VHDL

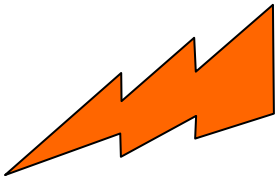
Instructions séquentielles
Logique séquentielle

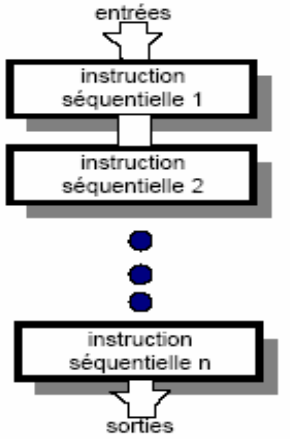
Logique séquentielle: le process

Le mot clé PROCESS

- Syntaxe:

```
MONETIQUETTE:process (signal1, signal2 etc)
-- zone déclarative
variable var1,var2: xxxxxxxx
Begin
 xxx
 xxx
 xxx
end process MONETIQUETTE;
```





- Le PROCESS est activé lors d'un changement d'état d'un des signaux de la liste de sensibilité
  - ✓ **process** (**signal1, signal2 etc**)
- Une fois dans le PROCESS, le déroulement est SEQUENTIEL
- Les instructions utilisables dans un PROCESS sont SPECIFIQUE (pas de when/else par exemple)
- Les signaux sont mis à jour uniquement à la fin du process

Écritures alternatives

```
process
begin
q <= d;
wait until HORLOGE = '1';
end process;
```

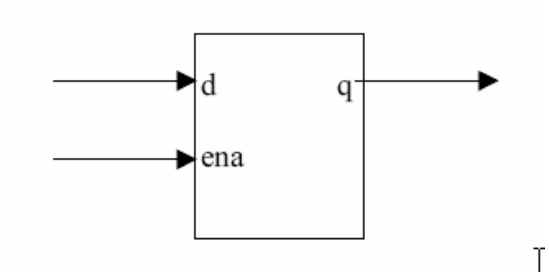
```
Process
begin
c <= a and b;
wait on a, b;
end process;
```

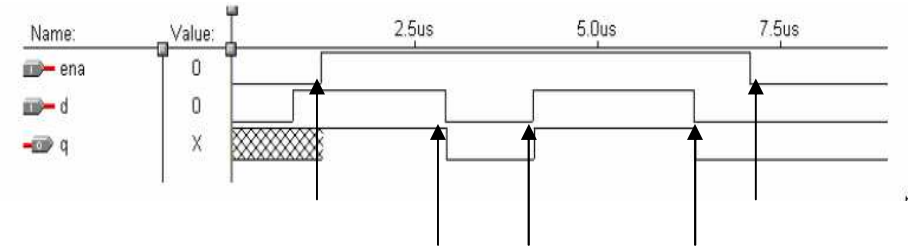
```
Process(a,b)
begin
c <= a and b;
end process;
```

Logique séquentielle: le process

Réveil du process

- Exemple: bascule D latch
  - ✓ **Fonctionnement sur niveau**





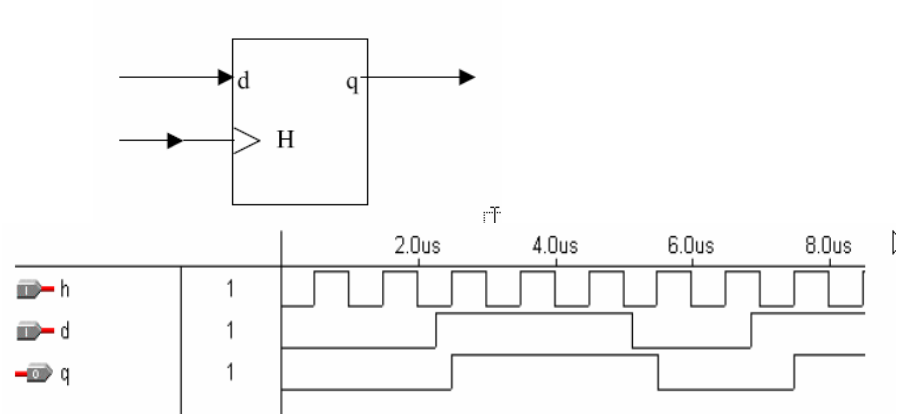
Processus activé

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY seq1 IS
        PORT (
              d,ena : IN STD_LOGIC;
              q: BUFFER STD_LOGIC
             );
END seq1;

ARCHITECTURE archi OF seq1 IS
BEGIN

PROCESS (ena,d)
BEGIN
IF ena = '1'   THEN q <= d;
               ELSE q <= q;
END IF;
END PROCESS;

END archi;
```

Liste de sensibilité

Logique séquentielle: le process

- **Rendre synchrone**
  - Bascule D edge: fonctionnement sur **front**



Autre façon:

```
Process(CLK)
Begin
If (CLK='1' AND CLK'event) then
  Q<=D;
End if;
End process;
```

DORENAVANT NOUS ADOPTERONS CETTE NOTATION SYSTEMATIQUEMENT EN COURS

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY seq2 IS
        PORT (
                d, h: IN STD_LOGIC;
                q:    OUT STD_LOGIC
               );
END seq2;

ARCHITECTURE archi OF seq2 IS
BEGIN
PROCESS
BEGIN
WAIT UNTIL h='1';
q <= d;
END PROCESS;

END archi;
```

Logique séquentielle: le process

Ecriture correcte des process

- Les compilateurs imposent une certaine rigidité dans la description des process
- Les règles a respecter

Chargement sur front :

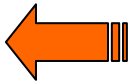
Un seul front
Une seule horloge

Pas de combinatoire avec l 'horloge

```
process(horl)
    if (horl'event and horl = '1') then
        contenu_registre <= valeur_in;
    end if;
    if (horl'event and horl = '0') then
        contenu_registre <= valeur_in;
    end if;
end process;
```

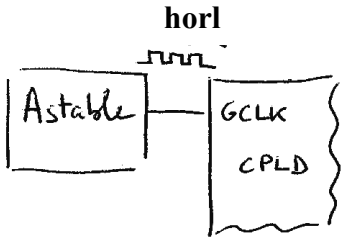
```
process(horl)
    if (horl'event and horl = '1'  and  ena = ' 1 ') then
        contenu_registre <= valeur_in;
    end if;
end process;
```

```
process(horl)
if (horl'event and horl = '1') then
   if (ena = ' 1 ') then
    contenu_registre <= valeur_in;
   end if;
  .....
  .....
end if;
end process;
```



Exemple de ce qu'il faut faire!

En pratique le FPGA (ou CPLD) possède une ou des broches spécifiques pour le signal d'horloge

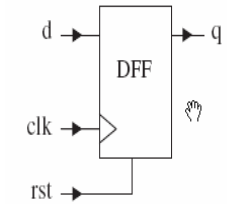


Logique séquentielle: le process

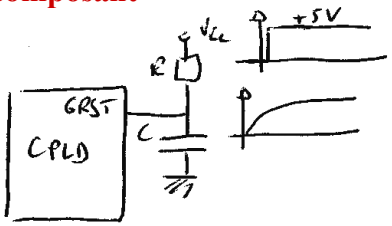
Penser à l'initialisation du systèmes

- Signaux reset et set: comportement défini par l'utilisateur

exemple



En pratique les FPGA (ou CPLD) possède une ou des broches spécifiques pour reset du composant



Reset et Set SYNCHRONE

On remarquera que RST a disparu de la liste de sensibilité

```
process (CLK)
Begin
if (CLK'event and CLK ='1') then
   if (RESET ='1') then
        S <= '0';
   elsif (SET ='1')then
        S <= '1';
   else
        S <= D;
end if;
end if;
end process ;
```

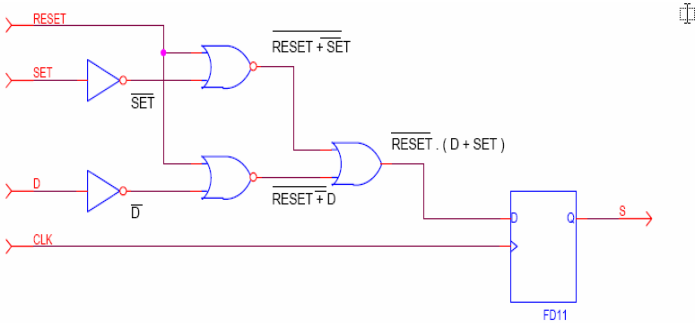
Actif à 1 ici pour l'exemple

```
LIBRARY ieee;
 USE ieee.std_logic_1164.all;
 ---------------------------------------
 ENTITY dff IS
 PORT ( d, clk, rst: IN STD_LOGIC;
 q: OUT STD_LOGIC);
 END dff;
 ---------------------------------------
 ARCHITECTURE behavior OF dff IS
 BEGIN
 PROCESS (rst, clk)
 BEGIN
 IF (rst='0') THEN
 q <= '0';
 ELSIF (clk'EVENT AND clk='1') THEN
 q <= d;
 END IF;
 END PROCESS;
END behavior;
```

Reset ASYNCHRONE

(ne dépend pas de clk)

et comportement **synchrone** de la bascule



Logique séquentielle: le process

- EN RESUME

- **DORENAVANT NOUS ADOPTERONS SYSTEMATIQUEMENT LES DEUX ECRITURES SUIVANTES**

Structure entièrement synchrone

```
process(clk)
if (clk'event and clk = '1') then
    .........
    .........
    .........
end if;
end process;
```

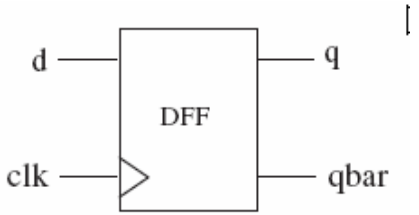
Structure synchrone avec reset asynchrone

```
PROCESS (rst, clk)
 BEGIN
 IF (rst='0') THEN
   .........
   .........
 ELSIF (clk'EVENT AND clk='1') THEN
   .........
   .........
END PROCESS;
```

