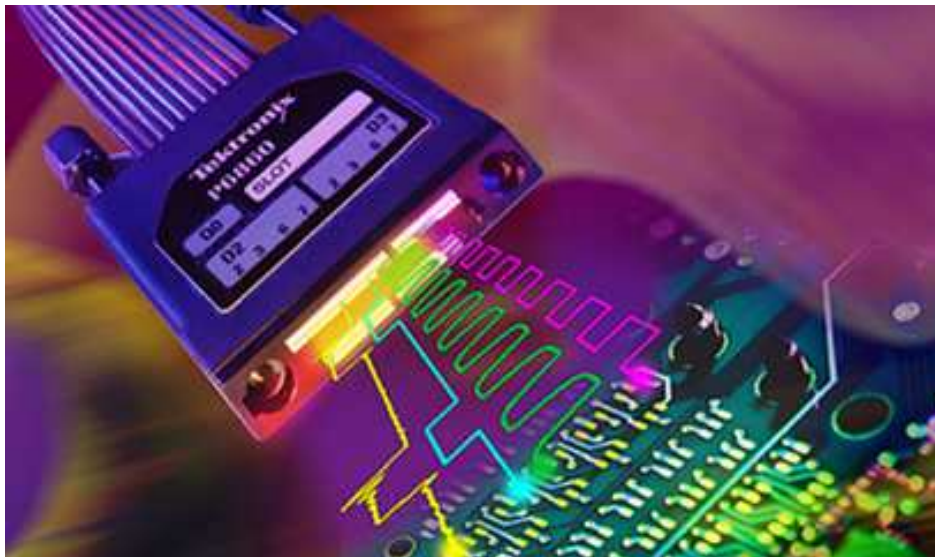

CIRCUITS COMBINATOIRES

Initiation et conceptions



Pascal GAGNAT, EISTI.
Cergy, France.

16 décembre 2004

Résumé

Le projet porte sur les composants de **Circuits Intégrés - IC**. Il nécessite des connaissances de base en électronique (**ICs**), en programmation bas niveau (**VHDL**), et en logique combinatoire.

L'objectif du projet consiste à réaliser un circuit combinatoire permettant d'afficher des chiffres sur un afficheur à partir de **Medium Scale Integration - MSI** spécifiques. Le projet nécessite une étude préalable sur l'ensemble du circuit en vue d'une simulation sous **MAX+plus II Baseline©** et d'une réalisation pratique du circuit à partir d'une plaquette de composants physiques.

La formalisation du problème sera énoncée dans la **Section 2**. La proposition des solutions se trouvera dans la **Section 3** et l'analyse du projet en **Section 4**. S'en suivra une démonstration du comportement des composants sous **MAX+plus II Baseline©** via le code **VHDL** associé au composant et ses flots de données en **Section 5** et **Section 6**. Et enfin une conclusion sur l'ensemble du projet en **Section 7**.

1 Introduction

Les cinquante dernières années de développement ont entraînées une augmentation spectaculaire des performances de processeurs et par la même occasion, une réduction spectaculaire de leur prix. Durant cette période de progrès continu dans la rentabilité des ordinateurs, le principe de logique combinatoire n'a jamais changé. La plupart des améliorations ont permies d'avancer dans la technologie de l'électronique, utilisation de lampes pour commencer, de transistors individuelles, d'**ICs** incorporant plusieurs transistors bipolaires. Puis, à travers la génération de la technologie **IC**, on se retrouve aujourd'hui avec des **Very Large Scale Integration - VLSI**, des circuits délivrant des millions de transistors sur une simple puce. Les transistors sont devenus aujourd'hui plus petits, moins cher, plus rapide, et consomment moins de courant. Cette histoire a porté l'industrie des ordinateurs en avant pour les trois dernières décennies et continuera encore durant les prochaines années.

Un **IC** est un circuit combinatoire dont la fonction de sortie s'exprime par une expression logique des seules variables d'entrées. Les circuits logiques, travaillent exclusivement avec deux signaux, le **1** et le **0** logique. Contrairement aux circuits analogiques qui eux travaillent avec des signaux électriques de toutes valeurs. Généralement le **0** logique est égal à 0 Volt et le **1** logique est égal à 5 volts, pour la logique positive. Les circuits d'ordinateurs, et de machines *intelligentes* fonctionnent avec des circuits logiques, car leur interprétation est relativement facile et ne permette pas d'ambiguïtés comparativement à des signaux analogiques.

Dans le projet, des circuits combinatoires de type **MSI** intervenant dans la réalisation des composants logiques d'un ordinateur comme un multiplexeur, un démultiplexeur, un inverseur, un décodeur et un afficheur sept segments seront utilisés.

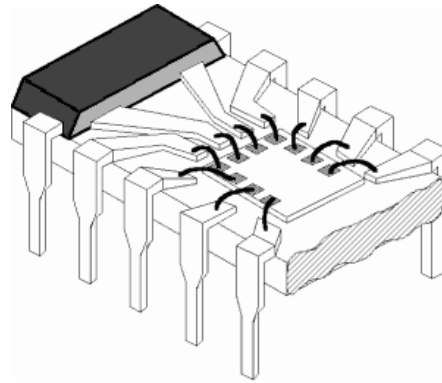


FIG. 1 – Représentation d'un composant TTL.

Le support de travail proposé est basé sur la technologie **Transistor to Transistor Logic - TTL** de la famille **Low Power Schottky - LS**. Un circuit logique de type **TTL** est donnée par la **Figure 1**.

