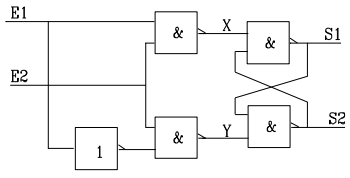
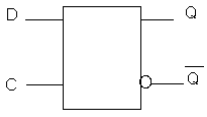
C4 – Les instructions séquentielles

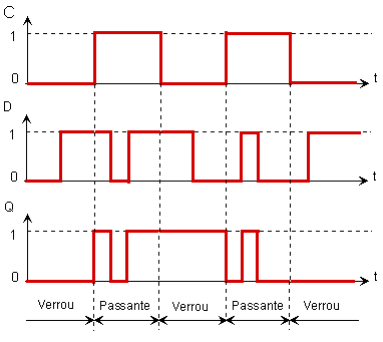
Yann DOUZE
VHDL

Rappel : Bascule D Latch

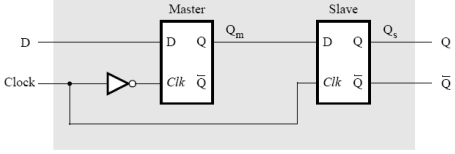


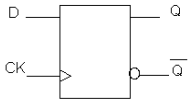


Entrées		Sorties	
C	D	Qn+1	$\overline{Qn+1}$
0	X	Qn	$\overline{Qn}$
1	0	0	1
1	1	1	0

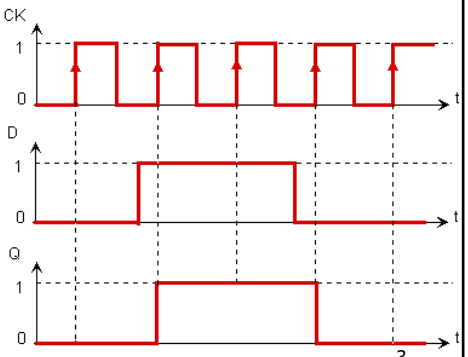


Rappel : Bascule D Flip-Flop





Entrées		Sorties	
CK	D	Qn+1	$\overline{Qn+1}$
0	X	Qn	$\overline{Qn}$
1	X	Qn	$\overline{Qn}$
↓	X	Qn	$\overline{Qn}$
↑	0	0	1
↑	1	1	0



3

3

Définition du process

- Un **process** est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement : les unes à la suite des autres.
- Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standard de la programmation structurée comme dans les systèmes à microprocesseurs.

```
Syntaxe:
[Nom_du_process
   :]process(Liste_de_sensibilité_nom_des_signaux)
Begin
   -- instructions du process
end process [Nom_du_process] ;
```

4

4

Règles de fonctionnement d'un process

- L'exécution d'un ***process*** a lieu à chaque changement d'état d'un signal de la liste de sensibilité.
- Les instructions du ***process*** s'exécutent séquentiellement.
- Les changements d'état des signaux par les instructions du ***process*** sont pris en compte à la **fin** du ***process***.

Role des processus

- Les processus sont utilisé avec 3 approches différentes :

1. Pour décrire un circuit combinatoire avec des instructions évolué de la programmation structuré : if, case, etc…
2. Pour décrire un circuit synchrone qui comporte une ou plusieurs cellules de mémorisation (bascules Dff).
3. Pour décrire une fonction non synthétisable tel qu'un banc de test ou un modèle.

Instructions séquentielles

- Attention !
- Les instructions **if** et **case** n'existent et ne peuvent exister que dans un process.

Instruction if

```
Syntaxe :
if condition then instructions
[elsif condition then instructions]
[else instructions]
end if ;

Exemple:
Process (A,B,E1,E2,E3)
Begin
   if  A='1'  then    SORTIE <= E1;
   Elsif B = '1' then SORTIE <= E2;
   Else SORTIE <= E3;
   end if ;
end process ;

Remarque : l'instruction if ne peut être utilisée que dans un
   process
```

Instruction case

Syntaxe:

CASE expression IS
WHEN choix => instructions;
[WHEN choix => instructions;]
[WHEN OTHERS => instructions;]
END CASE;

Remarque : pareil que l'instruction if, l'instruction case ne peut être utlisée que dans un process.

