Rapport Travaux Pratique n°3
Séquenceur de moteur pas à pas

Zacharia Benkirane, Thomas Petazzoni

7 janvier 2003

1 Présentation

Il s'agit dans ce TP de réaliser la commande d'un moteur pas à pas de manière à ce que sa vitesse soit proportionnelle à une fréquence d'horloge.

On souhaite une rotation dans les deux sens du moteur par demi pas.

Pour tourner dans un sens, la séquence de commande doit être la suivante :

Etat	B3	B2	B1	B0
X_0	1	0	0	0
X_1	1	1	0	0
X_2	0	1	0	0
X_3	0	1	1	0
X_4	0	0	1	0
X_5	0	0	1	1
X_6	0	0	0	1
X_7	1	0	0	1

On souhaite 4 fonctionnements possibles en fonction de 3 variables d'entrées :

- **CW** : lorsque CW = 1, on a une rotation dans le sens des aiguilles d'une montre. Lorsque CW = 0, on a une rotation en sens inverse.
- **A** : lorsque A = 1, le moteur est à l'arret avec couple.
- **L** : lorsque L = 1, le moteur est libre (pas de commande, et donc pas de couple).

Lorsque L et à sont à 0 on est en rotation, lorsque A et L sont tous les deux à 1, le moteur est libre (L est prioritaire sur A).

2 Réalisation du graphe des états

On définit **9 états** :

- Les 8 états correspondants aux 8 commandes possibles pour le moteur (cf la séquence de commande présentée dans la section précédente). Ces états sont appelés X_0, X_1, X_2, X_3, X_4, X_5, X_6 et X_7, et leurs sorties sont celles données dans le tableau de la séquence de commande ci dessus.
- Un état correspondant à une sortie nulle (B3, B2, B1 et B0 à 0), c'est à dire au moteur libre (pas de couple). Cet état est appelé *X'*.

Explications concernant les différentes transitions :

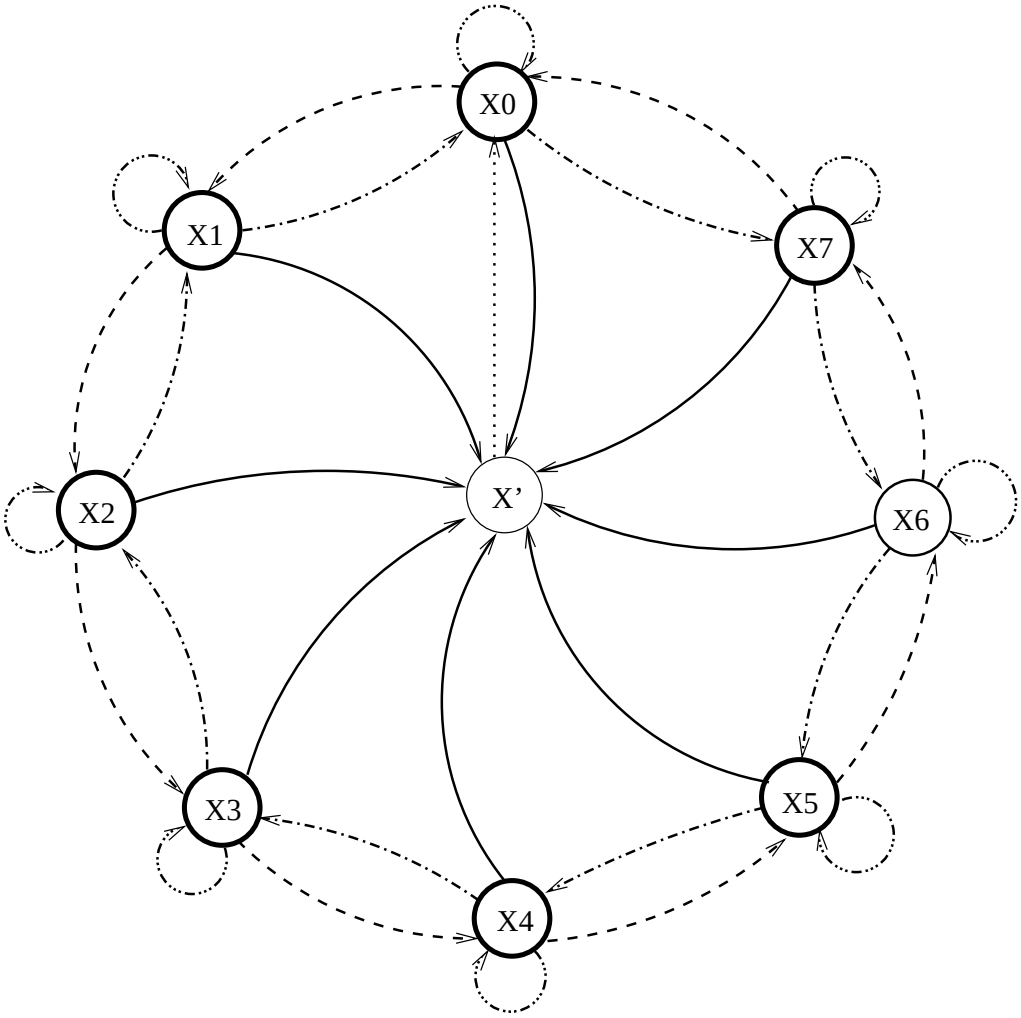
- Depuis tous les états X_i, on peut aller vers l'état *X'*, grâce à la transition L. En effet, depuis n'importe quel état, dès que *L=1*, on veut que le moteur soit libre.