Les **TTLs** sont en concurrence directe avec les **Complementary Metal Oxyde Semiconductor - CMOS**, de plus en plus utilisés sur le marché. Le CMOS requière une tension d'alimentation moindre (3V minimum contre 5V minimum pour les **TTLs**). La consommation d'énergie d'un CMOS au repos est inférieure de plusieurs puissances de dix à celle de n'importe quelle technologie concurrente. Il possède l'avantage d'être peu sensible aux perturbations électro-magnétiques. Le CMOS permet aussi d'atteindre une complexité de type **VLSI**. La vitesse de fonction d'un CMOS est environ trois à six fois plus rapide que celle d'une **TTL**.

Les principaux avantages d'un composant **TTL** de type **LS** par rapport à un CMOS résident dans sa plus grande tolérance au tension, à une faible dissipation de chaleur, et à un prix plus abordable.

2 Formalisation du problème

Les **MSI** que l'on utilise ont été conçus à partir de la logique Booléenne où un fil n'a qu'une de ces 2 valeurs : **1** ou **0**. Une tension de **0V** correspond à un **0** alors qu'une tension de **1V** correspond à un **1** pour la logique positive.

Leur applications sont régies par un ensemble de portes logiques **ET**, **OU**, **NON**, **AND**, **OR**, **XOR**. Ces portes logiques produisent une fonction d'une ou plusieurs variables d'entrées et une ou plusieurs variables de sorties.

L'interconnexion de ces portes logiques permet de réaliser le circuit illustré par la **Figure 2**.

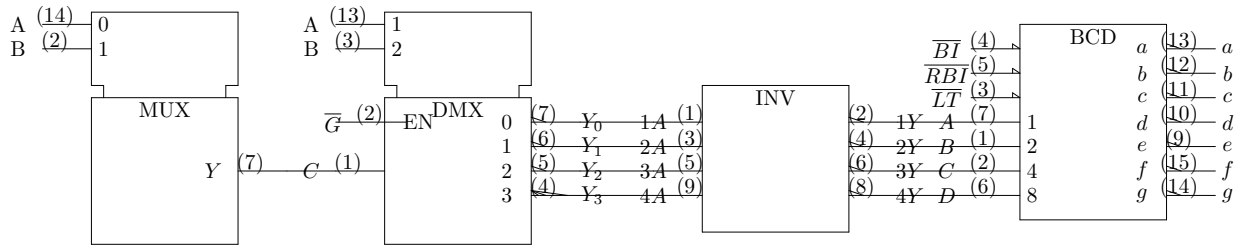


FIG. 2 – Schéma global du circuit à réaliser.

Ce circuit reçoit donc 4 signaux en entrée et 7 signaux en sortie. Le fonctionnement du circuit est basé sur la **Table 1**.

A	B	C	D	S_1	S_0	Afficheur
0	X	X	X	0	0	0
1	X	X	X	0	0	1
X	0	X	X	0	1	0
X	1	X	X	0	1	2
X	X	0	X	1	0	0
X	X	1	X	1	0	4
X	X	X	0	1	1	0
X	X	X	1	1	1	8

TAB. 1 – Table logique du circuit à réaliser.

Le circuit de la **Figure 2** se décompose en 5 parties correspondant à 5 **MSI** : le multiplexeur, le démultiplexeur, l'inverseur, le décodeur et l'afficheur 7 segments. **Section 3**.

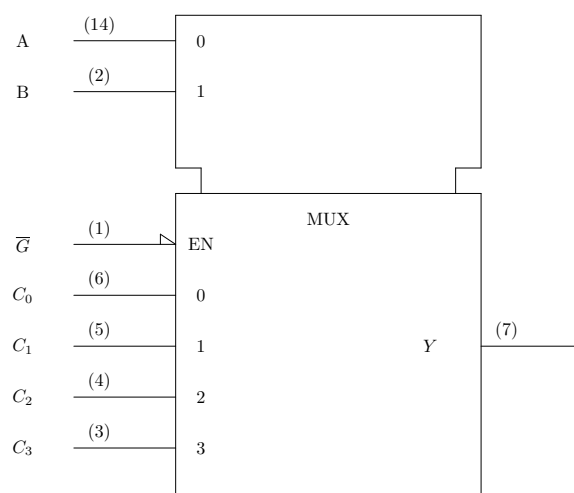


FIG. 3 – Schéma du multiplexeur **SN74LS153**.

La **Figure 3** représente le multiplexeur et la **Table 2** son fonctionnement interne. L'ensemble des schémas présentés sont conformes à la norme **ANSI/IEEE 91-1984**.

Select Inputs		Data Inputs				Strobe	Output
B	A	C_0	C_1	C_2	C_3	\overline{G}	Y
X	X	X	X	X	X	1	0
0	0	0	X	X	X	0	0
0	0	1	X	X	X	0	1
0	1	X	0	X	X	0	0
0	1	X	1	X	X	0	1
1	0	X	X	0	X	0	0
1	0	X	X	1	X	0	1
1	1	X	X	X	0	0	0
1	1	X	X	X	1	0	1

TAB. 2 – Table logique du multiplexeur **SN74LS153**.

