Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Système numérique

Résumé Chapitre 3 Fonctions combinatoires bascules

Auteurs :
Marc Roten

Professeur :
Nicolas Schroeter

20 novembre 2018

Table des matières

1 Fonction combinatoire

Il existe différents types de fonctions combinatoires :

- Equation logique
- table de vérité
- multiplexage
- comparaison
- encodage de priorité
- addition/soustraction

2 Fonctions d'affectation

```
A <= '1' when B='1' else
     '0' when C='1' else '0';
-- ATTENTION : l'affectation conditionnelle avec when else
-- ne peut etre utilisee qu'a l'exterieur d'une declaration
-- de process (instruction strictement concurrente)
```

FIGURE 1 – différentes fonctions d'affectation

```
with signal_sel select
    Sortie <= Entr_A when "00" | "01",
              '1' when "11",
              '0' when others; -- pour couvrir toutes les autres
                               -- combinaisons "UZ",…
```

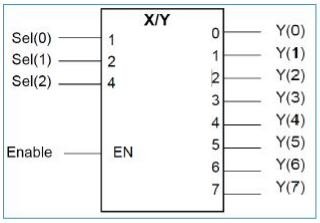
FIGURE 2 – différentes fonctions d'affectation

2.1 Exemple

```
with hexa select
  ascii <= "0110000" when "0000",
           "0110001" when "0001",
           "0110010" when "0010",
           "0110011" when "0011",
           "1000101" when "1110",
           "1000110" when others;
```

FIGURE 3 – Petit Exemple

3 Décodeur 3 a 8



EN	Sel(2)	Sel(1)	Sel(0)	Y(7)	Y(6)	Y(5)	Y(4)	Y(3)	Y(2)	Y(1)	Y(0)
'0'	x	x	x	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'
'1'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'0'	'1'
'1'	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'0'	'0'	'1'	'0'
'1'	'0'	'1'	'0'	'0'	'0'	'0'	'0'	'0'	'1'	'0'	'0'
'1'	'0'	'1'	'1'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'0'
'1'	'1'	'0'	'0'	'0'	'0'	'0'	'1'	'0'	'0'	'0'	'0'
'1'	'1'	'0'	'1'	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'0'
'1'	'1'	'1'	'0'	'0'	'1'	'0'	'0'	'0'	'0'	'0'	'0'
'1'	'1'	'1'	'1'	'1'	'0'	'0'	'0'	'0'	'0'	'0'	'0'

FIGURE 4 – Décodeur 3 vers 8

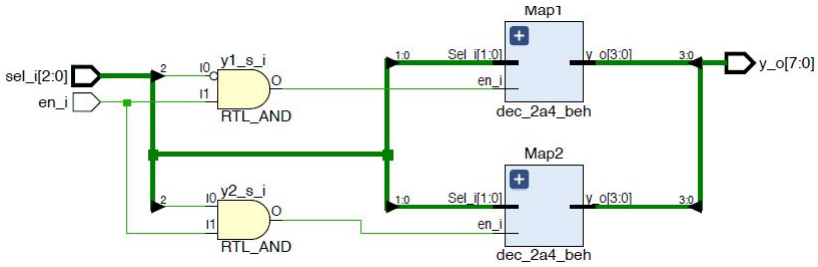


FIGURE 5 – Schéma rtl décodeur 3 à 8

```
library ieee;
use ieee.std_logic_1164.all;
entity dec_3a8 is
  port(
    sel_i: in std_logic_vector(2 downto 0);
    en_i : in std_logic;
    y_o : out std_logic_vector(7 downto 0)
  );
end entity;
architecture behav of dec_3a8 is
  component dec_2a4_beh is
    port(
        Sel_i : in std_logic_vector(1 downto 0);
        en_i: in std_logic;
        y_o : out std_logic_vector(3 downto 0)
    );
  end component;
  signal y_s : std_logic_vector(7 downto 0);
  signal y1_s, y2_s : std_logic;
begin
  Map1:dec_2a4_beh
  port map(
    sel_i => sel_i(1 downto 0),
    en_i => y1_s,
    y_o => y_s(3 downto 0)
  );
  Map2:dec_2a4_beh
  port map(
    sel_i => sel_i(1 downto 0),
    en_i => y2_s,
    y_o => y_s(7 downto 4)
  );

  y1_s <= not(sel_i(2)) and en_i;
  y2_s <= sel_i(2) and en_i;

  y_o <= y_s ;

end architecture;
```

