HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Reconfigurable & embedded
Digital Systems

Rapport Laboratoire Combinatoire hiérarchique : Calculateur de fonctions

Départements : TIC

Unité d'enseignement CSN

Auteurs : Pillonel Bastien & Brasey Loïc

Professeur : Etienne Messerli
Assistant : Anthony Jaccard Illan

Salle de labo : **A09**

Date : 29.03.2023

Introduction:

Description:

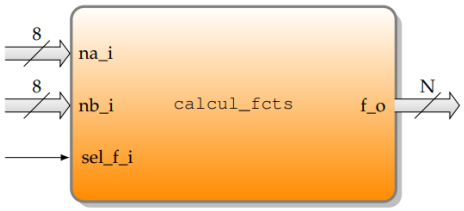
Le but de ce laboratoire est de créer un composant capable de réaliser les deux fonctions suivantes:

1. Si sel_f_i = ’0’ : f = na + 2 · nb − 84
2. Si sel_f_i = ’1’ : f = 3 · na − nb + 42

Comme indiqué plus haut notre composant possède une entrée de séléction (sel_f_i) permettant de choisir quelle fonction est réalisé par notre composant.

Rappel des contraintes:

“Vous n’osez pas utilisez de multiplicateur pour les multiplications et vous devez également optimiser la quantité de logique utilisée par votre solution. Ce critère doit guider vos choix lors de la décomposition du circuit. Les fonctions ont été choisies afin de permettre des optimisations.”



Nom	Dir.	Taille	Description
na_i	in	8	Valeur d’entrée signée na
nb_i	in	8	Valeur d’entrée signée nb
sel_f_i	in	1	Commande de sélection du calcul 1. ou 2.
f_o	out	N	Résultat signé du calcul, sur N bits (N à déterminer)

Conception:

Etape 1:

Ici il est demandé de trouver le nombre de bits nécessaire pour représenter la sortie f_o. Pour ce faire on peut calculer les valeurs min et max de chaque fonction:

	Min	Max
Fonction 1	-128 + 2 * -128 – 84 = -468	127 + 2 * 127 − 84 = 297
Fonction 2	3 * -128 − 127 + 42 = -469	3 * 127 − -128 + 42 = 551

On trouve donc qu’il nous faut **11 bits** afin de représenter le résultat signé max de la fonction 2.

Etape 2:

Après analyse des deux fonctions, nous avons découvert qu'il est possible de factoriser certaines opérations des deux fonction.

En effet l'operation 2*nb-84, de la fonction 1, peut aussi s'écrire 2*(nb-42). Conincidence dans la fonction 2 on retrouve -nb + 42 (même opération que pour f1 sans la multiplication par 2 et avec signe inversé). Il serait donc possible d'avoir un bloc qui s'occupe de cette opération.

On peut donc déjà séparer notre composant en trois blocs:

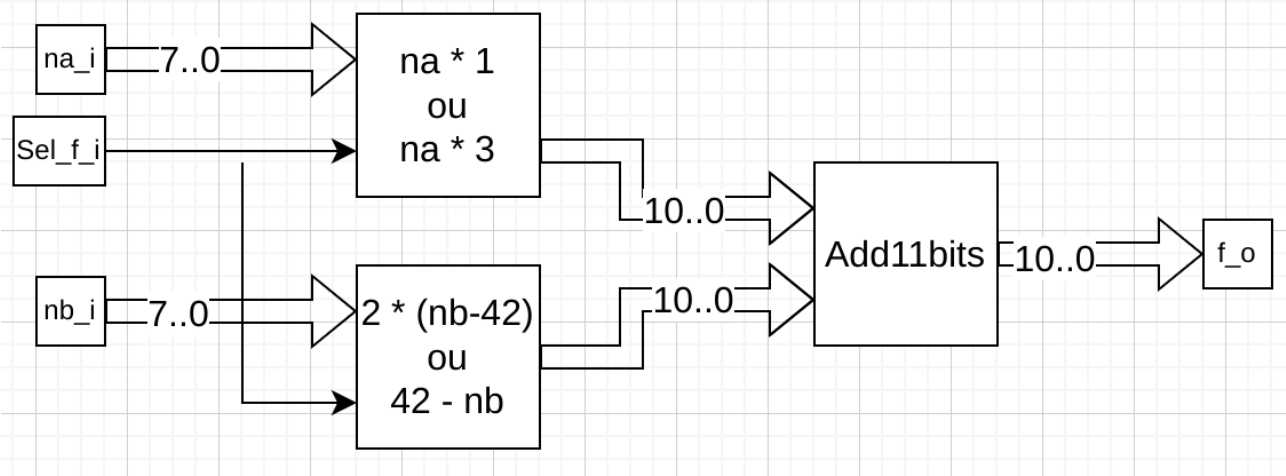
- Le premier s'occupe de sélectionner na ou 3*na en fonction de sel_f_i.

