TP2: Simulation et programmation de PLD

Les objectifs de ce TP sont :

- l'introduction aux composants de type "PLD" et à leur programmation ;
- l'utilisation d'un manuel de référence pour trouver l'information nécessaire ;
- l'introduction aux machines d'états.

1 Introduction

Un PLD (*Programmable Logic Device*) est un composant qu'on va pouvoir programmer selon les besoins d'une application spécifique. Il est livré vierge et il faut le programmer avant utilisation. Ceci peut être fait soit par un programmateur dédié (sous la forme d'un logiciel sur PC associé à un boitier externe dans lequel on place le composant), soit directement sur la carte applicative finale : on parle alors de *In-System Programming* (ISP), via une interface "JTAG".

La saisie des spécifications se fait via un fichier texte dans lequel on peut donner soit les expressions booléennes des sorties, soit directement la table de vérité. Ce fichier est ensuite "passé" comme entrée à un logiciel qui va générer le fichier de programmation au format normalisé **JEDEC**. Ce dernier sera utilisé pour programmer physiquement le composant ou le simuler.

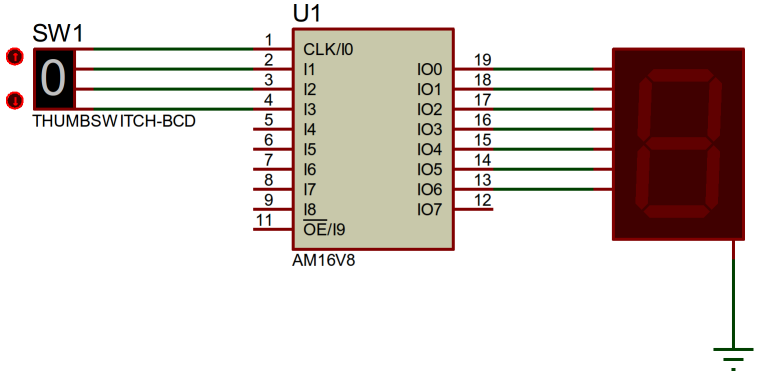
Les PLD peuvent implémenter des fonctions combinatoires ou séquentielles, et leur complexité est variable suivant les modèles : de 10 entrées/8 sorties comme celui que vous allez utiliser, jusqu'à plusieurs centaines de broches, utilisables comme entrées ou sorties.
Ces dernières versions sont désignées en général par les termes CPLD (*Complex Programmable Logic Device*) ou FPGA (*Field-Programmable Gate Array*).

Un CPLD de Xilinx, contenant 400 000 opérateurs élementaires.

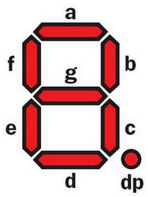
Dans ce TP, vous allez utiliser un PLD (virtuel) pour implémenter un décodeur BDC- 7 Segments et une machine d'états. Il faudra générer le fichier JEDEC et l'associer au composant placé sur le plan de travail.

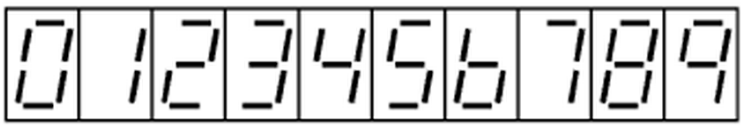
2 Décodeur BCD - 7SEG

Créer un nouveau projet sous ISIS, et le **sauvegarder immédiatement** dans le dossier **`T:\M1103\TP2`**. Faire la saisie du schéma ci-dessous. Le PLD se trouve dans la bibliothèque *PLDs & FPGAs*.



Cahier des charges On désire créer un dispositif permettant de gérer un afficheur 7 segments. Un nombre N de 4 bits (n_3, n_2, n_1, n_0) est envoyé au dispositif qui génère en sortie 7 variables (de a à g) afin de commander l'afficheur. On souhaite obtenir sur l'afficheur la valeur décimale de N. Le nombre N ne sera jamais > 9.





