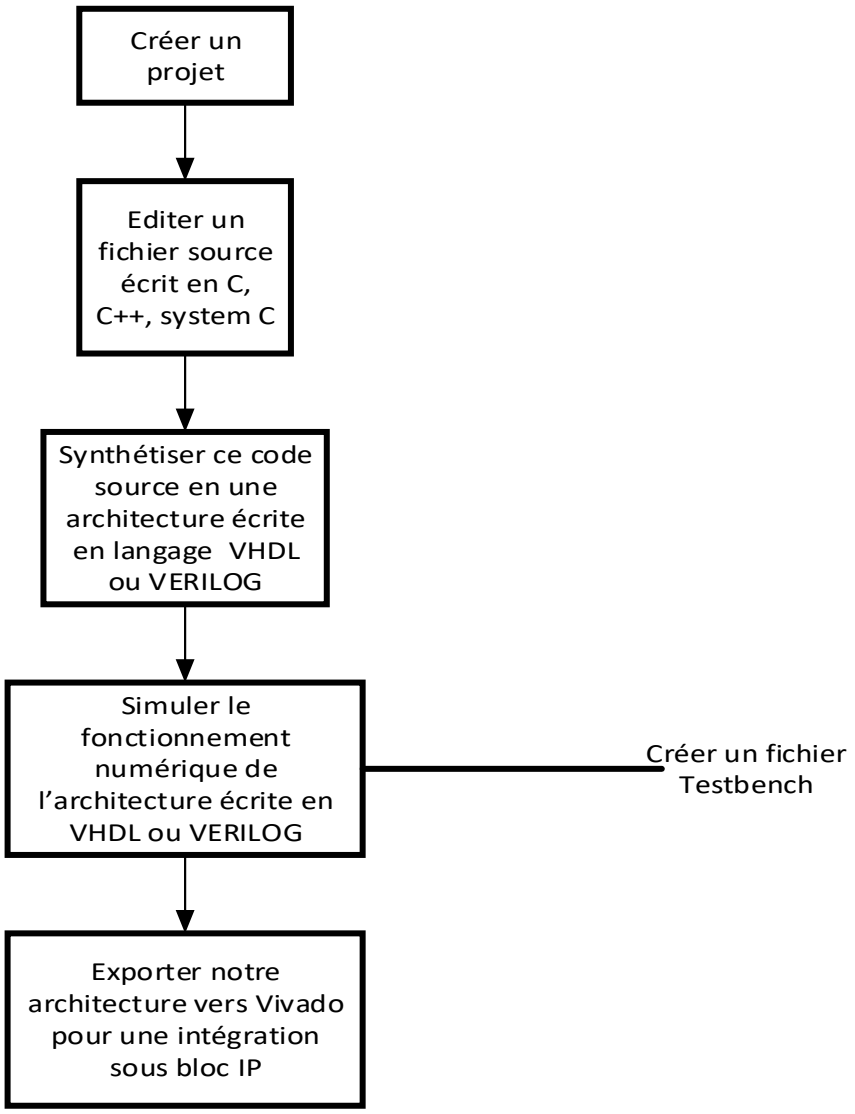
L'outil de synthèse de haut niveau (HLS High Level Synthesis)

1) Introduction

L'outil de synthèse de haut niveau (HLS High Level Synthesis) transforme une spécification système en C, C++, systemC ou openCL en une implémentation (RTL Register Transfer Level) que tu synthétises dans un FPGA.
Vivado HLS supporte les interfaces AXI que tu peux implémenter en utilisant une description C, C++, SystemC, ou OpenCL. Tu peux réaliser une conception sous Vivado HLS avec des interfaces AXI soit à travers IP integrator ou instancier directement le système en RTL.

2) La chaine opérationnelle sous vivado HLS

Les différentes opérations que l'on peut mener sous VIVADO sont résumées par le graphique ci-dessous :



L'écriture d'un programme en C, C++ ou system C doit répondre aux exigences de la synthèse. Durant cette première partie nous allons décrire de manière théorique les éléments qui nous permettrons par la suite de coder un programme C ou C++ en toute connaissance de cause.

3) Les éléments de base sous HLS

3.1) Les interfaces

C'est l'élément essentiel qui va nous permettre par la suite de créer des interfaces d'entrées, sorties, entrées sorties sur notre bloc IP.

On peut être amené à créer deux types d'interfaces :

- Les interfaces de synthèse qui sont réalisées en utilisant les arguments d'une fonction en langage C ou C++. Les interfaces de synthèse permettent de communiquer notre IPBLOC à d'autres IPBLOCS ou vers le processeur
- Les interfaces de block Level qui sont liées aux noms de la fonction de haut niveau en langage C. Ce type d'interface permet de contrôler l'exécution d'un sous-système de contrôler et de coordonner les échanges d'information entre chaque sous -système

3.2) Les interfaces de synthèse

Elles sont donc créées grâce aux arguments des fonctions. Suivant les opérations effectuées par chaque argument ou paramètres d'entrées de la fonction, automatiquement le synthétiseur détermine la direction de l'interface ou du port.

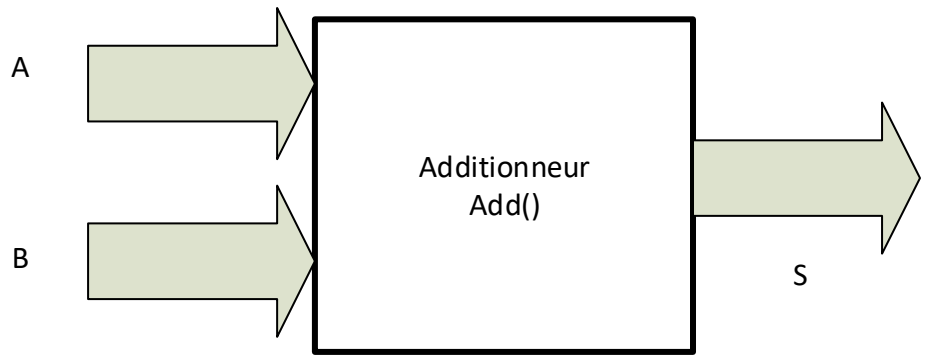
Exemple :

```
Void Add ( int A , int B, Int *S)

{
*S= A+B;

}
```

Après synthèse les arguments A, B, S deviennent des interfaces de synthèse.



Chaque port ou interface obéit à un protocole de communication que l'on doit choisir sur HLS

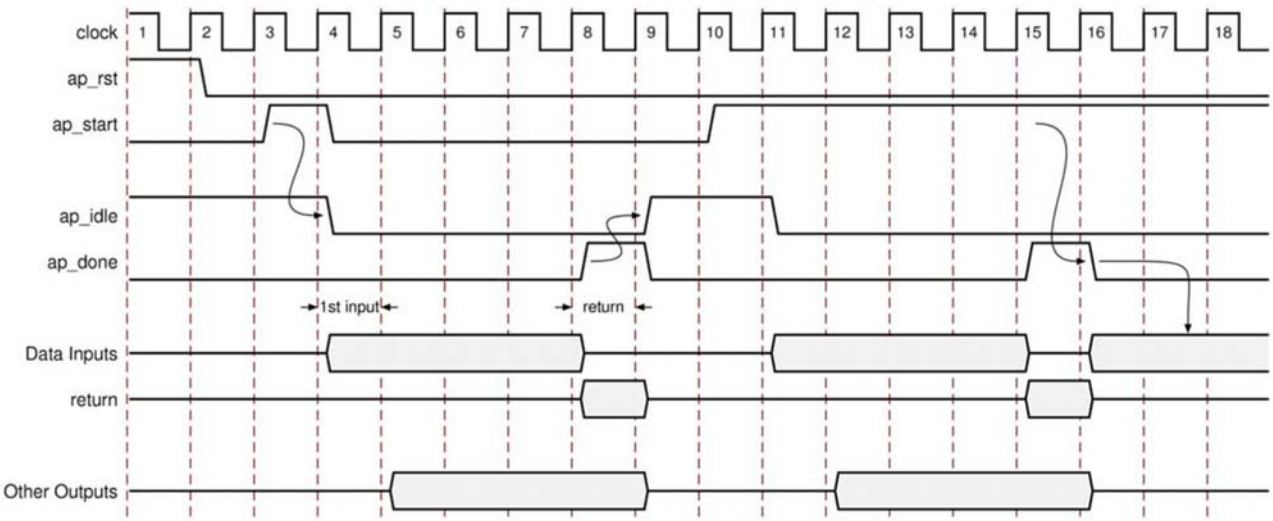
- Un passage de paramètre par valeur ne peut être qu'une entrée
- Un passage par référence peut devenir une entrée ou une sortie
- Une valeur de retour devient une sortie

3.3) Les principaux protocoles d'interface sous HLS

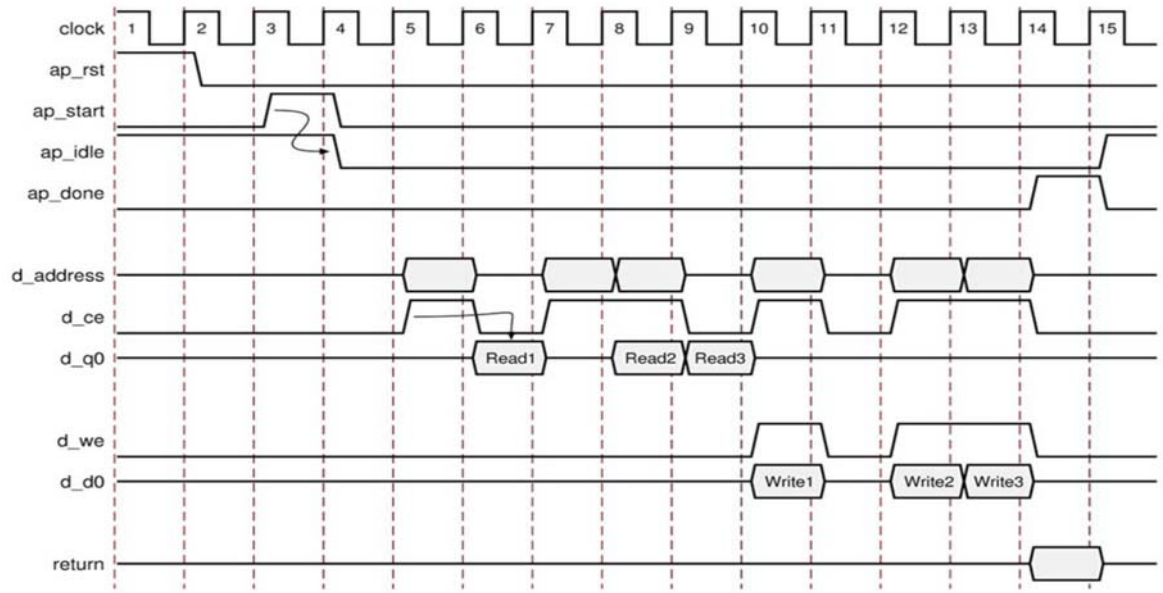
Ap_none C'est l'un des protocoles les plus simples avec aucun protocole d'interface et aucuns signaux de contrôle additionnels

Ap-ack Ce protocole se comporte différemment suivant si l'interface est en entrées ou en sorties. Pour les entrées un port d'acquittement de sortie est additionné et maintenu à l'état haut sur le même cycle d'horloge que l'entrée est lue. Pour un port en sortie un port d'acquittement en entrée est additionné.

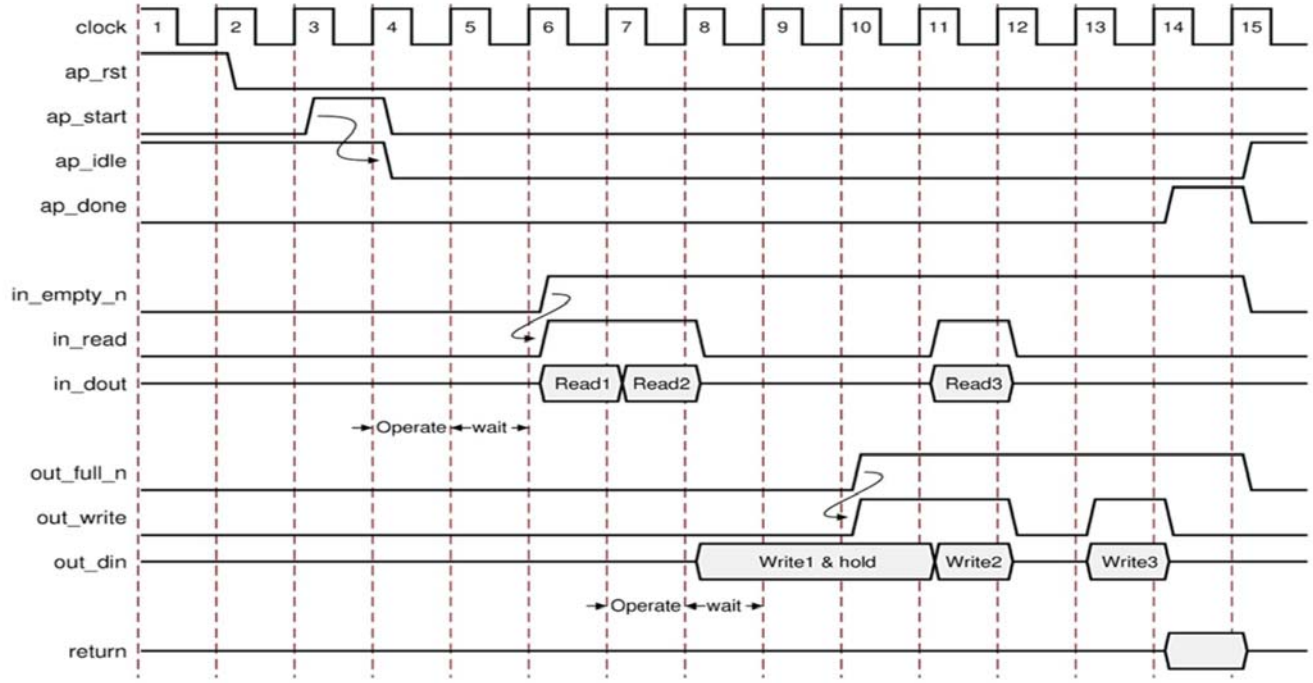
Ap_hs le suffixe HS veut dire handshaking ou poignée de main. Ce protocole permet un processus de poignée de main entre un producteur et un consommateur de données. Ils nécessitent deux ports de contrôle



Ap_memory Ce protocole basé sur une mémoire permet un accès aléatoire à une mémoire. Le seul argument compatible avec ce protocole est le type tableau. Ils nécessitent des signaux de validation d'horloge et d'adresses.



