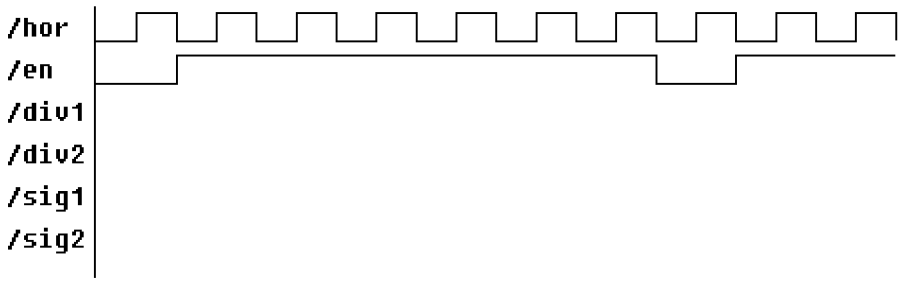
1. Etude d'un diviseur de fréquence

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

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' ;
```

```
               if en = '0' then
                  sig1 <= 0 ;
               else
                  case sig1 is
                     when 0 => if sig2 = '1' then
                                    sig1 <= 1 ;
                               end if ;
                     when 1 => sig1 <= 2 ;
                     when 2 => sig1 <= 0 ;
                     when others => sig1 <= 0 ;
                  end case ;
               end if ;
            end process ;
         d2 : process
            begin
               wait until hor = '1' ;
               if en = '0' then
                  sig2 <= '0' ;
               else
                  case sig2 is
                     when '0' => if sig1 = 0 then
                                    sig2 <= '1' ;
                                 end if ;
                     when others => sig2 <= '0' ;
                  end case ;
               end if ;
            end process ;
         end combien ;
```

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

2. 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
```

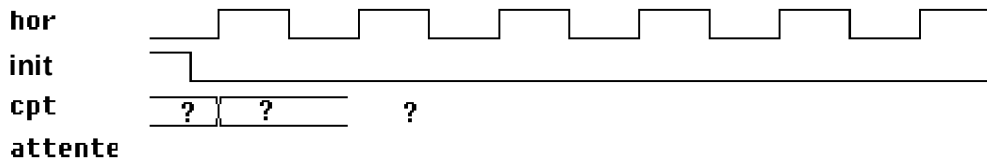
```
   signal compte : integer range 0 to 3 ;
   signal attente : bit ;
begin
   cpt <= compte ;

   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.

3. 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;
```

4. 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 ;
```

5. Analyse : comparaison de deux réponses à un même problème.

Pour réaliser un codeur AMI (la connaissance de ce code n'est pas nécessaire à la compréhension du sujet), deux concepteurs différents proposent deux solutions, toutes deux justes, concrétisées par deux architectures (amidec et amidir) décrivant la même entité. Les résultats de simulation fonctionnelle des deux solutions sont identiques (voir figures en annexe).

Les programmes sources sont donnés en annexe.

Chacune de ces deux solutions est synthétisée et implémentée dans un circuit programmable AMD 16V8H-25 (t_{PD} = 25 ns, t_{CO} = 12 ns, F_{maxint} = 40 MHz).

a Pour chaque solution indiquer la structure (pas les équations détaillées) du circuit généré. Préciser soigneusement les natures, combinatoires ou séquentielles, des différents blocs fonctionnels. Déduire de chaque programme la dimension du registre d'état correspondant. Construire les diagrammes de transitions associés.

b On teste les circuits réalisée avec une horloge à 40 Mhz.

Les résultats des deux tests sont fort différents, comme l'attestent les chronogrammes fournis en annexe. Interprétez quantitativement ces chronogrammes en vous appuyant sur les documents constructeur. Pour faciliter l'analyse on a placé des curseurs à des instants intéressants, et demandé l'affichage de l'écart temporel entre les curseurs.

c Conclure.

Annexe : programmes

entité commune :

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

architecture « amidec » :

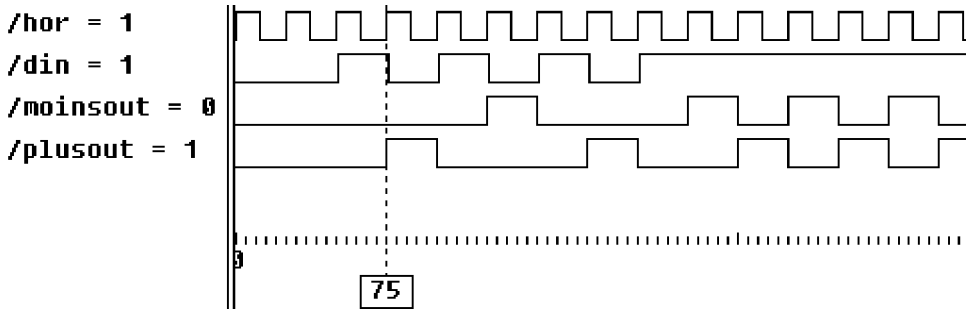
```
architecture amidec of amicod is
type ami is (mzero,pzero,moins,plus) ;
signal etat : ami ;
begin
plusout <= '1' when etat = plus else '0' ;
moinsout <= '1' when etat = moins else '0' ;
encode : process
begin
   wait until hor = '1' ;
   case etat is
      when mzero =>    if din = '1' then etat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

