



TP 5 IP core

Systèmes Numériques

Auteurs :

Marc ROTEN

Vincent VONLANTHEN

Professeur :

Nicolas SCHROETER

0/3



Table des matières

1	Introduction	5
2	Mode d'emploi	5
3	Création de l'IP : 4x7 segments	10
3.1	Structure des registres	10
3.2	Rendre visible slv_regx dans l'entité	10
3.3	Compléter en instanciant le composant 4x7 segments	11
3.4	Lien des registre AXI4-Lite avec driver 4x7 segments	13
4	Validation de l'IP avec l'interface	14
4.1	Création de la transaction	14
4.2	Lancement de la transaction	14
4.3	Analyse de Data de la transaction	14
5	Introduction à AXI4-Lite	15
5.1	Réaliser des chronogrammes en lançant des commandes TCL	15
5.1.1	Transaction d'écriture	15
5.1.2	Transaction de lecture	18
6	Validation	19
6.1	Résultat	19
6.2	Améliorations possibles	19
7	Conclusion	20



Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Filière Télécommunications

Systèmes Numériques

Classe T-2ad

Laboratoire 5 : IP core

Noms des étudiants :

Date du laboratoire :

0,3

Objectifs

- Familiarisation avec les IPs cores sous Vivado
- Création de l'IP core 4x7 segments
- Vérification du bon fonctionnement de l'IP avec l'interface JTAG
- Familiarisation avec l'interface AXI4-Lite

Travail

Ce laboratoire a pour objectif de développer l'IP core 4x7 segments et de vérifier son bon fonctionnement avec Vivado et l'interface JTAG. De plus, l'interface AXI4-Lite, qui permet de communiquer avec les registres de l'IP core, sera analysée et documentée.

Montage

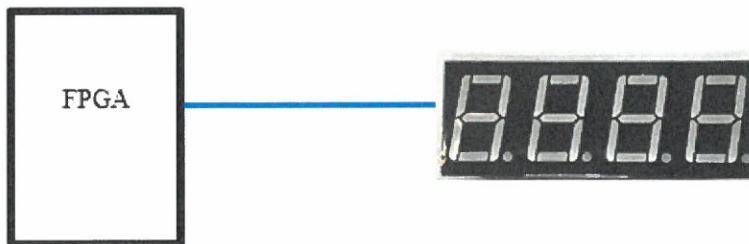


Figure 1 : Montage complet

Etape 1 : Création de l'IP

Le travail à effectuer est

- Proposer une définition de structure de registres pour l'IP 4x7 segments
- Créer l'IP dans Vivado
- Développer les différents composants VHDL de l'IP :
 - o Dans le composant de l'interface AXI4-Lite, rendre visible les registres (signaux slv_regx) dans son entité
 - o Compléter le composant principal de l'IP en instantiant le composant Driver4x7segm ; puis lier les registres du composant de l'interface AXI4-Lite avec le Driver4x7segm.
- Packager l'IP
- Documenter toutes ces étapes dans un mode d'emploi.

Etape 2 : Validation de l'IP avec l'interface JTAG

Afin de valider l'IP, un nouveau projet Vivado va être créé et son block design sera le suivant :

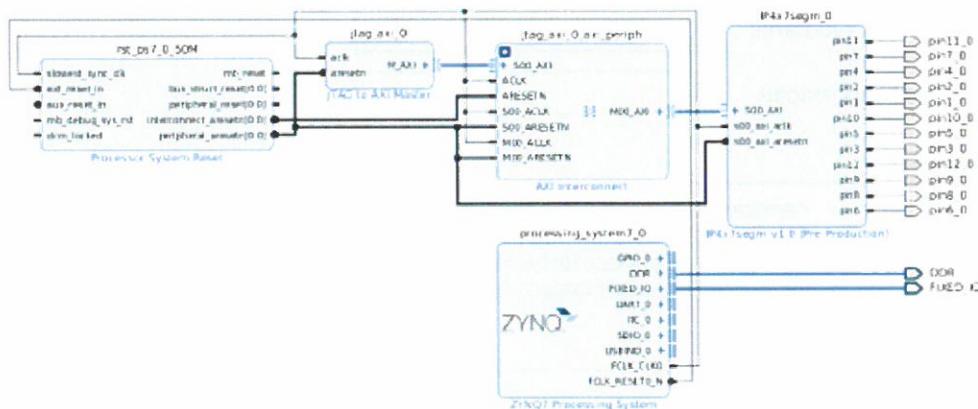


Figure 2 : Design de test

L'IP « jtag_axi_0 » permet de communiquer avec l'IP 4x7segments.
A l'aide de commandes TCL (voir le document pg174), il est possible de générer des transactions AXI d'écriture et de lecture. Ces transactions correspondront aux stimuli de tests pour l'IP 4x7 segments.

Cette étape correspond à la validation de l'IP:

- Créer un nouveau projet Vivado et produire le block design décrit ci-dessus ; Vivado ajoute de lui-même l'IP AXI Interconnect si l'IP « jtag_axi » est connecté directement à l'IP 4x7 segments
- Rechercher dans le document pg174 la description des commandes TCL permettant d'effectuer des transactions de lecture et d'écriture
- Décrire vos commandes TCL pour vos tests
- Exécuter vos commandes TCL et vérifier le bon fonctionnement de l'IP 4x7 segments
- Faire valider par le professeur le bon fonctionnement sur la Minized.

Etape 3 : Introduction à AXI4-Lite

Cette étape a pour objectif de se familiariser avec l'interface AXI4-Lite.

