

# Séance 6 :

## Introduction aux flux de données (Axi-Stream <-> DMA)

Christophe Jégo  
année 2025-2026

# 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 =>

The image shows the configuration interface for the `axi_dma_0` component in Vivado Vitis. The component name is `axi_dma_0`. The configuration is as follows:

- ☐ Show disabled ports
- ☐ Enable Asynchronous Clocks (Auto)
- ☐ Enable Scatter Gather Engine
- ☐ Enable Micro DMA
- ☐ Enable Multi Channel Support
- ☐ Enable Control / Status Stream
- Width of Buffer Length Register (8-26): 26 bits
- Address Width (32-64): 32 bits
- ☒ Enable Read Channel
  - Number of Channels: 1
  - Memory Map Data Width: 32
  - Stream Data Width: 32
  - Max Burst Size: 16
  - ☐ Allow Unaligned Transfers
- ☒ Enable Write Channel
  - Number of Channels: 1
  - Memory Map Data Width: 32
  - Stream Data Width: 32
  - Max Burst Size: 16
  - ☐ Allow Unaligned Transfers
  - ☐ Use Rlength In Status Stream
- ☐ Enable Single AXI4 Data Interface

On the left, a block diagram shows the connections for the `axi_dma_0` component. The component has two main data paths: `M_AXI_MM2S` and `M_AXI_S2MM`. The `M_AXI_MM2S` path is connected to `S_AXI_LITE` and `S_AXIS_S2MM`. The `M_AXI_S2MM` path is connected to `s_axi_lite_aclk`, `m_axi_mm2s_aclk`, `m_axi_s2mm_aclk`, and `axi_resetn`. The `axi_dma_0` component is also connected to `mm2s_primry_reset_out_n`, `s2mm_primry_reset_out_n`, `mm2s_introut`, and `s2mm_introut`.

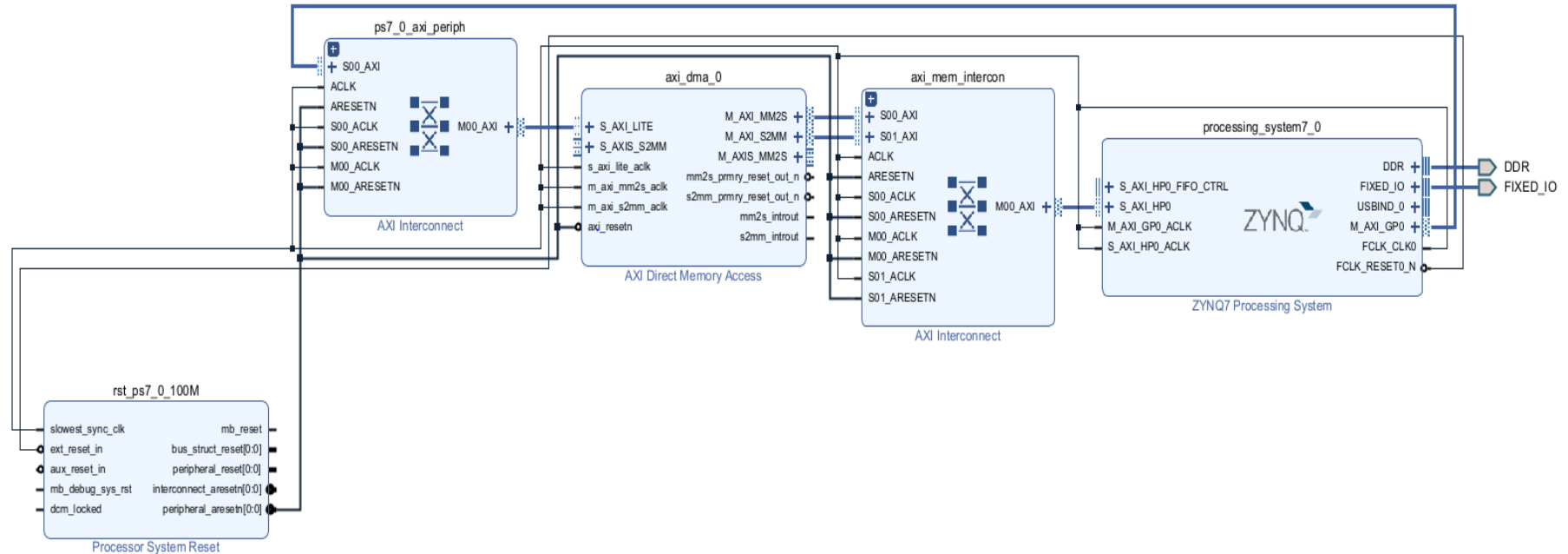
# Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