AP_FIFO Ce protocole basé sur une mémoire FIFO permet un accès séquentiel à une mémoire, ils ne nécessitent donc pas d'adresse



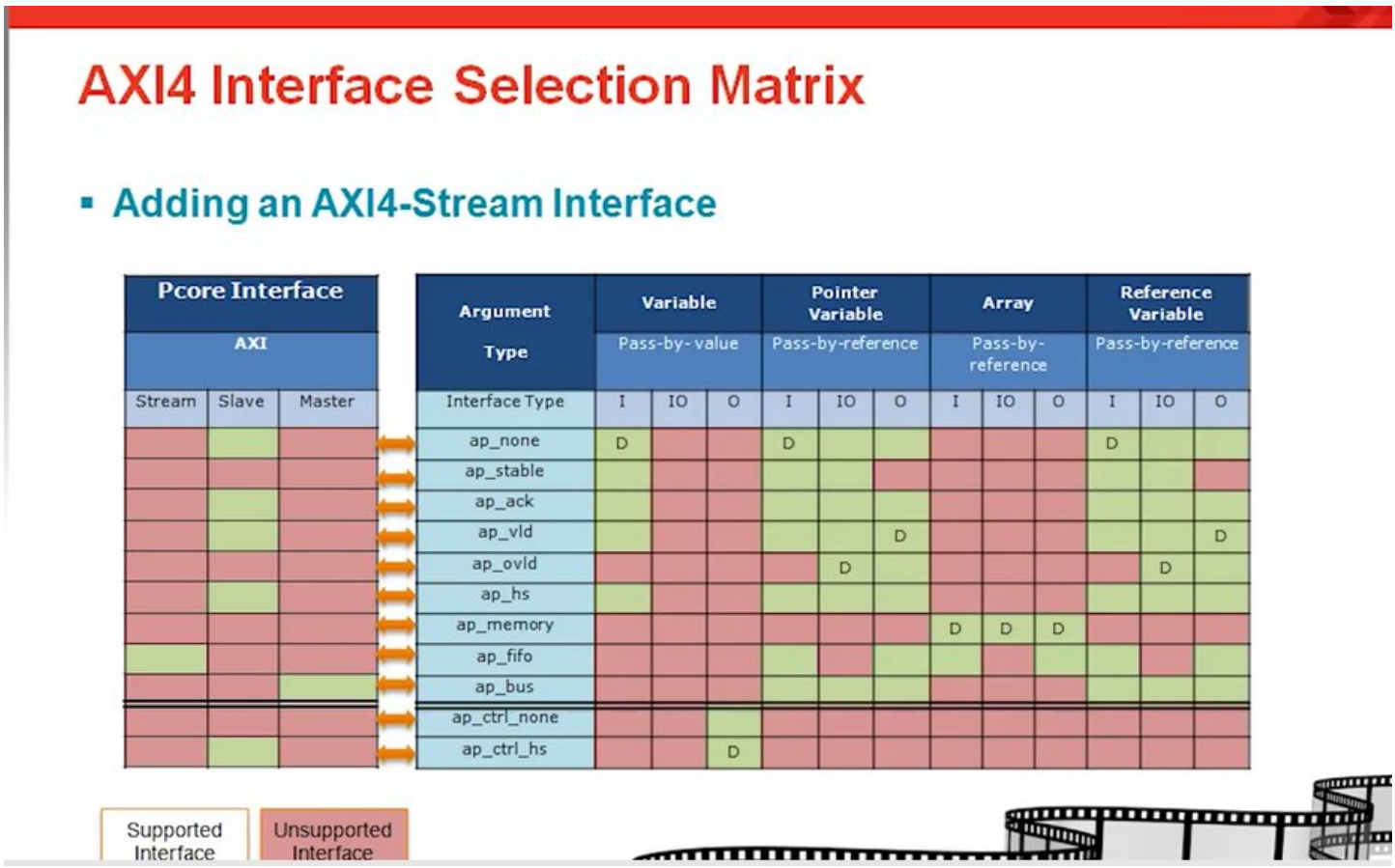
AP_BUS celui-ci est une simple interface de bus générique

Le bus **AXI** qui est du type AXI STREAM

Le bus **AXILITE**

Le bus **M_axi** qui spécifie une interface du type AXI4 maitre

Le tableau ci-dessous spécifie **les restrictions** sur le passage des paramètres pour chacun des protocoles :



Pcore Interface			Argument Type	Variable			Pointer Variable			Array			Reference Variable		
AXI				Pass-by- value			Pass-by-reference			Pass-by-reference			Pass-by-reference		
Stream	Slave	Master	Interface Type	I	IO	O	I	IO	O	I	IO	O	I	IO	O
			ap_none	D			D						D		
			ap_stable												
			ap_ack												
			ap_vld						D						D
			ap_ovld					D						D	
			ap_hs												
			ap_memory							D	D	D			
			ap_fifo												
			ap_bus												
			ap_ctrl_none												
			ap_ctrl_hs			D									

3.4) Les types de données en langage C ou C++ sur Vivado

On peut utiliser les mêmes types qu'en langage C ou C++. Si l'on apporte une certaine précision sur le format des données on peut utiliser la syntaxe suivante :

Pour les entiers

Language	Integer Data Type	Description	Required Header
C	int*N* (e.g. `int7`)	signed integer of *N* bits precision	`#include "ap_cint.h"`
	uint*N* (e.g. `uint7`)	unsigned integer of *N* bits precision	
C++	ap_int<*N*> (e.g. `ap_int<7>`)	signed integer of *N* bits precision	`#include "ap_int.h"`
	ap_uint<*N*> (e.g. `ap_uint<7>`)	unsigned integer of *N* bits precision	

```
// C code example
#include "ap_cint.h"

void top_level_function (..)
{
     // declarations
       int6 small_signed;
       uint10 big_unsigned;
       int22 vbig_signed;
       ...
}
```

```
// C++ code example
#include "ap_int.h"

void top_level_function (..)
{
     // declarations
       ap_int<6> small_signed;
       ap_uint<10> big_unsigned;
       ap_int<22> vbig_signed;
       ...
}
```

Les flottants à virgule fixe

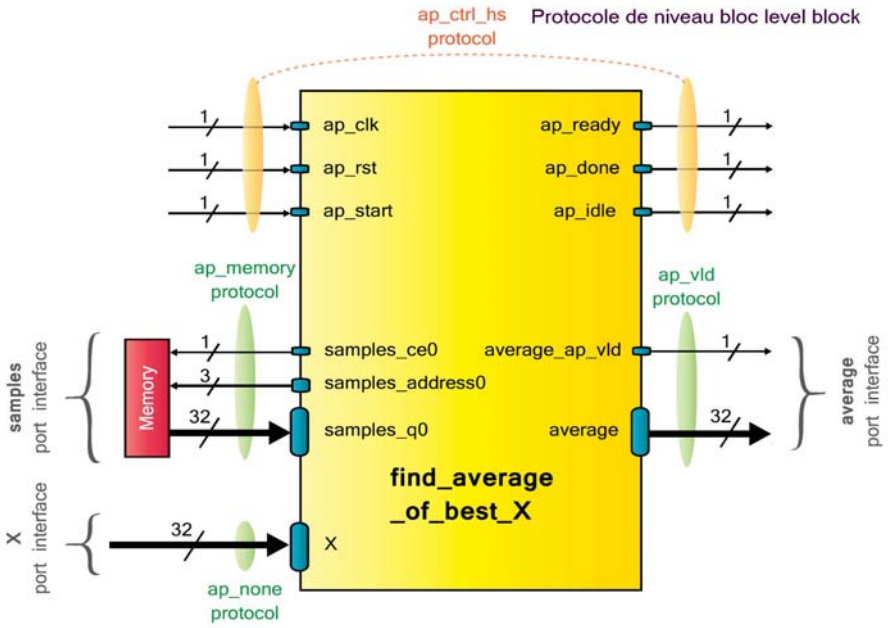
Language	Fixed Point Data Type	Description	Required Header
C++	ap_fixed<*W,I,Q,O,N*>	Signed fixed point number of *I* integer bits and *W-I* fractional bits.	`#include "ap_fixed.h"`
	ap_ufixed<*W,I,Q,O,N*>	Unsigned fixed point number of *I* integer bits and *W-I* fractional bits.	

Le I correspond au nombre de bits de la partie entière W au nombre de bits total (partie entière + fractionnaire).

Les autres lettres ne seront pas décrites

3.5) Les interfaces de block level

Elles sont créées en utilisant le nom de la fonction de plus haut niveau en langage C ou C++. La synthèse d'interface au niveau bloc applique un protocole d'entrées sorties à un IP bloc en additionnant des signaux de contrôle afin de contrôler les opérations de l'IPbloc. Une synthèse au niveau bloc est contrôlé par les modes d'interface ap_ctrl_hs et ap_ctrl_none qui sont appliqués à la function ou port de retour de la function.
Si un protocole de niveau bloc est utilisé il opère indépendamment des protocoles des interfaces de synthèse qui ont été adopté sur chaque port individuel. Il y a aussi deux signaux d'entrées qui sont rajoutés *ap_clk* et *ap_rst*. Ces deux signaux sont nécessaires à cause des opérations internes des blocs qui sont synchrones.
Il est important de noter qu'une interface de bus AXI4_lite Peut-être appliquée aussi pour un protocole d'interface de niveau bloc



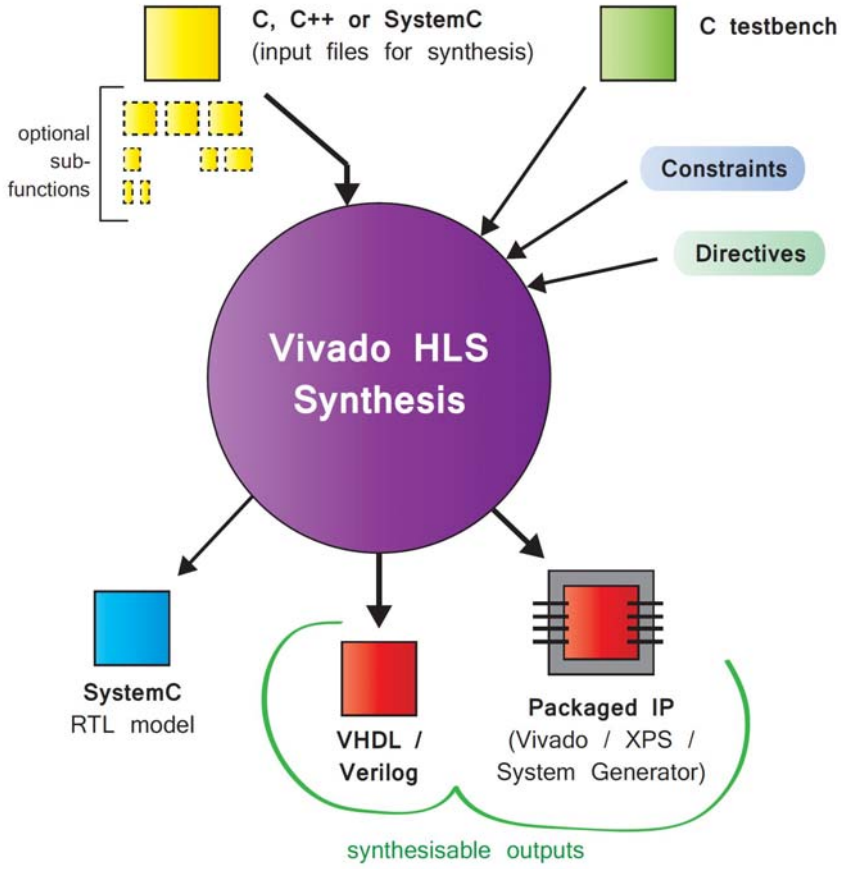
ap_axi_s is a signed interpretation of the AXI stream
• ap_axi_u is an unsigned interpretation of the AXI stream

```
template<int D,int U,int TI,int TD>
struct ap_axis{
ap_int<D> data;
ap_uint<D/8> keep;
ap_uint<D/8> strb;
ap_uint<U> user;
ap_uint<1> last;
ap_uint<TI> id;
ap_uint<TD> dest;
};
```

```
template<int D,int U,int TI,int TD>
struct ap_axiu{
ap_uint<D> data;
ap_uint<D/8> keep;
ap_uint<D/8> strb;
ap_uint<U> user;
ap_uint<1> last;
ap_uint<TI> id;
ap_uint<TD> dest;
};
```

4) Mise en place d'un décodeur sur Vivado HLS

Le traitement des flux de données est illustré ci-dessous



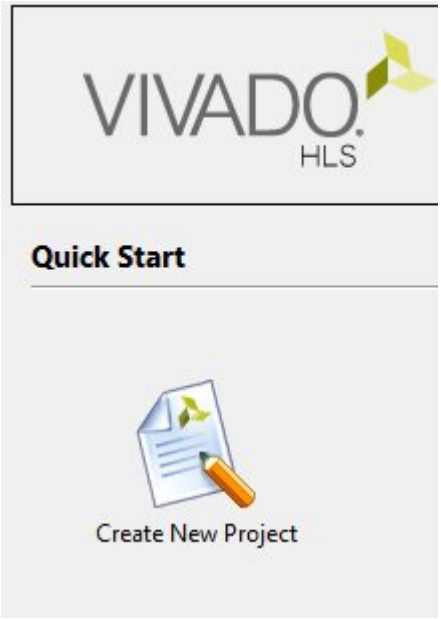
Le fichier de directive permettra d'établir sous forme de texte, les spécifications pour chacune des interfaces de synthèse. Le fichier C testbench fournira les signaux de test pour la validation de la synthèse