Compléter la table de vérité du dispositif :

N	n_3	n_2	n_1	n_0	a	b	c	d	e	f	g
0	0	0	0	0							
1	0	0	0	1							
2	0	0	1	0							
3	0	0	1	1							
4	0	1	0	0							
5	0	1	0	1							
6	0	1	1	0							
7	0	1	1	1							
8	1	0	0	0							
9	1	0	0	1							

Programmation sous Wincupl : Lancer Wincupl et créer un nouveau projet avec les paramètres suivants :

- Nom : 7seg
- Designer : Votre nom
- Device : g16v8
- Nombre d'entrées : 4
- Nombre de sorties : 7
- Nombre de pinnodes : 0

Compléter le fichier afin de définir :

- Les noms des entrées et leur numéro de broche (par exemple : **`PIN 1 = n0;`**),
- Les noms des sorties et leur numéro de broche (par exemple : **`PIN 19 = a;`**),
- La table de vérité du dispositif. Voir l'exemple de déclaration ci-contre, le **`'b'`** indiquant une valeur en binaire.
  Par exemple : **`'b'0000 => 'b'01010101`**

```
/* *************** OUTPUT PINS *******
PIN    X  =  A                        ;
PIN    X  =  B                        ;
PIN    X  =  C                        ;
PIN    X  =  D                        ;
PIN    X  =  E                        ;
PIN    X  =  F                        ;
PIN    X  =  G                        ;


FIELD entrees = [n3,n2,n1,n0];
FIELD sorties = [A,B,C,D,E,F,G];

TABLE entrees => sorties {
       'b'XXXX => 'b'XXXXXXX;
       ... à compléter ...
```

Immédiatement, sauvegardez ce fichier en **`TP2_A.pld`** dans votre dossier **`T:\M1103\TP2`**.

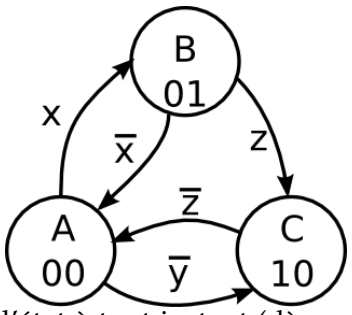
Compiler le programme (F9, ou bouton "*Device Dependent Compile*") en corrigeant les éventuelles erreurs de syntaxe. Ensuite, sous ISIS, charger le fichier JEDEC généré par WinCupl dans le composant et vérifier le bon fonctionnement de votre dispositif.

3 Machine d'états

3.1 Introduction

Une **machine d'états** est un automate qui ne peut se trouver que dans un nombre d'états finis. Chaque état est identifié par une valeur binaire unique, et on va passer d'un état à un autre si une certaine condition sur les entrées est remplie.

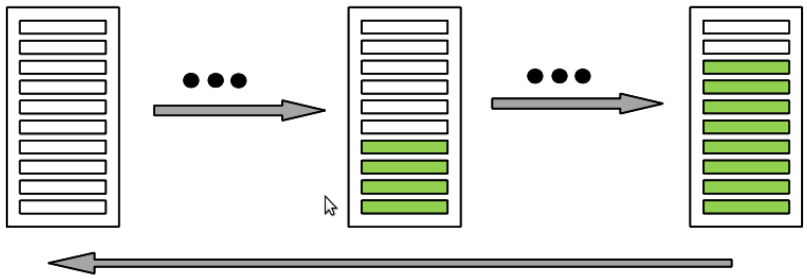
Par exemple, ci-contre la représentation d'une machine à 3 états (A, B, C) qui sont encodés par des valeurs sur 2 bits (A :00, B :01, C :10). On passera de l'état A à B si $x = 1$, et de B à A si $x = 0$. Par contre, si la machine se trouve dans l'état C, alors un changement de x n'aura aucune influence : il faudra obligatoirement avoir $z = 0$ pour retourner à l'état A.
De même, si on se trouve dans l'état A, alors un changement de z n'aura aucune influence.



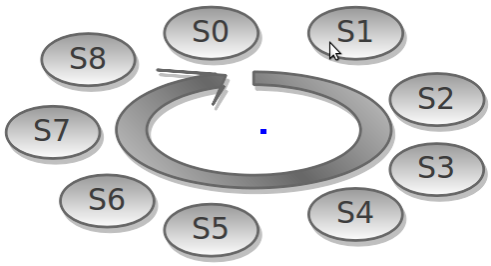
On distingue les machines d'état **asynchrones**, qui peuvent changer d'état à tout instant (dès que la condition est remplie) et les machines d'état **synchrones**, qui ne changeront d'état que sur les fronts d'horloge. Cette modélisation trouve son application dans de nombreuses situations réelles (feux de circulation, portillons, etc...)

