Synthèse - Utilisation de Vivado pour l'implémentation matérielle d'algorithmes d'IA et circuits neuromorphiques.

6 janvier 2026

Table des matières

1 Préface

Ce document est une synthèse de quelques conseils pour pouvoir bien utiliser Vivado et tirer parti de certaines fonctionnalités des FPGAs. Il est divisé en trois parties principales :

— La première partie explique comment utiliser les "blocks diagrams" pour créer des designs FPGA rapidement et efficacement.
— La deuxième partie détaille comment paramétrer une mémoire BRAM dans Vivado.
— La troisième partie donne quelques techniques et astuces dans le cadre tu TP sur la création de réseaux de neurones simples sur FPGA.

2 Utilisation des "blocks diagrams" dans Vivado

En plus d'être un outil de synthèse HDL classique pour FPGA, Vivado propose une interface graphique appelée "blocks diagrams" qui permet de créer des designs en assemblant des blocs IP (Intellectual Property) préconçus. Cette approche visuelle facilite la conception rapide de systèmes complexes sans avoir à écrire de macros composants permettant les connexions entre plusieurs composants.

2.1 Création d'un block diagram

Une fois un projet Vivado créé vous pouvez créer un nouveau diagramme en cliquant sur **Create Block Design** dans la section **IP Integrator** de l'onglet **Flow Navigator** (à gauche).

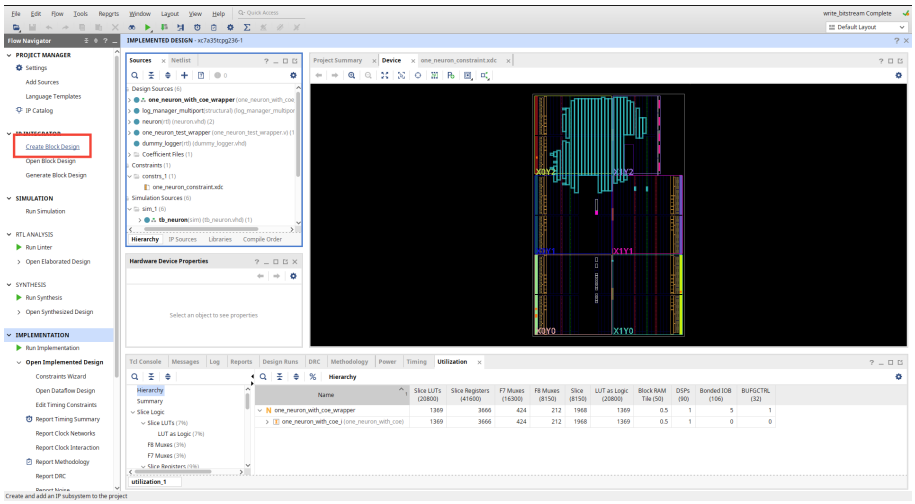

FIGURE 1 – Création d'un block design dans Vivado

Normalement, après avoir nommé votre design, un nouveau plan de travail s'ouvre où vous pourrez ajouter vos blocs et connexions.

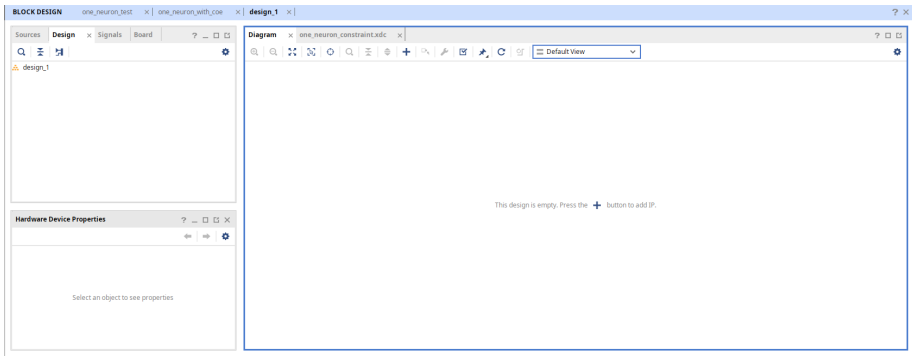

FIGURE 2 – Plan de travail vide d'un block design

2.2 Ajout de blocs IP

Dans votre fenêtre vide vous pouvez ajouter des blocs IP en cliquant sur l'icône "+" en haut (ou en appuyant sur la touche **Ctrl+I**). Une barre de recherche apparaît alors où vous pouvez taper le nom du bloc que vous souhaitez ajouter.

