0801 Simulation

Considérez le code VHDL suivant pour un module combinatoire et son banc de test associé.

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

entity module3 is
   port (
      A, B, C : in std_logic;
      F, G : out std_logic
   );
end module3;

architecture arch2 of module3 is
signal S1, S2, S3 : std_logic := '0';
begin
   S3 <= S1 or B;
   G <= S3 or not(A);
   process (A, B, C)
   variable V : std_logic := '0';
   begin
      S1 <= A and C;
      V := (not A) and (not B);
      S2 <= V or C;
      F <= B or C;
   end process;
end arch2;
```

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

entity module3_TB is
end module3_TB;

architecture arch of module3_TB is
signal A, B, C, F, G : std_logic;
begin

   UUT : entity module3(arch2)
         port map (A, B, C, F, G);

   A <= '1' after 0 ns;
   B <= '0' after 0 ns;
   C <= '0' after 0 ns, '1' after 10 ns;

end arch;
```

Donnez la valeur de tous les ports et signaux internes du module combinatoire en fonction du temps, en tenant compte des délais deltas.

temps	delta	A	B	C	S1	S2	S3	F	G
0 ns	0								
0 ns	1								
0 ns	2								
0 ns	3								
0 ns	4								
0 ns									
10 ns	0								
10 ns	1								
10 ns	2								
10 ns	3								
10 ns									

0802 signal et variable

Considérez le problème de la fonction majorité, pour lequel on veut établir si une majorité de bits d'un vecteur ont une valeur de '1'. Expliquez en vos propres mots pourquoi le code suivant ne fonctionne pas.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity vote is
    generic (W : positive := 4);
    port (
        lesvotes: in std_logic_vector(W - 1 downto 0);
        approbation : out std_logic
    );
end vote;

architecture comportementale1signal of vote is
   signal compte : integer;
begin
    process(lesvotes)
    begin
        compte <= 0;
        for k in 0 to lesvotes'length - 1 loop
            if lesvotes(k) = '1' then
                compte <= compte + 1;
            end if;
        end loop;
        if compte > lesvotes'length / 2 then
            approbation <= '1';
        else
            approbation <= '0';
        end if;
    end process;
end comportementale_signal_1;
```

0803

Considérez le code VHDL suivant. Écrivez un modèle synthétisable équivalent qui n'utilise pas de signal.

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

entity cctsequentielex1 is
    port (
        reset : in STD_LOGIC;
        CLK : in STD_LOGIC;
        X : in STD_LOGIC;
        Z : out STD_LOGIC
    );
end cctsequentielex1;
```

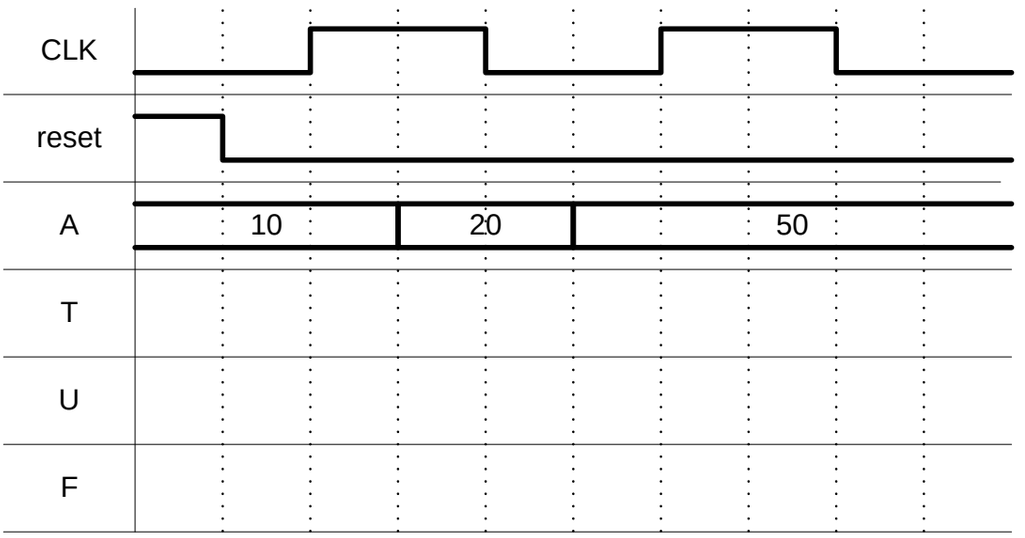
```
architecture arch1 of cctsequentielex1 is
   signal A : STD_LOGIC;
   signal B : STD_LOGIC;
begin
   process(CLK, reset) is
    begin
        if (reset = '0') then
           A <= '0';
           B <= '0';
        elsif (rising_edge(CLK)) then
           A <= A xor B;
           B <= x or not(B);
        end if;
    end process;

    z <= not(A or B);

end arch1;
```

0804

Considérez l'extrait de code VHDL suivant et les valeurs des signaux CLK, reset et A montrées sur le chronogramme.

