Travaux dirigés d'informatique industrielle
Analyse de programmes VHDL

1. Du code VHDL au circuit.

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).
-

Modélisations d'opérateurs simples

Les questions de ce paragraphe font référence aux exemples de programmes simples donnés en **annexe du TP_VHDL**.

Circuits combinatoires

Multiplexeurs

- Analyser et comparer les différentes versions du multiplexeur 2 vers 1 proposées.
- Utiliser trois multiplexeurs 2 vers 1 pour réaliser un multiplexeur 4 vers 1.
- Proposer d'autres solutions de réalisation de ce multiplexeur, au moyen d'instructions concurrentes et d'instructions séquentielles.

Décodeur

- Analyser le fonctionnement du modèle de décodeur 74138. Préciser les rôles des différentes entrées.
- Associer deux décodeurs de ce type pour réaliser un décodeur quatre vers 16. Programme VHDL correspondant.

Circuits séquentiels

Bascule D

- Analyser et tester les différentes versions de bascule D. En quoi diffèrent elles ?

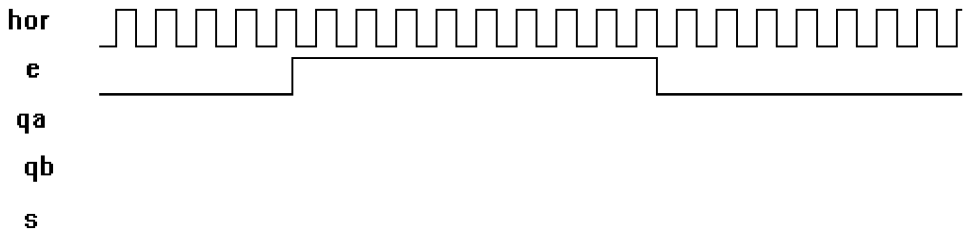
Bascule JK

- Proposer des programmes de description d'une bascule JK.
  - Entièrement synchrone.
  - Avec des entrées `preset` et `clear` asynchrones.

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.



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 reconnait-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.

Où l'on retrouve le monte-charge

Le programme ci-dessous reprend la commande de monte-charge vue en C.

```
-- montecharge
-- monte_c.vhd

entity monte_c is
      port (raz, hor, p0, p1, appel : in bit ;
              mot_monte, mot_desc, panne : out bit ) ;
end monte_c ;

architecture automate of monte_c is
      type monte_c_et is (init, arret, descend, monte, erreur) ;
      signal etat : monte_c_et ;
```

```
begin

fsm : process
begin
      wait until hor = '1' ;
      if raz = '1' then
            etat <= init ;
      else
            case etat is
                  when init =>
                        if p0 /= '1' then
                              etat <= descend ;
                        else
                              etat <= arret ;
                        end if ;
                  when arret =>
                        if p0 = '1' and p1 = '1' then
                              etat <= erreur ;
                        elsif p0 = '1' and appel = '1' then
                              etat <= monte ;
                        elsif p1 = '1' and appel = '1' then
                              etat <= descend ;
                        end if ;
                  when monte =>
                        if p1 = '1' then
                              etat <= arret ;
                        end if ;
                  when descend =>
                        if p0 = '1' then
                              etat <= arret ;
                        end if ;
                  when others => null ;
            end case ;
      end if ;
end process fsm ;

actions : process(etat)
begin
      mot_monte <= '0' ;
      mot_desc  <= '0' ;
      panne <= '0' ;
      case etat is
            when monte =>
                  mot_monte <= '1' ;
            when descend =>
                  mot_desc <= '1' ;
            when erreur =>
                  panne <= '1' ;
            when others => null ;
      end case ;
end process actions ;
end automate ;
```

- Préciser par une représentation structurelle les différents blocs qui apparaissent dans ce programme en distinguant bien leur nature combinatoire ou séquentielle.
- Un extrait du compte-rendu de synthèse figure ci-dessous, quelle est la dimension du registre d'état ?

```
      State variable 'etat' is represented by a Bit_vector (0 to 2).
      State encoding (sequential) for 'etat' is:
            init :=     b"000";
            arret :=    b"001";
            descend :=  b"010";
            monte :=    b"011";
```

```
            erreur :=   b"100";
```

- Modifier le calcul des sorties pour qu'elles soient mémorisées dans des bascules. (au moins deux solutions différentes)
- Quelle taille de circuit faut-il prévoir dans les différentes versions ?
- Les utilisateurs se plaignent du temps de réaction trop long (une période d'horloge) entre le moment où ils appuient sur le bouton d'appel et le moment où le monte-charge réagit, quelle solution proposez-vous ? (autre que d'augmenter la fréquence de l'horloge …)

2. Variables et signaux.

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.

