Travail écrit 2: Système Numérique 2

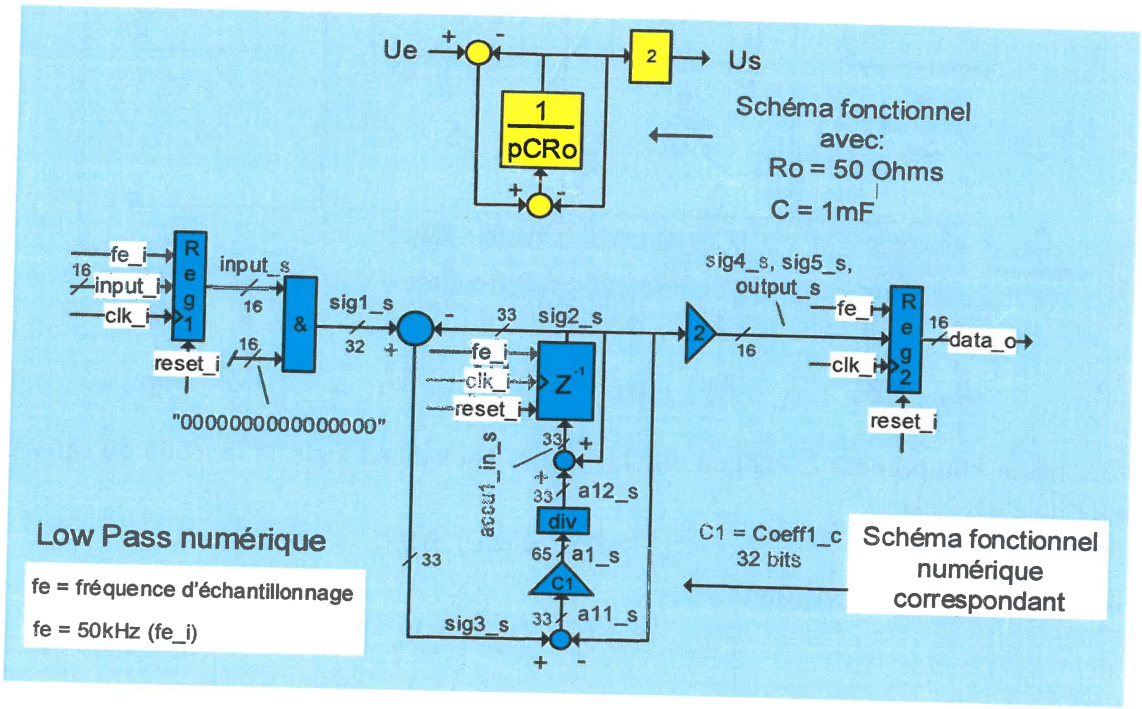
2017/2018

Filière : Télécommunication
Classe : T-2a, T-2d
Date : 5 juin 2018, 15h00 à 16h35
Professeur : Fabio Cunha

Nom et prénom : Zambon Yanick **Points :** 12.5/16 **Note :** 5.5

Problème 1: Filtre à intégrateur (3 pts) 2.5

Soit le filtre numérique ci-dessous :



a) Dessinez le schéma passif (résistance, capacité, inductance) analogique équivalent de ce filtre numérique, en illustrant les différentes étapes.

1/1 pt

b) A quoi sert ce facteur de 2 à la sortie du filtre ?