Les blocs IP sont des composants préconçus qui peuvent être utilisé dans votre design. Ces blocs peuvent aller de simples composants logiques (multiplexeurs, concaténation, ...) à des composants plus complexes (processeurs, contrôleurs de mémoire, interfaces de communication, ...).

2.3 Ajout de vos propres blocs

Vous pouvez également créer vos propres blocs IP à partir de vos fichiers HDL (VHDL ou Verilog) en utilisant l'outil **Create and Package IP** dans le menu **Tools**. Cependant, pour des designs simples et pour le prototypage, cette méthode n'est pas toujours nécessaire car elle ajoute une couche de complexité supplémentaire.

Dans votre cas le plus simple sera de simplement *drag and drop* vos fichier VHDL directement dans le block design. Pour ce faire il faudra retourner dans l'onglet **Sources**.

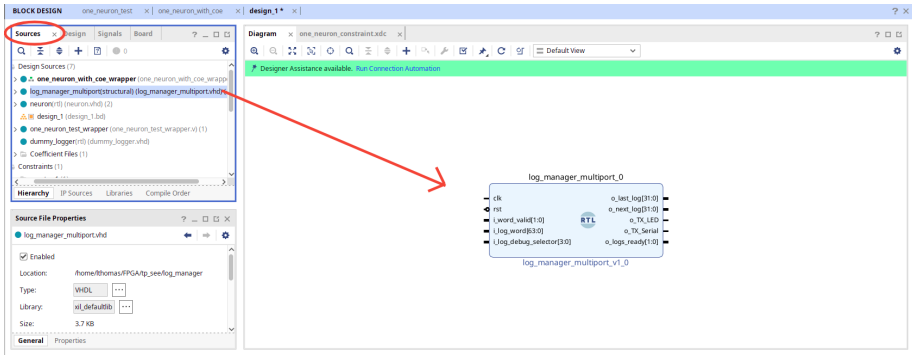

FIGURE 3 – Ajout de fichiers VHDL par drag and drop

2.4 Paramétrer et connecter les blocs

Il est possible de paramétrer les blocs IP en double-cliquant dessus dans le block design. Cela ouvre une fenêtre de configuration où vous pouvez ajuster les paramètres spécifiques du bloc.

Dans vos propres composants, les *in* et *out* ports seront automatiquement affichés dans le block design.

Très important : Uniquement les types STD_LOGIC et STD_LOGIC_VECTOR sont supportés dans les blocks diagrams. Si vous voulez gérer des *arrays* vous devrez les concatener manuellement en STD_LOGIC_VECTOR avant de les mettre en interface.

Les paramètres génériques cependant, ne sont pas contraints au STD_LOGIC. En double-cliquant sur un de vos blocs ce seront ces paramètres-là que vous changerez.

2.5 Création du wrapper HDL

Une fois votre block design terminé, vous devez générer un *wrapper* HDL qui encapsule le block design pour qu'il puisse être utilisé dans le reste de votre projet Vivado. Pour ce faire, faites un clic droit sur votre design et selectionnez **Create HDL Wrapper**.

Dans les faits, cela génerera un fichier VHDL ou Verilog qui instanciera votre block design et exposera ses ports pour qu'ils puissent être connectés à d'autres parties de votre projet. La gestion des ports est expliqué dans la partie suivante.