De la lisibilité du code.

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

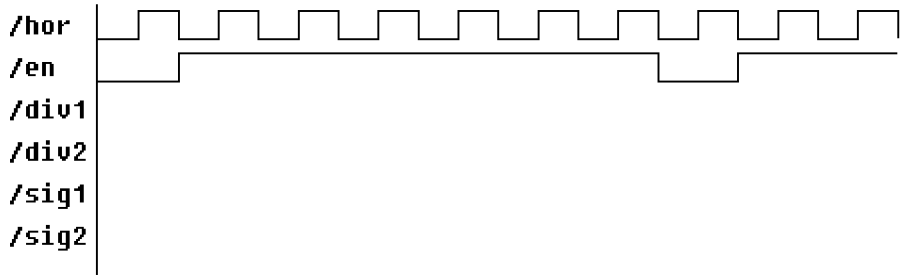
```
-- div_10.vhd

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. Etude d'un diviseur de fréquence

Le programme fourni en annexe est destiné à générer deux signaux à une fréquence différente de celle de l'horloge d'entrée.

a. Traduire ce programme en un schéma synoptique dans lequel apparaîtront des bascules, des blocs combinatoires dont on ne demande pas les détails internes et des signaux d'interconnexions nommée.
b. Représenter le fonctionnement logique de chaque processus par un diagramme de transitions (un par processus) commenté.
c. A partir de la réponse à la question précédente, compléter le chronogramme ébauché ci-dessous :



d. On synthétise le diviseur dans un PLD 22V10-15, dont une partie de la notice est fournie à l'annexe 3. Associer à chaque élément de la question a un temps caractéristique extrait de cette notice. Quelle est la fréquence maximum de fonctionnement du dispositif ?
e. La commande en a une action synchrone. Modifier le programme pour qu'elle ait une action asynchrone. Quelle propriété doit alors posséder le PLD utilisé ?

Annexes

1 Programme du diviseur de fréquence

```
1   entity divnn is
2      port (hor, en : in bit ;
3            div1, div2 : out bit ) ;
4   end divnn ;
5
6   architecture combien of divnn is
7      signal sig1 : integer range 0 to 3 ;
8      signal sig2 : bit ;
9   begin
10     div1 <= '1' when sig1 = 2 else '0' ;
11     div2 <= sig2 ;
12  d1 : process
13     begin
14        wait until hor = '1' ;
15        if en = '0' then
16           sig1 <= 0 ;
17        else
18           case sig1 is
19              when 0 => if sig2 = '1' then
20                           sig1 <= 1 ;
21                        end if ;
22              when 1 => sig1 <= 2 ;
23              when 2 => sig1 <= 0 ;
24              when others => sig1 <= 0 ;
```

```
1                  end case ;
2               end if ;
3            end process ;
4         d2 : process
5            begin
6               wait until hor = '1' ;
7               if en = '0' then
8                  sig2 <= '0' ;
9               else
10                 case sig2 is
11                    when '0' => if sig1 = 0 then
12                                   sig2 <= '1' ;
13                                end if ;
14                    when others => sig2 <= '0' ;
15                 end case ;
16              end if ;
17           end process ;
18        end combien ;
19
```

2 Extrait de la notice du PLD CE22V10-15

Commercial Switching Characteristics PALCE22V10

Parameter	Description	22V10-15		Unit
		Min.	Max.	
t_{PD}	Input to Output Propagation Delay	3	15	ns
t_{CO}	Clock to Output Delay	2	8	ns
t_{S1}	Input or Feedback Set-Up Time	10		ns
t_{S2}	Synchronous Preset Set-Up Time	10		ns
t_{H}	Input Hold Time	0		ns
t_{CF}	Register Clock to Feedback Input		4.5	ns

4. Synchronisation inter processus.

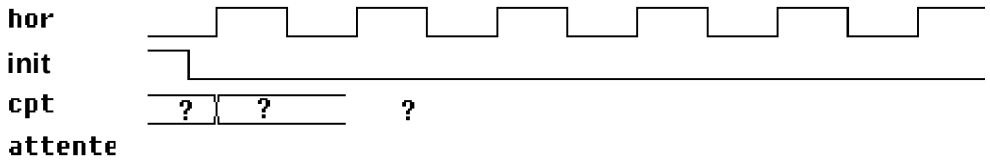
On considère le programme VHDL ci-dessous :

