Séance 6 :
Introduction aux flux de données (Axi-Stream <-> DMA)

Christophe Jégo

année 2025-2026

BORDEAUX INP Enseirb-Matmeca

Plan de la séance 6 :

L'objectif de cette séance est de :

- Utilisation de flux de données dans une architecture numériques
- Manipulation d'interfaces AXI-Stream dans Vitis HLS
- Exploitation de blocs DMA avec le ZYNQ dans VIVADO
- Réalisation de projets complets basés sur des flux de données

Les différents exercices ont pour objectif respectif :

- Exercice 1: Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS
- Exercice 2: Mise en place d'interfaces AXI4-Stream without side-channels avec Vitis HLS
- Exercice 3: Implémentation d'un filtre FIR générique disponible sous Vivado
- Exercice 4: Implémentation d'un filtre FIR8 avec interface AXI4-Stream synthétisé sous Vitis HLS
- Exercice 5: Remplacement du filtre FIR générique par un filtre FIR27 avec interface AXI4-Stream

Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

Ecrire un programme My_Full_Stream.cpp décrivant le fonctionnement de deux interfaces AXI4-Stream complètes

```
void Full_axi_stream (   hls::stream< ap_axis<32,2,5,6> > &A,
                         hls::stream< ap_axis<32,2,5,6> > &B)
        {
                #pragma HLS INTERFACE axis port=A
                #pragma HLS INTERFACE axis port=B
                #pragma HLS INTERFACE s_axilite port=return

                ap_axis<32,2,5,6> tmp;
                while(1)
                    {
                        A.read(tmp);
                        tmp.data = tmp.data.to_int() + 5;
                        B.write(tmp);
                        if(tmp.last)    {  break;  }
                     }
        }
```

Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

Ecrire un testbench TB_My_Full_Stream.cpp pour tester le fonctionnement des interfaces

```
int main()
 {
   int i=100;
   hls::stream<ap_axis<32,2,5,6> > A, B;
   ap_axis<32,2,5,6> tmp1, tmp2;

  for(int j=0;j<100;j++)
   {
      tmp1.data = i;
      tmp1.keep = 1;
      tmp1.strb = 1;
      tmp1.user = 1;
      if(j=99) { tmp1.last = 1; }
      else     { tmp1.last = 0; }
      tmp1.id = 0;
      tmp1.dest = 1;
   A.write(tmp1);
   Full_axi_stream(A,B);
   B.read(tmp2);

   if (tmp2.data.to_int() != 105)
    { cout << « KO: results mismatch" << endl;
      return 1;  }
    else
    {  cout << "OK" << endl; }

  }

  cout << "Success: results match" << endl;
  return 0;
}
```

Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

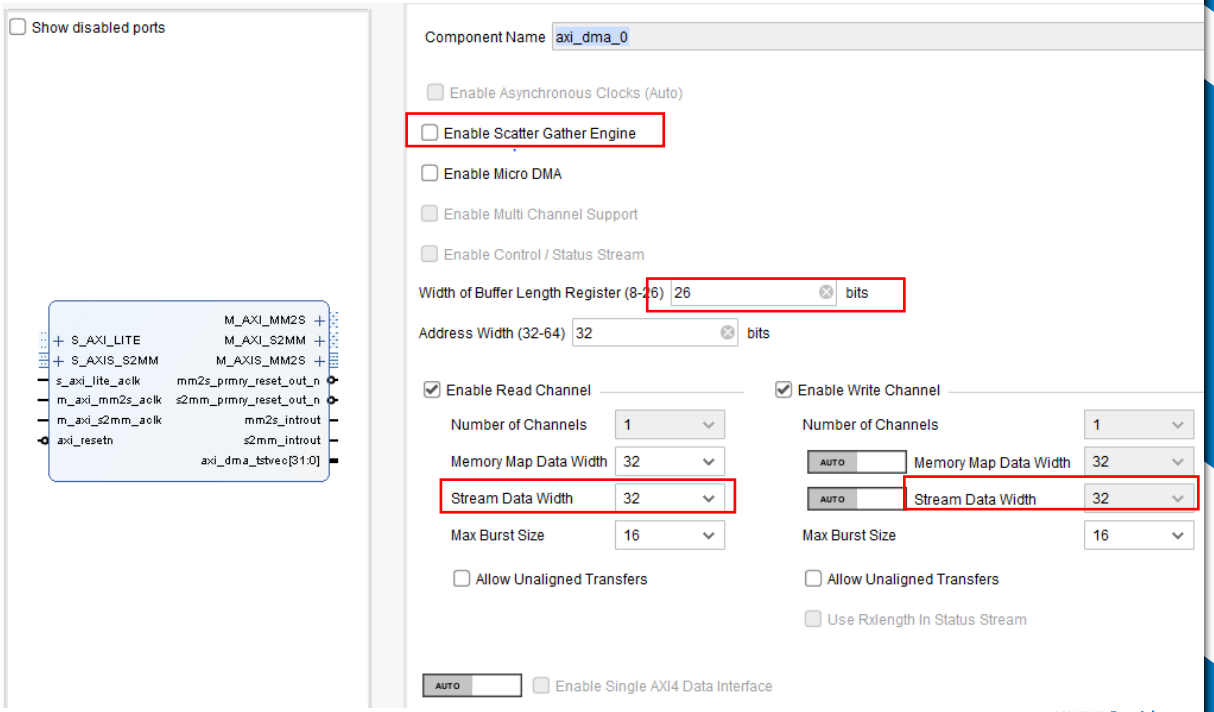
Exécuter l'environnement Vitis HLS :

- Créer un nouveau projet :
  - Nom du projet : My_Full_Stream
  - Top Function : Full_Stream (identique au nom de la fonction décrite dans le programme My_Stream_Int.cpp
  - Ajouter le fichier My_Full_Stream.cpp
  - Ajouter le testbench TB_Full_Stream.cpp
  - Choisir la carte Pynq Z2 comme cible technologique
- Exécuter la simulation C pour vérifier le bon fonctionnement des interfaces
- Exécuter la synthèse C (période : 10 ns, Incertitude : 1 ns)
  - Regarder le rapport de synthèse (nombre de cycles, ressources utilisées)
  - Regarder le diagramme d'ordonnancement des tâches
- Exécuter la co-simulation C pour vérifier le bon fonctionnement des interfaces
- Exécuter l'exportation au niveau RTL dans l'item implémentation

Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

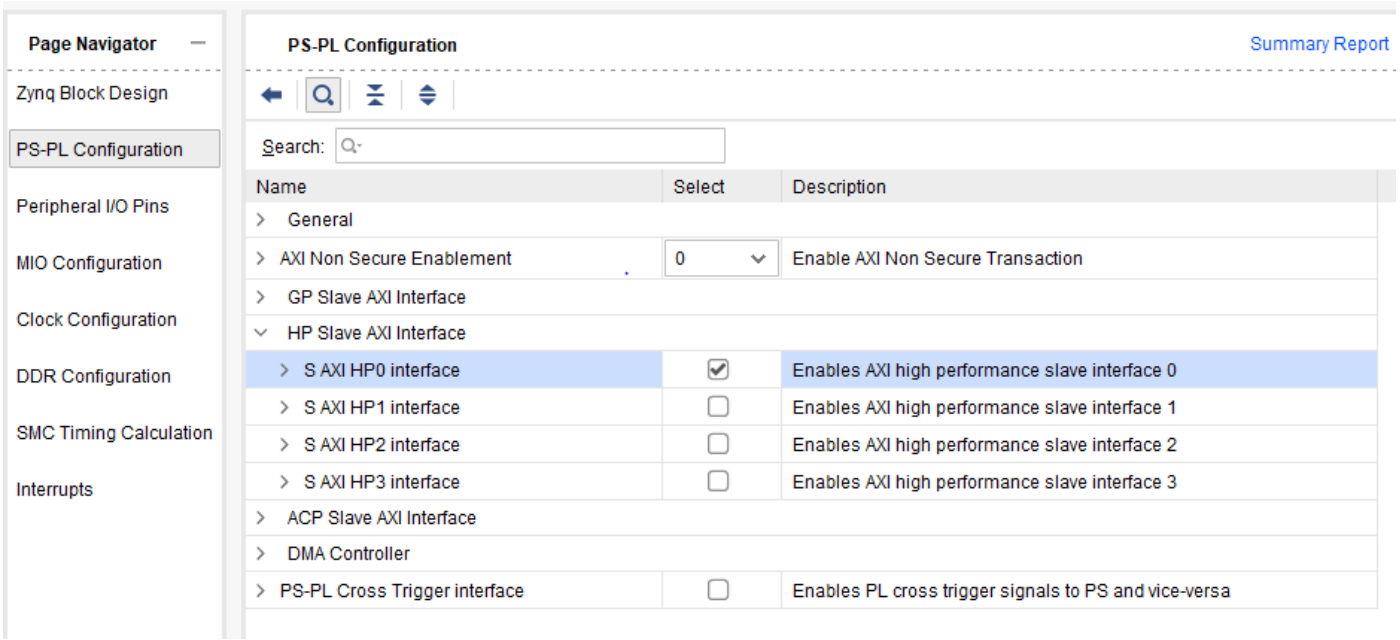
Définir une architecture sous Vivado qui associe une instance d'un processeur Zynq avec un bloc AXI DMA (Direct Memory Access).

Le bloc DMA doit être configuré comme indiqué =>

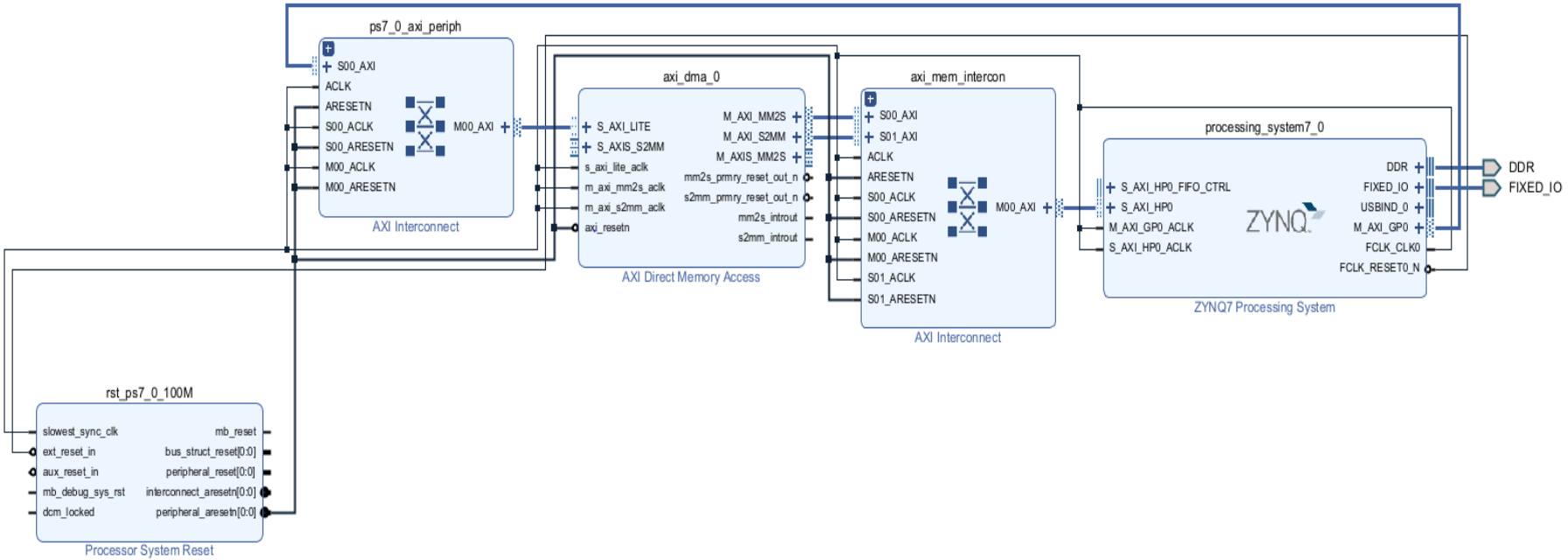


Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

Le processeur Zynq doit être configuré comme suit :



Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS



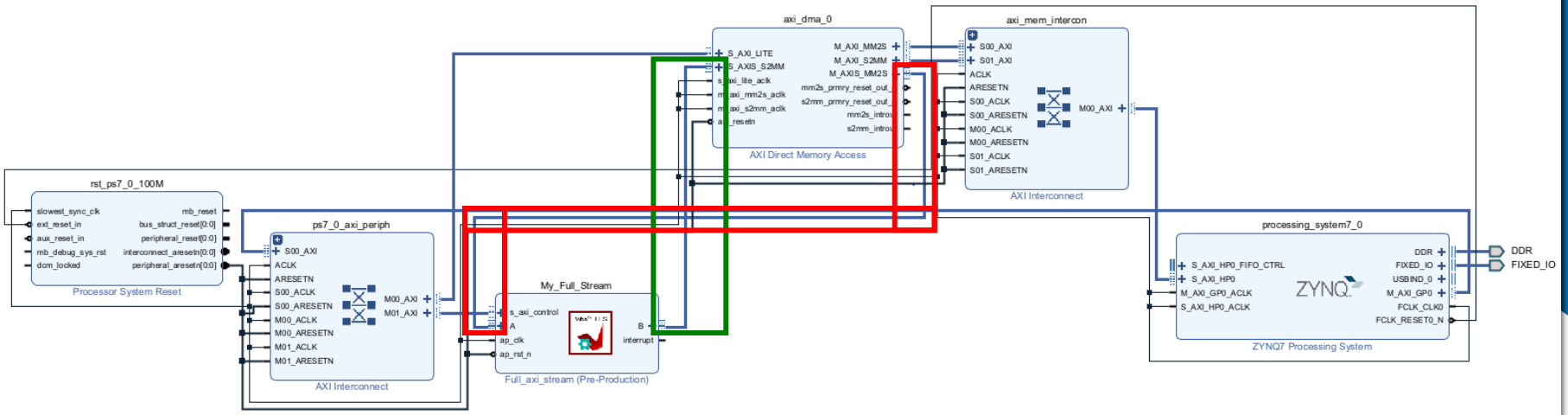
Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

Compléter l'architecture sous Vivado en ajoutant le bloc **My_Full_Stream**

provenant de la synthèse sous Vitis HLS

- Tools -> Settings
- IP -> Repository
- Sélectionner le bloc My_Full_Stream
- Ajouter l'IP au block design
- Ne pas oublier de relier l'entrée et la sortie du bloc My_Full_Stream à celles du module AXI_DMA :
  - M_AXI_MM2S →→→ A
  - B →→→ S_AXI_S2MM

Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS



Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

L'étape suivante consiste à configurer l'accès au bloc My_Full_Stream généré via Vitis HLS

Cela implique de disposer de l'overlay

```
from pynq import Overlay
overlay =Overlay("/home/xilinx/jupyter_notebooks/SE_EN325/S7_Ex1/design_name.bit")
```

Il faut ensuite configurer la plage d'adresses du bloc My_Full_Stream

```
Full_Stream_IP = overlay. design_name
BASE_ADDRESS = overlay.ip_dict['design_name']['phys_addr']
ADDRESS_LENGTH = 0x64000
```

Puis, il faut configurer le module DMA comme suit :

```
DMA = overlay.DMA_name
DMA?
DMA_send = overlay.axi_dma_0.sendchannel
DMA_recv = overlay.axi_dma_0.recvchannel
```

Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

Il est important de récupérer sous VIVADO la plage d'adressage dans le fichier vhdl :

Full_stream_control_s_axi.vhd

```
------------------------Address Info-------------------
-- 0x0 : Control signals
--        bit 0  - ap_start (Read/Write/COH)
--        bit 1  - ap_done (Read/COR)
--        bit 2  - ap_idle (Read)
--        bit 3  - ap_ready (Read/COR)
--        bit 7  - auto_restart (Read/Write)
--        bit 9  - interrupt (Read)
--        others - reserved
-- 0x4 : Global Interrupt Enable Register
--        bit 0  - Global Interrupt Enable (Read/Write)
--        others - reserved
-- 0x8 : IP Interrupt Enable Register (Read/Write)
--        bit 0 - enable ap_done interrupt (Read/Write)
--        bit 1 - enable ap_ready interrupt (Read/Write)
--        others - reserved
-- 0xc : IP Interrupt Status Register (Read/TOW)
--        bit 0 - ap_done (Read/TOW)
--        bit 1 - ap_ready (Read/TOW)
```

Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

L'étape suivante consiste à configurer l'accès au bloc My_Full_Stream généré via Vitis HLS

Il est nécessaire de vérifier l'état du bloc My_Full_Stream

```
Full_Stream_IP.register_map
```

Il faut lancer l'exécution du bloc My_Full_Stream

```
# Activation des bits 0 (ap_start) et 7 (auto_restart)
Full_Stream_IP.write (0x0,0b10000001)
Full_Stream_IP.register_map
```

Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

Objectif 1 : Exécuter successivement les différents programmes du fichier My_Stream.ipynb pour la configuration de l'overlay, la définition de la plage d'adresses, la configuration du module DMA et l'exécution du bloc My_Full_Stream

Objectif 2 : Ecrire un programme Python sous Jupyter. Ce dernier doit permettre l'exécution suivante sur la carte :

- ✓ Allouer 2 buffers de taille 20: (input_buffer et output_buffer);
- ✓ Fixer les valeurs 0 à 19 à input_buffer;
- ✓ Effectuer l'envoi puis la réception de la trame de taille 20;
- ✓ Afficher le contenu de output_buffer;
- ✓ Libérer l'espace mémoire allouée aux deux buffers.

Exercice 2 : Mise en place d'interfaces AXI4-Stream without side-channels avec Vitis HLS

Ecrire un programme My_Lite_Stream.cpp décrivant le fonctionnement de deux interfaces AXI4-Stream assignant que les signaux/bus obligatoires dans le protocole

```
void Lite_Stream(   hls::stream< ap_axis<32,0,0,0> > &A,
                    hls::stream< ap_axis<32,0,0,0> > &B)
        {
                    #pragma HLS INTERFACE axis port=A
                    #pragma HLS INTERFACE axis port=B
                    #pragma HLS INTERFACE s_axilite port=return

                    ap_axis<32,0,0,0> tmp;

                    A.read(tmp);
                    tmp.data = tmp.data.to_int() + 5;
                    B.write(tmp);
        }
```

Exercice 2 : Mise en place d'interfaces AXI4-Stream without side-channels avec Vitis HLS

Appliquer une approche méthodologique similaire à celle de l'exercice 1:

- Ecrire un testbench TB_My_Lite_Stream.cpp;
- Exécuter l'environnement Vitis HLS et comparer l'architecture générée avec celle de l'exercice 1.
- Définir une architecture sous Vivado qui associe une instance d'un processeur Zynq avec un bloc AXI DMA et un bloc My_Lite_Stream
- Configurer sous Jupyter l'accès au bloc My_Full_Stream généré
- Exécuter successivement les différents programmes pour la configuration de l'overlay, la définition de la plage d'adresses, la configuration du module DMA et l'exécution du bloc My_Lite_Stream
- Ecrire un programme Python dans le fichier My_Stream_Lit.ipynb sous Jupyter similaire à celui de l'exercice 1.