4.1) Création du projet

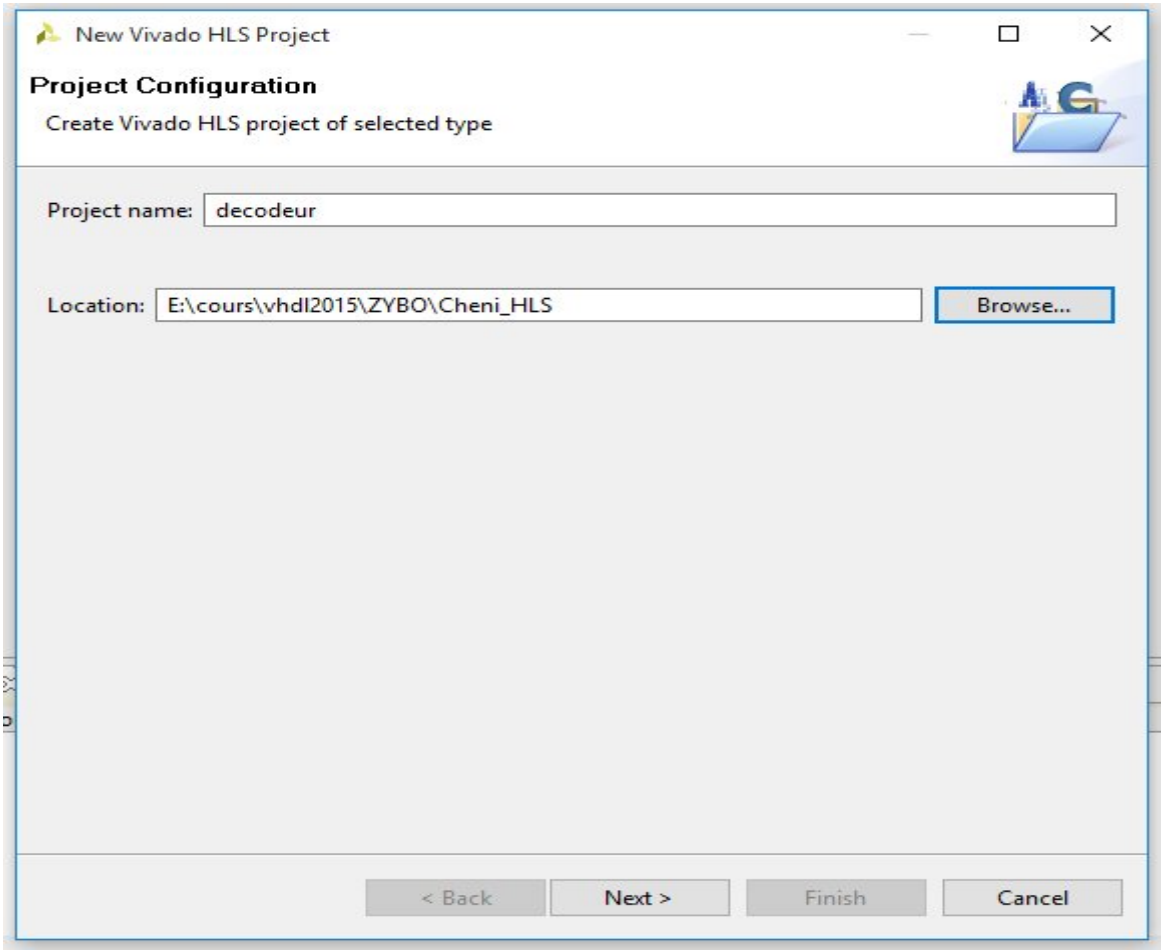
Lancez le logiciel Vivado HLS



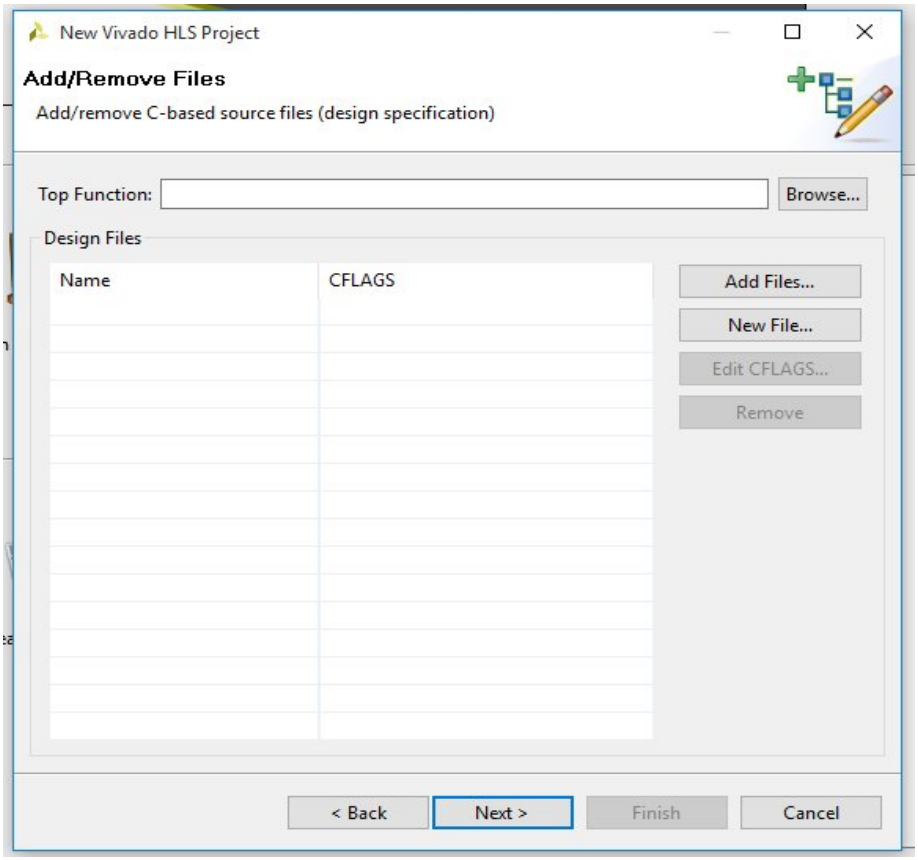
Créez un nouveau projet

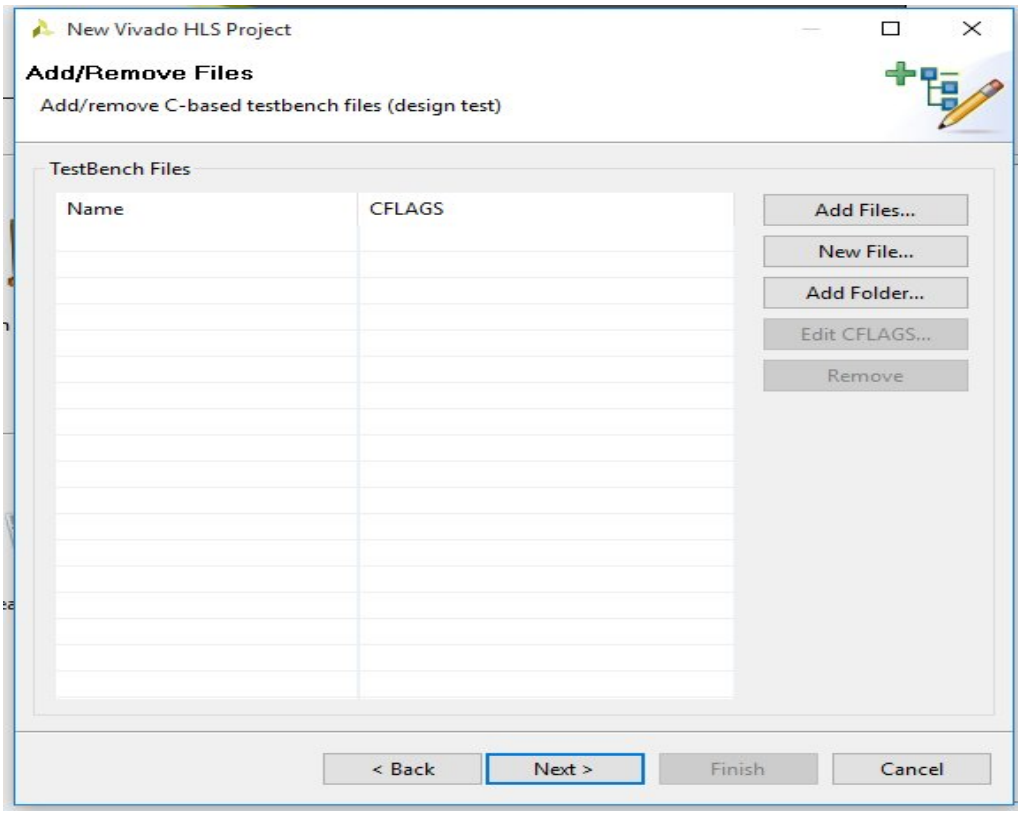


Fournir un nom et un chemin pour ce nouveau projet

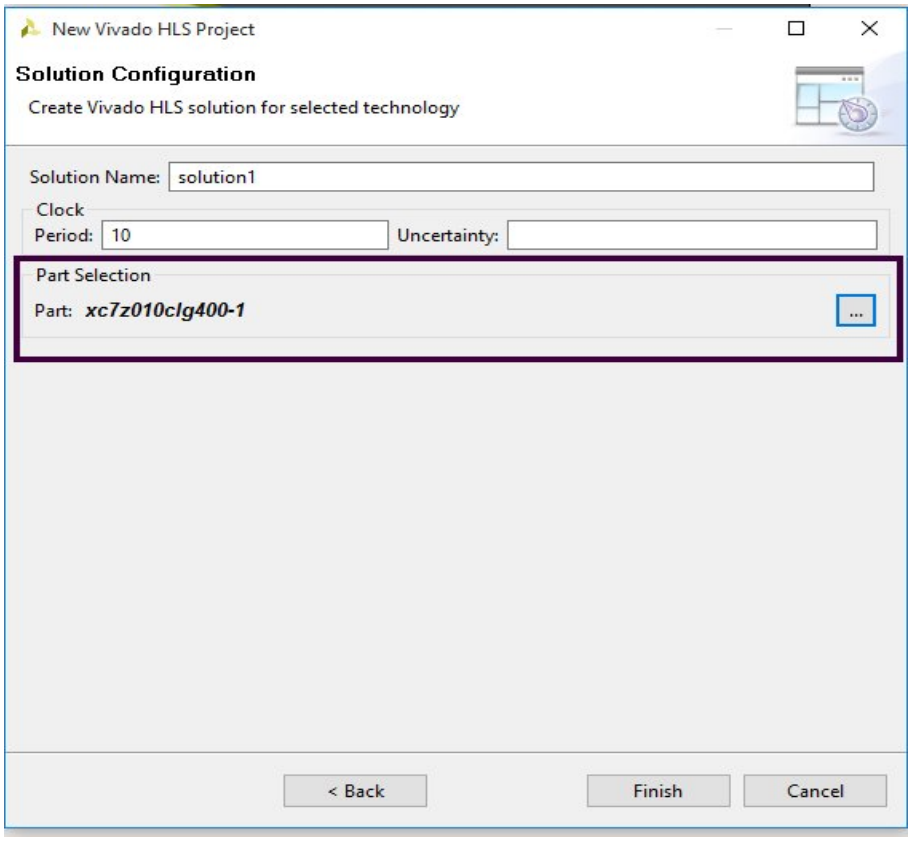


Laissez vierge les deux fenêtres suivantes car aucun fichier source n'a été créé pour l'instant