Peut être remplacé par *rising_edge(clk)* ou *falling_edege(clk)*

Logique séquentielle: le process

- **Mise à jour des signaux à la fin du process**
  - Exemple 1: bascule avec sortie complémentée



---- Solution 1: **NE MARCHE PAS**----------

```
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY dff IS
6 PORT ( d, clk: IN STD_LOGIC;
7 q: BUFFER STD_LOGIC;
8 qbar: OUT STD_LOGIC);
9 END dff;
10 ---------------------------------------
11 ARCHITECTURE not_ok OF dff IS
12 BEGIN
13 PROCESS (clk)
14 BEGIN
15 IF rising_edge(clk) THEN
16 q <= d;
17 qbar <= NOT q;
18 END IF;
19 END PROCESS;
20 END not_ok;
21 ---------------------------------------
```

LIGNE 17

Je me fais avoir car si d a changé q ne changera qu'à la fin du process

---- Solution 2: **OK** ----------

```
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY dff IS
6 PORT ( d, clk: IN STD_LOGIC;
7 q: BUFFER STD_LOGIC;
8 qbar: OUT STD_LOGIC);
9 END dff;
10 ---------------------------------------
11 ARCHITECTURE ok OF dff IS
12 BEGIN
13 PROCESS (clk)
14 BEGIN
15 IF rising_edge(clk) THEN
16 q <= d;
17 END IF;
18 END PROCESS;
19 qbar <= NOT q;
20 END ok;
21 ---------------------------------------
```

LIGNE 19

Je décris un relation COMBINATOIRE => je sors du PROCESS!!!

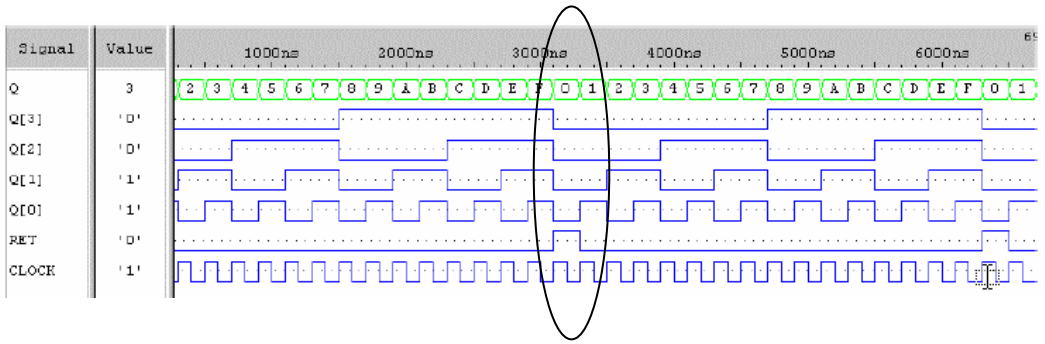
Logique séquentielle: le process

Mise à jour des signaux

- Cas des compteurs

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity CMP4BITSRET is
PORT (
RESET, CLOCK : in std_logic;
RET : out std_logic;
Q : out std_logic_vector (3 downto 0));
end CMP4BITSRET;
```

```
architecture DESCRIPTION of CMP4BITSRET is
signal CMP: unsigned (3 downto 0);
begin
process (RESET,CLOCK)
begin
if RESET ='1' then
CMP <= "0000";
elsif (CLOCK ='1' and CLOCK'event) then
CMP <= CMP + 1;
if (CMP = "1111") then
RET <= '1';
else
RET <= '0';
end if;
end if;
end process;
Q <= std_logic_vector (CMP);
end DESCRIPTION;
```



1110+1=1111 oui mais à la fin du process
Conclusion: etat 1111 et pas de RET ne passe pas à '1'!
Prochain front: 1111+1=0 je detecte 1111 , l'ancienne valeur et RET passe à 1
Oui mais trop tard!!

Logique séquentielle: le process

Des solutions

SOLUTION 1:

```
process (RESET,CLOCK)
begin
if RESET ='1' then
   CMP <= "0000";
elsif (CLOCK ='1' and CLOCK'event) then
   CMP <= CMP + 1;
    if (CMP = "1110") then
     -- la sortie RET passera à un quand CMP = 14
      --décimal
           RET <= '1';
    else
           RET <= '0';
    end if;
end if;
end process;
```

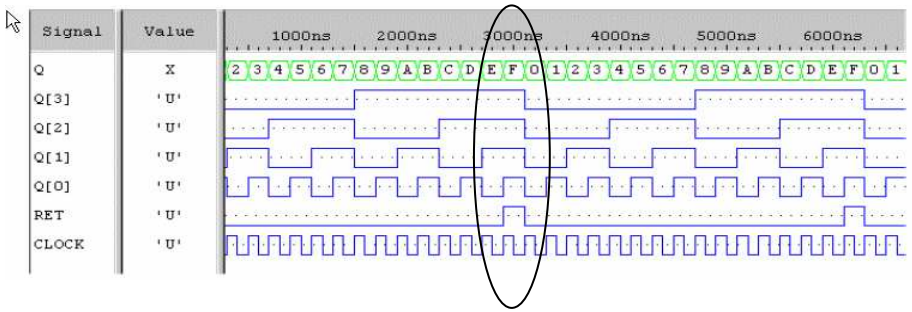
Version complètement synchrone:

J'anticipe pour avoir un résultat correct

Je décris le combinatoire HORS du PROCESS

SOLUTION 2:

```
process (RESET,CLOCK)
begin
if RESET='1' then
    CMP <= "0000";
elsif (CLOCK ='1' and CLOCK'event)
then
   CMP <= CMP + 1;
end if;
end process;

-- Validation de la retenue
RET <= '1' when (CMP = "1111") else '0';
```



Logique séquentielle: le process

Des solutions toujours

- Solution 3: EMPLOI DE VARIABLES

```
architecture DESCRIPTION of CMP4BITSRET is
-- le signal CMP a disparu
begin
process (RESET,CLOCK)
variable CMP: unsigned (3 downto 0);
begin
if RESET ='1' then
  CMP <= "0000";
elsif (CLOCK ='1' and CLOCK'event) then
   CMP := CMP + 1; -- ATTENTION
   if (CMP = "1111") then
      RET <= '1';
  else
     RET <= '0';
  end if;
end if;
Q <= std_logic_vector (CMP);
end process;
end DESCRIPTION;
```

On **peut** utiliser une variable à la place d'un signal

Affectation d'une variable

Mavariable:= ma valeur;

CONTRAIREMENT AU SIGNAUX LA VALEUR EST MISE A JOUR DE SUITE

En pratique pour vos opérations de comptage (compteur, diviseur de fréquence, timers et autres…) préférez les variables

CMP est déclaré au début du process. La portée de la variable est limitée à ce process.

L'instruction *Q <= std_logic_vector (CMP);* ***est déplacée à l'intérieur du process***

Logique séquentielle: les instructions

- **Assignations directes**
  - S<= signal;

Q est définie en sortie

OR Q<=Q+1 signifie « Relire état courant et incrémente »

Solution 1: définir BUFFER au lieu de OUT

Solution 2: couramment utilisée: passer par un signal ou une variable

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity CMP4BITS is
PORT (
CLOCK : in std_logic;
Q : out std_logic_vector (3 downto 0));
end CMP4BITS;

architecture DESCRIPTION of CMP4BITS is
signal Q_BUS_INTERNE : unsigned(3 downto 0));
begin
process (CLOCK)
begin
  if (CLOCK ='1' and CLOCK'event) then
     Q_BUS_INTERNE <= Q_BUS_INTERNE + 1;
  end if;
end process;

Q <= std_logic_vector (Q_BUS_INTERNE;)
 -- conversion et affectation du bus interne au
-- signal de sortie Q
end DESCRIPTION;
```

Nous aurions pu utiliser une variable.

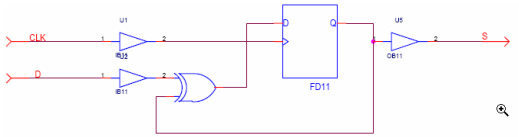
Voir les remarques des pages précédentes

Logique séquentielle: les instructions

Assignation conditionnelle

- Structure SI/SINON SI

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;
entity BASCULET is
port (
D,CLK : in std_logic;
S : buffer std_logic);
end BASCULET;

architecture DESCRIPTION of BASCULET is
begin
PRO_BASCULET : process (CLK)
Begin
if (CLK'event and CLK='1') then
    if (D='1') then
        S <= not (S);
    end if;
end if;
end process PRO_BASCULET;
end DESCRIPTION;
```



Bascule T

T comme TOGGLE (basculement)

La sortie change d'état à chaque front (utilisation pour la synthèse des compteurs)

Compteur de GRAY

Code binaire réfléchi

(codeur de position par exemple)

