

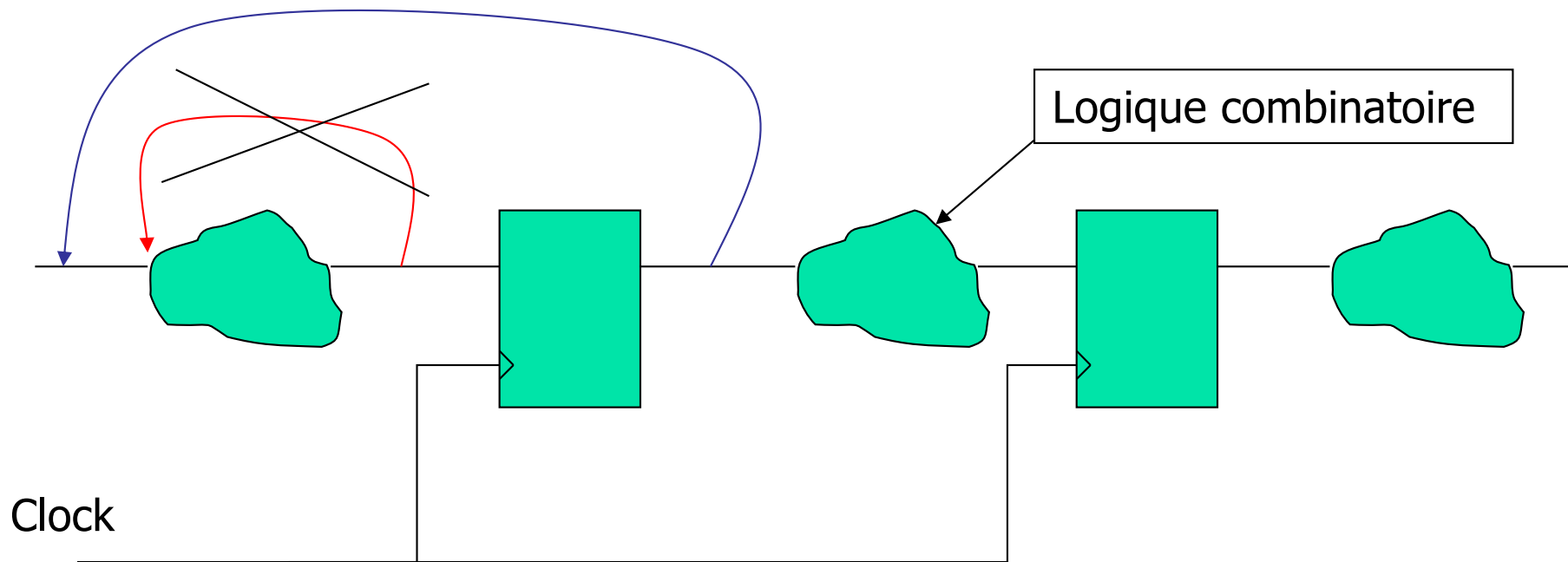


C7 – Les Process synchrones

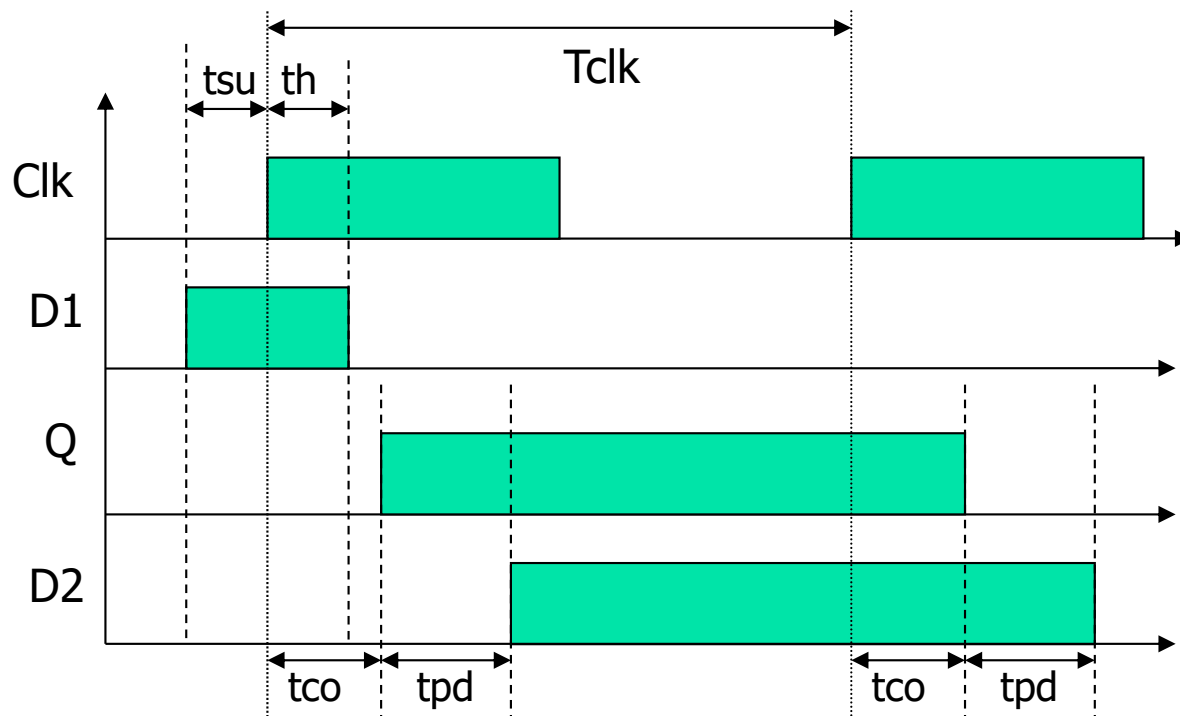
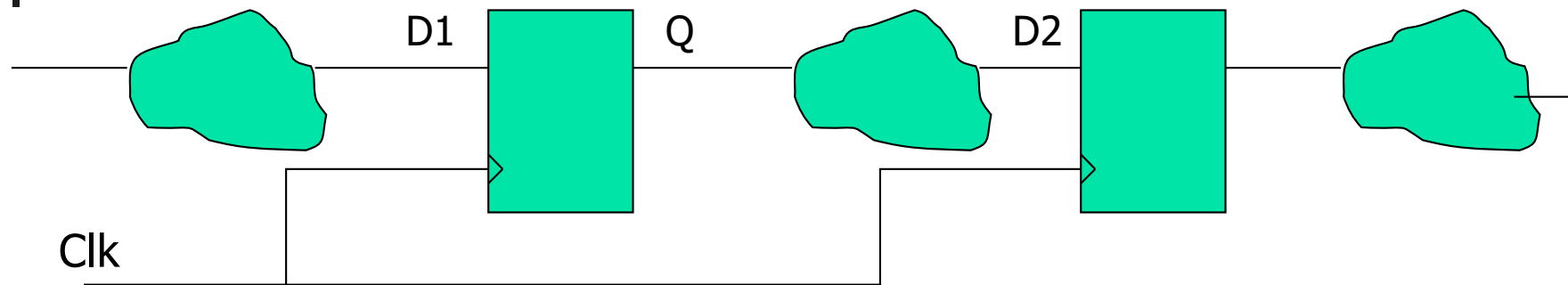
Yann DOUZE
VHDL

Design Synchrone

- Tous les registres se font dans des bascules D flip-flops avec une seule horloge externe.
- **Rebouclage sur du combinatoire interdit !**



Contraintes de temps



Tclk: période de l'horloge

tsu: time setup

th: time hold

tco: clock to output

tpd: propagation delay

$T_{clk} > t_{co} + t_{pd} + t_{su}$

