INF1500 : Logique des systèmes numériques

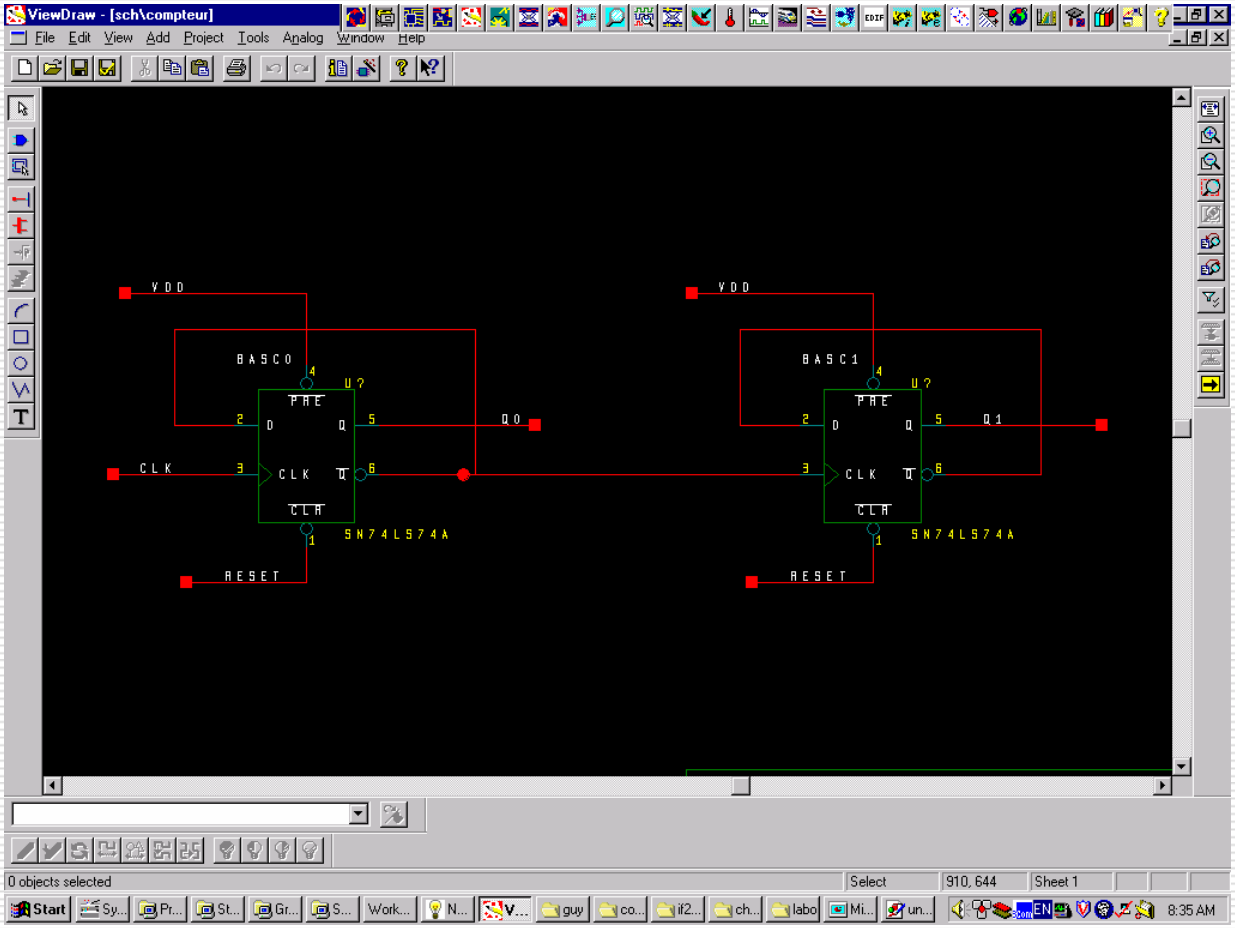
- Cours 8: VHDL

Entrées (Descriptions)

- Schéma
- HDL – Hardware Description Language (ABEL, PALASM, Verilog, VHDL, etc.)
  - Hard cores, soft cores
- Schéma + HDL

  - Note: VHDL (VHSIC Description Langage où VHSIC signifie Very High Speed Integrated Circuit)

Schéma


ViewDraw - [sch\compteur]
File Edit View Add Project Tools Analog Window Help
VDD
BASC0
U?
PRE
D
Q
CLK
CLR
Q0
RESET
SN74LS74A
BASC1
Q1
0 objects selected
Select
910, 644
Sheet 1
Start
8:35 AM

Introduction au VHDL

- **Principes de base et caractéristiques**
- **Flot de conception en VHDL**
- **Structure d'un programme**
- **Types et constantes**
- **Design niveau structurel**
- **Design niveau flots de données**

