REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE SAAD DAHLAB DE BLIDA
FACULTE DES SCIENCES
DEPARTEMENT D'INFORMATIQUE

MEMOIRE

En vue de l'obtention du diplôme
D'INGENIEUR D'ETAT EN INFORMATIQUE

THEME

ELABORATION D'UNE COMMANDE DE TRANSFERT DE DONNEES DANS UN CIRCUIT VISI.

Structure d'accueil :
CENTRE DE DEVELOPPEMENT DES TECHNOLOGIES AVANCEES
Division de Microélectronique & Nanotechnologie

Diplômants:
MECHTI AHMED.
MELZI DJAMEL.

y:

-M^r: BOUNOUAR président.
-M^r: OULD-AISSA. Membre.
-M^r: BOUKHLEF Membre
-M^r: MAHDOUM Ali promoteur.(CDTA).

PROMOTION 2003/2004

Remerciements

Nous remercions tout d'abord **Ellah** *de nous avoir donné la santé pour terminer ce modeste travail.*

Nous remercions vivement notre promoteur, **Ali Mahdoum**, *qui nous a extrêmement aidé et encouragé tout au long de ce travail.*

Nous voulons aussi remercier toute l'équipe des enseignants de l'université **Saad Dahlab de Blida** *qui nous a guidé tout au long de notre vie universitaire.*

Nous tenons également à exprimer nos remerciements au personnel du **CDTA**.

A tous nos collègues et amis qui nous ont soutenu et encouragé pour terminer ce travail et à toute la promotion 2003/2004.

SOMMAIRE

TABLE DES FIGURES

TABLE DES ALGORITHMES

Résumé

Un circuit VLSI de base est composé de deux parties : une partie opérative qui est un ensemble de registres, d'unités fonctionnelles et d'interconnexions, et une partie de contrôle.

La partie opérative consiste à effectuer des opérations arithmétiques, logiques et relationnelles sur des données. Les données et les résultats sont rangés dans des registres, et le transfert entre les registres et les unités fonctionnelles s'opère par le biais des interconnexions (bus).

La partie de contrôle assure le contrôle des opérations et le transfert des données s'effectuant dans la partie opérative.

Afin d'assurer un fonctionnement de la partie opérative répondant à l'algorithme à implémenter par un circuit intégré, les opérations ne doivent pas se faire d'une manière arbitraire, mais plutôt selon un ordre et un mode bien établis. Notre travail consiste alors à réaliser cette partie qui joue le rôle d'interface entre la partie opérative et la partie de contrôle. A partir de fichiers d'ordonnancement et d'allocation, il s'agit plus précisément de :

- déterminer les sources et drains des transistors de commande connectant directement les registres et les unités fonctionnelles aux différentes interconnexions,
- générer l'automate de séquencement des opérations,
- générer la table des signaux de commande,
- déterminer les grilles des transistors de commande

CHAPITRE 1

Introduction générale

Un circuit VLSI de base est composé de deux parties : une partie opérative qui est un ensemble de registres, d'unités fonctionnelles et d'interconnexions, et une partie de contrôle.

La partie opérative consiste à effectuer des opérations arithmétiques, logiques et relationnelles sur des données. Les données et les résultats sont rangés dans des registres, et le transfert entre les registres et les unités fonctionnelles s'opère par le biais des interconnexions (bus).

La partie de contrôle assure le contrôle des opérations et le transfert des données s'effectuant dans la partie opérative. Afin d'assurer un fonctionnement de la partie opérative répondant à l'algorithme à implémenter par un circuit intégré, les opérations ne doivent pas se faire d'une manière arbitraire, mais plutôt selon un ordre et un mode bien établis.

A partir d'un fichier de description des opérations ordonnancées dans des cycles et dont les données et les résultats sont rangés dans des registres bien définis, il s'agit alors d'élaborer une commande assurant un transfert de données correct entre les registres et les unités fonctionnelles via des interconnexions bien définies. Cette commande sera réalisée par l'utilisation de signaux de commande (provenant de la partie de contrôle) et d'autres éléments (transistors de passage) et ce, pour chaque cycle de d'exécution.

Les détails étant abordés dans les chapitres qui leur sont consacrés, nous énumérons ici les grandes lignes de notre travail. A partir de fichiers d'ordonnancement et d'allocation, la détermination des sources et drains des transistors de commande connectant les registres et les unités fonctionnelles aux bus est abordée au chapitre 3, après avoir donné quelques définitions et rappels au deuxième chapitre. Le quatrième chapitre traitera de la génération de l'automate décrivant le séquencement des opérations, de la génération de la table des signaux de commande, et de la détermination des grilles des transistors de commande. Suivra alors le chapitre cinq qui sera une synthèse de notre travail à partir de la présentation de quelques résultats. Une conclusion sera apportée au chapitre six, suivie par une liste de références bibliographiques.

CHAPITRE 2

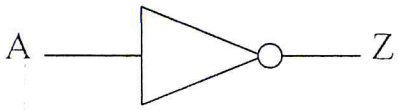
Définitions et rappels

2.1. Introduction:

Avant d'aborder les chapitres 3 et 4 qui constituent l'essentiel de notre travail, nous aimerions apporter dans le présent chapitre quelques définitions et rappels que nous jugeons utiles pour une meilleure compréhension de la description de nos travaux.

2.2. Inverseur:

* Représentation symbolique :



* Table de vérité :

A	Z
0	1
1	0

* Représentation structurelle (en technologie nMOS) :

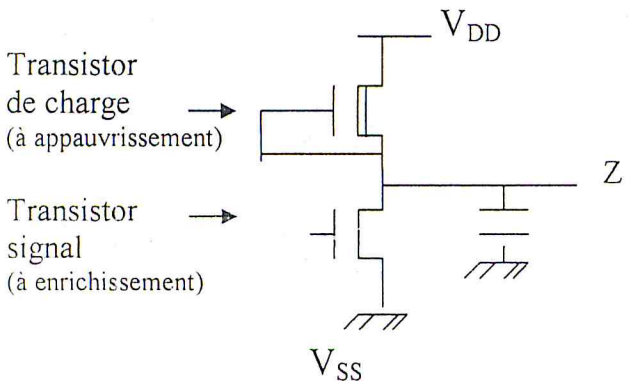


Fig.2.1. Inverseur NMOS

* Représentation structurelle (en technologie pseudo-nMOS) :

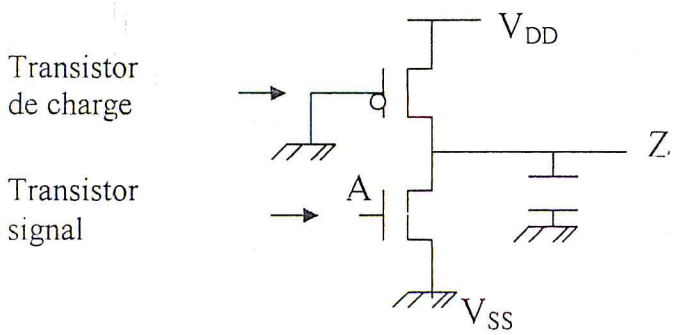


Fig.2.2. Inverseur pseudo-NMOS

* Représentation structurelle (en technologie CMOS) :

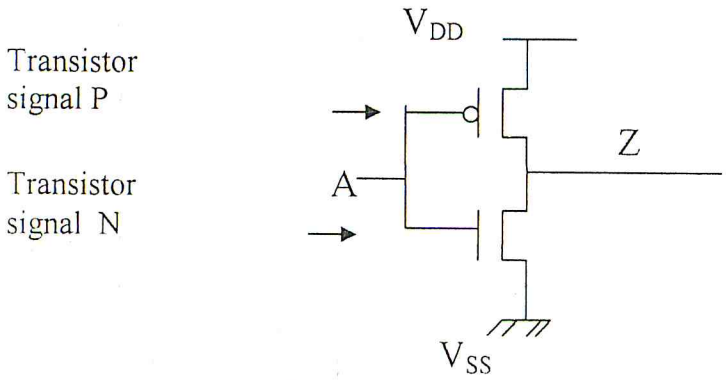
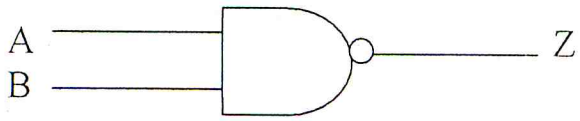


Fig.2.3. Inverseur CMOS

Notons que pour des raisons de consommation de puissance, la technologie CMOS est utilisée (les deux premières sont abandonnées).

2.3. Porte NAND à deux entrées :

* Représentation symbolique :



* Table de vérité :

A	B	Z
0	0	1
0	1	1
1	0	1
1	1	0

* Représentation structurelle (en technologie CMOS) :

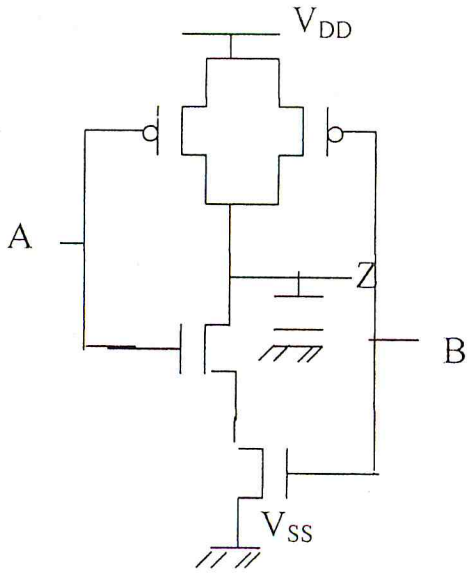
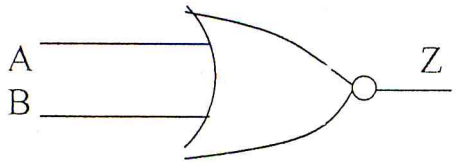


Fig.2.4. Porte NAND à deux entrées

2.4. Porte NOR à deux entrées :

* Représentation symbolique :



* Table de vérité :

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	0

* Représentation structurelle (en technologie CMOS):

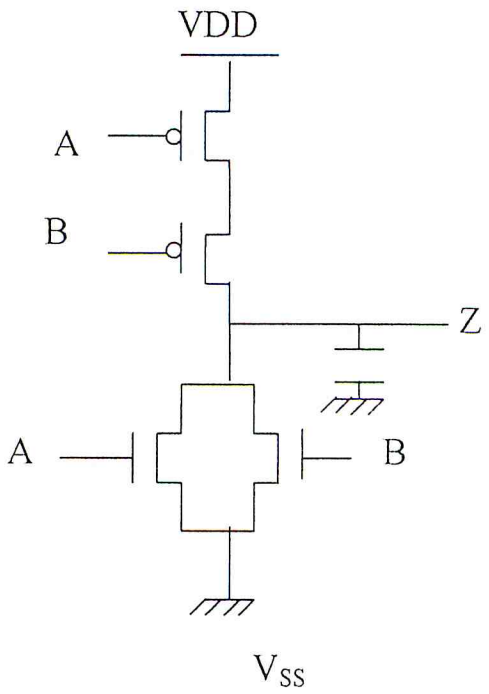
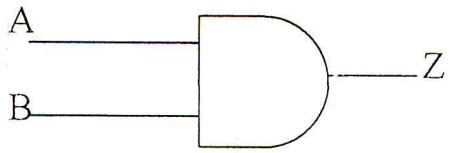


Fig.2.5. Porte NOR à deux entrées

2.5. Porte AND à deux entrées :

* Représentation symbolique :



* Table de vérité:

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

* Représentation structurelle (en technologie CMOS):

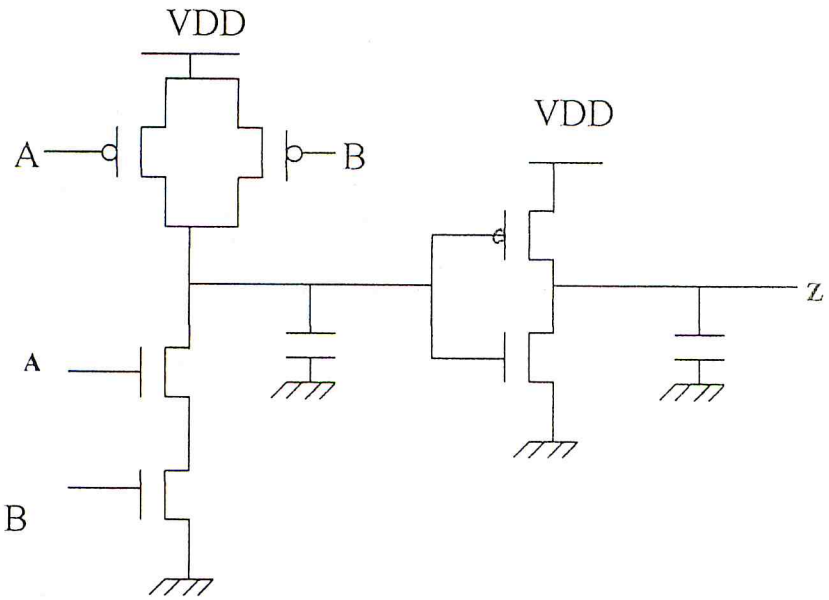
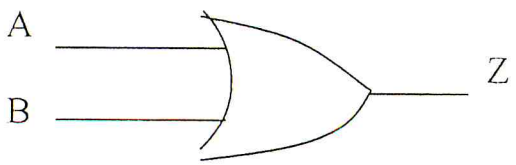


Fig.2.6. Porte AND à deux entrées

2.6. Porte OR à deux entrées :

* Représentation symbolique :



* Table de vérité:

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

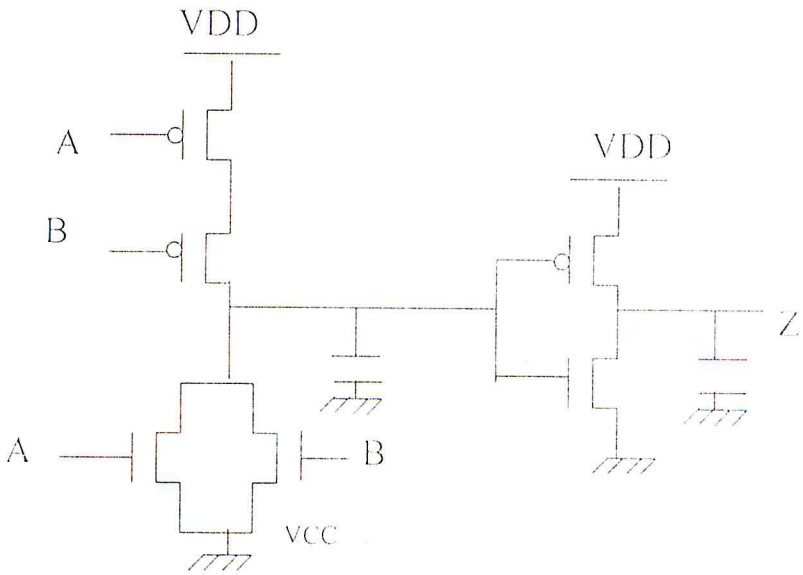


Fig.2.7 Porte OR à deux entrées

2.7. Porte de transmission :

* Représentation symbolique :

* Tables de vérité :

A	D	Z
0	0	X
0	1	X
1	0	0
1	1	1

A	Z
0	X
1	D

* Représentation structurelle (en technologie CMOS) :

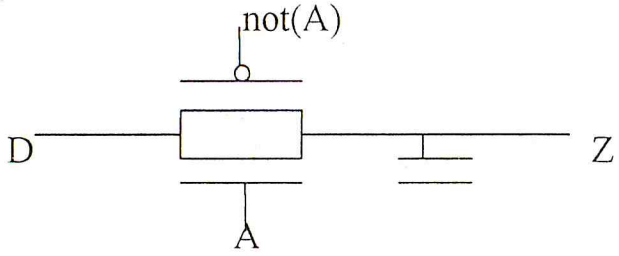


Fig.2.8. Porte de transmission CMOS

2.8. Nœuds de transistors:

Chaque transistor possède 4 Nœuds : sa source, son drain, sa grille, et son substrat. Un exemple en est donné ci-après.

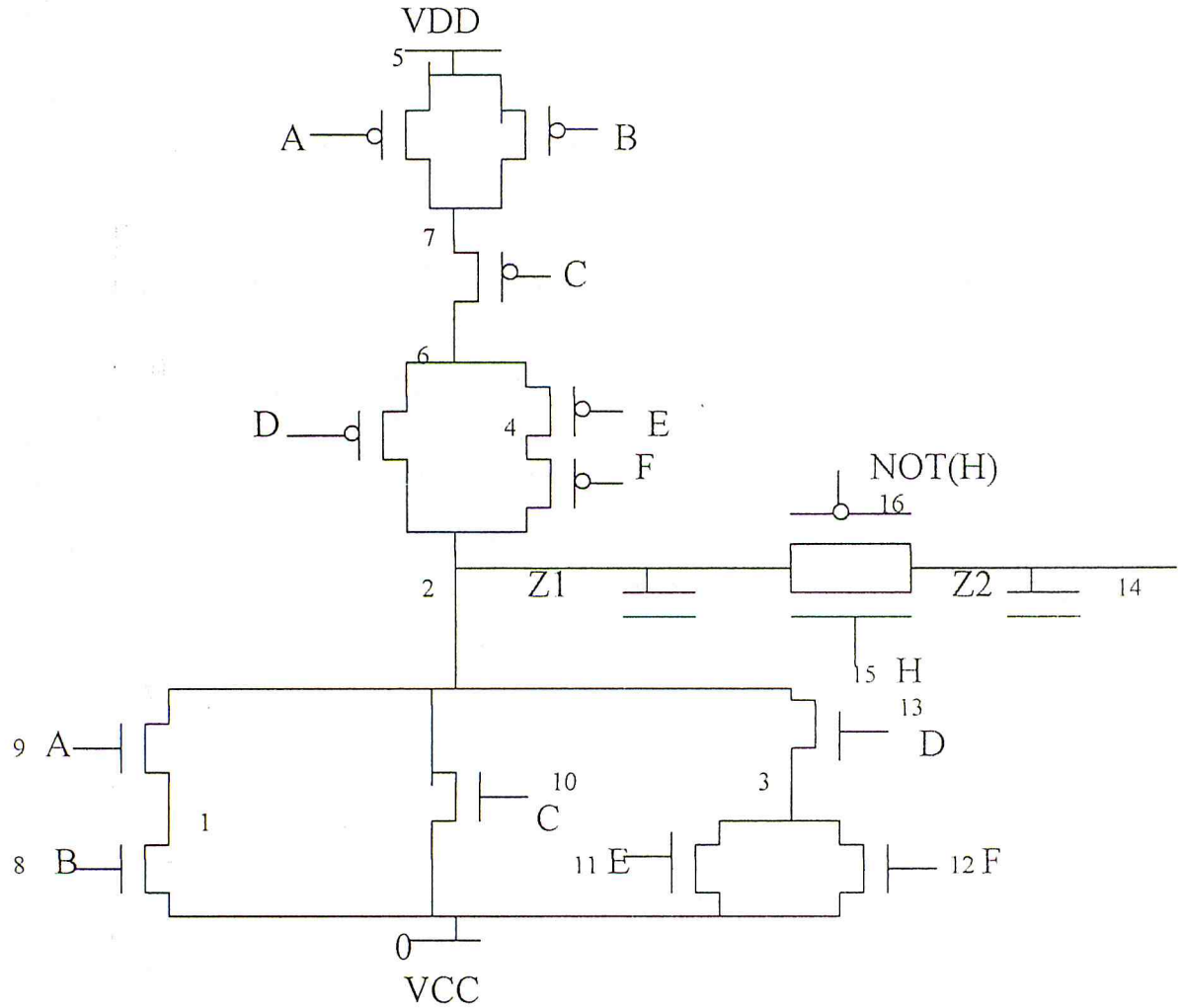


Fig.2.9. Noeuds des transistors d'un circuit CMOS

* Netlist:

Source	Grille	Drain	Substrat	type
0	8	1	0	N
1	9	2	0	N
0	10	2	0	N
0	11	3	0	N
0	12	3	0	N
3	13	2	0	N
6	13	2	5	P
6	11	4	5	P
4	12	2	5	P
7	10	6	5	P
5	8	7	5	P
5	9	7	5	P
2	15	14	0	N
2	16	14	5	P

2.9. Registre:

C'est un élément de mémoire constitué d'un ou de plusieurs bits pour stocker une donnée.

