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
d'un diplôme d'ingénieur d'état en informatique.
Option : Système d'information

Sujet :

Codification de données pour réduire la consommation de puissance dans un bus

Présenté par : LOUZRI Mohamed
SAADAOUI Messaoud

Promoteur : Mr.A. MAHDOUM
Encadreur : Mr.A. MAHDOUM

Organisme d'accueil : Centre de Développement des Technologies Avancées.

Soutenue le: 26/09/2007, devant le jury composé de :

Nom. M. MEHIEDDINE, C. C, USDB	**Président**
Nom CHERIF ZAHAR, C. C, USDB	**Examinateur**

- promotion 2006/2007 -

Remerciements

*Nous tenons à remercie **allah** le tout puissant de nous avoir guider a mener bien se modeste travail.*

*Nous exprimons nos sincères remerciement a **Mr H. BESSALEH** directeur du **CDTA** de nous avoir accepter de son établissement.*

*Nos remerciement et nos profondes gratitudes a notre promoteur **Mr A. MAHDOUM** de nous avoir proposer ce sujet, de nous avoir encadrer, pour ses remarques et ses conseils judicieux qu'il nous a octroyé le long de notre travail.*

Nous remercions l'équipe du laboratoire microélectronique du centre de Développent des Technologies Avancées.

Nos remerciements s'adressent également au président et membres du jury d'avoir accepter d'évaluer notre travail.

Nous sommes reconnaissant envers tous ceux qui ont contribués de prés ou de loin à la réalisation de ce projet.

Nous remercions également tous les enseignants qui ont contribués à notre formation depuis le jeune age et qu'ils trouvent dans ce mémoire le brin de ce qu'ils nous ont appris.

En fin, nos remerciements envers tous nos amis qui contribué de près ou de loin à la réalisation de ce projet.

Encore une fois merci à tous.

SOMMAIRE

Sommaire des figures.....1

Sommaire des tableaux.....2

RESUME.....3

INTRODUCTION GENERALE.....4

Chapitre 1 : RAPPELS ET DEFINITIONS.....6

1.1. Introduction.....6

1.2. Définitions et rappels.....6

1.2.1. Portes logiques.....6

1.2.2. Interconnexions.....8

1.3. Conclusion.....8

Chapitre 2 : PROBLEMATIQUE ET ETAT DE L'ART.....9

2.1. Introduction.....9

2.2. Problématique.....9

2.3. Etat de l'art.....10

2.3.1. Codage avec affectation de probabilités fixes.....12

2.3.2. Combinaison de codage avec affectation de probabilités fixes et retard de transfert de données.....14

2.3.3. Codage avec affectation de probabilités variables.....15

2.4. Conclusion.....15

Chapitre 3 : TECHNIQUES DEVLOPPEES.....16

3.1. Introduction.....16

3.2. Présentation de la méthodologie générale.....16

3.3. Méthode 1 : Codification Sans Probabilité.....19

3.3.1. L'algorithme de la méthode.....19

3.3.2. Description des structures de données.....21

3.3.3. Description des fichiers utilisés.....22

3.3.4. Les fonctions de cette méthode.....23

3.4. Méthode 2 : Codification Avec Probabilité Fixe....35
3.4.1. L'algorithme de la méthode....36
3.4.2. Description des structures de données....38
3.4.3. Description des fichiers utilisés....39
3.4.4. Les fonctions de cette méthode....39

3.5. Méthode 3 : Codification Avec Probabilité Variable....42
3.5.1. L'algorithme de la méthode....43
3.5.2. Description des structures de données....45
3.5.3. Description des fichiers utilisés....45
3.5.4. Les fonctions de cette méthode....45

3.6. Méthode 4 : Notre méthode....46
3.6.1. L'algorithme de la méthode....48
3.6.2. Description des structures de données....50
3.6.3. Description des fichiers utilisés....52
3.6.4. Les fonctions de cette méthode....52

3.7. Conclusion....59

Chapitre 4 : Tests RESULTATS....60
4.1. Introduction....60
4.2. Résultats obtenus....60
4.3. Conclusion....65

CONCLUSION GENERALE....66

BIBLIOGRAPHIE....67

Sommaire des figures

Chapitre 1 : RAPPELS ET DEFINITIONS

Fig.1.1. Transistor NMOS....6
Fig.1.2. Transistor PMOS....6
Fig.1.3. Inverseur CMOS....7
Fig.1.4. porte logique CMOS implémentant une fonction complexe....7
Fig.1.5. capacité intrinsèque et de couplage dans des interconnexions....8

Chapitre 2 : PROBLEMATIQUE ET ETAT DE L'ART

Fig.2.1. Codage et de décodage de données en vue de réduire l'énergie induite par les capacités parallèles des fils d'un bus....11
Fig.2.2. Codage et décodage de données avec l'utilisation de probabilités fixes en vue de réduire l'énergie due aux capacités parallèles....14

Chapitre 3 : TECHNIQUES DEVLOPPEES

Fig.3.1. Le schéma synoptique de la méthodologie générale adoptée....17
Fig.3.2. La structure nœud....21
Fig.3.3. La structure RESULTAT....21
Fig.3.4. Format d'un enregistrement du fichier cost_matrix....22
Fig.3.5. Format du fichier cost_matrix....23
Fig.3.6. Format du fichier résultat....23
Fig.3.7. La structure RECORD....38
Fig.3.8. Format du fichier INFO_COD....39
Fig.3.9. Format de la structure Node....51
Fig.3.10. Format de la structure Noeud1....52

Chapitre 4 : RESULTATS

Fig.4.1. La représentation graphique de l'énergie dynamique consommée....64
Fig.4.2. La représentation graphique de l'énergie parasite consommée....64
Fig.4.3. La représentation graphique de l'énergie totale consommée....65

Sommaire des tableaux

Chapitre 2 : PROBLEMATIQUE ET ETAT DE L'ART

Table 2.1. Valeurs de E_{par}/C_L en considérant toutes les paires de données possibles transmises successivement sur un bus de taille 3 bits.....13

Chapitre 4 : RESULTATS

Table 4.1.Energie dissipée pour chacune des quatre techniques avec n = 8 bits et m variant de n+1 bits jusqu'à n+9 bits.....61

Table 4.2. Résultats de comparaison (pourcentage de Gain d'énergie) entre les différentes méthodes et la méthode sans codification.....63

RESUME

Du fait du développement considérable de la technologie des circuits intégrés, l'intégration de plusieurs systèmes sur une même puce est devenue possible, donnant naissance à ce qu'on appelle les systèmes mono puce (*System On Chip* ou *SOC*). Toutefois, cette intégration engendre de nombreux problèmes, obligeant ainsi à mettre à jour de nombreux outils de CAO des circuits VLSI développés par le passé, voire de les refaire.

Parmi ces problèmes, le problème de couplage qui consiste à augmenter les capacités parasites sur un bus a un impact important sur les caractéristiques du système (forte consommation de puissance, transfert de données moins rapide) du fait que les distances entre les fils d'un bus sont de plus en plus réduites. La dissipation de puissance sur un bus est due à deux facteurs : la consommation dynamique qui résulte de la charge et de la décharge des capacités du bus et celle qui est due aux capacités parallèles présentes entre les fils voisins du bus. Les travaux de recherche actuels montrent qu'avec le développement technologique, le deuxième type de consommation de puissance est devenu plus important par rapport au premier. En effet, les capacités parallèles sont devenues respectivement 6 et 8 fois plus importantes que les capacités intrinsèques du bus avec les technologies CMOS 0.18 µm et 0.13 µm. Ceci montre l'intérêt à porter pour le deuxième type de consommation de puissance.

Alors que les capacités intrinsèques sont fixes pour un bus donné, celles qui se forment entre les fils voisins du bus varient par contre en fonction des données qui y transitent. Certaines données engendrant de plus faibles capacités parasites par rapport à d'autres, il est très évident d'exploiter cette observation pour diminuer la consommation de la puissance due au couplage. Pour ce faire, les travaux de recherche dans ce domaine consistent alors à opter pour un codage et un décodage de données. L'objet de ce mémoire est la présentation des méthodes essentielles développées dans ce domaine, ainsi que celle de notre contribution pour répondre à la problématique posée.

Mots-clés : *consommation de la puissance, dynamique, couplage, bus, codage, décodage de données, complexité algorithmique*

INTRODUCTION GENERALE

La conception des systèmes intégrés actuels est assujettie à la consommation de la puissance. Ceci est dû au développement technologique qui a engendré de nombreux paramètres parasites augmentant cette consommation, mais aussi au fait que de nombreux systèmes portables (téléphones, PCs, ...) inondant le marché actuel nécessitent une faible consommation de la puissance afin de leur assurer une plus grande autonomie, sachant que même les batteries les plus sophistiquées restent encore limitées. Outre ce problème d'autonomie, le problème de fiabilité du système pourrait se poser avec une conception quelconque du fait qu'une forte consommation de puissance entraînerait une augmentation de température pouvant à son tour affecter le bon fonctionnement du système.

