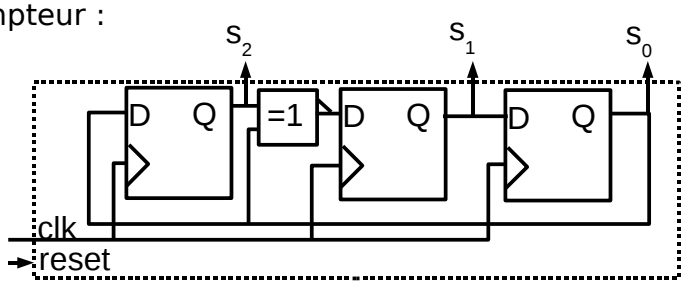
NOM : **Le Correcteur**
Prénom :
Groupe : **Correction en rouge dans ce document**

MCENSL1

PROBLEME I : Implantation d'un registres à décalage à rétroaction linéaire

On souhaite implanter un registres à décalage à rétroaction linéaire (normalement employé pour générer des bits aléatoires) comme indiqué dans la figure ci-dessous pour réaliser un compteur :



1°) Écrire les équations de récurrence de ce compteur
Réponses :
$S_2^+ = Q_0 = S_0$

$S_1^+ = /(Q_2 \text{ ouex } Q_0) = /(S_2 \text{ ouex } S_0)$

$S_0^+ = Q_1 = S_1$

2°) En déduire le programme VHDL (entité et architecture) correspondant si le reset est réalisé de manière asynchrone (et positionne $[S_2\ S_1\ S_0]_2$ à $[0\ 0\ 1]_2$).
Réponse :

Entité du registre à rétroaction	Architecture du registre à rétroaction

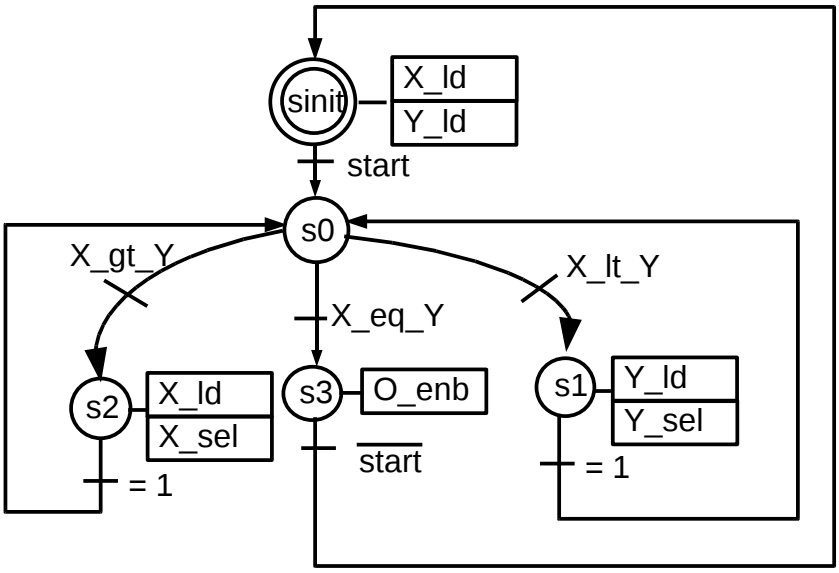
-- DS 19 Novembre 2009
ENTITY reg IS
PORT(clk,Reset : in BIT;
S : BUFFER BIT_VECTOR(2 DOWNTO 0));
END reg;
	-- DS 19 Novembre 2009
	ARCHITECTURE areg OF reg IS
	BEGIN
	PROCESS(clk,reset) BEGIN
	IF reset='1' THEN
	S<="001"
	ELSIF clk'event AND clk='1' THEN
	S(2) <= S(0);
	S(1) <= NOT (S(2) XOR S(0);
	S(0) <= S(1);
	END IF;
	END PROCESS;
	END areg;

PROBLÈME II : Calcul de PGCD

On reprend le calcul du PGCD du DS de l'année dernière (qui a été ajouté au polycopié de cette année) mais en utilisant un algorithme différent, présenté en anglais ici :
http://www.cs.ucr.edu/~vahid/sproj/SystemCLab/lab4.htm

Le calcul du PGCD est réalisé avec un circuit comportant deux parties : un séquenceur décrit par un graphe d'états et une partie opérative (ou chemin de données) composée de registres 8 bits. Il n'est pas nécessaire de se rappeler comment on calcule un PGCD pour continuer ce problème.

Le séquenceur est décrit par le graphe d'états ci-dessous.



1°) Peut-on réaliser une synthèse sans coder les états (ou avec codage one hot one) pour ce graphe d'états complet (états + sorties) si l'on veut utiliser une seule GAL22V10 ?

