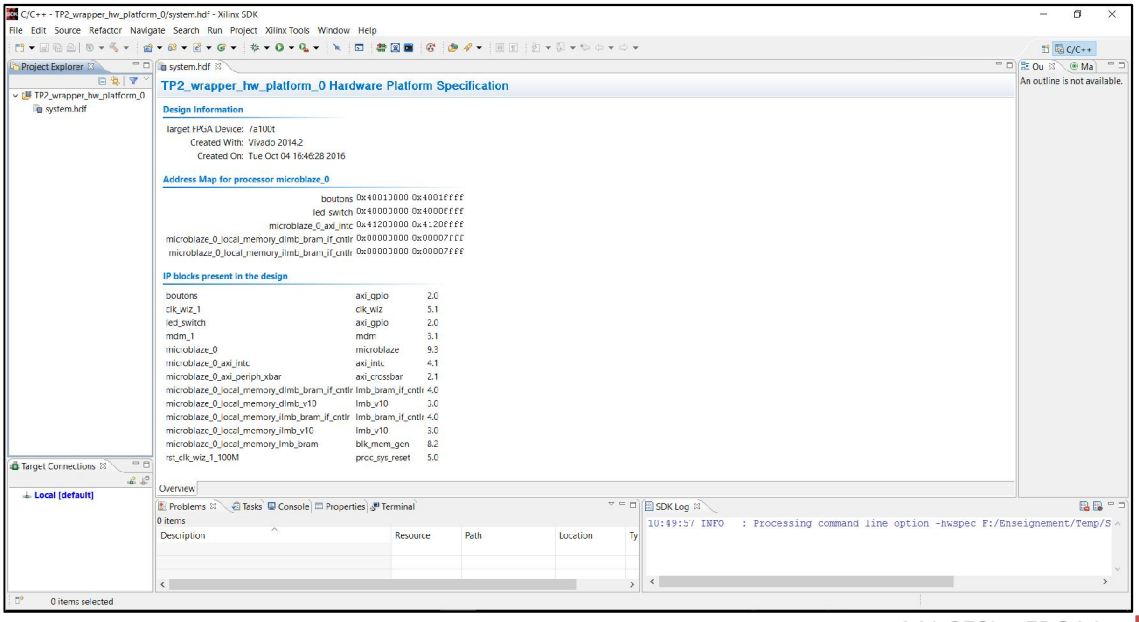
- Nous sommes maintenant prêts à implémenter le design.
  - Cliquer sur l'onglet ***Sources*** pour refaire apparaître le liste des sources et cliquer avec le bouton droit sur le ***Block Design TP2.bd*** pour sélectionner ***Create HDL Wrapper***. Vérifier que l'option ***Let Vivado manage wrapper*** est activée et lancer le processus.
  - Dans le ***Flow Navigator***, cliquer sur ***Generate Bitstream*** pour lancer toutes les phases de l'implémentation.
    - Attention : cela prend un certain temps. En effet :
      - L'outil doit générer le code VHDL de toutes les IP de votre système
      - L'architecture est beaucoup plus importante que celles vues au TP précédent…
  - Lorsque le bitstream est généré, vous pouvez choisir ***Open Implemented Design*** pour visualiser l'implantation du système dans le FPGA.

- Aller dans le menu ***File*** et choisir ***Export*** puis ***Export Hardware***.
  - Vérifier que la case ***Include bitstream*** est bien cochée puis valider.
  - Cette action permettra de récupérer les caractéristiques de la plate-forme matérielle nécessaires à la configuration de l'application logicielle.

- La partie conception matérielle est à présent terminée.
  - Pour passer au développement logiciel, cliquer dans le menu ***File*** sur ***Launch SDK***.