Introduction au VHDL

- **Design niveau comportemental**
- **Fonctions et procédures**
- **Librairies et *packages***
- **Notions de temps**
- **Synthèse**
- **Exemple de description VHDL pour des composantes MSI (chap. 5)**
- **Design niveau structurel (notions plus avancés)**

Principe de base du VHDL

- VHDL signifie VHSIC Description Langage

  où VHSIC signifie Very High Speed Integrated Circuit
- Milieu des années 1980 par le département de la défense américaine (en même temps que le langage ADA)
- Standard accepté en 1987 (VHDL-1987), étendu en 1993 (VHDL-1993) et plus récemment en 2002 (VHDL2002);

VHDL – Exemple: DFF

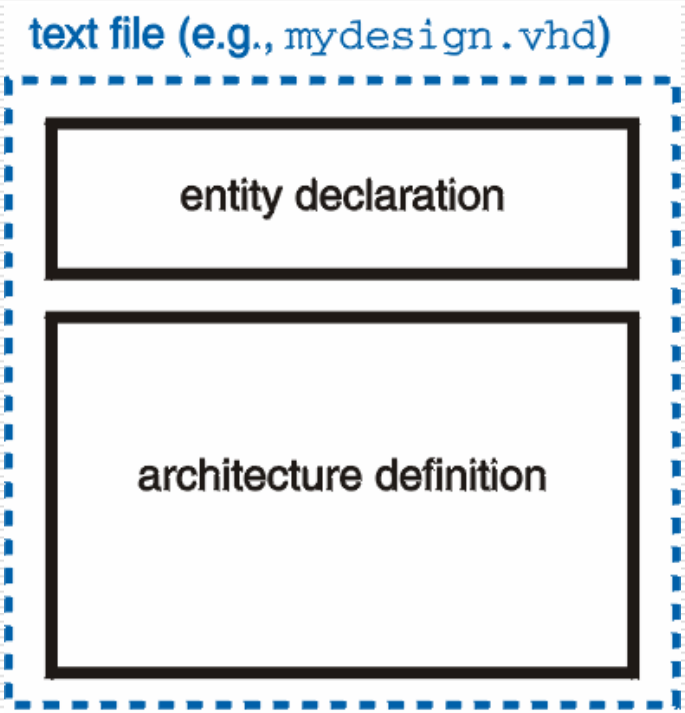
Table 8-7 VHDL model of a 74x74-like D flip-flop with preset and clear.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vdff74 is
  port (D, CLK, PR_L, CLR_L: in STD_LOGIC;
        Q, QN: out STD_LOGIC );
end Vdff74;