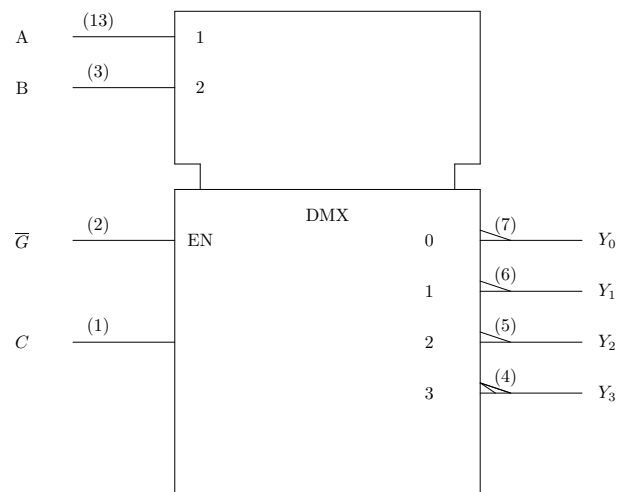


FIG. 4 – Représentation du démultiplexeur **SN74LS155a**.

La **Figure 4** représente le multiplexeur et la **Table 3** son fonctionnement interne.

Inputs			Outputs			
Select	Strobe	Data	Y_0	Y_1	Y_2	Y_3
B A	\overline{G}	C				
X X	1	X	1	1	1	1
0 0	0	0	0	1	1	1
0 1	0	0	1	0	1	1
1 0	0	0	1	1	0	1
1 1	0	0	1	1	1	0
X X	X	1	1	1	1	1

TAB. 3 – Table logique du démultiplexeur **SN74LS155a**.

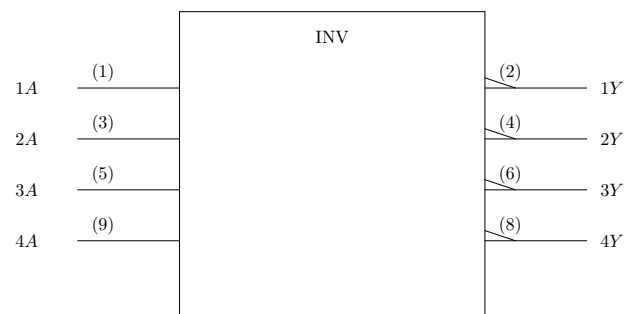


FIG. 5 – Représentation de l'inverseur **SN74LS04**.

La **Figure 5** représente l'inverseur et la **Table 4** son fonctionnement interne.

Input	Output
A	B
1	0
0	1

TAB. 4 – Table logique de l'inverseur **SN74LS04**.

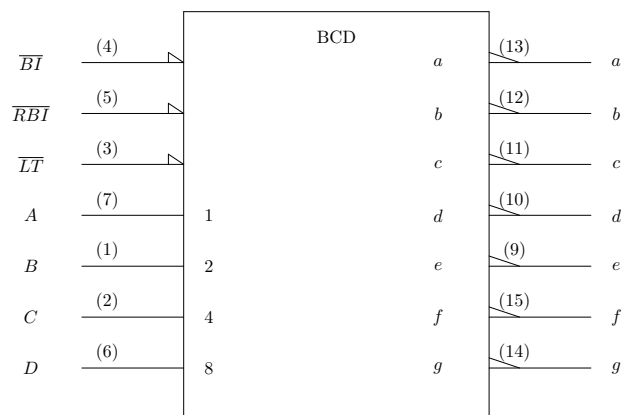


FIG. 6 – Représentation du décodeur **SN74LS48**.

La **Figure 6** représente le décodeur et la **Table 5** son fonctionnement interne.

Decimal	Input			$\overline{BI}/\overline{RBO}$	Output	Note
	\overline{LT}	\overline{RBI}	D C B A		a b c d e f g	
0	1	1	0 0 0 0	1	1 1 1 1 1 1 0	
1	1	X	0 0 0 1	1	0 1 1 0 0 0 0	
2	1	X	0 0 1 0	1	1 1 0 1 1 0 1	
3	1	X	0 0 1 1	1	1 1 1 1 0 1 1	
4	1	X	0 1 0 0	1	0 1 1 0 0 1 1	
5	1	X	0 1 0 1	1	1 0 1 1 0 1 1	
6	1	X	0 1 1 0	1	0 0 1 1 1 1 1	
7	1	X	0 1 1 1	1	1 1 1 0 0 0 0	
8	1	X	1 0 0 0	1	1 1 1 1 1 1 1	
9	1	X	1 0 0 1	1	1 1 1 0 0 1 1	
10	1	X	1 0 1 0	1	0 0 0 1 1 0 1	
11	1	X	1 0 1 1	1	0 0 1 1 0 0 1	
12	1	X	1 1 0 0	1	0 1 0 0 0 1 1	
13	1	X	1 1 0 1	1	1 0 0 1 0 1 1	
14	1	X	1 1 1 0	1	0 0 0 1 1 1 1	
15	1	X	1 1 1 1	1	0 0 0 0 0 0 0	
BI	X	X	X X X X	0	0 0 0 0 0 0 0	
RBI	1	0	0 0 0 0	0	0 0 0 0 0 0 0	
LT	0	X	X X X X	1	1 1 1 1 1 1 0	

TAB. 5 – Table logique du décodeur **SN74LS48**.

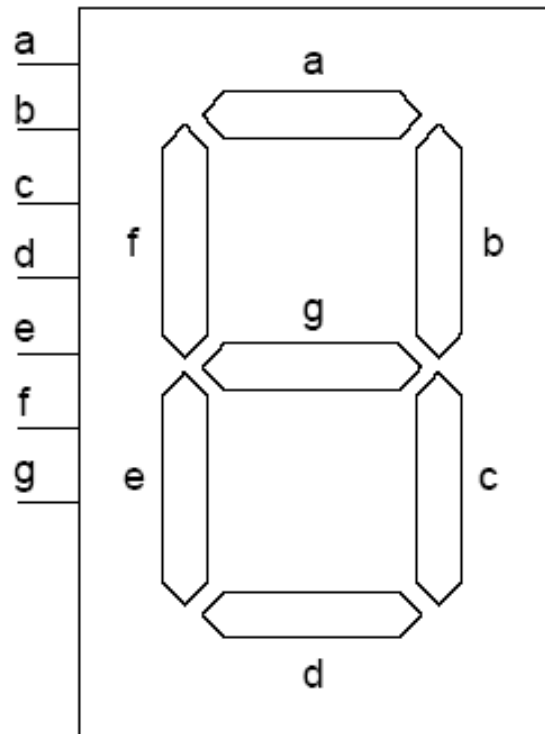


FIG. 7 – Représentation de l'afficheur **HDSP-A10X**.

