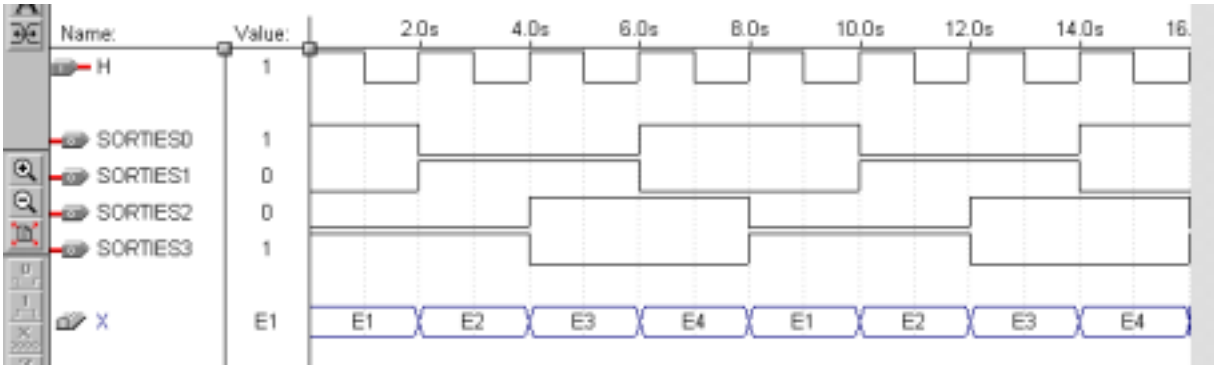
Machines d'état

Les machines d'états permettent d'effectuer la synthèse de systèmes numériques séquentiels, généralement synchrones (mais la synthèse de systèmes asynchrones est possible) à partir d'un cahier des charges.
Quelques exemples vont permettre de nous familiariser avec cette notion.

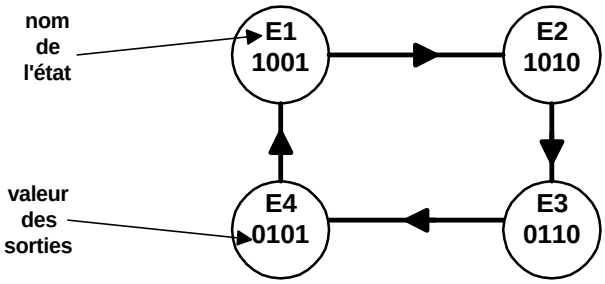
Exemple 1 : moteur pas à pas à sens de rotation unique.

Synthétisons dans ce premier exemple le séquenceur destiné à la commande d'un moteur pas à pas, unipolaire 4 phases, à un seul sens de rotation, en fonctionnement pas entier deux phases.
Nous avons ci-après les signaux recherchés S0, S1, S2 et S3 pour la commande des phases en fonction de l'horloge H :



On note 4 états possibles en sortie, s'enchaînant inconditionnellement dans l'ordre E1, E2, E3, E4, E1…au rythme de l'horloge H.

A partir de ce cahier des charges, on peut en déduire le diagramme (ou diagramme à bulles) suivant ;



Ce diagramme représente les différents états du système, en précisant éventuellement les niveaux logiques correspondants en sortie, et les conditions de passage d'un état à l'autre. Dans notre cas très simple, à chaque coup d'horloge, on passe inconditionnellement à l'état suivant.

La machine doit toujours se trouver au moins dans un état, et dans un seul état.