FIGURE 6 – Code décodeur 3 à 8

4 Multiplexeur 4 à 1

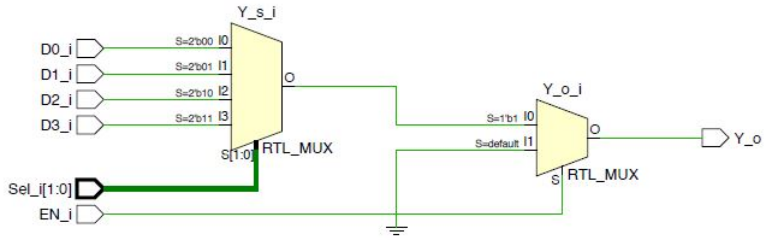


FIGURE 7 – Schéma rtl Multiplexeur 4 vers 1

Description VHDL pour un Mux 4 à 1

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Mux_4x1 is
    Port ( Sel_i : in STD_LOGIC_VECTOR (1 downto 0);
           EN_i : in STD_LOGIC;
           D0_i, D1_i, D2_i, D3_i : in STD_LOGIC;
           Y_o : out STD_LOGIC);
end Mux_4x1;

architecture Behavioral of Mux_4x1 is
    signal Y_s : std_logic;
begin
  with Sel_i select
    Y_s <=  D0_i when "00",
            D1_i when "01",
            D2_i when "10",
            D3_i when "11",
            'X'  when others; -- simulation
  --affectation de la sortie
  Y_o <= Y_s when EN_i = '1' else '0';
end Behavioral;
```

FIGURE 8 – Code Multiplexeur 4 vers 1

5 Comparateur 4 bits

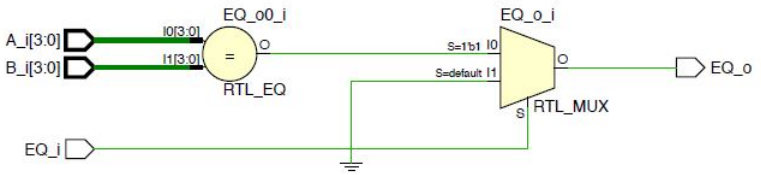


FIGURE 9 – Schéma RTL Comparateur

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Comp_4 is
    Port ( A_i : in STD_LOGIC_VECTOR (3 downto 0);
           B_i : in STD_LOGIC_VECTOR (3 downto 0);
           EQ_i : in STD_LOGIC;
           EQ_o : out STD_LOGIC);
end Comp_4;

architecture Behavioral of Comp_4 is
    signal EQ_s : std_logic;
begin
    -- test de l'égalité
    EQ_s <= '1' when A_i = B_i else '0';
    -- sortie avec l'entrée de chainage
    EQ_o <= EQ_s when EQ_i = '1' else '0';
end Behavioral;
```

FIGURE 10 – Code Comparateur

6 Encodeur de priorité

le But de l'encodeur de priorité est de déterminer l'indice de l'entrée active ayant le degré de priorité le plus élevé

6.1 Schéma et TV Encodeur

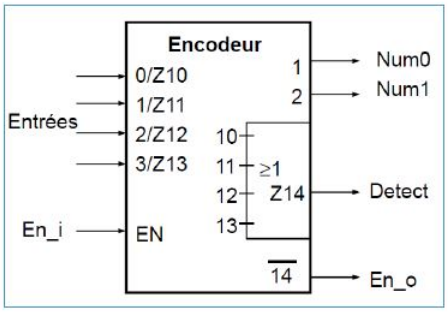


Table de vérité (TDV)

