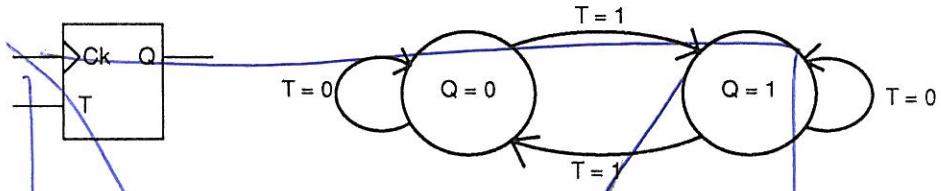

Figure III-26

Un exemple de réalisation

L'examen du diagramme de transitions de la figure III-26 nous montre que Q(t) = 1 si

Q(t-1) = '1' et T = '0',

ou

Q(t-1) = '0' et T = '1'

Ce qui nous fournit l'équation de l'entrée D d'une bascule D :

$$D = T \oplus Q$$

D'où le logigramme :

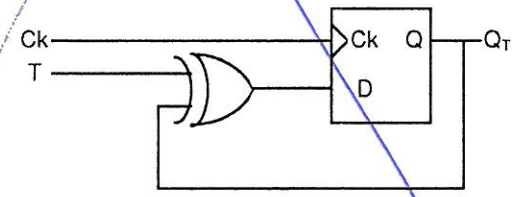

Figure III-27

Description en VHDL

La description d'une bascule T se déduit simplement du diagramme de transition :


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

architecture d_primitive of T_edge is
signal etat : bit;
begin
s <= etat ;
process
begin
wait until hor = '1' ;
if(T = '1') then
      etat <= not etat;
end if;
end process;
end d_primitive;
```

On rappelle que de tels exemples sont fournis à titre d'illustration du fonctionnement de l'opérateur considéré, et pour familiariser le lecteur avec le langage VHDL. On n'a jamais besoin, en pratique, de décrire chaque bascule utilisée dans ce langage !

Les applications

L'un des intérêts principaux des bascules de type T est qu'elles permettent de générer de façon extrêmement simple des compteurs binaires synchrones. Un compteur binaire est une fonction séquentielle synchrone dont l'état interne est un nombre entier naturel codé en binaire dont chaque chiffre binaire est matérialisé par une bascule. A chaque transition active de l'horloge ce nombre est incrémenté de 1, quand le nombre maximum est atteint, toutes les bascules sont à 1, la séquence recommence à partir de 0. Si le nombre de bits utilisés est n, on parlera d'un compteur modulo 2^n. La simple observation d'une table des entiers naturels écrits en base 2 nous fournit la clé du problème : la bascule de rang i doit changer d'état quand toutes les bascules de rang inférieur sont à 1.

D'où un exemple de réalisation d'un compteur synchrone modulo 16 dont on peut interrompre le comptage (en = '0') :

```
ENTITY cnt16 IS
PORT (ck, en : IN BIT;
      s : OUT BIT_VECTOR (0 TO 3)
      );
END cnt16;

ARCHITECTURE structurelle OF cnt16 IS
SIGNAL etat : BIT_VECTOR(0 TO 3);
SIGNAL inter: BIT_VECTOR(1 TO 3);
COMPONENT T_edge
```

```
-- la même que dans l'exemple précédent
port ( T,hor: in bit;
       s : out bit);
END COMPONENT;

BEGIN
-- Etablir le logigramme tout en lisant le texte
s <= etat ;
inter(1) <= etat(0) and en ;
inter(2) <= etat(1) and inter(1) ;
inter(3) <= etat(2) and inter(2) ;
g0 : T_edge port map (en,ck, etat(0));
g1 : for i in 1 to 3 generate
       g2 : T_edge port map (inter(i),ck,etat(i));
end generate;
END structurelle;
```

Là encore mettons en garde le lecteur, quand on a réellement besoin d'un compteur on écrit

