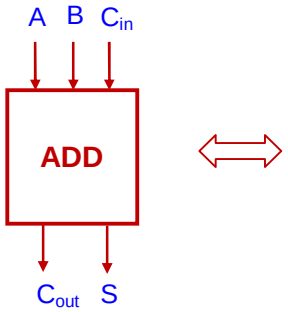
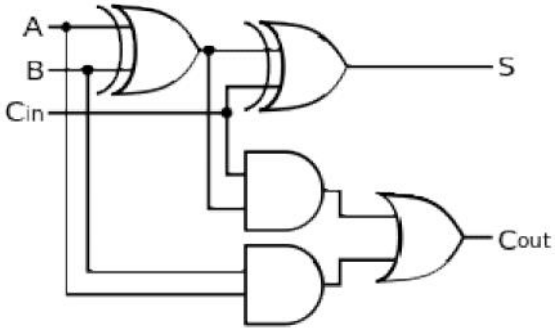
TP2 : EXPLOITATION DU SIMULATEUR DE VHDL
ADDITIONNEUR COMPLET ET MULTIPLEXEUR

Objectif : L'objectif essentiel de ce 2ème TP est toujours de bien maitriser la programmation VHDL via l'outil Altera Max+plus II. Nous nous intéressons à l'exploitation du simulateur VHDL pour la vérification des fonctions logiques.

Exercice 1 : Additionneur complet de 2 mots de 1 bit (affectation inconditionnelle)

Ecrire le code de description VHDL qui permet de réaliser un additionneur complet 1 bit, comme le montre la figure suivante :

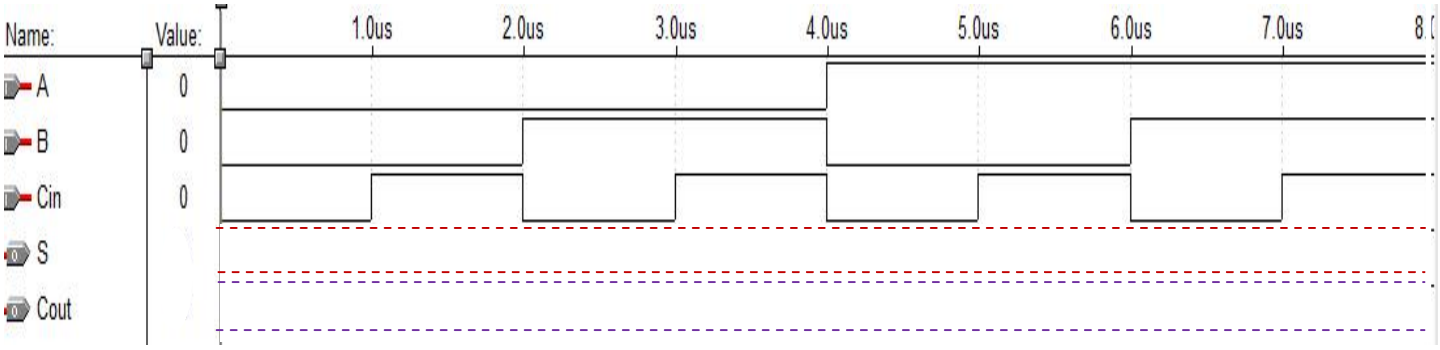




A	B	C_{in}	S	C_{out}
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

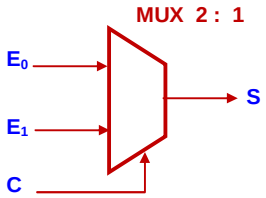
1- Sauvegarder, compiler, corriger les erreurs si nécessaire, simuler et vérifier les résultats via Waveform editor ?

2- Visualiser les chronogrammes des entrées/sorties pour un temps de 0 à 8 µs avec un pas 1 µs et remplir le tableau ?