Aussi, afin de pallier ce problème, différentes techniques de réduction de la consommation de la puissance sont appliquées à différents niveaux de conception, et ce, en considérant les différents composants du système. Parmi ces composants, on distingue les interconnexions (ou bus) permettant le transfert des données entre les différentes parties du système et qui constituent une source non moins importante du problème considéré. La dissipation de la puissance induite par un bus est de deux types : celle qui est due aux charges et aux décharges des capacités du bus, et celle résultant de la formation de capacités parasites entre les fils voisins du même bus. Des travaux de recherche actuels montrent que le deuxième type de consommation est devenu plus important par rapport au premier pour les technologies actuelles. Ainsi, pour les technologies CMOS 0.18 µm et 0.13 µm par exemple, les capacités parallèles sont respectivement 6 et 8 fois plus importantes que les capacités intrinsèques. Alors que les capacités intrinsèques sont fixes pour un bus donné, les capacités parallèles varient en revanche en fonction des données transitant sur le bus. Certaines données entraînant une plus faible dissipation de puissance, il est tout à fait évident d'exploiter cette observation pour réduire les capacités de couplage. Pour ce faire, différentes méthodes de codage et de décodage ont été utilisées. Toutefois, ce codage n'est possible qu'en augmentant la

taille du bus (nombre des fils le constituant) afin d'assurer la réception des données réellement transmises grâce à un décodage approprié. Cette augmentation du nombre de fils n'est pas sans problème du fait que la réduction des capacités de couplage pourrait s'accompagner de l'augmentation des capacités intrinsèques des différents fils du bus, d'où la nécessité de trouver le meilleur compromis.

Nous présenterons alors dans ce mémoire les principales méthodes développées ainsi que celle que nous proposons dans le deuxième et troisième chapitre, respectivement. Afin de faciliter au lecteur la compréhension du mémoire, des rappels et des définitions seront donnés au chapitre suivant. Le quatrième chapitre sera consacré à la présentation et la discussion des résultats obtenus, et nous terminerons par une conclusion générale suivie par la bibliographie étudiée.


CHAPITRE 1

1.1. Introduction :

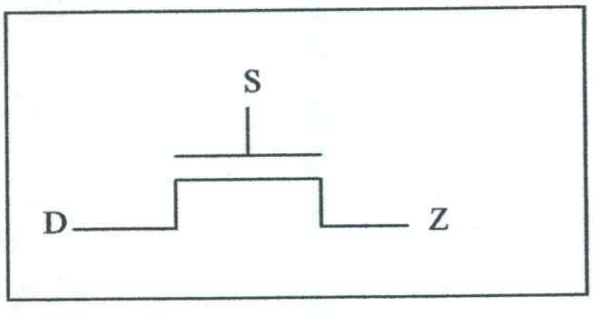
Afin de faciliter au lecteur la lecture et la compréhension de ce mémoire, nous avons jugé utile de donner des rappels et les définitions de certains termes qui seront rencontrés le long de ce mémoire.

1.2. Définitions et rappels :

Comme il a été dit dans l'introduction générale, la réduction de la consommation de la puissance intervient à différents niveaux de la conception du système, et ce, en considérant ses différents constituants. Parmi ces derniers, limitons-nous aux portes logiques et aux interconnexions.

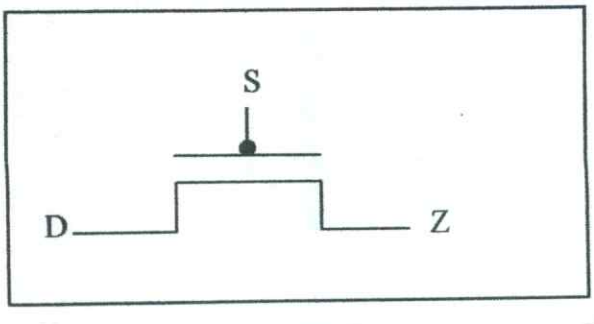
1.2.1. Portes logiques :

Une porte logique d'un circuit est un ensemble de transistors (NMOS et PMOS dans le cas de la technologie CMOS) permettant d'assurer la réalisation d'une fonction logique donnée. Cette dernière peut être soit simple (NOT, AND, NAND, OR, NOR, ...), soit complexe. Les caractéristiques électriques (temps de réponse et dissipation de puissance notamment) d'une porte dépendent des capacités attachées aux différents nœuds électriques.



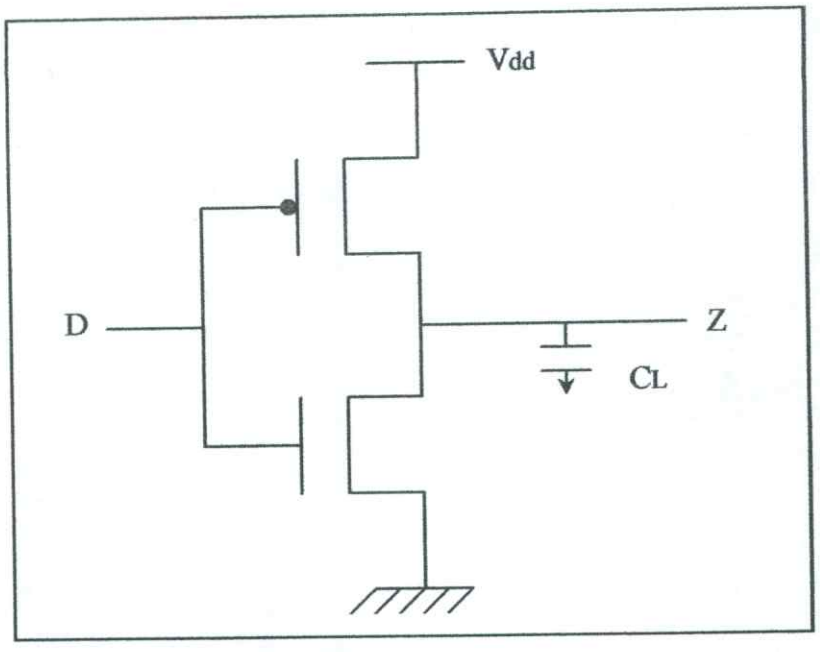
$$Z = \begin{cases} D \text{ si } S=1 \\ \text{Donnée précédente sinon} \end{cases}$$

Fig.1.1. Transistor NMOS



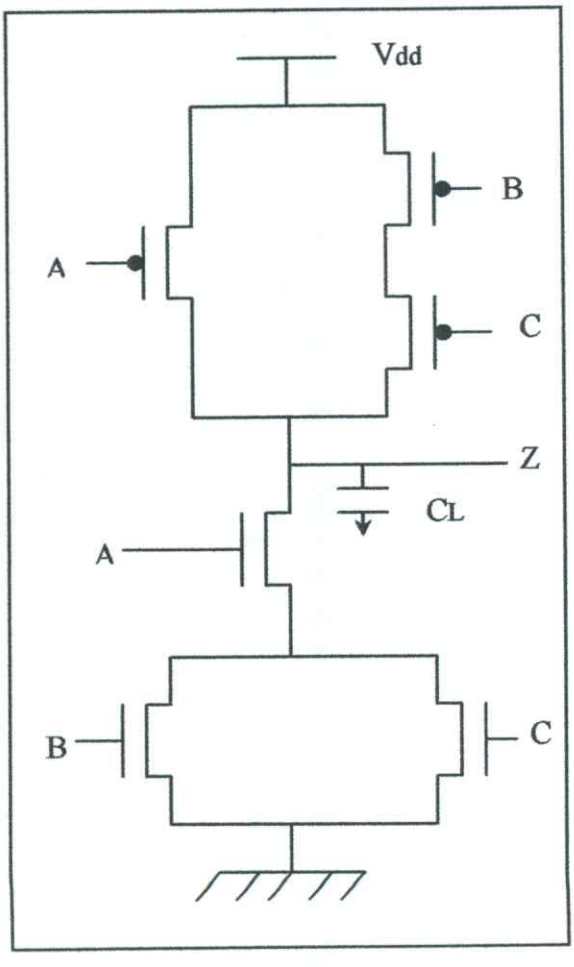
$$Z = \begin{cases} D \text{ si } S=0 \\ \text{Donnée précédente sinon} \end{cases}$$

