Chapitre IV

Fonctionnement concurrent et fonctionnement séquentiel

I. Fonctionnement concurrent

1. Définition

Dans un fonctionnement concurrent l'ensemble des instructions s'exécutent parallèlement; aucun ordre ne doit être respecté. Les instructions se trouvent entre le Begin et le End d'une architecture.

Exemple1

```
ARCHITECTURE fonctionnement_concurrent  OF exemple1 IS
Signal a,b,c: std_logic_vector( 7 downto 0);
Constant init: std_logic_vector (7downto 0):="0101010";
BEGIN
A <=B and C
B <= init when select ='1' else '0';
C <=A and B

END fonctionnement_concurrent;
```

Ce code est équivalent au code suivant:

```
ARCHITECTURE fonctionnement_concurrent  OF exemple1 IS
Signal a,b,c: std_logic_vector( 7 downto 0);
Constant init: std_logic_vector (7downto 0):="0101010";
BEGIN
C <=A and B
A <=B and C
B <= init when select ='1' else '0';

END fonctionnement_concurrent;
```

2. Affectation concurrentielle

Elle spécifie la relation logique entre les différents signaux dans un système logique

Exemple2

```
ARCHITECTURE affectation_concurrentielle OF exemple2 IS
Signal a,b,y1,y2: std_logic;
```

```
BEGIN

y1 <=not(a and b);
y2<=not(a or b);

END affectation_concurrentielle;
```

3. Affectation conditionnelle

Elle permet la description d'une séquence de conditions liées pour réaliser des affectations à un signal. Elle consiste à une affectation d'une valeur à une ou plusieurs sorties à travers des conditions testées et pour les combinaisons possibles; par l'instruction ***when***

Il est à noter que dans ce genre d'affectation la notion de priorité existe.

Exemple3

Ecrire le code VHDL pour un multiplexeur 4 voies avec des affectations conditionnelles

Solution

```
Entity Mux4 is
Port (sel: in std_logic_vector( 0 to 1);
A,b,c,d: in std_logic;
y: out std_logic);
end mux4;

ARCHITECTURE affectation_conditionnelle OF mux4 IS
BEGIN

y <=a when sel ="00" else
      b when sel="01" else
      c when sel="10" else
      d when sel="11";
END affectation_conditionnelle;
```

4. Affectation sélective

Il s'agit du même principe que l'affectation conditionnelle mais sans aucune priorité.

Exemple4

Ecrire le code VHDL pour un multiplexeur 4 voies avec des affectations sélectives

Solution

```
ARCHITECTURE affectation_sélective OF mux4 IS
BEGIN

with sel select
y <=a when "00",
      b when "01",
      c when "11",
      d when others;
END affectation_séléctive;
```

5. Comparaison entre l'affectation sélective et l'affectation conditionnelle

Outre la notion de priorité; l'affectation conditionnelle c'est la description naturelle pour décrire un fonctionnement, cependant pour éviter l'introduction des informations additionnelles et des logiques indésirables il est préférable d'utiliser l'affectation sélective.

Exemple5

Ecrire le code VHDL pour un circuit logique combinatoire de 3 entrées A,B,C; une seule entrée qui peut prendre '1' à instant donné. Les sorties Q_1 et Q_2 sont définies comme suit:

Q1 =Q2= 01 pour A=1, 10 pour B=1, 11 pour C=1, 00 pour les autres combinaisons

Solution

```
ARCHITECTURE affectation_sélective OF exemple5 IS
BEGIN

--Q est décalaré std_vector(0 to 1)

with Q select
Q <="01" when "100",
      "10" when "010",
      "11" when "001",
      "00" when others;
END affectation_séléctive;
```

Cette conception permet d'optimiser les ressources lors de l'implémentation

Remarques

a. Dans une affectation sélective on doit écrire toutes les possibilités, sinon on doit ajouter l'instruction **others** pour décrire les possibilités restantes.

b. Dans une affectation sélective on peut insérer l'instruction **to** pour définir une plage de variation

Exemple6

```
ARCHITECTURE affectation_sélective OF exemple6 IS
BEGIN

with entrée select
Q <='1'when "0000" to "1000",
      '0' when others;
END affectation_séléctive;
```

Pout spécifier que la sortie reste inchangée dans une affectation sélective on peut insérer l'instruction **unaffected**

Exemple7

```
ARCHITECTURE affectation_sélective OF exemple7 IS
BEGIN

with sel select
y <=a when "00",
      b when "01",
      c when "11",
      unaffected when others;
```

6. Les retards

Le langage VHDL permet l'introduction des retards dans la conception, deux types de retard existent:

after: modélisation des retards des portes logiques et les fils électriques, elle exige un minimum de largeur d'impulsion

transport: modélisation des retards dans les fils, il s'agit d'une propagation sans un minimum exigé de largeur d'impulsion.