La **Figure 7** représente le décodeur et la **Table 6** son fonctionnement interne avec l'affichage digital correspondant.

Output	Affichage
a b c d e f g	Digital
1 1 1 1 1 1 0	0
0 1 1 0 0 0 0	1
1 1 0 1 1 0 1	2
1 1 1 1 0 1 1	3
0 1 1 0 0 1 1	4
1 0 1 1 0 1 1	5
0 0 1 1 1 1 1	6
1 1 1 0 0 0 0	7
1 1 1 1 1 1 1	8
1 1 1 0 0 1 1	9
0 0 0 1 1 0 1	A
0 0 1 1 0 0 1	B
0 1 0 0 0 1 1	C
1 0 0 1 0 1 1	D
0 0 0 1 1 1 1	E
0 0 0 0 0 0 0	F

TAB. 6 – Table de résultats de l'afficheur 7 segments **HDSP-A10X**.

3 Proposition de solutions

D'abord, un rappel du concept de multiplexage est nécessaire. Ensuite, on expliquera brièvement l'inverseur pour finir sur le concept du décodage de données et de l'affichage de celles-ci.

3.1 Multiplexeur - Démultiplexeur

On ne peut définir un montage multiplexeur - démultiplexeur sans avoir définie au préalable ce qu'est le *multiplexage*.

On appelle *multiplexage*, la capacité à transmettre sur un seul support physique (voie haute vitesse), des données provenant de plusieurs paires d'équipements tels que des émetteurs et des récepteurs (voies basses vitesses).

Un multiplexeur est un équipement de *multiplexage* permettant de combiner les signaux provenant des émetteurs pour les faire transiter sur la voie haute vitesse. On nomme démultiplexeur l'équipement de *multiplexage* sur lequel les récepteurs sont raccordés à la voie haute vitesse. Le multiplexeur est souvent utilisé pour transformer une valeur codée sur plusieurs bits, i.e. un **bus** en une série de données binaires et séquentielles à diriger vers un circuit. Par exemple afin d'effectuer une transmission de données. C'est le principe de la transmission **série**, utilisé, entre autre, dans les Télécommunications.

Le multiplexeur, dans le domaine des circuits logiques, est un composant à 2^n entrées, n lignes de sélection et 1 sortie. Le démultiplexeur, lui, ne possède qu'une entrée, n lignes de sélection et 2^n sorties. Par exemple, un multiplexeur où $n=2$ à savoir 4 entrées, 2 lignes de sélections et 1 sortie de type **SN74LS153**; ainsi qu'un démultiplexeur avec la même caractéristique ($n=2$) tel que le **SN74LS155a**.

Les 4 premières broches du multiplexeur de la **Figure 3** que l'on nommera C_0 , C_1 , C_2 et C_3 représentent 4 entrées de données. Les 2 broches correspondant aux lignes de sélections que l'on nommera A et B représentent l'adresse, i.e. le numéro de l'entrée à commuter sur l'unique sortie. Ces 2 entrées (A et B) permettent de déterminer l'une des 4 entrées C_0 , C_1 , C_2 et C_3 qui sera connectée à la sortie S. Le **Tableau 2** illustre la table de vérité pour le composant **SN74LS153**.

Il existe une broche commune de commande appelée *strobe* qui sert à activer le multiplexeur. Ce *strobe* sera relié à la masse pour activer le multiplexeur.

La première broche d'entrée du démultiplexeur de la Figure 4 que l'on nommera DATA correspond au bus d'entrée des données envoyées par le multiplexeur. Les 2 broches correspondant aux lignes de sélections que l'on nommera A et B représentent l'adresse, i.e. le numéro de la sortie à commuter sur l'unique entrée. Ces lignes de sélections proviennent du multiplexeur. Les sorties seront nommés Y_0 , Y_1 , Y_2 et Y_3 et le **Tableau 3** illustre la table de vérité pour le composant **SN74LS155a**.

Il faut noter que le démultiplexeur **SN74LS155a** utilisé possède une contrainte : il inverse la valeur des signaux, i.e. un signal haut devient un signal bas et vice versa. Il faut donc rétablir les valeurs des signaux initiaux : c'est la fonction de l'inverseur.

3.2 Inverseur

L'inverseur est un composant à n entrée et n sortie. Il inverse tout simplement son signal d'entrée qu'il renvoie à sa sortie. Si un **1** logique est à son entrée, un **0** logique sera présent à sa sortie et vice versa, i.e. le comportement est similaire à un **NON** logique. On choisit un inverseur de type **SN74LS04** possédant 6 portes d'entrées et 6 portes de sorties. Les entrées seront nommées A_1, A_2, A_3, A_4 (car le signal en sortie du démultiplexeur **SN74LS155a** comporte 4 sorties), et les sorties Y_1, Y_2, Y_3, Y_4 comme l'illustre la **Figure 5**. Le **Tableau 4** illustre la table de vérité pour le composant **SN74LS04**.