Avant la propagation des circuits CPLD et FPGA et des langages HDL de programmation, on cherchait ensuite les équations des entrées et sorties des bascules de mémorisation des états. Aujourd'hui, il est possible de retranscrire directement le diagramme d'état en programme ; voici par exemple, un fichier VHLD associé à ce diagramme :

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

entity CDE_MOT_2 is
        port(   H               : in    std_logic;
                SORTIES         : out std_logic_vector (3 downto 0));
end CDE_MOT_2;

architecture ARC_CDE OF CDE_MOT_2 is
        type TYPE_ETAT is (E1, E2, E3, E4);
        signal X:TYPE_ETAT;
begin
        process (H)
        begin
                if H'event and H = '1' then
                        case X is
                                when E1 =>      X <= E2;
                                when E2 =>      X <= E3;
                                when E3 =>      X <= E4;
                                when others =>  X <= E1;
                        end case;
                end if;
        end process;

        with  X select
                SORTIES                 <=      "1001"  when    E1,
                                                "1010"  when    E2,
                                                "0110"  when    E3,
                                                "0101"  when    E4;

end ARC_CDE;
```

L'architecture du programme comprend deux parties distincts : l'instruction « process » gère le passage d'un état au suivant à chaque coup d'horloge, tandis l'instruction « with select » affecte aux sorties les valeurs correspondant à chaque état.

Remarques : le programme aurait pu être simplifié (et les ressources du circuit cible économisées) en remarquant que les sorties étaient complémentaires deux à deux.

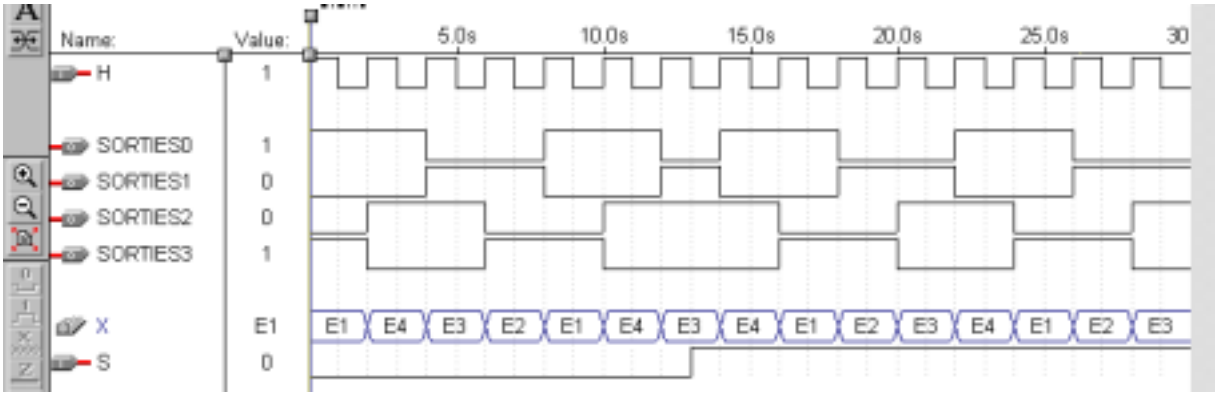
On notera que tous les états possibles des sorties n'ont pas été traités dans le diagramme d'état, afin de ne pas l'alourdir. Dans la description VHDL cependant, les états imprévus conduisent à l'état E1 (instruction « when others ») au coup d'horloge suivant. Cette solution n'est pas toujours la plus intéressante, un état imprévu conduisant alors à un fonctionnement aléatoire difficile à détecter et à supprimer ; il est parfois préférable d'avoir une panne franche, en prévoyant par exemple un état « Erreur ».

Reprendre le diagramme d'état et le programme dans le cas d'une commande en pas entier une phase.

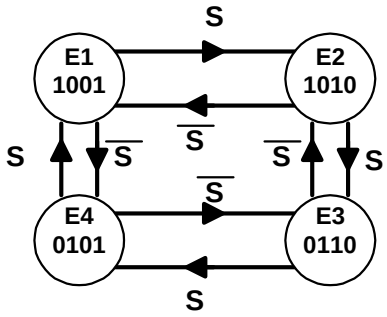
Exemple 2 : moteur pas à pas à double sens de rotation.