architecture « amidir » :

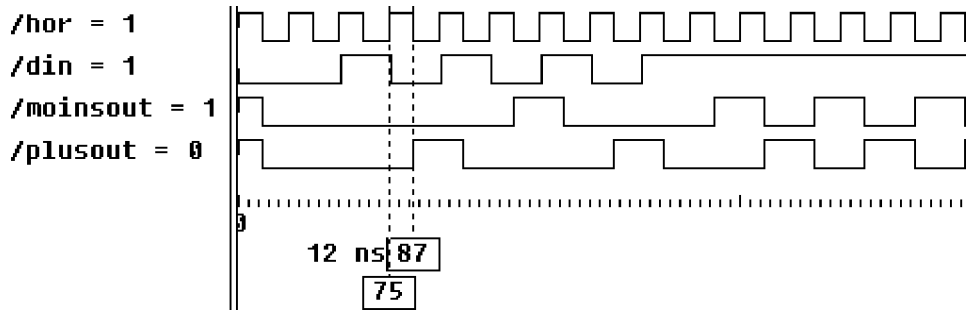
```
architecture amidir of amicod is
subtype ami is bit_vector(2 downto 0) ;
constant mzero : ami := "000" ;
constant pzero : ami := "001" ;
constant moins : ami := "010" ;
constant plus  : ami := "100" ;
signal etat : ami ;
begin
plusout <= etat(2) ;
moinsout <= etat(1) ;
encode : process
begin
   wait 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 amidir ;
```

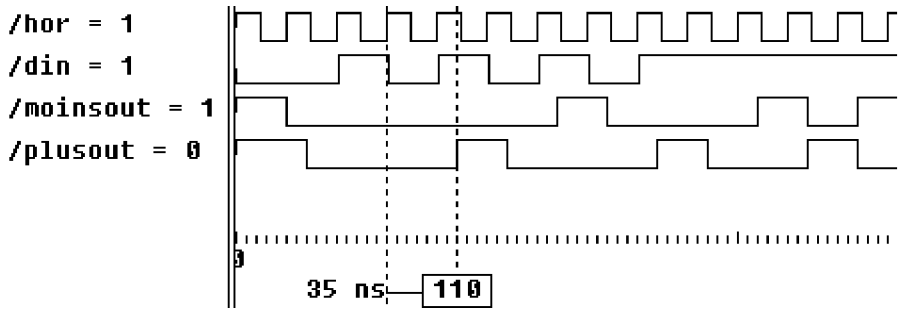
amidec ou amidir : simulation fonctionnelle du programme source



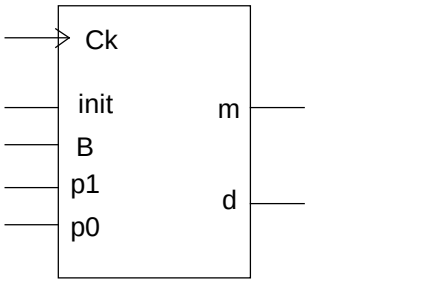
amidir : fonctionnement du circuit



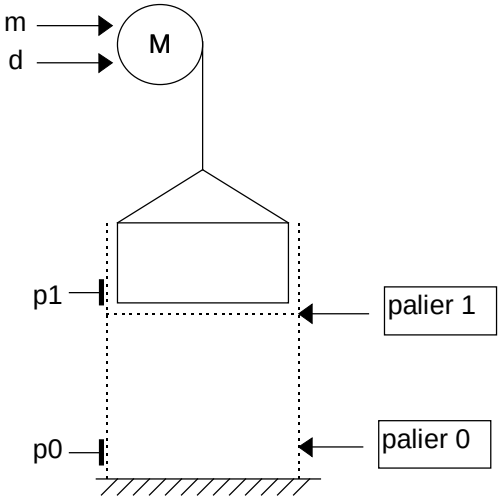
amidec : fonctionnement du circuit



6. Synthèse : commande de monte charge.



B : bouton de commande
pi : détecteurs de position
m, d : commandes du moteur
init : initialisation