In3	In2	In1	In0	Detect	Num1	Num0
'0'	'0'	'0'	'0'	'0'	'-'	'-'
'0'	'0'	'0'	'1'	'1'	'0'	'0'
'0'	'0'	'1'	x	'1'	'0'	'1'
'0'	'1'	x	x	'1'	'1'	'0'
'1'	x	x	x	'1'	'1'	'1'

Equations logiques

Detect = In3 + In2 + In1 + In0

Num1 = In3 + In2

Num0 = In3 + $\overline{\text{In2}}$. In1

FIGURE 11 – schéma encodeur priorité

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Encode_4 is
    Port ( In_i : in STD_LOGIC_VECTOR(3 downto 0);
           En_i : in STD_LOGIC;
           Num_o : out STD_LOGIC_VECTOR(1 downto 0);
           Detect_o : out STD_LOGIC;
           En_o : out STD_LOGIC);
end Encode_4;

architecture Behavioral of Encode_4 is
    signal Num_s : std_logic_vector(1 downto 0);
    signal Detect_s : std_logic;
begin
    Num_s <= "11" when In_i(3) = '1' else
             "10" when In_i(2) = '1' else
             "01" when In_i(1) = '1' else
             "00" when In_i(0) = '1' else
             "XX"; -- simulation

    Detect_s <= In_i(3) or In_i(2) or In_i(1) or In_i(0);
    Num_o <= Num_s when En_i = '1' else (others => '0');
    Detect_o <= Detect_s when En_i = '1' else '0';
    En_o <= En_i and not Detect_s;
end Behavioral;
Haute école d'ingénierie et d'architecture Fribourg
```

FIGURE 12 – code encodeur priorité

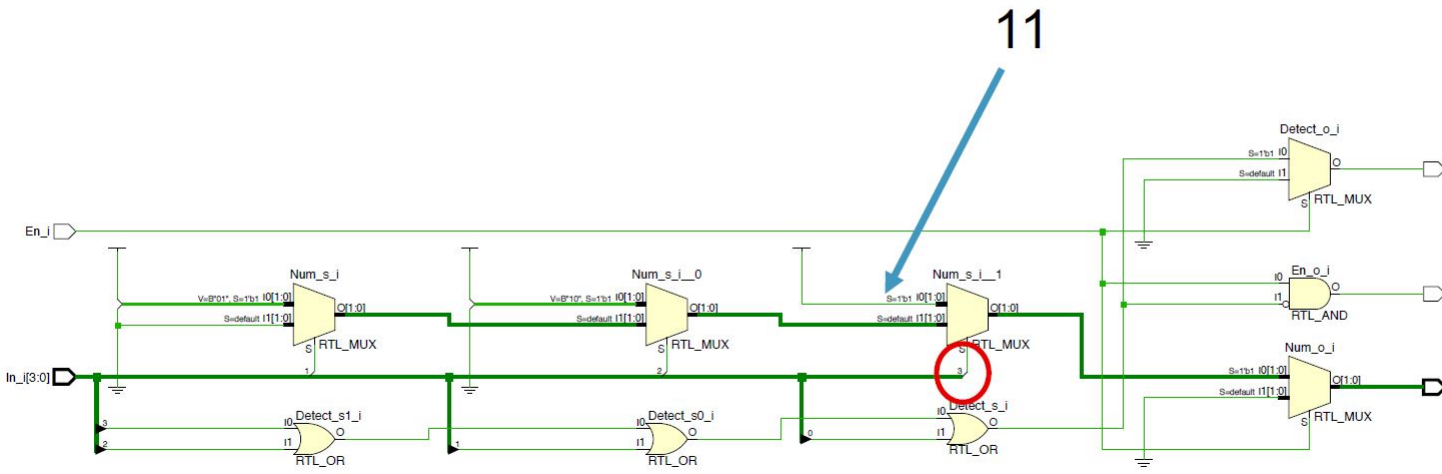


FIGURE 13 – Schéma RTL encodeur priorité

7 Les Bascules

7.1 Etat initial

- Au début d'une simulation, tous les signaux sont à l'état 'U'
- A l'enclenchement d'un système numérique, le contenu des bascules n'est pas déterministe.

→ Une initialisation est indispensable

FIGURE 14 – Etat initial bascules

7.2 Flip Flop

- La façon d'écrire une DFF le plus simplement est la suivante:

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity Flip_Flop is
    port (D : in Std_Logic;
          Clock : in Std_Logic;
          Q : out Std_Logic
    );
end Flip_Flop;

architecture Comport of Flip_Flop is
begin
    process(Clock)
    begin
        if Rising_Edge(Clock) then
            Q <= D;
        end if;
    end process;
end Comport;
```