Fig.1.2. Ttansistor PMOS



$$Z = not(D)$$

Fig.1.3. Inverseur CMOS

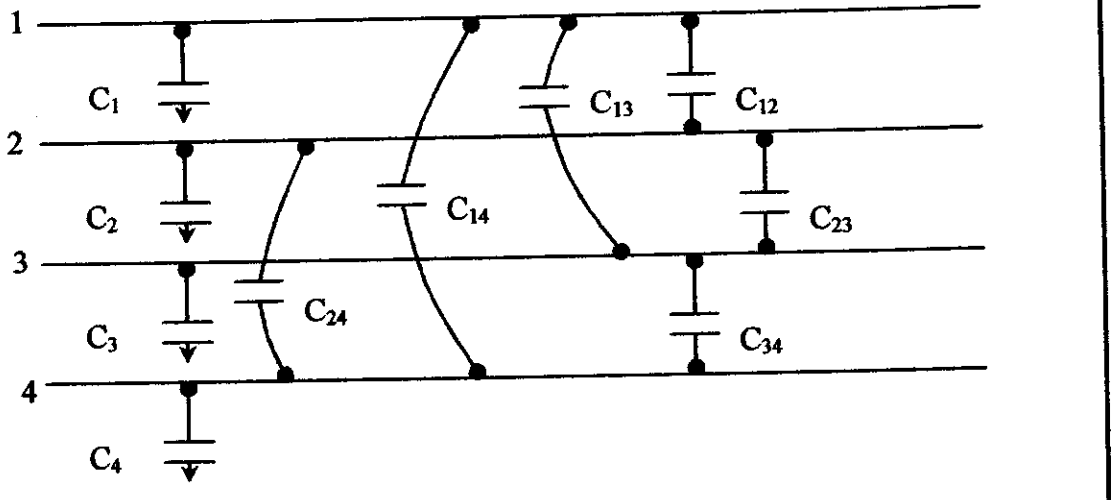


$$Z = not(A \ and \ (b \ or \ C))$$

Fig.1.4. porte logique CMOS implémentant une fonction complexe

1.2.2. Interconnexions :

Une interconnexion sert de support pour le transfert de données entre les différents composants du système. Chaque interconnexion se caractérise par une capacité intrinsèque dont la valeur dépend de la nature du masque utilisé (métal1, métal2, ...) et de ses dimensions géométriques (longueur et largeur). Des capacités parasites peuvent se former lorsque l'interconnexion se trouve dans un environnement donné (elle est en parallèle avec d'autres interconnexions, elle en croise d'autres, ...). La figure 1.5 montre un exemple où les deux types de capacités interviennent.



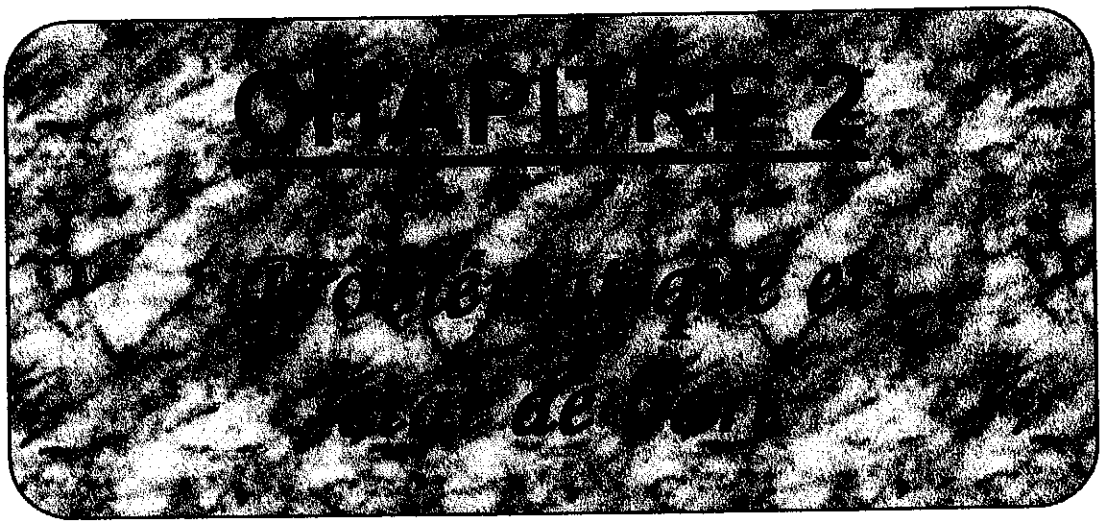
Ci : Capacité intrinsèque de la ligne numéro i (i = 1..4).

Cij : Capacité parasite entre les lignes i et j (i, j = 1..4, i≠j).

Fig.1.5. capacité intrinsèque et de couplage dans des interconnexions

1.3. Conclusion :

Dans ce chapitre, nous avions introduit les termes importants qui nous serviront de traiter l'aspect *dissipation de puissance* dans les prochains chapitres. Cet aspect concerne précisément la consommation de la puissance dynamique (charges et décharges des capacités intrinsèques) et celle due au couplage (capacité parallèle entre chaque paire d'interconnexions voisines). Des détails seront donnés au fur et à mesure que nous progresserons dans la lecture de ce mémoire.


CHAPITRE 2

2.1. Introduction :

Avant de présenter notre propre contribution pour solutionner le problème considéré, nous discuterons dans ce chapitre de manière plus détaillée de la problématique posée ainsi que des solutions qui lui ont été apportées.

2.2. Problématique :

Comme il a été dit dans l'introduction générale, le développement considérable de la technologie permet d'intégrer de plus en plus de transistors dans la même puce. Cette considérable intégration n'est toutefois pas sans problèmes, parmi lesquels celui de la dissipation de la puissance. Aussi, une conception classique d'un système VLSI (*Very Large Scale Integration*) dans les technologies actuelles engendrerait une forte dissipation de puissance pouvant entraîner une augmentation de la température, ce qui pourrait affecter la fiabilité du système. En outre, il existe actuellement sur le marché de nombreux systèmes portables (téléphones, PCs, ...) qui nécessitent tous une faible consommation de la puissance assurant ainsi une plus grande autonomie de leur fonctionnement quand on sait que les batteries les plus récentes restent encore limitées. Le problème de la dissipation de la puissance se pose pour les différents constituants du système, dont les interconnexions qui font l'objet de notre travail.

Alors que la consommation de la puissance d'un bus dans les anciennes technologies CMOS se limitait essentiellement à celle due à la puissance dynamique (charges et décharges des capacités intrinsèques des fils du bus), la dissipation due aux paramètres parasites est devenue non négligeable dans les technologies actuelles. Pire, elle est devenue nettement plus considérable que la consommation dynamique. Ainsi, dans les technologies CMOS 0.18 µm et 0.13 µm par exemple, la capacité parallèle est respectivement 6 et 8 fois plus importante que la capacité intrinsèque. Considérons la figure 1.5 où les deux types de capacités sont indiqués. L'estimation de la consommation de l'énergie du bus est alors obtenue en considérant celles qui sont dues aux deux types de capacités :

- énergie dynamique :

$$E_{dyn} = 0.5 * V_{dd}^2 * C_i * N_i \qquad (2.1)$$

Où V_{dd} est la tension d'alimentation,

C_i est la capacité intrinsèque du $i^{ème}$ fil du bus,

N_i vaut 1 si la donnée actuellement présente sur le $i^{ème}$ fil du bus est différente de la précédente, 0 sinon.

- énergie due aux capacités parallèles :

$$E_{par} = \lambda \sum_{i=1}^{n-1} \left(V_{i+1}^{new} - V_i^{new}\right)\left(\left(V_{i+1}^{new} - V_i^{new}\right) - \left(V_{i+1}^{old} - V_i^{old}\right)\right) * C_L \qquad (2.2)$$

Où n est le nombre de fils du bus

V_i^{new} : est la donnée actuelle sur le $i^{ème}$ fil du bus

V_i^{old} : est la donnée précédente sur le $i^{ème}$ fil du bus

C_L est la capacité intrinsèque d'un fil du bus

Considérant un bus à n bits, sa consommation totale d'énergie est alors : $E = E_{dyn} + E_{par}$, et sa puissance totale consommée est : $P = E * f$, f étant la fréquence de fonctionnement du système considéré.

Alors que la capacité intrinsèque d'une interconnexion est fixe, la capacité parallèle entre deux fils varie en revanche en fonction des données présentes sur les différentes interconnexions. Or il se trouve que pour certaines données, ces capacités parallèles sont faibles. Il est alors évident d'exploiter ces données pour réduire ces capacités parallèles, donc pour réduire la consommation d'énergie due à ces capacités parasites. Mais comment utiliser ces données de manière judicieuse tout en sachant exactement quelles sont les données réelles qui transitent sur le bus ? La réponse à cette question est donnée dans le prochain paragraphe ainsi que dans le troisième chapitre.

2.3. Etat de l'art:

Comme il a été dit dans l'introduction générale, le développement considérable de la technologie des semi-conducteurs permet de nos jours d'intégrer des systèmes complexes dans la même puce. Toutefois, cette gigantesque intégration s'accompagne de nombreux problèmes, dont celui de la dissipation d'énergie. Alors

que dans les anciennes technologies la consommation de la puissance due aux capacités parasites était quelque peu négligeable devant d'autres types de puissance, il n'en est rien avec les technologies submicroniques. Pire encore, les capacités parallèles induites par les fils d'un bus sont devenues nettement plus importantes que les capacités intrinsèques de ces fils. Nous avions mentionné auparavant que dans les technologies CMOS 0.18 µm et 0.13 µm par exemple, elles sont respectivement 6 et 8 fois plus importantes ! De ce fait, il est important, voire obligatoire de tenir compte de ce type de capacité dans la conception des systèmes actuels (systèmes embarqués, portables, ...) du fait que l'aspect *dissipation de puissance* est l'un des plus importants critères guidant leur conception. Pour ce faire, de nombreuses méthodes ont été développées aussi bien pour la consommation de l'énergie dynamique que pour celle de l'énergie due au couplage. Du fait que notre thème de travail concerne ce dernier type d'énergie, nous allons nous concentrer dans la suite de ce chapitre sur les principales et relativement récentes techniques développées en vue de réduire l'énergie due aux capacités parallèles. Le chapitre suivant traitera alors de notre propre contribution dans la problématique posée. Enfin, notons que ces techniques s'appuient toutes sur le modèle défini par les équations (2.1) et (2.2). Ce modèle a été établi à partir de certains travaux de recherche dont les principaux ont été publiés dans [12] et [13]. De plus, elles utilisent toutes le principe de codage et de décodage de données comme il est indiqué dans la figure 2.1 dont le contenu sera expliqué ultérieurement à travers un exemple.