Complétez le chronogramme pour les signaux et variables T, U, V et F.

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

entity VHDLEstMonAmi2 is
   port (
      clk, reset : in std_logic;
      A : in integer;
      F : out integer
   );
end VHDLEstMonAmi2;

architecture jaimeVHDL of VHDLEstMonAmi2 is

signal T, U : integer;

begin

process (clk, reset)
   variable V : integer := 10;
   begin
      if reset = '1' then
         T <= 0;
         U <= 0;
      elsif rising_edge(clk) then
         T <= T + V;
         V := V + 5;
         U <= T + V;
      end if;
   end process;

   process (A)
   begin
      F <= A + T + U;
   end process;

end jaimeVHDL;
```



0805 Synthèse

Consultez la norme IEEE 1076.6. Selon cette norme, est-il possible de modéliser, en vue de la synthèse, une bascule avec un énoncé wait ? Qu'en dit la documentation de Xilinx ?

0806 Synthèse

En justifiant clairement vos réponses, indiquez pour chacun des énoncés VHDL suivants si l'opération est synthétisable ou non. On supposera dans tous les cas que les signaux `sig_1`, `sig_2` et `sig_3` sont de type `unsigned` et qu'ils sont correctement dimensionnés.

```
sig_3 <= sig_1 + sig_2;
sig_3 <= sig_2 / 6;
sig_3 <= sig_2 mod 16;
sig_3 <= 6*(sig_1 - sig_2/16);
sig_3 <= sig_2 ** 4;
sig_3 <= sig_1 / sig_2;
sig_3 <= sig_1 * sig_2;
```