Exercice 2 : Multiplexeur 2 vers 1 (affectation inconditionnelle)

Ecrire le code de description VHDL qui permet de réaliser le multiplexeur 2 vers 1 de la figure suivante :

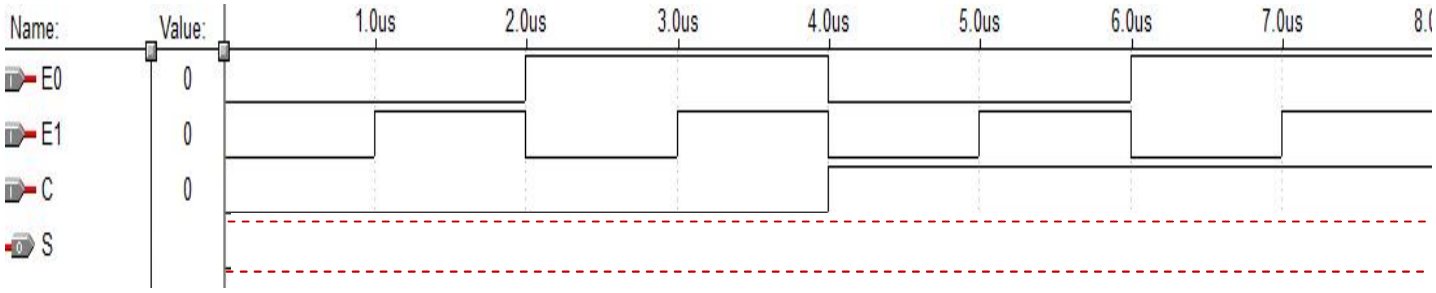


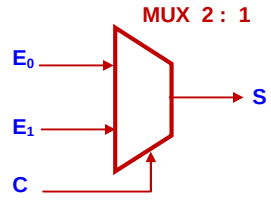
C	S
0	E_0
1	E_1

$$S = \overline{C}\, E_0 + C\, E_1$$

1- Sauvegarder, compiler, corriger les erreurs si nécessaire, simuler et vérifier les résultats via Waveform editor ?

2- Visualiser les chronogrammes des entrées / sorties pour un temps de 0 à 8 µs avec un pas 1 µs et remplir le tableau ?





C / E_0 E_1	0 0	0 1	1 0	1 1
0				
1				

3- Donner le code de description VHDL de ce circuit ?

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

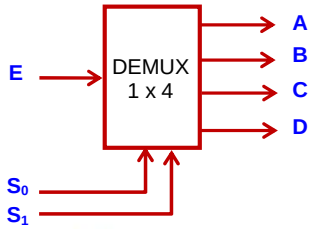
. . . . . .

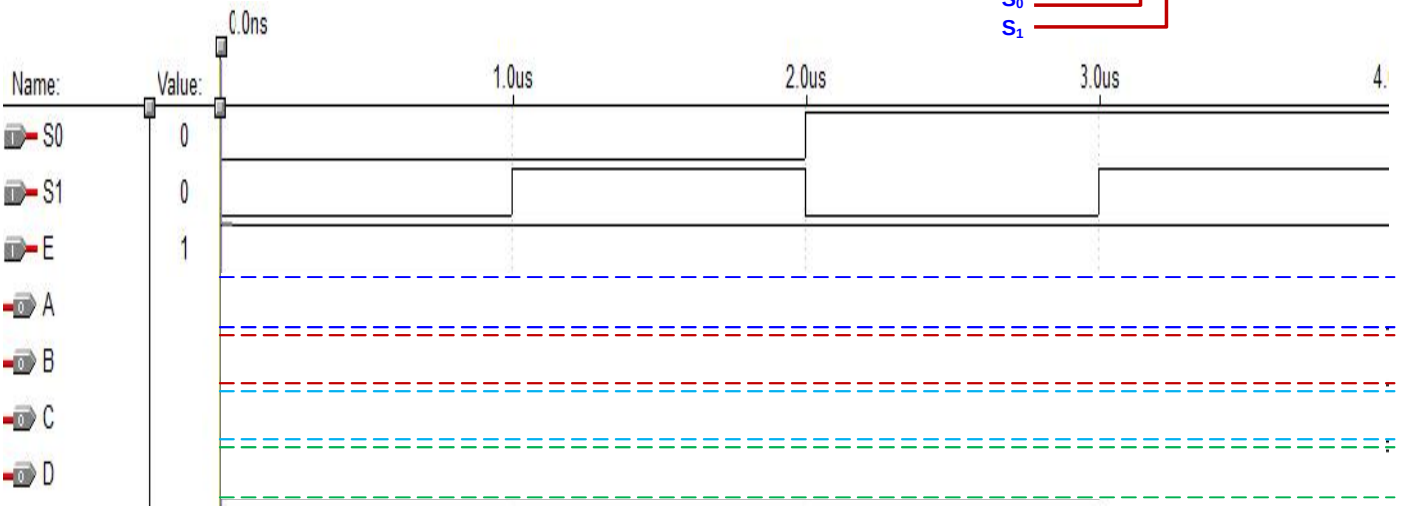
Exercice 3 : Démultiplexeur 1 vers 4 (affectation conditionnelle)