```
COMPTEUR GRAY
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.std_arith.all;
entity GRAY is
port (H,R :in std_logic;
Q :out std_logic_vector(2 downto 0));
end GRAY;
architecture ARCH_GRAY of GRAY is
signal X :std_logic_vector(2 downto 0);
begin
process(H,R)
begin
if R='1' then X <= "000";
elsif (H'event and H='1') then
if X = "000" then X <= "001";
elsif X = "001" then X <= "011";
elsif X = "011" then X <= "010";
elsif X = "010" then X <= "110";
elsif X = "110" then X <= "111";
elsif X = "111" then X <= "101";
elsif X = "101" then X <= "100";
elsif X = "100" then X <= "000";
end if;
end if;
end process;
Q <= X;
end ARCH_GRAY;
```

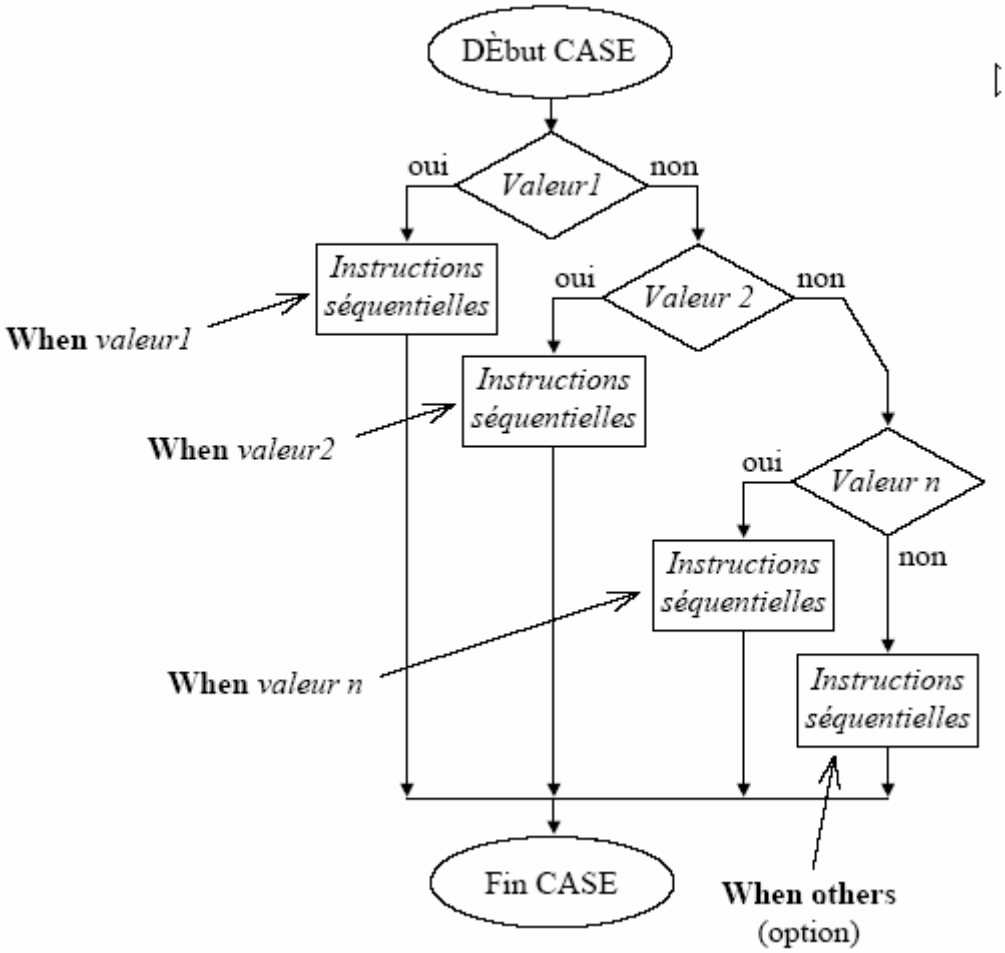
Logique séquentielle: les instructions

Assignation conditionnelle

- Structure CASE/IS
  - ✓ Utile pour décrire des grafcets

```
Case selecteur  is
when   condition1 =>          instructions ;
                              …….
                              instructions ;
when   condition2 =>          instructions ;
----
when others       =>          instructions ;
end case ;
```

CAS *possibles de l'expression* **EST**
LORSQUE *signal = valeur1* **=>** *instructions séquen*
LORSQUE *signal = valeur2* **=>***instructions séquen*
LORSQUE *signal = valeur3* **=>***instructions séquen*
LORSQUE *signal* = **AUTRES =>***instructions séque*
FIN DE CAS;



Logique séquentielle: les instructions

- Exemple de CASE/IS
  - ✓ Description d'un registre à décalage **à droite OU à gauche avec chargement parrallèle**