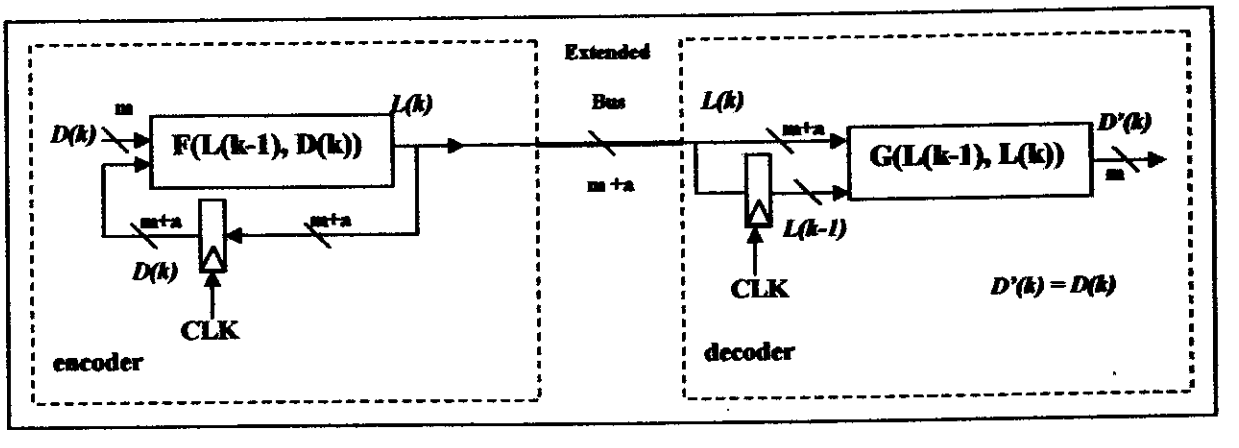


Fig.2.1. Codage et de décodage de données en vue de réduire l'énergie induite par les capacités parallèles des fils d'un bus.

2.3.1. Codage avec affectation de probabilités fixes:

Nous avions mentionné auparavant que certaines données transmises sur le bus induisent des capacités parallèles plus faibles par rapport à d'autres données. Il est alors évident d'exploiter ces données pour réduire la dissipation d'énergie. Néanmoins, le codage et le décodage des données doivent être faits de manière à ce qu'il n'y'ait aucune ambiguïté pour savoir quelles sont les données qui ont été réellement transmises. Pour ce faire, la taille du bus est augmentée de n fils à m fils ($m > n$) – la valeur de m sera discutée ultérieurement – et le principe de codage et de décodage est alors utilisé comme il est indiqué dans la figure 2.1. Avec cette technique, et en vue d'exploiter au maximum les données induisant de faibles capacités parallèles, la probabilité affectée à une donnée est d'autant plus grande que cette donnée induise une plus faible capacité parasite. Nous allons illustrer le principe de cette technique par l'exemple suivant :

Exemple:

Soit n=2 (n est la taille initiale du bus)

λ=3 (la capacité parallèle entre 2 fils du bus vaut 3 fois celle de la capacité intrinsèque d'un fil du bus)

Du fait que n=2, les données possibles à transmettre sur le bus sont : 00, 01, 10 et 11. Supposons que la séquence de données transmises sur le bus est la suivante : 01, 10, 01, 10, 00, 11, 10, 01, 01, 10.

Aucun codage:

L'énergie due aux capacités parallèles serait alors égale à 40 * C_L, et ce, en utilisant les équations (2.1) et (2.2).

Codage des données:

Supposons aussi que les données 01 et 10 soient celles qui seront les plus transmises et que les probabilités affectées aux différentes données possibles soient les suivantes : 00 $\rightarrow$ 0.1, 01 $\rightarrow$ 0.4, 10 $\rightarrow$ 0.4, 11 $\rightarrow$ 0.1. Supposons que la nouvelle taille du bus soit m=3. En utilisant les équations (2.1) et (2.2), la matrice qui suit donne toutes les valeurs possibles d'énergie de couplage en considérant toutes les paires de données (celle qui a été auparavant transmise et celle qui est en cours de transmission sur le bus).

Table 2.1. Valeurs de E_{Total}/C_L en considérant toutes les paires de données possibles transmises successivement sur un bus de taille 3 bits.

Donnée actuelle → / Donnée précédente ↓	000	001	010	011	100	101	110	111
000	00.00	03.50	06.50	04.00	03.50	07.00	04.00	01.50
001	00.50	00.00	10.00	03.50	04.00	03.50	07.50	01.00
010	00.50	07.00	00.00	00.50	07.00	13.50	00.50	01.00
011	01.00	03.50	03.50	00.00	07.50	10.00	04.00	05.00
100	00.50	04.00	10.00	07.50	00.00	03.50	03.50	07.00
101	01.00	00.50	13.50	07.00	00.50	00.00	07.00	00.50
110	01.00	07.50	03.50	04.00	03.50	10.00	00.00	00.50
111	01.50	04.00	07.00	03.50	04.00	06.50	03.50	00.00

Du fait que la donnée réelle à transmettre est codée sur 2 bits uniquement, la donnée qui est initialement transmise et qui doit être récupérée par le bloc récepteur est 00, 01, 10 ou 11, soit quatre types de données possibles. Du fait aussi que l'objet est de minimiser l'énergie dissipée à cause des capacités parallèles, il faudrait donc retenir, à partir de la table 2.1, les quatre paires de données successives qui minimisent le plus l'énergie considérée. Ainsi, si la donnée précédente (codée) est 010, les paires de vecteurs (010, 000, 011, 110) seraient les plus intéressantes pour le codage et le décodage. Parmi ces quatre valeurs, les deux premières valeurs (qui sont les plus faibles) seraient utilisées pour les données réelles 01 et 10 du fait que ces dernières ont les plus fortes probabilités, c'est-à-dire celles qui seront le plus souvent transmises. Il est donc tout à fait évident de procéder à un tel choix. En raisonnant de la même manière pour toutes les autres paires de vecteurs, on obtient l'affectation qui est indiquée dans la figure 2.2. Cette figure indique par exemple que si la donnée codée précédente était 000 et que la donnée codée actuelle est 001, alors la donnée qui est réellement transmise est 00, et l'énergie due aux capacités parallèles est de 03.50 C_L F. En nous appuyant sur cet exemple, la figure 2.1 nous indique ce qui suit :

- D (k) est 00,
- m = 2 bits ; m + a = m + 1 = 3 bits

- F (L (k-1), D (k)) = F (000, 00) = 001 (Fonction de codage)
- L (k) = 001
- G (L (k-1), L (k)) = G (000, 001) = 00 (Fonction de décodage)

La transmission de la séquence 01, 10, 01, 10, 00, 11, 10, 01, 01, 10 nous conduit alors à une dissipation d'énergie de 12.00 C_L F, qui est plus faible que celle obtenue sans codage de données, soit un gain d'énergie de 233% (gain= 100*(40 – 12)/12)).

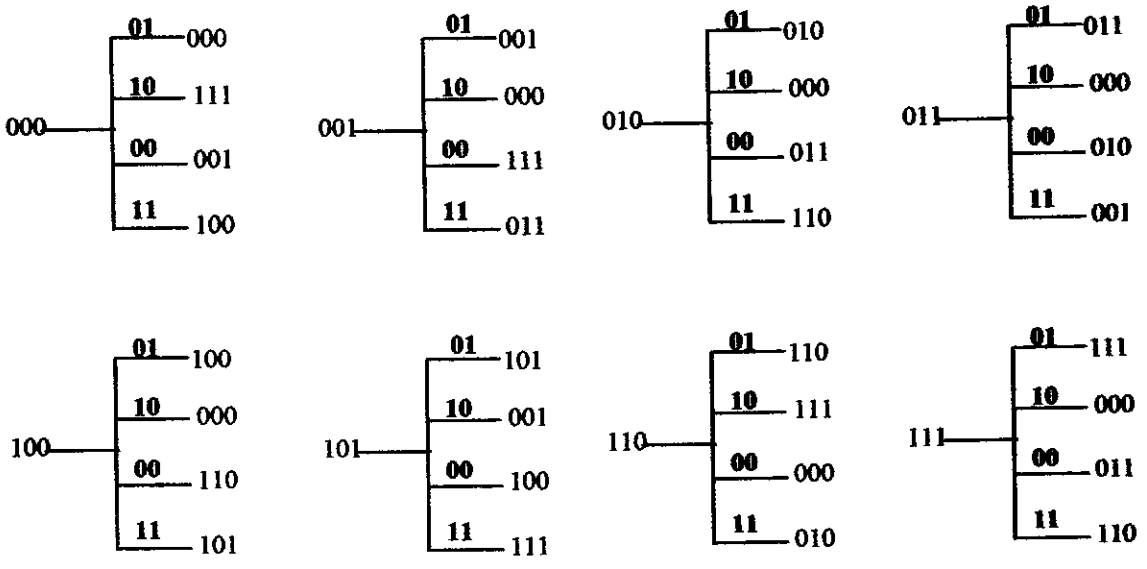


Fig.2.2. Codage et décodage de données avec l'utilisation de probabilités fixes en vue de réduire l'énergie due aux capacités parallèles.