2.10. Unité fonctionnelle:

C'est une entité permettant d'effectuer une opération arithmétique, logique ou relationnelle.

2.11. Bus :

C'est une entité physique permettant le transfert d'informations entre les différents registres et unités fonctionnelles d'une partie opérative.

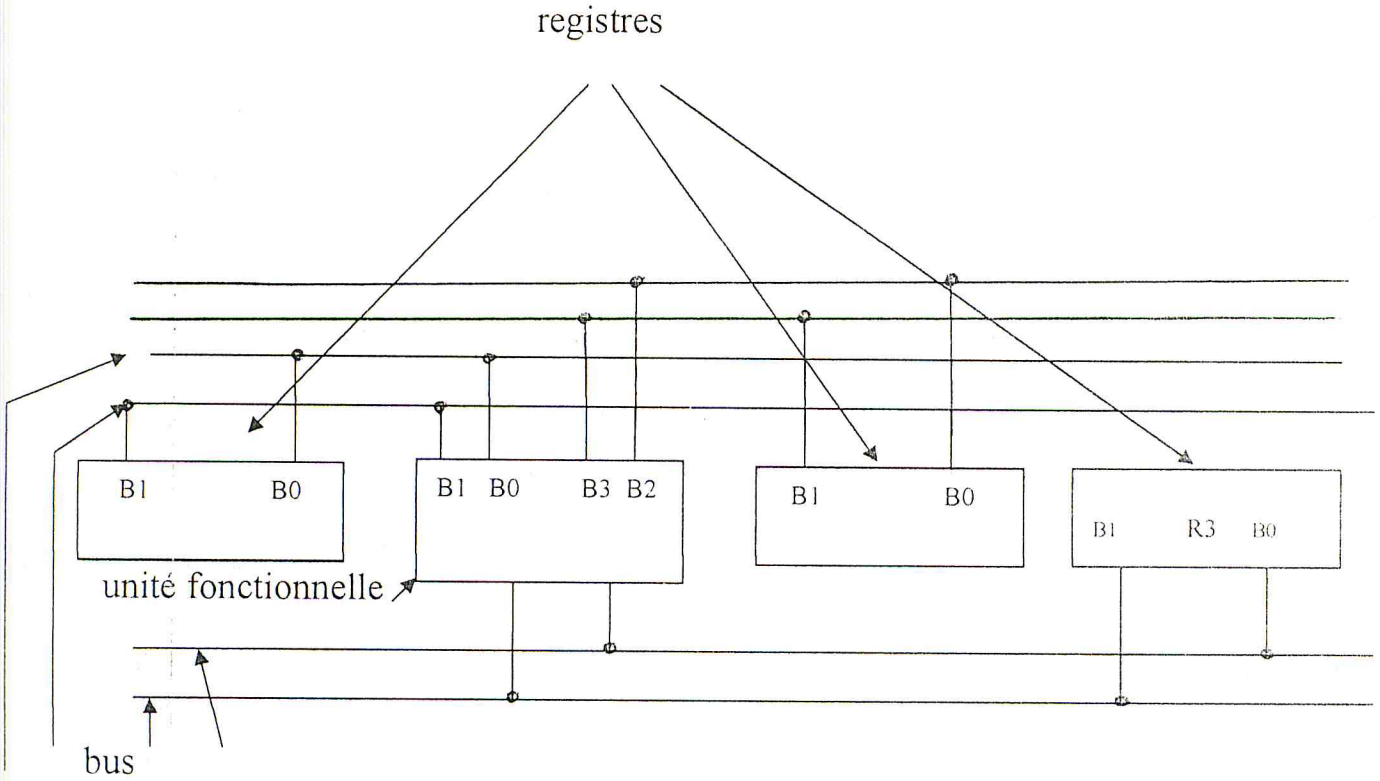


Fig.2.10. Exemple de partie opérative

2.12. Automate :

C'est une structure constituée essentiellement d'états et de transitions entre les différents états. Une transition d'un état à un autre s'effectue dès lors que la condition de la transition devient vraie. Dans un état donné, des tâches (ou instructions) sont exécutées.

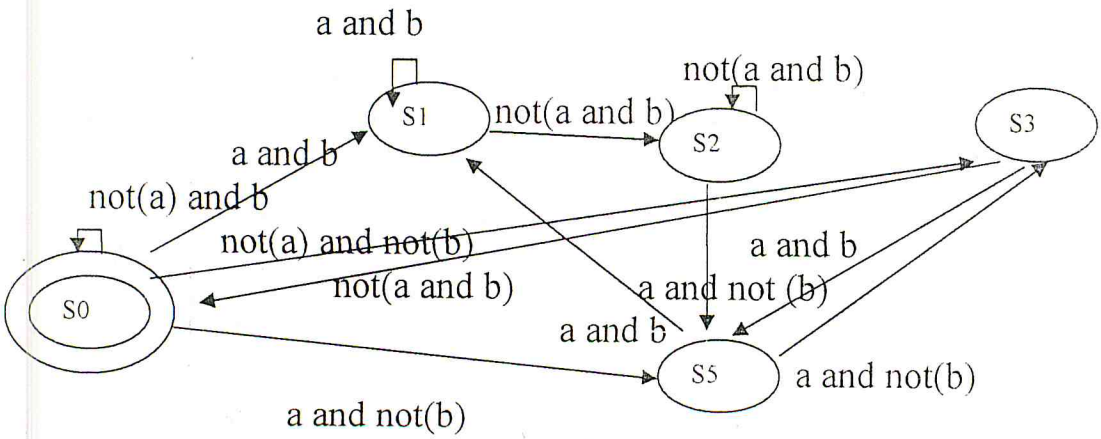


Fig.2.11. Exemple d'un automate

2.13. Machines à états finis:

Nous en distinguons deux types : la machine de Moore, et celle de Mealy :

2.13.1. Machine de Moore :

La fonction $\lambda : Y \rightarrow Y$ permet de déterminer l'état auquel il y'a transition à partir de l'état courant.

La fonction $\delta : Y \rightarrow Z$ permet de déterminer les valeurs des signaux de commande à partir de l'état courant de la machine.

2.13.2. Machine de Mealy :

La fonction $\lambda : X * Y \rightarrow Y$ permet de déterminer l'état auquel il y'a transition à partir de l'état courant, en fonction des valeurs des signaux de contrôle.

La fonction $\delta : X * Y \rightarrow Z$ détermine les valeurs des signaux de commande à partir de l'état courant de la machine et des signaux de contrôle de la machine.

Pour ces deux types de machines,
X : est l'ensemble des entrées primaires (pour la machine de Mealy uniquement)
Y : est l'ensemble des variables des états
Z : est l'ensemble de sorties primaires de la machine

Un exemple de machine de Mealy est le suivant :

Signaux de contrôle		Variables des états à l'entrée			Variables des états à la sortie			Signaux de commande	
C1	C2	Ye1	Ye2	Ye3	Ys1	Ys2	Ys3	F1	F2
0	0	x	x	0	0	0	1	0	1
0	0	0	0	1	0	1	0	1	0
0	1	0	1	0	0	1	1	0	0
0	1	1	1	0	0	0	0	1	1
0	0	0	1	1	0	0	0	1	0
0	1	0	0	x	1	1	0	1	0
1	x	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	1	0	1
1	1	0	0	1	0	0	0	1	0

x : veut dire 0 ou 1

Fig.2.12. Exemple de machine de Mealy

Dans le cadre de la conception des circuits intégrés, la machine de Mealy est utilisée pour implémenter la partie de contrôle du circuit. La description de cette partie de contrôle est faite à l'aide de ce type de machine, puis optimisée à l'aide d'outils de CAO appropriés. Parmi ces outils, on distingue les outils d'affectation efficace de codes aux états de la machine à états finis (exemple,

[7]), suivie par une synthèse logique (exemple, [9]) et une synthèse physique (exemple, [10]).
A titre d'illustration, nous donnons la représentation structurelle de la partie de contrôle (Fig.2.13) correspondant à la machine décrite dans la figure 2.12. La représentation physique d'une partie de contrôle est indiquée, elle, dans la figure 2.14.

2.14. Ordonnancement :

Avant d'implémenter un algorithme par un ASIC (circuit intégré à application spécifique), il s'agit d'ordonnancer les opérations de l'algorithme selon des critères de surface, vitesse et consommation de puissance, et ce, selon l'application considérée. La partie opérative implémentant l'algorithme peut être très rapide si le degré de parallélisme est considéré en priorité, mais peut être coûteuse en termes de surface et de consommation de puissance. De ce fait, des outils de CAO sont utilisés pour analyser les circuits avant leur fabrication, dont des outils d'analyse de la consommation de puissance (exemple, [12], [13] et [14]). Ainsi, l'ordonnancement d'instructions est souvent fait avec ajout de contraintes introduites par l'utilisateur afin de trouver le compromis qui correspond le mieux à l'application considérée. Plusieurs méthodes d'ordonnancement existent, et le lecteur intéressé par cet aspect peut se référer à [11].

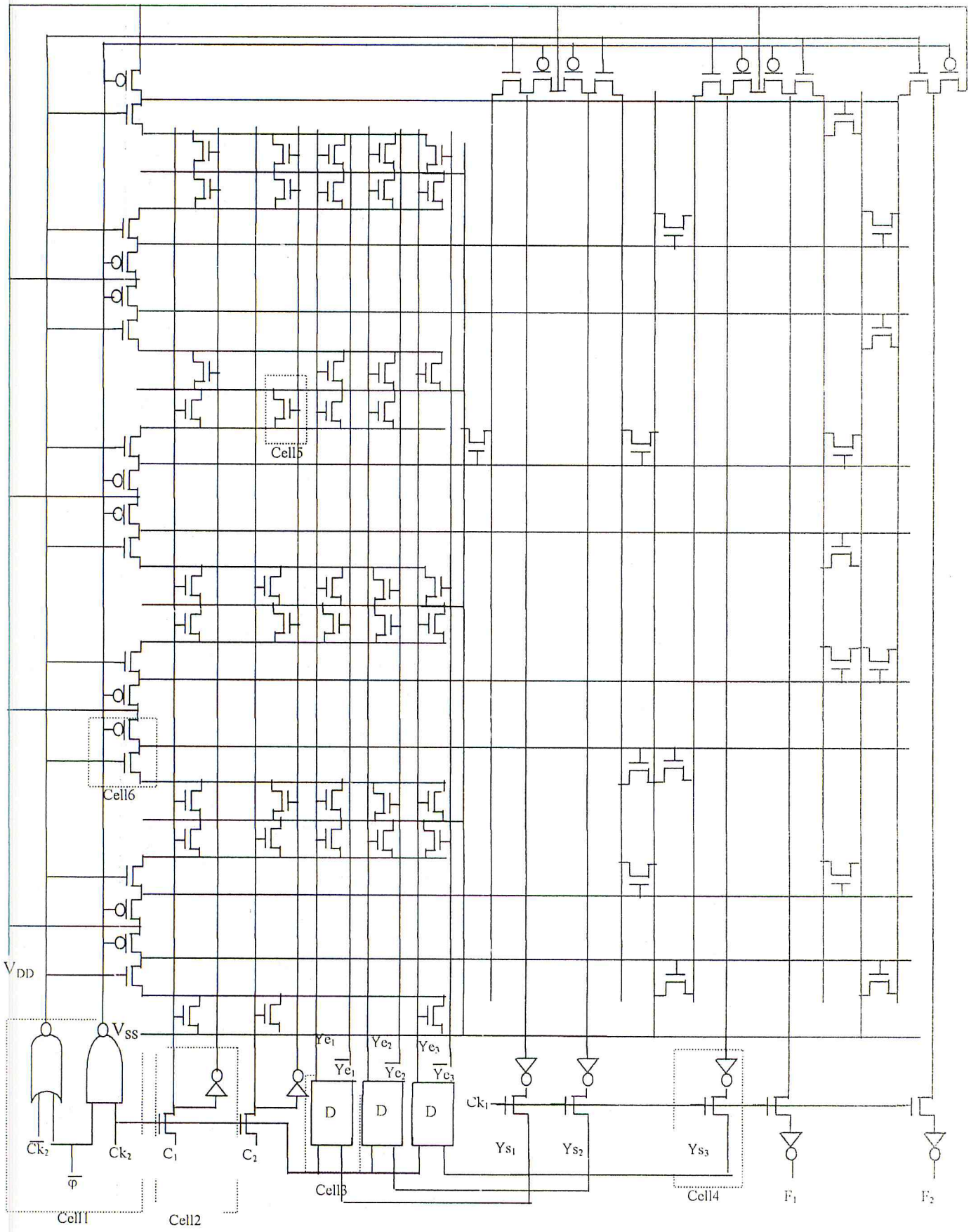


Fig.2.13. PLA implémentant la partie de contrôle décrite dans la Fig.2.12.

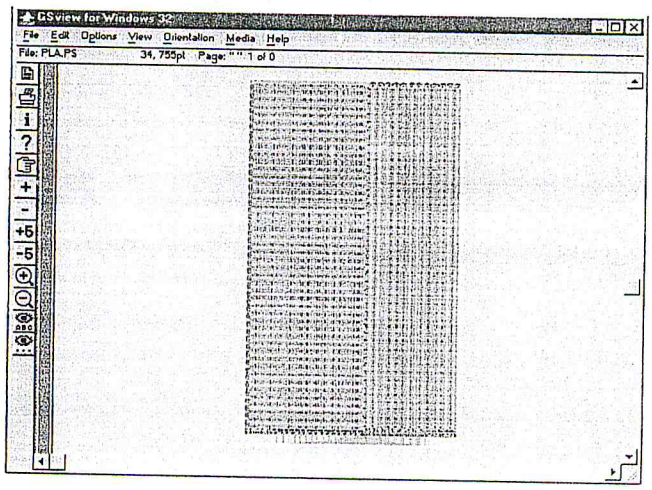


Fig.2.14. Représentation physique d'une partie de contrôle.

2.15. Allocation :

Les opérations étant ordonnancées, il n'est pas possible de stocker chaque valeur d'une variable dans un registre qui lui est propre car ceci rendrait le circuit très coûteux, voire impossible à réaliser (des millions de variables nécessiteraient alors autant de registres). Il en est de même pour les transferts de données entre les registres et les unités fonctionnelles d'une partie opérative : il n'est pas possible d'affecter des interconnexions (bus) propres à chaque transfert. L'allocation est alors une tâche consistant à allouer de manière efficace, les registres aux différentes variables à stocker, et les bus aux différents transferts des données, et ce, sans qu'il y'ait de conflit. Il ne faudrait pas affecter un registre à une autre variable tant que la valeur qui y est rangée sera encore utilisée comme opérande pour d'autres instructions, mais pouvoir affecter ce même registre à une autre variable si ce problème ne se pose plus. De même, un bus donné ne peut être affecté à deux transferts de données s'effectuant en parallèle, mais peut être utilisé pour un autre transfert dans le cas contraire.

2.16. Conclusion:

Dans ce chapitre, nous avons présenté des définitions et fait des rappels d'éléments nécessaires pour une meilleure compréhension des chapitres suivants qui constituent l'essentiel de notre travail.

CHAPITRE 3

Détermination des sources et drains des transistors de commande

3.1. Introduction :

Comme il a été dit au premier chapitre, les opérations doivent s'exécuter selon un ordre bien établi, conformément à l'algorithme à implémenter. Pour ce faire, des transistors de commande sont utilisés pour assurer un transfert correct de données entre les différents registres et unités fonctionnelles de la partie opérative à travers des interconnexions (bus). Il s'agit alors de définir les différents nœuds de chacun des transistors utilisés. Ces différents nœuds permettront ultérieurement (au niveau de conception physique) le routage des différents éléments du circuit intégré, c'est-à-dire de définir la manière dont sont connectés les différents composants du circuit.

Afin d'éviter tout conflit, les différents nœuds de chacun des transistors de commande doivent être correctement définis. Dans le présent chapitre, nous allons présenter notre méthode de détermination des sources et drains de ces transistors, celle de détermination de leur grille sera abordée dans le chapitre suivant.

3.2. Composants d'une partie opérative:

Une partie opérative d'un circuit intégré est constituée essentiellement de registres (pour ranger les différentes informations), d'unités fonctionnelles (pour effectuer des opérations arithmétiques, logiques ou relationnelles), de bus (pour assurer le transfert de données entre les registres et les unités fonctionnelles), et de transistors de passage (ou de multiplexeurs) pour la commande de transfert de ces données.

3.2.1. Registres :

La commande de lecture et d'écriture dans un registre se fait moyennant des portes de transmissions constituées chacune d'un transistor N et d'un transistor P (lorsque la technologie CMOS est utilisée) ayant les mêmes sources et drains, mais des grilles différentes (les signaux appliqués sur les deux grilles sont complémentaires). Afin de faciliter l'implémentation de l'algorithme déterminant les nœuds des différents transistors de commande, les sources sont les nœuds qui sont directement connectés aux entrées et aux sorties des registres (Cf. Fig.3.1) où :

S_i est la source commune des transistors P et N de la $i^{ème}$ porte de transmission
D_i le drain commun des transistors P et N de la $i^{ème}$ porte de transmission
G_{Pi} est la grille du transistor P de la $i^{ème}$ porte de transmission
G_{Ni} est la grille du transistor N de la $i^{ème}$ porte de transmission

Notons que les grilles des transistors (du même type) des différentes portes de transmission peuvent être alimentées par le même signal provenant de la partie de contrôle.

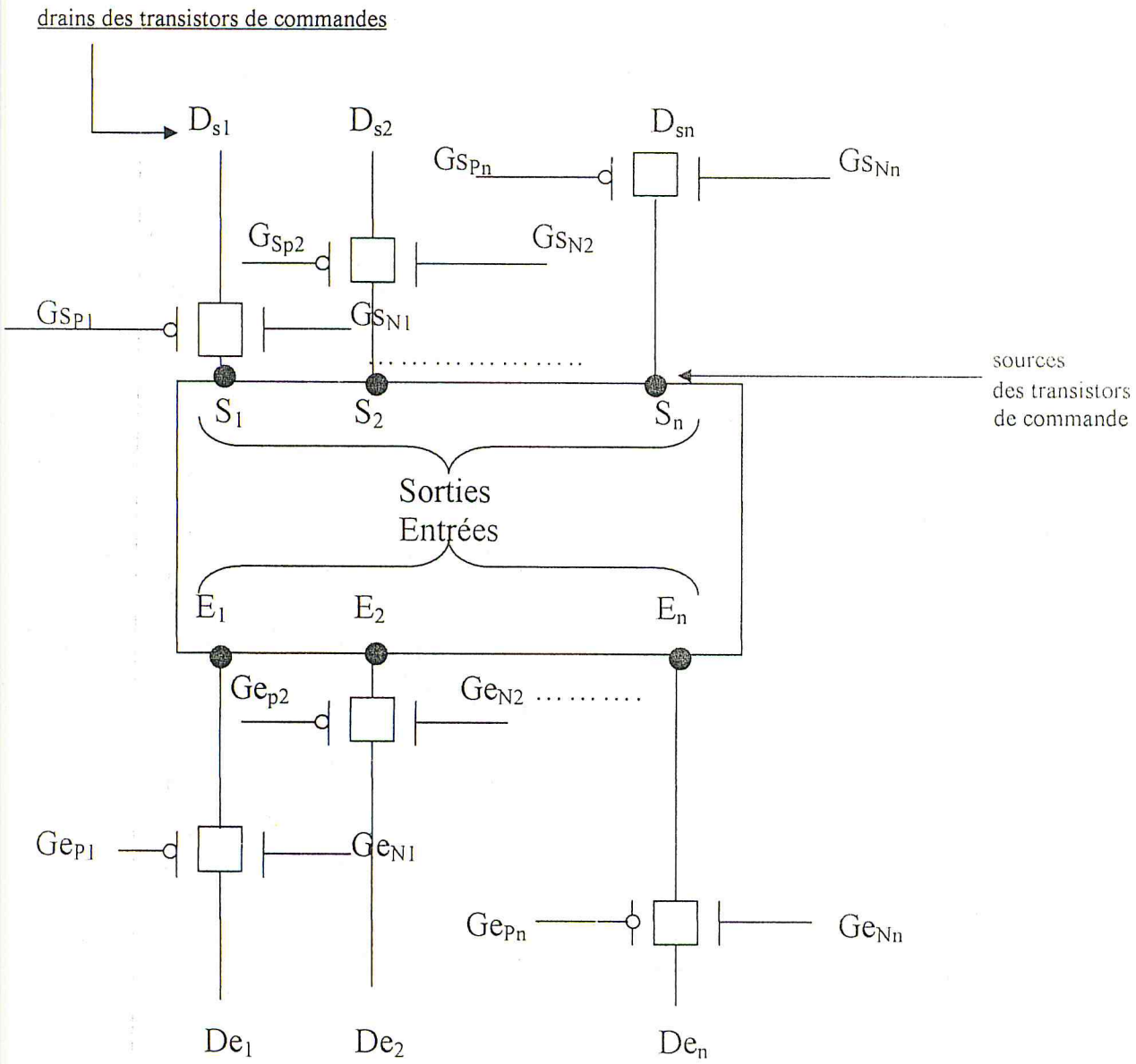


Fig.3.1 : Entrées et sorties d'un registre.

