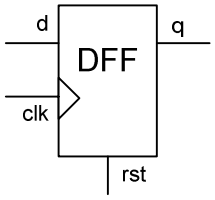
VHDL – TD2

1 Bascules

1.1 Bascule D avec remise à zéro asynchrone.

(DFF with asynchronous reset)



Avec un process ayant une liste de sensibilité.
(V.P 6-1)

```
1 ---------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY dff IS
6   PORT (d, clk, rst: IN STD_LOGIC;
7         q: OUT STD_LOGIC);
8 END dff;
9 ---------------------------------------
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12   PROCESS (clk, rst)
13   BEGIN
14     IF (rst='1') THEN
15       q <= '0';
16     ELSIF (clk'EVENT AND clk='1') THEN
17       q <= d;
18     END IF;
19   END PROCESS;
20 END behavior;
21 ---------------------------------------
```

Avec un process ayant un WAIT.
(V.P 6-4)

```
1 ---------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY dff IS
6   PORT (d, clk, rst: IN STD_LOGIC;
7         q: OUT STD_LOGIC);
8 END dff;
```

```
9 -------------------------------------
10 ARCHITECTURE dff OF dff IS
11 BEGIN
12   PROCESS
13   BEGIN
14     WAIT ON rst, clk;
15     IF (rst='1') THEN
16       q <= '0';
17     ELSIF (clk'EVENT AND clk='1') THEN
18       q <= d;
19     END IF;
20   END PROCESS;
21 END dff;
```

Avec un process incluant un CASE.
(V.P 6-6)

```
1 ---------------------------------------------
2 LIBRARY ieee;                 -- Unnecessary declaration,
3                               -- because
4 USE ieee.std_logic_1164.all; -- BIT was used instead of
5                               -- STD_LOGIC
6 ---------------------------------------------
7 ENTITY dff IS
8   PORT (d, clk, rst: IN BIT;
9         q: OUT BIT);
10 END dff;
11 ---------------------------------------------
12 ARCHITECTURE dff3 OF dff IS
13 BEGIN
14   PROCESS (clk, rst)
15   BEGIN
16     CASE rst IS
17       WHEN '1' => q<='0';
18       WHEN '0' =>
19         IF (clk'EVENT AND clk='1') THEN
20           q <= d;
21         END IF;
22       WHEN OTHERS => NULL; -- Unnecessary, rst is of type
23                            -- BIT
24     END CASE;
25   END PROCESS;
26 END dff3;
```

1.2 Bascule D avec sorties q et qbar .

(V.P 7-4)
Expliquer pourquoi la solution suivante est incorrecte. Proposer une solution correcte.

```
1 ---- Solution 1: not OK ---------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -------------------------------------
```

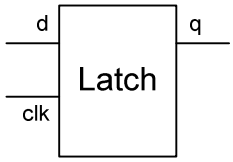
```
5 ENTITY dff IS
6   PORT ( d, clk: IN STD_LOGIC;
7          q: BUFFER STD_LOGIC;
8          qbar: OUT STD_LOGIC);
9 END dff;
10 ---------------------------------------
11 ARCHITECTURE not_ok OF dff IS
12 BEGIN
13   PROCESS (clk)
14   BEGIN
15     IF (clk'EVENT AND clk='1') THEN
16       q <= d;
17       qbar <= NOT q;
18     END IF;
19   END PROCESS;
20 END not_ok;
21 ---------------------------------------
```

Solution correcte (V.P 7-4)

```
1 ---- Solution 2: OK -------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------
5 ENTITY dff IS
6   PORT ( d, clk: IN STD_LOGIC;
7          q: BUFFER STD_LOGIC;
8          qbar: OUT STD_LOGIC);
9 END dff;
10 ---------------------------------------
11 ARCHITECTURE ok OF dff IS
12 BEGIN
13   PROCESS (clk)
14   BEGIN
15     IF (clk'EVENT AND clk='1') THEN
16       q <= d;
17     END IF;
18   END PROCESS;
19   qbar <= NOT q; -- concurrent setting
20 END ok;
21 ---------------------------------------
```

1.3 Latch

Le latch n'est pas une bascule, c'est une mémoire. Tant que clk est à '1', q prend les valeurs de d. Lorsque clk passe à '0', la dernière valeur de d est mémorisée. Programmer un Latch.



```
1 -------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -------------------------------------
5 ENTITY latch IS
6   PORT (d, clk: IN STD_LOGIC;
7         q: OUT STD_LOGIC);
8 END dff;
9 -------------------------------------
10 ARCHITECTURE behavior OF latch IS
11 BEGIN
12   PROCESS (clk, d)
13   BEGIN
14     IF (clk='1') THEN
15       q <= d;
16     END IF;
17   END PROCESS;
18 END behavior;
19 -------------------------------------
```