Exemple

```
process (TEST,A,B)
begin
  case TEST is
   when "00" => F <= A and B;
   when "01" => F <= A or B;
   when "10" => F <= A xor B;
   when "11" => F <= A nand B;
   when others => F <= '0';
  end case;
end process;
```

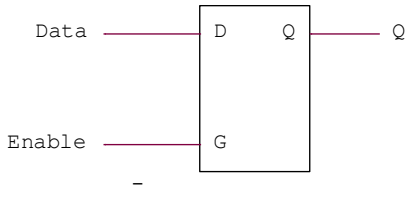
Process combinatoire

```
process (SEL,A,B,C)
begin
  if SEL ='1' then
    OP <= A and B;
  else
    OP <= C;
  end if;
end process;
```

- La liste de sensibilité doit être complète : tous les signaux lus dans le process doivent apparaître dans la liste de sensibilité.
- les sorties doivent être assignés dans tous les cas afin d'éviter les bascules D Latch.

Affectation incomplète

```
process (Enable,Data)
begin
  if Enable = '1' then
      Q <= Data;
  end if;
end process;
```



• Une affectation incomplète génère une bascule D Latch (verrou) non désiré (transparent latch)

• Certain FPGA n'ont pas de bascule D Latch (verrou), une boucle asynchrone sur du combinatoire est alors créé (interdit!)

Assignement par défaut

• Pour éviter les transparents Latch on préconise de faire un assignement par défaut.

```
Process (SELA, SELB, A, B)
begin
   OP <= '0';
   if SELA = '1' then
       OP <= A;
   end if;
   if SELB = '1' then
       OP <= B;
   end if;
end process;
```

Assignement par défaut

Remplace l'événement par défaut

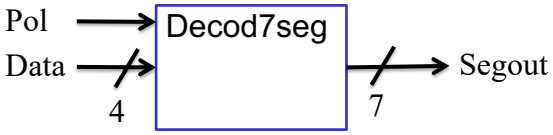
Remplace l'événement à nouveau

Liste de sensibilité complète

- Rajout du VHDL 2008
- -> process(all)

```
Process(all)
begin
   OP <= '0';
   if SELA = '1' then
       OP <= A;
   end if;
   if SELB = '1' then
       OP <= B;
   end if;
end process;
```

Exemple : décodeur 7 segments




```
Entity SEVEN_SEG is
port(
 Data : in  std_logic_vector(3 downto 0); --Expected within 0…9
 Pol  : in  std_logic;                    -- '0' if active LOW
 Segout: out std_logic_vector(1 to 7)); --Segments A,B,C,D,E,F,G
end entity;
```


Architecture combinatoire du décodeur 7 segments

```
Architecture COMB of SEVEN_SEG is
  signal sevseg : std_logic_vector(1 to 7);
begin
Process(Data, Pol,sevseg)
Begin
  case(Data) is
    when x”0”=> sevseg <= “1111110”;
    when x"1" => sevseg <= "0110000";
    when x"2" => sevseg <= "1101101";
    when x"3" => sevseg <= "1111001";
    when x"4" => sevseg <= "0110011";
    when x"5" => sevseg <= "1011011";
    when x"6" => sevseg <= "1011111";
    when x"7" => sevseg <= "1110000";
    when x"8" => sevseg <= "1111111";
    when x"9" => sevseg <= "1111011";
    when others => sevseg <= (others => '-');
  end case;
  if (Pol='1') then
    Segout <= sevseg;
  else
    Segout <= not(sevseg);
  end if;
end process;
end architecture;
```