Exemple8

```
ARCHITECTURE retard OF exemple8 IS
BEGIN

Y1 <=not (a and b) after 7 ns;
Y2 <=not (a and b) transport after 7 ns;
```

```
END retard;
```

II. Fonctionnement séquentiel

1. Définition

Contrairement au fonctionnement concurrent, le fonctionnement séquentiel impose un certain ordre dans l'exécution.

2. Structure de contrôle

a. L'instruction de contrôle if-elsif-then

Sa forme générale est donnée par :

```
If condition1 then
    Instructions ;
Elsif condition2 then
    Instructions ;
else
   Instructions ;
End if ;
```

Elle peut être aussi imbriquée :

```
 If condition_externe then
     Instructions;
Else
      If condition_intene then
            Instructions;
      End if ;
End if ;
```

b. L'instruction de contrôle case

Sa forme générale est donnée par :

```
Case expression de contrôle is
       When test1=>
                instructions ;
      When test2=>
                instructions ;
      When others =>
                instructions;
End case ;
```

Remarque

L'instruction if-elsif-then implique un ordre de priorité contrairement à l'instruction case qui ne nécessite aucun ordre de priorité.

3. Les boucles

a. La boucle For

Sa forme générale est donnée par :

```
Nom boucle : for i in plage de variation loop
Instructions ;
End loop ;
```

b. La boucle while

Sa forme générale est donnée par :

```
Nom boucle : while condition  loop
Instructions ;
End loop ;
```

Remarque

a. Nom de la boucle est optionnel.

b. La plage de variation peut être un type énuméré:

```
Type Etat is (init, clear, error, receive)
 Nom boucle : for i in etat loop
Instructions ;
End loop ;
```

c. La boucle infinie

Sa forme générale est donnée par :

```
Nom boucle : loop
Instructions ;
Exit when condition
End loop ;
```

Remarque

a. **Exit** peut être utilisée pour faire terminer la boucle for ou while

b. Dans le cas d'une boucle imbriquée, **exit** doit être accompagnée par le nom de la boucle, sinon elle fait fin uniquement à la boucle à laquelle elle est attachée.

4. Le processus

Il est défini par le mot clé ***process***, il représente une instruction concurrente dans une architecture mais il contient des instructions séquentielles. Le processus ne prend fin qu'à la fin de la simulation.

La forme générale d'un processus est:

```
Nom_processus: process (liste de sensibilité)
-- déclarations
BEGIN
-- instructions séquentielles
END process;
```

- ✓ Nom_processus: optionnel
- ✓ Process: mot clé
- ✓ Liste de sensibilité: ensemble de signaux provoquant le déclenchement du processus; elle est optionnelle et tout signal dans cette liste doit être utilisé dans le processus.
- ✓ Si la liste est vide, le processus doit contenir au moins une instruction ***wait***

Exemple9

```
Process(ck)
BEGIN
If ck ='1' and ck'event then – test du front montant
Q <=data;
End if;
END process;
```

5. Le processus pour les systèmes combinatoires

Le processus peut être utilisé dans la description des systèmes combinatoires, certaines règles doivent être respectées:

- ✓ La liste de sensibilité doit contenir tous les signaux d'entrée définis dans l'entité
- ✓ L'affectation des sorties couvre toutes les valeurs possibles définies dans la liste de sensibilité.
- ✓ Si le processus contient des variables, ces dernières doivent avoir une valeur.

Exemple10: Multiplexeur 4 voies

```
library ieee;
use ieee.std_logic_1164.all;
entity mux4 is
port (sel: in std_logic_vector( 0 to 1);
a,b,c,d: in std_logic;
y: out std_logic);
end mux4;

architecture processus_combinatoire of mux4 is
begin
process ( sel,a,b,c,d)
begin
if  sel ="00" then
     y  <= a ;

elsif sel ="01" then
     y  <= b ;

elsif sel ="10" then
     y  <= c ;

elsif sel ="11" then
     y  <= d ;
end if;
end process;
end processus_combinatoire;
```

6. Le processus pour les systèmes séquentiels

Pour l'utilisation du processus dans les systèmes séquentiels; il est recommandé de respecter les règles suivantes:

- ✓ La liste de sensibilité ne contient pas toute les entrées, les signaux qui se trouvent dans la liste de sensibilité interviennent au déclenchement du processus.
- ✓ Utiliser des boucles ***if- then-elsif*** incomplètes assurant que les signaux gardent leurs valeurs sous certaines conditions.
- ✓ Utiliser des variables qui gardent leurs valeurs entre deux itérations du processus.