```
entity couple is
   port ( hor,init : in bit ;
                cpt : out integer range 0 to 3 ) ;
end couple ;

architecture deux_proc of couple is
   signal compte : integer range 0 to 3 ;
   signal attente : bit ;
begin
   cpt <= compte ;

   compteur : process (hor, init)
   begin
```

```
   if init = '1' then
      compte <= 3 ;
   elsif hor'event and hor = '1' then
      case compte is -- équivalent du switch du C
         when 1      => if attente = '1' then
                           compte <= compte - 1 ;
                        end if ;
         when 0      => compte <= 3 ;
         when others => compte <= compte - 1 ;
      end case ;
   end if ;
   end process compteur ;

   retard : process
   begin
      wait until hor = '1' ;
      if compte = 1 then
         attente <= '1' ;
      else
         attente <= '0' ;
      end if ;
   end process retard ;
end deux_proc ;
```

Questions : (toutes les réponses doivent justifiées par une analyse du programme)

a. Etablir un schéma synoptique du circuit décrit. Dans cette question on ne demande pas les détails, seuls doivent figurer des blocs et les signaux nommés dans le programme.
b. Combien de bascules ce programme génère-t-il si les nombres entiers qui y figurent sont codés en binaire naturel ?
c. Certaines de ces bascules ont un fonctionnement entièrement synchrone, d'autres possèdent une commande asynchrone de l'horloge. Préciser ce point.
d. Compléter le chronogramme ci-dessous et commenter le résultat.



e. Représenter le fonctionnement synchrone de chaque processus par un diagramme de transitions commenté.
f. Déduire des diagrammes précédents les équations de commandes des bascules avec des bascules D, puis des bascules JK.
   Représenter les logigrammes correspondants (portes et bascules pourvues des commandes asynchrones éventuellement nécessaires).
g. Pour le schéma obtenu avec des bascules D :
   Les paramètres dynamiques des portes et bascules sont les suivants :

temps de propagation	Tp <= 10 ns
temps de maintien	Thold = 0 ns
temps de prépositionnement	Tsetup = 15 ns

   Représenter le fonctionnement du système pendant deux périodes caractéristiques de l'horloge, en faisant figurer ces temps (Echelle suggérée : 1 carreau = 5 ns, période d'horloge de 50 ns).

Quelle est la fréquence maximum de fonctionnement de ce montage ?

h. Le montage précédent est refusé en synthèse dans certains circuits programmables, bien que sa complexité ne soit pas en jeu. Expliquer pourquoi.

5. Largeur d'une impulsion.

On se propose de mesurer la largeur d'une impulsion et de fournir le résultat de la mesure, à la demande d'un microprocesseur.

L'impulsion à mesurer est asynchrone de l'horloge du circuit ; le résultat est arrondi à un nombre entier de périodes d'horloge. Si l'impulsion est trop longue et dépasse la capacité du circuit de mesure, la valeur transmise est toujours la même, égale au plus grand nombre entier que sait traiter le circuit.

Le programme VHDL complet est fourni en annexe (la syntaxe de (others => ...) se comprend en consultant le polycopié au paragraphe « agrégat »).

- Fournir un schéma synoptique (boites noires) du circuit réalisé.
- Quelle est l'utilité de la bascule triviale générée par le process « synchro » ?
- Pourquoi utilise-t-on ici le type « unsigned » plutôt qu'un type integer ?
- Décrire en quelques mots le fonctionnement du bloc attaché au process compteur. Noter, en particulier son comportement quand il y a dépassement de sa capacité.
- Décrire le fonctionnement du bloc généré par le process « moore » au moyen d'un diagramme de transitions. On repèrera les états par leurs noms symboliques.
- Comment le micro-processeur est il prévenu qu'une mesure est prête ? Quelle est alors la séquence normale des opérations ?
- Quelle est l'utilité de l'état « cohérence », en quoi est il nécessaire si on suppose que les impulsions dont on mesure la largeur arrivent « n'importe quand ».
- Préciser en quelques mots les principales caractéristiques que doit posséder un circuit programmable susceptible d'accueillir la fonction décrite.

Annexe