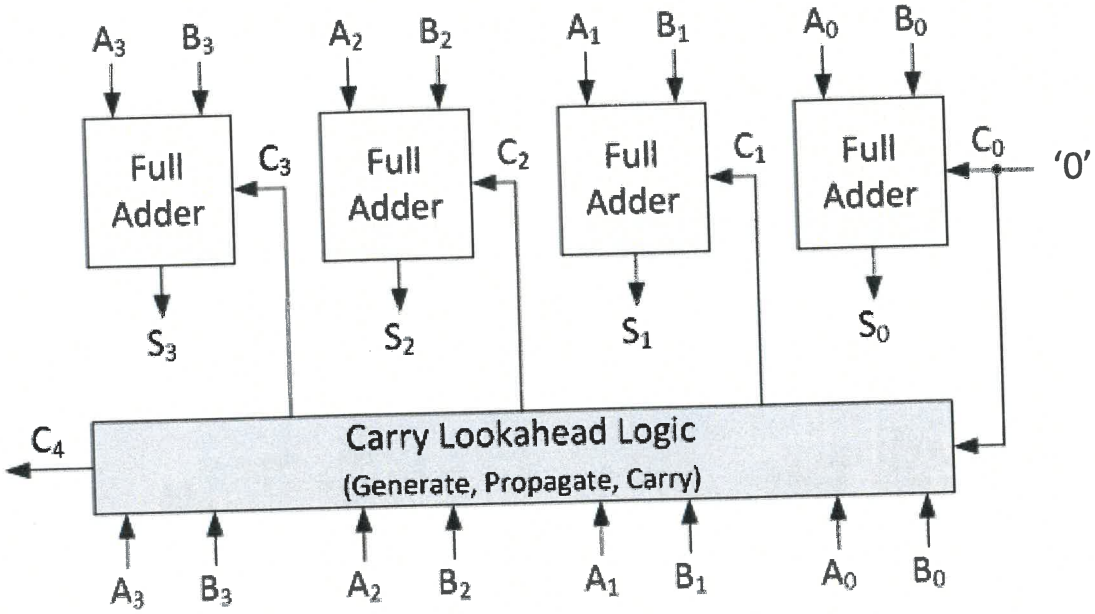
0.5/ 1 pt

c) Calculez le coefficient C1 = Coeff1_c et commentez les différentes opérations.

1/1 pt

Problème 2: Additionneur « Carry-Lookahead » (10 pts) 7.25

Afin de réduire le délai dû à la propagation du carry dans un additionneur « ripple-carry adder, il est possible d'évaluer rapidement pour chaque étage si le carry de l'étage précédemment vaut '0' ou '1'. Si une évaluation correcte peut être faite dans un temps relativement court, la performance de l'additionneur complet sera améliorée. Il s'agit en fait d'un additionneur de structure « Carry-Lookahead » :



Pour mieux comprendre la logique du block Carry Lookahead logic, la fonction du carry-out pour un étage i est :

$$c_{i+1} = a_i b_i + a_i c_i + b_i c_i$$

Si on factorise cette expression :

$$c_{i+1} = a_i b_i + (a_i + b_i) c_i$$

On peut réécrire l'expression sous cette forme :

$$c_{i+1} = g_i + p_i c_i$$

Où le terme generate (g) :

$$g_i = a_i b_i$$

Et le terme propagate (p) :

$$p_i = a_i + b_i$$

La fonction g_i est égale à '1' quand les deux entrées sont égales à '1', sans se préoccuper du carry d'entrée, d'où le nom generate pour la fonction puisqu'il génère le carry-out. Concernant le terme p_i est égale à '1' lorsqu'une des deux entrées vaut '1'. Toutefois, le carry est généré si le carry-in vaut '1', d'où le terme propagate puisque l'on propage le carry-in.

1) Vous devez réaliser le code VHDL du carry-lookahead adder en le rendant le plus paramétrable possible :

4.75/ **6 pts**

- en utilisant uniquement des mappings (3 pts)
- avec la fonction for...generate (2 pts)
- avec un paramètre générique (1 pt)

Notez que la fonction for...generate peut-être utiliser pour générer plusieurs cellules logiques si elles sont répétitives, par exemple :

```
GEN_CLA : for jj in 0 to g_WIDTH-1 generate
    G(jj)   <= a_in(jj) xor b_in(jj);
end generate GEN_CLA;
```

2) Réalisez le testbench de l'additionneur afin de vérifier le bon fonctionnement du circuit. Le testbench fonctionnera de la façon suivante :

- Stimulis et les valeurs de référence seront contenus dans deux fichiers différents
- Testbench lira les stimulis, exécutera l'addition et vérifiera si le calcul est correct
- Affichera dans la console si le test est réussi ou incorrect

2.5/ **4 pts**

Notez que le testbench doit contenir au moins 3 additions.

Problème 3: Arbiter (3 pts) 2.75

Soit la structure de l'arbiter que vous avez réalisée en TP.