Le moteur est commandé par deux signaux m et d pour montée et descente.
La commande élabore ces commandes à partir de ses entrées B, p1, p0 et init suivant le cahier des charges suivant :

- init provoque la descente du monte-charge jusqu'au palier p0. Ce signal est activé à la mise sous tension. On suppose qu'un plancher empêche le monte-charge d'être à un niveau inférieur à p0 !.
- Quand la cabine est entre deux étages, la dernière commande (m ou d) est maintenue.
- Quand la cabine est à un étage, si B est inactif elle s'arrête, si B est actif, elle change d'étage.

a Etablir un diagramme de transition général, discuter son fonctionnement et lever toutes les ambiguités du cahier des charges.
b Proposer des solutions sous forme de machines de Moore et de Mealy.
c Proposer des solutions telles que B doive effectuer une transition inactif→actif pour être pris en compte.
d En déduire, pour chaque solution, un synoptique et les équations de commandes des bascules.
e Proposer des solutions VHDL correspondantes.

7. Codeur HDB3

Dans les liaisons téléphoniques numériques à haut débit (2, 8 et 34 Mbits/s) on utilise en Europe un code à trois niveaux dérivé du code AMI (voir poly de travaux pratiques) :

- Les bits d'information égaux à '1' sont transmis par des impulsions alternativement positives ou négatives, de façon à maintenir une valeur moyenne nulle pour le signal véhiculé sur la ligne. Les bits égaux à '0' correspondent à une différence de potentiel nulle sur la ligne (« absence » d'impulsion).
- L'horloge de réception utilise les transitions du signal reçu pour maintenir une bonne synchronisation ; des longues séquences de '0' risquent donc de créer des glissements entre l'horloge d'émission et l'horloge de réception. Pour remédier à ce problème, toutes les séquences de quatre '0' successifs sont codées par des motifs (suites d'impulsions et de niveaux nuls) qui, pour être différenciés des motifs « normaux », ne respectent pas les règles d'alternance. On parle de *violation*.

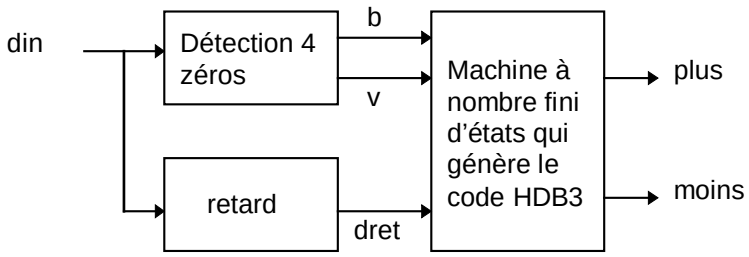
- Pour assurer le maintien de la valeur moyenne nulle, deux violations successives doivent être de polarités opposées. Si cette alternance ne correspond pas à la situation « naturelle », c'est un problème de parité du nombre des '1' qui séparent deux groupes successifs de quatre '0', on introduit dans le motif un élément de *bourrage* avant l'élément de « viol ».

Pour résumer, toute séquence de quatre zéros consécutifs génère dans le code de sortie le motif générique "b00v", où b est une éventuelle impulsion de bourrage (qui respecte la règle d'alternance) et v une impulsion (toujours présente) qui ne respecte pas la dite règle.

Dans l'exemple ci-dessous on représente les signaux ternaires émis par +, 0 et – :

données	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
code émis	**0**	**0**	**+**	**0**	**0**	**0**	**+**	**–**	**+**	**–**	**0**	**0**	**–**	**+**	**–**	**+**	**0**	**0**	**+**	**–**	**0**	**0**	**0**	**–**
bourrage viol							**v**			**b**			**v**			**b**			**v**					**v**

La structure du codeur à réaliser est la suivante :



Sur ce schéma de principe on n'a pas représenté l'horloge d'émission qui synchronise l'arrivée des données et l'ensemble du système. Les signaux b et v indiquent ici les positions que doivent avoir les éventuels motifs de bourrage et de viol. Les signaux binaires de sortie plus et moins servent à générer des impulsions de polarités correspondantes.
Le bloc détecteur de quatre zéros n'ayant pas la faculté de dire l'avenir, il est nécessaire de retarder d'un nombre ad-hoc de périodes d'horloge le signal d'entrée du codeur.