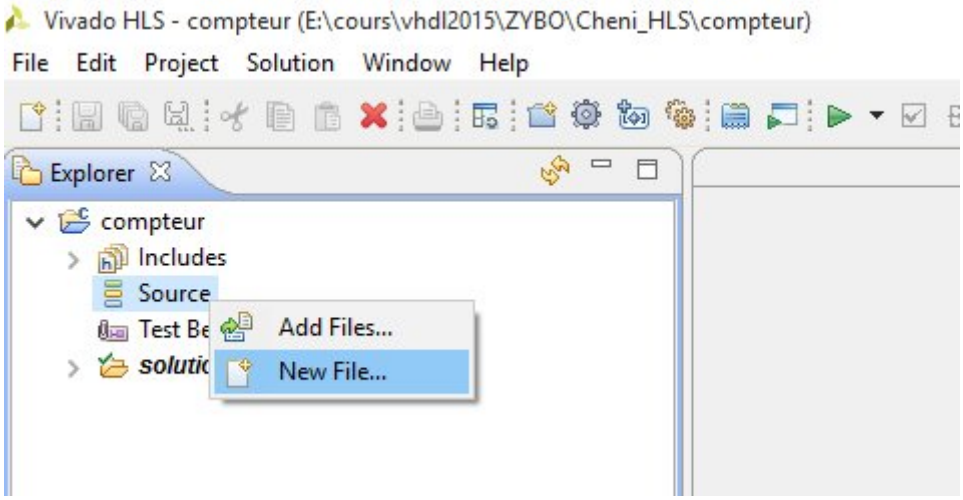


Fournir le type de FPGA pour la plateforme ZYBO

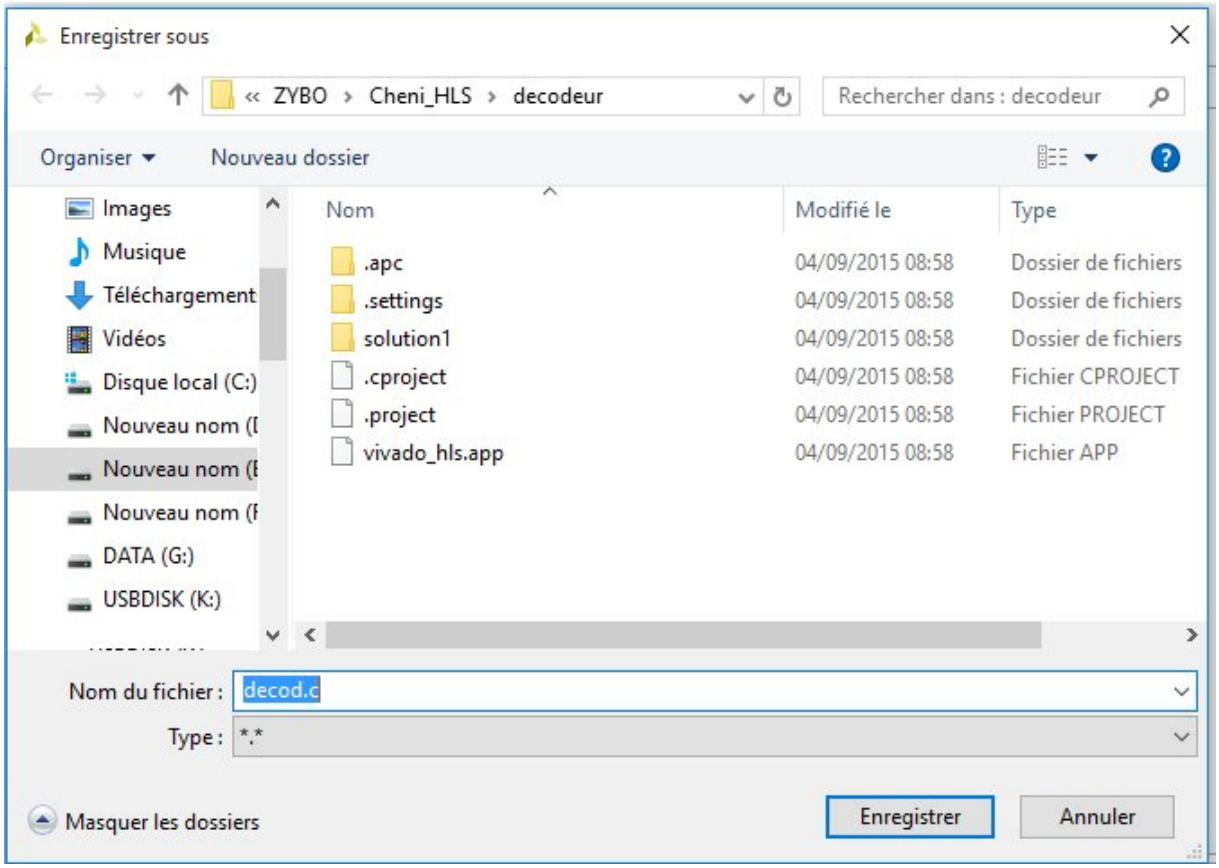


4.2) Création du fichier source et des interfaces de synthèse

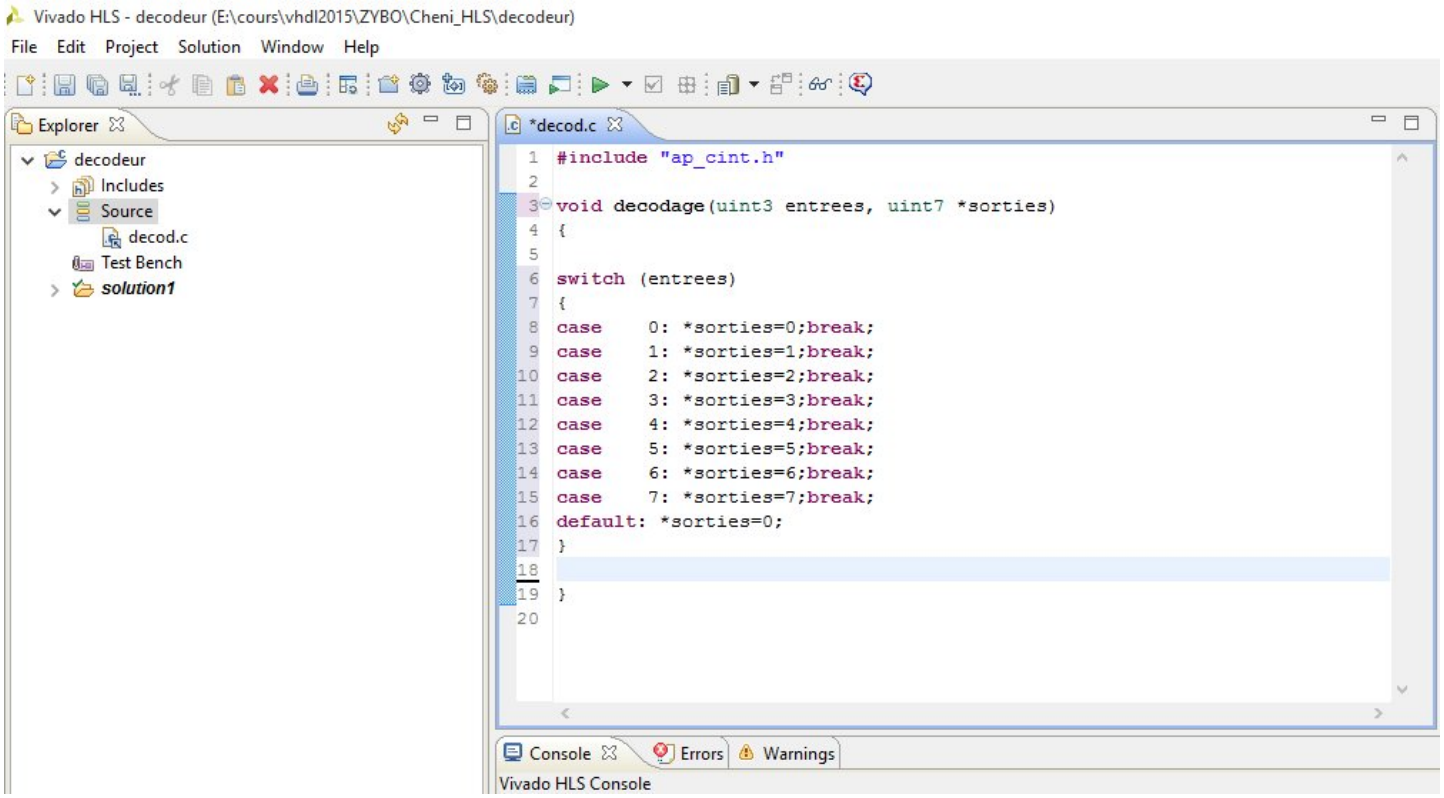
Créez un nouveau fichier source



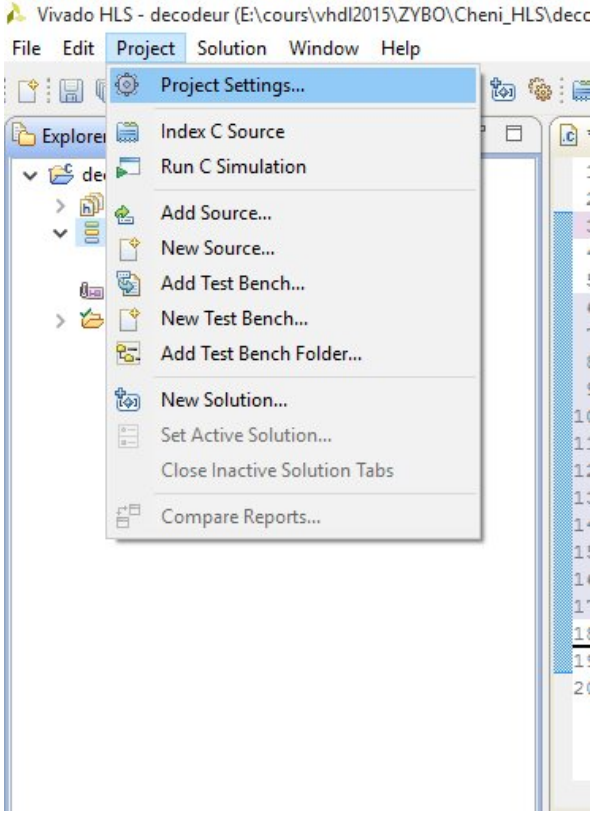
Donnez un nom à notre fichier source



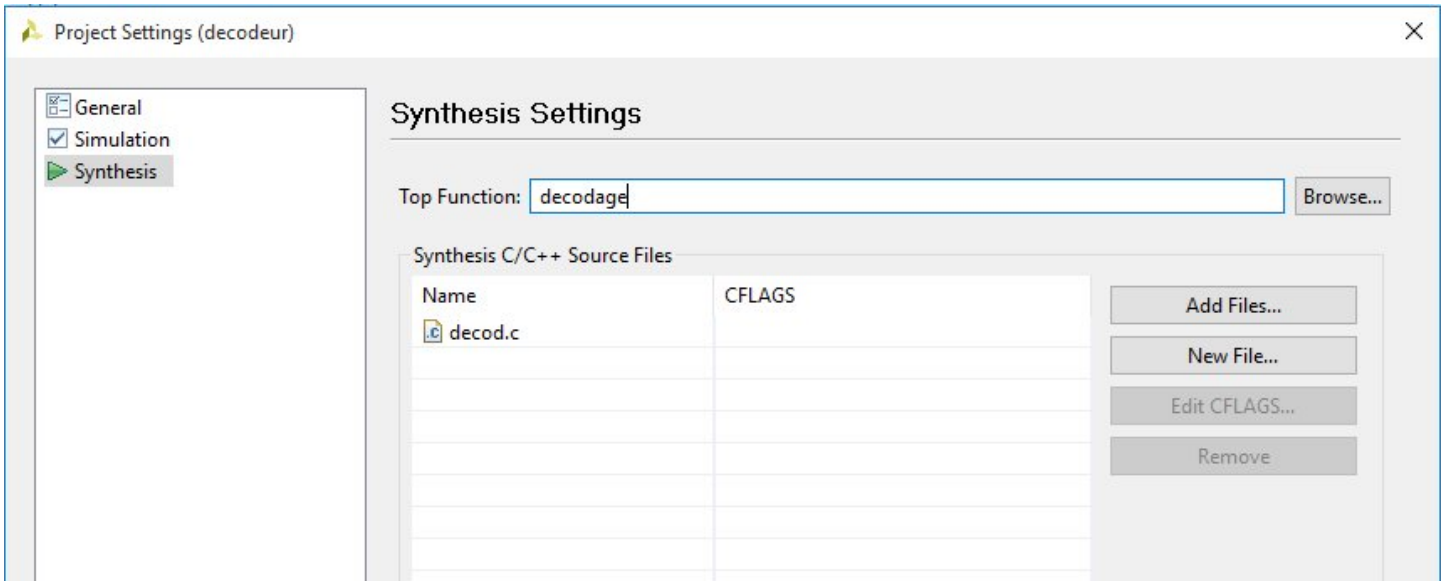
Complétez le fichier source



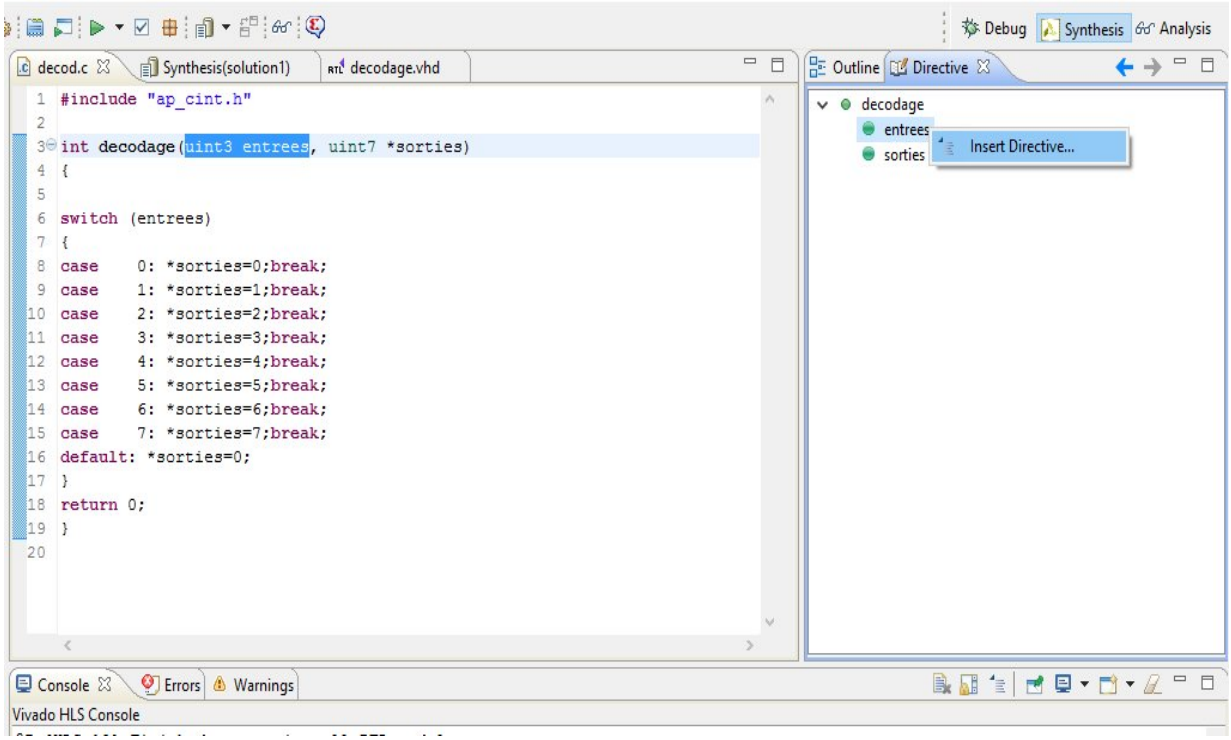
Fournir la fonction de block level (decodage dans notre cas particulier)



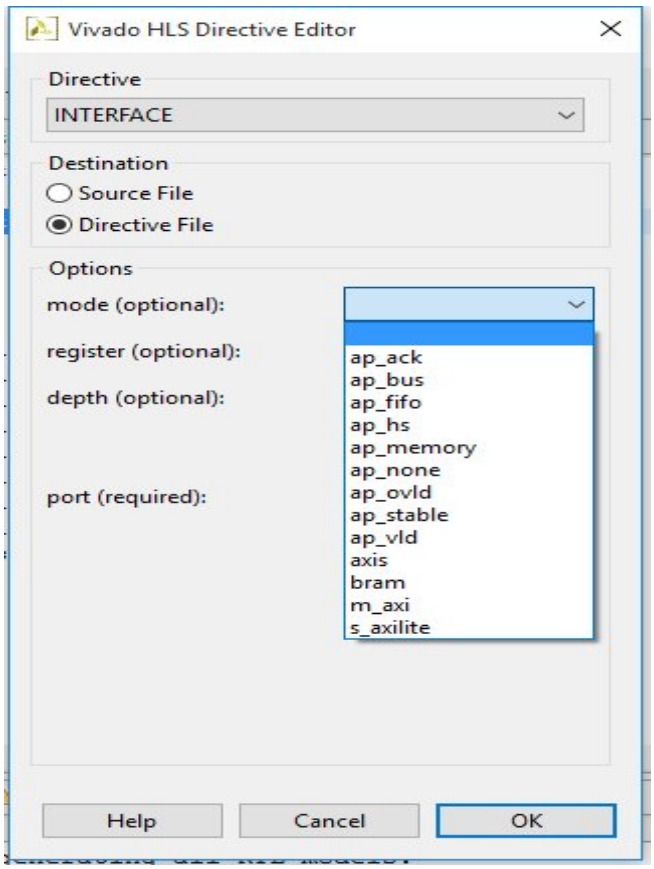
Complétez dans la rubrique Synthesis les champs suivants :



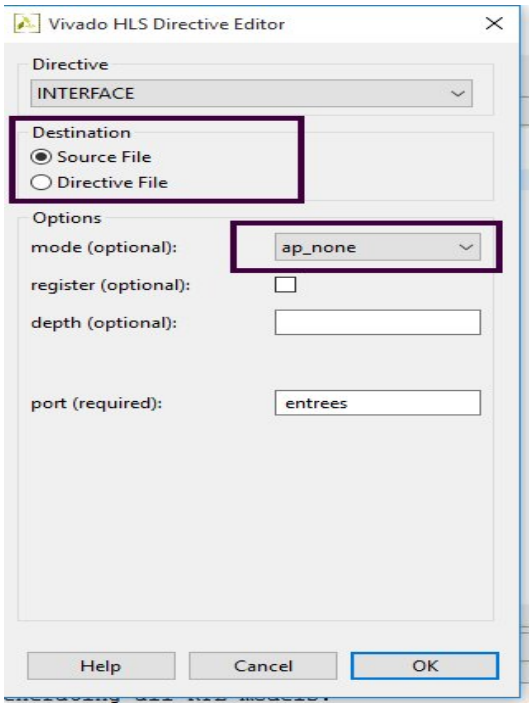
Cliquez droit sur le port d'entrée dans l'onglet directive et insérer une directive



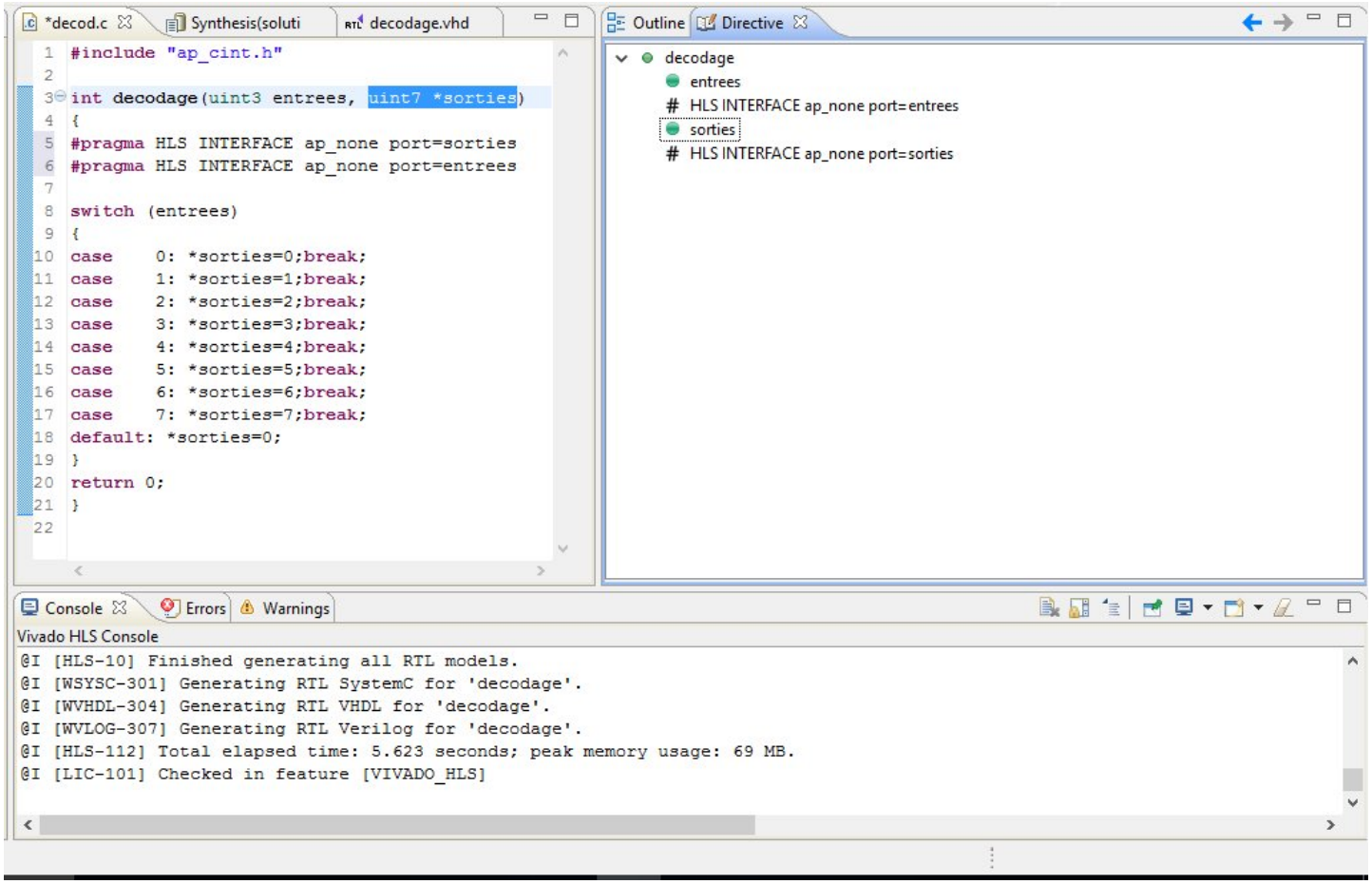
Choisir un protocole pour notre interface de synthèse (aucun dans ce cas)



Les directives pour notre interface de synthèse peuvent apparaitre dans notre fichier source ou dans le fichier de contrainte de directive. On choisira dans le fichier source