Reprenons le premier exemple en introduisant cette fois une commande S permettant de choisir le sens de rotation. Nous supposerons cette commande synchrone dans un premier temps.

Les chronogrammes recherchés sont alors :



Ce qui nous conduit au diagramme d'état suivant :



Remarques :

- lorsque plusieurs entrées interviennent dans un diagramme d'état, on suppose qu'une seule est susceptible de changer à un instant donné ;
- lorsque les conditions ne sont pas réalisées, l'état ne change pas (dans notre exemple, les conditions étant complémentaires, elles sont toujours réalisées) ;
- lorsque plusieurs branchements sont possibles à partir d'un état, les conditions ne doivent pas être vraies en même temps ;
- la présence de l'horloge est implicite, le passage d'un état à l'autre ne peut se faire qu'au front actif de l'horloge.

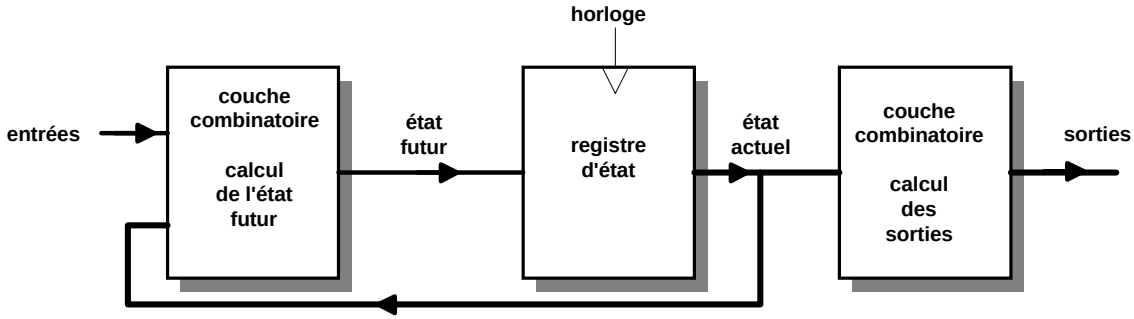
Une description VHDL possible est alors donnée ci-après :

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

entity CDE_MOT_2S is
        port(   H,S     : in    std_logic;
        SORTIES         : out std_logic_vector (3 downto 0));
end CDE_MOT_2S;

architecture ARC_CDE OF CDE_MOT_2S is
        type TYPE_ETAT is (E1, E2, E3, E4);
        signal X:TYPE_ETAT;
begin
        process (H)
        begin
                if H'event and H = '1' then
                        case X is
                                when E1 =>
                                        if S='1'  then     X <= E2;
                                                                else     X <=E4;
                                        end if;
```

```
                        when E2 =>
                                if S='1'  then     X <= E3;
                                                        else     X <=E1;
                                end if;


                        when E3 =>
                                if S='1'  then     X <= E4;
                                                        else     X <=E2;
                                end if;

                        when others =>
                                if S='1'  then     X <= E1;
                                                        else     X <=E3;
                                end if;

                end case;
        end if;
    end process;

    with  X select
        SORTIES             <=      "1001"  when    E1,
                                    "1010"  when    E2,
                                    "0110"  when    E3,
                                    "0101"  when    E4;

end ARC_CDE;
```