Exemple11: Registre à décalage cyclique

```
library ieee;
use ieee.std_logic_1164.all;
entity cyclique is
port (ck, rst, load: in std_logic;
data: in std_logic_vector (0 to 7);
q: out std_logic_vector (0 to 7));
end cyclique;

architecture processus_sequentiel  of cyclique  is

signal s:  std_logic_vector (0 to 7);

begin
seq : process ( ck,rst)
begin

if  rst ='1' then --if then incomplète
     s  <= "00000000" ;
elsif (ck ='1' and ck'event) then --if-then incomplète
     if (load ='1') then
          s  <= data ;
     else
         s  <= s(1 to7) &  s(0);
    end if;
end if;
end process;
q <= s ;
end processus_sequentiel;
```

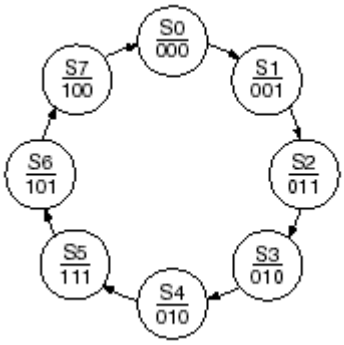
7. Le processus pour la machine d'état

Le processus peut être utilisé pour la description de la machine d'état à travers le type énuméré.

7.1 Machine de Moore

Exemple 12

Écrire son VHDL d'un un compteur synchrone en code gray avec une sortie Q de 3 bits seulement, son diagramme d'état est donnée par :



Solution

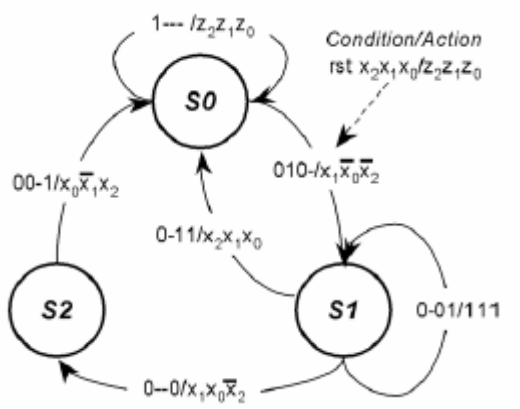
```
library ieee;
use ieee.std_logic_1164.all;
entity compteur_gray is
port( clk : in std_logic;
          q : out std_logic_vector(2 downto 0));
end compteur_gray;
architecture processus_machine_etat of compteur_gray is
type  etat  is (s0, s1, s2, s3, s4, s5, s6, s7);
signal state: etat ;
begin
process (clk)
begin
if clk'event and clk = '1' then
  case state is
     when s0 => state <= s1;
     when s1 => state <= s2;
     when s2 => state <= s3;
     when s3 => state <= s4;
     when s4 => state <= s5;
     when s5 => state <= s6;
     when s6=> state <= s7;
     when s7 => state <= s0;
  end case;
end if;
end process;

with state select
q <=" 000" when s0,
      "001" when s1,
      "011" when s2,
      "010" when s3,
      "110" when s4,
      "111" when s5,
     "101" when s6,
     "100" when s7;
end processus_machine_etat;
```

7.2 Machine de Mealy

Exemple 13

Soit le graphe des états suivants



Solution

```
library ieee;
use ieee.std_logic_1164.all;
entity exemple13  is
port ( clk, rst : in std_logic;
x : in std_logic_vector(2 downto 0);
z : out std_logic_vector(2 downto 0));
end exemple13;

architecture processus_mealy of exemple13 is
type etat is (S0, S1, S2);
signal state: etat;

begin
graphe_etat: process (rst, clk)
begin
if rst = '1' then state <= S0;
elsif clk'event and clk = '1' then
case state is

when S0 => if x(2) = '1' and x(1) = '0' then
state <= S1;
end if;

when S1 => if x(0) = '0' then
state <= S2;
elsif x(1) = '1'  and  x(0) = '1' then
state <= S0;
elsif x(1) = '0'  and  x(0) = '1' then
state <= S1;
end if;
```

```
when S2 => if x(2) = '0' and x(0) = '1' then
state <= S0;
end if;

when others => state <= S0;

end case;

end if;

end process graphe_etat;
sorties: process (state, x)

begin

case state is

when S0 => if x(2) = '0' and x(0) = '1' then
z(2) <= x(0);
z(1) <= not x(1);
z(0) <= x(2);
elsif x(1) = '1' and x(0) = '1' then
z(2) <= x(2);
z(1) <= x(1);
z(0) <= x(0);
end if;

when S1 => if x(2) = '1' and x(1) = '0' then
z(2) <= x(1);
z(1) <= not x(0);
z(0) <= not x(2);
elsif x(1) = '0' and x(0) = '1' then
z <= "111";
end if;

when S2 => if x(0) = '0' then
z(2) <= x(1);
z(1) <= x(0);
z(0) <= not x(2);
end if;

when others => z <= "---";

end case;

end process Sorties;
end architecture processus_mealy;
```

