Tronc commun 3A - Durée : 90 mn **Examen du** **Cours EE341** **NOM :**	**Session 1 année 2013 – 2014** **Un seul document autorisé :** **« VHDL résumé de syntaxe »** **Calculatrice interdite** **PRENOM :**
1 Les points donnés dans l'énoncé entre crochets [X] après chaque question représentent le barème et indiquent le temps à passer en minutes sur chaque question *2 La qualité de la rédaction (lisibilité, orthographe) sera prise en compte*	

I Partie cours [13/90]

1.1 Les FPGA SRAM sont composés de LUT (voir la figure 1.1.a) qui permettent d'implanter les fonctions combinatoires reconfigurables.

1.1.1 Précisez les significations des initiales dans les acronymes FPGA, SRAM, LUT [3]

FPGA : Field Programmable Gate Array

SRAM : Static Random Acces Memory

LUT : Look Up Table

1.1.2 Donnez la table de vérité de la fonction logique implantée dans la LUT de la figure 1.1.a et simplifiez la [2+4].

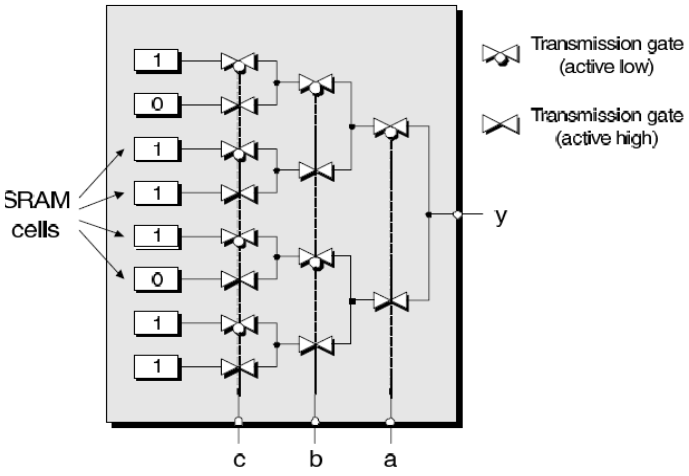


Figure 1.1.a

a	b	c	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Tableau de Karnaugh

c/ab	00	01	11	10
0	1	1	1	1
1	0	1	1	0

Y=b+/c

1.2 Indiquez lesquelles des caractéristiques ci-dessous sont relatives aux FPGA et lesquelles sont relatives aux CPLD [2+2] (**Attention -0,5 par réponses fausses**).

1.3 Je souligne les caractéristiques relatives aux FPGA

- 1 Peu de blocs logiques mais des blocs logiques avec un grand nombre d'entrées/sorties
- 2 Des interconnexions « omniprésentes » dans l'architecture
- 3 Plusieurs milliers de blocs logiques avec peu d'entrées/sorties
- 4 Un seul grand bloc d'interconnexions : peu flexibles
- 5 Souvent non volatiles
- 6 Performances élevées et reproductibles
- 7 Majoritairement volatiles
- 8 Temps dépendant du routage

II Partie exercices : [77/90]

2.1 PLD [12]

2.1.1 Indiquer directement sur le schéma ci-dessous quelles sont les interconnexions à programmer et les signaux à connecter aux entrées/sorties pour obtenir les fonctions suivantes [12] :