```
PROCESS (h)
VARIABLE stmp: std_logic_vector(3 DOWNTO 0);
BEGIN
If (h='1' and h'event) then
CASE selection IS
        WHEN "11"=> stmp := d_entree; --chargement paralelle
        WHEN "10"=>stmp:= stmp(2 DOWNTO 0) & edg; --gauche
        WHEN "01"=>stmp:= edd &stmp(3 DOWNTO 1); --droite
        WHEN OTHERS =>  ; --mémorisation
END CASE;
sortie <= stmp;
END PROCESS Reg_dec;
```

On **peut** utiliser une variable à la place d'un signal

Affectation d'une variable

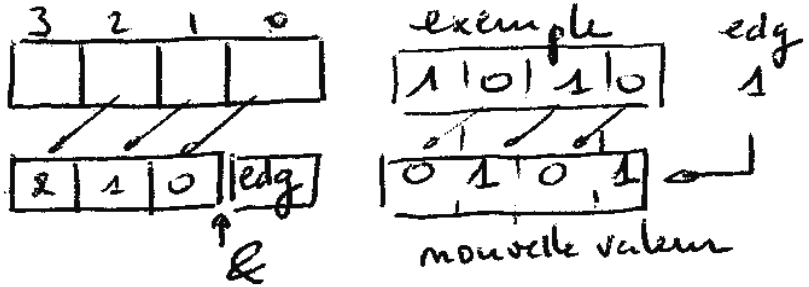
Mavariable:= ma valeur;

CONTRAIREMENT AU SIGNAUX LA VALEUR EST MISE A JOUR DE SUITE

Rappel: & « colle » les signaux ensemble

Cas où sel=« 10 »

DECALAGE à GAUCHE



RAPPEL avant les exemples

Complément sur les opérations arithmétiques

- Le rajout de use IEEE.numeric_std.all; permet
  - ✓ De travailler avec des valeurs signées et non signées
    - signal A,B: signed(3 downto 0);
    - signal tempo: unsigned(3 downto 0);
  - ✓ De convertir un std_logic_vector en signed ou unsigned
    - A<= signed(SW(3 downto 0));
    - B<=unsigned(RES);
  - ✓ De convertir des signed ou unsigned en std_logic_vector
    - LEDG(3 downto 0)<=std_logic_vector(tempo);
  - ✓ De redimensionner des vecteurs
    - Permet d'étendre le bit de signe correctement!
    - signal A,B: signed(LARG downto 0);
      A<= resize(signed(SW(LARG downto 1)),LARG+1);
  - ✓ De travailler avec les opérateurs arithmétiques standart
    - >, >=, =<,<, + ,- etc….
- Le rajout de use IEEE.std_logic_unsigned.all; permet
  - ✓ De travailler avec les opérateurs arithmétiques standard
  - ✓ de mélanger des entiers avec des std_logic_vector: A<= A +1;

IEEE.std_logic_unsigned.all et IEEE.std_logic_arith.all sont d'anciennes bibliothèques

Ne pas mettre en même temps:

IEEE.numeric_std.all;

IEEE.std_logic_arith.all;

Préfèrez l'emploi de IEEE.numeric_std.all;

Alternative à resize

A<=resize(signed(SW(LARG downto 1)),LARG+1);

Recopie du bit bit de poids forts

A<= A(3)&A

Logique séquentielle: des exemples

Des compteurs

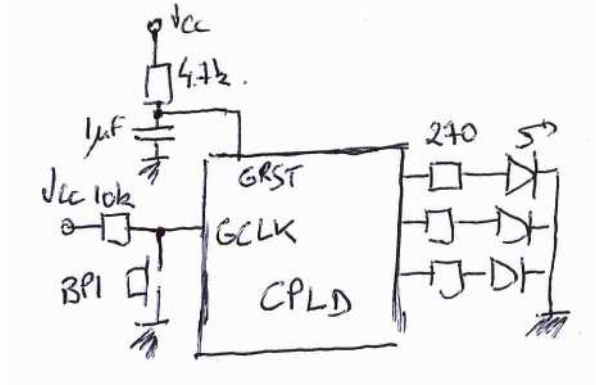
- Pour faire quoi?
  - ✓ Compter des événements
    - Exemple: signal d'horloge généré par un bouton poussoir et on compte le nombre d'appui
  - ✓ Pour mesurer un temps
    - Horloge stable délivré par un astable ou oscillateur (quartz)
    - Le nombre de coups nous renseigne sur le laps de temps écoulé

→ Fonction monostable

→ Fonction diviseur de fréquence

→ Fonction base de temps

Exemple 1: Compter des événements




```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity CMP3BITS is
PORT (
RESET, CLOCK : in std_logic;
Q : out std_logic_vector (2 downto 0));
end CMP3BITS;
```

```
architecture archCMP3BITS of CMP3BITS is
begin
process (CLOCK)
Variable Q_BUS_INTERNE : unsigned(2 downto 0));
begin
  if (CLOCK ='1' and CLOCK'event) then
     Q_BUS_INTERNE := Q_BUS_INTERNE + 1;
  end if;
Q <= std_logic_vector (Q_BUS_INTERNE );
end process;
end archCMP3BITS;
```

Logique séquentielle: des exemples

Exemple 1 bis

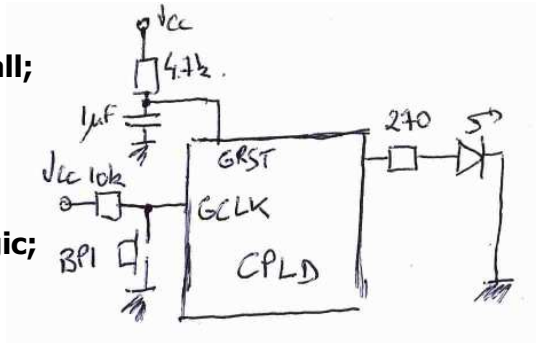
- Compter des événements
  - ✓ version avec integer

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity CMP3BITS is
PORT (
RESET, CLOCK : in std_logic;
Q : out std_logic_vector (2 downto 0));
end CMP3BITS;


architecture archCMP3BITS of CMP3BITS is
begin
process (CLOCK)
variable Q_BUS_INTERNE : integer range 0 to 7;
begin
  if (CLOCK ='1' and CLOCK'event) then
     Q_BUS_INTERNE := Q_BUS_INTERNE + 1;
  end if;
    Q <= std_logic_vector (Q_BUS_INTERNE );
end process;
end archCMP3BITS;
```