```
sig_3 <= sig_2 rem 3;
sig_3 <= 2*sig_2 + 6*sig_1;
sig_3 <= 2.0*sig_2 + 6.0*sig_1;
sig_3 <= shift_left(sig_2, sig_1);
sig_3 <= shift_right(sig_2, 10.0);
sig_3 <= sig_2 ** 2.0;
sig_3 <= shift_left(sig_2, to_integer( sig_1 ) );
sig_3 <= sin( real( to_integer( sig_2 ) ) );
sig_3 <= to_unsigned( integer( sin( 3.14 ) * 2.0**8 ), 8 );
```

0807 Implémentation

Considérez le modèle simplifié de FPGA donné verso et les délais de routage suivants.

Trajet	Délai
Entre deux CLB adjacents	1 ns
Entre un CLB et un bloc d'interconnexions	1.5 ns
Entre deux blocs d'interconnexions	2 ns
Entre un IOB et un bloc d'interconnexions	0.5 ns

Considérez le produit suivant résultant d'une étape préalable d'association. A, B, C et D sont des entrées du système. X, Y et Z sont des sorties. Chaque fonction f() doit être implémentée dans un CLB différent.

CLB1 = f(A, B)	CLB5 = f(CLB2, CLB6)	X = CLB8
CLB2 = f(C, D)	CLB6 = f(A, CLB4)	Y = CLB7
CLB3 = f(A, C, D)	CLB7 = f(CLB6, CLB2)	Z = CLB6
CLB4 = f(CLB1, C, D)	CLB8 = f(CLB1, CLB2)	

a. Proposez un placement sur le modèle de FPGA simplifié.

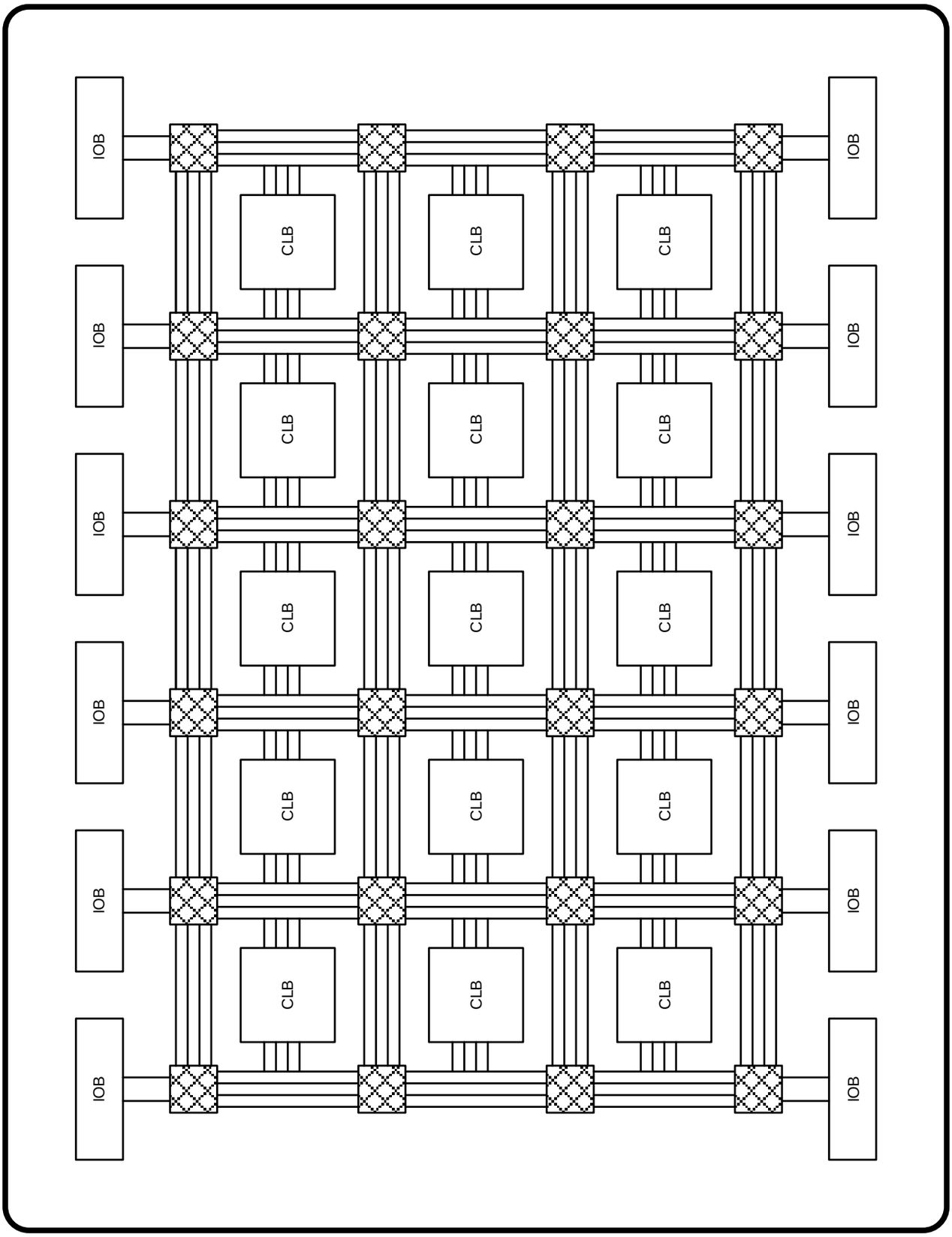
b. Proposez un routage qui correspond à votre placement

c. Mesurez le délai total et le délai maximum de votre placement et routage

d. Proposez la meilleure solution de placement-routage possible

   i. selon la métrique du délai total

   ii. selon la métrique du délai le plus long


IOB
CLB

Solutions

0801 Simulation

Time	Delta	A	B	C	S1	S2	S3	F	G
0 ps	0	U	U	U	0	0	0	U	U
0 ps	1	1	0	0	U	U	U	U	U
0 ps	2	1	0	0	0	0	U	0	U
0 ps	3	1	0	0	0	0	0	0	U
0 ps	4	1	0	0	0	0	0	0	0
10000 ps	0	1	0	1	0	0	0	0	0
10000 ps	1	1	0	1	1	1	0	1	0
10000 ps	2	1	0	1	1	1	1	1	0
10000 ps	3	1	0	1	1	1	1	1	1

0802

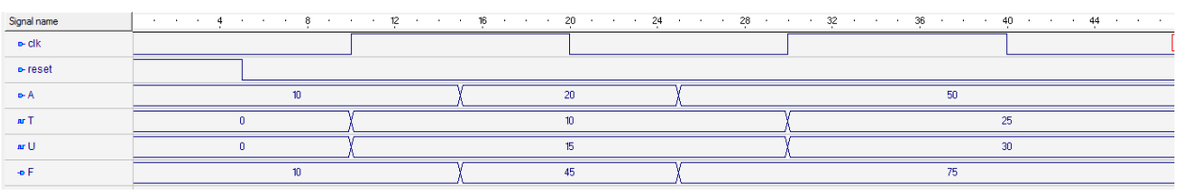
Le compteur compte est un signal.

Quand le processus est lancé par une modification de l'entrée lesvotes, la réinitialisation de compte est prévue pour être exécutée. Ensuite, la boucle s'exécute et l'incrémentation de compte est aussi prévue pour être exécutée, autant de fois qu'il y a de '1' dans le vecteur lesvotes. Le problème vient du fait qu'une seule assignation à compte aura lieu. S'il y a au moins un bit à '1' dans, alors le signal compte sera incrémenté d'une unité par rapport à sa valeur avant que le processus soit appelé. Sinon, le signal compte sera remis à 0. La valeur de compte est persistante entre deux exécutions du processus. Pour cet arrangement de code, il faudrait que compte soit une variable, pas un signal.

0803