Exercice

- Exercice 1 : Multiplexeur 8 vers 1

Process synchrone

```
process(CLK)
begin
if RISING_EDGE(CLK) then
   Q1 <= D;
end if;
end process;
```

```
process(CLK)
begin
if FALLING_EDGE(CLK) then
   Q2 <= D;
end if;
end process;
```

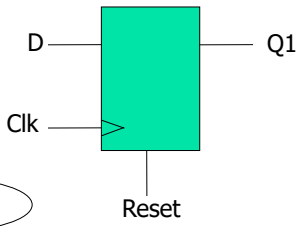
- **Un process synchrone est exécuté à chaque front montant de l'horloge**.
- Pour tester le front montant de l'horloge, on utilise la fonction rising_edge() qui est définit dans le package STD_LOGIC_1164.
- RISING_EDGE est VRAI lorsque l'horloge passe de l'état '0' à l'état '1'.
- Utile pour décrire une bascule D flip-flop (Dff).

Comment rajouter un reset asynchrone?

Reset Asynchrone

```
process(Reset,Clk)
begin
if Reset = '1' then
   Q1 <= '0';
elsif RISING_EDGE(Clk) then
   Q1 <= D;
end if;
end process;
```

- Le reset asynchrone est testé avant le front montant de l'horloge.

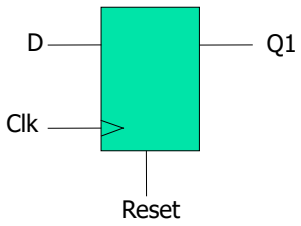


Comment faire un reset synchrone?

Reset Synchrone

```
process(Clk)
begin
if RISING_EDGE(Clk) then
   if Reset = '1' then
    Q1 <= '0';
   else
    Q1 <= D;
   end if;
end if;
end process;
```

- Le reset synchrone est testé après le front montant de l'horloge.



Transfert par registre (RTL)

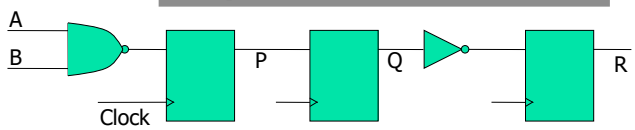
- Dans un process synchrone, à chaque affectation d'un signal, une bascule est créée.

```
process (Clock)
begin
   if Rising_edge(Clock) then
      P <= A nand B;
      Q <= P;
      R <= not Q;
   end if;
end process;
```

Clock

Registres

Logique combinatoire



Règles des process synchrones

- Liste de sensibilité doit inclure le clock et le reset et rien d'autre.
- Tout les signaux assignés dans le process doivent être initialisés par un reset.
- Pour la description d'un process synchrone, respectez la structure suivante :

```
process(clk,rst)
begin
if rst = '1' then
 -- valeur initiale
elsif rising_edge(clk) then
 -- instructions
end if;
end process;
```

- Attention les signaux prennent leurs nouvelles valeurs qu'à la fin du process.
- Complétez le code et dessinez le chronogramme d'un registre à décalage à 4 étages ?

Registre à décalage

```
-- Registre à décalage
entity registre_decalage is
port( D1, rst, clk : in std_logic;
      Q4           : out std_logic);
end entity;
architecture RTL of registre_decalage is
    signal Q1,Q2,Q3 : std_logic;
begin
process(Rst,Clk)
begin
if Rst = '1' then
    Q4 <= '0'; Q1<='0';Q2<='0';Q3<='0';
Elsif rising_edge(clk) then
    Q4 <= Q3;
    Q3 <= Q2;
    Q2 <= Q1;
    Q1 <= D1;
End if;
End process;
End architecture;
```

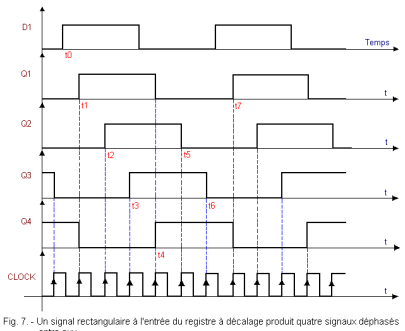


Fig. 7. - Un signal rectangulaire à l'entrée du registre à décalage produit quatre signaux déphasés entre eux.