2.3.2. Combinaison du codage avec affectation de probabilités fixes et retard de transfert de données:

La technique qui vient d'être présentée offre un avantage certain en matière de dissipation d'énergie par rapport à celle qui n'utilise pas de fonctions de codage et de décodage. Toutefois, certaines transitions de données continuent à induire une dissipation d'énergie relativement plus importante même quand les fonctions de codage et de décodage sont utilisées. Dans [14], on décrit une technique qui consiste à retarder toute transition 0 → 1 en vue de réduire la dissipation d'énergie due au couplage. Ainsi, pour la transition des données 1010 vers les données 0101, la transmission se fera comme suit : 1010 → 0000 → 0101 afin d'éviter les transitions

0 → 1 sur le deuxième et quatrième fil du bus. Ceci se fait alors en retardant la transmission des nouvelles données sur le deuxième et quatrième fil du bus.

2.3.3. Codage avec affectation de probabilités variables:

Dans les techniques utilisant des probabilités, il est clair que le but est d'aboutir à une plus faible dissipation d'énergie pour les données qui sont le plus souvent transmises. Dans l'exemple de la figure 2.2, le codage est fait de sorte que la plus faible consommation d'énergie (00.00 C_L F) est associée à la donnée 01 qui est susceptible d'être fréquemment transmise relativement aux autres données (probabilité égale à 0.4). Malheureusement, il n'y'a aucune information nous permettant d'affirmer que la donnée 01 sera celle qui sera toujours le plus souvent transmise. Ainsi, si au cours du temps la donnée 11 (à laquelle la plus forte dissipation d'énergie parmi les trois autres est associée) devient celle qui se transmet de plus en plus, le codage indiqué dans la figure 2.2 serait néfaste pour la dissipation d'énergie du fait qu'une forte énergie (relativement aux trois autres) est dissipée un nombre considérable de fois, c'est-à-dire autant de fois que la donnée 11 a été transmise ! Ceci a conduit au développement d'une autre technique qui consiste non pas à affecter des probabilités fixes pour les données, mais plutôt des probabilités variables en fonction de l'occurrence des données au cours du temps. Ainsi, le principe général de cette technique est le suivant :

- affecter des probabilités aux différentes données à la base de suppositions préalables sur leur occurrence pendant une période donnée
- après chaque période, ordonner dans l'ordre décroissant de leur occurrence les données et leur affecter des probabilités adéquates, celle qui a été transmise le plus recevant évidemment la plus forte probabilité, et donc associée à la plus faible énergie parmi les 2^n, n étant la taille initiale du bus

2.4. Conclusion:

Nous avions présenté dans ce chapitre la problématique ainsi que les techniques essentielles développées dans le but de répondre au problème. La dernière technique présentée semble la plus prometteuse mais recèle toujours des inconvénients. Notre technique qui sera présentée dans le chapitre qui suit consiste à éviter ces inconvénients tout en gardant les avantages de la dernière technique.


CHAPTER 3

3.1. Introduction :

Nous détaillons dans ce chapitre les principales techniques utilisées pour la problématique posée. Ces différentes techniques que nous avions programmées pour des fins de comparaison avec notre méthode ont été succinctement présentées dans le chapitre précédent.

3.2. Présentation de la méthodologie générale :

La méthodologie générale adoptée se base sur l'organigramme donné ci-après. Elle consiste en général à coder et décoder les données en vue de réduire la dissipation d'énergie. Les méthodes reposant sur cette méthodologie évidemment diffèrent par la manière dont le codage et le décodage sont réalisés.

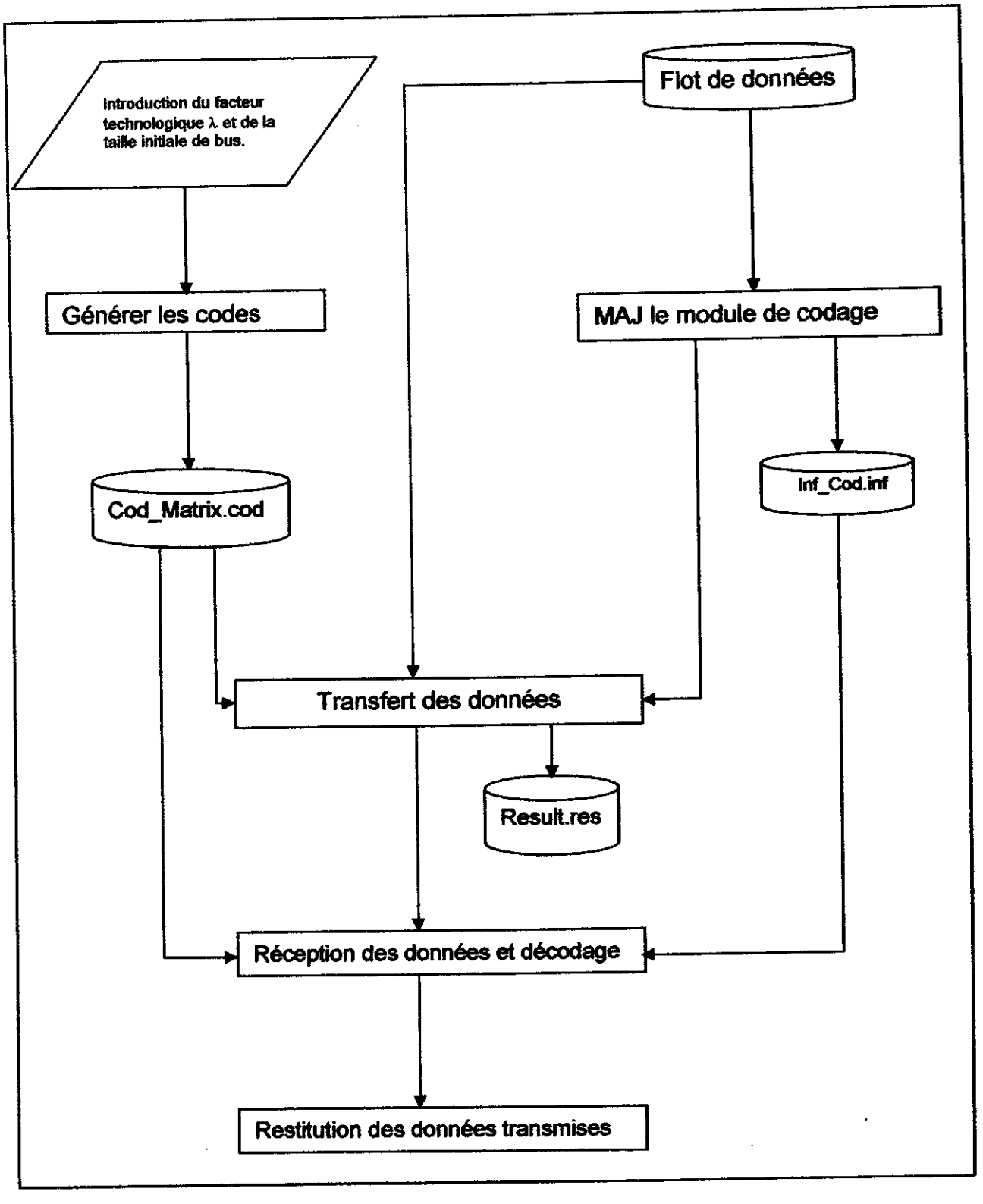


Fig.3.1. Le schéma synoptique de la méthodologie générale adoptée

Comme il est indiqué dans la figure 3.1, nous avons décomposé le problème en quatre grandes parties :

La première partie *(Générer les codes)* est une étape préliminaire pour la résolution du problème qui consiste à générer tous les codes possibles sur un bus de m bits, la consommation d'énergie pour chaque paire de codes $(c_i,c_j) \in B^m \times B^m$ avec B={0,1}, les trie par ordre croissant de leurs consommations d'énergie et stocke les résultats dans le fichier **Cod_Matrix.cod** (c'est la génération de la matrice des énergies associées aux différentes paires de codes). Cette étape nécessite la connaissance du facteur technologique λ (voir Equation 2.2) et de la taille initiale du bus.

La deuxième partie *(MAJ le module de codage)* consiste, pour chaque code $c_i \in B^m$, à sélectionner parmi 2^m codes appartenant chacun à B^m, les 2^n codes (n étant la taille initiale du bus) qui engendrent les plus faibles dissipations d'énergie associées au code c_i. Ces 2^n codes sont alors combinés avec c_i pour être affectés aux données codées sur n bits selon la probabilité de chacune de ces données. Ces informations sont alors stockées dans le fichier **info_cod.inf** pour qu'elles puissent être utilisées par la suite pour le décodage. Notons que l'exemple de la figure 2.2 illustre cette partie.