2.6 Gestion des interfaces/ports d'un design block

Il existe deux moyens principaux pour gérer les interfaces/ports d'un design block dans Vivado :

— Faire un clic droit sur le port d'un de vos blocs et sélectionner **Make External** pour exposer ce port comme une interface externe du block design. L'interface prendra le même nom que le port sélectionné (et y ajoutera un suffixe si nécessaire pour éviter les conflits de noms). Il est possible de renommer l'interface en cliquant dessus et en allant dans ses propriétés (en bas à gauche fig 4).

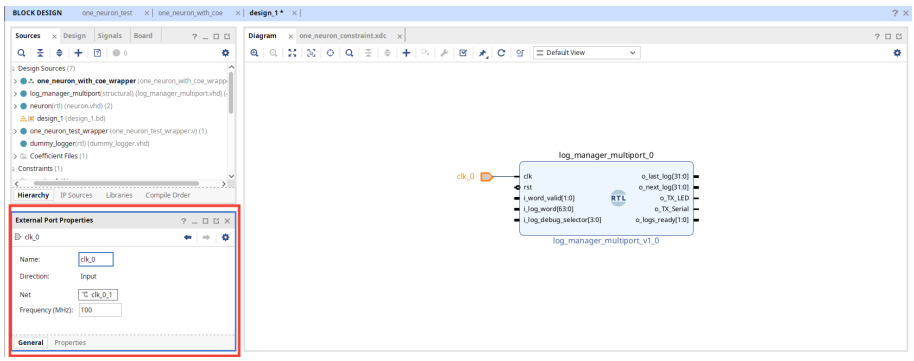

FIGURE 4 – Configurer un port externe

— Faire un clic droit dans le vide et sélectionner **Add Interface Port** pour ajouter un port externe manuellement. Ici vous pourrez paramétrer le nom, le type (input/output), et la largeur du port.

2.7 Intégration dans le projet Vivado

Une fois le wrapper HDL généré vous pouvez intégrer le block design comme un composant classique dans le reste de votre projet Vivado.

Si vous voulez y associer un fichier de contrainte (XDC) vous pouvez le faire de la même manière que pour un composant classique. Cependant, les ports à utiliser correspondront aux interfaces externes que vous avez créées dans le block design (voir section 2.6).

3 Comment paramétrer une mémoire BRAM dans Vivado

La plupart des FPGA modernes intègrent deux types de ressources particulièrement utiles la mémoire en blocs (BRAM) ainsi que les blocs DSP (Digital Signal Processing).

— La BRAM est une mémoire RAM embarquée dans le FPGA qui permet de stocker des données de manière efficace et rapide. Par exemple, la *BASYS 3* dispose de 50 blocs de mémoire BRAM, chacun pouvant stocker jusqu'à 36 Kbits de données. Un bloc BRAM peut être accédé en mode simple ou double port, ce qui permet des opérations de lecture et d'écriture simultanées.

— Les blocs DSP sont des unités de traitement spécialisées conçues pour effectuer des opérations mathématiques comme les multiplications, et ce, de manière très efficace. La BASYS 3 dispose de 90 DSPs capables de réaliser des multiplications 25x18 bits en un seul cycle d'horloge.

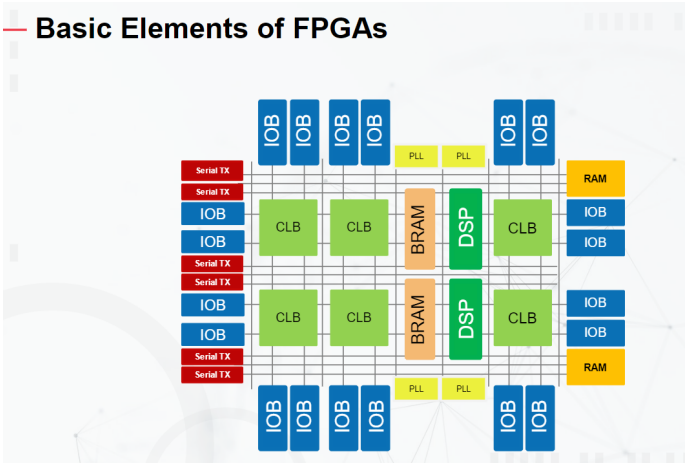

FIGURE 5 – Schéma simplifié d'un FPGA avec des blocs BRAM et DSP

Comme vous pouvez le voir sur la figure 5, les blocs BRAM et DSP sont intégrés proches dans la matrice logique du FPGA, ce qui permet des connexions rapides et efficaces avec les autres ressources logiques.