Le processeur Zynq doit être configuré comme suit :

The screenshot displays the Vitis PS-PL Configuration window. On the left is a 'Page Navigator' with a tree view containing: Zynq Block Design, PS-PL Configuration (selected), Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, and Interrupts. The main area is titled 'PS-PL Configuration' and includes a 'Summary Report' link. Below the title are navigation icons and a search bar. A table lists various configuration options:

Name	Select	Description
> General		
> AXI Non Secure Enablement	0	Enable AXI Non Secure Transaction
> GP Slave AXI Interface		
v HP Slave AXI Interface		
> S AXI HP0 interface	<input checked="" type="checkbox"/>	Enables AXI high performance slave interface 0
> S AXI HP1 interface	<input type="checkbox"/>	Enables AXI high performance slave interface 1
> S AXI HP2 interface	<input type="checkbox"/>	Enables AXI high performance slave interface 2
> S AXI HP3 interface	<input type="checkbox"/>	Enables AXI high performance slave interface 3
> ACP Slave AXI Interface		
> DMA Controller		
> PS-PL Cross Trigger interface	<input type="checkbox"/>	Enables PL cross trigger signals to PS and vice-versa

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



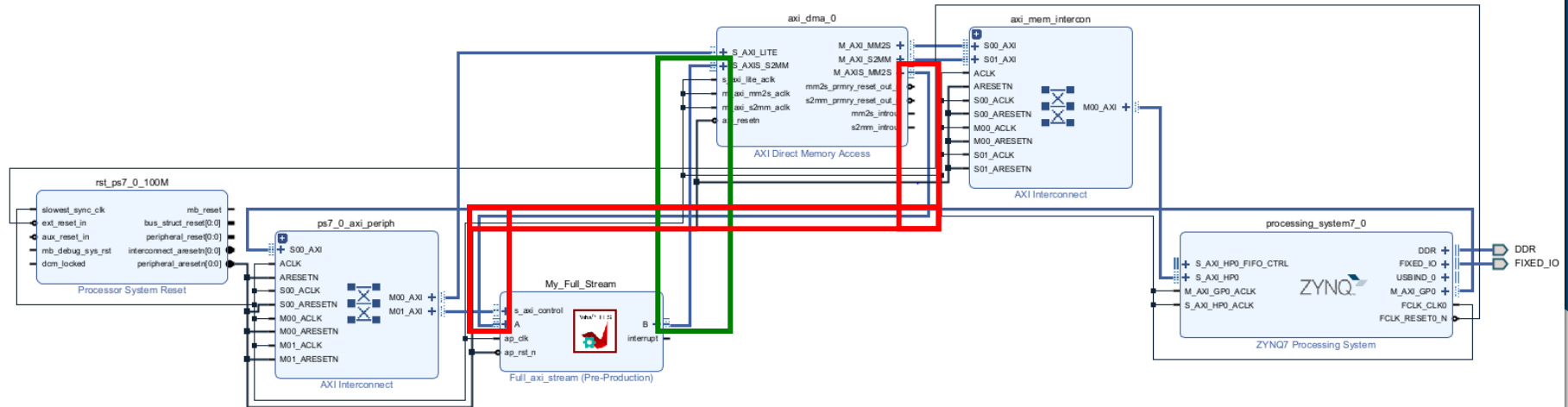


# Exercice 1 : Mise en place d'interfaces AXI4-Stream with side-channels avec Vitis HLS

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 pyng 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 vhd :

***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 :

IP Symbol   Freq. Response   Implement ▾

☐ Show disabled ports

Component Name: FIR

Filter Options   Channel Specification   Implementation   Detailed Implementation   Interface   Summary

Filter Coefficients

Select Source: Vector

Coefficient Vector: -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

Coefficient File: no\_coe\_file\_loaded

Number of Coefficient Sets: 1 [1 - 1024]