3.3 Décodeur BCD à 7 segments

Le décodeur **Binary Coded Decimal - BCD** à 7 segments est un circuit combinatoire à n entrées et 2^n sorties tout comme le démultiplexeur. Quelle que soit la combinaison d'entrée du circuit, une sortie et une seule peut se trouver à l'état haut **1** : chaque sortie du décodeur réalise un minterme différent des entrées respectives. Ce composant permet de traduire un nombre binaire codé sur 4 bits en un nombre décimal affiché sur 7 bits.

On utilisera un décodeur 7 segments **BCD** de type **SN74LS48**. Ce décodeur ne dispose pas de sorties à collecteurs ouverts. Il peut donc directement se relier à l'afficheur à anode commune via une *résistance pull-up* de $2k\Omega$ d'après les spécifications du constructeur. La **Figure 8** illustre un exemple de ce type de résistance lié à un **TTL**.

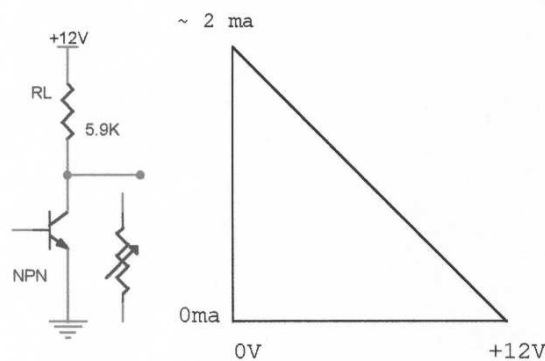


FIG. 8 – Schéma de fonctionnement d'une résistance pull-up.

On utilisera cependant une résistance de 390Ω pour ne pas atténuer l'intensité lumineuse des Light-Emitting Diode - **LEDs**. Cette résistance est en accord avec les contraintes d'intensité de courant supporté par la **LED**. La **LED** accepte un

pic de courant à 45mA et un seuil à 15mA. On déduit la valeur de la résistance approprié :

$$R = \frac{U}{I} = \frac{5}{15 \times 10^{-3}} = 334\Omega < 390\Omega \quad (1)$$

Ce décodeur illustré par la **Figure 6** a aussi la particularité de n'avoir que 7 (à savoir $2^n - 1$) sorties, car il ne prend pas en compte le *point* appelé **Decimal Point - DP** situé en bas à droite d'un 7 segments traditionnelles, contrairement au **SN74LS47**. Ce **DP** n'est autre que l'équivalent de la virgule décimale. Cela ne pose aucun problèmes dans le projet du circuit car toutes les valeurs qui seront affichées sur le 7 segments utilisent un affichage distinct, à savoir **0**, **1**, **2**, **4** et **8**. Le **DP** sert à distinguer un **8** d'un **B** comme illustrer sur la **Figure 9**.



FIG. 9 – Différentiation du 8 et du B avec le DP.

Il faut noter que les sorties sont à **0** et **1**, ce qui n'est pas le cas du **SN74LS47** qui possède des sorties de type **ON** et **OFF**. Cela s'explique par le fait que le **SN74LS47** ait des sorties à collecteur ouvert : **ON** allume le segment, et **OFF** l'éteint : d'où la nécessité d'utiliser des résistances extérieures, ce qui n'est pas le cas du **SN74LS48**. Le **Tableau 5** illustre la table de vérité pour le composant **SN74LS48**.

3.4 Afficheur 7 segments

Le terme afficheur (*display*) désigne tout circuit permettant d'afficher en clair une valeur numérique ou alphanumérique. La diode LED est l'élément le plus simple des afficheurs, elle permet en effet de visualiser un seul bit. Il existe également des afficheurs spécialisés, par exemple les cadrans de montres numériques par exemple.

Il existe un type de circuit permettant d'afficher des informations en code décimal ou hexadécimal au moyen d'une matrice de 7 segments constitués de 7

diodes LEDs : on appelle ces afficheurs, afficheurs 7 segments. L'afficheur à 7 segments permet de reproduire les signes **0** à **9**, **A** à **F**, l'élément vide (toutes **LEDs** éteintes) et comporte également le **DP** comme le montre la **Figure 10**. On exploitera uniquement les signes **0** à **9** dans le projet.

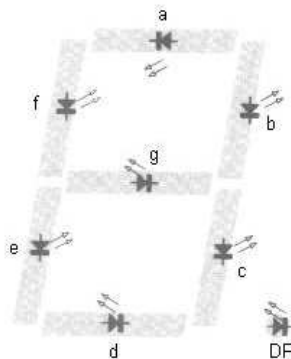


FIG. 10 – Représentation d'un 7 segments et de ses 7 diodes LEDs.

Il est clair que l'on ne peut commander un afficheur de ce type directement avec les sorties d'un compteur binaire. En effet, celui-ci requiert une commande spéciale pour faire apparaître le chiffre décimal choisi. C'est pourquoi on intercale entre ces deux circuits un décodeur 4 vers 7, i.e. le décodeur **SN74LS48** dans le cadre de la manipulation.

L'afficheur utilisé est issue de la série **HDSP-A10X** du constructeur **Hewlett-Packard®**. Chaque segment est désigné par une lettre : **a**, **b**, **c**, **d**, **e**, **f**, **g**, correspondant aux 7 **LEDs**. Celles-ci sont reliées entre elles par leur anode sur l'afficheur **HDSP-A10X** : on utilisera donc un afficheur à *anode commune*.