Ainsi, il est demandé

- d'ajouter une sonde ILA pour mesurer les signaux AXI4-Lite
- d'effectuer quelques chronogrammes en lançant les commandes TCL définies à l'étape 2
- documenter et expliquer les chronogrammes

Résultat demandé

- Fournir un rapport par groupe dans un délai d'1 semaine

Ce rapport doit comprendre :

- cette feuille de donnée avec les noms des étudiants et la date du TP
- une courte introduction
- documenter le travail réalisé :
 - les codes VHDL commentés et écrits selon les bonnes pratiques
 - le mode d'emploi de la création de l'IP sous Vivado
 - les commandes TCL pour effectuer des transactions de lecture et d'écriture
 - les chronogrammes de quelques transactions avec explications
- l'indication si votre IP est fonctionnel
- description des améliorations possibles
- une courte conclusion
- les signatures des étudiants

1 Introduction

Dans ce travail pratique nous allons créer notre premier IP sous Vivado. L'IP que nous allons créer et packager est l'**IP 4x7 segments**.

Vu que ce TP sera le premier dans lequel nous allons faire un IP, nous allons donc documenter en faisant un mode d'emploi vivado de comment créer un IP de bout-en-bout.

+ AXI4-lite

2 Mode d'emploi

Il faut tout d'abord, en Figure 1 créer un projet, puis aller dans Tools>Create and package new IP

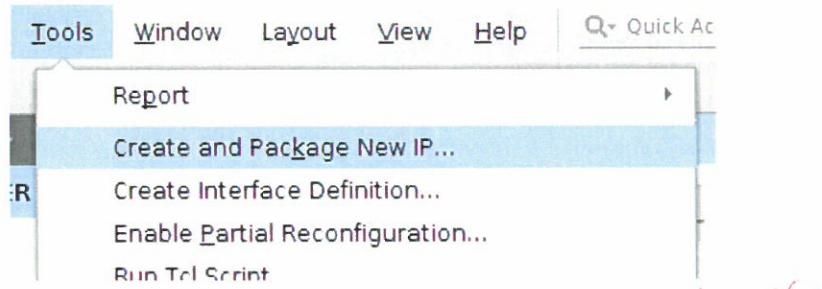


FIGURE 1 – Etape 1 Crédit de l'IP

Il faut ensuite, en Figure 2 laisser décocher les Packaging options et cocher **Create a new AXI4 peripheral**

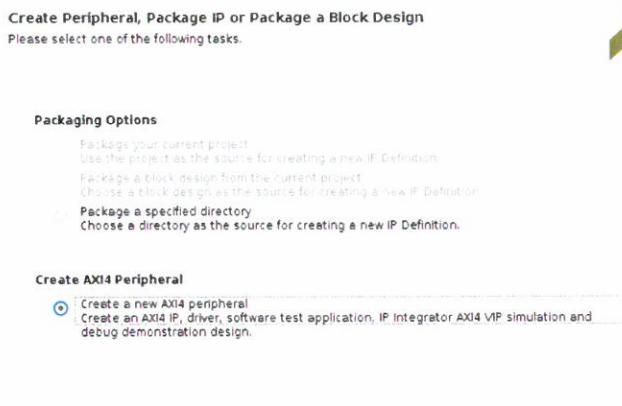


FIGURE 2 – Etape 2

Option pour créer des interfaces AXI4

Il nous faut ensuite, en Figure 3 renommer notre IP, indiquer sa version et cocher la case Overwrite existing

⚠ Déjà d'écaser

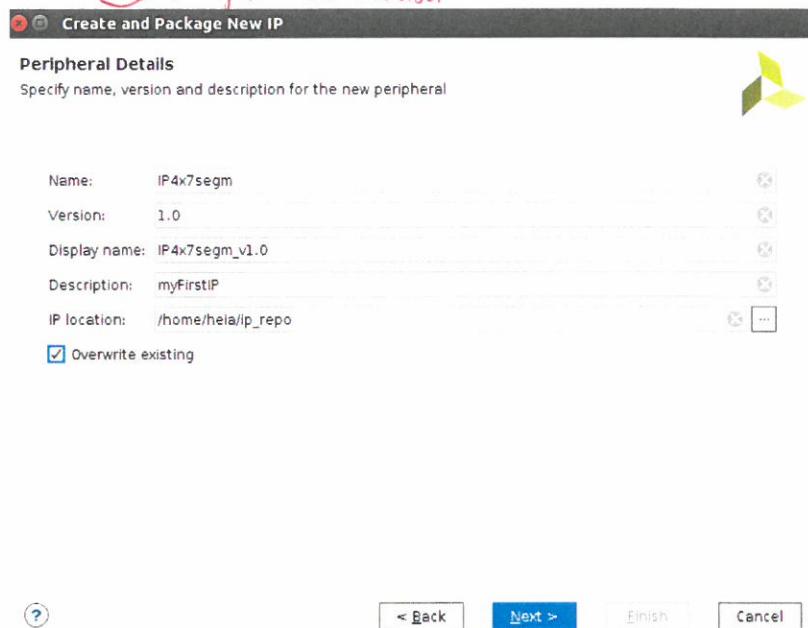


FIGURE 3 – Etape 3

Ensuite en Figure 4, on ne va pas changer d'options, 32 bits sont suffisants pour les cas simples.