Normalement, pour utiliser ces ressources dans Vivado, vous devez utiliser des blocs IP spécifiques. Sinon cela peut arriver que le "compilateur" reconnaisse des patterns spécifiques dans votre

code VHDL et décide d'implémenter automatiquement des BRAM ou des DSP pour optimiser les performances de votre design. Cependant, cette reconnaissance automatique n'est pas toujours garantie, et surtout sur des petits designs. Dans cette partie nous allons voir comment forcer l'utilisation de BRAM et DSP dans vos designs Vivado.

3.1 Utilisation des BRAM dans Vivado

3.1.1 Forcer l'utilisation de BRAM dans un code VHDL

Pour forcer l'utilisation de BRAM dans votre code VHDL, vous pouvez utiliser des attributs spécifiques qui indiquent au synthétiseur de mapper certaines structures de données vers des blocs BRAM.

Très important : Il est important de faire un composant autour qui imite bien le comportement de la BRAM (lecture synchrone).

Voici un exemple de code VHDL qui utilise un tableau pour stocker des données et force l'utilisation de BRAM :

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity rom_comp is
6      port(
7          clk    : in  std_logic;
8          addr   : in  unsigned(2 downto 0);
9          dout   : out std_logic_vector(7 downto 0)
10     );
11 end rom_comp;
12
13 architecture behavioral of rom_comp is
14
15     type rom_type is array (0 to 7) of std_logic_vector(7 downto 0);
16     signal rom : rom_type := (
17         0 => x"0F",
18         1 => x"C4",
19         2 => x"70",
20         3 => x"04",
21         4 => x"8C",
22         5 => x"46",
23         6 => x"0D",
24         7 => x"D6"
25     );
26
27 -- Force Block RAM
28 attribute rom_style : string;
29 attribute rom_style of rom : signal is "block";
30 begin
31     process(clk)
32     begin
33         if rising_edge(clk) then
34             dout <= rom(to_integer(addr));
35         end if;
36     end process;
37 end behavioral;
```

Ici ce sont les lignes *28* et *29* qui forcent l'utilisation de BRAM pour le signal `rom`.

3.1.2 Générer de la BRAM via l'IP block

La méthode plus conventionnelle pour utiliser des BRAM dans Vivado est d'utiliser l'IP block dédiée. Pour ce faire, ouvrez le **IP Catalog** dans Vivado et recherchez l'IP nommée **Block Memory Generator**.

Cela vous ouvrira une fenêtre de configuration où vous pourrez définir les paramètres de la mémoire BRAM, tels que la taille, le type d'accès (simple ou double port) ... Pour débloquer plus d'options il faut passer le mode de **BRAM Controller** à **Stand Alone** (premier paramètre en haut de la page **Basic**)

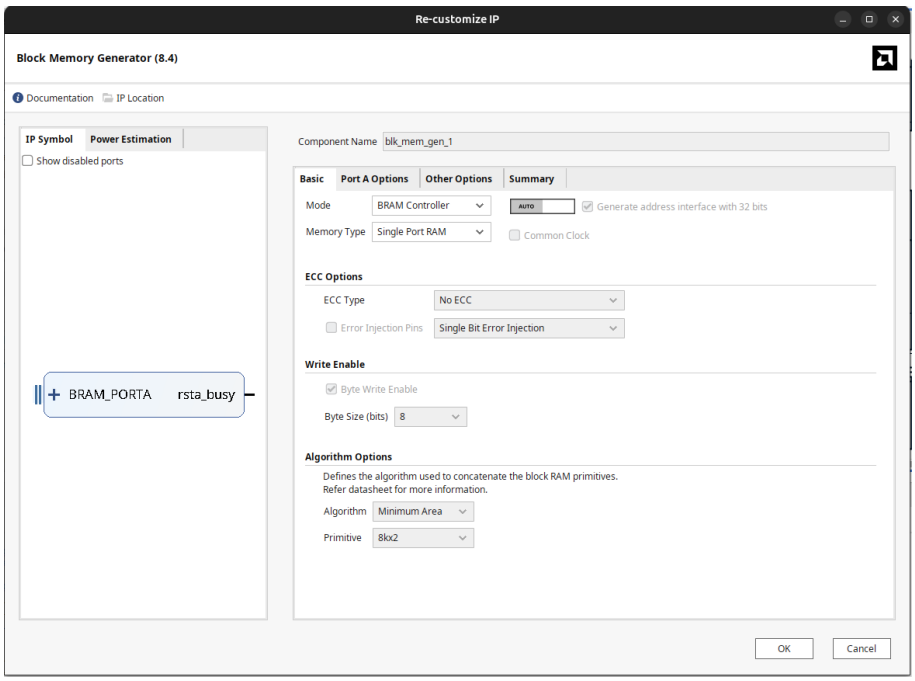

FIGURE 6 – Paramétrage de l'IP Block Memory Generator

Dans **Other options** il est possible d'initialiser votre mémoire à l'aide d'un fichier *.coe*. Cela peut être utile pour charger des poids de réseaux de neurones **Par exemple**.

Voici un exemple de fichier *.coe* :