3.5 RBI et BI-RBO

Pour effectuer la conversion d'un nombre binaire codé sur 4 bits en un nombre décimal affiché sur 7 bits, le décodeur utilise 3 entrées supplémentaires : **Read Blank Input - RBI**, **Blank Input/Read Blank Output - BI/RBO** et **Lamp Test - LT**.

- Quand l'entrée **RBI** est à l'état haut **1**, elle efface les **0** à gauche de l'afficheur si les entrées sont à **0**.
- Quand l'entrée **BI** est à l'état bas **0** (et l'entrée **LT** à **1**) alors tous les segments s'éteignent. Cette entrée permet l'effacement des segments de l'afficheur quelque soit l'état des autres entrées.
- Quand l'entrée **LT** est à l'état bas **0** alors tous les segments s'allument. Cette entrée permet de vérifier le fonctionnement de l'afficheur en allu-

mant tous les segments si **BI** est à l'état 1.

Pour effectuer le montage, il faudra donc placer : **RBI** à 0, **BI** sur 1 et **LT** sur 1.

EXEMPLE : La **Figure 11** montre comment utiliser les entrées et les sorties pour supprimer les **0** non significatifs sur un ensemble de décodage à 3 chiffres.

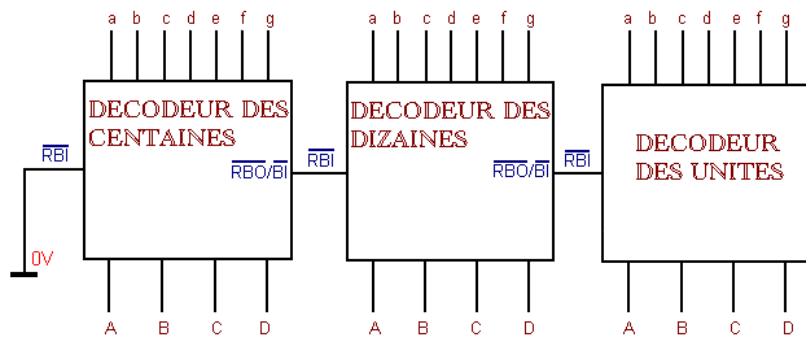


FIG. 11 – Utilisation des entrées \overline{RBI} et des sorties $\overline{RBO}/\overline{BI}$ pour le décodage d'un nombre à 3 chiffres.

Le **RBI** du décodeur des centaines est à 0 en permanence. A chaque fois qu'un 0 se présente sur le décodeur des centaines, ce 0 est effacé et la sortie **RBO** passe à 0 validant **RBI** du décodeur suivant. Si un 0 se présente sur le décodeur des dizaines, il est à son tour effacé et l'on va valider **RBI** du décodeur des unités.

Simulation du projet

4 Utilisation de la librairie Max+Plus 2

4.1 Le multiplexeur

Script 4.1 *Code VHDL du multiplexeur.*

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY multiplexeur IS
5      PORT (
6          a,b,c,d : IN std_logic;
7          selecteur : IN integer RANGE 0 TO 3;
8          sortie  : OUT std_logic
9      );
10 END multiplexeur;
11
12 ARCHITECTURE mul of multiplexeur IS
13 BEGIN
14 WITH selecteur SELECT
15     sortie <=      a WHEN 0,
16                   b WHEN 1,
17                   c WHEN 2,
18                   d WHEN 3,
19                   '0' WHEN OTHERS;
20
21 END mul;
```

4.2 Le démultiplexeur

Script 4.2 Code VHDL du démultiplexeur.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  ENTITY demultiplexeur IS
4      PORT(
5          entree: IN std_logic;
6          selecteur: IN integer RANGE 0 TO 3;
7          sortie: OUT std_logic_vector(3 DOWNT0 0)
8      );
9  END demultiplexeur;
10 ARCHITECTURE demux OF demultiplexeur IS
11 BEGIN
12 WITH selecteur SELECT
13 sortie <=      "111" & NOT entree WHEN 0,
14                "11" & NOT entree & '1' WHEN 1,
15                '1' & NOT entree & "11" WHEN 2,
16                NOT entree & "111" WHEN 3,
17                "1111" WHEN OTHERS;
18 END demux;
```

4.3 L'inverseur

Script 4.3 Code VHDL de l'inverseur.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY inverseur IS
5      PORT (
6          entree : IN std_logic_vector(3 downto 0);
7          sortie : OUT std_logic_vector(3 downto 0)
8      );
9  END inverseur;
10
11 ARCHITECTURE inv of inverseur IS
12 BEGIN
13     sortie <= not entree;
14 END inv;
```

4.4 Le décodeur BCD

Script 4.4 Code VHDL du décodeur BCD 7 segments.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY decodeur IS
5      PORT(
6          entree : IN std_logic_vector(3 downto 0);
7          lt,bi   : IN std_logic;
8          sortie  : OUT std_logic_vector(6 downto 0)
9      );
10 END decodeur;
11
12 ARCHITECTURE dec of decodeur IS
13 BEGIN
14     sortie<="1111110" WHEN entree="0000" and lt='1' and bi='1' ELSE
15         "0110000" WHEN entree="0001" and lt='1' and bi='1' ELSE
16         "1101101" WHEN entree="0010" and lt='1' and bi='1' ELSE
17         "1111001" WHEN entree="0011" and lt='1' and bi='1' ELSE
18         "0110011" WHEN entree="0100" and lt='1' and bi='1' ELSE
19         "1011011" WHEN entree="0101" and lt='1' and bi='1' ELSE
20         "0011111" WHEN entree="0110" and lt='1' and bi='1' ELSE
21         "1110000" WHEN entree="0111" and lt='1' and bi='1' ELSE
22         "1111111" WHEN entree="1000" and lt='1' and bi='1' ELSE
23         "1110011" WHEN entree="1001" and lt='1' and bi='1' ELSE
24         "1110111" WHEN entree="1010" and lt='1' and bi='1' ELSE
25         "1111111" WHEN entree="1011" and lt='1' and bi='1' ELSE
26         "1001110" WHEN entree="1100" and lt='1' and bi='1' ELSE
27         "1111110" WHEN entree="1101" and lt='1' and bi='1' ELSE
28         "1111001" WHEN entree="1110" and lt='1' and bi='1' ELSE
29         "1000111" WHEN entree="1111" and lt='1' and bi='1' ELSE
30         "1111111" WHEN lt='0' and bi='1' ELSE
31         "0000000" WHEN bi='0' ELSE
32         "0000000";
33 END dec;
```