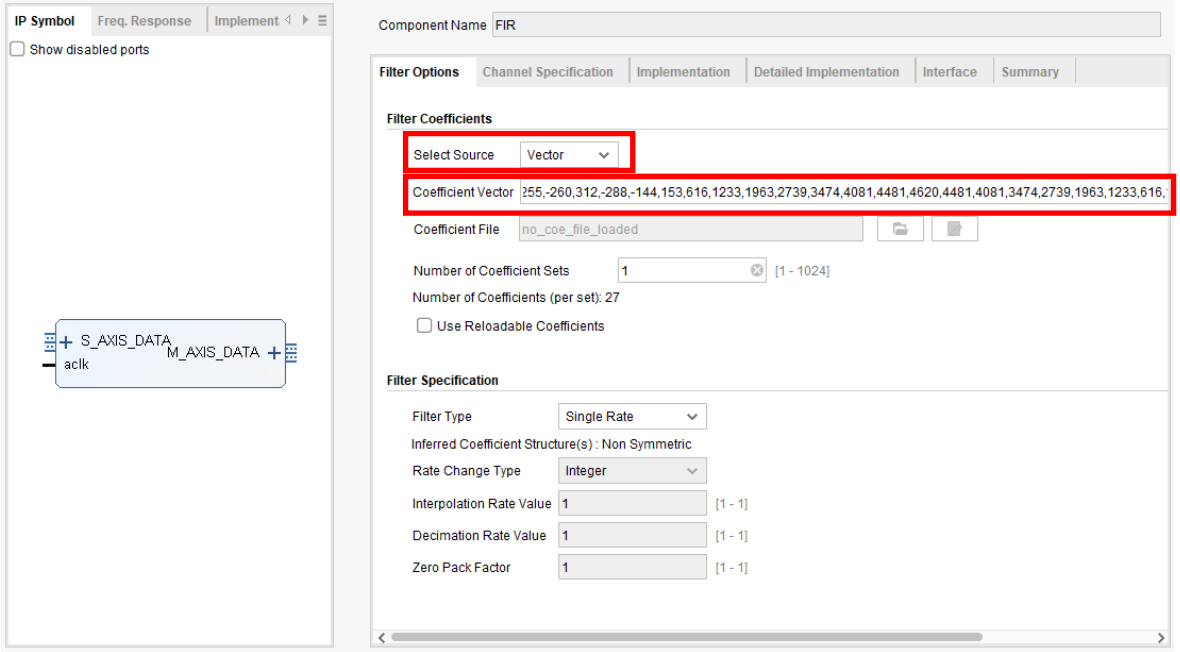
Exercice 3 : Implémentation d'un filtre FIR générique disponible sous Vivado

Définir une architecture sous Vivado qui associe une instance d'un processeur Zynq avec un bloc AXI DMA comme lors de l'exercice 1

Ajouter un bloc FIR configuré au niveau filter Options =>

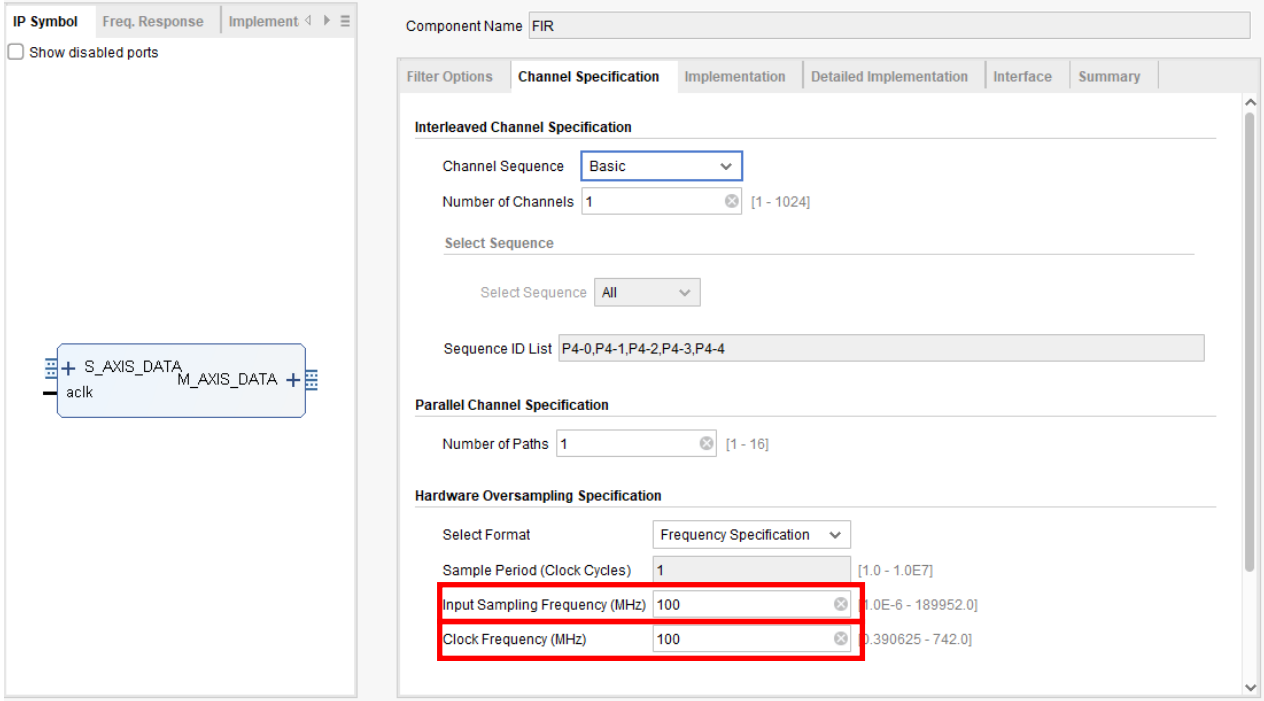
Les valeurs des 27 coefficients du filtre sont :

-255,-260,312,-288,-144,153,616,1233,1963,2739,3474,4081,4481,4620,4481,4081,3474,2739,1963,1233,616,153,-144,-288,-312,-260,-255