Mauvais style de process (1) ?

```
process(Clock,Reset)
Begin
      if Rising_edge(Reset) then
            ...
      elsif Rising_edge(Clock) then
            ...
      end if;
end process;
```

```
process(Clock,Reset,Ena)
begin
      if Reset = '1' then
            -- Actions asynchrone
      elsif Rising_edge(Clock) and Ena = '1' then
            -- Actions synchrones
      end if;
end process;
```

```
process(Clock,Reset)
begin
      if Reset = '1' then
            -- Actions asynchrone
      elsif Rising_edge(Clock) then
            -- Actions synchrones
      end if;
end process;
```

```
process(All_Inputs)
begin
 -- Logique purement combinatoire
end process;
```

25

Mauvais style de process (2) ?

```
process(Clock,Reset)
begin
      if Reset = '1' then
            -- Actions asynchrone
      elsif Rising_edge(Clock) then
            -- Actions synchrones
      end if;
            -- d'autres actions
end process;
```

```
process(Clock)
begin
      if Rising_edge(Clock) then
            -- Actions synchrones
      end if;
end process;
```

```
process(Clock,Reset)
begin
      if Reset = '0' then
            if Rising_edge(Clock) then
                  -- Actions synchrones
            end if;
      else
            -- Actions asynchrones
      end if;
end process;
```

26

Exercice

Exercice 2 : serial OR ou LFSR

Process non-synthétisable

- Les process non synthétisable sont utilisé pour :
  - décrire un modèle
  - écrire des bancs de test (test bench)
- Les process non synthétisable **ne possèdent pas de liste de sensibilité** et ce sont des **instructions WAIT** qui synchronisent le process.
- Trois formes de WAIT peuvent être combinées
  - WAIT ON *événement*; Exemple : wait on A,B,C,D
    - Remplace la liste de sensibilité d'un process classique
  - WAIT FOR *durée*; Exemple : wait for 100 ns
    - Permet de créer un délais
  - WAIT UNTIL *condition*; Exemple : wait until rst = '1'
    - Condition bloquante
  - WAIT;
    - Boucle infini, comme un while(1) en C

Exemples de Process non-synthétisable

```
-- Génération d'un reset
STIMULUS: process
begin
   reset <= '1';
   wait for 50 NS;
   reset <= '0';
   wait;
end process STIMULUS;
```

```
-- Génération d'une horloge
ClockGenerator: process
begin
Clock <= '0';
wait for 5 NS;
Clock <= '1';
wait for 5 NS;
end process;
```


Instructions de boucle

```
for PARAMETER in LOOP_RANGE loop
   ...
end loop;
```

```
While CONDITION loop
   ...
end loop;
```

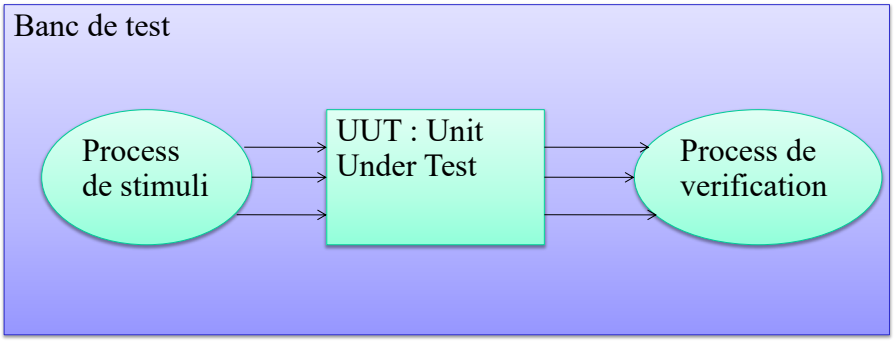
```
boucle1: -- étiquette optionnelle
FOR i IN 0 TO 10 LOOP
  -- calcul des puissances de 2
  b := 2**i;
  WAIT FOR 10 ns; --toutes les 10 ns
END LOOP;
```

```
I := 0;
WHILE b < 1025 LOOP
   b := 2**i;
   i := i+1;
   WAIT FOR 10 ns;
END LOOP;
```