architecture Vdff74_b of Vdff74 is
signal PR, CLR: STD_LOGIC;
begin
process(CLR_L, CLR, PR_L, PR, CLK)
  begin
    PR <= not PR_L; CLR <= not CLR_L;
    if (CLR and PR) = '1' then Q <= '0'; QN <= '0';
    elsif CLR = '1' then Q <= '0'; QN <= '1';
    elsif PR = '1' then Q <= '1'; QN <= '0';
    elsif (CLK'event and CLK='1') then Q <= D; QN <= not D;
    end if;
  end process;
end Vdff74_b;
```

Caractéristiques du VHDL

- Les designs peuvent être décomposés hiérarchiquement (diminue la complexité);
- Chaque design a son interface (connections) et une spécification comportementale précise (simulation);
- La spécification comportementale peut être un algorithme ou un circuit schématique;
- La concurrence, le temps et les horloges sont modélisées;
- Les opérations logiques et l'éxecution d'un comportement peuvent être simulées.

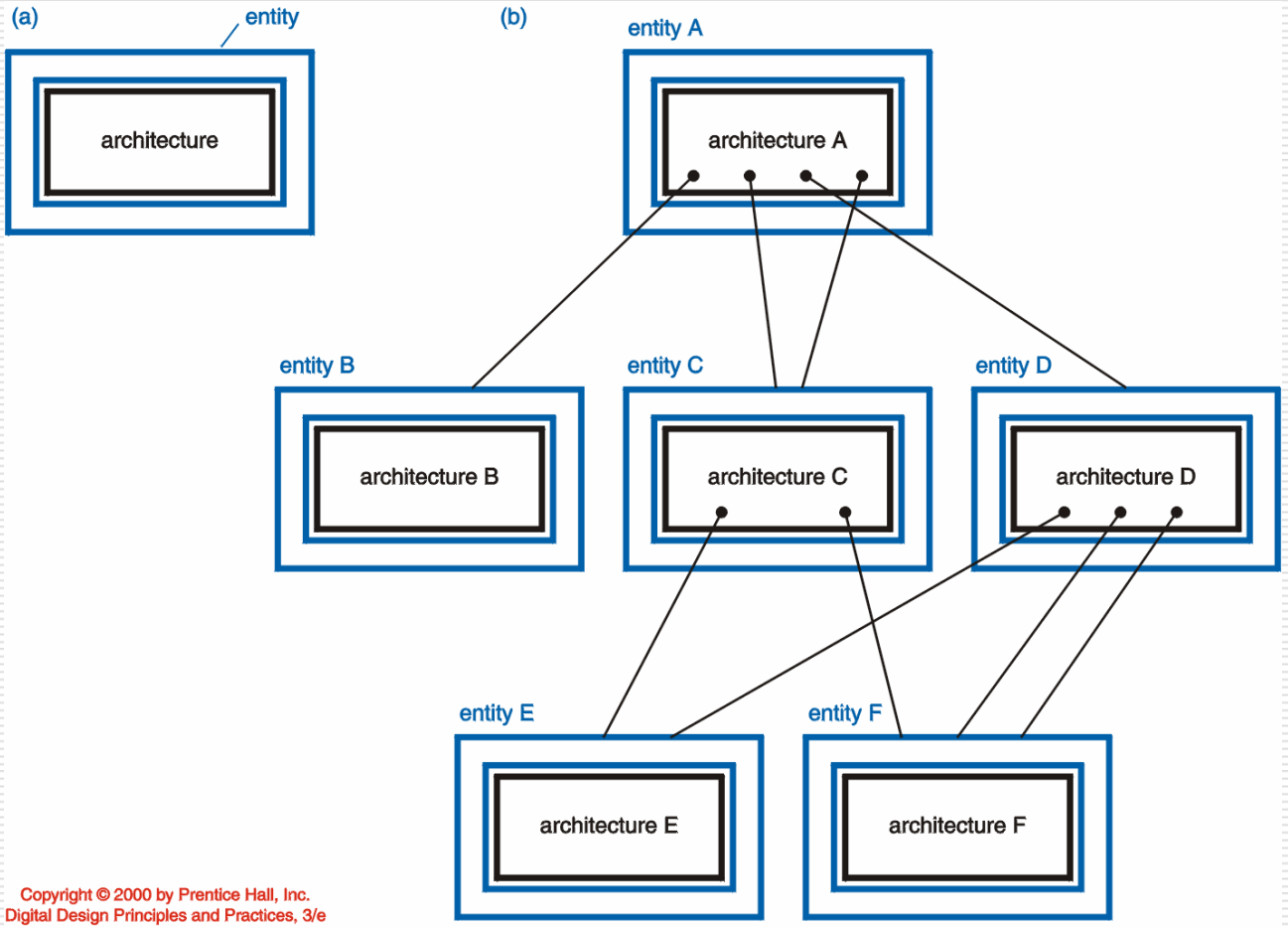
Structure d'un programme



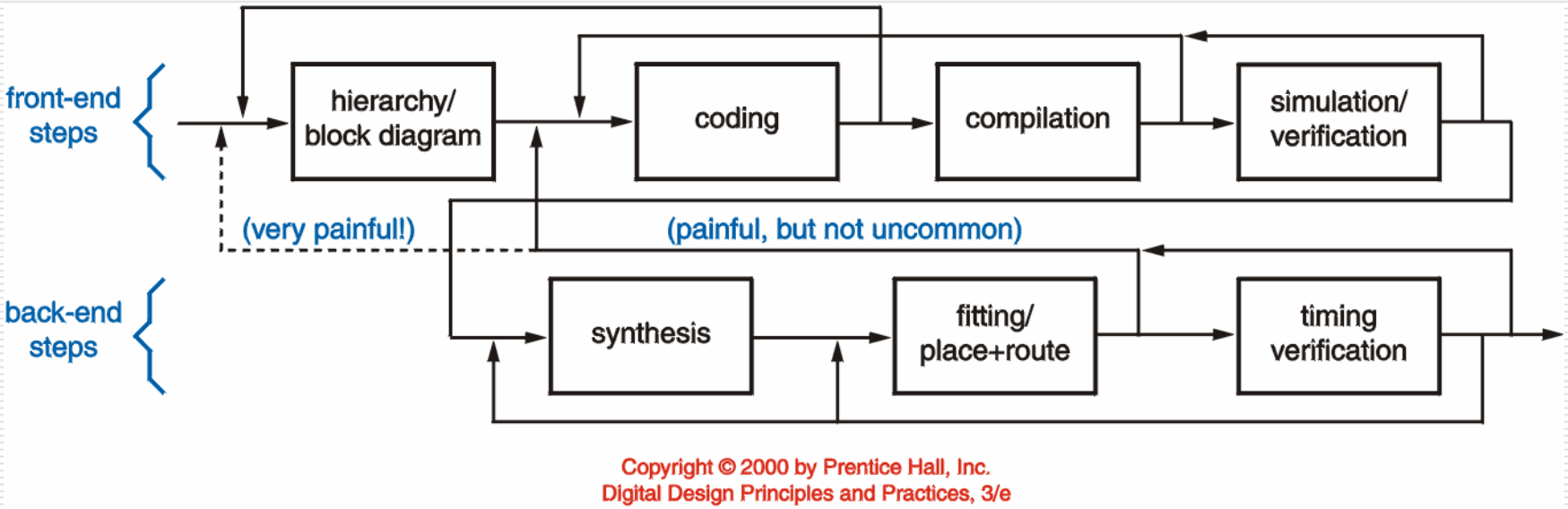
Exemple d'entité et d'architecture

