REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

Université Saad DAHLAB de BLIDA
Faculté des Sciences
Département d'Informatique

Mémoire en vue d'obtenir le diplôme de master en informatique

Option : Génie des systèmes informatiques

Sujet :

# Optimisation assujettie à des contraintes de surface, de vitesse, et de consommation de puissance d'une partie de contrôle d'un circuit intégré.

Présenté par :
BENCHERCHALI Sara.
OURARI Rachaa.

**Soutenu le :** 25/06/2016.

| | | |
|---|---|---|
| M. | M.Ould Khaoua | Président |
| Mme. | S.Aroussi | Examinatrice |
| M. | A.Mahdoum | Promoteur |

**Promotion**
2015 / 2016

# Remerciements

En préambule, nous souhaitons adresser ici tous nos remerciements aux personnes qui nous ont apportés leurs aide et qui ont ainsi contribués a l'élaboration de ce mémoire.

Tout d'abord nous tenons à exprimer notre gratitude et nos remerciements à Mr Mahdoum Ali, notre promoteur, pour son aide, ses conseils, son encouragement, et sa disponibilité dans ce projet.

Nous remercions également les membres de jury qui ont acceptés d'évaluer ce travail.

Nous présentons nos sincères remerciements à tous nos enseignants durant tous notre cursus scolaire et universitaire.

Enfin nous adressons nos plus sincères remerciements à tous nos proches et amis qui nous avons toujours soutenu et encouragé au cours de la réalisation de ce mémoire.

**Rachaa & Sara.**

# *Dédicaces*

*Avec un énorme plaisir, un cœur ouvert et une immense joie, que je dédiée ce travail :*

*À mon très cher et magnifique père Mohammed, mon premier encadrant, depuis ma naissance, qui m'a encouragé tout le temps à aller de l'avant.*

*À ma très chère mère Fatima Zahraa, qui m'a donnée tout son amour et m'a soutenue tout au long de ma vie.*

*À mes frères et ma petite sœur : Walid, Mouadh et Rihab, que je leur souhaite une vie pleine de joie du bonheur et de réussite.*

*À mon très cher binôme et soeur Sara, qui a accepté ma collaboration pour finir ce travail, et à mon amie et sœur Soumia, qu'on a passé ensemble des merveilleux moments durant toutes nos années universitaires.*

*À toutes personnes qui m'ont encouragées ou aidées au long de mes études.*

*Une spéciale dédicace à mon promoteur : Mr Mahdoum Ali, qu'il trouve dans ce modeste travail mes sincères gratitudes et reconnaissances.*

RACHAA

# Dédicaces

*Je dédie ce mémoire a :*

*Mes chères parent Tayeb et Farida que nulle dédicace ne puisse exprimer mes sincères sentiments, pour leur patience, leurs encouragements, leur aide, en témoignage de mon profond amour, et respect pour leur grand sacrifice.*

*Mon cher frère Mohamed, ma chère sœur Selma, ma très chère belle-sœur Faiza, et mon petit cher neveu Walid Anis.*

*Ma chère binôme Rachaa que sans son aide ce travail n'auras jamais vu le jour. Ma meilleure amie Soumia pour son encouragement et son aide tout au long de notre amitié. Et tous mes amis(es) et camarades de la promo 2015/2016.*

*Spécialement pour mon promoteur Mr Mahdoum Ali, trouver dans ce travail mes sincères gratitude et reconnaissance . Et à toutes les personnes qui mon encouragé ou aidé au long de mes études*

*Et à ma famille et toutes les personnes que j'aime.*

*Sara.*

# RESUME

Avec l'avènement du développement technologique, il est actuellement possible d'intégrer des millions de transistors sur la même puce. Outre cet avantage, les délais des portes logiques deviennent de plus en plus courts à cause de la commutation rapide des transistors due à des tensions de seuil des transistors de plus en plus réduites. A côté de ces avantages, il existe malheureusement de nombreux problèmes auxquels les concepteurs doivent y faire face. Nous nous contentons uniquement de citer le problème en relation avec le contexte de ce projet. Les courbes de l'ITRS (The International Technology Roadmap for Semiconductors) concernant les délais des portes logiques et des interconnexions montrent clairement que le premier type de délais ne cesse de s'améliorer au fur et à mesure que la longueur du canal du transistor se réduit. Malheureusement, ceci n'est pas le cas pour les interconnexions: la communication devient de plus en plus lente avec le développement technologique. Il a donc fallu opter pour une conception consistant à concevoir un réseau spécifique à l'application, répondant aux contraintes de largeur de bande, de consommation de puissance et de surface. Plusieurs méthodologies ont été proposées, dont les principales sont celles reposant sur les *Fat Trees*, les *Mesh Networks* et d'autres variantes de ces méthodologies. Dans ce contexte, il a été proposé, au CDTA, une nouvelle méthodologie pour laquelle un protocole de transfert de données entre les différents composants du système a été réalisé et généré sous forme de machines à états finis.

Dans le cadre de ce travail, il s'agit d'implémenter une machine à états finis donnée par une partie de contrôle régissant des transferts de données selon le protocole décrit dans cette machine. Cette implémentation doit être réalisée en tenant compte de contraintes portant sur la surface, la vitesse et la consommation de puissance. Ainsi pour dire, on verra qu'il s'agit d'un problème d'optimisation multi-critères pour lequel la solution optimale ne peut être obtenue en un temps polynomial. D'où la nécessité de développer, pour ce problème, un algorithme à base d'une heuristique ou d'une méta-heuristique générant une solution pré-optimale (voire optimale dans certains cas) en un temps CPU raisonnable.

**Mots-clés :** Partie de contrôle d'un circuit, dimensionnement automatique de transistors et d'interconnexions, optimisation multi-critères, contraintes de temps, surface, consommation de puissance.

# ABSTRACT

As technology scales, it is now possible to integrate millions of transistors on the same chip. Besides this advantage, the delay of the logic gates is becoming shorter due to the rapid switching of the transistors because of their smaller threshold voltages. Besides these advantages, there are unfortunately many issues that designers must cope with. We just mention only the problem in relation to the context of this project. The curves of the ITRS (The International Technology Roadmap for Semiconductors) concerning delays of logic gates and interconnections clearly show that the first type of time continues to improve gradually as the transistor channel length is reduced. Unfortunately, this is not the case for the connections: the communication becomes increasingly slower as technology scales. It was therefore necessary to opt for designing an application-specific network while meeting the bandwidth, power and area constraints. Several methodologies have been proposed: the main ones are those based on the Fat Trees, the Mesh Networks and other variations of these methodologies. In this context, it was proposed, at the CDTA, a new methodology for which a data transfer protocol between the different components of the system was performed and generated as finite state machines.

As part of this work, the aim is to implement a finite state machine by a control circuit governing data transfers, according to the behavior described by this machine. This implementation should be performed taking into account the surface, speed and power constraints. So to say, one will see that it is a problem of multi-criteria optimization for which the optimal solution can not be obtained in polynomial time. Hence the need to develop, for this problem, either an heuristic-based algorithm or a meta-heuristic-based one so as to generate a near optimal solution (or even optimal in some cases) in a reasonable CPU time.

**Key-words:** control unit of an integrated circuit, automating sizing of transistors and interconnections, multi-objective optimization, time, area and power dissipation constraints.

# SOMMAIRE


Introduction générale 1

I Etude bibliographique 3

1. Eléments de conception de circuits intégrés. . . . . .4
2. Heuristiques et méta-heuristiques. . . . . 9
   - 2.1. Heuristiques. . . . . .9
   - 2.2. Méta-heuristiques. . . . . .9
     - 2.2.1. Le recuit simulé. . . . . .9
     - 2.2.2. La méthode taboue (Tabu search). . . . . 11
     - 2.2.3. Les algorithmes génétiques et évolutionnaires. . . . . .13
     - 2.2.4. Les colonies de fourmis. . . . . .14

II Conception et implémentation de la solution 16

1. Etude du problème et solution proposée. . . . . .17
   - 1.1. Introduction. . . . . 17
   - 1.2. Contexte du problème. . . . . .17
   - 1.3. Démarche adoptée . . . . . .21
     - 1.3.1. Estimation de la surface . . . . . .22
     - 1.3.2. Estimation du temps de réponse . . . . . 24
     - 1.3.3. Estimation de la consommation de puissance. . . . . 27
     - 1.3.4. Heuristique donnant la priorité au paramètre *temps* . . . . . 29
     - 1.3.5. Heuristique donnant la priorité au paramètre *consommation de puissance* . . . . . .39
   - 1.4. Conclusion . . . . . .47
2. Mise en œuvre de la solution. . . . . . 47
   - 2.1. Introduction. . . . . 47
   - 2.2. Conception de la solution . . . . . .48
   - 2.3. Plateforme technique utilisée . . . . . 49
     - 2.3.1. Matériels . . . . . .49
     - 2.3.2. Systèmes d'exploitation . . . . . 49

2.4. Architecture technique de la solution .....50
2.5. Mise en œuvre ..... 51
2.6. Conclusion..... 58
III Résultats 59
1. Résultats.....60
1.1. Introduction ..... 60
1.2. Présentation des résultats..... 63
1.3. Conclusion..... .....65
Conclusion générale 66
Bibliographies 68

# Table des figures

Fig. 1.1. Structure générale d'un PLA. . . . . 4

Fig. 1.2. Inverseur en technologie CMOS. . . . . .5

Fig. 1.3. Exemple de porte NOR en CMOS statique. . . . . .7

Fig. 1.4. Exemple de porte NAND en CMOS statique. . . . . 7

Fig. 1.5. Exemple de porte NAND en CMOS dynamique. . . . . .8

Fig. 1.6. Représentation physique d'un inverseur. . . . . 8

Fig. 1.7. L'algorithme du recuit simulé. . . . . 10

Fig. 1.8. L'organigramme de l'algorithme "Recuit Simulé". . . . . 11

Fig. 1.9. L'algorithme Recherche Taboue. . . . . 12

Fig. 1.10. L'organigramme de l'algorithme "Méthode tabou". . . . . 12

Fig. 1.11. L'algorithme génétique. . . . . 13

Fig. 1.12. L'organigramme de l'algorithme génétique. . . . . 14

Fig. 1.13. Algorithme de fourmis. . . . . .15

Fig. 2.1. PLA associé au tableau 2.1 . . . . . 19

Fig.2.2. Effet de la largeur de l'interconnexion sur le temps de transfert de données. . . . . 31

Fig. 2.3. schéma synoptique. . . . . 50

Fig. 2.4. Model d'un fichier d'entrée. . . . . 55

Fig. 2.5. Format de fichier résultats. . . . . 57

# Liste des tableaux

Tableau 2.1. Table de vérité. . . . . 18

Tableau 3.1 Résultats obtenus en donnant la priorité au temps . . . . . 63

Tableau 3.2 Résultats obtenus en donnant la priorité à la consommation de puissance. . .64

# INTRODUCTION GENERALE

Les puces électroniques ce trouvant au cœur de nos objets de haute technologie sont constituées de composants semi-conducteurs « les transistors » et de composants conducteurs « les fils métalliques » qui servent à relier les premier entre eux. La structure de ces transistors, leur agencement et leur interconnexion permettent de réaliser des fonctions électroniques complexes telles que des processeurs, des mémoires, ou encore des amplificateurs utilisés pour transmettre de l'information par radiofréquence ou par optique.

Plus le nombre de transistors intégrés sur la même puce est élevé et le nombre d'interconnexion est élevé, plus il prend d'espace, de temps, et consomme beaucoup d'énergie. Ceci fait que la conception d'un réseau sur puce doit tenir compte de tous ces facteurs dans le but de satisfaire ses spécifications, mais aussi d'assurer un fonctionnement fiable (par exemple, une grande consommation de puissance entrainerait une élévation de température, ce qui engendrait une panne de système).

Dans le cadre de conception assistée par ordinateur d'une partie de contrôle d'un circuit assujettie à des contraintes de surface, de vitesse, et de consommation de puissance, le but de ce travail est de développer un algorithme à base d'une heuristique ou une méta-heuristique. En effet, il sera vu dans les prochains chapitres que ce problème d'optimisation multicritères n'est pas de complexité polynomiale.

# Partie I

# Etude bibliographique

Dans cette étude bibliographique, nous nous intéressons aux deux aspects "Conception de circuits intégrés" et "Optimisation combinatoire" ([1, 2]). Pour chacun de ces deux aspects, nous n'aborderons, évidemment, que les éléments en rapport avec notre travail.

## 1. Eléments de conception de circuits intégrés

Un circuit intégré peut être implémenté à l'aide d'une partie opérative et d'une partie de contrôle. La partie opérative, comme son nom l'indique, consiste à exécuter des opérations arithmétiques, logiques et relationnelles. L'exécution de ces opérations ne peut se faire de manière anarchique, mais plutôt sous le contrôle de la partie de contrôle. Celle-ci a pour rôle de contrôler et commander l'exécution des dites opérations selon le comportement et l'ordonnancement établis. Bien souvent, l'implémentation d'une partie de contrôle (en utilisant auparavant des méthodes de synthèse comme [4]) se fait à l'aide d'un PLA (Programmable Logic Array) à cause de la structure régulière qu'il offre. Notons que plus une structure est régulière et mieux seront les caractéristiques du circuit, en particulier en matière de diminution des paramètres parasites (résistances, capacités et inductances), ce qui est avantageux pour le temps et la consommation de la puissance. La structure générale d'un PLA est indiquée à la Figure 1.1. Le PLA se compose de deux principales parties ou plans AND et OR. Dans le plan AND seront disposées les portes logiques réalisant les termes-produit de chaque équation logique. Dans le plan OR seront implémentées les portes logiques réalisant les sorties désirées.

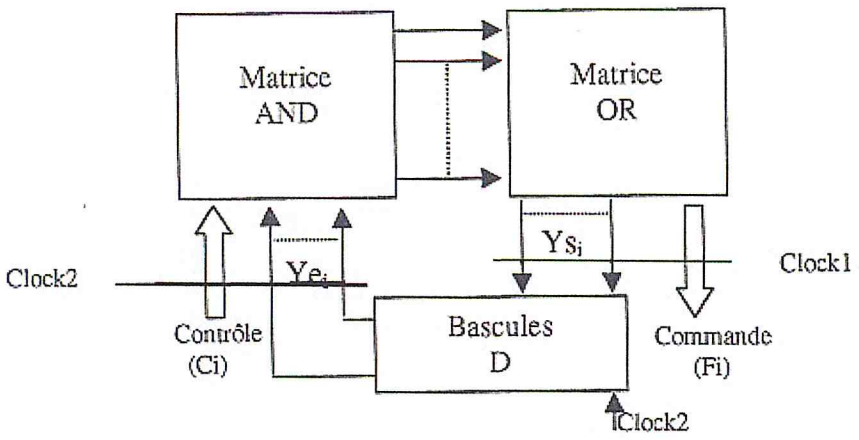


Fig.1.1. Structure générale d'un PLA.

Exemple:

$S_1$= (A and B and $\overline{C}$) or ($\overline{B}$ and C); $S_2$= ($\overline{B}$ and C) or (A and C)

Ainsi, les termes-produit (A and B and $\overline{C}$), ($\overline{B}$ and C) et (A and C) seront implémentés par des portes logiques dans le plan AND. Dans le plan OR, des portes logiques OR seront utilisées pour réaliser $S_1$ et $S_2$.