4.5 L'afficheur 7 segments

Script 4.5 *Code VHDL de l'afficheur.*

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY afficheur IS
5      PORT(
6          entree : IN std_logic_vector(6 downto 0);
7          sortie : OUT integer RANGE 0 to 15
8      );
9  END afficheur;
10
11 ARCHITECTURE aff OF afficheur IS
12 BEGIN
13     WITH entree SELECT
14         sortie <=      0 WHEN "1111110",
15                        1 WHEN "0110000",
16                        2 WHEN "1101101",
17                        3 WHEN "1111001",
18                        4 WHEN "0110011",
19                        5 WHEN "1011011",
20                        6 WHEN "0011111",
21                        7 WHEN "1110000",
22                        8 WHEN "1111111",
23                        9 WHEN "1110011",
24                        10 WHEN "1110111",
25 --                        11 WHEN "1111111",
26 -- Non implémenté car pas de DP.
27                        12 WHEN "1001110",
28 --                        13 WHEN "1111110",
29 -- Non implémenté car pas de DP.
30                        14 WHEN "1001111",
31                        15 WHEN "1000111",
32                        0 WHEN OTHERS;
33 END aff;
```

4.6 Le montage

Script 4.6 Code VHDL du montage, première partie.

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  ENTITY montage IS
5      PORT(
6          a,b,c,d : IN std_logic;
7          lt,bi : IN std_logic;
8          selecteur : IN integer RANGE 0 to 3;
9          sortie : OUT integer range 0 to 15
10     );
11 END montage;
12
13 ARCHITECTURE tp1 OF montage IS
14
15 COMPONENT multiplexeur IS
16     PORT (
17         a,b,c,d : IN std_logic;
18         selecteur : IN integer RANGE 0 TO 3;
19         sortie : OUT std_logic
20     );
21 END COMPONENT;
22
23 COMPONENT demultiplexeur IS
24     PORT(
25         entree: IN std_logic;
26         selecteur: IN integer RANGE 0 TO 3;
27         sortie: OUT std_logic_vector(3 DOWNTO 0)
28     );
29 END COMPONENT;
30
31 COMPONENT inverseur IS
32     PORT (
33         entree : IN std_logic_vector(3 downto 0);
34         sortie : OUT std_logic_vector(3 downto 0)
35     );
36 END COMPONENT;
```

Script 4.7 *Code VHDL du montage, deuxième partie.*

```
37  COMPONENT decodeur IS
38      PORT(
39          entree : IN std_logic_vector(3 downto 0);
40          lt,bi   : IN std_logic;
41          sortie  : OUT std_logic_vector(6 downto 0)
42      );
43  END COMPONENT;
44
45  COMPONENT afficheur IS
46      PORT(
47          entree : IN std_logic_vector(6 downto 0);
48          sortie : OUT integer RANGE 0 to 15
49      );
50  END COMPONENT;
51
52  SIGNAL donnee0 : std_logic;
53  SIGNAL donnee1 : std_logic_vector(3 downto 0);
54  SIGNAL donnee2 : std_logic_vector(3 downto 0);
55  SIGNAL donnee3 : std_logic_vector(6 downto 0);
56  SIGNAL donnee4 : integer range 0 to 15;
57
58  BEGIN
59
60  M01: multiplexeur PORT MAP (a,b,c,d,selecteur,donnee0);
61  M02: demultiplexeur PORT MAP (donnee0,selecteur,donnee1);
62  M03: inverseur PORT MAP (donnee1,donnee2);
63  M04: decodeur PORT MAP (donnee2,lt,bi,donnee3);
64  M05: afficheur PORT MAP (donnee3,sortie);
65
66  END tp1;
```

5 Analyse du projet

Le multiplexage est utilisé pour des questions de prix, à savoir transmettre des informations par n fils au lieu de 2^n . Il paraît donc inutile de l'utiliser ici puisque la carte est un support fixe où le multiplexeur et le démultiplexeur sont à quelques centimètres l'un de l'autre. Le multiplexage est pourtant essentiel car c'est ce procédé qui permet d'afficher le **0**, **1**, **2**, **4** et **8** sur le 7 segments en décomposant les signaux d'entrées.

Pour le démultiplexeur, il en existe qui n'inversent pas les signaux en sortie. Le fait d'utiliser le **SN74LS155a** rajoute un **MSI** inutilement, l'inverseur ce qui se traduit par une perte d'espace et une perte d'argent.

En ce qui concerne l'afficheur et le décodeur du signal, il est regrettable qu'une version avec le **DP** n'est pas été utilisé. Le fait de ne pas prendre en compte ce **DP** limite l'implémentation du système qui ne pourra pas afficher un ensemble de 16 valeurs sans ambiguïté.

