Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Filière Télécommunications **Systèmes Numériques 1**

Test de contrôle continu (21 pts)	Note : 3,7
Date : 22.11.2018	Nom : Roten Marc

10,625

Matériel autorisé :

- feuilles VHDL (référence et bibliothèque IEEE)

Réponses et brouillons sur ces feuilles, SVP.

1. Développement d'un composant (12 pts) 6

Développer un composant qui calcule le nombre de bits à 1 d'un std_logic_vector(31 downto 0)

Les contraintes de développement sont les suivantes :

- Le signal d'entrée std_logic_vector est combinatoire
- Le calcul du nombre de bits à 1 doit être un circuit séquentiel
- Ce composant est un worker et le traitement doit pouvoir être interrompu à tout instant.

a) Proposer une définition d'entité et expliquer chaque port

b) Dessiner un schéma bloc de ce composant en indiquant pour chaque bloc les signaux d'entrée et de sortie ; de plus, expliquer les fonctions liées à chaque bloc

c) Ecrire le code VHDL complet de ce composant (1 seul composant n'incluant aucune instanciation d'autres composants) en utilisant les bonnes pratiques.

Code

```
Library IEEE;
use ieee_std_logic_1164.all;   ← numeric_std
entity Ex1 is
    port( clk, again, rst: in std_logic;
          sign_in : out std_logic_vector(31 downto 0);
          numb    : out std_logic_vector(5 downto 0);
end Ex1;
architecture behavioral of Ex1 is
    signal countBit1: std_logic_vector(5 downto 0); signal fin_compteur: std_logic;
    types etats is (IDLE, COUNT, );
    signal P(etat_present, etat_futur) : etats ; signal countstate :std_logic_vector(5 downto 0);
begin
numb <= countBit1 ;
again <= fin_compteur ;
countbit: process (clk, countBit1, sign_in, etat_present)
    for (de 0 à 31) ⇒
        if rising_edge(clk) and sign_in(i) and etat_present = Count then
        countBit1 <= countBit1 + 1;
        else countBit <= (others => 0);
        end if;
    end for
end process count_bit;

Machine Etat: process (clk, rst, countstate, etat_present)
    if rst = '1' then
        etat_present <= idle ; numb <= (others => '0');
    elsif (rising_edge(clk) or again = '1') then
        countstate <= countstate + 1;
        if countstate = "100000" then
            etat_present <= IDLE; fin_compteur <= 1
        end if;
    end if;
end process Machine Etat;

Rst: process (fin_compteur, clk)
    if fin_compteur and rising_edge(clk) then
        fin_compteur <= 0;
end process Rst;

end behavioral;
```