- Le second permet de réaliser l'opération décrite au dernier paragraphe en fonction de sel_f_i.

- Le dernier fait l'addition des deux sortie des deux premiers blocs

Etape 3:

Schéma bloc grossier:



Bloc calcul de la première opérande:

Ici le but est de réaliser la fonction "na*1 ou na*3" en fonction de l'entrée de sélection sel_f_i.
Décomposition du calcul:

- Division de na par 2

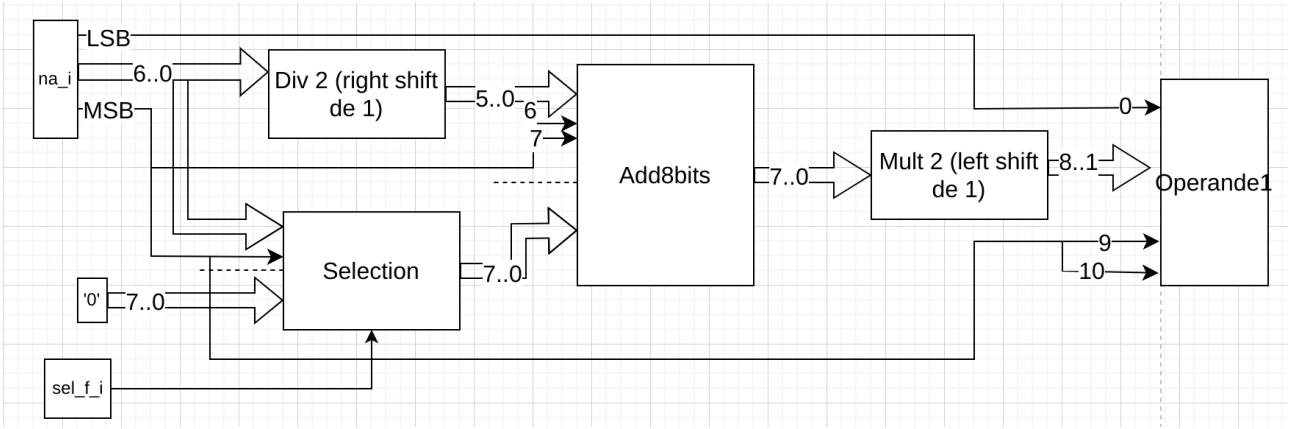
- Addition de na/2 avec :

    => 0 si sel_f_i vaut 0

    => na si sel_f_i vaut 1

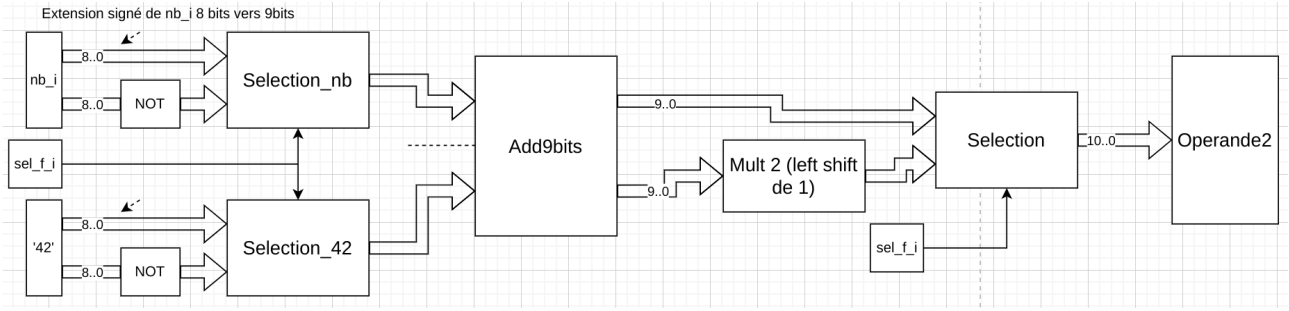
- Multiplication du resultat de l'addition par 2