Machine de Moore

Le fonctionnement décrit précédemment correspond à celui d'une machine dite de Moore dont le schéma est donné ci-après :



Avec ce type de machine, un registre d'état (composé de bascules) mémorise l'état actuel, tandis qu'une couche combinatoire positionne l'entrée du registre afin de pouvoir activer l'état suivant au coup d'horloge suivant.

Les sorties sont déterminées par une couche combinatoire en fonction des sorties du registre d'état. Elles sont donc synchrones avec l'horloge.

Les performances du système vont beaucoup dépendre de la simplicité ou de la complexité des couches combinatoires. Plusieurs stratégies sont possibles :

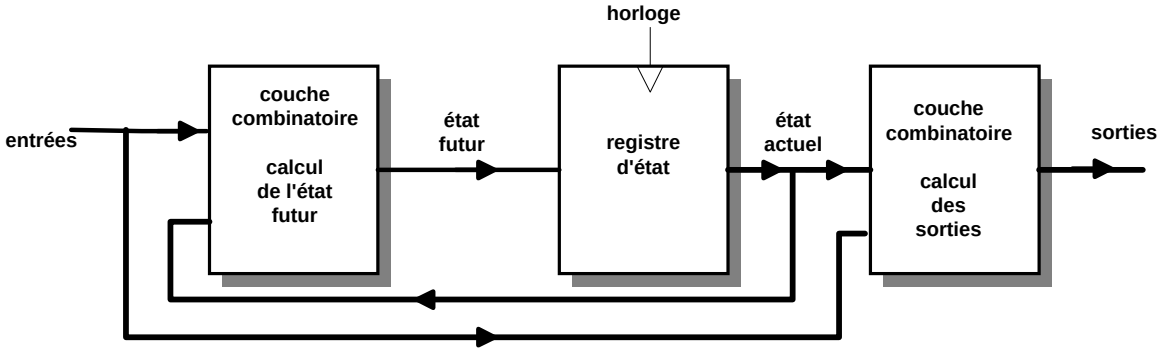
- affecter une sortie du système à chaque sortie du registre d'état (état codé avec les sorties), ce qui n'est pas toujours possible ; la couche combinatoire de calcul des sorties est alors supprimée, le temps entre le front d'horloge et l'évolution des sorties est optimal ;
- affecter une bascule par état dans le registre d'état (« one hot »), le nombre de bascules est alors généralement important, mais le décodage des sorties et surtout le calcul de l'état futur est facilité, permettant une fréquence de fonctionnement élevée ;

- minimiser le nombre de bascules (codage binaire) ; N bascules pouvant coder 2^N états ; les couches combinatoires sont alors complexes et le système lent.

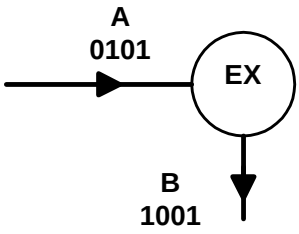
La solution choisie dépend de la manière dont va être écrit le programme.

Machine de Mealy

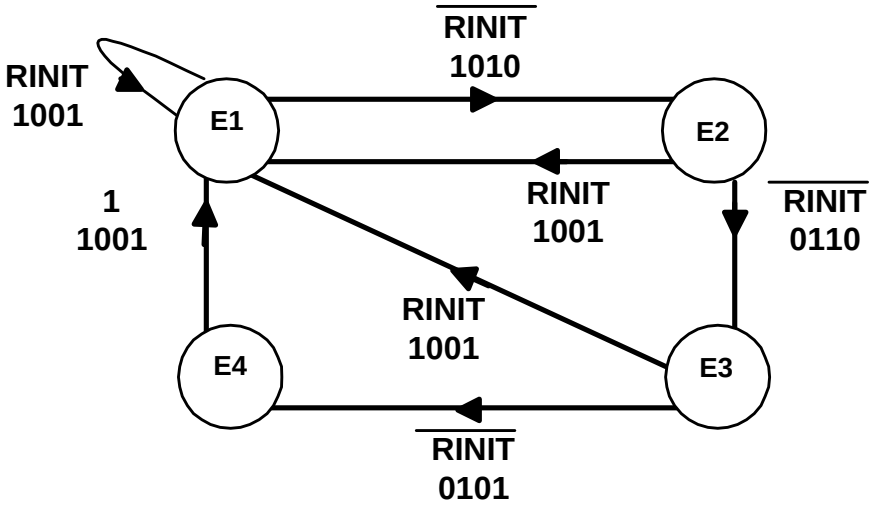
Dans certains cas, on peut souhaiter des sorties fonctionnant de manière asynchrones, la structure est alors modifiée pour conduire à une machine de type Mealy :



On modifie parfois le diagramme d'état de ce type de machine en précisant la valeur des sorties (0101 puis 1001 dans l'exemple) non plus dans la bulle d'état, mais dans la transition (ou vergent) avec la condition de passage (A puis B dans l'exemple).