$F_{max} = 1 / (t_{co} + t_{su} + t_{pd})$



Process synchrone

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



Style de code légal utilisant 'EVENT

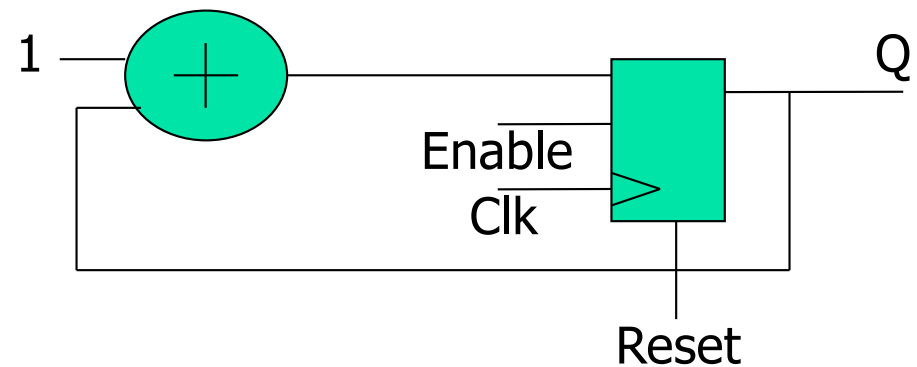
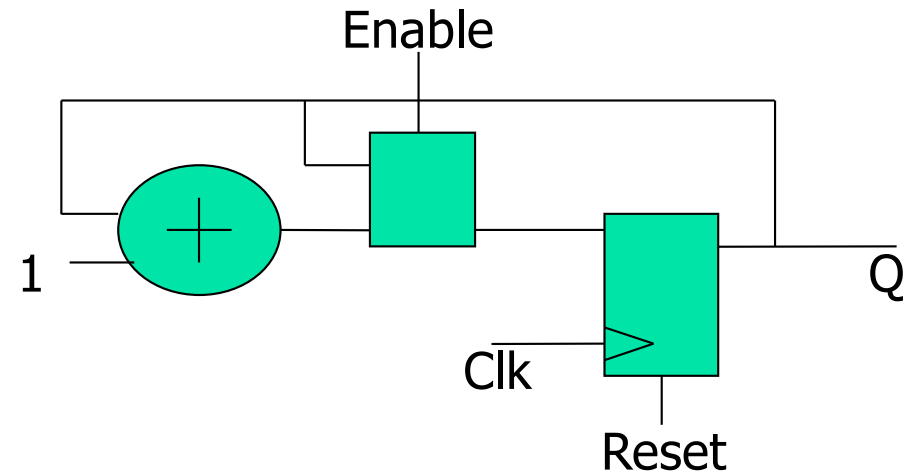
- S'EVENT est vrai si et seulement si il y a un événement sur S

```
process (Clk, rst)
Begin
    if rst = '1' then
        Q <= '0';
    elsif Clk'EVENT and Clk = '1' then
        Q <= D;
    end if;
end process;
```

```
process
begin
    wait until Clk'EVENT and Clk = '1';
    ...
end process;
```