Notre solution permet d'économiser 1 bit dans l'addition (donc economie de logiques) car le diviseur et multiplicateur par deux se réalise simplement en décalant les bits dans un sens ou l'autre alors que le rajout d'un bit dans l'additionneur ajoute des portes logiques.



Bloc calcul de la seconde opérande:

Ici le but est de réaliser la fonction “2 * (nb – 42) ou 42 – nb” en fonction de sel_f_i.



Etape 4:

Schéma détaillé calcul opérande 1:

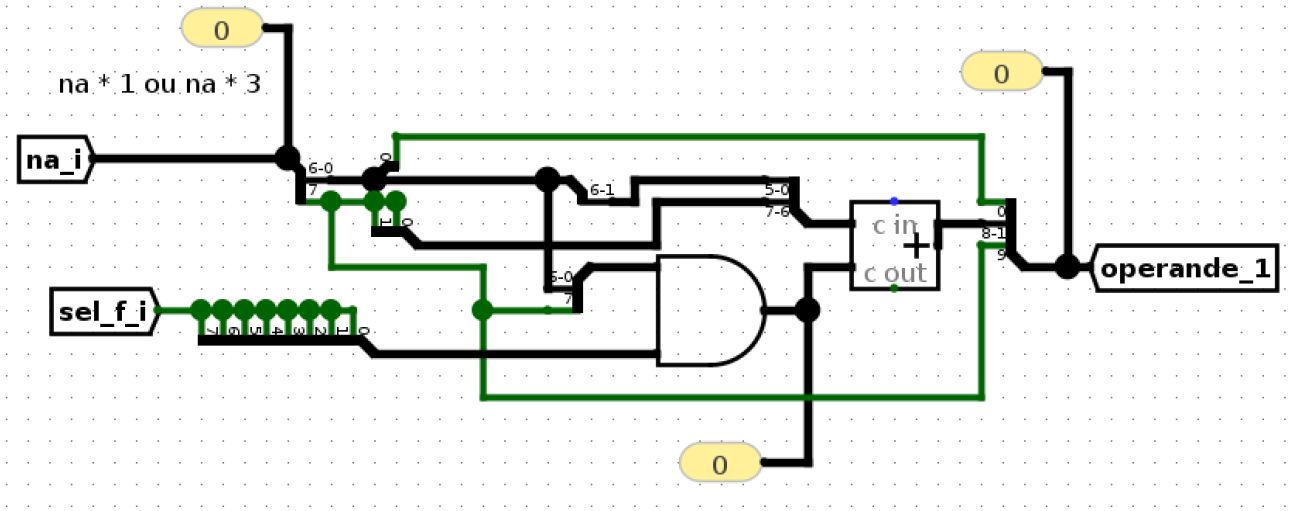


Schéma détaillé calcul opérande 2:

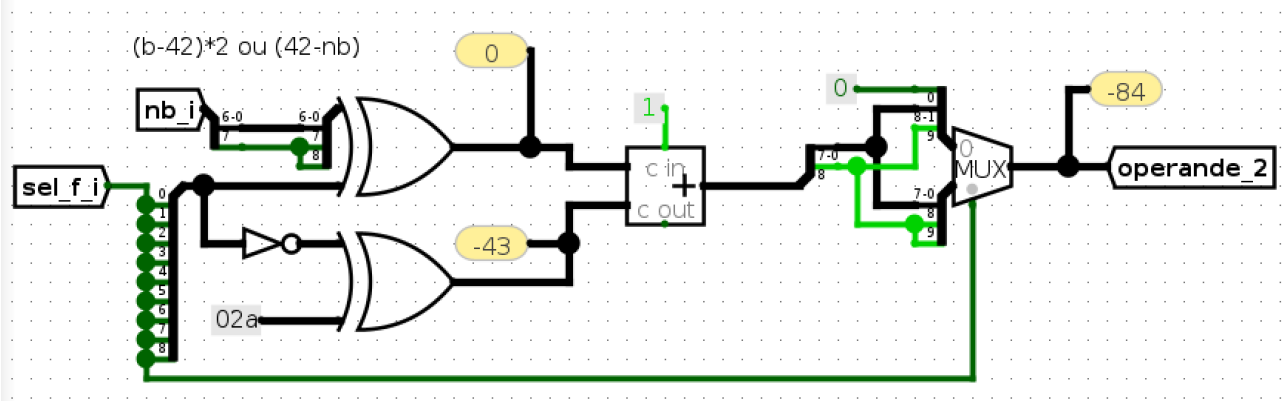
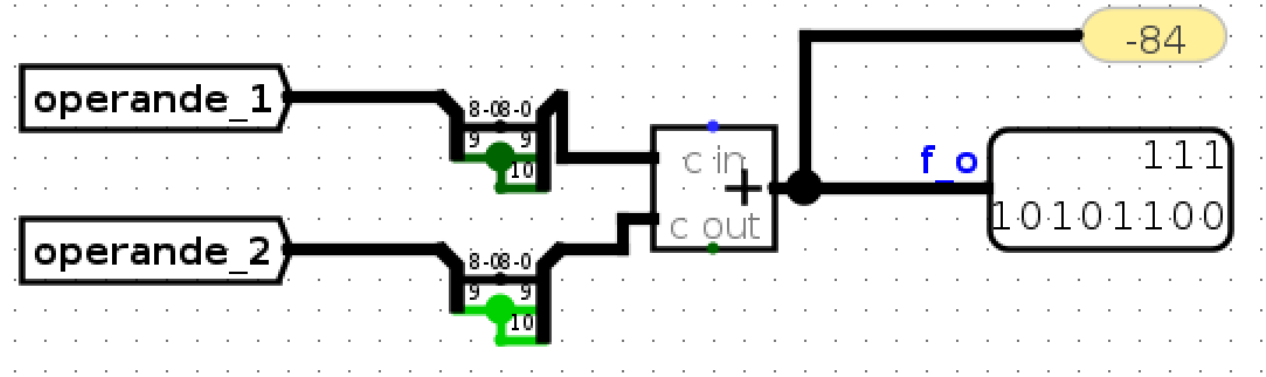


Schéma détaillé étage de sortie (add11bits):



Etape 5:

Table de vérité du bloc opérande 1:

Bloc opérande1

Input		Output				
na_i[7..0]	sel_f_i	Div2_o	Selection_o	Add8bits_o	Mult2_o	Operande1_o
"XXXX XXXX"	0	na_i'high & na_i'high & na_i[6..1]	"0000 0000"	Div2_o + Selection_o	na_i'high & Add8bits	Mult2_o'high & Mult2_o[9..0]
"XXXX XXXX"	1	na_i'high & na_i'high & na_i[6..1]	na_i[7..0]	Div2_o + Selection_o	na_i'high & Add8bits	Mult2_o'high & Mult2_o[9..0]

Table de vérité du bloc opérande 2:

Bloc opérande2

Input		Output					
nb_i[7..0]	sel_f_i	Selection_nb_o	Selection_42_o	Add9bits_o	Mult2_o	Selection	Operande2_o
"XXXX XXXX"	0	nb_i'high & nb_i[7..0]	Not("0 0010 1010")	Selection_nb_o + Selection_42_o	Add9bits_o[8..0] & '0'	Mult2_o[9..0]	Selection'high & Selection[9..0]
"XXXX XXXX"	1	not(nb_i'high & nb_i[7..0])	"0 0010 1010"	Selection_nb_o + Selection_42_o	Add9bits_o[8..0] & '0'	Add9bits'high & Add9bits[8..0]	Selection'high & Selection[9..0]

Nous n’avons pas réaliser la table de vérité du bloc d’étage final vu qu’il s’agit d’un simple additionneur 11bits.

