République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département informatique.

Mémoire pour l'obtention
du diplôme d'ingénieur d'état en informatique.

Option : Système d'information

Sujet :

Transformation d'une solution continue en une solution discrète pour la CAO des circuits digitaux VLSI à faible consommation de puissance, au niveau transistor.

Présenté par : TOUBALINE Nesrine
TOUAHRI Dalila

Promoteur : MAHDOUM Ali

Organisme d'accueil : CDTA (laboratoire de microélectronique)

Soutenue le: Mardi 27 Septembre 2005, devant le jury composé de :

Mr BENNOUAR	**Président**
Mr OULD-AISSA	**Examinateur**
Mr HAMMOUDA	**Examinateur**

2004-2005

Remerciements

Tous d'abord nous remercions le bon Dieu qui nous a protégé et guidé.

Nous remercions nos parents et toute la famille qui nous ont beaucoup aidé et encouragé durant nos études.

On exprime particulièrement nos profondes gratitudes et nos vifs remerciements à notre promoteur "MAHDOUM Ali" pour son aide, sa patience et ses précieux conseils tout au long de l'élaboration de ce travail.

On adresse nos sincères remerciements aux personnels du CDTA pour leurs soutiens.

Nous tenons également à remercier tous les enseignants qui nous ont suivis durant nos études.

Table des matières :

Chapitre III: Détermination de la solution discrète a partir de la solution continue

Chapitre IV: Résultats

Dédicaces

Je dédie ce mémoire à toute ma famille, mes très chers parents, ma sœur et mon frère, à mes amis et camarades, à tous ceux qui m'ont aidé à le réaliser, et à toute la section de cinquième année 2004/2005.

Je dédie ce modeste travail :
A mes très chers parents
A mon frère Ahcène
A mon oncle Mohamed
A toute ma famille
A tous mes amis et camarades

Table des organigrammes :

Chapitre II: Estimation du délai critique dans un circuit

Table des algorithmes

Chapitre II: Estimation du délai critique dans un circuit

Chapitre III: Détermination de la solution discrète a partir de la solution continue

Table des tableaux

Chapitre IV: Résultats

Table des figures

Chapitre I : Définitions et rappels

Chapitre II: Estimation du délai critique dans un circuit

Chapitre III: Détermination de la solution discrète a partir de la solution continue

Résumé

La consommation de la puissance est un paramètre déterminant dans la conception des circuits intégrés. De ce fait, cet aspect est abordé à chaque niveau d'abstraction, et dans notre cas, le problème est traité au niveau *transistor*.

En fait, il s'agit plus précisément de concevoir des circuits digitaux à faible consommation de puissance. Aussi, le problème consiste à minimiser la consommation totale de sorte que la contrainte portant sur le temps de réponse du circuit soit satisfaite. Le problème est alors formulé par une fonction objective à minimiser sous la contrainte *temps de réponse*. La solution continue générée donne alors les différentes valeurs des tensions d'alimentation à affecter aux différentes portes logiques du circuit, ainsi que celles des tensions de seuil des différents transistors. Cette solution ne peut malheureusement pas être réalisable car elle rendrait la fabrication du circuit très difficile et très coûteuse de par les nombreuses valeurs de tensions d'alimentation et de seuil obtenues. Aussi, la communauté scientifique travaillant dans ce domaine se limite généralement à deux bornes pour les tensions d'alimentation, et à deux bornes également pour les tensions de seuil de chaque type de transistor.

Le problème supplémentaire consiste alors à déterminer la solution *discrète* à partir de la solution *continue* obtenue. Nous verrons dans ce mémoire que cette *conversion de solution* n'est pas un problème polynomial, et nécessite de ce fait une heuristique adéquate. C'est ce dernier point que traite notre travail et qui sera développé le long de ce mémoire.

Introduction Générale

Avec la proéminence croissante des systèmes portables (PDAs, téléphones portables, PCs portables, etc ...) et la durée de vie limitée des batteries, la conception de circuits à faible consommation de puissance est devenue une nécessité. Bien que des efforts aient été faits pour augmenter la durée de vie des batteries (remplacement de batteries NiCd par des batteries NiMH), une autonomie intéressante n'est pas encore atteinte. De plus, les technologies actuelles permettent une intégration de millions de transistors sur une puce (implémentations SOC). Aussi, même si le système VLSI est alimenté par le secteur, une conception classique de ce système entraînerait une forte consommation de puissance, et de là, augmenterait la température de fonctionnement du système, ce qui engendrerait un disfonctionnement de celui-ci. De ce fait, l'aspect dissipation de puissance est abordé à chaque niveau d'abstraction soit pour proposer des méthodes aussi différentes que variées pour estimer ce paramètre, soit pour concevoir des circuits à faible consommation de puissance. Dans le cadre de ce travail, il s'agit de considérer le problème au niveau d'abstraction *transistor*.

La consommation de puissance d'un circuit est due principalement aux courts circuits, à l'activité dynamique du circuit, et aux fuites du courant dans les transistors. La consommation due aux courts circuits peut être réduite grâce à l'utilisation d'une technologie appropriée (exemple, technologie CMOS) et en réduisant les temps de transition des signaux d'entrée, ce qui aura pour effet de diminuer considérablement la durée où les transistors N et P conduisent simultanément. La consommation de la puissance dynamique peut être réduite en optant pour une faible valeur de l'alimentation du circuit, mais ce serait au détriment de la rapidité du circuit. Enfin, les fuites de courant peuvent être réduites en utilisant des transistors ayant des tensions de seuil assez élevées, mais ce serait aussi au détriment de la rapidité du circuit. La problématique consiste alors à réduire la consommation de puissance du circuit, tout en assurant à celui-ci une certaine performance en matière de rapidité. Aussi, le problème est formulé par une fonction objective (portant sur la consommation de puissance) à minimiser sous la contrainte *temps de réponse du circuit*. Le problème d'optimisation combinatoire est alors résolu par une méthode adéquate, générant ainsi une solution *continue* qui consiste à affecter des tensions d'alimentation aux

différentes portes logiques, ainsi que des tensions de seuil pour les différents transistors. Toutefois, cette solution n'est malheureusement pas réalisable dans la mesure où elle rendrait la fabrication du circuit très difficile et très coûteuse. De ce fait, seules deux bornes pour chacune des tensions d'alimentation et de seuil des transistors sont utilisées pour trouver le meilleur compromis possible entre la consommation de puissance du circuit et sa rapidité. Nous verrons que cette transformation de solution *continue* en solution *discrète* n'est pas un problème polynomial, et nécessite alors une heuristique appropriée. Notre travail porte précisément sur le développement de cette heuristique qui nécessite au préalable la réalisation de certaines tâches dont la détermination des temps de réponse des différentes portes logiques du circuit, le temps critique du circuit, etc....

Le présent mémoire est organisé comme suit. Au premier chapitre, nous donnerons les définitions et ferons quelques rappels qui seront nécessaires pour une meilleure compréhension du mémoire. Au deuxième chapitre, nous présenterons notre technique de détermination des temps de réponse des portes et celle du temps critique du circuit. Suivra alors le troisième chapitre qui traitera de l'heuristique proprement dite. Les résultats seront présentés au quatrième chapitre, et nous terminerons par une conclusion.

CHAPITRE I

Définitions et rappels

I.1. Introduction :

Avant d'aborder les chapitres suivants qui constituent l'essentiel de notre travail, nous aimerions apporter dans le présent chapitre quelques définitions et rappels que nous jugeons utiles pour une meilleure compréhension de la description de notre travail. Ces définitions et rappels ont un rapport avec tous les éléments qui interviennent dans l'aspect *consommation de puissance*.

I.2. Définition d'un circuit :

Un circuit intégré est un composant qui renferme (où "intègre") dans un unique petit boîtier, un nombre important de composants, notamment des transistors. Un processeur, par exemple, en intègre plusieurs millions.

I.3. Définition d'un transistor :

Un transistor est un composant électronique à semi-conducteur utilisé pour contrôler le passage d'un courant électrique. Il remplit deux fonctions vitales en électronique: celles d'amplificateur et de commutateur. Certains transistors sont spécialisés dans l'une ou l'autre de ces fonctions, d'autres sont aptes à les remplir toutes les deux.

I.4. Familles technologiques :

L'évolution rapide des technologies de fabrication a engendré plusieurs générations de circuits intégrés, appartenant à des "familles" différentes.

Plus récente, la famille CMOS (Complementary Metal-Oxide-Semiconductor) dont les propriétés la destinent à être une technologie très utilisée dans le développement des systèmes à très haute échelle d'intégration (VLSI). Parmi ces propriétés, on distingue particulièrement une basse consommation de puissance par rapport à d'autres technologies.

A titre d'illustration, nous présenterons les portes logiques suivantes :

-L'inverseur en technologie nMOS

- L'inverseur en technologie pseudo-nMOS

- Les portes de base en technologie CMOS qui sont successivement un inverseur, un NAND à deux entrées, un NOR à deux entrées, un AND à deux entrées et un OR à deux entrées.

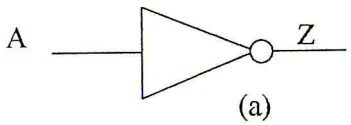
- La porte de transmission en technologie CMOS.

- Les portes complexes en technologie CMOS.

Notons que pour toutes les portes, chaque entrée est connectée à la grille d'un transistor à canal N et à la grille d'un transistor à canal P.

I.5. Inverseur:

* Représentation symbolique :

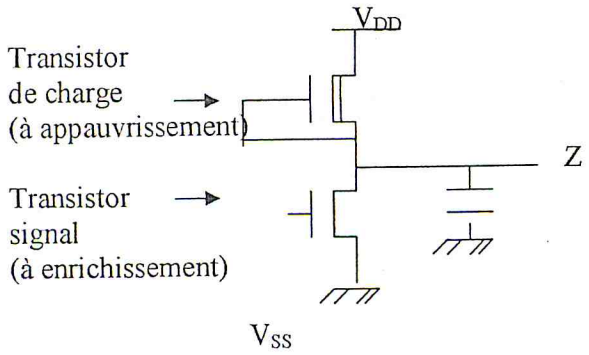


(a)

* Table de vérité :

A	Z
0	1
1	0

* Représentation structurelle (en technologie nMOS) :



(b)

Fig.I.1. Inverseur (a) : représentation symbolique (b) : représentation structurelle NMOS

- Représentation structurelle (en technologie pseudo-nMOS) :

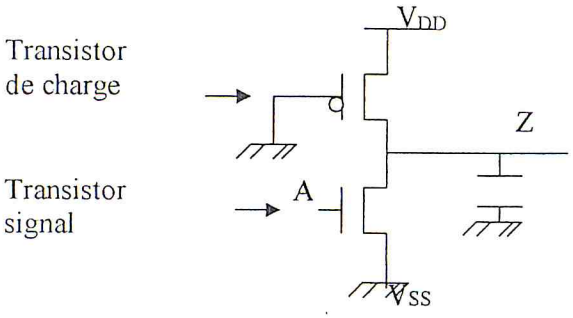


Fig.I.2. Inverseur pseudo-NMOS

* Représentation structurelle (en technologie CMOS) :

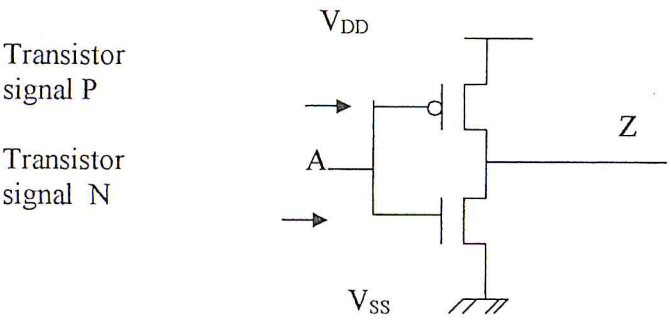
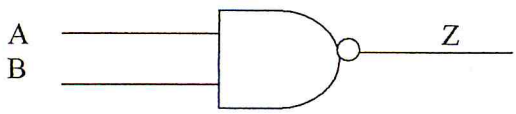


Fig.I.3. Inverseur CMOS

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

I.6.Porte NAND à deux entrées :

* Représentation symbolique :

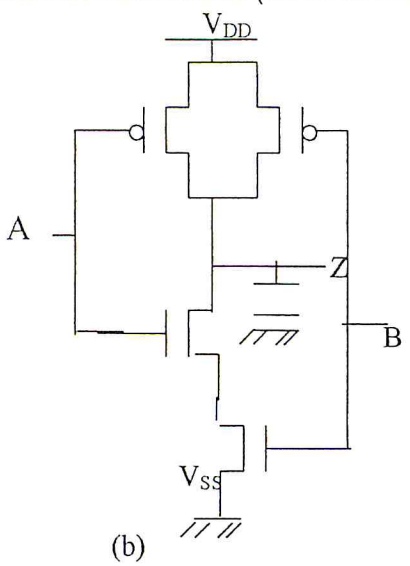


(a)

* Table de vérité :

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

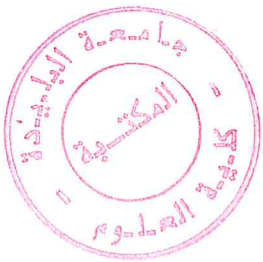
- Représentation structurelle (en technologie CMOS) :



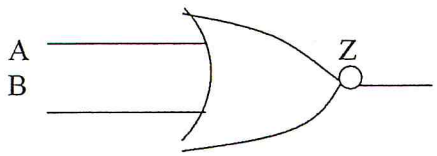
(b)

Fig.I.4. Porte NAND à deux entrées (a) : représentation symbolique (b) : représentation structurelle

I.7. Porte NOR à deux entrées :



* Représentation symbolique :



(a)

* Table de vérité :

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

* Représentation structurelle (en technologie CMOS):

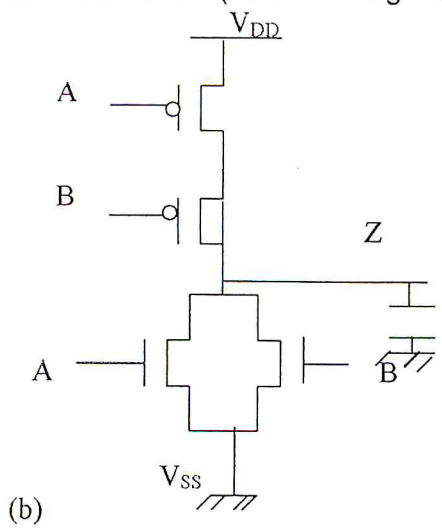
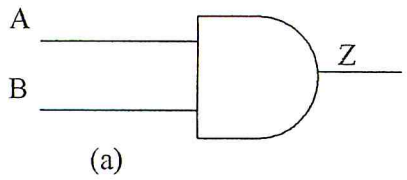


Fig.I.5. Porte NOR à deux entrées (a) : représentation symbolique (b) : représentation structurelle

I.8.Porte AND à deux entrées :

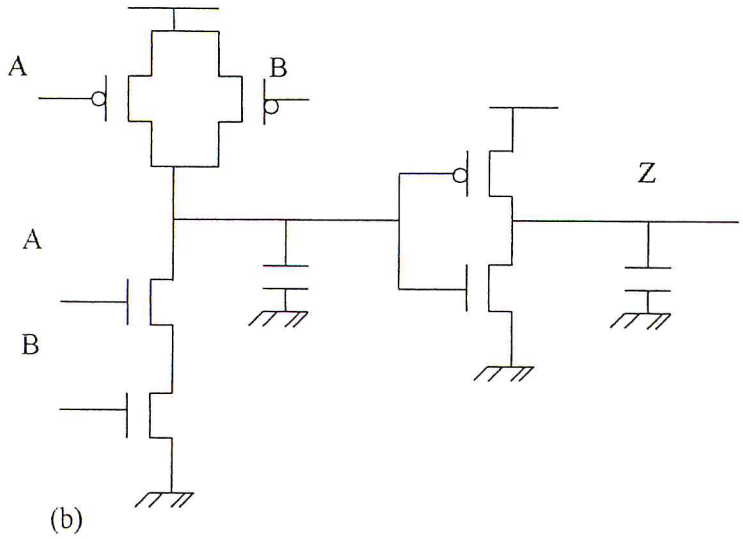
* Représentation symbolique :



* Table de vérité:

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

* Représentation structurelle (en technologie CMOS):

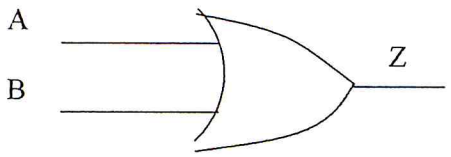


(b)

Fig.I.6. Porte AND à deux entrées (a) : représentation symbolique (b) : représentation structurelle

I.9.Porte OR à deux entrées :

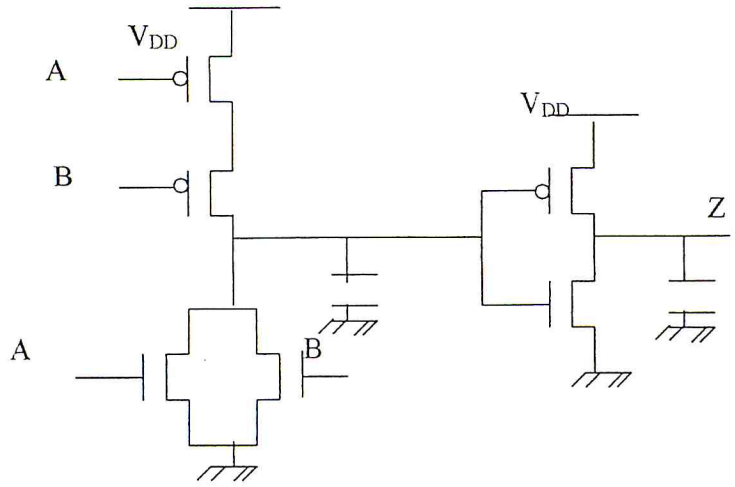
* Représentation symbolique :



(a)

* Table de vérité:

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



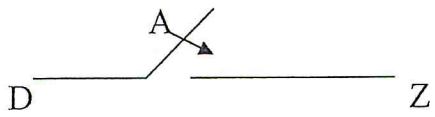
(b)

Fig.I.7. Porte OR à deux entrées (a) : représentation symbolique (b) : représentation structurelle CMOS

I.10.Porte de transmission :

* Représentation symbolique :

(a)



* Tables de vérité :

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

A	Z
0	X
1	D

* Représentation structurelle (en technologie CMOS) :

(b)

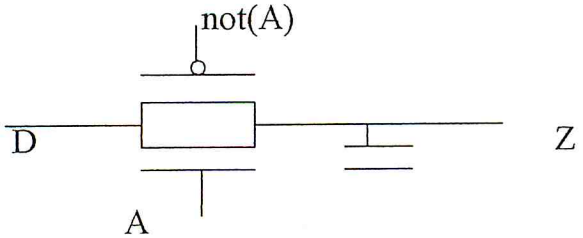


Fig.I.8. Porte de transmission (a) : représentation symbolique (b) : représentation structurelle CMOS

I.11.Les portes complexes :

On appelle porte complexe une porte nécessitant à la fois des symboles ET et des symboles OU pour leur dessin au niveau logique. Pour réaliser une fonction plus complexe, en logique complémentaire, un circuit est constitué de deux réseaux duaux : Un réseau N, constitué exclusivement de transistors NMOS, branché entre la sortie et le "moins de l'alimentation" (en général la masse) qui correspond au "0" logique, et un réseau P, constitué exclusivement de transistors PMOS, branché entre la sortie et le "plus de l'alimentation" (V_{DD}) qui correspond au "1" logique