```
memory_initialization_radix=16;
memory_initialization_vector=11 BE 7B 04 81 4D 0E D2;
```

Le *radix* correspond à la base des valeurs dans le vecteur d'initialisation (ici en hexadécimal). Le *vector* est la liste des valeurs initiales séparées par des espaces.

L'IP block générera alors automatiquement la mémoire BRAM avec les paramètres choisis et l'initialisation spécifiée (il vérifiera aussi que votre vecteur tient bien dans la mémoire que vous avez spécifié).

3.2 Utilisation des DSP dans Vivado

3.2.1 Forcer l'utilisation de DSP dans un code VHDL

Pour forcer l'utilisation de DSP dans votre code VHDL, vous pouvez utiliser ces lignes là :

```
architecture behavioral of multiplier is
  attribute use_dsp : string;
  signal mul_res : signed(15 downto 0);
  attribute use_dsp of mul_res : signal is "yes";
begin
  mul_res <= signed(IN1) * signed(IN2);
end behavioral;
```

4 Technique par rapport au TP

4.1 Composant de debug

Lors de la création de designs FPGA, il est souvent nécessaire de vérifier le comportement interne du circuit. Normalement nous utilisons des simulations pour cela mais il est aussi intéressant de voir ce qui se passe réellement sur le FPGA. Pour cela vous aurez à disposition un composant de debug appelé *log_manager_multiport* qui vous permettra de capturer des signaux de 32 bits et de les transmettre sur votre ordinateur via UART.

Voici la liste des paramètres à modifier :

— **Baud Rate** : Le baud rate que vous utiliserez pour communiquer en UART.

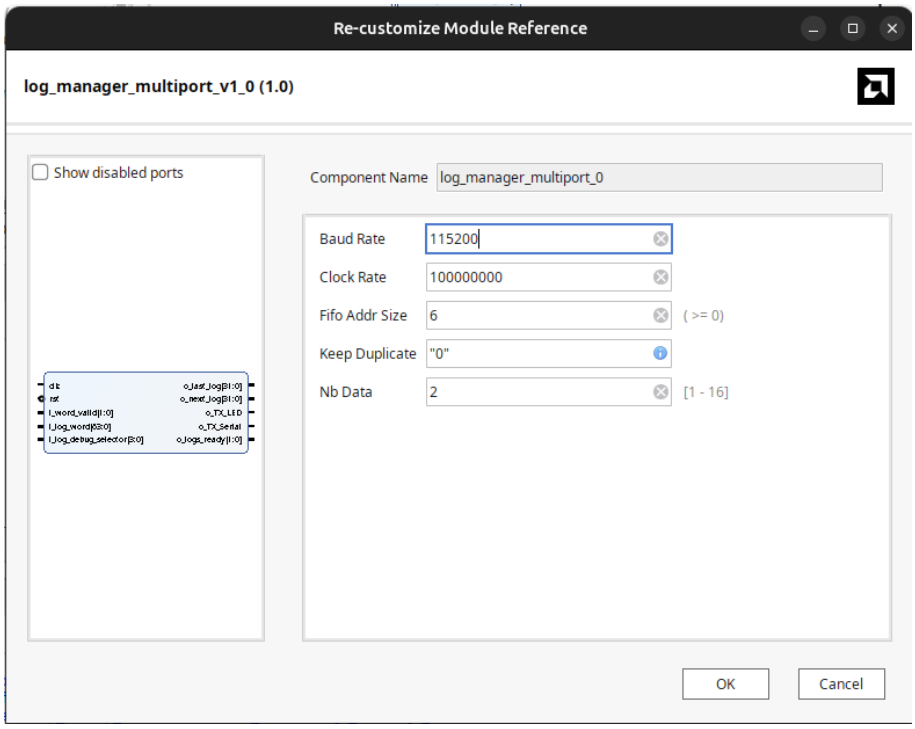

FIGURE 7 – Composant de debug log_manager_multiport dans un block diagram

— **Clock Rate** : La fréquence de la clock que vous utilisez.
— **Fifo Addr Size** : La taille du buffer qui stocke les entrées en attente d'être loguées.
— **Keep Duplicate** : Si activé ('1'), le composant loguera toutes les entrées même si elles sont identiques à la précédente.
— **Nb Data** : Le nombre de ports d'entrée de 32 bits (signaux à logger) que vous souhaitez utiliser.