Un registre peut être représenté par deux listes contenant les numéros affectés aux sources des transistors se trouvant en entrée et en sortie du registre.

Exemple 3.1 : R1 : 12 13 %% 14 15
Ceci signifie que le registre R1 est à deux bits, et que les sources des transistors en entrée sont numérotées 12 13, ceux en sortie, 14 15 (Cf. Fig.3.2).

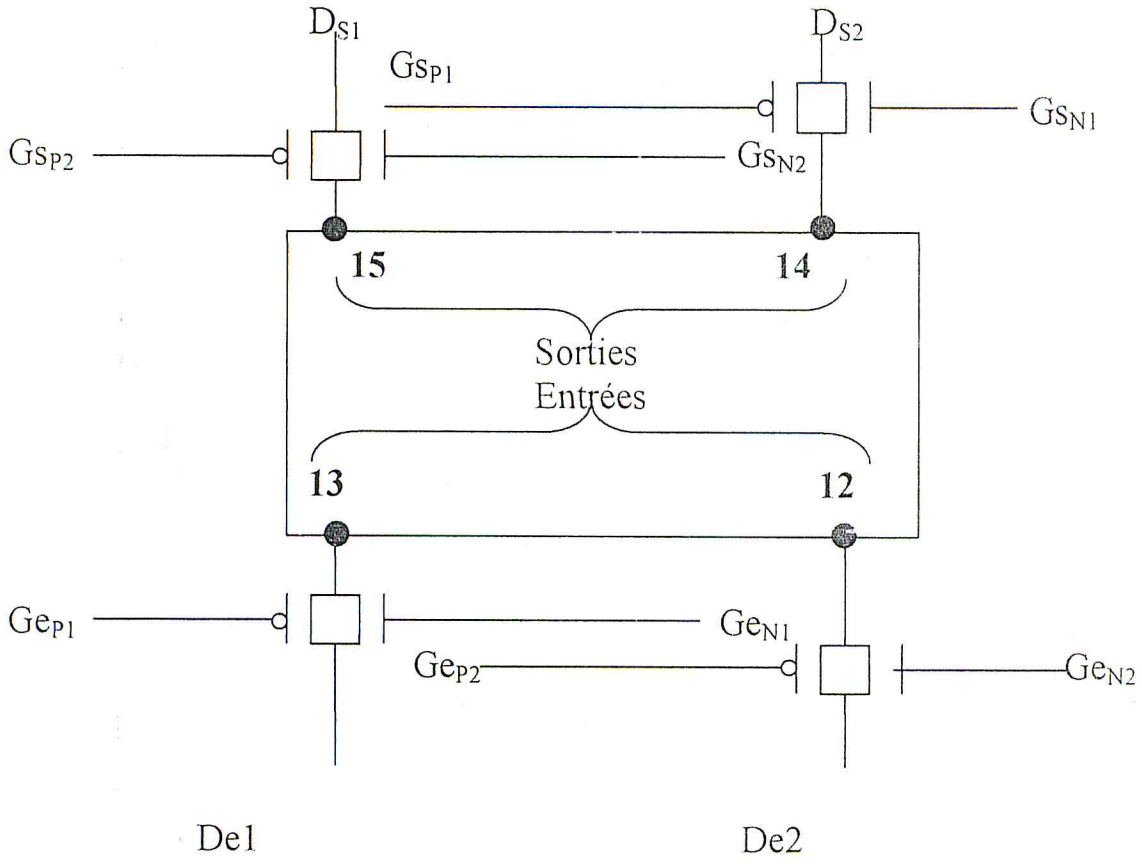


Fig.3.2. Exemple d'affectation des sources en entrée et en sortie d'un registre à 2 bits.

3.2.2. Unités fonctionnelles :

Pour ce qui est des unités fonctionnelles, la même définition que celle utilisée pour les registres est adoptée.

Exemple 3.2: and : 2 1 4 3 %% 5 6

Ceci signifie que l'unité fonctionnelle *and* opère sur des opérandes à deux bits, et les sources des transistors en entrée sont numérotées 1, 2,3,4, celles en sortie sont 5 et 6 (Cf. Fig.3.3).

Remarque :

Les entrées des registres sont reliées avec les sorties des unités fonctionnelles alors que les entrées des unités fonctionnelles sont reliées avec les sorties des registres

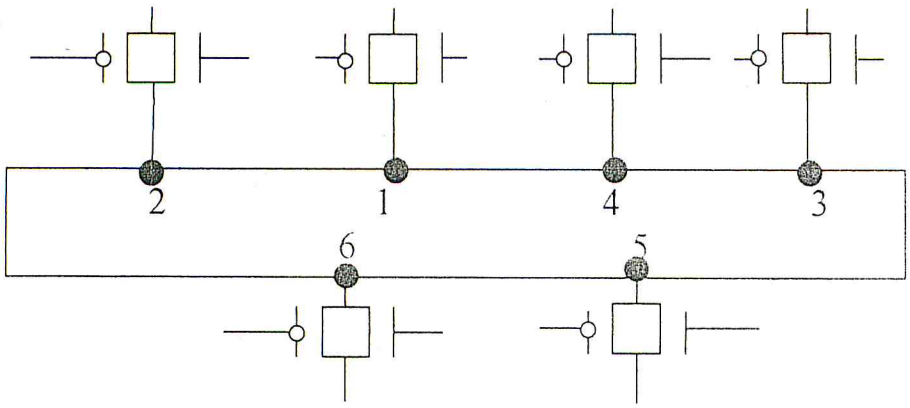


Fig.3.3. Exemple d'affectation des sources en entrée et en sortie d'une unité fonctionnelle opérant sur des opérandes à 2 bits.

3.2.3. Bus :

Un bus B permet de transférer des données depuis ou vers un registre de n bits sur n fils différents (un fil pour chaque bit). Nous représentons un bus B par la liste des numéros affectés à ses fils.

Exemple 3.3 :

B1 : 1 2 3 4

Ceci signifie que le bus B1 a 4 fils (nœuds électriques) numérotés de 1 à 4.

3.2.4. Transistors de commande :

Dans notre contexte, chaque transistor est représenté par 3 numéros correspondant à sa source, son drain et sa grille. Nous utilisons les deux caractères N, P pour distinguer un transistor de type NMOS d'un transistor de type PMOS.

Exemple 3.4:

T : 12 4 5 N

Ceci signifie qu'il s'agit d'un transistor de type NMOS ayant pour source, grille et drain les nœuds 12, 4 et 5, respectivement.

Avant d'aborder les problèmes relatifs à l'affectation des nœuds aux transistors de commande, notons que deux entités physiques quelconques (registres, unités fonctionnelles, bus, transistors) se partageant le même nœud (même numéro) seront physiquement connectées à travers ce nœud.

3.3. Problèmes d'affectation de noeuds:

3.3.1. Problème de conflit de bus :

Ce problème est partiellement résolu dans la phase d'allocation dans la mesure où on n'affecte pas le même bus à deux opérations différentes s'opérant en même temps. Toutefois, il est possible que les données d'un registre donné soient envoyées sur des bus différents (i, j, k) dans des cycles différents. De ce fait, quand les données sont envoyées sur le bus i, il faut garantir que le transfert se passe seulement à travers le bus i et non pas sur les autres bus (j et k), ce qui permet d'éviter tout conflit avec les transferts parallèles s'opérant sur les bus j et k.

Pour remédier à ce problème, il faut, lors d'un transfert sur le bus i, empêcher tout contact entre le registre en question et les bus j, k, ce qui nécessite l'utilisation d'*interrupteurs* (Cf. Fig.3.4).

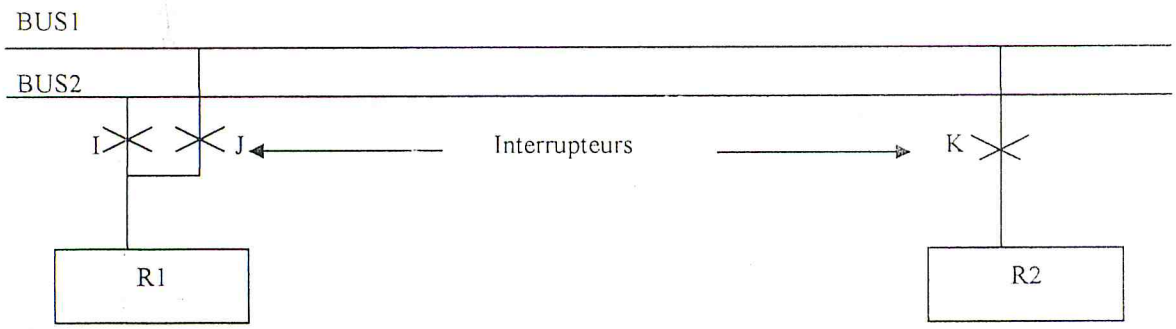


Fig.3.4. Solution au problème de conflit de bus

Ainsi, la figure 3.4 indique qu'il est possible d'effectuer le transfert des données de R1 sur le bus2 et celui des données de R2 sur le bus1 en même temps, sans qu'il n'y'ait aucun conflit du fait que l'interrupteur J est ouvert, et ne permet pas aux données de R1 de transiter sur le bus 1. Notons que ce cas peut se produire quand les données du registre R1 sont envoyées sur la bus 1 lors d'un traitement dans un autre cycle d'exécution de la partie opérative, et que les interrupteurs I,K et J sont commandés par des signaux de fermeture/ouverture provenant de la partie de contrôle.

3.3.2. Problème des poids forts et faibles :

Lors d'une opération comme : [R1] := [R2] - [R3], il est indispensable que chaque bit de R2 et de R3 soit transmis à l'unité fonctionnelle "-" d'une manière à ce qu'il arrive avec le même poids d'origine. Il en est de même pour les bits transmis par l'unité fonctionnelle "-" au registre R1.

Pour une affectation de nœuds correcte, les entrées d'un registre donné reçoivent des numéros croissants du poids le plus faible vers le poids le plus fort. La même manière d'affectation est utilisée pour les sorties.
Les sorties des registres sont alors raccordées convenablement aux entrées des unités fonctionnelles en se basant sur les poids faibles et forts des bits des données. De même, les bits de sortie des unités fonctionnelles sont raccordés à ceux des entrées des registres en respectant les valeurs des poids (Cf. Fig.3.5).

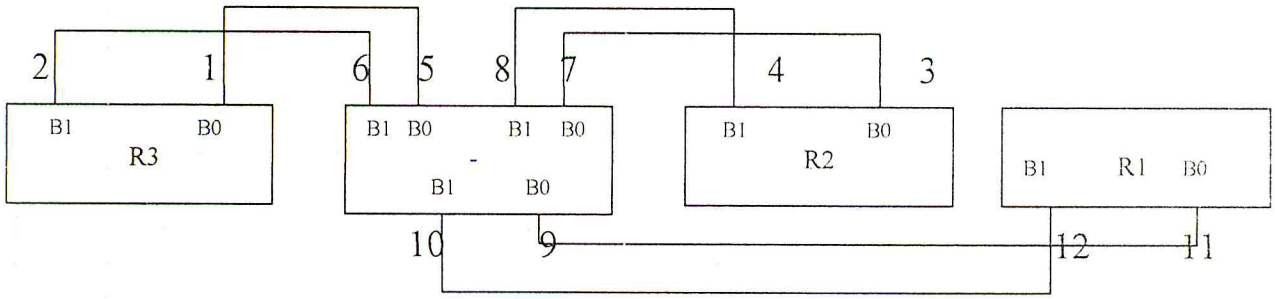


Fig. 3.5. Interconnexions correctes de bits en fonction de leurs poids faibles et forts

3.3.3. Problème des types des unités fonctionnelles :

Si le nombre de bits dans un circuit égal à *n*, alors il y'a :

- *n* bits en sortie de chaque registre et de chaque unité fonctionnelle unaire (unité opérant sur un seul opérande) ou binaire (unité opérant sur deux opérandes), à l'exception de la multiplication (nous aborderons ce cas ultérieurement)
- *n* bits en entrée de chaque unité fonctionnelle unaire et de chaque registre (non utilisé par la multiplication)

Un exemple en est donné à la figure 3.6.

Par contre, il y a 2**n* bits en entrée de chaque unité fonctionnelle binaire (*n* bits pour le registre i et *n* autres bits pour le registre j) (Cf. Fig.3.7).

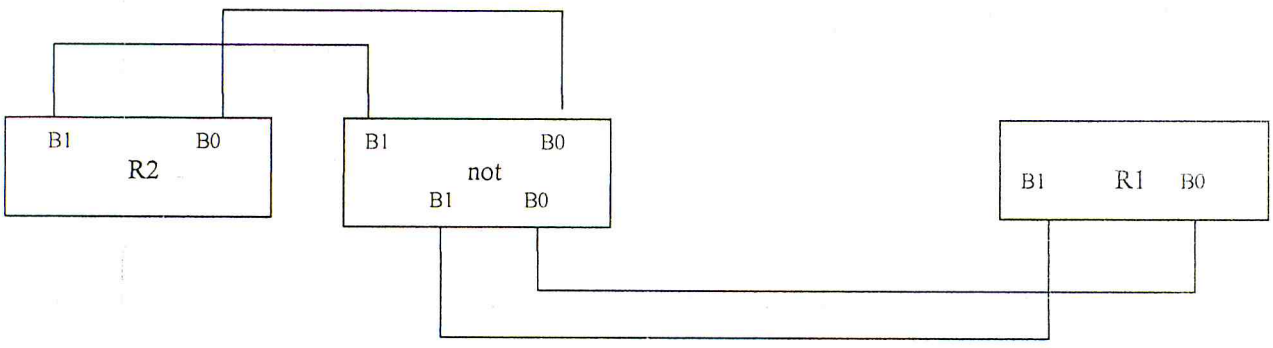


Fig.3.6 Exemple d'unité fonctionnelle unaire

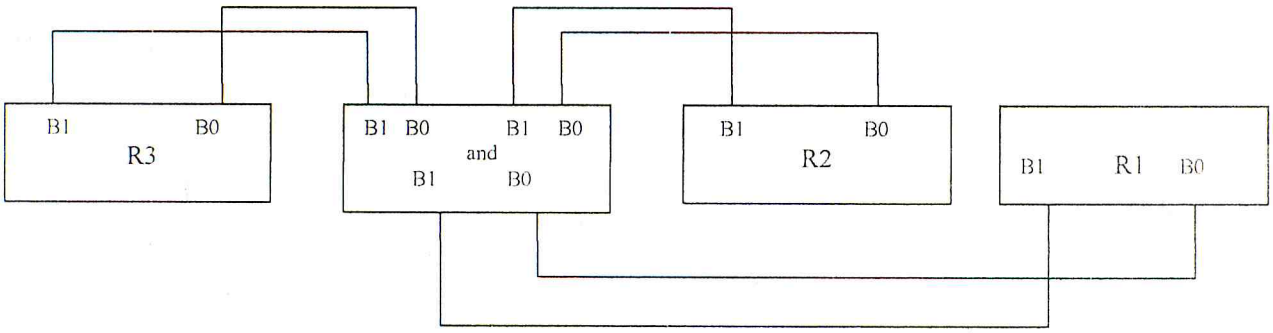


Fig.3.7 Exemple d'unité fonctionnelle binaire.

Par conséquent, pour assurer un transfert correct sur un nombre d'interconnexions correct, il faut prendre en considération ces différents cas.

3.3.4. Problème de nombre de bits pour une opération de multiplication :

Si le nombre de bit dans un circuit est égal à *n* alors il y'a :

- 2**n* bits en entrée et en sortie de chaque unité fonctionnelle exécutant la multiplication
- 2**n* bits en entrée de chaque registre utilisé pour stoker le résultat de la multiplication

Un exemple en est donné à la figure 3.8.

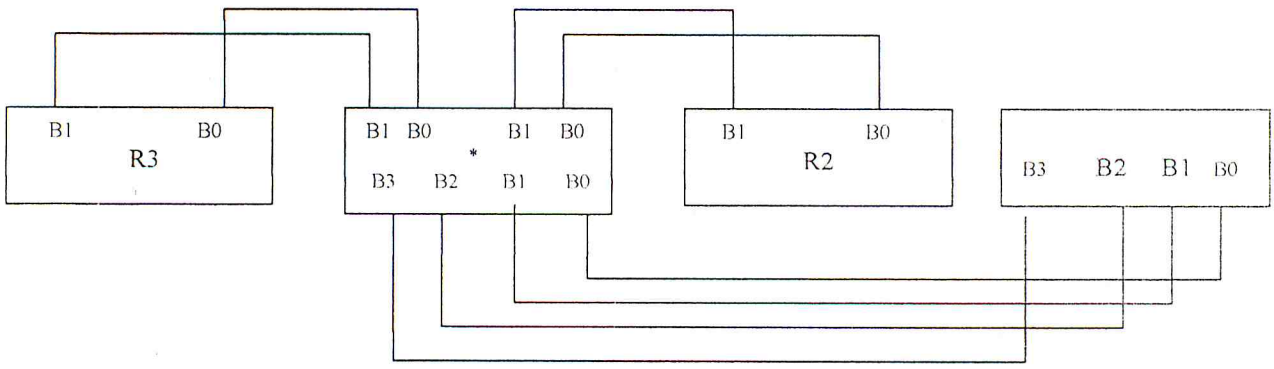


Fig.3.8 Cas d'une multiplication

De même, pour assurer un transfert correct sur un nombre correct d'interconnexions, il faut prendre en considération ce dernier cas.

3.3.5. Problème de l'utilisation exclusive des unités fonctionnelles :

Notons que le coût d'une partie opérative augmente avec les nombres de bits, de bus, de registres et d'unités fonctionnelles composant cette partie opérative. Il est aussi important de remarquer que seules les instructions ordonnancées dans le *même* cycle et soumises aux *mêmes* conditions de contrôle sont exécutées en *même* temps. Du fait que pour chaque unité fonctionnelle des transistors de commande sont utilisés en entrée et en sortie de cette unité, et afin d'éviter l'utilisation inutile d'un nombre abusif d'unités fonctionnelles et la génération d'un nombre plus important de transistors de commande, il est utile d'utiliser la même unité fonctionnelle (et donc les mêmes transistors de commande) lorsque celle-ci est sollicitée pour exécuter le même type d'opération dans des cycles différents. Ceci est illustré dans l'exemple ci-après où deux unités *and* seulement plutôt que trois peuvent implémenter l'algorithme original, sans qu'il n'y'ait aucun problème, du fait que les opérations 1 et 4 sont exclusives. Il en va de même pour les opérations 2 et 4.

Exemple 3.5 :

cycle 1 :
1: r1=r2 and r3 si A
2: r4=r5 and r6 si A
3: r7=not r8 si not(A)

cycle 2 :
4: r1=:r2 and r5 si A
5: r8=not r7 si A

Par conséquent, on a besoin de deux unités fonctionnelles *and* puisque les opérations 1 et 2 seront exécutées en parallèle (pour l'exécution de l'opération 4 au deuxième cycle, l'une de ces deux unités fonctionnelles pourra être utilisée), et d'une seule unité *not* pour exécuter les opérations 3 et 5 respectivement aux cycles 1 et 2.