```
etat <= etat + 1 ;
```

c'est nettement plus simple, le compilateur générera de lui même les interconnexions nécessaires entre les bascules.

L'ancêtre vénérable : la bascules J-K

Le principe

Quelque peu tombée en désuétude, la bascule J-K a régné en maître dans le monde de la logique séquentielle des décennies 60 et 70. Elle est l' héritière directe de la bascule R-S, que l'on a débarassé progressivement de ses difficultés asynchrones. Les premières bascules J-K n'étaient, en fait, pas de réels opérateurs synchrones, elles comportaient tout un mécanisme de mémorisation interne des commandes dans des bascules R-S (maître-esclave). Les versions actuelles sont construites au moyen d'une bascule D-edge et de logique combinatoire.

Une bascule J-K dispose de deux commandes, J et K, outre l'horloge. Son fonctionnement se décrit bien au moyen d'une table qui décrit la fonction réalisée en fonction des valeurs de la commande :

J(t-1)	K(t-1)	Fonction	Equation
0	0	Mémoire	Q(t) = Q(t-1)
0	1	Mise à zéro synchrone	Q(t) = '0'
1	0	Mise à un synchrone	Q(t) = '1'
1	1	Changement d'état	Q(t) = $\overline{Q(t-1)}$

Comme pour tout opérateur synchrone, l'état de la bascule ne change pas entre deux transitions actives du signal d'horloge. La fonction « mémoire », dans la table ci-dessus, signifie que la bascule conserve son état précédent même lors d'une transition d'horloge. Dans la construction du diagramme de transitions de la figure suivante, III-28, on a tenu compte de ce que chaque transition peut être obtenue par deux combinaisons différentes des commandes J et K, la transition '0' → '1', par exemple, peut être obtenue par mise à 1 explicite (JK = "10") ou par changement d'état (JK = "11") :

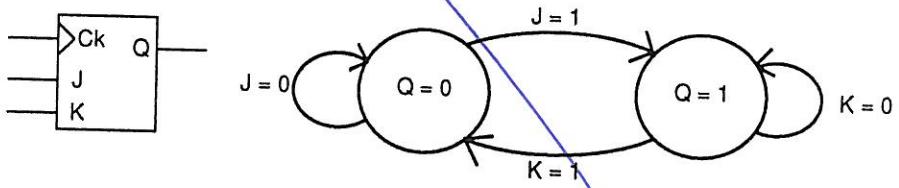

Figure III-28

On déduit aisément l'équation de la bascule J-K de son diagramme de transition :

$$Q(t) = J * \overline{Q(t-1)} + \overline{K} * Q(t-1)$$

Au lieu de raisonner sur l'équation de l'état futur, on peut décrire la bascule J-K par son équation de transition, si on introduit une variable binaire auxilliaire T_{JK}, égale à '1' si une transition doit avoir lieu, '0' autrement, on obtient :

$$T_{JK} = J * \overline{Q} + K * Q$$

Description en VHDL

La première description que nous donnerons est la simple traduction naïve de la table de vérité :

