VHDL - Logique programmable

Partie 6
• Systèmes combinatoires
• Systèmes séquentiels

Denis Giacona

ENSISA
École Nationale Supérieure d'Ingénieur Sud Alsace
12, rue des frères Lumière
68 093 MULHOUSE CEDEX
FRANCE

Tél. 33 (0)3 89 33 69 00

© *D.Giacona*

1. Systèmes combinatoires 4
1.1. Styles de description 4
1.2. Blocs combinatoires standard 5
1.3. Fonctions arithmétiques à l'aide d'opérateurs arithmétiques 6
1.3.1. Addition de 2 nombres de 8 bits 7
1.3.2. Surcharge de l'opérateur + ... 8
1.3.3. Addition de 4 nombres de 8 bits 10
1.3.4. Multiplication non signée et multiplication signée 12
1.3.5. Multiplication non signée 2bits * 2bits, résultat sur 4bits 13
1.3.6. Multiplication signée 8bits * 8bits, résultat sur 16bits 14
1.4. Fonctions arithmétiques à l'aide d'opérateurs de décalage 15
1.5. Comparaisons à l'aide d'opérateurs relationnels 16
1.6. Multiplexage 17
1.6.1. Multiplexage : processus avec instruction `case` 17
1.6.2. Multiplexage : processus avec fonction de conversion 18
1.6.3. Multiplexage : avec sortie 3 états 19
1.7. Partage de ressources logiques 20
1.8. Décodage 22
1.8.1. Décodage : processus avec instruction `case` 22
1.8.2. Décodage d'adresses 23
1.9. Décalage combinatoire 24
2. Systèmes séquentiels 25
2.1. Bloc logique séquentiel asynchrone 25
2.2. Bloc logique séquentiel synchrone 25
2.3. La mémorisation implicite 26
2.4. Description comportementale de la logique séquentielle asynchrone 27
2.4.1. Exemple de description à l'aide d'un processus 27
2.5. Description comportementale de la logique séquentielle synchrone 29
2.5.1. Description incorrecte d'un flip-flop D avec reset asynchrone 29
2.5.2. Méthode de synchronisation correcte 31
2.5.3. reset et preset asynchrones 32
2.5.4. reset et preset synchrones 33

© D.Giacona

2.6. Exemples de blocs séquentiels 34
2.6.1. D flip-flop (rising edge) 34
2.6.2. D flip-flop with enable 36
2.6.3. D flip-flop with Active High Synchronous Preset 40
2.6.4. D flip-flop with Active Low Synchronous Reset 41
2.6.5. D latch (level sensitive) 42
2.6.6. D latch (level sensitive) 43
2.6.7. RS latch (entrées actives à l'état bas) 44
2.6.8. RS latch (entrées actives à l'état bas) 45
2.6.9. RS latch (entrées actives à l'état bas) 46
2.6.10. Registre D synchrone, 8 bits, avec reset asynchrone 47
2.6.11. Registre D synchrone, 8 bits, avec reset synchrone 48
2.6.12. Registre D synchrone, 8 bits, avec reset synchrone et entrée de validation 49
2.6.13. Registre à décalage à droite 4 bits, avec reset asynchrone 51
2.6.14. Registre à décalage à droite 12 bits, avec reset asynchrone 53
2.6.15. Compteur BCD 4 bits, reset asynchrone, validation de comptage, sortie retenue à assignation combinatoire 54
2.6.16. Compteur BCD 4 bits, reset asynchrone, validation de comptage, sortie retenue à assignation combinatoire 58
2.6.17. Compteur BCD 4 bits, reset asynchrone, validation de comptage, sortie retenue synchronisée 61
2.6.18. Compteur binaire naturel 4 bits, synchrone, chargeable, sorties 3 états 66
2.6.19. Compteur 8 bits à commandes multiples 67

© D.Giacona

1.Systèmes combinatoires

1.1. Styles de description

La logique combinatoire peut être décrite dans tous les styles

- **flot de données**
- **structurelle**
- **comportementale**

```
architecture nom_architecture of nom_entité is

  begin

      { instruction_concurrente_d'assignation_de_signal
      | instruction_concurrente_d'instanciation_de_composant
      | instruction_concurrente_de_processus
      | instruction_de_génération}

  end [architecture] [nom_architecture];
```


© D.Giacona

1.2. Blocs combinatoires standard

	Instruction `case`	Instruction `if… elsif … else …`	Assignation conditionnelle `when…else…`	Assignation sélective `with...select…`	Description structurelle	Opérateur arithmétique
Additionneur					x	x
Soustracteur					x	x
Multiplexeur	x	x	x	x		
Démultiplexeur	x	x	x	x		
Codeur	x	x		x		
Décodeur	x	x		x		
Comparateur		x	x			
Multiplieur					x	x

© D.Giacona

1.3. Fonctions arithmétiques à l'aide d'opérateurs arithmétiques

Les opérateurs + (addition) et * (multiplication) sont initialement définis pour des données de type entier.

Lorsque le compilateur rencontre l'opérateur + avec de part et d'autre un signal de type `std_logic_vector`, il génère une erreur, sauf si on ouvre la bibliothèque contenant la surcharge de cet opérateur à l'aide des directives suivantes :

```
library ieee;
use ieee.std_logic_unsigned.all;
```


© D.Giacona

1.3.1. Addition de 2 nombres de 8 bits

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity add8_v2 is port(
    a   : in std_logic_vector(7 downto 0);
    b   : in std_logic_vector(7 downto 0);
    sum : out std_logic_vector(7 downto 0));
end add8_v2;

architecture arch_add8_v2 of add8_v2 is
begin

 sum <= a + b;

end arch_add8_v2;
```

```
DESIGN EQUATIONS FOR A CPLD

sum(0) =
          /a(0) * b(0)
        + a(0) * /b(0)

sum(1) =
          a(0) * a(1) * b(0) *
b(1)
        + a(0) * /a(1) * b(0) *
/b(1)
        + /a(0) * /a(1) * b(1)
        + /a(1) * /b(0) * b(1)
        + /a(0) * a(1) * /b(1)
        + a(1) * /b(0) * /b(1)
```


© D.Giacona

1.3.2. Surcharge de l'opérateur +

```
function "+" (L, R: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR;
  -- Result subtype: STD_LOGIC_VECTOR(MAX(L'LENGTH, R'LENGTH)-1 downto 0).
  -- Result: Adds two STD_LOGIC_VECTOR vectors that may be of different lengths.

function "+" (L, R: STD_LOGIC_VECTOR) return STD_LOGIC_VECTOR is
    constant SIZE: NATURAL := MAX(L'LENGTH, R'LENGTH);
    variable L01 : STD_LOGIC_VECTOR(SIZE-1 downto 0);
    variable R01 : STD_LOGIC_VECTOR(SIZE-1 downto 0);
  begin
    if ((L'LENGTH < 1) or (R'LENGTH < 1)) then return NAU;
    end if;
    L01 := TO_01(RESIZE(L, SIZE), 'X');
    if (L01(L01'LEFT)='X') then return L01;
    end if;
    R01 := TO_01(RESIZE(R, SIZE), 'X');
    if (R01(R01'LEFT)='X') then return R01;
    end if;
    return ADD_STD_LOGIC_VECTOR(L01, R01, '0');
  end "+";

-- this internal function computes the addition of two STD_LOGIC_VECTOR
  -- with input CARRY
  -- * the two arguments are of the same length
```

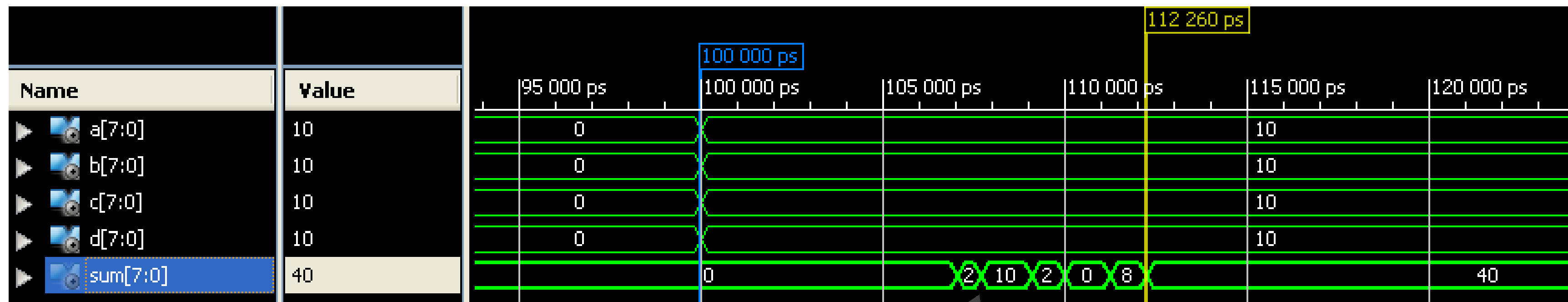

© D.Giacona

```
function ADD_STD_LOGIC_VECTOR (L, R: STD_LOGIC_VECTOR; C: STD_LOGIC) return STD_LOGIC_VECTOR is
  constant L_LEFT: INTEGER := L'LENGTH-1;
  alias XL: STD_LOGIC_VECTOR(L_LEFT downto 0) is L;
  alias XR: STD_LOGIC_VECTOR(L_LEFT downto 0) is R;
  variable RESULT: STD_LOGIC_VECTOR(L_LEFT downto 0);
  variable CBIT: STD_LOGIC := C;
begin
  for I in 0 to L_LEFT loop
    RESULT(I) := CBIT xor XL(I) xor XR(I);
    CBIT := (CBIT and XL(I)) or (CBIT and XR(I)) or (XL(I) and XR(I));
  end loop;
  return RESULT;
end ADD_STD_LOGIC_VECTOR;
```

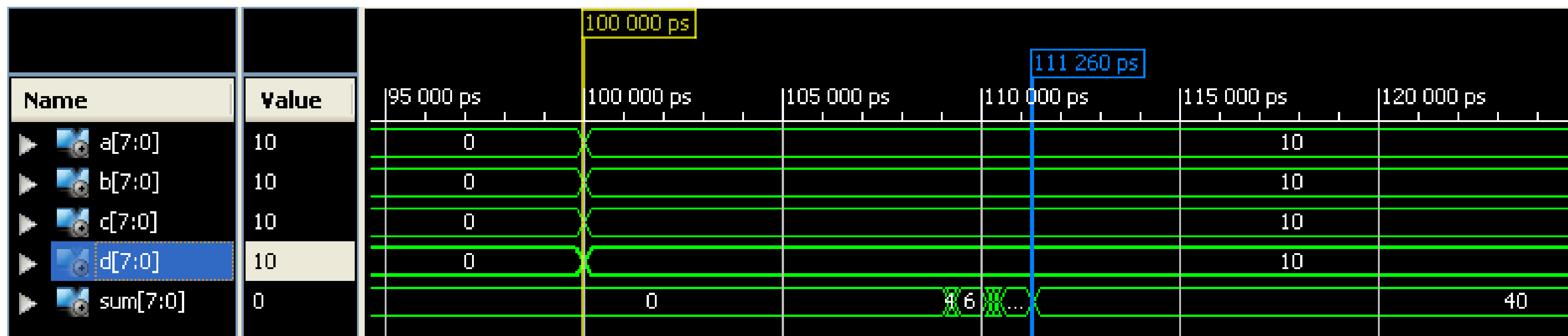

© D.Giacona

1.3.3. Addition de 4 nombres de 8 bits