4) Création des projets pour le développement logiciel

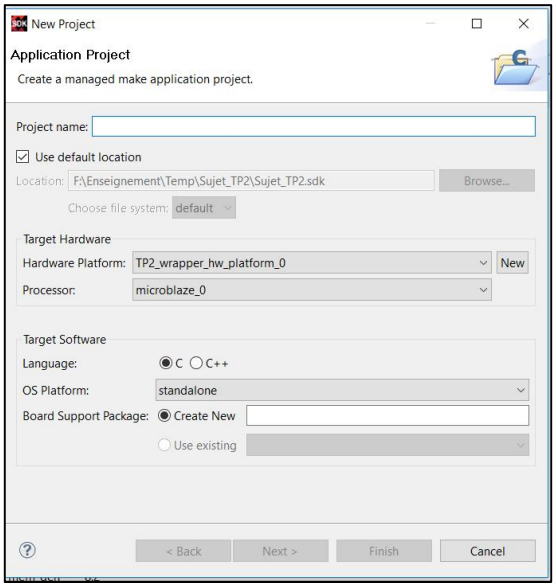
- Après avoir importé les informations de Vivado, ***SDK*** ouvre l'interface suivante



- Pour faire tourner une application logicielle sur la plate-forme ***Microblaze***, ***SDK*** va devoir s'appuyer sur 3 projets :
    - Un projet matériel correspondant à l'export réalisé sous ***Vivado***
    - Un projet logiciel dans lequel vous allez écrire le code de l'application
    - Un projet BSP (***Board Support Package***) qui contient les drivers de tous les composants de la plate-forme matérielle. Ce projet permet donc de faire le lien entre les deux autres projets.

- Dans la fenêtre ***Project Explorer***, vous pouvez constater qu'un projet a déjà été créé et est ouvert
    - Il s'agit du projet matériel, qui s'appelle ici ***TP2_wrapper_hw_platform_0***.
    - Il contient un fichier ***HDF***, ouvert dans la partie éditeur. Ce fichier résume les IP présentes dans votre design ainsi que les plages d'adresses qui ont été automatiquement assignées aux périphériques du ***Microblaze***.

- Nous allons à présent créer les deux autres projets.
    - Aller dans ***File*** → ***New*** → ***Application Project*** pour ouvrir la fenêtre ci-contre.
    - Vérifier que les cases Hardware Platform et Processor sont bien remplies avec les noms de votre projet matériel.
    - Dans ***Project Name***, écrire ***TP2***. Le nom du projet BSP s'initialise automatiquement en ***TP2_BSP***
    - Cliquer sur ***Next***. Puis choisir comme template ***Empty Application***. Cliquer sur ***Finish***.
    - Vous constaterez que ***TP2*** et ***TP2_BSP*** ont été ajoutés au ***Project Explorer***.



- Ouvrir le projet ***TP2***. Il est composé de deux sous-dossiers : ***includes*** et ***src***
    - C'est dans ***src*** qu'il faudra ajouter le code C de l'application

- Ouvrir le projet ***TP2_BSP***.
    - Double cliquer sur le fichier ***MSS***. Il fournit pour chaque périphérique la documentation du driver ainsi que quelques exemples d'utilisation qui pourront vous aider à développer vos programmes
    - Ouvrir le répertoire ***microblaze_0*** puis ***includes***. Chercher puis ouvrir le fichier ***xparameter.h***
        - Ce fichier est une suite de directives `#define` qui vont faciliter la programmation du système en associant des noms (relativement) intelligibles aux valeurs des paramètres à utiliser pour réaliser une configuration ou une action particulière.
        - Par exemple, en faisant une recherche (***Ctrl+F***) sur le texte ***GPIO***, vous allez arriver au texte suivant :

```
/* Definitions for driver GPIO */
#define XPAR_XGPIO_NUM_INSTANCES 2

/* Definitions for peripheral BOUTONS */
#define XPAR_BOUTONS_BASEADDR 0x40010000
#define XPAR_BOUTONS_HIGHADDR 0x4001FFFF
#define XPAR_BOUTONS_DEVICE_ID 0
#define XPAR_BOUTONS_INTERRUPT_PRESENT 1
#define XPAR_BOUTONS_IS_DUAL 0

/* Definitions for peripheral LED_SWITCH */
#define XPAR_LED_SWITCH_BASEADDR 0x40000000
#define XPAR_LED_SWITCH_HIGHADDR 0x4000FFFF
#define XPAR_LED_SWITCH_DEVICE_ID 1
#define XPAR_LED_SWITCH_INTERRUPT_PRESENT 1
#define XPAR_LED_SWITCH_IS_DUAL 1
```