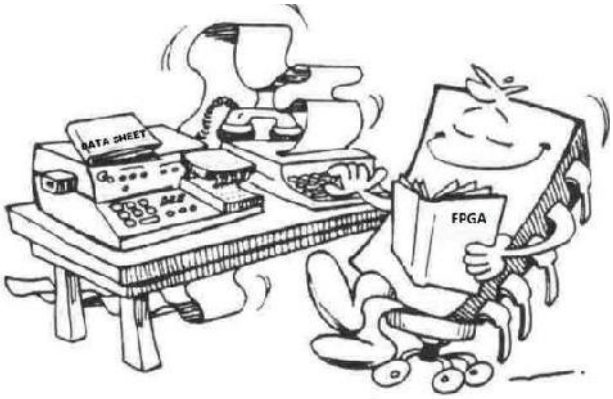
1- Ecrire et compiler le code de description VHDL du circuit de la figure en utilisant l'instruction When ... else ... ?

2- Simuler le code VHDL réalisé pour le cas suivant :

E = 1 pour les 4 situations suivantes : S_0S_1 = 00, 01, 10 et 11 (voir figure).





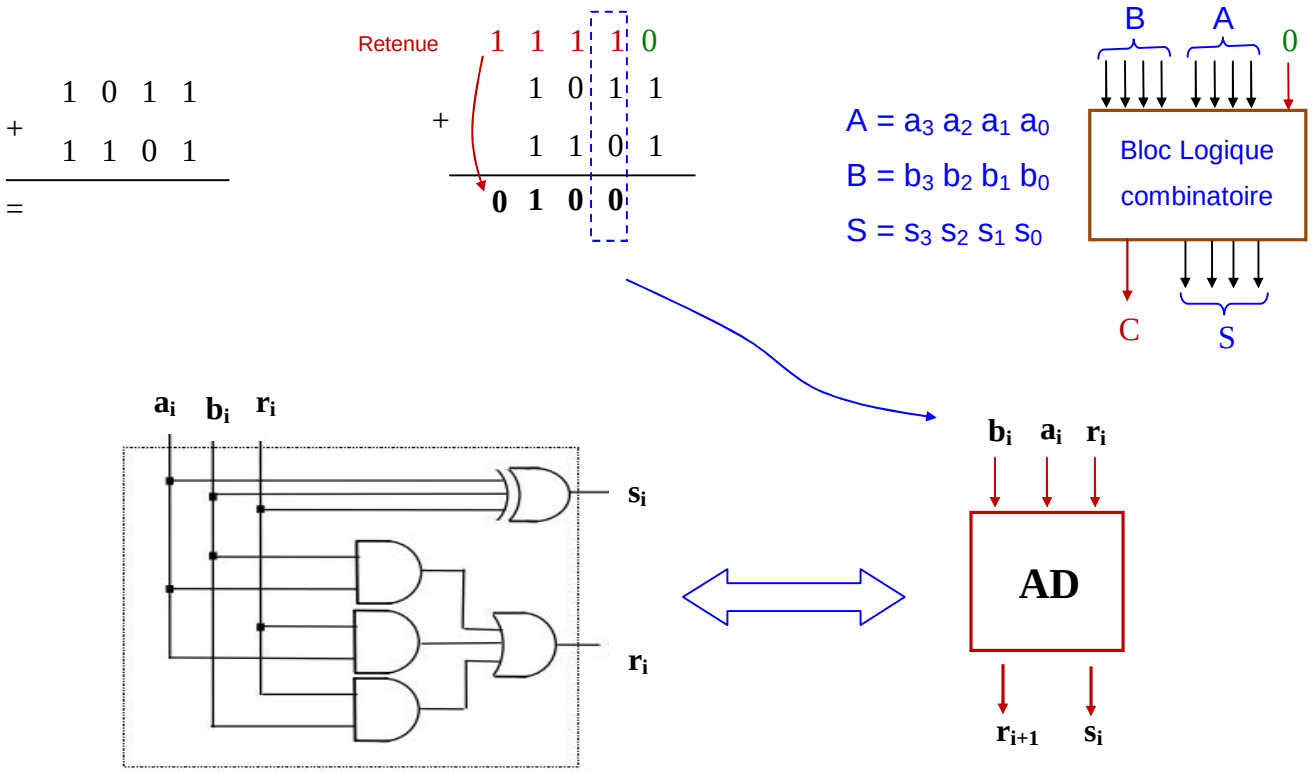


Présenté par : Dr. S. ABADLI.