```
sum <= a + b + c + d;
```

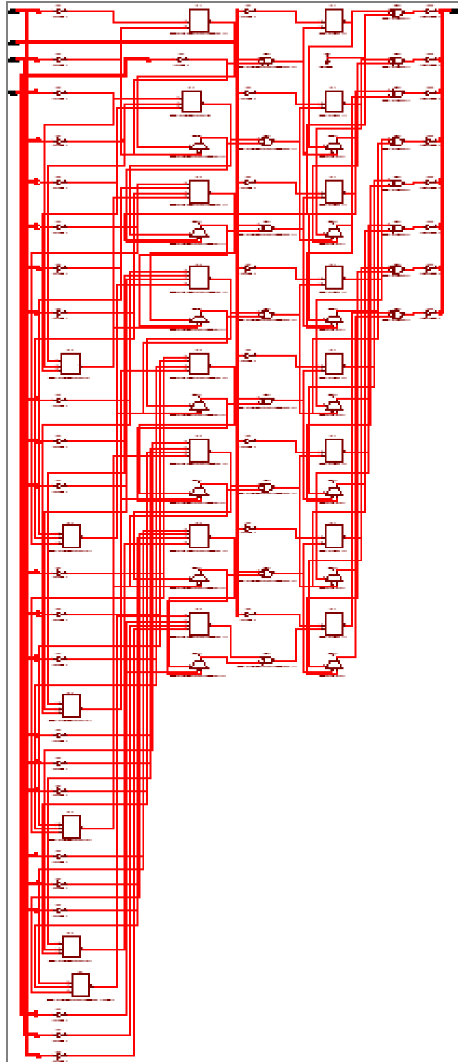



```
sum <= (a + b) + (c + d);      -- L'usage des parenthèses a une incidence sur la synthèse.
                               -- Cette description génère un additionneur plus rapide.
```



© D.Giacona

`sum <= a + b + c + d;`



`sum <= (a + b) + (c + d);`

© D.Giacona

1.3.4. Multiplication non signée et multiplication signée

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;       -- contient définitions du type signed et opérateur *
use ieee.std_logic_unsigned.all;    -- contient définition opérateur * pour std_logic_vector

entity nombres_signes is port(
  x   : out std_logic_vector(7 downto 0);
  y   : out std_logic_vector(7 downto 0));
end nombres_signes;

architecture arch_nombres_signes of nombres_signes is
     signal x1_int, x2_int : std_logic_vector(3 downto 0);
     signal y1_int, y2_int : signed(3 downto 0);
     signal y_int : signed(7 downto 0);
begin

-- La multiplication avec * est non signée si le type est : std_logic_vector
x1_int    <= "1111";          -- Valeur : 15
x2_int    <= "1111";
x         <= x1_int * x2_int; -- Résultat : 225

-- La multiplication avec * est signée si le type est : signed
y1_int    <= "1111";          -- Valeur : -1
y2_int    <= "1111";
y_int     <= signed(y1_int) * signed(y2_int);     -- Résultat : + 1
y         <= std_logic_vector(y_int);

end arch_nombres_signes;
```

x(0) = VCC
x(1) = GND
x(2) = GND
x(3) = GND
x(4) = GND
x(5) = VCC
x(6) = VCC
x(7) = VCC

y(0) = VCC
y(1) = GND
y(2) = GND
y(3) = GND
y(4) = GND
y(5) = GND
y(6) = GND
y(7) = GND

© D.Giacona

1.3.5. Multiplication non signée 2bits * 2bits, résultat sur 4bits

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_unsigned.all;

entity mult_2bits is port(
  a, b : in std_logic_vector(1 downto 0);
  y    : out std_logic_vector(3 downto 0));
end mult_2bits;


architecture arch_mult_2bits of mult_2bits is
begin

  y <= a * b;

end arch_mult_2bits;
```

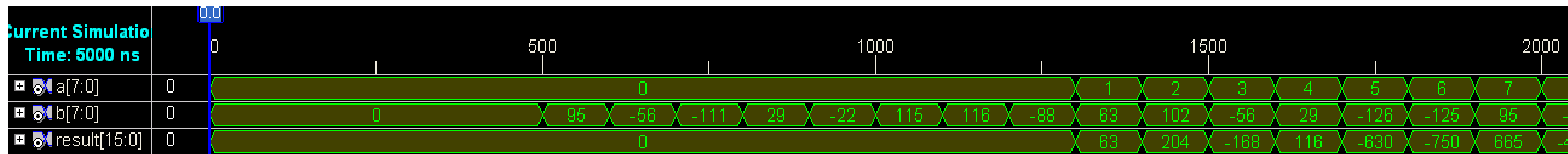
```
DESIGN EQUATIONS

    y(0) =
          a(0) * b(0)

    y(1) =
          /a(0) * a(1) * b(0)
        + a(0) * /a(1) * b(1)
        + a(0) * /b(0) * b(1)
        + a(1) * b(0) * /b(1)

    y(2) =
          /a(0) * a(1) * b(1)
        + a(1) * /b(0) * b(1)

    y(3) =
          a(0) * a(1) * b(0) * b(1)
```

UHA Université de Haute-Alsace Mulhouse-Colmar — ensisa école nationale supérieure d'ingénieurs sud alsace

© D.Giacona

1.3.6. Multiplication signée 8bits * 8bits, résultat sur 16bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY signed_mult IS
     PORT (
          a:        IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          b:        IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          result: OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
     );
END signed_mult;

ARCHITECTURE rtl OF signed_mult IS
     SIGNAL a_int, b_int:      SIGNED (7 downto 0);
     SIGNAL pdt_int:           SIGNED (15 downto 0);

BEGIN
     a_int <= SIGNED(a);       -- appel à une fonction de conversion
     b_int <= SIGNED(b);
     pdt_int <= a_int * b_int;
     result <= STD_LOGIC_VECTOR(pdt_int);
END rtl;
```



© D.Giacona

1.4. Fonctions arithmétiques à l'aide d'opérateurs de décalage

❒ Fonctions de décalages (`shr` et `shl`) et de concaténations (`&`) de vecteurs de bits pour effectuer des multiplications et des divisions par des puissances de 2

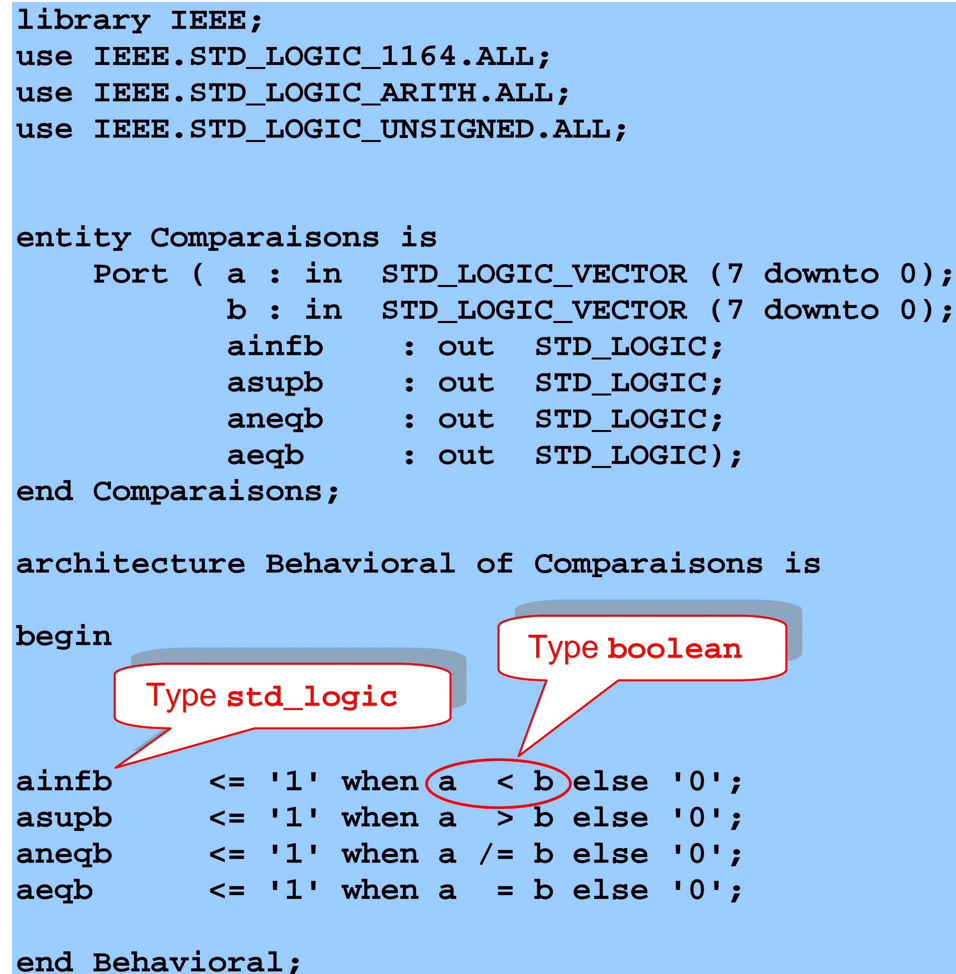
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity OperationsArithmetiques is
    Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           x1 : out  STD_LOGIC_VECTOR (7 downto 0);
           x2 : out  STD_LOGIC_VECTOR (7 downto 0);
           x3 : out  STD_LOGIC_VECTOR (7 downto 0);
           x4 : out  STD_LOGIC_VECTOR (7 downto 0);
           x5 : out  STD_LOGIC_VECTOR (7 downto 0));
end OperationsArithmetiques;

architecture Behavioral of OperationsArithmetiques is
begin

  x1 <= "00" & a(7 downto 2);       -- décalage à droite de 2 positions (division par 4)
  x2 <= shr(a,"101");               -- décalage à droite de 5 positions (shift right function)
  x3 <= b(5 downto 0) & "00";       -- décalage à gauche de 2 positions (multiplication par 4)
  x4 <= shl(b,"11");                -- décalage à gauche de 3 positions (shift left function)
  x5 <= shr(a,"10") + shl(b,"10");

end Behavioral;
```

Name	Value	999 997 ps
a[7:0]	00111100	00111100
b[7:0]	00000101	00000101
x1[7:0]	00001111	00001111
x2[7:0]	00000001	00000001
x3[7:0]	00010100	00010100
x4[7:0]	00101000	00101000
x5[7:0]	00100011	00100011

© D.Giacona

1.5. Comparaisons à l'aide d'opérateurs relationnels

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity Comparaisons is
    Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           ainfb    : out  STD_LOGIC;
           asupb    : out  STD_LOGIC;
           aneqb    : out  STD_LOGIC;
           aeqb     : out  STD_LOGIC);
end Comparaisons;

architecture Behavioral of Comparaisons is

begin


ainfb     <= '1' when a  < b else '0';
asupb     <= '1' when a  > b else '0';
aneqb     <= '1' when a /= b else '0';
aeqb      <= '1' when a  = b else '0';

end Behavioral;
```



1.6. Multiplexage

1.6.1. Multiplexage : processus avec instruction case

```
library ieee;
    use ieee.std_logic_1164.all;
    USE ieee.std_logic_arith.ALL;

entity mux4to1_v8 is port(
  a : in std_logic_vector(3 downto 0);
  s : in unsigned(1 downto 0);
  x : out std_logic);
end mux4to1_v8;

architecture arch_mux4to1_v8 of mux4to1_v8 is
begin
  process (s,a)
    variable i : integer range 0 to 3;
  begin
    i := conv_integer(s);  -- fonction de conversion vecteur de bits vers entier
    case i is
      when 0 => x <= a(0);
      when 1 => x <= a(1);
      when 2 => x <= a(2);
      when 3 => x <= a(3);
    end case;
  end process;
end arch_mux4to1_v8;
```


© D.Giacona

1.6.2. Multiplexage : processus avec fonction de conversion

```
library ieee;
  use ieee.std_logic_1164.all;
  use ieee.std_logic_unsigned.all;

entity mux16to1_v1 is port(
  a : in std_logic_vector(15 downto 0);
  s : in std_logic_vector(3 downto 0);
  x : out std_logic);
end mux16to1_v1;

architecture arch_mux16to1_v1 of mux16to1_v1 is
begin
  process (s,a)
    variable i : integer range 0 to 15;
  begin
    i := conv_integer(s);
    x <= a(i);
  end process;

end arch_mux16to1_v1;
```

```
DESIGN EQUATIONS
  x =
      a(15) * s(0) * s(1) * s(2) * s(3)
    + a(14) * /s(0) * s(1) * s(2) * s(3)
    + a(13) * s(0) * /s(1) * s(2) * s(3)
    + a(12) * /s(0) * /s(1) * s(2) * s(3)
    + a(11) * s(0) * s(1) * /s(2) * s(3)
    + a(10) * /s(0) * s(1) * /s(2) * s(3)
    + a(9) * s(0) * /s(1) * /s(2) * s(3)
    + a(8) * /s(0) * /s(1) * /s(2) * s(3)
    + a(7) * s(0) * s(1) * s(2) * /s(3)
    + a(6) * /s(0) * s(1) * s(2) * /s(3)
    + a(5) * s(0) * /s(1) * s(2) * /s(3)
    + a(4) * /s(0) * /s(1) * s(2) * /s(3)
    + a(3) * s(0) * s(1) * /s(2) * /s(3)
    + a(2) * /s(0) * s(1) * /s(2) * /s(3)
    + a(1) * s(0) * /s(1) * /s(2) * /s(3)
    + a(0) * /s(0) * /s(1) * /s(2) * /s(3)
```