- On comprend bien que :
  - Il y a deux instances de périphériques ***GPIO***.
  - Le 1er (***BOUTONS***) a un numéro d'ID qui est 0, peut générer une interruption et n'est pas dual port.
  - Le 2nd (***LED_SWITCH***) a une ID qui est 1, peut générer une interruption et est dual port (1 port LED + 1 port Interrupteurs).
  - Figurent également les adresses de début (***BASEADDR***) et de fin (***HIGHADDR***) de chaque module.

- En descendant un peu plus bas dans la zone des définitions associées au contrôleur d'interruptions, on peut trouver, en plus d'un alias pour le numéro d'ID, les canaux d'interruptions associés à chaque périphérique ***GPIO***
  - Pour le module ***LED_SWITCH*** (canal 0)
    - `XPAR_MICROBLAZE_0_AXI_INTC_LED_SWITCH_IP2INTC_IRPT_INTR 0`
  - Pour le module ***BOUTONS*** (canal 1)
    - `XPAR_MICROBLAZE_0_AXI_INTC_BOUTONS_IP2INTC_IRPT_INTR 1`

- Toujours dans le dossier ***microblaze_0/includes*** de ***TP2_BSP***, chercher puis ouvrir le fichier ***xgpio.h***
  - ***xgpio.h*** déclare toutes les données et fonctions du driver des contrôleurs ***GPIO***.
  - Le code de ce driver a été automatiquement ajouté au ***projet BSP*** car votre plate-forme matérielle comporte au moins un module de ce type.
  - Les drivers de tous les composants présents dans la plate-forme ont ainsi été ajoutés au ***projet BSP*** pour que vous puissiez les utiliser dans le projet logiciel.
  - Ces drivers permettent au développeur de simplifier l'utilisation des périphériques en s'abstrayant en partie du matériel.
    - Le programmeur n'a ainsi plus à écrire ou lire explicitement dans les registres du périphérique.
    - A la place, il fait appel à une fonction du driver, en fournissant éventuellement quelques paramètres, et c'est le driver qui gère l'accès aux registres.

5) Développement de l'application logicielle

- Nous allons à présent écrire un premier programme s'exécutant sur le ***Microblaze***.
  - Dans le projet logiciel ***TP2***, cliquer avec le bouton droit sur le sous-dossier ***src***. Choisir ***New*** puis ***Source file***. Donner un nom à votre programme (avec l'extension ***.c***) et validez.