Par contre dans l'exemple 3.6, *une seule* unité fonctionnelle *and* et *une seule* unité *not* seront utilisées du fait que :

- les opérations 1 et 2 sont exclusives (même si elles sont ordonnancées dans le même cycle) car les conditions A et not(A) ne peuvent être vraies en même temps
- les opérations 1 et 4 sont soumises au même signal de contrôle, mais ordonnancées dans des cycles différents
- les opérations 2 et 4 sont ordonnancées dans des cycles différents
- les opérations 3 et 5 sont ordonnancées dans des cycles différents

Exemple 3.6 :
cycle 1:
1: r1=r2 and r3 si A
2: r4=r5 and 6 si not(A)
3: r7=not r8 si not(A)
cycle 2 :
4: r1=:r2 and r5 si A
5: r8=not r7 si A

3.4. Structures de données et algorithmes:

3.4.1. Structures des données en mémoire principale :

Les structures de données qui ont permis d'implémenter nos algorithmes sont les suivantes :

int_char
int nœud :numéro de nœud
char* chaîne :chaîne de caractèresreprésentant un signal

transistor :
int_char* source : source de transistor
int_char* gate : grille de transistor
int_char* drain : drain de transistor
char type: type de transistor N ou P
transistor* next: pointeur sur un transistor

unité fonctionnelle :
int num_uf: numero de l'unité fonctionnelle
transistor* imput_head : pointeur sur la liste des transistors utilisés en entrée de l'unité fonctionnelle.
transistor* output_head : pointeur sur la liste des transistors utilisés en sortie de l'unité fonctionnelle.
bus1* head_imput1: pointeur sur un bus1 utilisé en entrée1
bus1* head_imput2: pointeur sur un bus1 utilisé en entrée2
bus1*head_output: pointeur sur un bus1 utilisé en sortie
char* label: nom de l'unité fonctionnelle
uf* next: pointeur sur unité fonctionnelle
char type:type de l'unité fonctionnelle (unaire ou binaire)

registre :
int num_reg: numéro de registre.
transistor* imput_head : pointeur vers la liste des transistors utilisés en entrée du registre

transistor* output_head : pointeur vers la liste des transistors utilisés en sortie du registre
bus1* head_imput : pointeur sur un Bus1 utilisé en entrée.
bus1* head_output : pointeur sur un Bus1 utilisé en sortie.
registre * next : pointeur sur un registre.

fil :
int num_fil : numéro de fil
fil*next : pointeur sur une structure fil

bus :
int num_bus : numéro de bus
fil* head : pointeur vers la liste des fils de bus
bus* next : pointeur sur un bus

bus1 :
bus*ptr : pointeur sur un Bus
int nbr_bit : nombre de bit de bus
transistor*head : pointeur sur une liste des transistors
char*signal : chaîne de caractères représentant un signal
bus1*next : pointeur sur un Bus1

condition :
char*cond : chaîne de caractères représentant une condition
cond*next : pointeur sur une structure condition

entrée :
int num_ent : numéro de l'entrée du circuit
entre*next : pointeur sur une structure entrée

circuit :
int nbr_entre : numéro représentant le nombre des entrées du circuit
Entre*tete : pointeur sur une structure entre
int sorte : numéro représentant la sortie du circuit
circuit*next : pointeur sur une structure circuit

info_ins :
char* num_ins : numéro représentant le numéro de l'instruction.
char* cond : chaîne de caractères représentant une condition.
int traite : Boolean.
Noeud* next : pointeur sur une structure info_ins.

instruction :
char* num_ins: numéro de l'instruction
instruction*next : pointeur sur une structure instruction

ls :
char car : un caractère représentant une condition d'entrée
int num : numéro de caractère
I s* next : pointeur sur une structure ls

chaine :
char*etat : chaîne de caractères représentant une condition
ch* next : pointeur sur une structure chaîne

3.4.2. Structures des fichiers :

Nous allons décrire dans ce qui suit certains fichiers utilisés en entrée par nos algorithmes et ceux qui sont générés après l'exécution. Notons que la notation "ABCD" veut dire : écrire dans le fichier concerné la chaîne de caractères ABCD.

- Structure d'un enregistrement du fichier généré par la tâche d'**allocation :**

"Cycle"	numéro de cycle					
Numéro d'instruction	Rs	:=	R_i	op	R_j	condition
	Br_k					
	Br_l					
	Bf_m					

Numéro d'instruction	Rs	:=	op	R_i	condition
	Br_k				
	Bf_m				

où :

numéro de cycle : indique le numéro du cycle dans lequel sont exécutées les instructions indiquées dans les champs *Numéro d'instruction*
Rs : indique le registre dans lequel le résultat de l'opération "op" est stocké
R_i : indique le registre contenant l'opérande 1 de l'opération "op" si elle est binaire, et contenant l'opérande de "op" si elle est unaire
R_j : indique le registre contenant l'opérande 2 de l'opération "op" si elle est binaire
Op : indique le nom de l'unité fonctionnelle qui va exécuter l'opération
Br_k : indique le bus sur lequel se fait le transfert du contenu de R_i vers l'unité fonctionnelle "op"
Br_l : indique le bus sur lequel se fait le transfert du contenu de R_j à l'unité fonctionnelle "op"

Bf_m : indique le bus sur lequel se fait le transfert du résultat de l'opération "op" vers le registre Rs

Les fichiers des résultats permettant de décrire la partie opérative sont les suivants:

Structure d'un enregistrement du fichier *bus*:

"Bus en entrée"
Numéro de bus | liste de fils
"Bus en sortie"
Numéro de bus | liste de fils

Numéro de bus : indique le numéro de bus sur lequel se fait le transfert des données
Liste des fils : indique les numéros des fils constituant le bus indiqué dans le champ "numéro de bus"

Structure d'un enregistrement du fichier *registre* :

"R"Numéro de registre | liste 1 | liste 2

Numéro de registre : indique le numéro de registre
Liste1 : indique la liste des numéros affectés aux sources des transistors utilisés en entrée du registre indiqué dans le champ "numéro de registre"
Liste2 : indique la liste des numéros affectés aux sources des transistors utilisés à la sortie du registre indiqué par le champ "numéro de registre "

Structure d'un enregistrement du fichier *unité fonctionnelle uf* :

Nuf | liste 1 | liste 2

Nuf : indique le nom de l'unité fonctionnelle
Liste1 : liste des numéros affectés aux sources des transistors utilisés en entrée de l'unité fonctionnelle "Nuf"
Liste2 : liste des numéros affectés aux sources des transistors utilisés à la sortie de l'unité fonctionnelle "Nuf"

Structure d'un enregistrement du fichier *circuit* :

"Circuit ou " liste1 sortie
ou :
"Circuit inverseur " liste1 sortie

Liste1 : liste des numéros des signaux utilisés en entrée du circuit "ou " ou du circuit "inverseur"
Sortie : indique la sortie du circuit ("ou" ou "inverseur")

Structure d'un enregistrement du fichier *netlist* :

S G D type signal
S : source du transistor
G : grille du transistor
D : drain du transistor
Type : type du transistor
Signal : l'ensemble des signaux issus de l'unité de commande qui contrôlent la grille du transistor considéré

Nous décrivons ci-après les différents algorithmes que nous avons conçus pour la génération des résultats à partir du fichier d'allocation. Ces fichiers-résultats sont précisément les fichiers bus, registre, circuit, unité fonctionnelle, et netlist.

3.4.3. Les algorithmes de génération des sources et drains des transistors de commande :

inser_reg(R,nbr) :
Cette fonction sert à créer un registre R de *nbr* bits et à l'insérer dans la liste des registres.

inser_busR(B,nbr) :
Cette fonction sert à créer un bus B de *nbr* bits et à l'insérer dans la liste des bus.

retourner_nuf(operation):
Cette fonction sert à créer une unité fonctionnelle et à l'insérer dans la liste des unités fonctionnelles (dans cette fonction, il est tenu compte de l'utilisation exclusive des unités fonctionnelles –cf 3.3.5-).

Les fonctions **Inser_entre_registre_bus(R,B,n,signal)**,
Inser_sortie_registre_bus(R,B,signal) ,
inser_entre_uf_bus(U,B,E,signal) ,
inser_sorte_uf_bus(U,B,signal)
permettent de résoudre le problème de conflit de bus par l'insertion des interrupteurs en entrée et à la sortie de registres et unités fonctionnelles considérés.

Algorithme 3-1 : // Connecter les entrées des registres aux bus

Inser_entre_registre_bus(R, B,n,signal) :

```
Début
 Localiser le registre R
 Si R a déjà utilisé en entrée le bus B alors
   mettre à jour son signal de commande et son nombre de bits et sortir
 Sinon
   ajouter le bus B à la liste contenant les bus utilisés en entrée par R et insérer
    son signal de commande signal et son nombre de bits n.
 Fin si
 si le bus B est le premier bus utilisé par R en entrée alors sortir.
  sinon
     Si B est le deuxième bus utilisé par R en entrée alors
        insérer à tous les bus utilisés par R en entrée des interrupteurs dont le
        nombre est égal à leur nombre de bits.
       Sinon
        insérer à B des interrupteurs dont le nombre est égal à n.
     Fin de si.
 Fin de si
Fin.
```

Algorithme 3-2 : // Connecter les sorties des registres aux bus
Inser sortie registre bus(R,B,signal) :

```
Début
  Localiser le registre R
 Si R a déjà utilisé en sortie le bus B alors
 |Mettre à jour  son signal de commande  et sortir
  Sinon
    Ajouter le bus B à la liste contenant les bus utilisés  en sortie par R et
    Insérer son signal de commande signal
 Fin de si.
 | Si le bus B est le premier bus utilisé par R en sortie alors  sortir
  Sinon
        Si B est le deuxième bus utilisé par R en sortie  alors
            insérer à tous les bus utilisés par R en sortie, des interrupteurs dont le
            nombre est égal à nombre de bit (utilisée comme variable globale).
          Sinon
              insérer à B des interrupteurs dont le nombre est égal à nombre de
              bits (variable globale)
       Fin de si
Fin de si
Fin.
```

Algorithme 3-3 // Connecter les entrées des unités fonctionnelles aux bus *inser_entre_uf_bus(U,B,E,signal) :*

```
Début
 Localiser l'unité fonctionnelle U
 Si E=2 aller à 1
   Sinon
     Si U a déjà utilisé en entrée1 le bus B alors
       Mettre à jour son signal de commande (en prenant en compte
        signal) et sortir
       Sinon
        Ajouter le bus à la liste contenant les bus utilisés en entrée1 par U et
        insérer son signal de commande signal
       Fin de si
     Si le bus B est le premier bus utilisé par U en entrée1 alors
       sortir.
       Sinon
        Si le bus B est le deuxième bus utilisé par U en entrée1 alors
           insérer à tous les bus utilisés par U en entrée1 des interrupteurs dont
           le nombre est égal à nombre de bit(variable globale) et sortir.
         Sinon
           insérer au bus B des interrupteurs dont le nombre est égal à nombre
            de bits (variable globale) et sortir.
         Fin de si.
    Fin de si.
Fin de si.
1 :Si U a déjà utilisé en entrée2 le bus B alors
       mettre à jour son signal de commande(en prenant en compte
       signal) et sortir.
  Sinon
       ajouter le bus à la liste contenant les bus utilisés en entrée2 par U et
       insérer son signal de commande signal.
 Fin de si.
 Si le bus B est le premier bus utilisé par U en entrée2 alors sortir.
   Sinon
      Si le bus B est le deuxième bus utilisé par U en entrée2 alors
        insérer à tous les bus utilisés par U en entrée2 des interrupteurs dont le
        nombre est égal à nombre de bits (variable globale).
      sinon
         insérer au bus B des interrupteurs dont le nombre de bits est égal à
         nombre de bits (variable globale).
      Fin de si.
Fin de si.
Fin.
```

Algorithme 3-4 // Connecter les sorties des unités fonctionnelles aux bus *inser_sorte_uf_bus(U,B,signal):*

```
Début
 Localiser l'unité fonctionnelle U.
 Si U a déjà utilisé en sortie le bus B alors
        mettre à jour son signal de commande et sortir.
 Sinon
       ajouter le bus B à la liste contenant les bus utilisés en sortie par U et
       insérer son signal de commande signal.
 Fin de si.
 Si le bus B est le premier bus utilisé par U en sortie alors
       sortir.
  Sinon
    Si B est le deuxième bus utilisé par U en entrée alors
      insérer à tous les bus utilisés par U en sortie des interrupteurs dont le
      nombre est égal à nombre de bits(variable globale).
     Sinon
      insérer à B des interrupteurs dont le nombre est égal à nombre de
       bits (variable globale).
    Fin de si.
 Fin de si.
Fin.
```

La fonction *MAJbus* :

Cette fonction sert a énumérer les fils des bus par des numéros différents du fait que les fils constituent des nœuds différents.

Algorithme 3-5 : // affectation de nœuds aux fils du même bus

MAJbus() :

```
Var cpt:entier
        fil:Fil
Début
    Initialiser la valeur du compteur cpt par le résultat retourner par la fonction
     Traiter_signaux()
    Pour chaque bus b en entrée
     Faire
        fil=tête de b
        Tant que (la liste des fils de b n'est pas vide)
         Faire
            fil->numéro :=cpt
            cpt:=cpt +1
            fil:= fil->svt
        Fait
     Fait

    Pour chaque bus b en sortie
    Faire
        fil :=tête de b
        Tant que (la liste des fils de b n'est pas vide)
         Faire
            fil ->numéro :=cpt
            cpt:=cpt +1
            fil:= fil->svt
        Fait

     Fait

Fin
```

La fonction *MAJsource* :

Cette fonction énumère les sources des transistors des registres et des unités fonctionnelles par des numéros différents (par ordre croissant).

Notons que les sources des transistors reliés aux registres et aux unités fonctionnelles sont énumérées d'une manière croissante, du poids le plus faible au poids le plus fort.

Algorithme 3-6://Mise à jour des sources des transistors

MAJsource() :

```
Var :  cpt :entier
       R :pointeur sur registre
       U : pointeur sur unité fonctionnelle
       LR : pointeur sur liste de registre
       Luf :liste des unités fonctionnelles
       Lt : pointeur sur liste de transistors
       T : pointeur sur transistor
Début
    Initialiser cpt par la valeur retournée par la fonction MAJbus()
    Tant que (LR n'est pas vide)
      Faire
        R :=LR
        Lt :=R->tete_entre      (liste des transistors en entrée)
         Tant que(Lt n'est pas vide)
            Faire
              T :=Lt
               T ->source->nœud :=cpt
                cpt :=cpt+1
               Lt :=Lt->svt
            Fait
         Lt :=R->tete_sortie      (liste des transistors en sortie)
         Tant que(Lt n'est pas vide)
            Faire
              T :=Lt
               T ->source->nœud=cpt
                cpt :=cpt+1
               Lt :=Lt->svt
            fait
           LR :=LR->svt
      fait
      Tant que (Luf n'est pas vide)
      Faire
        U :=Luf
        Lt :=U->tête_entrée      (liste des transistors en entrée)
         Tant que (Lt n'est pas vide)
            Faire
              T :=Lt
               T ->source->nœud :=cpt
                cpt :=cpt+1
               Lt :=Lt->svt
            Fait
```

```
        Lt :=U->tête_sortie      (liste des transistors en sortie)
        Tant que (Lt n'est pas vide)
          Faire
            T :=Lt
            T ->source->nœud :=cpt
            cpt :=cpt+1
            Lt :=Lt->svt
          Fait
        Luf :=Luf->svt
    Fait
Fin
```

La fonction *Majdrain* :

Cette fonction met à jour les drains des transistors de chaque registre et chaque unité fonctionnelle, et ce, en leur affectant les numéros des fils des bus auxquels ce registre ou cette unité fonctionnelle est raccordé. Cette fonction est utilisée dans le cas où il ne peut exister de possibilité de conflit.

Les valeurs des drains des transistors connectant les registres et les unités fonctionnelles aux bus sont celles affectées aux fils des bus considérés.

Algorithme 3-7 // Mise à jour des drains des transistors
MAJdrain() :

```
Début
 Considérer  la liste des registres
 Prendre R  le premier registre de cette liste
1:Si R utilise un seul bus en entrée alors
|          connecter le bus avec les entrées du registre en respectant
|          le poids fort et le poids faible des bits
 Fin de si
 Si R utilise un seul bus en sortie alors
|          Connecter le bus avec les sorties du registre en respectant le poids
|          fort et le poids faible des bits
Fin de si.
 S'il  y a un registre suivant dans la liste des registres alors
|        R ← ce registre ; aller à 1
Fin de si.
Considérer la liste des unités fonctionnelles
Prendre  U  la première unité fonctionnelle
3 :Si U est unaire alors
|     aller à 2.
 Sinon
|      Si U utilise un seul bus avec la 2ème entrée  alors
|      | connecter le bus avec l'entrée 2 de U, en respectant le poids
|      | fort et le poids faible des bits
|      Fin de si.
|  2 :  Si U utilise un seul bus avec la 1ère entrée alors
|      | connecter le bus avec l'entrée 1 de U, en respectant le
|      | poids fort et le poids faible des bits
|       Fin de si.
|      Si U utilise un seul bus en sortie alors
|      |      Connecter le bus  avec la sortie de U, en respectant le
|      |       poids fort et le poids faible des bits
|      Fin de si.
|    S'il y a une unité fonctionnelle suivante dans la liste des unités fonctionnelles
|    | alors
|    |   U ← cette unité fonctionnelle ;  aller à 3.
|   Fin de si.
 Fin de si.
Fin.
```

Fonction traiterP() :

Pour ne pas alourdir inutilement nos structures de données, celles-ci ne considèrent que les transistors de type N. En effet, les nœuds des transistors de type P sont déduits directement à partir de ceux des transistors de type N. Cette déduction des noeuds est assurée par la fonction traiterP().

Algorithme 3.8://Le programme principal :

Algorithme du programme principal :
Le programme principal fait appel aux fonctions décrites précédemment.
Début

Ouvrir le fichier d'*ordonnancement* pour lecture.
Simplifier le fichier d'*ordonnancement* en faisant appel à la fonction **optimiser()**.
Générer un automate à partir du fichier d'*ordonnancement* simplifié en faisant appel à la fonction **transition()**.
Générer à partir du fichier *automate* le fichier *table_sgnaux* en faisant appel à la fonction **traiter_signaux()**
Ouvrir le fichier d'*allocation* pour lecture.
1: Lire l'enregistrement courant (le format de l'enregistrement a été décrit précédemment)
Extraire le numéro de registre R_i qui contient l'opérande 1.
Extraire le numéro de registre R_j qui contient l'opérande 2.(selon le type de l'opération : unaire ou binaire).
Extraire le numéro de l'unité fonctionnelle U qui va effectuer l'opération.
Extraire le numéro de l'opération.
Extraire son signal en faisant appel à la fonction **retourner_signal()**
// cette fonction est décrite dans le chapitre 4
Extraire le numéro du bus br_k sur lequel le transfert des données de R_i vers l'unité U va se faire.
Extraire le numéro du bus br_l sur lequel le transfert de R_j vers l'unité U va se faire (selon le type de l'opération).
Extraire le numéro du registre R_s dans lequel le résultat de l'opération sera stocké.
Extraire le numéro du bus bf_m sur lequel le résultat de l'opération sera transféré vers le registre R_s.
Insérer dans la liste des registres les trois registres R_i, R_j et R_s en faisant appel à la fonction **inser_reg()**
Insérer dans la liste des bus les bus br_k et br_l connectés respectivement aux sorties des registres R_i et R_j , en faisant appel à la fonction **inser_sortie_reg_bus()**
Insérer le bus bf_m connecté à l'entrée du registre R_s en faisant appel à la fonction **inser_entre_registre_bus()**

Insérer le bus br_k connecté à l'entrée 1 de l'unité fonctionnelle U en faisant appel à la fonction **insert_entre_uf_bus()**
Insérer le bus br_l connecté à l'entrée 2 de l'unité fonctionnelle U en faisant appel à la fonction **insert_entre_uf_bus()** (dans le cas d'une opération binaire)
Insérer le bus bf_m connecté à la sortie de l'unité fonctionnelle U en faisant appel à la fonction **inser_sorte_uf_bus()**
Si non fin de fichier d'*allocation*, aller à **1**.
Fermer le fichier d'*allocation*
Faire appel à **MAjbus()**
Faire appel à **MAjsource()**
Faire appel à **MAJdrain()**
Faire appel à **MAJgate()**
Faire appel à **traiterP()**
Faire appel à **imprimer()**
Fin.