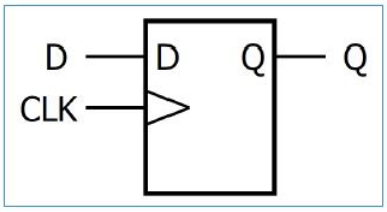
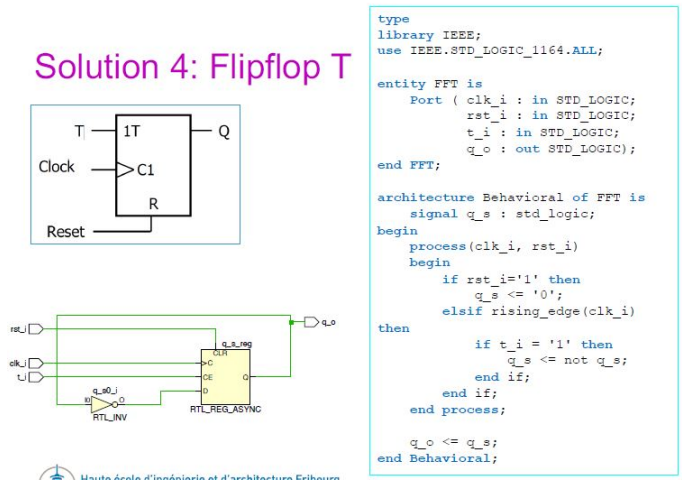


FIGURE 15 – Description simple DFF

- Cette description se synthétise parfaitement sur tous les synthétiseurs actuels
- Deux informations confirment la notion de flip-flop:
  - Le processus ne réagit que sur l'horloge CLK
  - La condition du test spécifie clairement un changement de valeur (rising_edge)

FIGURE 16 – particulatirés de cette definition

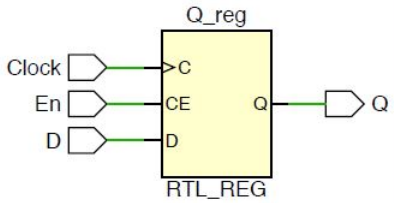
7.3 FlipFlop T




```
type
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity FFT is
    Port ( clk_i : in STD_LOGIC;
           rst_i : in STD_LOGIC;
           t_i : in STD_LOGIC;
           q_o : out STD_LOGIC);
end FFT;

architecture Behavioral of FFT is
    signal q_s : std_logic;
begin
    process(clk_i, rst_i)
    begin
        if rst_i='1' then
            q_s <= '0';
        elsif rising_edge(clk_i)
then
            if t_i = '1' then
                q_s <= not q_s;
            end if;
        end if;
    end process;

    q_o <= q_s;
end Behavioral;
```