Chacune des portes logiques sera constituée par un ensemble de transistors. Dans la technologie CMOS (Complementary Metal Oxide Semi-Conductor), sont utilisés deux types de transistors : les transistors NMOS (canal N où la conduction est assurée par des électrons) et les transistors PMOS (canal P où la conduction est assurée par des trous qui sont des charges positives). Cette technologie a remplacé les technologies NMOS et pseudo-NMOS à cause du problème de consommation de puissance que ces dernières engendrent. La Figure 1.2 illustre un inverseur en technologie CMOS.

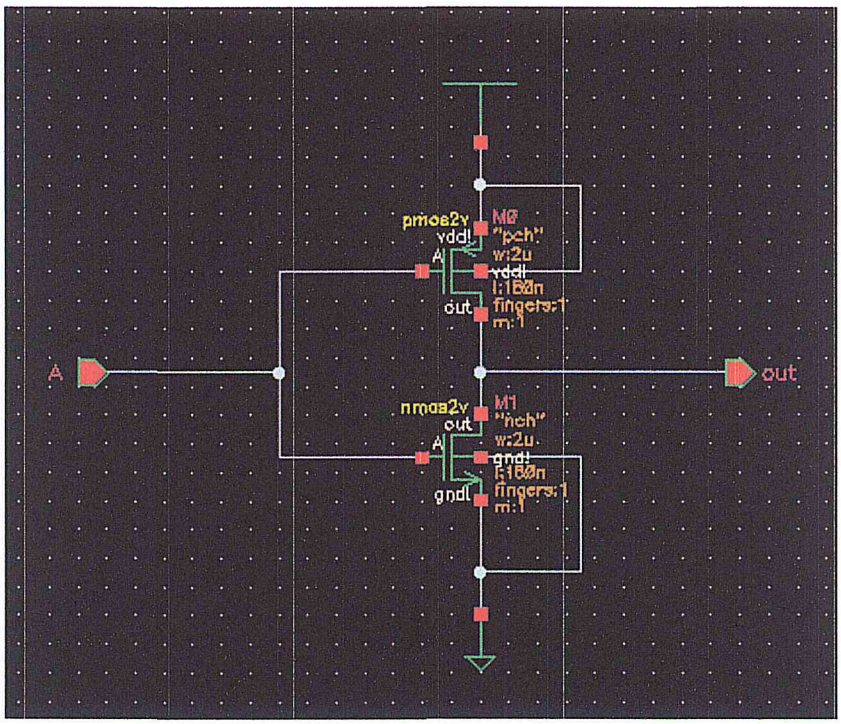


Fig.1.2. Inverseur en technologie CMOS

Dans la Figure 1.2, on distingue, pour chacun des deux transistors NMOS et PMOS, trois nœuds électriques : la grille, la source et le drain.
Chacun des deux transistors est passant ou bloqué selon la tension appliquée sur sa grille et sa tension de seuil. Notons que la tension de seuil d'un transistor est la tension à partir de laquelle le transistor commence à conduire. Pour le transistor NMOS, il commence à conduire dès que $V_{GS} > V_{th}$ ($V_{Gs}$ est la tension appliquée sur la grille du transistor, $V_{th}$ étant la tension de seuil de ce dernier). Pour le transistor PMOS (ayant une tension de seuil négative), la conduction est assurée si $V_{GS} < V_{th}$.

Ainsi, pour l'inverseur de la Figure 1.2, si E vaut '1', le transistor NMOS serait passant alors que le transistor PMOS serait bloqué. Ceci entraînerait une décharge de la capacité au nœud de sortie de l'inverseur, ce qui se traduirait par le '0' logique. Par contre, si E vaut '0', le transistor NMOS serait bloqué alors que le transistor PMOS serait passant. Ceci permettrait de charger la capacité au nœud de sortie de la porte, ce qui se traduirait par l'obtention du '1' logique. Les Figures 1.3 et 1.4 représentent les portes logiques NOR et NAND en technologie CMOS statique, respectivement. Notons qu'il existe aussi le style de conception dynamique. Un exemple implémentant la porte NAND en est donné à la Figure 1.5. Le principe de fonctionnement est le suivant:

- Lorsque $\varphi$=0, le nœud de sortie de la porte est préchargé à '1'.
- Lorsque $\varphi$=1, le transistor d'évaluation est passant, permettant de garder la valeur logique '1' préalablement générée (cas où A=0 et/ou B=0) ou de décharger la capacité (obtention du '0' logique si A=1 et B=1).

Notons que chacun des deux styles de conception statique et dynamique présente des avantages et des inconvénients. L'utilisation de l'un ou de l'autre style se fait selon le contexte [1].
Nous verrons dans la suite que l'estimation de certains paramètres est faite à partir de la représentation physique d'un élément (transistor, circuit, etc.). Aussi, nous indiquons dans la Figure 1.6 la représentation physique d'un inverseur.

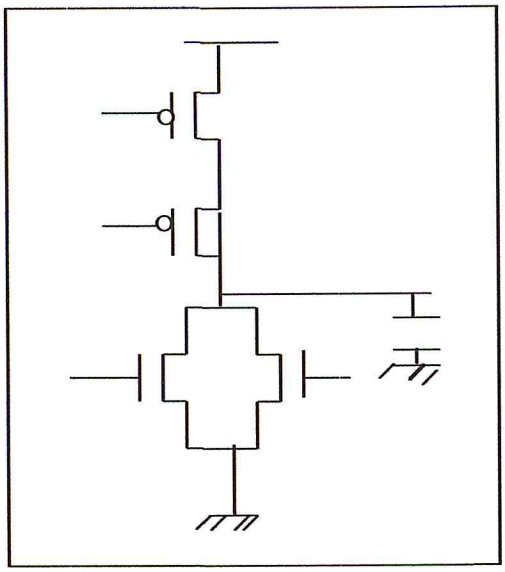

Fig.1.3. Exemple de porte NOR en CMOS statique.

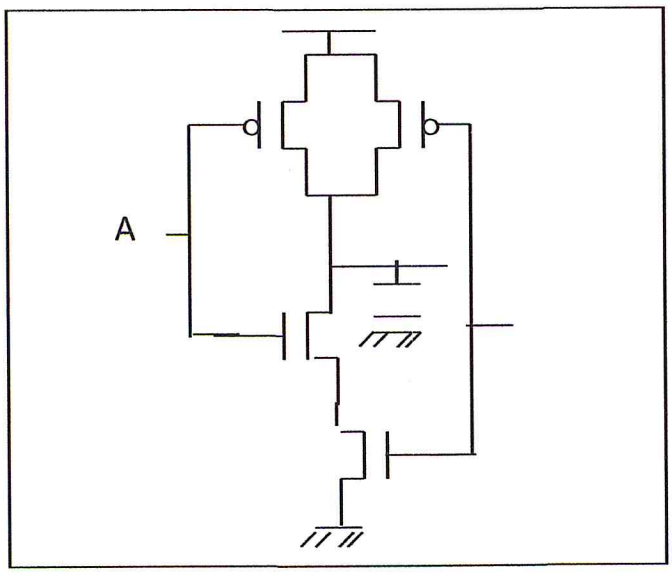


Fig.1.4. Exemple de porte NAND en CMOS statique.

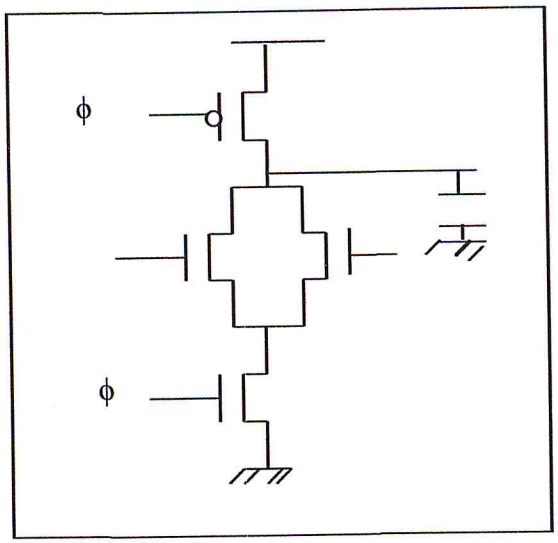


Fig.1.5. Exemple de porte NAND en CMOS dynamique.

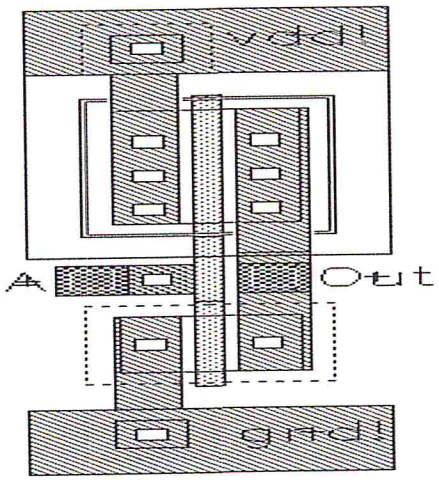


Fig.1.6. Représentation physique d'un inverseur.

Dans le cas de la conception d'un circuit intégré, plusieurs étapes de conception sont requises pour garantir la fonctionnalité du circuit, mais aussi pour que cette conception réponde aux spécifications requises par le client en matière de surface, vitesse et consommation de puissance. Bien souvent, la satisfaction de ces contraintes n'est pas aisée (pour ne pas dire impossible) à obtenir avec une conception manuelle. D'où l'utilisation incontournable de techniques relevant de l'informatique et de la recherche opérationnelle. Dans le cadre de ce travail, il s'agit précisément de développer un outil d'aide à la conception d'une partie de contrôle assujettie à des contraintes de surface, temps et consommation de puissance. Nous verrons ultérieurement que ce problème n'est pas de complexité algorithmique

polynomiale, d'où la nécessité de développer un algorithme à base d'heuristique ou de méta-heuristique.

## 2. Heuristiques et méta-heuristiques

### 2.1. Heuristiques

Une heuristique est une méthode à base d'un algorithme développé non pas à la base d'un canevas établi (cas des méta-heuristiques), mais plutôt de manière spécifique. Quoiqu'une heuristique est une méthode approchée, elle doit générer une solution de qualité (pré-optimale, voire optimale dans certains cas) en un temps CPU raisonnable. Selon l'application concernée (profil, difficulté), ce temps raisonnable peut être de quelques secondes à quelques .... jours (pour justement viser l'obtention d'une solution de qualité tout en notant que cette heuristique doit être de complexité algorithmique polynomiale. Autrement, autant opter pour une méthode exhaustive).

### 2.2. Méta-heuristiques

Tout comme les heuristiques, le développement d'un algorithme à base d'une méta-heuristique doit répondre à la double difficulté de devoir être de complexité algorithmique polynomiale, tout en assurant de générer une solution de qualité. Toutefois, une méta-heuristique se présente comme un canevas ou un moule pour l'algorithme à développer. On distingue plusieurs types de méta-heuristiques. Nous en citerons les principales ci-après.

#### 2.2.1. Le recuit simulé

Le recuit simulé est la première méta-heuristique qui a été proposée par des physiciens. Elle consiste à améliorer la qualité d'un solide, en partant d'une haute température qui rend la matière liquide, avec une phase de refroidissement lente pour avoir la forme solide de notre matière avec l'objectif d'atteindre un état d'énergie minimal. L'idée est d'effectuer un mouvement selon une distribution de probabilité qui dépend de la qualité des différents voisins :

_ Les meilleurs voisins ont une probabilité plus élevée.

_ Les moins bons ont une probabilité plus faible.

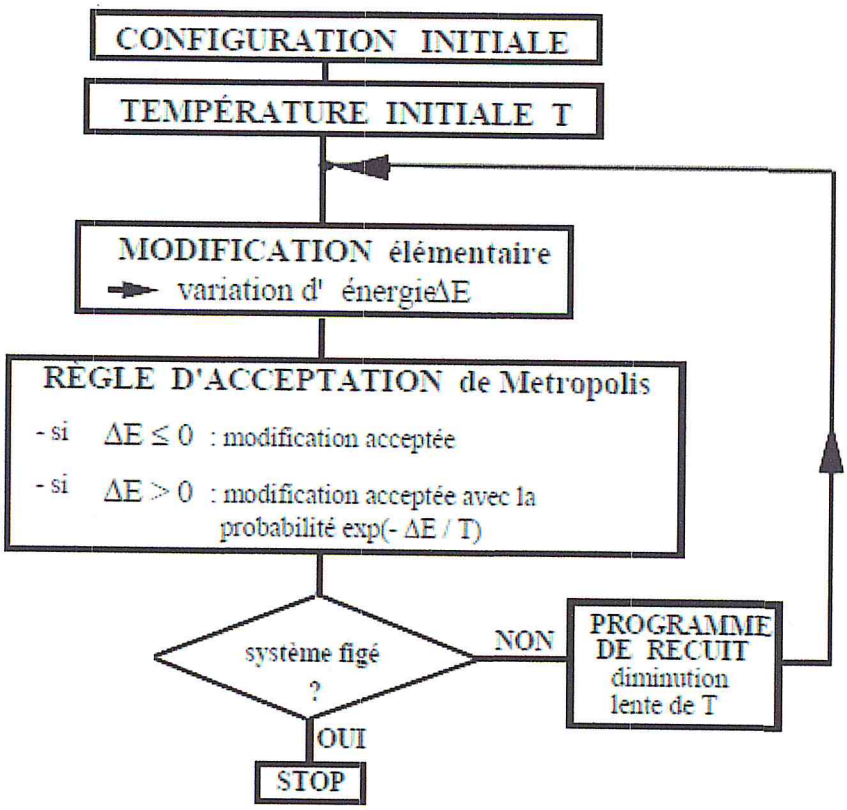


Fig. 1.8 : L'organigramme de l'algorithme "Recuit Simulé".

### 2.2.2. La méthode taboue (Tabu search)

La méthode de recherche avec tabou (ou tabu search) vient de corriger les lacunes du recuit simulé. Elle a le même principe mais utilise le mécanisme de mémoire. La notion de liste taboue permet de mémoriser les régions déjà visitées et d'interdire de retourner rapidement vers ses régions. A chaque itération l'algorithme choisit le meilleur voisin non tabou, même si celui-ci dégrade la fonction.

**Algorithm 2** Méthode Tabou

$S* \leftarrow S$ : $F* \leftarrow F(S*)$ :
$T \leftarrow$ :
**repeat**
  $t \leftarrow$ *le meilleur mouvement parmi les mouvement non tabous* :
  **if** $F(t) < F(S*)$ **then**
    $S* \leftarrow t$ : $F* \leftarrow F(t)$ :
  **end if**
  Mettre $T$ a jour :
**until** condition fin
**return** $S*$ :

Fig. 1.9 : L'algorithme Recherche Taboue.