Exemple 2

- Détecter un événement

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

entity DETECT is
PORT (
RESET, CLOCK : in std_logic;
S1 : out std_logic);
end DETECT ;
```



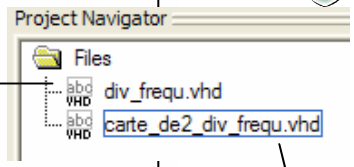

```
architecture archCMP3BITS of CMP3BITS is
begin
process (CLOCK)
variable COMPT : integer range 0 to 7;
-- unsigned(…), std_logic_vector(…) marche aussi!
begin
  if (CLOCK ='1' and CLOCK'event) then
     COMPT <= COMPT + 1;
     if (COMPT=3) then
        S1<='1';
     else
        S1<='0';
  end if;
end process;
end archCMP3BITS;
```

Peut-être mis en dehors du process car COMBINATOIRE

Logique séquentielle: des exemples

Exemple 3: Diviseur de fréquence

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity div_frequ is
6  PORT(
7        clk_in: in std_logic;
8        clk_div: out std_logic);
9  end entity div_frequ;
10
11 architecture arch_div_frequ of div_frequ is
12 BEGIN
13 PROCESS (clk_in)
14 variable cpt :integer range 0 to 7;
15 BEGIN
16 IF (clk_in'EVENT AND clk_in='1') THEN
17      cpt := cpt + 1;
18      IF (cpt=7) THEN
19           cpt := 7;
20           clk_div <= '1';
21      ELSE
22           clk_div <= '0';
23      END IF;
24 END IF;
25 END PROCESS;
26
27 END architecture arch_div_frequ;
```



Préferez des variables aux signaux

La manipulation des *integer* est plus souple que les *unsigned*

Comment utiliser le composant diviseur de fréquence

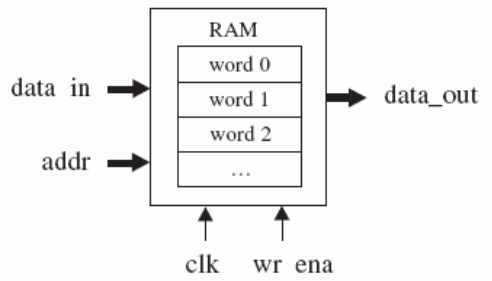
- On évitera de cascader la sortie du diviseur sur l'horloge du bloc suivant
- La bonne méthode:
  - Même horloge pour tous le monde
  - La sortie du diviseur est une entrée de validation du bloc suivant

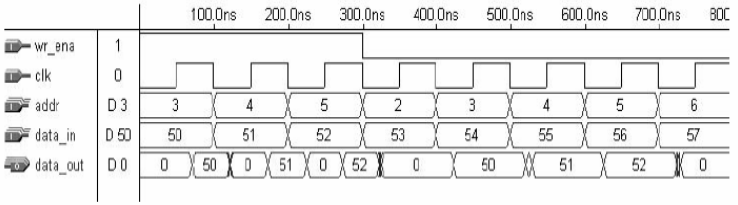
Exemple d'utilisation du composant div-freq

```
29 signal Fil : std_logic;
30 signal Del : std_logic;
31
32 Begin
33 TD_RESET <= '1';
34 U0: div_frequ port map(clk_in =>CLOCK_27 , clk_div => Fil);
35 process(CLOCK_27)
36 Begin
37 if (CLOCK_27'EVENT AND CLOCK_27='1') THEN
38     if(Fil='1') then
39         Del <= NOT Del; -- change etat led
40     end if;
41 end if;
42 End process;
43 LEDR(0)<=Del ;
```

Logique séquentielle: des exemples

Une RAM






```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
 ---------------------------------------------------
ENTITY ram IS
GENERIC ( bits: INTEGER := 8; -- # of bits per word
                words: INTEGER := 16); -- # of words in the memory
PORT ( wr_ena, clk: IN STD_LOGIC;
                 addr: IN INTEGER RANGE 0 TO words-1;
                data_in: IN STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
                data_out: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
12 END ram;
 ---------------------------------------------------
ARCHITECTURE ram OF ram IS
TYPE vector_array IS ARRAY (0 TO words-1) OF STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
SIGNAL memory: vector_array;
BEGIN
PROCESS (clk)
BEGIN
IF (clk'EVENT AND clk='1') THEN
                 IF (wr_ena='1') THEN
                                memory(addr) <= data_in;
                END IF;
END IF;
END PROCESS;
 data_out <= memory(addr);
END ram;
```

Création d'un nouveau type: TYPE

C'est un tableau de vecteurs

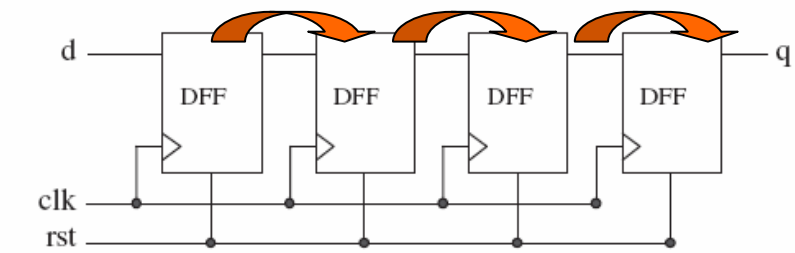
Déclaration d'un signal du type créé précédemment