$w = ABC' + A'B'CD'$

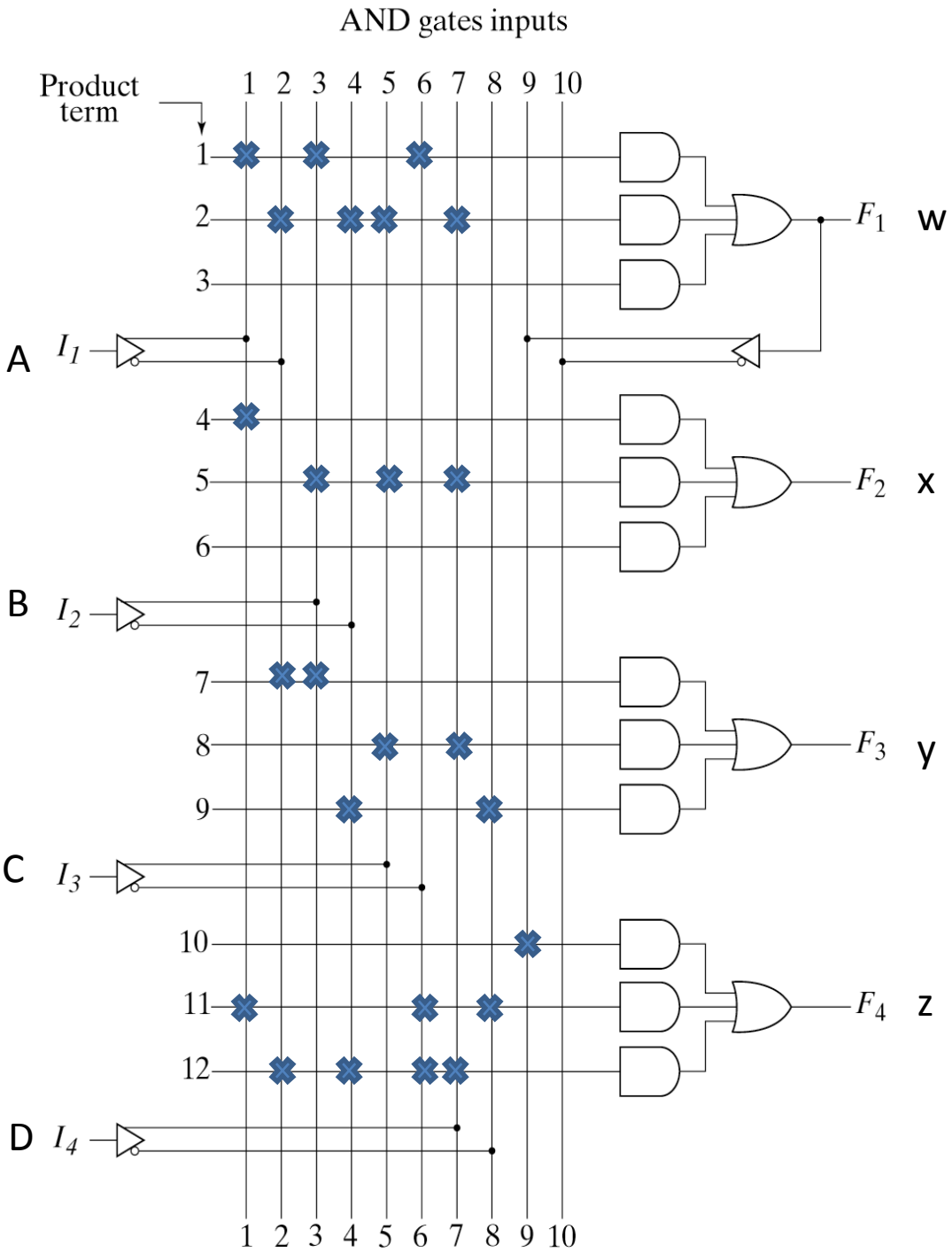
$x = A + BCD$

$y = A'B + CD + B'D'$

$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$

$= w + AC'D' + A'B'C'D$

Remarque : ‘ correspond au complément



2.2 Description VHDL [36]

L'objectif de cet exercice (et des questions suivantes) est de décrire le circuit représenté ci-dessous en VHDL : d'abord avec une description structurelle, puis avec une description RTL.

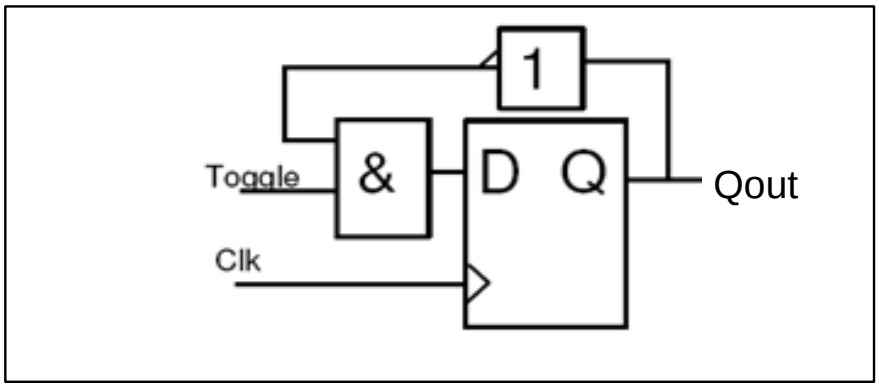


Figure 2.2.a : schéma du circuit toggle

La bascule D est une bascule Flip-Flop sensible sur front montant de l'horloge.

La porte « & » correspond à une porte ET et la porte « 1 » (avec le triangle en sortie) correspond à un inverseur.

Ces trois composants (entités et architectures RTL) ont été compilés dans la bibliothèque de travail WORK et leurs entités sont les suivantes :

entity FF is port(D, CLK : in std_logic ; Q : out std_logic); end entity;	entity AND is port(A, B : in std_logic ; S : out std_logic); end entity;	entity INV is port(A : in std_logic ; S : out std_logic); end entity;

2.1.1 Donnez en VHDL l'entité de ce circuit [3]

```
Library IEEE;
Use IEEE.std_logic_1164.all;

entity toggle_comp is
        port(   toggle, clk : in std_logic:
                Qout : out std_logic);
end entity;
```