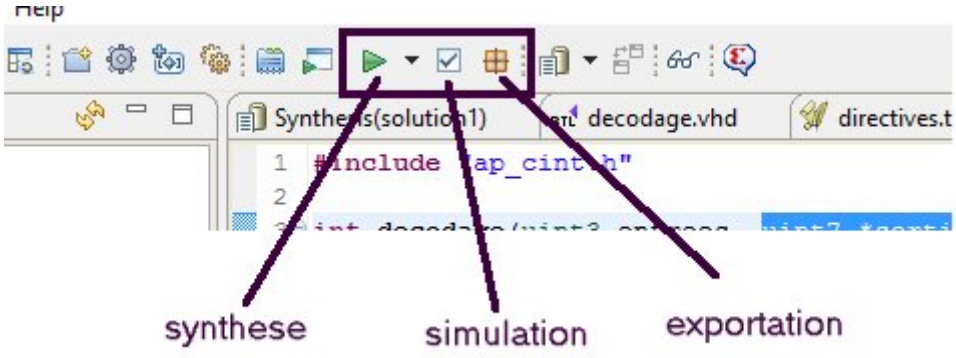


De la même façon créez une interface de synthèse pour le port sorties

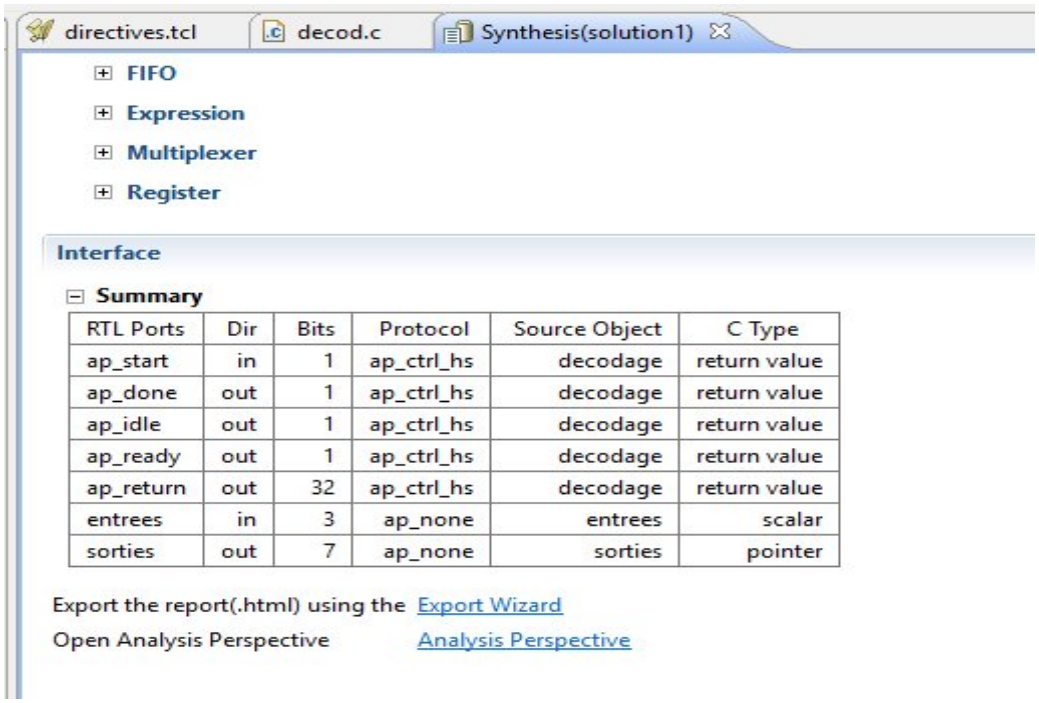


4.3) Synthèse en code VHDL de notre fichier source en C

Cliquez sur le bouton vert pour la synthèse

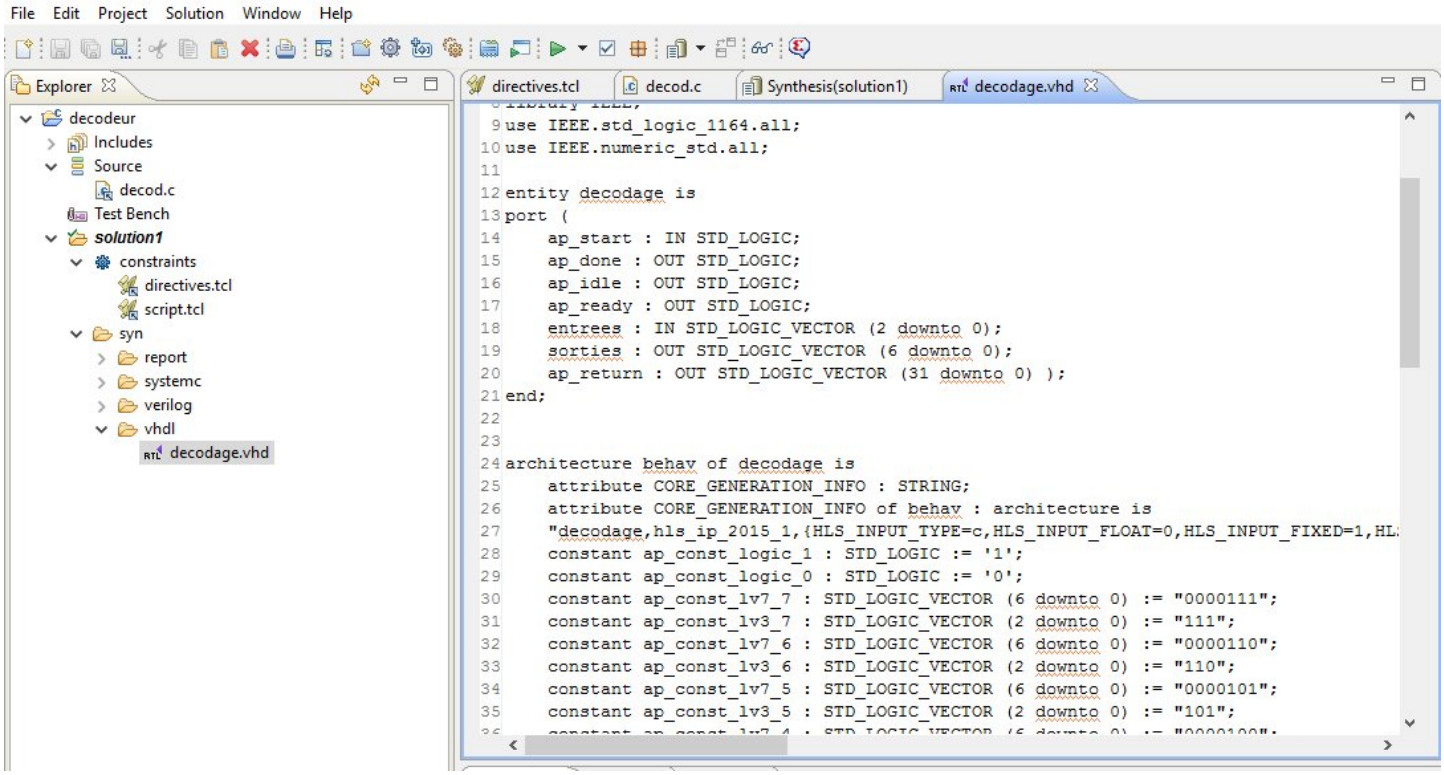


Le rapport de la synthèse nous est donné dans la fenêtre ci-dessous :
On voit apparaitre les différentes interfaces de synthèse



RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_start	in	1	ap_ctrl_hs	decodage	return value
ap_done	out	1	ap_ctrl_hs	decodage	return value
ap_idle	out	1	ap_ctrl_hs	decodage	return value
ap_ready	out	1	ap_ctrl_hs	decodage	return value
ap_return	out	32	ap_ctrl_hs	decodage	return value
entrees	in	3	ap_none	entrees	scalar
sorties	out	7	ap_none	sorties	pointer

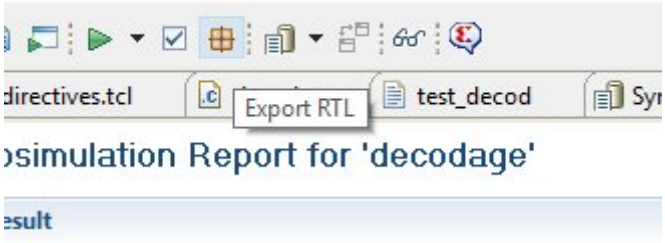
Le code VHDL synthétisé se trouve dans le fichier decodage.vhd



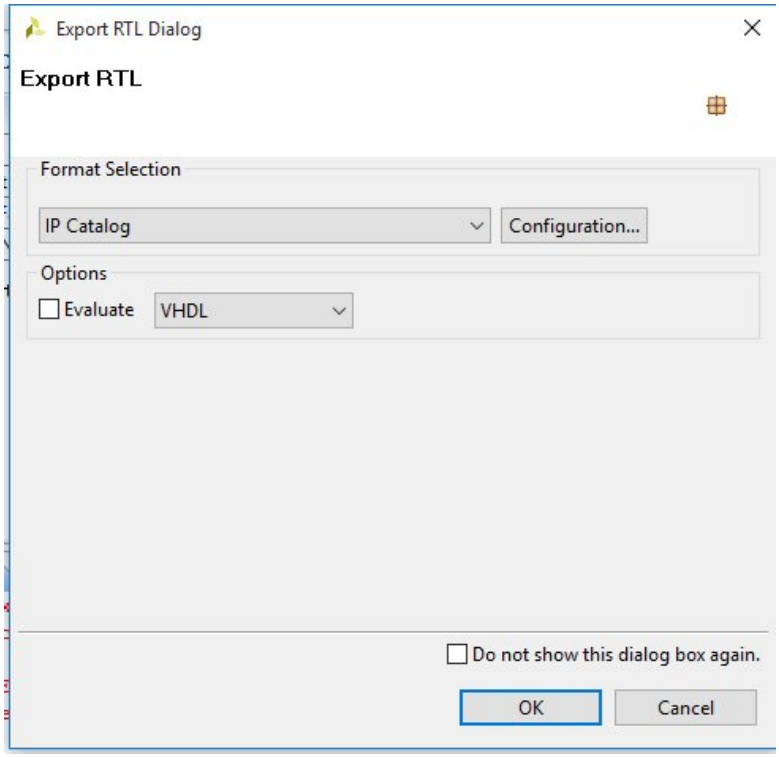
4.4) Simulation de la fonction synthétisée

4.5) Exportation du fichier pour la création d'un IPBLOCK

Cliquez sur exportRTL

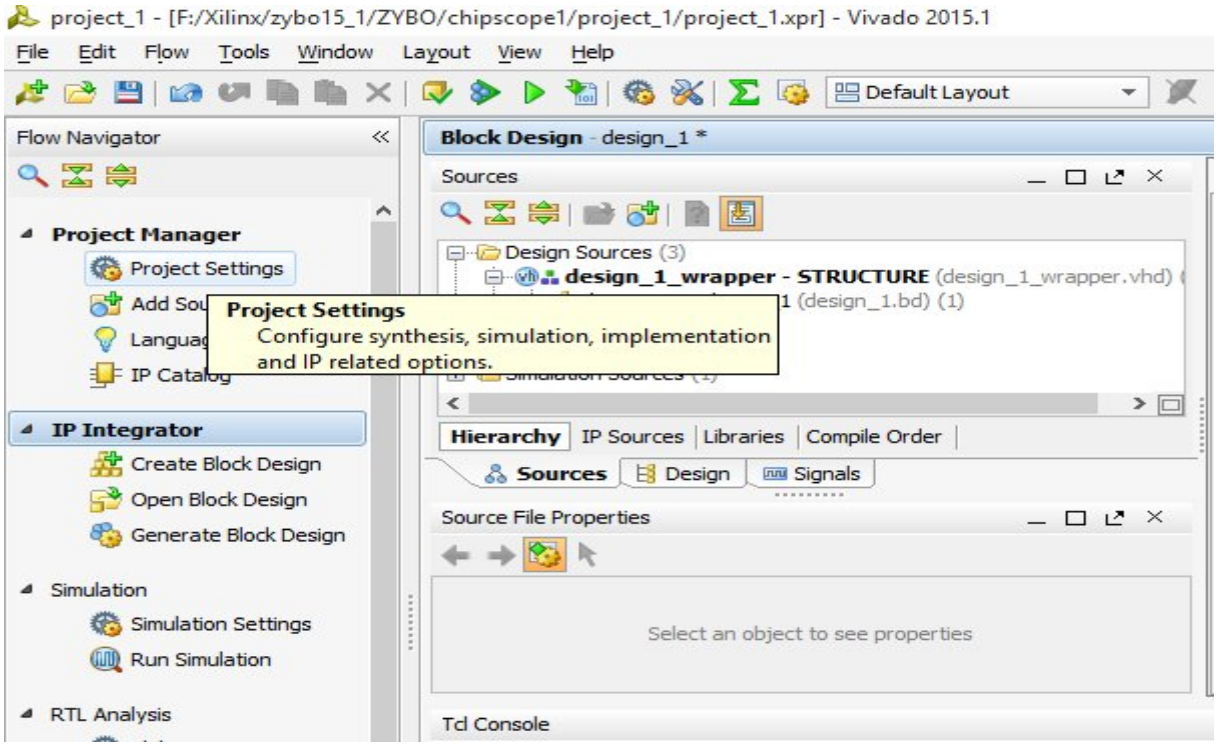


Choisir une exportation en code VHDL pour l'IP Catalog de Xilinx

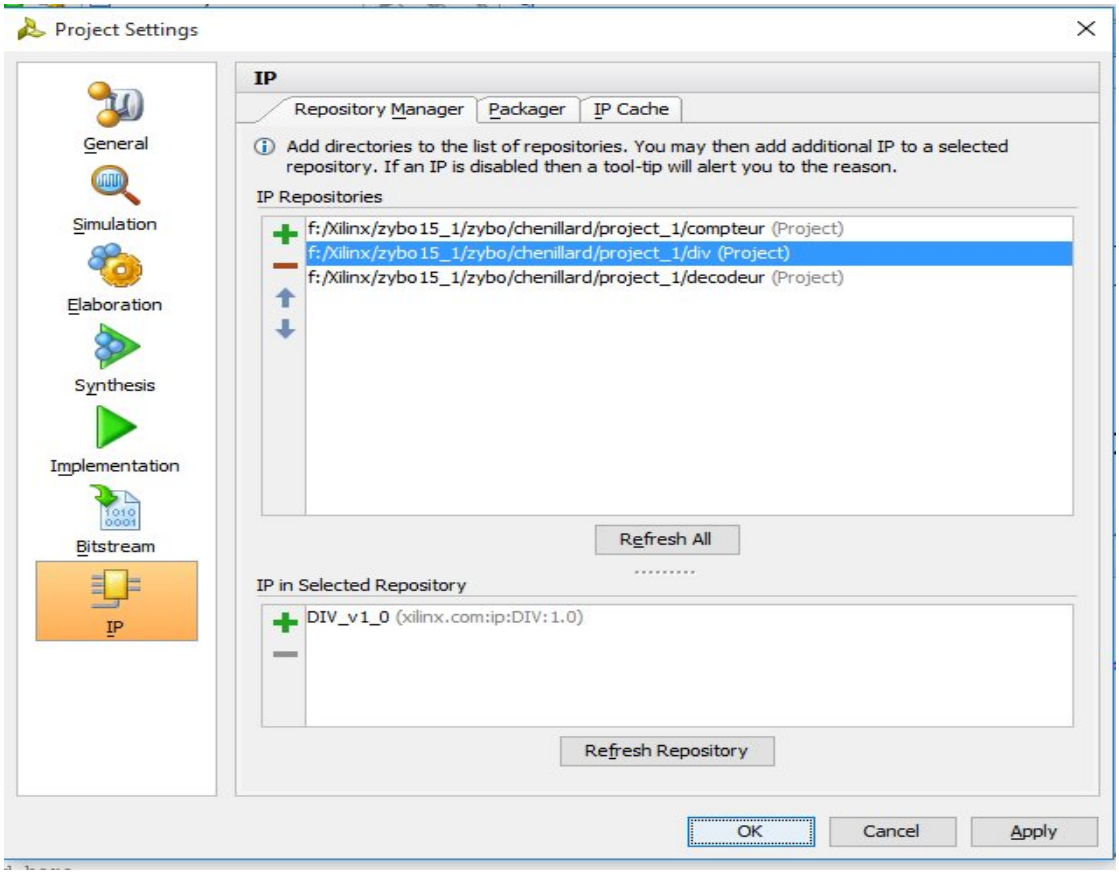


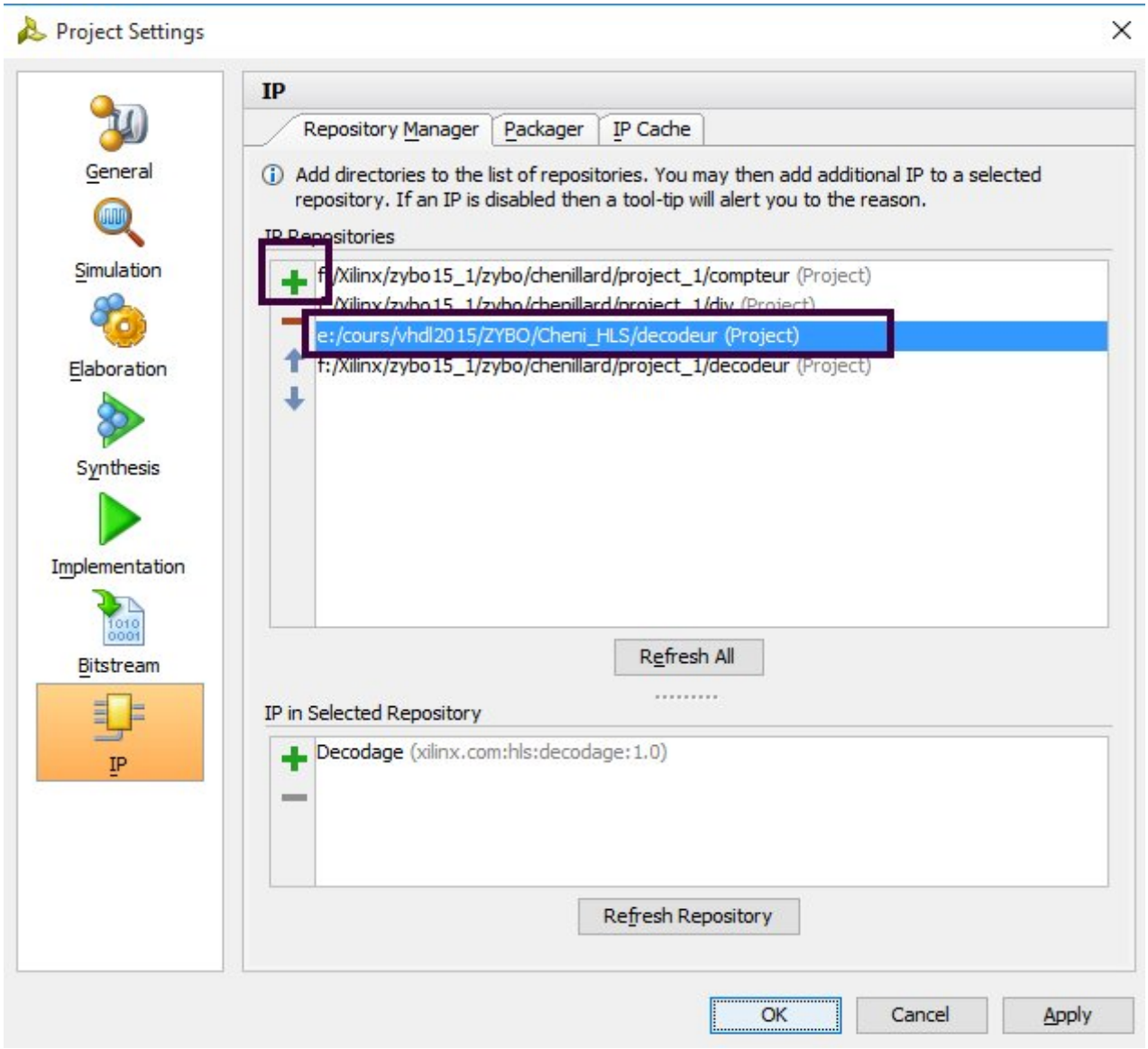
4.6) Importation de notre IPBLOCK sous Vivado