© D.Giacona

1.6.3. Multiplexage : avec sortie 3 états

```
library ieee;
  use ieee.std_logic_1164.all;

entity mux4to1_v9 is port(
    a, b, c, d : in std_logic;
    oe         : in std_logic;    -- output enable
    s          : in std_logic_vector(1 downto 0);
    x          : out std_logic);
end mux4to1_v9;

architecture arch_mux4to1_v9 of mux4to1_v9 is
begin

x <= 'Z' when oe = '0' else
       a when s  = 0 else
       b when s  = 1 else
       c when s  = 2 else
       d;

end arch_mux4to1_v9;
```

```
DESIGN EQUATIONS
    x =    d * s(0) * s(1)
         + c * /s(0) * s(1)
         + b * s(0) * /s(1)
         + a * /s(0) * /s(1)

    x.OE = oe
```

UHA Université de Haute-Alsace Mulhouse-Colmar — ensisa école nationale supérieure d'ingénieurs sud alsace

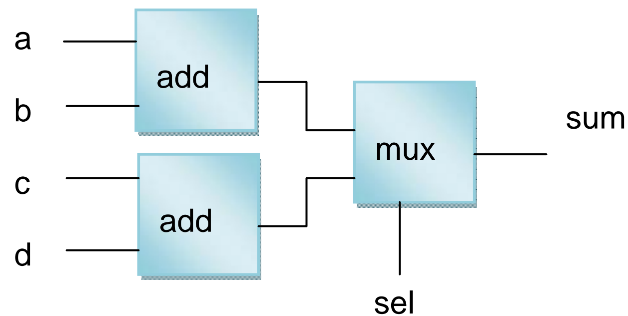
© D.Giacona

1.7. Partage de ressources logiques

❒ Cahier des charges

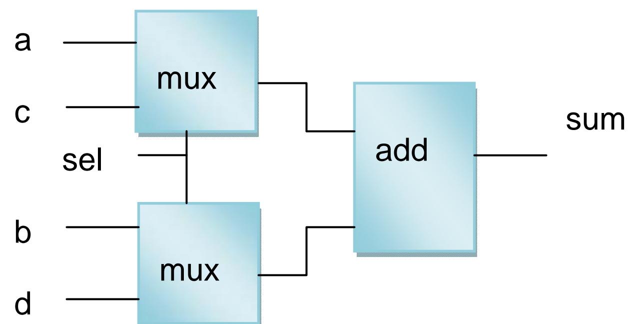
sum = a + b si sel = '0'
sum = c + d si sel = '1'

☹ Solution 1



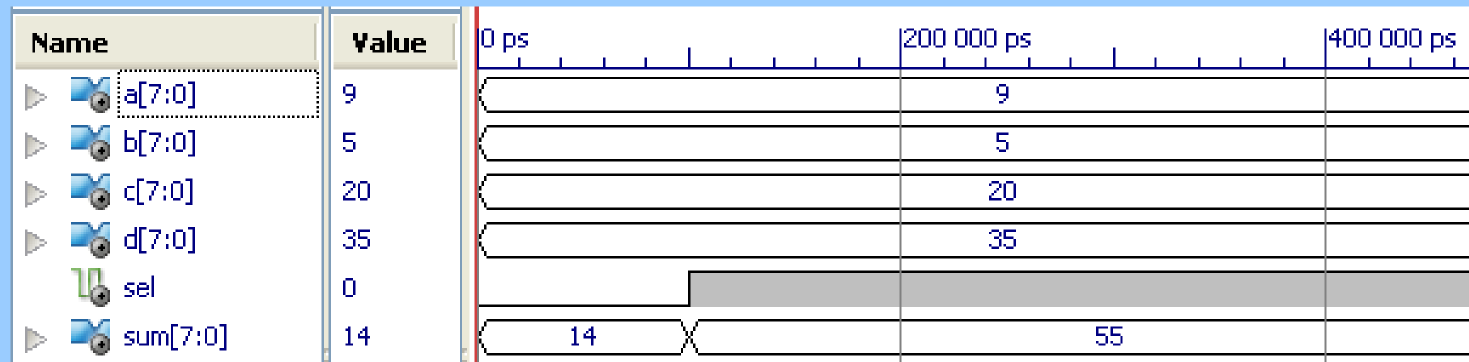
☺ Solution 2 (plus économique)

Un additionneur consomme plus de ressources logiques qu'un multiplexeur !



© D.Giacona

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PartageRessources is
    Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
           b : in  STD_LOGIC_VECTOR (7 downto 0);
           c : in  STD_LOGIC_VECTOR (7 downto 0);
           d : in  STD_LOGIC_VECTOR (7 downto 0);
           sel : in  STD_LOGIC;
           sum : out  STD_LOGIC_VECTOR (7 downto 0));
end PartageRessources;

architecture Behavioral of PartageRessources is
     signal op1, op2 : std_logic_vector(7 downto 0);
begin

process (a,b,c,d,sel)
begin
     if sel = '0' then
          op1 <= a;
          op2 <= b;
     else
          op1 <= c;
          op2 <= d;
     end if;
end process;

     sum <= op1 + op2;

end Behavioral;
```



1.8. Décodage

1.8.1. Décodage : processus avec instruction case

```
library ieee;
use ieee.std_logic_1164.all;

entity dec1of8 is port(
  en : in std_logic;        -- enable (entrée de validation)
  a  : in std_logic_vector(2 downto 0);
  y  : out std_logic_vector(7 downto 0));
end dec1of8;

architecture arch_dec1of8 of dec1of8 is
begin
decode: process(en, a)
begin
  if en = '0' then
    y <= x"00";
  else
    case a is
      when "000" => y <= x"01";
      when "001" => y <= x"02";
      when "010" => y <= x"04";
      when "011" => y <= x"08";
      when "100" => y <= x"10";
      when "101" => y <= x"20";
      when "110" => y <= x"40";
      when "111" => y <= x"80";
      when others => y <= x"00";
    end case;
  end if;
end process;
end arch_dec1of8;
```

```
DESIGN EQUATIONS
    y(0) = /a(0) * /a(1) * /a(2) * en

    y(1) = a(0) * /a(1) * /a(2) * en

    y(2) = /a(0) * a(1) * /a(2) * en

    y(3) = a(0) * a(1) * /a(2) * en

    y(4) = /a(0) * /a(1) * a(2) * en

    y(5) = a(0) * /a(1) * a(2) * en

    y(6) = /a(0) * a(1) * a(2) * en

    y(7) = a(0) * a(1) * a(2) * en
```


© D.Giacona

1.8.2. Décodage d’adresses

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity decodeur_adresse is port(
  en : in std_logic;
  a  : in unsigned(3 downto 0);
  ad_0to3 : out std_logic;
  ad_6to7 : out std_logic;
  ad_8to15 : out std_logic);
end decodeur_adresse;

architecture arch_decodeur_adresse of decodeur_adresse is
begin
decode: process(en, a)
begin
ad_0to3 <= '0';
ad_6to7 <= '0';
ad_8to15 <= '0';

if en = '0' then
  null;
else
  case conv_integer(a) is
    when 0 to 3 => ad_0to3 <= '1';
    when 6 to 7 => ad_6to7 <= '1';
    when 8 to 15 => ad_8to15 <= '1';
    when others => null;
  end case;
end if;
end process;
end arch_decodeur_adresse;
```

```
DESIGN EQUATIONS

ad_0to3 = /a(2) * /a(3) * en

ad_6to7 = a(1) * a(2) * /a(3) * en

ad_8to15 = (3) * en
```


© D.Giacona

1.9. Décalage combinatoire

```
library ieee;
use ieee.std_logic_1164.all;

entity decalage is port(
  dg, dd : in std_logic;
  a : in bit_vector(7 downto 0);
  x : out bit_vector(7 downto 0));
end decalage;


architecture arch_decalage of decalage is
begin

x <= a sla 1 when (dg,dd) = ('1','0') else -- décalage arithmétique gauche
     a sra 1 when (dg,dd) = ('0','1') else -- décalage arithmétique droite
     a;

end arch_decalage;
```

DESIGN EQUATIONS

```
x(0) = a(1) * dd * /dg
   + a(0) * dg
   + a(0) * /dd

x(1) = a(1) * dd * dg
   + a(0) * /dd * dg
   + a(2) * dd * /dg
   + a(1) * /dd * /dg

x(2) = a(2) * dd * dg
   + a(1) * /dd * dg
   + a(3) * dd * /dg
   + a(2) * /dd * /dg

x(3) = a(3) * dd * dg
   + a(2) * /dd * dg
   + a(4) * dd * /dg
   + a(3) * /dd * /dg

x(4) = a(4) * dd * dg
```

✋ **Ces opérateurs ne fonctionnent que sur le type `bit_vector`**

© D.Giacona

2.Systèmes séquentiels

2.1. Bloc logique séquentiel asynchrone

- **Un bloc logique séquentiel asynchrone est activé dès que l'une quelconque de ses entrées change d'état**

- **Styles de description**
  - **comportementale**
  - **structurelle**
  - **flot de données**

2.2. Bloc logique séquentiel synchrone

- **Un bloc logique séquentiel synchrone est activé sur occurrence d'un front d'horloge et non pas sur un changement d'état de l'une quelconque de ses entrées**

- **Conception synchrone ⇔ une seule horloge et un seul front**

- **Styles de description**
  - **comportementale (processus avec signal d'horloge)**
  - **structurelle (avec des composants comportant des processus)**

© D.Giacona

2.3. La mémorisation implicite

- **En VHDL, les signaux ont une valeur courante et une valeur prochaine déterminée par l'opérateur d'assignation.**

- **Soit une instruction conditionnelle en mode concurrent (`when…else…`) ou en mode séquentiel (`if…then…else…`, `case…when`) :**
  - **Description exhaustive**
    - **Tout signal assigné dans une branche reçoit aussi une assignation dans toutes les autres branches**
  - **Description incomplète**
    - **Si un signal est assigné dans une branche, chaque absence d'assignation de ce même signal dans une autre branche signifie que la prochaine valeur du signal est identique à la valeur courante. C'est l'effet mémoire, propre aux systèmes séquentiels.**

© D.Giacona

2.4. Description comportementale de la logique séquentielle asynchrone

2.4.1. Exemple de description à l'aide d'un processus

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

Il n'y a pas de clause `else`
➔ **mémorisation implicite**

- **Activation du processus**

Liste de sensibilité **(e,d)** :

➔ le processus est activé lorsque **e** ou **d** change d'état
➔ tant que **e** et **d** ne changent pas d'état, **q** ne change pas d'état

© D.Giacona

• Exécution du processus

Instruction `if e = '1' then q <= d; end if;`

↑`d`, `e = '0'` → (condition = `false`) → <u>mémorisation implicite</u>
↑`d`, `e = '1'` → (condition = `true`) → `q` prend la valeur de `d` (c.-à-d. `'1'`)

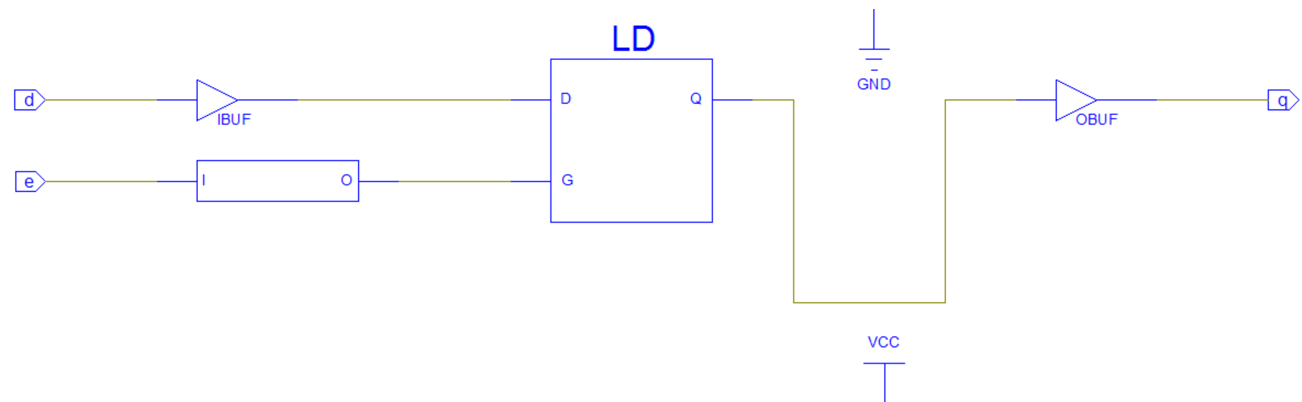
↓`d`, `e = '0'` → (condition = `false`) → <u>mémorisation implicite</u>
↓`d`, `e = '1'` → (condition = `true`) → `q` prend la valeur de `d` (c.-à-d. `'0'`)

↑`e`, `d = '0'` → (condition = `true`) → `q` prend la valeur de `d` (c.-à-d. `'0'`)
↑`e`, `d = '1'` → (condition = `true`) → `q` prend la valeur de `d` (c.-à-d. `'1'`)

↓`e`, `d = '0'` → (condition = `false`) → <u>mémorisation implicite</u>
↓`e`, `d = '1'` → (condition = `false`) → <u>mémorisation implicite</u>

• Interprétation

Il s'agit du comportement d'un **latch D**.



