Éléments de correction
Évaluation Type Codesign

-

## Partie A – *QCM Quartus et VHDL*

**Questions QCM : (Case à cocher pour répondre). Aucune rature autorisée !**

a)

☒ *Transformer un fichier de schéma en description VHDL.*
☒ *Transformer un fichier VHDL en fichier de schéma*

b)

☒ *Le signal a représente l'adresse de la sortie active.*
☒ *L'instruction with... select est une instruction concurrente de VHDL.*
☒ *Si le signal a vaut "110", aucune sortie n'est active.*
☒ *Ici, le cas OTHERS est obligatoire si l'on veut être certain de fabriquer un circuit combinatoire.*

c)

| abc | s |
|---|---|
| xx0 | 0 |
| 001 | 0 |
| 101 | 1 |
| 011 | 1 |
| 111 | 1 |

☐
```
ENTITY myFunc IS
  PORT(
    a, b, c: IN std_logic;
    s: OUT std_logic
  );
END myFunc;

ARCHITECTURE ar OF myFunc IS
BEGIN
  s <= a OR (b AND c);
END ar;
```

■
```
ENTITY myFunc IS
  PORT(
    a, b, c: IN std_logic;
    s: OUT std_logic
  );
END myFunc;

ARCHITECTURE ar OF myFunc IS
  SIGNAL abc: std_logic_vector(2 DOWNTO 0);
BEGIN
  abc <= a & b & c;
  WITH abc SELECT
  s <=
    '1' WHEN "011" | "101" | "111",
    '0' WHEN OTHERS;
END ar;
```

■
```
ENTITY myFunc IS
  PORT(
    a, b, c: IN std_logic;
    s: OUT std_logic
  );
END myFunc;

ARCHITECTURE ar OF myFunc IS
BEGIN
  s <= c AND (a OR b);
END ar;
```

■
```
ENTITY myFunc IS
  PORT(
    a, b, c: IN std_logic;
    s: OUT std_logic
  );
END myFunc;

ARCHITECTURE ar OF myFunc IS
  SIGNAL ab : std_logic;
BEGIN
  s <= ab AND c;
  ab <= a OR b;
END ar;
```

d) ☒ *Avec un tel circuit, il est possible de faire n'importe quel opérateur logique qui comporte au plus 3 entrées et une sortie.*
☒ *La description ci-dessus est un multiplexeur 8 vers 1.*
☒ *Même si le cas OTHERS n'a aucune chance de se présenter, il ne faut pas le supprimer sinon la description ne correspond plus à un circuit combinatoire.*
☒ *Le signal b correspond à l'adresse de l'entrée sélectionnée pour être en sortie.*

**Partie B -*Exercices* - 1h max**

## Exercice N°1.

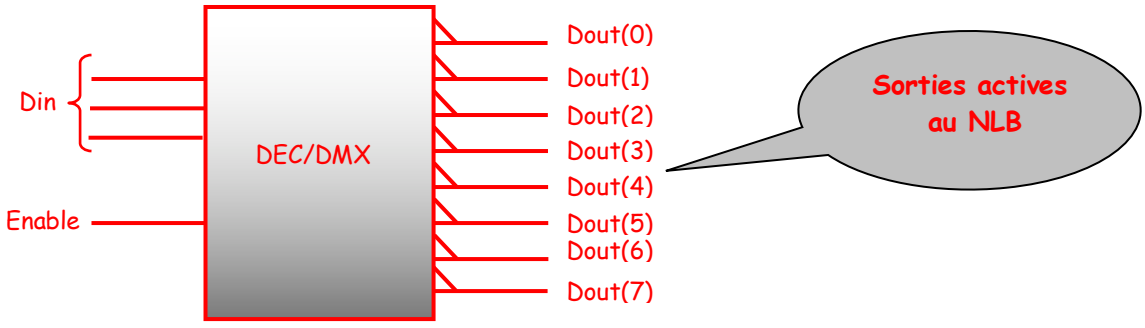
a)

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

ENTITY Exo1_PartB is
      port( Enable : in std_logic ;
            Din: in STD_LOGIC_VECTOR (2 downto 0);
            Dout: out STD_LOGIC_VECTOR (7 downto 0));
end entity Exo1_PartB;