Number of Coefficients (per set): 27

☐ Use Reloadable Coefficients

Filter Specification

Filter Type: Single Rate

Inferred Coefficient Structure(s): Non Symmetric

Rate Change Type: Integer

Interpolation Rate Value: 1 [1 - 1]

Decimation Rate Value: 1 [1 - 1]

Zero Pack Factor: 1 [1 - 1]

-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

The screenshot displays the Vivado FIR Filter Wizard configuration for the 'Channel Specifications' page. The component name is 'FIR'. The 'Interleaved Channel Specification' section shows 'Channel Sequence' set to 'Basic' and 'Number of Channels' set to '1'. The 'Parallel Channel Specification' section shows 'Number of Paths' set to '1'. The 'Hardware Oversampling Specification' section shows 'Select Format' set to 'Frequency Specification', 'Sample Period (Clock Cycles)' set to '1', 'Input Sampling Frequency (MHz)' set to '100', and 'Clock Frequency (MHz)' set to '100'. The last two fields are highlighted with a red box.

Section	Parameter	Value	Range
Interleaved Channel Specification	Channel Sequence	Basic	
	Number of Channels	1	[1 - 1024]
Parallel Channel Specification	Number of Paths	1	[1 - 16]
	Sample Period (Clock Cycles)	1	[1.0 - 1.0E7]
Hardware Oversampling Specification	Input Sampling Frequency (MHz)	100	[1.0E-6 - 189952.0]
	Clock Frequency (MHz)	100	[0.390625 - 742.0]

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

Configuration de la page  
Implementation

The screenshot displays the Vivado IP configuration interface for a FIR filter. The left pane shows the IP Symbol, Freq. Response, and Implement tabs, with a checkbox for 'Show disabled ports'. The right pane shows the Component Name as 'FIR' and the Implementation tab selected. The Configuration Options section includes Coefficient Type (Signed), Quantization (Integer Coefficients), Coefficient Width (16), and Coefficient Fractional Bits (0). The Coefficient Structure section has radio buttons for Inferred (selected) and Non Symmetric. The Data Path Options section includes Input Data Type (Signed), Input Data Width (32), Input Data Fractional Bits (0), Output Rounding Mode (Non Symmetric Rounding Up), and Output Width (32). The Output Fractional Bits are set to 0. Red boxes highlight the Input Data Width, Output Rounding Mode, and Output Width fields.

IP Symbol Freq. Response Implement

☐ Show disabled ports

Component Name FIR

Filter Options Channel Specification Implementation Detailed Implementation Interface Summary

**Coefficient Options**

Coefficient Type Signed

Quantization Integer Coefficients

Coefficient Width 16 [14 - 49]

☐ Best Precision Fraction Length

Coefficient Fractional Bits 0 [0 - 0]

**Coefficient Structure**

☒ Inferred

☐ Non Symmetric

**Data Path Options**

Input Data Type Signed

Input Data Width 32 [2 - 49]

Input Data Fractional Bits 0 [0 - 32]

Output Rounding Mode Non Symmetric Rounding Up

Output Width 32 [1 - 47]

Output Fractional Bits : 0

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

Configuration de la  
page Interface

The screenshot displays the Vivado FIR Filter Wizard interface. On the left, the 'IP Symbol' tab shows a block diagram with ports: S\_AXIS\_DATA (input), M\_AXIS\_DATA (output), and aclk (clock). The main panel is the 'Interface' tab, which is configured for a component named 'FIR'. The 'Data Channel Options' section is highlighted with a red box, showing 'TLAST' selected in the 'Packet Framing' dropdown and 'Output TREADY' checked. The 'TUSER' section shows 'Input' and 'Output' both set to 'Not Required' and 'User Field Width' set to 1. The 'Configuration Channel Options' section shows 'Synchronization Mode' set to 'On Vector' and 'Configuration Method' set to 'Single'. The 'Reload Channel Options' section shows 'Reload Slots' set to 1. The 'Control Signals' section shows 'ARESETn (active low)' and 'ACLKEN' both unchecked, with a note that 'ARESETn must be asserted for a minimum of 2 cycles'.