Nous allons illustrer ces différentes procédures par l'exemple 3.7 suivant. Quoique notre programme est général, nous considérons dans cet exemple des registres à 1 bit, et ce, afin de faciliter au lecteur la compréhension de notre travail.

3.5. Exemple d'illustration :

Exemple 3.7 :

3.5.1. Fichiers de données :

Soit le fichier d'ordonnancement suivant :

```
cycle 1 :
-----------------------------------------------------------------
1 : x=aorb [ not(A)andnot(B) ]
1 : x=aorb [ not(A)andB ]
1 : x=aorb [ Aandnot(B) ]
1 : x=aorb [ AandB ]
-----------------------------------------------------------------

cycle 2 :
-----------------------------------------------------------------
2 : z=xandy [ not(A)andB ]
2 : z=xandY [ Aandnot(B) ]
2 : z=xandy [ AandB ]
3 : j=oandj [ not(A)andnot(B) ]
3 : j=oandj  [ Aandnot(B) ]
-----------------------------------------------------------------
```

```
cycle 3 :
-----------------------------------------------------------------
1 : x=aorb [ not(A)andnot(B) ]
4 : y=aorz [ not(A)andB ]
4 : y=aorz [ Aandnot(B) ]
-----------------------------------------------------------------

cycle 4 :
-----------------------------------------------------------------
5 : x=a*4 [ not(A)andnot(B) ]
5 : x=a*4 [ not(A)andB ]
5 : x=a*4 [ Aandnot(B) ]
-----------------------------------------------------------------

cycle  5 :
-----------------------------------------------------------------
6 : z=anandb [ not(A)andnot(B) ]
7 : t=anorb  [ not(A)andnot(B) ]
7 : t=anorb  [ not(A)andB ]
-----------------------------------------------------------------

cycle 6 :
-----------------------------------------------------------------
8 : x=aandu [ not(A)andB ]
-----------------------------------------------------------------

cycle 7 :
-----------------------------------------------------------------
9 : y=a*h [ not(A)andB ]
-----------------------------------------------------------------
```

Le fichier d'allocation associé est le suivant :

```
cycle 1
     1 [r4] := [r5] or [r6] 1
        br4
        br3
        bf2
cycle 2
     2 [r3] := [r4] and [r2] BorA
        br3
        br4
        bf2

     3 [r1] := [r4] and [r6] not(B)
        br2
        br1
        bf1


cycle 3
     1 [r4] := [r5] or [r6] not(A)andnot(B)
```

```
        br4
        br3
        bf2

     4 [r2] := [r3] or [r1] Aandnot(B)or (not(A)andB)
        br1
        br2
        bf1
cycle 4
     5 [r6] := [r4] * [r1] not(B)or not(A)
        br2
        br1
        bf2
cycle 5
     6 [r4] := [r2] nand [r6] not(A)and not(B)
        br4
        br3
        bf2

     7 [r5] := [r1] nor [r3] not(A)
        br2
        br1
        bf1

cycle 6
     8 [r6] := [r5] and [r4] not(A)and B
        br4
        br3
        bf1
cycle 7
     9 [r1] := [r4] * [r6] not(A)and B
        br2
        br1
       bf1
```

Les fichiers qui sont générés par notre programme sont cités ci-dessous. Notons que d'autres fichiers sont aussi générés, et seront présentés au chapitre 4.

3.5.2. Fichiers de résultats partiels :

Fichier *bus*:

```
BUS EN ENTREE

br1:
     19
br2:
     20
```

```
br3:
     21
br4:
     22

BUS EN SORTIE
bf1:
     23   24
bf2:
     25   26
```

Fichier *registres*:

```
R:1
     27   28     %%  29
R:2
     30     %%  31
R:3
     32     %%  33
R:6
     34   35     %%  36
R:5
     37     %%  38
R:4
     39     %%  40
```

Fichier *unités fonctionnelles* :

```
nor:
     41   42     %%  43
nand:
     44   45     %%  46
*:
     47   48     %%  49  50
and:
     51   52     %%  53
and:
     54   55     %%  56
or:
     57   58     %%  59
```

Fichier *netlist*:

```
*************************************************
S    G     D  TYPE  SIGNAL
*************************************************
27         23  N
28         24  N
30         23  N
31         22  N
32         25  N
33         19  N
```

37	23	N
38	22	N
39	25	N
41	20	N
42	19	N
43	23	N
44	22	N
45	21	N
46	25	N
47	20	N
48	19	N
53	23	N
54	21	N
55	22	N
56	25	N
27	23	P
28	24	P
30	23	P
31	22	P
32	25	P
33	19	P
37	23	P
38	22	P
39	25	P
41	20	P
42	19	P
43	23	P
44	22	P
45	21	P
46	25	P
47	20	P
48	19	P
53	23	P
54	21	P
55	22	P
56	25	P
29	19	N
29	20	N
34	23	N
34	25	N
35	26	N
36	19	N
36	21	N
40	20	N
40	21	N
49	23	N
50	24	N
49	25	N
50	26	N

51	22	N
51	20	N
52	21	N
52	19	N
57	19	N
57	22	N
58	20	N
58	21	N
59	23	N
59	25	N
29	19	P
29	20	P
34	23	P
34	25	P
35	26	P
36	19	P
36	21	P
40	20	P
40	21	P
49	23	P
50	24	P
49	25	P
50	26	P
51	22	P
51	20	P
52	21	P
52	19	P
57	19	P
57	22	P
58	20	P
58	21	P
59	23	P
59	25	P

3.5.3. Commentaires sur les résultats:

1- Les colonnes *G* et *signal* du fichier *netlist* concernent les grilles des transistors de commande et seront expliqués au chapitre4.

2- Le nombre de fils des bus br1, br2, br3, br4 est égal 1 puisqu'on a considéré que le nombre de bits est égal à 1.

3- Le nombre de fils des bus bf1, bf2 est égal 2 puisque ils sont utilisés pour transférer le résultat d'une multiplication de 2 opérandes à 1 bit (voir les cycles 4 et 7 dans le fichier d'allocation).

4- Le nombre des unités fonctionnelles "and" dans le fichier d'allocation est 3 (cycles 2, 2 et 6), mais le nombre des unités "and" générées effectivement est 2. Ceci est dû au fait que dans le cycle 2, les 2 opérations *and* peuvent être exécutées en même temps au 2ème cycle car les 2 conditions logiques 'BorA' et 'not(B)' peuvent être vraies en même temps (B=0 et A=1). Ceci implique l'utilisation de deux unités fonctionnelles "and" distinctes. Dans le cycle 6, une opération "and" y est ordonnancée. L'une des 2 unités fonctionnelles *and* nécessaires au 2ème cycle peut être utilisée sans aucun problème de conflit du fait que l'opération 8 est exclusive avec les opérations 2 et 3.

5- Le nombre des unités fonctionnelles "or" dans le fichier d'allocation est 3, mais le nombre de ces unités effectivement générées est 1. Ceci est dû au fait que dans le cycle1, une seule opération "or" est exécutée. Dans le cycle 3, 2 opérations de ce type y sont ordonnancées, mais du fait que les conditions logiques "*not(A) and not(B)*" et "*A and not(B) or (not(A)andB*) " ne peuvent jamais être vraies en même temps, les opérations 1 et 4 sont donc exclusives, et alors une seule unité fonctionnelle *or* suffit. Les opérations ordonnancées dans des cycles différents ne pouvant s'exécuter en parallèle, une seule unité *or* permettra l'exécution des 3 opérations *or*.

6- Le même raisonnement que celui qui vient d'être décrit est fait pour l'unité "*".

Cette optimisation du nombre d'unités fonctionnelles nécessaires effectivement est garantie par la fonction **retourner_nuf()** qui renvoie toujours le numéro de l'unité libre (elle utilise même des routines de simplification pour pouvoir déterminer si deux conditions logiques peuvent être vraies en même temps).

7- Pour la multiplication de deux opérandes à 1 bit chacun, le résultat est sur 2 bits. Il y a 2 bits en entrée (1 bit pour chaque opérande) et 2 bits en sortie. Le résultat étant rangé soit dans R1 (cycle 7), soit dans R6 (cycle 4), il y'a alors 2 bits aux entrées de chacun de ces 2 registres.

8- Toutes les unités fonctionnelles dans cet exemple sont binaires. Il y'a alors 2 bits en entrée et un bit en sortie pour chaque unité fonctionnelle, sauf pour la sortie d'une opération de multiplication (2 bits en sortie).

9- Puisque R1 est connecté au bus br2 dans les cycles 3 et 5 et au bus br1 dans le cycle 4, il y' a un interrupteur à la sortie de R1 qui sert à éviter le problème de

conflit de bus (cet interrupteur sert à transférer exclusivement, les données de R1 sur le bus br2 dans les cycles 3 et 5, et sur le bus br1 dans le 4ème cycle).

10- Le même raisonnement que celui qui vient juste d'être décrit est fait pour les registres R4 (cycles 2, 4, 6 et 7, bus br2 et br3) et R6 (cycles 1, 2, 3, 4, 5 et 6,7 bus br1, br3, bf2 et bf1), ainsi que pour les unités fonctionnelles "*" (cycles 4 et 7, bus br1, br2, bf1 et bf2), "or" (cycles 1 et 3, bus br3, br4, bf1 et bf2) et "and" (cycles 2 et 6, bus br3, br4, bf1 et bf2).

11- L'affectation des nœuds commence par celles des signaux. Du fait qu'il y'a 18 signaux (ceux-ci sont expliqués au chapitre 4), la numérotation des fils des bus commence par 19.

12- La numérotation des sources des transistors commence par 27 puisqu'il y'a 18 signaux et 8 fils de bus (27=18+8+1).

13-Le drain de tout transistor directement connecté à un registre ou à une unité fonctionnelle reçoit le numéro du fil du bus auquel il est directement connecté (voir Fig. 3.9).

14- La règle de connexion des registres aux unités fonctionnelles selon les valeurs des poids des bits est respectée (voir Fig.3.9). Pour la multiplication par exemple, nous avons ce qui suit :

49 (poids faible de l'unité de multiplication) → 34 (poids faible de R6)
50 (poids fort de l'unité de multiplication) → 35 (poids fort de R6)
49 (poids faible de l'unité de multiplication) →27 (poids faible de R1)
50 (poids fort de l'unité de multiplication) → 28 (poids fort de R1).

3.6. Conclusion :

Nous avons présenté dans ce chapitre la détermination des sources et drains des transistors de commande d'exécution des opérations d'une partie opérative. Nous avons énuméré les différents problèmes et montré comment nous les avons résolus. La détermination des grilles de ces transistors de commande nécessitant un traitement particulier, celle-ci est traitée dans le chapitre qui suit immédiatement.

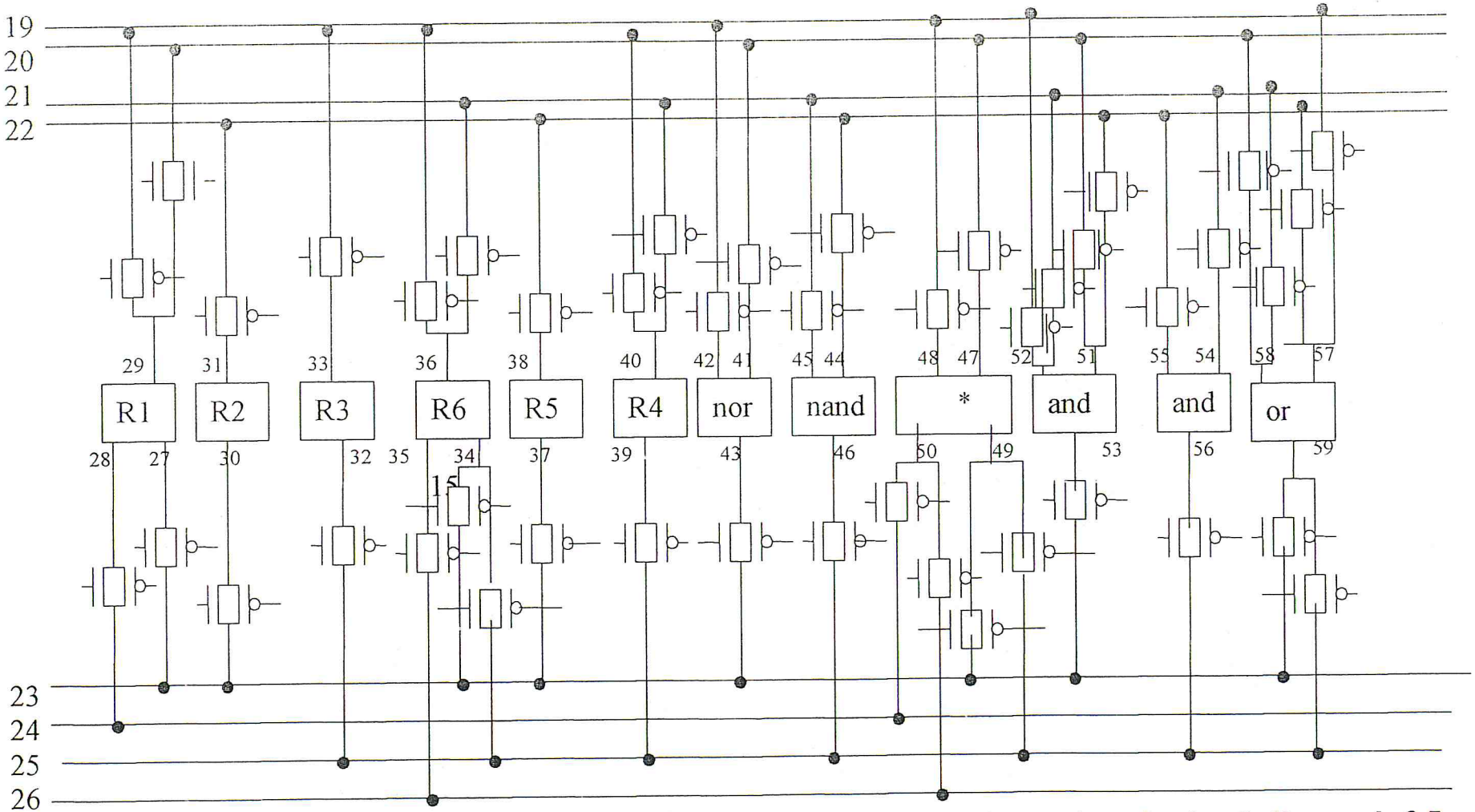


Fig. 3.9. Détermination des sources et drains pour la partie opérative de l'exemple 3.7

CHAPITRE 4

Affectation des nœuds aux grilles des transistors de commande

4.1. Introduction :

Dans le précédent chapitre, nous avions discuté de la détermination des sources et drains des transistors de commande. Ces nœuds permettent de montrer comment les registres et les unités fonctionnelles sont connectés aux bus de la partie opérative pour le transfert adéquat des différentes données entre ces différentes ressources physiques. En d'autres termes, ces sources et drains répondent à la question "*de quel registre ou unité fonctionnelle se fait tel transfert, vers quelle ressource doit-il se faire, et sur quel bus ?* " Toutefois, la réponse au problème n'est que partielle, du fait qu'on ne sait toujours pas répondre à la question "*à quel moment tel transfert doit-il se faire ?* ". Le présent chapitre a alors pour but de répondre à cette question.

4.2. Simplification des conditions d'exécution d'une opération:

Nous rappelons que la détermination des nœuds des transistors de commande se fait après les tâches d'ordonnancement et d'allocation. Dans un flot de données contrôlées, il existe un certain nombre de chemins de données à ordonnancer. L'ordonnancement de chacun de ces chemins, puis le regroupement des différents résultats de l'ordonnancement conduit à un fichier du type de celui montré par la figure 4.1 et dont le format d'enregistrement est le suivant :

Structure d'un enregistrement du fichier d'ordonnancement :

"Cycle" : numéro de cycle
numéro d'instruction instruction condition
où :

numéro de cycle : indique le numéro du cycle dans lequel sont ordonnancées les instructions indiquées dans les champs suivants.
numéro d'instruction : indique le numéro de l'instruction.
instruction : indique l'instruction.
condition : indique la condition sous laquelle l'instruction est exécutée.

```
cycle 1 :
-----------------------------------------------------
1 : a=x*y [ AandB ]
1 : a=x*y [ not(A)andnot(B) ]
1 : a=x*y [ not(A)andB ]
1 : a=x*y [ Aandnot(B) ]
-----------------------------------------------------
cycle 2 :
-----------------------------------------------------
2 : z=a-b [  B ]
2 : z=a-b [ not(B) ]
-----------------------------------------------------
```

```
cycle 3 :
-------------------------------------------------
3 : t=not(b) [ not(A)andnot(B) ]
3 : t=not(b) [ not(A)andB ]
3 : t=not(b) [ Aandnot(B) ]
-------------------------------------------------
cycle 4 :
-------------------------------------------------
4 : r=not(t) [ not(A)andnot(B) ]
4 : r=not(t) [ Aandnot(B) ]
-------------------------------------------------
cycle 5 :
-------------------------------------------------
5 :  x=a*c  [ Aandnot(B) ]
-------------------------------------------------
```

Fig.4.1. Exemple de fichier d'ordonnancement.

Dans cet exemple, nous constatons que l'instruction 1 sera exécutée dans le cycle 1 si l'une de ces quatre conditions est vraie: [A and B], [not(A) and not(B)], [not(A) and B], [A and not(B)].
Elle sera donc exécutée si la condition (A and B) **or** (not(A) and not(B)) **or** (not(A) and B) **or** (A and not(B)) est vraie, c'est-à-dire si la condition logique '1' est vraie, donc dans tous les cas.
De même, l'instruction 3 sera exécutée dans le cycle 3 si la condition (not(A) and not(B)) **or** (not(A) and B) **or** (A and not(B)) est vraie, donc si (not(A) or not(B)) est vraie.
Par conséquent, avant de déterminer *quand* une instruction donnée est exécutée, il y'a lieu de déterminer ses conditions d'exécution simplifiées, et ce, afin d'obtenir une architecture du circuit intégré plus optimale, comme nous le verrons ultérieurement.
Pour ce faire, nous avions utilisé *ESPRESSO*[9], un outil de simplification d'équations logiques. L'utilisation de cet outil conduit alors à un autre fichier d'ordonnancement dont les conditions d'exécution des instructions peuvent être éventuellement simplifiées (Cf Fig.4.2).