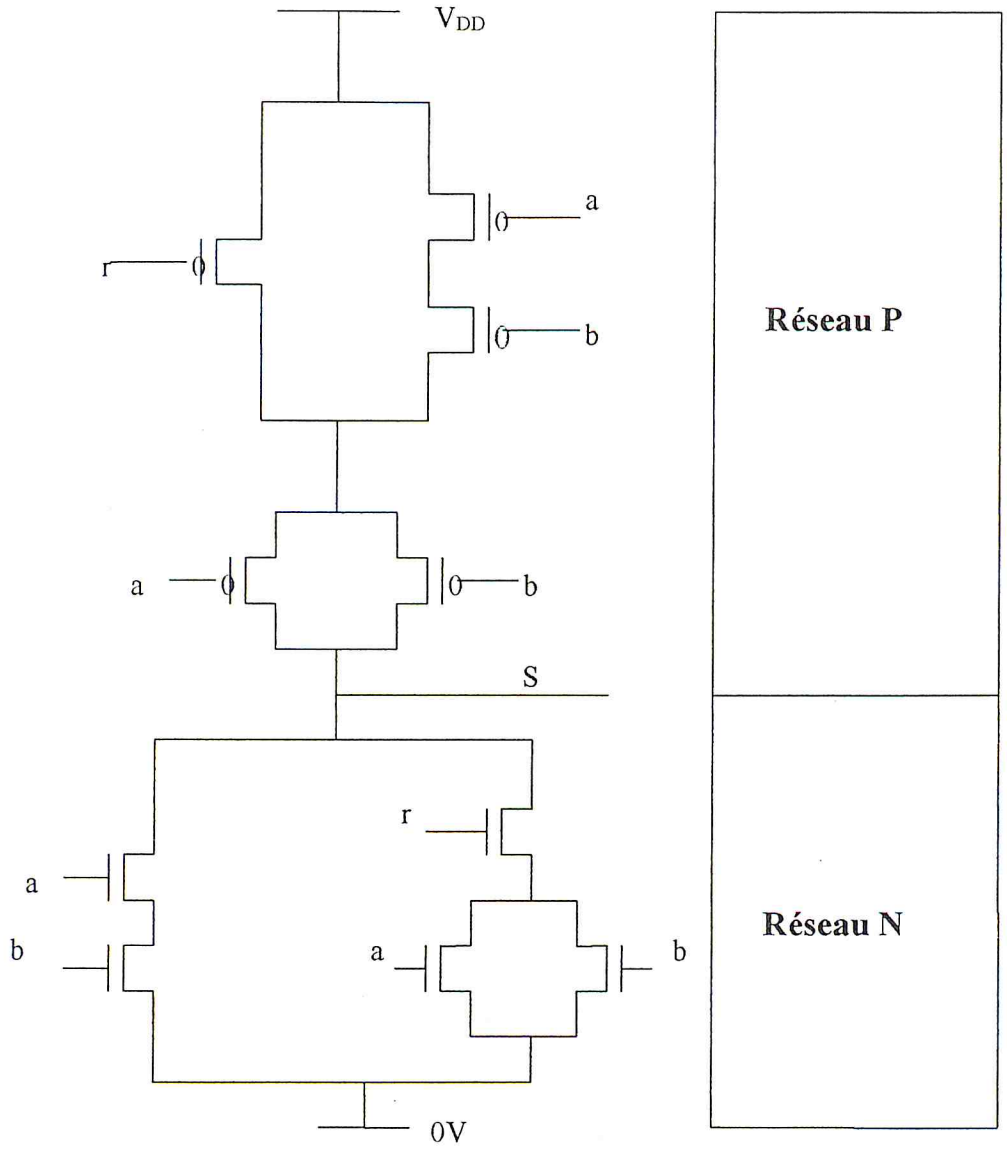


Fig.I.9. : une porte complexe S= not((a and b) or (r and (a or b)))

I.12.Nœuds de transistors:

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

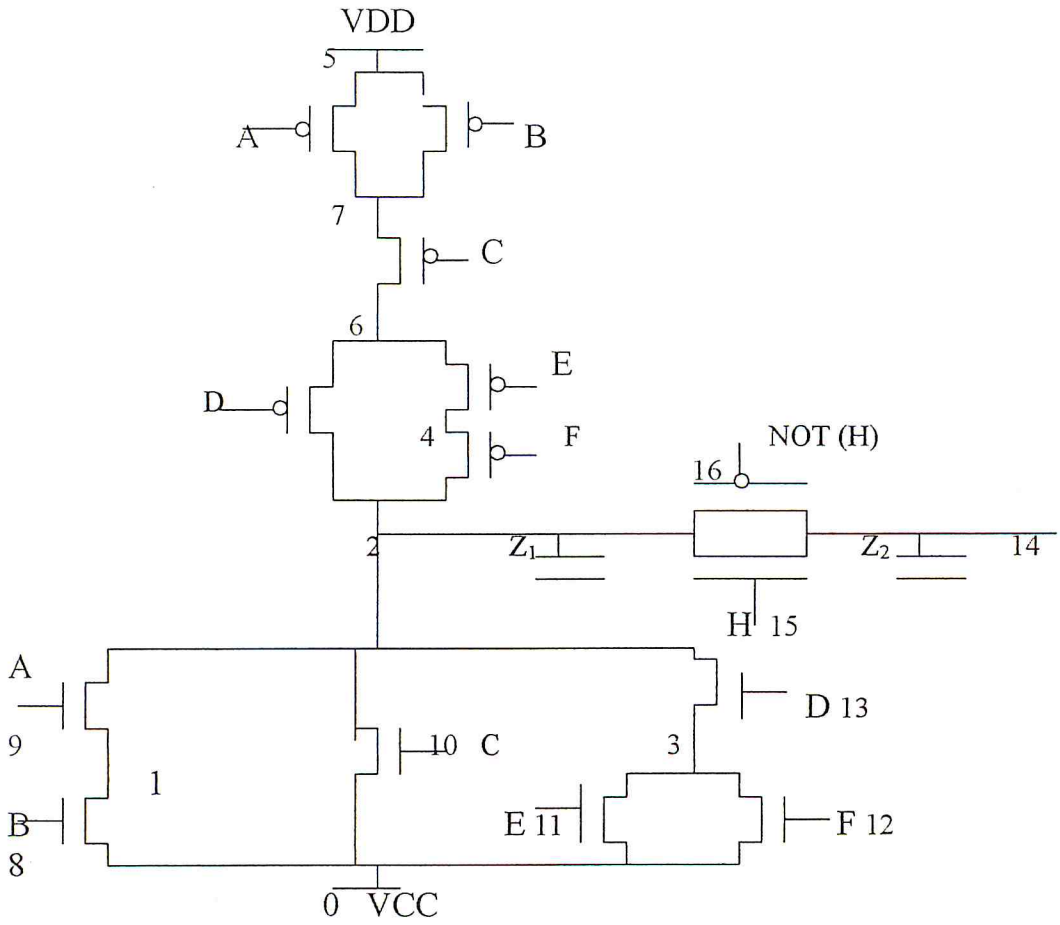


Fig.I.10. Noeuds des transistors d'un circuit CMOS

* Netlist:

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

4	12	2	5	P
7	10	6	5	P
5	8	7	5	P
5	9	7	5	P
2	15	14	0	N
2	16	14	5	P

I.13.Tension de seuil d'un transistor :

La tension de seuil V_{th} d'un transistor est la tension à partir de laquelle le transistor commence à conduire. Plus précisément, il y'a conduction pour un transistor MOS lorsque $|V_{GS}| > |V_{th}|$.

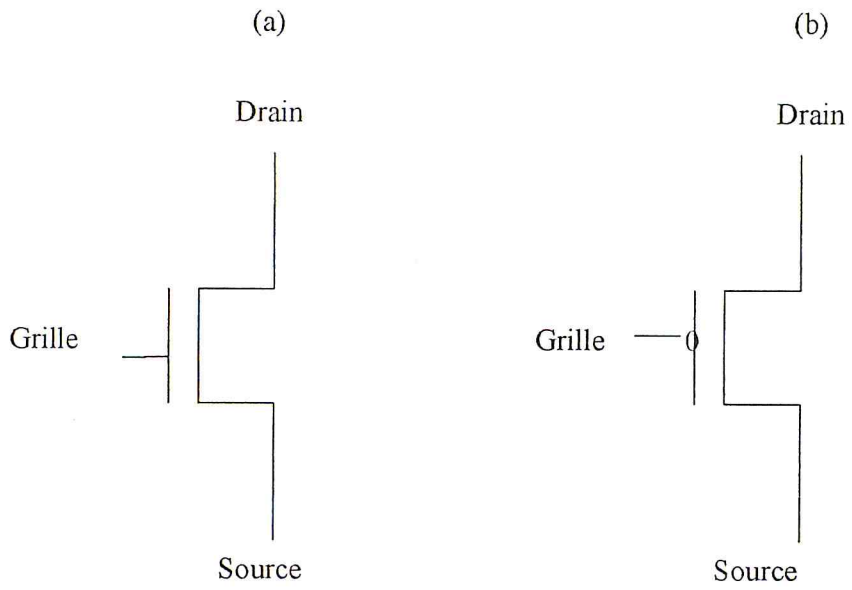


Fig.I.11. : les transistors : (a) NMOS et (b) PMOS

Le transistor MOS à canal N :

V_{TN} est la tension de seuil du transistor NMOS

Si $V_{GS} > V_{TN}$ le canal existe, le NMOS est passant.

Si $V_{GS} \leq V_{TN}$ le canal n'existe pas, le NMOS est bloqué.

Le transistor MOS à canal P :

V_{TP} est la tension de seuil de PMOS

Si $V_{GS} < V_{TP}$ le canal existe, le PMOS est passant.

Si $V_{GS} \geq V_{TP}$ le canal n'existe pas, le PMOS est bloqué

Notons que V_{GS} est la tension entre la grille et la source du transistor considéré.

I.14.Consommation de la puissance :

Il existe trois sources majeures de consommation dans les circuits numériques CMOS qui sont décrites dans l'équation (I.1). Le premier terme représente la composante due aux charges ou décharges des capacités du circuit, le deuxième terme est dû au courant du court-circuit et le dernier aux courants de fuite.

$$P = P_{dynamique} + P_{court-circuit} + P_{fuite} \qquad (I.1)$$

I.14.1.Courant de court circuit:

Dans le cas de l'inverseur CMOS, une transition montante (ou descendante) à son entrée entraîne la commutation simultanée des deux transistors (les transistors PMOS et le NMOS sont passants) sur un court intervalle de temps, ce qui crée un chemin direct entre l'alimentation et la masse.

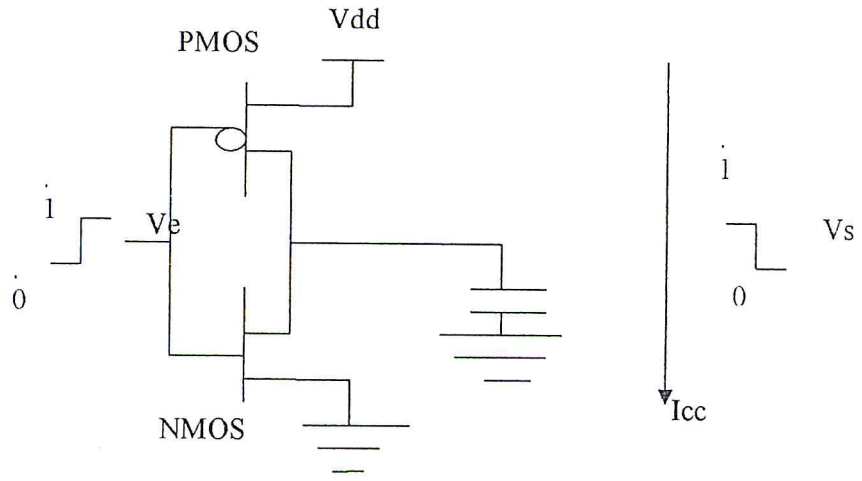


Fig.I.12: courant de court circuit

Vdd : tension d'alimentation
Ve : tension d'entrée
Vs : tension de sortie
Icc : courant de court circuit

I.14.2.Activité dynamique :

La puissance dynamique est causée par la charge ou la décharge de la capacité C_l. Elle est déterminée par :

$$P_{sw} = 0.5 * V_{DD}^2 * f * \sum_{i=1}^{\#portes} (C_{gi} * N_{gi}) \qquad (I.2)$$

où : - P_{sw} est la puissance dynamique du circuit
- V_{DD} est la tension d'alimentation
- f est la fréquence du circuit
- C_{gi} est la capacité de charge de la porte i
- N_{gi} est le nombre de transitions de la sortie de la porte i (nombre de charges et décharges de C_{gi})

I.14.3.Fuites du courant :

Les causes de l'augmentation des courants de fuite sont essentiellement dues à la réduction des tentions de seuil des transistors. En effet, du fait qu'un transistor conduit lorsque $|V_{GS}| > |V_{th}|$ et comme pratiquement V_{GS} n'est pas nul, il est possible que $|V_{GS}|$ reste supérieur à de très petites valeurs de $|V_{th}|$ (ce qui est le cas dans les technologies actuelles), et donc le transistor continue à conduire. La consommation de puissance due aux fuites de courant est donnée par l'expression suivante :

$$P_{leak} = 0.28125 * V_{DD} * K * Nb_{trans} * W_{avg} * L * 10^{-vt/\alpha v} \qquad (I.3)$$

où :
- P_{leak} est la consommation due aux fuites du courant.
- V_{DD} est la tension d'alimentation.
- K= I0 µA/µm.
- Nb_{trans} est le nombre de transistors dans le circuit.
- W_{avg} est la largeur moyenne des transistors.
- L est la longueur du canal (en µm) du transistor dans la technologie considérée.
- Vt , est la tension de seuil initiale du transistor.
- α =0.095 V.

I.15.Conclusion :

Dans ce chapitre, nous avons présenté des définitions et fait des rappels pour comprendre les principaux éléments qui interviennent en matière de consommation de puissance dans les circuits intégrés, et pour une meilleure compréhension des chapitres suivants qui constituent l'essentiel de notre travail.

CHAPITRE II

Estimation du délai critique dans un circuit

II.1.Introduction :

Comme indiqué dans l'introduction générale, le chapitre qui suivra celui-ci traitera de l'heuristique permettant de transformer la solution continue en une solution discrète. Cette dernière consiste à affecter des tensions d'alimentation aux portes logiques ainsi que des tensions de seuil aux transistors du circuit. Nous verrons dans ce chapitre que le temps de réponse d'un circuit dépend justement de ces tensions d'alimentation et de seuil. Par conséquent, le temps de réponse d'un circuit dépend de cette affectation de tensions. Comme il s'agit de minimiser la consommation de la puissance sous la contrainte du temps de réponse d'un circuit, il est nécessaire de déterminer le temps de réponse du circuit (qui est aussi le temps critique) à partir de ces tensions pour décider si une affectation de tensions est valable, c'est-à-dire si le temps de réponse induit du circuit est inférieur ou égal à celui désiré. De ce fait, nous discuterons dans ce chapitre des techniques développées pour la détermination du délai de chaque porte logique, ainsi que celle du chemin critique du circuit.

II.2. Détermination du délai d'une porte

Une méthode simple et efficace pour déterminer le délai de chaque porte consiste à trouver, à partir d'un modèle basé sur les résistances et les capacités, un rapport exprimant la rapidité ou la lenteur de cette porte par rapport à un inverseur de base. A partir du délai de l'inverseur et de ce rapport, le délai de la porte sera totalement défini.

II.2.1 Délai de l'inverseur

Notons le délai d'un inverseur par T_{inv} . Ce délai peut être exprimé en fonction des tensions de seuil du transistor (N ou P), ou en fonction du modèle basé sur les résistances et les capacités [1].

$$T_{invN} = \frac{C_L}{\beta_n * (V_{DD} - V_{tn})} \left[\frac{2 * V_{tn}}{V_{DD} - V_{tn}} + \ln \frac{4 * (V_{DD} - V_{tn})}{V_{DD}} - 1 \right] \quad \text{(II.1)}$$

où :

T_{invN} est le temps de décharge de la capacité de charge de l'inverseur

C_L est la capacité de charge de l'inverseur

$\beta_n = \mu_N * C_{ox} * W / L$

μ_N est la mobilité des électrons

C_{ox} est la capacité de l'oxyde

L est la longueur du transistor N de l'inverseur

W est la largeur du transistor N de l'inverseur

V_{DD} est la tension d'alimentation

V_{tn} est la tension de seuil du transistor N

$$T_{invP} = \frac{C_L}{\beta_p * (V_{DD} + V_{tp})} \left[\frac{-2 * V_{tp}}{V_{DD} + V_{tp}} + \ln \frac{4 * (V_{DD} + V_{tp})}{V_{DD}} - 1 \right] \quad \text{(II.2)}$$

où :

T_{invP} est le temps de charge de la capacité de charge de l'inverseur

β_p = μ_P * C_{ox} * W / L

μ_P est la mobilité des trous (charges positives dans le canal du transistor P)

V_{tp} (< 0) est la tension de seuil du transistor P

Les autres paramètres sont tels que définis précédemment.

Le temps de décharge de l'inverseur est proportionnel à T_{invRCN}

où:

$$T_{invRCN} = R_n * (C_D + C_L) \quad \text{(II.3)}$$

R_n est la résistance modélisant le transistor N

C_D est la capacité modélisant le transistor N

C_L est la capacité de charge de l'inverseur

De même, le temps de charge de l'inverseur est proportionnel à T_{invRCP}

où :

$$T_{invRCP} = R_p * (C_D + C_L) \quad \text{(II.4)}$$

R_p est la résistance modélisant le transistor P

C_D est la capacité modélisant le transistor P

C_L est la capacité de charge de l'inverseur

II.2.2. Délai d'une porte quelconque :

Le délai d'une porte est le cas le plus défavorable parmi les temps de charge (*rise time*) et de décharge (*fall time*) .

Nous avons :

$$T_r = K_{r1} * R_p * (K_{r2} * C_D + C_L) \quad \text{(II.5)}$$

$$T_f = K_{f1} * R_n * (K_{f2} * C_D + C_L) \quad \text{(II.6)}$$

où :

K_{r1}, K_{r2}, K_{f1} et K_{f2} sont des constantes (qui seront déterminées dans le paragraphe suivant)

On calcule alors le rapport (Rapport) entre la proportionnalité du délai de la porte considérée et celle du délai de l'inverseur.

Supposons que $T_r > T_f$. Alors,

$$\text{Rapport} = \frac{T_r}{T_{invRCP}} = \frac{K_{r1} * R_p * (K_{r2} * C_D + C_L)}{R_p * (C_D + C_L)} = \frac{K_{r1} * (K_{r2} * C_D + C_L)}{(C_D + C_L)} \quad \text{(II.7)}$$

$\Rightarrow T_{Porte}$ = Rapport * T_{invP} , c'est à dire :

$$T_{Porte} = \frac{K_{r1} * (K_{r2} * C_D + C_L)}{(C_D + C_L)} \left[\frac{C_L}{\beta_p * (V_{DD} + V_{tp})} \left[\frac{-2 * V_{tp}}{V_{DD} + V_{tp}} + \ln \frac{4 * (V_{DD} + V_{tp})}{V_{DD}} - 1 \right] \right] \quad \text{(II.8)}$$

Si $T_r \leq T_f$, alors

$$T_{Porte} = \frac{K_{f1} * (K_{f2} * C_D + C_L)}{(C_D + C_L)} \left[\frac{C_L}{\beta_n * (V_{DD} - V_{tn})} \left[\frac{2 * V_{tn}}{V_{DD} - V_{tn}} + \ln \frac{4 * (V_{DD} - V_{tn})}{V_{DD}} - 1 \right] \right] \quad \text{(II.9)}$$

De ce fait, il suffit de déterminer les quatre constantes pour pouvoir estimer le délai d'une porte logique à partir de l'équation (II.8) ou (II.9). Enfin, les délais et chemins critiques du circuit sont calculés à partir des délais des portes logiques constituant le circuit. Le schéma synoptique de ce traitement est illustré par la figure II.1. Notons que les détails des procédures, des structures de données et des enregistrements des fichiers seront donnés au fur et à mesure que nous progresserons dans ce chapitre.

II.2.3. Calcul des constantes :

Pour calculer K_{r1}, K_{r2}, K_{f1} et K_{f2}, nous avons besoin de connaître le comportement de la porte (l'expression logique).

K_{r1} est le nombre de transistors de type P dans le chemin le plus défavorable pour la charge de la capacité de charge

K_{r2} est le nombre de capacités dans le chemin le plus défavorable pour la charge de la capacité de charge

K_{f1} est le nombre de transistors de type N dans le chemin le plus défavorable pour la décharge de la capacité de charge

K_{f2} est le nombre de capacités dans le chemin le plus défavorable pour la décharge de la capacité de charge

Remarque: Pour le calcul des quatre constantes, nous considérons que :

1) le nombre de transistors est le nombre de variables dans l'expression logique. Si une variable se répète, on la comptabilise à nouveau (le nombre de transistors est le nombre d'opérandes).

2) un transistor a deux capacités à ses bornes. Quand il est en série avec un autre transistor, il partage avec lui une capacité.