Clock Enables

```
process (Clk, Reset)
begin
    if Reset = '1' then
        Q <= "00000000";
    elsif RISING_EDGE(Clk) then
        if Enable = '1' then
            Q <= Q + 1;
        end if;
    end if;
end process;
```



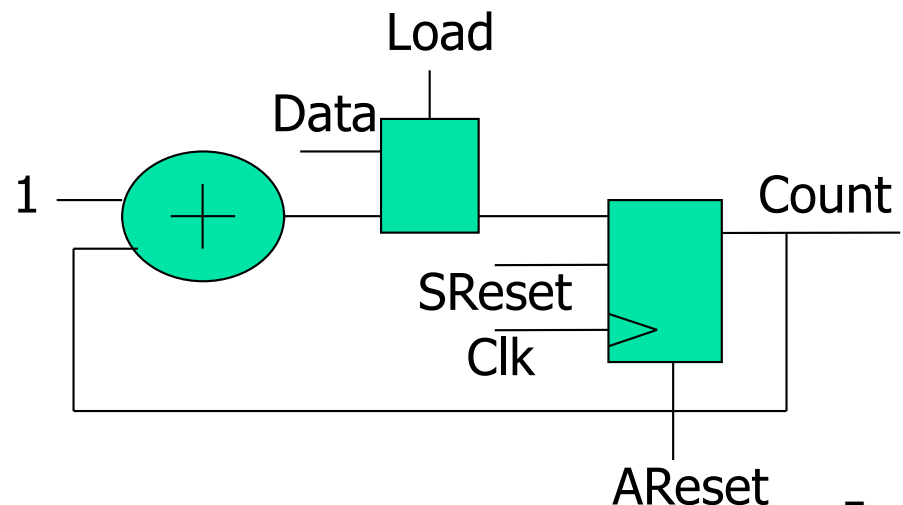
Actions Synchrones et Asynchrones

```
signal Count : unsigned(7 downto 0);
process (Clk, AReset)
begin
    if AReset = '1' then
        Count <= "00000000";
    elsif RISING_EDGE(Clk) then
        if SReset = '1' then
            Count <= "00000000";
        elsif Load = '1' then
            Count <= UNSIGNED(Data);
        else
            Count <= Count + 1;
        end if;
    end if;
end process;
```

Reset Asynchrone

Reset Synchrone

Load Synchrone





Compteur 8 bits

```
entity COUNTER8BIT is
    port ( CLK, RST : in      Std_logic;
           Q       : out     Std_logic_vector( 7 downto 0) );
end entity;
architecture RTL of COUNTER8BIT is
    signal CNT: unsigned( 7 downto 0 );
begin
    process (CLK,RST)
    begin
        if RST = '1' then
            CNT <= "00000000";
        elsif rising_edge(CLK) then
            CNT <= CNT + 1;
        end if;
    end process;
    Q <= std_logic_vector(CNT);
end architecture;
```




Compteur génériques

```
entity COUNTER is
  generic (N : integer:=8);
  port ( CLK, RST : in      Std_logic;
         Q       : out     Std_logic_vector (N-1 downto 0) );
end entity;

architecture RTL of COUNTER is
  signal CNT: unsigned(N-1 downto 0);
begin
  process (CLK,RST)
  begin
    if RST = '1' then
      CNT <= (others => '0');
    elsif rising_edge(CLK) then
      CNT <= CNT + '1';
    end if;
  end process;
  Q <= std_logic_vector(CNT);
end architecture;
```



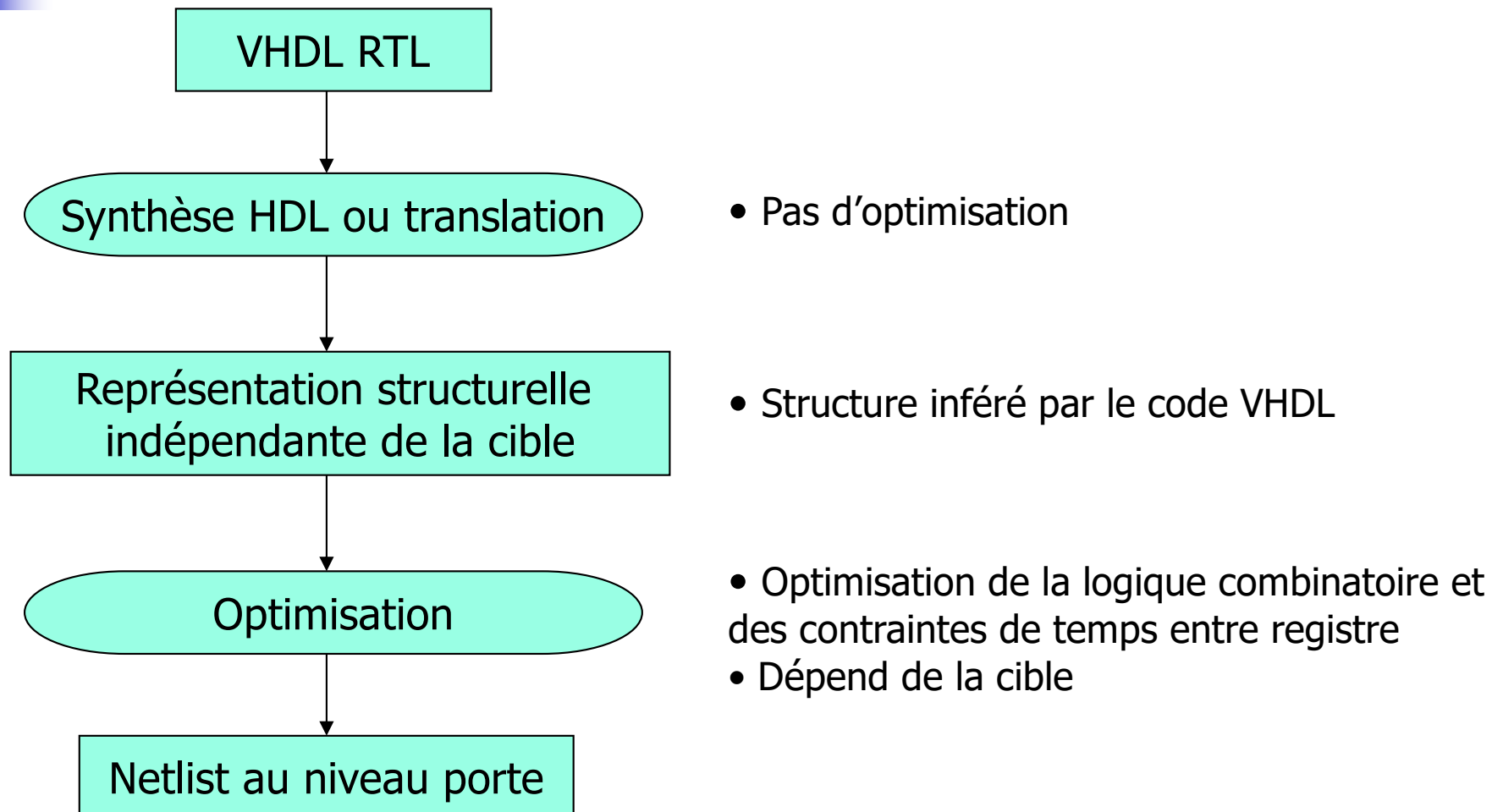
Instanciation d'un composant générique

```
-- l'entité du composant COUNTER
entity COUNTER is
    generic(N : integer:=8);
    port (CLK, RST : in  Std_logic;
          Q       : out  Std_logic_vector(N-1 downto 0));
end entity;
```

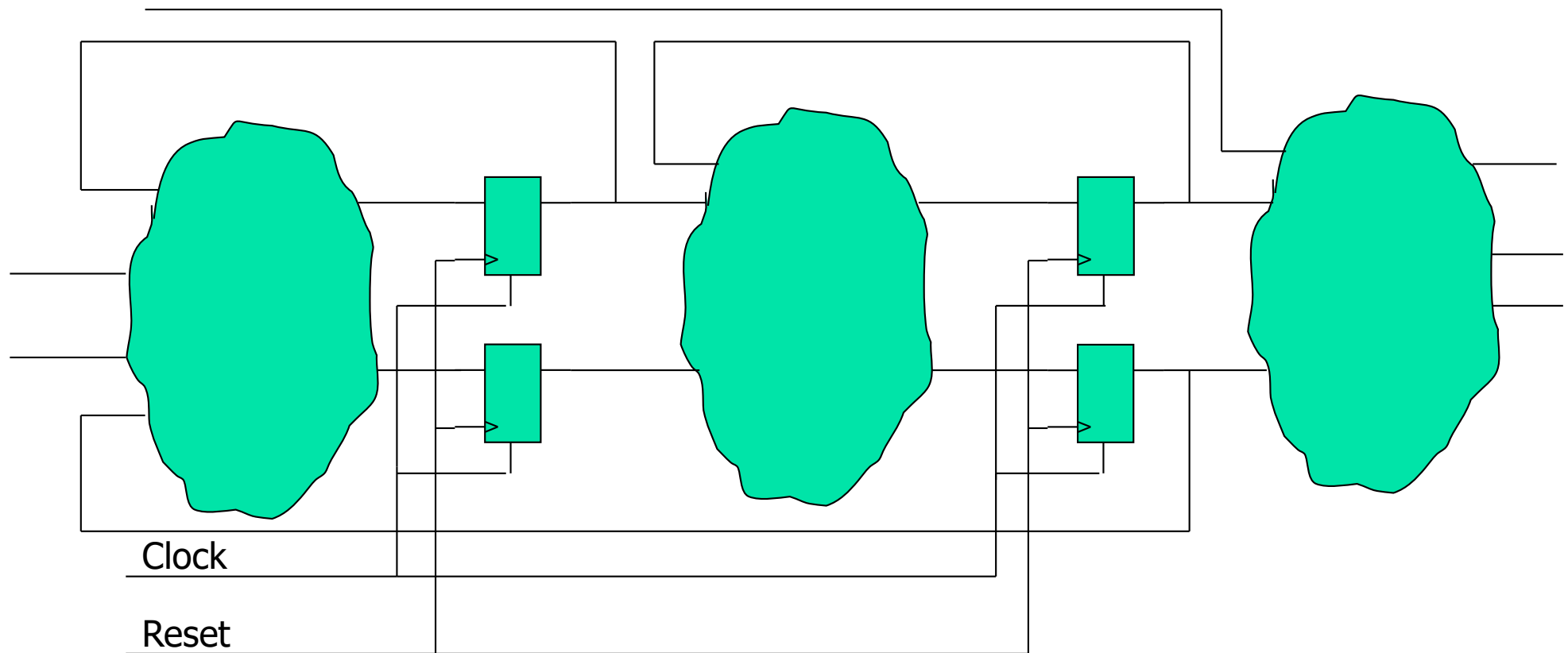
```
-- Utilisé dans l'architecture STRUCT d'un composant BLOK
architecture STRUCT of BLOK is
    signal Count4: std_logic_vector(3 downto 0);
    signal Count6: std_logic_vector(5 downto 0);
begin
    -- association par position
    U1: entity work.COUNTER generic map(4)
        port map(CLK , RST, Count4);
    -- association par nom
    U2: entity work.COUNTER generic map (N => 6)
        port map (CLK => CLK, RST => RST, Q => Count6);
end architecture;
```



Fonctionnement des outils de synthèse



Synthèse RTL



- La synthèse RTL n'ajoute, ne supprime ou ne bouge pas de registre.
- La synthèse RTL optimise uniquement la logique combinatoire.

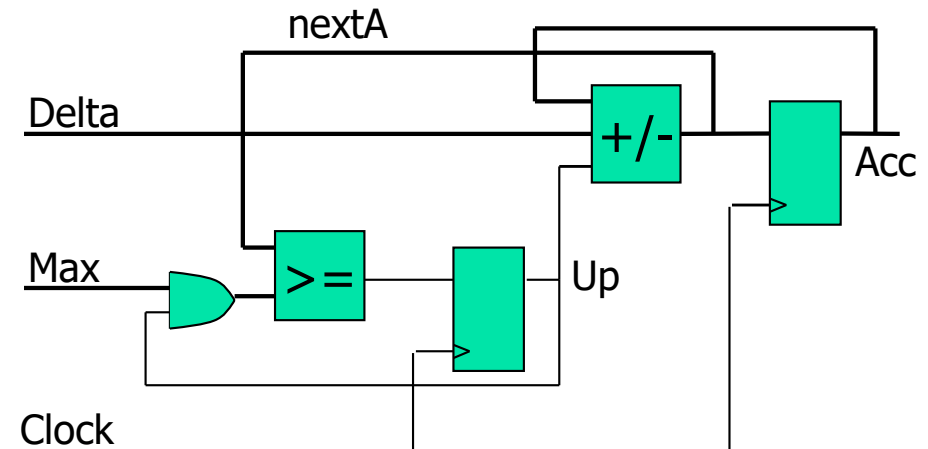


Règles : Les registres à la synthèse

- Les outils de synthèse RTL infèrent les registres directement à partir du code VHDL suivant certaine règle élémentaire :
 1. Des **registres** sont **inférés** à la **synthèse** que dans les **process synchrone**.
 2. **Signaux**: Tout les signaux assignés dans un **process synchrone** sont synthétisés par des registres.
 3. **Variables** : les variables assignés dans un **process synchrone** peuvent être synthétisées soit par un fil, soit par un registre.
 - Les variables sur lesquelles sont assignées une nouvelle valeur avant d'être lues sont synthétisées par un fil. (**assigné avant d'être lu => fil**)
 - Les variables qui sont lues avant d'être assignées sont synthétisées par un registre. (**lu avant d'être assigné => registre**)

Les registres à la synthèse

```
signal Acc, Delta, Max: signed(11 downto 0);
process (Clock)
    variable Up : std_logic;
    variable nextA : signed(11 downto 0);
begin
    if RISING_EDGE(Clock) then
        if Up = '1' then
            nextA := Acc + Delta;
            if nextA >= Max then
                Up := '0';
            end if;
        else
            nextA := Acc - Delta;
            if nextA < 0 then
                Up := '1';
            end if;
        end if;
        Acc <= nextA;
    end if;
end process;
```





Exercice 1

```
signal reg: std_logic_vector(3 downto 0);
begin
process(clk, rst)
begin
    if rst='1' then
        reg <= (others => '0');
    elsif Rising_edge(clk) then
        reg(3) <= D;
        reg(2 downto 0) <= reg(3 downto 1);
        Q <= reg(0);
    end if;
end process;
```

Combien de flip-flops ?



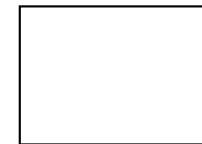
Flip-flops



Exercice 2

```
Signal INPUT : std_logic_vector(7 downto 0)
Signal REG : std_logic;
process (clk,rst)
variable V : STD_LOGIC;
begin
    if rst='1' then
        REG <= '0';
    elsif RISING_EDGE(clk) then
        V := '1';
        for I in 0 to 7 loop
            V := V and INPUT(I);
        end loop;
        REG <= V;
    end if;
end process;
```

Combien de bascule D
flip-flops ?



Flip-flops



Exercice 3

```
Signal OUTPUT : std_logic;
COUNTER : process (CLK, RST)
    variable COUNT : UNSIGNED(7 downto 0);
Begin
    if RST = '1' then
        COUNT := "00000000";
    elsif RISING_EDGE(CLK) then
        COUNT := COUNT + 1;
        OUTPUT <= COUNT(7);
    end if;
end process;
```

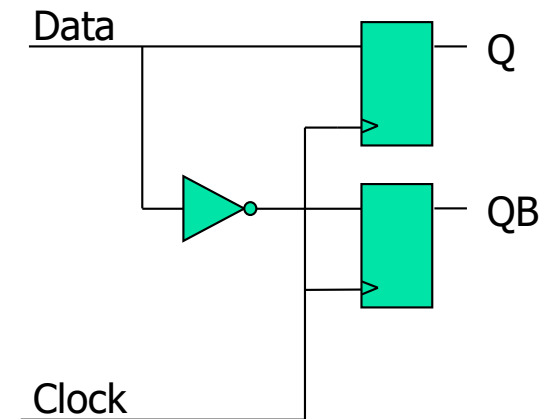
Combien de flip-flops ?



Flip-flops

La synthèse n'optimise pas les registres !

```
process (clk, rst)
begin
  if rst = '1' then
    Q <= '0';
    QB <= '0';
  elsif RISING_EDGE(clk) then
    Q <= Data;
    QB <= not Data;
  end if;
end process;
```

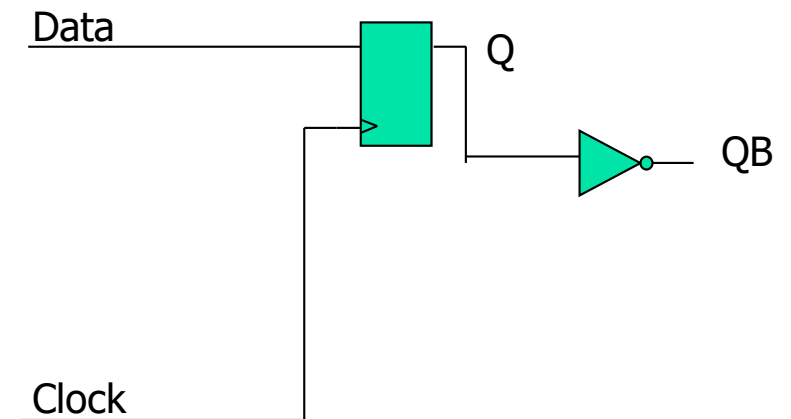


Comment faut il réécrire le code pour s'assurer qu'il n'y est qu'une seule bascule D ?

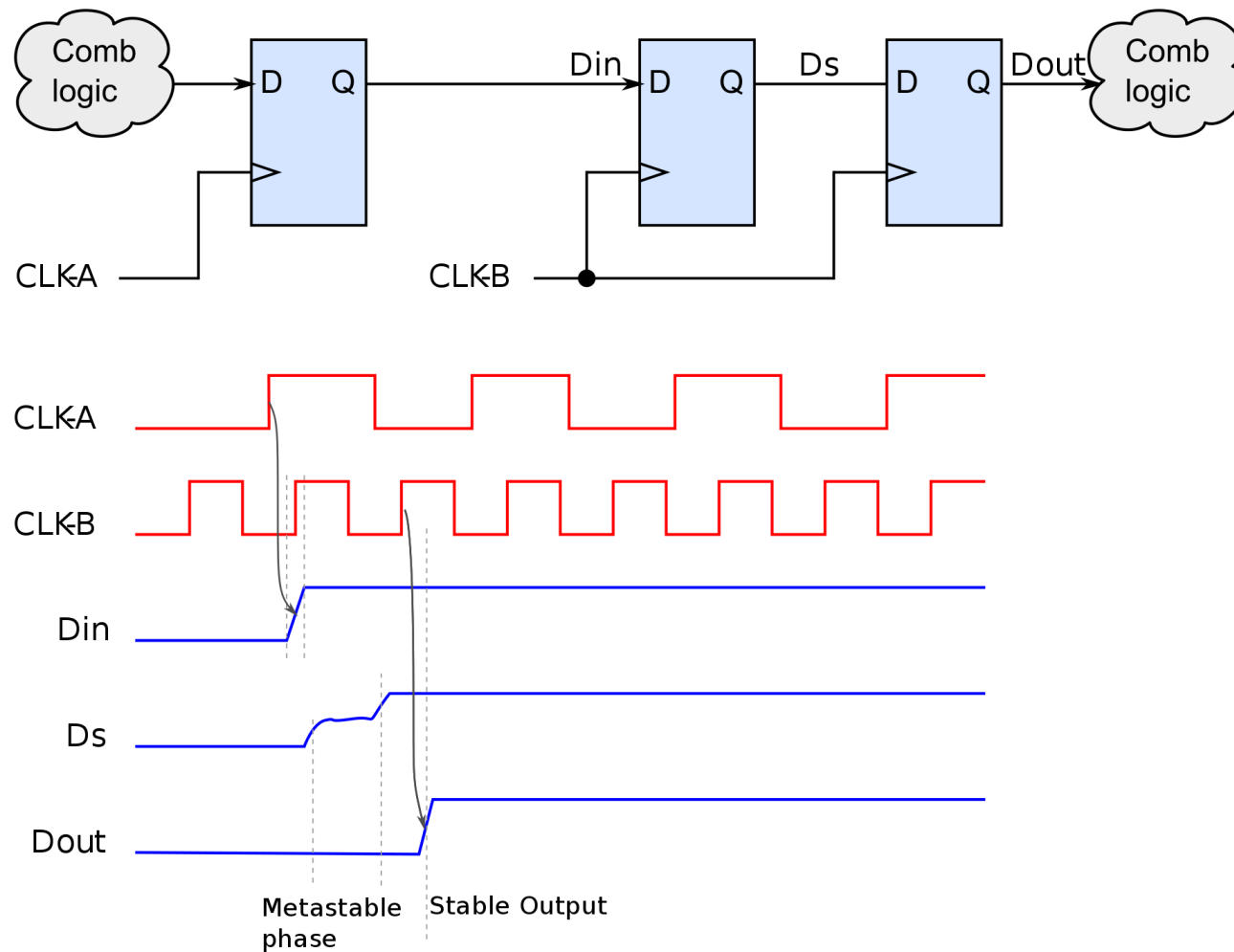
Solution !

```
process (clk, rst)
begin
  if rst = '1' then
    Q <= '0';
  elsif RISING_EDGE(clk) then
    Q <= Data;
  end if;
end process;

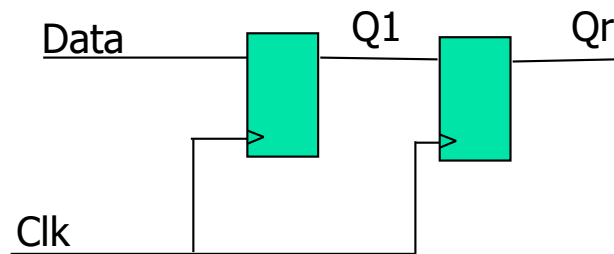
QB <= not Q;
```



Problèmes de métastabilité



Solution : Resynchronisation



- Avantage : sûreté de fonctionnement
- Inconvénient : introduit du délai (pipeline)