Ouvrir sous Vivado le projet chenillard

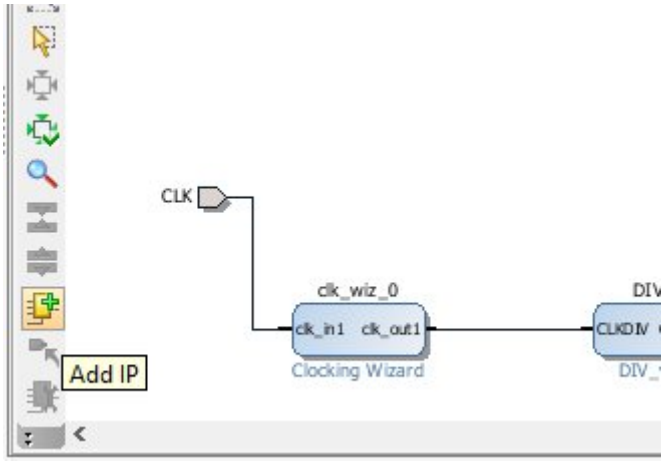


Importez notre IPBLOCK cliquez sur projet settings et additionnez l'IP

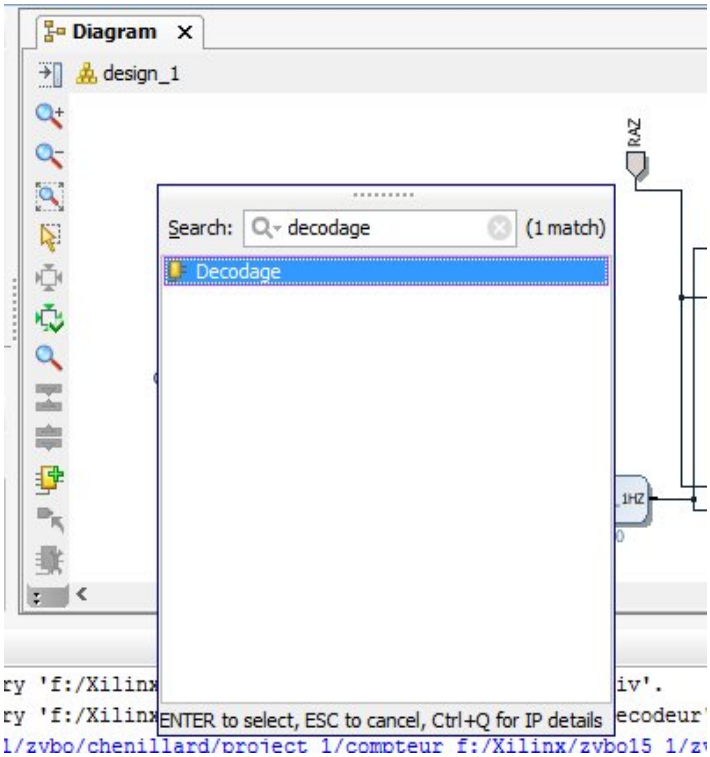




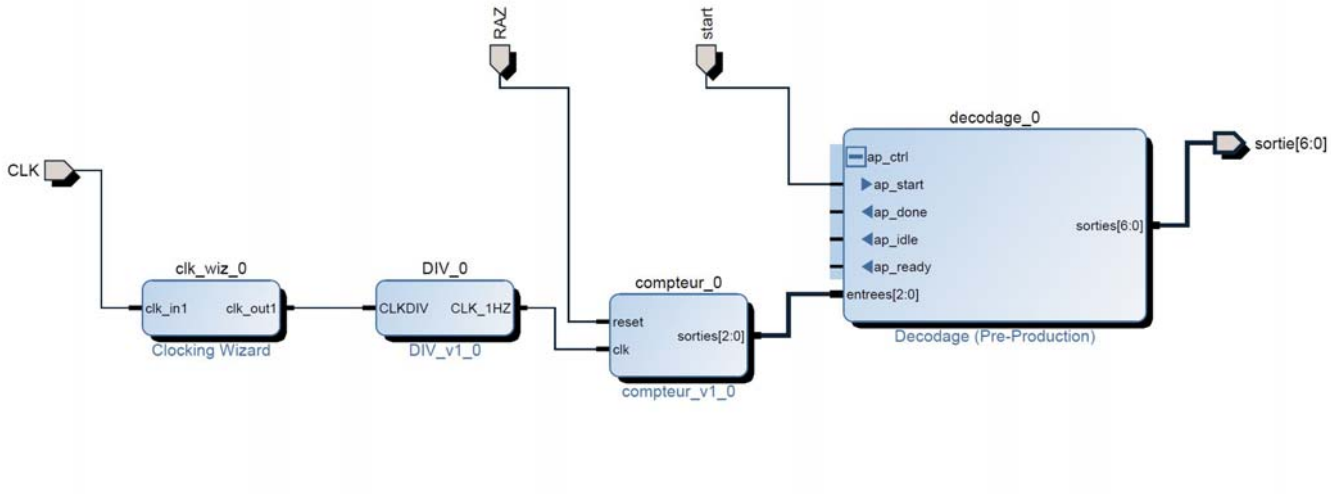
Ouvrir le bloc design system et additionnez un IP bloc



Sélectionnez la function decodage élaborez précédemment



Réalisez le bloc design suivant:



Recréez le HDL WRAPPER, inclure le signal start dans le fichier de contrainte de la carte ZYBO. Générez le bitstream, programmez et vérifiez le bon fonctionnement sur la carte.

The IP-XACT standard is an eXtensible Markup Language (XML) schema for documenting IP using metadata, which is both human readable and machine accessible [2], along with an Application Programming Interface (API) which allows software tools access to the stored meta-data [1]. The tools that are compatible with the IP-XACT standard are able to interpret, configure, implement and change IP blocks that are compliant with the IP meta-data description [1].

Decod.c

```
#include "ap_cint.h"

void decodage(uint3 entrees, uint7 *sorties)
{
#pragma HLS INTERFACE ap_none port=sorties
#pragma HLS INTERFACE ap_none port=entrees

switch (entrees)
{
case  0: *sorties=0;break;
case  1: *sorties=1;break;
case  2: *sorties=2;break;
case  3: *sorties=3;break;
case  4: *sorties=4;break;
case  5: *sorties=5;break;
case  6: *sorties=6;break;
case  7: *sorties=7;break;
default: *sorties=0;
}
//return 0;
}
```

Test_decod.c

```
#include "ap_cint.h"

void decodage(uint3 , uint7 );

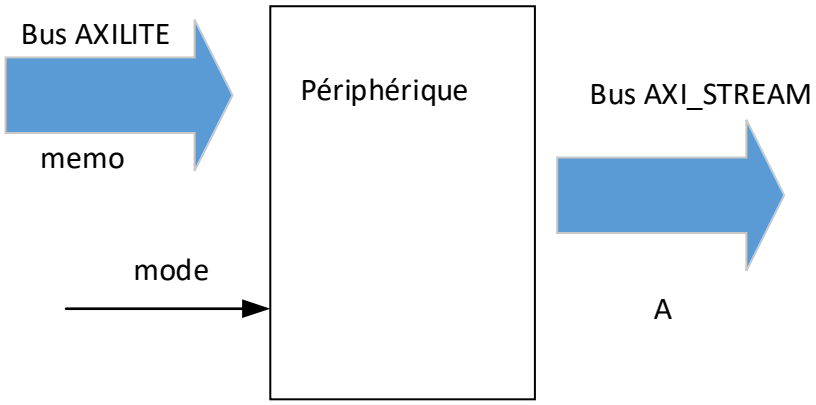

int main(void)
{
uint4 i=0;
uint7 out;
for (i=0; i<8; i++)
{
decodage(i,&out);
}
return 0;

}
```

Création d'un périphérique AXI_LITE et AXI_STREAM sous HLS

1) Problématique

Le périphérique que l'on souhaite créer permettra de réceptionner douze données dans un tampon grâce à l'interface AXI_LITE. Les données dans le tampon seront transmises via une interface du type AXI_STREAM vers l'extérieur. La simulation sera réalisée grâce à l'outil BFM qui permet d'émuler des bus du type AXI. Le corps du modèle aura la forme suivante :



Afin de faciliter le développement on utilisera l'outil HLS pour générer le modèle de notre périphérique

2) Création du périphérique sous HLS

Créez un projet sous HLS, et complétez le code suivant pour le fichier source

```
#include "ap_axi_sdata.h"

int indic=1;

void example(ap_axis<32, 1, 1, 1> *A, int memo[12], bool mode) {
#pragma HLS INTERFACE s_axilite depth=12 port=memo
#pragma HLS INTERFACE axis port=A
int i;
if (mode ==1 && indic==1){
      i=0;
            for(i=0;i<12;i++) {
# pragma HLS PIPELINE
                  A->data = memo[i];
                  A->strb = i;
                  A->keep = 1;
                  A->user = 0;
                  A->id = 0;
                  A->dest = 1;
```

```
                if (i==11) {A->last = 1;}
                else {A->last = 0;}

            }
            indic=0;

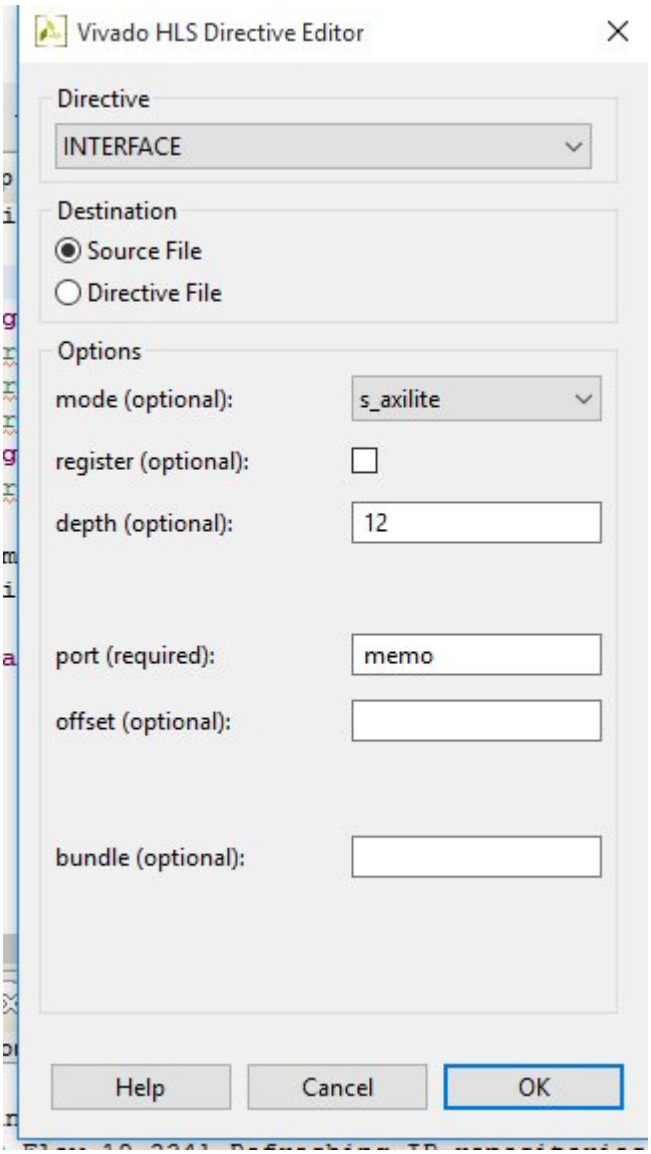
}
if (mode==0)
 indic=1;

      }
```

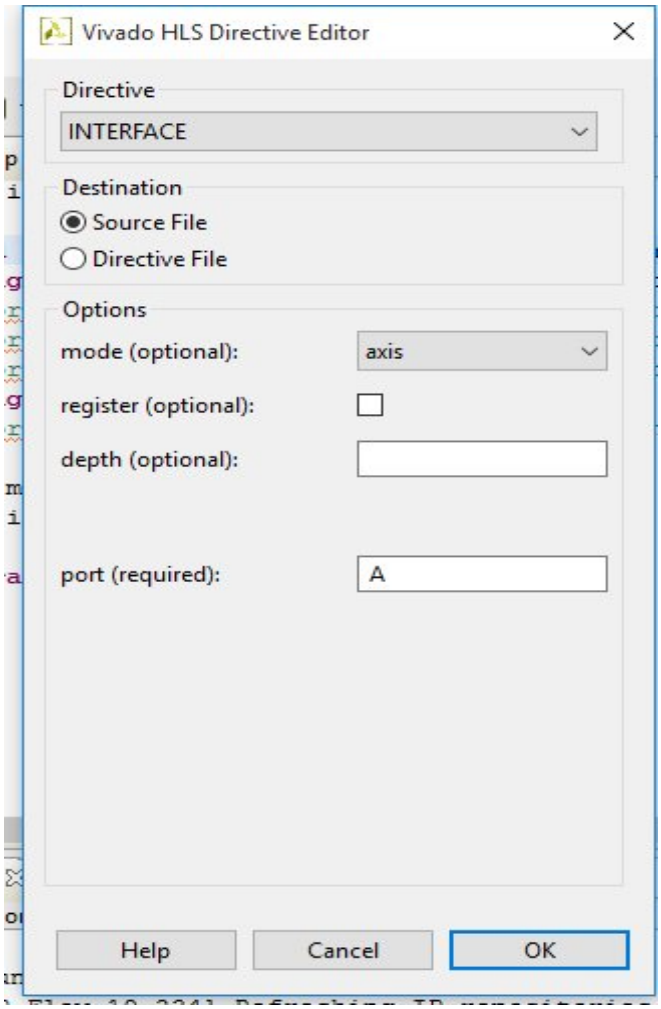
Le bus AXI_STREAM peut être décomposé en plusieurs signaux qui sont déclarés dans une structure ap_axis dans le fichier d'en tête "ap_axi_sdata.h".
La syntaxe de la déclaration est la suivante :

```
ap_axis<nbre de bits de data, nbre de bits de user, nbre de bits de id , nbre de bits de dest>
```

la variable A est du type pointeur car c'est une sortie avec une interface AXI_STREAM



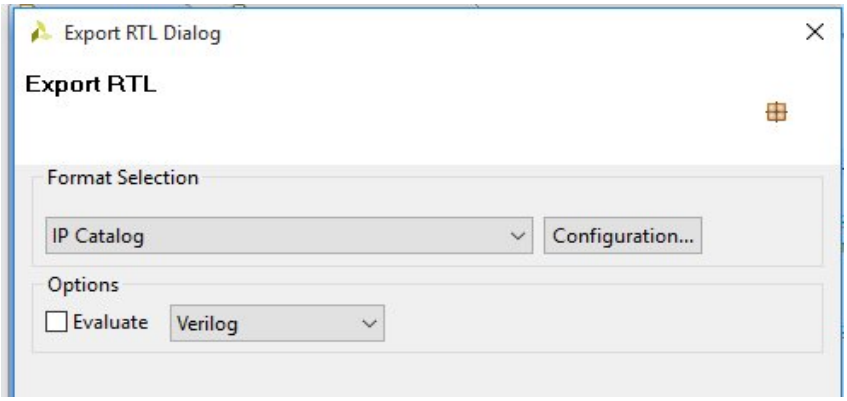
on peut remarquer que le tableau `memo[12]` utilise une interface du type AXI_lite avec une profondeur de 12 cases mémoires.



La variable `bool` mode permettra de créer un simple signal sans aucun protocole ni interface.

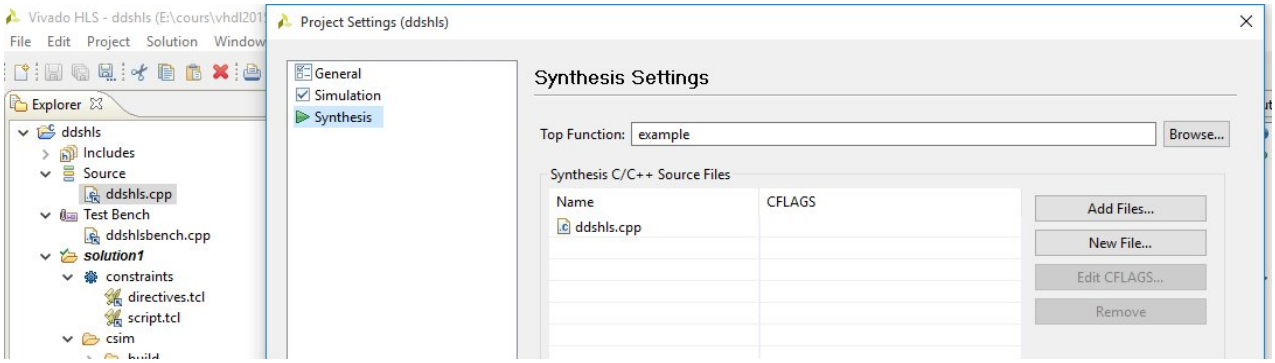
A noter que la directive `# pragma` HLS PIPELINE permettra d'accélérer les échanges dans la mémoire.

Lancez la phase de synthèse puis l'exportation de notre IPCORE :



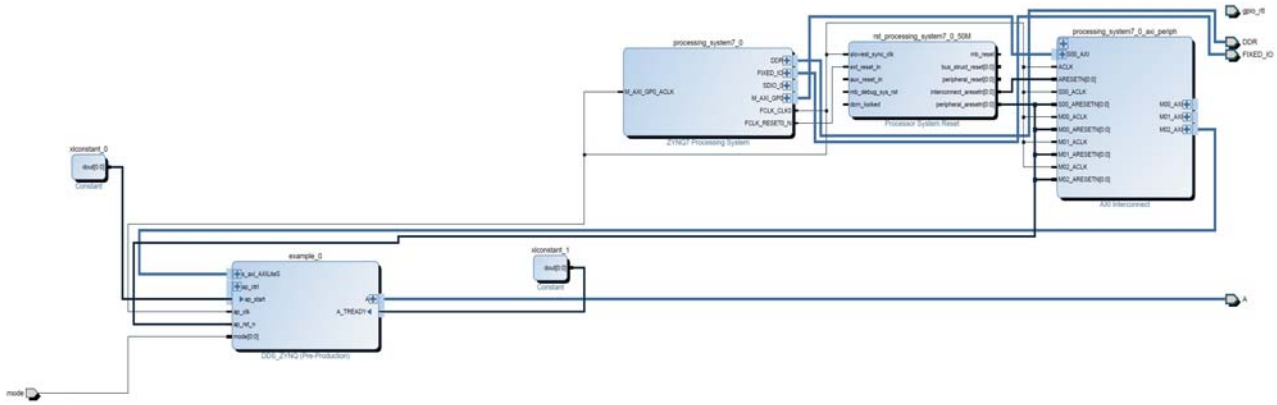
Le langage Verilog n'a pas été choisi au hasard car l'émulateur de bus AXI nommé BFM (Bus fonctionnal model) a été écrit en langage verilog notamment son fichier testbench.

Attention au projet settings et à la top fonction



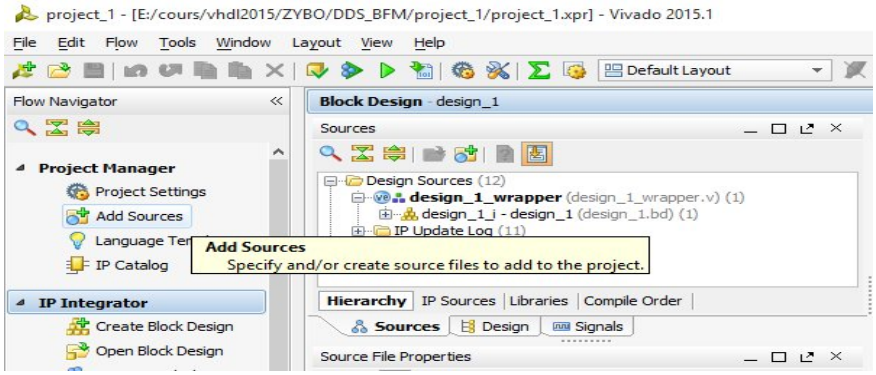
3) Création de notre simulation sous Vivado

Créez un projet en verilog et le board design suivant avec notre périphérique example

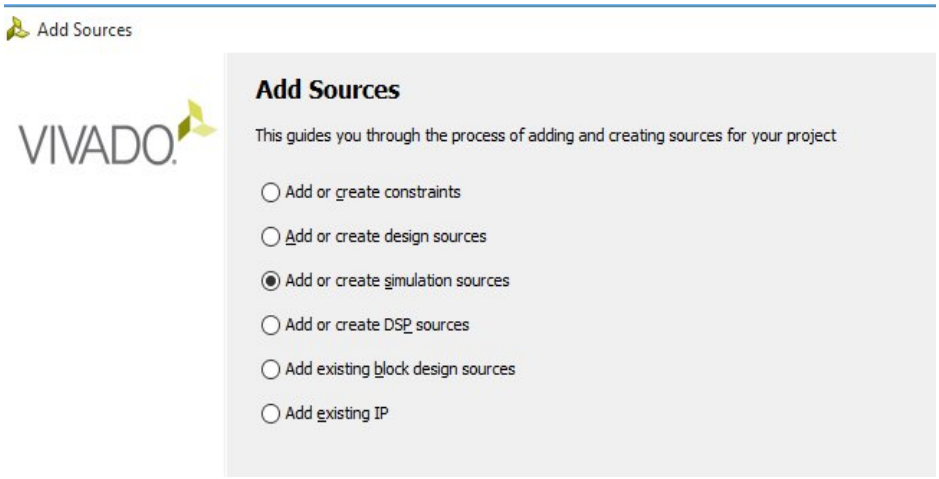


Attention à ne pas brancher les signaux du bus AXI sur une sortie pour visualiser les chronogrammes car ceux-ci sont déconnectés du bus
Créez un HDL Wrapper pour ce board design

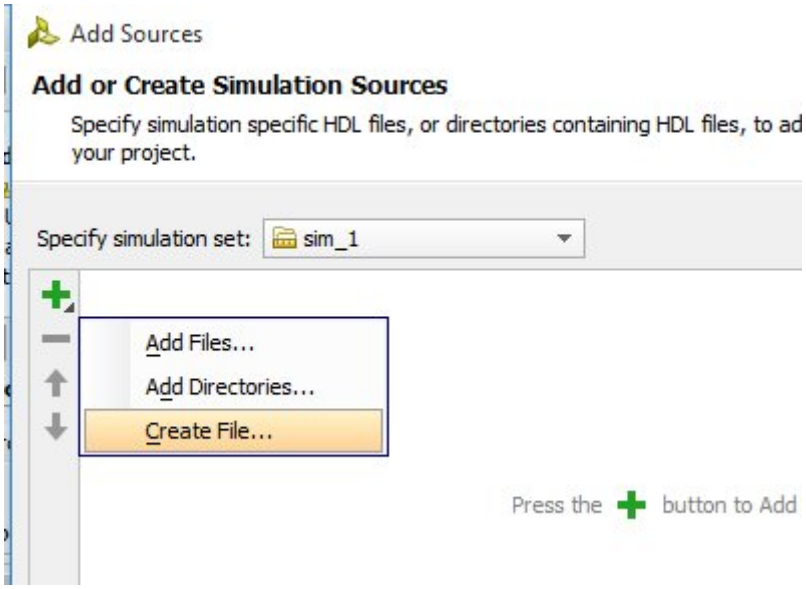
Puis cliquez sur add sources



Et sélectionnez un fichier de simulation



Créez ce fichier de simulation



Le fichier sera baptisé tb.v

Complétez ce fichier avec le code suivant :

```
timescale 1ns / 1ps

`uselib lib=unisims_ver

module tb;  // obligatoire

reg [0:0]PS_CLK;
reg  [0:0]tb_ARESETn;
reg  [0:0]mode;
//reg [31:0]data;
/*reg  [31:0]A_tdata;
reg  [0:0]A_tdest;
reg  [0:0]A_tid;
reg  [3:0]A_tkeep;
reg  [0:0]A_tlast;
reg  [3:0]A_tsrtrb;
reg  [0:0]A_tuser;
reg  [0:0]A_tvalid;*/
wire PS_CLK1;
wire tb1_ARESETn;
wire mode1;
//wire [31:0] A_tdata1;
wire A_tdest;
wire A_tid;
wire [3:0] A_tkeep;
wire [31:0] A_tdata;
wire A_tlast;
wire [3:0]  A_tsrtrb;
wire A_tuser;
wire A_tvalid;
//wire data1;
reg      [31 : 0]       response;
integer                          i;

// attention à prendre comme nom design_1_wrapper correspondant au HDL wrapper

design_1_wrapper zynq_sys
     (.A_tdata(A_tdata),
      .A_tdest(A_tdest),
      .A_tid(A_tid),
      .A_tkeep(A_tkeep),
      .A_tlast(A_tlast),
      .A_tstrb(A_tstrb),
      .A_tuser(A_tuser),
      .A_tvalid(A_tvalid),
```

```
      .FIXED_IO_ddr_vrn(),
      .FIXED_IO_ddr_vrp(),
      .FIXED_IO_mio(),
      .FIXED_IO_ps_clk(PS_CLK1),
      .FIXED_IO_ps_porb(tb1_ARESETn),
      .mode(mode1),
      .FIXED_IO_ps_srstb(tb1_ARESETn));

//------------------------------------------------------------------------
        // Simple Clock Generator
        //------------------------------------------------------------------------
        initial begin
          PS_CLK = 1'b0;
          forever #10 PS_CLK = !PS_CLK;
        end

       assign PS_CLK1 =PS_CLK;

      assign tb1_ARESETn = tb_ARESETn ;


      initial begin
      mode = 1'b0;

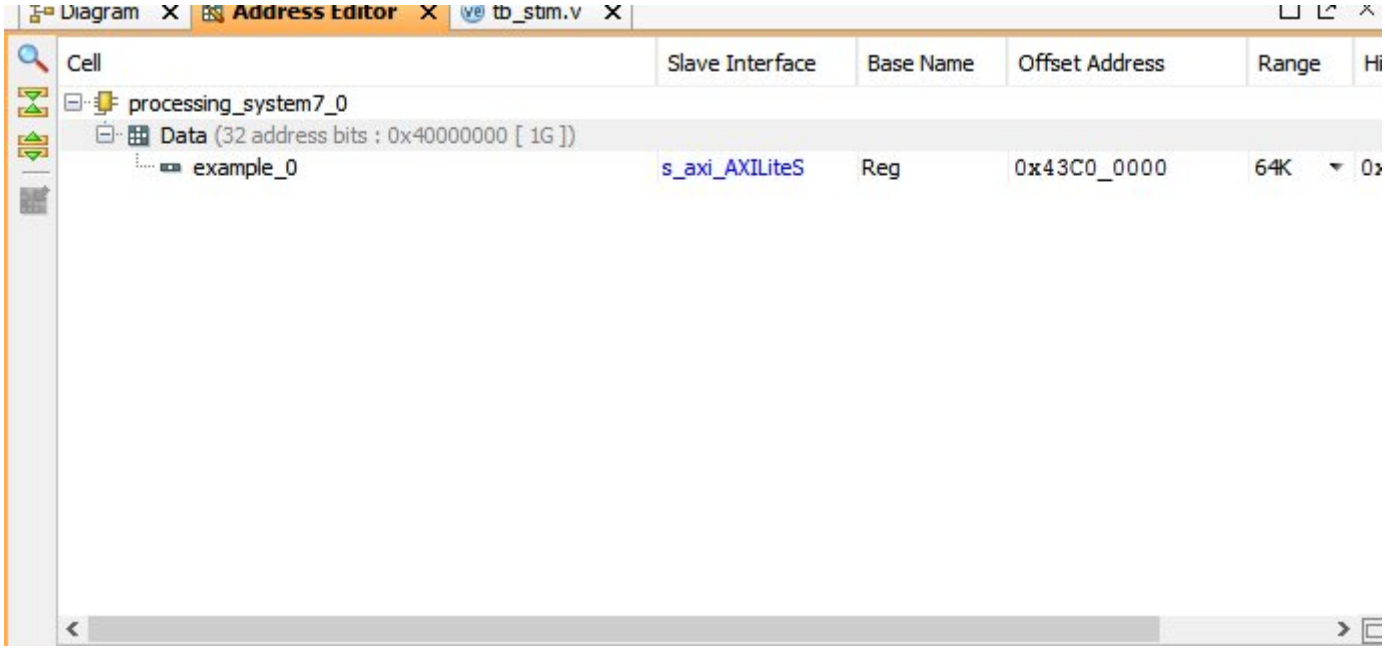
        tb_ARESETn = 1'b0;
        // Release the reset on the posedge of the clk.
        repeat(80)@(posedge PS_CLK);
        tb_ARESETn = 1'b1;
        @(posedge PS_CLK);

        //Reset the PL
        zynq_sys.design_1_i.processing_system7_0.inst.fpga_soft_reset(32'h1);
        zynq_sys.design_1_i.processing_system7_0.inst.fpga_soft_reset(32'h0);

        // Single Read/writes:

for (i = 0; i < 12 ; i = i+1) begin
         zynq_sys.design_1_i.processing_system7_0.inst.write_data(32'h43C00040+i*4,4,
32'h00000000+i, response);
      end
        mode = 1'b1;
         end
assign mode1 = mode  ;
```

Lors de la phase de synthèse sous HLS celui-ci fournit les pilotes pour SDK. Pour adresser le périphérique il nous faut l'adresse de base du périphérique :



Dans Adress editor soit l'adresse 0x43C00000.

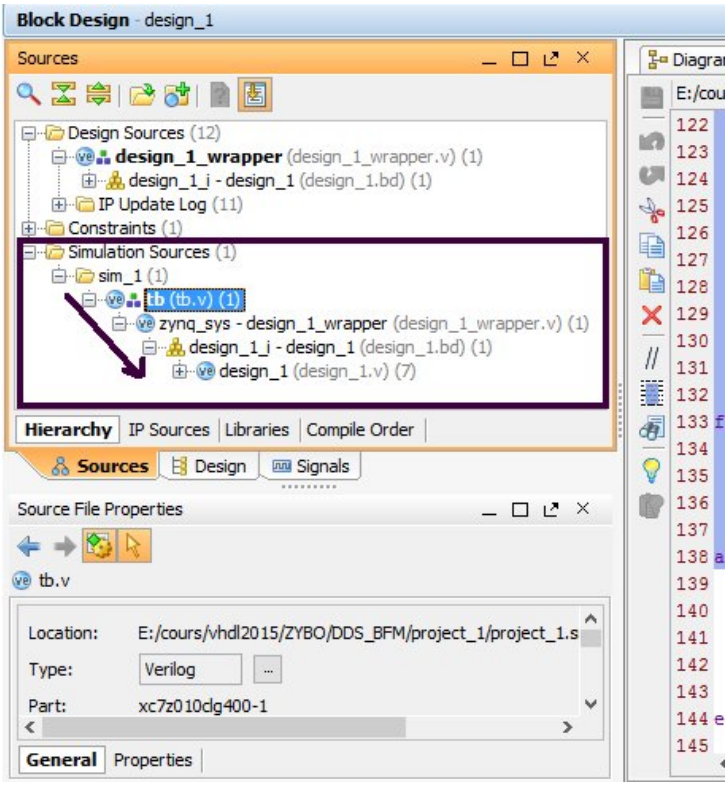
Pour l'offset d'adresse le fichier xexample_hw.h nous donne :

```
#define XEXAMPLE_AXILITES_ADDR_MEMO_BASE 0x40
#define XEXAMPLE_AXILITES_ADDR_MEMO_HIGH 0x7f
#define XEXAMPLE_AXILITES_WIDTH_MEMO     32
#define XEXAMPLE_AXILITES_DEPTH_MEMO     12
```

On retrouve bien dans notre programme l'instruction:
zynq_sys.design_1_i.processing_system7_0.inst.write_data(32'**h43C00040**

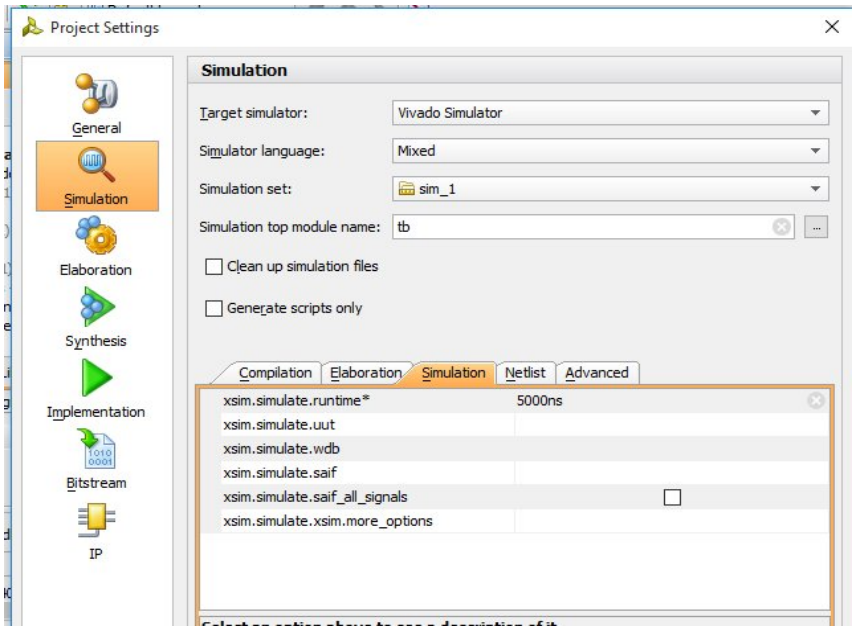
On doit respecter la syntaxe suivante pour que les commandes agissent sur le PS (initialisation, lecture, écriture …..)

zynq_sys.design_1_i.processing_system7_0. . Il doit respecter l'arborescence donnée par la flèche:

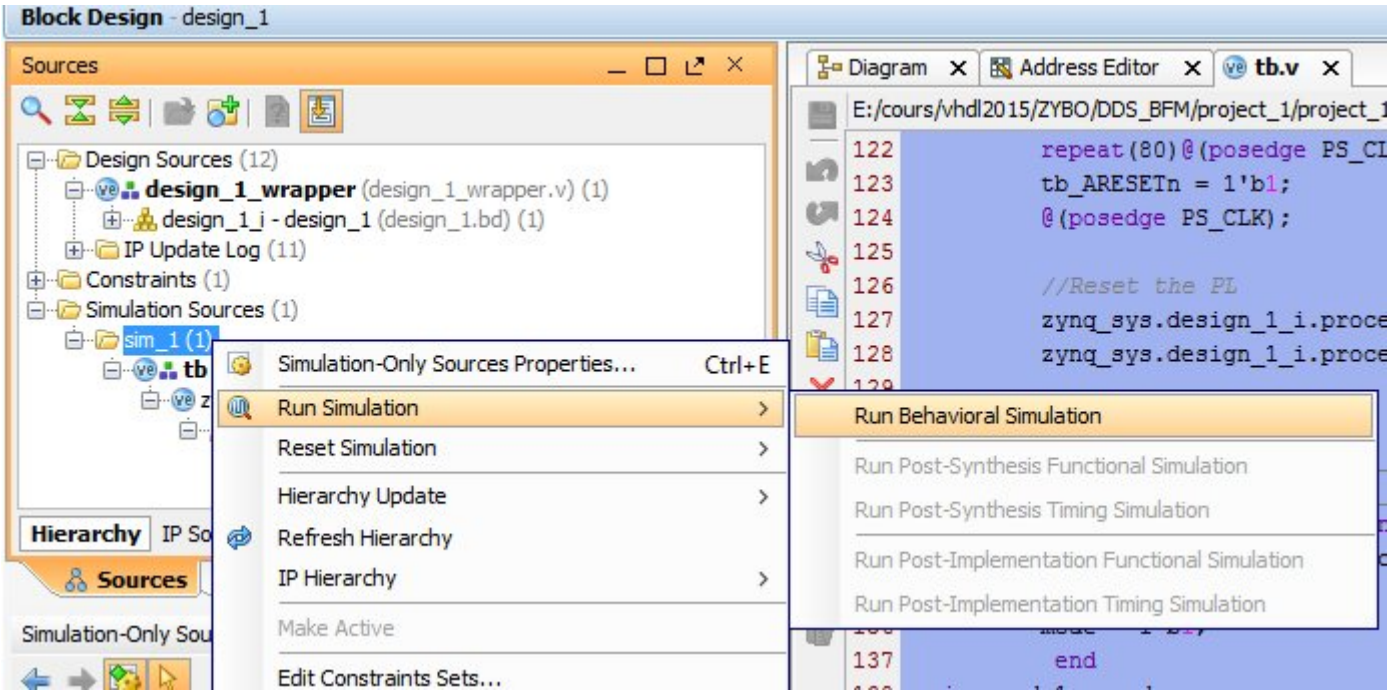


Les différentes commandes qui agissent sur le PS sont données dans la documentation du ZYNQ BFM.

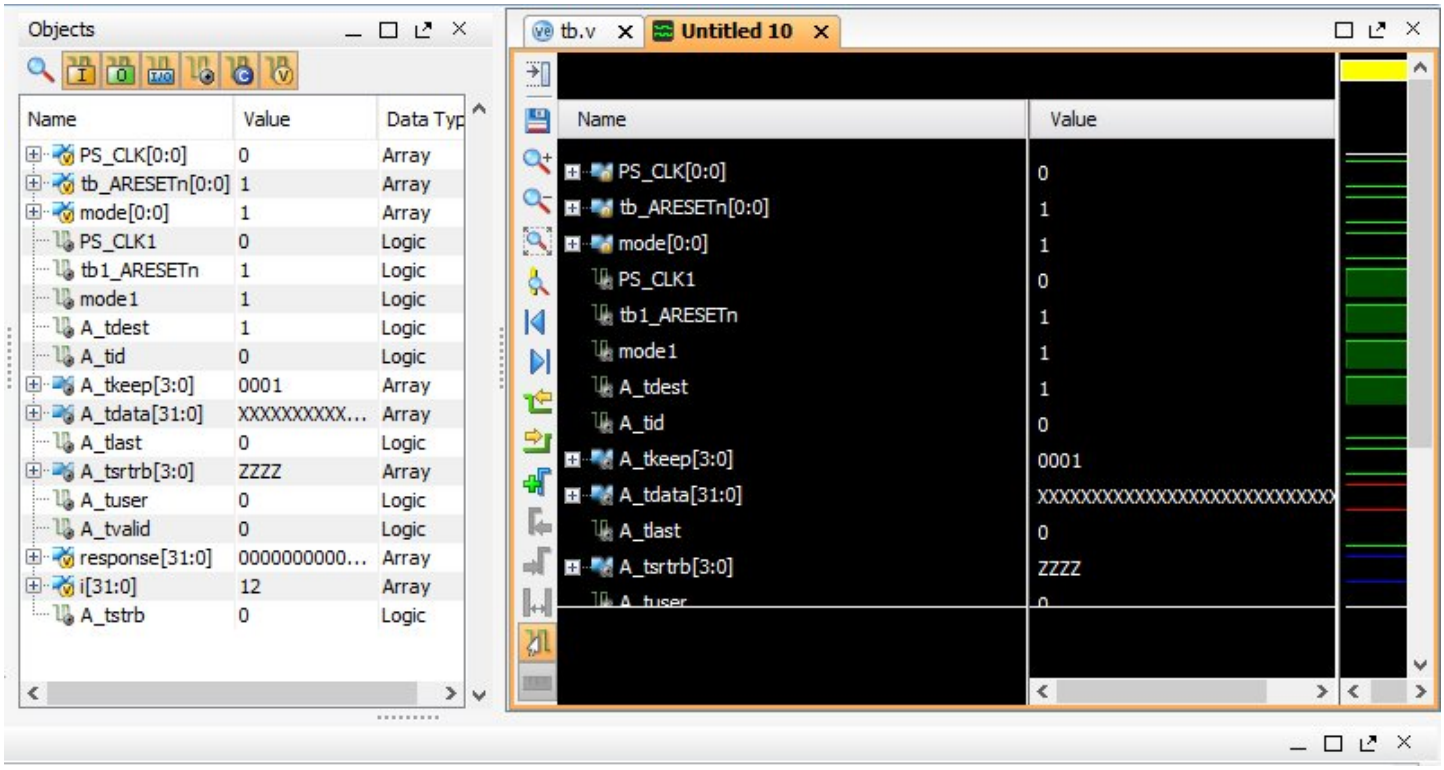
Paramétrez la simulation avec le fichier testbench tb.v:



Lancez la simulation après un file -→ save all files :
Cliquez droit sur sim_1



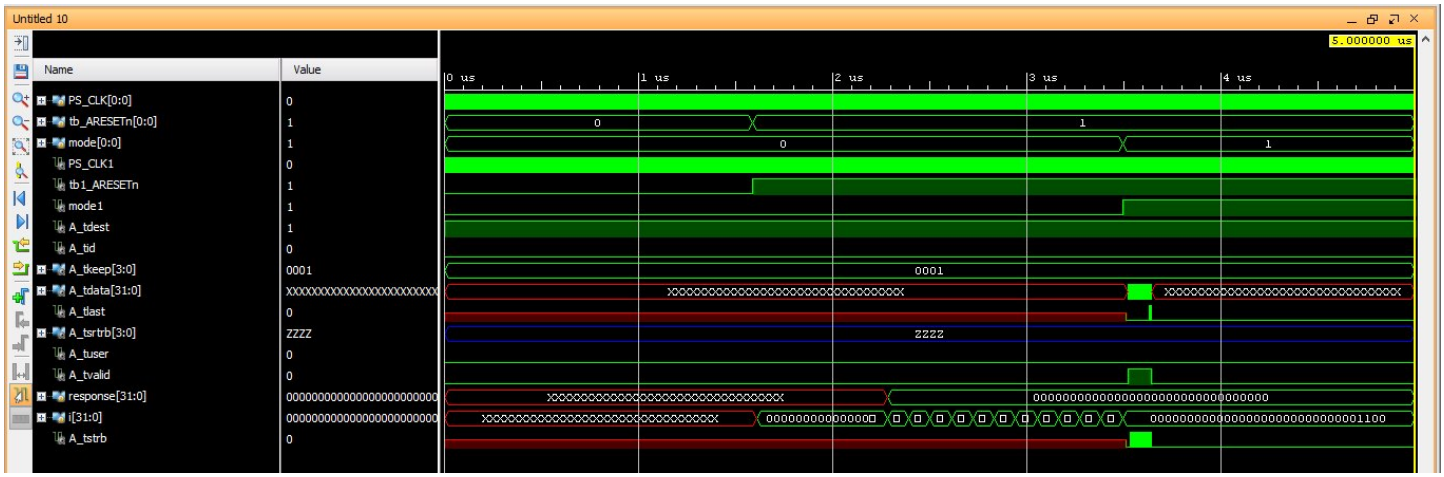
La fenêtre de simulation apparait :



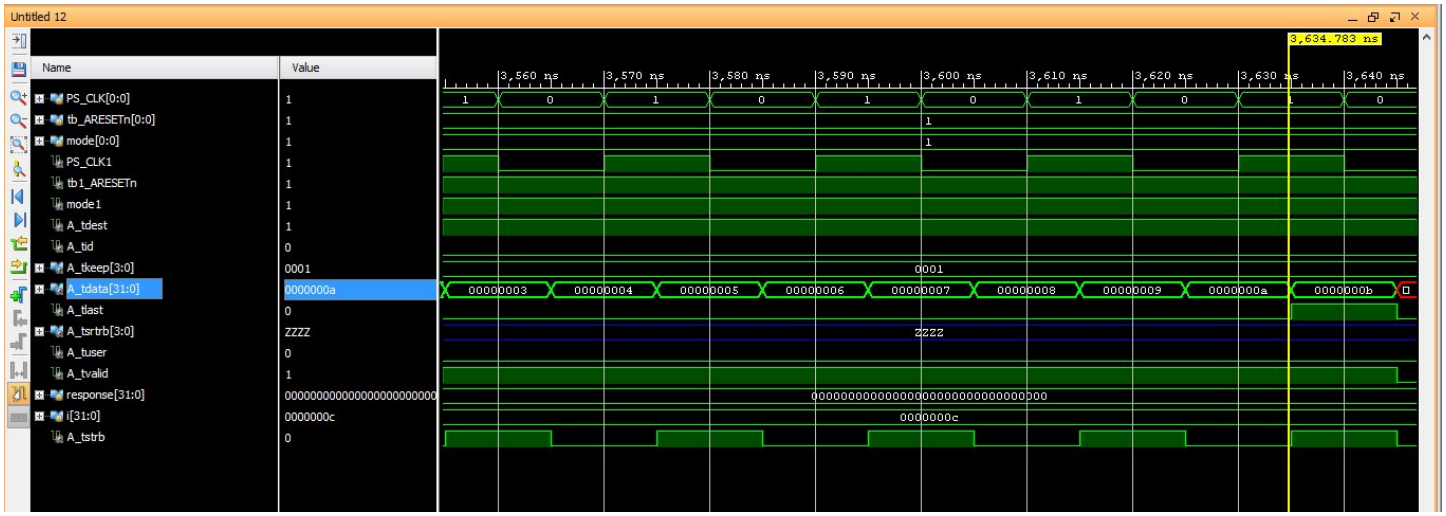
Pour agrandir la fenêtre cliquez sur :



En fin tout y est :



Le bus AXI4_STREAM en action :



Attention le modèle de simulation processing_system7_axi_periph sur la version 15_2 n'a pas l'air de fonctionner (à vérifier). Dans ce cas précis revenir à la version 15.1

Exemple de simulation pour un modèle MLI

HLS

```
#include "ap_axi_sdata.h"

//int bitcounteur=0;
//int bitcounteur=0;

void example( int memo[4], bool *LED, volatile int *outpudata, bool mode) {
#pragma HLS INTERFACE s_axilite depth=4 port=memo
#pragma HLS INTERFACE ap_none register port=LED
#pragma HLS INTERFACE ap_none register port=outpudata

//#pragma HLS INTERFACE s_axilite  depth=4 port=memo
#pragma HLS INTERFACE ap_none register port=mode

      int bitcounteur;
 //int bitcounteur=0;
//#pragma HLS INTERFACE s_axilite register port=memo metadata="-bus_bundle slv0"
//#pragma HLS ARRAY_PARTITION variable=memo complete dim=1
//#pragma HLS INTERFACE ap_fifo depth=32 port=memo

            //*outpudata=memo[1];
            while(mode==1){
                  *LED = 0;
                  volatile int i=0;
//# pragma HLS PIPELINE
                  for (i=0;i<memo[2];i++){
                  *outpudata=i;
            //*LED = 0;
            if (i>memo[1])*LED = 1;
            else *LED = 0;}
            }

}
```

Sous VIVADO BFM fichier de simulation verilog

```
`timescale 1ns / 1ps
`uselib lib=unisims_ver
module tb_stim;  // obligatoire

reg [0:0]PS_CLK;
reg  [0:0]tb_ARESETn;
reg  [0:0]mode;
wire PS_CLK1;
wire tb1_ARESETn;
wire OUT_MLI1;
//wire [5:0] ADDRW;
//wire [31:0] DATAAXI;
wire [31:0] data;
//wire readyAW;
wire mode1;
//wire [31:0] data;
//wire [31:0] data;
///wire data1;
reg[31 : 0] response;
integer i;

design_1_wrapper zynq_mli
       (
       //.ADDRW(ADDRW),
       //.DATAAXI(DATAAXI),
       //.readyAW(readyAW),
        .FIXED_IO_ddr_vrn(),
        .FIXED_IO_ddr_vrp(),
        .FIXED_IO_mio(),
        .FIXED_IO_ps_clk(PS_CLK1),
        .FIXED_IO_ps_porb(tb1_ARESETn),
        .FIXED_IO_ps_srstb(tb1_ARESETn),
        .data(data),
        .mode(mode1),
        .OUT_MLI(OUT_MLI));


 //------------------------------------------------------------------------
                  // Simple Clock Generator
                  //------------------------------------------------------------------------
----

                  initial begin
                    PS_CLK = 1'b0;
                    forever #10 PS_CLK = !PS_CLK;
                  end

                assign PS_CLK1 =PS_CLK;

               assign tb1_ARESETn = tb_ARESETn ;
               assign OUT_MLI1 = OUT_MLI ;
               // assign data1 = data ;
                 assign mode1 = mode ;
               // assign DATAAXI1 = DATAAXI ;
               //   assign ADDRW1 = ADDRW ;

            initial begin
          tb_ARESETn = 1'b0;
          mode = 1'b0;
               // Release the reset on the posedge of the clk.
```

```
                repeat(80)@(posedge PS_CLK);
                tb_ARESETn = 1'b1;
                @(posedge PS_CLK);
                //repeat(10)@(posedge PS_CLK);
                //Reset the PL
                zynq_mli.design_1_i.processing_system7_0.inst.fpga_soft_reset(32'h1);
                zynq_mli.design_1_i.processing_system7_0.inst.fpga_soft_reset(32'h0);
                //repeat(10)@(posedge PS_CLK);
                zynq_mli.design_1_i.processing_system7_0.inst.write_data(32'h43C00010,4,
32'h00000000, response);
                //repeat(80)@(posedge PS_CLK);
                 zynq_mli.design_1_i.processing_system7_0.inst.write_data(32'h43C00014,4,
32'h0000050, response);
               // repeat(10)@(posedge PS_CLK);
               zynq_mli.design_1_i.processing_system7_0.inst.write_data(32'h43C00018,4,
32'h000000100, response);
                //repeat(30)@(posedge PS_CLK);
              zynq_mli.design_1_i.processing_system7_0.inst.write_data(32'h43C00010,4,
32'h00000001, response);
            mode = 1'b1;
end

endmodule
```