```
architecture arch1_nosig of cctsequentielex1 is

begin

    process(CLK, reset) is
    variable A, B : std_logic;
    begin
        if (reset = '0') then
           A := '0';
           B := '0';
        elsif (rising_edge(CLK)) then
           A := A xor B;
           B := x or not(B);
        end if;
       z <= not(A or B);
    end process;

end arch1_nosig;
```

0804



0805

Oui, voir le paragraphe 6.1.3.2 de IEEE 1076.6-2004. Mais ce n'est pas recommandé par Xilinx, voir le chapitre 4 du *Vivado Design Suite User Guide – Synthesis* – UG901 (v2020.2) 28 janvier 2021, et le chapitre 5 du même manuel, pp. 99+

0806

a) `sig_3 <= sig_1 + sig_2;`

Énoncé synthétisable car il s'agit de l'addition de deux signaux de type unsigned.

b) `sig_3 <= sig_2 / 6;`

Énoncé non synthétisable car il s'agit de la division d'un signal de type unsigned par une constante qui n'est pas une puissance de 2.

c) `sig_3 <= sig_2 mod 16;`

Énoncé synthétisable car il s'agit d'une opération modulo puissance de 2 effectuée sur un signal de type unsigned.

d) `sig_3 <= 6*(sig_1 – sig_2/16);`

Énoncé synthétisable. La division est opérée sur une constante puissance de deux. La soustraction est effectuée sur des signaux de type unsigned. La multiplication par une constante est également supportée.

e) `sig_3 <= sig_2 ** 4;`

Énoncé non synthétisable car la puissance (même pas une puissance de 2) n'est pas synthétisable.

f) `sig_3 <= sig_1 / sig_2;`

Énoncé non synthétisable car il s'agit de la division de deux signaux de type unsigned.

g) `sig_3 <= sig_1 * sig_2;`

Énoncé synthétisable car il s'agit de la multiplication de deux signaux de type unsigned.

h) `sig_3 <= sig_2 rem 3;`

Énoncé non synthétisable car il s'agit d'une operation « reste » d'un signal de type unsigned sur une constante qui n'est pas une puissance de 2.

i) `sig_3 <= 2*sig_2 + 6*sig_1;`

Énoncé synthétisable car il s'agit d'opérations de multiplications par une constante et d'addition effectuées sur des signaux de type unsigned.

j) `sig_3 <= 2.0*sig_2 + 6.0*sig_1;`

Énoncé n'est pas synthétisable. Bien que cet exemple ressemble à l'énoncé précédent, ici les constantes sont de type real et ne peuvent être synthétisés.

k) `sig_3 <= shift_left(sig_2, sig_1);`

Énoncé non synthétisable. Le second argument doit être de type integer.

l) `sig_3 <= shift_right(sig_2, 10.0);`

Énoncé non synthétisable. Le second argument doit être de type integer.

m) `sig_3 <= sig_2 ** 2.0;`

Énoncé non synthétisable car la puissance (même pas une puissance de 2) n'est pas synthétisable.

n) `sig_3 <= shift_left(sig_2, to_integer( sig_1 ) );`

Énoncé synthétisable car le second paramètre est bien de type integer.

o) `sig_3 <= sin( real( to_integer( sig_2 ) ) );`

Énoncé non synthétisable car la fonction sinus ne l'est pas.

p) `sig_3 <= to_unsigned( integer( sin( 3.14 ) * 2.0**8 ), 8 );`

Énoncé synthétisable. Nous avons ici l'initialisation d'un signal à une constante dont la valeur est définie par la fonction sinus.

0807

On montre ici une disposition possible des 8 blocs.

Les deux chemins les plus longs sont indiqués :

de l'entrée D au bloc #3, délai de 8 ns

du bloc #1 au bloc #8, délai de 9 ns

Ce n'est pas nécessairement la meilleure solution.