Exercices d'électronique numérique. VHDL.

1. Du code VHDL au circuit.

a Du combinatoire au séquentiel

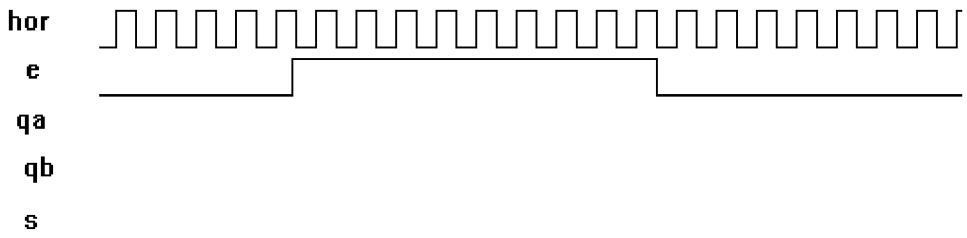
```
-- comb_seq.vhd

entity comb_seq is
      port (
            e1, e2 : in  bit ;
            s_et, s_latch, s_edge : out bit
            ) ;
end comb_seq ;

architecture ww of comb_seq is
begin

et : process
begin
      wait on e1, e2 ;
      if( e1 = '1' ) then
            s_et <= e2 ;
      else
            s_et <= '0' ;
      end if ;
end process ;

latch : process
begin
      wait on e1, e2 until e1 = '1' ;
            s_latch <= e2 ;
end process ;

edge : process
begin
      wait on e1 until e1 = '1' ;
            s_edge <= e2 ;
end process ;
end ww ;
```

- Montrer que les noms des processus correspondent aux opérateurs décrits.
- Proposer une écriture équivalente qui utilise des listes de sensibilités sans instructions « wait ».
- Noter les libertés prises par certains petits compilateurs de synthèse de circuits programmables (discussion avec l'enseignant).

b On considère le programme ci-dessous (écrit en VHDL) :

```
entity transitm is
   port (   hor, e : in bit ;
            s : out bit );
end transitm ;
architecture quasi_struct of transitm is
   signal qa, qb : bit ;
begin
   s <= qa xor qb ;
   schem : process
   begin
      wait until hor = '1' ;
      qa <= e ;
      qb <= qa ;
   end process schem ;
end quasi_struct ;
```

- Déduire de ce programme, par une construction méthodique, un schéma (bascules et portes logiques).
- Compléter le chronogramme ci-dessous.



c On considère le programme VHDL suivant qui décrit le fonctionnement d'une bascule :

```
entity basc is
     port ( T,hor,raz : in bit;
          s : out bit);
end basc;

architecture primitive of basc is
signal etat : bit;
begin
s <= etat ;
process
begin
wait until (hor = '1') ;
     if(raz = '1') then
          etat <= '0';
```

```
            elsif(T = '1') then
                  etat <= not etat;
            end if;
      end process;
      end primitive;
```

- A quoi reconnaît-on qu'il s'agit d'un circuit séquentiel synchrone ?
- La commande « raz » est-elle synchrone ou asynchrone ?
- Etablir le diagramme de transition de cette bascule.
- Déduire du diagramme précédent les équations logiques et le schéma d'une réalisation avec une bascule D.

2. Variables et signaux.

a Operateur OU exlusif generalise

```
-- ouexpar.vhd

ENTITY ouex IS
      generic(taille : integer := 8) ;
      PORT ( a : IN  BIT_VECTOR(0 TO taille - 1) ;
                  s : OUT BIT );
END ouex;

ARCHITECTURE parite of ouex is
BEGIN
process(a)
variable parite : bit ;
begin
      parite := '0' ;
      FOR i in a'range LOOP
            if a(i) = '1' then
                  parite := not parite;
            end if;
      END LOOP;
      s <= parite;
end process;
END parite;

ARCHITECTURE FAUSSE of ouex is
signal parite : bit ; -- MAUVAIS CHOIX
BEGIN
process(a)
begin
      parite <= '0' ;
      FOR i in a'range LOOP
            if a(i) = '1' then
                  parite <= not parite;
            end if;
      END LOOP;
      s <= parite;
end process;
END FAUSSE;
```

- Analyser le fonctionnement de la première architecture proposée. Quelle est la structure du schéma sous-jacente ?

- Pourquoi la deuxième architecture est-elle fausse ?
- Conclure quant aux comportements respectifs des variables et des signaux.

b De la lisibilité du code.