Réponse :

5 états + 5 sorties fait 10 sorties en tout, c'est justement le 10 de 22V10 donc OK.

On décide d'utiliser un outil automatique pour générer le code VHDL correspondant à ce graphe d'état. Pour cela, on dessine le graphe d'évolution correspondant et on demande une génération automatique.
Voici le fichier VHDL généré (en partie seulement) ainsi que le fichier rapport en compilant le fichier VHDL. Remarquez que ce fichier rapport permet de trouver le brochage du composant, quelles variables VHDL sur quelles broches.

NOM :
Prénom :

Devoir surveillé MCENSL1 (suite)

Programme VHDL généré	*Fichier Rapport*

```
-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S0, S1, S2, S3, sInit);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments
--diagram ACTIONS;


Sreg0_machine: process (CLK)

begin

if CLK'event and CLK = '1' then
       if Init='1' then
              Sreg0 <= sInit;
       else
       case Sreg0 is
              when S0 =>
                     if X_gt_Y='1' then
                            Sreg0 <= S2;
                     elsif X_lt_Y='1' then
                            Sreg0 <= S1;
                     elsif X_eq_Y='1' then
                            Sreg0 <= S3;
                     end if;
              when S1 =>
                     Sreg0 <= S0;
              when S2 =>
                     Sreg0 <= S0;
              when S3 =>
                     if start='0' then
                       Sreg0 <= sInit;
                     end if;
              when sInit =>
                     if start='0' then
                       Sreg0 <= sInit;
                     elsif start='1' then
                       Sreg0 <= S0;
                     end if;
              when others => null;
       end case;
       end if;
end if;
end process;
```

```
State variable 'sreg0' is represented by a Bit_vector
(0 to 2).
State encoding (sequential) for 'sreg0' is:
       s0 :=  b"000";
       s1 :=  b"001";
       s2 :=  b"010";
       s3 :=  b"011";
       sinit :=       b"100";
....
DESIGN EQUATIONS           (16:10:02)
o_enb = sreg0SBV_1.Q * sreg0SBV_2.Q
x_sel = sreg0SBV_1.Q * /sreg0SBV_2.Q
y_sel = /sreg0SBV_1.Q * sreg0SBV_2.Q
x_ld = sreg0SBV_1.Q * /sreg0SBV_2.Q
        + sreg0SBV_0.Q
y_ld = /sreg0SBV_1.Q * sreg0SBV_2.Q
        + sreg0SBV_0.Q

sreg0SBV_0.D =
          sreg0SBV_1.Q * sreg0SBV_2.Q * /start
        + sreg0SBV_0.Q * /start
        + init

sreg0SBV_1.D =
          /sreg0SBV_1.Q * /sreg0SBV_2.Q *
/sreg0SBV_0.Q * /init * x_eq_y * /x_lt_y
        + /sreg0SBV_1.Q * /sreg0SBV_2.Q *
/sreg0SBV_0.Q * /init * x_gt_y
        + sreg0SBV_1.Q * sreg0SBV_2.Q * /init * start

sreg0SBV_2.D =
          /sreg0SBV_1.Q * /sreg0SBV_2.Q *
/sreg0SBV_0.Q * /init * /x_gt_y * x_lt_y
        + /sreg0SBV_1.Q * /sreg0SBV_2.Q *
/sreg0SBV_0.Q * /init * x_eq_y *
          /x_gt_y
        + sreg0SBV_1.Q * sreg0SBV_2.Q * /init * start
...
```

2°) Les états sont mémorisés dans Sreg0. On rappelle que si l'on ne précise pas le codage des états c'est la même chose que de spécifier l'enum_encoding suivant :

```
1       type Sreg0_type is (S0, S1, S2, S3, sInit);
2       attribute enum_encoding of state:type is "000 001 010 011 100";
3       signal Sreg0: Sreg0_type;
```

Vers quel état va-t-on si on se trouve dans un autre état que S0, S1, S2, S3, sInit ?

(C'est la ligne "when others =>nul;" du programme VHDL.)

Réponse : vers S0

3°)Modifier au choix l'enum encoding ou le when others =>null; du programme pour que tous les états non spécifiés ramènent à l'état initial "sInit".