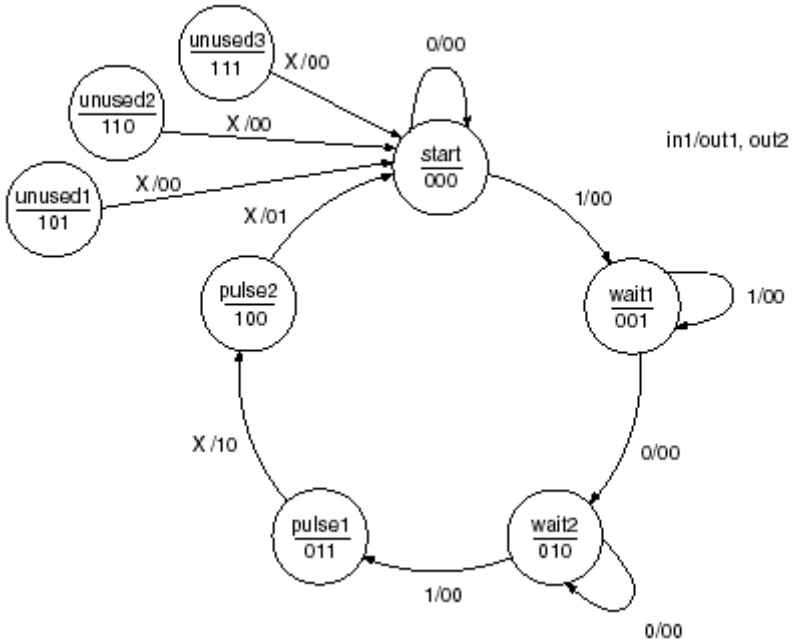
7.3 Cas des états inutilisés

Dans la conception des systèmes séquentiels synchrones; les états doivent être assignés sur un nombre de bits défini par le nombre de bascules à utilisées, les états restants sont définis comme des états indéterminés.

Pour une conception VHDL d'une machine d'état, on peut définir les états inutilisés par deux manières :

- ✓ Soit ne sont pas pris en compte (indéterminés)
- ✓ Soit faire orienter tous ces états vers un état dans la machine, et pour une raison de sécurité il est préférable de les orienter vers le premier état appelé état de départ.

Exemple14




```
Library ieee;
Use ieee.std_logic_1164.all;
Entity two_pulse is
Port( clk, in1 : in std_logic;
Output : out std_logic_vector (1 to 2));
End two_pulse;
Architecture etat_inutilises of two_pulse is
Type sequence is (start, wait1, wait2, pulse1, pulse2);
Signal pulse_state : sequence;
Begin
Process(clk)
Begin
If (clk'event and clk = '1') then
Case pulse_state is
When start =>
If in1 = '0' then
```

```
Pulse_state <= start;
Output <= "00";
Elsif in1 = '1' then
Pulse_state <= wait1;
Output <= "00";
End if;
When wait1 =>
If in1 = '0' then
Pulse_state <= wait2;
Output <= "00";
Elsif in1 = '1' then
Pulse_state <= wait1;
Output <= "00";
End if;
When wait2 =>
If in1 = '0' then
Pulse_state <= wait2;
Output <= "00";
Elsif in1 = '1' then
Pulse_state <= pulse1;
Output <= "00";
End if;
When pulse1 =>
Pulse_state <= pulse2;
Output <= "10";
When pulse2 =>
Pulse_state <= start;
Output <= "01";
When others =>
Pulse_state <= start;
Output <= "00";
End case;
End if;
End process;
End etat_inutilises;
```

7.4 Encodage des états

L'encodage des états pour une machine d'état peut avoir un impact important sur la taille de la conception, il consiste à attribuer à chaque état un nombre binaire unique.