IP Symbol   Freq. Response   Implement ▾

☐ Show disabled ports

Component Name: FIR

Filter Options   Channel Specification   Implementation   Detailed Implementation   **Interface**   Summary

**Data Channel Options**

TLAST   Packet Framing ▾

☒ Output TREADY   ☒ Input FIFO

**TUSER**

Input: Not Required ▾   Output: Not Required ▾

User Field Width: 1 [1 - 256]

**Configuration Channel Options**

Synchronization Mode: On Vector ▾

Configuration Method: Single ▾

**Reload Channel Options**

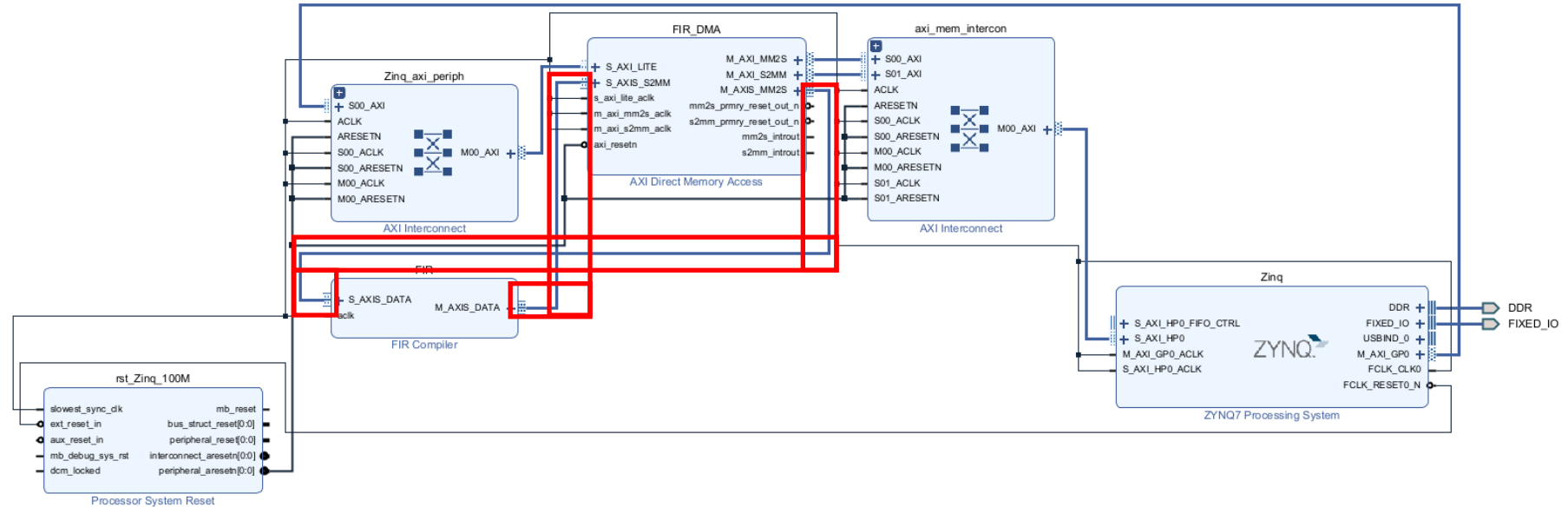
Reload Slots: 1 [1 - 256]