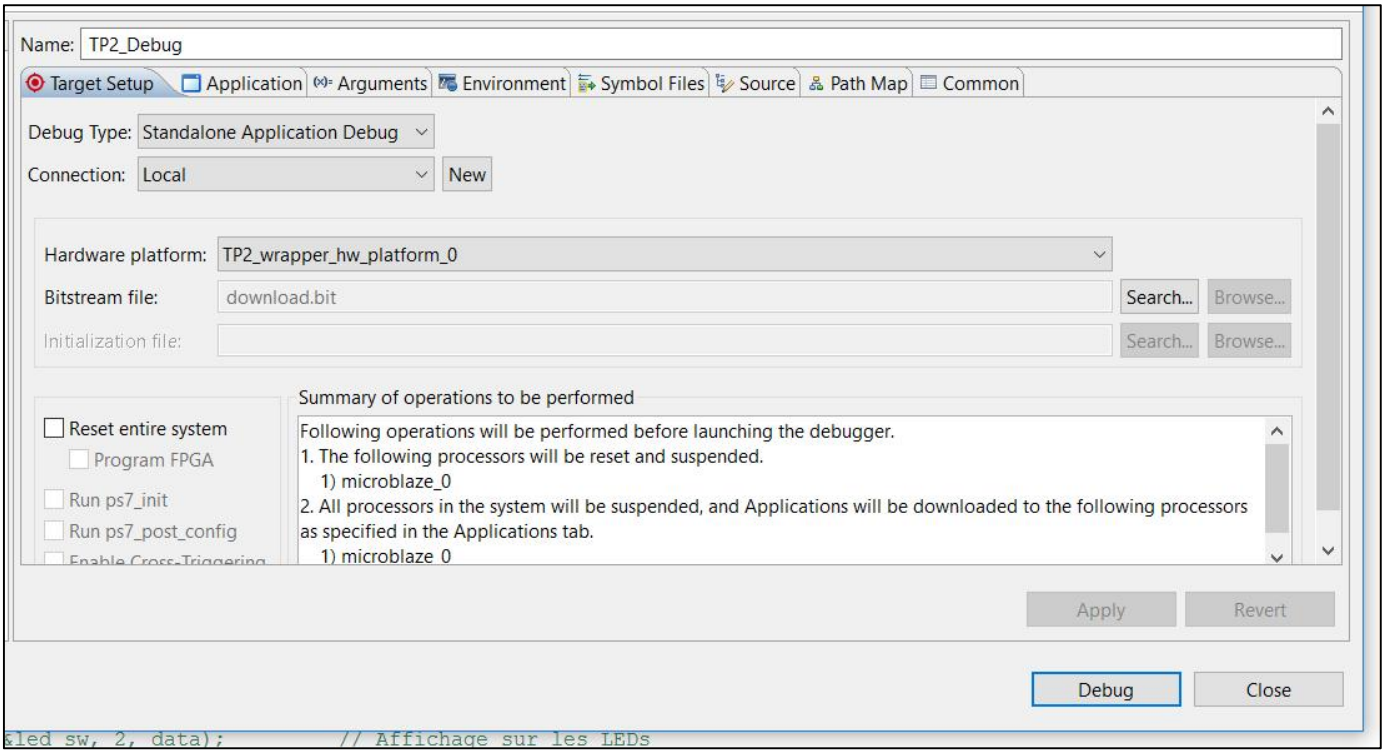
- Dans ce premier programme, nous allons piloter les ***LED*** en fonction de l'état des ***boutons*** et des ***interrupteurs*** de la carte.
  - Nous allons donc utiliser les fonctions du driver de ***GPIO***
  - Pour cela, écrire en tête de votre programme C les directives :
    - `#include "xgpio.h"` et `#include "xparameters.h"`
  - Vous pouvez à présent créer des variables de type ***XGpio***, par exemple : `XGpio led,button;`
    - Ces variables permettent de piloter une instance de contrôleur GPIO.
    - Les fonctions du driver qui utilisent les variables ***XGpio*** sont résumées dans le tableau ci-dessous, en prenant l'exemple d'un périphérique ***USER_GPIO*** possédant 2 ports (le port 1 en sortie et le port 2 en entrée), et piloté par une variable ***my_gpio***.