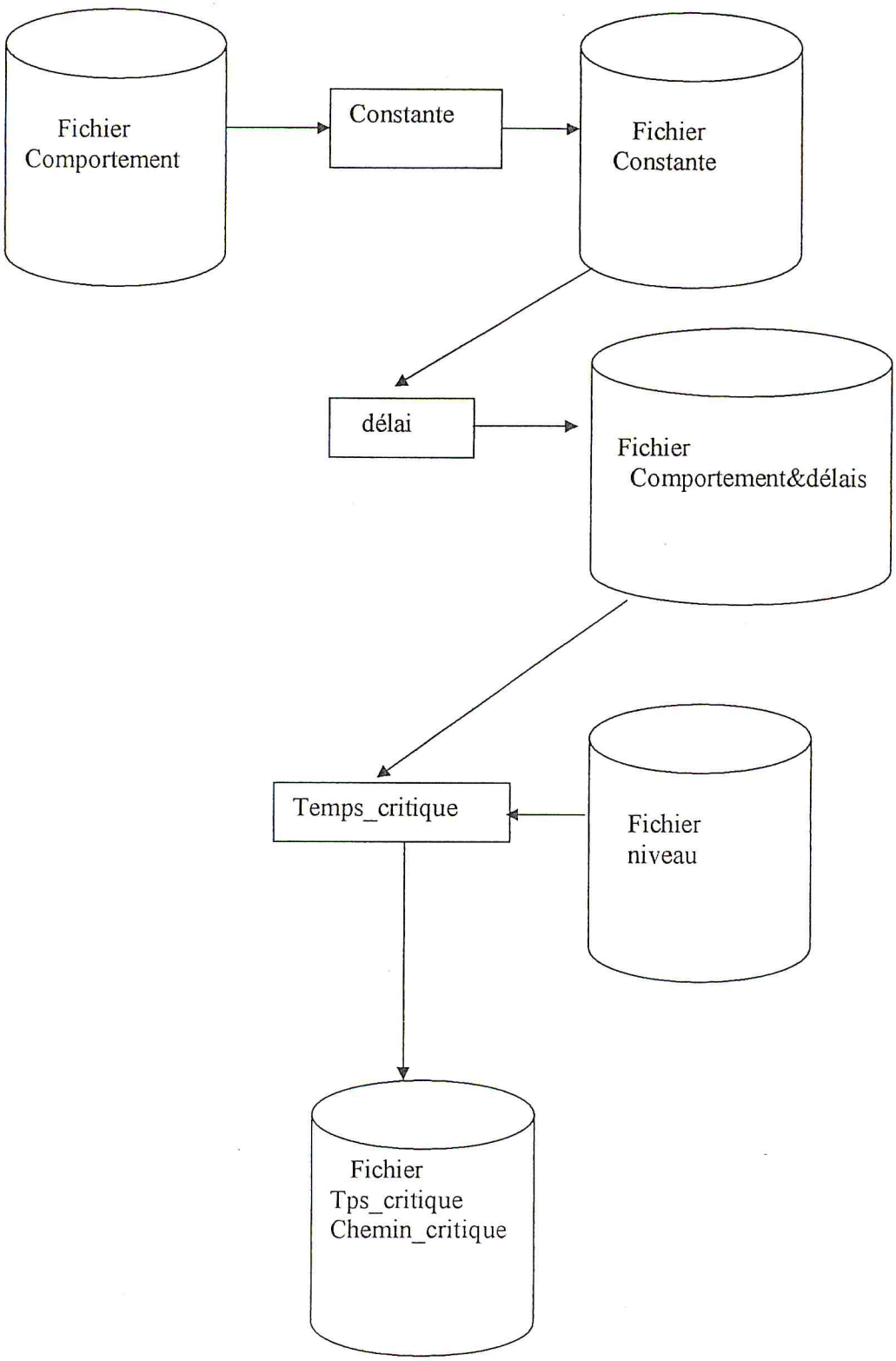


Fig II.1.: Organisation des modules développés pour le calcul des délais et chemins critiques

II.2.3.1. Algorithme général pour le calcul des constantes:

Algorithme II.1 Algorithme général pour le calcul des constantes

Cet algorithme est le suivant :

```
DEBUT
 Entrée : l'expression (comportement de la porte)
 Construction de l'arbre n-aire associé à l'expression (opérateur (père nœud)
  variable (feuille))
 calcul de Kf1          // kf1=poids :
 faire
   les variables ont un poids=1
   les opérateurs : AND (poids=somme des poids de ses fils)
                    OR  (poids = maximum des poids de ses fils)
   Kf1=poids de la racine ;
 fait
 calcul de kf2 // kf2=cap  :
 faire
   marquer le chemin qui a participé au calcul de Kf1.
   cap=0 ;
   Si on rencontre  un AND marqué (racine)
   alors si son fils gauche  est une variable
         alors cap=cap+1 ;
         fsi
   fsi
 // soit p un nœud et pf son fils
  pour tout fils de p
  faire si ( p est marqué  et  pf est une variable)
        alors si p == OR
              alors cap=cap+2 ;
              sinon si p == AND
                    alors si frère droit de pf est différent  de OR
                          alors cap=cap+1 ;
                           fsi
                    fsi
              fsi
       sinon si ((pf == AND ou pf == OR) et pf marqué)
              alors {i= calcul de Kf2(pf) ; // appel récursif
                    cap=cap+i ;
                    }
              sinon si (pf=AND ou OR et pf non marqué )
                    alors si pf == AND
                          alors {K=capacité_N (pf) ;
                                cap=cap+K ;
                                }
                           Fsi
                    Fsi
              Fsi
       fsi
  Fait
FIN
```

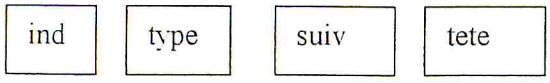
```
capacité_N (P2)
 DEBUT
   cap=0 ;
   si (p2 == AND)
   alors {Aller plus à droite (fd est le fils le plus à droite) ;
          Si (fd est une variable)
          Alors cap=cap+1 ;
          Sinon {i=capacité_N (fd) ;  // appel récursif
                 cap=cap+i;
                }
          fsi
         Aller plus à gauche (fg est le fils le plus à gauche) ;
         Si (fg est une variable)
         Alors cap=cap+1 ;
         Sinon {i=capacité_N (fg) ; // appel récursif
                cap=cap+i;
               }
         fsi
        }
    sinon  // p2 = OR
           parcourir tous les chemins ;
           pf fils de p2
           Si (pf est une variable)
           Alors cap=cap+1 ;
           Sinon {i=capacité_N(pf) ; // appel récursif
                  cap=cap+i;
                 }
           fsi
     fsi
     return cap ;
FIN
```

Notons que le calcul de Kr1 et Kr2 se fait de la même manière que Kf1 et Kf2 à la seule différence que l'arbre n_aire sera inversé (un nœud AND devient OR et vice-versa).

II.2.3.2. Structure de données, de fichiers et algorithmes :

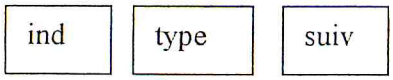
II.2.3.2.1.Structure de données

Type NOEUD1



int ind : indice de nœud
char type[5] : type de nœud (variable ou opérateur)
NOEUD1 *suiv : pointeur sur un nœud de type NOEUD1
NOEUD2 *tete : pointeur sur un nœud de type NOEUD2

Type NOEUD2

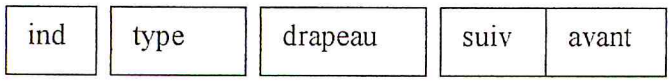


int ind: indice de nœud
char type[5] : type de nœud
NOEUD2 *suiv : pointeur sur un nœud de type NOEUD2

Type liste1

NOEUD1 *tete: pointeur sur le début de la liste des noeuds de type NOEUD1

Type NOEUD3

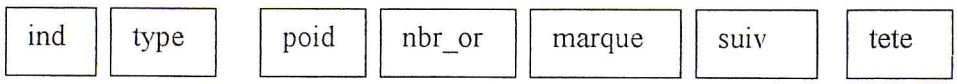


int ind : indice de nœud
char type[5] : type de nœud
char drapeau : un drapeau pour marquer les noeuds
NOEUD3 *suiv: pointeur sur un nœud de type NOEUD3
NOEUD3 *avant : pointeur sur un nœud de type NOEUD3

Type liste2

NOEUD3 *tete: pointeur sur le début de la liste des noeuds de type NOEUD3

Type NOEUD4



int ind :indice de nœud
char type[5] : type de nœud
int poid : utilisé pour calculer les valeurs des constantes
int nbr_or : nombre de or dans un chemin
char marque :(M ou N) utilisé pour marquer un chemin
NOEUD4 *suiv : pointeur sur un nœud de type NOEUD4
NOEUD4 *tete : pointeur sur un nœud de type NOEUD4

Type list3

tete

NOEUD4 *tete: pointeur sur le début de la liste des noeuds de type NOEUD4

II.2.3.2.2. Structure des fichiers :

Fichier comportement de la porte :

Entrées	%	Sortie	%	Comportement	%

Entrées : les entrées de la porte
Sortie : la sortie de la porte
Comportement : le comportement de la porte (l'expression logique)

Fichier des constantes :

n°porte	Kf1	Kf2	Kr1	Kr2

n°porte : numéro de la porte
Kf1 : nombre de résistances dans le chemin défavorable pour la décharge de la capacité de charge de la porte
Kf2 : nombre de capacités dans le chemin défavorable pour la décharge de la capacité de charge de la porte
Kr1 : nombre de résistances dans le chemin défavorable pour la charge de la capacité de charge de la porte
Kr2 : nombre de capacités dans le chemin défavorable pour la charge de la capacité de charge de la porte

Fichier des délais (à accès direct) :

delai

delai : le délai de la porte

Fichier comportement&délai de la porte:

Entrées	%	Sortie	%	Comportement	%	Capacité	%	delai	%

Capacité : c'est la capacité de la porte

II.2.3.2.3. Les fonctions du programme:

La fonction pile() :

A partir de l'expression logique (comportement de la porte), on détermine la notation postfixée en utilisant une pile.

Algorithme II.2 ://détermination de la notation postfixée

```
Char pile1[n][m] , pile2[n][m], ch[m] ;
Int nb_variable=0, indice1=0, indice2=0, Int fct
DEBUT
    Tant que (non fin d'expression)
    Faire {Lecture (ch) ;
           fct=generer_pile (ch) ;
           Si (nb_variable ==2 )
           Alors {indice2 -- ;//décrémenter l'indice de la pile 2
                  Si (indice2 >= 0)
                  Alors {pile1[indice1]=pile2[indice2] ; //déplacer le contenu de la pile2
                                                         // dans la pile1
                         indice1++ ;//incrémenter l'indice de la pile1
                         nb_variable- - ;
                        }
                  fsi
                 }
           fsi
          }
     Fait
FIN
```

La fonction generer_pile()

La fonction generer_pile() remplit deux piles selon le type du caractère lu.

Algorithme II.3://generation de la pile

```
DEBUT  char  AND, OR, (, );
      Si (ch == AND ou ch == OR)
      Alors {pile2[indice2]=ch ;
             indice2 ++;
            }
      Sinon si (ch == '(')
            alors nb_variable=0 ;
            sinon si (ch == ')')
                  alors nb_variable=2 ;
                  sinon //variable
                        {Pile1[indice1]=ch ;
                         indice1++ ;
                         nb_variable++ ;
                        }
                  fsi
             fsi
      fsi
      return 0 ;
FIN
```

Fonction arbre binaire

À partir de la notation postfixée, on construit un arbre binaire

Algorithme II.4:// construction de l'arbre binaire

```
DEBUT
   int indice=0 ;//pour parcourir la pile1 de 0 a indice1
   LISTE1 *p_list1 ;
   LISTE2 *p_list2 ;
   NOEUD1 *p ;
   NOEUD2 *q ;
   indice1-- ;
  // Créer les nœuds de liste1 d'abord (les pères)
  Tant que (indice <= indice1)
   Faire {Si (pile1[indice] == AND ou pile1[indice] == OR)
          Alors {Créer p1 de type NOEUD1;
                 p1->tete=0;
                 p1->ind=indice;
                 p1->type=pile1[ indice] ;
                 Appeler la fonction insérer  pour insérer p1 dans la liste p_list1
                }
          Fsi
          indice ++ ;
         }
  Fait
// créer les noeud2 de list1
p=p_list1->tete ;
indice= indice1 ;
Tant que (p)
Faire {Si (pile1[indice] != p->type)
        Alors {indice - - ;
               Tant que (pile1[indice] != p->type)
                Faire {Créer p3 de type NOEUD3 ;
                       p3->ind=indice ;
                       p3->drapeau='M';//marquer un fils gauche inséré dans p_list2
                       p3->type = pile1[indice] ;
                       Appeler la fonction insérer  pour insérer p3 dans p_list2
                       indice - - ;
                      }
                Fait
                Créer p3 de type NOEUD3 ;
                p3->ind=indice ;
                p3->drapeau='M';
                Appeler la fonction insérer  pour insérer p3 dans la liste p_list2
               }
        Fsi
        indice - - ;
        créer p2 de type NOEUD2 ; //fils droit
        p2->type=pile1 [indice] ;
        p2->ind =indice;
        Appel la fonction insérer  pour relier p2 à p
```

```
        si (pile1[indice] != AND ou pile1[indice] != OR )
        alors {indice - - ;
               créer p22 de type NOEUD2; //fils gauche
               p22->type=pile1 [indice] ;
               p22->ind =indice;
               Appeler la fonction insérer  pour relier p2 à p22
              }
       Sinon {indice - - ;
               créer p3 de type NOEUD3 ;  //fils droit
               p3->ind =p->ind;
               p3->drapeau ='G';// marquer un fils droit  insérer dans la p_list2
               p3->type=p->type ;
               Appeler la fonction insérer  pour insérer p3 dans la liste p_list2
              }
        fsi
        p=p->suiv ;
       }
 Fait
Tant que (indice > 0)
Faire {indice - -
        créer p3 de type NOEUD3 ;
         p3->ind =indice ;
         p3->drapeau ='M';
         p3->type=pile1[indice] ;
        Appeler la fonction insérer  pour insérer p3 dans la liste p_list2 }
 Fait
//compléter la liste1 avec les fils gauches
q=p_list2->tete ;
Tant que (q)
 Faire {
         Appeler la fonction completer_list1(q) ;
         q=q->suiv ;
        }
 Fait
FIN

La fonction completer_list1(NOEUD3 *q)
DEBUT
  NOEUD2 *qtr1
  NOEUD3  *qq
    Si (q->drapeau == 'M')
    Alors {Créer qtr2 de type NOEUD2 ;
           qtr2->type=q->type ;
           qtr2->ind=q->ind ;
           qq=q->avant ;
           tant que (qq->drapeau == 'G')
           faire qq=qq->avant ;
           fait
```

```
            qtr1=chercher(q->ind) ; // chercher un nœud d type NOEUD2 dans la
                                    // liste p_list1
            Appeler la fonction insérer pour relier qtr2 à qtr1
          }
    fsi
FIN
```

La fonction arbre n_aire()

```
// À partir de l'arbre binaire, construire l'arbre n_aire
```

Algorithme II.5://construction de l'arbre n_aire

```
DEBUT
  int i ;
  NOEUD1 *q ;
  q=p_list1->tete ;
  Tant que (q)
   Faire {
          Appeler la fonction modifier avec i=modifier (q) ;
          q=q->tete ;
         }
   Fait
FIN
```

La fonction modifier(NŒUD1 *y)

Modifie la liste p_list1 pour passer de l'arbre binaire à l'arbre_naire

```
DEBUT
 NOEUD1   *x, *x_avant ;
 NOEUD2 *p, *p2;
 int z ;
 p=y->tete ;
  Tant que (p)
   Faire si (y->type == p->type)
         alors { x=chercher(p->ind)   // chercher un nœud dans p_list1.
                 z= modifier (x) ;  // appel récursif
                 remplacer (p,x,y) ;
                 p2=p ;
                 p=p->suiv ;
                 free(p2)     //libérer l'espace mémoire réservé à p2
                 x_avant=chercher(x)  // chercher un nœud dans p_list1.
                 x-avant->suiv=x->suiv
                 free(x) ;
                }
          sinon p=p->suiv ;
          fsi
    Fait
    return 0 ;
FIN
```

La fonction remplacer (NOEUD2 *p, NOEUD1 *y, *x)

Remplacer un fils (opérateur) par ses fils s'il est du même type que son père.

```
DEBUT
        NOEUD2   *fils,*pm,*p_preced;
        fils =x->tete ;
        si (p == y->tete)
        alors {y->tete=fils ;
                fils=fils->suiv ;
                Tant que (fils)
                 Faire fils =fils->suiv ;
                 Fait
                 fils->suiv=p->suiv ;
                }
         sinon {p_preced=y->tete;
                 pm=p_preced->suiv;
                 Tant que (pm != p)
                 Faire {p_preced=pm ;
                         pm=pm->suiv ;
                        }
                 Fait
                p_preced->suiv=fils ;
               }
        fsi
FIN
```

La fonction arbre_naire2() :

Change la structure de représentation de l'arbre n_aire (les nœuds de l'arbre sont tous du même type)

Algorithme II.6:// Change la structure de représentation de l'arbre n_aire

```
DEBUT
        NOEUD1 *p1;   NOEUD4 *p4;
        p1=p_list1->tete;
        creer p_list3 de type list3
        p_list3->tete=0;
        créer p4 de type NOEUD4;
        p4->ind=p1->ind;
        p4->type=p1->type ;
        p4->suiv=0 ;
        p4->tete=0 ;
        p4->marque='M' (p4 est la racine) ;
        p-list3->tete =p4 ;
        remplir_list3() ;
FIN
```

La fonction remplir()

```
DEBUT
    NOEUD1  *pp ;
    NOEUD2  *p2 ;
    NOEUD4  *qq, *p_sauv ;
   pp=p_list1->tete ;
   Tant que(pp)
    Faire { Faisant appel à la fonction  chercher chercher_list3(pp->ind) pour chercher
un nœud de type NOEUD4 dans la liste list3;
           qq=x ;
           p2=pp->tete ;
          Tant que (p2)
           Faire {Creer p4 de type NOEUD4;
                  p4->ind=p2->ind;
                  p4->type=p2->type;
                  p4->suiv=0;
                  p4->tete=0 ;
                  p4->marque='N' ;
                  si( !qq->tete)
                  alors qq->tete=p4 ;
                  sinon {pp4=qq->tete ;
                         Tant que (pp4)
                          Faire {p_sauv=pp4 ;
                                 pp4=pp4->suiv ;
                                }
                         Fait
                         p_sauv->suiv=p4 ;
                        }
                 Fsi
          p2=p2->suiv ;
               }
          Fait
          pp=pp->suiv
         }
 Fait
FIN
```

La fonction calculer_poid(NOEUD4 *p) // Calculer kf1

Algorithme II.7: //calcul de Kf1

```
DEBUT
  NOEUD4  *pf;
   int y, poid ,somme;
   poid=0 ; Somme=0;
   pf=p->tete;
   Tant que(pf)
    Faire {Si(pf->tete == 0)    // il s'agit d'une variable
           Alors {Y=1;
                  Pf->nbr_or=0;   }
          Sinon  Y=calculer_poid(pf);  // appel récursif
```

```
            fsi
            pf->poid=y;
            si(p->type == AND)
            alors {poid=poid+y;
                   somme= somme+pf->nbr_or;
                   p->nbr_or=somme;
                  }
            Sinon si(p->type == OR)
                  Alors {si(y >= poid)
                         Alors {Poid=y;
                                somme=somme+pf->nbr_or;
                                p->nbr_or=somme;
                               }
                         fsi
                         somme=somme+1;
                        }
                  fsi
           fsi
           pf=pf->suiv;
          }
Return poid;
 FIN
```

La fonction capacité_F(NOEUD4 *p)

Algorithme II.8: //calcul de Kf2 pour le chemin marqué

```
DEBUT
  NOEUD4  *p1, *p2, *p3, *p4;
  int cap;
  int i, k;
 cap=0;
 p2=p->tete;
 p3=p2;
 Si(p->type == AND)
 Alors {p4=p->tete;
        Tant que(p4)
         Faire {p1=p4;
                p1=p4->suiv;
               }
         Fait
         Si(p1->type != AND ou p1->type != OR)
         Alors cap=cap+1 ;
         fsi
        }
 fsi
Tant que(p2)
Faire Si(p->marque == 'M' et p2->type != AND et p2->type != OR)
       Alors Si(p->type == OR)
              alors cap=cap+2 ;
             Sinon Si( p3 existe et p3->type != OR)
                    alors cap=cap+1 ;
```

```
                    fsi
                fsi
        Sinon Si( p2->type == AND ou p2->type == OR et p2->marque == 'M')
                Alors {i=capacité_F(p2) ;
                       cap=cap+i ;
                      }
                Sinon {Si ( p2->type == AND ou p2->type == OR et p2->marque == 'N')
                       Alors {k=capacité_N(p2) ;
                              cap=cap+k ;
                             }
                       fsi
                       p3=p2->suiv ;
                      }
                fsi
        fsi
 Fait
return cap ;
FIN
```

La fonction capacité_N(NOEUD4 *p2)

Algorithme II.9: //calcul de Kf2 pour le chemin non marqué

```
DEBUT
  NOEUD4 *pf1, *pf2, pfd, pfg;
   int i, cap;
   cap=0 ;
   Si(p2->type == AND)
   Alors { //aller à droite
           pfd=p2->tete ;
           si(pfd->type == AND et pfd != OR)
           alors cap=cap+1;
           sinon {i=capacité_N(pfd) ;
                  cap=cap+i ; }
           Fsi
           // aller à gauche
            pf1=p2->tete;
            tant que(pf1)
            faire {pfg=pf1 ;
                   pf1=pf1->suiv ;
                  }
            fait
             si(pfg->type != AND et pfg->type != OR)
             alors cap=cap+1 ;
             sinon {i=capacité_N(pfg) ;
                    cap=cap+i ;
                   }
            Fsi
          }

  Sinon {     // type = OR
         pf2=p2->tete ;
```

```
        Tant que(pf2)
         Faire {si(pf2->type != AND et pf2->type != OR)
                alors cap=cap+1 ;
                Sinon {i=capacité_N(pf2) ;
                       cap=cap+i ;
                      }
                Fsi
                pf2=pf2->suiv
               }
          Fait
 Fsi
 return cap ;
FIN
```

La fonction principale() // Fonction calculant les constantes

Algorithme II.10: // Programme principal pour le calcul des constantes

```
DEBUT
  NOEUD4 *p
  int indice1=0, indice2=0, nbr_or=0;
  int Kf1, Kf2, Kr1, Kr2, d1, fct1,fct2;
  Ouvrir les fichiers comportement et constante ;
  Tant que(non fin du fichier comportement)
  Faire {Pile() ;
         arbre_binaire() ;
         arbre_naire() ;
         arbre_naire 2() ;
         p=p_list3->tete ;
         Kf1=calcule_poid(p);
         p->poid=Kf1 ;
        marquer le chemin qui à participé au calcule de Kf1
         Kf2=capacité_F(p) ;
         Kr1=calculer_poid(p) ;
         p->poid=Kr1 ;
         p->marque='M' ;
         marquer le chemin qui à participé au calcule de Kr1
         Kr2=capaciter_F(p) ;
         écrire dans le fichier constantes ;
         // Remettre à 0 les variables modifiées durant les appels aux fonctions
         indice1=0 ; indice2=0 ; nb_variable=0 ;
        }
    Fait
FIN
```