Conception d'un circuit additionneur complet

La conception d'un circuit logique combinatoire additionneur est très simple à réaliser.

En première approche, un additionneur complet de deux mots de 4 bits est comme suite :



Additionneur complet de 2 mots de 1 bit

Les équations logiques permettant d'exprimer le bit s_i en fonction de a_i, b_i et de la retenue r_{i+1} sont :

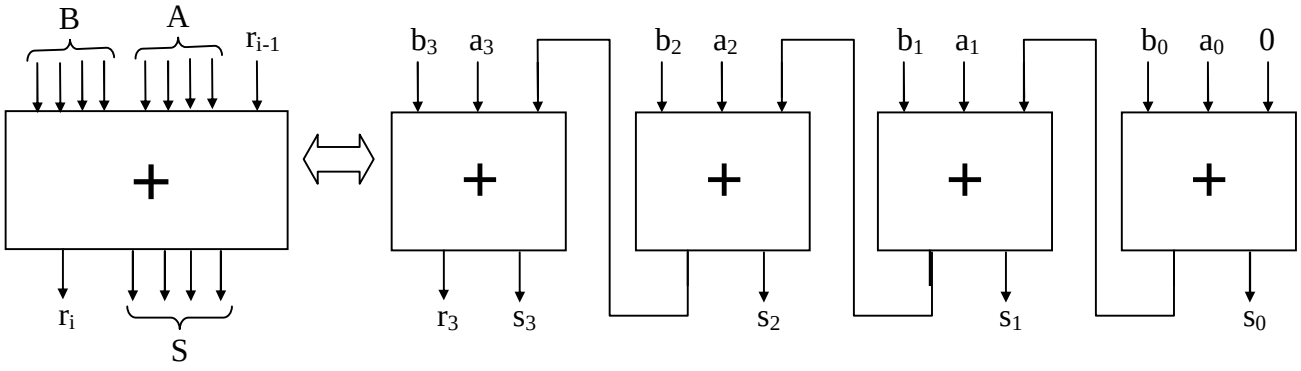
r_i \ a_i b_i	0 0	0 1	1 0	1 1
0	0	1	1	0
1	1	0	0	1

$s_i = a_i \oplus b_i \oplus r_i$

r_i \ a_i b_i	0 0	0 1	1 0	1 1
0	0	0	0	1
1	0	1	1	1

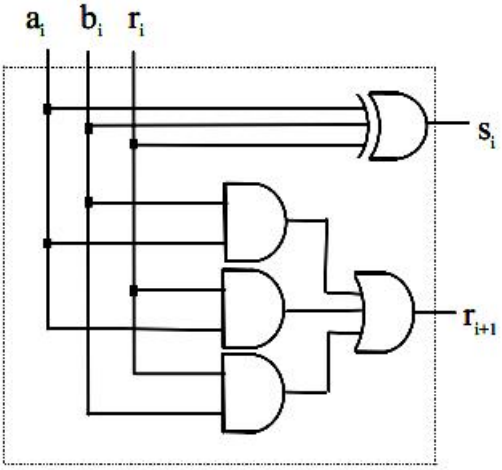
$r_{i+1} = a_i b_i + r_i a_i + r_i b_i$

Un additionneur de deux mots 4 bits est donc constitué de 4 blocs d'additionneurs de deux mots de 1 bit.