```
entity Inhibit is     -- also known as 'BUT-NOT'
  port (X,Y: in BIT;     --  as in 'X but not Y'
        Z:   out BIT);   --  (see [Klir, 1972])
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

Conception VHDL



Flot de conception VHDL



Exemple d'entité et d'architecture

```
entity Inhibit is     -- also known as 'BUT-NOT'
  port (X,Y: in BIT;     --  as in 'X but not Y'
        Z:   out BIT);   --  (see [Klir, 1972])
end Inhibit;

architecture Inhibit_arch of Inhibit is
begin
  Z <= '1' when X='1' and Y='0' else '0';
end Inhibit_arch;
```

Table 5-3

Syntaxe exacte

```
entity entity-name is
  port (signal-names : mode signal-type;
        signal-names : mode signal-type;
        ...
        signal-names : mode signal-type);
end entity-name;
```

Table 5-14

```
architecture architecture-name of entity-name is
  type declarations
  signal declarations
  constant declarations
  function definitions
  procedure definitions
  component declarations
begin
  concurrent-statement
  ...
  concurrent-statement
end architecture-name;
```

Table 5-15

Types et constantes

`bit`	`character`	`severity_level`
`bit_vector`	`integer`	`string`
`boolean`	`real`	`time`

Table 5-16

integer **Operators**	
+	addition
-	subtraction
*	multiplication
/	division
mod	modulo division
rem	modulo remainder
abs	absolute value
**	exponentiation

boolean **Operators**	
and	AND
or	OR
nand	NAND
nor	NOR
xor	Exclusive OR
xnor	Exclusive NOR
not	complementation

Table 5-17

Type et sous-type prédéfinis *std_logic*

```
type STD_ULOGIC is ( 'U',  -- Uninitialized
                     'X',  -- Forcing  Unknown
                     '0',  -- Forcing  0
                     '1',  -- Forcing  1
                     'Z',  -- High Impedance
                     'W',  -- Weak     Unknown
                     'L',  -- Weak     0
                     'H',  -- Weak     1
                     -     -- Don't care
                   );
subtype STD_LOGIC is resolved STD_ULOGIC;
```

Table 5-19

Type et sous-type prédéfinis *std_logic*

```
SUBTYPE UX01 IS resolved std_ulogic RANGE 'U' TO '1';
                                              -- ('U','X','0','1')
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;

-- truth table for "and" function
CONSTANT and_table : stdlogic_table := (
--      ---------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -      |   |
--      ---------------------------------------------------
        ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ),  -- | U |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | X |
        ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | 0 |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 1 |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | Z |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | W |
        ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | L |
        ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | H |
        ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )   -- | - |
);
FUNCTION "and" ( L : std_ulogic; R : std_ulogic ) RETURN UX01 IS
BEGIN
    RETURN (and_table(L, R));
END "and";
```

Table 5-24

p.278, tables 4-37

Type et sous-type prédéfinis *std_logic*

```
type monthly_count is array (1 to 12) of integer;
type byte is array (7 downto 0) of STD_LOGIC;

constant WORD_LEN: integer := 32;
type word is array (WORD_LEN-1 downto 0) of STD_LOGIC;

constant NUM_REGS: integer := 8;
type reg_file is array (1 to NUM_REGS) of word;

type statecount is array (traffic_light_state) of integer;
```

Table 5-21

Design niveau structurel

- Équivalent à un circuit en format schématique (block diagram);
- *Signal* pour décrire les interconnexions (nœuds);
- *Component* et *port map* pour décrire les portes;

Exemple: Détecteur de nombres premiers (sans optimisation)