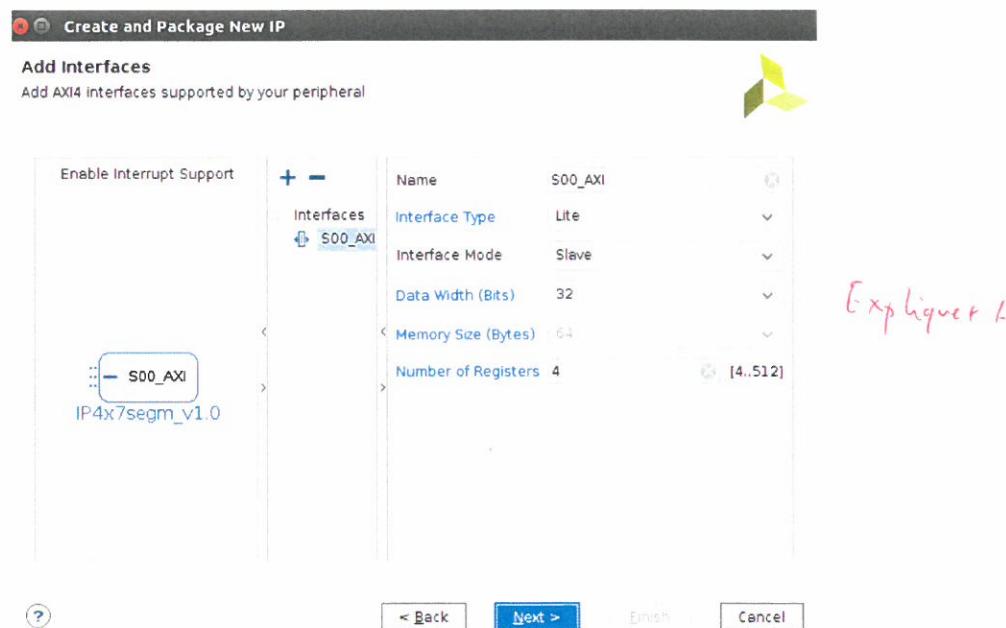


FIGURE 4 – Etape 4

Mise à page

Sélectionnez ensuite comme indiqué en Figure 5 l'option Edit IP>Finish

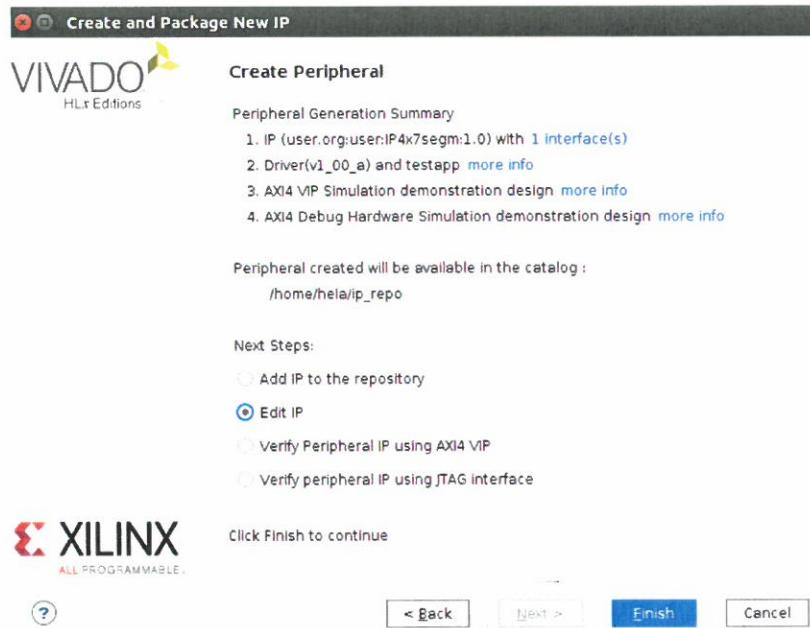
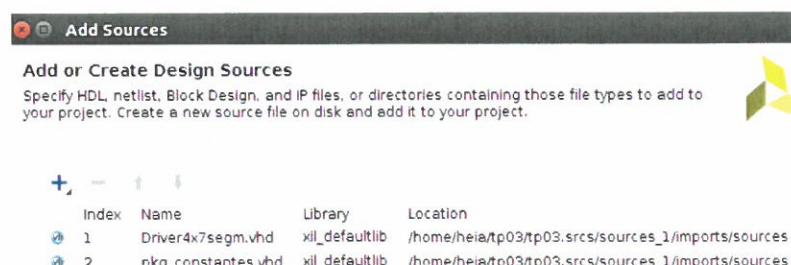


FIGURE 5 – Etape 5

Si notre IP nécessite différents package, ou différents composants, dans le cas de ce tp, le package des constantes et le composant driver4x7segments comme indiqué en Figure 6



FIGURE 6 – Import Sources



Add or Create Design Sources

Specify HDL, netlist, Block Design, and IP files, or directories containing those file types to add to your project. Create a new source file on disk and add it to your project.

Index	Name	Library	Location
1	Driver4x7segm.vhd	xil_defaultlib	/home/heia/tp03/tp03.srcc/sources_1/imports/sources
2	pkg_constants.vhd	xil_defaultlib	/home/heia/tp03/tp03.srcc/sources_1/imports/sources



FIGURE 7 – Import Sources

Il faut ensuite adapter le code VHDL du topmodule et du composant Slave. D'une fois cette opération terminée, nous pouvons packager notre IP, comme montré en Figure 8, il faut vérifier que les ports que l'on veut mettre en sortie de notre **minized** apparaissent.

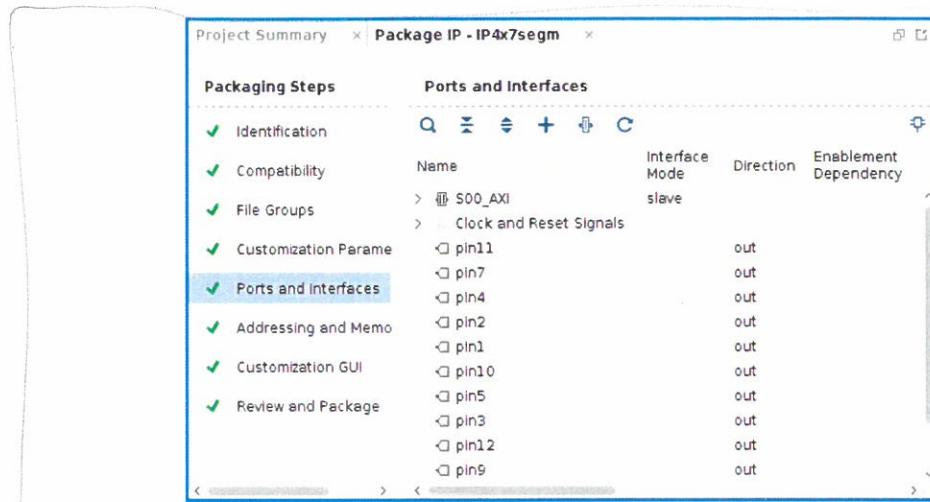


FIGURE 8 – Etape 6

Si les ports n'apparaissent pas, une erreur doit se trouver au niveau du topmodule ou des composants internes, si les ports n'apparaissent pas, vérifier le code.