Le composant dispose de ces entrées :

— **clk** : La clock principale du design **(doit avoir la même fréquence que vous avez spécifié en paramètres)**.
— **rst** : Un signal de reset asynchrone actif haut.
— **i_word_valid** : Un signal d'activation pour chaque port d'entrée (largeur = Nb Data).
— **i_log_word** : Les signaux à logger (largeur = Nb Data × 32 bits).
— **i_log_debug_selector** : Un signal de sélection pour choisir quel port d'entrée ira dans *o_last_log* et *o_next_log*.

Le composant dispose de ces sorties :

— **o_last_log** : le dernier log debuggé (32 bits).
— **o_next_log** : le prochain log debuggé (32 bits).
— **o_TX_LED** : Un signal qui indique l'état de la transmission UART (1 = en cours de transmission).
— **o_TX_Serial** : La sortie UART.
— **o_logs_ready** : un signal indiquant si les buffers de logs sont pleins et prêts à être transmis (1 = prêt).

Dans le fichier de contrainte vous deverez assigner la sortie *o_TX_Serial* à la broche UART de la carte FPGA. Pour la *BASYS3* on aurait :

```
1    set_property PACKAGE_PIN A18 [get_ports o_TX_Serial] ;
2    set_property IOSTANDARD LVCMOS33 [get_ports {o_TX_Serial}]
```

Pour envoyer une donnée vous devrez activer le bit correspondant dans *i_word_valid* et mettre la donnée à envoyer sur le port *i_log_word*.

Quand vous mettez Nb Data à 2 ou plus, il faudra concatener les signaux à logger dans *i_log_word* et les signaux de validation *i_word_valid*. Vous pouvez utilsier l'IP block *Inline Concatenate* pour cela.

A noter que les trames de 32 bit sont elles mêmes transmises par paquets de 8 bits, en little endian. Il est possible donc que vous receviez les octets dans un ordre différent de celui attendu. Il

est donc conseillé de faire un script afin de mieux lire les différentes trames.

Vous pourrez aussi trouver la trame **0x10101010**. Qui arrive une fois tous les buffers de transmission vidés. C'est un bug mais c'est devenu une feature intéressante pour savoir quand la transmission est terminée !