**Control Signals**

☐ ARESETn (active low)   ☐ ACLKEN

ARESETn must be asserted for a minimum of 2 cycles

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

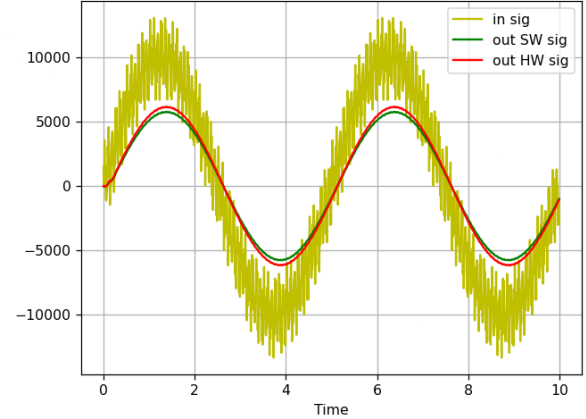
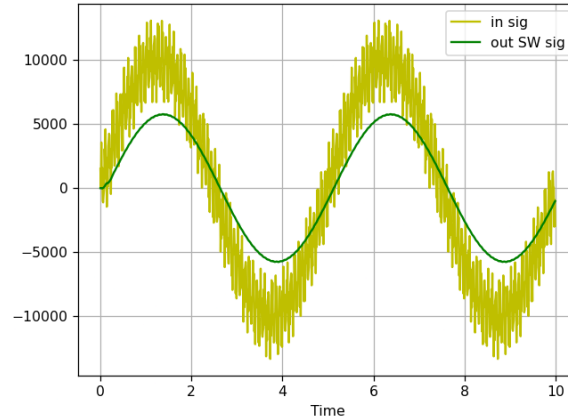
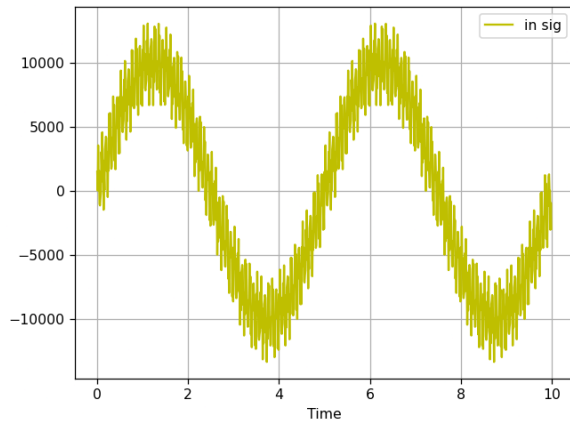
Puis, il faut configurer le module DMA comme suit :

```
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.

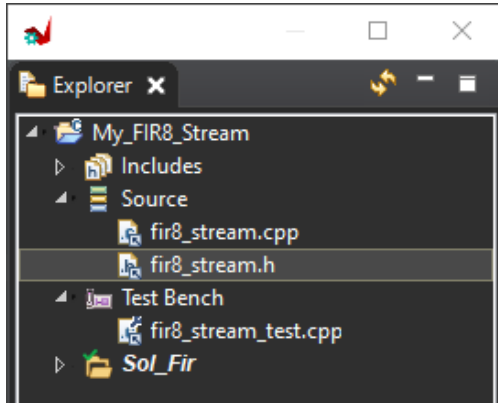
Objectif 2 : Modifier le programme Python sous Jupyter afin de prendre en compte les configurations décrites dans le transparent précédent. Comparer le temps d'exécution des implémentations logicielles et matérielles. Vous devez normalement pouvoir visualiser les figures suivantes:



# 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

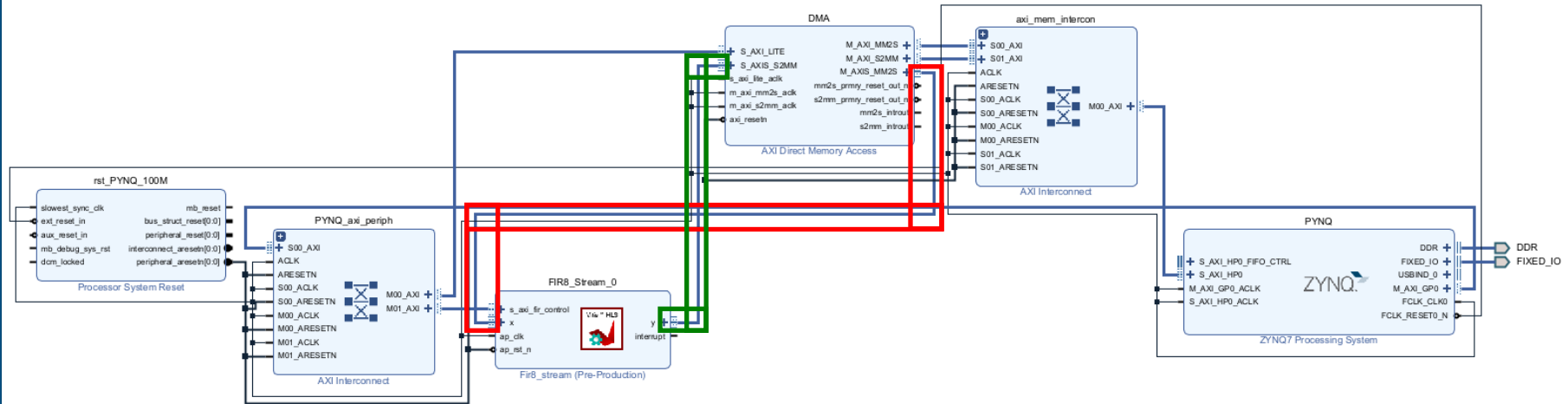