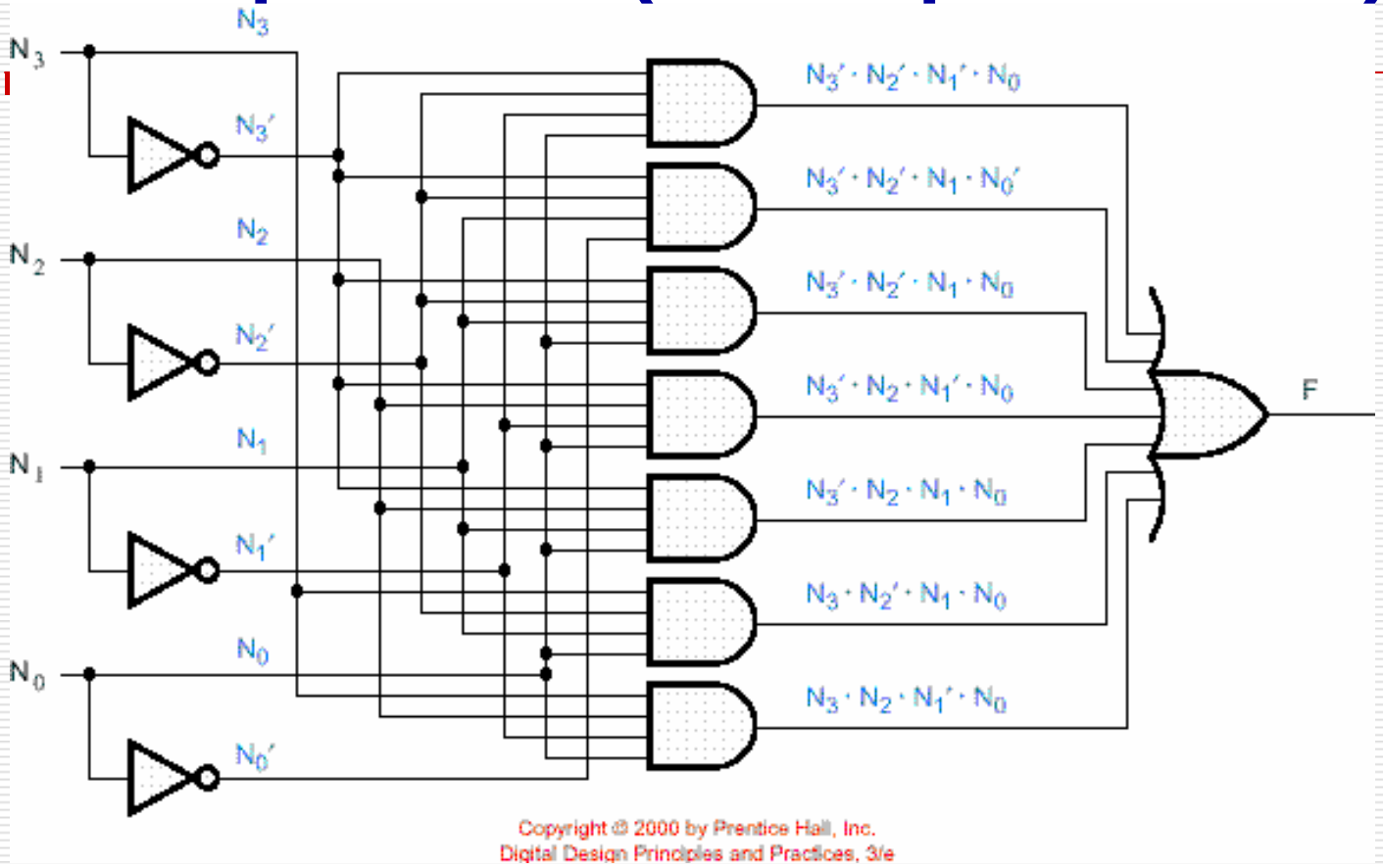


Fig. 4-18

Exemple: Détecteur de nombres premiers (après optimisation)