© D.Giacona

2.5. Description comportementale de la logique séquentielle synchrone

2.5.1. Description incorrecte d'un flip-flop D avec reset asynchrone

```
process (ar,clk)
begin
  if ar = '1' then          -- condition 1 (prioritaire)
    q <= '0';
  elsif clk = '1' then      -- condition 2
    q <= d;
  end if;
end process;
```

● **Activation du processus**

Liste de sensibilité **(ar,clk)** :

➔ le processus est activé lorsque **ar** ou **clk** change d'état
➔ tant que **ar** et **clk** ne changent pas d'état, **q** ne change pas d'état

© D.Giacona

- **Exécution du processus pour le cas particulier suivant :**

↓**`ar`** , **`clk`** = **`'1'`** → (condition1 = **`false`**) et (condition2 = **`true`**) → **`q`** prend la valeur de **`d`**

- **Interprétation**

Ce cas de figure montre que ce n'est pas le comportement d'un flip-flop D, puisque la sortie `q` réagit sur un simple état `'1'` de l'horloge `clk` et non pas sur un front montant.

© *D.Giacona*

2.5.2. Méthode de synchronisation correcte

Les actions synchrones sont déclenchées sur un front actif de l'horloge (*edge sensitive*) que l'on peut spécifier de deux façons :

❒ **à l'aide de l'attribut `'event`**

☝ **Seul le signal `clk` doit figurer dans ce `if`**

```
process (clk)
begin …

if (clk'event and clk = '1') then …   -- front montant
if (clk'event and clk = '0') then …   -- front descendant
```

❒ **à l'aide d'une fonction**

```
if rising_edge(clk)  then …           -- front montant
if falling_edge(clk) then …           -- front descendant
```


© D.Giacona

2.5.3. reset et preset asynchrones

Les signaux de mise à zéro ou de mise à un sont prioritaires vis-à-vis de l'horloge.

Priorité de `ar` par rapport à `clk`

```
process (ar,clk)        -- ar doit figurer dans la liste
begin …
if ar = '1' then        -- reset asynchrone
  q <= (others => '0');
elsif (clk'event and clk= '1') then
  …
```

```
process (ap,clk)  -- ap doit figurer dans la liste
begin …
if ap = '1' then  -- preset asynchrone
  q <= (others => '1');
elsif (clk'event and clk= '1') then
```


© D.Giacona

2.5.4. reset et preset synchrones

La mise à zéro ou la mise à un ne prend effet que sur un front actif de l'horloge.

```
process (clk)  -- sr ne doit pas figurer dans la liste de sensibilité
begin …
if (clk'event and clk= '1') then
  if sr = '1'then  -- reset synchrone
    q <= '0';
  else
    …
  end if;
end if;
```

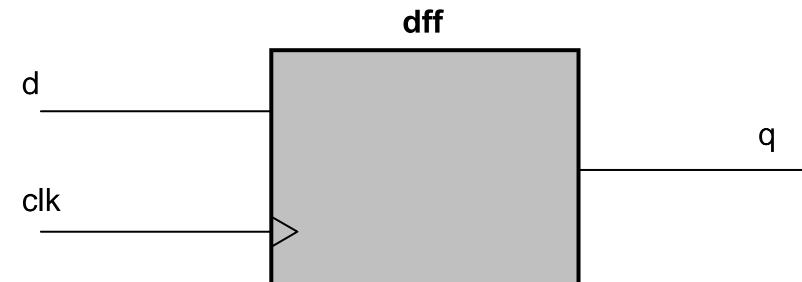
☝ Le test du signal `sr` est réalisé obligatoirement dans un `if` imbriqué

```
process (clk)  -- sp ne doit pas figurer dans la liste de sensibilité
begin …
if (clk'event and clk= '1') then
  if sp = '1'then  -- preset synchrone
    q <= '1';
  else
    …
  end if;
end if;
```


© D.Giacona

2.6. Exemples de blocs séquentiels

2.6.1. D flip-flop (rising edge)

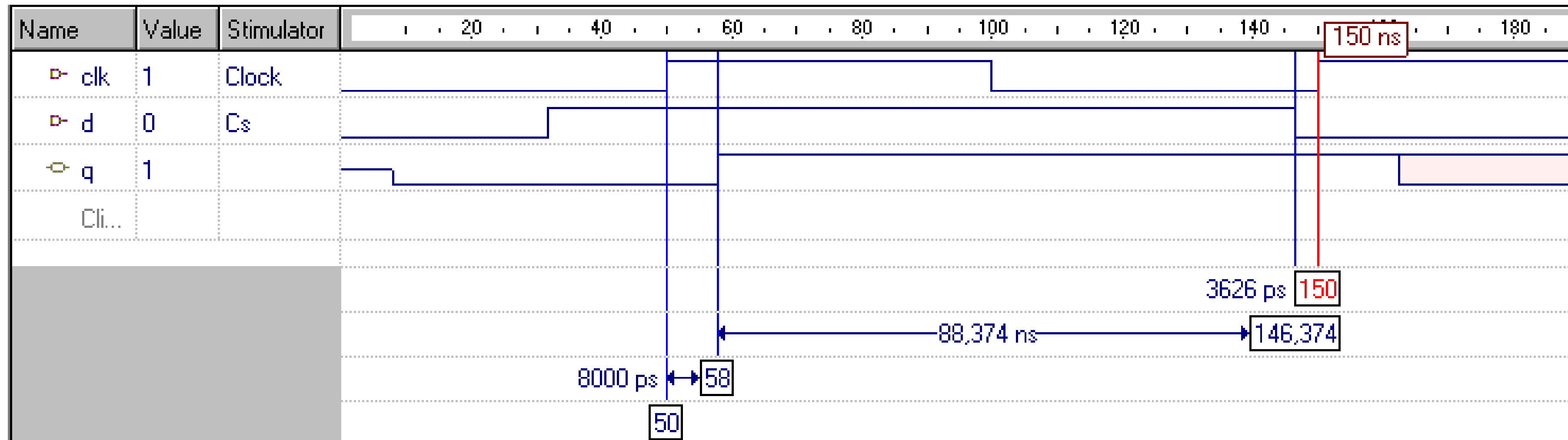



```
library ieee;
use ieee.std_logic_1164.all;

entity dff is port(
 d,clk  : in std_logic;
 q      : out std_logic);
end dff;

architecture arch_dff of dff is

begin

ffd: process (clk)
begin
  if (clk'event and clk = '1') then        -- si front montant de clk
    q <= d;
  end if;
end process ffd;

end arch_dff;
```


© D.Giacona

Simulation temporelle

```
Target Device:   CY37256P160-83AC
DESIGN EQUATIONS
    q.D = d
    q.C = clk

----------------------------------------------------------------------------
Signal Name | Delay Type  |    tmax    | Path Description
----------------------------------------------------------------------------
reg::q[143]
inp::d              tS                8.0 ns      1 pass
out::q              tCO               8.0 ns
```



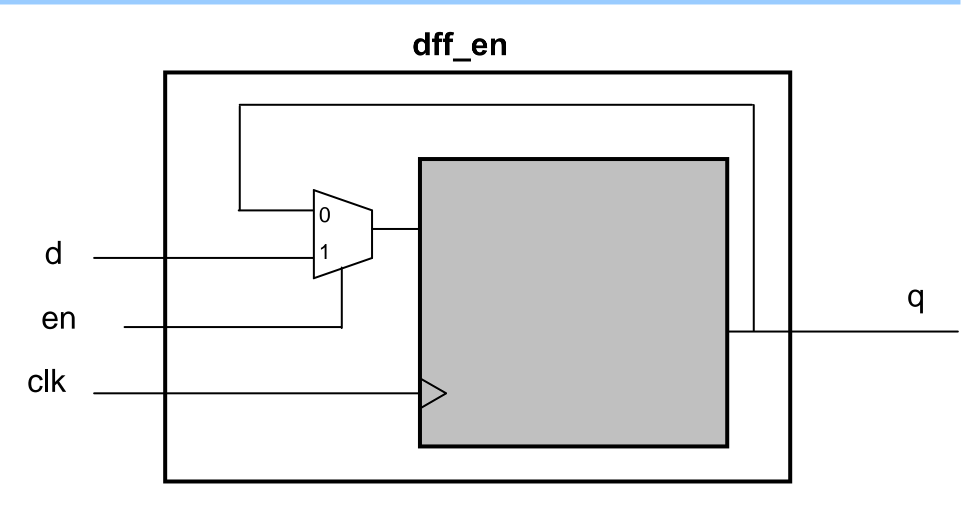
Le simulateur :

- **indique le temps de retard du signal de sortie `q` par rapport au front actif de l'horloge : `tco` = 8ns**
- **montre que le non respect du paramètre `ts` = 8ns (setup time) entraîne un état métastable de la sortie**

© *D.Giacona*

2.6.2. D flip-flop with enable

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dff_en is
    Port ( clk : in  STD_LOGIC;
           en : in  STD_LOGIC;
           d : in  STD_LOGIC;
           q : out  STD_LOGIC);
end dff_en;

architecture Behavioral of dff_en is

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

end Behavioral;
```

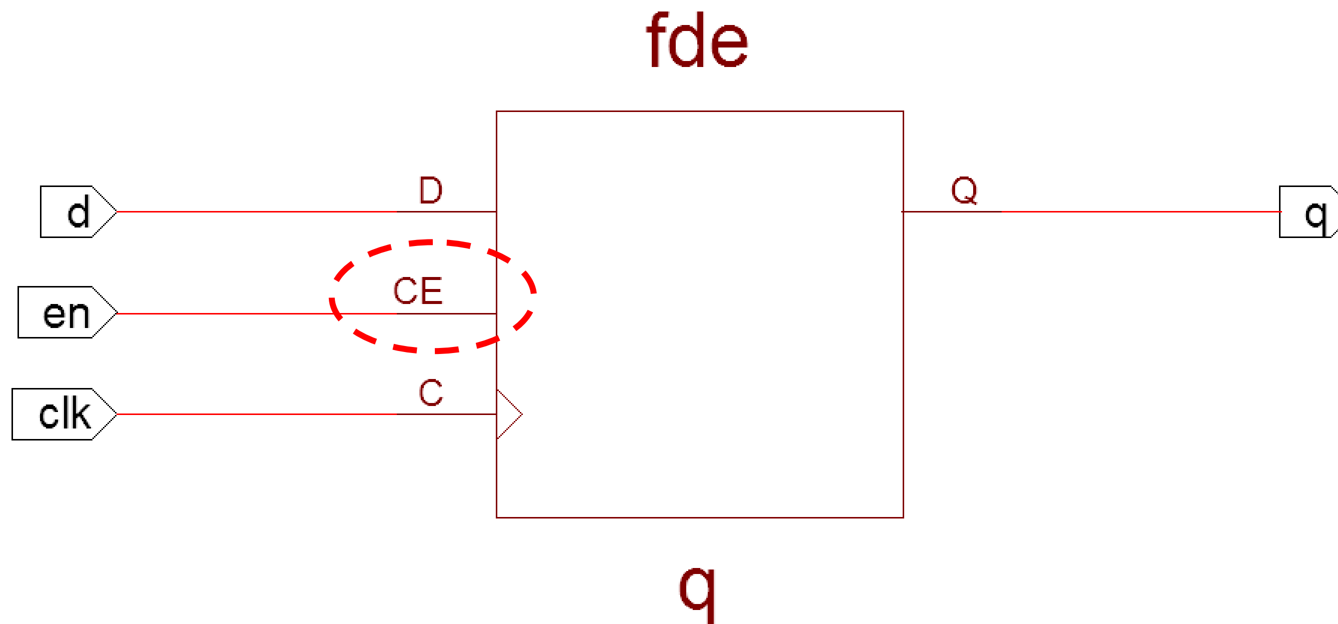


Priorité de `clk` par rapport à `en`

© D.Giacona

Note

Dans les circuits de la famille SPARTAN (Xilinx), les flip-flops sont tous munis d'une entrée **CE** pour la validation de la donnée.



© D.Giacona

☹ Mauvais code

La description suivante est à proscrire !