Il faudra donc remplacer le décodeur pour qu'il puisse prendre en compte le **DP**.

L'afficheur est très important dans le circuit car il permet à l'étudiant d'avoir une vision directe du branchement de son circuit : l'étudiant a alors une vision concrète du montage qu'il est en train de réaliser suivant les résultats que l'afficheur **BCD** à 7 segments retournent.

6 Conclusion

Le projet permet de se familiariser avec les circuits combinatoires en donnant une approche aussi bien théorique (recherche de résultats) que pratique (installation de **MSIs** et codage en **VHDL**). Il permet aussi de se familiariser avec les spécifications des composants - *Datasheet*.

C'est une très bonne application du cours sur la logique combinatoire car il regroupe l'ensemble des **MSIs** les plus utilisés dans les **ICs** : le projet permet de poser les bases d'un projet de plus grande envergure, la conception d'un microprocesseur.

Références

- [1] Thierry Lefebvre, "Le multiplexeur" [online] disponible sur [http ://perso.wanadoo.fr/thierry.lefebvre/digital/mux.htm](http://perso.wanadoo.fr/thierry.lefebvre/digital/mux.htm) vérifié le 15 Décembre 2004.
- [2] Patrick Marcel, "Multiplexeur", [online] disponible sur [http ://www.blois.univ-tours.fr/marcel/archi/node61.html](http://www.blois.univ-tours.fr/marcel/archi/node61.html) vérifié le 15 Décembre 2004.
- [3] Benjamin Drieu "Multiplexeur", [online] disponible sur [http ://www.grassouille.org/docs/cours-ii-html/node43.html](http://www.grassouille.org/docs/cours-ii-html/node43.html) vérifié le 15 Décembre 2004.
- [4] Jean-François Pillou "Transmission de données", [online] disponible sur [http ://www.commentcamarche.net/transmission/transmux.php3](http://www.commentcamarche.net/transmission/transmux.php3) vérifié le 15 Décembre 2004.
- [5] Catherine Douillard - ENST Bretagne "Electronique numérique", [online] disponible sur [http ://perso.enst-bretagne.fr/douillar/ISP304/Cours1.pdf](http://perso.enst-bretagne.fr/douillar/ISP304/Cours1.pdf) vérifié le 15 Décembre 2004.
- [6] Mr Acohen - INRIA Rocquencourt "Traitement des données", [online] disponible sur [http ://www-rocq.inria.fr/acohen/teach/archi/enonce_td1.pdf](http://www-rocq.inria.fr/acohen/teach/archi/enonce_td1.pdf) vérifié le 8 Décembre 2004.
- [7] A. Dipanda "Circuits combinatoires", [online] disponible sur [http ://depinfo.u-bourgogne.fr/DEUG-MIAS-II/DEUG-MIAS-I4/architecture3-6X.pdf](http://depinfo.u-bourgogne.fr/DEUG-MIAS-II/DEUG-MIAS-I4/architecture3-6X.pdf) vérifié le 15 Décembre 2004.
- [8] D. Bertrand "L'unité de contrôle", [online] disponible sur [http ://web.iuhe.ac.be/Bertrand/charleroi6.pdf](http://web.iuhe.ac.be/Bertrand/charleroi6.pdf) vérifié le 15 Décembre 2004.
- [9] L'équipe Dream "Composants logiques", [online] disponible sur [http ://membres.lycos.fr/dtlogic/tech/multiplexeur/composants.html](http://membres.lycos.fr/dtlogic/tech/multiplexeur/composants.html) vérifié le 15 Décembre 2004.
- [10] Guy Bois - Ecole Polytechnique de Montréal "Logique électronique", [online] disponible sur [http ://www.cours.polymtl.ca/inf2501/documents/examens/final_h03.pdf](http://www.cours.polymtl.ca/inf2501/documents/examens/final_h03.pdf) vérifié le 15 Décembre 2004.
- [11] Patrick Furon - ENSI de Caen "Encodeurs", [online] disponible sur [http ://www.ecole.ensicaen.fr/~furon/_elecnumerique/_cours_electronum/](http://www.ecole.ensicaen.fr/~furon/_elecnumerique/_cours_electronum/) vérifié le 15 Décembre 2004.
- [12] Département des miracles "Logique combinatoire", [online] disponible sur [http ://www.miracle.qc.ca/index.php?path=logicc/lc.htm](http://www.miracle.qc.ca/index.php?path=logicc/lc.htm) vérifié le 15 Décembre 2004.
- [13] Jean Provost "Traitement matériel de l'information", [online] disponible sur [http ://www.comelec.enst.fr/tpsp/eni/poly/enich1.html](http://www.comelec.enst.fr/tpsp/eni/poly/enich1.html) vérifié le 15 Décembre 2004.

- [14] Equipe AbcElectronique "Logique et circuits combinatoires", [online] disponible sur http://www.abcelectronique.com/dossiers/logique_combinatoire/-logique_combinatoire.phtml vérifié le 15 Décembre 2004.
- [15] J. Steeman, "Guide des circuits intégrés", Publitrone, ISBN 2-86661-021-0, 1983.
- [16] J. Steeman, "Guide des circuits intégrés 2", Publitrone, ISBN 2-86661-031-8, 1987.
- [17] Steve Furber, "ARM System Architecture", Addison-Wesley, ISBN 0-201-40352-8, 1996.
- [18] Solution for education, "Understanding Transistor Circuits", [online] disponible sur <http://www.sfedcs.com/Theory/Transistors.html> vérifié le 15 Décembre 2004.