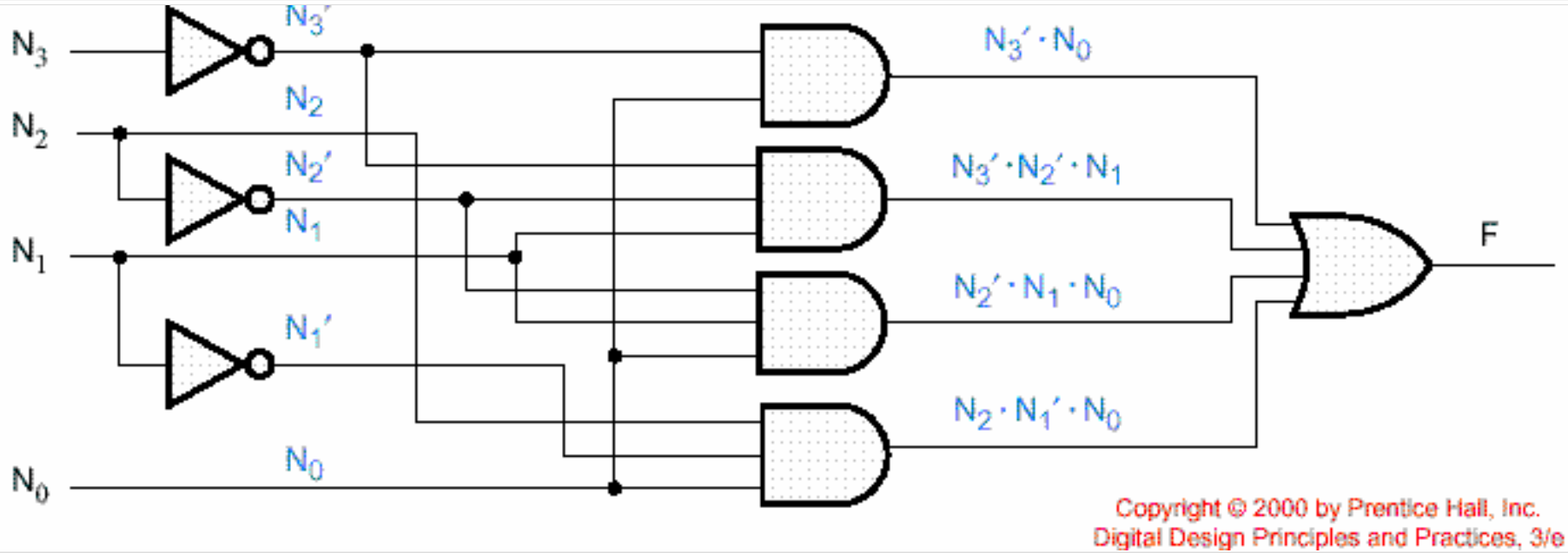


Fig. 4-25

VHDL structurel du détecteur de nombre premiers optimisé

```
library IEEE;
use IEEE.std_logic_1164.all;

entity prime is
    port ( N: in STD_LOGIC_VECTOR (3 downto 0);
           F: out STD_LOGIC );
end prime;

architecture prime1_arch of prime is
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INV port (I: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND2 port (I0,I1: in STD_LOGIC; O: out STD_LOGIC); end component;
component AND3 port (I0,I1,I2: in STD_LOGIC; O: out STD_LOGIC); end component;
component OR4 port (I0,I1,I2,I3: in STD_LOGIC; O: out STD_LOGIC); end component;
begin
  U1: INV port map (N(3), N3_L);
  U2: INV port map (N(2), N2_L);
  U3: INV port map (N(1), N1_L);
  U4: AND2 port map (N3_L, N(0), N3L_N0);
  U5: AND3 port map (N3_L, N2_L, N(1), N3L_N2L_N1);
  U6: AND3 port map (N2_L, N(1), N(0), N2L_N1_N0);
  U7: AND3 port map (N(2), N1_L, N(0), N2_N1L_N0);
  U8: OR4 port map (N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0, F);
end prime1_arch;
```

Table 5-30

Design niveau flot de données

- **Le design est décrit en termes du flot des données et des opérations;**
- **Peut être exprimé par un assignation de signaux concurrents (voir *prime3_arch*);**
- **Peut aussi être simplement une table de vérité (voir *prime4_arch ou prime5_arch*)**

Prime3_arch

```
architecture prime3_arch of prime is
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
begin
  N3L_N0     <= '1' when N(3)='0' and N(0)='1' else '0';
  N3L_N2L_N1 <= '1' when N(3)='0' and N(2)='0' and N(1)='1' else '0';
  N2L_N1_N0  <= '1' when N(2)='0' and N(1)='1' and N(0)='1' else '0';
  N2_N1L_N0  <= '1' when N(2)='1' and N(1)='0' and N(0)='1' else '0';
  F <= N3L_N0 or N3L_N2L_N1 or N2L_N1_N0 or N2_N1L_N0;
end prime3_arch;
```

Table 5-38

Prime4_arch (instruction *with*)

```
architecture prime4_arch of prime is
begin
  with N select
    F <= '1' when "0001",
         '1' when "0010",
         '1' when "0011" | "0101" | "0111",
         '1' when "1011" | "1101",
         '0' when others;
end prime4_arch;
```

Table 5-41

Prime5_arch

```
architecture prime5_arch of prime is
begin
  with CONV_INTEGER(N) select
    F <= '1' when 1 | 2 | 3 | 5 | 7 | 11 |
         '0' when others;
end prime5_arch;
```

Table 5-41

Design niveau comportemental

- Peut être exprimé par un algorithme (voir prime7_arch, prime8_arch et prime9_arch);
- On utilise le concept de processus
  - Chaque processus exécute ses instructions de manière séquentielle;
  - Rebouclage implicite à la fin du processus
  - Plusieurs processus définissent des actions qui se déroulent en parallèle dans un système.

Design niveau comportemental

- **Liste de sensibilité statique**
  - **Soit process(N);**
  - **Au départ tous les processus sont exécutés une fois;**
  - **Une fois le corps du processus exécuté, celui-ci attend un événement sur le signal N pour poursuivre l'exécution à partir du début.**

Process en VHDL

```
process (signal-name, signal-name, ..., signal-name)
   type declarations
   variable declarations
   constant declarations
   function definitions
   procedure definitions
begin
   sequential-statement
   ...
   sequential-statement
end process;
```

Table 5-42

Prime7_arch (instruction *if then else*)

```
architecture prime7_arch of prime is
begin
  process(N)
    variable NI: INTEGER;
  begin
    NI := CONV_INTEGER(N);
    if NI=1 or NI=2 then F <= '1';
    elsif NI=3 or NI=5 or NI=7 or NI=11 or NI=13 then F <= '1';
    else F <= '0';
    end if;
  end process;
end prime7_arch;
```