```
entity jk is port (
```

```
                moore_state <= O"7" ;
            end if ;
when O"2" => if man = '0' -- etat inutiles
             then
                moore_state <= O"6" ;
            else
                moore_state <= O"5" ;
            end if ;
when O"3" => if man = '0' then -- bis
                moore_state <= O"4" ;
            else
                moore_state <= O"7" ;
            end if ;
   end case ;
end process moore_mach ;
end comporte ;
```

Codage des états

Quand on utilise des circuits standard, des compteurs programmables, par exemple, pour réaliser une machine séquentielle, le codage des états du diagramme de transitions est, de fait, imposé par le circuit cible. Il en va tout autrement quand la dite machine doit être implantée dans un circuit programmable ou un ASIC. Libéré des contraintes liées à une quelconque fonction prédéfinie, le concepteur peut, à loisir, adapter le codage des états à l'application qu'il est en train de réaliser.

Le choix d'un code est particulièrement important quand on s'oriente vers la réalisation d'une machine de Moore. L'exemple du décodeur Manchester nous a appris que l'un des avantages de cette architecture réside dans la possibilité de générer les sorties directement à partir du registre d'état, donc dénuées de tout parasite lié à leur calcul. Mais, comme nous le verrons dans deux exemples, l'identification des sorties du système à celles des bascules du registre d'état ne suffit généralement pas pour définir le codage des états.

Ce choix du codage mérite une grande attention, il conditionne grandement la complexité de la réalisation, sa bonne adaptation au problème posé ; un choix judicieux conduira à un résultat simple et facilement testable, alors qu'aucun logiciel d'optimisation ne compensera des erreurs de décision à ce niveau.

Le nombre d'états nécessaires et le type de code adopté fixent, en premier lieu, la taille du registre d'état. Schématiquement, si n est la taille, en nombre de bits, du registre d'état, et N_e le nombre d'états nécessaires, ces deux nombres (entiers !) doivent vérifier la double inégalité :

$$n \leq N_e \leq 2^n$$

Si l'inégalité de gauche n'est pas vérifiée, certaines bascules sont probablement inutiles ; quand cette inégalité se transforme en égalité, on utilise un code très

« dilué », une bascule par état, qui présente l'avantage de la lisibilité, mais le danger de générer en grand nombre des états accessibles inutilisés (rappelons ici qu'il y a toujours 2^n états accessibles).

Si l'inégalité de droite n'est pas vérifiée, la tentative est sans espoir ; si elle se transforme en égalité, on utilise un encodage « fort », auquel il faudra très probablement adjoindre des fonctions combinatoires de calcul des sorties ; on ne réalise pas que des compteurs binaires ou des codeurs de position absolue (code de Gray).

Les situations intermédiaires correspondent en général à des codes adaptés aux sorties.

Encodage « fort » ou code « dilué » ? En caricaturant un peu, on peut dire que les tenants de la première solution préfèrent les fonctions combinatoires, et que les seconds sont des adeptes des bascules. Il n'est pas évident, à priori, de prévoir la complexité des équations engendrées par tel ou tel code. On gagne souvent à suivre le fonctionnement « naturel » de la machine[24], et, surtout, on gagne à se souvenir que les ordinateurs, et leurs compilateurs, ne sont pas posés sur un bureau à titre de décoration ; ils permettent de voir très vite quelle est la complexité sous jacente d'un choix, sans pour celà tomber dans le BAO[25].

Codes adaptés aux sorties

L'idée qui vient naturellement à l'esprit est de choisir le codage en fonction des sorties à générer. C'est souvent la méthode la plus souple, celle qui conduit aux équations les plus faciles à interpréter, et pas forcément plus compliquées que celles que l'on obtiendrait avec d'autres codes.

Une commande de feux tricolores.

Pour satisfaire à une tradition bien établie, nous prendrons comme premier exemple une commande de feux de circulation routière.