La troisième partie *(transfert)* exploite les fichiers **cod_matrix.cod** et **info_cod.inf** pour déterminer l'énergie associée à une paire de codes $(c_i,c_j) \in B^m \times B^m$, laquelle paire est déterminée en fonction de la donnée de taille n bits. L'énergie consommée pour la donnée en cours de transfert sur le bus est alors écrite dans le fichier **result.res**.

La quatrième partie *(réception des données et décodage)* consiste à décoder les données pour la réception des données réellement transmises avant leur codage. Cette partie exploite les deux fichiers **cod_matrix.cod** et **info_cod.inf** et les codes réellement transmis.

3.3. Méthode 1 : Codification Sans Probabilité

Description de la méthode :

Le transfert des données de taille *n* sont codées sur *m* (m > n) bits en utilisant le procédé décrit en 2.3.1. Toutefois, il est supposé pour cette méthode que toutes les données sont équiprobables. De ce fait, les 2^n faibles valeurs d'énergie parmi les 2^m valeurs possibles pour les couples de codes $(c_i, c_j) \in B^m \times B^m$ (i fixé, j=1, 2, ...,2^m) sont affectées indifféremment aux couples (c_i,c_j).

3.3.1. Algorithme:

```
Algorithme Méthode_Avec_Codification_Sans_Probabilité
Var
    Cod, Data : char*; // Tableaux utilisés pour les codes à transmettre
    ancien_vect, nouv_vect : integer*; // Tableaux utilisés pour les différents
                                       // états de bus
    vnew, vold : integer; //  Entiers associés aux différents états du bus
    Nbr, S2 : integer init 0; //S2 : contient l'énergie parasite consommée
    S, S1: real init 0;       //S1 : contient l'énergie intrinsèque consommée
                              //S=S1+S2
    R : RESULTAT;             //R : contient le résultat (énergie consommée pour un
                              //flot de données transmises sur le bus)
Début
  Ecrire ("donnez n :", n);     // Lecture de la taille des données avant la codification
  Ecrire ("donnez lambda :", lambda); // Lecture de la valeur du facteur
                                      // technologique λ
    // Génération des codes aléatoires pour les transmettre
  Pour i := 0 jusqu'à DATA_NBR
  Faire
      Nbr := rand (); // Génération d'un nombre entier
      Cod := int2bin (Nbr, n, p2); // Transformation du nombre entier en binaire
      Mettre le contenu du tableau Cod dans le tableau Data;
  Fait
  Open (result, L/E) ; // Ouverture du fichier des résultats en mode L/E
```

```
Pour m := n + 1 jusqu'à n + 10
Faire
    S := 0; S1 :=0 ; S2 :=0; // Initialisation à 0 de l'énergie consommée
    vold := 0;  //  L'état initial de bus est (000...0)
    Open (cost_matrix, L/E);  // Ouverture du fichier des codes en mode L/E
    Remplir (cost_matrix, 2^m, 2^n);  // Remplissage du fichier des codes
    Tant que (il y'a des données à envoyer)  // Envoi des données
     Faire
        ancien_vect := int2bin (vold, m, p); // Mettre état courant dans
                                              // ancien_vect
        Mettre la donnée courante dans Cod ;
        Vnew := bin2int (Cod, n); //mettre l'équivalent du nouvel état dans
                                  // Vnew
       // Charger tampon dans le cas où les données demandées ne s'y
       // trouvent pas
        Charger_tampon () ;
        nouv_vect :=int2bin (ENERGI_MAT[vold][vnew].num_colonne, m,p1);
       //transformation d'une valeur entière en binaire
       // num correspond au nouvel état du bus
        S1 + := ENERGI_MAT [vold] [vnew].e1; // S1:énergie intrinsèque
        S2 + := ENERGI_MAT [vold] [vnew].e2; // S1:énergie parasite
        S + := S1 + S2; // S: énergie totale
        vold := bin2int (nouv_vect, m);
     Ftq
    R.intrinseq := S1;
    R.parasit := S2;
    Write (R, size of (RESULTAT), 1, result);
    Close (cost_matrix);
 Fin
 Close (result);
FIN
```

3.3.2. Description des structures de données :

La structure Nœud :

Cette structure contient les informations concernant un élément de la matrice d'énergie (voir **Fig.3.2**)

```
struct nœud {
        int num_ligne;
        int num_colonne;
        int e2;
        float e1;
        };
```

Fig.3.2. La structure *nœud*

Où :

- **num_ligne** (resp. **num_colonne**) indique le numéro de la ligne (resp. colonne) de cet élément dans la matrice d'énergie.
- **e1** (resp. **e2**) indique l'énergie dynamique (resp. parasite) consommée quand l'état du bus change de l'état associé à **num_ligne** vers celui associé à **num_colonne**.

La structure RESULTAT :

Cette structure contient le résultat de l'exécution (voir **Fig.3.3**)

```
struct RESULTAT {
        float intrinseq;
        int parasit;
        };
```

Fig.3.3. La structure RESULTAT

Où :

- **intrinseq** (resp. **parasit**) indique l'énergie dynamique (resp. parasite) consommée pour un flot de données considéré

Les variables :

n : le nombre de bits avant la codification.

m : le nombre de bits après la codification.

Lambda : facteur technologique variant d'une technologie à une autre.

3.3.3. Description des fichiers utilisés:

Nous allons décrire dans ce qui suit les fichiers utilisés en entrée par nos algorithmes et ceux qui sont générés après l'exécution.

Le fichier cost_matrix :

Ce fichier contient les 2^n éléments qui engendrent les plus faibles valeurs d'énergie pour chaque ligne de la matrice d'énergie. Il contient alors $2^n \times 2^m$ éléments.

Structure d'un enregistrement du fichier cost_matrix:

L'enregistrement de ce fichier contient (04) quatre champs comme il est indiqué dans la **Figure 3.4.**

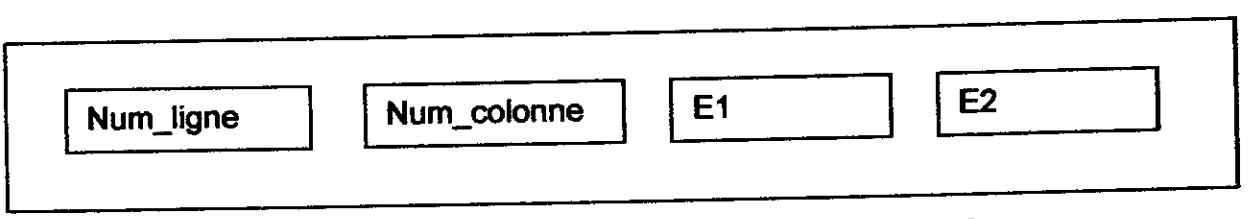


Fig.3.4. Format d'un enregistrement du fichier cost_matrix

Où :

- **Num_ligne** (resp. **Num_colonne**) : indique le numéro de la **ligne** (resp. **colonne**) d'un élément de la matrice d'énergie.
- **E1** (resp. **E2**) est la valeur de E/C_L, E étant l'énergie **dynamique** (resp. **parasite**) consommée quand l'état du bus change de l'état associé à **Num_ligne** vers celui associé à **Num_colonne**.

Le format du fichier cost_matrix est présenté dans la **Figure 3.5**

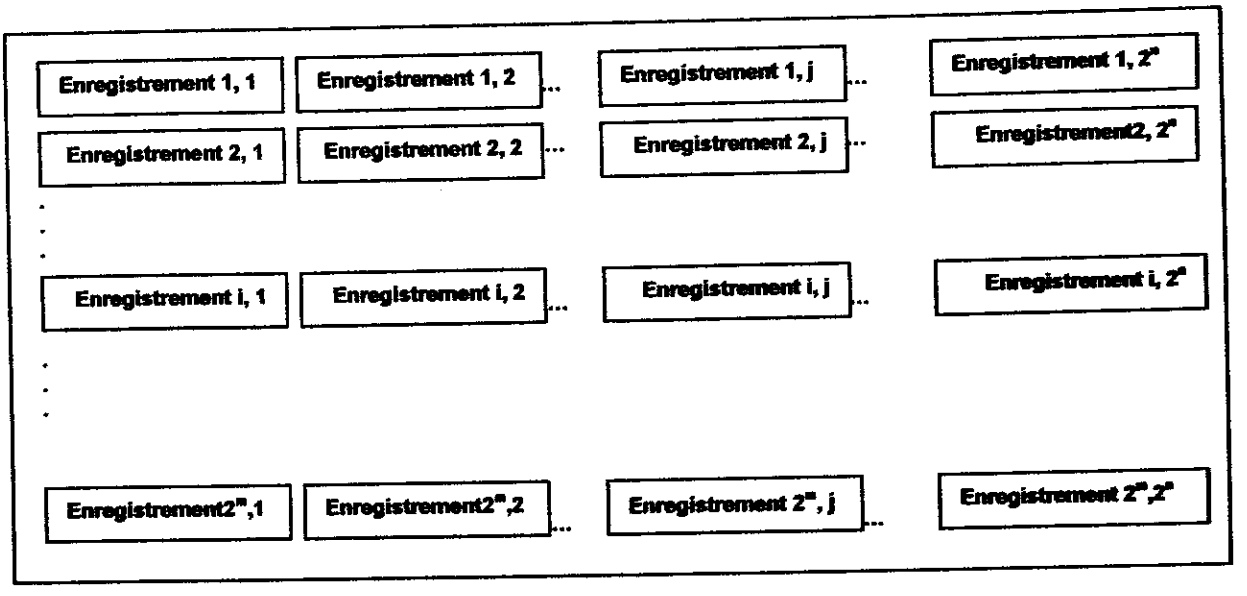


Fig.3.5. Format du fichier cost_matrix

Le fichier résultat :

Ce fichier contient le résultat obtenu après exécution (E/C_L pour le transfert d'un flot de données considéré).