- Depuis l'état *X'*, on peut revenir en l'état X_0 une fois que *L=0*, d'où la transition $\overline{L}$.

- Depuis tout état X_i, on reste dans cet état si *A=1*, mais que *L=0*, d'où les transitions-boucles $A.\overline{L}$ apparaissant sur chaque état X_i. On doit bien spécifier $A.\overline{L}$, et pas seulement A, car on veut que L soit prioritaire sur A.

- Depuis tout état X_i, on peut aller vers l'état X_{i+1} (avec boucle de X_7 à X_0) lorsque *CW=1* (rotation dans le sens des aiguilles d'une montre), mais que *A=0* et *L=0*, d'où les transitions $CW.\overline{L}.\overline{A}$.

- Depuis tout état X_i, on peut aller vers l'état X_{i-1} (avec boucle de X_0 à X_7) lorsque *CW=0* (rotation en sens inverse), et que *A=0* et *L=0*, d'où les transitions $\overline{CW}.\overline{A}.\overline{L}$.


X0
X1
X7
X2
X'
X6
X3
X5
X4
A.(not L)
CW.(not A).(not L)
(not CW).(not A).(not L)
L
(not L)

3 Implémentation en logique cablée

Nous avons choisi de réaliser l'implémentation en logique cablée de manière décodée : il y a quatres sorties *B0, B1, B2* et *B3*, nous utiliserons donc quatres bascules D, une pour chaque sortie.

3.1 L'entrée L

Nous allons tout d'abord nous occuper du signal L, qui est simple à implémenter. Lorsque L est au niveau haut, alors le moteur est libre, c'est à dire que toutes les sorties doivent être à 0. Il suffit donc de relier l'entrée L aux entrées *Reset* de toutes les bascules. Plus exactement, l'entrée L est active niveau haut, et les entrées *Reset* sont actives niveau bas, donc c'est $\overline{L}$ qu'il faut connecter aux entrées *Reset* des bascules.

3.2 Conditions de maintien et de transition

Pour chaque bascule, il faut étudier deux choses :
- Les conditions de maintien, lorsque A vaut 1.
- Les conditions de transition, lorsque A vaut 0.

Les conditions de maintien sont simple à implémenter : lorsque A vaut 1, la valeur à l'entrée de la bascule doit être égale à la sortie, afin que le moteur reste bloqué dans la position courante.