```
if clk'event and clk = '1' and en = '1' then
    q <= d;
end if;
```

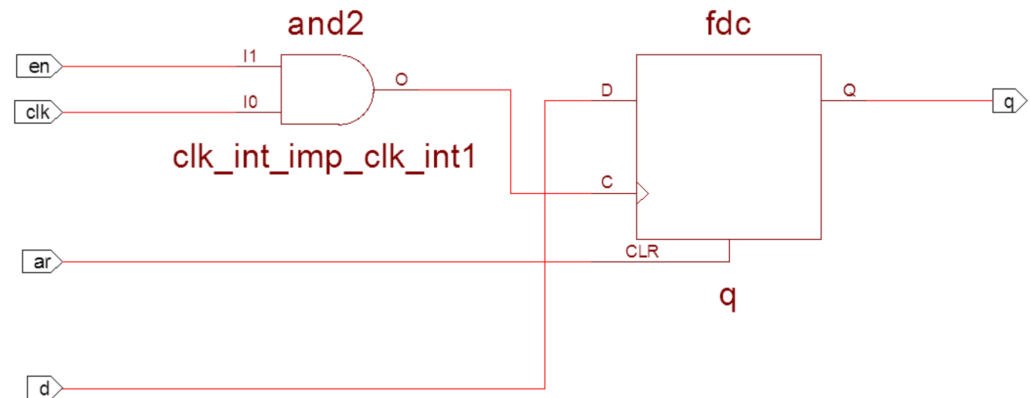
- La priorité des signaux n'est pas explicite !
- Certains compilateurs, comme par exemple Warp, génèrent même une erreur !

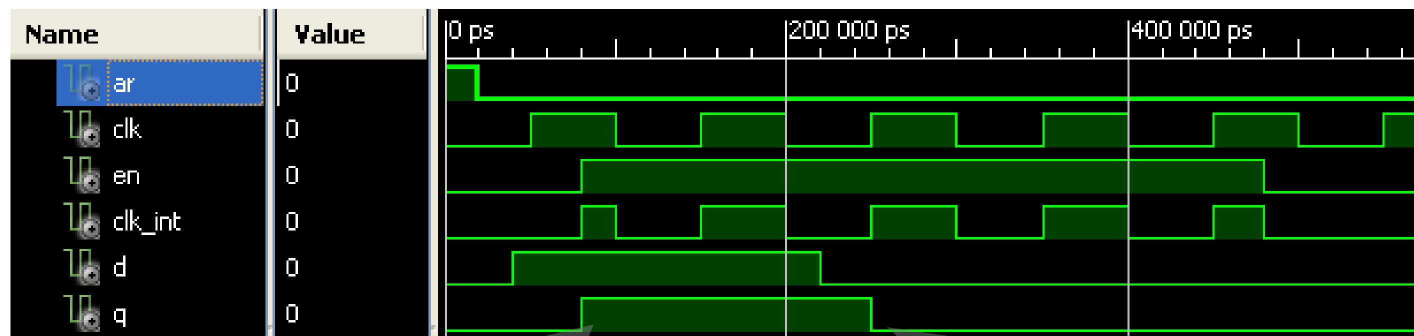
© D.Giacona

☹ Mauvaise conception : (*avoid gated clock*)

- La description suivante est contraire à la règle : "dans un système séquentiel synchrone, les flip-flops sont toujours commandés sur le même front du signal d'horloge `clk`"

```
architecture Behavioral of dff_en is
  signal clk_int : STD_LOGIC;
begin
clk_int <= clk and en;
dflip_flop: process (clk_int,ar)
  begin
    if ar = '1' then
      q <= '0';
    elsif clk_int'event and clk_int = '1' then
      q <= d;
    end if;
end process;
end Behavioral;
```



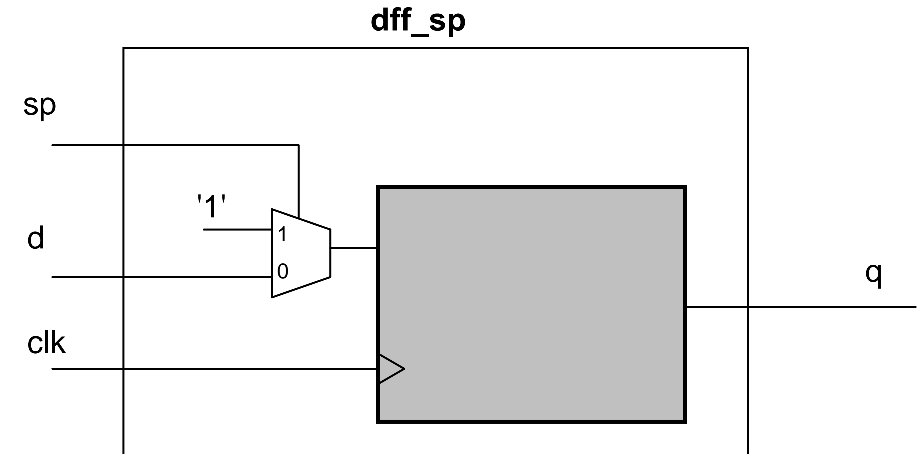


Changement d'état de `q` en dehors de ↑`clk` : NOK

Changement d'état de `q` sur ↑`clk` : OK

© D.Giacona

2.6.3. D flip-flop with Active High Synchronous Preset



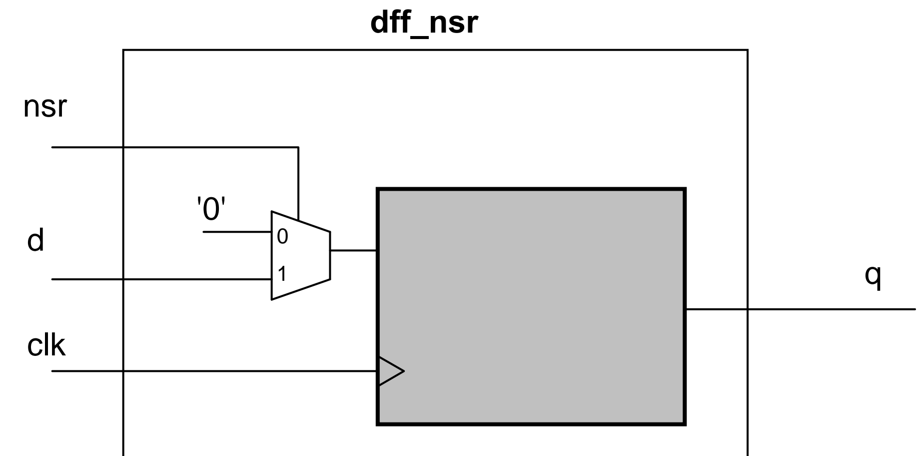

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity dff_sp is port(
 d, clk, sp : in std_logic;
 q          : out std_logic);
end dff_sp;

architecture arch_dff_sp of dff_sp is
begin

dff: process (clk)
begin
  if (clk'event and clk = '1') then
    if sp = '1' then       -- preset actif à l'état haut
      q <= '1';
    else
      q <= d;
    end if;
  end if;
end process dff;
end arch_dff_sp;
```

```
DESIGN EQUATIONS
    q.D = sp + d
    q.C = clk
```


© D.Giacona

2.6.4. D flip-flop with Active Low Synchronous Reset

```
library ieee;
use ieee.std_logic_1164.all;

entity dff_nsr is port(
 d, clk, nsr : in std_logic;
 q           : out std_logic);
end dff_nsr;

architecture arch_dff_nsr of dff_nsr is

begin
dff: process (clk)
begin
  if (clk'event and clk = '1') then
    if nsr = '0' then      -- reset actif à l'état bas
      q <= '0';
    else
      q <= d;
    end if;
  end if;
end process dff;

end arch_dff_nsr;
```




```
DESIGN EQUATIONS

    q.D = d * nsr

    q.C = clk
```


© D.Giacona

2.6.5. D latch (level sensitive)

```
-- description par un processus, mémorisation implicite

library ieee;
use ieee.std_logic_1164.all;

entity d_latch is port(
  le    : in std_logic;          -- le : latch enable
  d     : in std_logic;          -- d : data
  q     : out std_logic);
end d_latch;


architecture arch_d_latch of d_latch is
begin
latchd:  process (le,d)
begin
  if (le = '1') then
    q <= d;
  end if;
end process latchd;

end arch_d_latch;
```

le	d	état présent q	état futur q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

```
DESIGN EQUATIONS
  q.D = d

  q.LH = le
```

UHA UNIVERSITE DE HAUTE-ALSACE Mulhouse-Colmar ensisa école nationale supérieure d'ingénieurs sud alsace

© D.Giacona

2.6.6. D latch (level sensitive)

```
library ieee;
use ieee.std_logic_1164.all;

entity d_latch is
  port (d     : in std_logic;
        le    : in std_logic;
        q     : out std_logic);
end d_latch;


ARCHITECTURE arch_d_latch OF d_latch IS
  SIGNAL qx : std_logic;

BEGIN
  qx <= (d and le) or (d and qx) or (not(le) and qx);
  q  <= qx;
END arch_d_latch;
```

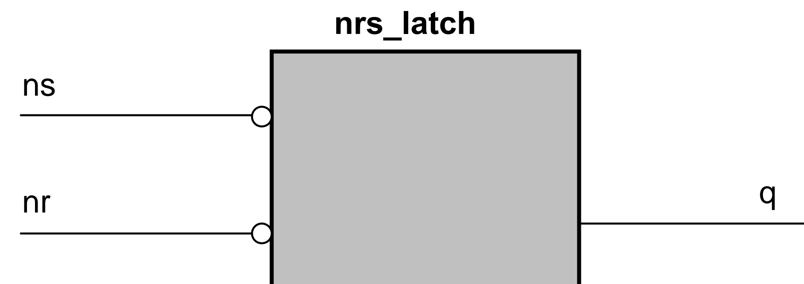
UHA ensisa école nationale supérieure d'ingénieurs sud alsace

© D.Giacona

2.6.7. RS latch (entrées actives à l'état bas)

ns : negative set
nr : negative reset

ns	nr	q+	
0	0	(1)	(priorité à la mise à un)
0	1	1	mise à un
1	0	0	mise à zéro
1	1	q	maintien




```
library ieee;
use ieee.std_logic_1164.all;

entity nrs_latch is port(
  ns, nr  : in std_logic;
  q       : out std_logic);
end nrs_latch;

architecture arch_nrs_latch of nrs_latch is
signal sel : std_logic_vector(1 downto 0);
begin
  sel <= (ns,nr);
  process (sel)
    begin
      case sel is
        when "00" | "01" => q <= '1';
        when "10"        => q <= '0';
        when others      => null;
      end case;
  end process;
end arch_nrs_latch;
```

```
DESIGN EQUATIONS

    q = nr * q.CMB + /ns
```