II.2.3.3. Exemple d'illustration

Fichier de données :

Soit le fichier suivant décrivant le comportement d'une porte :

a b c d e % s % NOT ((a AND b) OR d OR (e AND (c OR d))) %

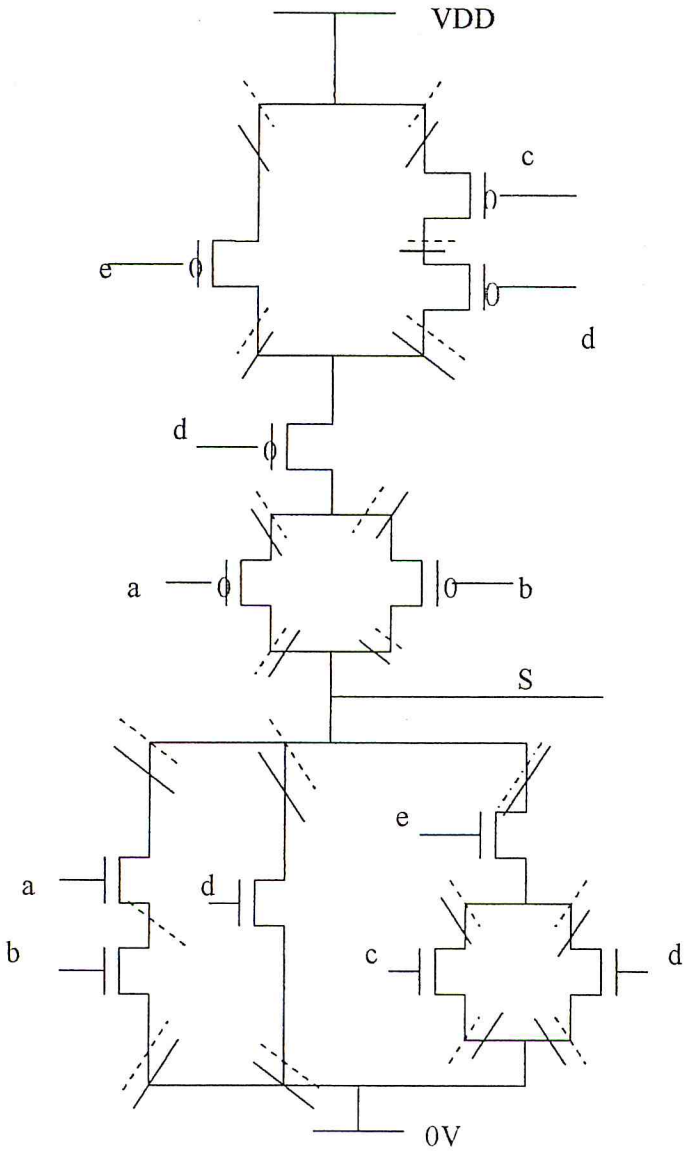


Fig II.2 Calcul des constantes pour la porte réalisant la fonction s=not((a and b)or d or (e and (c or d)))

Représentante les cas défavorables

Représentante tous les cas possibles

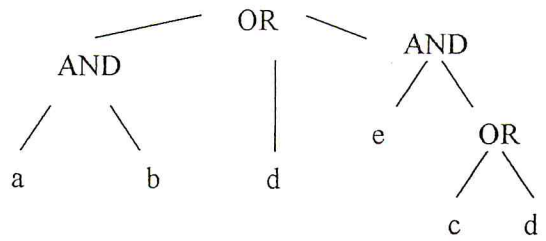


Fig II.3: l'arbre n_aire correspondant au réseau N

Fichier des résultats partiels (détermination des constantes) :

2 9 4 9

II.2.4. Algorithme général pour le calcul du délai d'une porte :

Algorithme II.11:// Algorithme général pour le calcul du délai d'une porte

DEBUT

// calcul du délai d'une porte

// les différentes variables sont telles que définies précédemment

si ($T_r > T_f$)

alors {Calculer T_{invRCP} ;

Calculer T_{invP} ;

Rapp= $\frac{T_r}{T_{invRCP}}$;

T_{Porte} =Rapp * T_{invP} ;

}

Sinon {Calculer T_{invRCN} ;

Calculer T_{invN} ;

Rapp= $\frac{T_f}{T_{invRCN}}$;

T_{Porte} =Rapp * T_{invN} ;

}

fsi

FIN

La fonction délai :

Algorithme II.12: //Algorithme de la fonction délai

DEBUT

Entrées : Fichier des constantes (Kr1, Kr2, Kf1, Kf2)

Les tensions V_{DD}, V_{tn}, V_{tp}

Les valeurs de C_D, C_L, β_p et β_n

$$T_r = K_{r1} * \left(\frac{4}{\beta_p * V_{DD}} \right) * (K_{r2} * C_D + C_L)$$

$$T_f = K_{f1} * \left(\frac{4}{\beta_n * V_{DD}} \right) * (K_{f2} * C_D + C_L)$$

si ($T_r > T_f$)

alors {

$$T_{invRCP} = R_p * (C_D + C_L)$$

$$T_{inv\,P} = \frac{C_L}{\beta_p * (V_{DD} + V_{tp})} \left[\frac{-2 * V_{tp}}{V_{DD} + V_{tp}} + \ln \frac{4 * (V_{DD} + V_{tp})}{V_{DD}} - 1 \right]$$

Rapp= $\frac{K_{r1} * (K_{r2} * C_D + C_L)}{(C_D + C_L)}$

$$T_{Porte} = \frac{K_{r1} * (K_{r2} * C_D + C_L)}{(C_D + C_L)} \left[\frac{C_L}{\beta_p * (V_{DD} + V_{tp})} \left[\frac{-2 * V_{tp}}{V_{DD} + V_{tp}} + \ln \frac{4 * (V_{DD} + V_{tp})}{V_{DD}} - 1 \right] \right]$$

}

Sinon {

$$T_{invRCN} = R_n * (C_D + C_L)$$

$$T_{inv\,N} = \frac{C_L}{\beta_n * (V_{DD} - V_{tn})} \left[\frac{2 * V_{tn}}{V_{DD} - V_{tn}} + \ln \frac{4 * (V_{DD} - V_{tn})}{V_{DD}} - 1 \right]$$

Rapp= $\frac{K_{f1} * (K_{f2} * C_D + C_L)}{(C_D + C_L)}$

$$T_{Porte} = \frac{K_{f1} * (K_{f2} * C_D + C_L)}{(C_D + C_L)} \left[\frac{C_L}{\beta_n * (V_{DD} - V_{tn})} \left[\frac{2 * V_{tn}}{V_{DD} - V_{tn}} + \ln \frac{4 * (V_{DD} - V_{tn})}{V_{DD}} - 1 \right] \right]$$

}

Fsi

Ecrire T_{Porte} dans le fichier delai ;

FIN

II.3. Détermination du délai critique d'un circuit intégré :

Un problème fondamental qui se pose fréquemment dans les applications est celui de la recherche de chemins de longueur minimale ou maximale. On va citer ci-dessous quelques algorithmes simples permettant d'obtenir de tels chemins.

II.3.1. Algorithmes de Ford et Bellman-Kalaba : [2]

II.3.1.1. Formulation du problème :

Soit G= (X, U) un l-graphe orienté sans boucle comportant n sommets. A tout arc $(X_i , X_j) \in U$ est associé un nombre réel ℓ_{ij} appelé longueur de l'arc (X_i, X_j). La longueur d'un chemin μ quelconque, notée $\ell(\mu)$, est alors définie comme étant la somme de longueurs des arcs qui le composent :

$$\ell(\mu) = \sum_{(X_i, X_j) \in \mu} \ell_{ij}$$

Un chemin joignant un sommet X_a à un sommet X_b est dit de longueur maximale s'il maximise cette longueur $\ell(\mu)$ dans l'ensemble de tous les chemins μ joignant X_a à X_b. La longueur d'un tel chemin est appelée distance maximale de X_a à X_b.

Il sera commode d'introduire une matrice L, dite matrice des longueurs, telle que :

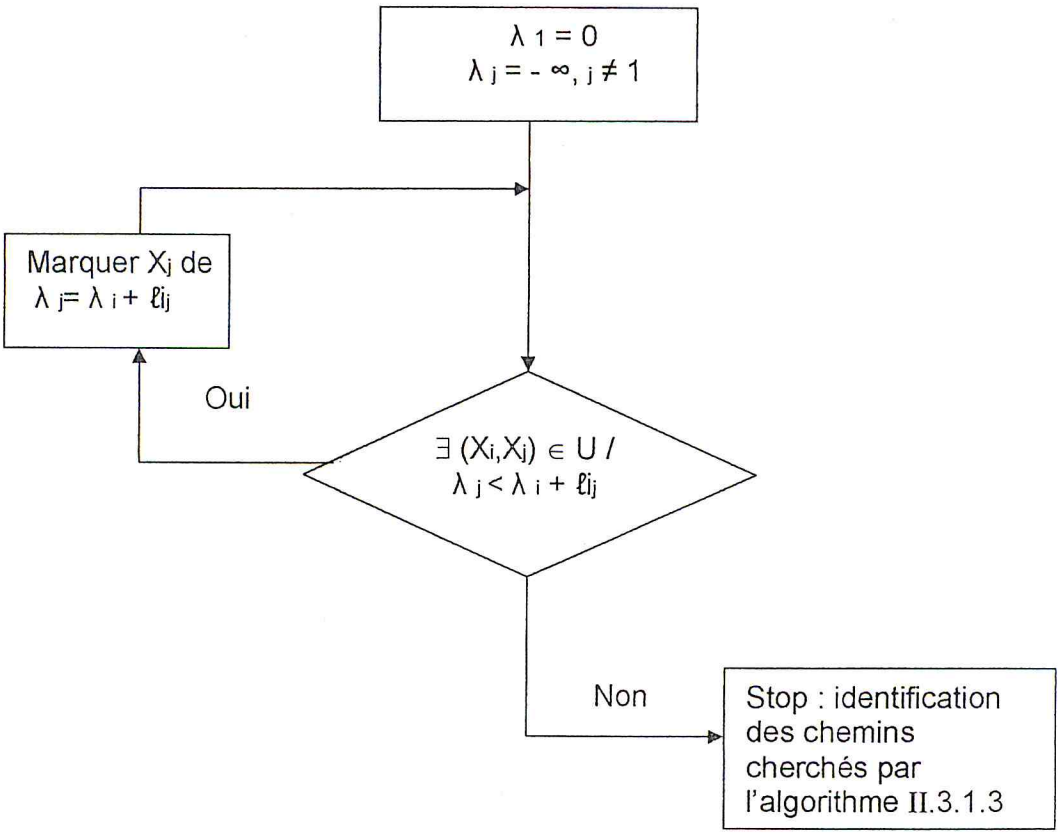
$$\ell_{ij} = \begin{cases} \ell_{ij} & \text{si } (X_i, X_j) \in U \\ +\infty & \text{si } (X_i, X_j) \notin U \\ 0 & \text{si } i=j \end{cases}$$

II.3.1.2. Organigramme de l'algorithme de Ford : obtention de chemins de longueur maximale :

Peu efficace, nous mentionnons cet algorithme ici pour des raisons d'ordre historique. Il permet l'obtention de chemins de longueur maximale, quels que soient les signes des longueurs ℓ_{ij}.

Son principe : On marque tout sommet X_j d'une marque λ_j qui correspond à une borne inférieure pour la distance maximale de X_1 à X_j. On augmente progressivement ces marques, à raison d'une par étape, et en suivant la procédure décrite dans l'organigramme ci-dessous. Lorsqu'une telle augmentation n'est plus possible, la marque λ_j est la distance maximale de X_1 à X_j. Un chemin de longueur maximale peut être construit, à partir des distances λ_j ainsi obtenues, par application de l'algorithme d'identification décrit en II.3.1.3.

L'organigramme ci-dessous présente cet algorithme de recherche de chemins de longueur maximale.



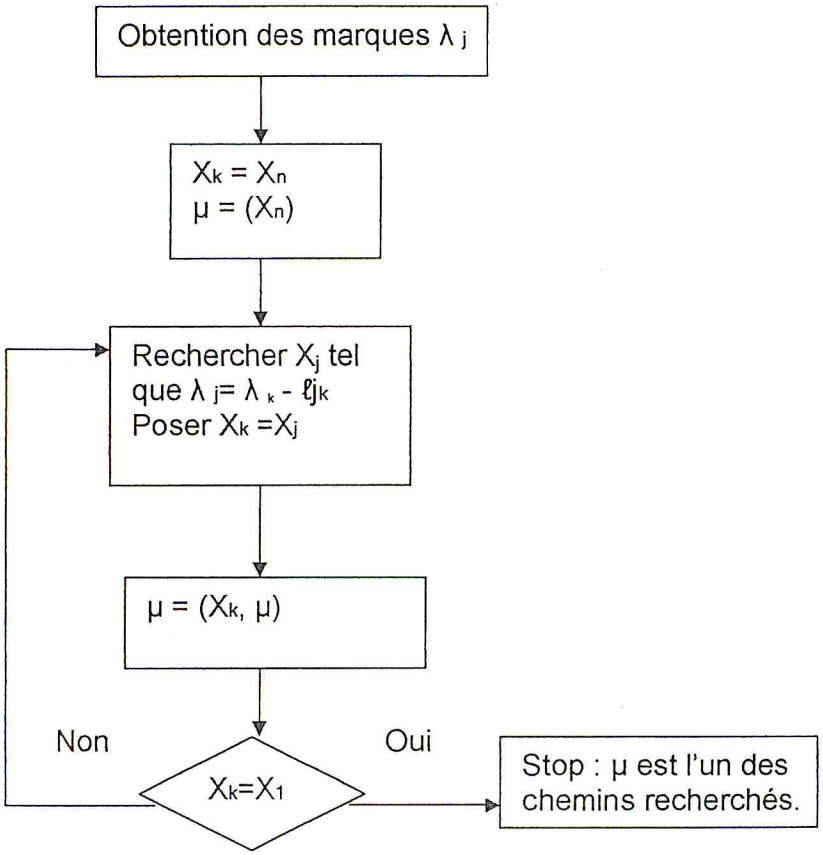
Organigramme II.1 de l'algorithme de Ford

L'inconvénient de cet algorithme est sa relative lenteur. Si le sommet marqué à chaque étape est choisi au hasard, le nombre d'itérations peut être fort élevé. Les algorithmes de Bellman-Kalaba repris en II.3.1.4 et II.3.1.5 sont, en moyenne, beaucoup plus rapides.

II.3.1.3.Organigramme de l'algorithme d'identification d'un chemin de longueur maximale :

L'identification d'un chemin de longueur maximale se fait à partir des distances maximales de X_1 aux autres sommets X_j. Les distances correspondent aux marques λ_j obtenues en fin d'application des algorithmes de Ford ou Bellman-Kalaba.

Son principe : Les chemins recherchés se reconstituent, à reculons, à partir de X_n, par juxtaposition d'arcs (X_j, X_k) tels que $\lambda j = \lambda_k - \ell_{kj}$.



Organigramme II.2 de l'algorithme d'identification d'un chemin de longueur maximale :

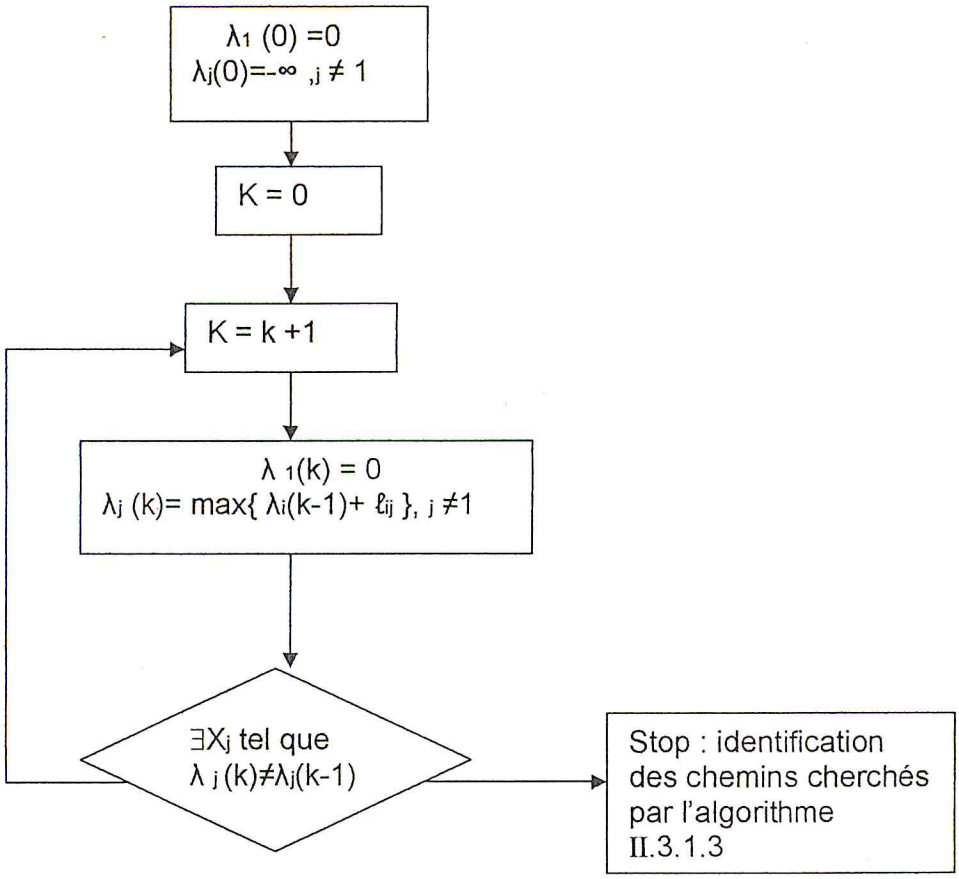
Rappelons que le chemin recherché n'est pas nécessairement unique.

II.3.1.4.Organigramme de l'algorithme de Bellman-Kalaba: obtention de chemins de longueur maximale :

Cet algorithme permet l'obtention de chemins de longueur maximale, quels que soient les signes de longueur ℓ_{ij}.

Son principe : On considère, au départ de X_1, les chemins d'un arc (étape 1), puis ceux de deux arcs au plus (étape 2),..., et ceux de k arcs au plus (étape k). A l'étape k, on marque tout sommet X_j d'une marque λj (k) qui représente la longueur du plus long chemin joignant X_1 à X_j en k arcs au plus. Lorsque toutes les marques obtenues aux étapes k-1 et k sont identiques, λj (k) est la distance maximale de X_1 à X_j. Un chemin de longueur maximale peut donc être construit, à partir de ces distances, par application de l'algorithme II.3.1.3.

L'organigramme ci-dessous se présente comme celui de l'algorithme II.3.1.2.



Organigramme II.3 de l'algorithme de Bellman-Kalaba pour l'obtention de chemins de longueur maximale

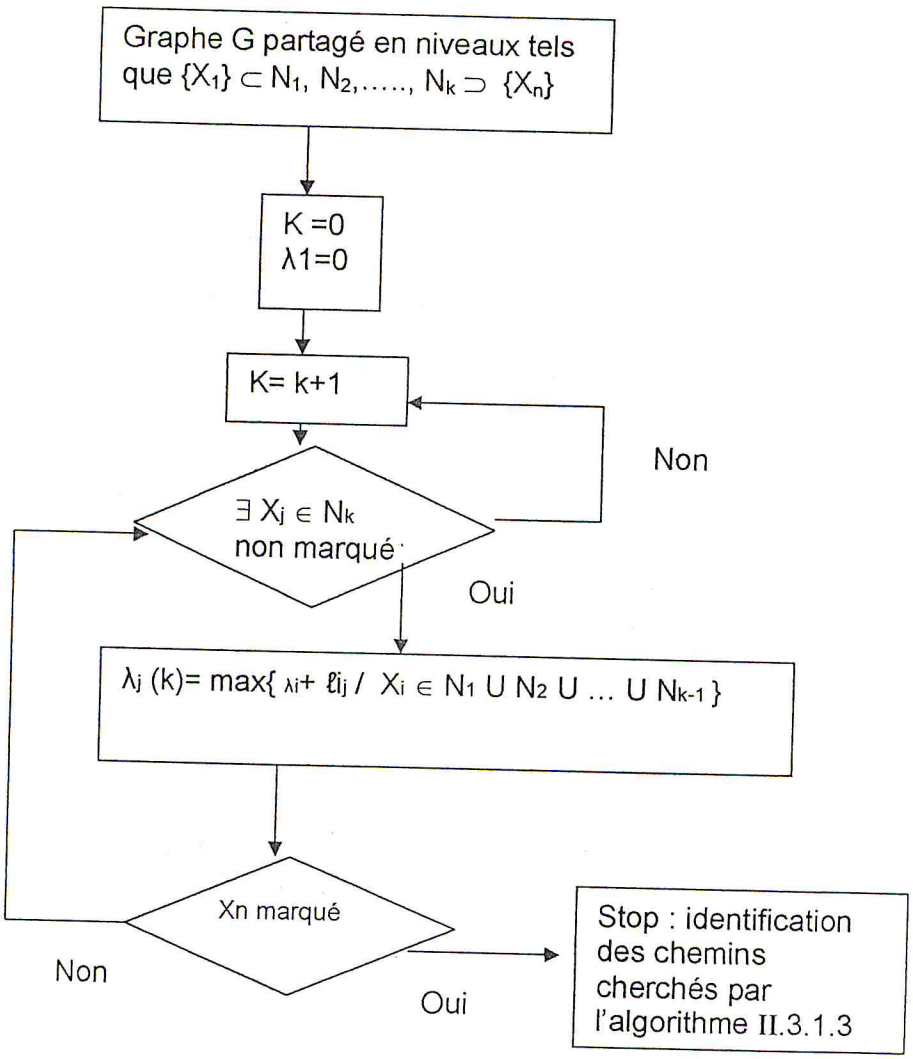
Soulignons que chaque sommet, dans cette version de l'algorithme, est marqué à chaque étape. Ceci peut représenter un volume de calculs assez important. La procédure simplifiée décrite en II.3.1.5 permet de remédier en grande partie à cet inconvénient.