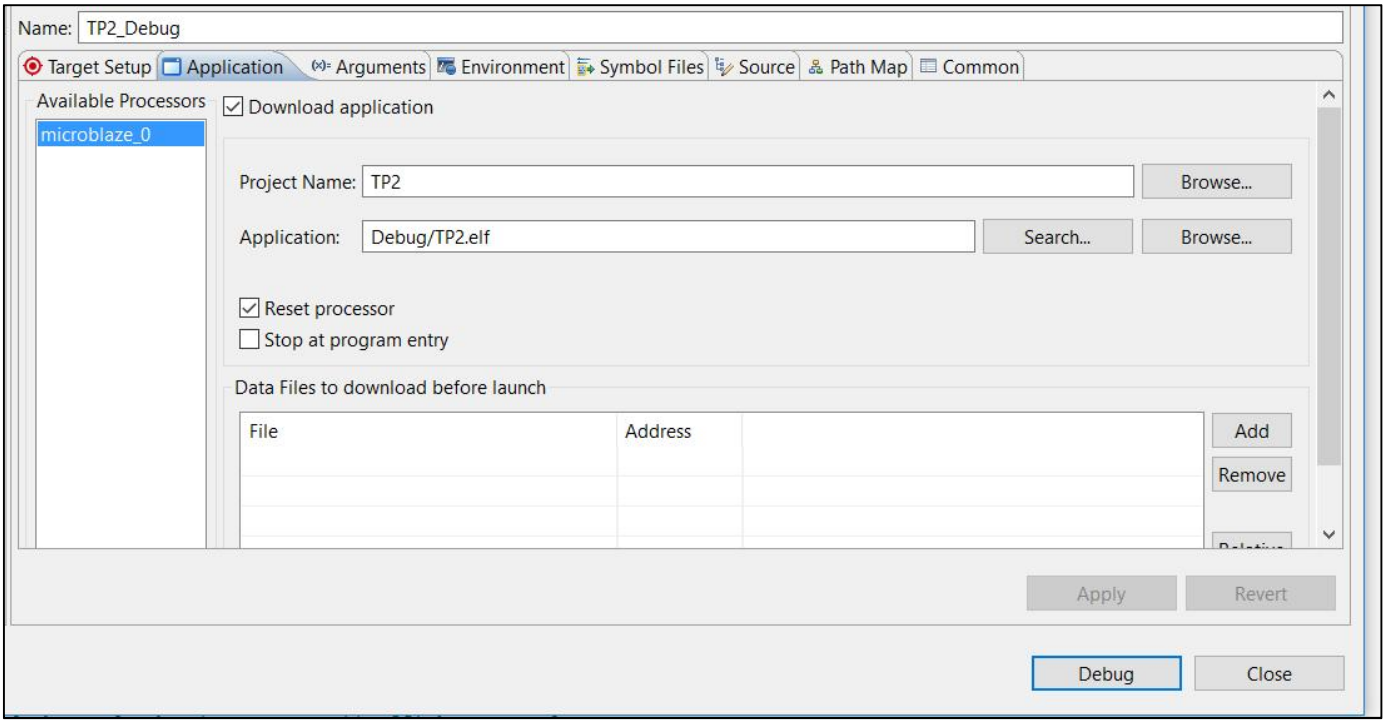
Fonction	Rôle
XGpio_Initialize(&my_gpio, XPAR_USER_GPIO_DEVICE_ID)	Associe la variable led au périphérique LED du système
XGpio_SetDataDirection(&my_gpio,1,0)	Fixe la direction des prots d'E/S du contrôleur GPIO Paramètres de la fonction 1) Indique le périphérique GPIO concerné 2) Indique quel port (1 ou 2) est concerné par l'instruction (Se référer à votre plate-forme matérielle pour savoir à quelles E/S correspondent les ports 1 ou 2) 3) Fixe la direction des broches du port (Bits à 0→sorties / Bits à 1→entrées)
XGpio_DiscreteWrite(&my_gpio,1,data)	Fonction d'écriture vers les ports de sortie du GPIO Paramètres de la fonction 1) Indique le périphérique GPIO concerné 2) Indique quel port (1 ou 2) est concerné par l'instruction 3) Valeur écrite en sortie du périphérique (integer)
XGpio_DiscreteRead(&my_gpio, 2)	Fonction de lecture des ports d'entrée du GPIO Paramètres de la fonction 1) Indique le périphérique GPIO concerné 2) Indique quel port (1 ou 2) est concerné par l'instruction La fonction renvoie un integer (la valeur lue dans le périphérique)

- A l'aide de ce descriptif, écrire un premier programme dans votre fichier C***,*** qui allume une des ***LED*** si on commute un ***interrupteur*** vers le haut, et qui l'éteint si ***l'interrupteur*** est tiré vers le bas.
    - A chaque sauvegarde, le fichier est compilé, ce qui vous permet de corriger d'éventuelles erreurs.