Dès que cette étape est validée, on peut packager notre IP, procéder comme suit en figure 9.

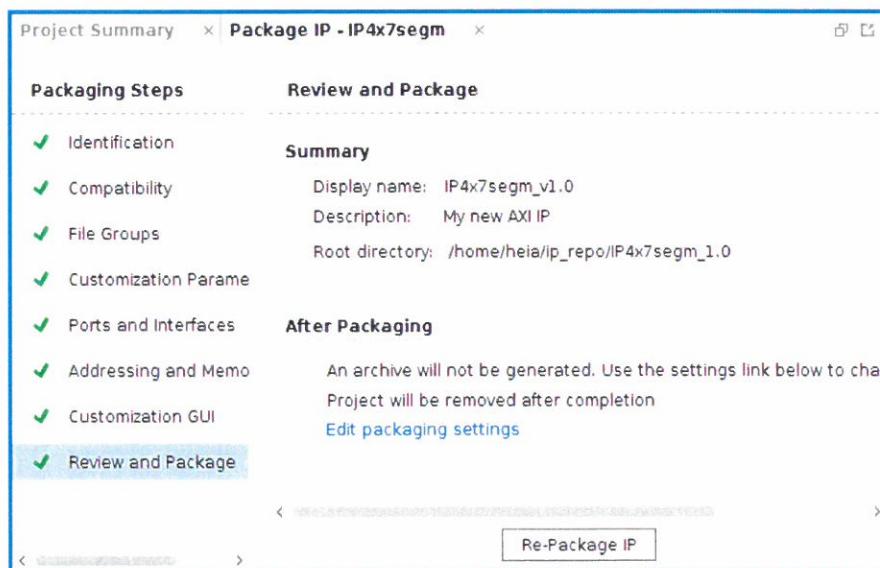


FIGURE 9 – Etape 7

Pour utiliser notre IP, nous l'importons après avoir créer notre block design, à la même manière que notre ZYNQ, voir Figure 10

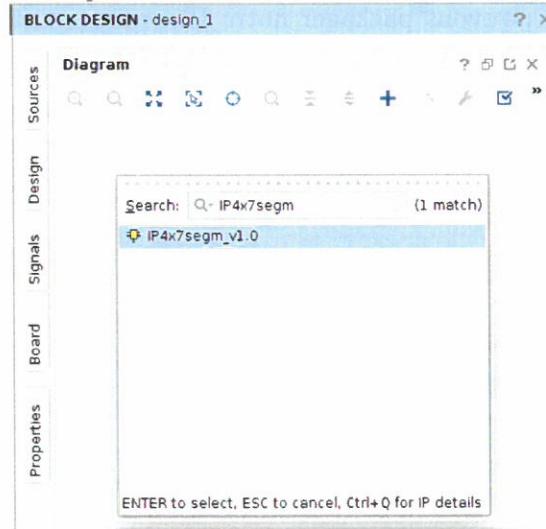


FIGURE 10 – Etape 8

Insérez ensuite l'IP JTAG to axi Master. Cliquez ensuite sur *Run Block Automation* comme suit en Figure 11

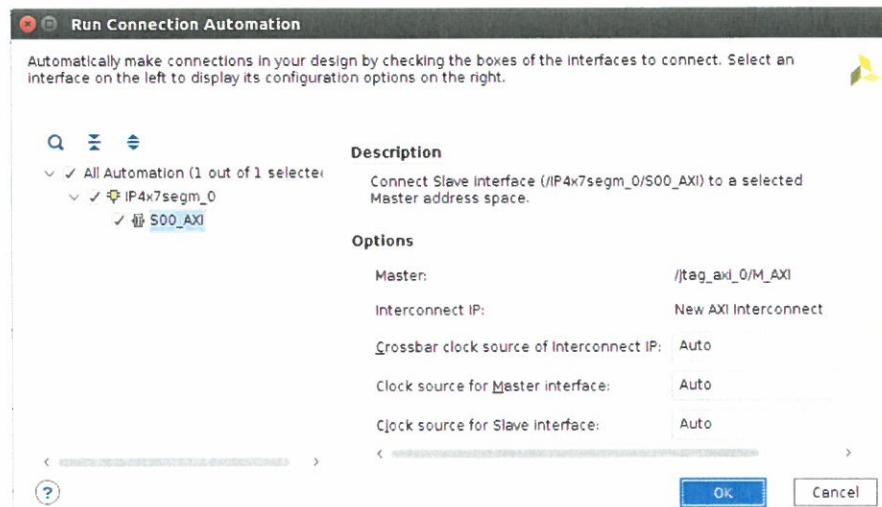


FIGURE 11 – Run Block automation

Avec l'outil Vivado, lorsqu'on lance *Run Block Automation* avec un IP avec une interface Slave et l'IP *JTAG to axi master*, le composant AXI interconnect vient se greffer à notre design.

Expliquer comment se fait la mise à jour dans un projet.