Un passage pour piétons traverse une avenue ; il est protégé par un feu tricolore qui fonctionne à la demande des piétons : En l'absence de toute demande, les feux sont à l'orange clignotant (un nombre Tor de secondes allumés, Tor secondes éteints). Quand un piéton souhaite traverser l'avenue, il est invité à appuyer sur un bouton, ce qui provoque le déclenchement d'une séquence (vue des voitures) :

- orange fixe pendant 2*Tor secondes,
- rouge pendant Tr secondes,
- vert pendant Tv secondes, pour laisser passer le flot de voitures pendant un minimum de temps,
- retour à la situation par défaut.

[24] Mais qu'est-ce que ce fonctionnement naturel ? Sa recherche est, sans doute, l'une des parties les plus intéressantes, et donc souvent difficile, du travail.

[25] Bricolage Assisté par Ordinateur

Profitons de cet exemple pour subdiviser la solution du problème en sous ensembles. Trois blocs fonctionnels peuvent être identifiés :

1. La commande des feux proprement dite, les sorties de trois bascules du registre d'état commandent directement l'allumage, ou l'extinction, des lampes rouge, verte et orange.
2. Une temporisation qui, suite à une commande d'initialisation, fournit les trois durées Tor, Tr et Tv.
3. Une mémorisation de l'appel des piétons, qui évite de se poser des questions concernant la durée pendant laquelle le demandeur appuie sur le bouton ; une simple pression suffit, l'appel est alors enregistré, quel que soit l'état d'avancement de la séquence de gestion des feux.

Outre les commandes des feux proprement dites, le bloc principal fournit un signal d'initialisation (cpt) à la temporisation, qui doit durer une période d'horloge[26], et un signal d'annulation (raz) de la requête, mémorisée, d'un piéton.

Les signaux d'entrée de ce bloc sont la requête (piet) et les trois indications de durée Tor Tr et Tv ; nous supposerons que ces dernières passent à '1', pendant une période d'horloge, quand les durées correspondantes se sont écoulées.

D'où le synoptique de la figure V-17 :

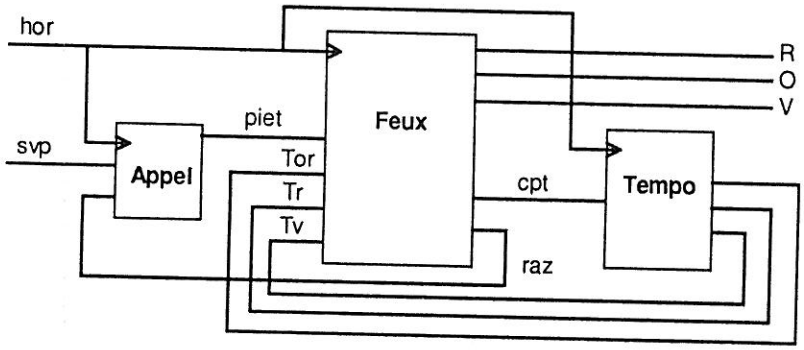

Figure V-17

[26] Nous sommes en train de définir trois processus qui se commandent et/ou s'attendent mutuellement. Le danger de ce type d'architecture, très fréquente, est de générer des interblocages : un processus en initialise un second et attend une réponse de ce dernier. Si le demandeur oublie de relâcher la commande d'initialisation, le système est bloqué. Ce type de situation porte, en informatique, le doux nom d'étreinte fatale (*deadly embrace*). La solution adoptée ici est d'envoyer des signaux fugaces (mais synchrones !), ce qui oblige le demandeur à attendre la réponse dans un état différent de celui où il a passé la commande d'initialisation

Nous nous contenterons d'étudier, ici, le bloc principal, feux, laissant la synthèse des deux autres blocs à titre d'exercice.

Première ébauche :

Le fonctionnement général peut être celui illustré par la figure V-18 :

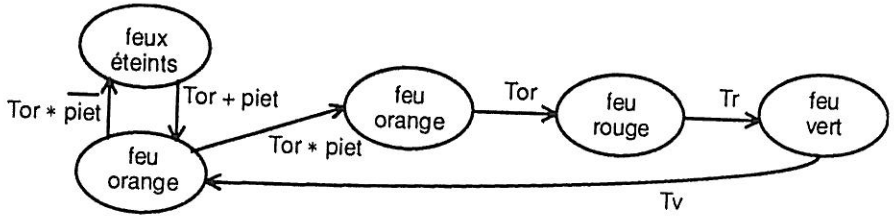

Figure V-18

Précisions :

A partir de l'ébauche précédente, il nous reste à préciser le mode de calcul des signaux gérés par le processus feux, et à en déduire le codage des états. Le signal cpt se prête bien à une réalisation sous forme de sortie de Mealy, les signaux de commande des feux à une réalisation sous forme de sorties de Moore. Les deux états où le feu orange est allumé doivent être distingués, une bascule supplémentaire, qui n'est attachée à aucune sortie, doit être rajoutée à cette fin. La sortie raz peut être identique à la sortie qui correspond au feu rouge ; il n'est pas utile de mémoriser une demande de piéton quand les voitures sont arrêtées au feu rouge. D'où une version plus élaborée du diagramme de transitions (figure V-19) :

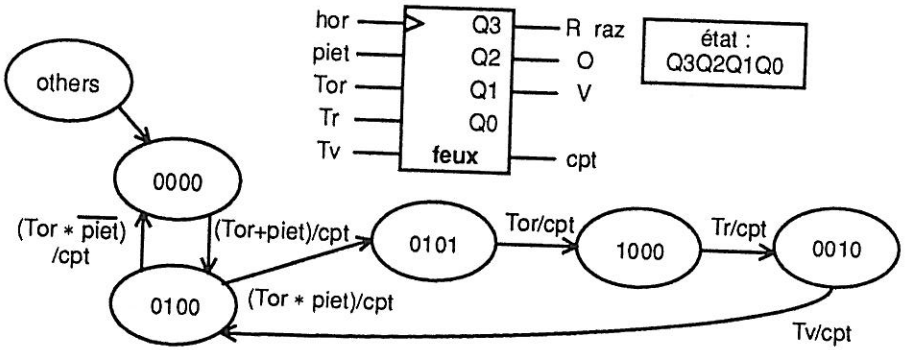

Figure V-19

Programme VHDL :

Un exemple de programme VHDL, qui correspond au module feux uniquement, est fourni ci-dessous ; il se déduit directement du diagramme de transitions précédent.