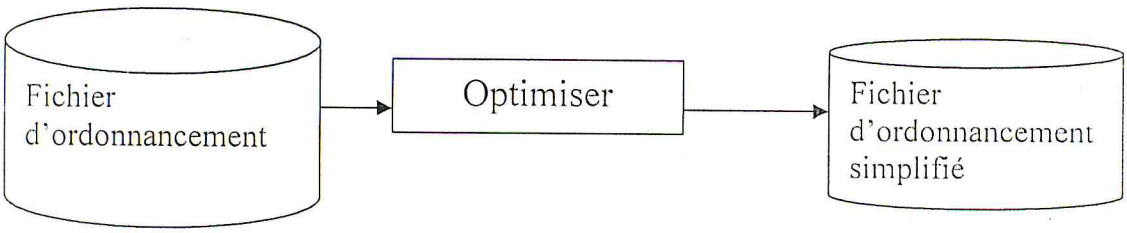


Fig.4.2. Simplification des conditions d'exécution des instructions

La génération de ce dernier fichier d'ordonnancement se fait par la fonction *optimiser()* présentée ci-après. Elle consiste à :

- concaténer les conditions qui ont le même numéro d'instruction dans le même cycle par des « or » dans une chaîne *condition*,
- simplifier *condition* en utilisant l'outil ESPRESSO

Algorithme 4-1:// Simplification des conditions d'exécution des instructions

Optimiser( ):

```
Var i, j : entier.
    C : chaîne de caractères.
Début
        Ouvrir le fichier d'ordonnancement en lecture.
        Ouvrir le fichier d'ordonnancement simplifié en mode écriture.
  Tant que (non fin de fichier d'ordonnancement)
     Faire
        |   Lire (du fichier d'ordonnancement, numéro de cycle j)
        |   Ecrire (dans fichier d'ordonnancement simplifié,
        |           numéro de cycle j)
        |   Lire les numéros des instructions du cycle  j
        |  Pour chaque numéro instruction i
        |  Faire
        |    | Considérer les conditions des instructions dont
        |    |   le numéro est égal à i.
        |    | Concaténer ces conditions par des or  en une condition C
        |    | Simplifier la condition C (appel à ESPRESSO)
        |    | Ecrire (dans fichier d'ordonnancement simplifié, i)
        |    | Ecrire (dans fichier d'ordonnancement simplifié, C)
        |
        |  Fait
     Fait
        Fermer le fichier d'ordonnancement.
        Fermer le fichier le fichier d'ordonnancement simplifié.
Fin
```

Le fichier d'ordonnancement simplifié correspondant au fichier de la Fig.4.1 est celui indiqué dans la figure 4.3.

```
cycle 1
-------------------------------------------------------------
      1      [1]
-------------------------------------------------------------
cycle 2
-------------------------------------------------------------
      2     [1]
-------------------------------------------------------------
cycle 3
-------------------------------------------------------------
      3     [not(A)ornot(B)]
-------------------------------------------------------------
cycle 4
 ------------------------------------------------------------
      4     [ not(B)]
-------------------------------------------------------------
cycle 5
-------------------------------------------------------------
      5    [Aandnot(B)]
-------------------------------------------------------------
```

Fig.4.3. Fichier d'ordonnancement avec simplification des conditions d'exécution des instructions

4.3. Détermination de l'automate de commande d'exécution des instructions:

Nous avions dit en introduction de ce chapitre qu'il fallait déterminer les conditions d'exécution d'un transfert donné. Pour ce faire, nous utilisons le concept d'*automate* dans la mesure où celui-ci nous permet de déterminer les actions à faire dans un état donné de l'automate, les transitions d'états, et les conditions de ces transitions.

Notons que dans notre cas, l'automate est déduit à partir de l'ordonnancement des instructions incluses dans les différents chemins de données, dont chacun sera exécuté si la condition correspondante est vraie.

Dans un premier temps, l'état initial de l'automate est S_0. Il y'a alors transition vers l'état S_1 dans tous les cas, puisque pour toute condition, il existe au moins une instruction dans le premier cycle du chemin qui lui est associé. Pour un chemin donné, il y'a transition de l'état S_i ($i \geq 1$) à l'état :

- S_{i+1} s'il existe au moins une instruction ordonnancée dans le cycle S_{i+1}
- S_0 sinon

Ceci est illustré par la figure 4.4.

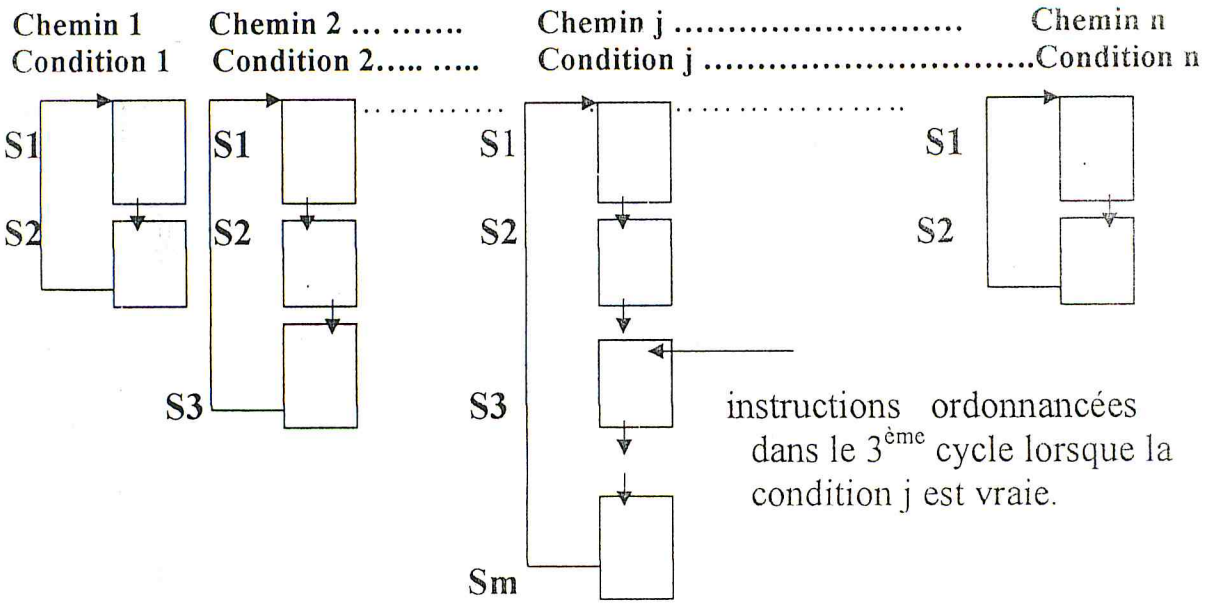


Fig.4.4. Transitions d'états dans l'ordonnancement de chemins.

De manière générale, il y'a transition de l'état :

- S_0 à l'état S_1 dans tous les cas
- S_i à S_{i+1} ($i \geq 1$) pour tout chemin contenant au moins une instruction dans le cycle S_{i+1} (la condition de transition est donc l'union des conditions d'exécution de tels chemins)
- S_i ($i \geq 1$) à S_0 pour tout chemin dont le nombre maximal de cycles est égal à S_i (la condition de transition à l'état S_0 est donc l'union des conditions d'exécution de tels chemins)

Avant de donner l'algorithme correspondant à la génération d'un automate à partir d'un fichier d'ordonnancement, nous donnons ci-après la structure d'un enregistrement du fichier *automate*:

Structure d'un enregistrement du fichier *automate*:

Etat courant [état suivant condition de transition [numéro d'instruction condition d'instruction]$^+$ "fin1"]$^+$ | état initial condition de retour "NOOP" condition de retour "fin1" | "fin2"

[]$^+$: occurrence pouvant se produire 1 une ou plusieurs fois
| | : occurrence pouvant se produire 1 ou aucune fois
Etat courant : état courant de l'automate
Etat suivant : état suivant de l'automate
Condition de transition : condition de transition de l'état courant à l'état suivant de l'automate

Numéro d'instruction : représente le numéro de l'instruction exécutée après le passage à l'état suivant.
Condition d'instruction : condition sous laquelle l'instruction est exécutée.
"Fin1" : indique la fin des instructions exécutées lors du passage à l'état suivant.
"Fin2" : indique la fin de l'enregistrement.
Condition de retour : indique la condition de transition à l'état initial
"NOOP" : indique qu'il n'y' a pas d'instruction à exécuter pour la transition à l'état initial

Le module permettant de générer le fichier *automate* correspondant à l'ordonnancement d'instructions (Cf Fig. 4.5) est implémenté par la fonction *transition()* que nous présentons ci-après :

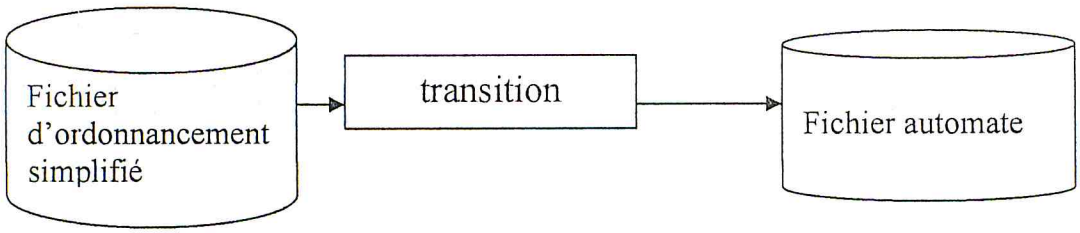


Fig.4.5. Génération de l'automate correspondant à un fichier d'ordonnancement

Algorithme 4-2: **//Génération de l'automate correspondant à un fichier d'ordonnancement**

Transition() :

```
Var : Z,T, R : chaîne de caractères
  Début
        Etat courant :=0
        Z :=1        //Condition de transition précédente.
        R :=1        // Condition de retour à l'état initial.
        T :=1        // Condition de transition.
        Ouvrir le fichier d'ordonnancement simplifié pour lecture.
        Ouvrir le fichier  automate pour écriture.
    Tant que (non fin du fichier d'ordonnancement simplifié)
       Faire
          Lire (du fichier d'ordonnancement simplifié, numéro de cycle).
          Etat suivant : = numéro de cycle
          concaténer toutes les conditions du cycle par or
          en une condition T.
          Simplifier la condition T (appel à ESPRESSO)
          Si T=R  alors
              Début
                  Ecrire (dans fichier automate, Etat courant,
                          Etat suivant, Condition T,
                          La liste des instructions avec leur
                          condition, "fin1 fin2").
                          Etat courant := numéro de cycle.
                          Z :=T
               Fin
            sinon
              Debut
                  R:=Z and not(T).
                  Ecrire (dans fichier automate, Etat courant,
                          Etat suivant, Condition c,

                  La liste des instructions avec leur condition,
                       " Fin1" , " 0", R, " NOOP",R, " Fin1 fin2")
                  Etat courant := numéro de cycle.
                  Z :=T.
               Fin
          Fin de si
        Fait
        Ecrire (dans fichier automate, "0", R, "NOOP",R, "fin1 fin2").
        Fermer le fichier d'ordonnancement simplifié et  automate
  Fin
```

A titre d'illustration, l'automate décrit dans la figure 4.6 est généré à partir du fichier d'ordonnancement de la figure 4.3.

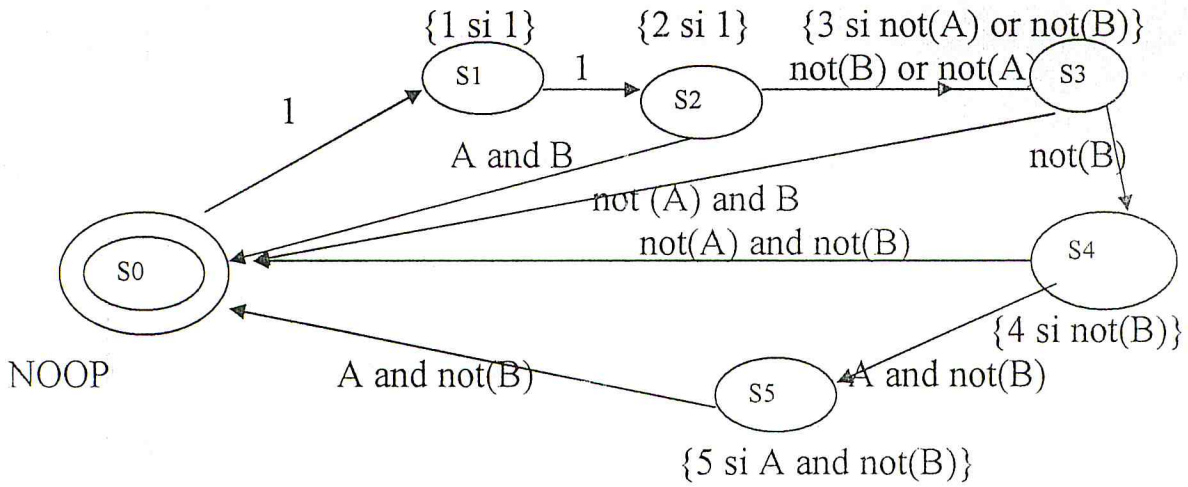


Fig.4.6. Automate correspondant au fichier d'ordonnancement de la figure 4.3

```
0 1 1 1 1 fin1 fin2
1 2 1 2 1 fin1 fin2
2 3 not(B)ornot(A) 3 not(B)ornot(A) fin1 0  BandA NOOP BandA
fin1 fin2
3 4 not(B) 4 not(B) fin1 0  not(A)andB NOOP not(A)andB fin1
fin2
4 5 Aandnot(B) 5 Aandnot(B) fin1 0  not(A)andnot(B) NOOP
not(A)andnot(B) fin1 fin2
5 0 Aandnot(B) NOOP  Aandnot(B) fin1 fin2
```

Fig.4.7. Exemple du fichier *automate*

La ligne "`2 3 not(B)ornot(A) 3 not(B)ornot(A) fin1 0  BandA NOOP BandA fin1 fin2`" signifie ce qui suit :
Si l'état courant est S_2, alors

- si la condition de transition "not(B) or not(A)" est vraie, alors l'état suivant est S_3, et l'instruction 3 est exécutée si la condition "`not(B) or not(A)`" est vraie
- si la condition de transition "B and A" est vraie, alors l'état suivant est S_0, et aucune exécution d'instruction n'est faite (NOOP)

Note : La condition d'exécution d'une instruction n'est pas toujours égale à la condition de transition de l'état courant à l'état suivant dans lequel elle est ordonnancée. En effet, supposons que dans l'exemple de la figure 4.3 une autre instruction (6) était ordonnancée avec l'instruction 4 au quatrième cycle comme indiqué ci-après :

```
cycle 4
------------------------------------------------------------------
      4     [ not(B)]
      6     [ A and not(B) ]
```

On obtiendrait alors l'automate suivant:

```
0 1 1 1 1 fin1 fin2
1 2 1 2 1 fin1 fin2
2 3 not(B)ornot(A) 3 not(B)ornot(A) fin1 0  BandA NOOP BandA
fin1 fin2
3 4 not(B) 4 not(B) 6 A and not(B) fin1 0  not(A)andB NOOP
not(A)andB fin1 fin2
4 5 Aandnot(B) 5 Aandnot(B) fin1 0  not(A)andnot(B) NOOP
not(A)andnot(B) fin1 fin2
5 0 Aandnot(B) NOOP  Aandnot(B) fin1 fin2
```

Fig.4.8. Autre exemple d'automate

Ainsi, la ligne "`3 4 not(B) 4 not(B) 6 A and not(B) fin1 0 not(A)andB NOOP not(A)andB fin1 fin2`" montre que si la condition d'exécution de l'instruction 6 (`A and not(B)`) est vraie, la condition de transition de l'état S_3 à l'état S_4 (`not(B)`) est forcément vraie, mais l'inverse est faux si la variable logique A est fausse. En d'autres termes, si on est dans l'état S_3 et que la condition not(B) est vraie, alors il y'a transition vers l'état S_4 et

- exécution des instructions 4 et 6 si la variable logique A est vraie
- uniquement l'exécution de l'instruction 4 si A est fausse

4.4. Génération de la table des commandes:

Une fois que l'automate est généré, les signaux de commande d'exécution des différentes opérations de la partie opérative en sont déduits. La table des signaux indiquée dans la figure 4.10 est une autre représentation de l'automate de la figure 4.7. Cette table est générée par la fonction **traiter_signaux()** à partir du fichier *automate* (Cf Fig.4.9) et est décrite ci-après.

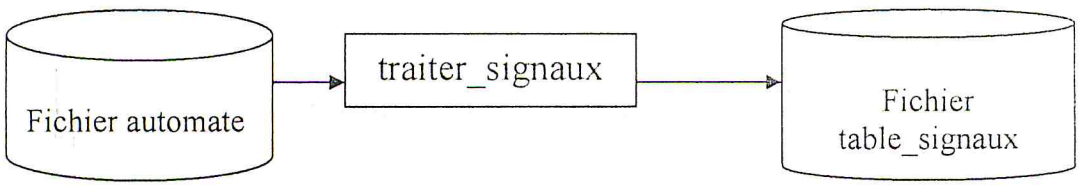


Fig.4.9. Génération du fichier *table_signaux*

La structure d'un enregistrement du fichier *table_signaux* est la suivante :

Structure d'un enregistrement du fichier table signaux :

Condition	état courant	état suivant	liste d'instructions	Signal de commande

Condition : indique la condition d'exécution des instructions énumérées dans le champ "liste d'instructions"

Etat courant : indique l'état courant de l'automate

Etat suivant : indique l'état suivant de l'automate

Liste d'instructions : indique les numéros des instructions exécutées lorsque la condition indiquée dans le champ "condition" est vraie et l'automate se trouve dans l'état indiqué dans le champ "état suivant"

Signal de commande : un signal de l'unité de commande permettant l'exécution des instructions énumérées dans le champ "liste d'instructions" lorsque l'automate se trouve dans l'état "état suivant" et que la condition associée est vraie

Nous décrivons maintenant l'algorithme implémentant la fonction *traiter_signaux()*.

Algorithme :4-3// Génération de la table des signaux de commande

traiter_signaux(cpt)

```
Var : etat_courant , etat_svt : chaîne de caractère.
      cpt :entier
Début
    Ouvrir le fichier automate pour lecture.
    Ouvrir le fichier table_signaux pour écriture.
    Ecrire (dans le fichier table_signaux, "condition etat_courant  etat_suivant
     instruction signal")
     Tant que (non fin automate)
      Faire
          Lire (du fichier automate , etat_courant).
          Tant que (chaîne lue ≠ "fin2")
             Faire
                Tant que (chaîne lue ≠ "fin1")
                   Faire
                      Lire (du fichier automate, etat_svt).
                      Sauter la condition de transition.
                      Grouper toutes les instructions qui ont la même
                      Condition de transition dans le même groupe G.
                      Pour chaque groupe G.
                         Faire
                            Générer un signal de sortie Fcpt.
                              cpt :=cpt+1
                               coder La condition d'exécution des
                               instructions.
                              Ecrire(dans table_signaux ,
                              La condition d'exécution des instructions,
                              Etat_courant, Etat_svt,
                               La liste des instructions du groupe G,
                               Fcpt)

                         Fait
                   Fait
             Fait
       Fait
    Fermer le fichier  automate
    Fermer le fichier  table_signaux
Fin
```

L'exemple de la figure 4.10 représente la table de commande correspondant à l'automate de la figure 4.7.

AB	etat_crt	etat_svt	instruction	signal
--	0	1	1	F 1
--	1	2	2	F 2
0-	2	3	3	F 3
-0	2	3	3	F 4
11	2	0	NOOP	F 6
-0	3	4	4	F 7
01	3	0	NOOP	F 8
10	4	5	5	F 9
00	4	0	NOOP	F 10
10	5	0	NOOP	F 11

Figure 4.10. Table de commande correspondant à l'automate de la Fig.4.7

Notes :

- la notation 01 signifie si la condition not(A) and B est vraie
- la notation – signifie 0 ou 1
- la notation -0 signifie si la condition not(B) est vraie (quelque soit la valeur de A)
- l'expression not(A) or not(B) est représentée par 0- (pour not(A)) et par -0 (pour not(B))
- la ligne 0- 2 3 3 F3 signifie que si not(A) est vraie, et si l'état courant de l'automate est S_2, il y'a alors transition vers l'état S_3 et exécution de l'instruction 3 qui sera matérialisée par le signal F_3 qui prendra la valeur logique 1
- la ligne -0 2 3 3 F4 signifie que si not(B) est vraie, et si l'état courant de l'automate est S_2, il y'a alors transition vers l'état S_3 et exécution de l'instruction 3 qui sera matérialisée par le signal F_4 qui prendra la valeur logique 1
- ces 2 dernières lignes signifient tout simplement que si *not(A) or not(B)* est vraie, et si l'état courant de l'automate est S_2, alors il y'a transition vers l'état S_3 et exécution de l'instruction 3 qui sera matérialisée par le signal (*F_3 or F_4*) qui prendra la valeur logique 1.
- la ligne 11 2 0 NOOP F 6 veut dire que si la condition *A and B* est vraie, et si l'état courant est S_2, alors il y'aura transition vers l'état S_0 et aucune instruction ne sera exécutée (NOOP)
- les 3 lignes "0- 2 3 3 F 3", "-0 2 3 3 F4" et "11 2 0 NOOP F 6" correspondent tout simplement à la ligne "2 3 not(B)ornot(A) 3 not(B)ornot(A) fin1 0 BandA NOOP BandA fin1 fin2" de l'automate de la Fig.4.7

Enfin, supposons que dans le quatrième cycle, une autre instruction (6) y était ordonnancée comme suit :

```
Cycle4 :
-------------------------------------------------------------
      4     [ not(B)]
      6     [ A and not(B) ]
```

Nous avions vu que la ligne "3 4 not(B) 4 not(B) 6 A and not(B) fin1 0 not(A)andB NOOP not(A)andB fin1 fin2" remplacerait alors la ligne "3 4 not(B) 4 not(B) fin1 0 not(A)andB NOOP not(A)andB fin1 fin2" de l'automate de la Fig.4.7.