2 Compteurs

2.1 Compteur décimal 1 chiffre

(V.P 6-2)

```
1 -------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -------------------------------------------
5 ENTITY counter IS
6   PORT (clk : IN STD_LOGIC;
7         digit : OUT INTEGER RANGE 0 TO 9);
8 END counter;
9 -------------------------------------------
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12   count: PROCESS(clk)
13     VARIABLE temp : INTEGER RANGE 0 TO 10;
14   BEGIN
15     IF (clk'EVENT AND clk='1') THEN
16       temp := temp + 1;
17       IF (temp=10) THEN temp := 0;
18       END IF;
19     END IF;
20     digit <= temp;
21   END PROCESS count;
22 END counter;
23 -------------------------------------------
```

2.2 Compteur générique

On crée un paquetage “decimal” défini par:

```
PACKAGE decimal IS
  TYPE chiffre IS RANGE 0 TO 9;
  TYPE nombre IS ARRAY (INTEGER RANGE <>) OF chiffre;

  COMPONENT mod_cnt IS
    GENERIC(modulo: INTEGER:=16);
    PORT (clk, clear, cen: IN BIT;
          cout: OUT INTEGER RANGE 0 TO modulo-1;
          rco : OUT BIT);
    END COMPONENT mod_cnt;
END DECIMAL;
```

Le composant “mod_cnt” est un compteur modulo générique pour lequel on demande d’écrire la spécification en VHDL.

```
ENTITY mod_cnt IS
  GENERIC(modulo: INTEGER:=16);
  PORT (clk, clear, cen: IN BIT;
        cout: OUT INTEGER RANGE 0 TO modulo-1;
        rco : OUT BIT);
END mod_cnt;

-------------------------------------------------
ARCHITECTURE cnt_beh OF mod_cnt IS
  SIGNAL state: INTEGER RANGE 0 TO modulo-1;
BEGIN
  rco <= cen WHEN state = modulo-1 ELSE ’0’;
  cout <= state;
  cnt: PROCESS
  BEGIN
    WAIT UNTIL clk = ’1’;
    IF clear = ’1’ THEN
      state <= 0;
    ELSIF cen = ’1’ THEN
      state <= (state + 1) mod modulo;
    END IF;
  END PROCESS cnt;
END cnt_beh;
```

Ce compteur est compilé dans une bibiothèque genlib. Expliquer comment l’instancier pour obtenir un compteur modulo 10 (entité decade).

```
USE work.decimal.all;

ENTITY decade IS
  PORT (clk, raz, en: IN BIT;
        cout: OUT chiffre;
        dix : OUT BIT);
  END decade;

LIBRARY genlib ;
```

```
ARCHITECTURE instance OF decade IS
BEGIN
  dec : mod_cnt
     GENERIC MAP (modulo=>10)
     PORT MAP (clk=>clk, clear=>raz, cen=>en,
               cout=> integer(count), rco=>dix);
END instance;


-------------------------------------------------
ARCHITECTURE cnt_beh OF mod_cnt IS
  SIGNAL state: INTEGER RANGE 0 TO modulo-1;
BEGIN
  rco <= cen WHEN state = modulo-1 ELSE '0';
  cout <= state;
  cnt: PROCESS
  BEGIN
    WAIT UNTIL clk = '1';
    IF clear = '1' THEN
      state <= 0;
    ELSIF cen = '1' THEN
      state <= (state + 1) mod modulo;
    END IF;
  END PROCESS cnt;
END cnt_beh;
```