FIGURE 17 – FlipFlop

7.4 Flip Flop Enable




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

entity DFFE_I is
    Port ( D, Clock, En: in STD_LOGIC;
           Q : out STD_LOGIC);
end DFFE_I;

architecture Behavioral of DFFE_I is
begin
    process(Clock)
    begin
        if rising_edge(clock) then
            if En = '1' then
                Q <= D;
            end if;
        end if;
    end process;
end Behavioral;
```

FIGURE 18 – Flip Flop Enable

7.5 DFFb

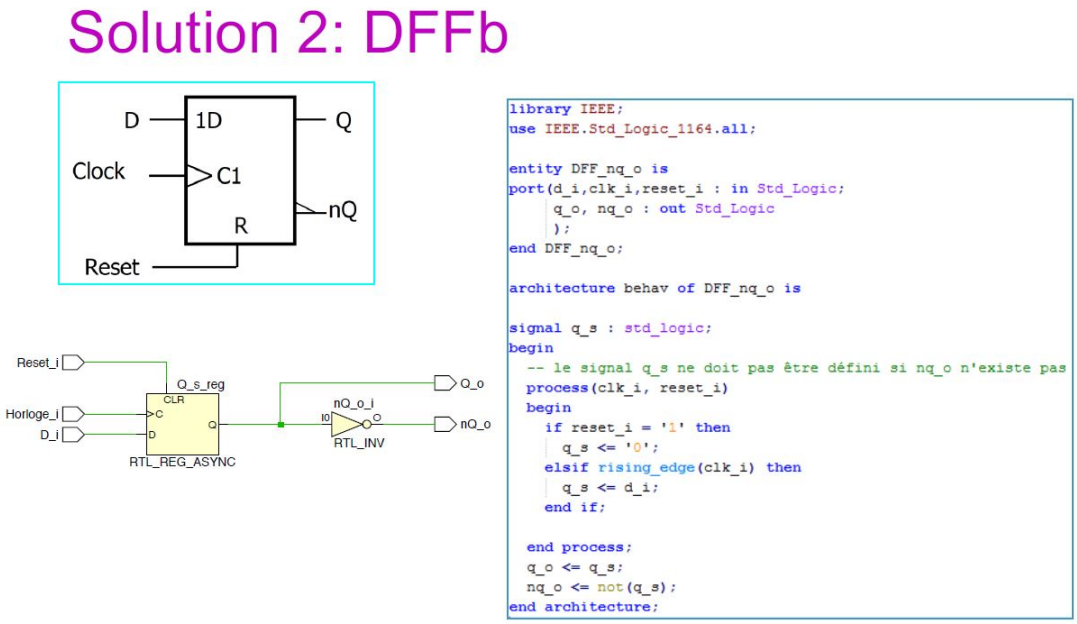


FIGURE 19 – Flip Flop b

7.6 D Latch

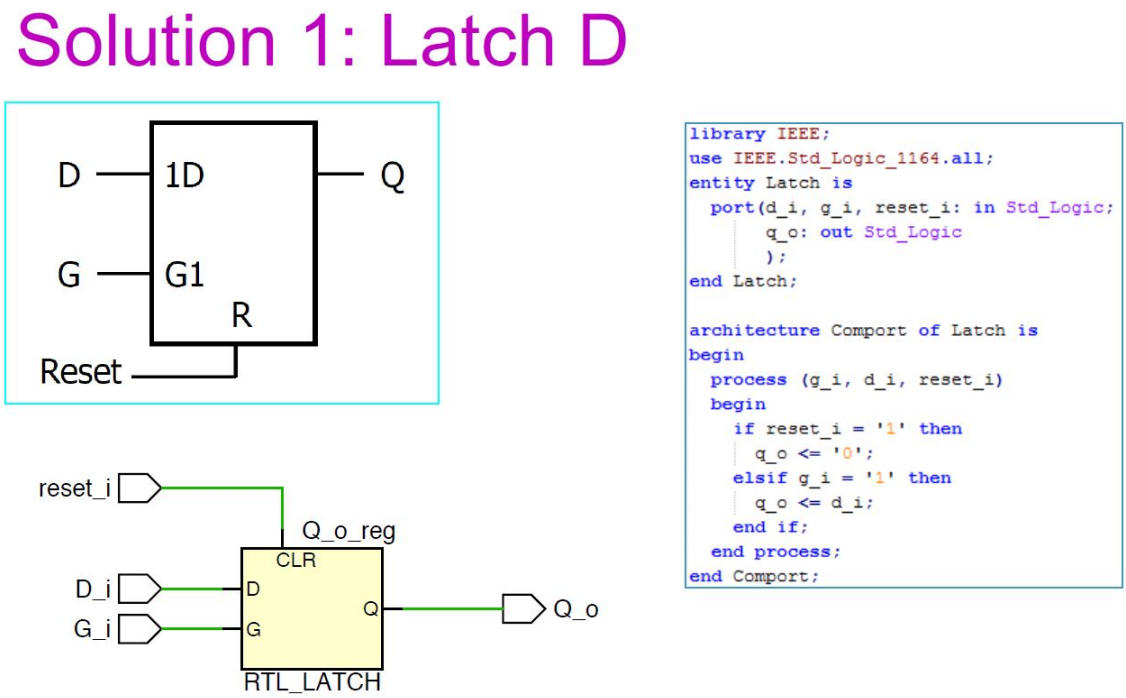


FIGURE 20 – D-Latch

8 Reset synchrone vs asynchrone

Synchrone