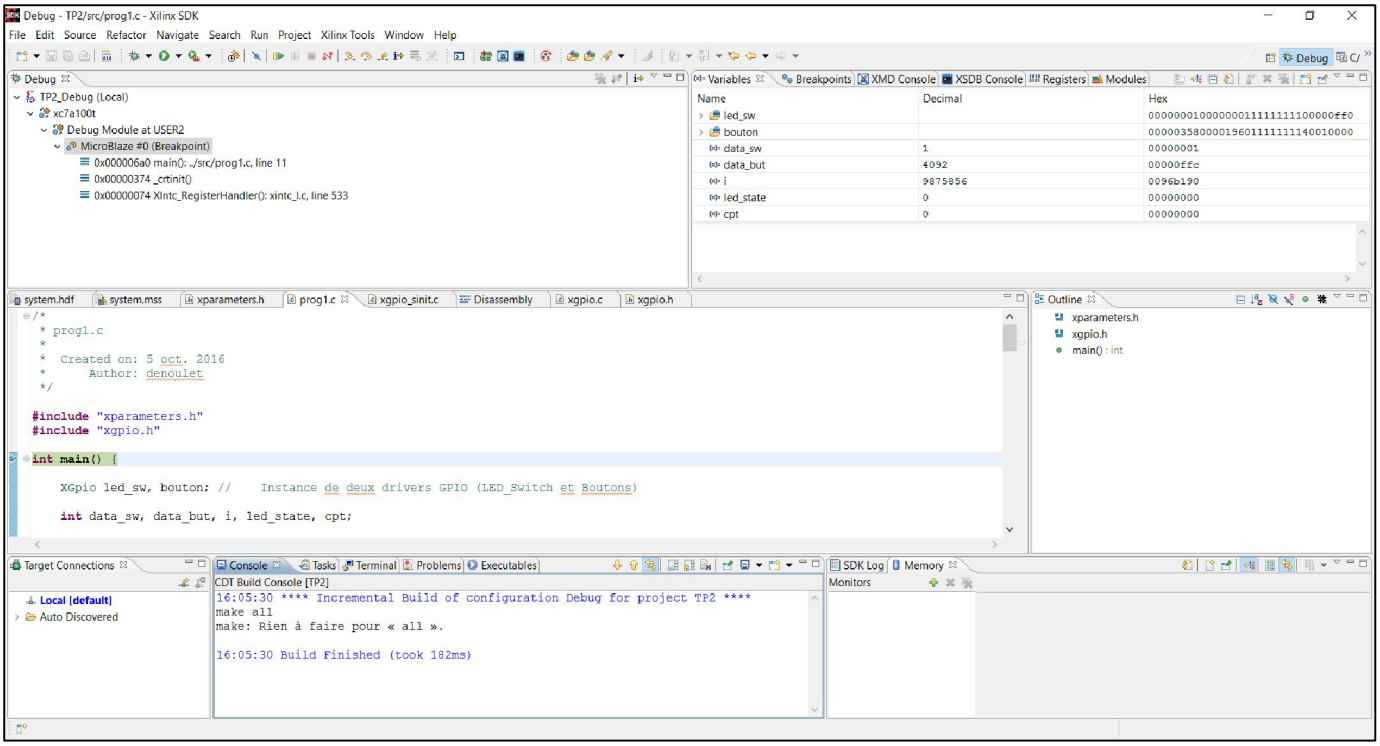
- Pour exécuter le programme, il faut d'abord programmer le ***FPGA*** pour qu'il contienne le système ***Microblaze*** sur lequel sera exécuté le code.
    - Dans le menu ***Xilinx Tools*** sélectionner ***Program FPGA***
        - (Si éventuellement, il y a le choix entre plusieurs plates-formes matérielles, choisir celle sur laquelle vous voulez travailler)
    - Vérifier que les cases ***Bitstream*** et ***BMM/MMI File*** sont correctement remplies. Vérifier également que le microblaze est en mode ***bootloop*** (attente d'un programme à télécharger)
    - NB : Une fois que le FPGA est programmé, il n'y a plus à faire cette étape (du moins tant que la carte est en marche)