3.2 Cahier des charges

On désire créer un dispositif permettant de gérer un bargraphe. Un signal d'horloge cadencé à 5Hz fait évoluer une machine d'état synchrone qui dispose de 8 sorties (Q0..Q7). On souhaite obtenir les affichages suivants :



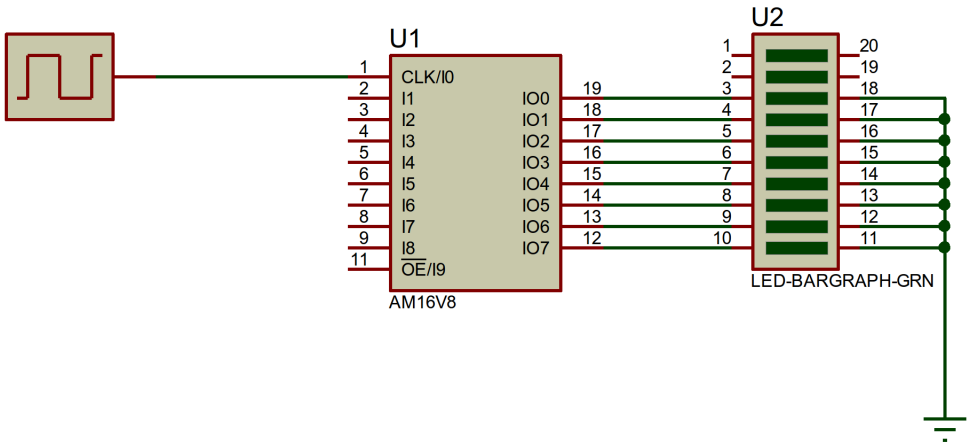
Ci-dessous le graphe des états correspondants : il n'y a aucune condition, les changements se feront lors des transitions d'horloge. Compléter la table de verité suivante et donner le code hexa. Les 9 états (S0 à S8) seront encodés directement par les broches sur lesquelles sont connectées les dels. Par exemple, l'état S0 correspond à 8 zeros, et l'état S1 par une seule sortie à "1".



Etat	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	code hexa
S0									
S1									
S2									
S3									
S4									
S5									
S6									
S7									
S8									

3.3 Travail à effectuer

Faire la saisie du schéma ci-dessous sous ISIS.



Programmation sous Wincupl :

1. Lancer Wincupl et créer un nouveau projet avec les paramètres adéquats (toujours un g16v8 comme *device*). Le sauvegarder en **`T:\M1103\TP2\TP2_B1.pld`**

2. Compléter le fichier afin de définir :
   - Les noms des 8 sorties et leur numéro de broche (pour l'entrée d'horloge, celle-ci étant obligatoirement sur la broche CLK, il sera inutile de le spécifier, c'est implicite).
   - La machine d'état (**consulter l'aide** pour voir la syntaxe : menu *Help* → *CUPL Programmers Reference Guide*, section 1.1.30, *State Machine Syntax*).

   Il faudra prédéfinir les 9 états avec neuf lignes avec la syntaxe suivante (le 'h' indique la notation hexa) :
   ```
   $DEFINE S0 'h'00
   $DEFINE S1 'h'01
   ...
   ```
   Ici, les transitions entre états ne seront pas liés à une variable d'entrée mais à une période d'horloge : chaque front d'horloge devra faire passer à l'état suivant (sauf pour l'état S8 pour lequel le suivant sera l'état S0). Ceci se fera avec une succession de PRESENT/NEXT (Voir exemples dans l'aide).

3. Compiler le programme (F9) en corrigeant les éventuelles erreurs de syntaxe.

4. Sous ISIS, charger le fichier JEDEC dans le composant ("Editer propriétés", puis "JEDEC FuseMap File") et faire valider par l'enseignant le fonctionnement de votre dispositif.

5. Ajouter ensuite sur le schéma une entrée connectée à un "LOGIC STATE" (voir TP1).
   Dupliquez le fichier de programmation (.pld) en **`TP2_B2.pld`**. Dans ce fichier, modifier la machine d'état de façon à ce que le sens d'évolution dépende du niveau logique de cette entrée. Vous utiliserez la structure suivante (voir doc. en ligne) :
   ```
   IF condition1 NEXT state_n1;
   ```