done −0,25

rst −0,25

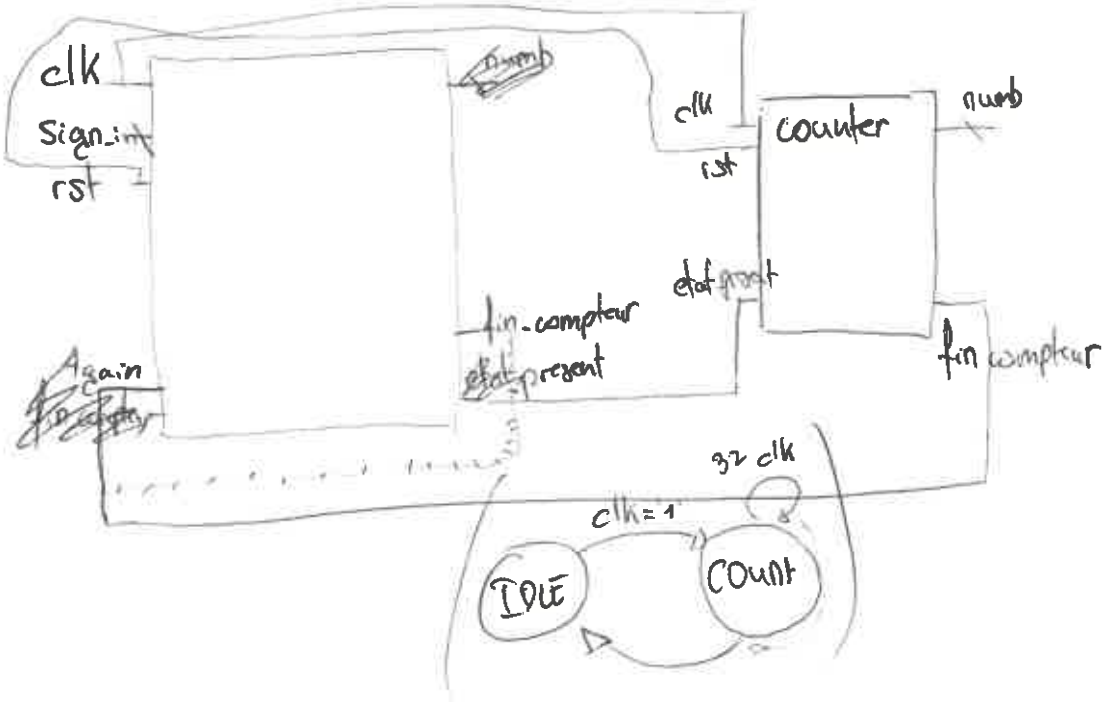
Manque compteur i −1,5

*if etat_present = IDLE then
etat present <= Count,
−0,5

Manque contrôle interm. fin et fin correcte
−1,25

Manque Val_Reg −2

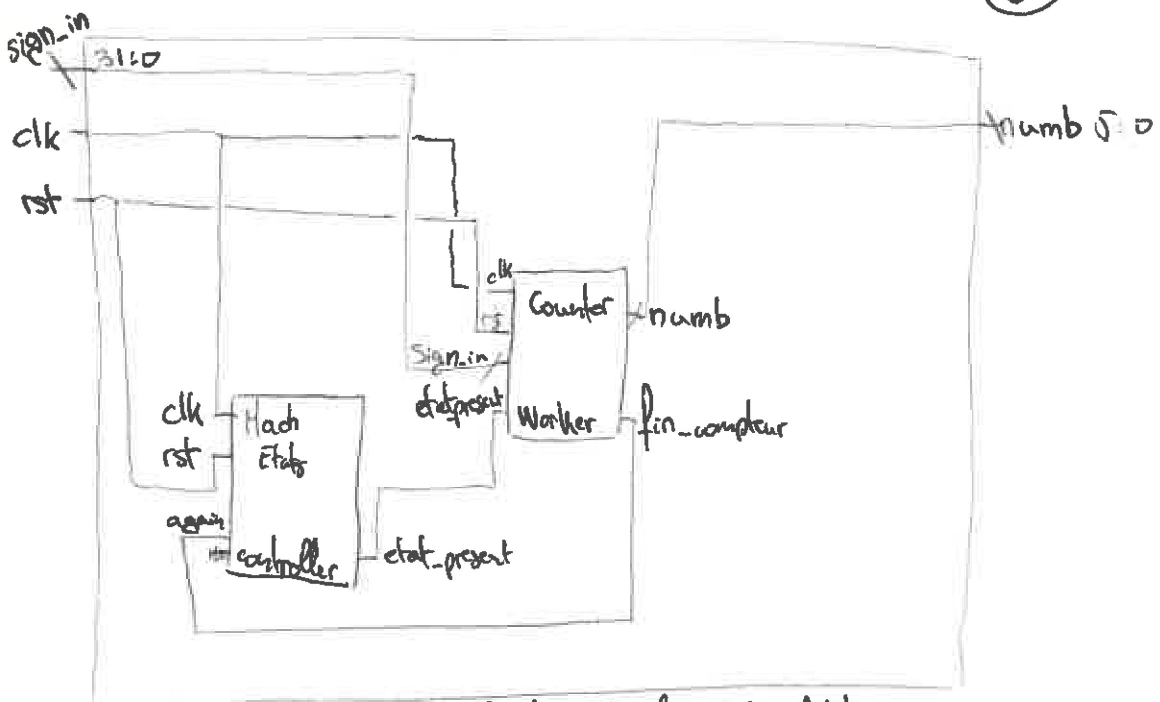
b) schema bloc



a) à chaque coup de clock, on prend le signal d'entrée, et on passe dans l'état COUNT.
Again est une entrée qui dit à notre composant de recommencer, et donc de repasser à l'état IDLE, et donc de pouvoir recompter une nouvelle fois.
rst permet de repartir à IDLE
fin_compteur est un std_logic qui indique dès qu'il a passé les 32 bits du vecteur
fin_compteur lié à Again