Le programme suivant propose trois versions d'un diviseur de fréquence par 10 :

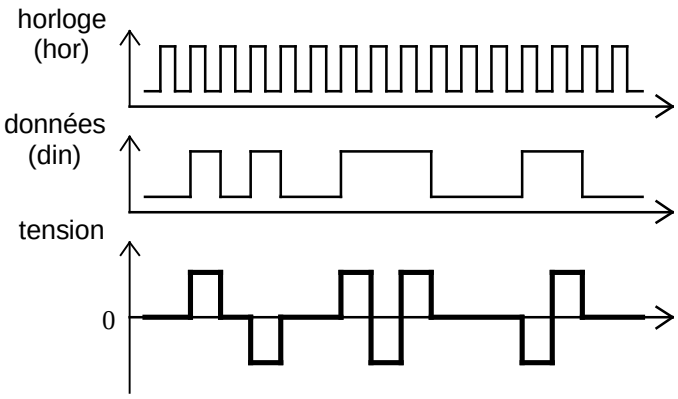
```
-- div_10.vhd

entity div_10 is
     port(      hor : in bit ;
          sort : buffer bit );
end div_10 ;

architecture piege of div_10 is
begin
diviseur : process
variable compte : integer range 0 to 5 := 0 ;
begin
     wait until hor = '1' ;
          compte := compte + 1 ;
          if compte = 5 then
               compte := 0 ;
               sort <= not sort ;
          end if ;
end process diviseur ;
end piege ;

architecture perverse of div_10 is
signal compte : integer range 0 to 4 := 0 ;
begin
diviseur : process
begin
     wait until hor = '1' ;
          compte <= compte + 1 ;
          if compte = 4 then
               compte <= 0 ;
               sort <= not sort ;
          end if ;
end process diviseur ;
end perverse ;

architecture correcte of div_10 is
signal compte : integer range 0 to 4 := 0 ;
begin
diviseur : process
begin
     wait until hor = '1' ;
          if compte = 4 then
               compte <= 0 ;
               sort <= not sort ;
          else
               compte <= compte + 1 ;
          end if ;
end process diviseur ;
end correcte ;
```

- Discuter les différentes versions. Réalisent-elles toutes la fonction annoncée ?
- Quel est le piège de la première version en synthèse ?

3. Exercice de synthèse

Dans les transmissions téléphoniques à grande distance, les informations transitent sous forme numérique, transmises en série (un bit à la fois), au rythme d'une horloge. Le code binaire utilisé est transformé en un code à 3 niveaux de tension sur la ligne (câble coaxial, par exemple) :

⇒ un ZERO logique correspond toujours à une tension nulle,



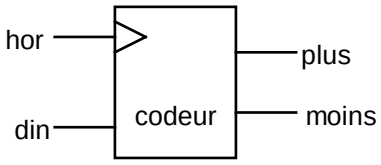
⇒ les niveaux logiques UN sont représentés par des impulsions, qui durent une période de l'horloge de transmission, alternativement positives et négatives, d'où le nom du code. On notera que le système doit se « souvenir » de la polarité de la dernière impulsion transmise pour fonctionner correctement.

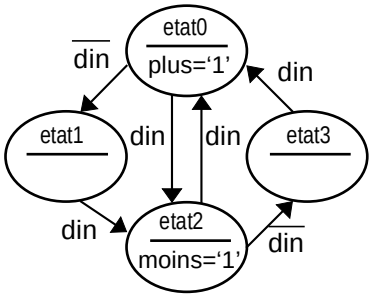
La création des impulsions alternées passe par un changement de code : le codeur reçoit l'horloge d'émission, hor, et les données à transmettre, din. Il fournit en sortie deux signaux binaires que nous nommerons plus et moins, générés suivant l'algorithme ci-dessous :

⇒ si din = '0' : plus = '0' , moins = '0' ;

⇒ si din = '1' : plus = '1' , moins = '0' ou plus = '0' , moins = '1', en alternance.

On se propose d'étudier plusieurs solutions pour réaliser ce codeur.





L'idée générale est de réaliser une machine synchrone à quatre états, conformément à la figure ci-contre.

L'ébauche de diagramme de transitions proposé est évidemment incomplète.

- Compléter le diagramme de transitions proposé, en indiquant les conditions de transitions et de maintiens éventuels.
- Quel est le nombre minimum de bascules nécessaires à la réalisation du codeur ?
- Proposer un codage des états et une solution complète du problème qui utilise ce nombre minimum de bascules. Proposer un programme VHDL qui réponde au problème.
- On souhaite que les sorties du codeur soient issues directement de bascules du registre d'état. Proposer une solution en précisant bien le nombre de bascules utilisées et un diagramme de transitions complet.
- Proposer un programme VHDL qui réalise cette nouvelle solution.
- Quels sont les avantages et inconvénients respectifs des deux solutions étudiées ?