2.1.2 Donnez en VHDL une description structurelle **(instanciant les 3 composants)** correspondant au schéma précédent [12]

```
architecture struct of toggle_comp is

signal d_int, qout_int, toggle_2 : std_logic;
begin

i1 : entity work.FF(bhv) port map (D=>d_int; CLK => clk, Q => qout);
i2 : entity work.AND(bhv) port map (A=> toggle_2, B => Toggle, S => d_int);
i3: entity work.INV(bhv) port map (A => qout_int, S => toggle_2);

Qout <= qout_int:
end architecture;
```

2.1.3 Donnez en VHDL une description RTL (cad **synthétisable**) du même circuit [6].

Architecture rtl of toggle_comp is

```
signal qout_int : std_logic;
begin

process(clk)
begin
If clk'event and clk='1' then
        qout_int <= Toggle and not(qout_int);
end if;
end process;

Qout <= qout_int;
end architecture;
```

2.1.4 Ecrire en VHDL un testbench permettant de simuler la description RTL précédente [9].

```
Library IEEE;
Use IEEE.std_logic_1164.all;

Entity tb_toggle is
End entity ;

Architecture bhv of tb_toggle is
Signal toggle, clk, qout : std_logic := '0';

Begin

Clk<=not(clk) after 10 ns;

Toggle <= '0', '1' after 40 ns

I1 : entity work.toggle_comp(rtl) port map (Toggle => toggle, clk => clk, Qout => qout);

End architecture;
```

2.1.5 Représenter sur un chronogramme les signaux d'entrées et de sorties (Clk, Toggle, Qout) obtenus lors de la simulation de votre testbench [6]

tant que toggle en entrée vaut 0 alors la sortie vaut 0
quand toggle vaut 1 alors à chaque front montant qout change d'état : à 50 ns (front montant de l'horloge) qout prend la valeur 1…

Conseil : vérifiez que les timings sur vos chronogrammes correspondent à ceux de votre testbench

2.2 Etude d'un Registre à décalage à rebouclage linéaire (LFSR : Linear Feedback Shift Register) [29]

Le code ci-dessous implémente un LFSR.

2.2.1 Représentez le schéma à base de portes logiques (bascules, xor, mux…) de ce circuit [12].

Un registre à décalage avec un rebouclage avec des xor + des mux pour sélectionner entre Din ou la sortie de la précédente cellule

2.2.2 Le RESET est-il synchrone ou asynchrone [2] ? Est-il actif au niveau haut ou au niveau bas [1] ? Justifiez.

Reset asynchrone et actif au niveau haut car dans la liste de sensibilité et avant la condition clk=1 and clk'event dans le if

2.2.3 Que se passerait-il (comment évaluerait la sortie PRN) si toutes les bascules du circuit étaient initialisées à '0' au démarrage ? [2]

Alors le LFSR resterait à 0 tant qu'aucun 1 n'est chargé (car 0 xor 0 = 0)

2.2.4 Quel est le rôle de l'entrée LD ? [2]

LD permet de chargé la valeur sur l'entrée Din dans les bascules du LFSR

2.2.5 Quel est le rôle de l'entrée EN ? [2]

EN à 1' autorise le LFSR à évoluer sur chaque front de l'horloge

2.2.6 Suite à une initialisation du registre à la valeur x « 0001 » (soit en binaire « 0000_0000_000_0001 »), représentez en binaire l'évolution de la sortie PRN pendant 16 coups d'horloge (avec RESET='0', LD='0, EN='1') [4]

```
t_prn(0) <= t_prn(15) xor t_prn(4) xor t_prn(2) xor t_prn(1);
t_prn(1 to 15) <= t_prn(0 to 14);
```

XOR:
1000_0000_ 0001 0110

0000_0000_0000_0001 = 1

0000_0000_0000_0010 = 2

0000_0000_0000_0101 = 5

0000_0000_0000_1011 = 11

0000_0000_0001_0111 = 23

0000_0000_0010_1111 = 47

0000_0000_0101_1110 = 94

0000_0000_1011_1101 = 189

2.2.7 Convertir les 8 premières valeurs binaires précédentes en nombre décimal [4]

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

entity LFSR16 is
port(
     CLK : in std_logic;
     RESET : in std_logic;
     LD : in std_logic;
     EN : in std_logic;
```

```
      DIN : in std_logic_vector (0 to 15);
      PRN : out std_logic_vector (0 to 15);
      ZERO_D : out std_logic);
end LFSR16;

architecture RTL of LFSR16 is
signal t_prn : std_logic_vector(0 to 15);
begin

PRN <= t_prn;

ZERO_D <= '1' when (t_pr = x"0000") else '0';

process(CLK,RESET)
begin
if RESET='1' then
      t_prn <= x"0001";
elsif CLK'event and CLK='1' then
      if (LD = '1') then
            t_prn <= DIN;
      elsif (EN = '1') then
            t_prn(0) <= t_prn(15) xor t_prn(4) xor t_prn(2) xor t_prn(1);
            t_prn(1 to 15) <= t_prn(0 to 14);
      end if;
end if;
end process;

end RTL;
```