Réponses :

```
4        attribute enum_encoding of state:type is "001 010 011 100 000";
5        -- ou
6        when others =>sReg0 <= sInit;
```

4°) Réécrire les équations de récurrence en remplaçant sreg0SBV_0.Q par q0, ... sreg0SBV_2.Q par q2 et naturellement sreg0SBV_0.D par q_0^+ en utilisant les notations habituelles :

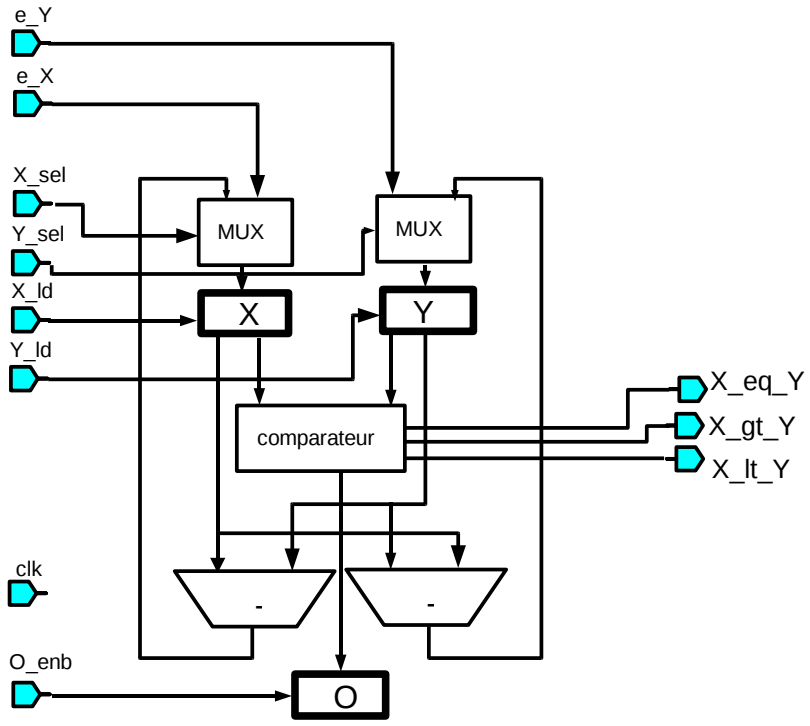
Réponses :

q_0^+ = Q1 * Q2 * /start + Q0 * /start + init

q_1^+ = /Q1 * /Q2 * /Q0 * /init * x_eq_y * /x_lt_y
+ /Q1 * /Q2 * /Q0 * /init * x_gt_y
+ Q1 * Q2 * /init * start

q_2^+ = /Q1 * /Q2 * /Q0 * /init * /x_gt_y * x_lt_y
+ /Q1 * /Q2 * /Q0 * /init * x_eq_y *
/x_gt_y
+ Q1 * Q2 * /init * start

5°) La partie opérative (ou chemin de données) est présentée maintenant

Partie opérative	*Explications*
	Tous les rectangles épais représentent des registres 8 bits, il y en a donc trois : X, Y et O. Les trapèzes représentent un soustracteur 8 bits réalisant respectivement la différence X-Y (à gauche) et Y-X (à droite). C'est donc une partie combinatoire. Le comparateur compare X et Y et positionne X_eq_Y à 1 si l'on a égalité, X_lt_Y à 1 si X<Y et X_gt_Y à 1 si X>Y. Facile à mémoriser quand on connait la notation anglaise gt signifie "greater than" lt "less than" eq equal. Le multiplexeur gauche sort X_i si X_sel=0 ou le résultat du soustracteur gauche si X_sel=1.

NOM :

Prénom :

Devoir surveillé MCENSL1 (suite)

Écrire la partie entité (seulement) décrivant la partie opérative.

Réponse :

```
ENTITY chemin_donnees IS
  port(     X_sel: in STD_LOGIC;
        Y_sel: in STD_LOGIC;
        X_ld: in STD_LOGIC;
        Y_ld: in STD_LOGIC;
      O_enb:in STD_LOGIC;
-- beaucoup d'étudiants on ratés les 8 bits ci-après !!!!!
      e_X: in STD_LOGIC_VECTOR(7 DOWNTO 0);
      e_Y: in STD_LOGIC_VECTOR(7 DOWNTO 0);
      X_lt_Y: out STD_LOGIC;
      X_gt_Y: out STD_LOGIC;
      X_eq_Y: out STD_LOGIC;
      clk: in STD_LOGIC);
end chemin_donnees;
```

6°) Les registres sont gérés par des process. On vous donne le process gérant le registre X et le multiplexeur qui le précède. On vous demande d'écrire celui qui gère le registre Y si la sortie du soustracteur droit s'appelle "s_soustr_droit".

process du registre X	*process du registre Y*

`-- gestion des registres`
`reg_X:process(clk)begin`
`  if clk'event and clk='0' then`
`    if X_ld='1' then`
`      if X_sel = '1' then`
`          regX <= s_soustr_gauche;`
`        else`
`          regX <= e_X;`
`        end if;`
`      else`
`        regX <= regX;`
`      end if;`
`  end if;`
`end process;`
	`-- gestion des registres`
	`reg_X:process(clk)begin`
	`  if clk'event and clk='0' then`
	`    if Y_ld='1' then`
	`      if Y_sel = '1' then`
	`          regY <= s_soustr_droit;`
	`        else`
	`          regY <= e_Y;`
	`        end if;`
	`      else`
	`        regY <= regY;`
	`      end if;`
	`  end if;`
	`end process;`

7°) C'est naturellement un front descendant qui fait évoluer le registre X. Précisez ce qui est chargé dans le registre lorsque X_ld='0'.