Le processus est invoqué à chaque fois que N va changer de valeur

Table 5-45

Fonction CONV_INTEGER

```
function CONV_INTEGER (X: STD_LOGIC_VECTOR) return INTEGER :
  variable RESULT: INTEGER;
begin
  RESULT := 0;                                                    R
  for i in X'range loop
    RESULT := RESULT * 2;
    case X(i) is
      when '0' | 'L'  => null;
      when '1' | 'H'  => RESULT := RESULT + 1;
      when others     => null;
    end case;
  end loop;
  return RESULT;
end CONV_INTEGER;
```

Autre exemple: une bascule D

VHDL model of a 74x74-like D flip-flop with preset and clear.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Vdff74 is
  port (D, CLK, PR_L, CLR_L: in STD_LOGIC;
        Q, QN: out STD_LOGIC );
end Vdff74;

architecture Vdff74_b of Vdff74 is
signal PR, CLR: STD_LOGIC;
begin
process(CLR_L, CLR, PR_L, PR, CLK)
  begin
    PR <= not PR_L; CLR <= not CLR_L;
    if (CLR and PR) = '1' then Q <= '0'; QN <= '0';
    elsif CLR = '1' then Q <= '0'; QN <= '1';
    elsif PR = '1' then Q <= '1'; QN <= '0';
    elsif (CLK'event and CLK='1') then Q <= D; QN <= not D;
    end if;
  end process;
end Vdff74_b;
```

Le processus est invoqué à chaque fois qu'un signal d'entrée (D, CLK, PR_L ou CLR_L) de la bascule change.

Prime8_arch (instruction *case*)

```
architecture prime8_arch of prime is
begin
  process(N)
  begin
    case CONV_INTEGER(N) is
      when 1 => F <= '1';
      when 2 => F <= '1';
      when 3 | 5 | 7 | 11 | 13 => F <= '1';
      when others => F <= '0';
    end case;
  end process;
end prime8_arch;
```

Table 5-47

Prime9_arch (instruction *for*)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity prime9 is
    port ( N: in STD_LOGIC_VECTOR (15 downto 0);
           F: out STD_LOGIC );
end prime9;

architecture prime9_arch of prime9 is
begin
  process(N)
  variable NI: INTEGER;
  variable prime: boolean;
  begin
    NI := CONV_INTEGER(N);
    prime := true;
    if NI=1 or NI=2 then null; -- take care of boundary cases
    else for i in 2 to 253 loop
           if NI mod i = 0 then
             prime := false; exit;
           end if;
         end loop;
    end if;
    if prime then F <= '1'; else F <= '0'; end if;
  end process;
end prime9_arch;
```

C'est donc beaucoup plus simple pour déterminer n'importe quel nombre premier

Table 5-50

Fonctions et procédures

- Semblables à ce qu'on retrouve dans les langages de programmation de haut niveau.
- On les utilise surtout pour la conversion de type, la résolution de vecteur ou encore pour définir des expressions booléennes à haut niveau (OR, AND, etc.).

Librairies et packages

- **Permet la réutilisation de code.**
- **On utilise l'énoncé suivant:**

```
package package-name is
   type declarations
   signal declarations
   constant declarations
   component declarations
   function declarations
   procedure declarations
end package-name;
package body package-name is
   type declarations
   constant declarations
   function definitions
   procedure definitions
end package-name;
```

Librairies et packages

- Exemple: voir le fichier spartan3.vhd

- Une fois que la librairie est définie, les instructions suivantes permettent de l'utiliser:
  - library IEEE;
  - use IEEE.STD_LOGIC_1164.all;

Notion de temps

- **Pour tenir compte des délai on a l'énoncé *after*:**
  - **Z <= ´1´ after 4 ns when X= ´1´ and Y = ´1´ else ´0´ after 3 ns;**
- **Pour la synchronisation de processus, on a les énoncés *wait*:**
  - **wait for 10 ns;**
  - **wait on <nom de signal>**

Notion de temps

- **Exemple du producteur/consommateur à travers un seul tampon (*cons* et *prod* sont deux signaux définis dans une architecture contenant P1 et P2):**

process P1

.
écriture d'une 1er
valeur dans le tampon
prod <= not prod
wait on cons
écriture d'une 2e
valeur dans le tampon
.
.

process P2

.
lecture de la 1er
valeur dans le tampon
cons <= not cons
wait on prod
écriture de la 2e
valeur dans le tampon
.
.