ARCHITECTURE comp_Exo1_PartB of Exo1_PartB is
begin
      PROCESS (Din, Enable)  -- sensible à Din et enable
      variable Temp : STD_LOGIC_VECTOR(7 downto 0); -- un vecteur temporaire
      begin            -- début de l'algorithme
            if (Enable = '1') then            -- si enable vrai alors...
                  Temp := "11111111";         -- Temp est initialisé
                  Temp(TO_INTEGER(UNSIGNED(Din))):= '0' ;
            -- puis le N° présent sur Din est extrait en format integer
            -- et le bit correspondant de Temp est mis à zéro (actif).
                  Dout <= Temp ;  -- La sortie est affectée :Temp(2)
                                  -- par ex=11111011
            else
                  Dout <= (others => 'Z'); -- dans le cas ou Enable est faux  les
                                           --sorties sont à haute impédance.
            end if;
      end PROCESS;     -- fin du process
end architecture comp_Exo1_PartB; -- fin de l'architecture
```

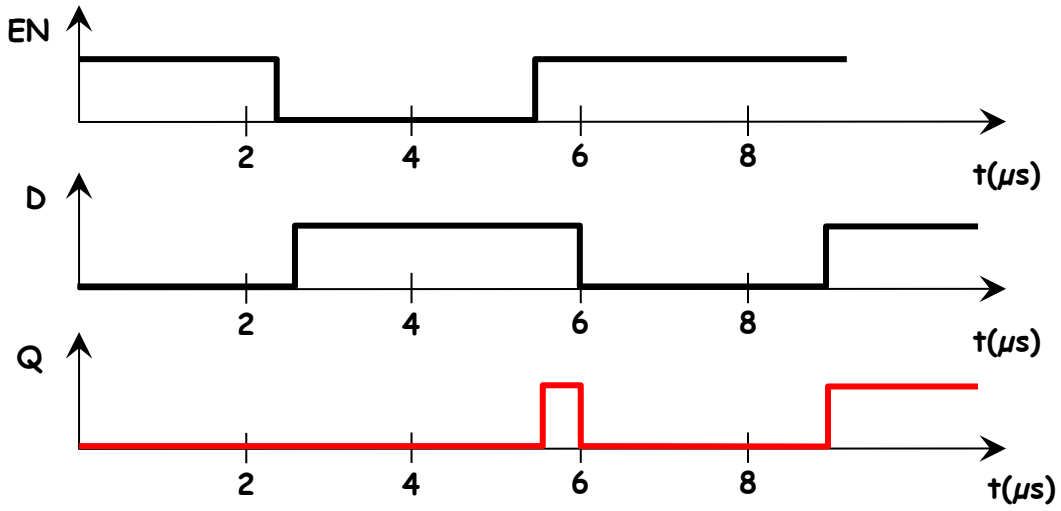
***Réponse attendue.***

***Il s'agit d'un décodeur 3 vers 8 ou 1 parmi 8 dont les sorties sont actives au NLB.***

**Une sortie active parmi 8.**



## Exercice N°2.




## Exercice N°3.

Le code ci-dessous décrit le comportement d'une machine à états à 2 **process** :

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

entity Exo3_PartB is
      port (
            X, CLOCK   : in std_logic;
            Z          : out std_logic
           );
End Exo3_PartB;

architecture comp_Exo3_PartB of Exo3_PartB is
type STATE_TYPE is (S0, S1, S2, S3);
signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;
begin
      process(CURRENT_STATE, X) -- Process Asynchrone(bloc combinatoire de la FSM)
      begin
      case CURRENT_STATE is
      when S0 => Z <= '0';
            if X = '0' then NEXT_STATE <= S0;
            else NEXT_STATE <= S2;
            end if;
      when S1 => Z <= '1';
            if X = '0' then NEXT_STATE <= S0;
            else NEXT_STATE <= S2;
            end if;
      when S2 => Z <= '1';
            if X = '0' then NEXT_STATE <= S2;
            else NEXT_STATE <= S3;
            end if;
      when S3 => Z <= '0';
            if X = '0' then NEXT_STATE <= S3;
            else NEXT_STATE <= S1;
            end if;
      end case;
```

```
    end process;

    process(CLOCK) -- Process Synchrone(bloc séquentiel de la FSM)
    begin
        if CLOCK'event and CLOCK = '1'then
        CURRENT_STATE <= NEXT_STATE;
        End if;
    end process;
end comp_Exo3_PartB;
```

a) **Réponse attendue:**

**Il s'agit d'une machine de Moore car le processus combinatoire du calcul des sorties ne dépend pas des entrées (ici Z ne dépend pas de X dans le processus combinatoire).**

**On peut dire aussi que le codage des sorties s'effectue directement dans les états.**

b) **Réponse attendue :**

**Il suffit de rajouter une entrée RAZ (in : stc_logic) et la ligne suivante dans le process synchrone avant le if CLOCK'event and CLOCK='1', et placer RAZ dans la liste de sensibilité du process**

```
process(CLOCK, RAZ)
begin
if RAZ='0' then CURRENT_STATE<=S0 ;
elsif CLOCK'event and CLOCK='1' then
…
```

c) **Réponse attendue :**

**Les processus sont concurrents. Ils seront donc exécutés en même temps !**

d) et e)**Réponses attendues :**

| Etat Présent | Etat Futur | | Sortie (Z) |
|---|---|---|---|
| | X=0 | X=1 | |
| S0 | S0 | S2 | 0 |
| S1 | S0 | S2 | 1 |
| S2 | S2 | S3 | 1 |
| S3 | S3 | S1 | 0 |