Si l'on appelle D_i les entrées des bascules, et Q_i les sorties, alors on exprimera les entrées des bascules de la manière suivante :

$$D_i = Q_i.A + (ConditionsDeTransition).\overline{A}$$

3.2.1 Conditions de transition pour la bascule 0

La bascule 0 correspond à la sortie B0. Lorsque A vaut 0, on doit faire évoluer l'état de la bascule pour l'amener dans l'état suivant ou l'état précédent en fonction de *CW*.

En particulier, pour la bascule 0, on doit passer à 1 ou rester à 1 lorsque :
- On est en X_4 et qu'on passe en X_5 (CW = 1)
- On est en X_5 et qu'on passe en X_6 (CW = 1)
- On est en X_5 et qu'on passe en X_7 (CW = 1)
- On est en X_0 et qu'on passe en X_7 (CW = 0)
- On est en X_7 et qu'on passe en X_6 (CW = 0)
- On est en X_6 et qu'on passe en X_5 (CW = 0)

On va simplifier séparement les cas CW=1 et CW=0 :

$\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0} + \overline{Q_3}.\overline{Q_2}.Q_1.Q_0 + \overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0$ $\overline{Q_3}.\overline{Q_2}.(\overline{Q_0}.Q_1 + Q_1.Q_0 + Q_0.\overline{Q_1})$ $\overline{Q_3}.\overline{Q_2}.(Q_1 + \overline{Q_1}.Q_0)$ $\overline{Q_3}.\overline{Q_2}.(Q_1 + Q_0)$	CW = 1
$\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0 + Q_3.\overline{Q_2}.\overline{Q_1}.Q_0 + Q_3.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}$ $\overline{Q_2}.\overline{Q_1}.(\overline{Q_3}.Q_0 + Q_3.Q_0 + Q_3.\overline{Q_0})$ $\overline{Q_2}.\overline{Q_1}.(Q_0 + Q_3.\overline{Q_0})$ $\overline{Q_2}.\overline{Q_1}.(Q_0 + Q_3)$	CW = 0

Au total, on a donc :

$$D_0 = A.Q_0 + \overline{A}.(CW.(\overline{Q_3}.\overline{Q_2}.(Q_1 + Q_0)) + \overline{CW}.(\overline{Q_2}.\overline{Q_1}.(Q_0 + Q_3)))$$

3.2.2 Conditions de transition pour la bascule 1

La bascule 1 correspond à la sortie B1. Lorsque A vaut 0, on doit faire évoluer l'état de la bascule pour l'amener dans l'état suivant ou l'état précédent en fonction de CW.

En particulier, pour la bascule 1, on doit passer à 1 ou rester à 1 lorsque :
- On est en X_2 et qu'on passe en X_3 (CW = 1)
- On est en X_3 et qu'on passe en X_4 (CW = 1)
- On est en X_4 et qu'on passe en X_5 (CW = 1)
- On est en X_6 et qu'on passe en X_5 (CW = 0)
- On est en X_5 et qu'on passe en X_4 (CW = 0)
- On est en X_4 et qu'on passe en X_3 (CW = 0)

On va simplifier séparement les cas CW=1 et CW=0 :

$\overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0} + \overline{Q_3}.Q_2.Q_1.\overline{Q_0} + \overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0}$ $\overline{Q_3}.\overline{Q_0}.(Q_2.\overline{Q_1} + Q_2.Q_1 + \overline{Q_2}.Q_1)$ $\overline{Q_3}.\overline{Q_0}.(Q_2 + \overline{Q_2}.Q_1)$ $\overline{Q_3}.\overline{Q_0}.(Q_2 + Q_1)$	CW = 1
$\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0 + \overline{Q_3}.\overline{Q_2}.Q_1.Q_0 + \overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0}$ $\overline{Q_3}.\overline{Q_2}.(\overline{Q_1}.Q_0 + Q_1.Q_0 + Q_1.\overline{Q_0})$ $\overline{Q_3}.\overline{Q_2}.(Q_0 + Q_1.\overline{Q_0})$ $\overline{Q_3}.\overline{Q_2}.(Q_0 + Q_1)$	CW = 0