Etape 6:

Notre description vhdl comprend deux fichier. Le premier, addn.vhd qui permet d’instancier un additionneur générique n bits repris du labo précédant. Le second, calcul_fcts_top.vhd décrit le composant réalisant les 2 fonctions décrites dans la donnée en réutilisant des additionneurs.

Fichier addn.vhd:

```
--------------------------------------------------------------------------------
-- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
-- Institut REDS, Reconfigurable & Embedded Digital Systems
--
-- File         : addn.vhd
--
-- Description  : Additionneur n bits avec carry in & carry out
--
-- Author       : Brasey Loic et Bastien Pillonel
-- Date         : 06.02.2023
-- Version      : 1.0
--------------------------------------------------------------------------------

--| Library |-----------------------------------------------------------------
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.numeric_std.all;
--------------------------------------------------------------------------------

--| Entity |------------------------------------------------------------------
entity addn is
  generic( N : positive range 1 to 31 := 26);
  port (nbr_a_i          : in  std_logic_Vector(N-1 downto 0);
        nbr_b_i          : in  std_logic_Vector(N-1 downto 0);
        cin_i            : in  std_logic;
        somme_o          : out std_logic_Vector(N-1 downto 0);
        cout_o           : out std_Logic
       );
end addn;
--------------------------------------------------------------------------------

--| Architecture |---------------------------------------------------------
architecture comportement of addn is

  --| Signals |-------------------------------------------------------------
  signal nbr_a_s : unsigned(N downto 0);
```

```
  signal nbr_b_s   : unsigned(N downto 0);
  signal somme_s : unsigned(N downto 0);
  signal cin_s       : unsigned(0 downto 0);
  ------------------------------------------------------------------------

begin

  -- Assignation of intern signals
  nbr_a_s       <= "0" & unsigned(nbr_a_i);
  nbr_b_s       <= "0" & unsigned(nbr_b_i);
  cin_s(0)      <= cin_i;
  somme_s     <= nbr_a_s + nbr_b_s + cin_s;

  -- Output
  somme_o     <= std_logic_Vector(somme_s(N-1 downto 0));
  cout_o        <= somme_s(N);

end comportement;
--------------------------------------------------------------------------------
```

Fichier calcul_fcts.vhd:

```
--------------------------------------------------------------------------------
-- HEIG-VD, Haute Ecole d'Ingenierie et de Gestion du canton de Vaud
-- Institut REDS, Reconfigurable & Embedded Digital Systems
--
-- File        : calcul_fcts_top.vhd
--
-- Description  : Description d'un system avec deux entrées na et na ainsi
--                que une entrée sel. Lorque sel et à 1: f = na*3-nb+42 et
--                f = na-84+2*nb lorsque sel = 0.
--
-- Author      : Brasey Loic et Bastien Pillonel
-- Date        : 06.02.2023
-- Version     : 1.0
--------------------------------------------------------------------------------

--| Library |-----------------------------------------------------------------
library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;
--------------------------------------------------------------------------------

--| Entity |------------------------------------------------------------------
entity calcul_fcts_top is
   port(
```

```
                na_i  : in  std_logic_vector(7 downto 0);
                nb_i  : in  std_logic_vector(7 downto 0);
                sel_i : in  std_logic;
                f_o   : out std_logic_vector(10 downto 0)-- A définir
    );
end calcul_fcts_top;
--------------------------------------------------------------------------------

--| Architecture |--------------------------------------------------------------
architecture struct of calcul_fcts_top is

    --| Signals |-----------------------------------------------------------
    signal na_sur_2_s           : std_logic_vector(7 downto 0);
    signal na_and_sel_s         : std_logic_vector(7 downto 0);
    signal nb_s                 : std_logic_vector(8 downto 0);
    signal n42_s                : std_logic_vector(8 downto 0);
    signal rslt_add9_s          : std_logic_vector(8 downto 0);
    signal operande_1_s         : std_logic_vector(10 downto 0);
    signal operande_2_s         : std_logic_vector(10 downto 0);
    signal f_s                  : std_logic_vector(10 downto 0);

    ------------------------------------------------------------------------

    --| Components |----------------------------------------------------------
    component addn is
       generic( N : positive range 1 to 31 := 26);
       port (nbr_a_i      : in  std_logic_Vector(N-1 downto 0);
             nbr_b_i      : in  std_logic_Vector(N-1 downto 0);
             cin_i        : in  std_logic;
             somme_o      : out std_logic_Vector(N-1 downto 0);
             cout_o       : out std_Logic
       );
    end component;
    for all : addn use entity work.addn(comportement);
    ----    ------------------------------------------------------------------

begin

    -- affectation des signeaux du premier bloc additionneur
    na_sur_2_s(7 downto 6)    <=  (others => na_i(7));
    na_sur_2_s(5 downto 0)    <=  na_i(6 downto 1);
    na_and_sel_s              <= na_i when sel_i = '1' else (others => '0');

    -- instanciation de l'additionneur du premier bloc
    add8: addn
    generic map ( N => 8 )
```