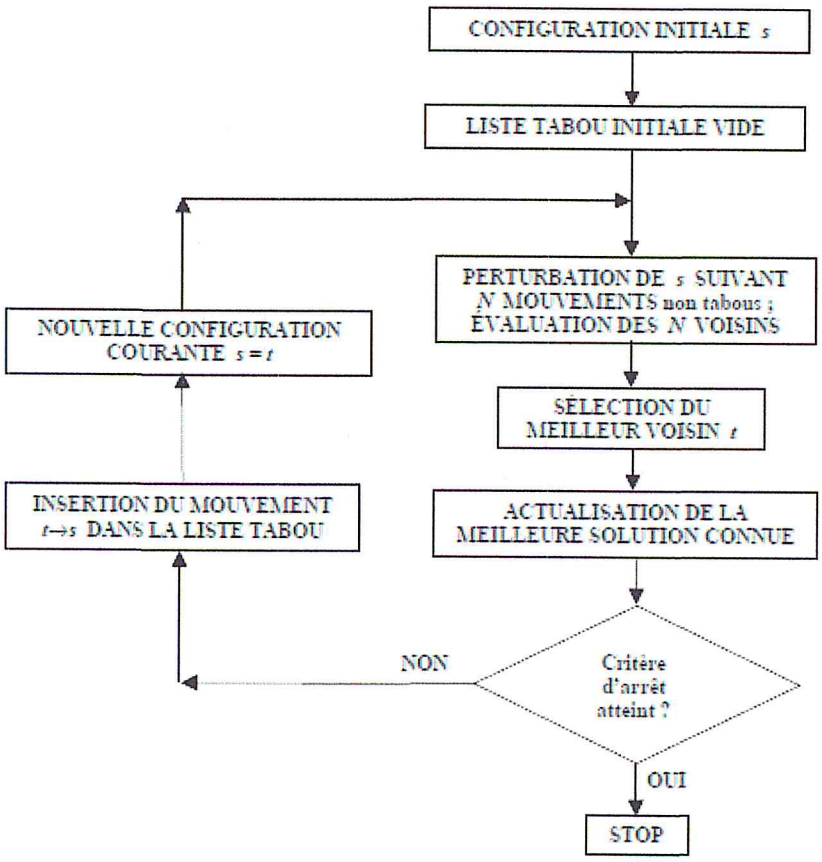


Fig. 1.10 : L'organigramme de l'algorithme "Méthode tabou".

### 2.2.3. Les algorithmes génétiques et évolutionnaires

Comme le recuit simulé a été inspiré de la thermodynamique, les algorithmes évolutionnaires (dont les plus connus sont les algorithmes génétiques) ont été inspirés de l'évolution Darwinienne des populations biologiques. Le principe de cet algorithme est : premièrement faire évoluer un ensemble de solutions candidates, appelé une « population d'individus ». Un « individu » n'est autre qu'une solution possible du problème à résoudre. Chaque individu de cette population se voit attribuer une fonction appelée fonction d'adaptation (fitness) qui permet de mesurer sa qualité ou son poids ; cette fonction d'adaptation peut représenter la fonction objectif à optimiser. Ensuite, les meilleurs individus de cette population sont sélectionnés, subissent des croisements et des mutations et une nouvelle population de solutions est produite pour la génération suivante. Ce processus se poursuit, génération après génération, jusqu'à ce que le critère d'arrêt soit atteint.

**Algorithm 3** Génétique

```
Initialiser la population
while critère d'arrêt non atteint do
  Évaluer les individus ;
  Sélectionner les individus ;
  Faire des opérations des croisement et de mutation ;
  Creer une nouvelle population ;
end while
```

Fig. 1.11 : L'algorithme génétique.

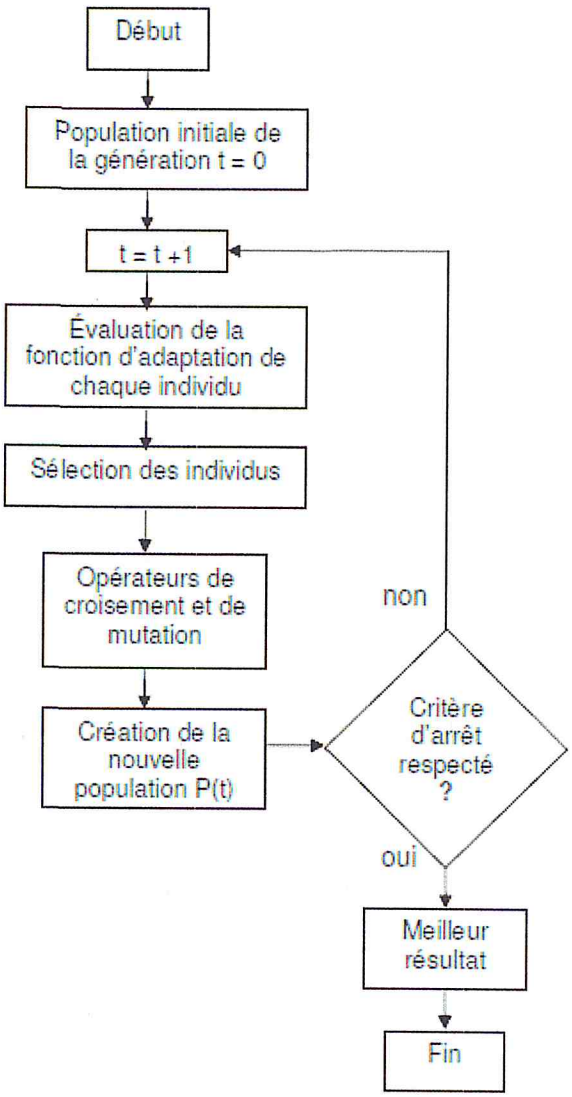


Fig. 1.12 : L'organigramme de l'algorithme génétique.

### 2.2.4. Les colonies de fourmis

Cette méthode est inspirée du comportement des fourmis à la recherche de nourriture. Il est connu que les fourmis sont capables de déterminer le chemin le plus court entre leur nid et une source de nourriture. Ceci est possible grâce à la phéromone qui est une substance que les fourmis déposent sur le sol lorsqu'elles se déplacent. Lorsqu'une fourmi doit choisir entre deux directions, elle choisit avec une plus grande probabilité celle comportant une plus forte concentration de phéromone. Dans le coté informatique : Chaque fourmi est un algorithme constructif, elle doit

prendre une décision pour compléter une solution précise, cette décision dépendra a deux facteurs :

_ La force gloutonne est une valeur qui représente l'intérêt de la fourmi à prendre la décision

_ La trace représente l'intérêt historique de la fourmi pour prendre la décision d. Plus cette quantité est grande, plus il a été intéressant dans le passé de prendre cette décision.

Lorsqu'une fourmi termine la construction de sa solution, elle laisse une trace $\tau_d$ sur le chemin emprunté. Cette trace est proportionnelle à la qualité de la solution construite. De plus, il est important de mettre en place un processus d'évaporation de la trace afin d'oublier les choix réalisés dans un lointain passé et de donner plus d'importance aux choix réalisés récemment.

**Algorithm 4** Colonies de fourmis

Initialiser les traces $\tau_d$ à 0 pour toute décision possible d
**while** critère d'arrêt non atteint **do**
  Construire $|A|$ solutions en tenant compte de la force gloutonne et de la trace ;
  Mettre à jour les traces $\tau_d$ ainsi que la meilleure solution rencontrée ;
  exécuter le processus d'évaporation ;
**end while**

Fig. 1.13 : Algorithme de fourmis.

# Partie II

# Conception et implémentation de la solution

# 1. Etude du problème et solution proposée

## 1.1. Introduction

Comme indiqué précédemment, la conception d'un circuit doit être menée de manière à assurer la fonctionnalité de celui-ci tout en satisfaisant les spécifications du client. Ainsi, la satisfaction de ces contraintes (notamment la surface, la vitesse et la consommation de puissance du circuit) incite l'utilisation de techniques informatiques et de recherche opérationnelle. En effet, et comme nous le verrons, la complexité algorithmique du problème posé n'est pas polynomiale. D'où le recours à des algorithmes à base d'heuristiques ou de méta-heuristiques. Ce point sera développé après avoir précisé le contexte du problème.

## 1.2. Contexte du problème

La partie de contrôle d'un circuit intégré est généralement implémentée à l'aide d'un PLA (Programmable Logic Array) pour la structure régulière qu'il offre. Cette régularité permet de réduire les paramètres parasites (résistances, capacités et inductances), ce qui est bénéfique pour la rapidité et la consommation de puissance du circuit. Ceci étant, les composants du PLA doivent avoir certaines caractéristiques permettant de satisfaire les contraintes de surface, vitesse et consommation de puissance. Pour mieux situer le problème, considérons le comportement du PLA indiqué au Tableau 2.1. A ce tableau, est déduit le PLA indiqué à la Figure 2.1. Notons, toutefois, que dans les deux parties du PLA (parties AND et OR), seules les portes NOR sont utilisées. Ceci est dû au fait que de telles portes sont plus intéressantes que les portes NAND puisque les résistances et les capacités sont réduites. Pour implémenter une équation en une porte NOR qui devait être implémentée, à l'origine, en une porte NAND, il suffit juste de transformer l'équation. Par exemple, A and not(B) = not [not(A and not(B))] = not [not(A) or B].

**Tableau 2.1. Table de vérité**

| | ENTREES | | | | | SORTIES | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| N° de la ligne | C1 | C2 | Ye1 | Ye2 | Ye3 | Ys3 | Ys2 | Ys1 | F1 | F2 |
| 1 | 0 | 0 | X | X | 0 | 0 | 0 | 1 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 5 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 6 | 0 | 1 | 0 | 0 | X | 1 | 1 | 0 | 1 | 0 |
| 7 | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 8 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 9 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

Précisons, par exemple, que l'équation du signal $F_2$ déduite du PLA est la suivante :

$F_2 = \text{not}\,[\text{not}(F_{21} \text{ or } F_{22} \text{ or } F_{23} \text{ or } F_{24})] = F_{21} \text{ or } F_{22} \text{ or } F_{23} \text{ or } F_{24}$

Le premier not est dû à l'inverseur et le deuxième est dû à l'utilisation d'une porte NOR

$F_{21} = \text{not}(C_1 \text{ or } C_2 \text{ or } Y_{e3})$
$= \text{not}(C_1) \text{ and not}(C_2) \text{ and not}(Y_{e3})$

$F_{22} = \text{not}(C_1 \text{ or not}(C_2) \text{ or not}(Y_{e1}) \text{ or not}(Y_{e2}) \text{ or } Y_{e3})$
$= \text{not}(C_1) \text{ and } C_2 \text{ and } Y_{e1} \text{ and } Y_{e2} \text{ and not}(Y_{e3})$

$F_{23} = \text{not}(\text{not}(C_1) \text{ or } Y_{e1} \text{ or } Y_{e2} \text{ or } Y_{e3})$
$= C_1 \text{ and not}(Y_{e1}) \text{ and not}(Y_{e2}) \text{ and not}(Y_{e3})$

$F_{24} = \text{not}(\text{not}(C_1) \text{ or not}(C2) \text{ or Ye1 or } Y_{e2} \text{ or } Y_{e3})$

= $C_1$ and $C_2$ and not($Y_{e1}$) and not($Y_{e2}$) and not($Y_{e3}$)

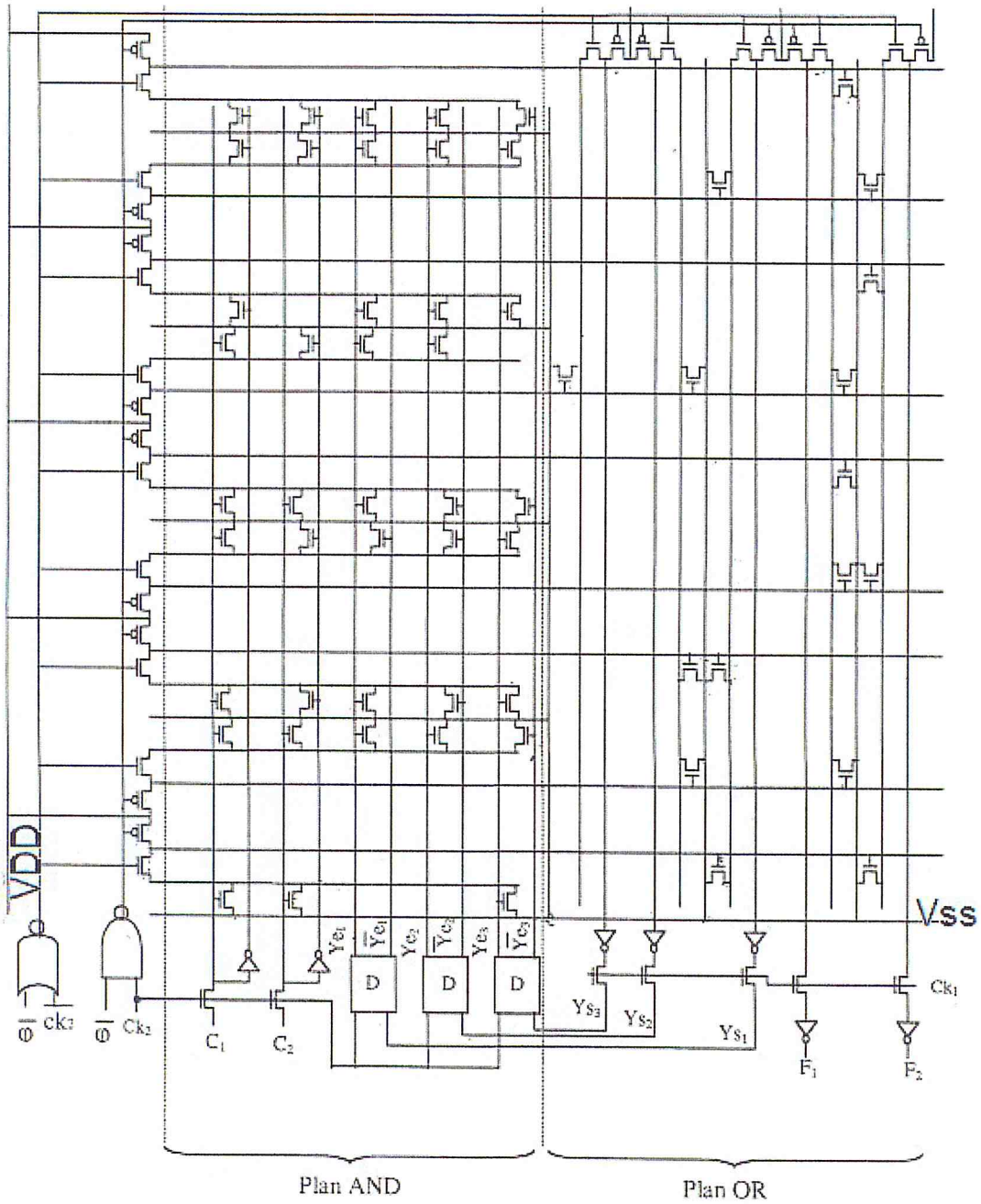


Fig. 2.1. PLA associé au tableau 2.1

Ainsi, les composants principaux du PLA (transistors et interconnexions) sont déduits à partir des équations logiques permettant de générer les signaux $F_i$ ; i=1, 2, 3, 4 et les caractéristiques du PLA dépendront des dimensions des transistors et des

interconnexions. Pour un nombre appréciable de ces composants, il n'est pas aisé, pour ne pas dire impossible, de trouver les dimensions adéquates permettant de satisfaire les contraintes imposées par le client. Bien plus, même en utilisant un processeur, la méthode exhaustive ne serait pas de complexité polynomiale, interdisant, de ce fait, d'obtenir la solution en un temps CPU raisonnable.
Du fait que l'espace des solutions est très large et que les dimensions des transistors et des interconnexions ne peuvent être quelconques (elles se situent dans des intervalles de valeurs dans le cas pratique), nous avons opté comme suit :

- le transistor peut avoir une largeur minimale $W_m$ ou une largeur maximale $W_M$
- l'interconnexion a aussi une largeur minimale $L_m$ ou maximale $L_M$

Notons que la longueur du transistor est définie par la technologie utilisée et que celle de l'interconnexion sera déduite de la topologie du PLA et aussi de la technologie utilisée.