Dans ce cas, on aurait : "00 3 4 4 F7" et "10 3 4 4,6 F8" à la place de la ligne "-0 3 4 4 F7" dans la table des signaux de la Fig.4.10 (les signaux F_8, F_9, F_{10} et F_{11} de la table seraient remplacés respectivement par F_9, F_{10}, F_{11} et F_{12}).

La table des signaux étant générée, il suffit de mettre à jour les grilles des transistors de commande en fonction des nœuds qui leur sont associés. On distingue alors 3 cas :

- un signal F_i provenant de la partie de contrôle est directement connecté à la grille d'un transistor qui commande un transfert de données dans un seul cycle. Dans ce cas, cette grille reçoit le numéro affecté au signal F_i
- un registre (ou une unité fonctionnelle) est utilisé(e) dans plusieurs cycles. Dans ce cas, les grilles des transistors directement connectés au registre (ou à l'unité fonctionnelle) sont commandées par l'union des signaux de commande contrôlant le registre (ou l'unité fonctionnelle) dans les différents cycles lorsque le transfert se fait plus d'une fois sur le même bus. Ces grilles reçoivent alors pour numéros ceux des nœuds de sortie des portes *or* réalisant l'union des signaux de commande concernés
- un transistor PMOS reçoit le signal complémentaire à celui contrôlant le transistor NMOS de la même porte de transmission. La grille du transistor PMOS aura alors pour numéro celui du nœud de sortie de l'inverseur ayant pour entrée le signal commandant le transistor NMOS de cette même porte de transmission

Nous donnons ci-après l'algorithme d'affectation de nœuds aux grilles des transistors de commande.

Algorithme 4-4 //mise à jour des grilles des transistors

MAJgate() :

Début
Considérer la liste des registres.
Considérer R, le premier registre de cette liste.
1: Si R utilise un seul bus en entrée alors
    mettre à jour les grilles des transistors de l'entrée de R.
Sinon
    mettre à jour les grilles des interrupteurs de chaque bus utilisé par R en entrée.
Fin de si.
Si R utilise un seul bus en sortie alors
    mettre à jour les grilles des transistors à la sortie de R.
Sinon
    mettre à jour les grilles des interrupteurs de chaque bus utilisé par R à sa sortie.
Fin de si.
s'il y' a un registre suivant dans la liste des registres alors
    R ← ce registre ; aller à 1.
Fin de si.
Considérer la liste des unités fonctionnelles.
Considérer U la première unité fonctionnelle de cette liste
2: Si U est unaire alors
    aller à 3.
Sinon
    Si U utilise un seul bus à sa 2ème entrée alors
        mettre à jour les grilles des transistors constituant l'entrée 2 de U.
    Sinon
        mettre à jour les grilles des interrupteurs de chaque bus utilisé par U à sa 2ème entrée .
    Fin de si.
3: Si U utilise un seul bus à sa 1ère entrée alors
        mettre à jour les grilles des transistors constituant l'entrée 1 de U.
    sinon
        mettre à jour les grilles des interrupteurs de chaque bus utilisé par U à sa 1ère entrée.
    Fin de si.
    Si U utilise un seul bus en sortie alors
        mettre à jour les grilles des transistors de la sortie de U.
    sinon
        mettre à jour les grilles des interrupteurs de chaque bus utilisé par U à sa sortie.
    Fin de si.

s'il y' a une unité fonctionnelle suivante dans la liste des unités fonctionnelles
alors
U ← cette unité ; aller à 2.
Fin de si.
Fin.

Le module qui permet de déterminer les signaux de commande est implémenté par la fonction *retourner_signal()*.

Fonction *retourner_signal():*

Cette fonction consiste à rechercher tous les signaux de commande de l'instruction i à partir du fichier *table_signaux* (Cf Fig.4.11). Il y'a alors concaténation de ces signaux par l'opérateur logique "+" dans la chaîne *signal* qui est retournée par la fonction.

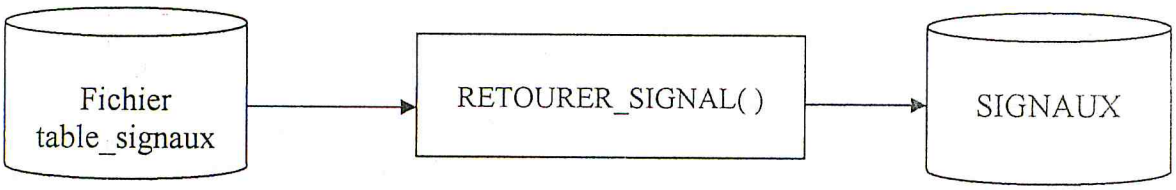


Fig.4.11. Détermination du signal de commande d'une opération donnée

Algorithme 4-5:// Détermination du signal de commande d'une opération donnée
retourner_signal(i)

Ouvrir le fichier *table_signaux* pour lecture ;
signal ← signal de commande de l'instruction i ;
s'il existe d'autres signaux de commande de l'instruction i
alors Concaténer tous les signaux de commande de l'instruction i par des opérateurs logiques "+" et mettre le résultat de la concaténation dans la chaîne *signal*;
fin de si
Fermer le fichier *table_signaux*;
Retourner *signal;*

Nous donnons ci-après un exemple pour illustrer l'algorithme que nous venons de décrire. Le fichier d'allocation correspondant au fichier d'ordonnancement de

la Figure 4.1 et auquel correspond la table de commande indiquée dans la Figure 4.10 est le suivant :

```
cycle 1
      1 [r2] := [r1] * [r3] 1
        br1
        br2
        bf1
cycle 2
      2 [r3] := [r2] - [r1] 1
        br1
        br2
        bf1
cycle 3
      3 [r1] := not [r3] not(A)ornot(B)
        br1
        bf1
cycle 4
      4 [r2] := not [r3] not(B)
        br1
        bf2
cycle 5
      5 [r2] := [r1] * [r3] Aandnot(B)
        br1
        br2
        bf3
```

Figure 4.12. Fichier d'allocation issu du fichier d'ordonnancement de la figure 4.3

Dans cet exemple, nous avons ce qui suit:

	r1	r2	r3	*	-	not
br1	1, 5	2	3, 4	1, 5	2	3, 4
br2	2	X	1, 5	1, 5	2	X
bf1	3	1	2	1	2	3
bf2	X	4	X	X	X	4
bf3	X	5	X	5	X	X

Pour ce qui est de la 1ère ligne et la 1ère colonne, ceci veut dire que le registre r1 est directement connecté au bus br1 pour assurer les transferts des données de r1 sur br1 afin d'exécuter l'instruction 1 au cycle 1, et l'instruction 5 au cycle 5. Le résultat de l'opération 4 est transféré au cycle 4 vers le registre r2 via le bus bf2. Le même raisonnement est fait pour les autres cas.

Par conséquent, les grilles des transistors de commande sont déterminées comme suit :

- la grille du transistor N qui est directement relié au registre r1 et au bus br1 reçoit le numéro du nœud de sortie du circuit *ou* réalisant l'union des signaux F1 et F9 permettant respectivement l'exécution des instructions 1 et 5
- la grille du transistor N qui est directement relié au registre r2 et au bus br1 reçoit le numéro affecté au signal F2 qui permet l'exécution de l'instruction 2
- la grille du transistor N qui est directement relié au registre r3 et au bus br1 reçoit le numéro du nœud de sortie du circuit *ou* réalisant l'union des signaux F3, F4 et F7. Notons que F3 et F4 permettent l'exécution de l'instruction 3 au cycle 3 respectivement quand la condition not(A) ou not(B) est vraie, et F7 celle de l'instruction 4 au cycle 4
- le raisonnement qui vient d'être décrit s'applique aussi pour les unités fonctionnelles el les autres registres
- la grille de tout transistor P est alimentée par le signal complémentaire issu de l'inverseur ayant pour entrée le signal alimentant la grille du transistor N de la même porte de transmission

Les figures 4.13, 4.14, 4.15, 4.16 et 4.17 représentent respectivement les fichiers *Bus*, *registre*, *unité fonctionnelle*, *circuit* et *netlist*.
La figure 4.18 représente le résultat de l'affectation des sources, drains et grilles des transistors commandant les transferts de données de la partie opérative dont les fichiers d'ordonnancement et d'allocation sont respectivement indiqués dans les figures 4.3 et 4.12.

```
BUS EN ENTREE
br2:
     12
br1:
     13
BUS EN SORTIE
bf3:
     14   15
bf2:
     16
bf1:
     17   18
```

Fig.4.13. Fichier *Bus*.

```
R:3
     19      %%   20
R:1
     21      %%   22
R:2
     23   24      %%   25
```

Fig.4.14. Fichier *Registres.*

```
not:
     26      %%   27
-:
     28   29      %%   30
*:
     31   32      %%   33 34
```

Fig. 4.15. Fichier *unités fonctionnelles.*

```
LES CIRCUITS OU:

 4   3      %%   37
 9   1      %%   36
 7   4   3      %%   35

LES INVERSEURS :

 1    %% 45
 7    %% 44
 9    %% 43
 37   %% 42
 36   %% 41
 35   %% 40
 2    %% 38
```

Fig.4.16. Fichier *Circuit.*

S	G	D	TYPE	SIGNAL
19	2	17	N	F2
21	37	17	N	F3+F4
25	2	13	N	F2
26	35	13	N	F3+F4+F7
28	2	13	N	F2
29	2	12	N	F2
30	2	17	N	F2
31	36	13	N	F1+F9
32	36	12	N	F1+F9
19	38	17	P	not(F2)
21	42	17	P	not(F3+F4)
25	38	13	P	not(F2)
26	40	13	P	not(F3+F4+F7)
28	38	13	P	not(F2)
29	38	12	P	not(F2)
30	38	17	P	not(F2)
31	41	13	P	not(F1+F9)
32	41	12	P	not(F1+F9)
20	35	13	N	F3+F4+F7
20	36	12	N	F1+F9
22	2	12	N	F2
22	36	13	N	F1+F9
23	9	14	N	F9
24	9	15	N	F9
23	7	16	N	F7
23	1	17	N	F1
24	1	18	N	F1
27	7	16	N	F7
27	37	17	N	F3+F4
33	9	14	N	F9
34	9	15	N	F9
33	1	17	N	F1
34	1	18	N	F1
20	40	13	P	not(F3+F4+F7)
20	41	12	P	not(F1+F9)
22	38	12	P	not(F2)
22	41	13	P	not(F1+F9)
23	43	14	P	not(F9)

```
24   43   15   P    not(F9)
23   44   16   P    not(F7)
23   45   17   P    not(F1)
24   45   18   P    not(F1)
27   44   16   P    not(F7)
27   42   17   P    not(F3+F4)
33   43   14   P    not(F9)
34   43   15   P    not(F9)
33   45   17   P    not(F1)
34   45   18   P    not(F1)
```

Fig.4.17. Fichier *Netlist.*

4.5. Conclusion:

Le présent chapitre est un complément du chapitre précédent pour la détermination complète des nœuds des transistors de commande d'exécution des opérations d'une partie opérative donnée. Il a consisté à déterminer les signaux alimentant les grilles des différents transistors de commande. Nous avons montré en particulier comment un fichier d'ordonnancement simplifié est généré à partir du fichier d'ordonnancement initial, et comment l'automate de séquencement de l'exécution des différentes opératives en est déduit. S'en est alors suivie la description de la génération de la table de commande permettant l'affectation des signaux aux transistors de commande de la partie opérative concernée.

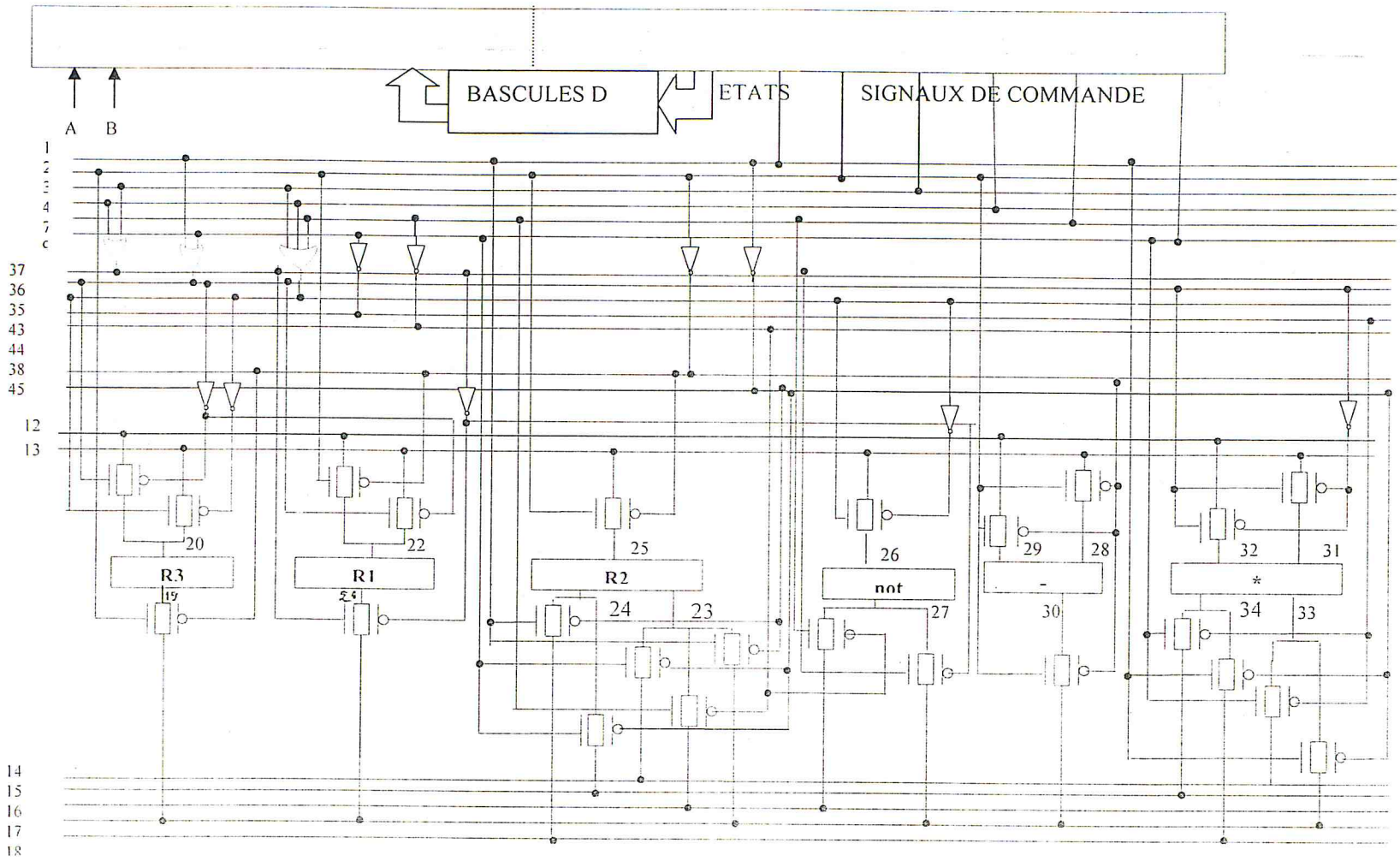


Fig.4.18. Détermination des sources, drains et grilles des transistors de commande d'une partie opérative

CHAPITRE 5

Résultats

5.1. Introduction :

Dans les chapitres précédents, des exemples ont été donnés en fonction du problème traité. Dans ce présent chapitre, nous présentons quelques résultats générés par les algorithmes que nous avions développés.

5.2. Discussion de résultats :

La table 5.1 indique certains de nos résultats. Celle-ci montre que pour un nombre assez important de transistors (8960), le temps CPU pour l'exécution de notre programme est de zéro seconde, ce qui montre l'efficacité de nos algorithmes, bien que les tâches abordées dans le cadre de ce travail soient polynômiales.

Nombre de registres	Nombre d'unités fonctionnelles	Nombre de bits	Nombre de transistors	Temps CPU (s)
3	3	32	1536	0
6	5	32	2176	0
6	6	32	2816	0
15	6	32	2880	0
16	8	64	8960	0

Table5.1. Présentation de quelques résultats

5.3. Exemple d'illustration :

5.3.1. Fichiers générés :

L'exemple donné ci-après a pour but de montrer *tous les fichiers* générés par notre programme. Du fait que les fichiers générés sont de taille importante pour des fichiers de données considérant plusieurs bits, nous nous limitons ici à des registres à 2 bits.

Soit le fichier d'ordonnancement suivant :

```
cycle 1 :
-----------------------------------------------------
 1 : r4=r5 or r6   [ 1 ]
-----------------------------------------------------

cycle 2 :
-----------------------------------------------------
  2 : r3=r4 and r2 [ not(A)and not(B) ]
  2 : r3=r4 and r2 [ not(A)and B ]
```

```
  2 : r3=r4 and r2 [ A and not(B) ]
  3 : r1=r4 and r6 [ not(A)and not(B) ]
  3 : r1=r4 and r6 [ not(A)and B ]
  3 : r1=r4 and r6 [ A and B ]
-------------------------------------------------

cycle 3 :
-------------------------------------------------
   4 : r4=r5 or r6 [ not(A)and not(B)
   4 : r4=r5 or r6 [ A and not(B) ]
   4 : r4=r5 or r6 [ A and B ]
   5 : r2=r3 or r1 [ not(A)and B ]
   5 : r2=r3 or r1 [ A and not(B) ]
   5 : r2=r3 or r1 [ A and B ]
-------------------------------------------------

cycle 4 :
-------------------------------------------------
  6 : r4=r2 nand r6 [ not(A)and not(B) ]
  6 : r4=r2 nand r6 [ A and B ]
  7 : r5=r1 nor r3 [ A and not(B) ]
  7 : r5=r1 nor r3 [ A and B ]

-------------------------------------------------

cycle 5 :
-------------------------------------------------
8 : r6=r5 and r4 [ not(A)and not(B) ]
8 : r6=r5 and r4 [ A and B ]

-------------------------------------------------
```

Au précédent fichier d'ordonnancement, correspond le fichier d'allocation suivant :

```
cycle 1
      1 [r4] := [r5] or [r6] 1
        br4
        br3
        bf2
```

```
cycle 2
      2 [r3] := [r4] and [r2] not(A)or not(B)
        br3
        br4
        bf2

      3 [r1] := [r4] and [r6] B or not(A)
        br2
        br1
        bf1
cycle 3
      4 [r4] := [r5] or [r6] not(B)or A
        br4
        br3
        bf2

      5 [r2] := [r3] or [r1] B or A
        br1
        br2
        bf1
cycle 4
      6 [r4] := [r2] nand [r6] (not(A)and not(B))or
                                              (A and B)
        br4
        br3
        bf2

      7 [r5] := [r1] nor [r3] A
        br2
        br1
        bf1
cycle 5
      8 [r6] := [r5] and [r4] (not(A)and not(B))or
                                              (A and B)
        br4
        br3
        bf1
```

Les fichiers qui sont alors générés par notre programme sont les suivants :

Fichier *automate* :

```
0 1 1 1 1 fin1 fin2
1 2 1 2 not(B)ornot(A) 3 Bornot(A) fin1 fin2
2 3 1 4 not(B)orA 5 BorA fin1 fin2
3 4 not(B)orA 6 (not(A)andnot(B))or(AandB) 7 A fin1 0
Bandnot(A) NOOP Bandnot(A) fin1 fin2
4 5 (not(A)andnot(B))or(AandB) 8 (not(A)andnot(B))
or(AandB) fin1 0  Aandnot(B) NOOP Aandnot(B) fin1
fin2
5 0 (not(A)andnot(B))or(AandB) NOOP
(not(A)andnot(B))or(AandB) fin1 fin2
```

Fichier *table signaux* :