On pourrait par exemple souhaiter une ré-initialisation asynchrone vers la valeur « 1001 » des sorties des phases de notre moteur par une entrée RINIT (l'entrée de sens à été supprimée afin de simplifier).

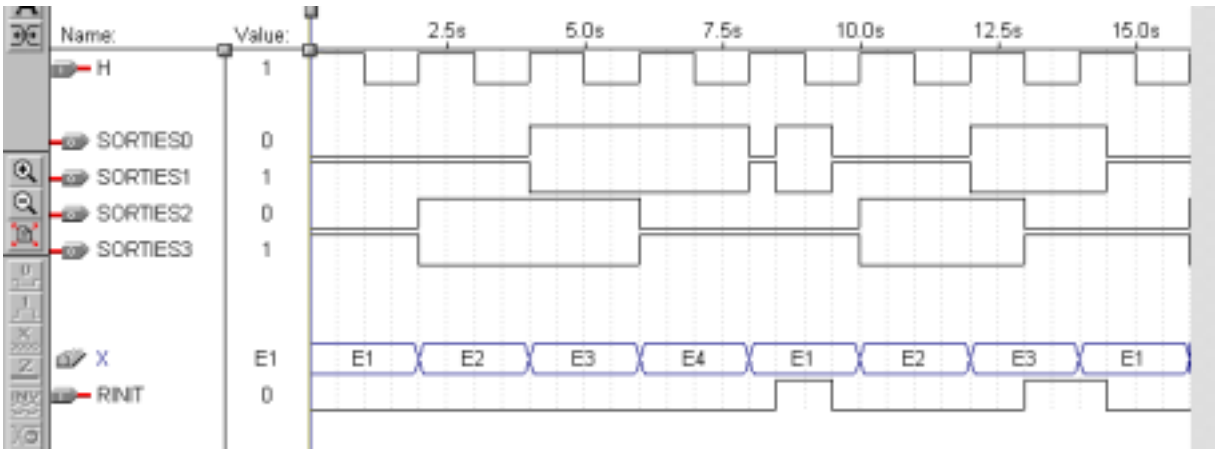


Les sorties ne dépendent plus cette fois de l'état actif, mais également de l'entrée RINIT.

Une réalisation possible étant donnée par le programme ci-après :

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

entity CDE_MOT_M4 is
        port(   H, RINIT: in    std_logic;
                SORTIES                 : out std_logic_vector (3 downto 0));
end CDE_MOT_M4;

architecture ARC_CDE OF CDE_MOT_M4 is
        type TYPE_ETAT is ( E1, E2, E3, E4);
        signal X:TYPE_ETAT;
begin
        process (H)
        begin
                if H'event and H = '1' then
                        case X is
                                when E1 =>      if RINIT='1'    then    X <= E1;
                                                                else    X <= E2;
                                                end if;

                                when E2 =>      if RINIT='1'    then    X <= E1;
                                                                else    X <= E3;
                                                end if;

                                when E3 =>      if RINIT='1'    then    X <= E1;
                                                                else    X <= E4;
                                                end if;

                                when others =>  X <= E1;
                        end case;
                end if;
        end process;