Différentes techniques existent pout l'encodage :

L'encodage séquentiel est l'encodage par défaut pour les outils de synthèse, à chaque état est assigné un nombre binaire croissant à partir d'une chaîne de "n" zéros.

Les encodages de Gray et de Johnson assignent des valeurs binaires successives qui ont la particularité de ne changer qu'un seul bit en passant d'un nombre à un autre. L'avantage

est de réduire les erreurs de transitions possibles dues à des changements des entrées asynchrones.

L'encodage de Gray utilise tous les 2^N états possibles alors que l'encodage de Johnson revient à implémenter un registre d'état à décalage et requiert ainsi plus de bascules. L'usage du code de Gray permet aussi de minimiser la consommation des ressources du circuit.

L'encodage "one-hot" n'utilise qu'une seule bascule par état. Ceci permet de simplifier la logique de décodage et de réduire le temps de transition d'un état à l'autre.

Les outils de synthèse permettent aussi de définir un encodage personnalisé.

La Table ci-dessous donne les valeurs d'encodage pour 16 états et pour chacun des styles d'encodages ci-dessus.

No.	Séquentiel	Gray	Johnson	One-hot
0	0000	0000	00000000	0000000000000001
1	0001	0001	00000001	0000000000000010
2	0010	0011	00000011	0000000000000100
3	0011	0010	00000111	0000000000001000
4	0100	0110	00001111	0000000000010000
5	0101	0111	00011111	0000000000100000
6	0110	0101	00111111	0000000001000000
7	0111	0100	01111111	0000000010000000
8	1000	1100	11111111	0000000100000000
9	1001	1101	11111110	0000001000000000
10	1010	1111	11111100	0000010000000000
11	1011	1110	11111000	0000100000000000
12	1100	1010	11110000	0001000000000000
13	1101	1011	11100000	0010000000000000
14	1110	1001	11000000	0100000000000000
15	1111	1000	10000000	1000000000000000

Exemple 15: encodage séquentiel pour l'exemple 12

```
architecture encodage_sequentiel  of compteur_gray is
type  etat  is array (2 downto 0) of std_logic;
constant s0: etat:="000";
constant s1: etat:="001";
constant s2: etat:="010";
constant s3: etat:="011";
constant s4: etat:="100";
constant s5: etat:="101";
constant s6: etat:="110";
constant s7: etat:="111";
signal state: etat;
```

```
begin
process (clk)
begin
if clk'event and clk = '1' then
  case state is
      when s0 => state <= s1;
      when s1 => state <= s2;
      when s2 => state <= s3;
      when s3 => state <= s4;
      when s4 => state <= s5;
      when s5 => state <= s6;
      when s6=> state <= s7;
      when s7 => state <= s0;
      when others=> null; -- aucune action
 end case;
end if;
end process;

with state select
q <="000" when s0,
       "001" when s1,
       "011" when s2,
       "010" when s3,
       "110" when s4,
       "111" when s5,
      "101" when s6,
      "100" when s7,
     "000" when others;-- pour rester à l'état initial mais on peut choisir les sorties
                                --selon notre cas
end encodage_sequentiel;
```

Exemple 16: encodage one-hot pour l'exemple 12

```
architecture encodage one_hot  of compteur_gray is
type  etat  is array (7 downto 0) of std_logic;
constant s0: etat:="00000001";
constant s1: etat:="00000010";
constant s2: etat:="00000100";
constant s3: etat:="00001000";
constant s4: etat:="00010000";
constant s5: etat:="00100000";
constant s6: etat:="01000000";
constant s7: etat:="10000000";
signal state:etat;

begin
process (clk)
begin
if clk'event and clk = '1' then
  case state is
      when s0 => state <= s1;
```

```
    when s1 => state <= s2;
    when s2 => state <= s3;
    when s3 => state <= s4;
    when s4 => state <= s5;
    when s5 => state <= s6;
    when s6=> state <= s7;
    when s7 => state <= s0;
    when others=> state <= s0;-- si on fait null le compteur n'avance pas car dans ce
                                --type d'encodage un seul bit doit avoir la valeur 1
  end case;
end if;
end process;

with state select
q <="000" when s0,
     "001" when s1,
     "011" when s2,
     "010" when s3,
     "110" when s4,
     "111" when s5,
    "101" when s6,
    "100" when s7,
"000" when others;
end encodage one_hot;
```