Au premier abord mal lu la consigne

(b)



sign_in : Bus 32 bit où il faut compter le nombre de bits à 1

clk : horloge de notre système utile pour notre counter et notre machine d'état

rst : reset synchrone utile pour nos composants Internes

numb : bus qui nous sort le nombre de bits à 1 en binaire

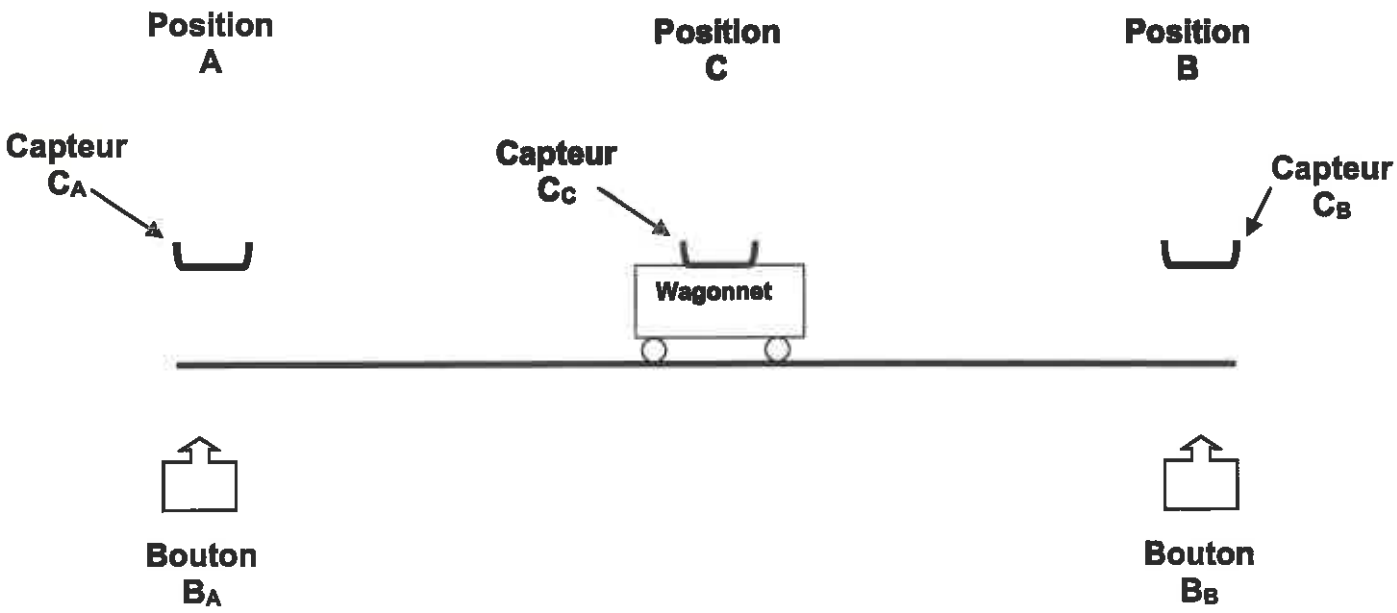
amélioration possible

~~pour le worker, mettre un signal qui sert de controller : comme un STOP qui servirait asynchrone surtout pas, on oublie~~

2. Conception d'un système synchrone (6 pts) 3,125

Un wagonnet motorisé se déplace sur un rail entre les positions A, C et B (voir figure ci-dessous). Les capteurs C_A, C_C, et C_B permettent de localiser le wagonnet lors de ses déplacements.
Les boutons B_A et B_B permettent d'appeler le wagonnet pour faire un transport de C à A et retour, respectivement de C à B et retour.
Le système séquentiel pilote le moteur du wagonnet en fonction des valeurs des boutons et des capteurs.
Les règles de fonctionnement sont :

- Lorsque le wagonnet est arrêté en position C :
  - si les 2 boutons sont sur OFF (état 0), le wagonnet reste en C ; ✓
  - si un seul bouton est sur ON (état 1), le wagonnet se déplace jusqu'à la position du bouton correspondant et s'arrête;
  - si les 2 boutons sont sur ON, alors le système enverra le wagonnet dans la direction inverse de la dernière position atteinte (A ou B).
- Lorsque le wagonnet est arrêté en position A ou B et que le bouton correspondant passe à OFF, alors le wagonnet retourne à la position C quelle que soit la valeur de l'autre bouton.
- Lorsque le wagonnet se déplace, les boutons n'ont pas d'effet sur le déplacement, le contrôle se faisant uniquement avec les valeurs des capteurs.



a) Définir clairement toutes les variables d'entrée et de sortie pas encore parfaitement spécifiées dans la donnée.

b) Etablir un diagramme d'états (sans états parasites) selon une machine de Moore (pas besoin de coder les états).