3 Création de l'IP : 4x7 segments

Dans cette partie nous allons aborder les différents points importants à prendre en compte avant de continuer.

En Figure 12, on retrouve le schéma bloc, on y retrouve les éléments importants tels que le composant driver 4x7 segments. On retrouve en sortie du topmodule : les 12 pins de sortie du 4x7 segments, on met aussi en sortie du topmodule le registre slv_reg0, représenté ici par la flèche entrante *TCL*.

On rajoute aussi un *AXI-INTERCONNECT*, utile si l'on dispose de plusieurs IP. Dans le cas où l'on se trouve avec une interface *JTAG* et 2 ou plus d'IP, une autre flèche partirait de l'*AXI-INTERCONNECT* vers l'interface Slave de notre second IP.

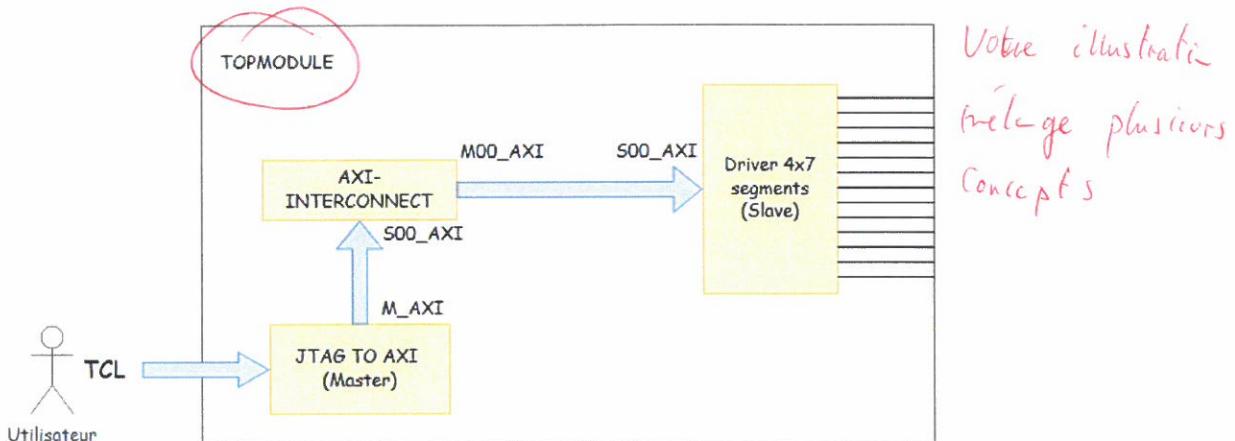


FIGURE 12 – Schéma bloc

3.1 Structure des registres

Vu que l'IP à déployer est relativement simple et léger, et que les bus mis à disposition est de 32 bits, nous sommes en mesure de faire passer l'intégralité des données nécessaires sur un seul ~~bus~~
register. Nous avons choisi le bus slv_reg0.

Nous avons choisi de répartir les données comme indiqué en figure 13

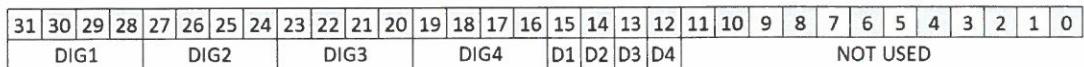


FIGURE 13 – Registre slv_reg0

3.2 Rendre visible slv_regx dans l'entité

```

port (
    -- Users to add ports here
    registre0: out std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    -- User ports ends
    -- Do not modify the ports beyond this line
    ...
    -- Add user logic here
    registre0 <= slv_reg0;
    -- User logic ends
end arch_imp;

```

ligne de la
position dans
le code source

3.3 Compléter en instanciant le composant 4x7 segments

```

port (
    -- Users to add ports here
    pin1, pin7, pin4, pin2, pin1, pin10, pin5, pin3 : out std_logic;
    pin12, pin9, pin8, pin6: out std_logic;

    -- User ports ends
    -- Do not modify the ports beyond this line
    ...

architecture arch_imp of IP4x7segm_v1_0 is
    signal regOutTopModule : std_logic_vector(C_S00_AXI_DATA_WIDTH -1 downto 0);
    signal nrst:std logic; Pas décrit

    -- component declaration
    component Driver4x7segm is
        Port ( clk : in STD LOGIC;
                rst : in STD_LOGIC;
                dig1, dig2, dig3, dig4 : in STD_LOGIC_VECTOR (3 downto 0);
                dot1, dot2, dot3, dot4 : in STD LOGIC;
                pin1, pin7, pin4, pin2, pin1, pin10, pin5, pin3 : out std_logic;
                pin12, pin9, pin8, pin6: out std_logic);
    end component;
    ...
begin
    ...
    -- Add user logic here
    myFirstIP: Driver4x7segm
        Port map ( clk      => s00_axi_aclk,
                   rst      => nrst,
                   dig1     => regOutTopModule(C_S00_AXI_DATA_WIDTH-1 downto
                                              C_S00_AXI_DATA_WIDTH-4),
                   dig2     => regOutTopModule(C_S00_AXI_DATA_WIDTH-5 downto
                                              C_S00_AXI_DATA_WIDTH-8),
                   dig3     => regOutTopModule(C_S00_AXI_DATA_WIDTH-9 downto
                                              C_S00_AXI_DATA_WIDTH-12),
                   dig4     => regOutTopModule(C_S00_AXI_DATA_WIDTH-13 downto
                                              C_S00_AXI_DATA_WIDTH-16),
                   dot1    => regOutTopModule(C_S00_AXI_DATA_WIDTH-17),
                   dot2    => regOutTopModule(C_S00_AXI_DATA_WIDTH-18),
                   dot3    => regOutTopModule(C_S00_AXI_DATA_WIDTH-19),
                   dot4    => regOutTopModule(C_S00_AXI_DATA_WIDTH-20),
                   pin1    => pin1,
                   pin2    => pin2,
                   pin3    => pin3,
                   pin4    => pin4,
                   pin5    => pin5,
                   pin6    => pin6,
                   pin7    => pin7,
                   pin8    => pin8,
                   pin9    => pin9,
                   pin10   => pin10,
                   pin11   => pin11,