Format du fichier :

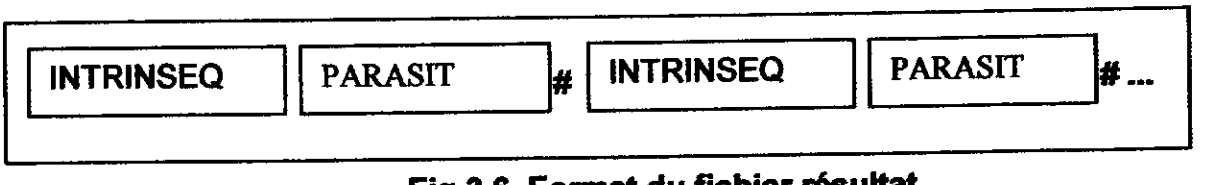


Fig.3.6. Format du fichier résultat

Où:

- **INTRINSEQ** (resp. **PARASIT**) correspond à E/C_L, où E indique l'énergie **dynamique** (resp. **parasite**) consommée pour un flot de données.

3.3.4. Les fonctions de cette méthode :

int2bin (int elt, int Taille, int *p) :

Cette fonction permet de convertir un entier en binaire.

Où :

- **elt** : l'élément à convertir.
- **Taille** : indique le nombre de bits utilisés pour le codage en binaire.
- **p** : la table qui contient le code binaire (le résultat retourné par la fonction).

Algorithme:

```
Algorithme int2bin;
        Var  k : integer init 1;
Début
    Pour i := 0 jusqu'à Taille
    Faire // initialiser le tableau par le vecteur nul (00...0)
        p [i] := 0 ;
    Fait
    Tant que (elt > 0)
    Faire //calculer le code binaire
        p [Taille – k++] := elt mod 2;
        elt := elt div 2;
    Fait
Fin
```

bin2int (char *T, int Taille) :

Cette fonction permet de convertir une suite de '0' et de '1' (un nombre binaire) en un nombre entier.

Où :

- **T** : une table qui contient le code à convertir.
- **Taille** : nombre de bits du code.

Algorithme:

```
Algorithme bin2int;
    Var S : integer init 0;
Début
    Pour i := 0 jusqu'à Taille –1
    Faire
        Si T [i] = 1 Alors
            S + := 2^(Taille – (i + 1));
        FSi
    Fait
Fin
```

Remplir (FILE *str, int T1, int T2) :

C'est la fonction qui calcule la matrice d'énergie et trie ses lignes par ordre croissant de valeurs d'énergie consommée et qui met les 2^n (n étant le nombre de bits avant la codification) éléments de chaque ligne dans le fichier de pointeur **str**.

Où :

- **Str** : pointeur sur le fichier **cod_matrix** où on stocke les résultats.
- **T1** : le nombre de lignes (et de colonnes) de la matrice d'énergie ($T1 = 2^m$).
- **T2** : le nombre de codes stockés dans **cod_matrix** pour chaque ligne triée ($T2 = 2^n$).

Algorithme :

```
Algorithme Remplir ;
Var tab : nœud* ;
Début
    Pour i := 0 jusqu'à T1 // boucle sur les lignes de la matrice d'énergie
    Faire
        Pour j := 0 jusqu'à T1 // calcul de la i^eme ligne de la matrice
        Faire
            Tab [j].e1 := facteur1 (i, j, m);
            Tab [j].e2 := facteur2 (i, j, m, lambda);
            Tab [j].num_ligne := i;
            Tab [j].num_colonne := j;
        Fait
        trirapide (Tab, 0, T1 – 1); // trier la ligne calculée
        write (Tab, sizeof (Noeud), T2, str);
        // Mettre T2 enregistrements dans le fichier pointé par str
    Fait
FIN
```

Facteur1 (int x, int y, int Taille) :

C'est la fonction utilisée pour déterminer l'énergie dynamique consommée quand le bus change de l'état X vers l'état y.

Où :

- **x** (resp. **y**) : représente l'état **courant** (resp. **prochain**) du bus
- **Taille** : le nombre de bits utilisés pour le codage

Algorithme facteur1;

```
Var  vold, vnew : integer*;
     som1, som2 : real init 0;
Début
     vold := int2bin (x, Taille, p0) ; // mettre l'état courant dans vold
     vnew := int2bin (y, Taille, p1); // mettre l'état suivant dans vnew

     Pour i := 0 jusqu'à Taille
     Faire
          som1 +:= vnew [i] * (vnew [i] – vold [i]); // l'énergie due à la charge de la
                    // capacité intrinsèque associée à la i^ème interconnexion du bus
          som2 +:= vold [i] * (vold [i] – vnew [i]); // l'énergie due à la décharge de
                    // la capacité intrinsèque associée à la i^ème interconnexion du bus
     Fait
     Return 0.5 * (som1 + som2) ;
Fin
```

Facteur2 (int x, int y, int Taille, int Lambda) :

C'est la fonction utilisé pour déterminer l'énergie **parasite** consommée quand le bus change d'un état x vers un état y.

Où :

- **x** (resp. **y**) représente l'état **courant** (resp. **prochain**) du bus.
- **Lambda** : un facteur technologique.
- **Taille** : le nombre de bits utilisés pour le codage.

Algorithme facteur2;

```
Var  vold, vnew : integer*;
     Som3 : real init 0;
Début
     vold := int2bin (x, Taille, p0) ; // mettre l'état courant dans vold
     vnew := int2bin (y, Taille, p1); // mettre l'état suivant dans vnew

   Pour i:= 0 jusqu'à Taille – 1
   Faire
   som3 +:= (vnew [i+1]–vnew [i])*((vnew [i+1]–vnew [i]) – (vold [i+1] – vold [i]));
   Fait

   Return (lambda * som3);
Fin
```

La fonction Trier :

Nous allons brièvement expliquer la méthode dite du **"tri rapide" (Quick Sort)** du fait que la complexité algorithmique de notre application dépend fortement de la complexité de cette fonction. Ceci, d'autant plus que le nombre et la taille des tableaux manipulés sont importants.

Rappelons ici les cinq principales méthodes de tri qui existent:

- le tri à bulles
- le tri à bulles optimisé
- le tri dichotomique
- le tri par la méthode de Shell Metzner
- le tri rapide (Quick Sort)

Nous expliquons ci-après le principe du tri rapide en considérant l'ordre croissant (le même principe s'applique pour le tri par ordre décroissant). Parmi les cinq méthodes précédemment citées, la dernière est la plus rapide. Cependant, cette fonction étant récurrente, elle utilise plus de ressources que la méthode de Shell Metzner.

Le principe consiste à faire un tri par rapport à un élément du tableau, les plus forts d'un coté, les autres de l'autre. On recommence ensuite pour chacun de ces cotés et ainsi de suite... Notons que cette procédure s'appelle elle-même, c'est-à-dire qu'elle est récurrente.

Pour simplifier sa programmation, nous avons décomposé la fonction en trois fonctions élémentaires qui sont **echangerElements**, **Partition** et la fonction récurrente **triRapide** dont ses détails sont donnés ultérieurement.

echangerElements (Noeud T [ ], int m, int n):

Cette fonction permet de faire la permutation de deux éléments du tableau T d'indices m et n.

- Le paramètre **m** correspond au numéro du premier élément.
- Le paramètre **n** correspond au numéro du deuxième élément.
- Le paramètre **T [ ]** est le tableau qui contient les deux éléments.

Algorithme :

```
Algorithme echangerElements
Var  temp : Nœud;
Début
    temp := T [m];
    T [m] := T [n];
    T [n] := temp;
Fin
```

Partition (Nœud T [ ], int m, int n):

- Le paramètre **m** correspond au numéro du premier élément à trier (à l'appel il vaut 1).
- Le paramètre **n** correspond au numéro du dernier élément à trier (à l'appel il vaut Nb_Element).
- Le paramètre **T [ ]** est le tableau à trier. Il est modifié puis retourné à la procédure appelante.