UHA ensisa école nationale supérieure d'ingénieurs sud alsace

© D.Giacona

2.6.8. RS latch (entrées actives à l'état bas)

```
library ieee;
use ieee.std_logic_1164.all;

entity nrs_latch is port(
  ns,nr     : in std_logic;
  q         : out std_logic);
end nrs_latch;

architecture arch_nrs_latch of nrs_latch is
begin

  process (ns,nr)
    begin
      if (ns,nr) = ('0','0') then
        q <= '1';
      elsif (ns,nr) = ('0','1') then
        q <= '1';
      elsif (ns,nr) = ('1','0') then
        q <= '0';
      end if;
  end process;
end arch_nrs_latch;
```

```
DESIGN EQUATIONS

    q = nr * q.CMB + /ns
```


© D.Giacona

2.6.9. RS latch (entrées actives à l'état bas)

```
library ieee;
use ieee.std_logic_1164.all;

entity nrs_latch is port(
  ns, nr  : in std_logic;
  q       : out std_logic);
end nrs_latch;


architecture arch_nrs_latch of nrs_latch is
  signal sq, snq : std_logic;
begin

  snq <= nr nand sq;
  sq <= ns nand snq;
  q <= sq;

end arch_nrs_latch;
```


© D.Giacona

2.6.10. Registre D synchrone, 8 bits, avec reset asynchrone

```
library ieee;
use ieee.std_logic_1164.all;

entity dreg8_ar is port(
  clk, ar    : in std_logic;
  d          : in std_logic_vector(7 downto 0);
  q          : buffer std_logic_vector(7 downto 0));
end dreg8_ar;

use work.std_arith.all;


architecture arch_dreg8_ar of dreg8_ar is
begin

process (clk, ar)
begin
  if ar = '1' then
    q <= (others => '0');
  elsif (clk'event and clk= '1') then
    q <= d;
  end if;
end process;

end arch_dreg8_ar;
```

```
DESIGN EQUATIONS

    q(0).D = d(0)

    q(0).AP = GND

    q(0).AR = ar

    q(0).C = clk

etc…
```


© D.Giacona

2.6.11. Registre D synchrone, 8 bits, avec reset synchrone

```
library ieee;
use ieee.std_logic_1164.all;

entity dreg8_sr is port(
  clk, sr : in std_logic;
  d       : in std_logic_vector(7 downto 0);
  q       : out std_logic_vector(7 downto 0));
end dreg8_sr;

use work.std_arith.all;


architecture arch_dreg8_sr of dreg8_sr is
begin

process (clk)
begin
  if (clk'event and clk= '1') then
     if sr = '1'then
       q <= "00000000";
     else
       q <= d;
     end if;
  end if;
end process;

end arch_dreg8_sr;
```

Rappel : le test du signal `sr` est réalisé obligatoirement dans un `if` imbriqué

```
DESIGN EQUATIONS

    q(0).D = d(0) * /sr

    q(0).C = clk

    q(1).D = d(1) * /sr

    q(1).C = clk …
```

Le test sur `sr` est effectué si le test sur `clk` est positif, ce qui détermine un reset synchrone. De plus, le reset est prioritaire sur la mémorisation.

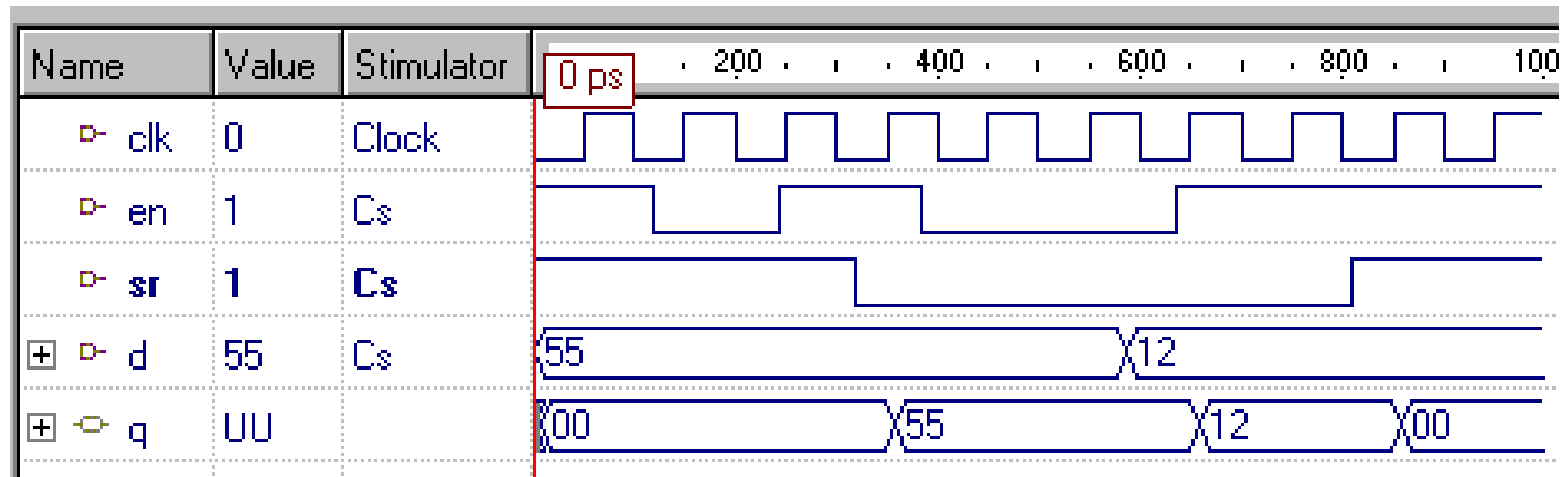
© D.Giacona

2.6.12. Registre D synchrone, 8 bits, avec reset synchrone et entrée de validation

```
library ieee;
use ieee.std_logic_1164.all;

entity dreg8_sr_en is port(
  clk, sr, en : in std_logic;
  d           : in std_logic_vector(7 downto 0);
  q           : buffer std_logic_vector(7 downto 0));
end dreg8_sr_en;

use work.std_arith.all;


architecture arch_dreg8_sr_en of dreg8_sr_en is
begin

process (clk)
begin
  if (clk'event and clk = '1') then
    if sr = '1'then
      q <= "00000000";
    elsif en = '1' then
      q <= d;
    else
      q <= q;
    end if;
  end if;
end process;

end arch_dreg8_sr_en;
```

```
DESIGN EQUATIONS

q(0).D = d(0) * en * /sr + /en * q(0).Q * /sr
q(0).C = clk
q(1).D = d(1) * en * /sr + /en * q(1).Q * /sr
q(1).C = clk…
```


© D.Giacona



© D.Giacona

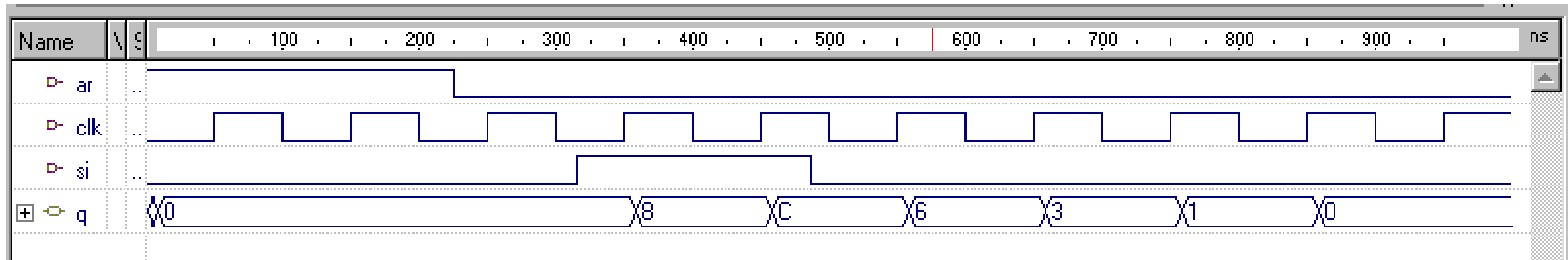
2.6.13. Registre à décalage à droite 4 bits, avec reset asynchrone

```
library ieee;
use ieee.std_logic_1164.all;

library ieee;
use ieee.std_logic_1164.all;

entity shift_right_reg4_ar is port(
  clk, ar, si   : in std_logic;
  q             : buffer std_logic_vector(3 downto 0));
end shift_right_reg4_ar;

use work.std_arith.all;

architecture arch_shift_right_reg4_ar of shift_right_reg4_ar is
begin

process (clk, ar)
begin
  if ar = '1' then
    q <= "0000";
  elsif (clk'event and clk= '1') then
    q(3) <= si;
    q(2) <= q(3);
    q(1) <= q(2);
    q(0) <= q(1);
  end if;
end process;

end arch_shift_right_reg4_ar;
```

```
DESIGN EQUATIONS

    q(0).D = q(1).Q
    q(0).AP = GND
    q(0).AR = ar
    q(0).C = clk

    q(1).D = q(2).Q
    q(1).AP = GND
    q(1).AR = ar
    q(1).C = clk

    q(2).D = q(3).Q
    q(2).AP = GND
    q(2).AR = ar
    q(2).C = clk

    q(3).D = si
    q(3).AP = GND
    q(3).AR = ar
    q(3).C = clk
```


© D.Giacona

q(3)	q(2)	q(1)	q(0)	**valeurs à l'instant d'activation du processus**
si	q(3)	q(2)	q(1)	**valeurs prises à la fin du processus**



© D.Giacona

2.6.14. Registre à décalage à droite 12 bits, avec reset asynchrone

```
library ieee;
use ieee.std_logic_1164.all;

entity shift_right_reg12_ar is port(
  clk, ar, si : in std_logic;
  q           : buffer std_logic_vector(11 downto 0));
end shift_right_reg12_ar;

architecture arch_shift_right_reg12_ar of shift_right_reg12_ar is
begin

process (clk, ar)
begin
  if ar = '1' then
    q <= (others => '0');
  elsif (clk'event and clk = '1') then
    q(11) <= si;       -- si (serial input)
    shift : for i in 0 to 10 loop
      q(i) <= q(i+1);
    end loop shift;
  end if;
end process;
end arch_shift_right_reg12_ar;
```


© D.Giacona

2.6.15. Compteur BCD 4 bits, reset asynchrone, validation de comptage, sortie retenue à assignation combinatoire

❒ **Utilisation d'un signal interne de type `std_logic_vector`**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;


entity bcdcnt4_ar_en_comb is port(
  clk, ar, en : in std_logic;
  co          : out std_logic;
  q           : out std_logic_vector(3 downto 0));
end bcdcnt4_ar_en_comb;


architecture arch_bcdcnt4_ar_en_comb of bcdcnt4_ar_en_comb is

  signal count : std_logic_vector(3 downto 0);

begin
```


© D.Giacona

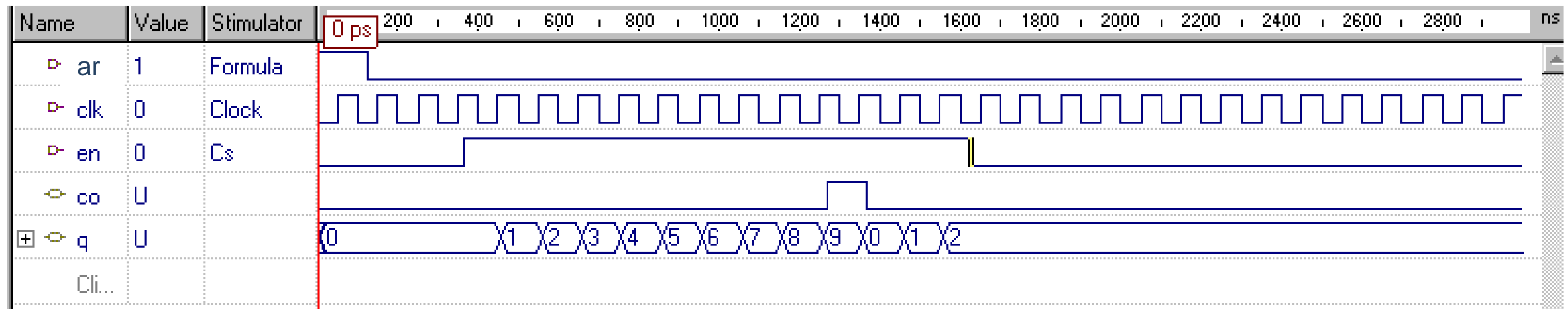
```
process (clk, ar)
begin
  if ar = '1' then
    count <= (others => '0');
  elsif (clk'event and clk = '1') then
    if en = '1' then
      if count < x"9" then    -- les littéraux sont de
        count <= count + 1;   --  type std_logic
      else
        count <= x"0";
      end if;
    end if;
  end if;

end process;