III. Choisir une variable ou un signal

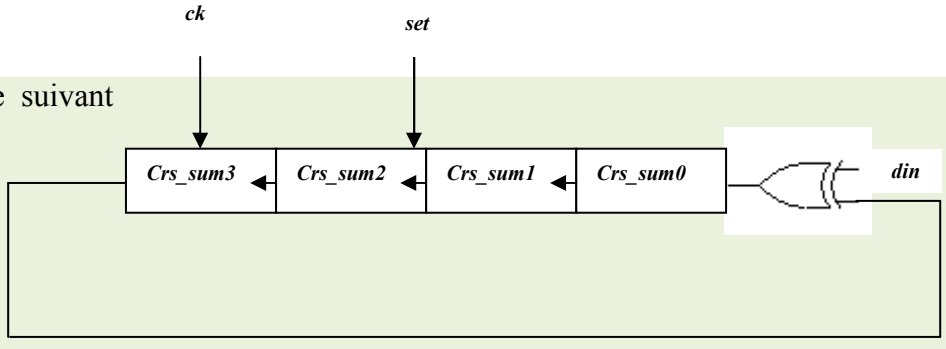
Dans un processus ou sous-programme, il est important de faire la différence entre variable et signal, le choix présente des considérations importantes et toute erreur peut engendrer des situations délicates lors de la simulation.

Notons que:

- ✓ Une variable dans un processus change de valeur immédiatement après l'affectation
- ✓ Un signal garde la valeur affectée jusqu'à l'arrêt du processus

Exemple 17

Soit le registre suivant



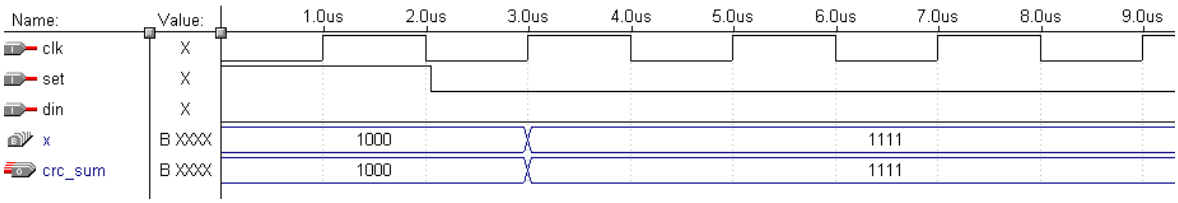
- ✓ Si : Set=1 alors crc_sum=1000
- ✓ Sinon si le front montant fonctionnement selon le schéma

Solution

Utilisation d'une variable

```
library ieee;
use ieee.std_logic_1164.all;
entity crc is
port(clk,set,din: in std_logic;
    crc_sum: out std_logic_vector(3 downto 0));
end crc;

architecture crc_variable of crc is
begin
process (clk,set)
variable x : std_logic_vector (3 downto 0);
 begin
if set='1' then
x := "1000";
elsif rising_edge(clk) then
x(0):=din xor x(3);
x(1):=x(0);
x(2):=x(1);
x(3):=x(2);
end if;
crc_sum <= x;
end process;
end crc_variable;
```



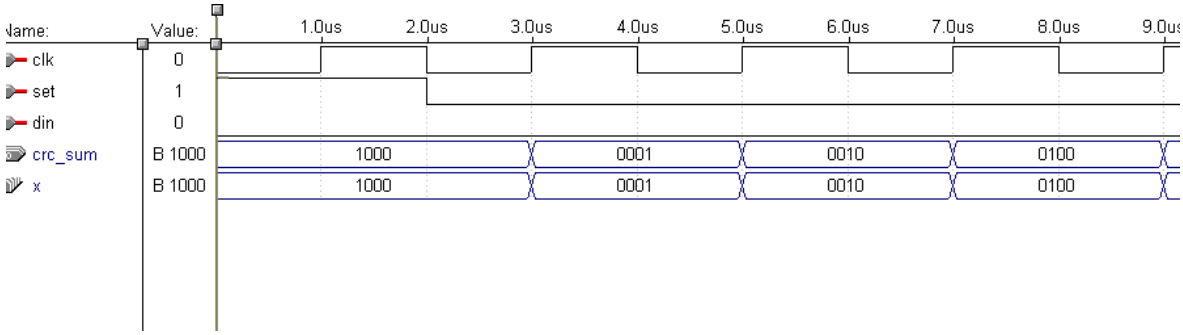
Utilisation d'un signal

```
architecture crc_signal of crc_signal is

signal x : std_logic_vector (3 downto 0)
process (clk,set)
begin
if set='1' then
x <= "1000";
elsif rising_edge(clk) then
x(0)<=din xor x(3);
x(1)<=x(0);
x(2)<=x(1);
```

```
x(3)<=x(2);
end if;
end process;
crc_sum <= x;

--  crc_sum doit être situé dans l'architecture et nom dans le process
end crc_signal;
```