Au total, on a donc :

$$D_1 = A.Q_1 + \overline{A}.(CW.(\overline{Q_3}.\overline{Q_0}.(Q_2 + Q_1)) + \overline{CW}.(\overline{Q_3}.\overline{Q_2}.(Q_0 + Q_1)))$$

3.2.3 Conditions de transition pour la bascule 2

La bascule 2 correspond à la sortie B2. Lorsque A vaut 0, on doit faire évoluer l'état de la bascule pour l'amener dans l'état suivant ou l'état précédent en fonction de CW.

En particulier, pour la bascule 2, on doit passer à 1 ou rester à 1 lorsque :
- On est en X_0 et qu'on passe en X_1 (CW = 1)
- On est en X_1 et qu'on passe en X_2 (CW = 1)
- On est en X_2 et qu'on passe en X_3 (CW = 1)
- On est en X_4 et qu'on passe en X_3 (CW = 0)
- On est en X_3 et qu'on passe en X_2 (CW = 0)
- On est en X_2 et qu'on passe en X_1 (CW = 0)

On va simplifier séparement les cas CW=1 et CW=0 :

$Q_3.\overline{Q_2}.\overline{Q_1}.\overline{Q_0} + Q_3.Q_2.\overline{Q_1}.\overline{Q_0} + \overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0}$ $\overline{Q_1}.\overline{Q_0}.(Q_2.\overline{Q_3} + Q_2.Q_3 + \overline{Q_2}.Q_3)$ $\overline{Q_1}.\overline{Q_0}.(Q_2 + \overline{Q_2}.Q_3)$ $\overline{Q_1}.\overline{Q_0}.(Q_2 + Q_3)$	CW = 1
$\overline{Q_3}.\overline{Q_2}.Q_1.\overline{Q_0} + \overline{Q_3}.Q_2.Q_1.\overline{Q_0} + \overline{Q_3}.Q_2.\overline{Q_1}.\overline{Q_0}$ $\overline{Q_3}.\overline{Q_0}.(\overline{Q_1}.Q_2 + Q_2.Q_1 + Q_1.\overline{Q_2})$ $\overline{Q_3}.\overline{Q_0}.(Q_2 + Q_1.\overline{Q_2})$ $\overline{Q_3}.\overline{Q_0}.(Q_2 + Q_1)$	CW = 0

Au total, on a donc :

$$D_2 = A.Q_2 + \overline{A}.(CW.(\overline{Q_1}.\overline{Q_0}.(Q_2 + Q_3)) + \overline{CW}.(\overline{Q_3}.\overline{Q_0}.(Q_2 + Q_1)))$$

3.2.4 Conditions de transition pour la bascule 3

La bascule 3 correspond à la sortie B3. Lorsque A vaut 0, on doit faire évoluer l'état de la bascule pour l'amener dans l'état suivant ou l'état précédent en fonction de CW.

En particulier, pour la bascule 3, on doit passer à 1 ou rester à 1 lorsque :
- On est en X_6 et qu'on passe en X_7 (CW = 1)
- On est en X_7 et qu'on passe en X_0 (CW = 1)
- On est en X_1 et qu'on passe en X_1 (CW = 1)
- On est en X_2 et qu'on passe en X_1 (CW = 0)
- On est en X_1 et qu'on passe en X_0 (CW = 0)
- On est en X_7 et qu'on passe en X_7 (CW = 0)

On va simplifier séparement les cas CW=1 et CW=0 :