AB	etat_crt	etat_svt	instruction	signal
--	0	1	1	F 1
0-	1	2	3	F 2
-1	1	2	3	F 3
0-	1	2	2	F 4
-0	1	2	2	F 6
1-	2	3	5	F 7
-1	2	3	5	F 8
1-	2	3	4	F 9
-0	2	3	4	F 10
1-	3	4	7	F 11
11	3	4	6	F 12
00	3	4	6	F 13
01	3	0	NOOP	F 14
11	4	5	8	F 15
00	4	5	8	F 16
10	4	0	NOOP	F 17
11	5	0	NOOP	F 18
00	5	0	NOOP	F 19

Fichier *bus*:

```
BUS EN ENTREE
br1:
    20  21
br2:
    22  23
br3:
    24  25
```

```
br4:
     26   27
BUS EN SORTIE
bf1:
     28   29
bf2:
     30   31
```

Fichier *registres*:

```
R:1
     32   33      %%   34 35
R:2
     36   37      %%   38 39
R:3
     40   41      %%   42 43
R:6
     44   45      %%   46 47
R:5
     48   49      %%   50 51
R:4
     52   53      %%   54 55
```

Fichier *unités fonctionnelles*:

```
nor:
     56   57   58   59      %%   60 61
nand:
     62   63   64   65      %%   66 67
or:
     68   69   70   71      %%   72 73
and:
     74   75   76   77      %%   78 79
and:
     80   81   82   83      %%   84 85
or:
     86   87   88   89      %%   90 91
```

Fichier *circuit* :

```
LES CIRCUITS OU:


 10   9   1      %%   103
 16   15   3   2      %%   102
 13   12      %%   101
 16   15   6   4      %%   100
 16   15   10   9   1      %%   99
 13   12   10   9   1      %%   98
 16   15      %%   97
 6   4      %%   96
 13   12   6   4      %%   95
 8   7      %%   94
 11   8   7      %%   93
 3   2      %%   92

LES INVERSEURS :

 103  %% 117
 102  %% 116
 101  %% 115
 100  %% 114
 99   %% 113
 11   %% 112
 98   %% 111
 97   %% 109
 96   %% 108
 95   %% 107
 94   %% 106
 93   %% 105
 92   %% 104
```

Fichier *netlist* :

```
*************************************************************
S    G     D   TYPE SIGNAL
*************************************************************
32   92    28  N    F2+F3
33   92    29  N    F2+F3
34   93    22  N    F7+F8+F11
35   93    23  N    F7+F8+F11
36   94    28  N    F7+F8
```

37	94	29	N	F7+F8
38	95	26	N	F4+F6+F12+F13
39	95	27	N	F4+F6+F12+F13
40	96	30	N	F4+F6
41	96	31	N	F4+F6
42	93	20	N	F7+F8+F11
43	93	21	N	F7+F8+F11
44	97	28	N	F15+F16
45	97	29	N	F15+F16
48	11	28	N	F11
49	11	29	N	F11
50	99	26	N	F1+F9+F10+F15+F16
51	99	27	N	F1+F9+F10+F15+F16
52	98	30	N	F1+F9+F10+F12+F13
53	98	31	N	F1+F9+F10+F12+F13
56	11	22	N	F11
57	11	23	N	F11
58	11	20	N	F11
59	11	21	N	F11
60	11	28	N	F11
61	11	29	N	F11
62	101	26	N	F12+F13
63	101	27	N	F12+F13
64	101	24	N	F12+F13
65	101	25	N	F12+F13
66	101	30	N	F12+F13
67	101	31	N	F12+F13
68	94	20	N	F7+F8
69	94	21	N	F7+F8
70	94	22	N	F7+F8
71	94	23	N	F7+F8
72	94	28	N	F7+F8
73	94	29	N	F7+F8
78	102	28	N	F2+F3+F15+F16
79	102	29	N	F2+F3+F15+F16
80	96	24	N	F4+F6
81	96	25	N	F4+F6
82	96	26	N	F4+F6
83	96	27	N	F4+F6
84	96	30	N	F4+F6
85	96	31	N	F4+F6
86	103	26	N	F1+F9+F10
87	103	27	N	F1+F9+F10
88	103	24	N	F1+F9+F10

89	103	25	N	F1+F9+F10
90	103	30	N	F1+F9+F10
91	103	31	N	F1+F9+F10
32	104	28	P	not(F2+F3)
33	104	29	P	not (F2+F3)
34	105	22	P	not (F7+F8+F11)
35	105	23	P	not (F7+F8+F11)
36	106	28	P	not (F7+F8)
37	106	29	P	not (F7+F8)
38	107	26	P	not (F4+F6+F12+F13)
39	107	27	P	not (F4+F6+F12+F13)
40	108	30	P	not (F4+F6)
41	108	31	P	not (F4+F6)
42	105	20	P	not (F7+F8+F11)
43	105	21	P	not (F7+F8+F11)
44	109	28	P	not (F15+F16)
45	109	29	P	not (F15+F16)
48	112	28	P	not (F11)
49	112	29	P	not (F11)
50	113	26	P	not (F1+F9+F10+F15+F16)
51	113	27	P	not (F1+F9+F10+F15+F16)
52	111	30	P	not (F1+F9+F10+F12+F13)
53	111	31	P	not (F1+F9+F10+F12+F13)
56	112	22	P	not (F11)
57	112	23	P	not (F11)
58	112	20	P	not (F11)
59	112	21	P	not (F11)
60	112	28	P	not (F11)
61	112	29	P	not (F11)
62	115	26	P	not (F12+F13)
63	115	27	P	not (F12+F13)
64	115	24	P	not (F12+F13)
65	115	25	P	not (F12+F13)
66	115	30	P	not (F12+F13)
67	115	31	P	not (F12+F13)
68	106	20	P	not (F7+F8)
69	106	21	P	not (F7+F8)
70	106	22	P	not (F7+F8)
71	106	23	P	not (F7+F8)
72	106	28	P	not (F7+F8)
73	106	29	P	not (F7+F8)
78	116	28	P	not (F2+F3+F15+F16)
79	116	29	P	not (F2+F3+F15+F16)

```
80   108   24  P    not (F4+F6)
81   108   25  P    not (F4+F6)
82   108   26  P    not (F4+F6)
83   108   27  P    not (F4+F6)
84   108   30  P    not (F4+F6)
85   108   31  P    not (F4+F6)
86   117   26  P    not (F1+F9+F10)
87   117   27  P    not (F1+F9+F10)
88   117   24  P    not (F1+F9+F10)
89   117   25  P    not (F1+F9+F10)
90   117   30  P    not (F1+F9+F10)
91   117   31  P    not(F1+F9+F10)


46   92   20   N    F2+F3
47   92   21   N    F2+F3
46   98   24   N    F1+F9+F10+F12+F13
47   98   25   N    F1+F9+F10+F12+F13
54   92   22   N    F2+F3
55   92   23   N    F2+F3
54   100  24   N    F4+F6+F15+F16
55   100  25   N    F4+F6+F15+F16
74   97   26   N    F15+F16
75   97   27   N    F15+F16
74   92   22   N    F2+F3
75   92   23   N    F2+F3
76   97   24   N    F15+F16
77   97   25   N    F15+F16
76   92   20   N    F2+F3
77   92   21   N    F2+F3

46   104  20   P    not (F2+F3)
47   104  21   P    not (F2+F3)
46   111  24   P    not (F1+F9+F10+F12+F13)
47   111  25   P    not (F1+F9+F10+F12+F13)
54   104  22   P    not (F2+F3)
55   104  23   P    not (F2+F3)
54   114  24   P    not (F4+F6+F15+F16)
55   114  25   P    not (F4+F6+F15+F16)
74   109  26   P    not (F15+F16)
75   109  27   P    not (F15+F16)
74   104  22   P    not (F2+F3)
75   104  23   P    not (F2+F3)
76   109  24   P    not (F15+F16)
```

```
77   109  25   P    not (F15+F16)
76   104  20   P    not (F2+F3)
77   104  21   P    not (F2+F3)
```

5.3.2. Commentaires sur les résultats :

1- A partir de la table des signaux, les signaux de commande de chaque instruction sont les suivants :

Instruction 1: F1;
Instruction 2: F4+F6;
Instruction 3: F2+F3;
Instruction 4: F9+F10
Instruction 5: F7+F8;
Instruction 6: F12+F13;
Instruction 7: F11;
Instruction 8: F15+F16;

Pour l'instruction 2 (*[r3] := [r4] and [r2]*) par exemple, nous avons à considérer les noeuds des transistors de commande relatifs aux registres *r2*, *r3* et *r4*, à l'unité fonctionnelle *and*, et aux bus *br3*, *br4* et *bf2* (voir le fichier d'allocation). En considérant les différents fichiers générés, nous avons ce qui suit :

* Fichier *registres* :
- r2: les nœuds des transistors de commande connectés à la sortie de r2 sont 38 et 39 (noter que pour l'instruction 2, ce sont les sorties de r2 qui sont utilisées)
- r4: les nœuds des transistors de commande connectés à la sortie de r4 sont 54 et 55 (noter que pour l'instruction 2, ce sont les sorties de r4 qui sont utilisées)
- r3: les nœuds des transistors de commande connectés à l'entrée de r3 sont 40 et 41 (noter que pour l'instruction 2, ce sont les entrées de r3 qui sont utilisées)

* Fichier *bus* :
- br3 : les 2 fils de ce bus ont pour numéros de nœuds 24 et 25
- br4 : les 2 fils de ce bus ont pour numéros de nœuds 26 et 27
- bf2 : les 2 fils de ce bus ont pour numéros de nœuds 30 et 31

* Fichier *unités fonctionnelles* :
2 unités exécutant l'opération *and* sont utilisées par la partie opérative. Les numéros des nœuds des transistors directement connectés à ces 2 unités fonctionnelles sont 74, 75, 76, 77, 78, 79 pour l'une, et 80, 81, 82, 83, 84 et 85 pour l'autre. Du fait que l'instruction 2 est exécutée lorsque le signal (F4+F6)

est vrai (voir la table des signaux), du fait que le signal du nœud 96 vaut 1 lorsque (F4+F6) est vrai, et comme les transistors ayant pour nœuds 80, 96, 24 ; ...., 85, 96, 31 existent (voir le fichier *netlist*) alors que les transistors connectés à l'unité *and* par les nœuds 74, 75, ... , 79 ont des grilles qui ne sont pas alimentées par un signal prenant la valeur *vrai* lorsque (F4+F6) l'est, on en déduit aisément quelle unité *and* est utilisée pour l'instruction 2.

* Fichier *netlist :*

Il suffit de considérer toutes les lignes où le signal (F4+F6) apparaît pour déterminer les transistors de commande de l'instruction 2. Ces lignes sont celles où la grille vaut 95(signal `F4+F6+F12+F13`), 96(signal `F4+F6`) ou 100(signal `F4+F6+F15+F16`).

* Fichier *circuit* :
A partir de ce fichier, nous avons les informations suivantes :
- 95 est le nœud de sortie du circuit OU ayant pour entrées les signaux F13, F12, F6 et F4
- 96 est le nœud de sortie du circuit OU ayant pour entrées les signaux F4 et F6
- 100 est le nœud de sortie du circuit OU ayant pour entrées les signaux F16, F15, F6 et F4

De ce fait, tous les transistors de type N ayant pour numéros de grille 95, 96 ou 100 vont contribuer pour l'exécution correcte de l'instruction 2. Du fait que chaque porte de transmission est constituée d'un transistor N et d'un transistor P, et que les signaux de grille de ces deux types de transistors doivent être complémentaires, les grilles des transistors P contrôlant l'exécution de l'instruction 2 sont tout simplement celles qui résultent de l'inversion des signaux ayant pour numéros de nœuds 95, 96 ou 100. Ce sont donc les nœuds 107, 108 et 114, respectivement.

2- lorsque les signaux F4 ou F6 sont vrais, les autres signaux de commande sont faux (voir la table des signaux), à l'exception du signal F2 qui permet d'exécuter en parallèle l'instruction 3 lorsque la condition *not(A)* est vraie.

3- le nombre des fils des bus br1, br2, br3, br4, bf1, bf2 est égal à 2 du fait que nous avons considéré des registres à 2 bits.

4- les nœuds générés montrent qu'il y'a une bonne correspondance des poids des différents bits.

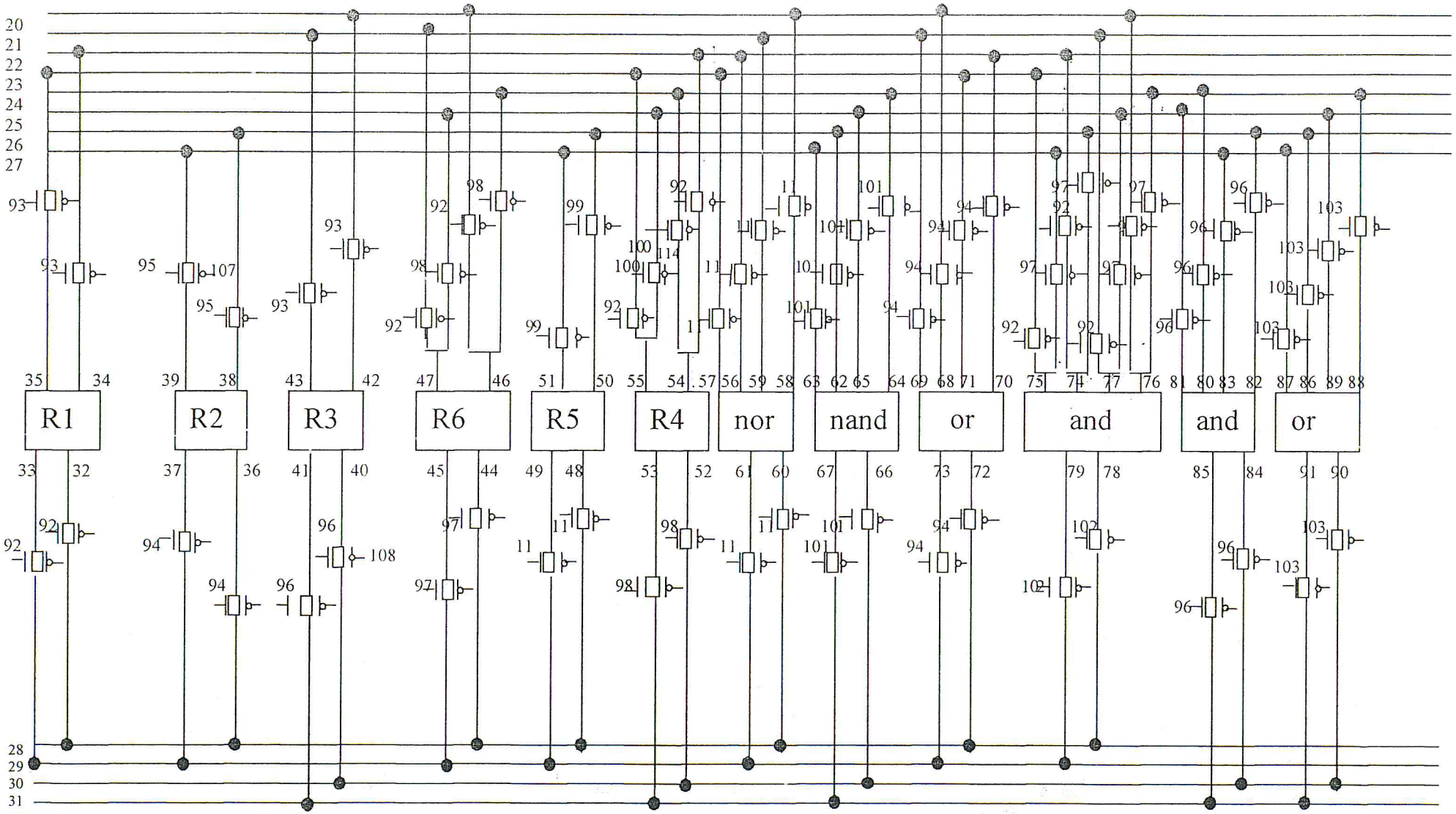


Fig.5.1. Affectation de nœuds aux transistors de commande d'une partie opérative

5.4. Conclusion :

Dans ce chapitre, nous avions énuméré tous les fichiers représentant les résultats obtenus à partir des fichiers d'ordonnancement et d'allocation. Ces résultats montrent que nos algorithmes sont efficaces (le temps CPU est égal à 0 seconde dans tous les cas présentés) même si les tâches abordées dans le cadre de notre travail sont polynômiales.

En outre, nous avions expliqué comment les nœuds sont affectés aux transistors de commande à travers l'exécution d'une instruction (instruction 2).

CHAPITRE 6

Conclusion générale

Nous avons présenté dans ce mémoire un travail portant sur l'élaboration d'une commande de transfert de données dans un circuit VLSI.

Après une introduction, nous avions donné quelques définitions et rappels pour une meilleure compréhension de ce mémoire. Suivirent alors les chapitres 3 et 4 qui constituent l'essentiel de notre travail.

Nous avions alors montré comment, à partir des fichiers d'ordonnancement et d'allocation, se fait l'affectation des nœuds aux transistors de commande directement connectés aux registres et unités fonctionnelles de la partie opérative. L'affectation des nœuds aux différents bus a été aussi expliquée.

Par la suite, l'affectation des nœuds aux grilles des transistors de commande a été abordée. Celle-ci s'est faite en plusieurs étapes, dont :

- la simplification d'équations logiques correspondant aux conditions d'exécution des différentes instructions,
- la génération de l'automate d'exécution des différentes opérations,
- la génération de la table des signaux
- la génération des nœuds affectés aux grilles des transistors de commande

Nous avions présenté quelques résultats montrant l'efficacité de nos algorithmes, et donné un exemple d'illustration complémentant les nombreux exemples donnés dans les chapitres précédents au fur et à mesure que la description d'une tâche y était abordée.

Ce travail nous a permis de maîtriser certains aspects du domaine de la CAO des circuits VLSI, tout en mettant en pratique certaines de nos connaissances acquises au cours de notre cursus universitaire.

Bibliographie

[1] C. Mead & L. Conway "Introduction aux systèmes VLSI" Addison-Wesley Publishing Company

[2] Weste & Eshraghian "Principles of CMOS VLSI Design: A System Perspective" Reading, Addison-Wesley Publishing Company

[3] D. Gajski, N. Dutt, A. Wu, S. Lin "High Level Synthesis: Introduction to Chip and System Design" Kluwer Academic Publishers

[4] A.V. Aho, J.E. Hopcroft, J.D. Ullman "Data Structures and Algorithms" Addison-Wesley Publishing Company

[5] Ritchie,kenighan "Langage C",ED Masson.

[6] "Linux & vi" Quick Reference Guide

[7] A. Mahdoum "SAFT: An Efficient State Assignment for Finite State Machine Tool" INFORMATION Journal, Nippon Press, Vol.3, N°3, July 2000

[8] A. Mahdoum "Démonstrations sur l'Outil SAFT" Abstract in the Designer's Forum Proceedings of DATE'02, 4-8 March 2002, Palais des Congrès, Paris, France

[9] R. Ruddel, "ESPRESSO" U.C.Berkeley, California, USA

[10] H. Djamah, H. Hadj Moussa, A. Mahdoum "CMORGEN: A CMOS Regular Structure Generator" SYNTHESE, Publication de l'Université Badji Mokhtar, Annaba, Vol.6, N°6, Novembre 1999

[11] M. Bougherara, L. Beha "Contribution à l'ordonnancement des instructions d'un flot de données contrôlées" Mémoire d'Ingénieur d'Etat en Informatique, Université Saad Dahleb, Blida, Promotion 2002/2003

[12] A. Mahdoum "SPOT : An Estimation of the Switching Power Dissipation in CMOS Circuits and Data Paths Tool" SASIM'97 1-2 Dec.1997, Osaka, Japan

[13] A. Mahdoum "SPOT: Un Outil à Base d'un Algorithme Génétique pour Estimer la Consommation Maximale de la Puissance Dynamique des Circuits CMOS" CSCA'99, 8-9 Nov.1999, Hôtel Sheraton, Alger

[14] A. Mahdoum "SPOT : A Tool for Estimating the Average and the Maximal Switching Power Dissipation" Démonstrations sur l'Outil SPOT, Abstract in the Designer's Forum Proceedings of DATE'02, 4-8 March 2002, Palais des Congrès, Paris, France