                SORTIES <=      "1010" when (X=E1 and RINIT='0') else
                                "0110" when (X=E2 and RINIT='0') else
                                "0101" when (X=E3 and RINIT='0') else
                                "1001";

end ARC_CDE;
```

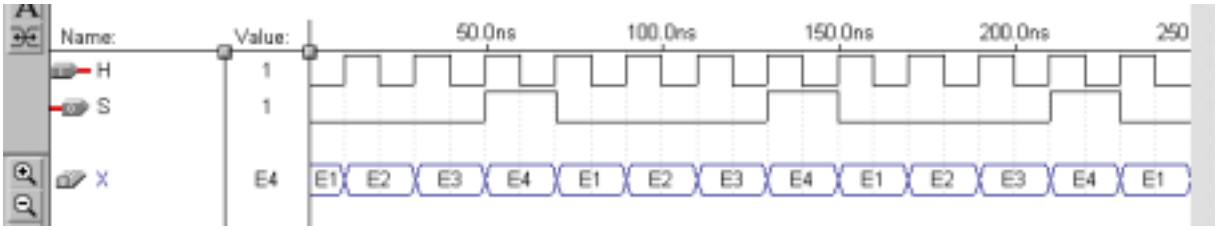
Sur les chronogrammes suivant, on note bien le caractère asynchrone des changements de valeurs en sortie lors de l'initialisation, avec des changements d'état qui restent par contre synchrones de l'horloge.



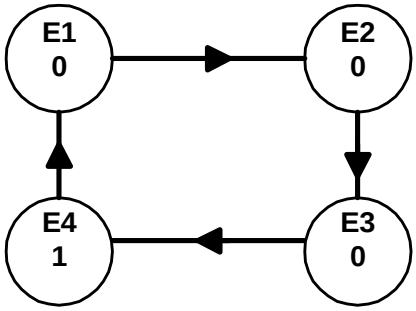
Proposer un séquenceur complet permettant le fonctionnement dans les deux sens de rotation, en mode pas entier une ou deux phases, ainsi qu'en mode demi-pas. On s'inspirera du fonctionnement d'un circuit spécialisé tel que le L297.

Exemple 3 : diviseur de fréquence.

Le nombre d'états d'un système n'est pas lié au nombre d'états possibles des sorties.
Considérons par exemple un diviseur de fréquence fournissant une impulsion en sortie toutes les quatre impulsions du signal d'horloge.



Pour synthétiser un tel diviseur, il ne suffit évidemment pas de prendre en compte les deux états possibles de la sortie, mais tous les états intermédiaires du système.



Un exemple de programme est donné ci-après :

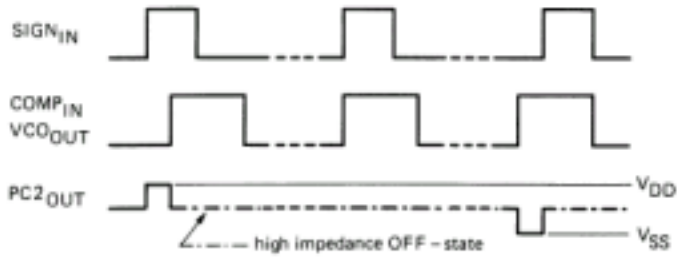
```
library ieee;
use ieee.std_logic_1164.all;

entity div4 is
        port(   H               : in    std_logic;
                S               : out std_logic);
end div4;

architecture ARC_div OF div4 is
        type TYPE_ETAT is (E1, E2, E3, E4);
        signal X:TYPE_ETAT;
begin
        process (H)
        begin
                        if H'event and H = '1' then
                        case X is
                                when E1 =>      X <= E2;
                                when E2 =>      X <= E3;
                                when E3 =>      X <= E4;
                                when others =>  X <= E1;
                        end case;
                end if;
        end process;
```