```

il



```
    pin12    =>  pin12
);
-- User logic ends
end arch_imp;
```

3.4 Lien des registre AXI4-Lite avec driver 4x7 segments

```
architecture arch_imp of IP4x7segm_v1_0 is
begin
    regOutTopModule : std_logic_vector(C_S00_AXI_DATA_WIDTH -1 downto 0);
    nrst:std_logic;
    ...
    component IP4x7segm_v1_0_S00_AXI is
        generic (
            C_S_AXI_DATA_WIDTH : integer      := 32;
            C_S_AXI_ADDR_WIDTH : integer      := 4
        );
        port (
            registre0      : out std_logic_vector(C_S_AXI_DATA_WIDTH -1 downto 0);
            S_AXI_ACLK     : in std_logic;
            S_AXI_ARESETN  : in std_logic;
            S_AXI_AWADDR   : in std_logic_vector(C_S_AXI_ADDR_WIDTH-1 downto 0);
            ...
        );
    begin
        nrst <= not s00_axi_aresetn;
        -- Instantiation of Axi Bus Interface S00_AXI
        IP4x7segm_v1_0_S00_AXI_inst : IP4x7segm_v1_0_S00_AXI
            generic map (
                C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
                C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
            )
            port map (
                registre0      => regOutTopModule,
                S_AXI_ACLK     => s00_axi_aclk,
                S_AXI_ARESETN  => s00_axi_aresetn,
                ...
            );
    end;
end;
```

4 Validation de l'IP avec l'interface

Ci-après, on retrouve les commandes d'écriture et de lecture

4.1 Création de la transaction

utilisé
`create_hw_axi_txn wr_txn_lite [get_hw_axis hw_axi_1] -address 00000000 -data 12346000 -type write`

create_hw_axi_txn : Create hardware AXI transaction, étape nécessaire pour toute création de transaction.

wr_txn_lite : C'est simplement le nom de la transaction.

-address : C'est l'adresse du registre sur lequel on va executer notre commande TCL, pour le registre slv_reg0, l'adresse est 00000000 *Le votre registre ne se trouve pas à cette adresse*

-data : C'est les data que l'on va envoyer à l'adresse mentionnée précédemment.

-type write : On précise là si on se trouve face à une transaction d'écriture ou de lecture.

4.2 Lancement de la transaction

`run_hw_axi wr_txn_lite`

run_hw_axi : run hardware axi, étape nécessaire, il faut pas uniquement créer la transaction, il faut aussi la lancer

wr_txn_lite : C'est simplement le nom que l'on a donné à notre transaction

4.3 Analyse de Data de la transaction

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

FIGURE 14 – Commande

- **De 1 à 4** : Contenu qui sera affiché sur les digits(1 pour 1, 2 pour 2, etc... et F pour éteindre le digit)
- **5** : Utilisé pour la gestion des dots. Pour savoir quels dots s'allument, il suffit de voir le tableau de correspondance de la Figure 15 sachant que le premier bit est égal au premier dot et ainsi de suite.

N° commande	Dots allumés sur le 4x7 segments
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

FIGURE 15 – Tableau de correspondance entre la commande et le(s) dot(s) allumé(s)

5 Introduction à AXI4-Lite

5.1 Réaliser des chronogrammes en lançant des commandes TCL

5.1.1 Transaction d'écriture

On a disposé 2 sondes ILA, pour la connexion JTAG to axi master <=> interconnect et interconnect <=> Driver 4x7 segments, respectivement slot 0 et slot 1.

Commande TCL pour le premier chronogramme :

```
create_hw_axi_txn transactionWrite [get_hw_axis hw_axi_1] -address 00000000 -data DABA2222
-type write
run_hw_axi_txn transactionWrite
```