VHDL : éléments de correction

1. Du code VHDL au circuit.

a Du combinatoire au séquentiel

- Le processus « et » est combinatoire : s_et prend une valeur définie quelques soient les valeurs de e1 et e2.
  Pour le processus latch : si e1 = '1' tout changement est reporté en sortie, c'est le mode transparent. Si e1 = '0' la valeur de s_latch n'est pas spécifiée, elle ne change pas, donc est mémorisée.
  Pour le processus edge : seuls des changements de e1, suivis par une valeur de e1 égale à '1' sont susceptibles de modifier la sortie. C'est une bascule sensible aux fronts montants de e1.
-

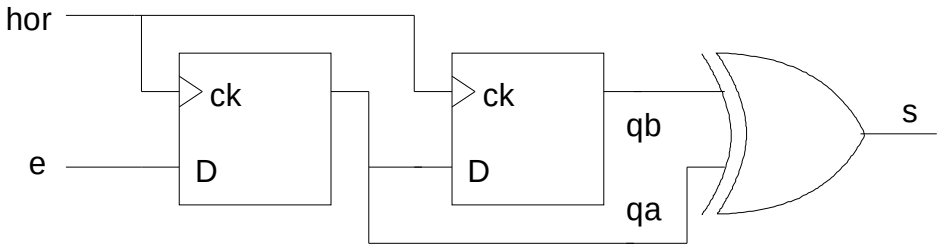
```
architecture ww of comb_seq is
begin

et : process (e1,e2)
begin
     if( e1 = '1' ) then
          s_et <= e2 ;
     else
          s_et <= '0' ;
     end if ;
end process ;

latch : process (e1,e2)
begin
     if( e1 = '1' ) then
          s_latch <= e2 ;
     end if ;
end process ;

edge : process (e1)
begin
     if( e1 = '1' ) then
          s_edge <= e2 ;
     end if ;
end process ;
end ww ;
```

- Des bascules sûres :

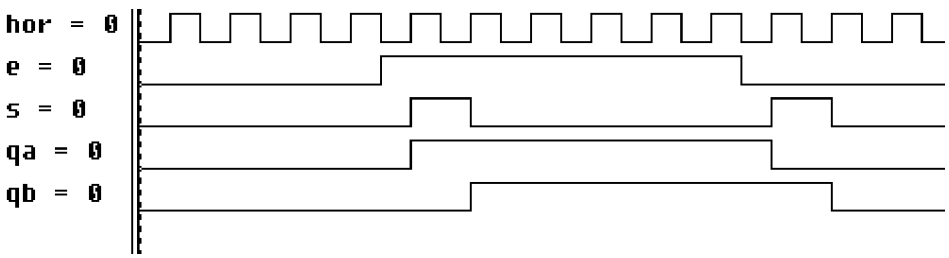
```
entity d_edge is
   port ( d,hor : in bit ;
          s : out bit) ;
end d_edge;

architecture d_wait of d_edge is
begin
process
begin
wait until hor = '1';
     s <= d ;
end process ;
end d_wait ;
```

```
architecture d_liste of d_edge is
begin
process (hor)
begin
   if hor = '1' and hor'event then
      s <= d ;
   end if ;
end process ;
end d_liste ;

-- commandes asynchrones
-- dedgeraz.vhd

entity d_edge_raz is
   port ( d, hor, raz : in bit ;
          q : out bit) ;
end d_edge_raz ;

architecture d_liste of d_edge_raz is
begin
process (hor,raz)
begin
   if raz = '1' then
      q <= '0' ;
   elsif hor = '1' and hor'event then
      q <= d ;
   end if ;
end process ;
end d_liste ;
```

b Schéma correspondant :



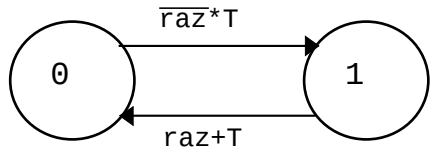
-
- Chronogramme :



c On considère le programme VHDL suivant qui décrit le fonctionnement d'une bascule :

- Toutes les affectations relatives au signal « etat » sont conditionnées par un front montant du signal hor.