- La dernière étape consiste à télécharger le code logiciel dans la mémoire du ***Microblaze***.
    - Pour cela, tirer l'interrupteur 15 vers le bas et l'interrupteur 14 vers le haut afin que le processeur ***Microblaze*** ne soit pas en mode reset (auquel cas on ne pourrait pas le programmer)
        - Cela correspond aux niveaux logiques pour lesquels les ports ***reset*** et ***reset_rtl*** de la plate-forme matérielle ne sont pas actifs (cf. page 5).
    - Aller dans le menu ***Run → Debug Configurations***
        - Double cliquer sur ***Xilinx C/C++ application (System Debugger)*** pour créer une nouvelle configuration.
        - Appeler cette configuration ***TP2_Debug***, puis remplir les onglets ***Target Setup*** et ***Application*** comme indiqué ci-dessous.





- Cliquer sur ***Apply*** puis ***Debug***
  - Cela lance le téléchargement du programme dans le ***Microblaze*** et l'ouverture du débuggeur.
  - Un message vous demande si vous souhaitez ouvrir la ***Debug Perspective***. Cliquer sur ***Yes***
    - Une "***perspective***" sur ***SDK*** est une organisation des fenêtres adaptée à l'outil en cours d'utilisation.
    - La ***perspective*** associée au débuggeur est identique à celle de la figure ci-dessous
    - Si vous souhaitez revenir à la ***perspective*** d'origine (pour par exemple retravailler votre code C, vous pouvez à tout moment cliquer sur le menu ***Window*** puis ***Close Perspective***.



- Vous pouvez exécuter le programme de deux façons

Mode pas à pas → Cliquer sur l'icône	***Mode Run*** → Cliquer sur l'icône
	

- En mode pas à pas, vous aurez accès à l'état et l'évolution de vos variables, ajouter ou retirer des points d'arrêt, etc…

- Vérifier ainsi que votre programme fonctionne correctement.

- Modifier votre programme pour que :
  - Les ***LED*** clignotent si ***l'interrupteur*** 0 est relevé, et qu'elles affichent un motif fixe sinon.
  - Si ***l'interrupteur*** 1 est relevé alors
    - Si on appuie sur le ***bouton Left***, les 4 ***LED*** de gauche s'allument
    - Si on appuie sur le ***bouton Right***, les 4 ***LED*** de gauche s'éteignent
    - Si on appuie sur le ***bouton Center***, on incrémente un compteur modulo 16 qui s'affiche sur les 4 ***LED*** de droite. On réfléchira notamment à la gestion des rebonds des boutons.

6) Utilisation du contrôleur d'interruptions

- Nous allons à présent utiliser les ***boutons poussoirs*** en mode ***interruption***. Nous avons en effet dans ***Vivado*** configuré une sortie interruption pour le contrôleur ***GPIO*** des ***boutons***, et nous avons connecté cette sortie au ***contrôleur d'interruptions*** du ***Microblaze***.

- Cliquer avec le bouton droit sur le projet logiciel ***TP2*** et choisir ***Close Project***.

- Créer un nouveau projet logiciel (***TP2_Exo2*** par exemple) en conservant le même ***projet BSP***.

- Modifier le programme précédent pour que le compteur modulo 16 soit géré par une interruption.
    - Pour cela, vous pourrez vous appuyer sur le programme ***Interrupt_Example.c*** présent sur le disque ***I:\denouletj\4I108 – FPGA1***.
        - Pour plus d'explications sur le code, vous consulter également la documentation des ***drivers GPIO*** et ***INTC*** accessibles depuis le fichier ***.mss*** du projet ***BSP.***
        - Regarder également le ***datasheet PDF*** du ***périphérique GPIO*** afin de comprendre comment sont déclenchées puis gérées les requêtes d'interruption.