Synthèse

- **Le code décrit à un niveau d'abstraction trop élevé (e.g. comportemental) peut réserver des surprises (e.g. surface du circuit) à la synthèse.**

- **Par exemple à la table 5-50, l'opération *modulo* demande l'utilisation d'un diviseur entier. Or, beaucoup d'outils de synthèse sont incapables de synthétiser la division (sauf la division par 2, i.e. le shift).**

- **De plus la boucle *i in 2 to 253* implique que l'outil de synthèse va dérouler la boucle et générer 252 diviseurs combinatoires…**

Exemple de description VHDL pour des composantes MSI (chap. 5)

- **Bascule D (74LS74)**
  - **Section 8.2.7 et Tables 8-6 et 8-7**
- **Décodeur (74LS138)**
  - **Section 6.4.6 et Tables 6-13 à 6-19**
- **Encodeur (74LS148)**
  - **Section 6.5.4 et Table 6-30**
- **Multiplexeur**
  - **Section 6.7.5 et Table 6-49 et 6-50**
- **Démultiplexeur à partir d'un 74LS138**
  - **À faire en exercice**
- **Compteur 4 bits (74LS163)**
  - **Section 8.4.6 et Table 8-16**

Design niveau structurel (notions plus avancées)

- Énoné *generic* et *generic map*

Énoncé generic

- Il est aussi possible de rendre variable le nombre de signaux d'entrée d'une porte
- Par exemple, on souhaite réutiliser la même porte avec deux ou trois entrées selon le contexte.
- Ou encore, on souhaite parfois pouvoir assigner un délai à chaque porte
- Exemple: Dans *prime1_arch*, on veut faire de la réutilisation de portes au niveau du AND ainsi qu'ajouter des délais à chaque porte:

```
entity ANDGATE is
      generic (N:   Positive := 2;                          -- number of inputs
       tLH: Time := 0 ns;                                   -- rise inertial delay
       tHL: Time := 0 ns;                                   -- fall inertial delay
       strn: STRENGTH := strn_X01);                                 -- output
      strength
      port (Input: in STD_LOGIC_VECTOR (1 to N);                    -- inputs
          Output: out STD_LOGIC);                           -- output
end ANDGATE;
```

Voir définition complète dans le fichier *std_logic_entities.vhd*

Énoncé generic

- Une fois que notre composante est définie avec l'énoncé *generic*, elle peut ensuite être instantiée avec *generic map* (tout comme *port map*).
- Voir l'exemple de *prime1_arch*

```
-- prime1_arch
architecture prime1_arch of prime is
signal INP1: STD_LOGIC_VECTOR (1 to 2);
signal INP2, INP3, INP4: STD_LOGIC_VECTOR (1 to 3);
signal INP5: STD_LOGIC_VECTOR (1 to 4);
signal N3_L, N2_L, N1_L: STD_LOGIC;
signal N3L_N0, N3L_N2L_N1, N2L_N1_N0, N2_N1L_N0: STD_LOGIC;
component INVGATE generic (tLH : time;
                           tHL : time);
          port (Input: in STD_LOGIC;          --inputs
          Output: out STD_LOGIC); end component;
component ANDGATE generic (N : positive;
                  tLH : time;
          tHL : time);
     port (Input: in STD_LOGIC_VECTOR (1 to N);          -- inputs
          Output: out STD_LOGIC); end component;
component ORGATE  generic (N : positive;
                           tLH : time;
          tHL : time);
     port (Input: in STD_LOGIC_VECTOR (1 to N);          -- inputs
          Output: out STD_LOGIC); end component;
```

```
-- prime1_arch (suite)
begin
 U1: INVGATE generic map (tLH => 10 ns, tHL => 10 ns) port map (N(3), N3_L);
 U2: INVGATE generic map (tLH => 10 ns, tHL => 10 ns) port map (N(2), N2_L);
 U3: INVGATE generic map (tLH => 10 ns, tHL => 10 ns) port map (N(1), N1_L);
 INP1(1) <= N3_L;
 INP1(2) <= N(0);
 U4: ANDGATE generic map (N => 2, tLH => 15 ns, tHL => 15 ns) port map (INP1,
      N3L_N0);
 INP2(1) <= N3_L;
 INP2(2) <= N2_L;
 INP2(3) <= N(1);
 U5: ANDGATE generic map (N => 3, tLH => 15 ns, tHL => 15 ns) port map (INP2,
      N3L_N2L_N1);
 INP3(1) <= N2_L;
 INP3(2) <= N(1);
 INP3(3) <= N(0);
 U6: ANDGATE generic map (N => 3, tLH => 15 ns, tHL => 15 ns) port map (INP3,
      N2L_N1_N0);
 INP4(1) <= N(2);
 INP4(2) <= N1_L;
 INP4(3) <= N(0);
 U7: ANDGATE generic map (N => 3, tLH => 15 ns, tHL => 15 ns) port map (INP4,
      N2_N1L_N0);
 INP5(1) <= N3L_N0;
 INP5(2) <= N3L_N2L_N1;
 INP5(3) <= N2L_N1_N0;
 INP5(4) <= N2_N1L_N0;
 U8: ORGATE generic map (N => 4, tLH => 20 ns, tHL => 20 ns) port map (INP5, F);
end prime1_arch;
```