$\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.Q_0 + Q_3.\overline{Q_2}.\overline{Q_1}.Q_0 + Q_3.\overline{Q_2}.Q_1.\overline{Q_0}$ $\overline{Q_2}.\overline{Q_1}.(Q_3.\overline{Q_0} + Q_0.Q_3 + \overline{Q_3}.Q_0)$ $\overline{Q_2}.\overline{Q_1}.(Q_3 + \overline{Q_3}.Q_0)$ $\overline{Q_2}.\overline{Q_1}.(Q_3 + Q_0)$	CW = 1
$\overline{Q_3}.\overline{Q_2}.\overline{Q_1}.\overline{Q_0} + Q_3.\overline{Q_2}.\overline{Q_1}.\overline{Q_0} + Q_3.\overline{Q_2}.\overline{Q_1}.\overline{Q_0}$ $\overline{Q_1}.\overline{Q_0}.(\overline{Q_2}.Q_3 + Q_2.Q_3 + Q_2.\overline{Q_3})$ $\overline{Q_1}.\overline{Q_0}.(Q_3 + Q_2.\overline{Q_3})$ $\overline{Q_1}.\overline{Q_0}.(Q_3 + Q_2)$	CW = 0

Au total, on a donc :

$$D_3 = A.Q_3 + \overline{A}.(CW.(\overline{Q_2}.\overline{Q_1}.(Q_3 + Q_0)) + \overline{CW}.(\overline{Q_1}.\overline{Q_0}.(Q_3 + Q_2)))$$

3.2.5 Un dernier ajout

Lorsque L a été mis à 1, toutes les bascules sont à 0, or aucun des états X_i ne correspondont à une sortie *0000*, et donc aucune transition ne permettra de faire "repartir" la machine à état. Il faut donc que lorsque toutes les bascules sont à 0 et que L est à 0, on repasse en l'état X_0, c'est à dire en placant B_3 à 1. On va donc modifier l'équation de D_3 de la manière suivante :

$$D_3 = A.Q_3 + \overline{A}.(CW.(\overline{Q_2}.\overline{Q_1}.(Q_3 + Q_0)) + \overline{CW}.(\overline{Q_1}.\overline{Q_0}.(Q_3 + Q_2))) + \overline{Q_3 + Q_2 + Q_1 + Q_0 + L}$$

3.3 Schéma général

Le schéma général de l'implémentation en logique cablée est disponible en annexe. Le schéma a été simplifié : les expressions comme $\overline{Q_0}.\overline{Q_2}$ ou $Q_1 + Q_0$ n'ont pas été cablées, et tous les fils n'ont pas été positionnés, le tout dans un souci de clarté.

4 Implémentation à l'aide d'une mémoire

Le mot mémoire est constitué de
- 3 bits désignants l'adresse suivante dans la mémoire (D0, D1, D2).
- 1 bit inutilisé (D3)
- 4 bits désignants la commande envoyée à l'interface de puissance pour la commande du moteur (D4, D5, D6, D7).

Le cablage proposé dans le TP est très astucieux : l'adresse mémoire courante est sélectionnée par deux choses. Les 3 bits de poids faible de l'adresse sont désignés par le micro programme lui même, tandis que les 3 bits de poids forts sont directement connectés aux entrées CW, L et A. Ceci facilite grandement la programmation de la mémoire.

Ainsi, on sait que lorsque *L=0, A=0 et CW=0*, on se trouve entre les adresses 0x00 et 0x07 dans la mémoire (rotation dans le sens inverse). Dans cette plage on code simplement les 4 bits de poids fort en recopiant à l'envers le tableau des états, et pour les 3 bits de poids faible on donne l'adresse suivante pour avancer dans le graphe.

Lorsque *L=0,A=0, CW=1*, on se trouve entre les adresses 0x08 et 0x0F dans la mémoire (rotation dans le sens des aiguilles d'une montre). Dans cette plage on code simplement les 4 bits de poids fort en recopiant le tableau des états (dans le bon sens), et pour les 3 bits de poids faible on donne l'adresse suivante pour avancer dans le graphe.