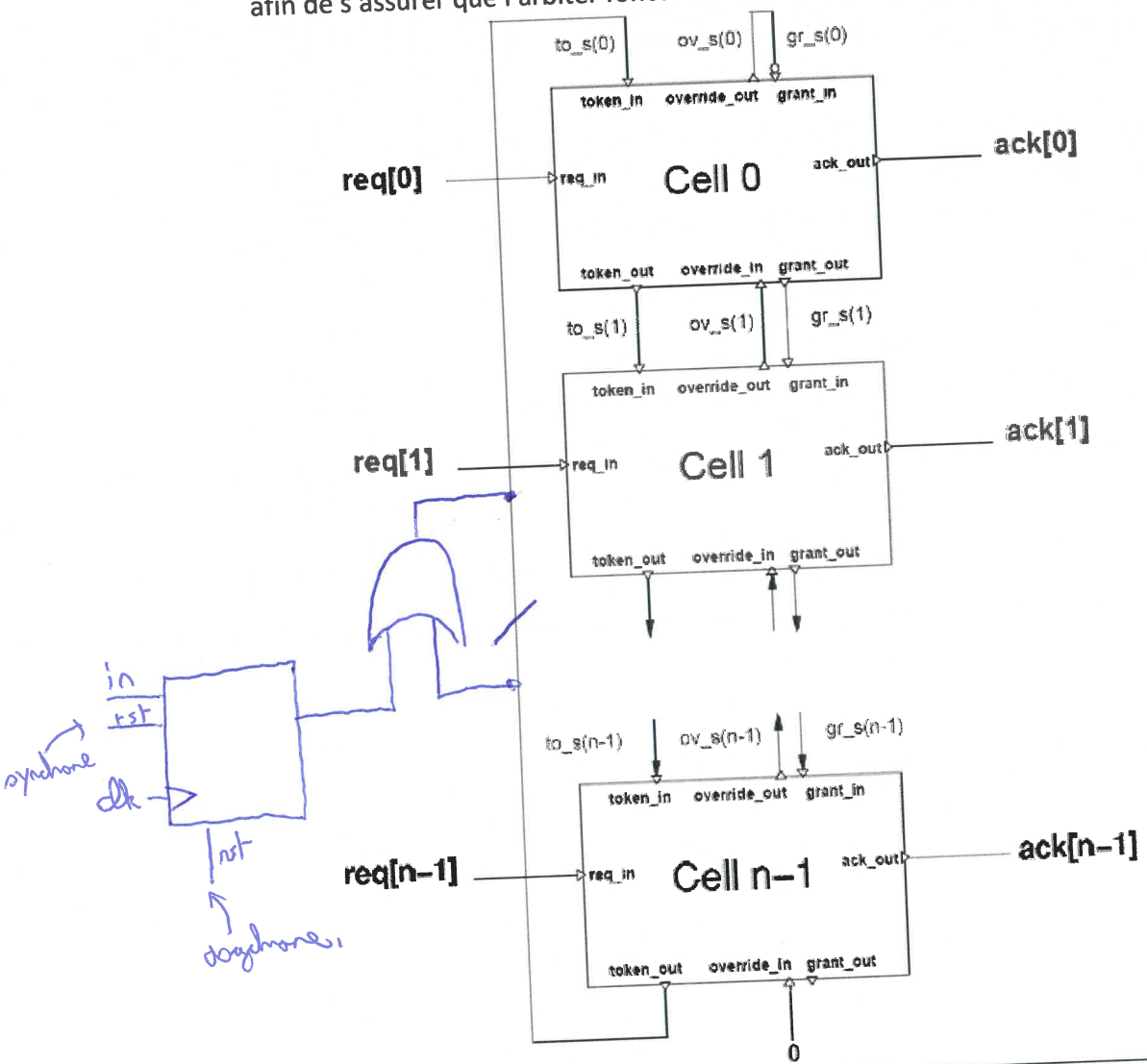
1) Expliquez l'utilité et le fonctionnement de ce circuit, ainsi que les 3 règles à suivre pour assurer le bon fonctionnement du circuit **2 pts** 1.75/

- Cela simule un type de réseau nommé le "token ring" qui n'est plus qu'utilisé dans des pays reculés d'Afrique centrale. Il permet à plusieurs composants de "parler" à tour de rôle.

Règles: 1) Si un seul veut "parler", il peut immédiatement
2) Si plusieurs veulent "parler", on les hiérarchise dans un ordre précis (ex: id le plus bas)
3) Si plusieurs veulent "parler" en continu, ils opposent comme des gentlemen et alternent.

token -1/4

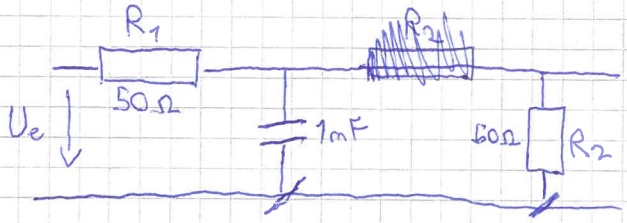
2) Ce schéma est incomplet et n'assure pas le bon fonctionnement de l'arbiter, notamment la règle 3. Indiquez directement sur le schéma le changement à apporter afin de s'assurer que l'arbiter fonctionnement correctement. **1 pt** 1/



TE04 Sys num 2

Exercice 1

a)



✓

b) Etant donné ma connaissance nulle en électronique et mon incapacité totale à comprendre les gribouillis de la forme de ceux que l'on trouve par exemple au point a), je vais suivre les indications de monsieur Cunha, qui affirme avec force qu'un courant continue + $\frac{1}{\infty} = 1$, il ne reste donc que deux rectangles négatifs qui sont hélas dénués de tension (j'ai appris jusqu'à $U = R\,\Sigma$, et oui.) et ça, c'est pas bien. Alors pour compenser on fait $\cdot 2$. Voilà voilà.