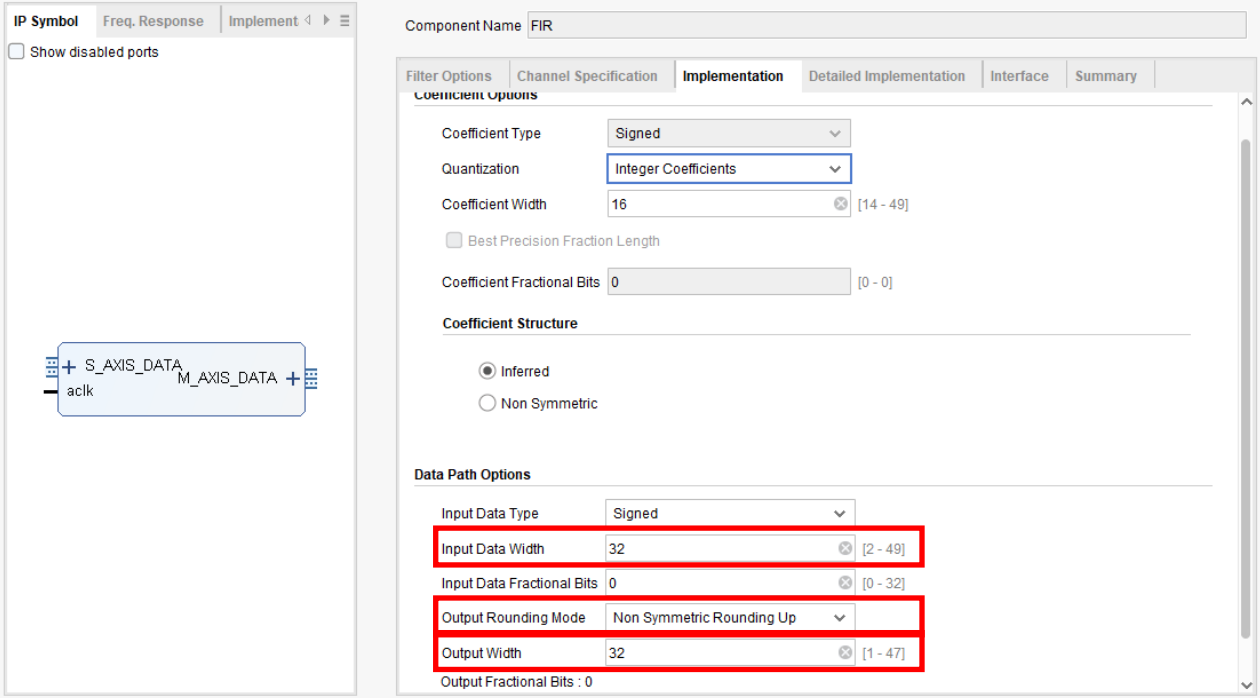
Exercice 3 : Implémentation d'un filtre FIR générique disponible sous Vivado

Configuration de la page Channel Specifications



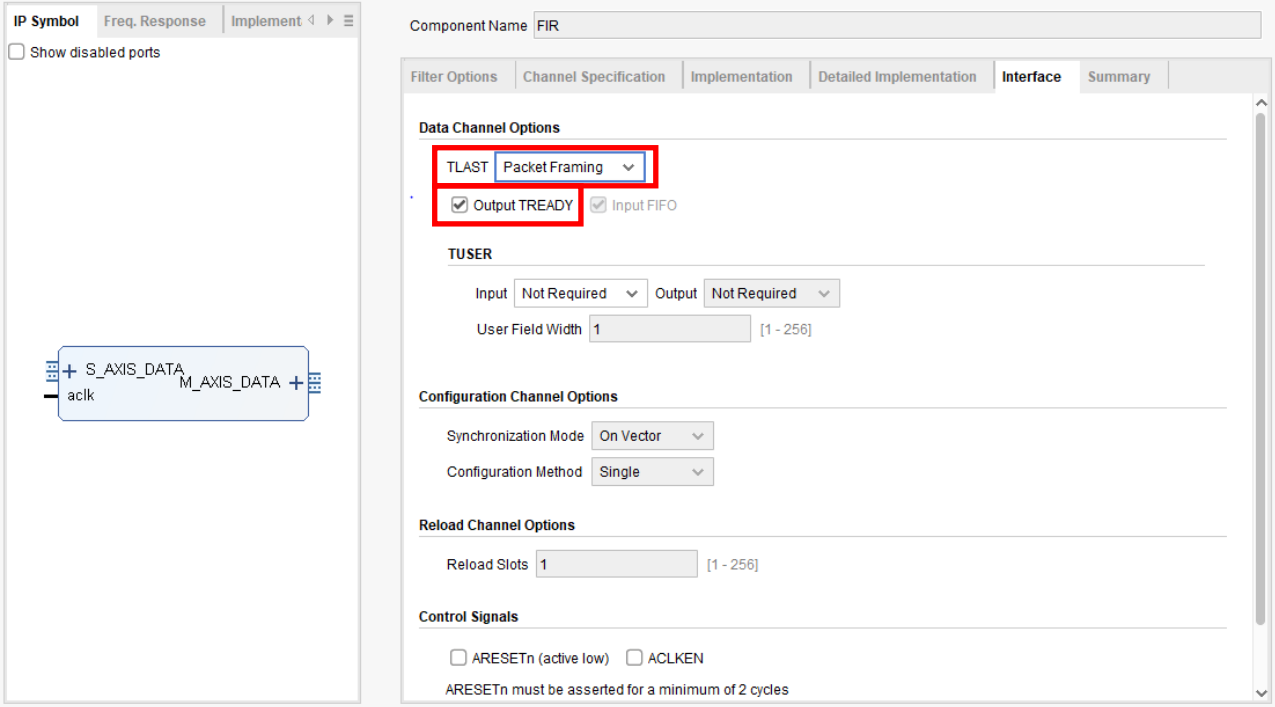
Exercice 3 : Implémentation d'un filtre FIR générique disponible sous Vivado