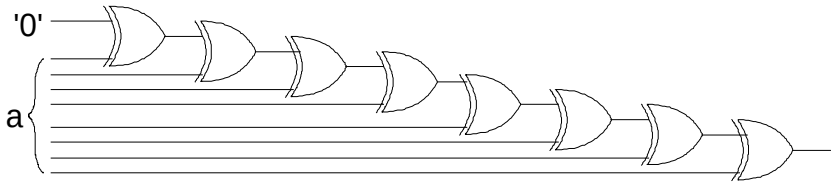
- La commande « raz » est synchrone. Pour créer une commande asynchrone, voir question 1-a.



- $D = \overline{Q} * \overline{raz} * T + Q * \overline{raz} * \overline{T} = \overline{raz} * (Q \oplus T)$

2. Variables et signaux.

a Operateur OU exlusif generalise



- La deuxième architecture est fausse : le signal « parite » ne change pas de valeur dans le corps de la boucle. Seul reste un schéma pathologique :

```
s =  (a_7 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1 + a_0 )* /s
```

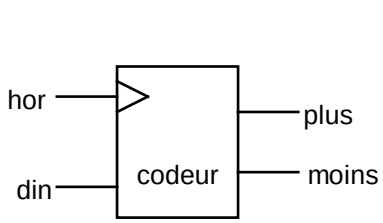
qui crée un oscillateur si l'une des entrées au moins est à '1'.

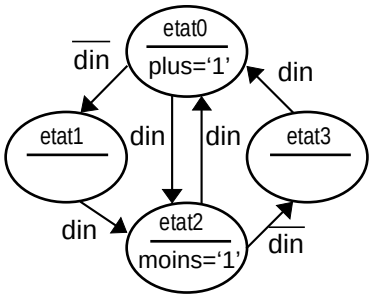
- L'affectation à une variable a un effet immédiat, comme dans un langage classique, l'affectation à un signal ne prend effet que quand le processus se remet en attente. Dit autrement, un signal ne change pas de valeur au cours de l'exécution d'un processus.

b De la lisibilité du code.

- Les trois versions réalisent le même opérateur.
- En synthèse, la première version génère un signal à la place de la variable, si l'outil de synthèse accepte cette construction, ce qui n'est pas toujours le cas. Le signal généré compte de 0 à 4, ce qui est correct, pas de 0 à 5 comme la variable. La deuxième version provoque une erreur en simulation, compte + 1 dépasse 4, il y aurait donc débordement s'il n'y avait pas l'instruction « if » qui modifie la valeur projetée de compte.

3. Exercice de synthèse





- 4 états, donc 2 bascules au minimum.
- Deuxième solution plus simple et, surtout, beaucoup plus rapide, il n'y a pas de couche de logique combinatoire en sortie.

```
entity amicod is port(hor, din : in bit ;
        plusout, moinsout : out bit ); end amicod ;
```

```
architecture sort_decode of amicod is
type ami is (mzero,pzero,moins,plus) ;
signal etat : ami ;
begin
plusout  <= '1' when etat = plus  else '0' ;
moinsout <= '1' when etat = moins else '0' ;
encode : process
begin
   wait until hor = '1' ;
   case etat is
      when mzero =>    if din = '1' then etat <= plus  ;
            end if ;
      when pzero =>    if din = '1' then etat <= moins ;
            end if ;
      when moins =>    if din = '1' then etat <= plus ;
            else etat <= mzero ;
            end if ;
      when plus  =>    if din = '1' then etat <= moins ;
            else etat <= pzero ;
            end if ;
      when others =>   etat <= mzero ;
   end case ;
end process encode ;
end sort_decode ;
```

- sorties directes :

```
architecture sort_direct of amicod is
subtype ami is bit_vector(2 downto 0) ;
constant mzero : ami := "000" ;
constant pzero : ami := "001" ;
constant moins : ami := "010" ;
constant plus  : ami := "100" ;
signal etat : ami ;
begin
plusout  <= etat(2) ;
moinsout <= etat(1) ;
encode : process
begin
   wait until hor = '1' ;
   case etat is
      when mzero =>    if din = '1' then etat <= plus  ;
            end if ;
      when pzero =>    if din = '1' then etat <= moins ;
            end if ;
      when moins =>    if din = '1' then etat <= plus ;
            else etat <= mzero ;
            end if ;
      when plus  =>    if din = '1' then etat <= moins ;
            else etat <= pzero ;
            end if ;
      when others =>   etat <= mzero ;
   end case ;
end process encode ;
end sort_direct ;
```