-1/2

c) Opération 1 : je regarde mon formulaire avec espoir.
Opération 2 : je trouve que pour calculer mon coefficient, il faut calculer son "temps" τ_1
Opération 3 : je calcule avec ma calculatrice : $\tau_1 = CR_0 = 0.05$. Je suis content.
Opération 4 : je vois que $C_1 = \frac{T_e}{\tau_1}$ or $T_e = \frac{1}{f_e} = \frac{1}{50kHz} = 2 \cdot 10^{-5}$ s.
Opération 5 : j'applique : $\frac{2.10^{-5}}{0.05} = 4 \cdot 10^{-4} = C_1$
Opération 6 : Au cas-où, j'écris également sa version sous virgule en le multipliant par 2^{32} : 1'717'986 ✓
Opération 7 : Je trouve la page car l'exercice est terminé.

Exercice 2

1)
```
1:library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all
entity CL is
        generic ( LEN : integer := 8);
        Port ( A ;B ;in std_logic_vector (LEN-1 downto 0);
          S_out, C_out: out std_logic_vector (LEN-1 downto 0);
              C_in: in  std_logic ));
end CL;
```

C_out std_logic

-1/4

architecture behavioral of CL is

signal [redacted], P_o, S_o, G_o: std_logic_vector (LEN-1 downto 0);
C_o: std_logic_vector (LEN downto 0);

component fulladder is
port (A_i : in std_logic;
B_i : in std_logic;
C_i : in std_logic;
S_i : out std_logic);
end component;

G_o(I) <= A(I) and B(I);
P_o(I) <= A(I) or B(I);

begin

Q1: for I in 0 to LEN-1 generate

Q2: if I=0 generate — port map — C_o(0)

adder1st : fulladder (A(0), B(0), [redacted], S_o(0));

end generate;

Q3: if I = LEN-1 generate — port map

adderlast : fulladder (A(LEN-1), B(LEN-1), [redacted] C

[redacted] C_out (LEN-1 downto 0) <= C_o (LEN downto 1); -1/4

C_o(0) <= C_in; S_out <= S_o;

Q1: for I in 0 to LEN-1 generate

G_o(I) <= A(I) and B(I);

P_o(I) <= A(I) or B(I);

C_o(I+1) <= G_o(I) or (P_o(I) and C_o(I));

adder : fulladder

port map (A_i => A(I), -1/4
B_i => B(I),
C_i => C_o(I),
S_i => S_o(I));
C_o => open -1/4

end generate;

end architecture;

Daniela Zamboni

12-a

2) library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

use IEEE.std_logic_textio.all;
use STD.textio.all;

entity tb is
end tb;

architecture behavioral of tb is
component add is
Generic (Len: integer:=8);
Port (A, B: in std_logic_vector(LEN-1 downto 0);
C_in : in std_logic;
C_out, S_out: out std_logic_vector (LEN-1 downto 0));
end component;
constant L : integer := 8;
Signal A_s, B_s, C_out_s, S_out_s : std_logic_vector (L-1 downto 0);
Signal C_in_s: std_logic;
file F_in: text;
file F_out: text;

begin
file_open (F_in, "in.txt", read_mode);
file_open (F_out, "out.txt", write_mode);

adder: add
Generic map (LEN => L);
Port map (A => A_s,
B => B_s,
C_in => C_in_s,
C_out => C_out_s,
S_out => S_out_s);

lettura: process
variable A_v, B_v : integer;
variable L_in, L_out: line;
variable S_v: integer; variable C_out_v : integer;

begin

```
while not endfile (F_in) loop
    readline (F_in, L_in);
    read (L_in, A_v);
    readline (F_in, L_in);
    read (L_in, B_v);
    A_s <= std_logic_vector (to_signed (A_v, L));
    B_s <= std_logic_vector (signed (B_v, L));
    C_in_s <= '0';
    wait 100 ns;
    S_v <= to_integer (signed (S_out_s));
    C_out_v <= to_integer (signed (C_out_s));
    WriteLine (L_out, S_v);
    write (F_out, L_out);
    WriteLine (L_out, C_out_v);
    write (F_out, L_out);
    assert (A_v + B_v + (2**L) * C_out_v = S_v) report "erreur";
end loop;
end process;
end architecture;
```

✓

pas exactement ce qui est demandé (-1) mais (ok)

pas compris (-1/2)