Ne dessiner pas les transitions impossibles dues à des erreurs des capteurs !

c) Expliquer verbalement sous forme d'une table à quelles situations correspond chaque état et indiquer la valeur de la/des variable(s) de sortie pour chaque état.

d) Si le diagramme d'états est établi selon une machine de Mealy, pouvons-nous diminuer le nombre d'états par rapport à la machine de Moore? Si oui, lesquels n'existent plus. Justifier votre réponse.

a) B_A, B_B : in std_logic

C_A, C_B, C_C : out std_logic

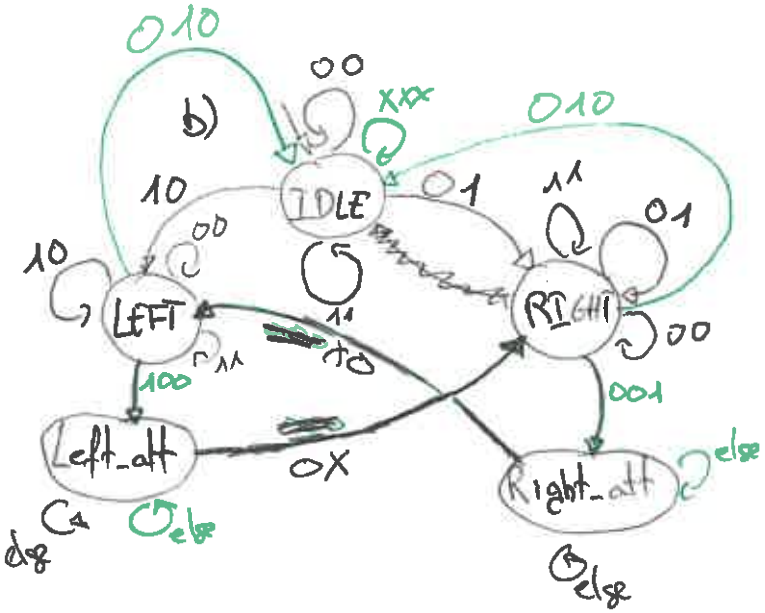
X-1

Brouillon

B_A	B_B	state
0	0	IDLE
0	1	RIGHT
1	0	LEFT
1	1	IDLE

Moteur -0,25

b)



$B_A B_B$

$C_A C_B C_C$

X-1,5

RST -0,125

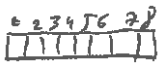
Etat	Explication	Sorties
IDLE	Etat de départ de notre système, on attend les instructions des boutons B_A et B_B	
RIGHT	Bouton B_B pressé. le Wagonnet continue à avancer vers la droite jusqu'à la détection par C_B	
LEFT	Bouton B_A pressé. le wagonnet continue à avancer vers la gauche jusqu'à détection par C_A. Lors de la détection par C_A, dès que "0X" en entrée pour "$B_A B_B$", on retourne vers la droite jusqu'à ce que 010	
Left_att		
Right_att		

Lors de la détection par C_B, dès que "x0" en entrée pour les boutons BA et B_B, on retourne vers la gauche, jusqu'à ce que 010

3. VHDL (3 pts) 1,5

Soit les définitions suivantes :

```
TYPE byte IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
TYPE mem1 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
TYPE mem2 IS ARRAY (0 TO 3) OF byte;
TYPE mem3 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(0 TO 7);
SIGNAL a: STD_LOGIC;
SIGNAL b: BIT;
SIGNAL x: byte;
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL v: BIT_VECTOR (3 DOWNTO 0);
SIGNAL z: STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);
SIGNAL w1: mem1;
SIGNAL w2: mem2;
SIGNAL w3: mem3;
```



Déterminez parmi les affectations suivantes, celles qui sont légales ou illégales. Justifiez votre réponse uniquement si vous estimez que l'affectation est illégale.

Affectation ✓ ✗	Légale ou illégale	Explications
`y(0) <= x(0);`	✗	pas le même type
`b <= a;`	✗	pas le même type
`y(5 TO 7) <= z(6 DOWNTO 4);`	✓	
`w2 <=((OTHERS=>'0'), (OTHERS=>'0'), (OTHERS=>'0'), (OTHERS=>'0'));`	✓	
`w1(0)(2) <= x(2);`	✗	pas même type
`w3 <= ("11111100", ('0','0','0','0','Z','Z','Z','Z'), (OTHERS=>'0'), (OTHERS=>'0'));`	✓	
`W1(0, 7 DOWNTO 0) <= "11110000";`	✗	0 => pas correct
`y <= (7 =>'0', 1 =>'0', OTHERS => '1');`	✓	
`z <= "1111" & "000";`	✓	
`w1 <= (OTHERS => '1');`	✓	