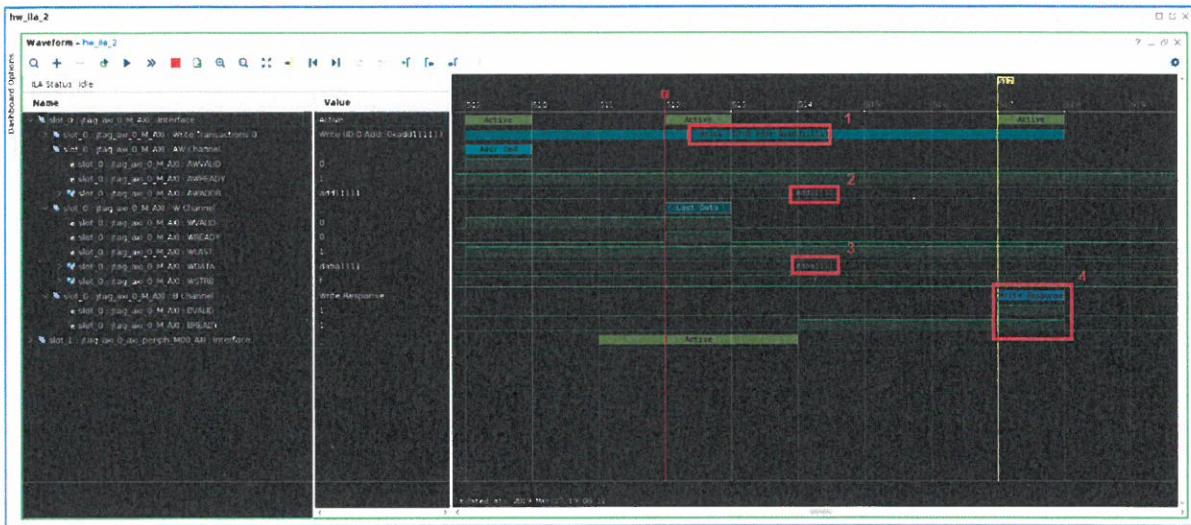


FIGURE 16 – JTAG - Interconnect Write

- **Point 1 :** Indique que l'on se trouve dans une étape d'écriture des données à une certaine adresse, *0xadd1111* sur la Figure 16.
- **Point 2 :** Il s'agit de l'adresse, en hexadécimal, à laquelle l'information est écrite. Il s'agit de l'adresse *0xadd1111* sur la Figure 16.
- **Point 3 :** Il s'agit de l'information qui est écrite et envoyée au slave pour qu'elle puisse être affichée sur le 4x7 segments. Pour simplifier la lecture et la compréhension de nos captures, DABA1111 pour DATA1 et 0xadd1111 pour l'adresse 1.
- **Point 4 :** Ici on se trouve dans le channel *Write Response*. C'est dans ce dernier qu'il est indiqué si la réponse du slave est valide ou non. Le signal *m_axi_bvalid* prend la valeur 1 lorsque la réponse est valide sinon la valeur 0. Le signal *m_axi_bready* prend la valeur 1 lorsque le maître est prêt à recevoir une réponse sinon la valeur 0. Dans notre cas toute la partie réponse est positive.

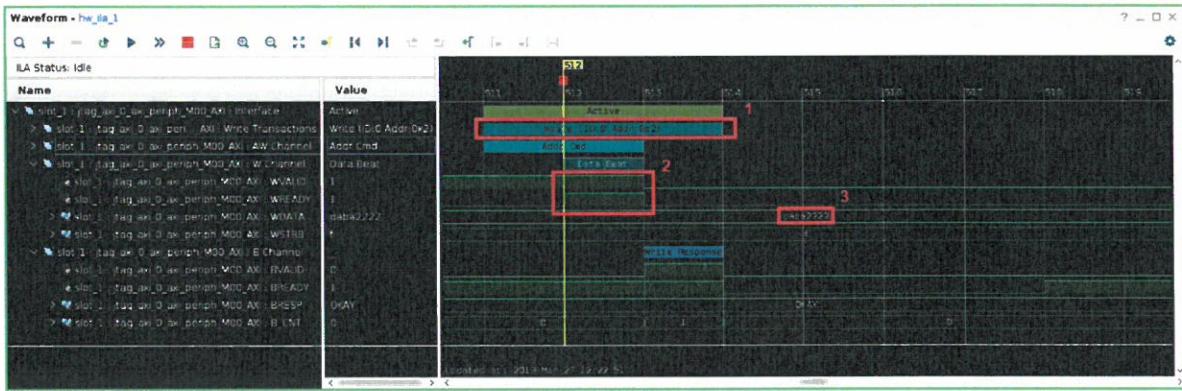


FIGURE 17 – Interconnect - 4x7 segments Write

- Point 1 :** Indique que l'on se trouve dans une étape d'écriture des données à une certaine adresse.
- Point 2 :** Ici on a les signaux qui nous indiquent si l'adresse à laquelle on veut écrire l'information est valide et si la cible est prête à recevoir cette adresse. Pour la première partie, c'est le signal *m_axi_awvalid* qui prend la valeur 1 si l'adresse est valide et sinon la valeur 0. Pour la deuxième partie, c'est le signal *m_axi_awready* qui prend la valeur 1 quand la cible est prête à recevoir l'adresse indiquée et sinon la valeur 0.
- Point 3 :** Il s'agit de l'information qui est écrite et envoyée au slave pour qu'elle puisse être affichée sur le 4x7 segments. Dans le cas de la Figure 17, pour simplifier la lecture et la compréhension de nos captures, DABA2222 pour DATA2.

✓



5.1.2 Transaction de lecture

La commande que nous lancerons pour lire le registre slv_reg0 est la suivante :

```
create_hw_axi_txn rd_txn_lite [get_hw_axis hw_axi_1] -address 00000000 -type read  
run_hw_axi rd_txn_lite
```