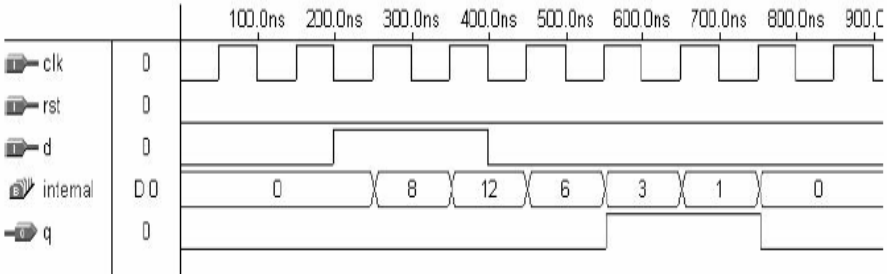
Pas de combinatoire avec l 'horloge

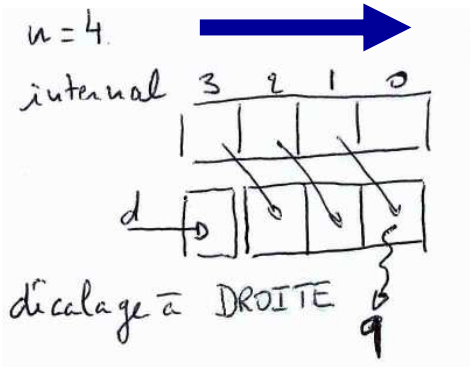
```
process (horl)
    if (horl'event and horl = '1'  and  ena = ' 1 ') then
        contenu_registre <= valeur_in;
    end if;
end process;
```

Logique séquentielle: des exemples

- **Registre à décalage simple**
  - Sortie série (1 seule sortie)








```
1 ------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------------
ENTITY shiftreg IS
6 GENERIC (n: INTEGER := 4);
7 PORT (d, clk, rst: IN STD_LOGIC;
8 q: OUT STD_LOGIC);
9 END shiftreg;
10 ------------------------------------------------
11 ARCHITECTURE behavior OF shiftreg IS
12 SIGNAL internal: STD_LOGIC_VECTOR (n-1 DOWNTO 0);
13 BEGIN
14 PROCESS (clk, rst)
15 BEGIN
16 IF (rst='1') THEN
17                internal <= (OTHERS => '0');
18 ELSIF (clk'EVENT AND clk='1') THEN
19                internal <= d & internal(internal'LEFT DOWNTO 1);
20 END IF;
21 END PROCESS;
22 q <= internal(0);
23 END behavior;
24 ------------------------------------------------
```

Logique séquentielle: des exemples

- **Registre à décalage double sens**
  - Illustration du mode *inout*
  - Choix du sens

Emploi de signaux et non de variables

Affectation en dehors du process

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity DECAL_DG is
port (
                H,R,SENS :in std_logic;
                IN_OUT,OUT_IN :inout std_logic);
end DECAL_DG;

architecture ARCH_DECAL_DG of DECAL_DG is
signal Q :std_logic_vector(3 downto 0);
begin
process(H,R)
begin
if R='1' then Q <= "0000";
elsif (H'event and H='1') then
                if SENS = '1'  then
                      Q <= Q(2 downto 0) & IN_OUT;
                else Q <= OUT_IN & Q(3 downto 1);
                end if;
end if;
end process;
OUT_IN <= Q(3) when SENS = '1' else 'Z';
IN_OUT <= Q(0) when SENS = '0' else 'Z';
end ARCH_DECAL_DG;
```

Logique séquentielle: des exemples

Détection d'un front

- Détection d'un changement d'état d'un signal
- Contrainte: DESCRIPTION SYNCHRONE

```
process (CLOCK)
VARIABLE detect : std_logic_vector(1 DOWNTO 0);
begin
if (CLOCK ='1' and CLOCK'event) then
          front_montant <= '0';
          front_descendant <= '0' ;
         detect(1) := detect(0);
         detect(0) := signal_lent;
         IF detect = "01" THEN
                  front_montant <= '1';
         END IF;
         IF detect = "10" THEN
                  front_descendant <= '1';
        END IF;
end if;
END PROCESS;
```

La détection de front porte sur le signal signal_lent (par exemple bouton poussoir)

Il faut donc que la fréquence système CLOCK soit beaucoup plus grande que signal_lent.

Rappel:

la variable prend sa valeur instantanément

Le signal prend sa valeur à la sortie du process

Bibliographie

- Certaines illustrations et exemples proviennent de cours ou d'ouvrages présents ci-dessous

  - ✓ Introduction à la Synthèse logique **Philippe LECARDONNEL & Philippe LETENNEUR**
  - ✓ Le langage de description VHDL **T. BLOTIN**
  - ✓ VHDl **J.maillefert**
  - ✓ Circuit Design with VHDL **Volnei A. Pedroni**

- **Un lien bien utile pour l'étudiant GEII**
  - ✓ **http://perso.orange.fr/xcotton/electron/coursetdocs.htm**