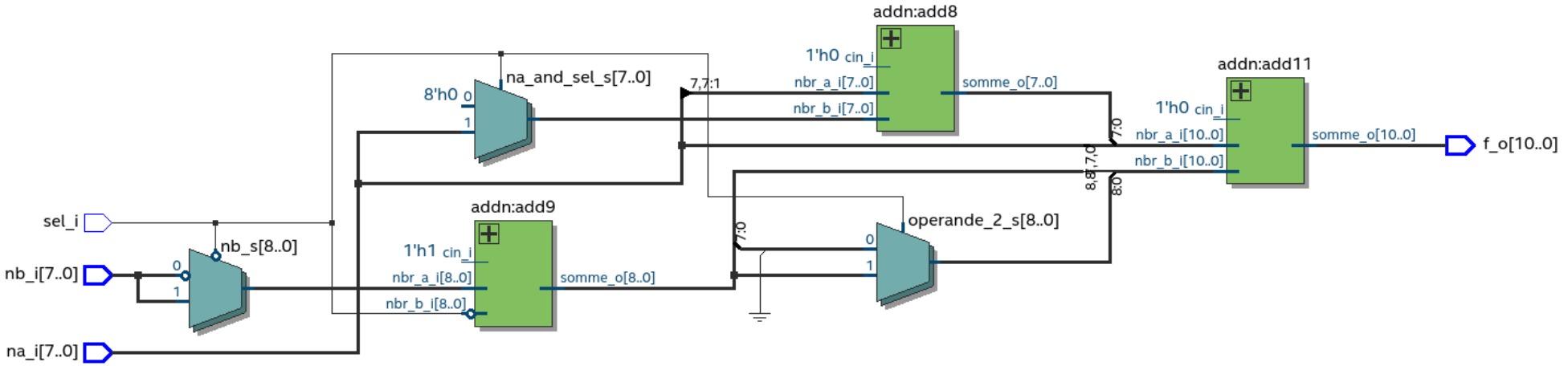
```
    port map(nbr_a_i   => na_sur_2_s,
             nbr_b_i   => na_and_sel_s,
             cin_i     => '0',
             somme_o   => operande_1_s(8 downto 1)
             -- cout_o
             );

    -- multiplication par 2 et extension du signe
    operande_1_s(0)                 <= na_i(0);
    operande_1_s(10 downto 9)       <= (others => na_i(na_i'high));

    -- assignation des signaux du deuxième bloc (inverser un des deux en fct de sel)
    nb_s          <= nb_i(7) & nb_i when sel_i = '0' else not(nb_i(7) & nb_i);
    n42_s         <= '0' & x"2a" when sel_i = '1' else not( '0' & x"2a");

    -- instanciation du deuxieme additionneur
    add9: addn
    generic map ( N => 9 )
    port map(nbr_a_i   => nb_s,
             nbr_b_i         => n42_s,
             cin_i           => '1',
             somme_o         => rslt_add9_s
             -- cout_o
             );

    -- multiplication du resulat par 2 en fonction de sel et extension de signe
    operande_2_s(9 downto 0) <= rslt_add9_s(8) & rslt_add9_s when sel_i ='1' else rslt_add9_s &
'0';
    operande_2_s(10) <= operande_2_s(9);

    -- instanciation du dernier additionneur
    add11: addn
    generic map ( N => 11 )
    port map(nbr_a_i   => operande_1_s,
             nbr_b_i         => operande_2_s,
             cin_i           => '0',
             somme_o         => f_s
             -- cout_o
             );

    -- sortie
    f_o <= f_s;

end struct;
--------------------------------------------------------------------------------
```