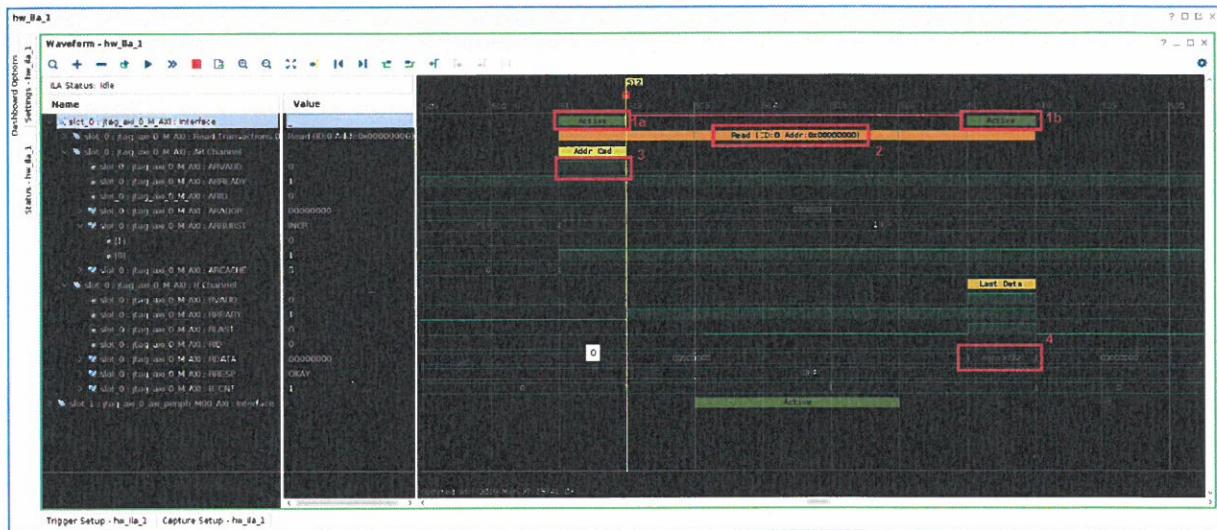


FIGURE 18 – JTAG - Interconnect READ

- **Point 1a & 1b :** Il s'agit des balises qui indique que l'interface AXI est active lors de la lecture du bus de données.
- **Point 2 :** Indique que l'on se trouve dans une étape de lecture des données à une certaine adresse. Sur la Figure 18, on peut observer que l'adresse est la *0x00000000*.
- **Point 3 :** Le signal *m_axi_arvalid* indique si l'adresse à laquelle on lit les données est valide. Si c'est les cas, alors le signal prend la valeur 1 sinon la valeur 0.
- **Point 4 :** Il s'agit du bus de donnée qui est lu par le slave. Sur la Figure 18 les données sont les suivantes : **databus2222**.



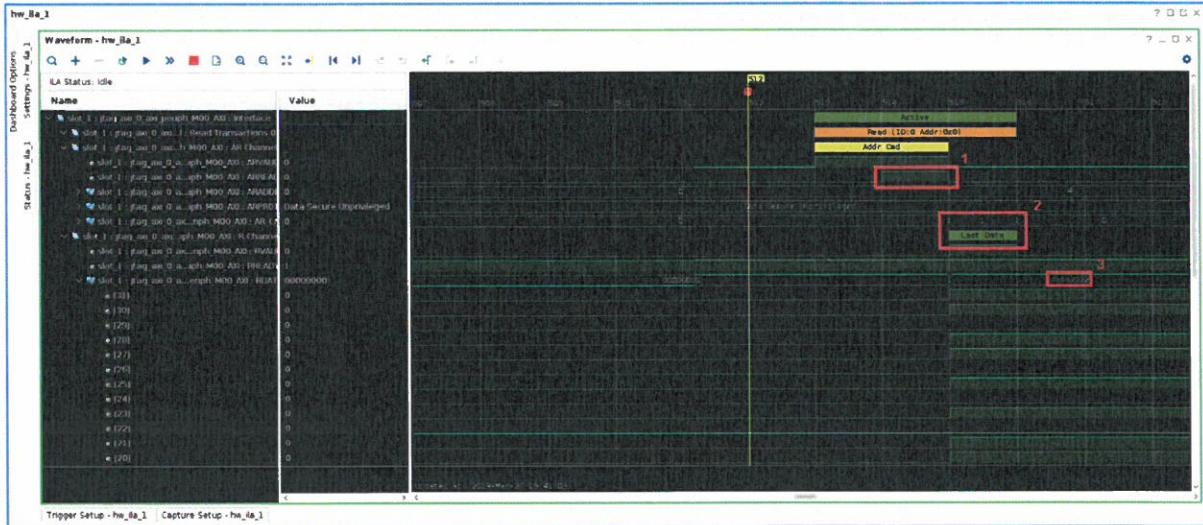


FIGURE 19 – Interconnect - 4x7 segments READ

- **Point 1 :** Le signal *m_axi_arready* indique si la cible est prête à recevoir l'adresse mentionnée ci-dessus. Si la cible est prête alors le signal prend la valeur 1 sinon la valeur 0. Dans le cas de la Figure 19, la cible est prête.
- **Point 2 :** Le signal *m_axi_arcache* contient l'indication que l'adresse, où sont écrites les données, se trouve bien dans le cache. Cela indique à quelle position se trouve les données reçues. Au moment de la capture de la Figure 19, il s'agit des dernières données reçues.
- **Point 3 :** Il s'agit du bus de donnée qui est lu par le slave. Sur la Figure 19 les données sont les suivantes : **daba2222**. Les données lues apparaissent correctement sur le bus ..._M00_AXI

6 Validation

6.1 Résultat

Notre IP est totalement fonctionnel pour ce cas précis, comme indiqué en Partie 5.1, le bus AXI4 est fonctionnel, les adresses sont transmises de manière correcte.

Après avoir réalisé le câblage sur le 4x7 segments, on peut, en entrant les commandes TCL, afficher ce que l'on veut sur le 4x7segments.

6.2 Améliorations possibles

- Nous ne voyons pas de grosses améliorations pour cet IP

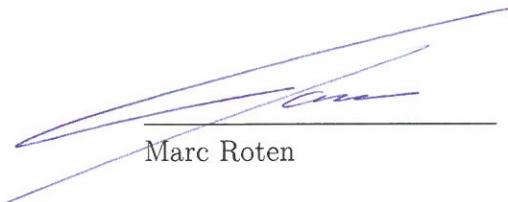
- L'interface AXI4-Lite est fonctionnelle, donc pas d'améliorations possible (selon nous)
- Si nous devions trouver une amélioration possible, rajouter des lettres ou plutôt différents codage pour avoir un éventail plus large de caractère.

7 Conclusion

En conclusion, ce travail pratique nous a bien permis d'assimiler comment créer un IP et comment s'en servir dans un projet. Ce qui nous sera très utile lors du projet de semestre. Nous avons à la suite de ce TP plein d'idées d'IP développable pour la multitudes de composant mis à disposition dans le kit fourni avec la *MiniZed*.

Il nous a en outre permis d'observer et de comprendre comment les informations sont transmises à travers une interface AXI4-Lite, relation maître-esclave, et à quel moment elles sont écrites et lues.

On a aussi mieux compris le fonctionnement du debugging hardware *ILA*



Marc Roten



Vincent Vonlanthen