Exercices

Exercice1

Écrire le code VHDL d'un décodeur 2-4 en utilisant:

- ✓ L'affectation concurrentielle
- ✓ L'affectation conditionnelle
- ✓ L'affectation sélective

Exercice2

Écrire le code VHDL d'un décodeur BCD-7 segments en utilisant l'affectation sélective

Exercice3

Écrire le code VHDL de la bascule JK.

Exercice4

Écrire le code VHDL d'un compteur synchrone binaire 8 bits (qui compte de 0 à 255) et ayant une entrée clear; en utilisant une description comportementale.

Exercice5

Soit un compteur synchrone binaire 4 bits ayant les entrées suivantes :

P(4bits), load(1bit), ck, clear, count_ena(1bit), direction (1bit)

Le compteur fonctionne comme suit :

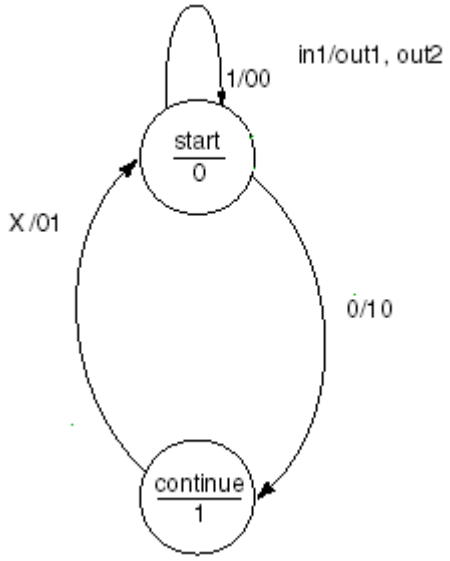
- ✓ Si clear =0 on une remise à zéro
- ✓ Si load=1 et clear=1 alors chargement par p (sortie=p)
- ✓ Si le front montant alors :

- Si count_ena=1 et direction =0 décomptage
- Si count_ena=1 et direction =1 comptage

Écrire son code VHDL en utilisant une description comportementale

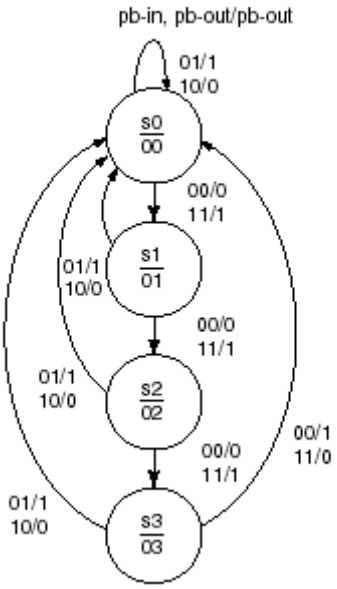
Exercice6

Écrire le code VHDL pour la machine d'état suivante :



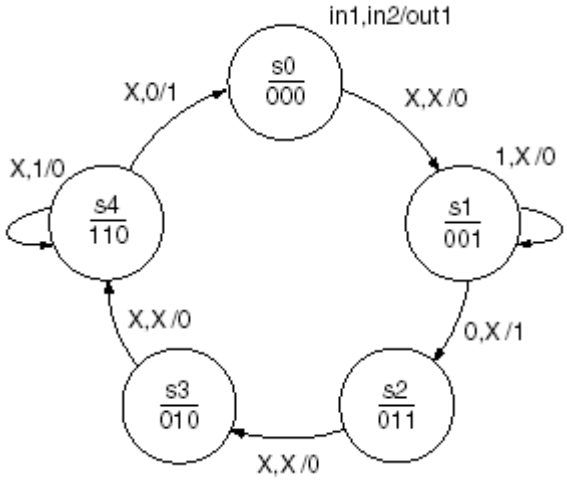
Exercice7

Soit un système fonctionnant selon le graphe des états ci-dessous, ce système ayant une entrée principale P_in et une sortie P_out utilisée comme buffer (pour le test de passage d'un état vers un autre avec P_in), écrire le code VHDL