Lorsque *L=0, A=1, CW=0*, on se trouve entre les adresses 0x10 et 0x17 dans la mémoire (bloquage du moteur dans la position courante). Dans cette plage, on recopie les états du tableau (en sens inverse, puisqu'on doit tourner dans le sens inverse des aiguilles d'une montre), mais l'adresse sur 3 bits est identique à l'adresse courante, pour que l'on puisse rester bloqué sur cet état.

De même pour *L=0,A=1 et CW=1*, on se trouvera entre les adresses 0x18 et 0x1F (bloquage du moteur dans la position courante). Même principe que précédemment, sauf qu'on recopie le tableau dans le sens normal (rotation dans le sens des aiguilles d'une montre).

Enfin, lorsque *L=1*, on se retrouve entre les adresses 0x20 et 0x3F. Lorsque *L=1*, on souhaite que le moteur soit completement libre, c'est pourquoi cette plage d'adresses est entièrement remplie de zéros.

Addresse	Val (binaire)	Val (hex)	Addresse	Val (binaire)	Val (hex)
0x00	0110 0001	0x61	0x20	0000 0000	0x00
0x01	0100 0010	0x42	0x21	0000 0000	0x00
0x02	1100 0011	0xC3	0x22	0000 0000	0x00
0x03	1000 0100	0x84	0x23	0000 0000	0x00
0x04	1001 0101	0x95	0x24	0000 0000	0x00
0x05	0001 0110	0x16	0x25	0000 0000	0x00
0x06	0011 0111	0x37	0x26	0000 0000	0x00
0x07	0010 0000	0x20	0x27	0000 0000	0x00
0x08	0010 0000	0x21	0x28	0000 0000	0x00
0x09	0011 0010	0x32	0x29	0000 0000	0x00
0x0A	0001 0011	0x13	0x2A	0000 0000	0x00
0x0B	1001 0100	0x94	0x2B	0000 0000	0x00
0x0C	1000 0101	0x85	0x2C	0000 0000	0x00
0x0D	1100 0110	0xC6	0x2D	0000 0000	0x00
0x0E	0100 0111	0x47	0x2E	0000 0000	0x00
0x0F	0110 0000	0x60	0x2F	0000 0000	0x00
0x10	0110 0000	0x60	0x30	0000 0000	0x00
0x11	0100 0001	0x41	0x31	0000 0000	0x00
0x12	1100 0010	0xC2	0x32	0000 0000	0x00
0x13	1000 0011	0x83	0x33	0000 0000	0x00
0x14	1001 0100	0x94	0x34	0000 0000	0x00
0x15	0001 0101	0x15	0x35	0000 0000	0x00
0x16	0011 0110	0x36	0x36	0000 0000	0x00
0x17	0010 0111	0x27	0x37	0000 0000	0x00
0x18	0010 0000	0x20	0x38	0000 0000	0x00
0x19	0011 0001	0x31	0x39	0000 0000	0x00
0x1A	0001 0010	0x12	0x3A	0000 0000	0x00
0x1B	1001 0011	0x93	0x3B	0000 0000	0x00
0x1C	1000 0100	0x84	0x3C	0000 0000	0x00
0x1D	1100 0101	0xC5	0x3D	0000 0000	0x00
0x1E	0100 0110	0x46	0x3E	0000 0000	0x00
0x1F	0110 0111	0x67	0x3F	0000 0000	0x00

5 Programmation VHDL

La programmation en VHDL d'un tel séquenceur ne pose pas de problème particulier. Nous avons choisi d'utiliser des états codés (d'où le *type* `type_etat`. Les etats *x0, x1, x2, x3, x4, x5, x6* et *x7* correspondent à ceux du graphe, tandis que l'état *x8* correspond à l'état *x'* dans le graphe.

A chaque battement d'horloge on regarde tout d'abord si *L=1*, dans ce cas on passe directement dans l'état *x8*. Sinon si *A=1*, l'état reste identique à lui même. Enfin en fonction de *CW* et de l'état courant, on détermine l'état suivant, que ce soit dans le sens normal, ou dans le sens inverse.