```
entity feux is
   port ( hor, piet, Tor, Tr, Tv : in bit ;
     R, O, V, cpt : out bit );
end feux ;

architecture comporte of feux is
signal etat : bit_vector(3 downto 0) ;
begin
R <= etat(3) ;
O <= etat(2) ;
V <= etat(1) ;

machine : process -- diagramme de transitions.
begin
   wait until hor = '1' ;
      case etat is
         when X"0" -- états en hexadécimal.
                 => if (Tor or piet) = '1' then
                       etat <= X"4" ;
                    end if ;
         when X"4" => if (Tor and piet) = '1' then
                       etat <= X"5" ;
            elsif (Tor and not piet) = '1' then
                       etat <= X"0" ;
                    end if ;
         when X"5" => if Tor = '1' then
                       etat <= X"8" ;
                    end if ;
         when X"8" => if Tr = '1' then
                       etat <= X"2" ;
                    end if ;
         when X"2" => if Tv = '1' then
                       etat <= X"4" ;
                    end if ;
         when others => etat <= X"0" ;
            -- pour les états inutilisés.
      end case ;
end process machine ;

mealy : process -- calcul de la sortie cpt.
begin
   wait on etat, piet, Tor, Tr, Tv ; -- liste de sensibilité
```



```
   cpt <= '0' ; -- assure un bloc combinatoire.
      case etat is
         when X"0" => if Tor = '1' or piet = '1' then
                         cpt <= '1' ;
                      end if ;
         when X"4" => if Tor = '1' then
                         cpt <= '1' ;
                      end if ;
         when X"5" => if Tor = '1' then
                         cpt <= '1' ;
                      end if ;
         when X"8" => if Tr = '1' then
                         cpt <= '1' ;
                      end if ;
         when X"2" => if Tv = '1' then
                         cpt <= '1' ;
                      end if ;
         when others => null ; -- case complet.
      end case ;
end process mealy ;
end comporte ;
```

Le décodeur Manchester réexaminé.

Comme deuxième exemple, reprenons, en la complétant un peu, l'étude du décodeur Manchester différentiel. Nous avions omis, dans la version précédente, un deuxième signal de sortie, rx, qui indique aux utilisateurs la cadence de transmission. Comme on peut le voir sur la figure V-20 , ce signal a une fréquence moitié de celle de l'horloge, mais il ne peut pas s'agir d'un simple diviseur par deux : un diviseur par deux est incapable de distinguer les transitions systématiques des transitions signifiantes du signal d'entrée man, il est incapable de se synchroniser.

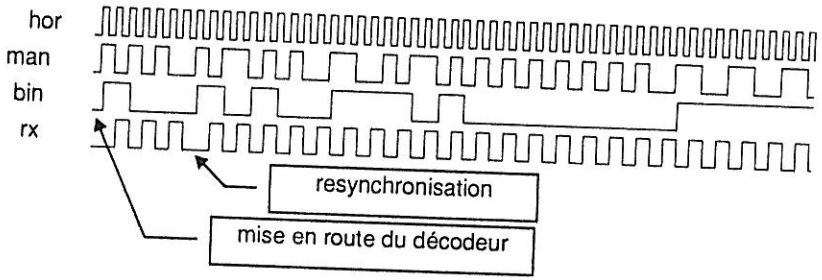

Figure V-20