```
process(Clock)
begin
    if rising_edge(Clock) then
        if reset = '1' then
            Q1 <= '0';
        else
            Q1 <= D;
        end if;
    end if;
end process;
```

Asynchrone

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

NB: Respectez rigoureusement la liste de sensibilité !

FIGURE 21 – Différences entre les deux

IL EST NECESSAIRE DE N'UTILISER QUUN SEUL TYPE DE RESET PAR PROJET ! ! !

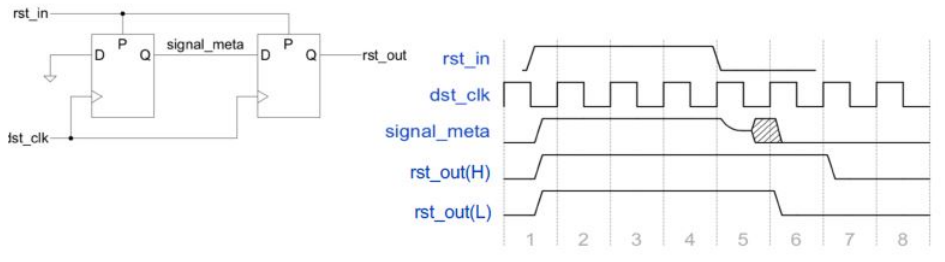
Fonctions d'un Reset

- initialiser FSM
- initialiser Compteur
- initialiser bascules

Problèmes liés aux resets asynchrone :

- Si plusieurs entrées changent en meme temps, avec un système à reset asynchrone, notre machine d'état ne sera plus déterministe.

On peut / doit donc rendre le signal de reset synchrone, pour se faire, procéder comme suit :



- C'est rst_out qui est appliqué au reste du système.

FIGURE 22 – asynchrone to synchrone

9 Bonne partique bien Schroeterproof

- Quelques recommandations de Xilinx :
  - Pour des raisons de performance, utiliser la logique positive pour les signaux de reset, clock enable (CE du DFFE) et un flanc actif montant pour l'horloge.
  - Si le design nécessite de la logique négative:
    - décrire en logique positive tous les composants
    - dans le composant TOP-LEVEL, faire les adaptations.
  - Eliminer les latchs sauf si c'est consciemment voulu; les outils Xilinx avertissent de la présence de latchs.
- Consulter les schémas RTL peut faire économiser beaucoup de temps de développement!

FIGURE 23 – Bonnes pratiques