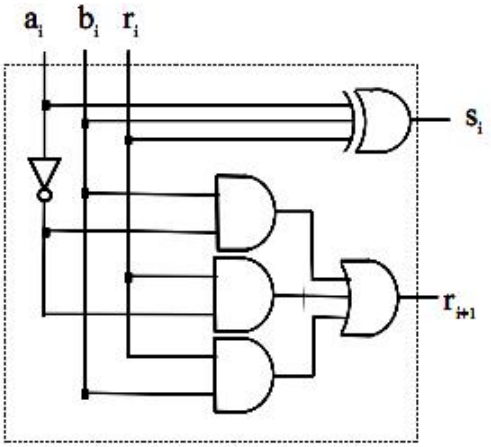
Structure d'un Additionneur de deux nombre binaires codés sur 4 bits.

Pour réaliser un soustracteur, on utilise un additionneur et on lui ajoute un complimenteur à 2.



Structure d'un **additionneur** 1 bit plus retenue

$$\begin{cases} s_i = a_i \oplus b_i \oplus r_i \\ r_{i+1} = a_i \,.\, b_i + r_i \,.\, a_i + r_i \,. \end{cases}$$

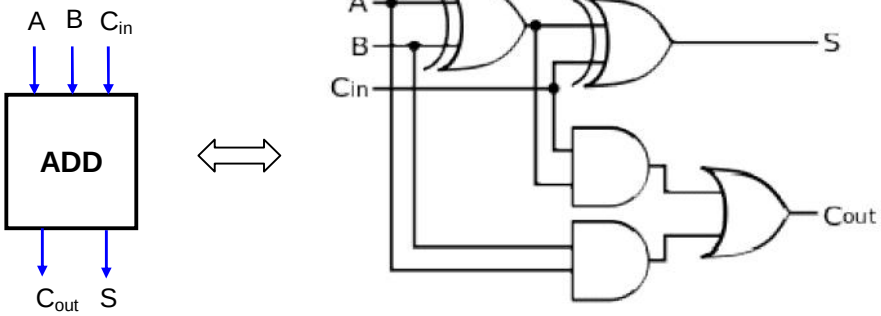


Structure d'un **soustracteur** 1 bit plus retenue

$$\begin{cases} s_i = a_i \oplus b_i \oplus r_i \\ r_{i+1} = \bar{a}_i.b_i + r_i.a_i + \bar{r}_i.b_i \end{cases}$$

Exercices corrigés

Exercice 1

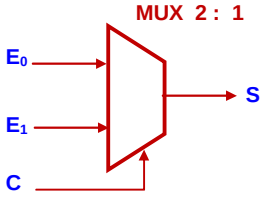



```
library ieee;
use ieee.std_logic_1164.all;

entity tp2_1 is
port( A, B, Cin : in std_logic;
      S, Cout : out std_logic);
end tp2_1;

architecture ARCHI of tp2_1 is
begin
   S <= (A xor B) xor Cin;
   Cout <= (A and (B or Cin)) or (Cin and B);
end ARCHI;
```

Exercice 2



C	S
0	E_0
1	E_1

```
library ieee;
use ieee.std_logic_1164.all;

entity tp2_2 is
port( E0, E1, C : in std_logic;
      S : out std_logic);
end tp2_2;

architecture ARCHI of tp2_2 is
begin
   --S <= E0 when C='0' else E1 when C='1';
   S <= ((not C) and E0) or (C and E1);
end ARCHI;
```

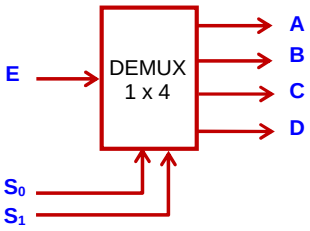
Exercice 3

1ère possibilité

```
library ieee;
  use ieee.std_logic_1164.all;

entity demux is
   port (E, S0, S1 : in std_logic;
         A, B, C, D : out std_logic);
end demux;

architecture ARCHI of demux is
  begin
     A <= E and (not S1) and (not S0);
     B <= E and (not S1) and (S0);
     C <= E and (S1) and (not S0);
     D <= E and (S1) and (S0);
end ARCHI;
```



2ème possibilité

```
library ieee;
use ieee.std_logic_1164.all;

entity exo3 is
  port( E,s0,s1 : in std_logic;
        A,B,C,D : out std_logic) ;
end exo3;

architecture ARCHI of exo3 is
  begin
   A <= E when (s0='0' and s1='0') else '0';
   B <= E when (s0='0' and s1='1') else '0';
   C <= E when (s0='1' and s1='0') else '0';
   D <= E when (s0='1' and s1='1') else '0';
end ARCHI;
```