Configuration de la page Implementation

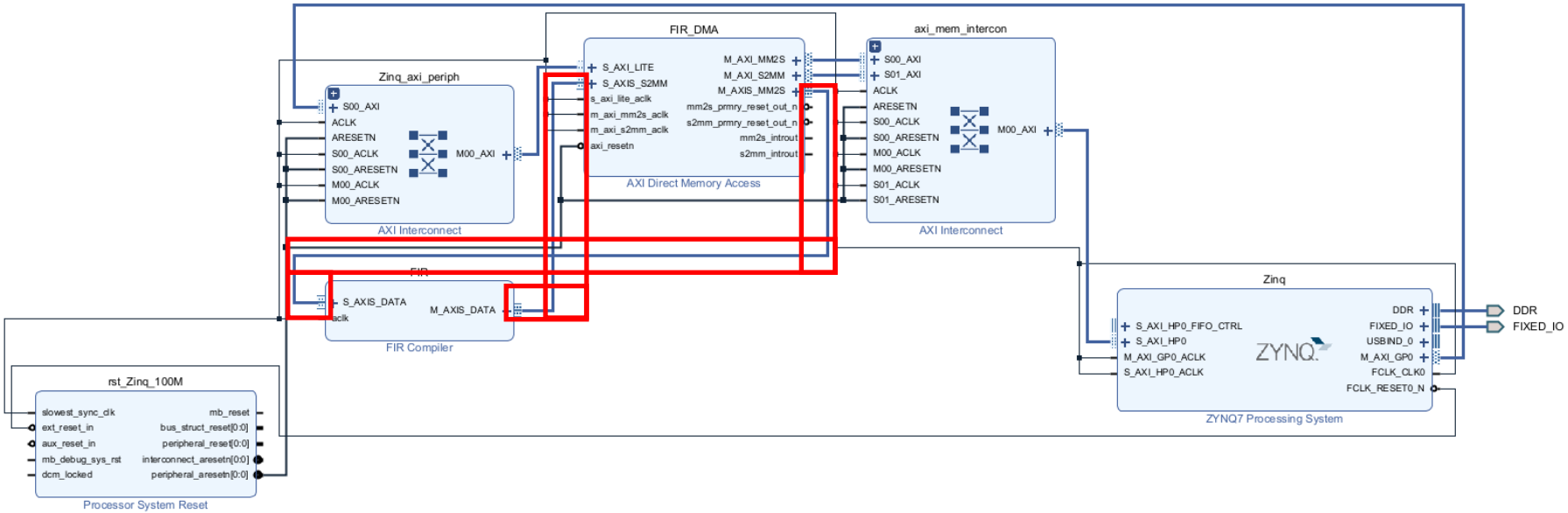


Exercice 3 : Implémentation d'un filtre FIR générique disponible sous Vivado

Configuration de la page Interface



Exercice 3 : Implémentation d'un filtre FIR générique disponible sous Vivado



Exercice 3 : Implémentation d'un filtre FIR générique disponible sous Vivado

L'étape suivante consiste à configurer l'accès au bloc FIR générique de votre projet Vivado

Cela implique de disposer de l'overlay

```
from pynq import Overlay
overlay =Overlay("/home/xilinx/jupyter_notebooks/SE_EN325/S7_Ex3/design_name.bit")
```

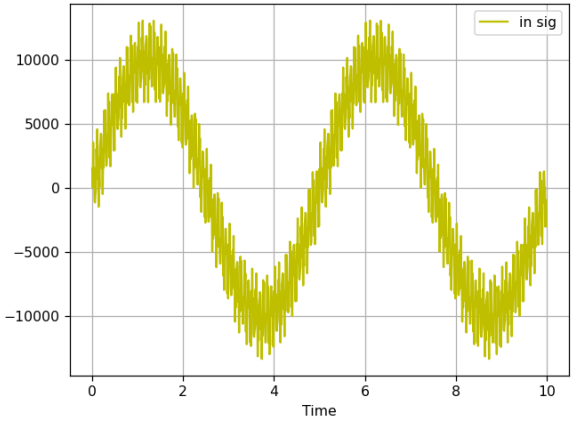
Puis, il faut configurer le module DMA comme suit :

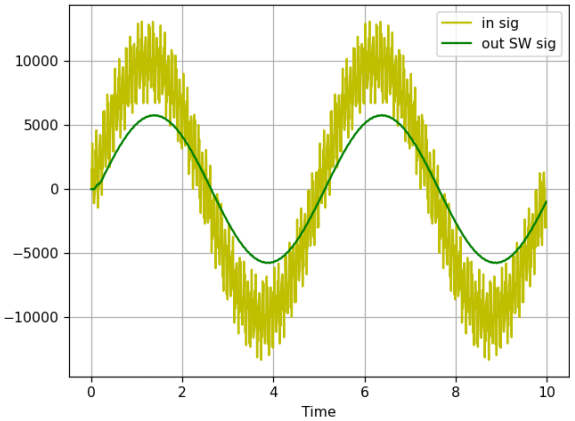
```
DMA = overlay.DMA_name
DMA?
```

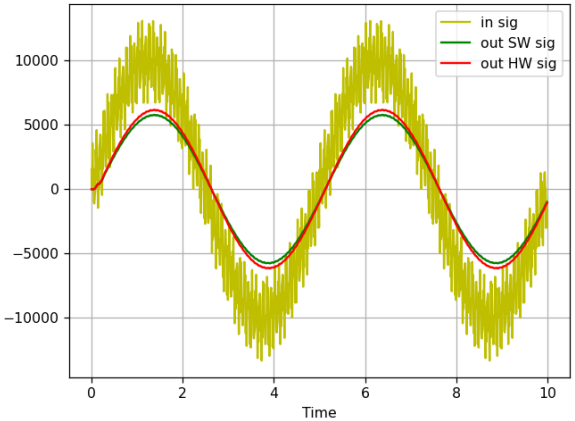
Exercice 3 : Implémentation d'un filtre FIR générique disponible sous Vivado

Objectif 1 : Exécuter successivement les différents programmes proposés dans le fichier fir_filter.ipynb afin de vous familiariser avec la syntaxe python et les fonctions du fichier.

Objectif 2 : Modifier le programme Python sous Jupyter afin de prendre en compte les configurations décrites dans le transparent précédent. Comparer le temps d'exécution des implémentations logicielles et matérielles. Vous devez normalement pouvoir visualiser les figures suivantes:



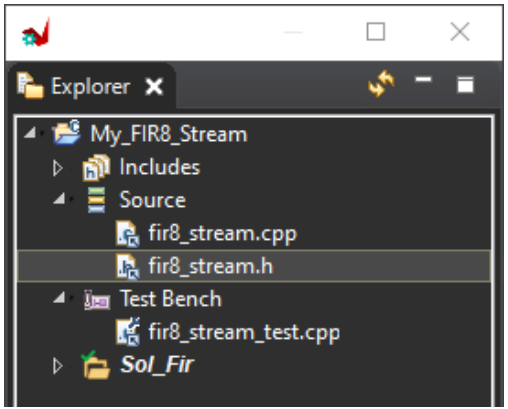




Exercice 4 : Implémentation d'un filtre FIR8 avec interface AXI4-Stream synthétisé sous Vitis HLS

Cet exercice implique de reprendre les descriptions du filtre FIR8 de l'exercice 1 de la séance 4 : (fir8.cpp, fir8.h et fir8_test.cpp)