Réponse :

garde la même valeur car `regX <= regX;`

8°) A l'aide de la partie du programme suivant, on vous demande de trouver quelles sont les valeurs d'initialisation des registres X et Y ?

```
begin
  i1:chemin_donnees port map(X_sel=>X_sel,Y_sel=>Y_sel,X_ld=>X_ld,Y_ld=>Y_ld,s_O=>oregO,
                  s_X=>oregX,s_Y=>oregY,e_X=>"00001001",e_Y=>"00010001",
                  X_lt_Y=>X_lt_Y,X_gt_Y=>X_gt_Y,O_enb=>O_enb,X_eq_Y=>X_eq_Y,clk=>clk);
  i2:sequenceur port map(X_lt_Y=>X_lt_Y,X_gt_Y=>X_gt_Y,X_eq_Y=>X_eq_Y,clk=>clk,Init=>init,
                    start=>start,O_enb=>O_enb,X_sel=>X_sel,Y_sel=>Y_sel,X_ld=>X_ld,Y_ld=>Y_ld);
end top_arch;
```

Réponses :
X = 9 $(00001001)_2$
Y = 17 $(00010001)_2$

9°) Remplir la partie signal du programme réunissant le chemin de données et le séquenceur.

```
entity top is
  port(clk:in std_logic;
       init:in std_logic;
       start:in std_logic;
       oregX,oregY,oregO:out STD_LOGIC_VECTOR(7 DOWNTO 0));
end top;

architecture top_arch of top is
-- TOUS CES SIGNAUX étaient à trouver :
signal X_sel, Y_sel,X_ld,Y_ld,X_lt_Y,X_gt_Y,X_eq_Y,O_enb : std_logic;

component chemin_donnees
  port( X_sel: in STD_LOGIC;
        Y_sel: in STD_LOGIC;
        X_ld: in STD_LOGIC;
        Y_ld: in STD_LOGIC;
       O_enb:in STD_LOGIC;
       e_X: in STD_LOGIC_VECTOR(7 DOWNTO 0);
       e_Y: in STD_LOGIC_VECTOR(7 DOWNTO 0);
       s_O: out STD_LOGIC_VECTOR(7 DOWNTO 0);
       -- pour la simulation
       s_X: out STD_LOGIC_VECTOR(7 DOWNTO 0);
       s_Y: out STD_LOGIC_VECTOR(7 DOWNTO 0);
       X_lt_Y: out STD_LOGIC;
       X_gt_Y: out STD_LOGIC;
       X_eq_Y: out STD_LOGIC;
       clk: in STD_LOGIC);
end component;
component sequenceur
  port (X_lt_Y: in STD_LOGIC;
       X_gt_Y: in STD_LOGIC;
       X_eq_Y: in STD_LOGIC;
        CLK: in STD_LOGIC;
        Init: in STD_LOGIC;
        start: in STD_LOGIC;
        X_sel: out STD_LOGIC;
        Y_sel: out STD_LOGIC;
        X_ld: out STD_LOGIC;
        Y_ld: out STD_LOGIC;
              O_enb:out STD_LOGIC);
end component;
begin
  i1:chemin_donnees port map(X_sel=>X_sel,Y_sel=>Y_sel,X_ld=>X_ld,Y_ld=>Y_ld,s_O=>oregO,
                    s_X=>oregX,s_Y=>oregY,e_X=>"00000101",e_Y=>"00010000",
                    X_lt_Y=>X_lt_Y,X_gt_Y=>X_gt_Y,O_enb=>O_enb,X_eq_Y=>X_eq_Y,clk=>clk);
   i2:sequenceur port map(X_lt_Y=>X_lt_Y,X_gt_Y=>X_gt_Y,X_eq_Y=>X_eq_Y,clk=>clk,Init=>init,
                      start=>start,O_enb=>O_enb,X_sel=>X_sel,Y_sel=>Y_sel,X_ld=>X_ld,Y_ld=>Y_ld);
end top_arch;
```