## ↓ Exécuter la co-simulation C pour vérifier le bon fonctionnement des interfaces

## ↓ 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 pyng 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 :

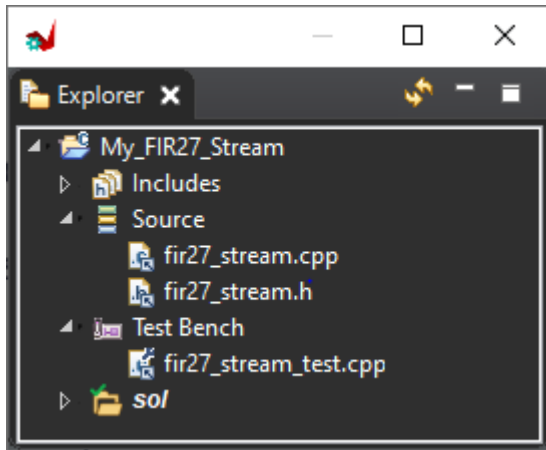
- ✓ Allouer 2 buffers de taille 32: (input\_buffer et output\_buffer);
- ✓ Fixer les valeurs 0 à 31 à input\_buffer;
- ✓ Effectuer l'envoi puis la réception de la trame de taille 32;
- ✓ Afficher le contenu de output\_buffer;
- ✓ Comparer le contenu avec le fichier output\_fir8\_stream\_gold.dat
- ✓ 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

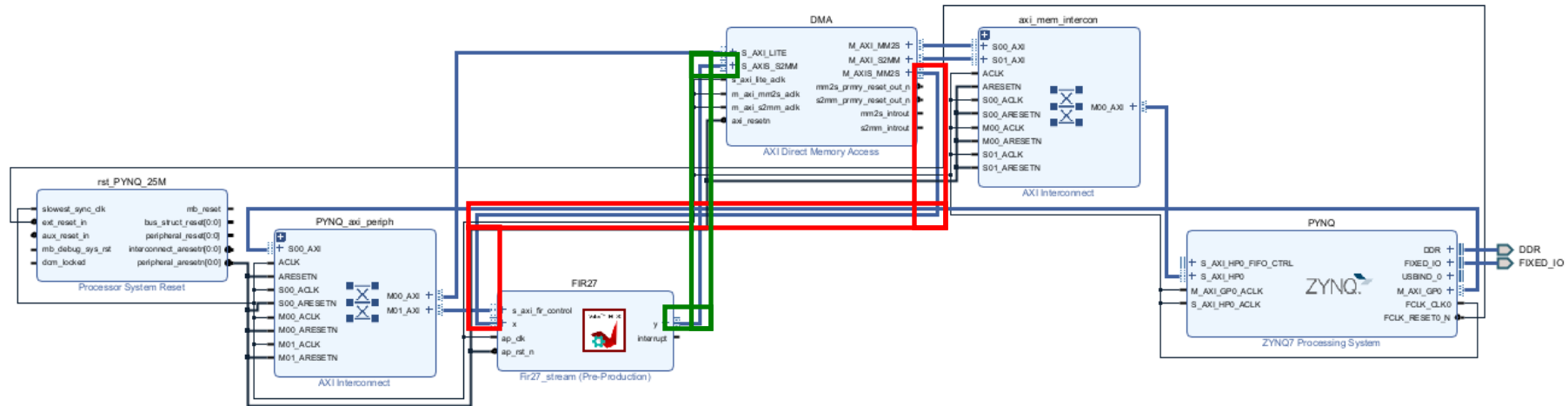
## ↓ Exécuter la co-simulation C pour vérifier le bon fonctionnement des interfaces

## ↓ 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:

