SN360/SN361 Introduction à la conception des circuits numériques

---

Module « Systèmes matériels et logiciels »

Crédit SN360 : 4/4 / SN361 : 2/6

Vincent Beroulle

Bureau : D202

vincent.beroulle@esisar.grenoble-inp.fr

GRENOBLE INP UGA Esisar VALENCE

UGA Université Grenoble Alpes

Plan

I Introduction

II Représentation des nombres en binaire

III Circuits combinatoires

IV Circuits séquentiels

V Machine à états finis

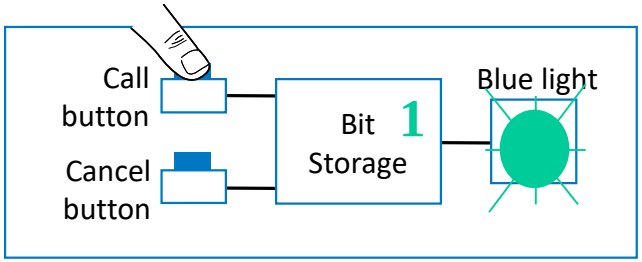
VI Complément sur les HDL

VII Circuits reconfigurables (optionnel)

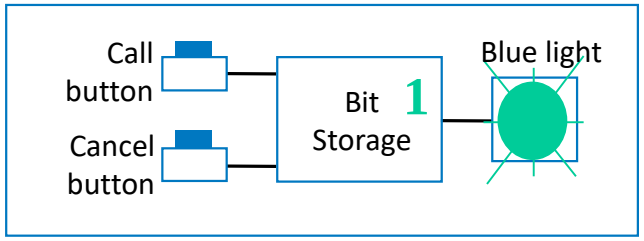
IV Circuits séquentiels

Définitions

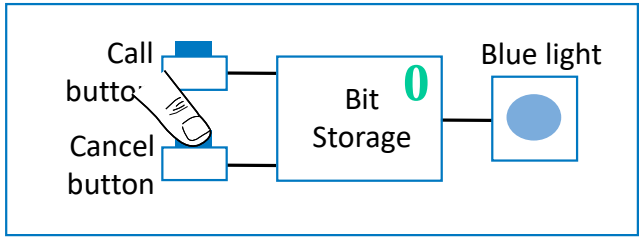
- Circuit séquentiel :
  - Les valeurs des sorties dépendent des valeurs d'entrée **et** de la séquence de valeurs d'entrée depuis le reset.
  - C'est-à-dire S=**F**(E, séquence des entrées).
  - On a donc une **mémorisation** et **une initialisation du circuit**.



1. Call button pressed – light turns on



*2. Call button released – light **stays on***

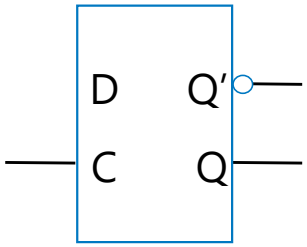


3. Cancel button pressed – light turns off

IV Circuits séquentiels

Elément mémorisant : D latch

- Fonctionnement



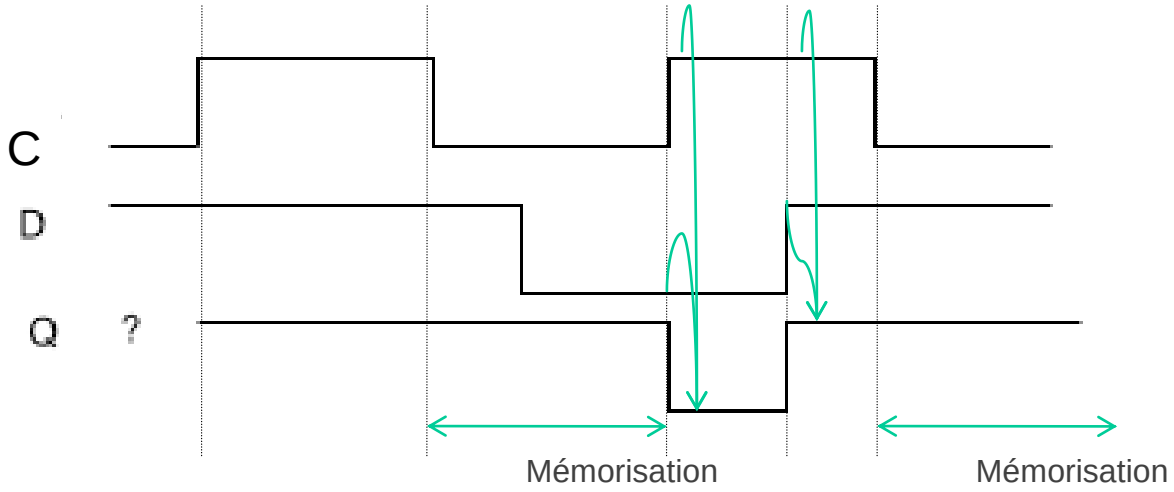
D latch symbol

C	D	Q
Oui	0	0
Oui	1	1
Non	*	Q à l'instant précédent

– Sensible au niveau de l'activation (appelé D LATCH ou BASCULE VERROU)

IV Circuits séquentiels
Elément mémorisant : D latch

- Exemple de chronogramme :



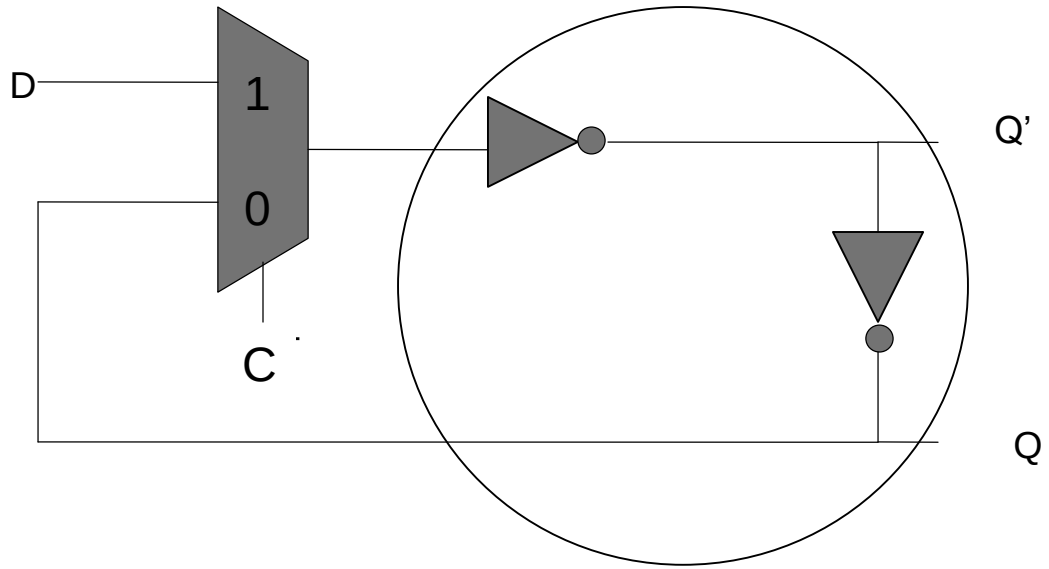
- Problème :
  - **Si D change quand C passe de 1 à 0, Q est indéterminé.**

5

IV Circuits séquentiels

Elément mémorisant : D latch

- Principe :



Elément mémoire

6

IV Circuits séquentiels

Codes VHDL&Verilog – D latch

VHDL

```
process(d,c)
begin
if c='1' then
   q<=d;
end if;
end process;
```

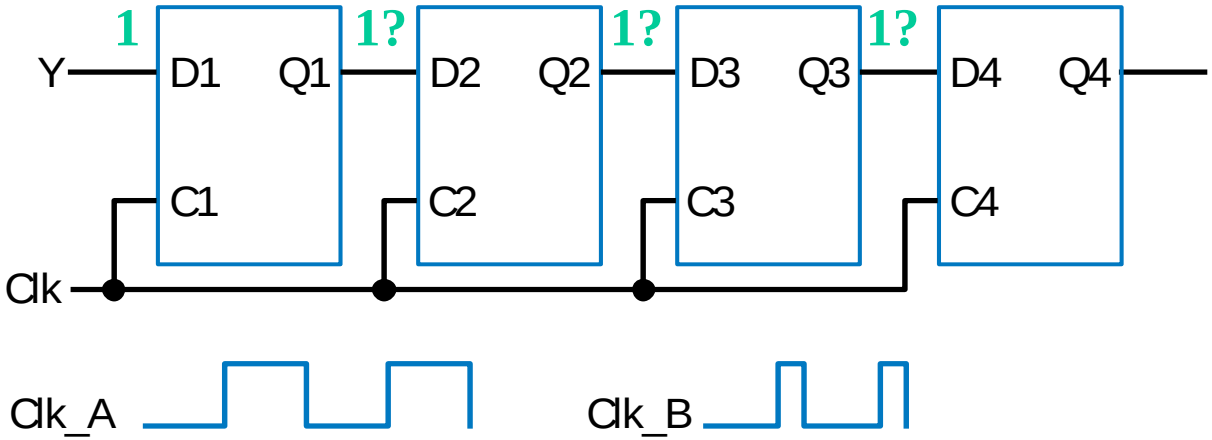
Verilog

```
always @(*)
begin
   if (c) begin
      q=d;
   end
end
```

IV Circuits séquentiels

Elément mémorisant : D latch

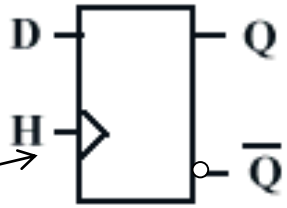
- Les D latch sont “transparentes”
  - Quand C=1, au travers de combien de bascules le signal va-t-il se propager?
  - Cela dépend de la longueur de l’impulsion C=1!
    - Clk_A – le signal traverse plusieurs bascules
    - Clk_B – le signal traverse moins de bascules



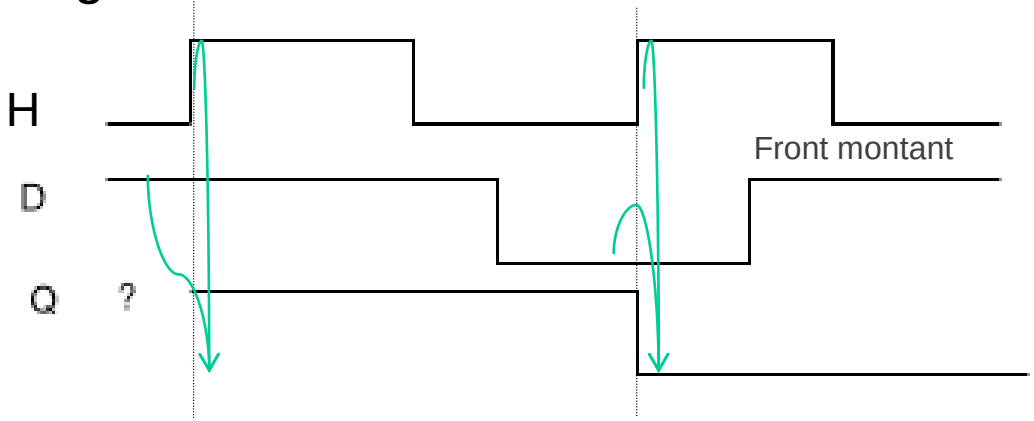
IV Circuits séquentiels

Elément mémorisant : D Flip Flop (DFF)

Le triangle signifie l'activation sur front montant montant



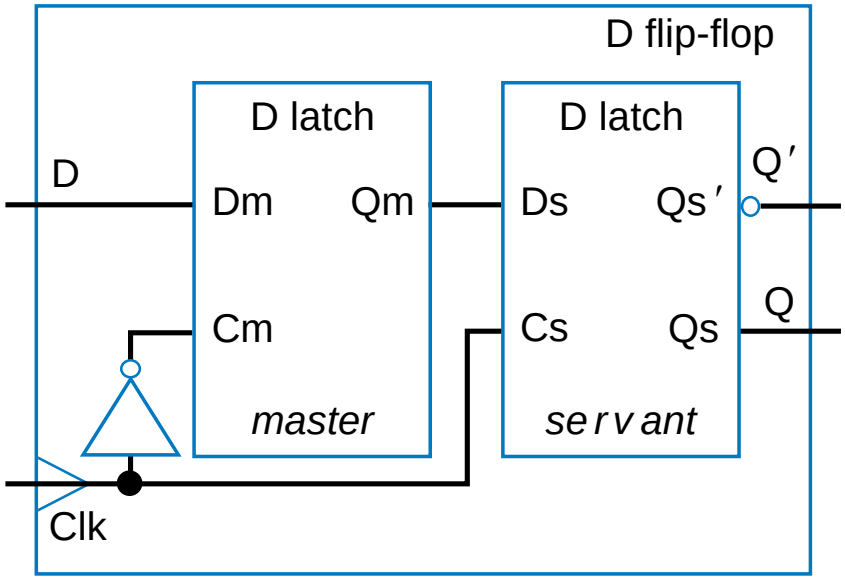
- Exemple de chronogramme :



- Problème :
  - **Si D change quand H présente un front montant, Q est indéterminé**
  - **Intervalle de commutation interdit de D T_{setup} avant et T_{hold} après le front d'horloge actif**

IV Circuits séquentiels

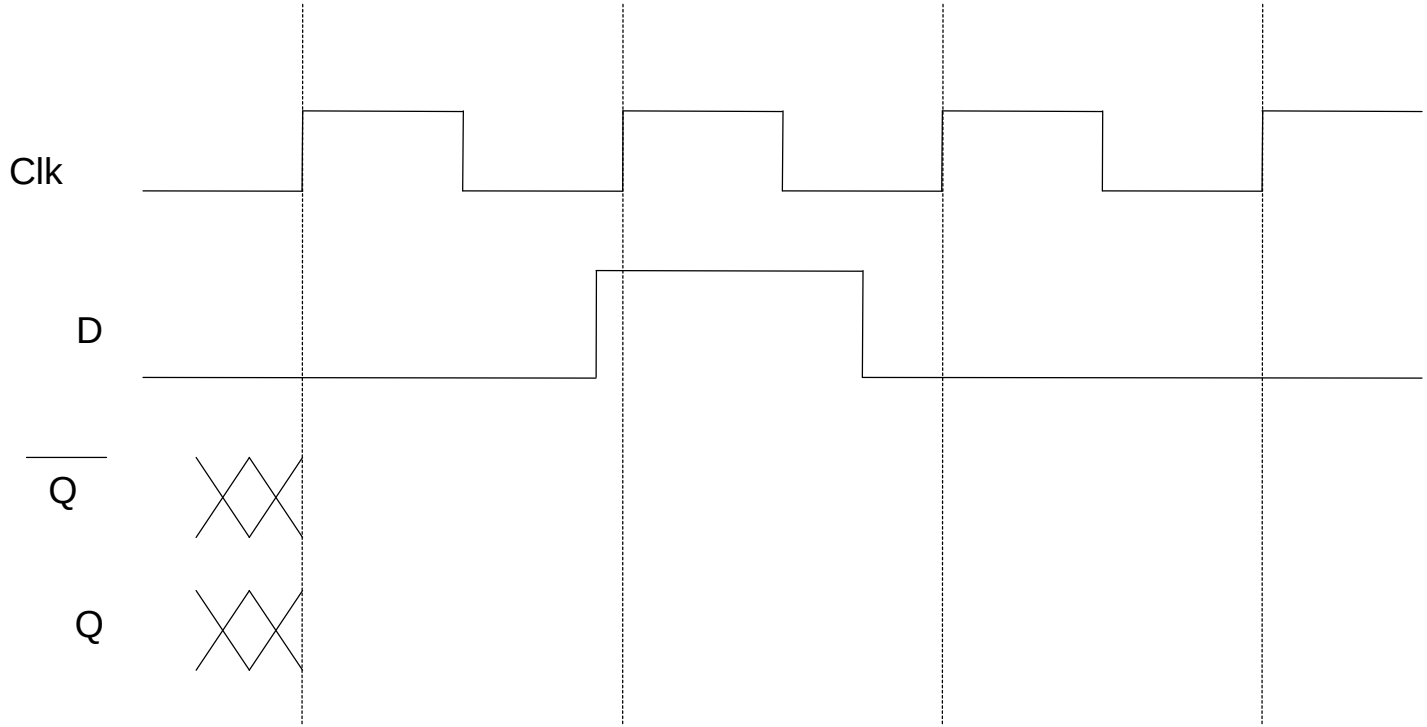
Elément mémorisant : D Flip Flop (DFF)



IV Circuits séquentiels

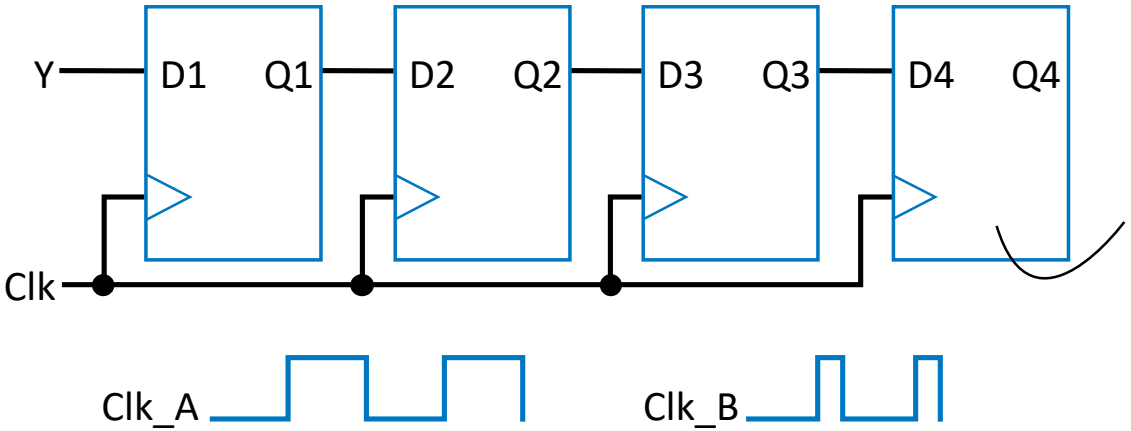
Elément mémorisant : D Flip Flop

- Complétez le chronogramme suivant:



IV Circuits séquentiels

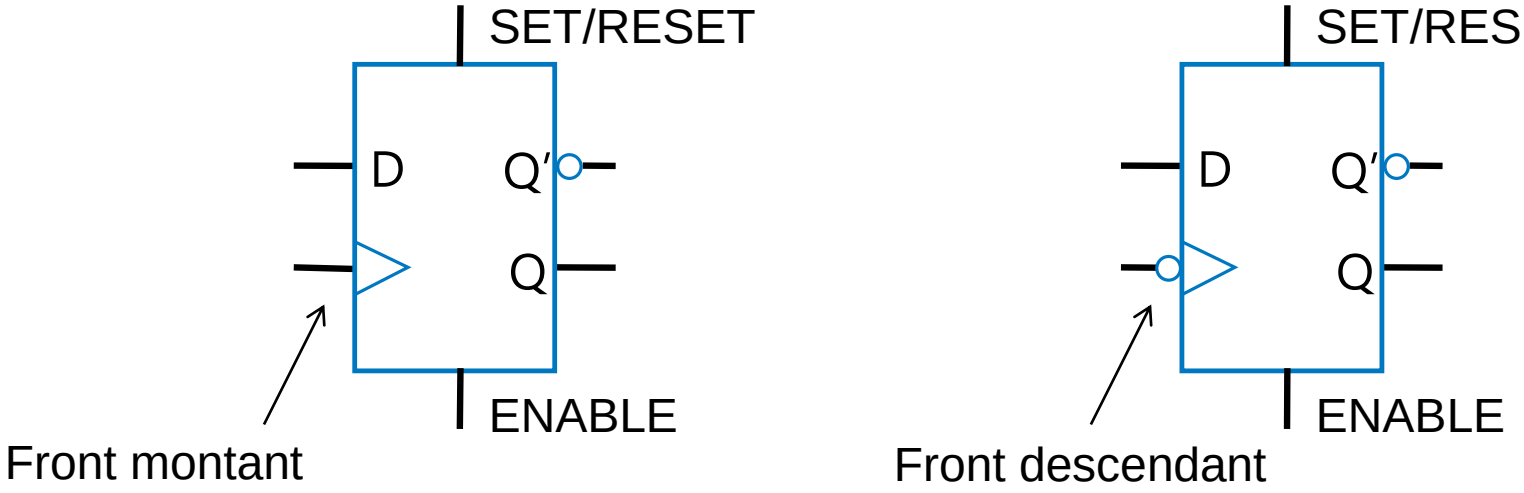
Elément mémorisant : D Flip Flop



- Que ce passe-t-il pour les 2 horloges Clk_A et Clk_B?
- Les bascules DFF résolvent le problème de la transparence des D latchs

IV Circuits séquentiels

Elément mémorisant : D Flip Flop



- Initialisation (**TOUJOURS PRIORITAIRE SUR ENABLE**)
  - SET / RESET : mise à ‘1’ ou à ‘0’ de Q
  - Synchrone / Asynchrone <=> il faut un front montant de H / indépendant de H
- Enable
  - à ‘1’ : autorise le fonctionnement de la bascule comme vu précédemment.
  - à ‘0’ : inhibe le fonctionnement de la bascule : pas de chargement au front montant.
- Activation = front d’horloge

IV Circuits séquentiels

Codes VHDL&Verilog – D Flip Flop

VHDL

```
-- reset synchrone
process(clk)
begin
if clk'event and clk='1' then
   if reset = '1' then
      q <= '0';
   elsif enable = '1' then
      q <= d;
   end if;
end if;
end process;
```

Verilog

```
reg q;
-- reset synchrone
always @(posedge clk) begin
   if (reset) begin
         q <= 1'b0;
   end else if (enable) begin
      q <= d;
   end
end
```

-- utilisation d'**affectation non bloquante** (affectation non immédiate) : <=

IV Circuits séquentiels

Codes VHDL&Verilog – D Flip Flop

VHDL

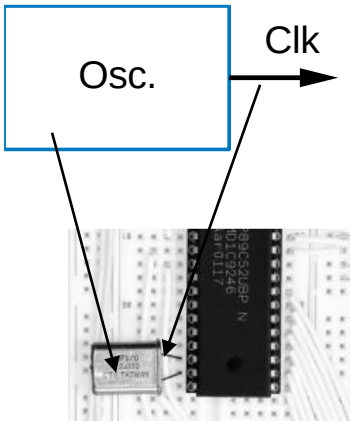
```
-- reset asynchrone
process(rest, clk)
begin
if reset = ‘1’ then
   elsif h'event and h='1' then
    if enable = '1' then
      q <= d;
    end if;
end if;
end process;
```

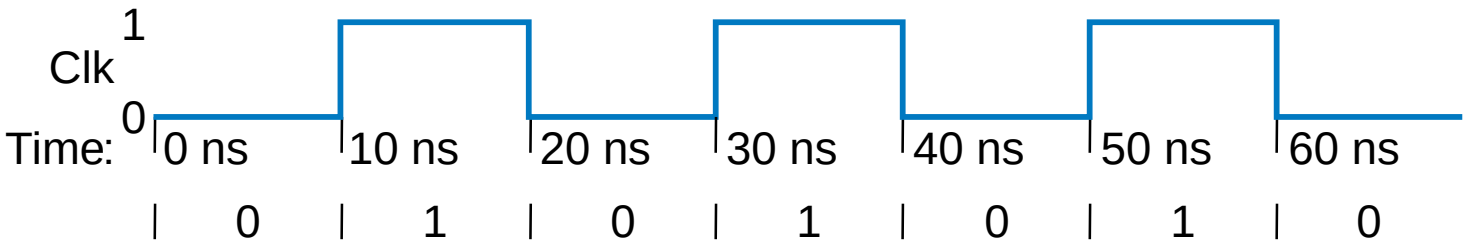
Verilog

```
reg q;
-- reset asynchrone
always @(posedge clk or negedge reset) begin
   if (reset) begin
         q <= 1'b0;
   end else if (enable) begin
      q <= d;
   end
end
```

IV Circuits séquentiels
Signal d'horloge

- Le signal d'horloge provient toujours d'un oscillateur
  - L'oscillateur génère un signal périodique
    - "Période" = 20 ns, "Frequence" = 1/20 ns = 50 MHz
    - Un "cycle" est la durée d'une période (20 ns)





Period/Freq : Se souvenir de 1 ns → 1 GHz

Freq.	Period
100 GHz	0.01 ns
10 GHz	0.1 ns
1 GHz	**1 ns**
100 MHz	10 ns
10 MHz	100 ns

IV Circuits séquentiels

Codes VHDL&Verilog – signal d'horloge

VHDL

```
signal clk : bit; -- par défaut initialisé à '0'
clk <= not clk after 10 ns;
-- période de 20 ns
-- ou avec un process
process
begin
 wait for 10 ns;
 clk <= not clk;
end process;
```

Verilog

```
initial begin clk = 0;
// Initialisation de l'horloge à 0
end

always begin
#10 clk = ~clk;
// clk change toutes les 10ns
end
```

IV Circuits séquentiels

Test bench

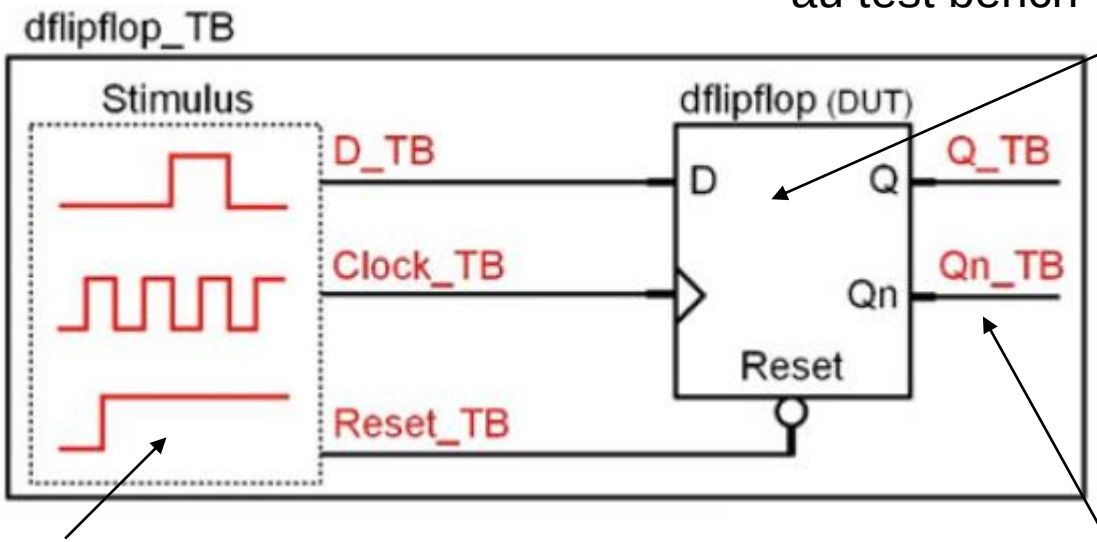
- Test bench = Simulation à différents niveaux d'abstraction
  - Génération des stimuli
  - Connection des signaux aux entrées du circuit à tester
  - Observation des signaux
- Test bench = un fichier HDL sans entrée et sans sortie
  - Non synthétisable!

IV Circuits séquentiels

Test bench

Même nom que le DUT suivi de _TB

Le composant à tester (DUT pour Device Under Test) est instancié dans le test bench, ses ports sont connectés à des signaux internes au test bench



Les stimuli sont créés dans le test bench et connectés au DUT

Les sorties du DUT peuvent être analysées sous forme de chronogramme dans le simulateur, ou directement vérifiées dans le test bench

IV Circuits séquentiels

Test bench - VHDL

VHDL

```
entity dflipflop_TB is
end dflipflop_TB;

architecture bhv of dflipflop_TB is
signal Reset_TB, Clock_TB, D_TB,
Q_TB, Qn_TB : bit;

component dflipflop
port(Clock, Reset, D: in bit;
     Q, Qn : out bit);
end component;
```

Mêmes signaux que dans l'entité

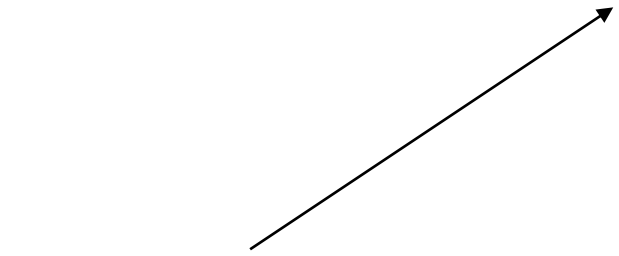
```
begin
Reset_TB <= '0', '1' after 15 ns;

Clock_TB <= not Clock_TB after
10 ns;

D_TB <= '0', '1' after 55 ns, '0'
after 105 ns;

DUT: dflipflop
Port map(Clock_TB, Reset_TB,
D_TB, Q_TB, Qn_TB);

End bhv;
```

IV Circuits séquentiels

Test bench - VHDL

VHDL

```
entity dflipflop_TB is
end dflipflop_TB;

architecture bhv of dflipflop_TB is
signal Reset_TB, Clock_TB, D_TB, Q_TB, Qn_TB : bit;

begin
Reset_TB <= '0', '1' after 15 ns;

Clock_TB <= not Clock_TB after 10 ns;

D_TB <= '0', '1' after 55 ns, '0' after 105 ns;

DUT: entity dflipflop(rtl)
Port map(Clock_TB, Reset_TB, D_TB, Q_TB, Qn_TB);

End bhv;
```



Instanciation directe

IV Circuits séquentiels

Test bench - Verilog

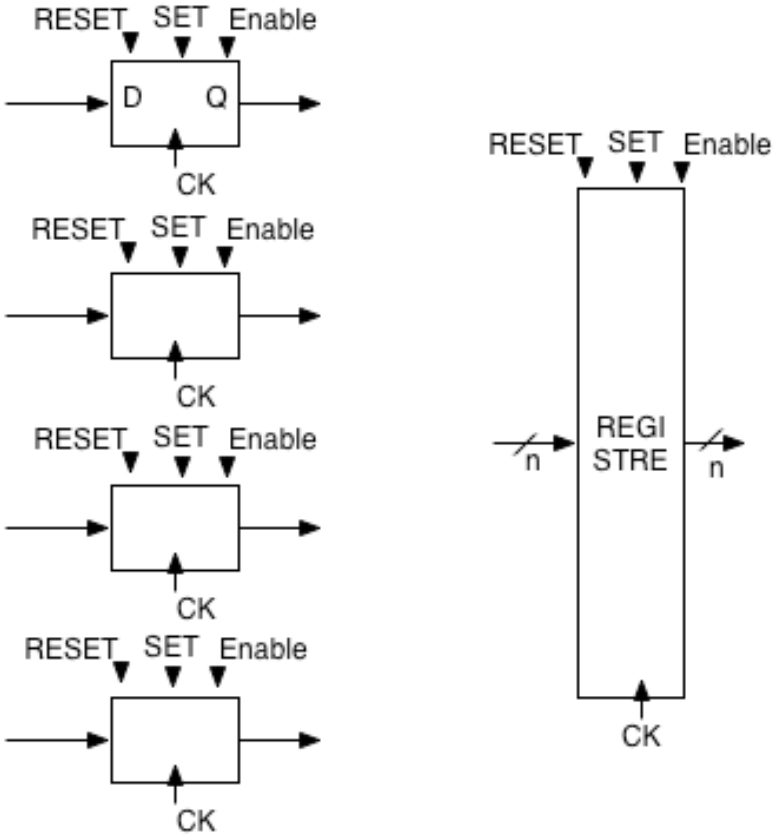
Verilog

```
‘timescale 1ns/1ps
module dflipflop_TB ();
wire Q_TB, Qn_TB;
reg Clock_TB, Reset_TB, D_TB;
Dflipflop DUT (Clock_TB, Reset_TB,
D_TB, Q_TB, Qn_TB,);

initial
begin
Reset_TB = 1’b0;
#15 Reset_TB = 1’b1;
end

initial
begin
Clock_TB = 1’b0;
End

always
begin
#10 Clock_TB = ~Clock_TB;
end

initial
begin
D_TB = 1’b0;
#55 D_TB = 1’b1;
#50 D_TB = 1’b0;
end
end module
```

IV Circuits séquentiels

Registres

- Un registre *n* bits = *n* bascules avec les mêmes entrées set, reset, clk, enable … (c'est-à-dire même initialisation et activation)



IV Circuits séquentiels

Codes VHDL&Verilog – Registre

VHDL

```
-- registre de taille à définir
signal d, q : bit_vector(…);
process(h)
begin
if clk'event and clk='1' then
-- ou if rising_edge(clk) then
   if reset = '1' then
      q <= (others=>'0');
   elsif enable = '1' then
         q<=d;
      end if;
end if;
end process;
```

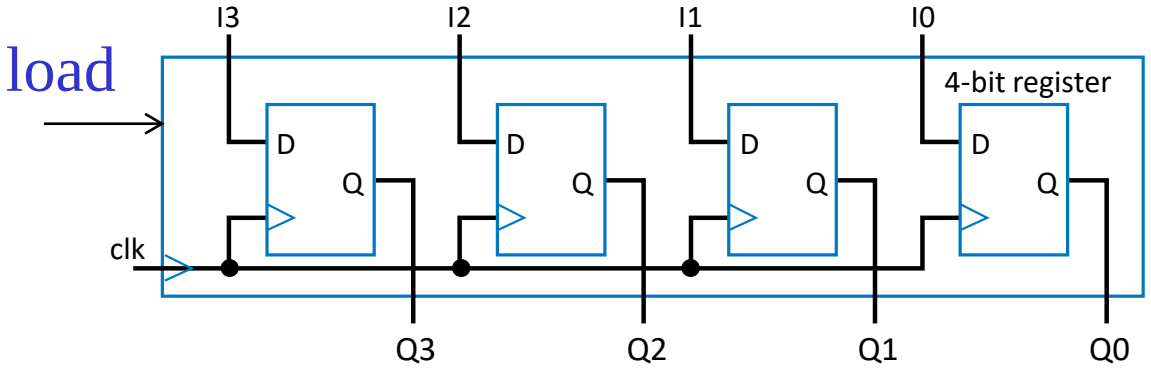
Verilog

```
-- registre 4 bits
reg [3:0] q;
wire [3;0] d;

always @(posedge clk) begin
   if (reset) begin
         q <= 4'b0000;
   end else if (enable) begin
      q <= d;
   end
end
```

Pour déterminer le nombre de DFF on compte le nombre de bits affectés dans les « rising_edge »

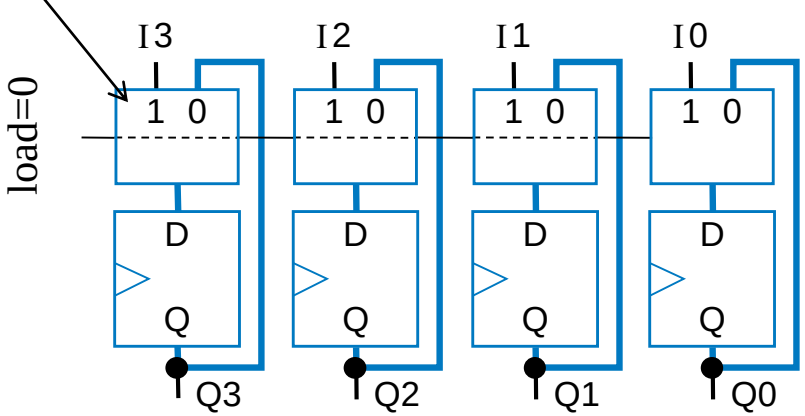
IV Circuits séquentiels

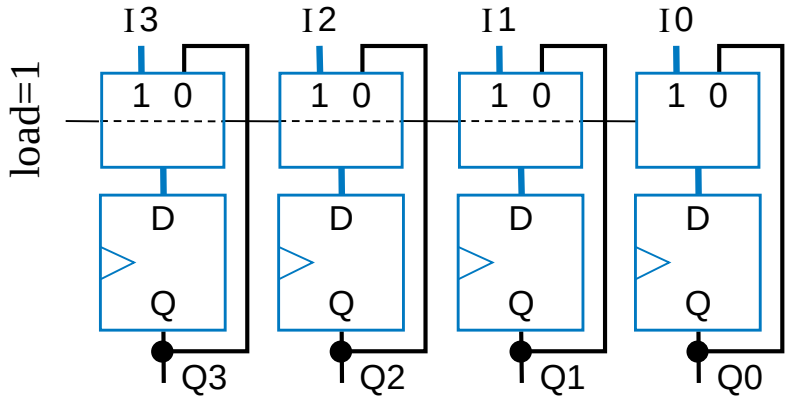
Registres à chargement parallèle - PIPO



Registre
PIPO
=
Parallel In
Parallel Out

MUX 2x1





IV Circuits séquentiels

Codes VHDL&Verilog – Registre à chargement parallèle

VHDL

```
-- registre
signal d, q : bit_vector(…);

process(clk)
begin
if clk'event and clk='1' then
   if load = '1' then
      q <= d;
   end if;
end if;
end process;
```

Verilog

```
-- registre 4 bits
reg [3:0] q;
wire [3;0] d;

always @(posedge clk) begin
   if (load) begin
         q <= d;
   end
end
```

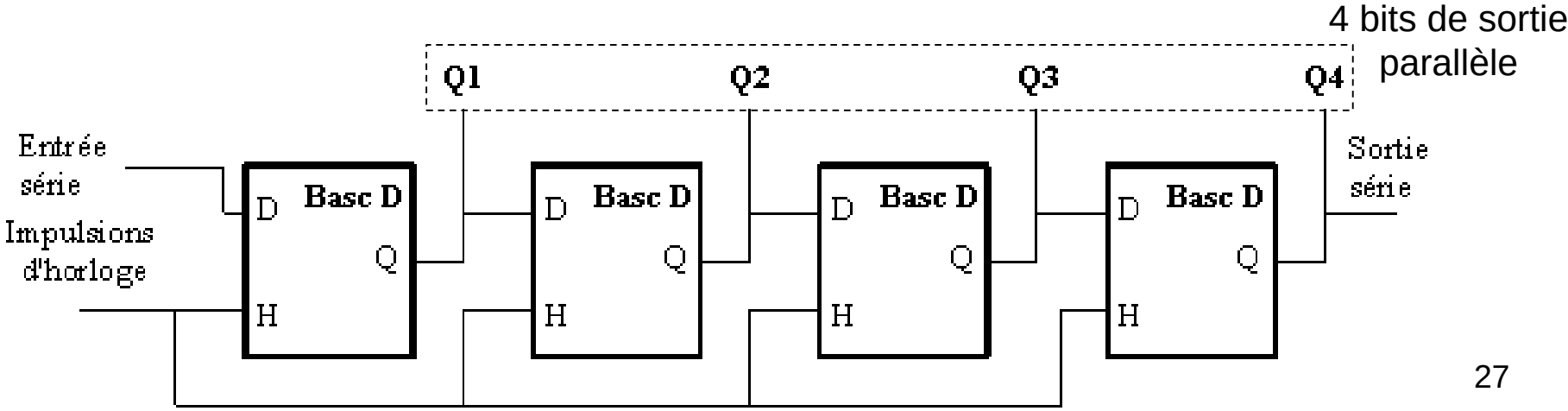
IV Circuits séquentiels

Registres à décalage

- **Un registre à décalage *N* bits** est un registre *N* bits dans lequel les bits sont décalés à chaque coup d'horloge.

- Principe : la sortie de la bascule N est reliée à l'entrée de la bascule N-1 etc…

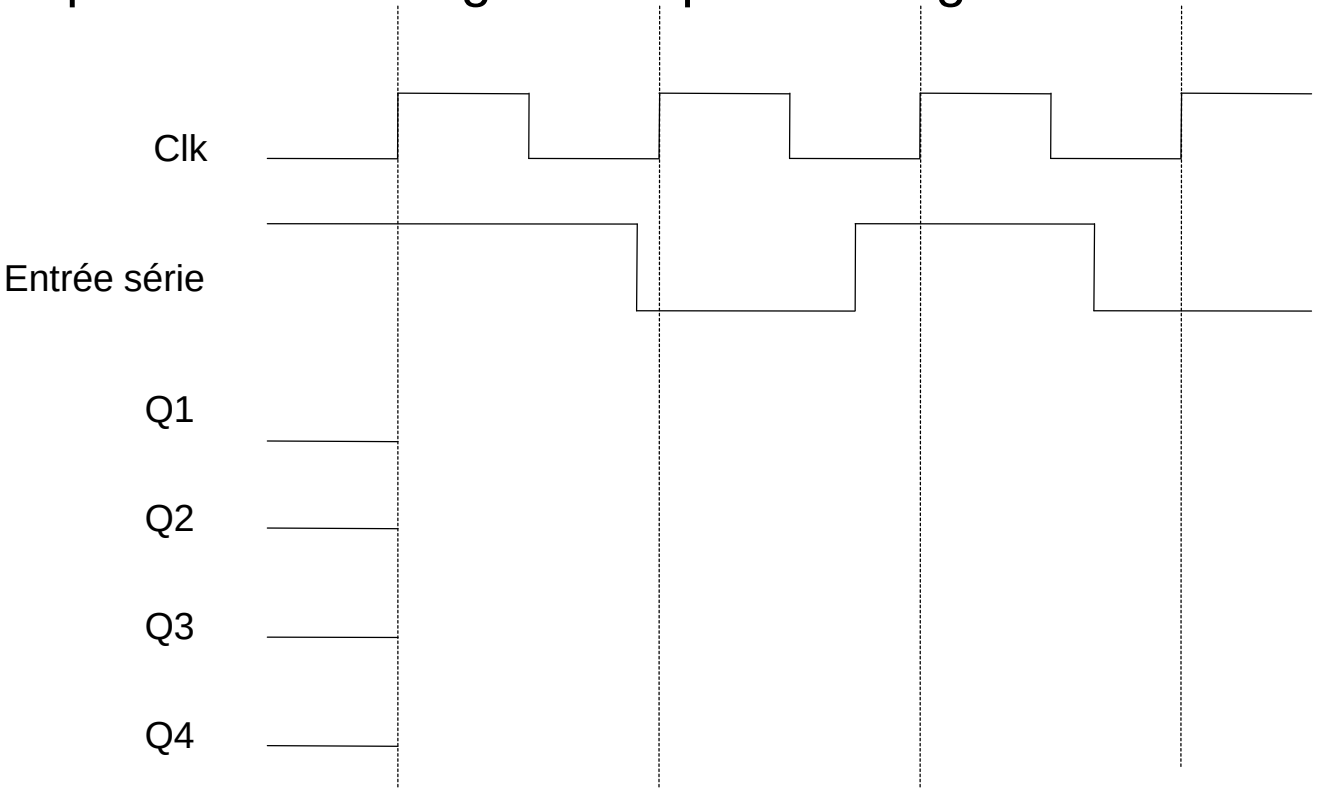
- Exemple ci-dessous : Registre 4 bits « Serial IN – Serial OUT » (**SISO**) et « Serial IN – Parallel OUT » (**SIPO)**



IV Circuits séquentiels

Registres à décalage

- Complétez le chronogramme pour un registre SIPO :



IV Circuits séquentiels

Code VHDL – SISO & SIPO

VHDL

```
entity SISO-SIPO is
port(clk, d : in bit;
      s : out bit;
      q : out bit_vector(4 downto 1));
end SISO-SIPO;

architecture RTL of SISO-SIPO is
signal q_temp : bit_vector(4 downto 1);
```

VHDL - suite

```
begin
process(clk)
begin
if clk'event and clk='1' then
q_temp <= q_temp(3 downto 1)&d;
end if;
end process;
q<=q_temp;
s<=q(4);
end RTL;
```

& : opérateur de concaténation

IV Circuits séquentiels

Code Verilog – SISO & SIPO

Verilog

```
module SISO_SIPO (
    input wire clk,
    input wire d,
    output wire s,
    output wire [4:1] q
);
```

Verilog - suite

```
    reg [4:1] q_temp;
    always @(posedge clk) begin
        q_temp <= {q_temp[3:1], d};
    end
    assign q = q_temp;
    assign s = q_temp[4];
endmodule
```

IV Circuits séquentiels

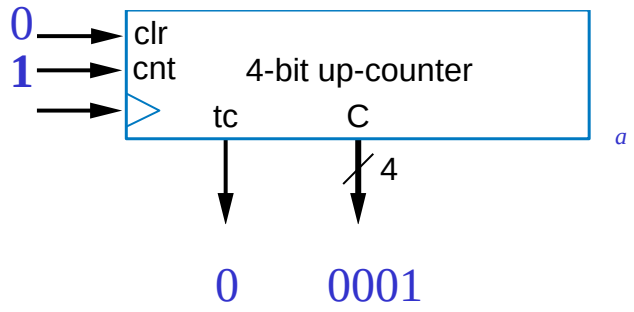
Registres à décalage

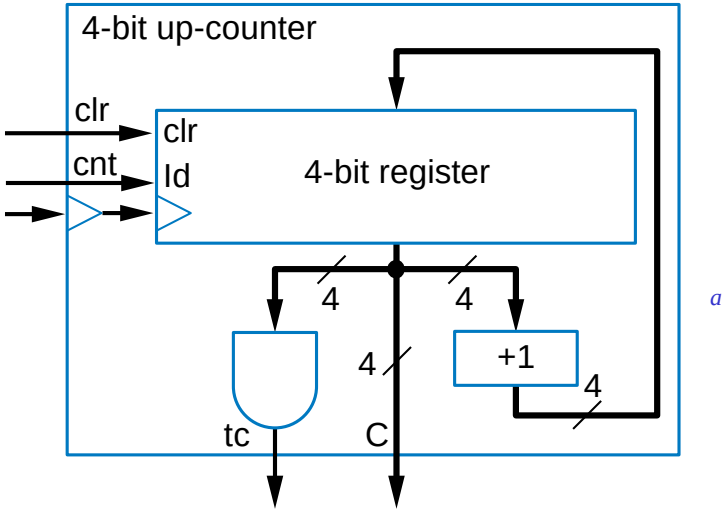
“En utilisant quatre registres, concevez un circuit qui stocke les quatre valeurs présentes à une entrée D de 8 bits lors des quatre derniers cycles d’horloge. Le circuit doit avoir une seule sortie de 8 bits qui peut être configurée à l’aide de deux entrées S1 et S0 pour sortir n’importe lequel des quatre registres. [Astuce : utilisez un multiplexeur 8 bits 4x1.]”

IV Circuits séquentiels

Compteurs

- Compteur 4 bits, principe :
  - un signal incrémentation **cnt** est envoyé au compteur (autorisation de comptage)





IV Circuits séquentiels

Code VHDL – Compteur 4 bits

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity COUNTER
    port (
      clk, clr, cnt : in STD_LOGIC;
       C   : out STD_LOGIC_VECTOR
    (3 downto 0);
       tc  : out STD_LOGIC);
end COUNTER;
architecture RTL of COUNTER is
signal count :
    UNSIGNED (3 downto 0) :
begin
    process(clk)
    begin
       if rising_edge(clk) then
          if clr = '1' then
             count <= (others => '0');
          elsif cnt = '1' then
             if count = "1111" then
                count <= (others => '0');
             else
                count <= count + 1;
             end if;
          end if;
       end if;
    end process;
    C <= std_logic_vector(count);
    tc <= '1' when count = "1111" else '0';
end RTL;
```

IV Circuits séquentiels

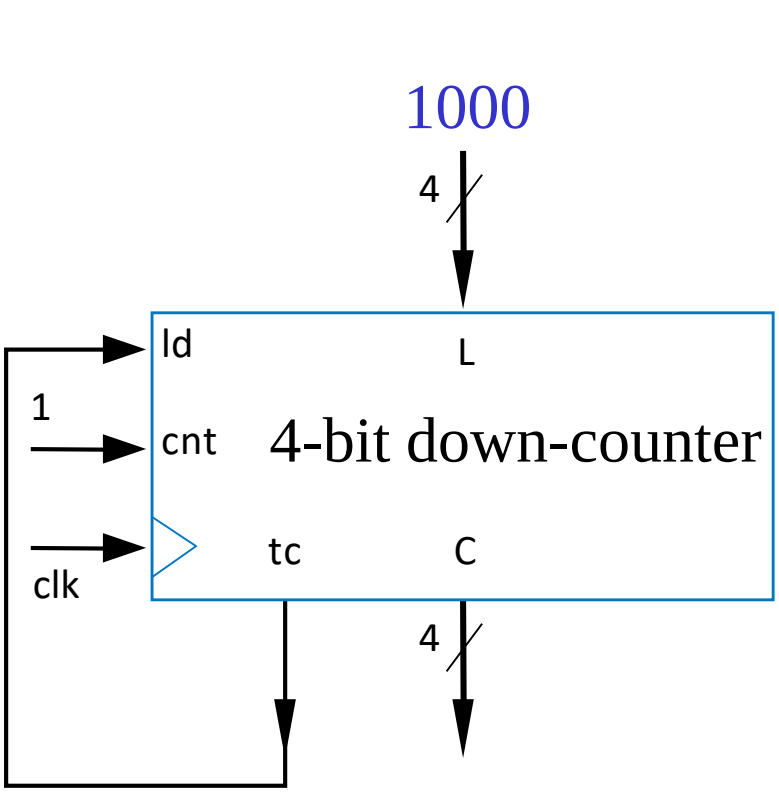
Code Verilog – Registre 4 bits

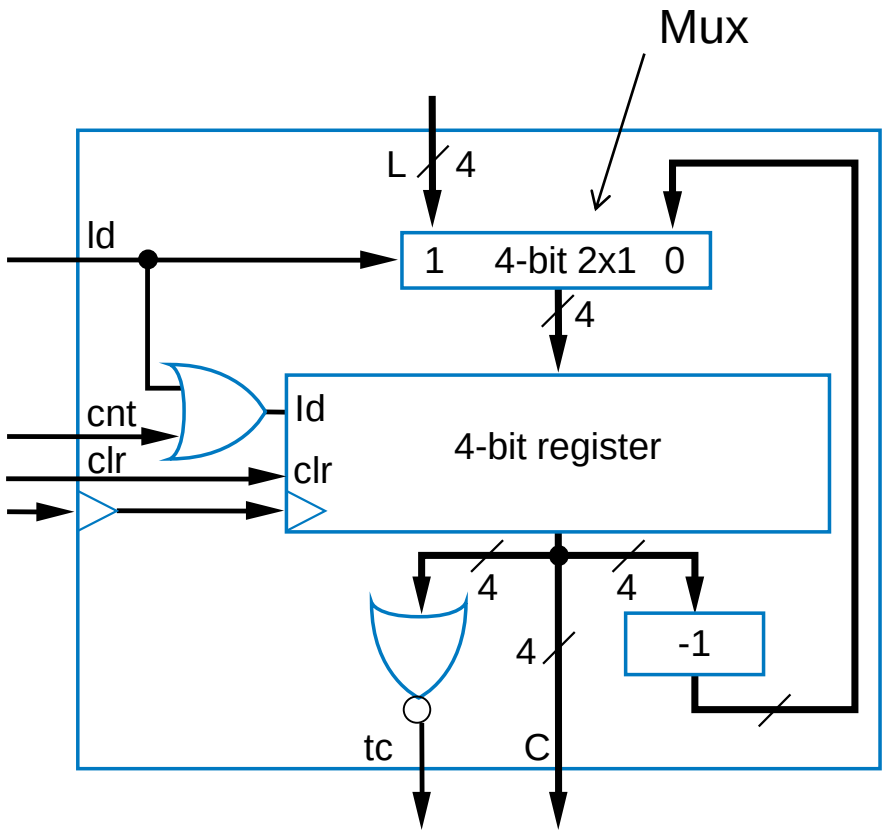
Verilog

```
module COUNTER (
    input wire clk,
    input wire clr,
    input wire cnt,
    output wire [3:0] C,
    output wire tc
);
    reg [3:0] count;
    always @(posedge clk) begin
     if (clr) begin
        count <= 4'b0000;
     end else if (cnt) begin
        if (count == 4'b1111) begin
           count <= 4'b0000;
        end else begin
           count <= count + 1;
        end
     end
    end
    assign C = count;
    assign tc = (count == 4'b1111)
     ? 1'b1 : 1'b0;
endmodule
```

IV Circuits séquentiels

Décompteur avec chargement parallèle





IV Circuits séquentiels

Code VHDL – Décompteur 4 bits

VHDL

```
library ...
entity DownCounter is
port (clk, cnt, clr, load     : in
    std_logic;
L : in std_logic_vector(3 downto 0);
C : out std_logic_vector(3 downto
    0);
tc      : out std_logic);
end DownCounter;
architecture RTL of DownCounter is
signal count : unsigned(3 downto
    0);
begin
process(clk)
begin
    if rising_edge(clk) then
        if clr = '1' then
            count <= (others => '0');
        elsif load = '1' then
            count <= unsigned(L);
        elsif cnt = '1' then
            count <= count - 1;
        end if;
    end if;
end process;
tc <= '1' when count = 0 else '0';
C <= std_logic_vector(count);
end RTL;
```

IV Circuits séquentiels

Code Verilog – Décompteur 4 bits

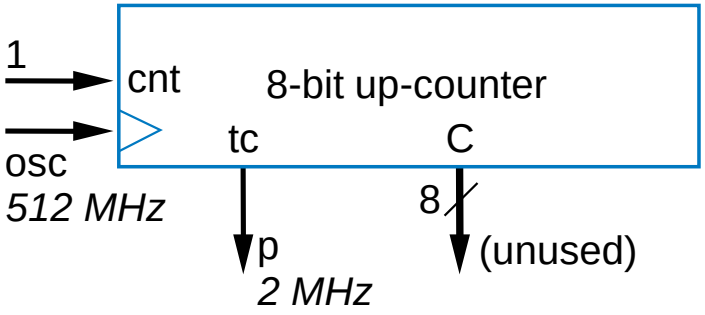
Verilog

```
module DownCounter (
    input wire clk, cnt, load,clr,
    input wire [3:0] L,
    output wire [3:0] C,
    output reg tc
);
    reg [3:0] count;

always @(posedge clk) begin
        if (clr) begin
            count <= 4'b0000;
        end else if (load) begin
            count <= L;
        end else if (cnt) begin
            count <= count - 1;
        end
    end
    always @* begin
        tc = (count == 4'b0000);
    end
    assign C = count;
endmodule
```

IV Circuits séquentiels

Compteur – Diviseur de fréquence

- Si la fréquence de l'horloge est de 512 MHz et que l'on veut générer des impulsions de 2 MHz
  - Il faut diviser la fréquence par 256
  - Utilisation d'un compteur 8-bit module 256 (de 0 à 255)
  - tc est l'impulsion voulue (2 MHz)



IV Circuits séquentiels

Code VHDL – Compteur modulo N

```
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if clr = '1' or (count = N-1 and cnt='1')) then
                count <= (others => '0');
            elsif cnt = '1' then
                    count <= count + 1;
                end if;
            end if;
        end if;
    end process;
    C <= std_logic_vector(count);
    tc <= '1' when count = N-1 else '0';
end RTL;
```

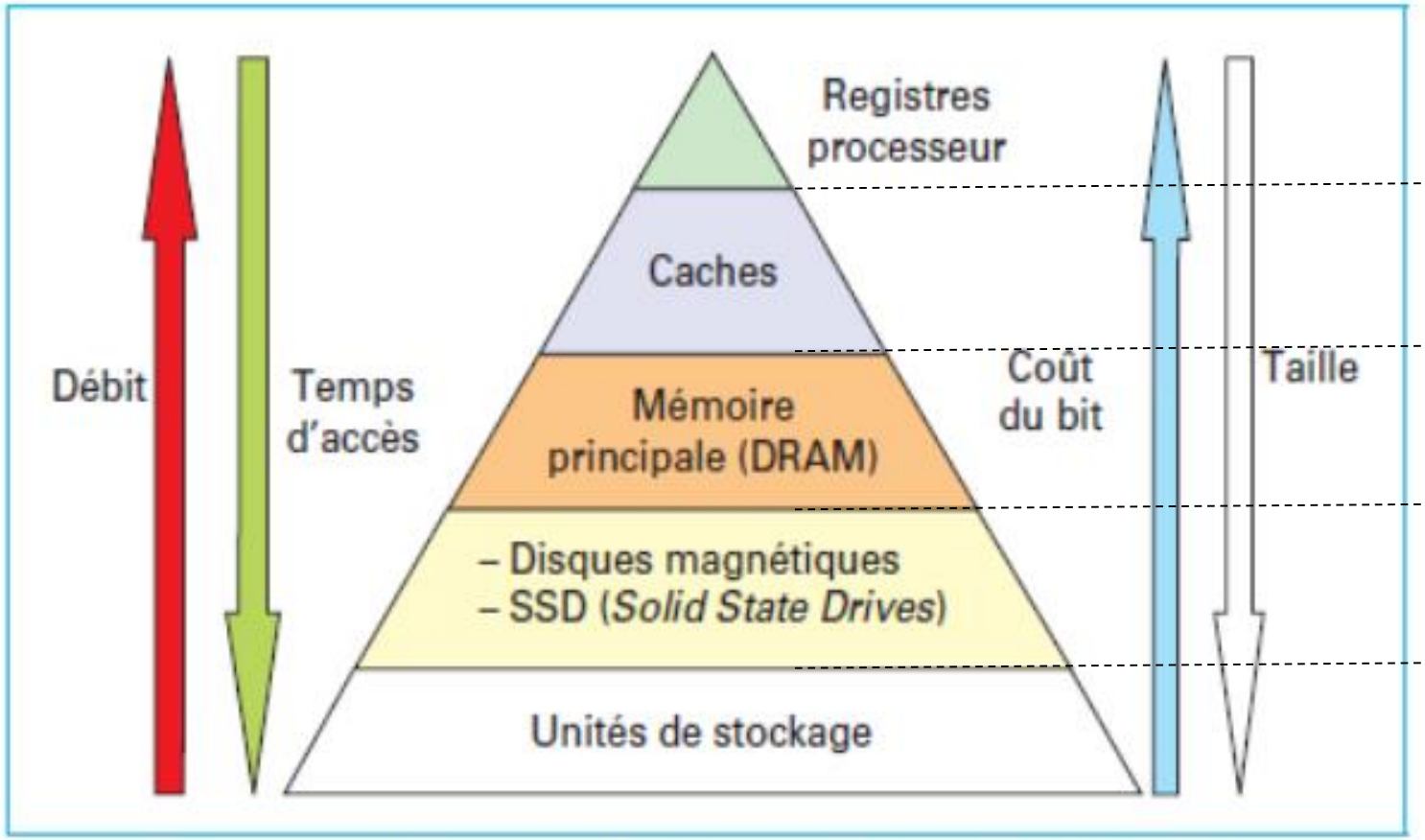
IV Circuits séquentiels

Code Verilog – Compteur modulo N

```
parameter LAST_VALUE = 4'b1110;
    reg [3:0] count;
    always @(posedge clk) begin
        if (clr == 1'b1 || (count == LAST_VALUE && cnt == 1'b1)) begin
            count <= 4'b0000;
        end else if (cnt == 1'b1) begin
            count <= count + 1'b1;
        end
    end
    assign C = count;
    assign tc = (count == LAST_VALUE) ? 1'b1 : 1'b0;
    endmodule
```

IV Circuits séquentiels

Les mémoires



Techniques de l'ingénieur, H1002, hiérarchie mémoire : les caches

IV Circuits séquentiels

Mémoires

- Principe :
  - Choix de l'adresse par le bus d'adresses.
  - Ecriture/lecture des données par un bus données.

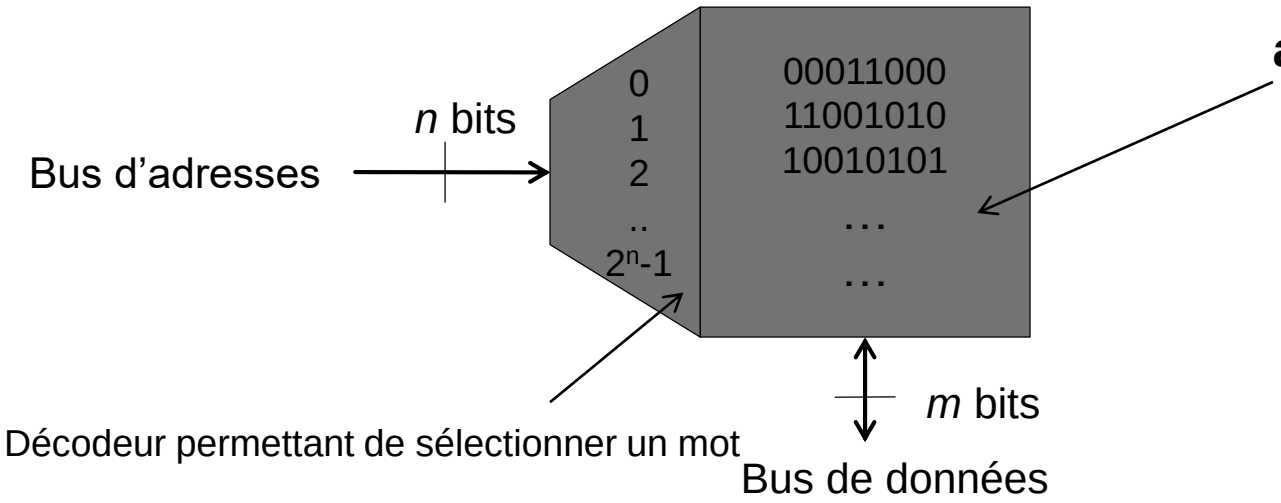


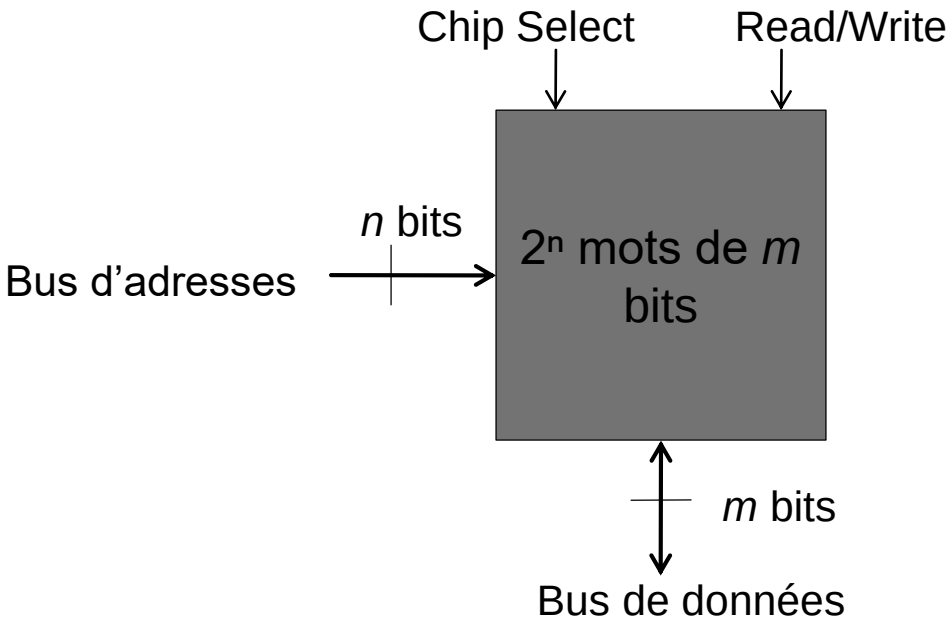
Tableau constitué d'un ensemble de registres à chargement parallèle : **architecture optimisée (très compacte)**

- Bus d'adresses de largeur *n* * Bus données de largeur *m*
  => 2^n mots de *m* bits.

III Circuits séquentiels

Mémoires

- Signaux de contrôle de la lecture/écriture en mémoire :



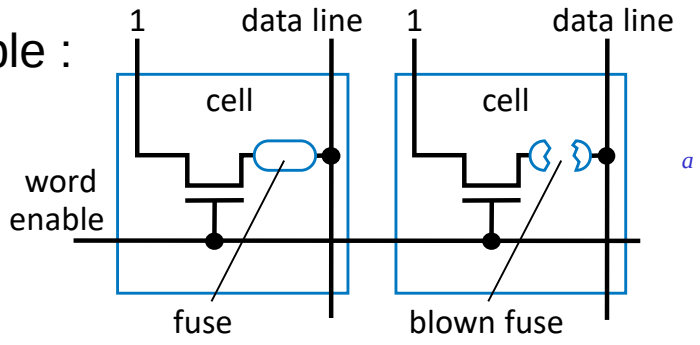
Read/Write	CS	
0	1	Écriture
1	1	Lecture
1/0	0	Rien

IV Circuits séquentiels

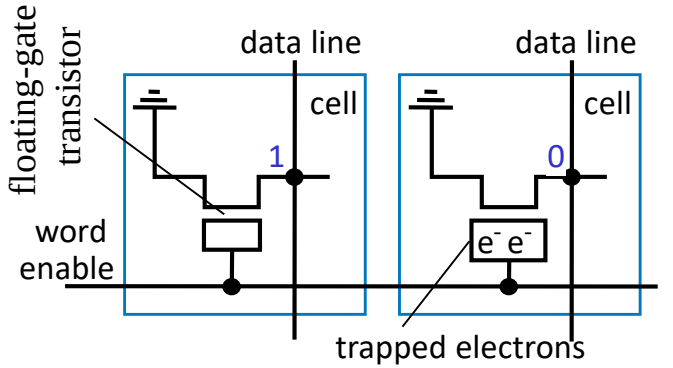
Mémoires ROM

- Les ROMs (mémoires mortes) ⇔ Read Only Memory :
  - Non volatiles (informations mémorisées conservées quand l'alimentation disparait)
  - Pas de signal Read/Write (*chip select* suffit).
  - Ce sont des circuits combinatoires (avec pour entrées les bits d'adresse, et sorties les bits stockés).
- Les différentes ROMs :
  - Non configurables (fusible ou antifusible)
  - Configurables électriquement : EPROMs (programmation définitive).
  - Configurables et effaçable électriquement : EEPROMs (ou Mémoire Flash)
  - Temps d'écriture plus long que pour les RAM

Fusible :



EPROM ou EEPROM :



IV Circuits séquentiels

Code Verilog – modélisation d'une ROM (read only)

```
type rom_array is array (0 to 3) of std_logic_vector(3 downto 0);
signal ROM : rom_array := ( "1110", -- ROM[0]
    "1010", -- ROM[1]
    "0110", -- ROM[2]
    "1100" -- ROM[3] );
begin
process(address)
begin
case address is
    when "00" => data_out <= ROM(0);
    when "01" => data_out <= ROM(1);
    when "10" => data_out <= ROM(2);
    when "11" => data_out <= ROM(3);
    when others => null;
end case;
end process;
```

ROM peut être initialisé avec un fichier binaire data.txt en utilisant les fonctions de lecture read/readline (ou par scripting)

IV Circuits séquentiels

Code Verilog – modélisation d'une ROM (read only)

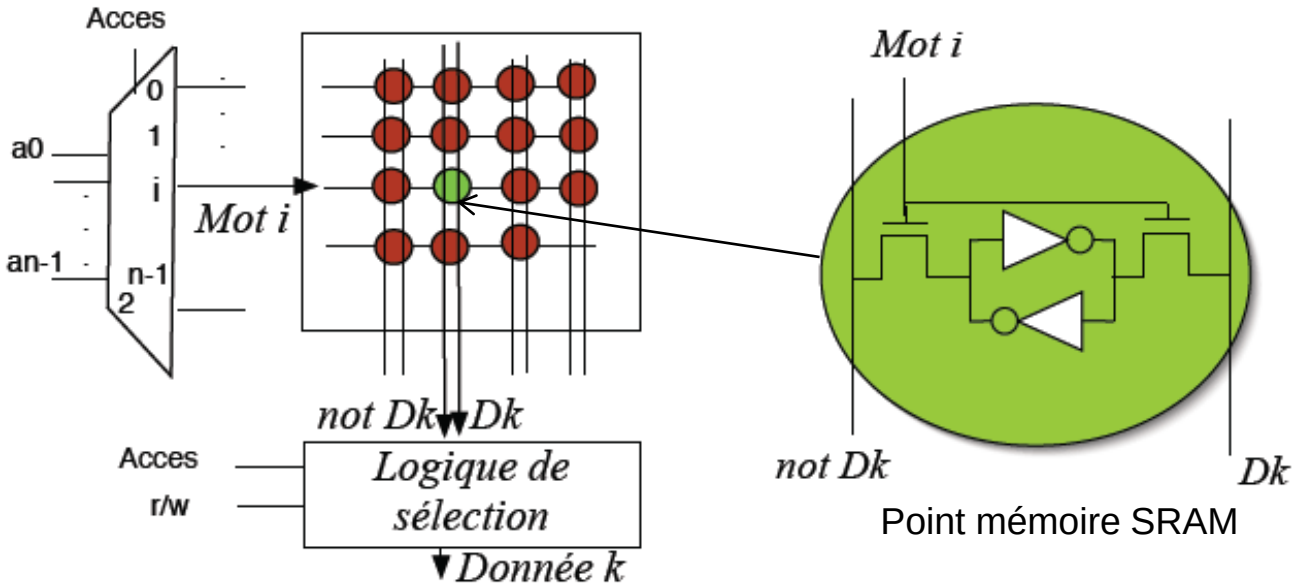
```
module rom_4x4_async
    (output reg [3:0] data_out,
    input wire [1:0] address);
reg[3:0] ROM[0:3]; //déclaration d'un tableau 4x4
initial
    begin
      ROM[0] = 4'b1110;
      ROM[1] = 4'b1010;
      ROM[2] = 4'b0110;
      ROM[3] = 4'b1100;
    end
always @(address)
    data_out = ROM[address];
endmodule
```

ROM peut être initialisé avec un fichier binaire data.txt :
$readmemb("data_bin.txt", memory_bin);
//non synthétisable

IV Circuits séquentiels

Mémoires SRAM

- Mémoire Statique RAMs :



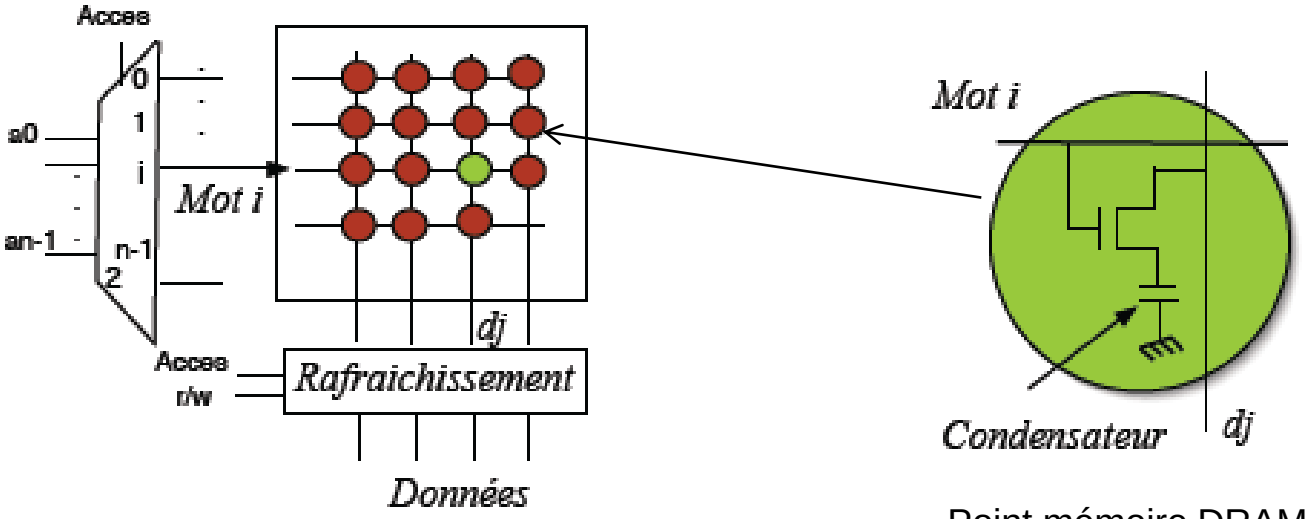
Point mémoire SRAM

Structure robuste aux perturbations

IV Circuits séquentiels

Mémoires DRAM

- Dynamique RAMs :
  - 1 transistor + 1 condensateur = 1 point mémoire
  - Besoin de rafraîchir pour palier à la décharge du condensateur toutes les 2 à 4 ms.
  - Lecture puis re-écriture par circuit spécialisé de rafraîchissement interne à la mémoire
  - Le rafraîchissement consomme ~5% du temps d'accès à la mémoire => long temps d'accès dans le pire cas



Point mémoire DRAM

IV Circuits séquentiels

Code VHDL – modélisation d'une mémoire read/write

```
-- Déclaration de la mémoire RW
type rw_array is array (0 to 3) of std_logic_vector(3 downto 0);
signal RW : rw_array := ( "0000", "0000", "0000", "0000" );
begin
process (address, WE, data_in)
    begin
    if WE = '1' then
     RW(to_integer(unsigned(address))) <= data_in;
    else
     data_out <= RW(to_integer(unsigned(address)));
    end if;
end process;
```

IV Circuits séquentiels

Code Verilog – modélisation d'une mémoire read/write

```
module rw_4x4_async
    (output reg [3:0] data_out,
    input wire [1:0] address,
    input wire WE,
    input wire [3:0] data_in;
reg[3:0] RW[0:3]; //déclaration d'un tableau 4x4
always @(address or WE or data_in)
    if (WE)
     RW[address] = data_in;
    else
     data_out = RW[address];
endmodule
```

IV Circuits séquentiels

Mémoires RAM

- Les RAMs (mémoires vives) ⇔ Random Acces Memory
  - Volatile (les informations mémorisées disparaissent quand l'alimentation disparait)
  - Registres stockant un grand nombre de mots

- Mémoires Statiques RAMs (SRAM) :
  - Réalisation avec deux inverseurs (voir D latch)
  - En tout 6 transistors pour stocker un bit (voir suite)
  - Rapide => utilisation en cache

- Mémoires Dynamiques RAMs (DRAM) :
  - Moins couteuses mais moins rapides que la SRAM
  - Meilleure densité d'intégration

Faire le QCM : QCM4 SN360/SN361 Circuits séquentiels