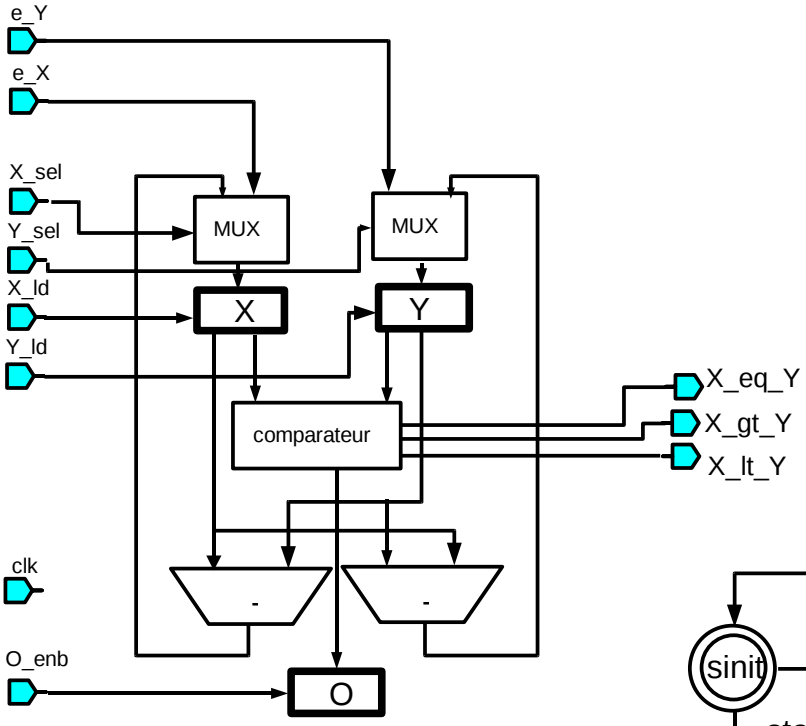
10°) On initialise le registre X à 5 et le registre Y à 16 pendant qu'on est dans l'état "sInit". Puis un petit coup de start nous emmène dans s0. Donner la trace de cet algorithme, c'est à dire les valeurs prises par les registres durant l'exécution du graphe d'état de la question 1 avec cette partie opérative.
Pour remplir : je parts de s0, comme X_lt_Y=1 je vais en S1 et comme alors Y_ld=1 alors je charge Y avec ce qui arrive quand Y_sel=1 soit Y-X=11. X est inchangé. Je déduis alors X-Y, Y-X, X_eq_Y, X_lt_Y et X_gt_Y.
Si le résultat se trouve dans le registre O, quel est le PGCD de 5 et de 16 ?

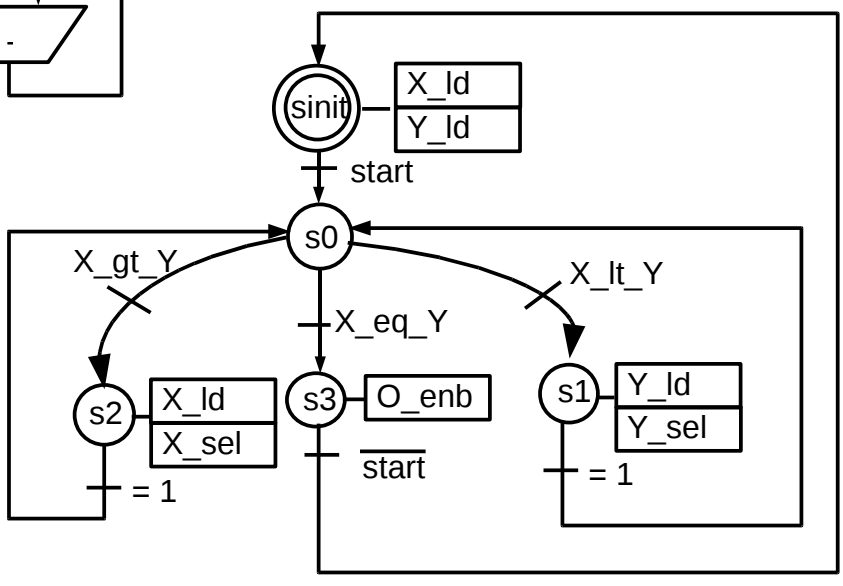
Réponse :(remplir le tableau en page suivante)
On vous a remis le graphe d'état ainsi que la partie opérative sur cette même page pour vous faciliter la réponse (vous éviter de tourner les pages).

NOM :
Prénom :

Devoir surveillé MCENSL1 (suite)

Etats du graphe	***sInit***	***s0***	***s1***	***s0***	***s1***	***s0***	***s1***	***s0***	***s2***	***s0***	***s2***	***s0***	***s2***	***s0***	***s2***	***s 0***	***s 3***	***s 3***
X	5	5	5	5	5	5	5	5	4	4	3	3	2	2	1	1	1	1
Y	16	16	11	11	6	6	1	1	1	1	1	1	1	1	1	1	1	1
O	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1
X-Y	-11	-11	-6	-6	-1	-1	4	4	4	3	3	2	2	1	1	0	0	0
Y-X	11	11	6	6	1	1	-4	-4	-4	-3	-3	-2	-2	-1	-1	0	0	0
X_eq_Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
X_lt_Y	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
X_gt_Y	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
O_enb	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1