Il faut les modifier pour synthétiser une architecture comprenant : deux AXI4-Stream complètes x et y (cf exercice 1) et une interface AXI-lite pour le contrôle



fir8_stream.h

```
#include "ap_axi_sdata.h"
#include "hls_stream.h"

#ifndef FIR_H_
#define FIR_H_

const int N=8;

typedef hls::stream< ap_axis<32,2,5,6>> data_t;
typedef ap_axis<32,2,5,6> var_t;

void FIR8_Stream(data_t *y, data_t x);

#endif
```

Exercice 4 : Implémentation d'un filtre FIR8 avec interface AXI4-Stream synthétisé sous Vitis HLS

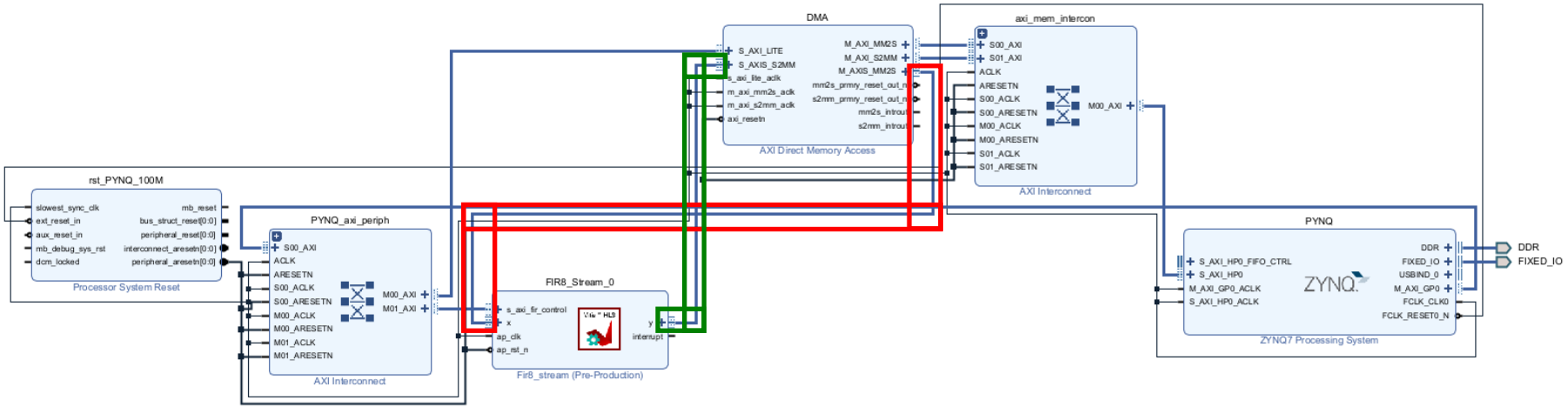
Exécuter l'environnement Vitis HLS :

- Créer un nouveau projet :
  - Nom du projet : My_FIR8_Stream_IP
  - Top Function : FIR8_Stream_IP (identique au nom de la fonction du programme fir8_stream.cpp
  - Ajouter les fichiers : fir8_stream.cpp et fir8_stream.h
  - Ajouter le testbench fir8_stream_test.cpp
  - Choisir la carte Pynq Z2 comme cible technologique
- Exécuter la simulation C pour vérifier le bon fonctionnement des interfaces
- Exécuter la synthèse C (période : à déterminer ns, Incertitude : 1 ns)
  - Regarder le rapport de synthèse (nombre de cycles, ressources utilisées)
  - Regarder le diagramme d'ordonnancement des tâches
- Exécuter la co-simulation C pour vérifier le bon fonctionnement des interfaces
- Exécuter l'exportation au niveau RTL dans l'item implémentation

Exercice 4 : Implémentation d'un filtre FIR8 avec interface AXI4-Stream synthétisé sous Vitis HLS

Définir une architecture sous Vivado qui associe une instance d'un processeur Zynq, un bloc AXI DMA configuré comme pour les exercices précédent.

Ajouter une instance de l'IP filtre FIR8 puis effectuer les connexions entre le filtre FIR8 et le bloc DMA comme indiqué sur la figure ci-dessous. Enfin, générer le bistream.



Exercice 4 : Implémentation d'un filtre FIR8 avec interface AXI4-Stream synthétisé sous Vitis HLS

L'étape suivante consiste à configurer l'accès au bloc FIR8 généré à l'aide de Vitis HLS

Cela implique de disposer de l'overlay

```
from pynq import Overlay
overlay =Overlay("/home/xilinx/jupyter_notebooks/SE_EN325/S7_Ex4/design_name.bit")
```

Il faut ensuite configurer la plage d'adresses du bloc My_FIR8

```
FIR28_Stream_IP = overlay. design_name
BASE_ADDRESS = overlay.ip_dict['design_name']['phys_addr']
ADDRESS_LENGTH = 0x64000
FIR28_Stream_IP.write (0x0,0b10000001)
FIR28_Stream_IP.register_map
```

Enfin, il faut configurer le module DMA comme suit :

```
DMA = overlay.DMA_name
DMA_send = DMA.sendchannel
DMA_recv  = DMA.recvchannel
DMA?
```

Exercice 4 : Implémentation d'un filtre FIR8 avec interface AXI4-Stream synthétisé sous Vitis HLS

Objectif 1 : Exécuter successivement les différents programmes du fichier My_FIR8.ipynb pour la configuration de l'overlay, la définition de la plage d'adresses, la configuration du module DMA et l'exécution du bloc FIR8_Stream_IP

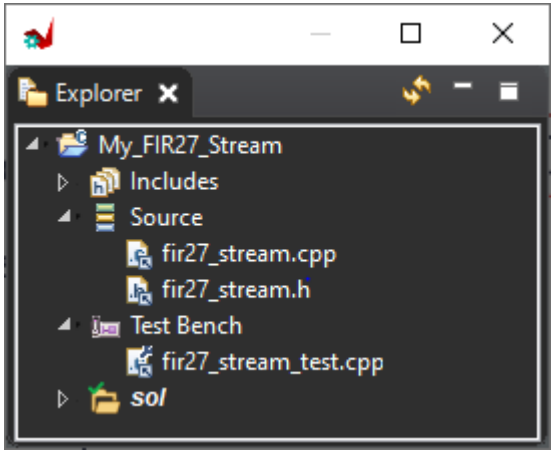
Objectif 2 : Ecrire un programme Python sous Jupyter. Ce dernier doit permettre l'exécution suivante sur la carte :

- ✓ Allouer 2 buffers de taille 32: (input_buffer et output_buffer);
- ✓ Fixer les valeurs 0 à 31 à input_buffer;
- ✓ Effectuer l'envoi puis la réception de la trame de taille 32;
- ✓ Afficher le contenu de output_buffer;
- ✓ Comparer le contenu avec le fichier output_fir8_stream_gold.dat
- ✓ Libérer l'espace mémoire allouée aux deux buffers.