3 Additionneurs

3.1 Additionneur avec propagation de retenue

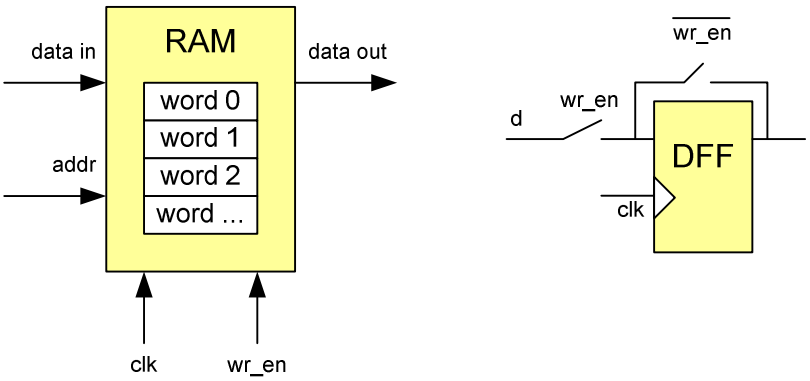
(Carry Ripple Adder) La solution doit être élaborée à partir d'additionneurs complets 1 bit (1-bit full-adder). On demande une solution générique utilisant des STD_LOGIC_VECTORs et une solution non-générique utilisant les entiers.
(V.P 6-8)

```
1 ----- Solution 1: Generic, with VECTORS --------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ------------------------------------------------
5 ENTITY adder IS
6   GENERIC (length : INTEGER := 8);
7   PORT ( a, b: IN STD_LOGIC_VECTOR (length-1 DOWNTO 0);
8          cin: IN STD_LOGIC;
9          s: OUT STD_LOGIC_VECTOR (length-1 DOWNTO 0);
10         cout: OUT STD_LOGIC);
11 END adder;
12 ------------------------------------------------
13 ARCHITECTURE adder OF adder IS
14 BEGIN
15   PROCESS (a, b, cin)
16     VARIABLE carry : STD_LOGIC_VECTOR (length DOWNTO 0);
```

```
  BEGIN
    carry(0) := cin;
    FOR i IN 0 TO length-1 LOOP
      s(i) <= a(i) XOR b(i) XOR carry(i);
      carry(i+1) := (a(i) AND b(i)) OR (a(i) AND
                    carry(i)) OR (b(i) AND carry(i));
    END LOOP;
    cout <= carry(length);
  END PROCESS;
END adder;
-------------------------------------------------
```

```
---- Solution 2: non-generic, with INTEGERS ----
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------
ENTITY adder IS
  PORT ( a, b: IN INTEGER RANGE 0 TO 255;
         c0: IN STD_LOGIC;
         s: OUT INTEGER RANGE 0 TO 255;
         c8: OUT STD_LOGIC);
END adder;
-------------------------------------------------
ARCHITECTURE adder OF adder IS
BEGIN
  PROCESS (a, b, c0)
    VARIABLE temp: INTEGER RANGE 0 TO 511;
  BEGIN
    IF (c0='1') THEN temp:=1;
    ELSE temp:=0;
    END IF;
    temp:= a + b + temp;
    IF (temp > 255) THEN
      c8 <= '1';
      temp:= temp - 256;
    ELSE c8 <= '0';
    END IF;
    s <= temp;
  END PROCESS;
END adder;
-------------------------------------------------
```

4 RAM



La mémoire a un bus d'entrée (data in), un bus d'adresse (addr), une horloge (clk) et une validation d'écriture (wr_en). Quand wr_en est activé, data in est rangé dans le mot d'adresse addr lors du front suivant de clk. Quant à la sortie data out elle doit en permanence afficher le mot sélectionné par addr. Le schéma de droite résume ce comportement.
Programmer une mémoire RAM générique de `words` mots de `bits` bits (défauts: `bits`=8 et `words`=16).

```
1 ---------------------------------------------------
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 ---------------------------------------------------
5 ENTITY ram IS
6   GENERIC ( bits: INTEGER:= 8; -- # of bits per word
7             words: INTEGER:= 16); -- # of words in the memory
8   PORT ( wr_ena, clk: IN STD_LOGIC;
9          addr: IN INTEGER RANGE 0 TO words-1;
10         data_in: IN STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
11         data_out: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
12 END ram;
13 ---------------------------------------------------
14 ARCHITECTURE ram OF ram IS
15   TYPE vector_array IS ARRAY (0 TO words-1) OF
16     STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
17   SIGNAL memory: vector_array;
18 BEGIN
19   PROCESS (clk, wr_ena)
20   BEGIN
21     IF (wr_ena='1') THEN
22       IF (clk'EVENT AND clk='1') THEN
23         memory(addr) <= data_in;
24       END IF;
25     END IF;
26   END PROCESS;
27   data_out <= memory(addr);
28 END ram;
29 ---------------------------------------------------
```

Cette RAM générique est compilée dans la bibliothèque `mem`. Expliquer comment instancier une RAM de 16 mots de 8 bits.

```
USE work.mem.all;

ENTITY ram16x8 IS
  PORT ( wr_ena, clk: IN STD_LOGIC;
         addr: IN INTEGER RANGE 0 TO words-1;
         data_in: IN STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
         data_out: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
  END ram;
-----------------------------------------------------

ARCHITECTURE ram16x8 OF ram IS
BEGIN
  Inst: ram
    GENERIC MAP (
      words => 16,
      bits => 8
    )
    PORT MAP (
      addr =>  addrBus,
      data_in => dataBus1,
      wr_en => w,
      clk => h,
      data_out => dataBus2
   );
END ram16x8;
```