```
        S           <=      '1'     when X=E4 else '0';
end ARC_div;
```

Exemple 4 : comparateur 3 états d'une boucle à verrouillage de phase.

Cet exemple met en évidence la possibilité d'utiliser les machines d'état afin de synthétiser un système asynchrone, ici le comparateur 3 états d'une boucle à verrouillage de phase semi-numérique. Nous nous contenterons d'étudier le diagramme d'état, la transcription en VHDL étant fastidieuse.

Les chronogrammes suivants rappellent le fonctionnement :si les fronts des signaux d'entrée arrivent en même temps, la sortie $PC2_{OUT}$ du comparateur reste en haute impédance (pas de changement à l'entrée de l'OCT), si le front montant du signal référence $SIGN_{IN}$ arrive en premier, la sortie passe au niveau logique 1 jusqu'à l'arrivée du front montant du signal de contre réaction $COMP_{IN}$ (augmentation de la tension d'entrée de l'OCT), la sortie passant à 0 entre les deux fronts si le signal de contre réaction est en avance.



Le constructeur donne alors le diagramme d'état suivant :

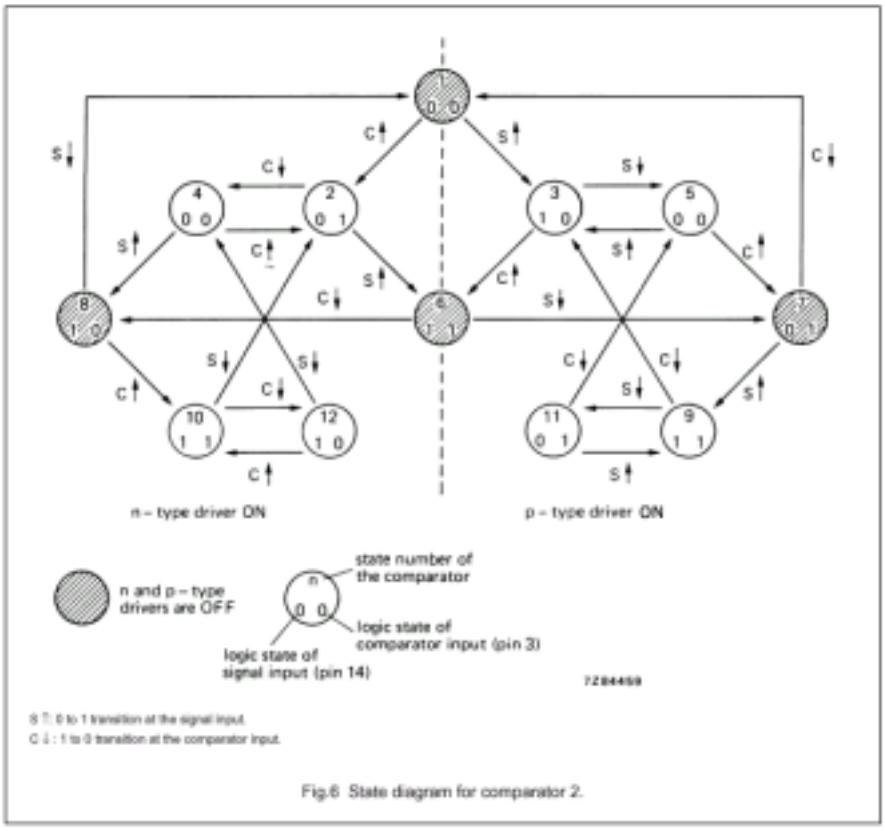


Fig.6 State diagram for comparator 2.

On notera que le constructeur s'est autorisé quelques liberté avec le formalisme habituel des diagrammes d'état. Les états hachurés correspondent à une sortie en haute impédance, les états blancs de gauche à un niveau logique 1 en sortie, ceux de droite à un niveau logique 0. Les changements d'état sont provoqués par les fronts montant des signaux S ($SIGN_{IN}$) et C ($COMP_{IN}$).

Bibliographie

Electronique numérique et séquentielle par N. Richard chez Dunod

Logique programmable par L. Dutrieux et D. Demigny chez Eyrolles

Modélisation de systèmes intégrés numériques par A. Vachoux sur :
http://lsmwww.epfl.ch/Site01OCT10/Courses/modelnum01.book.pdf