-- Elaboration du signal carry "co" hors processus
--   (assignation combinatoire)
co     <= '1' when (count = x"9") else '0';
q      <= count;

end arch_bcdcnt4_ar_en_comb;
```

```
DESIGN EQUATIONS
co = q(3).Q * /q(2).Q * /q(1).Q * q(0).Q
```

La retenue co **est élaborée par un bloc combinatoire. Elle est mise à jour après la mise à jour de la sortie** q**.**

© D.Giacona

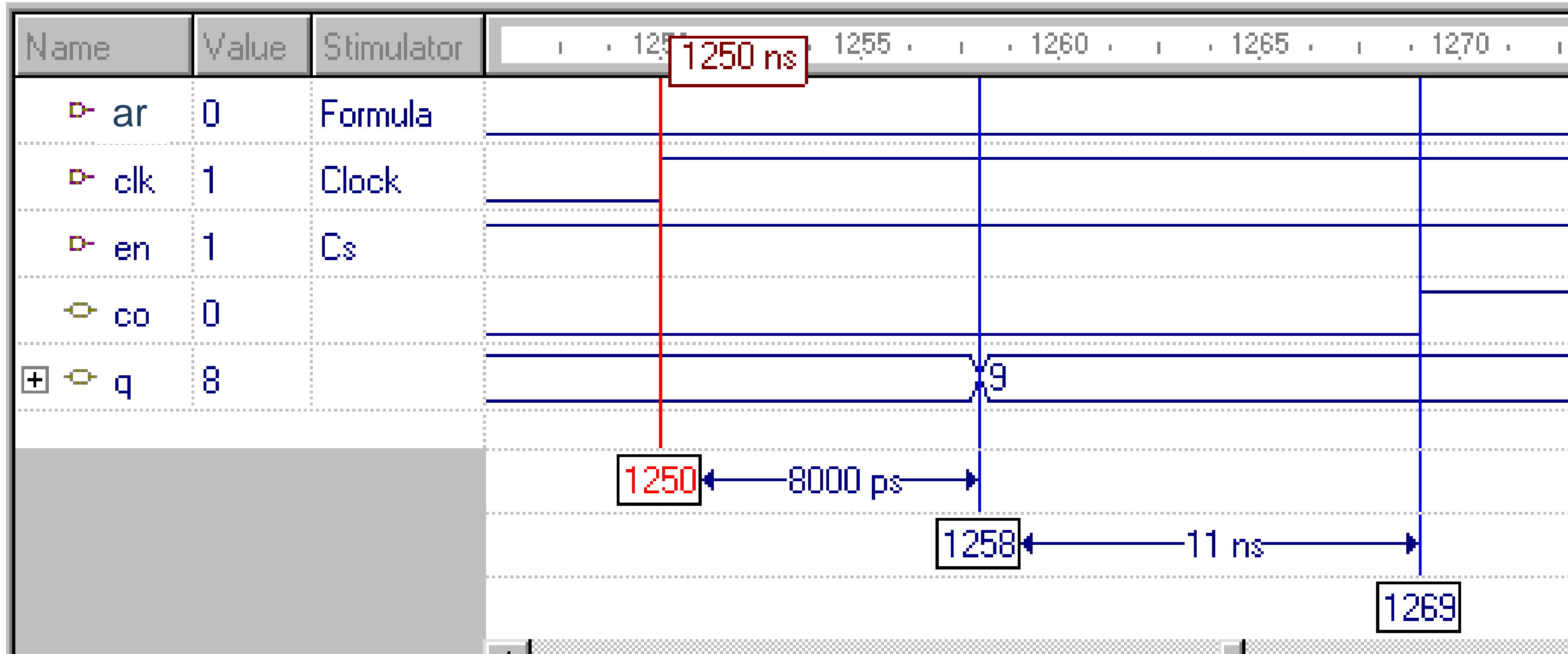
☞ Résultat de simulation




```
TIMING PATH ANALYSIS        using Package: cy37256p160-83ac

Worst Case Path Summary
-----------------------

                      tS = 8.0 ns for q(0).D
                    tSCS = 12.0 ns for q(0).D
                     tCO = 19.0 ns for co
                     tRO = 21.5 ns for q(0).AR
```

f_{max} = 1 / tSCS

© D.Giacona



© D.Giacona

2.6.16. Compteur BCD 4 bits, reset asynchrone, validation de comptage, sortie retenue à assignation combinatoire

❒ Utilisation d'un signal interne de type `integer`

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity bcdcnt4_ar_en_comb is port(
  clk, ar, en   : in std_logic;
  co            : out std_logic;
  q             : out std_logic_vector(3 downto 0));
end bcdcnt4_ar_en_comb;

architecture arch_bcdcnt4_ar_en_comb of bcdcnt4_ar_en_comb is

    signal count : integer range 0 to 9;  -- limite l'ensemble des valeurs
```

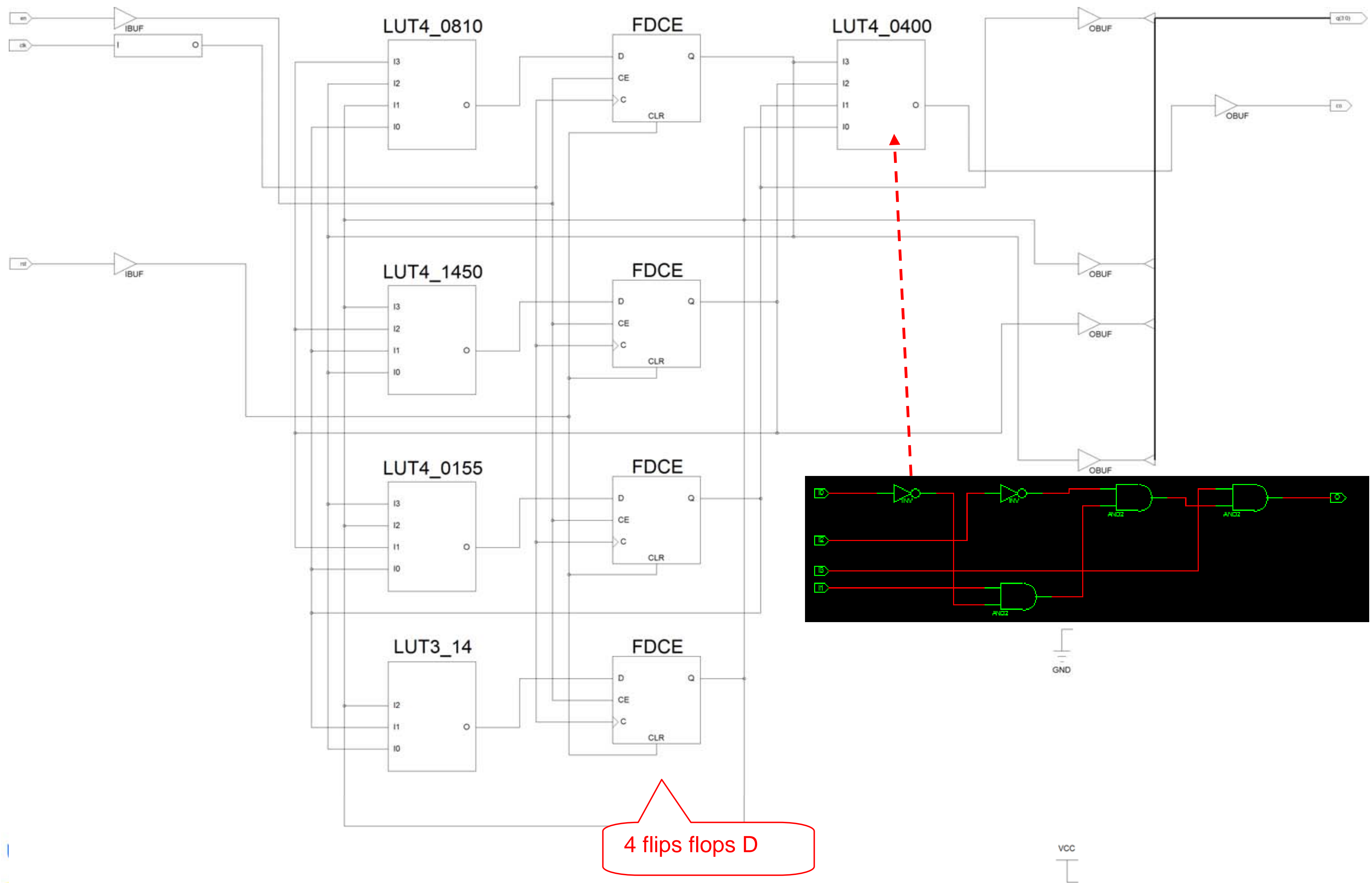
En l'absence de contrainte (`range … to …`), un outil de synthèse convertirait l'entier `count` en un vecteur de 32 bits.
➔ c.-à-d. produirait 32 flips-flops !

© D.Giacona

```
begin

process (clk, ar)
begin
  if ar = '1' then
    count <= 0;
  elsif (clk'event and clk = '1') then
    if en = '1' then
      if count < 9 then
        count <= count + 1;
      else
        count <= 0;
      end if;
    end if;
  end if;
end process;

co <= '1' when count = 9 else '0';    -- assignation combinatoire

q <= conv_std_logic_vector(count,4);  -- conversion en vecteur 4 bits

end arch_bcdcnt4_ar_en_comb;
```

UHA UNIVERSITE DE HAUTE-ALSACE Mulhouse-Colmar ensisa école nationale supérieure d'ingénieurs sud alsace

© D.Giacona


LUT4_0810
FDCE
LUT4_0400
IBUF
OBUF
LUT4_1450
FDCE
LUT4_0155
FDCE
LUT3_14
FDCE
D
CE
C
CLR
Q
I3
I2
I1
I0
O
GND
VCC
4 flips flops D

2.6.17. Compteur BCD 4 bits, reset asynchrone, validation de comptage, sortie retenue synchronisée

❒ Utilisation d'un signal interne de type `std_logic_vector`

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity bcdcnt4_ar_en_sync is port(
  clk, ar, en    : in std_logic;
  co             : out std_logic;
  q              : out std_logic_vector(3 downto 0) );
end bcdcnt4_ar_en_sync;


architecture arch_bcdcnt4_ar_en_sync of bcdcnt4_ar_en_sync is

  signal count : std_logic_vector(3 downto 0);

begin
```


© D.Giacona

```
process (clk, ar)

begin
  if ar = '1' then
    co <= '0';
    count <= (others => '0');
  elsif (clk'event and clk = '1') then
    if en = '1' then
      if count < x"9" then    -- les littéraux sont de type std_logic
        count <= count + 1;
        if count = x"8" then  -- anticipation du calcul de la retenue
          co <= '1';
        end if;
      else
        count <= x"0";
        co <= '0';
      end if;
    end if;
  end if;
end process;

q <= count;

end arch_bcdcnt4_ar_en_sync;
```

Si `count` = 8, alors au prochain front montant d'horloge

- **`count` passera à 9**
- **`co` passera à '1'**

© D.Giacona

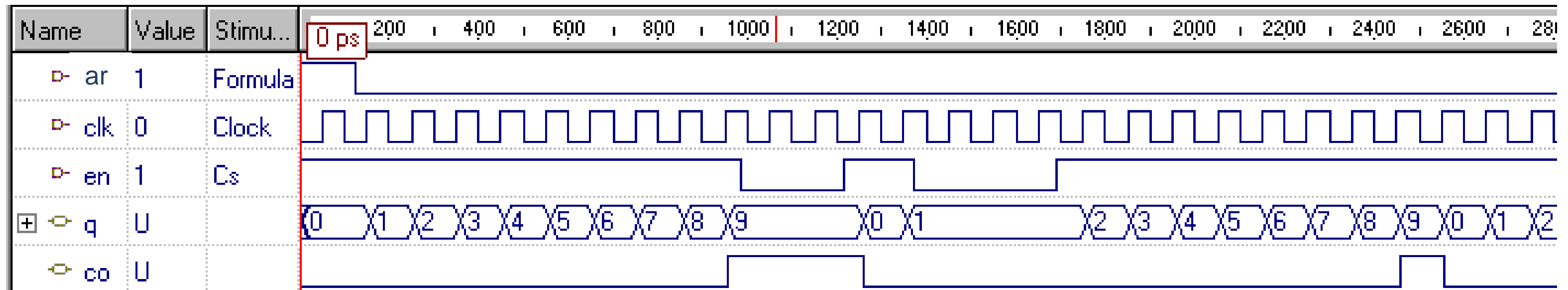
❒ Rapport de compilation

La sortie co est issue d'un flip-flop D