II.3.1.5. Organigramme de l'algorithme de Bellman-kalaba simplifié dans le cas d'un partage en niveaux de G pour l'obtention de chemins de longueur maximale :

L'algorithme de Bellman-Kalaba est d'application beaucoup plus directe lorsque le graphe G peut être préalablement partitionné en niveaux.

Son principe : Un graphe G sans circuit peut être partagé en niveaux. Soient X (O), X (I),..., X (K - 1) les K niveaux. Pour faciliter la présentation, renumérotons ces niveaux en sens inverse et notons-les respectivement N_k, N_{k-1},..., N_1. Supposons que $X_1 \in N_1$ et $X_n \in N_k$. L'algorithme II.3.1.4 peut alors se simplifier comme suit :

A l'étape k, (k=1,2,...), on marque chaque sommet Xj du niveau N_k d'une marque λj qui représente la distance maximale de X_1 à X_j. Les marques ainsi attribuées sont donc définitives, ce qui réduit fortement le nombre d'opérations à effectuer.



Organigramme II.4 de l'algorithme de Bellman-kalaba simplifié dans le cas d'un partage en niveaux de G pour l'obtention de chemins de longueur maximale

II.3.2. Algorithme de Pert :

L'algorithme de Pert consiste à résoudre des problèmes d'ordonnancement et de planification de tâches (liées par des contraintes de précédence) composant un projet complexe.

Les sommets représentent des événements de début et/ou de fin de tâche.

Les arcs représentent les tâches et leurs durées.

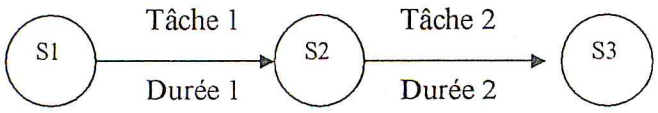


Fig II.4. Graphe d'ordonnancement

Un arc entrant pour un sommet donné indique que la tâche correspondante précède toutes les tâches qui correspondent aux arcs sortant du même sommet.

Algorithme II.13: // Algorithme du chemin critique de Pert
Données : graphe G = (V, E), sans circuits, des activités avec leur durée d_{ik}.
Résultat :

δ_i début des activités correspondant aux arcs (i, k) partant de i,

φ_i fin des activités correspondant aux arcs (k, i) arrivant à i,

durée du chemin critique.

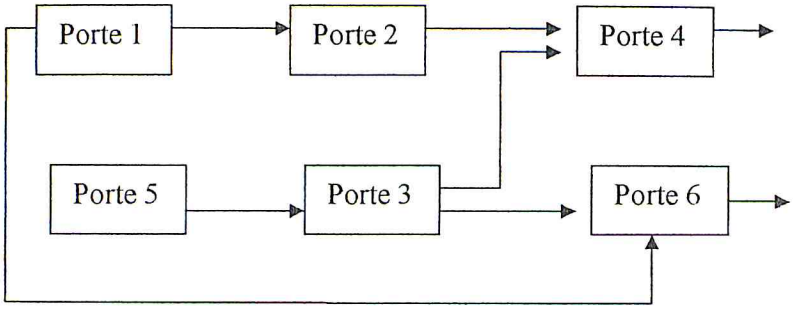
Début
I. Calcul des dates de début
$\delta_1 := 0$
Pour k := 2 à n faire $\delta_k := \max \{\delta_j + dik \mid j \ P(k)\}$
II. Calcul des dates de fin
$\varphi_n := \delta_n$
Pour k := n-1 à 1 faire $\varphi_k := \min \{\varphi_j - dik \mid j \ S(k)\}$
Fin.
Notations:
P (i) = {k V | (k, i) E}: ensemble des sommets prédécesseurs de i.
S (i) = {k V | (i, k) E}: ensemble des sommets successeurs de i.

- Un sommet i est critique si $\delta_i = \varphi_i$.
- Un arc (i, j) est critique si ses extrémités sont des sommets critiques et $d_{ij} = \delta_j - \delta_i$.
- Un chemin critique est un chemin de 1 à n n'utilisant que des arcs critiques, c'est-à-dire des activités telles que tout retard dans leur exécution provoquerait un retard de la fin du projet.

La durée du chemin critique est donnée par δ_n (ou par φ_n, les deux valeurs étant toujours égales). Elle correspond à la durée minimale du projet étant données les durées des tâches le composant et les précédences respectives.

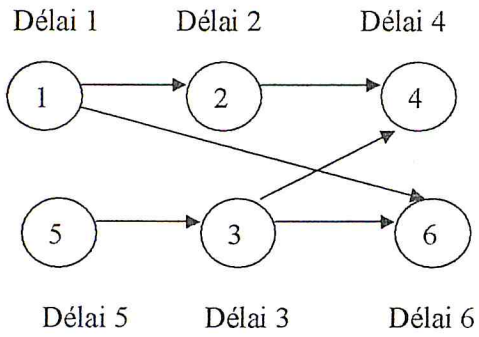
II.3.3. Formulation du problème pour un circuit intégré :

Un circuit intégré est composé d'un ensemble de portes dont chacune correspond à une fonction logique. Pour estimer le délai critique du circuit, le délai de chaque porte ainsi que les liens entre les portes du circuit doivent être connus.



FigII.5. Exemple de portes connectées

On peut modéliser le circuit par un graphe dont les sommets du graphe sont les portes et les arcs sont les liens entre les portes. Un arc entrant vers un sommet représente une entrée de la porte. Par contre, un arc sortant représente une sortie de la porte. A chaque sommet du graphe on associe le délai de la porte correspondante.



FigII.6. modélisation du circuit par un graphe

Notons que les sommets qui n'ont pas d'arcs entrants sont les sommets du premier niveau, et les sommets qui n'ont pas d'arcs sortants sont les sommets du dernier niveau.

II.3.4. Algorithme général pour le calcul du délai critique d'un circuit intégré :

Similairement aux algorithmes présentés précédemment, nous calculons les délais critiques pour chaque porte (sommet) du dernier niveau. Le délai critique du circuit sera le maximum des délais de ces sommets.

L'algorithme suivant est un algorithme récursif (la condition d'arrêt est la rencontre d'une des portes du premier niveau). Il a comme entrée un sommet du dernier niveau et en sortie le délai critique et les chemins critiques.

AlgorithmeII.14 : Algorithme général pour le calcul du délai critique d'un circuit intégré

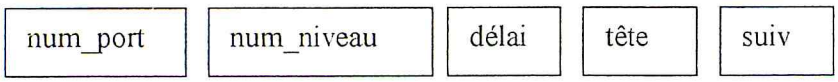
```
// Si est l'ensemble des sommets de niveau i
// s est un sommet de Si
Début

    Soit S_critique (i) l'ensemble des sommets du chemin critique jusqu' à Si
    si (niveau(s) == 1)
    alors {temps_critique=délai(s) ;
           return temps_critique ;
          }
    fsi
    max=0 ;
    Tant que (∃ t ∈ Si-1)
    Faire {temps_critique=calcul_délai_recursif (t) ;
           si (max <  temps_critique)
           alors {max=temps_critique ;
                  Remplacer les éléments de S_critique (i) par t ;
                 }
           sinon si (max == temps_critique)
                  alors Ajouter t à S_critique (i) ; // il existe un arc de t à s
                  fsi
           fsi
          }
    Fait
   Temps_critique=max+ délai(s) ;
    return temps_critique ;
Fin
```

II.3.5. Structure de données, de fichiers et algorithmes :

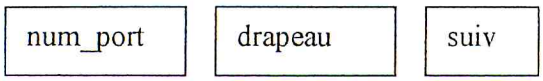
II.3.5.1. Structure de données :

Type NOEUD1 :



int num_port : numéro de la porte.
int num_niveau : numéro de niveau de la porte.
float délai : le délai de la porte.
NOEUD2 *tete : pointeur sur un nœud de type NOEUD2.
NOEUD1 *suiv : pointeur sur un nœud de type NOEUD1.

Type NOEUD2 :

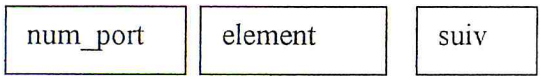


int num_port : numéro de la porte.
int drapeau : un drapeau pour marquer les portes.
NOEUD2 *suiv : pointeur sur un nœud de type NOEUD2.

Type LISTE :

NOEUD1 *tete : pointeur sur la liste des nœuds de type NOEUD1.

Type TABLEAU :

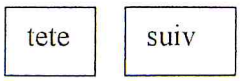


int num_porte : numéro de la porte.
float element : un élément du tableau.
TABLEAU *suiv : pointeur sur un nœud de type TABLEAU.

Type LISTE_T :

TABLEAU *tete : pointeur sur la liste des nœuds (ou élément) de type TBLEAU.

Type NOEUD3 :



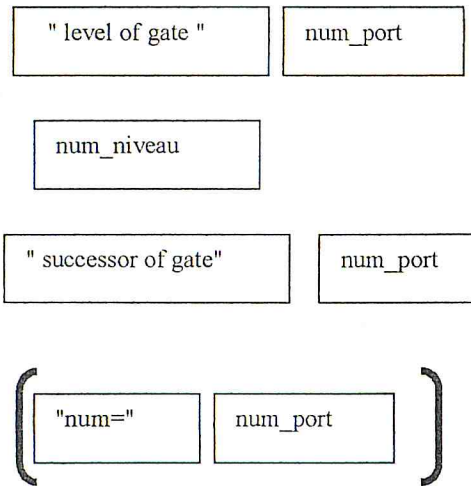
NOEUD2 *tete : pointeur sur un nœud de type NOEUD2.
NOEUD3 *suiv : pointeur sur un nœud de type NOEUD3.

Type LISTE_SAUV :

NOEUD3 *tete : pointeur sur la liste des nœuds de type NOEUD3.

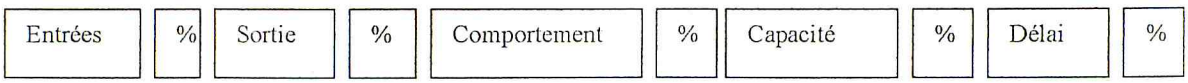
II.3.5.2.Structure des fichiers :

Fichier des niveaux et successeurs des portes :



num_port : numéro de porte.
num_niveau : niveau de la porte.
( )* : Occurrence pouvant se produire une ou plusieurs fois ou aucune fois

Fichier des délais : (le fichier à accès séquentiel)



Entrées : les entrées de la porte.
Sortie : la sortie de la porte.
Comportement : le comportement de la porte.
Capacité : la capacité de la porte.
Délai : le délai de la porte.

II.3.5.3.Les fonctions du programme :

La fonction lecture () :
Elle fait la lecture de deux fichiers (fichier des niveaux et successeurs des portes et fichier des délais), crée une liste de portes avec leurs prédécesseurs et une liste des portes avec leurs successeurs. Cette dernière sera par la suite réduite à une liste des portes du dernier niveau.

Algorithme II.15 : // Lecture des fichiers de données et la représentation du circuit par une liste

```
Début
  Créer deux listes :
  LISTE *p_list
  LISTE *p_list2
  NOEUD1 * p1,*pn1 ;
  NOEUD2 *p2 ;
 Ouvrir le fichier des niveaux et successeurs des portes
 Tant que (non fin de fichier des niveaux)
    Faire
     Créer deux nouveaux nœuds p1 et pn1 ;
     Lecture du numéro de porte ;
    Affecter numéro de porte au champ num_port de p1 et pn1 ;
     Lecture de niveau de porte ;
     Affecter niveau de porte au champ num_niveau  de p1 et pn1 ;
     Lecture de délai en faisant appel à la fonction lecture_délai (n°porte)
    Affecter délai au champ correspondant de p1 et pn1 ;
    Faisant appel à la fonction insérer  pour insérer p1 dans la liste p_list
    Faisant appel à la fonction insérer  pour insérer pn1 dans la liste p_list2
    Si (il existe des successeurs)
     alors
       Faire
         Lecture de niveau de porte
         Créer le nœud p2 ;
         Affecter numéro de porte à p2 ;
         Faisant appel à la fonction insérer  pour relier
         p2 à p1
     Fait
   Fin si
Fait
Faire appel à la fonction chercher_predecesseur () pour  compléter   list2
Faire appel à la fonction libérer liberer_list (p_list) pour libérer les nœuds de la liste
p_list qui ne sont pas du dernier niveau.
Fin
```

La fonction chercher_predecesseur (LISTE *list) :

Fait la recherche des portes qui alimentent une porte donnée du circuit.

Algorithme II.16:// rechercher les portes qui alimentent une porte donnée du circuit

```
Début

  NOEUD1 *p1,*p2
  NOEUD2 *p3
  int niveau_courant ,porte_courante

  p1=p-list2-> tete
  Tant que (non fin list2)
        Si (le niveau de p1 est supérieur a 1 ) alors
   Faire
      niveau_courant = p1->num_niveau
      Porte_courante  =p1->num_port
      p2 = p_list->tete
     Tant que (p2)
       Faire
         p3=p2->tete
          Tant que (p3)
            Faire
              Si (numéro de porte de p3 est égal à celui de la porte courante) alors
                Faire
                  Créer un noeud2 * p4
                  Affecter à p4 numéro de porte de p2
                  Initialiser le drapeau de p4 à 0
                  Faisant appel à la fonction insérer  pour relier p4 à p1
                Fait
             p3=p3->suiv
           Fait
            p2=p2->suiv
       Fait
           p1=p1->suiv
    Fait
Fin
```

La fonction délai_critique (NOEUD1 *ptr) :

C'est une fonction récursive qui renvoie le délai critique des nœuds du dernier niveau et met le drapeau des nœuds participant au chemin critique à 1 pour pouvoir les retrouver et les sauvegarder dans le fichier résultat.

Algorithme II.17 : // Détermination des délais et chemins critiques

```
Début
 NOEUD1 *ptr2,NOEUD2 *ptr1,*ptr2_sauv, NOEUD3 *ptr3_sauv ;
 float t_cr, max ;
  Si (ptr est de niveau 1) alors
    Faire
      t_cr =ptr->delai ;
      return t_cr ;
    Fait
  max=0 ;
 Créer une liste pour garder les nœuds (de type NOEUD3) qui participent au chemin
 critique :
   LISTE_SAUV *list_sauv ;
   ptr1=ptr->tete ;
    Tant que (ptr1)
   Faire
     ptr2=chercher (ptr1->num_port) ;
     t_cr= delai_critique (ptr2) ;
      Si (max < t_cr) alors
        Faire
          max=t_cr ;
           Si (il existe d'autres nœuds qui ont été insérés a la liste list_sauv) alors
             libérer (list_sauv) ;
           Fin si
          Créer un nœud de type NOEUD3 *p ;
           p->tete =ptr1 ;
           inserer_sauv (list_sauv, p) ;
       Fait
         Si non si (max == t_cr) alors
      Faire
        Créer un nœud de type NOEUD3 *p ;
        p->tete=ptr1 ;
        inserer_sauv (list_sauv,p) ;
     Fait
    t_cr=ptr->delai+max ;
     Si (il existe des nœuds dans la list_sauv) alors
    Faire
        ptr3_sauv=list_sauv->tete ;
         Tant que (ptr3_sauv)
            Faire
              ptr2_sauv =ptr3_sauv->tete ;
             prt2_sauv->drapeau=1 ;
              ptr3_sauv=ptr3_sauv->suiv ;
            Fait
```

```
          libérer (list_sauv) ;
      Fait
     ptr1=ptr1->suiv ;
 Fait
  return t_cr ;
Fin.
```

La fonction générer (int num) :
Génère les chemins critiques vers la porte de numéro num.

Algorithme II.18: Générer les chemins critiques

```
Début
   int indice, vect[n] ;
   int retour=0 ;
   NOEUD1 *p1,*ptr3 ;
   NOEUD2 *p2 ;
   int  i, y ;
     p1=chercher (num) ;
    Si (p1 est de niveau1) alors
      Faire
        Ecrire numéro de porte de p1 dans le fichier résultat ;
         retour=1 ;
        return 0 ;
      Fait
    Fin si
  p2=p1->tete ;
  Tant que (p2)
    Faire
      Si (drapeau de p2 est égal à 1) alors
        Faire
          prt3=chercher (p2->num_port) ;
           // Enregistrer le numéro de porte trouvé dans le vecteur vect
            vect [++indice]=ptr3->num_port ;
           y=générer (p2->num_port) ;
            Si (retour vrai)
               Pour i=indice-1 à 0
               Faire
                  Ecrire dans le fichier résultat la valeur de vect[i] ;
               Fait
             Fin si
              indice - -
          Fait
      Fin si
     p2=p2->suiv ;
     Fait
    retour=0 ;
    return 0 ;
Fin.
```

La fonction principale :

La fonction principale fait appel aux fonctions décrites précédemment.

Algorithme II.19 : // Programme principal du calcul du temps critique

```
Début
  int indice=0 vect[n] ;
  LISTE *p_list ;
  TABLEAU * p ;
   NOEUD1 * pt, * ptr ;
   float x, f ;
    lecture () ;
  // Créer un tableau dynamique pour sauvegarder les délais critiques pour les
     portes de sortie.
   LISTE_T * p_tab ;
   pt=p_list-> tete ;
   Tant que (pt)  // boucle sur les noeuds du dernier niveau
   Faire
     Faisant appel à la fonction chercher avec  ptr=chercher (pt->num_port) pour
chercher un nœud de type NOEUD1 dans la liste p_list2
      x=delai_critique (ptr) ;
     sauvegarder x dans un tableau
      pt=pt->suiv ;
  Fait
 Calcul le max du tableau
Ecrire le max dans le fichier résultat
    p=p_tab->tete ;
 Tant que (p)
   Faire
    Si (élément de p  == max) alors
      Faire
        indice=0 ;
        vect[indice]=p->num_port ;
        f=générer (p->num_port) ;
     Fait
  Fait
Fin.
```

II.3.6.Exemple d'illustration :

Pour faciliter aux lecteurs la compréhension de l'exemple, nous avons choisi un jeu de test « circuit » qui contient 10 portes seulement.

Les fichiers de données :

Soit le fichier suivant qui contient les délais des portes :

```
a b c % s1 % NOT ( a OR b OR c ) % 40. %
a b c d e % s5 % NOT ( a AND ( b OR ( c AND d ) OR e ) ) % 40. %
s6 k e f % s7 % NOT ( s6 AND ( k OR f ) OR e ) % 40. %
s10 e % s6 % NOT ( s10 AND e ) % 40. %
a b c e f % s2 % NOT ( a AND b AND c AND ( e OR f ) ) % 40. %
s2 s1 a % s10 % NOT ( s1 AND s2 AND a ) % 40. %
s5 b % s9 % NOT ( s5 OR b ) % 40. %
s7 a b % s12 % NOT ( s8 AND a AND b ) % 40. %
s2 b c d % s13 % NOT ( s3 OR ( b AND c ) OR d ) % 40. %
s9 s7 % s11 % NOT ( s9 AND s7 ) % 40. %
```

L'un des fichiers déduit de SPOT ([3], [4], [5]) contient les numéros des portes, leur niveau, ainsi que leurs successeurs :

```
 level of gate 5
1

successors of gate 5
num= 9
num= 6

 level of gate 2
1

 successors of gate 2
num= 7

 level of gate 1
1

 successors of gate 1
num= 6

 level of gate 9
2

 successors of gate 9

 level of gate 7
2

 successors of gate 7
num= 10

 level of gate 6
2
```

```
 successors of gate 6
num= 4

 level of gate 4
3

 successors of gate 4
num= 3

 level of gate 3
4

 successors of gate 3
num= 8
num= 10

 level of gate 10
5

 successors of gate 10

 level of gate 8
5

 successors of gate 8
```

Le fichier qui contient les constantes est le suivant :

```
1 : 1  6  3  4
2 : 3  8  3  8
3 : 2  7  3  6
4 : 2  3  1  4
5 : 4  7  2  9
6 : 3  4  1  6
7 : 1  4  2  3
8 : 3  4  1  6
9 : 2  7  3  6
10 : 2  3 1   4
```

Le fichier délai (à accès direct) correspondant est alors :

```
2.09
3.14
2.61
0.70
2.26
0.87
1.22
```

0.87
2.61
0.70

Le graphe correspondant au circuit et montrant le délai de chaque porte est alors celui indiqué dans la Fig.II.7.

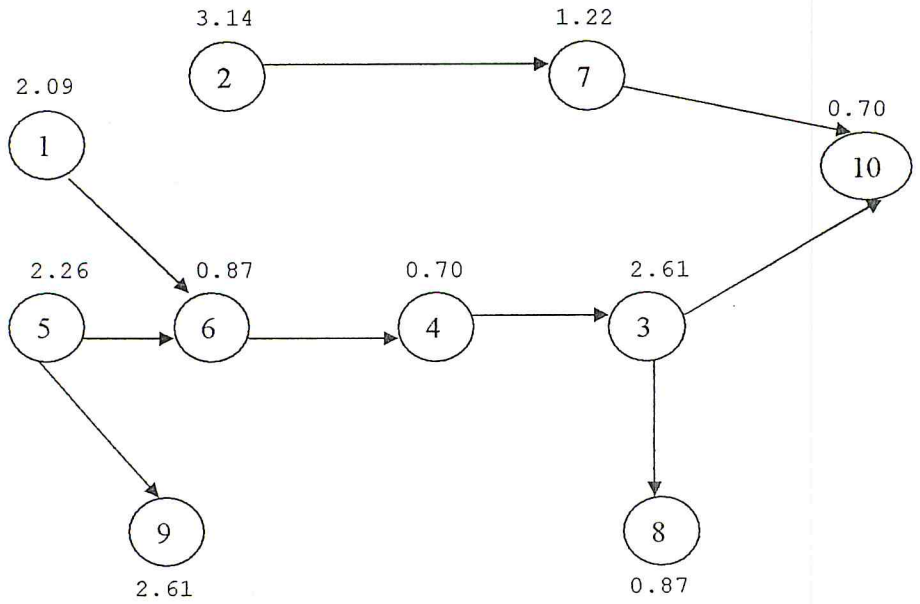


Fig II.7. : le graphe correspondant

Fichier des résultats partiels

7.31
5 6 4 3 8

Le délai critique du circuit est égal à 7.31ns, et le chemin critique est 5 6 4 3 8.

II.4.Conclusion

Nous avons présenté dans ce chapitre les techniques développées pour la détermination du délai de chaque porte logique, ainsi que celle des délais et chemins critiques du circuit. Enfin, notons que ces techniques ont été implémentées dans FREEZER1 qui est une première version d'un outil d'aide à la conception de circuits digitaux à faible consommation de puissance [6].

CHAPITRE III

Détermination de la solution discrète à partir de la solution continue

III.1. Introduction

On a vu au chapitre précédent la détermination des délais des portes et le calcul du temps critique d'un circuit en fonction des tensions d'alimentation des portes logiques et de celles de seuil des transistors.

On verra dans ce chapitre comment, à partir d'une solution continue, déterminer les valeurs des tensions d'alimentation à affecter aux portes logiques du circuit ainsi que celles des tensions de seuil des transistors. En effet, le problème d'optimisation de la consommation de puissance peut être formulé par une fonction objective qu'il faut minimiser en tenant compte de la contrainte du temps de réponse du circuit. Toutefois, la solution obtenue peut ne pas être réalisable, et l'objet de ce chapitre porte précisément sur le développement d'une heuristique permettant de transformer une solution continue en une solution discrète, réalisable.

III.2. Problèmes d'optimisation combinatoire

III.2.1. Définition

Avant de présenter notre heuristique, nous aimerions tout d'abord faire quelques rappels sur les problèmes d'optimisation combinatoire, puis voir comment ceux-ci peuvent être appliqués au problème de la consommation de la puissance.

L'optimisation d'une fonction $f : A \rightarrow B$ (définie de *A* vers B) consiste à chercher un élément x_0 dans A tel que pour tout élément *x* de A, $f(x_0) > f(x)$ (maximisation) ou $f(x_0) \leq f(x)$ (minimisation).

Notons que l'ensemble A peut être fini, dénombrable ou pas, et que les éléments de A sont appelés les *solutions possibles* et la fonction f est appelée la *fonction objective*.

III.2.2. Méthodes de résolution des problèmes d'optimisation combinatoire

Selon le fait qu'elles garantissent ou pas l'optimalité de la solution obtenue, on distingue les méthodes exactes et les méthodes approchées. Les méthodes exactes parcourent parfois l'espace de recherche de manière exhaustive et par conséquent, la solution trouvée est optimale. En général, le temps de calcul d'un algorithme augmente exponentiellement avec la taille du problème. Ceci rend souvent impossible l'application d'une méthode exacte, et des méthodes approchées sont alors proposées. Celles-ci ne parcourent pas tout l'espace de recherche et sont en général basées sur des heuristiques permettant de trouver des solutions assez rapidement mais peuvent toutefois ne pas générer une solution optimale

III..3. Complexité Algorithmique

On appelle complexité d'un algorithme son temps de calcul (autrement dit le temps nécessaire à son exécution), indépendamment de la vitesse de la machine sur laquelle il est exécuté et de la qualité du code produit par le compilateur. On s'intéresse à l'évolution de cette complexité en fonction de la taille des données : étant donnée une procédure p, on s'intéresse à la variation de la durée du calcul de

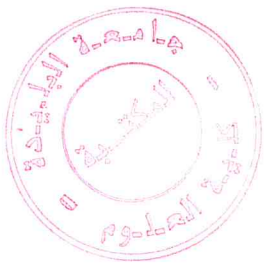
La consommation statique est notamment importante pour les technologies récentes où les tensions de seuil des transistors sont tellement faibles, qu'il y'ait une possibilité d'existence d'un courant dans le canal du transistor même lorsque le circuit est au repos.

Les équations déterminant les trois types de consommation de puissance sont les suivantes :

** consommation due aux courts circuits :*

$$P_{sc} = V_{dd} * I_{sc} \qquad \textbf{(III.1)}$$

où : - V_{dd} est la tension d'alimentation du circuit
- I_{cs} est le courant du court-circuit



** consommation dynamique :*

$$P_{sw} = 0.5 * f * V_{dd}^2 * \sum_{i=1}^{Nb\_portes} C_{gi} N_{gi} \qquad \textbf{(III.2)}$$

où : - f est la fréquence de fonctionnement du circuit
- C_{gi} est la capacité de charge de la porte i
- N_{gi} est le nombre de transitions de la porte i (nombre de charges/décharges de C_{gi})

** consommation statique :*

$$P_{leak} = 0.28125 * K * W_{avg} * L * V_{dd} * Nb_{trans} * 10^{-V_t/\alpha_V} \qquad \textbf{(III.3)}$$

où : - K=10 µA/µm
- W_{avg} est la largeur moyenne des transistors
- L est la longueur du canal (en µm) du transistor dans la technologie considérée
- Nb_{trans} est le nombre de transistors dans le circuit
- V_t est la tension de seuil initiale du transistor
- α_V = 0.095 V

III.6. Minimisation de la consommation de la puissance

III.6.1. Nécessité [6]

Avec la proéminence croissante des systèmes portables (PDAs, téléphones portables, PCs portables, etc ...) et la durée de vie limitée des batteries, la conception de circuits à faible consommation de puissance est devenue une nécessité. Bien que des efforts aient été faits pour augmenter la durée de vie des batteries (remplacement de batteries NiCd par des batteries NiMH), une autonomie intéressante n'est pas encore atteinte. De plus, les technologies actuelles permettent une intégration de millions de transistors sur une puce (implémentations SOC).

Aussi, même si le système VLSI est alimenté par le secteur, une conception classique de ce système entraînerait une forte consommation de puissance, et de là, augmenterait la température de fonctionnement du système, ce qui engendrerait un disfonctionnement de celui-ci. De ce fait, l'aspect dissipation de puissance est abordé à chaque niveau d'abstraction soit pour proposer des méthodes aussi différentes que variées pour estimer ce paramètre, soit pour concevoir des circuits à faible consommation de puissance, et ce, à différents niveaux d'abstraction. Notons que dans le cadre de ce travail, il s'agit de considérer le problème au niveau d'abstraction *transistor*.

La consommation de puissance d'un circuit est due principalement aux courts circuits, à l'activité dynamique du circuit, et aux fuites du courant dans les transistors. La consommation due aux courts circuits peut être réduite grâce à l'utilisation d'une technologie appropriée (exemple, technologie CMOS) et en réduisant les temps de transition des signaux d'entrée, ce qui aura pour effet de diminuer considérablement la durée où les transistors N et P conduisent simultanément.

La consommation de la puissance dynamique peut être réduite en optant pour une faible valeur de l'alimentation du circuit, mais ce serait au détriment de la rapidité du circuit.

Enfin, les fuites de courant peuvent être réduites en utilisant des transistors ayant des tensions de seuil assez élevées, mais ce serait aussi au détriment de la rapidité du circuit. La problématique consiste alors à réduire la consommation de la puissance du circuit, tout en assurant à celui-ci une certaine performance en matière de rapidité.

III.6.2. Méthodologie [6]

Certaines solutions existantes consistent à résoudre un problème d'optimisation combinatoire permettant d'affecter des valeurs différentes de tensions d'alimentation pour les portes logiques, et également des tensions de seuil différentes pour les transistors du circuit de sorte à minimiser la consommation de puissance du circuit tout en satisfaisant la contrainte *temps*. Cette solution peut rendre la fabrication du circuit très coûteuse, voire impossible du fait de l'utilisation de plusieurs tensions d'alimentation et de plusieurs tensions de seuil. Aussi, la communauté scientifique travaillant dans ce domaine se limite généralement à deux valeurs d'alimentation (bornes supérieure et inférieure) et à deux valeurs de tension de seuil pour chaque type de transistor. Ensuite, la solution *continue* est remplacée par une solution *discrète* du fait que les valeurs des tensions d'alimentation et celles des tensions de seuil obtenues peuvent être différentes des bornes assignées.

Au niveau d'abstraction *transistor*, plusieurs méthodes se basent sur l'interpolation. A partir d'une fonction objective modélisée par un posynôme (Eq. III.4), des contraintes sont introduites pour définir complètement le problème d'optimisation combinatoire. Ensuite, une méthode de programmation convexe est utilisée pour résoudre ce problème.

$$f_{obj} = \sum_{i=1}^{m} c_i \prod_{j=1}^{n} x_j^{\alpha_{ij}} \qquad c_i \geq 0 \,,\, x_j > 0 \qquad (III.4)$$

Notons que les variables x_i portent sur les tensions d'alimentation, les tensions d'entrée, les capacités de charge, les dimensions des transistors et les tensions de

seuil, et que le nombre de valeurs à choisir pour chaque variable doit être assez élevé pour pouvoir espérer une bonne interpolation.

Il est aussi à noter que la résolution de ce problème d'optimisation combinatoire peut conduire à une solution continue, c'est-à-dire à une solution où les valeurs des tensions d'alimentation et de celles des tensions de seuil sont différentes des bornes assignées. De ce fait, il s'agit de transformer la solution continue en une solution discrète réalisable, sans altérer sérieusement l'optimum (concernant la consommation de la puissance) trouvé. Ceci est l'objet du paragraphe qui suit.

III.7. Présentation d'une heuristique générant une solution réalisable

Comme nous l'avions dit précédemment, la consommation due aux courts circuits peut être réduite en choisissant une technologie appropriée (par exemple, la technologie CMOS) pour la conception du circuit.

Du fait que les paramètres *consommation de puissance dynamique* et *temps de réponse du circuit* sont antagonistes, il s'agira de trouver un bon compromis. Celui-ci peut être atteint en optant pour deux bornes V_{ddL} et V_{ddH} en ce qui concerne les tensions d'alimentation du circuit. La valeur V_{ddH} ($V_{ddH} > V_{ddL}$) affectée à certaines portes logiques aura pour effet de rendre le circuit rapide, mais au détriment de la consommation de la puissance dynamique (voir Eq.III.2). Aussi, afin de compenser, V_{ddL} est affectée à certaines portes logiques afin de diminuer la consommation due à l'activité dynamique du circuit. De ce fait, l'équation III.2 est transformée comme suit :

$$P_{sw}^{'} = 0.5 * f * \left[V_{DD,L}^2 \sum_{i=1}^{|E_L|} C_{gLi} * N_{gLi} + V_{DD,H}^2 \sum_{i=1}^{|E_H|} C_{gHi} N_{gHi} \right] \qquad \textbf{(III.5)}$$

où : - E_L et E_H sont respectivement les ensembles des portes alimentées par V_{ddL} et V_{ddH}
- C_{gLi} et C_{gHi} sont respectivement les capacités de charge des portes alimentées par V_{ddL} et V_{ddH}
- N_{gLi} et N_{gHi} sont respectivement les transitions des portes alimentées par V_{ddL} et V_{ddH}

De même, afin de trouver un bon compromis entre le temps de réponse du circuit et la consommation de la puissance statique, deux tensions de seuil pour chaque type de transistor (N ou P) sont utilisées. L'équation III.3 est alors transformée comme suit :

$$P_{leak}^{'} = 0.28125 * K * W_{avg} * L * \sum_{i=1}^{nb\_portes} (Nb_{N,i} * 10^{-v_{tN,i}/\omega v} + Nb_{P,i} * 10^{v_{tP,i}/\omega v}) * V_{DD,i} \qquad \textbf{(III.6)}$$

où : - $Nb_{N,i}$ est le nombre de transistors N dans la porte i
- $Nb_{P,i}$ est le nombre de transistors P dans la porte i
- $v_{tN,i}$ est la tension de seuil initiale des transistors N de la $i^{ème}$ porte
- $v_{tP,i}$ est la tension de seuil initiale des transistors P de la $i^{ème}$ porte
- $V_{DD,i}$ est la tension d'alimentation (V_{ddL} ou V_{ddH}) de la $i^{ème}$ porte

La consommation totale considérée est alors :

$$P_{tot} = P'_{sw} + P'_{leak} \qquad \textbf{(III.7)}$$

Les équations (III.5), (III.6) et (III.7) sont intégrées dans l'outil SPOT ([3], [4] et [5]) pour déterminer les consommations des puissances dynamique, statique et totale.

A partir de :
- la solution continue,
- des procédures déterminant le temps critique dans un circuit
- et de l'outil SPOT qui nous permet de connaître la valeur de P_{tot} une fois l'affectation des tensions d'alimentation et des tensions de seuil faite,

il est question de connaître justement cette affectation (si elle existe) de sorte que P_{tot} soit la plus minimale possible, tout en satisfaisant la contrainte *temps*. Avant d'aborder cet aspect d'affectation de tensions, considérons la complexité algorithmique de ce traitement d'affectation.

Soit un circuit à N portes logiques. Soient V_{ddL}et V_{ddH} les bornes des tensions d'alimentation, V_{tNL} et V_{tNH} les bornes des tensions de seuil des transistors N, et V_{tPL} et V_{tpH} celles des tensions de seuil des transistors P. Du fait que :

- $V_{dd} \in \{V_{ddL}, V_{ddH}\}$,
- $V_{tN} \in \{V_{tNL}, V_{tNH}\}$,
- $V_{tP} \in \{V_{tPL}, V_{tPH}\}$,

une méthode consisterait à considérer toutes les combinaisons possibles d'affectation de tensions, puis de retenir celle qui minimise le plus P_{tot}, tout en satisfaisant la contrainte du temps de réponse du circuit. Le nombre de ces combinaisons serait alors égal à 2^{3*N}. De ce fait, l'implémentation de cette méthode ne pourrait donner la solution en un temps de réponse raisonnable pour un nombre conséquent de portes logiques. Aussi, il s'agira de développer une heuristique qui permettra de générer une solution pré-optimale en un temps polynomial. Cette heuristique devrait donc être développée selon une stratégie telle que la solution obtenue soit bonne. Ceci fait l'objet de ce qui suit dans ce chapitre.

La stratégie globale adoptée dans l'heuristique est la suivante :

- viser d'abord le cas idéal :
  Affecter V_{ddL} à toutes les portes
  Affecter V_{tNH} à tous les transistors de type N
  Affecter V_{tPL} à tous les transistors de type P

Si le temps de réponse du circuit (déterminé à partir des procédures décrites dans le chapitre précédent) est inférieur ou égal à un temps de réponse fixé par l'utilisateur, alors le cas idéal est atteint : la plus petite valeur de P_{tot} est obtenue tout en satisfaisant la contrainte du temps de réponse du circuit.

Dans le cas contraire, c'est-à-dire dans le cas où cette affectation viole la contrainte *temps*, il s'agira d'affecter V_{ddH} à toutes les portes pour lesquelles la solution

continue a généré la plus grande tension (soit n_i le nombre de ces portes) et V_{ddL} à toutes les autres portes. Ceci permettrait d'augmenter les chances de satisfaire la contrainte *temps*, tout en visant une affectation de faibles valeurs d'alimentation (donc pour minimiser P_{tot}) aux autres portes. En d'autres termes, la plus grande valeur de V_{dd} (qui est V_{ddH}) n'est affectée à certaines portes qu'en cas de violation de la contrainte portant sur le temps de réponse du circuit. Si la contrainte *temps* est satisfaite, l'affectation des tensions de seuil est abordée. Sinon, à partir de l'affectation de V_{ddH} pour certaines portes, il s'agira alors de déterminer de nouveau la solution continue, mais pour uniquement $(N-n_i)$ portes en ce qui concerne l'affectation de V_{dd}. Cette procédure est réitérée jusqu'à l'affectation de V_{dd} à *toutes* les portes logiques du circuit.

Du fait que l'algorithme d'affectation des différentes tensions sera donné ultérieurement, nous nous limitons à dire que le même principe adopté pour les tensions d'alimentation est utilisé pour l'affectation des tensions de seuil des transistors N et P. C'est-à-dire que pour les transistors N, la valeur V_{tNL} est affectée aux transistors pour lesquels la solution continue a généré la plus petite valeur de tension (soit n_j ce nombre) alors que V_{tNH} est affectée à tous les autres transistors. Si la contrainte *temps* est satisfaite, l'affectation des tensions de seuil aux transistors P est abordée. Sinon, la solution continue est de nouveau déterminée pour $(N-n_j)$ portes, et le processus continue ainsi, en n'affectant V_{tNL} à des transistors qu'au besoin, c'est-à-dire que lorsque la contrainte *temps de réponse du circuit* n'est pas satisfaite.
En ce qui concerne les transistors P, le même principe décrit précédemment est adopté, sauf que V_{tPH} est affectée aux transistors P au fur et à mesure que la contrainte du temps de réponse n'est pas satisfaite.

III.8.Algorithme général de l'heuristique :

Algorithme III.1 : // Algorithme général de l'heuristique

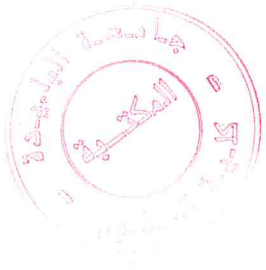
```
Début
 Entrées : fichier comportement
           Tfixé
           Constantes VDDL, VDDH ,VTHNL,VTHNH,VTHPL et VTHPH

  Affecter VDDL à toutes les portes
  Affecter VTHNH à tous transistors N
  Affecter VTHPL à tous transistors P
  determiner_constantes() ; // calculer Kr1, Kr2, Kf1, Kf2
  determiner_delai() ;      // calculer les délais des portes
  determiner_TCR() ;        // calculer le temps critique du circuit
  Si Tcr ≤ Tfixé
  alors Determiner_Puissance()  // exécuter SPOT pour déterminer Ptot
  Si non
    Faire
      Determiner_solution_continue() ;  // Génération aléatoire d'une sol. continue
```

fin=0 ;
Tant que (non fin)
Faire
Affecter V_{ddH} aux portes de S_1= {P_i ; P_i a la plus grande valeur de V_{dd}} ;
Affecter V_{ddH} à toutes les portes de S_2= {P_j ; P_j alimente une ou plusieurs portes de S_1} ;
Affecter V_{ddL} aux autres portes de S_3 = {P_k ; $P_k \notin (S_1 \cup S_2)$ } ;
determiner_delai() ;
determiner_TCR() ;
Si Tcr ≤ Tfixé alors fin =1 ;
Si non
Faire
Si S_3 = Ø alors { pas de solution ; exit ; }
Sinon determiner_solution_continue () ;
Fin si
Fait
Fin si
Fait
fin =0
Tant que (non fin)
Faire
Affecter V_{thNL} aux portes de S'_1= {P_i ; transistors de P_i ont la plus petite valeur de V_{thN}} ;
Affecter V_{thNH} aux autres transistors des portes de S'_2= {P_j ; $P_j \notin S'_1$} ;
determiner_delai() ;
determiner_TCR() ;
Si Tcr ≤ Tfixé alors fin =1
Sinon
Faire
Si S'_2 = Ø alors { pas de solution ; exit ; }
Sinon determiner_solution_continue() ;
Fin si
Fait
Fin si
Fait
fin =0
Tant que (non fin)
Faire
Affecter V_{thPH} aux portes S'_1= {P_i ; transistors de P_i ont la plus grande valeur de V_{thP}} ;
Affecter V_{thPL} aux autres transistors des portes de S'_2= {P_j ; $P_j \notin S'_1$} ;
determiner_delai() ;
determiner_TCR() ;
Si Tcr ≤ Tfixé alors fin =1
Si non
Faire
Si S'_2 = Ø alors { pas de solution ; exit ; }
Sinon determiner_solution_continue () ;
Fin si
Fait

```
            Fin si
           Fait
        determiner_Puissance() ;
      Fait
    Fin si

      Sorties : Puissance du circuit
            Les valeurs de Vdd pour toutes les portes
            Les valeurs de VthN et VthP pour tous les transistors
Fin
```



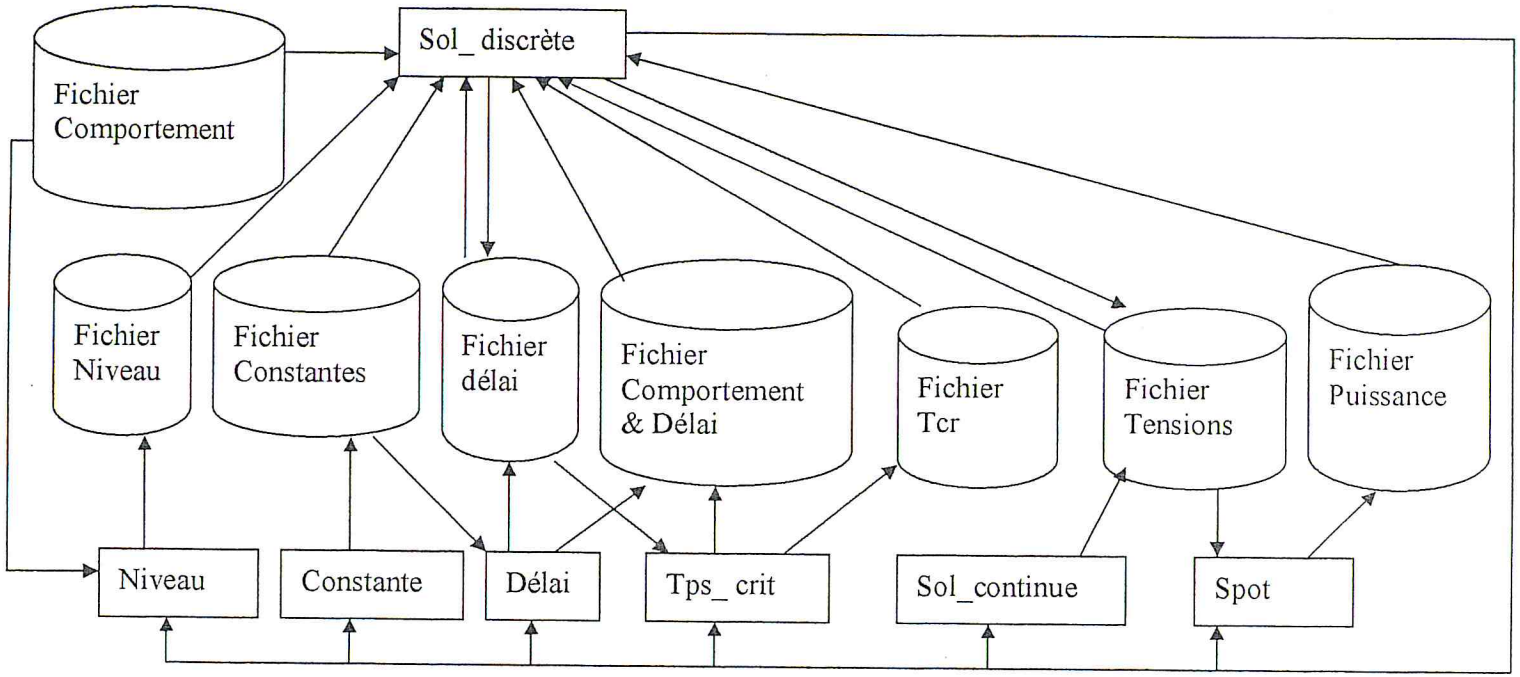


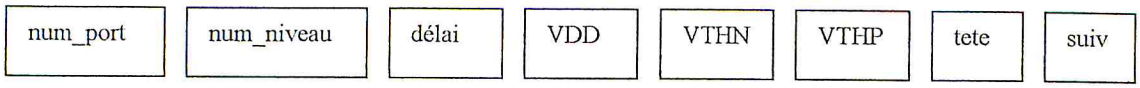
Fig III.1: Organisation des modules de transformation de la solution continue en une solution discrète.

Le module *solution discrète* fait appel aux autres modules (constante,temps critique, ...) qui génèrent les fichiers nécessaires pour son exécution. L'organisation des différents modules est donnée dans la Fig.III.1.

III.9.Structure de données , de fichiers et algorithmes:

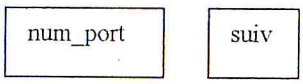
III.9.1.Structure des données :

Type NOEUD1 :



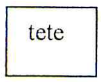
int num_port : numéro de la porte
int num_niveau : numéro de niveau de la porte
float délai : le délai de la porte
double VDD : l'alimentation de la porte
double VTHN : la tension de seuil de transistor N
double VTHP : la tension de seuil de transistor P
NOEUD2 *tete : pointeur sur un nœud de type NOEUD2
NOEUD1 *suiv : pointeur sur un nœud de type NOEUD1

Type NOEUD2 :



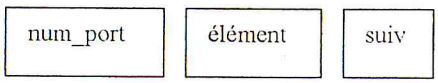
int num_port : numéro de la porte
NOEUD2 *suiv : pointeur sur un nœud de type NOEUD2

Type LISTE :



NOEUD1 *tete : pointeur sur la liste des nœuds de type NOEUD1

Type TABLEAU :



int num_port : numéro de la porte
float élément : l'élément du tableau
TABLEAU *suiv : pointeur sur un nœud de type TABLEAU

Type LISTE_T :

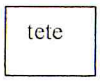
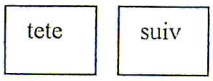


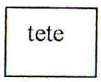
TABLEAU *tete : pointeur sur la liste des nœuds de type TABLEAU

Type NOEUD3 :



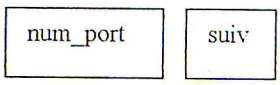
NOEUD2 *tete : pointeur sur un nœud de type NOEUD2
NOEUD3 *suiv : pointeur sur un nœud de type NOEUD3

Type LISTE_SAUV :



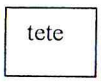
NOEUD3 *tete : pointeur sur la liste des nœuds de type NOEUD3

Type NOEUD4 :



int num_port : numéro de la porte
NOEUD4 *suiv : pointeur sur un nœud de type NOEUD4

Type LISTE_PORT :



NOEUD4 *tete : pointeur sur la liste des nœuds de type NOEUD4

III.9.2. Structure de fichiers :

Remarque :
Les fichiers de ce module sont les mêmes que ceux présentés précédemment.

III.9.3. Les fonctions du programme :

La fonction lecture () :

Elle fait la lecture de deux fichiers (fichier des niveaux et successeurs des portes, et fichier des délais), et crée une liste de portes avec leurs prédécesseurs et une liste des portes avec leurs successeurs. Cette dernière sera par la suite réduite à une liste des portes du dernier niveau.

Algorithme III.2 // Lecture des fichiers de données et la représentation du circuit par une liste

```
Début
  Créer deux listes :
  LISTE *p_list
  LISTE *p_list2
  NOEUD1 * p1,*pn1 ;
  NOEUD2 *p2 ;
 Ouvrir le fichier des niveaux et successeurs des portes
 Tant que (non fin de fichier des niveaux)
    Faire
     Créer deux nouveaux nœuds p1 et pn1 ;
     Lecture du numéro de porte ;
    Affecter numéro de porte au champ num_port de p1 et pn1 ;
     Lecture de niveau de porte ;
     Affecter niveau de porte au champ num_niveau  de p1 et pn1 ;
     Lecture de délai en faisant appel à la fonction lecture_délai (n°porte)
    Affecter délai au champ correspondant de p1 et pn1 ;
    Faisant appel à la fonction insérer  pour insérer p1 dans la liste p_list
    Faisant appel à la fonction insérer  pour insérer pn1 dans la liste p_list2
    Si (il existe des successeurs)
     alors
       Faire
         Lecture de niveau de porte
         Créer le nœud p2 ;
         Affecter numéro de porte à p2 ;
         Faisant appel à la fonction insérer  pour relier
         p2 à p1
     Fait
   Fin si
Fait
Faire appel à la fonction chercher_predecesseur () pour  compléter   list2
Faire appel à la fonction libérer liberer_list (p_list) pour libérer les nœuds de la liste
p_list qui ne sont pas du dernier niveau.
Fin
```

La fonction chercher_predecesseur (LISTE *list) :

Fait la recherche des portes qui alimentent une porte donnée du circuit.

Algorithme III.3 : // Rechercher les portes qui alimentent une porte donnée du circuit

```
Début
  NOEUD1 *p1,*p2
  NOEUD2 *p3
  int niveau_courant ,porte_courante

  p1=p-list2-> tete
  Tant que (non fin list2)
        Si (le niveau de p1 est supérieur a 1) alors
   Faire
      niveau_courant = p1->num_niveau
      Porte_courante  =p1->num_port
      p2 = p_list->tete
     Tant que (p2)
      Faire
        p3=p2->tete
         Tant que (p3)
          Faire
            Si (numéro de porte de p3 est égal à celui de la porte courante) alors
              Faire
                Créer un noeud2 * p4
                Affecter à p4 numéro de porte de p2
                Initialiser le drapeau de p4 à 0
                Faire appel à la fonction insérer  pour relier p4 à p1
              Fait
            p3=p3->suiv
          Fait
           p2=p2->suiv
       Fait
           p1=p1->suiv
    Fait
Fin
```

La fonction Créer list port (float var) :

Créer une liste des portes à partir des nœuds dont le champ élément est égal à var (le max ou le min)

```
Début
 TABLEAU *psuiv ;
  Créer list_port de type LISTE_PORT ;
  list_port->tete=0 ;
  psuiv = list->tete ;
```

```
Tant que (psuiv)
    Faire
     Si (psuiv->élément=var) alors
        Faire
          Créer p de type NOEUD4 ;
          p->num_port = psuiv->num_port ;
         Faire appel à la fonction insérer  pour insérer p dans la liste list_port
        Fait
     Fin si
    psuiv=psuiv->suiv ;
   Fait
Fin
```

La fonction affecter_VDDh (int num) :

// Affecter V_{DDh} à la porte de numéro num ainsi qu'à toutes les portes qui l'alimentent

```
Algorithme III.4://affectation de  VDDh
Début
 NOEUD1 *p1 ;
 NOEUD2 *p2 ;
p1=chercher (num)    // chercher un nœud de type NOEUD1 dans p_list2
Si (p1 est de premier niveau) alors
      Faire
       Si (p1->VDD != VDDh) alors
         p1->VDD = VDDh ;
         Appeler la fonction libérer avec liberer (p1->num_port) pour libérer le nœud de
              la liste list qui comporte le même numéro que le nœud p1
       Fin si
     Fait
  Si non
     Faire
      Si (p1->VDD != VDDh) alors
        Faire
         p1->VDD=VDDh ;
         Appeler la fonction libérer avec liberer (p1->num_port) pour libérer le nœud de
la liste list qui comporte le même numéro que le nœud p1
        Fait
       Fin si
      p2=p1->tete ;
       Tant que (p2)
        Faire
          affecter_VDDh (p2->num_port) ; // appel récursif
          p2=p2->suiv ;
        Fait
     Fait
   Fin si
 Fin
```

La fonction affecter_VthNL (int num) :

// Affecter VthNL aux transistors N de la porte num
Algorithme III.5://affectation de V_{thNL}
Début
NOEUD1 *p1 ;
p1=chercher (num) // chercher un nœud de type NOEUD1 dans p_list2
p1->VthN=VthNL ;
Appeler la fonction libérer avec liberer (num) pour libérer le nœud de la liste list qui comporte numéro num.
Fin

La fonction affecter_VthPH (int num) :

// Affecter VthPH aux transistors P de la porte num
Algorithme III.6: // Affectation de V_{thPH}

Début
NOEUD1 *p1 ;
Appeler la fonction chercher avec p1=chercher (num) pour chercher un nœud de type NOEUD1 dans p_list2
p1->VTHP=VthPH ;
Appeler la fonction libérer avec liberer (num) pour libérer le nœud de la liste list qui comporte numéro num.
Fin

Le programme principal :
Algorithme III.7: //Programme principal de la détermination de la solution discrète
Début
Entrée : le fichier comportement
Les constantes : VDDL= 1, VDDH =3.3, VTHNL=0.1, VTHNH=0.3,
VTHPL= -0.3 et VTHPH= -0.1
Tfixé

```
NOEUD1 *pt ;
TABLEAU *ptr ;
int fin=0 ;
lecture () ; //construire la liste p_list2
pt=p_list2->tete ;
Tant que (pt)
 Faire
  pt->VDD=VDDL ;
  pt->VTHN=VTHNH ;
  pt->VTHP=VTHPL ;
 Fait
déterminer_Constantes() ;
déterminer_Délai() ;
```

```
déterminer_TCR() ;
Si (Tcr ≤ Tfixé) alors déterminer_Puissance() ;
Si non
 Faire
  Déterminer_Solution_Continue()
  // Affectation de VDD aux portes logiques du circuit
 Tant que (non fin)
   Faire
    Affecter les valeurs VDD, VTHN et VTHP de la solution continue à la liste p_list2 ;
    Construire une liste de type TABLEAU contenant les valeurs continues de VDD ;
    calculer_max () ; // déterminer les portes dont VDD est maximale
    créer_list_port () ; // créer une liste de portes dont VDD est maximale
    Affecter-VDDh () ; // affecter VDDH aux portes de list_port
    Affecter VDDL aux autres portes() ; // affecter VDDL aux portes non insérées
                                                                    dans list_port
    déterminer_Délai() ;
    Tcr=déterminer_TCR() ;
    Si (Tcr ≤ Tfixé) alors
      Faire
       fin=1 ;
       liberer_listv ()  // libération de l'espace-mémoire alloué pour list
      Fait
    Si non
      Faire
       Si (list non vide) alors
         Faire
           Déterminer_Solution_Continue() ; // pour les portes dont l'affectation de VDD
                                                                    n'a pas été faite
         Fait
       Si non //liste vide
         Faire
          Afficher "Erreur : contrainte de temps non satisfaite" ; exit ;
         Fait
       Fin si
     Fait
Fait

// Affectation de VTHN aux transistors N du circuit
fin=0 ;
 Tant que (non fin)
   Faire
    Construire une liste de type TABLEAU contenant les valeurs continues de VTHN
    calculer_min () ; //  déterminer les portes dont les transistors N ont la tension de
                                                                    seuil la plus faible
    créer_list_port () // créer une liste de portes dont les transistors N ont la plus
                                                                    faible valeur de VTHN
    Affecter_VTHNL() ; //  affecter VTHNL aux transistors N des portes de list_port
    Affecter VTHNh aux autres portes() ; // affecter VTHNH aux transistors N des
                                                    portes non insérées  dans list_port
```

```
    déterminer_Délai() ;
    Tcr=déterminer_TCR() ;

  Si (Tcr ≤ Tfixé) alors
      Faire
       fin=1 ;
       liberer_listv () ;  // libération de l'espace-mémoire alloué pour list
      Fait
   Si non
     Faire
      Si (list non vide) alors
        Déterminer_Solution_Continue() ; // pour les portes dont l'affectation de
                                          VTHN n'a pas été faite pour leurs transistors N
        Fait
       Si non //liste vide
        Faire
         Afficher "Erreur : contrainte de temps non satisfaite" ; exit ;
        Fait
       Fin si
     Fait
Fait

// Affectation de VTHP aux transistors P du circuit
fin=0
 Tant que (non fin)
   Faire
    Construire list du type TABLEAU contenant les valeurs continues de VTHP
    calculer_max () ; //  déterminer les portes dont les transistors P ont la tension de
                                                              seuil la plus grande
    créer_list_port () ; // créer une liste de portes dont les transistors P ont la plus
                                                          grande valeur de VTHP
    affecter-VTHPH () ;  //   affecter VTHPH aux transistors P des portes de list_port
    Affecter VTHPL aux autres portes() ; // affecter VTHPL aux transistors P des
                                                  portes non insérées  dans list_port
    déterminer_Délai() ;
    Tcr=déterminer_TCR() ;
  Si (Tcr ≤ Tfixé) alors
     Faire
      fin=1 ;
      liberer_listv ()   // libération de l'espace-mémoire alloué pour list
     Fait
  Si non
    Faire
     Si (list non vide) alors
      Faire
       determiner_Solution_Continue() ; // pour les portes dont l'affectation de
                                         VTHP n'a pas été faite pour leurs transistors P
      Fait
```

```
 Si non //liste vide
        Faire
         Afficher "Erreur : contrainte de temps non satisfaite" ; exit ;
        Fait
       Fin si
     Fait
 Fait
 déterminer_Puissance() ;
Fait
Fin si
```

Sortie : puissance du circuit, les valeurs de V_{DD} pour les portes logiques, et les valeurs de V_{THN} et V_{THP} pour tous les transistors du circuit.
Fin

III.10. Exemple d'illustration

Les fichiers de données :

Fichier circuit contient le comportement des portes du circuit

```
a b c % s1 % NOT ( a OR b OR c ) % 40. %
a b c d e % s5 % NOT ( a AND ( b OR ( c AND d ) OR e ) ) % 40. %
s6 k e f % s7 % NOT ( s6 AND ( k OR f ) OR e ) % 40. %
s10 e % s6 % NOT ( s10 AND e ) % 40. %
a b c e f % s2 % NOT ( a AND b AND c AND ( e OR f ) ) % 40. %
s2 s1 a % s10 % NOT ( s1 AND s2 AND a ) % 40. %
s5 b % s9 % NOT ( s5 OR b ) % 40. %
s7 a b % s12 % NOT ( s8 AND a AND b ) % 40. %
s2 b c d % s13 % NOT ( s3 OR ( b AND c ) OR d ) % 40. %
s9 s7 % s11 % NOT ( s9 AND s7 ) % 40. %
```

L'un des fichiers déduit de SPOT ([3], [4], [5]) contient les numéros des portes, leur niveau, ainsi que leurs successeurs :

```
 level of gate 5
1

successors of gate 5
num= 9
num= 6

 level of gate 2
1

 successors of gate 2
num= 7
```

```
 level of gate 1
1

 successors of gate 1
num= 6

 level of gate 9
2

 successors of gate 9

 level of gate 7
2

 successors of gate 7
num= 10

 level of gate 6
2

 successors of gate 6
num= 4

 level of gate 4
3

 successors of gate 4
num= 3

 level of gate 3
4

 successors of gate 3
num= 8
num= 10

 level of gate 10
5

 successors of gate 10

 level of gate 8
5

 successors of gate 8
```

Fichier CONSTANTES : Il contient les valeurs de constantes suivantes

1 : 1 6 3 4
2 : 3 8 3 8
3 : 2 7 3 6
4 : 2 3 1 4
5 : 4 7 2 9
6 : 3 4 1 6
7 : 1 4 2 3
8 : 3 4 1 6
9 : 2 7 3 6
10 : 2 3 1 4

Fichier des résultats partiels :

Fichier TENSIONS : il contient les valeurs des différentes tensions affectées aux différentes portes du circuit.

V_{DD} (V)	V_{THN} (V)	V_{THP} (V)
1.000000e+00	3.000000e-01	-3.000000e-01
3.300000e+00	3.000000e-01	-1.000000e-01
1.000000e+00	3.000000e-01	-1.000000e-01
1.000000e+00	3.000000e-01	-1.000000e-01
1.000000e+00	3.000000e-01	-1.000000e-01
1.000000e+00	3.000000e-01	-1.000000e-01
3.300000e+00	3.000000e-01	-1.000000e-01
1.000000e+00	1.000000e-01	-1.000000e-01
1.000000e+00	3.000000e-01	-1.000000e-01
1.000000e+00	3.000000e-01	-1.000000e-01

Fichier circuit_M

C'est le fichier décrivant le comportement des portes avec leur délai. Notons que ces délais sont obtenus suite à l'affectation des différentes tensions d'alimentation et de seuil, et que ce fichier est une donnée pour le module *estimation de la consommation de puissance* (SPOT).

3.3
32.
static1
a b c % s1 % NOT (a OR b OR c) % 40. % 20.268854 %
a b c d e % s5 % NOT (a AND (b OR (c AND d) OR e)) % 40. % 3.135170 %
s6 k e f % s7 % NOT (s6 AND (k OR f) OR e) % 40. % 11.181246 %
s10 e % s6 % NOT (s10 AND e) % 40. % 2.981666 %
a b c e f % s2 % NOT (a AND b AND c AND (e OR f)) % 40. % 9.690413 %
s2 s1 a % s10 % NOT (s1 AND s2 AND a) % 40. % 3.727082 %
s5 b % s9 % NOT (s5 OR b) % 40. % 1.219233 %
s7 a b % s12 % NOT (s8 AND a AND b) % 40. % 3.727082 %
s2 b c d % s13 % NOT (s3 OR (b AND c) OR d) % 40. % 11.181246 %
s9 s7 % s11 % NOT (s9 AND s7) % 40. % 2.981666 # 1

Fichier temps critique

41.885929
1 6 4 3 8

Fichier PUISSANCE contient la consommation de puissance du circuit

```
*****************************************************
    Sw (W)          Leak (W)          Tot (W)
*****************************************************
2.282262e-05   1.884803e-05   4.167066e-05
```

III.11. Conclusion

Nous venons de présenter dans ce chapitre l'heuristique que nous avions développée pour générer une solution discrète à partir de la solution continue, et ce, pour la minimisation de la consommation de puissance d'un circuit intégré sous la contrainte de son temps de réponse. Par ailleurs, nous avons vu que le développement de cette heuristique se justifie par un problème technologique rendant difficile, voire impossible la fabrication d'un circuit intégré fonctionnant sous plusieurs tensions d'alimentation et présentant plusieurs tensions de seuil pour ses transistors. Enfin, nous avons vu que la transformation de la solution continue en une solution discrète pour le problème concerné n'est pas de complexité polynomiale, d'où le recours à une heuristique.

CHAPITRE IV

Résultats

IV.1. Introduction

Dans les chapitres précédents, des exemples ont été donnés en fonction du problème traité. Dans ce présent chapitre, nous présentons et commentons les résultats obtenus par l'heuristique développée (en fait, il s'agit de six heuristiques basées sur la même stratégie, mais affectant les différentes valeurs de tensions selon des ordres différents).

Notons que l'algorithme présenté au troisième chapitre affecte les valeurs discrètes aux tensions d'alimentation et aux tensions de seuil dans l'ordre suivant :
- les valeurs discrètes des tensions d'alimentation des portes logiques du circuit (V_{dd})
- les valeurs discrètes des tensions de seuil des transistors du type N de chaque porte logique du circuit (V_{thN}).
- les valeurs discrètes des tensions de seuil des transistors du type P de chaque porte logique du circuit (V_{thP}).

Toutefois, l'implémentation (en langage C) du même algorithme tient compte des six combinaisons possibles, et ce, afin d'obtenir plusieurs résultats possibles, et en tirer des conclusions. Ces six différentes combinaisons possibles sont les suivantes :

- Affectation de V_{dd}, puis celle de V_{thN} , suivies par l'affectation de V_{thP} (cette combinaison correspond à H1 dans les tableaux des résultats)
- Affectation de V_{dd}, puis celle de V_{thP} , suivies par l'affectation de V_{thN} (cette combinaison correspond à H2 dans les tableaux des résultats)
- Affectation de V_{thN}, puis celle de V_{dd} , suivies par l'affectation de V_{thP} (cette combinaison correspond à H3 dans les tableaux des résultats)
- Affectation de V_{thN}, puis celle de V_{thP} , suivies par l'affectation de V_{dd} (cette combinaison correspond à H4 dans les tableaux des résultats)
- Affectation de V_{thP}, puis celle de V_{dd} , suivies par l'affectation de V_{thN} (cette combinaison correspond à H5 dans les tableaux des résultats)
- Affectation de V_{thP}, puis celle de V_{thN} , suivies par l'affectation de V_{dd} (cette combinaison correspond à H6 dans les tableaux des résultats)

Notons que les quatre circuits Cir1, Cir2, Cir3 et Cir4 utilisés pour l'obtention des résultats ont respectivement cinq, dix, quinze et vingt portes logiques.

Le premier tableau donne les différents temps critiques obtenus avec les différentes combinaisons, et ce, pour les quatre circuits considérés. Par exemple, pour le premier circuit, l'utilisation de la première heuristique (H1) donne un temps critique égal à 3.92 ns. De ce fait, toute valeur de $T_{fixé}$ (qui est un temps fixé par le concepteur) inférieure à 3.92 ns, entraînerait une violation de la contrainte portant sur le temps de réponse du premier circuit lorsque H1 est utilisée.

Dans le cas où il n'existe pas de contrainte sur le temps de réponse du circuit, la puissance totale minimale serait celle obtenue avec l'affectation de V_{ddL} à toutes les portes logiques du circuit, V_{thNH} à tous les transistors NMOS, et V_{thPL} à tous les transistors PMOS du circuit (voir dans le chapitre III les équations relatives aux consommations dynamique et statique de la puissance dans un circuit CMOS). Cependant, à cette consommation minimale, correspondrait un temps de réponse du

circuit qui ne pourrait pas être intéressant (T_{crMax}). Les temps critiques obtenus dans ce cas sont alors les suivants :
Cir1 : 30.40 ns
Cir2 : 74.31 ns
Cir3 : 59.11 ns
Cir4 : 116.54 ns

En tenant compte de l'affectation des différentes tensions *discrètes* à partir de la solution *continue*, nous avions déterminé les valeurs minimales des délais critiques des circuits, délais en dessous desquels les valeurs de $T_{fixé}$ entraîneraient une violation de la contrainte *temps*. Ces valeurs critiques sont indiquées dans le tableau suivant :

Tableau IV1 : Valeurs critiques de temps de réponse de circuits

	Cir1 T_{cr}**(ns)**	**Cir2** T_{cr}**(ns)**	**Cir3** T_{cr}**(ns)**	**Cir4** T_{cr}**(ns)**
H1	3.92	9.60	6.97	13.65
H2	3.92	9.60	6.97	13.65
H3	10.95	20.29	16.72	39.68
H4	10.95	20.29	16.69	39.53
H5	7.78	16.12	14.04	24.73
H6	7.78	16.12	14.04	24.73

Notons que le temps de réponse d'une porte est estimé à l'aide de l'équation donnée au deuxième chapitre. De ce fait, en plus des paramètres impliqués dans cette équation, le temps de réponse d'un circuit dépend d'autres paramètres, dont le nombre de portes logiques, le nombre de portes en cascade, les délais des interconnexions,....
Pour notre part, nous avions testé notre programme pour différentes valeurs de $T_{fixé}$ (inférieur, supérieur ou égal à T_{cr}) pour un circuit donné. Nous donnons ci-après les résultats de la consommation de la puissance obtenus pour différentes valeurs de $T_{fixé}$ en considérant les différentes combinaisons possibles, et ce, pour chaque circuit. Ainsi, le tableau2 indique que la consommation totale de la puissance est de **22.52703 µW** (19.06548 µW pour la consommation dynamique et 3.46155 µW pour celle due aux fuites du courant) pour le premier circuit, avec un temps de réponse du circuit égal à 10 ns. Par contre, cette consommation est de **3.14275µW** pour un temps de réponse égal à 35 ns (notons le compromis temps de réponse – consommation de puissance). Dans le tableau4, le problème n'a pas de solution lorsque la valeur du temps fixée par l'utilisateur est de 10 ns pour le premier circuit, et lorsque l'heuristique H3 est utilisée. Ceci est dû est au fait que $T_{fixé}$ = 10 ns est inférieur à T_{cr} = 10.95 ns (voir case H3- cir1 dans le premier tableau). Enfin, notons que les résultats indiqués dans les tableaux ci-dessous ont été obtenus avec les valeurs des tensions suivantes :

- V_{ddL}= 1 V
- V_{ddH}= 3.3V
- V_{thNL}= 0.1V
- V_{thNH}= 0.3V
- V_{thPL}= - 0.3V
- V_{thPH}= - 0.1V

Tableau IV .2 : les résultats du programme H1

H1	T fixé (ns)			Type	Puissance (µWatts)		
Cir1 5 portes	10	25	35	Sw	19.06548	15.37171	3.08175
				Leak	3.46155	3.43349	0.06100
				Tot	**22.52703**	**18.80520**	**3.14275**
Cir2 10 portes	21	50	75	Sw	32.77279	16.51805	3.00944
				Leak	10.84480	18.84803	0.20131
				Tot	**43.61759**	**35.36608**	**3.21075**
Cir3 15 portes	25	30	60	Sw	33.46007	29.47581	7.62607
				Leak	17.86480	20.38186	0.18911
				Tot	**51.32487**	**49.85767**	**7.81518**
Cir4 20 portes	30	67	117	Sw	97.46159	44.02372	8.94964
				Leak	7.59079	16.18641	0.37212
				Tot	**105.05238**	**60.21013**	**9.32176**

Tableau IV .3 : les résultats du programme H2

H2	T fixé (ns)			Type	Puissance (µWatts)		
Cir1 5 portes	10	25	35	Sw	19.06548	15.37171	3.08175
				Leak	3.46155	3.43349	0.06100
				Tot	**22.52703**	**18.80520**	**3.14275**
Cir2 10 portes	21	50	75	Sw	32.77279	16.51805	3.00944
				Leak	10.84480	18.84803	0.20131
				Tot	**43.61759**	**35.36608**	**3.21075**
Cir3 15 portes	25	30	60	Sw	33.46007	29.47581	7.62607
				Leak	17.86480	20.38186	0.18911
				Tot	**51.32487**	**49.85767**	**7.81518**
Cir4 20 portes	30	67	117	Sw	97.46159	44.02372	8.94964
				Leak	7.59079	16.18641	0.37212
				Tot	**105.05238**	**60.21013**	**9.32176**

Tableau IV.4 : les résultats du programme H3

H3	T fixé (ns)			Type	Puissance (µWatts)		
Cir1 5 portes	10	25	35	Sw	Erreur T fixé	15.37171	3.08175
				Leak		3.43349	0.06100
				Tot		**18.80520**	**3.14275**
Cir2 10 portes	21	50	75	Sw	32.77279	16.51805	3.00944
				Leak	10.84480	18.84803	0.20131
				Tot	**43.61759**	**35.36608**	**3.21075**
Cir3 15 portes	25	30	60	Sw	33.46007	29.47581	7.62607
				Leak	17.86480	20.38186	0.18911
				Tot	**51.32487**	**49.85767**	**7.81518**
Cir4 20 portes	30	67	117	Sw	Erreur T fixé	44.02372	8.94964
				Leak		16.18641	0.37212
				Tot		**60.21013**	**9.32176**

Tableau IV.5 : les résultats du programme H4

H4	T fixé (ns)			Type	Puissance (µWatts)		
Cir1 5 portes	10	25	35	Sw	Erreur T fixé	15.37171	3.08175
				Leak		3.43349	0.06100
				Tot		**18.80520**	**3.14275**
Cir2 10 portes	21	50	75	Sw	37.54827	45.34415	3.00944
				Leak	25.91210	3.66513	0.20131
				Tot	**63.46037**	**49.00928**	**3.21075**
Cir3 15 portes	25	30	60	Sw	64.84086	41.32895	7.62607
				Leak	4.55734	1.95610	0.18911
				Tot	**69.39820**	**43.28504**	**7.81518**
Cir4 20 portes	30	67	117	Sw	Erreur T fixé	85.21817	8.94964
				Leak		7.52636	0.37212
				Tot		**92.74453**	**9.32176**

Tableau IV.6 : les résultats du programme H5

H5	T fixé (ns)			Type	Puissance (µWatts)		
Cir1 5 portes	10	25	35	Sw	11.83186	15.37171	3.08175
				Leak	10.06622	3.43349	0.06100
				Tot	**21.89808**	**18.80520**	**3.14275**
Cir2 10 portes	21	50	75	Sw	37.54827	45.34415	3.00944
				Leak	25.91210	3.66513	0.21311
				Tot	**63.46037**	**49.00928**	**3.22255**
Cir3 15 portes	25	30	60	Sw	64.84086	41.32895	7.62607
				Leak	4.55734	1.95610	1.89110
				Tot	**69.39820**	**43.28504**	**9.51717**
Cir4 20 portes	30	67	117	Sw	84.62245	85.21817	8.94964
				Leak	70.61567	7.52063	0.37212
				Tot	**155.23812**	**92.73880**	**9.32176**

Tableau IV.7 : les résultats du programme H6

H6	T fixé (ns)			Type	Puissance (µWatts)		
Cir1 5 portes	10	25	35	Sw	11.83186	15.37171	3.08175
				Leak	10.06622	3.43349	0.06100
				Tot	**21.89809**	**18.80520**	**3.14275**
Cir2 10 portes	21	50	75	Sw	37.54827	45.34415	3.00944
				Leak	25.91210	3.66513	0.20131
				Tot	**63.46037**	**49.00928**	**3.21075**
Cir3 15 portes	25	30	60	Sw	64.84086	41.32895	7.62607
				Leak	4.55734	1.95610	0.18911
				Tot	**69.39820**	**43.28504**	**7.81518**
Cir4 20 portes	30	67	117	Sw	84.62245	85.21817	8.94964
				Leak	70.61567	7.52063	0.37212
				Tot	**155.23810**	**92.73880**	**9.32176**

IV.2. Commentaires sur les résultats

1) Quand on augmente la valeur de $T_{fixé}$ (condition portée sur le temps de réponse du circuit) la consommation de la puissance totale diminue à cause du relâchement sur la contrainte de temps. Formellement, on sait qu'une relaxation d'un problème d'optimisation conduit à une solution plus optimale que celle obtenue à partir du problème original. On remarque que les résultats obtenus suivent cette règle.

2) Quand $T_{fixé}$ est supérieur à T_{crMax}, le temps critique du circuit (T_{cr}) et la consommation de la puissance totale deviennent stables pour les six heuristiques.

3) Selon les $T_{fixés}$ calculés précédemment (voir Tableau 1), on remarque que les programmes H1 et H2 donnent de meilleurs résultats que ceux obtenus par les programmes H5 et H6 qui donnent eux-mêmes de meilleurs résultats que les programmes H3 et H4. Ceci est dû aux relations de calcul de la consommation de la puissance (chapitre précédent) qui, avec les heuristiques H1 et H2, satisfont rapidement la contrainte du temps et donc permettent d'affecter un plus grand nombre de valeurs V_{ddL}, V_{thNH} et V_{thPL} aux différentes tensions, ce qui a pour conséquence de réduire la consommation de la puissance totale.

4) On remarque que les programmes H1 et H2 donnent les mêmes valeurs de consommation de puissance pour le même circuit et pour chaque $T_{fixé}$. Il en est de même pour H5 et H6.

5) Quand la consommation de la puissance totale du circuit diminue, les causes peuvent être :
-la réduction de la consommation dynamique
-la réduction de la consommation de puissance due aux fuites du courant
-ou simultanément les deux cas cités ci-dessus

6) Les temps CPU utilisés pour l'exécution de nos programmes sont négligeables (0 seconde). Notons que Linux permet d'obtenir des valeurs de temps CPU exprimées en millisecondes en utilisant la procédure ci-dessous :
cc –p nom de fichier → obtention des fichiers *a.out* et *mon.out*
prof a.out → obtention des temps CPU exprimés en ms pour l'exécution des différentes fonctions et procédures
Toutefois, quoique les différents fichiers nécessaires pour connaître le profil (*profile*) de nos programmes ont été obtenus normalement, la commande *prof* n'a pas fonctionné normalement sur la plateforme que nous avons utilisée. De ce fait, tous les temps CPUs sont exprimés en secondes.

IV.3. Conclusion

Nous avons présenté dans ce chapitre six heuristiques permettant de transformer chacune, une solution continue en une solution discrète et avions justifié au préalable la nécessité de recourir à une (des) heuristique(s). De même, nous avons vu comment on peut obtenir un bon compromis entre la consommation de la puissance totale et le temps de réponse d'un circuit. Enfin, les temps CPU obtenus montrent que nos programmes sont efficaces.

Conclusion générale

Avec la prolifération des systèmes portables (PCs, téléphones, PDAs, etc ....), la durée de vie limitée des batteries et la possibilité de concevoir des systèmes monopuce, la conception de circuits intégrés à faible consommation de puissance est devenue plus que nécessaire. Ceci, afin d'augmenter l'autonomie du système utilisé, mais aussi pour réduire sa température, sans quoi, un disfonctionnement du système serait engendré.

Après avoir donné des définitions et fait quelques rappels jugés nécessaires pour une meilleure compréhension de ce mémoire, nous avions vu que la consommation de la puissance est due aux courts circuits, aux fuites de courant dans les transistors et à l'activité dynamique du circuit. Du fait que la consommation due aux courts circuits peut être réduite avec l'utilisation de la technologie CMOS, nous nous sommes intéressés aux deux autres composantes de la consommation de puissance. Nous avions alors vu que la consommation dynamique peut être minimisée en affectant de faibles valeurs d'alimentation, mais que ce serait au détriment du temps de réponse du circuit. La consommation due aux fuites du courant peut être réduite elle, en augmentant (en diminuant) les tensions de seuil des transistors NMOS (PMOS), mais ce serait aussi au détriment du temps de réponse du circuit. De ce fait, nous avions vu que ce problème pouvait être formulé par un problème d'optimisation combinatoire à optimiser sous la contrainte temps de réponse du circuit. Ce problème étant résolu, nous avions vu que la solution continue obtenue posait toujours un problème, à savoir qu'il est coûteux, difficile, voire impossible de fabriquer un circuit fonctionnant sous plusieurs valeurs de tensions. Afin de contourner cette problématique, on a alors recours à l'utilisation de deux bornes de valeurs pour les tensions d'alimentation et de seuil des transistors NMOS et PMOS.

Après avoir montré que la transformation de la solution continue en une solution discrète n'est pas polynomiale, nous avions présenté nos six heuristiques. Celles-ci, quoique basées sur la même stratégie, opèrent sur des ordres d'affectation de tensions différents, et génèrent des résultats différents. Du fait que la réduction de la consommation de la puissance est faite sous la contrainte *temps*, il était nécessaire d'estimer le temps de réponse du circuit. A cet effet, nous avions alors présenté une technique d'estimation de ce temps en calculant des constantes qui servent pour le calcul des délais des portes logiques du circuit, ces derniers permettant eux-mêmes la détermination du délai critique dans un circuit.

Enfin, nous avions vu que les temps CPU obtenus pour les différents jeux d'essais présentés étaient intéressants, ce qui montre l'efficacité de nos méthodes.

Nous terminons en soulignant que ce mémoire nous a permis d'aborder certains aspects nouveaux pour nous, tels que la conception des circuits intégrés, et nous a permis en même temps de concrétiser différentes notions acquises au cours de notre cursus universitaire.

Bibliographie

[1] N. Weste & K. Eshraghian "Principles of CMOS VLSI design: A systems perspective" Reading, MA Addison Wesley, 1993.

[2] F. Droesbeke, M. Hallin et Ci. Lefebvre "Les graphes par l'exemple" ELLIPSE 2001.

[3] A. Mahdoum "SPOT: A tool for estimating the switching power dissipation in CMOS circuits and data paths" 1-2 Dec.97, Osaka, Japan.

[4] A. Mahdoum "SPOT: Un outil à base d'un algorithme génétique pour estimer la consommation maximale de la puissance dynamique des circuits CMOS" CSCA'99, 8-9 Nov. 99, Hôtel Sheraton, Alger, Algérie.

[5] A. Mahdoum "SPOT: A Tool for estimating the maximal and the average switching power dissipation in CMOS circuits" Designers's Forum Proceedings, DATE'02, 4-8 March 2002, Palais des Congrès, Paris, France.

[6] A. Mahdoum, A. Boutammine, D. Touahri et N. Toubaline "FREEZER1 : un outil d'aide à la conception de circuits digitaux à faible consommation de puissance" IEEE/FTFC'05, 18-20 Mai 2005, Paris, France.

[7] M.Sakarovitch " Optimisation combinatoire, Méthodes mathématiques et algorithmiques : programmation discrète " HERMANN, 1984.

[8] I.Charon, A.Germa et O.hudry "Méthodes d'optimisation combinatoire" MASSON, 1996.

[9] D.Galland "Le langage C : pratique et environnement" ,DUNOD 1990.

[10] B.W.Kernighan D.M.Ritche " Le langage C", MASSON 1983.

[11] P.Guitton "Estimation et Optimisation de la consommation lors de la conception globale des systèmes autonomes ",Université de Nice-Sophia Antipolis-UFR Sciences, thèse de doctorat octobre 2004.

[12] " vi & linux" quick references guide UNIX.

[13] T.Mahnke & all, "Power Optimization through dual supply voltage scaling using power compiler",European Synopsys Users Group Meeting,2002.

[14] Q.Wang & S.B.K. Vrudhula, "Static Power Optimization of Deep Submicron CMOS Circuits dor dual Vt Technology ", in Proc,IEEE ICCAD,pp 490-496,1998.

[15] S.Sirichotiyakul & all, "Stand-by Power minimization through simultaneous threshold voltage selection and circuit sizing ",in Proc.DAC,pp 436-441,1999.

[16] K.Usami & all, "Automated Low-Power Technique Exploiting Multiple Supply voltages applied to a Media Processor ",IEEE Journal SSC, vol.33, n° 3, pp 463-472,1998.

[17] "Résolution de problèmes difficiles: algorithmes d'approximation, algorithmes probabilistes, heuristique et métaheuristiques". (http://www.lacim.uqam.ca/~chauve/Enseignement/INF7440/H05/HEURISTIQUES-PROBA/GUY-approximations-heuristiques.pdf)

[18] C.Piguet, S.Cserveny, J.F.Perotto et J.M.asgonty "Techniques de circuits et méthodes de conception pour réduire la consommation statique dans les technologies profondément sub-microniques", IEEE/FTFC'03, 14-16 Mai 2003, Paris, France. (http://r2d2.enssat.fr/conferences/ftfc2003/papers/2.pdf)

[19] S.Pillement, R.David et O.Sentieys "Architectures reconfigurables :opportunités pour la faible consommation" IEEE/FTFC'03, 14-16 Mai 2003, Paris, France. (http://r2d2.enssat.fr/conferences/ftfc2003/papers/21.pdf)

[20] M.V.Caneghem "Le voyageur de commerce: Algorithme branch and bound , Algorithme Glouton, Méthodes de recherche locale", Décembre 2003. (http://www.lif-sud.univ-mrs.fr/~vancan/mait/documents/cours4_8.pdf)

[21] S.Gailhard, N.Julien et E.Martin "Intégration de technologies d'optimisation faible consommation dans l'outil de synthese architecturale GAUT_W" AA'98, Saclay, 29-30 Janvier 1998. (http://lester.univ-ubs.fr:8080/publications/Lowpower/aaa.pdf)

[22] A.Mechti & D.Melzi "Elaboration d'une commande de transfert de données dans un circuit" Mémoire d'ingénieur d'Etat en informatique, Université Saad Dahleb, Blida, promotion 2003/2004.