Code : re-synchronisation des entrées

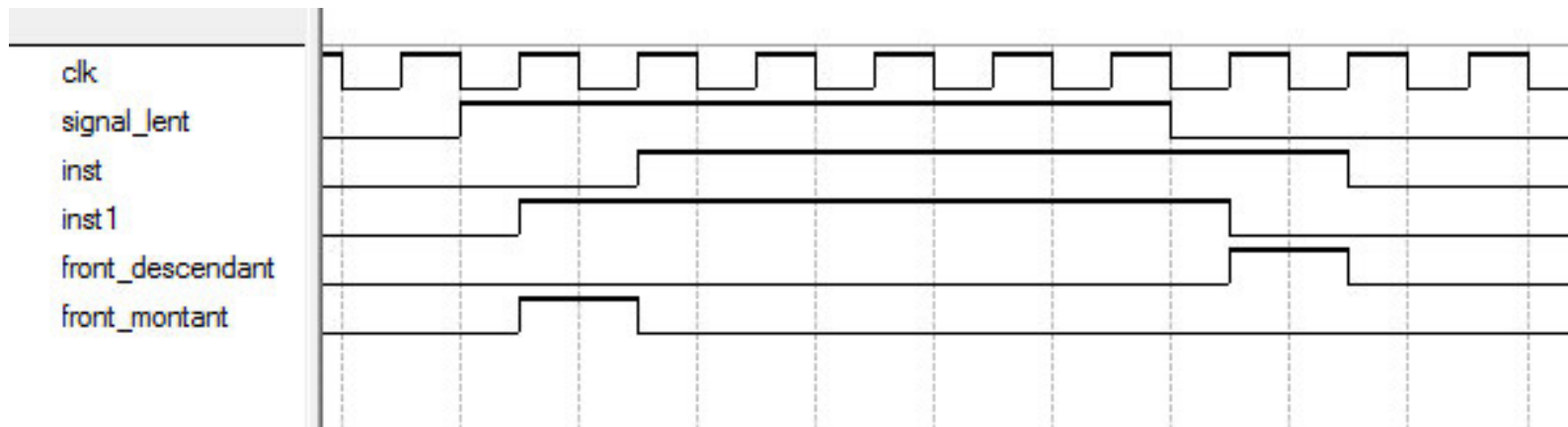
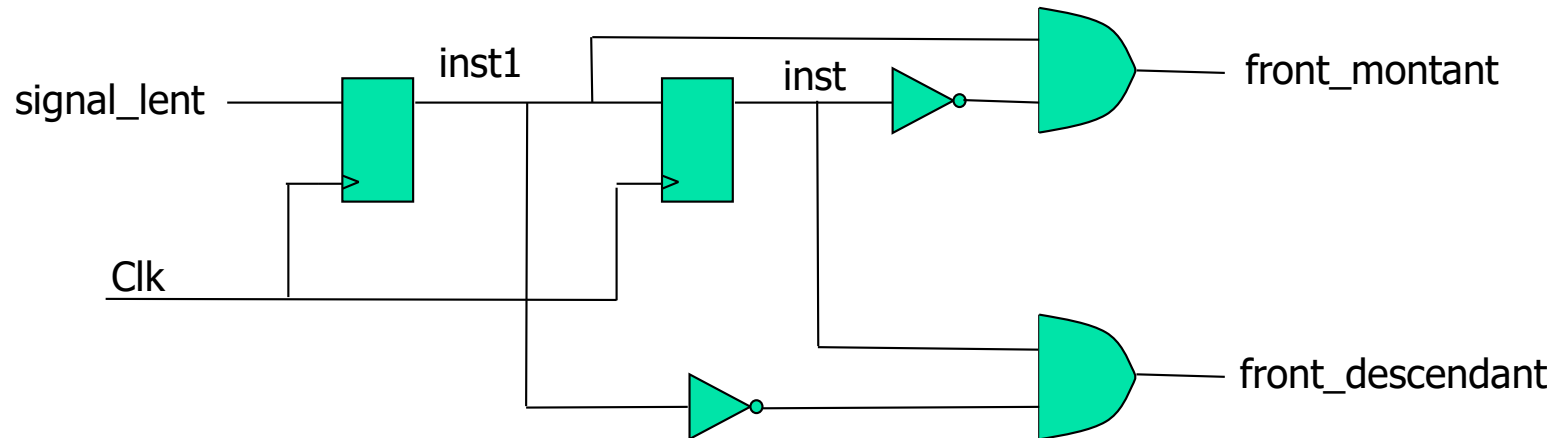
```
entity resynchro is
port(
    clk, rst, Data: in std_logic;
    Qr : out std_logic);
end entity;
architecture RTL of resynchro is
    Signal Q1 : std_logic;
begin
    process (clk,rst)
    begin
        if (rst='1') then
            Q1 <= '0'; Qr <= '0';
        elsif rising_edge(clk) then
            Q1 <= Data;
            Qr <= Q1;
        end if;
    end process;
end architecture;
```



Détection de fronts

- Les opérateurs de détection de fronts (`Rising_edge(clk)` et `Clk'event and clk='1'`) ne doivent être utilisé que pour tester le front d'une horloge (`clk`).
- Chaque fois que l'on fait un test de front, l'outil de synthèse comprend qu'il s'agit d'une horloge.
- Horloge = le signal le plus rapide du circuit.

Exercice : Détection des fronts d'un signal lent





Détection de front lent : code

```
Entity detect_fronts is
port(
    clk, rst, signal_lent : in std_logic;
    front_descendant : out std_logic;
    front_montant : out std_logic);
end entity;
architecture RTL of detect_fronts is
    signal inst, inst1 : std_logic;
begin
    PROCESS ( clk , rst)
    BEGIN
        if rst = '1' then
            inst <= '0'; inst1 <= '0';
        elsif rising_edge (clk) then
            inst1 <= signal_lent;
            inst <= inst1;
        end if;
    end process;
    front_montant <= inst1 and (not inst);
    front_descendant <= inst and (not inst1);
end architecture;
```



Exemple d'utilisation : compteur d'événements

```
entity cnt_evt is
port( clk, rst, sig_lent : std_logic;
      cnt : std_logic_vector(7 downto 0));
end entity;
architecture RTL of cnt_evt is
  signal Q : unsigned (7 downto 0);
  signal fm : std_logic;
Begin
U1 : entity work.detect_front port map (
  clk => clk, rst => rst, signal_lent=>sig_lent,
  front_montant => fm);
Process (clk,rst)
begin
  if (rst='1') then
    Q <= (others => '0');
  elsif rising_edge(clk) then
    if (fm='1') then
      Q<= Q +'1';
    end if;
  end if;
end process;
cnt <= std_logic_vector(Q);
End architecture;
```