```
Target Device: CY37256P160-83AC

DESIGN EQUATIONS

    co.D = en * /q(0).Q * /q(1).Q * /q(2).Q * q(3).Q
         + co.Q * /en
         + co.Q * /q(3).Q
    co.AP = GND
    co.AR = ar
    co.C = clk

    q(0).D =  en * /q(0).Q * /q(1).Q * /q(2).Q
           + en * /q(0).Q * /q(3).Q
           + /en * q(0).Q
    q(0).AP = GND
    q(0).AR = ar
    q(0).C = clk

    q(1).T =  en * q(1).Q * q(3).Q
           + en * q(0).Q * /q(3).Q
    q(1).AP = GND
    q(1).AR = ar
    q(1).C = clk

    q(2).T = en * q(0).Q * q(1).Q * /q(3).Q
           + en * q(2).Q * q(3).Q
    q(2).AP = GND
    q(2).AR = ar
    q(2).C = clk
```

© D.Giacona

❐ Résultat de simulation

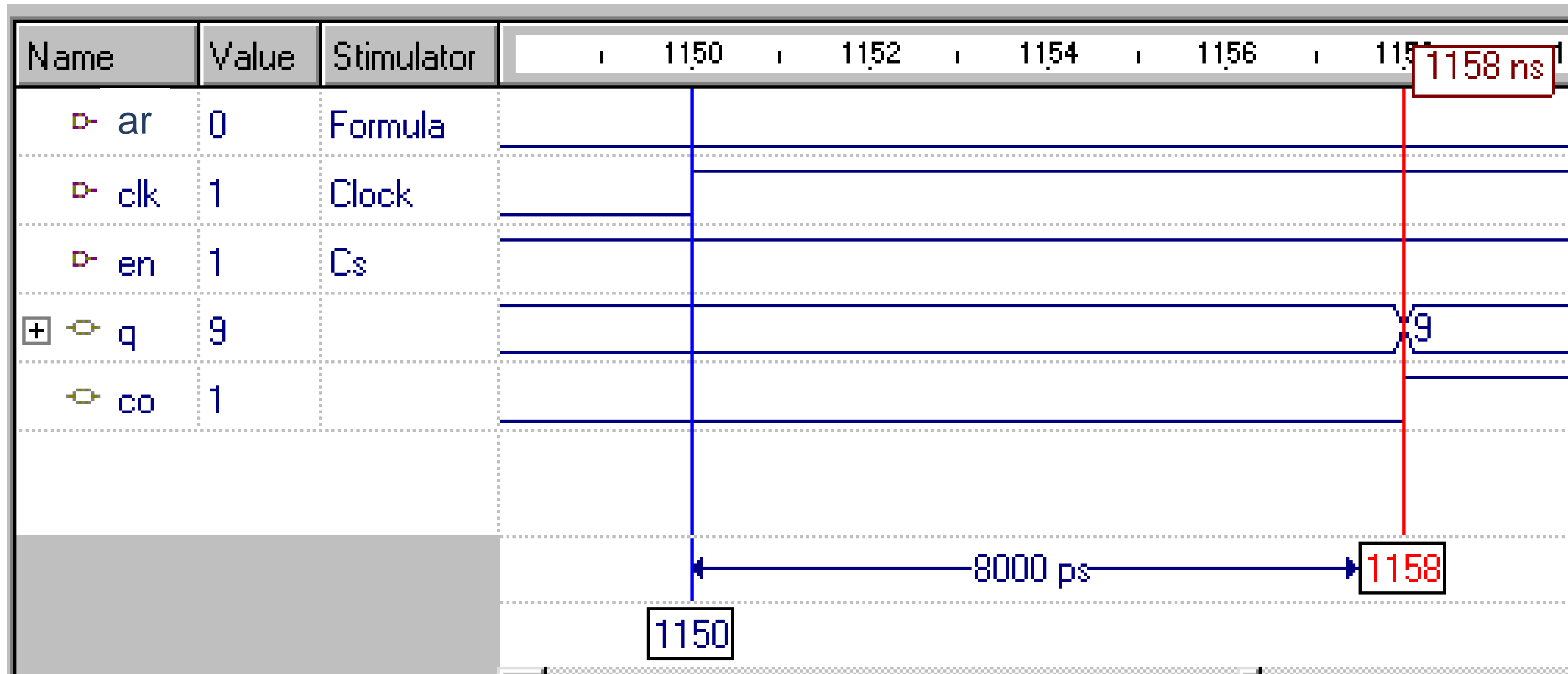



```
TIMING PATH ANALYSIS using Package: cy37256p160-83ac

rst Case Path Summary
-----------------------
                     tS = 8.0 ns for q(0).D
                   tSCS = 12.0 ns for q(0).D
                    tCO = 8.0 ns for q(0).C
                    tRO = 21.5 ns for q(0).AR
```

f_{max} = 1 / tSCS

La sortie `q` **et la retenue** `co` **changent d'état en même temps.**

© *D.Giacona*



© D.Giacona

2.6.18. Compteur binaire naturel 4 bits, synchrone, chargeable, sorties 3 états

oe = '1' : le port d'entrée/sortie fournit la valeur de comptage
oe = '0' : le port d'entrée/sortie reçoit une donnée de chargement

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity cnt4_ld_oe is port(
      clk, oe, ld      : in std_logic;
      count_io         : inout std_logic_vector(3 downto 0));
end cnt4_ld_oe;

architecture arch_cnt4_ld_oe of cnt4_ld_oe is
signal count, data : std_logic_vector(3 downto 0);

begin
process (clk)
begin
  if (clk'event and clk = '1') then
    if ld = '1' then
      count <= data;
    else
      count <= count + 1;
    end if;
  end if;
end process;

-- Validation de la sortie
count_io <= count when (oe = '1') else "ZZZZ";
data <= count_io;
end arch_cnt4_ld_oe;
```

Remarque : Le signal `data` peut être remplacé par `count_io`.
Il n'est utilisé que pour clarifier la description.

```
DESIGN EQUATIONS

count_io(0).D = count_io(0) * ld
              + /count_io(0).Q * /ld

count_io(0).C = clk

count_io(1).D =  /count_io(0).Q *
count_io(1).Q * /ld
              + count_io(0).Q *
/count_io(1).Q * /ld
              + count_io(1) * ld

count_io(1).C = clk

count_io(1).OE = oe

count_io(2).T = /count_io(2) * count_io(2).Q
* ld
              + count_io(2) * /count_io(2).Q
* ld
              + count_io(0).Q *
count_io(1).Q * /ld

count_io(2).C = clk
```

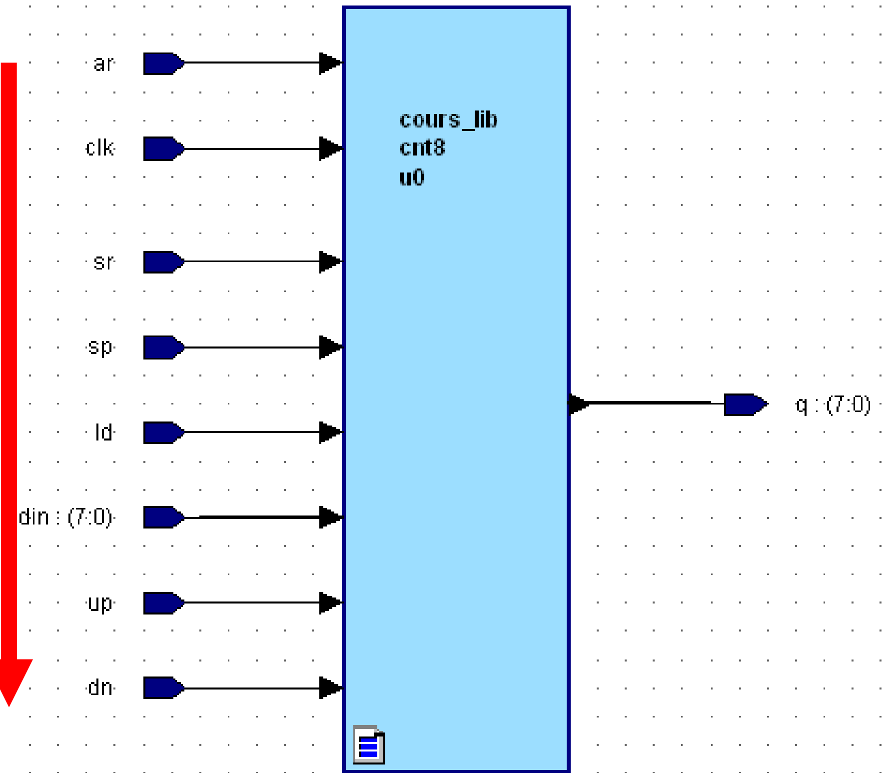
UHA UNIVERSITÉ DE HAUTE-ALSACE Mulhouse-Colmar ensisa école nationale supérieure d'ingénieurs sud alsace

© D.Giacona

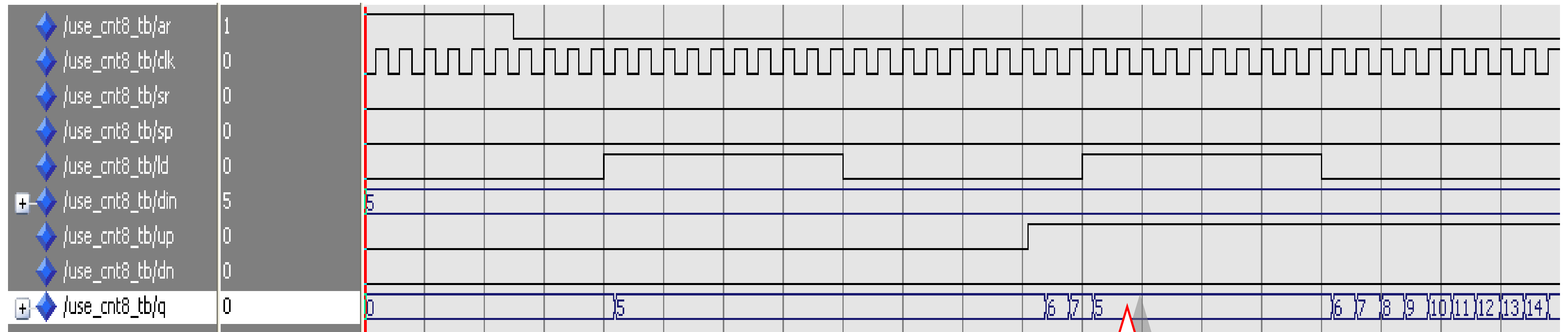
2.6.19. Compteur 8 bits à commandes multiples

```
ARCHITECTURE behavioral OF cnt8 IS
  signal cnt : std_logic_vector(7 downto 0);
BEGIN

  count_proc: process(ar,clk)
  begin
    if ar = '1' then
      cnt <= (others => '0');
    elsif clk'event and clk =
      if sr = '1' then
        cnt <= (others => '0');
      elsif sp = '1' then
        cnt <= (others => '1');
      elsif ld = '1' then
        cnt <= din;
      elsif up = '1' then
        cnt <= cnt + 1;
      elsif dn = '1' then
        cnt <= cnt - 1;
      end if;
    end if;
  end process;

  q <= cnt;

END ARCHITECTURE behavioral;
```

Priorités décroissantes
⇨ Utiliser la structure
if … then … elsif …



© D.Giacona



Priorité de la commande `ld` par rapport à la commande `up`

© D.Giacona