Programme VHDL complet de correction et sa simulation :

```
library IEEE;
use IEEE.std_logic_1164.all;
--use work.pgcd.all;
entity top is
  port(clk:in std_logic;
       init:in std_logic;
       start:in std_logic;
       oregX,oregY,oregO:out STD_LOGIC_VECTOR(7 DOWNTO 0));
end top;

architecture top_arch of top is
signal X_sel, Y_sel,X_ld,Y_ld,X_lt_Y,X_gt_Y,X_eq_Y,O_enb : std_logic;
component chemin_donnees
  port( X_sel: in STD_LOGIC;
        Y_sel: in STD_LOGIC;
        X_ld: in STD_LOGIC;
        Y_ld: in STD_LOGIC;
              O_enb:in STD_LOGIC;
              e_X: in STD_LOGIC_VECTOR(7 DOWNTO 0);
              e_Y: in STD_LOGIC_VECTOR(7 DOWNTO 0);
              s_O: out STD_LOGIC_VECTOR(7 DOWNTO 0);
              -- pour la simulation
              s_X: out STD_LOGIC_VECTOR(7 DOWNTO 0);
              s_Y: out STD_LOGIC_VECTOR(7 DOWNTO 0);
              X_lt_Y: out STD_LOGIC;
              X_gt_Y: out STD_LOGIC;
              X_eq_Y: out STD_LOGIC;
              clk: in STD_LOGIC);
end component;
component sequenceur
  port (X_lt_Y: in STD_LOGIC;
              X_gt_Y: in STD_LOGIC;
              X_eq_Y: in STD_LOGIC;
        CLK: in STD_LOGIC;
        Init: in STD_LOGIC;
        start: in STD_LOGIC;
        X_sel: out STD_LOGIC;
        Y_sel: out STD_LOGIC;
        X_ld: out STD_LOGIC;
        Y_ld: out STD_LOGIC;
              O_enb:out STD_LOGIC);
end component;
begin
  i1:chemin_donnees port
map(X_sel=>X_sel,Y_sel=>Y_sel,X_ld=>X_ld,Y_ld=>Y_ld,s_O=>oregO,s_X=>oregX,s_Y=>oregY,
                    e_X=>"00000101",e_Y=>"00010000",X_lt_Y=>X_lt_Y,X_gt_Y=>X_gt_Y,O_enb=>O_enb,
                                      X_eq_Y=>X_eq_Y,clk=>clk);
  i2:sequenceur port map(X_lt_Y=>X_lt_Y,X_gt_Y=>X_gt_Y,X_eq_Y=>X_eq_Y,clk=>clk,Init=>init,

start=>start,O_enb=>O_enb,X_sel=>X_sel,Y_sel=>Y_sel,X_ld=>X_ld,Y_ld=>Y_ld);
end top_arch;



library IEEE;
use IEEE.std_logic_1164.all;
-- si warp
--use work.std_arith.all;
-- si aldec HDL-sim
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity chemin_donnees is
  port( X_sel: in STD_LOGIC;
        Y_sel: in STD_LOGIC;
        X_ld: in STD_LOGIC;
        Y_ld: in STD_LOGIC;
              O_enb: in STD_LOGIC;
              e_X: in STD_LOGIC_VECTOR(7 DOWNTO 0);
              e_Y: in STD_LOGIC_VECTOR(7 DOWNTO 0);
              s_O: out STD_LOGIC_VECTOR(7 DOWNTO 0);
              -- pour la simulation
              s_X: out STD_LOGIC_VECTOR(7 DOWNTO 0);
              s_Y: out STD_LOGIC_VECTOR(7 DOWNTO 0);
              X_lt_Y: out STD_LOGIC;
              X_gt_Y: out STD_LOGIC;
              X_eq_Y: out STD_LOGIC;
              clk: in STD_LOGIC);
```