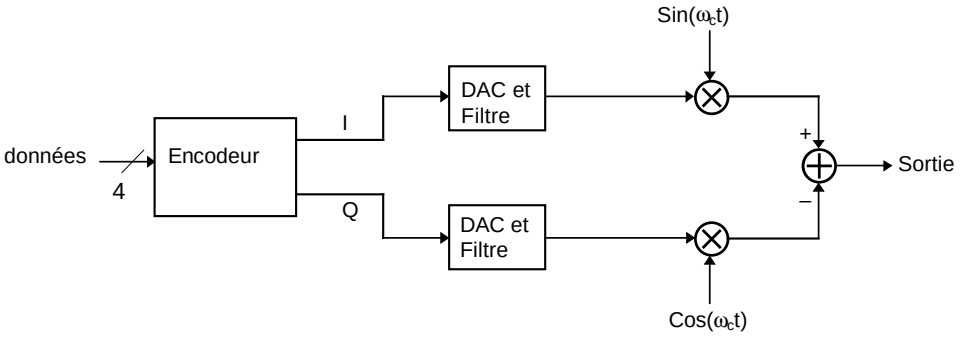
Une ébauche d'algorithme décrivant le codeur proprement dit pourrait être quelque chose du genre :

```
Si le dernier viol est positif
        Si la dernière impulsion de sortie est positive
                générer un bourrage négatif,
                générer un viol négatif.
        Autrement
                ne pas générer de bourrage,
                générer un viol négatif.
Autrement
        Si la dernière impulsion de sortie est négative
                générer un bourrage positif,
                générer un viol positif.
        Autrement
                ne pas générer de bourrage,
                générer un viol positif.
```

8. Codeur pour modulateur QAM différentiel

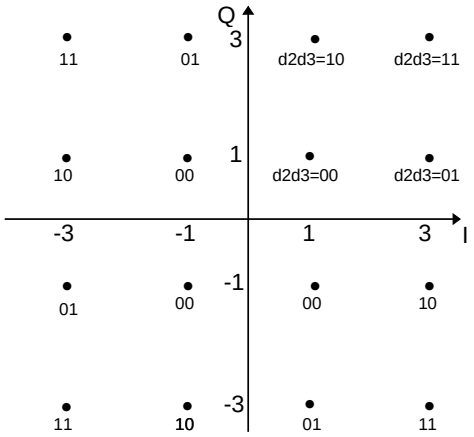
Dans nombre de systèmes de transmissions numériques, l'objectif est de transmettre le plus haut débit binaire possible dans un canal de bande passante donnée (et compte tenu d'un rapport signal/bruit non infini).
Une méthode classique consiste à moduler en amplitude et en phase (modulation vectorielle) une porteuse. Le « vecteur de Fresnel » associé à la porteuse peut avoir un nombre fini de valeurs, les points représentatifs de l'extrémité de ce vecteur, dans le plan complexe, constituent une

constellation qui caractérise la modulation. Les informations binaires d'entrée provoquent des changements d'état dans la constellation, changements qui obéissent à une règle de codage.
Le schéma de principe typique d'un tel système est représenté ci-dessous.



L'exercice proposé (inspiré des modems téléphoniques) consiste à réaliser un codeur différentiel qui fixe des trajectoires dans une constellation à 16 états en calculant les signaux I (in phase) et Q (quadrature), codés sur deux bits (I_2I_1 et Q_2Q_1), en fonction des données d'entrée qui sont regroupées par paquets de quatre bits ($d_3d_2d_1d_0$).

La constellation du modulateur est décrite ci-dessous :



Les points sont répartis dans quatre quadrants ; les valeurs de d2 et d3 déterminent le point dans le quadrant concerné, celles de d0 et d1 déterminent les changements de quadrant entre un code et le suivant (codage différentiel de la phase) :

d0 d1	changement de phase
0 0	+ 90°
0 1	0
1 1	+270°
1 0	+180°

Les valeurs de I et Q sont des entiers signés, théoriquement sur trois bits, mais les valeurs étant toujours impaires, le poids faible est une constante câblée de façon fixe à l'entrée des convertisseurs numériques analogiques.

Le travail consiste à réaliser le codeur, piloté par l'horloge qui cadence l'arrivée (par paquets de 4 bits) des données d'entrée.

9. Décodeur pour modulateur QAM différentiel

Même problème que le précédent, « à l'envers ». Les données d'entrée sont les valeurs de I et Q, celles de sortie sont les valeurs de d3d2d1d0.

N.B. : autant le codeur qui intervient dans un vrai modem suit d’assez près la structure présentée ici, autant le décodage est en réalité nettement plus complexe si on tient compte des perturbations apportées par le canal de transmission (bruit et interférences entre symboles).