Etape 7:

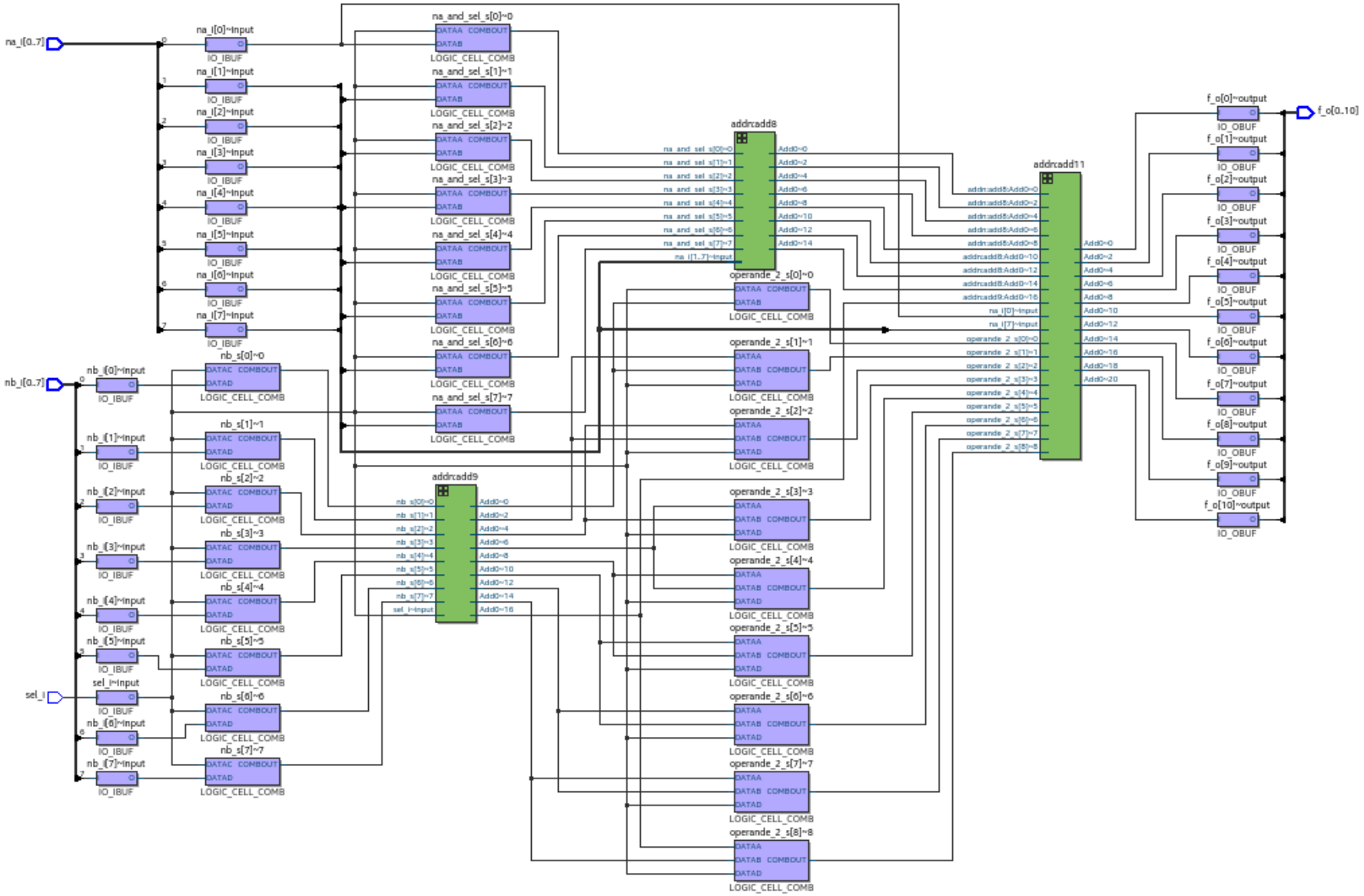
Après synthèse de du composant nous nous retrouvons avec un total de 53 logic éléments.