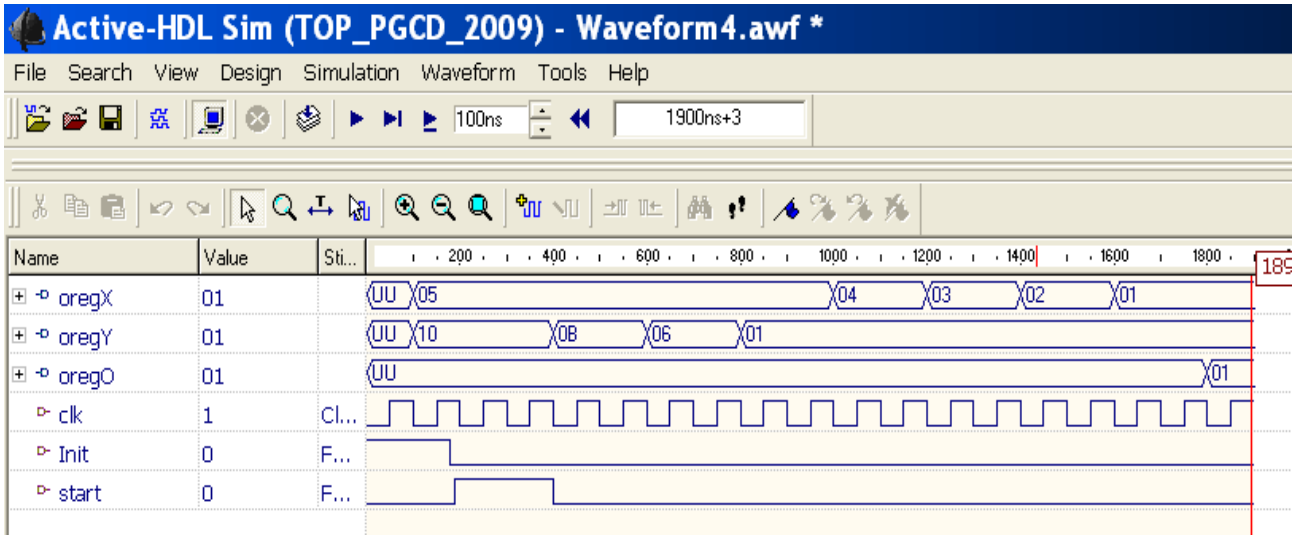
```
end chemin_donnees;

architecture chemin_donnees_arch of chemin_donnees is
-- declaration des registres
signal regX,regY,regO,s_comp,s_soustr_gauche,s_soustr_droite : STD_LOGIC_VECTOR(7 DOWNTO 0);
begin
-- gestion des registres
reg_X:process(clk)begin
  if clk'event and clk='0' then
    if X_ld='1' then
      if X_sel = '1' then
           regX <= s_soustr_gauche;
         else
           regX <= e_X;
         end if;
       else
         regX <= regX;
       end if;
  end if;
end process;
reg_Y:process(clk)begin
  if clk'event and clk='0' then
    if Y_ld='1' then
      if Y_sel = '1' then
           regY <= s_soustr_droite;
         else
           regY <= e_Y;
         end if;
       else
         regY <= regY;
       end if;
  end if;
end process;
reg_O:process(clk,O_enb)begin
  if clk'event and clk='0' then
    if O_enb='1' then
          regO <= s_comp;
       else
         regO <= regO;
       end if;
  end if;
end process;
-- partie combinatoire
comp:process(regX,regY)begin
  if regX < regY then
    X_lt_Y <='1';
  else
    X_lt_Y <='0';
  end if;
  if regX > regY then
    X_gt_Y <='1';
  else
    X_gt_Y <='0';
  end if;
  if regX = regY then
    X_eq_Y <='1';
       s_comp <= regX;
  else
    X_eq_Y <='0';
       s_comp <= (others =>'0');
  end if;
end process;

soustr_gauche:process(regX,regY)begin
  s_soustr_gauche <= regX - regY;
end process;

soustr_droite:process(regX,regY)begin
  s_soustr_droite <= regY - regX;
end process;

  s_X <= regX;
  s_Y <= regY;
  s_O <= regO;
end chemin_donnees_arch;


-- le mux est deja dans la gestion des registres X et Y
```

```
--
--  File: C:\Program Files\Aldec\Active-HDL Sim\PGCD_FSM_2009.vhd
-- created: 11/11/09 11:45:11
--  from: 'C:\Program Files\Aldec\Active-HDL Sim\PGCD_FSM_2009.asf'
--  by fsm2hdl - version: 2.0.1.45
--
library IEEE;
use IEEE.std_logic_1164.all;

entity sequenceur is
  port (CLK: in STD_LOGIC;
        init: in STD_LOGIC;
        start: in STD_LOGIC;
        X_eq_Y: in STD_LOGIC;
        X_gt_Y: in STD_LOGIC;
        X_lt_Y: in STD_LOGIC;
        O_enb: out STD_LOGIC;
        X_ld: out STD_LOGIC;
        X_sel: out STD_LOGIC;
        Y_ld: out STD_LOGIC;
        Y_sel: out STD_LOGIC);
end;

architecture pgcd_fsm_2009_arch of sequenceur is

-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S0, S1, S2, S3, sInit);
signal Sreg0: Sreg0_type;

begin
--concurrent signal assignments
--diagram ACTIONS;


Sreg0_machine: process (CLK)

begin

if CLK'event and CLK = '1' then
        if Init='1' then
                Sreg0 <= sInit;
        else
        case Sreg0 is
                when S0 =>
                        if X_gt_Y='1' then
                                Sreg0 <= S2;
                        elsif X_lt_Y='1' then
                                Sreg0 <= S1;
                        elsif X_eq_Y='1' then
                                Sreg0 <= S3;
                        end if;
                when S1 =>
                        Sreg0 <= S0;
                when S2 =>
                        Sreg0 <= S0;
                when S3 =>
                        if start='0' then
                                Sreg0 <= sInit;
                        end if;
                when sInit =>
                        if start='0' then
                                Sreg0 <= sInit;
                        elsif start='1' then
                                Sreg0 <= S0;
                        end if;
                when others =>
                        null;
        end case;
        end if;
end if;
end process;

-- signal assignment statements for combinatorial outputs
X_sel_assignment:
X_sel <= '0' when (Sreg0 = S0) else
         '0' when (Sreg0 = S1) else
         '1' when (Sreg0 = S2) else
```