A partir de ces indications, considérons que le PLA est constitué de $N_T$ transistors et de $N_I$ interconnexions. Nous avons alors ce qui suit :

Transistor 1 : $W_m$ $W_M$
Transistor 2 : $W_m$ $W_M$
……..
Transistor $N_T$ : $W_m$ $W_M$
Interconnexion 1 : $L_m$ $L_M$
Interconnexion 2 : $L_m$ $L_M$
……..
Interconnexion $N_I$ : $L_m$ $L_M$

La méthode exhaustive permettant de considérer tous les cas possibles puis de retenir la solution la plus intéressante consiste à considérer 2 * 2 * 2 …… * 2 = $2^{NT + NI}$ cas

possibles, ce qui n'est pas polynomial. En effet, le temps d'exécution T(N) est O(f(N)) s'il existe une constante $c > 0$ telle que : $T(N) \leq c * f(N) \; \forall \; N \geq N_0 \; ; N_0 \geq 0$

Dans notre cas, $N = N_T + N_I$ et on a bien $T(N) \leq 1 * 2^{NT + NI} \; \forall \; N \geq 0 \Rightarrow T(N)$ est $O(2^{NT + NI})$.

Il s'agit alors de développer une méthode basée sur une heuristique ou une méta-heuristique permettant de trouver, pour chaque transistor et chaque interconnexion, la largeur permettant de satisfaire les contraintes de surface, vitesse et consommation de puissance. Ceci fait l'objet de la section prochaine.

### 1.3. Démarche adoptée

Il existe de nombreuses méta-heuristiques utilisées pour la résolution de problèmes d'optimisation. Le choix de l'une ou de l'autre se fait selon sa facilité d'adaptation (codage, etc.) au problème concerné. Il serait néanmoins erroné de dire que l'une d'entre elles est la meilleure (si c'était le cas, la communauté universitaire travaillant dans ce domaine utiliserait précisément la dite méta-heuristique au détriment de toutes les autres). En fait, la qualité de la solution dépend beaucoup plus du choix de la solution initiale, du passage d'une solution à une autre, du critère d'arrêt et surtout d'éviter de recourir à l'aspect aléatoire. Concernant cet aspect de génération aléatoire, il est utile de noter qu'il engendre plusieurs problèmes [3]:

- possibilité de générer une solution non réalisable (perte inutile de temps CPU)
- possibilité de générer une (des) solution(s) précédemment étudiée(s) (perte de temps CPU sans pouvoir améliorer la qualité de la solution)
- possibilité de tomber dans un scénario stationnaire (périodiquement, les mêmes solutions précédemment étudiées sont générées, ce qui ne permet pas aussi d'améliorer la qualité de la solution quand bien même le nombre d'itérations est élevé)

Dans le cadre de notre travail, nous avions opté pour le développement d'algorithmes à base d'heuristiques tout en tenant compte des considérations qui viennent juste d'être citées. Nous avions préalablement dit qu'il s'agit de développer un outil d'aide à la conception d'une partie de contrôle d'un circuit intégré assujettie à des

contraintes de surface, vitesse et consommation de puissance. Dans le cas pratique, il existe essentiellement deux types de systèmes intégrés :

- les systèmes temps-réel (le temps de réponse est un paramètre critique)
- les systèmes portables (la consommation de puissance est un paramètre critique)

Pour ce faire, nous avions développé deux algorithmes à base d'heuristiques:

- l'un donnant la priorité au paramètre *temps*, mais assurant que les contraintes de surface et de consommation de puissance sont satisfaites,
- l'autre donnant la priorité au paramètre *consommation de puissance*, mais assurant que les contraintes de surface et de temps sont satisfaites

Avant de présenter ces deux heuristiques, nous allons étudier comment les différents paramètres influent sur la surface, la vitesse et la consommation de puissance du circuit.

### 1.3.1. Estimation de la surface

Considérons la partie de contrôle indiquée à la Figure 2.1. Nous allons commencer par l'estimation de la largeur du PLA. La largeur de la partie droite de celui-ci dépend du :

- nombre d'interconnexions (horizontales) véhiculant le signal de sortie d'une porte logique située dans la partie gauche du PLA (soit $N_{S1}$ ce nombre),
- nombre de lignes connectées à $V_{SS}$ qui est égal à $\lceil$Nombre de signaux de sortie / 2$\rceil$ à cause de la technique *miroir*[1] utilisée (soit $N_{S2}$ ce nombre)

Comme les interconnexions n'ont pas la même largeur, il s'agira de tenir compte de la largeur de chacune de ces $N_{S1}$ interconnexions (la largeur d'une ligne connectée à $V_{SS}$ étant fixe).

Soit $N_1 = N_{S1} + N_{S2}$. Ceci engendrera une largeur $L_{N1}$

Il s'agira aussi de tenir compte de l'espace requis entre deux interconnexions verticales voisines, ce qui donne :

[1] La technique miroir consiste à partager une ligne Vss ou Vdd à deux blocs adjacents d'un circuit.

$N_2$ = Distance1_entre_2_interconnexions * ($N_1$ – 1).

Ceci engendrera une largeur $L_{N2}$.

La largeur de la partie gauche du PLA dépend :

- du nombre d'interconnexions verticales qui se trouvent dans cette partie : $N_3$ = 2 * Nombre_de signaux_d'entrée + 3. Ceci engendrera une largeur $L_{N3}$.
- de la distance entre ces $N_3$ interconnexions : $N_4$ = Distance2_entre_les_$N_3$_interconnexions * ($N_3$ – 1). Ceci engendrera une largeur $L_{N4}$

La largeur du PLA sera alors estimée par: $L_S = \sum_{i=1}^{4} L_{Ni}$ .

Pour ce qui est de la hauteur du PLA, elle dépend de

- de la longueur des transistors :

$$H_1 = \sum_{i=1}^{Nb\_portes\_logiques} 3 * longueur\_Transistor_i$$

= 3 * Nb_portes_logiques * longueur_Transistor

- de la distance Poly_to_Active (règle de dessin à respecter dans la technologie ciblée) :

$$H_2 = \sum_{i=1}^{Nb\_portes\_logiques} 2 * Distance\_Poly\_to\_Active * 3$$

= 6 * Nb_portes_logiques * Distance_Poly_to_Active

Notons que la longueur du transistor et la distance Poly_to_Active (Active : diffusion N ou diffusion P) dépendent de la technologie ciblée.

Ainsi, la hauteur du PLA sera estimée par : H = $H_1$ + $H_2$.

La surface totale du PLA sera alors estimée par : S = $L_S$ * H (2.1)

### 1.3.2. Estimation du temps de réponse

Ce temps est estimé depuis que les signaux d'entrée sont introduits au PLA jusqu'au moment de l'obtention de tous les signaux de sortie. Ce délai implique aussi bien les portes logiques que les interconnexions.

#### 3.3.2.1. Estimation du délai d'une porte logique

Considérons une porte logique. Comme toutes les portes logiques sont implémentées en un style de conception dynamique, le délai d'une porte logique implique :

- le temps de précharge de la sortie de la porte (obtention du '1' logique)
- le temps d'évaluation du signal de sortie de la porte

Le temps de précharge implique un seul transistor. Un modèle de délai pour un transistor est le suivant :

$$D_{Tr} = \left[\frac{C_{load} * L}{\mu C_{ox} W (V_{dd} - V_{th})}\right]\left[\frac{2V_{th}}{V_{dd} - V_{th}} + \ln\frac{4(V_{dd} - V_{th})}{V_{dd}}\right] \quad (2.2)$$

Avec $\mu C_{ox} =$ { 35 $\mu A/V^2$ pour un transistor NMOS ; 11.67 $\mu A/V^2$ pour un transistor PMOS }

$C_{load}$ est la capacité de charge

L est la longueur du canal du transistor

$\mu$ est la mobilité des électrons (trous) pour le transistor NMOS (PMOS)

$C_{ox}$ est la capacité de l'oxyde

W est la largeur du transistor

$V_{dd}$ est la tension d'alimentation

$V_{th}$ est la tension de seuil du transistor

En considérant la Figure 2.1, la valeur de $C_{load}$ connectée aux transistors de précharge et d'évaluation est estimée comme suit :

$$\begin{aligned} C_{load1} = \quad & (W_p * Active\_To\_poly) * C_{puS\_diffusionP} \\ & + 2(W_p + Active\_To\_Poly) * C_{puL\_diffusionP} \\ & + (W_n * Active\_To\_Poly) * C_{puS\_diffusionN} \\ & + 2(W_n + Active\_To\_Poly) * C_{puL\_diffusionN} \end{aligned}$$

$C_{puS\_diffusionX}$ ($C_{puL\_diffusionX}$) est la capacité par unité de surface (longueur) de la diffusion X (P ou N).

En injectant la valeur de $C_{load}$ dans l'équation $D_{Tr}$, nous obtiendrons une estimation du temps de précharge $T_{Pr}$.

Pour ce qui est du temps d'évaluation du signal de sortie, celui-ci implique deux délais :

- le temps de réponse $T_{E1}$ du transistor d'évaluation
- le temps de réponse $T_{E2}$ de l'un des transistors de la porte NOR (notons que ces transistors sont en parallèle)

$T_{E1}$ se calcule à l'aide de l'équation de $D_{Tr}$ dans laquelle est injectée la valeur de $C_{load1}$

$T_{E2}$ se calcule de la même manière que $T_{E2}$, mais avec la valeur de $C_{load}$ suivante :

$$C_{load2} = L * H * C_{puS\_diffusionN} + 2(L + H) * C_{puL\_diffusionN}$$

$$\begin{aligned} \text{Avec: } L &= Nb * W_N + Nb * Distance_{Active\_To\_Active} \\ &= Nb * (W_N + Distance_{Active\_To\_Active}) \end{aligned}$$

où Nb est le nombre de transistors NMOS constituant la porte logique

$H = 2 * Distance_{Poly\_To\_Active}$

Le délai de la porte i sera donc : $D_{Pi} = T_{Pr} + T_{E1} + T_{E2}$

### 3.3.2.1. Estimation du délai d'une interconnexion

Ce délai est estimé par le modèle suivant :

$$d_{ij} = \frac{1}{2} r_{Wij} * l_{Wij} * \left(C_{Wij/ul} * l_{Wij}\right) + \left(C_{Wij/uS} * l_{Wij} * W_{interc}\right) + r_{Wij} * l_{Wij} * C_{Polyj} \qquad (2.3)$$

Avec $C_{Polyj} = \left(l_{Tr} * Poly\_To\_Active\right) * C_{Poly/uS} + 2 * \left(l_{Tr} + Poly\_To\_Active\right) * C_{Poly/uL}$

où $l_{Tr}$ est la longueur du transistor

Ainsi, considérant le PLA indiqué à la Figure 2.1 et le signal $F_2$ par exemple, on obtiendra :

Délai ($Y_{S1}$) = Délai ($P_{12}$) + Max (Délai ($P_1$), Délai ($P_3$), Délai ($P_8$))
+ Max ($d_{i, 12\_j}$) ; $i \in \{1, 3, 8\}$ ; $j \in \{1, 2, 3\}$

En considérant la table de vérité indiquée dans le Tableau 2.1, nous avons ce qui suit :

- $P_i$ ($1 \leq i \leq 9$) est une porte logique dont le comportement est décrit par la $i^{ème}$ ligne de la table de vérité
- $P_i$ ($10 \leq i \leq 14$) indique les transistors (représentés par un '1' dans la colonne i) composant cette porte logique $P_i$
- $d_{i, 12\_j}$ est le temps de transfert du signal issu de la porte i et alimentant la grille du $j^{ème}$ transistor de la porte 12 générant le signal de sortie $Y_{S1}$

En définitive, le délai obtenu avec les dimensions des transistors et celles des interconnexions déterminées à une itération donnée de l'algorithme sera :

Délai = max (Délai ($Y_{S1}$), Délai ($Y_{S2}$), Délai ($Y_{S3}$), Délai ($F_1$), Délai ($F_2$) )

### 1.3.3. Estimation de la consommation de puissance

Dans la technologie CMOS, il existe essentiellement deux types de consommation de puissance:

- la puissance dynamique due à l'activité du circuit (charges et décharges de capacités aux noeuds de sortie des portes en fonction des signaux d'entrée)
- la puissance due aux fuites de courant. Celle-ci n'est pas à négliger dans les technologies nouvelles des semi-conducteurs à cause du fait que les transistors, dans ces technologies, présentent des tensions de seuil très faibles. D'où la possibilité d'une fuite de courant entre les nœuds source et drain du transistor

Un modèle-type pour représenter la consommation dynamique est celui-ci :

$$P_{dyn} = 0.5 * V_{dd}^2 * f \sum_{i=1}^{Nb\_Portes} C_i * N_i \qquad (2.4)$$

- $V_{dd}$ est la tension d'alimentation
- $f$ est la fréquence de fonctionnement du circuit
- $C_i$ est la capacité attachée au nœud de sortie de la porte $i$
- $N_i$ est le nombre de transitions (charges, décharges) de $C_i$

Celui représentant la dissipation due aux fuites du courant est le suivant :

$$P_{leak} = 0.28125 * V_{dd} * K * W_{avg} * L * \left( Nb_{Trans\_N} * 10^{-V_{tN}/\alpha_V} + Nb_{Trans\_P} * 10^{-V_{tP}/\alpha_V} \right) \quad (2.5)$$

- $K$= 10 µA/µm

- $W_{avg}$ est la largeur moyenne des transistors
- $L$ est la longueur du transistor dans la technologie ciblée
- $Nb_{Trans\_N}$ ($Nb_{Trans\_P}$) est le nombre de transistors NMOS (PMOS) dans le circuit
- $V_{tN}$ ($V_{tP}$) est la tension de seuil du transistor NMOS (PMOS)
- $\alpha_V = 0.095$ V

Pour ce qui est du calcul de la puissance dynamique, le problème est bien plus important que ne laisse apparaître l'équation associée. En effet, le problème se pose avec le paramètre $N_i$ (nombre de charges et de décharges) qui dépend de plusieurs paramètres (vecteur des signaux d'entrée, style de conception, délais des portes logiques, ...). Il se trouve qu'il n'est pas possible d'exprimer ce paramètre comme étant une fonction analytique de ces paramètres. Ainsi, pour calculer la consommation moyenne ou maximale du circuit, le seul moyen est de considérer ce procédé :

- introduire un vecteur d'entrée $V_1$ au circuit: selon le type du circuit (statique, dynamique avec leurs variantes), les sorties des portes logiques seront à l'état logique '1' ou '0'
- réinjecter un autre vecteur d'entrée $V_2$ au circuit: certaines capacités vont se charger ou se décharger, ce qui permet de déterminer $N_i$, puis la consommation dynamique

Ainsi, pour un circuit à M entrées, le nombre de possibilités serait de $2^{2*M}$. Par exemple, pour un petit circuit comme un additionneur à 2 entrées, chacune étant de taille 32 bits (la taille totale serait alors de 64 bits), il faudrait alors considérer $2^{2*64} = 2^{128}$ cas, ce qui est déjà énorme pour un petit circuit. Pour un système complexe tel que les systèmes sur puce qui peuvent contenir des centaines de millions de portes logiques, il est impensable de procéder ainsi, le problème considéré étant en fait de complexité non polynomiale. Celle-ci est $O(2^{2*M})$, M étant le nombre de signaux d'entrée.