Exercice8

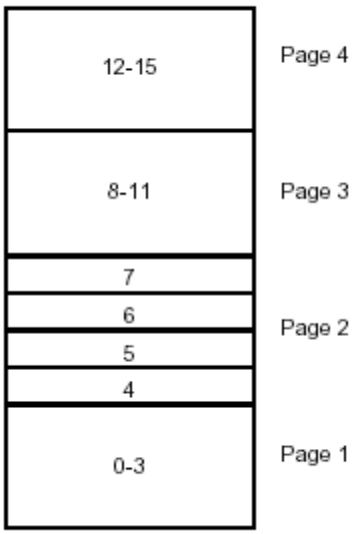
Écrire le code VHDL pour la machine d'état suivante en complétant les états non utilisés



Exercice9

Soit un décodeur d'adresse mémoire 4 bits. On suppose que la mémoire considérée est décomposée en quatre pages, la deuxième page étant elle-même décomposée en quatre segments.

Table des adresses de la mémoire.



Écrire le code VHDL du décodeur

Exercice10

L'unité arithmétique et logique (Arithmetic and Logic Unit - ALU) constitue le cœur d'une unité de calcul telle que l'on trouve dans un microprocesseur. Elle est capable d'effectuer des opérations arithmétiques (addition, soustraction, incrémentation, décrémentation, décalages) et logiques (AND, OR, NOT, XOR) de base sur deux bus de données.

Écrire le code VHDL de cette ALU pour des arguments non signés

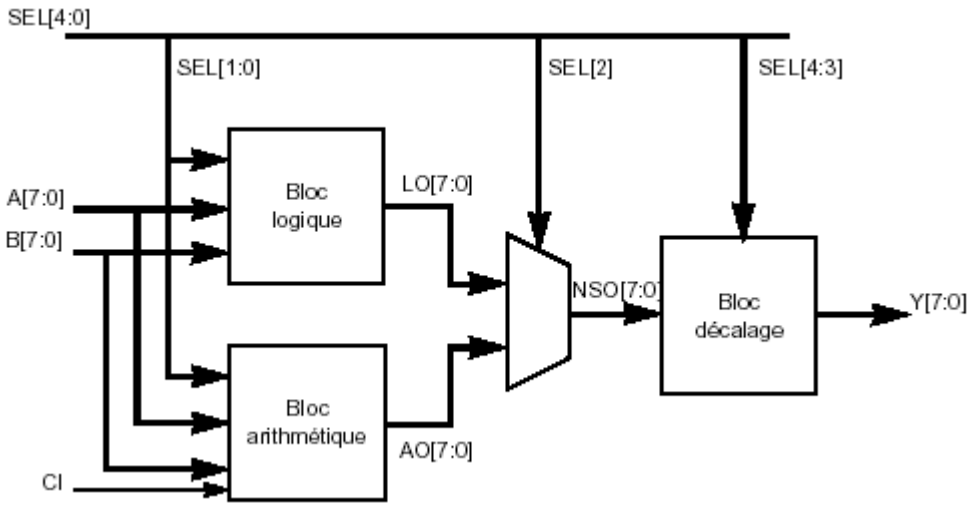
Exercice11

Soit une ALU de 8 bits fonctionnant comme suit :

Table 4.2: Table des fonctions d'une ALU 8 bits.

S4	S3	S2	S1	S0	CI	Opération	Fonction	Unité
0	0	0	0	0	0	Y <= A	Transfert de A	arithmétique
0	0	0	0	0	1	Y <= A + 1	Incrémentation de A	
0	0	0	0	1	0	Y <= A + B	Addition	
0	0	0	0	1	1	Y <= A + $\overline{B}$ + 1	Addition avec retenue	
0	0	0	1	0	0	Y <= A + $\overline{B}$	A + compl. à 1 de B	
0	0	0	1	0	1	Y <= A + $\overline{B}$ + 1	Soustraction	
0	0	0	1	1	0	Y <= A - 1	Décrémentation de A	
0	0	0	1	1	1	Y <= B	Transfert de B	
0	0	1	0	0	0	Y <= A and B	AND	logique
0	0	1	0	1	0	Y <= A or B	OR	
0	0	1	1	0	0	Y <= A xor B	XOR	
0	0	1	1	1	0	Y <= $\overline{A}$	Complément	
0	0	0	0	0	0	Y <= A	Transfert de A	décalage
0	1	0	0	0	0	Y <= shl A	Décalage à gauche de A	
1	0	0	0	0	0	Y <= shr A	Décalage à droite de A	
1	1	0	0	0	0	Y <= 0	Transfert de zéros	

Son diagramme bloc est donné par :



Donner le code VHDL correspondant