Nous avons pu comparer avec d'autres groupe et avons constater que le nombre de logique est plutôt proche des leurs (diff d'environs 5 à 10 éléments). Cependant nous nous trouvons tout de même plus haut qu'eux. Nous pensons qu'avec une décomposition un peu différente il serait possible d'avoir des additionneur moins grand et donc moins de logique.

Vue RTL:


addn:add8
1'h0 cin_i
nbr_a_i[7..0]
nbr_b_i[7..0]
somme_o[7..0]
addn:add11
1'h0 cin_i
nbr_a_i[10..0]
nbr_b_i[10..0]
somme_o[10..0]
f_o[10..0]
8'h0
na_and_sel_s[7..0]
7,7:1
addn:add9
1'h1 cin_i
nbr_a_i[8..0]
nbr_b_i[8..0]
somme_o[8..0]
operande_2_s[8..0]
sel_i
nb_s[8..0]
nb_i[7..0]
na_i[7..0]
7:0
8:0

Vue Technology:



Après analyse des vue on retrouve bien nos 3 blocs additionneurs 8,9 et 11 bits (je ne les ai pas détaillé étant donné que nous avons simplement repris le composant générique du labo précédent). Ainsi que nos 3 selecteurs qui correspondent au reste de la logique présente dans les vues.

Etape 9:

Logs de la simulation:

```
# vsim -voptargs=""+acc"" work.calcul_fcts_tb
# Start time: 17:32:10 on Mar 27,2023
# ** Note: (vsim-3812) Design is being optimized...
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading ieee.numeric_std(body)
# Loading work.calcul_fcts_tb(test_bench)#1
# Loading work.calcul_fcts_top(struct)#1
# Loading work.addn(comportement)#1
# Loading work.addn(comportement)#2
# Loading work.addn(comportement)#3
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ns  Iteration: 0  Instance: /calcul_fcts_tb
# ** Note: Debut de la simulation pour le calculateur du cours SYSLOG2/CSN-2023
#    Time: 0 ns  Iteration: 0  Instance: /calcul_fcts_tb
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ns  Iteration: 3  Instance: /calcul_fcts_tb
# ** Note: Fin de la simulation ! Observez le log pour voir si vous avez des erreurs
#    Time: 200500 ns  Iteration: 0  Instance: /calcul_fcts_tb
# ** Note: end of verification process
#    Time: 201 us  Iteration: 1  Instance: /calcul_fcts_tb
# ** Note: Error detected : 0
#    Time: 201 us  Iteration: 1  Instance: /calcul_fcts_tb
```

On remarque bien que notre composant calculateur de fonction, nos trois additionneur et le test bench ont été chargé correctement.

La simulation automatique s’est réalisée **sans erreur**.

Etape 13:

Nous avons pu faire valider le laboratoire par l’assistant sans problème particulier.

Conclusion:

En conclusion, notre composant calculateur de fonction est fonctionnel néanmoins il pourrait être améliorer (reduction des éléments logique) avec quelques petits ajustement sur notre code vhdl et peut-être une autre décomposition. Nous avons fait le choix de partir avec celle-ci afin d’éviter de passer trop de temps sur la phase de conception qui aura déjà occupé une grande partie de ce labo.