Pour l'estimation de la puissance totale (dynamique et celle due aux fuites de courant), nous avons utilisé un outil développé au CDTA [5, 6]. Il repose sur un algorithme génétique qui présente les caractéristiques suivantes:

- la population initiale est générée de manière judicieuse et non aléatoire: des individus représentatifs constituent chaque génération
- le passage d'une génération à une autre se fait aussi de manière argumentée (les opérations de croisement et de mutation arbitraires sont évitées)
- l'outil a été testé sur des benchmarks

### 1.3.4. Heuristique donnant la priorité au paramètre *temps*

Comme il a été indiqué précédemment, les systèmes sur puce actuels sont de deux types :

- ceux qui sont destinés pour des applications temps-réel et qui nécessitent donc que chaque tâche s'exécute dans le délai imparti
- ceux qui pour lesquels la consommation d'énergie pose un problème (exemple de systèmes portables) et qui donc nécessitent une conception à faible consommation de puissance

Dans le cadre de notre travail, nous avons développé deux grands algorithmes à base d'heuristiques :

- le premier algorithme repose sur une heuristique qui consiste à satisfaire, si possible (les contraintes doivent être vraisemblables), les contraintes de surface, vitesse et consommation de puissance tout en donnant la priorité au paramètre *temps*
- le deuxième consiste plutôt à satisfaire les mêmes trois contraintes, mais en donnant la priorité au paramètre *consommation de puissance*

Avant de donner plus de détails sur ces deux algorithmes, notons que pour le premier algorithme, la meilleure solution obtenue $S_i$ est remplacée par une nouvelle solution

$S_j$ si cette dernière améliore le paramètre *temps* même si les valeurs de la surface et de la consommation de puissance augmentent, mais toutefois satisfaisant les trois contraintes.

Il en est de même pour le deuxième algorithme qui consiste à remplacer la solution courante $S_i$ par une nouvelle solution $S_j$ si celle-ci améliore la consommation de puissance même si les valeurs de surface et de temps augmentent, mais toutefois satisfaisant les trois contraintes.

Nous verrons dans le prochain chapitre que le logiciel se compose de trois exécutables:

- le premier exécutable est généré à partir d'un programme codé en C qui s'exécute avec l'option –t ou –p. Si l'option –t (-p) est choisie, ce fichier exécutera, de manière automatique, le programme donnant la priorité au temps (respectivement à la consommation de puissance)
- le deuxième exécutable implémente l'algorithme donnant la priorité au *temps*
- le troisième exécutable implémente l'algorithme donnant la priorité à la *consommation de puissance*

Nous rappelons que la qualité de la solution dépend essentiellement de l'initialisation, du passage d'une solution à une autre et du critère d'arrêt. Dans le développement de ces deux algorithmes, nous avions surtout évité l'aspect *génération aléatoire de solution*, celui-ci pouvant engendrer des problèmes tels que [3, 7] :

- la génération d'une solution non réalisable
- la génération de solutions n'améliorant pas la fonction objective et gaspiller, inutilement, du temps CPU
- le risque de tomber dans un scénario stationnaire (régénérer, périodiquement, des ensembles de solutions auparavant générées)

Considérons maintenant comment la solution initiale a été définie. A partir des modèles de délai, nous savons parfaitement quels sont les paramètres qui influent sur le délai. Certains paramètres (tensions d'alimentation et de seuil de transistor, ...) étant fixés par le procédé technologique ciblé, nous nous intéressons à l'influence des dimensions des transistors et celle des interconnexions sur le délai.

L'équation (2.2) exprimant le temps de réponse d'un transistor montre clairement que ce temps est d'autant plus petit que la largeur du transistor W est grande. Pour une conception réelle d'un circuit, il n'est pas possible de considérer n'importe quelle largeur. Aussi, la larguer d'un transistor est choisie parmi deux valeurs extrêmes $W_{min}$ et $W_{max}$.

Pour ce qui est de la définition de la solution initiale en termes de dimensions de transistors, il est judicieux d'affecter, à chaque transistor, la largeur $W_{max}$ puisque l'algorithme consiste à donner la priorité au paramètre *temps*.

L'autre paramètre qui affecte le délai concerne les interconnexions. Pour une longueur donnée de cette interconnexion, une largeur $L_{max}$ assure un délai de transfert de données meilleur que si on optait pour une largeur $L_{min}$. En effet, considérons la Figure 2.2 :

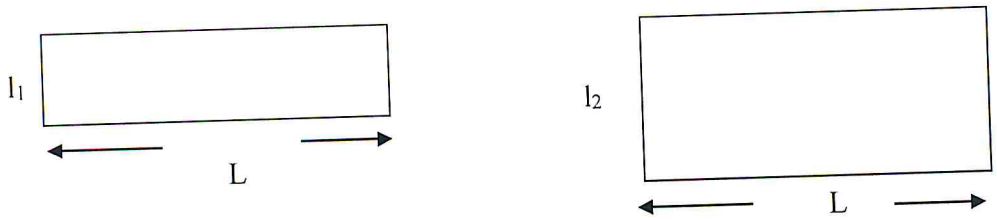


Fig.2.2. Effet de la largeur de l'interconnexion sur le temps de transfert de données.

Le temps de transfert d'une donnée étant proportionnel au produit R * C (Résistance * Capacité), nous avons pour :

- la première structure : $T_1 = K\,[2(L+l_1) * C_{puL} + (L*l_1) * C_{puS}] * 6\,R_{\blacksquare}$

- la deuxième structure : $T_2 = K\,[2(L+l_2) * C_{puL} + (L*l_2) * C_{puS}] * 3\,R_{\blacksquare}$

où $C_{puL}$ ($C_{puS}$) est la capacité par unité de longueur (surface) du matériau considéré (Aluminium, Cuivre) et $R_{\blacksquare}$ (ou $R_c$) est la résistance par carré.

Comme $l_2 = 2*l_1$, nous obtenons : $T_2 = K\,[2(L+2*l_1) * C_{puL} + (L*2*l_1) * C_{puS}] * 3\,R_{\blacksquare}$

Nous avons donc : $$\frac{T_2}{T_1} = \frac{K*\left((2L+4l_1)C_{puL} + 2L*l_1*C_{puS}\right)*3R_c}{K*\left((2L+2l_1)C_{puL} + L*l_1*C_{puS}\right)*6R_c}$$

$$= \frac{(L+2l_1)C_{puL} + L*l_1*C_{puS}}{(2L+2l_1)C_{puL} + L*l_1*C_{puS}}$$

Comme $2L + 2l_1 > L + 2*l_1$, nous avons alors : $T_2 / T_1 < 1$, et donc $T_1 > T_2$

Ainsi, pour le même type de matériau implémentant l'interconnexion et pour la même longueur, l'interconnexion ayant la plus grande largeur offre un meilleur temps de transfert de données.

Ainsi donc, la définition de la solution initiale, qui est en fait la meilleure solution pour le paramètre *temps*, se fait en affectant :

- la largeur $W_{max}$ à chaque transistor
- la largeur $L_{max}$ à chaque interconnexion

Quel est l'intérêt d'une telle initialisation ? Il est clair que si les trois contraintes sont satisfaites avec cette initialisation, la meilleure *solution* serait obtenue en *une seule* itération !

Dans le cas contraire, nous visons aussi une solution de qualité. Aussi, afin de ne pas trop s'éloigner de la meilleure solution (qui n'a malheureusement pas satisfait la

contrainte de surface et / ou celle de la consommation de la puissance), le déplacement de la solution courante vers une autre solution se fait de manière argumentée (et non aléatoire). Ce déplacement se fait aussi en opérant, graduellement, sur les éléments de la solution. Nous donnons, ci-après, les explications nécessaires concernant l'algorithme décrit par la suite.

- Partie A : La contrainte de temps (très forte, mal estimée) n'a pas été satisfaite avec la meilleure solution ($W_{max}$ pour tous les transistors et $L_{max}$ pour toutes les interconnexions). L'algorithme s'arrête en donnant une notification à l'utilisateur.
- Partie B : Tester si la contrainte de consommation de puissance est satisfaite. Si c'est le cas, considérer la partie C.
- Partie C : Tester si la contrainte de surface est satisfaite. Si c'est le cas, écrire la solution et arrêter l'exécution (pas la peine d'itérer encore car la solution ne peut être améliorée : rappelons que la solution initiale est la meilleure).
- Partie D : La contrainte de puissance est satisfaite, mais celle de la surface ne l'est pas. Si toutes les dimensions des transistors et toutes les interconnexions sont minimales, il n'est pas possible de diminuer la surface : l'algorithme s'arrête avec une notification donnée à l'utilisateur.
- Partie E : Il est possible de diminuer la surface. Afin de ne pas trop s'éloigner de la solution courante (qui est avantageuse au paramètre *temps*), nous allons changer les dimensions des transistors d'une *seule* porte seulement. Puisqu'il y'a eu changement de dimensions, le temps du circuit est recalculé. Si la contrainte de temps est satisfaite, ne plus faire de changements (qualité de la solution) et recalculer la consommation de puissance (recommencer le procédé indiqué à partir de la partie B). Sinon, considérer les transistors d'une autre porte logique. Si après avoir considéré les transistors de toutes les portes et qu'il n'a pas été possible de satisfaire la contrainte de temps en essayant de satisfaire celle de la surface, l'algorithme analyse les largeurs des interconnexions (Partie H). S'il a été possible de satisfaire la contrainte de temps en diminuant la (les) largeur(s) d'une (de certaines) interconnexion(s), reprendre le procédé à partir de la partie B (recalculer la consommation de

puissance et la surface, etc.). Si au cours du processus le changement de la largeur d'une interconnexion affecte la contrainte de temps, la largeur $L_{max}$ est réaffectée à l'interconnexion en cours de considération (pour garder la contrainte de temps satisfaite). Si la considération de toutes les interconnexions n'a pas permis de procéder à un changement de largeur (la contrainte de temps ne serait plus satisfaite), l'algorithme s'arrête en notifiant à l'utilisateur que la (les) contrainte(s) de temps et / ou de surface est (sont) très forte(s).

- Partie K : La contrainte de consommation de puissance n'est pas satisfaite : A partir de la solution courante, changer les dimensions des transistors $W_{max}$ en $W_{min}$ d'une seule porte à la fois (ne pas trop s'éloigner de la qualité de la solution). Recalculer, à chacun de ces changements, le temps du circuit. Si la contrainte de temps est satisfaite, reprendre le procédé à partir du calcul de la consommation de puissance, etc. (Partie B). Noter qu'à chaque fois qu'un changement de dimension de transistor ne satisfait plus la contrainte de temps, la dimension précédente est réaffectée à la porte en cours de considération (assurer que la contrainte de temps soit toujours satisfaite) et passer à une autre porte. Si la considération de toutes les portes ne résout pas le problème, une tentative est faite au niveau des interconnexions. De la même manière, une seule interconnexion est considérée à la fois (pour ne pas trop altérer la solution, intéressante, en cours). A chaque fois qu'un changement de largeur d'interconnexion ne permet plus de satisfaire la contrainte de temps, la largeur précédente est réaffectée à l'interconnexion en cours de considération, et une autre interconnexion est envisagée. Si un changement de largeur d'interconnexion a permis de garder la contrainte de temps satisfaite, le processus est repris à partir du recalcul de la consommation de puissance (Partie B). Si aucun changement de largeur d'interconnexion n'a été opéré sans affecter la contrainte de temps, l'algorithme s'arrête en notifiant à l'utilisateur que la (les) contrainte(s) de temps et / ou de puissance est (sont) très forte(s).

Rappelons que les caractéristiques d'un tel algorithme sont les suivantes:

- la solution initiale est définie telle qu'on obtienne le meilleur temps de réponse du circuit,
- le passage d'une solution à une autre se fait de manière graduelle et non brusque (ne pas trop affecter la qualité de la solution qui découle de la meilleure solution),
- l'algorithme s'arrête dans certains cas en donnant des notifications à l'utilisateur tout en assurant que l'exécution d'autres itérations est inutile

Enfin, notons que l'algorithme tel que décrit précédemment, considère, dans l'ordre, les dimensions des transistors, puis celles des interconnexions. Une question évidente se poserait d'elle-même : que serait la qualité de la solution si on examinait, d'abord, les interconnexions puis les transistors ?

Notons que dans le cas où la contrainte de consommation de puissance :

- est satisfaite, la partie E (considérer les transistors) est examinée avant la partie H (considérer les interconnexions)
- n'est pas satisfaite, la partie K (considérer les transistors) est examinée avant la partie N (considérer les interconnexions)

Afin d'assurer l'obtention d'une bonne solution, l'algorithme est exécuté quatre fois, en tenant compte des quatre possibilités suivantes :

- 1 : examiner, dans l'ordre, les parties E, H, K, puis N
- 2 : examiner, dans l'ordre, les parties E, H, N, puis K
- 3 : examiner, dans l'ordre, les parties H, E, K, puis N
- 4 : examiner, dans l'ordre, les parties H, E, N, puis K

Ainsi, la solution générant le meilleur temps du circuit tout en satisfaisant toutes les contraintes sera retenue.

**Algorithme :**

**Variables** T, P, S : double ;

**Constantes** : Wmax ← la largeur maximum des transistors, Wmin ← la largeur minimum des transistors, Lmax ← la largeur maximum des interconnexions, Lmin ← la largeur minimum des interconnexions;

**Début**

Initialiser les largeurs des transistors à Wmax ;

Initialiser les largeurs des interconnexions à LargMax ;

T ← Calculer_Temps() ;

**Si** ( T > TFixé )

**Alors** {écrire (" Contrainte du temps très forte ") ;

exit ;

}

**Fsi**

Etiq : P ← Calculer_Puissance() ;

**Si** ( P <= PFixé )

**Alors** { S ← Calculer_Surface() ;

**Si** (S <= SFixé )

**Alors** {écrire les résultats ;

exit ;

}

**Fsi** ;

**Si** (la largeur de chaque transistor égale à Wmin **et** la largeur de chaque interconnexion égale à LargMin)

**Alors** { écrire (" Contrainte de surface très forte ") ;

exit ;

}

**Fsi** ;

C_tps ← 0 ;

**/* Début de la partie E*/**

```
Tant que (C_tps == 0 et il existe une porte contenant des
          transistors de largeurs Wmax)
 Faire { changer ces W en Wmin ;
        T ← Calculer_Temps() ;
        Si (T <= TFixé )
        Alors C_tps ← 1 ;
        Sinon {réaffecter Wmax à ces W ;
                passer à une autre porte ;
               }
        Fsi
        }
 Fait
/* La fin de la partie E*/
 Si (C_tps == 1)
 Alors aller à étiq ;
 Fsi
  /*    Le début de la partie H   */
 Tant que ( C_tps == 0  et il existe une interconnexion de
           largeur LargMax)
 Faire { changer cette largeur en LargMin ;
         T ← Calculer_Temps() ;
         Si ( T <= TFixé )
         Alors C_tps ← 1 ;
         Sinon {Réaffecter LargMax à cette interconnexion ;
                 Passer à une autre interconnexion ;
                }
          Fsi
        }
  Fait
  /* La fin de la partie H   */
  Si (C_tps == 1)
```

```
        Alors aller à étiq ;
        Sinon {écrire (" Contrainte(s) de surface et/o du temps très
                        forte(s) ") ;
                exit ;
               }
        Fsi
       }
Sinon { C_tps ← 0 ;
        /* Le début de la partie K */
       Tant que ( C_tps == 0 et il existe une porte contenant des
                transistors de largeurs Wmax )
       Faire {Changer ces W en Wmin ;
              T ← Calculer_Temps() ;
              Si (T <= TFixé )
              Alors C_tps ← 1 ;
              Sinon { Réaffecter Wmax à ces W ;
                      passer à une autre porte ;
                     }
              Fsi
             }
        Fait
        /* La fin de la partie K */
        Si ( C_tps == 1)
        Alors aller à étiq ;
        Fsi
        /* Le début de la partie N * /
        Tant que ( C_tps == 0 et il existe une interconnexion de
                  largeur LargMax )
        Faire { Changer cette largeur en LargMin ;
                T ← Calculer_Temps() ;
                Si ( T <= TFixé  )
```

```
            Alors C_tps ← 1 ;
           Sinon { Réaffecter LargMax à cette interconnexion ;
                    Passer à une autre interconnexion ;
                   }
            Fsi
           }
      Fait
      /* La fin de la partie N */
      Si ( C_tps == 1 )
      Alors aller à étiq ;
      Sinon { écrire (" Contrainte(s) du temps et/ou de puissance
                       très forte(s) ") ;
              exit ;
             }
       Fsi
       }
  Fsi
Fin
```

### 1.3.5. Heuristique donnant la priorité au paramètre *consommation de puissance*

Comme pour la première heuristique, la solution initiale n'est pas générée de manière aléatoire, mais plutôt à obtenir la *meilleure* valeur de consommation de puissance. Celle-ci est obtenue en affectant :

- la largeur $W_{min}$ à chaque transistor
- la largeur $L_{min}$ à chaque interconnexion

Encore une fois, l'intérêt d'une telle initialisation est clair : si les trois contraintes sont satisfaites avec cette initialisation, la meilleure *solution* serait obtenue en *une seule* itération !

Dans le cas contraire, nous visons aussi une solution de qualité. Aussi, afin de ne pas trop s'éloigner de la meilleure solution (qui n'a malheureusement pas satisfait la contrainte de surface et / ou celle du temps), le déplacement de la solution courante vers une autre solution se fait de manière argumentée (et non aléatoire). Ce déplacement se fait aussi en opérant, graduellement, sur les éléments de la solution. Nous donnons, ci-après, les explications nécessaires concernant l'algorithme décrit par la suite.

- Partie A : La contrainte de puissance (très forte, mal estimée) n'a pas été satisfaite avec la meilleure solution ($W_{min}$ pour tous les transistors et $L_{min}$ pour toutes les interconnexions). L'algorithme s'arrête en donnant une notification à l'utilisateur.
- Partie B : Tester si la contrainte de temps est satisfaite. Si c'est le cas, considérer la partie C.
- Partie C : Tester si la contrainte de surface est satisfaite. Si c'est le cas, écrire la solution et arrêter l'exécution (pas la peine d'itérer encore car la solution ne peut être améliorée : rappelons que la solution initiale est la meilleure).
- Partie D : La contrainte de temps est satisfaite, mais celle de la surface ne l'est pas. Si toutes les dimensions des transistors et toutes les interconnexions sont minimales, il n'est pas possible de diminuer la surface : l'algorithme s'arrête avec une notification donnée à l'utilisateur.
- Partie E : Il est possible de diminuer la surface. Afin de ne pas trop s'éloigner de la solution courante (qui est avantageuse au paramètre *temps*), nous allons changer les dimensions des transistors d'une *seule* porte seulement. Puisqu'il y'a eu changement de dimensions, le temps du circuit est recalculé. Si la contrainte de temps est satisfaite, ne plus faire de changements (qualité de la solution) et recalculer la consommation de puissance (recommencer le procédé à partir du calcule de la surface indiqué dans la partie B). Sinon, considérer les transistors d'une autre porte logique. Si après avoir considéré les transistors de toutes les portes et qu'il n'a pas été possible de satisfaire la contrainte de temps en essayant de satisfaire celle de la surface, l'algorithme

analyse les largeurs des interconnexions (Partie H). S'il a été possible de satisfaire la contrainte de temps en diminuant la (les) largeur(s) d'une (de certaines) interconnexion(s), recalculer la consommation de puissance et reprendre le procédé à partir du calcul de la surface indiqué dans la partie B. Si au cours du processus le changement de la largeur d'une interconnexion affecte la contrainte de temps, la largeur $L_{max}$ est réaffectée à l'interconnexion en cours de considération (pour garder la contrainte de temps satisfaite). Si la considération de toutes les interconnexions n'a pas permis de procéder à un changement de largeur (la contrainte de temps ne serait plus satisfaite), l'algorithme s'arrête en notifiant à l'utilisateur que la (les) contrainte(s) de temps et / ou de surface est (sont) très forte(s).

- Partie K : La contrainte de temps n'est pas satisfaite : A partir de la solution courante, changer les dimensions des transistors $W_{min}$ en $W_{max}$ d'une seule porte à la fois (ne pas trop s'éloigner de la qualité de la solution). Recalculer, à chacun de ces changements, le temps du circuit. Si la contrainte de temps est satisfaite, recalculer la consommation de puissance. Si elle dépasse la valeur maximale fixée, l'algorithme s'arrête en donnant un message à l'utilisateur. Sinon, reprendre le procédé à partir du calcul de la surface, etc. (Partie B). Noter qu'à chaque fois qu'un changement de dimension de transistor ne satisfait plus la contrainte de temps, la dimension précédente est réaffectée à la porte en cours de considération (assurer que la contrainte de temps soit toujours satisfaite) et passer à une autre porte. Si la considération de toutes les portes ne résout pas le problème, une tentative est faite au niveau des interconnexions. De la même manière, une seule interconnexion est considérée à la fois (pour ne pas trop altérer la solution, intéressante, en cours). Si un changement de largeur d'interconnexion a permis de satisfaire la contrainte de temps, la consommation de puissance est recalculée. Si celle-ci dépasse la valeur maximale fixée, une notification est donnée à l'utilisateur et l'algorithme s'arrête. Sinon, le processus est repris à partir du calcul de la surface (Partie B). Si avec la considération de toutes les interconnexions la contrainte de temps n'a pas été satisfaite, l'algorithme s'arrête en notifiant à

l'utilisateur que la (les) contrainte(s) de temps et / ou de puissance est (sont) très forte(s).

Rappelons aussi que les caractéristiques d'un tel algorithme sont les suivantes :

- la solution initiale est définie telle qu'on obtienne la meilleure consommation de puissance du circuit,
- le passage d'une solution à une autre se fait de manière graduelle et non brusque (ne pas trop affecter la qualité de la solution qui découle de la meilleure solution),
- l'algorithme s'arrête dans certains cas en donnant des notifications à l'utilisateur tout en assurant que l'exécution d'autres itérations est inutile

Enfin, notons que tout comme le premier algorithme, l'algorithme qui vient d'être décrit, considère, dans l'ordre, les dimensions des transistors, puis celles des interconnexions. Or, il est possible que la solution serait différente si on considérait, tout d'abord, les interconnexions avant les transistors. En effet, dans le cas où la contrainte de temps :

- est satisfaite, la partie E (considérer les transistors) est examinée avant la partie H (considérer les interconnexions)
- n'est pas satisfaite, la partie K (considérer les transistors) est examinée avant la partie O (considérer les interconnexions)

De même que pour le premier algorithme, et afin d'assurer l'obtention d'une bonne solution, l'algorithme est exécuté quatre fois, en tenant compte des quatre possibilités suivantes :

- 1 : examiner, dans l'ordre, les parties E, H, K, puis O
- 2 : examiner, dans l'ordre, les parties E, H, O, puis K
- 3 : examiner, dans l'ordre, les parties H, E, K, puis O
- 4 : examiner, dans l'ordre, les parties H, E, O, puis K

Ainsi, la solution générant la meilleure consommation de puissance du circuit tout en satisfaisant toutes les contraintes sera retenue.

**Algorithme :**

**Variables** T, P, S : double ;

**Constantes** : Wmax ← la largeur maximum des transistors, Wmin ← la largeur minimum des transistors, Lmax ← la largeur maximum des interconnexions, Lmin ← la largeur minimum des interconnexions;

**Début**

Initialiser les largeurs des transistors à Wmin ;

Initialiser les largeurs des interconnexions à LargMin ;

P ← Calculer_Puissance() ;

**Si** ( P > PFixé )

**Alors** { écrire (" Contrainte de puissance très forte ") ;

exit ;

}

**Fsi**

T ← Calculer_Temps() ;

**Si** (T <= TFixé )

**Alors** { étiq : S ← Calculer_Surface() ;

**Si** ( S <= SFixé )

**Alors** { écrire les résultats ;

exit ;

}

**Fsi**

**Si** ( la largeur de chaque transistor égale à Wmin et la largeur de chaque interconnexion égale à LargMin )

**Alors** { écrire (" Contrainte de surface très forte ") ;

exit ;

```
        }
 Fsi
 C_tps ← 0 ;
 /* Le début de la partie E */
 Tant que ( C_tps == 0 et il existe une porte contenant des
            transistors de largeurs Wmax )
 Faire { changer ces W en Wmin ;
         T ← Calculer_Temps() ;
         Si ( T <= TFixé )
         Alors C_tps ← 1 ;
         Sinon { réaffecter Wmax à ces W ;
                 passer à une autre porte ;
               }
         Fsi
        }
 Fait
 /* La fin de la partie E */
 Si ( C_tps == 1 )
 Alors { P ← Calculer_Puissance() ;
         Aller à étiq ;
        }
  Fsi
  /* Le début de la partie H */
  Tant que ( C_tps == 0 et il existe une interconnexion de
             largeur LargMax )
  Faire { changer cette largeur en LargMin ;
          T ← Calculer_Temps() ;
          Si ( T <= TFixé )
          Alors C_tps ← 1 ;
          Sinon { réaffecter LargMax à cette interconnexion ;
                  passer à une autre interconnexion ;
                }
```

```
            Fsi
           }
      Fait
      /* La fin de la partie H */
      Si (C_tps == 1 )
      Alors { P ← Calculer_Puissance ;
              Aller à étiq ;
            }
      Sinon { écrire (" Contrainte(s) de surface et/ou du temps très
                       forte(s) ") ;
              exit ;
            }
       Fsi
      }
Sinon {C_tps ← 0 ;
        /* Le début de la partie K */
       Tant que (C _tps == 0 et il existe une porte contenant des
                 transistors de largeurs Wmin )
       Faire { changer ces W en Wmax ;
               T ← Calculer_Temps() ;
                Si (T <= TFixé )
                Alors C_tps ← 1 ;
                Fsi
               }
        Fait
        /* La fin de la partie K */
        Si  ( C_tps == 1 )
       Alors { P ← Calculer_Puissance() ;
                Si  ( P <= PFixé )
                Alors aller à étiq ;
```

```
        Sinon { écrire (" Contrainte(s) du temps et/ou de  puissance très
                  forte(s) ") ;
                 exit ;
                }
         Fsi
        }
Fsi
/* Le dédut de la partie O */
Tant que ( C_tps == 0 et il existe une interconnexion de
            largeur LargMin )
Faire { changer cette largeur en LargMax ;
        T ← Calculer_Temps() ;
        Si ( T <= TFixé )
        Alors C_tps ← 1 ;
        Fsi
       }
Fait
/* La fin de la partie O */
Si (C_tps == 1 )
Alors { P ← Calculer_Puissance() ;
         Si ( P <= PFixé )
         Alors aller à étiq ;
         Sinon {écrire (" Contrainte(s) du temps et/ou de puissance très
                   forte(s) ") ;
                   exit ;
                  }
          Fsi
        }
  Sinon { écrire ("Contrainte du temps très forte ") ;
          exit ;
        }
  Fsi
```

}

**Fsi**

**Fin**

### 1.4. Conclusion

Après avoir abordé les aspects essentiels portant sur le temps, la consommation de puissance et la surface d'un circuit intégré, nous avons montré que le problème considéré n'est pas de complexité polynomiale. Il est donc essentiel de le résoudre par des algorithmes à base d'heuristiques ou de méta-heuristiques. Nous avons alors décrit deux algorithmes qui ont pour caractéristiques de viser l'obtention d'une solution de qualité. Ceci, grâce à :

- une définition adéquate de la solution initiale,
- un changement graduel de paramètres (dimensions des transistors et des interconnexions) lors du passage d'une solution à une autre (ne pas trop altérer la qualité de la solution courante qui découle de la solution idéale)
- l'évitement de génération aléatoire de solutions [3]

L'implémentation de ces deux algorithmes sera abordée dans le chapitre suivant.

## 2. Mise en œuvre de la solution

### 2.1. Introduction

Nous abordons dans ce chapitre les aspects portant sur les structures de données, les fichiers et les modules utilisés.

### 2.2. Conception de la solution

Le rôle de la phase de la conception est de démystifier le fonctionnement du système, pour cela nous allons décrire l'ensemble des moyens et des procédures utilisées.

Afin de rendre l'application lisible, réutilisable, facile à maintenir et à modifier, nous avons fractionnés notre programme en plusieurs

modules, que l'on compile séparément. Chacun a son rôle. Ces modules sont :

- **principal.c :** fait l'appel des objets autosizeA_t, autosizeB_t, autosizeC_t et autosizeD_t, ou autosizeA_p, autosizeB_p, autosizeC_p et autosizeD_p, il contient la fonction main().

**2.2.1 t_autosizeA.c :** spécialisé dans la génération de l'objet autosizeA_t. Celui-ci contient les fonctions primitives suivantes : Calculer_Temps (), Calculer_Surface(), Calculer_Puissance().

**2.2.2 t_autosizeB.c :** spécialisé dans la génération de l'objet autosizeB_t. Celui-ci contient les fonctions primitives suivantes : Calculer_Temps (), Calculer_Surface(), Calculer_Puissance().

**2.2.3 t_autosizeC.c :** spécialisé dans la génération de l'objet autosizeC_t. Celui-ci contient les fonctions primitives suivantes : Calculer_Temps (), Calculer_Surface(), Calculer_Puissance().

**2.2.4 t_autosizeD.c :** spécialisé dans la génération de l'objet autosizeD_t. Celui-ci contient les fonctions primitives suivantes : Calculer_Temps (), Calculer_Surface(), Calculer_Puissance().

**2.2.5 p_autosizeA.c :** spécialisé dans la génération de l'objet autosizeA_p. Celui-ci contient les fonctions primitives suivantes : Calculer_Temps (), Calculer_Surface(), Calculer_Puissance().

**2.2.6 p_autosizeB.c :** spécialisé dans la génération de l'objet autosizeB_p. Celui-ci contient les fonctions primitives suivantes : Calculer_Temps (), Calculer_Surface(), Calculer_Puissance().

**2.2.7 p_autosizeC.c :** spécialisé dans la génération de l'objet autosizeC_p. Celui-ci contient les fonctions primitives

suivantes : Calculer_Temps (), Calculer_Surface(), Calculer_Puissance().

**2.2.8 p_autosizeD.c :** spécialisé dans la génération de l'objet autosizeD_p. Celui-ci contient les fonctions primitives suivantes : Calculer_Temps (), Calculer_Surface(), Calculer_Puissance().

Tous ces modules seront définis et présentés à la suite de ce chapitre.

## 2.3. Plateforme technique utilisée

Dans cette étape nous allons décrire toutes les plateformes utilisées car la diversité des plateformes influe sur la performance de l'application.

**2.3.1 Matériels :** Les plateformes utilisées sont :

- 2 Machines **Acer** :
  - Processeur : Inter (R) CPU 2.60 GHz
  - Mémoire : 4.00 Go

- Une machine Dell
  - Processeur : AMD CPU 3.19 GHz
  - Mémoire : 4.00 Go

**2.3.2 Systèmes d'exploitation :** Dans notre travail nous avons utilisées les deux systèmes d'exploitation suivants :

- LINUX : OpenSUSE 11.
- LINUX : OpenSUSE 12.

## 2.4. Architecture technique de la solution

Le schéma ci-après va mieux expliquer la structure et l'architecture de la solution proposée :

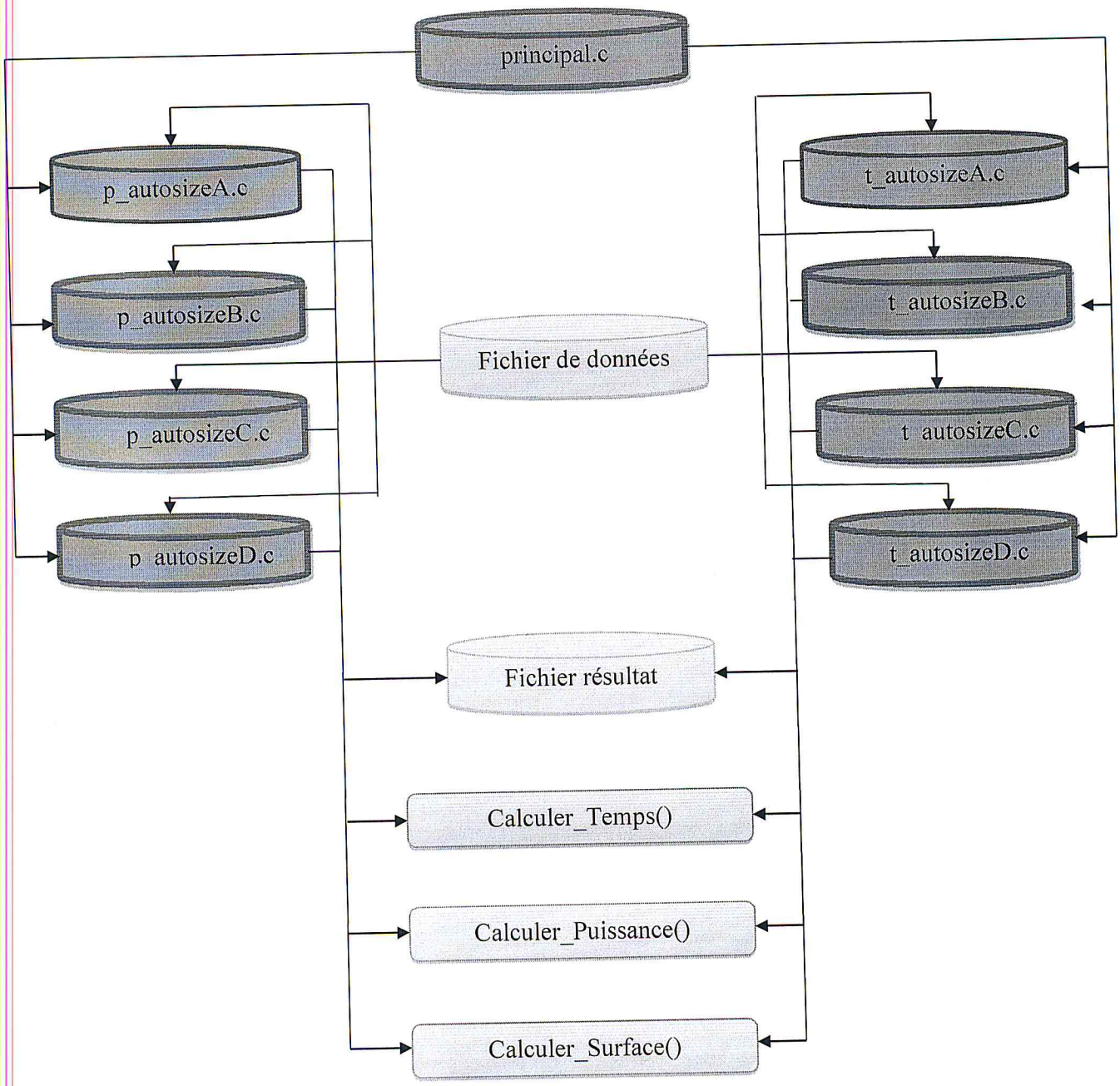


Fig. 2.3 : schéma synoptique.

## 2.5. Mise en œuvre :

La phase de la mise en œuvre est pour l'objectif de montrer comment mettre en œuvre les principes mentionnés dans la phase de la conception.

**Étape 1 : les fichiers "headers"**

Les fichiers headers ou les fichiers en-tête se sont des fichiers de ressources partagés dans un même projet ou parmi plusieurs projets, qui contiennent toutes les variables des valeurs constantes et toutes les structures de données utilisées dans notre travail. Il faut inclure ces fichiers dans tous les modules de notre application qui nécessitent un ou plusieurs composants de ces fichiers.

Pour cela nous avons créés deux fichiers headers :

1. **DeclVal. h :** qui comporte les déclarations de toutes les variables utilisées dans notre application, pour calculer la surface, le délai, ou la puissance, nous citons par exemple : les capacités et résistances utilisés, les largeurs des transistors et des interconnexion…etc.
   On l'inclure dans les modules adéquats de notre application, en ajoutant au début de chacun l'instruction : #include " DeclVal . h ".
   Notons que les valeurs des variables dans ces fichiers sont définies par la technologie utilisée.

2. **decl.h :** qui contient toutes les structures de données utilisées dans notre programme. On l'inclure dans les modules adéquats de notre application, en ajoutant au début de chacun l'instruction : #include " decl . h ".
   Nous avons utilisées cinq structures de données, quatre pour les transistors, et une pour les interconnexions.

   Les structures « TransistorsPartie1 », et « TransistorsPartie2 » décrient les porte d'entrées et de sorties, qui contiennent un transistor PMos et un transistor d'évaluation. Les transistors NMos de la partie

d'entrée sont représentés par la structure « TransistorsNmos », et ceux de la partie de sortie sont représentés par la structure « EntreePartie2 » ainsi que les portes d'entrées associées à une porte de sortie donnée.

**TransistorsPartie1 :**

- **Ligne** : un entier qui indique le numéro de la ligne d'une porte d'entrée donnée.
- **WP** : un caractère de valeur '0' ou '1' qui indique la largeur du transistor PMos d'une porte d'entrée donnée.
- **WE** : un caractère de valeur '0' ou '1' qui indique la largeur du transistor d'évaluation d'une porte d'entrée donnée.
- **Delai** : un réel qui indique le délai d'une porte d'entrée donnée.
- **Nmos*** : un pointeur qui pointe vers les transistors NMos d'une porte d'entrée donnée.
- **Next*** : un pointeur qui pointe vers la porte d'entrée suivante.

**TransistorsPartie2 :**

- **colonne** : un entier qui indique le numéro de la colonne d'une porte de sortie donnée.
- **WP** : un caractère de valeur '0' ou '1' qui indique la largeur du transistor PMos d'une porte de sortie donnée.
- **WE** : un caractère de valeur '0' ou '1' qui indique la largeur du transistor d'évaluation d'une porte de sortie donnée.
- **Delai** : un réel qui indique le délai d'une porte de sortie donnée.
- **Entree*** : un pointeur qui pointe vers la structure EntreePartie2.
- **Next*** : un pointeur qui pointe vers la porte d'entrée suivante.

**TransistorsNmos :**

- **num :** un entier qui indique le numéro de la colonne d'un transistor NMos dans une porte d'entrée donnée.
- **WN :** un caractère qui prend la valeur '0' ou '1' et qui indique la largeur d'un transistor NMos donnée.
- **Next* :** un pointeur qui pointe vers le prochain transistor NMos d'une porte d'entrée donnée.

**EntreePartie2 :**

- **WN :** un caractère qui prend la valeur '0' ou '1' et qui indique la largeur d'un transistor NMos donnée.
- **adressePorte* :** un pointeur qui pointe vers une porte d'entrée associée à une porte de sortie donnée.
- **I* :** un pointeur qui pointe vers l'interconnexion de sortie d'une porte d'entrée donnée.
- **Next* :** un pointeur qui pointe vers le prochain transistor NMos d'une porte donnée.

**Interconnexion :**

- **VH :** un caractère qui prend la valeur 'V' pour une interconnexion Verticale ou 'H' pour une interconnexion Horizontale.
- **num :** un entier qui indique le numéro d'une interconnexion donnée.
- **W :** un caractère de valeur '0' ou '1' qui indique la largeur d'une interconnexion donnée.
- **Next* :** un pointeur qui pointe vers l'interconnexion suivante.

### Étape 2 les fichiers sources

Nous représentons ci-après les différents fichiers sources :

- **principal . c** : fait la vérification des paramètres entrés par l'utilisateur, et fait appel soit aux fichiers t_autosizeA.c, t_autosizeB.c, t_autosizeC.c et t_autosizeD.c, si l'utilisateur tape « -t », ou bien aux fichers p_autosizeA.c, p_autosizeB .c, p_autosizeC.c et p_autosizeD.c s'il tape « -p ».
- **t_autosizeA.c** : ce fichier comporte l'algorithme qui examine, dans l'ordre, les parties **E,H,K** et **N**.
- **t_autosizeB.c** : comporte l'algorithme qui examine, dans l'ordre, les parties **E,H,N** puis **K.**
- **t_autosizeC.c** : comporte l'algorithme qui examine, dans l'ordre, les parties **H,E,K** puis **N**.
- **t_autosizeD.c** : comporte l'algorithme qui examine, dans l'ordre, les parties **H,E,N** puis **K**.
- **p_autosizeA,c** : comporte l'algorithme qui examine, dans l'ordre, les parties **E,H,K** et **O**.
- **p_autosizeB,c** : comporte l'algorithme qui examine, dans l'ordre, les parties **E,H,O** et **K**.
- **p_autosizeC,c** : comporte l'algorithme qui examine, dans l'ordre, les parties **H,E,K** et **O .**
- **p_autosizeD,c** : comporte l'algorithme qui examine, dans l'ordre, les parties **H,E,O** et **K**.

**Étape 3 définition des formats des fichiers d'enregistrement**

**Les fichiers d'entrées :**

- **Fichier de données :** C'est un fichier de nom quelconque, il inclut des informations sur le circuit donné, comme la table de vérités, le nombre d'éléments d'entré et de sortie dans cette table, ainsi que le nombre de ligne de la table. Le fichier contient aussi les contraintes de temps, surface, et de puissance spécifiées par l'utilisateur.

  La figure suivante montre un modèle de ce fichier :

fichier d'entrée:

contraintes pour le circuit:
temps= 0.02
puissance= 23.0
surface= 0.12

informations sur la table:
nombre de variables d'entrées= 5
nombre de variables de sorties= 5
nombre de lignes= 4

table de verité:
0 0 x x 0 0 0 1 0 1
0 0 0 0 1 0 1 0 1 0
0 1 0 0 x 1 1 0 1 0
1 x 0 0 0 0 0 0 0 1

Fig. 2.4 : Model d'un fichier d'entrée

**Les fichiers de sortie :**

C'est dans ces fichiers où nous obtiendront nos résultats finaux qui concernent tous les informations sur les numéros et les largeurs des interconnexions, ainsi que les lignes, les colonnes et les largeurs des transistors, plus les valeurs calculés de la surface, du temps et la puissance d'un circuit donné.

Nous avons définis deux types de fichiers de résultats qui contiennent tous les deux les mêmes types de résultats sauf que :

- **Les fichiers du type Résultats_X_FichierDeDonnées_t :** ce sont les fichiers de résultats qui sont générer après l'exécution des modules

**t_autosizeA.c, t_autosizeB.c, t_autosizeC.c** et **t_autosizeD.**c, qui contient toutes les résultats déjà mentionnées. Où le X peut prendre les valeurs A,B,C ou D selon le module exécuté, et FichierDeDonnées prend le nom du fichier de données entrée par l'utilisateur.

Ces quatre fichiers sont :

1. **Résultats_A_FichierDeDonnées_t :** contient les résultats du fichier **t_autosizeA.c**.
2. **Résultats_B_FichierDeDonnées_t :** contient les résultats du fichier **t_autosizeB.c**.
3. **Résultats_C_FichierDeDonnées_t :** contient les résultats du fichier **t_autosizeC.c**.
4. **Résultats_D_FichierDeDonnées_t :** contient les résultats du fichier **t_autosizeD.c**.

Ainsi, le fichier qui a le meilleur temps du circuit tout en satisfaisant toutes les contraintes sera retenu.

- **Les fichiers du type Résultats_X_FichierDeDonnées_p :** ce sont les mêmes que les précédents fichiers de résultats, sauf que ces derniers sont générer après l'exécution des modules p_autosizeA.c , p_autosizeB.c , p_autosizeC.c et p_autosizeD.c .

Ces fichiers sont :

1. **Résultats_A_FichierDeDonnées_p :** contient les résultats du fichier **p_autosizeA.c**.
2. **Résultats_B_FichierDeDonnées_p :** contient les résultats du fichier **p_autosizeB.c**.
3. **Résultats_C_FichierDeDonnées_p :** contient les résultats du fichier **p_autosizeC.c**.
4. **Résultats_D_FichierDeDonnées_p :** contient les résultats du fichier **p_autosizeD.c**.

Ainsi, le fichier qui a la meilleure consommation de puissance du circuit tout en satisfaisant toutes les contraintes sera retenu.

Un exemple de fichier de résultat quelconque est le suivant :

temps= 4.071497e-11 , puissance= 1.495292e-01, surface= 2.661093e-08

Ligne 1 :
Transistor Pmos : Largeur=1.600000e-05, Transistor d'evaluation :
Largeur=1.600000e-05
Transistors Nmos : Largeur T1 =1.600000e-05 Largeur T2 =1.600000e-05 Largeur T3 =1.600000e-05
Ligne 2 :
Transistor Pmos : Largeur=1.600000e-05, Transistor d'evaluation :
Largeur=1.600000e-05
Transistors Nmos : Largeur T1 =1.600000e-05 Largeur T2 =1.600000e-05 Largeur T3 =1.600000e-05 Largeur T4 =1.600000e-05 Largeur T5 =1.600000e-05
Colonne 1 :
Transistor Pmos : Largeur=1.600000e-05, Transistor d'evaluation :
Largeur=1.600000e-05
Transistors Nmos : Largeur T1 =1.600000e-05
Interconnexion numero 1 : largeur =7.000000e-06
Interconnexion numero 2 : largeur =7.000000e-06

Fig. 2.5 : Format de fichier résultats.

## 2.6. Conclusion

Nous avons décrit dans ce chapitre les différents éléments ayant servi pour la conception de notre application. Notre application a été implémentée en langage C sous Suse/Linux et testée sur différents jeux d'essais. La présentation des résultats obtenus fait l'objet du chapitre suivant.

# Partie III

# RESULTATS

## 1. Résultats

### 1.1. Introduction

Nous présentons dans ce chapitre les résultats obtenus en utilisant la plateforme suivante :

- Système d'exploitation : Linux Suse 11
- Processeur : AMD CPU 3.19 GHz.
- Mémoire : 4.00 Go RAM.

Les algorithmes présentés dans le précédent chapitre ont été implémentés dans le langage C. Pour la compilation des différents programmes, nous avons utilisé l'utilitaire Makefile qui a pour avantage de générer de manière automatique l'exécutable après chaque modification d'un ou de plusieurs fichiers sans se soucier des différentes compilations du fait que les dépendances entre les différents fichiers sont déjà établies et indiquées dans le fichier Makefile que nous présentons ci-après :

```
autosizeA_p:  p_autosizeA.o
        cc -g -o autosizeA_p p_autosizeA.o  -lm
autosizeB_p:  p_autosizeB.o
        cc -g -o autosizeB_p p_autosizeB.o  -lm
autosizeC_p:  p_autosizeC.o
        cc -g -o autosizeC_p p_autosizeC.o  -lm
autosizeD_p:  p_autosizeD.o
        cc -g -o autosizeD_p p_autosizeD.o  -lm
p_autosizeA.o: p_autosizeA.c decl.h declVal.h
        cc -g -c p_autosizeA.c
p_autosizeB.o: p_autosizeB.c decl.h declVal.h
        cc -g -c p_autosizeB.c
p_autosizeC.o: p_autosizeC.c decl.h declVal.h
        cc -g -c p_autosizeC.c
p_autosizeD.o: p_autosizeD.c decl.h declVal.h
        cc -g -c p_autosizeD.c

autosizeA_t:  t_autosizeA.o
```

```
        cc -g -o autosizeA_t t_autosizeA.o -lm
autosizeB_t:  t_autosizeB.o
        cc -g -o autosizeB_t t_autosizeB.o -lm
autosizeC_t:  t_autosizeC.o
        cc -g -o autosizeC_t t_autosizeC.o -lm
autosizeD_t:  t_autosizeD.o
        cc -g -o autosizeD_t t_autosizeD.o -lm
t_autosizeA.o: t_autosizeA.c decl.h declVal.h
        cc -g -c t_autosizeA.c
t_autosizeB.o: t_autosizeB.c decl.h declVal.h
        cc -g -c t_autosizeB.c
t_autosizeC.o: t_autosizeC.c decl.h declVal.h
        cc -g -c t_autosizeC.c
t_autosizeD.o: t_autosizeD.c decl.h declVal.h
        cc -g -c t_autosizeD.c

principal: principal.c
        cc -g -o principal principal.c
```

L'exécution se fait à l'aide de la commande suivante :

principal – t Fichier_De_Données  ou

principal –p Fichier_De_Données

Fichier_De_Données est le fichier décrivant le comportement de la partie de contrôle ainsi que les contraintes de temps, de consommation de puissance et de surface. Nous en donnons ci-après un exemple :

fichier d'entrée:

contraintes pour le circuit:

temps= 45.e-12

puissance= 350.e-3

surface= 5.e-6

informations sur la table:

nombre de variables d'entrées= 5

nombre de variables de sorties= 5

nombre de lignes= 9

table de vérité:

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

- Les 3 contraintes sont respectivement de 45 ps, 350 mW et 5 mm2
- Le nombre de lignes indique le nombre de lignes de la partie de contrôle implémentée par un PLA
- Il y'a 5 signaux d'entrée et 5 signaux de sortie
- Chacun des 5 signaux de sortie dépend des 5 signaux d'entrée

L'option –t lance l'exécution de 4 exécutables donnant la priorité au paramètre *temps* tout en satisfaisant, si possible, les 3 contraintes (temps, consommation de puissance, surface). Pour ce qui est des 4 exécutables, ils sont issus de la même heuristique mais portent sur 4 combinaisons possibles, selon que le traitement commence d'abord par les dimensions des transistors ou celles des interconnexions. Il est évident que la meilleure solution sera celle donnant la plus faible valeur de temps tout en satisfaisant les 3 contraintes.

Il en est de même pour l'option –p mais donnant la priorité au paramètre *consommation de puissance*.

### 1.2 Présentation des résultats

Nous présentons dans ce qui suit 2 tableaux indiquant les résultats obtenus pour la même partie de contrôle, mais avec différentes contraintes. Les résultats du premier tableau (deuxième tableau) ont été respectivement obtenus avec l'option –t et l'option –p.

**Tableau 3.1 Résultats obtenus en donnant la priorité au temps**

| $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) | Combinaison A | | | Combinaison B | | | Combinaison C | | | Combinaison D | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) | $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) | $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) | $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) |
| 45 | 950 | 5 | 40.71 | 245.24 | 0.027 | 40.71 | 245.24 | 0.027 | 40.71 | 245.24 | 0.027 | 40.71 | 245.24 | 0.027 |
| 35 | 950 | 5 | Pas de solution : contrainte de temps très forte | | | | | | | | | | | |
| 50 | 150 | 0.015 | Pas de solution : contrainte de surface très forte | | | | | | | | | | | |
| 50 | 150 | 5 | 41.77 | 149.71 | 0.017 | 40.71 | 148.30 | 0.020 | 41.77 | 149.71 | 0.017 | 40.71 | 148.30 | 0.020 |
| 45 | 90 | 5 | Pas de solution : contrainte de puissance très forte | | | | | | | | | | | |
| 41 | 150 | 5 | 40.71 | 138.54 | 0.020 | 40.71 | 148.30 | 0.020 | 40.71 | 138.54 | 0.020 | 40.71 | 148.30 | 0.020 |
| 60 | 140 | 10 | 41.77 | 134.23 | 0.017 | 40.71 | 131.22 | 0.020 | 41.77 | 134.23 | 0.017 | 40.71 | 131.22 | 0.020 |
| 42 | 92 | 10 | 41.77 | 91.71 | 0.016 | 41.77 | 91.71 | 0.016 | 41.77 | 91.71 | 0.016 | 41.77 | 91.71 | 0.016 |
| 41 | 92 | 10 | Pas de solution : contrainte de temps ou de puissance très forte | | | | | | | | | | | |
| 42 | 150 | 10 | 41.77 | 149.71 | 0.017 | 40.71 | 148.30 | 0.020 | 41.77 | 149.71 | 0.017 | 40.71 | 148.30 | 0.020 |
| 42 | 150 | 5 | 41.77 | 149.71 | 0.017 | 40.71 | 148.30 | 0.020 | 41.77 | 149.71 | 0.017 | 40.71 | 148.30 | 0.020 |
| 50 | 250 | 0.030 | 40.71 | 245.24 | 0.027 | 40.71 | 245.24 | 0.027 | 40.71 | 245.24 | 0.027 | 40.71 | 245.24 | 0.027 |

Dans le tableau 3.1, on peut observer ce qui suit :

- dans la 1ère ligne, toutes les contraintes ont été satisfaites (il s'agit en fait de la meilleure solution qu'on puisse obtenir pour le temps)
- dans la 2ème ligne, il n'y'a pas de solution à cause de la contrainte de temps qui est très forte (la valeur minimale possible est 40.71 ps)
- dans la 3ème ligne, il n'y'a pas de solution à cause de la contrainte de surface qui est très forte (la valeur minimale possible est 0.016 mm2) - dans la 5ème ligne, il n'y'a pas de solution à cause de la contrainte de puissance qui est très forte (la valeur minimale possible est 91.71 mW)
- les combinaisons A, B, C et D ne donnent pas toujours les mêmes résultats : dans la 7ème ligne, la modification des dimensions pour satisfaire la contrainte de consommation de puissance a commencé par celle des interconnexions – combinaisons B et D- ce qui a permis de diminuer certaines capacités, donc la

consommation de puissance sans faire aucun ou peu de changements de dimensions de transistors (initialisées à Wmax). Ceci a permis d'obtenir de meilleures valeurs de temps et de consommation de puissance qu'avec les combinaisons A et C
De même, nous allons interpréter, dans ce qui suit, les résultats obtenus avec l'option –p.

**Tableau 3.2 Résultats obtenus en donnant la priorité à la consommation de puissance**

| $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) | Combinaison A | | | Combinaison B | | | Combinaison C | | | Combinaison D | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) | $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) | $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) | $T_F$ (ps) | $P_F$ (mW) | $S_F$ ($mm^2$) |
| 45 | 350 | 5 | 41.77 | 91.71 | 0.016 | 41.77 | 91.71 | 0.016 | 41.77 | 91.71 | 0.016 | 41.711 | 91.71 | 0.016 |
| 45 | 90 | 5 | Pas de solution : contrainte de puissance très forte | | | | | | | | | | | |
| 45 | 900 | 0.010 | Pas de solution : contrainte de surface très forte | | | | | | | | | | | |
| 41 | 990 | 5 | 40.71 | 126.84 | 0.020 | 40.71 | 245.24 | 0.027 | 40.71 | 126.84 | 0.020 | 40.71 | 245.24 | 0.027 |
| 40.6 | 150 | 5 | Pas de solution : contrainte de temps très forte | | | | | | | | | | | |
| 41 | 150 | 5 | 40.71 | 126.84 | 0.020 | Pas de solution | | | 40.71 | 126.84 | 0.020 | Pas de solution | | |
| 40.8 | 1500 | 5 | 40.71 | 126.84 | 0.020 | 40.71 | 245.24 | 0.020 | 40.71 | 126.84 | 0.020 | 40.71 | 245.24 | 0.027 |
| 50 | 150 | 5 | 41.77 | 91.71 | 0.016 | 41.77 | 91.71 | 0.016 | 41.77 | 91.71 | 0.016 | 41.77 | 91.71 | 0.016 |
| 50 | 150 | 0.015 | Pas de solution : contrainte de surface très forte | | | | | | | | | | | |
| 40.7 | 150 | 5 | Pas de solution : contrainte de temps très forte | | | | | | | | | | | |
| 40.8 | 150 | 5 | 40.71 | 126.84 | 0.020 | Pas de solution | | | 40.71 | 126.84 | 0.020 | Pas de solution | | |
| 40.8 | 250 | 5 | 40.71 | 126.84 | 0.020 | 40.71 | 245.24 | 0.027 | 40.71 | 126.84 | 0.020 | 40.71 | 245.24 | 0.027 |

Dans le tableau 3.2, on peut observer ce qui suit :
- dans la 1ère ligne, toutes les contraintes ont été satisfaites (il s'agit en fait de la meilleure solution qu'on puisse obtenir pour la consommation de puissance)
- dans la 2ème ligne, il n'y'a pas de solution à cause de la contrainte de puissance qui est très forte (la valeur minimale possible est 91.71 mW)
- dans la 3ème ligne, il n'y'a pas de solution à cause de la contrainte de surface qui est très forte (la valeur minimale possible est 0.016 mm2)
- dans la 5ème ligne, il n'y'a pas de solution à cause de la contrainte de temps qui est très forte (la valeur minimale possible est 40.71 ps)
- les combinaisons A, B, C et D ne donnent pas toujours les mêmes résultats : dans la 7ème ligne, la modification des dimensions pour satisfaire la contrainte de temps a commencé par celle des transistors – combinaisons A et C- ce qui a permis de

satisfaire la contrainte de temps sans faire aucun ou peu de changements de dimensions des interconnexions (initialisées à Lmin). Ceci a permis d'obtenir de meilleures valeurs de consommation de puissance qu'avec les combinaisons A et C

- dans la 11ème ligne, on remarque que les combinaisons A et C donnent une solution alors que les combinaisons B et D ne le permettent pas : ces dernières, pour satisfaire la contrainte de temps (très proche de la valeur minimale) ont transformé toutes les dimensions des interconnexions (initialisées à Lmin), ce qui a permis d'augmenter les valeurs des capacités Cload et donc la valeur de la consommation de puissance, dépassant la contrainte imposée. En revanche, les combinaisons A et C ont commencé à changer certaines dimensions des transistors, ce qui a eu pour effet de satisfaire la contrainte de temps sans aucun changement (ou peu de changements) sur les dimensions des interconnexions.

Enfin, notons que tous les résultats indiqués dans les 2 précédents tableaux ont été obtenus en un temps CPU très intéressant, égal à 0 seconde - la fonction système *clock()* - utilisée affiche le temps CPU en secondes.

### 1.3 Conclusion

Nous avons présenté dans ce chapitre les résultats obtenus en optant pour l'option –t (priorité donnée au temps) ou l'option –p (priorité donnée à la consommation de puissance). Pour chacune des 2 options, 4 combinaisons A, B, C et D ont été utilisées, selon le cas où les changements considèrent tout d'abord les dimensions des transistors ou celles interconnexions. Les tableaux 5.1 et 5.2 montrent clairement l'efficacité de ces combinaisons : A et C sont efficaces pour l'option –p alors que B et D le sont pour l'option –t.

Comme il a été mentionné dans les chapitres précédents, l'initialisation de la solution, le passage de la solution courante (ne satisfaisant pas une ou des contraintes) à une autre solution ne se font pas de manière aléatoire, mais plutôt de façon réfléchie. Ceci nous a permis d'obtenir des résultats de qualité attendue à partir des heuristiques conçues.

# CONCLUSION GENERALE

Nous avons essentiellement présenté dans ce mémoire deux heuristiques, chacune constituée de 4 variantes, en vue de dimensionner les transistors et les interconnexions d'une partie de contrôle d'un circuit. Ce travail nous a permis de mettre en pratique nos connaissances acquises au cours de notre cursus universitaire aussi bien du point de vue Software (optimisation multi-critères) que du point de vue Hardware (circuits digitaux). Les résultats montrés dans le chapitre précédent montrent clairement l'efficacité des heuristiques développées.

# Bibliographies

[1] Weste & Eshraghian "Principles of CMOS VLSI Design" Kluwer Academis Publishers

[2] Garey & Johnson "Computers and intractability : The theory of NP-completness" Editions Freeman

[3] A. Mahdoum "Critique du livre : Representations for genetic and evolutionary algorithms écrit par Franz Rothlauf, édité par Springer Verlag Editions" critique publiée dans The Computer, Vol.49 N°5, 2006, journal édité par l'université d'Oxford

[4] A. Mahdoum "SAFT : An efficient state assigment for finite state machine tool", INFORMATION, 2002, journal édité par l'université Hosei, Japon

[5] A. Mahdoum "SPOT : A tool for estimating the maximal switching power dissipation" SASIMI, 1-2 Déc.97, Osaka, Japon

[6] A. Mahdoum "SPOT : A tool for estimating the maximal and average switching power dissipation" Design Automation and Test Conference, Paris, 2002

[7] A. Mahdoum "Combined Heuristics for Synthesis of SoCs with Time and Energy Constraints" Computers and Electrical Engineering, ELSEVIER, Vol. 38, 2012, pp.1687-1702.

[8] Johann Dréo, Alain Pétrowski, Éric D Taillard, and Patrick Siarry. Avantpropos. In Métaheuristiques pour l'optimisation diffcile, pages 6_8. 2003.

[9] Agoston E Eiben and JE Smith. What is an evolutionary algorithm ? In Introduction to Evolutionary Computing, pages 15_35. Springer, 2003.

[10] Serap Ulusam Seçkiner, Yunus Ero§lu, Merve Emrullah, and Türkay Dereli. Ant colony optimization for continuous functions by using novel pheromone updating. Applied Mathematics and Computation, 219(9) :4163_4175, 2013.