Exercice 5 : Remplacement du filtre générique par un filtre FIR27 avec interface AXI4-Stream

Cet exercice implique de reprendre les descriptions du filtre FIR8 de l'exercice 4 : (fir8_stream.cpp, fir8_stream.h et fir8_stream_test.cpp)

Il faut modifier la taille du filtre dans le .h et les valeurs des 27 coefficients dans le fichier .cpp : {-255,-260,312,-288,-144,153,616,1233,1963,2739,3474,4081,4481,4620,4481,4081,3474,2739,1963,1233,616,153,-144,-288,-312,-260,-255};



fir27_stream.h

```
#include "ap_axi_sdata.h"
#include "hls_stream.h"

#ifndef FIR_H_
#define FIR_H_

const int N=27;

typedef hls::stream< ap_axis<32,2,5,6>> data_t;
typedef ap_axis<32,2,5,6> var_t;

void FIR8_Stream(data_t *y, data_t x);

#endif
```

Exercice 5 : Remplacement du filtre générique par un filtre FIR27 avec interface AXI4-Stream

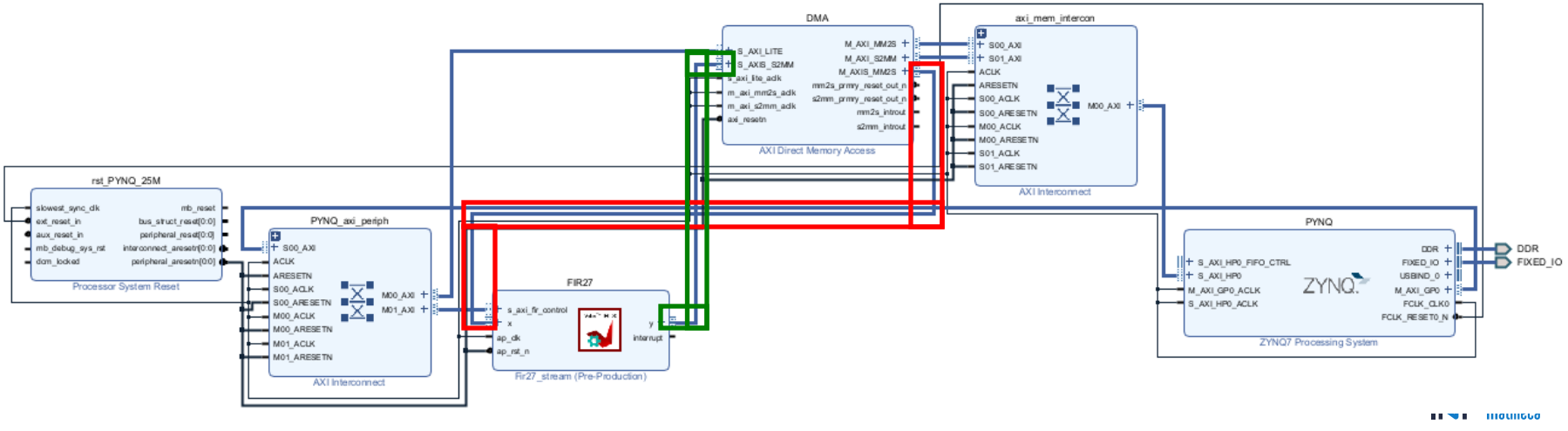
Exécuter l'environnement Vitis HLS :

- Créer un nouveau projet :
  - Nom du projet : My_FIR27_Stream_IP
  - Top Function : FIR27_Stream_IP (identique au nom de la fonction du programme fir27_stream.cpp)
  - Ajouter les fichiers : fir27_stream.cpp et fir27_stream.h
  - Ajouter le testbench fir27_stream_test.cpp
  - Choisir la carte Pynq Z2 comme cible technologique
- Exécuter la simulation C pour vérifier le bon fonctionnement des interfaces
- Exécuter la synthèse C (période : à déterminer ns, Incertitude : 1 ns)
  - Regarder le rapport de synthèse (nombre de cycles, ressources utilisées)
  - Regarder le diagramme d'ordonnancement des tâches
- Exécuter la co-simulation C pour vérifier le bon fonctionnement des interfaces
- Exécuter l'exportation au niveau RTL dans l'item implémentation

Exercice 5 : Remplacement du filtre générique par un filtre FIR27 avec interface AXI4-Stream

Définir une architecture sous Vivado qui associe une instance d'un processeur Zynq, un bloc AXI DMA configuré comme pour les exercices précédent.

Ajouter une instance de l'IP filtre FIR27 puis effectuer les connexions entre le filtre FIR27 et le bloc DMA comme indiqué sur la figure ci-dessous. Enfin, générer le bistream.



Exercice 5 : Remplacement du filtre générique par un filtre FIR27 avec interface AXI4-Stream

L'étape suivante consiste à configurer l'accès au bloc FIR27 généré à l'aide de Vitis HLS

Cela implique de disposer de l'overlay

```
from pynq import Overlay
overlay =Overlay("/home/xilinx/jupyter_notebooks/SE_EN325/S7_Ex5/design_name.bit")
```

Il faut ensuite configurer la plage d'adresses du bloc My_FIR27

```
FIR27_Stream_IP = overlay. design_name
BASE_ADDRESS = overlay.ip_dict['design_name']['phys_addr']
ADDRESS_LENGTH = 0x64000
FIR27_Stream_IP.write (0x0,0b10000001)
FIR27_Stream_IP.register_map
```

Enfin, il faut configurer le module DMA comme suit :

```
DMA = overlay.DMA_name
DMA?
```

Exercice 5 : Remplacement du filtre générique par un filtre FIR27 avec interface AXI4-Stream

Objectif 1 : Exécuter successivement les différents programmes proposés dans le fichier My_fir_filter_HLS.ipynb afin de vous familiariser avec la syntaxe python et les fonctions du fichier.

Objectif 2 : Modifier le programme Python sous Jupyter afin de prendre en compte les configurations décrites dans le transparent précédent. Comparer le temps d'exécution des implémentations logicielles et matérielles. Vous devez normalement pouvoir visualiser les figures suivantes: