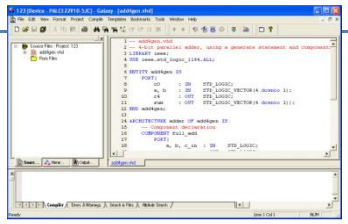
Le premier projet VHDL



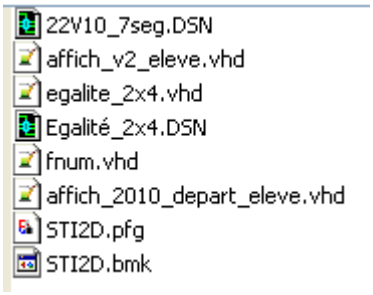
Ouvrir la machine virtuelle

- Ouvrir la machine virtuelle XP_PROG_ALL

Pour préparer le travail

- Recopier le dossier \Répertoire à envoyer sur les postes\VHDL_STI2D

  Et le mettre sur le bureau de la machine virtuelle

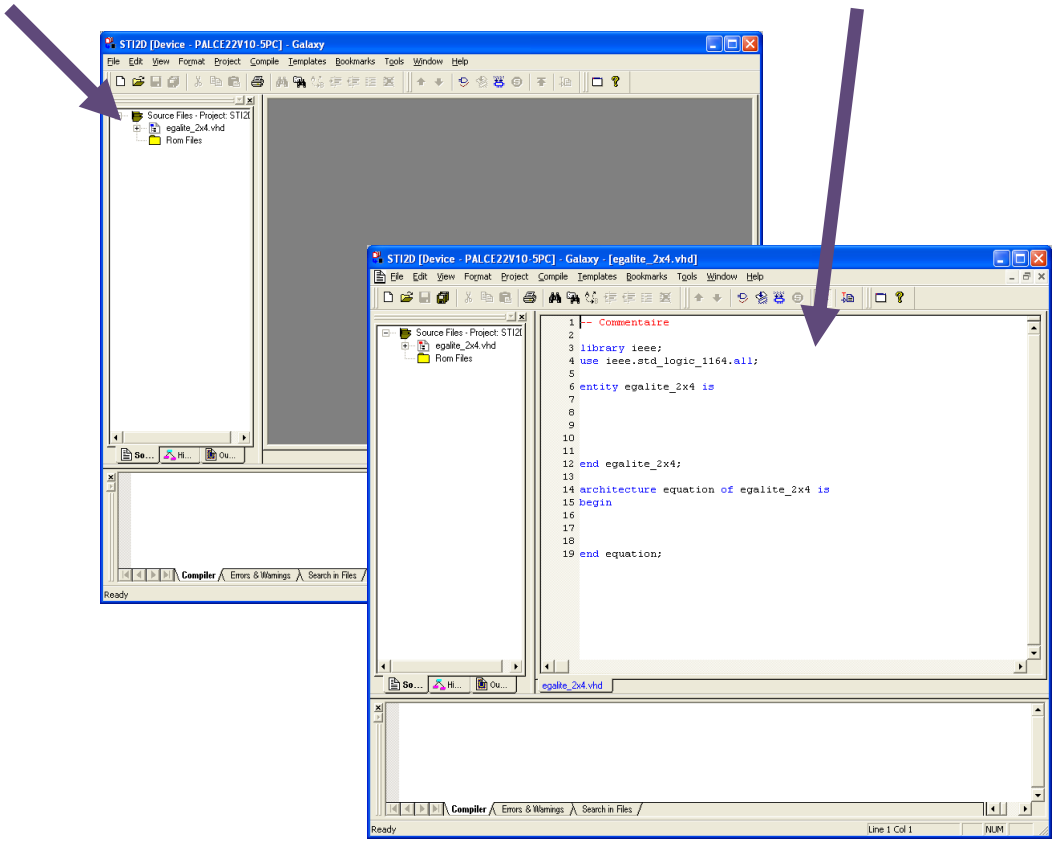


Pour lancer Warp

- Double cliquer sur le fichier de projet STI2D.pfg

  Le logiciel se lance

- Double cliquer sur le fichier egalite_2x4 pour ouvrir l'éditeur de texte

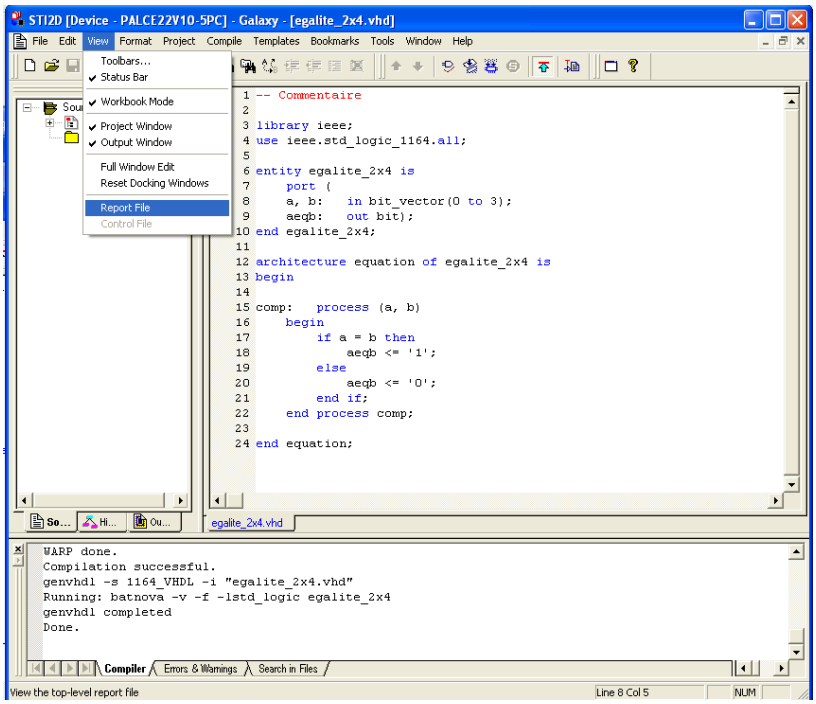


- Compléter ensuite le fichier comme ci-dessous :

```
 2
 3 library ieee;
 4 use ieee.std_logic_1164.all;
 5
 6 entity egalite_2x4 is
 7     port (
 8     a, b:    in bit_vector(0 to 3);
 9     aeqb:    out bit);
10 end egalite_2x4;
11
12 architecture equation of egalite_2x4 is
13 begin
14
15 comp:    process (a, b)
16     begin
17         if a = b then
18             aeqb <= '1';
19         else
20             aeqb <= '0';
21         end if;
22     end process comp;
23
24 end equation;
```

- Pour compiler appuyer sur [F7]

- Ensuite ouvrir le fichier de rapport réalisé par le compilateur



- Observer en particulier les équations trouvées automatiquement par le compilateur.

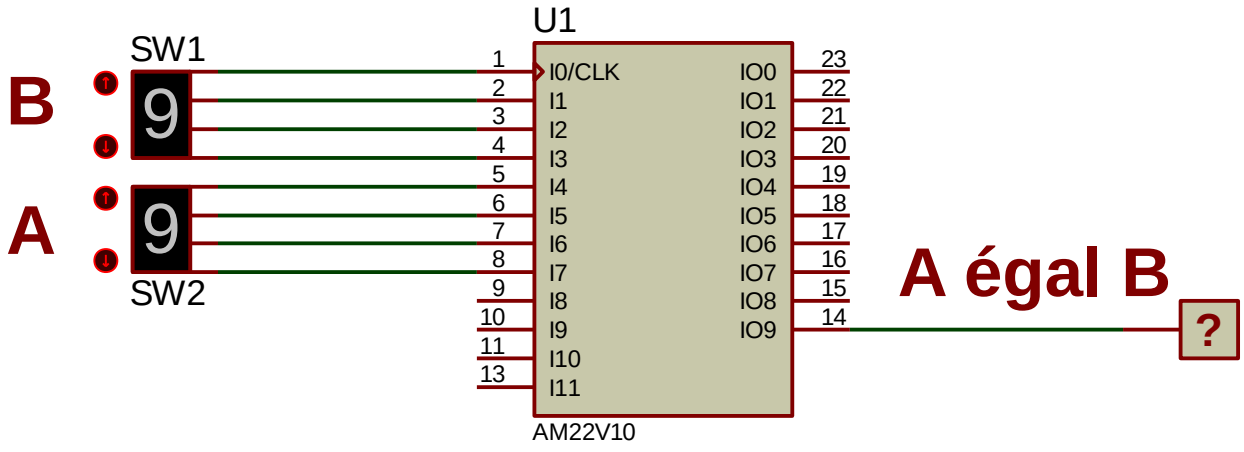
Le fichier rapport donne aussi le brochage proposé pour le projet :

```
PINOUT INFORMATION     (22:24:55)

Messages:
  Information: Checking for duplicate NODE logic.
               None.

                          C22V10

          b(3) =|  1|                          |24|* not used
          b(2) =|  2|                          |23|* not used
          b(1) =|  3|                          |22|* not used
          b(0) =|  4|                          |21|* not used
          a(3) =|  5|                          |20|* not used
          a(2) =|  6|                          |19|* not used
          a(1) =|  7|                          |18|* not used
          a(0) =|  8|                          |17|* not used
      not used *|  9|                          |16|* not used
      not used *| 10|                          |15|* not used
      not used *| 11|                          |14|= aeqb
      not used *| 12|                          |13|* not used
```

```
DESIGN EQUATIONS           (22:24:55)


    /aeqb =
          /a(0) * b(0)
        + /a(1) * b(1)
        + /a(2) * b(2)
        + /a(3) * b(3)
        + a(0) * /b(0)
        + a(1) * /b(1)
        + a(2) * /b(2)
        + a(3) * /b(3)


Completed Successfully
```

Pour simuler il faut modifier le fichier vhdl voir page 16 fiche 6 Flux de simulation Warp => ISIS

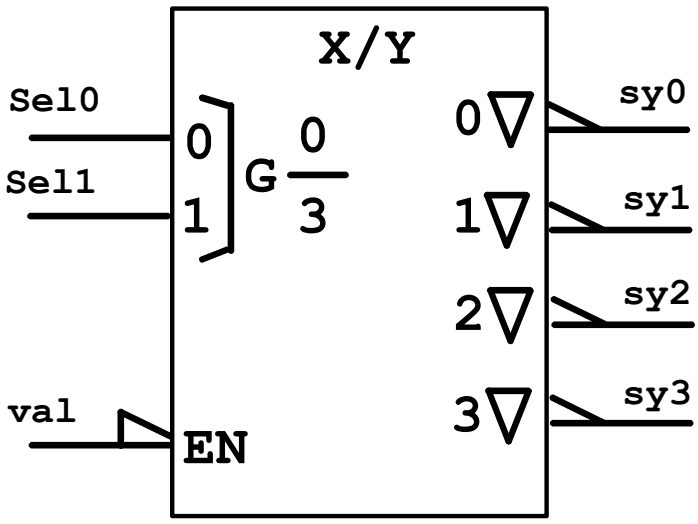
On pourra ensuite simuler avec le schéma ISIS ci-contre Egalité_2x4.DSN



Synthèse et simulation de décodeur en vhdl

Synthèse d'un décodeur 1 vers 4 avec sortie haute impédance

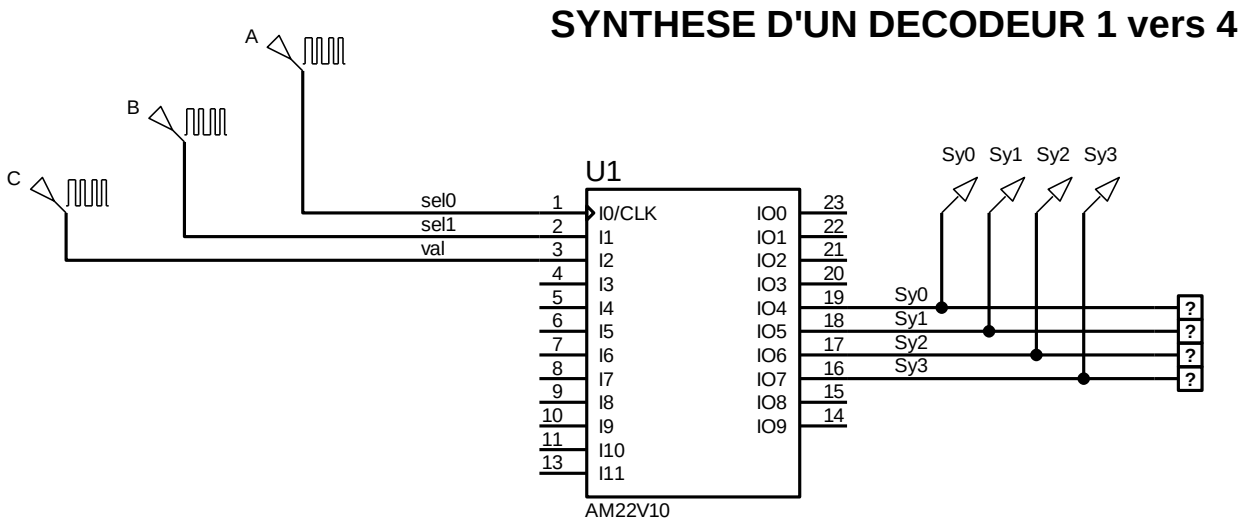
Le symbole :



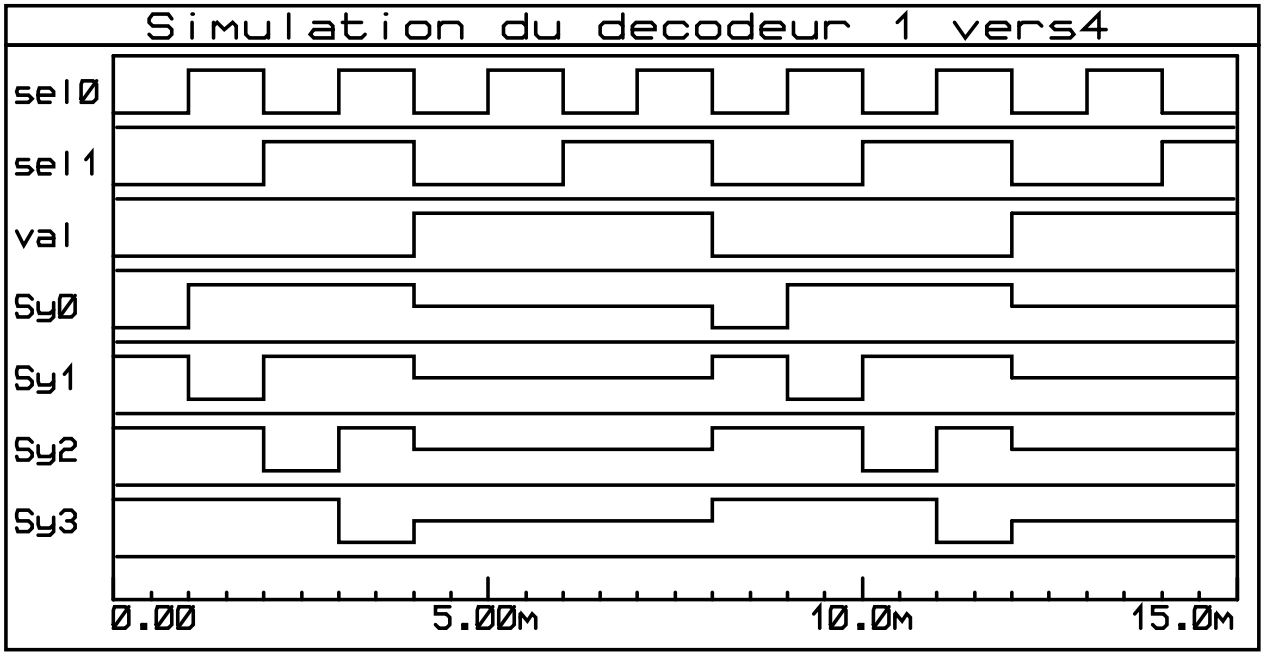
La description en vhdl :

```
 1 library ieee;
 2 use ieee.std_logic_1164.all;
 3
 4
 5 entity decodeur1 is
 6
 7     port ( sel0,sel1,val : in  std_logic;
 8            sy : out std_logic_vector(3 downto 0));
 9
10     attribute pin_numbers of decodeur1:entity is
11     "sel0:1 " & "sel1:2 " & "val:3 " &
12     "sy(0):19 " & "sy(1):18 " & "sy(2):17 " & "sy(3):16 ";
13
14
15 end decodeur1;
16
17
18 architecture box of decodeur1 is
19 begin
20
21
22 p1:process(sel0,sel1,val)
23 begin
24     if (val='1') then sy <= "ZZZZ";
25     elsif ( sel1='0' and sel0='0' ) then sy <= "1110";
26     elsif ( sel1='0' and sel0='1' ) then sy <= "1101";
27     elsif ( sel1='1' and sel0='0' ) then sy <= "1011";
28     elsif ( sel1='1' and sel0='1' ) then sy <= "0111";
29     else sy <= "1111";
30     end if;
31 end process p1;
32 end;
```

La simulation sous proteus :



Le chronogramme du fonctionnement :



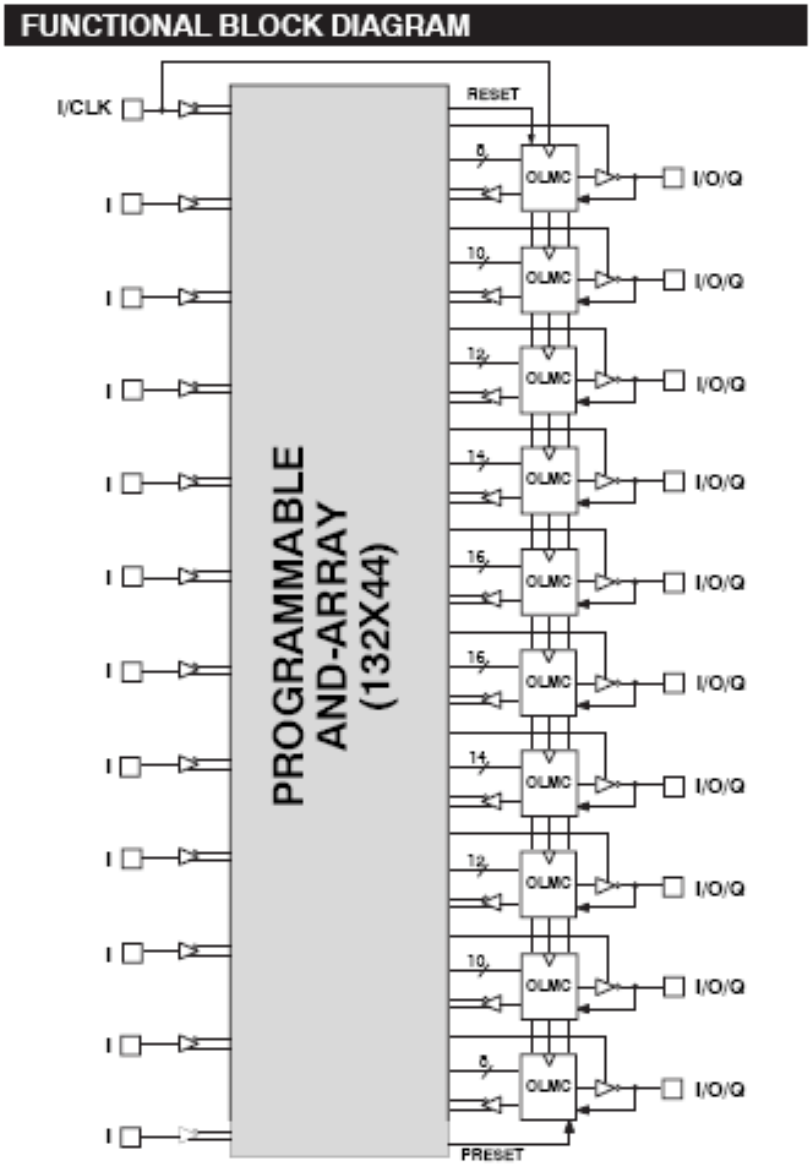
Travail à effectuer :

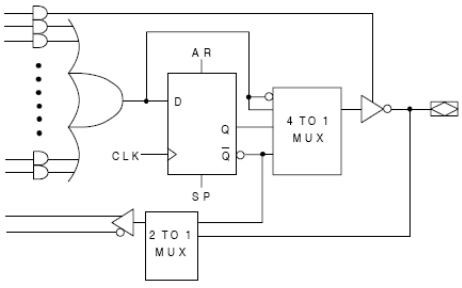
1. Faire toute la synthèse de ce décodeur.
2. Consigner les résultats dans un dossier en commentant les résultats obtenus.
3. Etendre cette synthèse à un décodeur de même type mais de 1 vers 8

Synthèse d'une séquence binaire pseudo-aléatoire

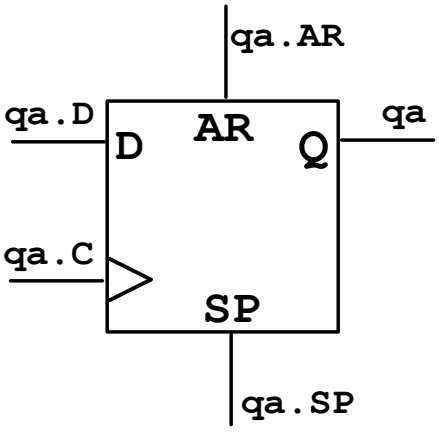
La logique séquentielle, description succincte du 22v10:

La description fonctionnelle indique les 10 sorties qui sont en fait des macros cellules configurables soient en fonctions combinatoires soient en fonctions séquentielles donc avec une bascule de type D, appelée aussi registre. Le panachage des fonctions est possible.





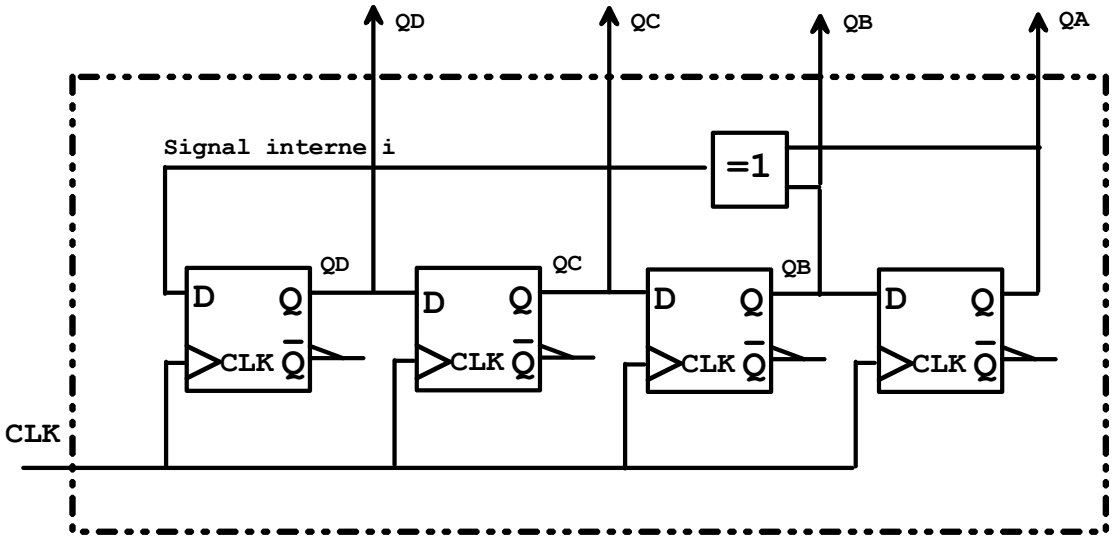
GAL22V10 OUTPUT LOGIC MACROCELL (OLMC)



Le synthétiseur de Warp traduit chaque macro cellule de la manière suivante dans le cas d'une configuration séquentielle, si qa est le nom de la broche de sortie alors :

- qa.D = l'équation d'entrée de la bascule
- qa.AR = reset asynchrone
- qa.SP = preset synchrone
- qa.C =l'entrée de l'horloge (broche 1 uniquement)

Le schéma à synthétiser :



Le code vhdl :

Le code complet conforme au schéma ci-dessus :

```
2  -- Séquence binaire pseudo aléatoire
3  -- sur 4 bits
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7
8  use work.portes.all;
9
10 entity sbpa4 is
11     port (
12         init : in std_logic;
13         clk : in std_logic;
14         qa, qb, qc, qd : inout std_logic
15         );
16 end sbpa4;
17
18
19
20 architecture composant of sbpa4 is
21
22 signal i : std_logic;
23
24 attribute synthesis_off of i : signal is true;
25
26 begin
27
28     i <= qa xor qb;
29     -- port map ( Entrée, Horloge, Reset, Preset, Sortie )
30     -- conformément à la description dans la bibliothèque
31     ic1 : dff2_t1 port map ( qb, clk, '0',  init, qa);
32     ic2 : dff2_t1 port map ( qc, clk, init, '0',  qb);
33     ic3 : dff2_t1 port map ( qd, clk, init, '0',  qc);
34     ic4 : dff2_t1 port map ( i, clk,  init, '0',  qd);
35
36 end composant;
```

Il faut utiliser une bibliothèque portes.vhd, c'est un fichier indépendant à décrire en vhdl.

Noter la déclaration du signal interne à l'architecture appelé i, à quoi sert-il ?

Le synthétiseur à obligation de le conserver à la fin de la synthèse, il sera donc visible sur une broche.

L'instruction **port map** donne l'interconnexion pour les signaux, à renseigner conformément au schéma soit dans l'ordre : (data, clock, reset, preset, sortie)

La bibliothèque portes :

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package portes is
5
6  component dff2_t1
7      port (
8          d      : in std_logic;
9          clk    : in std_logic;
10         reset  : in std_logic;
11         preset : in std_logic;
12         q      : out std_logic
13         );
14     end component;
15 end portes;
```

Package : déclaration des modules, c'est le catalogue du contenu de la bibliothèque. Pour l'instant il n'y a qu'un seul élément.

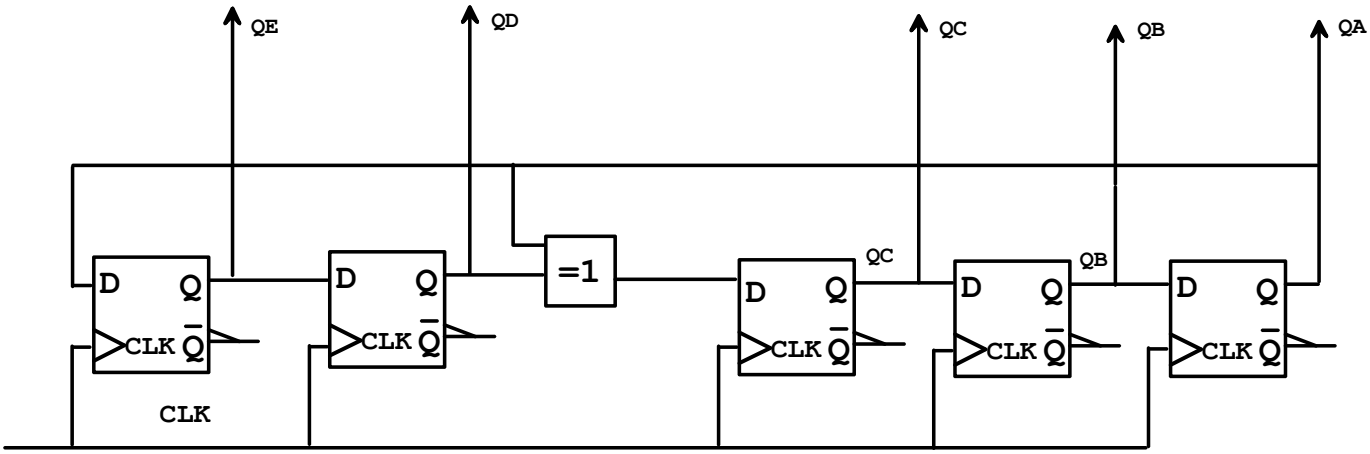
```
19 --bascule D active sur fronts montants
20 --avec reset et preset
21 library ieee;
22 use ieee.std_logic_1164.all;
23
24 entity dff2_t1 is
25 port (
26         d       : in std_logic;
27         clk     : in std_logic;
28         reset   : in std_logic;
29         preset  : in std_logic;
30         q       : out std_logic
31         );
32 end dff2_t1;
33
34
35 architecture archdff of dff2_t1 is
36 begin
37
38 p1 : process (clk,reset,preset)
39 begin
40     if (clk'event and clk='1')
41     then
42         if ( reset = '1' )
43             then q <= '0';
44         elsif ( preset = '1' )
45             then q<='1';
46         else q <= d;
47         end if;
48     end if;
49 end process p1;
50
51 end archdff;
```

Description complète du premier 'composant' de la bibliothèque la bascule D.

Travail à effectuer :

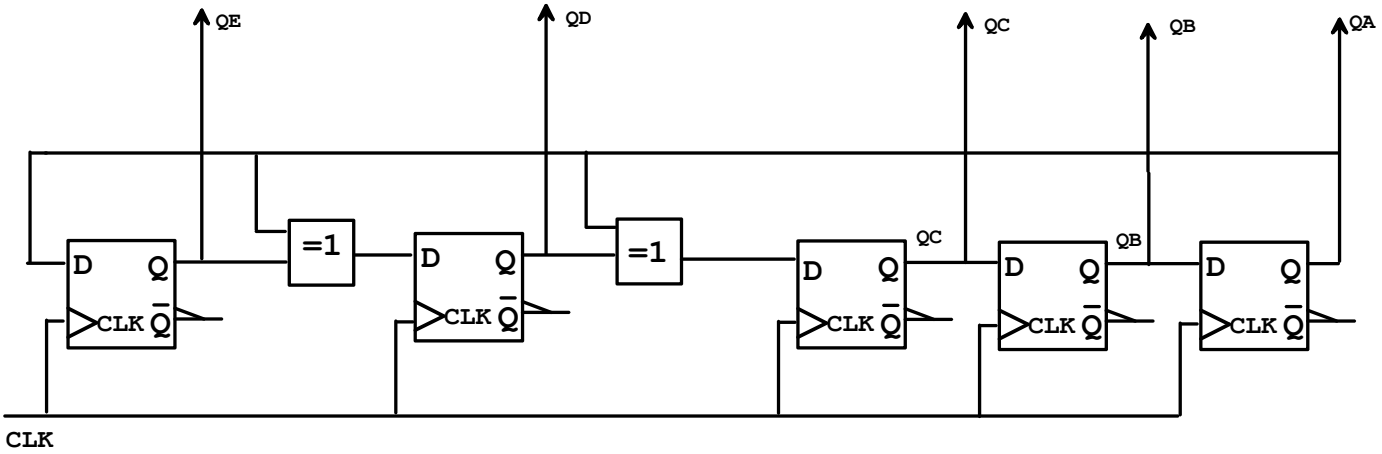
1. Faire toute la synthèse du générateur de SBPA, quelle est la valeur initiale obtenue avec init=1 ?
2. Déterminer la séquence obtenue avec Proteus, pourquoi appelle-t-on une telle séquence

   Séquence **B**inaire **P**seudo **A**léatoire ?
3. Faire la synthèse de l'exemple ci-dessous :



La valeur initiale est de 10000, la séquence est-elle de longueur maximale ?

4. Autre exemple :

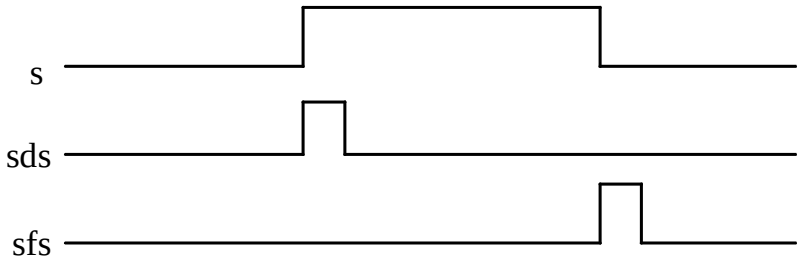


Noter ici la présence de deux signaux interne, on pourra prendre comme valeur initiale la valeur 10101.

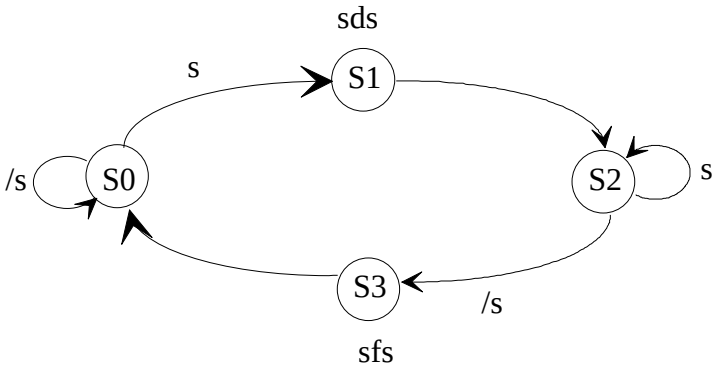
Automate séquentiel

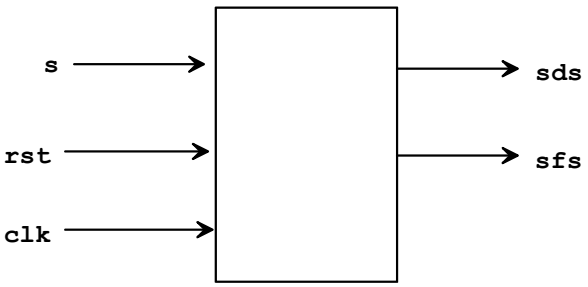
Description de l'automate par une machine de Moore

Réaliser un automate qui détecte le début et la fin d'une impulsion s de durée quelconque.



Le codage en vhdl






```
 6 library ieee;
 7 use ieee.std_logic_1164.all;
 8
 9 entity moore1 is port(
10     clk,rst : in std_logic;
11     s :       in std_logic;
12     sds,sfs : out std_logic);
13 end moore1;
14
15
16 library ieee;
17 use ieee.std_logic_1164.all;
18
19 architecture archmoore1 of moore1 is
20     type states is (s0,s1,s2,s3);
21     signal state : states:=s0;
22
23 begin
24
25     moore:process(rst,clk)
26     begin
27         if (rst='1') then
28             state <= s0;
29         elsif ((clk'event) and (clk='1')) then
30             case state is
31                 when s0 => if (s='1') then state <= s1;
32                                       else state <= s0; end if;
33                            when s1 => state <= s2;
34                 when s2 => if (s='0') then state <= s3;
35                                       else state <= s2; end if;
36                 when s3 => state <= s0;
37             end case;
38         end if;
39     end process;
40
41     sds <= '1' WHEN (state=s1) else '0';
42     sfs <= '1' WHEN (state=s3) else '0';
43
44 end archmoore1;
```

Encodage des états

Cela signifie donner un numéro binaire aux états pour pouvoir les identifier. Il y a plusieurs solutions exposées ci-dessous. Noter que ces numéros sont mémorisés donc utilise des registres, pour nous des sorties du 22V10.

Encodage personnalisé :

syntaxe : *attribute enum_encoding of type_name:type is "string"*

Choix complètement libre du code

```
21    type states is (s0,s1,s2,s3);
22
23    attribute enum_encoding of states:type is " 00 01 10 11 ";
24
25    signal state : states:=s0;
```

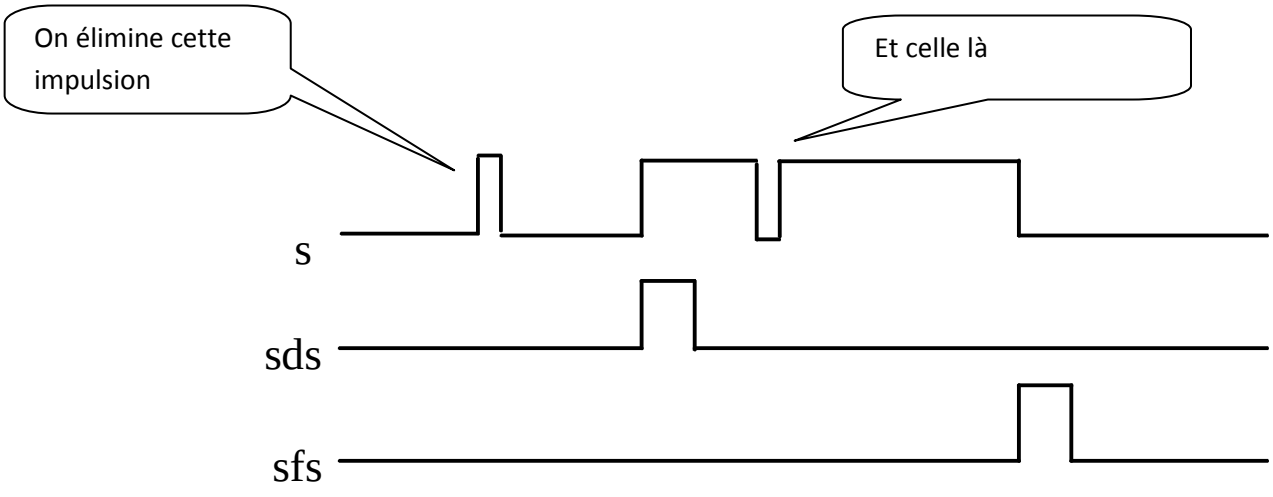
Encodage prédéfini : one_hot_zero, one_hot_one, gray, sequential

Syntaxe : *attribute state_encoding of type-name is value;*

```
21    type states is (s0,s1,s2,s3);
22
23    attribute state_encoding of states:type is gray;
24
25    signal state : states:=s0;
```

Travail à réaliser

1. Faire la synthèse et la simulation
2. Améliorer le dispositif précédent pour filtrer les bruits numériques, éliminer des impulsions parasites de durée inférieure ou égale à un coup d'horloge selon le chronogramme ci-dessous :



Il faut tout d'abord dessiner la machine d'états modifiée puis faire la synthèse.

Analyse d'une boite noire

Faire la synthèse et la simulation du code boite-noire et tout d'abord dessiner le bilan des entrées sorties à partir de l'entité.

```
 3 library ieee;
 4 use ieee.std_logic_1164.all;
 5
 6 library cypress;
 7 use cypress.std_arith.all;
 8
 9 entity fnum is
10     generic (N:integer:=6);
11     port (
12         clock,sig_in : in std_logic;
13         sig_out : out std_logic);
14 end fnum;
```

Puis faire la synthèse et simuler le résultat pour déterminer la fonction de cette boite, la description de l'architecture est donnée ci-dessous. (Vous n'avez rien à saisir, le code fhum.vhd vous est donné)

```
16 architecture arch_fnum of fnum is
17
18 signal compte: integer range 0 to N;
19 signal etat : std_logic;
20
21 begin
22
23 process (clock)
24 begin
25     if (clock'event and clock='1') then
26
27         if ((etat='0') and (compte<N))
28             then
29                 if (sig_in='0')
30                 then compte <= 0;
31                     else compte <= compte + 1;
32                 end if;
33             elsif ((etat='0') and (compte = N))
34                 then etat <= '1'; compte <= 0;
35         end if;
36
37         if ((etat='1') and (compte<N))
38             then
39                 if (sig_in='1')
40                     then compte <= 0;
41                     else compte <= compte + 1;
42                 end if;
43             elsif ((etat='1') and (compte = N))
44                 then etat <= '0'; compte <= 0;
45         end if;
46     end if;
47 end process;
48
49 sig_out <= etat;
50
51 end arch_fnum;
```

Synthèse de compteur

Comparer le fonctionnement des compteurs ci-dessous,

Les codes vhdl sont disponibles dans le dossier examples/counter de warp galaxy

```
 3 library ieee;
 4 use ieee.std_logic_1164.all;
 5 entity counter is port(
 6     clk, reset, load:   in std_logic;
 7     data:           in std_logic_vector(3 downto 0);
 8     count:          buffer std_logic_vector(3 downto 0));
 9 end counter;
10
11 use work.std_arith.all;
12
13 architecture archcounter of counter is
14 begin
15 upcount: process (clk)
16     begin
17         if (clk'event and clk= '1') then
18             if load = '1' then
19                 count <= data;
20             else
21                 count <= count + 1;
22             end if;
23         end if;
24     end process upcount;
25 end archcounter;
```

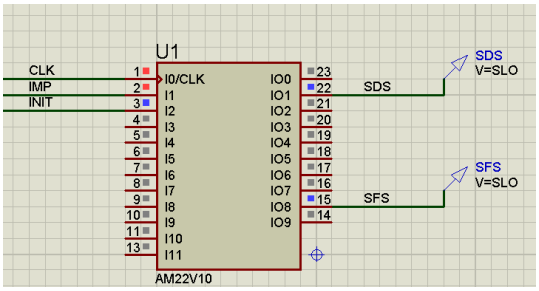
Ne pas oublier de faire tout d'abord le bilan des entrées sorties

```
 4 library ieee;
 5 use ieee.std_logic_1164.all;
 6 USE work.STD_ARITH.all;
 7
 8 entity counter is port(
 9     clk, rst, pst, load,counten:    in std_logic;
10     data:               in std_logic_vector(3 downto 0);
11     count:              buffer std_logic_vector(3 downto 0));
12 end counter;
13
14 architecture archcounter of counter is
15 begin
16 upcount: process (clk, rst, pst)
17     begin
18         if rst = '1' then
19             count <= "0000";
20         elsif pst = '1' then
21             count <= "1111";
22         elsif (clk'event and clk= '1') then
23             if load = '1' then
24                 count <= data;
25             elsif counten = '1' then
26                 count <= count + 1;
27             end if;
28         end if;
29     end process upcount;
30 end archcounter;
```

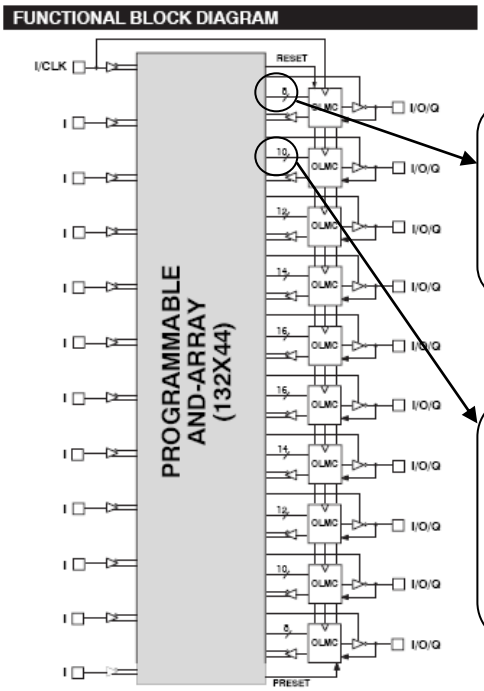
Visualisation des registres internes

Sur l'exemple ci-dessous les sorties broche 23 et 14 du Pal22V10 sont utilisés en interne par le synthétiseur vhdl et donc ne sont pas visibles sous proteus.

```
                        C22V10
                 ___________________________________
          clk =| 1|                                |24|* not used
            s =| 2|                                |23|= (stateSBV_0)
          rst =| 3|                                |22|= sds
    not used *| 4|                                |21|* not used
    not used *| 5|                                |20|* not used
    not used *| 6|                                |19|* not used
    not used *| 7|                                |18|* not used
    not used *| 8|                                |17|* not used
    not used *| 9|                                |16|* not used
    not used *|10|                                |15|= sfs
    not used *|11|                                |14|= (stateSBV_1)
    not used *|12|                                |13|* not used
                 -----------------------------------
```



Un examen du fichier de programmation au format jedec généré par warp permet de comprendre pourquoi :



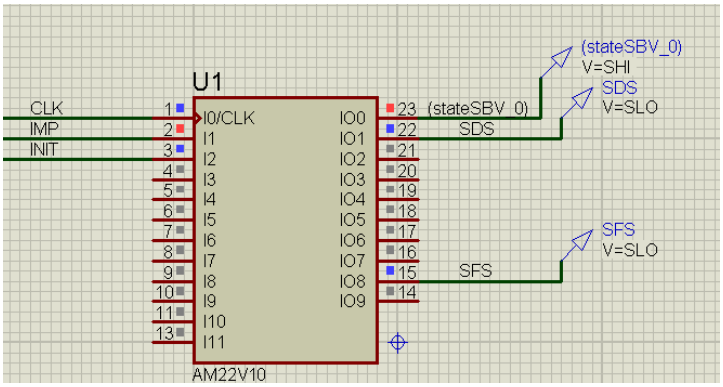

```
L0044
0000000000000000000000000000000000000000000000
0111111111111111111111111111111111111111101111
1011111111111111111111111111111111111111011111
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
Node stateSBV_0[23] => OE : 1 ,LOGIC : 8 *

L0440
1111111111111111111111111111111111111111111111
0111111111111111111111111111111111111101111111
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000
Node sds[22] => OE : 1 ,LOGIC : 10 *
```

Cette ligne de fusible n°22 commande la sortie haute impédance, ici elle est entièrement remplie de 0, tous les fusibles sont conservés le terme produit correspondant vaut donc 0, le

Ligne 31 : commentaire indiquant le numéro de la broche et le nom de la variable vhdl utilisée, ainsi que le nombre de termes produits

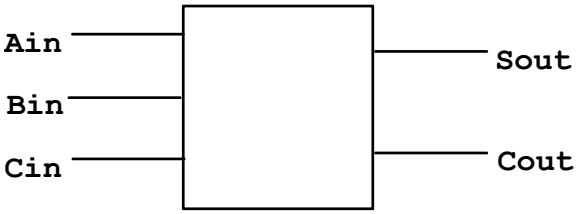
La modification de la ligne de fusible directement dans le fichier jedec généré par warp permet d'activer la sortie haute impédance. La broche 23 visualise donc l'état interne de state_BV_0. (Cette modification doit être réalisée après chaque compilation, avant le traitement par Galep32)

```
20
21 L00044
22 1111111111111111111111111111111111111111111111
23 1101111111111111111111111111111111111111101111
24 1110111111111111111111111111111111111111011111
25 0000000000000000000000000000000000000000000000
26 0000000000000000000000000000000000000000000000
27 0000000000000000000000000000000000000000000000
28 0000000000000000000000000000000000000000000000
29 0000000000000000000000000000000000000000000000
30 0000000000000000000000000000000000000000000000
31 * Node stateSBV_0[23] => OE : 1 ,LOGIC : 8 *
32
33 L00440
34 1111111111111111111111111111111111111111111111
35 1101111111111111111111111111111111111101111111
36 0000000000000000000000000000000000000000000000
37 0000000000000000000000000000000000000000000000
38 0000000000000000000000000000000000000000000000
39 0000000000000000000000000000000000000000000000
40 0000000000000000000000000000000000000000000000
41 0000000000000000000000000000000000000000000000
42 0000000000000000000000000000000000000000000000
43 0000000000000000000000000000000000000000000000
44 0000000000000000000000000000000000000000000000
45 * Node sds[22] => OE : 1 ,LOGIC : 10 *
```



Additionneur

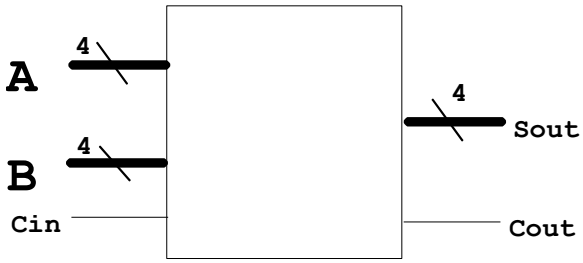
Etude de l'additionneur élémentaire



1. Déterminer les équations d'un additionneur de deux bits.
2. Réaliser la synthèse en vhdl. Simuler.
3. Intégrer cet additionneur dans votre bibliothèque porte.

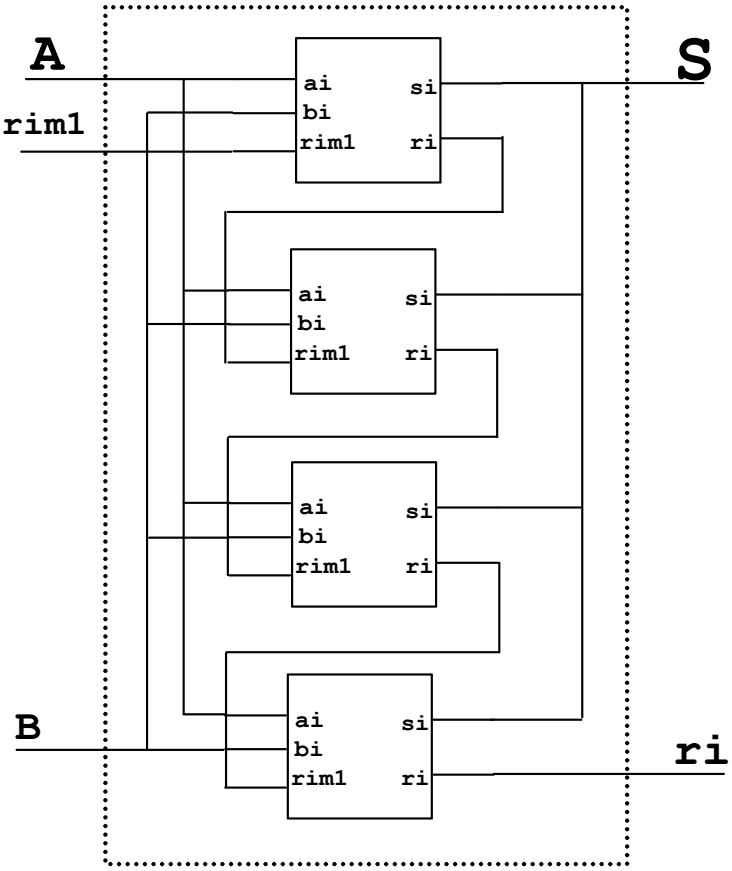
Synthèse d'un bloc d'addition

Utiliser l'élément d'addition élémentaire pour synthétiser un additionneur de deux mots de quatre bits.



La méthode de synthèse est identique au chapitre SBPA. Il faut :

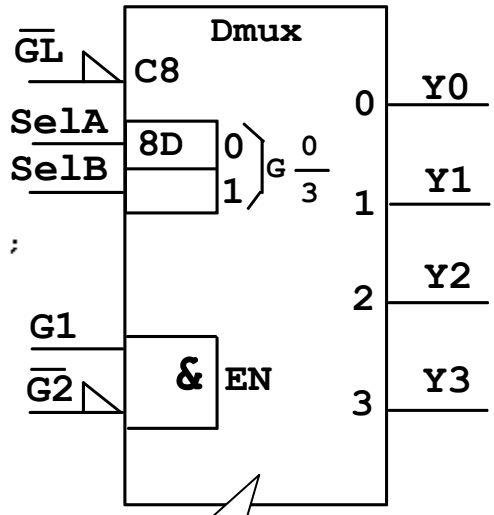
1. Faire appel à votre bibliothèque porte
2. Interconnecter les différents additionneurs selon le schéma ci-dessous
3. Compiler puis faire la synthèse, (l'utilisation de l'attribut Synthesis_off = true est indispensable pourquoi ?)



Synthèse d'un décodeur avec latch

Vérifier par la simulation que le code ci-dessous correspond bien au fonctionnement de ce décodeur.

```
 3 library ieee;
 4 use ieee.std_logic_1164.all;
 5
 6
 7 entity decod_latch is
 8
 9     port ( selA,selB,G1,G2b,GLb : in  std_logic;
10            sy : out std_logic_vector(3 downto 0));
11
12 end decod_latch;
13
```



Il faut expliquer ce symbole avant toute chose !

```
15 architecture box of decod_latch is
16
17 signal adresse : std_logic_vector(1 downto 0);
18
19 begin
20
21
22 p1:process(selA,selB,GLb)
23 begin
24     if (  GLb= '0' )
25     then
26         adresse <= selB & selA;
27     end if;
28 end process p1;
29
30
31 p2:process(adresse,G1,G2b)
32 begin
33     if ( G1='1' and  G2b='0' )
34     then
35         if     ( adresse = "00" ) then sy <= "0001";
36         elsif ( adresse = "01" ) then sy <= "0010";
37         elsif ( adresse = "10" ) then sy <= "0100";
38         else  sy <= "1000";
39         end if;
40     else
41         sy <= "0000";
42     end if;
43 end process p2;
44
45
46
47 end;
```

Modifier le code ci-dessus pour faire un décodeur identique mais avec la fonction haute impédance en plus, gérée avec une entrée supplémentaire de validation.

1 Redessiner le symbole complet avec l'entrée supplémentaire et les sorties avec la haute impédance.

2 Modifier le code vhdl

3 Simuler le nouveau décodeur