```
-- mesure de la largeur d'une impulsion
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity duree is
   port( hor, read, inpulse, raz : in std_ulogic;
         fini : out std_ulogic;
         largeur : out unsigned(15 downto 0) );
end duree ;

architecture multiproc of duree is
   type fsm5 is (pret,comptage,attente_lect,lecture,coherence) ;
   signal commande : fsm5 ;
   signal compte : unsigned(15 downto 0);
   constant maxcpt : unsigned(15 downto 0) := (others => '1');
   signal syncpulse : std_ulogic ;
   signal on_y_va : boolean ; -- valeurs FALSE ou TRUE, test logique
begin
   largeur <= compte when read = '1' else (others => 'Z');
   on_y_va <= (commande = pret) or (commande = comptage);
   fini <= '1' when commande = attente_lect else '0';
synchro : process
   begin
   wait until rising_edge(hor);
      syncpulse <= inpulse;
   end process synchro ;
compteur : process
```

```
   begin
   wait until rising_edge(hor);
   if commande = coherence then
      compte <= (others => '0');
   elsif on_y_va and syncpulse = '1' then
      if compte < maxcpt then
         compte <= compte + 1;
      end if;
   end if;
   end process compteur;
moore : process
   begin
   wait until rising_edge(hor);
   if raz = '1' then
      commande <= coherence;
   else
      case commande is
         when pret         => if syncpulse = '1' then
                                 commande <= comptage ;
                              end if;
         when comptage     => if syncpulse = '0' then
                                 commande <= attente_lect ;
                              end if;
         when attente_lect => if read = '1' then
                                 commande <= lecture ;
                              end if;
         when lecture      => if read = '0' then
                                 commande <= coherence ;
                              end if;
         when coherence    => if syncpulse = '0' then
                                 commande <= pret ;
                              end if;
         when others       => commande <= coherence ;
      end case;
   end if;
   end process moore ;
end multiproc;
```

6. Synthèse de circuits décrits en VHDL : timer

- Expliquez, en vous appuyant éventuellement sur un exemple bien choisi, en quoi il peut être préférable de prévoir plus de bascules que le minimum nécessaire lors de la conception d'une machine à nombre fini d'états.
- Comment contrôle-t-on, dans le source VHDL, la dimension et le codage du registre d'états d'un opérateur séquentiel ?
- Montrer que les deux architectures du programme fourni en annexe réalisent la même fonction : un diviseur de fréquence par un nombre modifiable par l'utilisateur. Préciser les valeurs de division que l'on peut obtenir.
- L'architecture « simple » est synthétisable dans un circuit 22V10, l'architecture « lourde » ne l'est pas, le nombre de produits générés par les équations combinatoires étant trop élevé. Pourquoi ?

```
package timerpkg is
   constant nbits : integer := 9 ; -- paramètre constant
end package ;

library ieee;
use ieee.numeric_bit.all ;
use work.timerpkg.all ;

entity timer is
```

```
   port ( hor, en : in bit ;
          fin : in unsigned(nbits-1 downto 0) ;
          hordiv : out bit ) ;
end timer ;

architecture simple of timer is
   signal cpt : unsigned(nbits-1 downto 0) ;
   signal div : bit ;
begin

hordiv <= div ;

compte : process
   begin
   wait until rising_edge(hor) ;
      if en = '0' then
         cpt <= fin ;
         div <= '0' ;
      elsif cpt = 0 then
         cpt <= fin ;
         div <= not div ;
      else
         cpt <= cpt - 1 ;
      end if ;
   end process compte ;
end simple ;

architecture lourde of timer is
   signal cpt : unsigned(nbits-1 downto 0) ;
   signal div : bit ;
begin

hordiv <= div ;

compte : process
   begin
   wait until rising_edge(hor) ;
      if en = '0' then
         cpt <= (others => '0') ;
         div <= '0' ;
      elsif cpt >= fin then
         cpt <= (others => '0') ;
         div <= not div ;
      else
         cpt <= cpt + 1 ;
      end if ;
   end process compte ;
end lourde ;
```