Après le *process*, un *with* sert à décoder les états afin de produire les sorties adaptées.

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

entity moteur is

port( cw,a,l,clk : in std_logic;
                b: out std_logic_vector (3 downto 0));
end moteur;

architecture a of moteur is
type type_etat is (x0,x1,x2,x3,x4,x5,x6,x7,x8);
signal etat : type_etat;
begin
        process (clk)
        begin

        if(clk'event and clk='1') then
                if(l='1') then etat <= x8;
                elsif (a='1') then etat <= etat;
                -- Sens normal
                elsif (cw='1') then
                        case etat is
                                when x0 => etat <= x1;
                                when x1 => etat <= x2;
```

```
                        when x2 => etat <= x3;
                        when x3 => etat <= x4;
                        when x4 => etat <= x5;
                        when x5 => etat <= x6;
                        when x6 => etat <= x7;
                        when x7 => etat <= x0;
                        when x8 => etat <= x0;
                end case;
        -- Sens inverse
        else
                case etat  is
                        when x0 => etat <= x7;
                        when x1 => etat <= x0;
                        when x2 => etat <= x1;
                        when x3 => etat <= x2;
                        when x4 => etat <= x3;
                        when x5 => etat <= x4;
                        when x6 => etat <= x5;
                        when x7 => etat <= x6;
                        when x8 => etat <= x0;
                end case;
        end if;
end if;
end process;

with etat  select
        b <= "1000" when x0,
                "1100" when x1,
                "0100" when x2,
                "0110" when x3,
                "0010" when x4,
                "0011" when x5,
                "0001" when x6,
                "1001" when x7,
                "0000" when x8;
end a;
```

Conclusion

Ce TP nous a permis de réaliser l'implémentation d'un séquenceur pour moteur pas à pas en utilisant différentes méthodes :

- En logique cablée
- A l'aide d'une mémoire
- A l'aide d'un circuit, programmé à partir du langage VHDL

Comparaison des solutions

En terme de prix, la première solution semble être la moins couteuse : les circuits intégrés comportant de nombreuses portes logiques et des bascules sont peu chers. Cependant, la solution est difficile et longue à concevoir et peu évolutive. La moindre modification à apporter amènerait à reconsidérer presque l'ensemble du schéma.

Les deux solutions suivantes sont certainement plus couteuses, et nécessitent l'utilisation d'un ordinateur pour programmer la mémoire ou la puce. Le codage du micro-programme est assez long et fastidieux, et la aussi toute évolution du séquenceur pour le moteur amènerait à reconsidérer tout le cablage de la mémoire. Cette solution est donc assez fastidieuse et peu évolutive.

La dernière solution, programmer en VHDL, puis flasher une puce avec ce code (une fois synthétisé) semble être la plus efficace et la plus evolutive. Le langage VHDL est clair, simple, et on peut facilement l'améliorer et le comprendre dès la première lecture. Les contraintes sont donc beaucoup moins grandes qu'avec les deux autres solutions.

Controle de la vitesse du moteur

Le principe du moteur pas à pas consiste à alimenter à tour de rôle les bobines du moteur. Ainsi, pour obtenir un fonctionnement par demi-pas, on alimente alternativement une bobine simple et deux bobines adjacentes.

Pour contrôler approximativement la vitesse du moteur à l'aide d'une horloge à 1 Khz, il suffit de réaliser une division d'horloge. Ceci permettra de ramener la fréquence de 1 Khz à la fréquence souhaitée pour la rotation du moteur (entre 32 et 500 demi-pas par seconde). On utilisera pour cela un compteur qui s'incrémente à chaque front montant. Par exemple, si l'on souhaite avoir une vitesse de 250 demi-pas par seconde, on utilisera un compteur par 4. Dès que le compteur aura atteint la valeur 3, il enverra 1 sur sa sortie, le reste du temps il enverra 0. Ce contrôle de vitesse peut donc être implanté dans le circuit programmable à l'aide d'une séquence VHDL.