Exemple de boucle

```
ClockGenerator: process
begin
  while not Stop loop
    Clock <= '0';
    wait for 5 NS;
    Clock <= '1';
    wait for 5 NS;
  end loop;
  wait;
end process;
```


Banc de test (Test Bench)

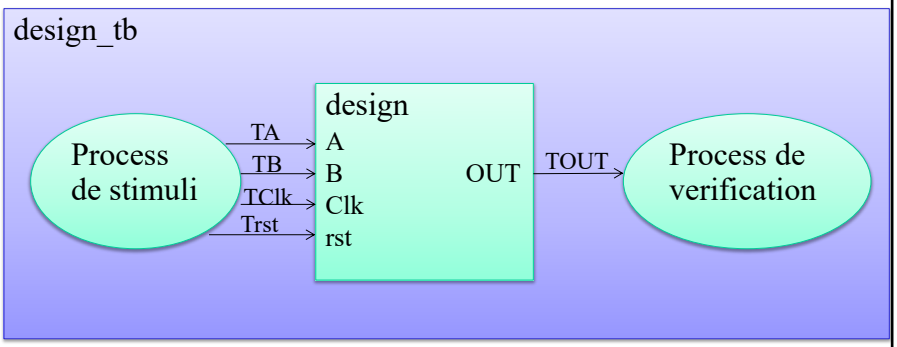


Verification = Assertion

- **ASSERT** : permet d'écrire des messages à l'écran en function d'une condition.
- Syntaxe :
  - `ASSERT condition REPORT message <SECURITY …>`
- Forcer un message:
  - `ASSERT FALSE REPORT "Toujours à l'écran " SEVERITY note;`
  - `REPORT "Toujours à l'écran " SEVERITY note;`
- Test en contexte séquentiel,
  - `ASSERT s = '1' REPORT "la sortie vaut '0' et ce n'est pas normal " SEVERITY warning; -- mentionne un warning`
  - `ASSERT s = '1' REPORT "la sortie vaut '0' et ce n'est pas normal " SEVERITY failure; -- Mentionne une erreur et arrête l'exécution`

33

Exemple



34

Banc de Test (1)

```
Entity design_tb is
End entity;
Architecture bench of design_tb is
    Signal Tclk : std_logic := '0';
    Signal Trst : std_logic;
    signal TA,TB,TOUT : std_logic_vector(3 downto 0);
    signal Done : boolean := False;
Begin
-- instanciation du composant à tester
UUT: entity work.design port map (
    A => TA, B => TB,
    OUT => TOUT,
     clk => Tclk, rst => Trst);
-- Génération d'une horloge
Tclk <= '0' when Done else not Tclk after 5 ns;
-- Génération d'un reset au début
Trst <= '1', '0' after 5 ns;
```


Banc de Test (2)

```
Stimuli_&_verification: process
Begin
TA <= "0000";
TB <= "0000";
wait for 5 NS;
assert TOUT = "0000" report "erreur" severity warning;
wait for 5 NS;
TA <= "1111";
wait for 5 NS;
assert TOUT = "1111" report "erreur" severity warning;
wait for 5 NS;
TA <= "0101";
TB <= "1010";
wait for 5 NS;
assert TOUT = "1010" report "erreur" severity warning;
Done <= True;
wait;
end process;
End architecture;
```

Exercice

- Exercice 3 : circuit Min Max

37