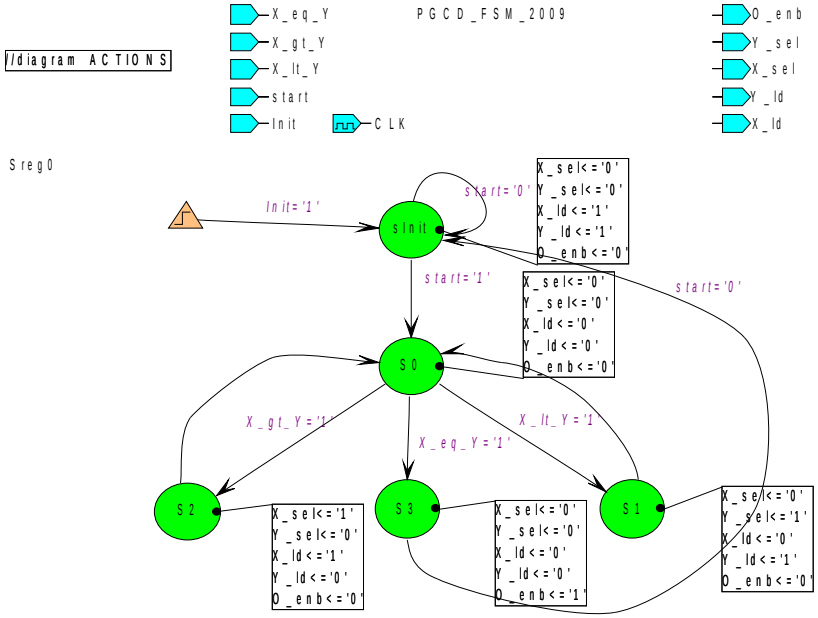
```
          '0' when (Sreg0 = S3) else
          '0';

Y_sel_assignment:
Y_sel <= '0' when (Sreg0 = S0) else
          '1' when (Sreg0 = S1) else
          '0' when (Sreg0 = S2) else
          '0' when (Sreg0 = S3) else
          '0';

X_ld_assignment:
X_ld <= '0' when (Sreg0 = S0) else
           '0' when (Sreg0 = S1) else
           '1' when (Sreg0 = S2) else
           '0' when (Sreg0 = S3) else
           '1';

Y_ld_assignment:
Y_ld <= '0' when (Sreg0 = S0) else
           '1' when (Sreg0 = S1) else
           '0' when (Sreg0 = S2) else
           '0' when (Sreg0 = S3) else
           '1';

O_enb_assignment:
O_enb <= '0' when (Sreg0 = S0) else
          '0' when (Sreg0 = S1) else
          '0' when (Sreg0 = S2) else
          '1' when (Sreg0 = S3) else
          '0';

end pgcd_fsm_2009_arch;
```

Trace de l'algorithme :

Lire OB (11) pour la deuxième valeur de oregY et non pas 08



Voici le graphe d'états.


//diagram ACTIONS
X_eq_Y
X_gt_Y
X_lt_Y
start
Init
CLK
PGCD_FSM_2009
O_enb
Y_sel
X_sel
Y_ld
X_ld
SregO
Init='1'
sInit
start='0'
X_sel<='0'
Y_sel<='0'
X_ld<='1'
Y_ld<='1'
O_enb<='0'
start='1'
S0
X_sel<='0'
Y_sel<='0'
X_ld<='0'
Y_ld<='0'
O_enb<='0'
start='0'
X_gt_Y='1'
X_eq_Y='1'
X_lt_Y='1'
S2
S3
S1
X_sel<='1'
Y_sel<='0'
X_ld<='1'
Y_ld<='0'
O_enb<='0'
X_sel<='0'
Y_sel<='0'
X_ld<='0'
Y_ld<='0'
O_enb<='1'
X_sel<='0'
Y_sel<='1'
X_ld<='0'
Y_ld<='1'
O_enb<='0'