Algorithme :

```
Algorithme partition
        Var v  : Nœud;
             i, j : integer ;
Début
   v := T [m]; // valeur pivot (on prend comme pivot initial l'élément du premier indice
                //du tableau)
   i := m – 1;
   j := n + 1; // indice final du pivot
        Tant que (vrai)
        Faire
              Faire
                    j–;
              Tant que (T [j] > v);
              Faire
                    i++;
              Tant que (T [j] < v);
              Si (i < j) Alors
                    échangerÉléments (T, i, j);
              Sinon
                    Return j;
              FSi
        Fait
FIN
```

triRapide (Nœud T [ ], int m, int n) // fonction récurrente

Algorithme :

```
Algorithme triRapide
    var P : integer ;
Début
    Si (m < n) Alors
        P := partition (T, m, n);
        // Diviser le tableau en deux parties et mettre les éléments qui sont
        // inférieurs au pivot dans l'une et les autres dans l'autre partie
        // Trier les éléments de chacune des deux parties
        triRapide (T, m, P); // le tri de la première partie du tableau
        triRapide (T, P + 1, n); // le tri de la deuxième partie du tableau
    FSi
FIN
```

Charger tampon ():

C'est une procédure qui permet de charger le tampon dans le cas où les données dont on a besoin ne s'y trouvent pas.

Algorithme :

```
Algorithme Charger_tampon;
Début
Si (Donnée demandée n'est pas dans le tampon) Alors
    // MAX_ROW est le nombre de lignes du tampon où on charge les données à partir
    // du fichier des codes
    // MAX_COLUMN est le nombre de colonnes du tampon

    Si (MAX_ROW > = 2^m) Alors
        // Cas où le nombre de lignes du tampon est plus grand que le nombre de
        // lignes de la matrice d'énergie
        Si (MAX_COLUMN < 2^n) Alors
            // Cas où le nombre de colonnes du tampon est plus petit que le
            // nombre de colonnes de la matrice d'énergie

            Pointer la tête de Lect./Ecrit. sur le premier enregistrement demandé ;
            Pour i := 0 jusqu'à 2^m // copier une partie de la matrice
            Faire
                fread (Tampon[i], sizeof (Noeud), MAX_COLUMN, cost_matrix);
                fseek (cost_matrix, ( 2^n – MAX_COLUMN) * sizeof (Noeud),
                    SEEK_CUR);
            Fait
        Sinon
            // Cas où le nombre de colonnes du tampon est plus grand que le
            // nombre de colonnes de la matrice d'énergie
            Pointer la tête de Lect./Ecrit. sur le premier enregistrement du fichier ;
            Pour k := 0 jusqu'à 2^m //copier toutes les données de la matrice
            Faire
                fread (Tampon [k], sizeof (Noeud), 2^n, cost_matrix );
            Fait
        FSi
```

```
Sinon
    // Cas où le nombre de lignes du tampon est plus petit que le nombre de
    //lignes de la matrice d'énergie

    Si (MAX_COLUMN > = 2^n) Alors
        // Cas où le nombre de colonnes du tampon est plus grand que le
        // nombre de colonnes de la matrice d'énergie
        Pointer la tête de Lect./Ecrit. sur le premier enregistrement demandé ;
        Pour i := 0 jusqu'à MAX_ROW //copier MAX_ROW x 2^n éléments
        Faire
            fread (Tampon [i], sizeof (Noeud), 2^n, cost_matrix );
        Fait
    Sinon
        Pointer la tête de Lect./Ecrit. sur le premier enregistrement demandé ;
        Pour i := 0 jusqu'à MAX_ROW
            //copier MAX_ROW x MAX_COLUMN éléments
        Faire
            fread(Tampon [i], sizeof (Noeud),MAX_COLUMN, cost_matrix );
            fseek (cost_matrix , (2^n – MAX_COLUMN) * sizeof (Noeud),
                SEEK_CUR);
        Fait
    FSi
FSi
Fixer les paramètres indiquant la partie copiée dans le tampon ;
FSi
FIN
```

Exemple :

On donne un exemple pour n=2 bits (n étant le nombre de bits avant codification) et lambda = 3(un facteur technologique). Codons les données sur m = 3 bits puis transmettons-les sur un bus de 3 lignes.

La consommation en énergie due aux différents changements d'état de bus, la consommation totale, les différents états de bus pendant les différentes étapes de transmission et tous les résultats sont représentées dans ce qui suit :

```
Donnez n      : 2
Donnez lambda: 3
*************************************************************
*état courant  *donnée  *état prochain *dynamique     *parasit
*************************************************************
*000           * 11      * 001          * 0.50 CL      * 3 CL
*001           * 10      * 111          * 1.00 CL      * 0 CL
*111           * 01      * 000          * 1.50 CL      * 0 CL
*000           * 11      * 001          * 0.50 CL      * 3 CL
*001           * 01      * 000          * 0.50 CL      * 0 CL
*000           * 11      * 001          * 0.50 CL      * 3 CL
*001           * 10      * 111          * 1.00 CL      * 0 CL
*111           * 00      * 111          * 0.00 CL      * 0 CL
*111           * 01      * 000          * 1.50 CL      * 0 CL
*000           * 01      * 111          * 1.50 CL      * 0 CL
*111           * 10      * 011          * 0.50 CL      * 3 CL
*011           * 11      * 001          * 0.50 CL      * 3 CL
*001           * 10      * 111          * 1.00 CL      * 0 CL
*111           * 11      * 110          * 0.50 CL      * 3 CL
*110           * 11      * 100          * 0.50 CL      * 3 CL
*100           * 10      * 111          * 1.00 CL      * 0 CL
*111           * 00      * 111          * 0.00 CL      * 0 CL
*111           * 10      * 011          * 0.50 CL      * 3 CL
*011           * 00      * 011          * 0.00 CL      * 0 CL
*011           * 00      * 011          * 0.00 CL      * 0 CL
*011           * 11      * 001          * 0.50 CL      * 3 CL
*001           * 00      * 001          * 0.00 CL      * 0 CL
*001           * 11      * 101          * 0.50 CL      * 3 CL
*101           * 01      * 100          * 0.50 CL      * 0 CL
*100           * 10      * 111          * 1.00 CL      * 0 CL
```

*111	* 10	* 011	* 0.50 CL	* 3 CL
*011	* 10	* 000	* 1.00 CL	* 0 CL
*000	* 11	* 001	* 0.50 CL	* 3 CL
*001	* 11	* 101	* 0.50 CL	* 3 CL
*101	* 11	* 001	* 0.50 CL	* 0 CL

-RESULTAT FINAL:

-L'énergie ch/dech: 18.50 CL

-L'énergie parasite: 39 CL

-TOTAL 57.50 CL

3.4. Méthode 2 : Codification Avec Probabilité Fixe

Description de la méthode :

Dans cette méthode on reçoit des données codées sur n bits, on les code sur m bits comme il est indiqué dans 2.3.1, puis on les envoie sur une interconnexion de m lignes. Il est à noter que tenant compte des prédictions des valeurs des données transitant sur le bus, des probabilités fixes sont associées aux différentes valeurs des données afin d'en faire un bon usage pour la réduction de l'énergie : plus la probabilité associée à une donnée est forte, et plus l'énergie associée au couple $(c_i,c_j) \in B^m \times B^m$ codant cette donnée est faible.

3.4.1. Algorithme de la méthode

```
Algorithme Méthode_Avec_Codification_Et_Probabilité_Fixe
Var
    Cod, data: char*;
    //tableaux de caractères pour contenir les codes à transmettre
    ancien_vect, nouv_vect : integer *; // contiennent les différents états de bus
    vnew, vold : integer;
    Nbr, S2 : integer init 0; // S2: énergie parasite
    S, S1 : real init 0;          // S1:énergie dynamique
    R : RECORD ; //sert à sauvegarder les informations concernant le codage
    R1: RESULTAT;  // contient l'énergie consommée
Début
  Ecrire ("donnez n :", n); // nombre de bits avant la codification
  Ecrire ("donnez lambda :", lambda);  // lambda est un facteur technologique

 // Génération des codes aléatoires à transmettre
  Pour i := 0 jusqu'à DATA_NBR
  Faire
      Nbr := rand (); // générer un entier
      Cod := int2bin (Nbr, n, p2);  //convertir un entier en binaire
      Mettre le contenu de Cod dans Data;
  Fait
  Open (info_cod, L/E) ; // Ouverture du fichier info_ cod en mode Lect./Ecrit.
  Affecter les probabilités pour chaque occurrence de code ;
  Affecter les codes selon les probabilités et mettre les informations d'affectation dans
  R ;
  Write (R, size of (RECORD), 1, info_cod);
    //sauvegarder les informations de codage dans le fichier info_cod.inf

     Pour m := n + 1 jusqu'à n + 10
          //m étant le nombre de bits utilisés pour coder les données
          //on fait varier m de n+1 jusqu'à n+10
```

```
Faire
    S := 0; S1 := 0;  S2 := 0;   // Initialement, l'énergie consommée est nulle
    vold := 0;  //  L'état initial du bus est (000…0)
    Open (cost_matrix, L/E) ; // Ouverture du fichier des codes en mode L/E
    Open (result, L/E) ; // Ouverture du fichier résultat en mode L/E
    Remplir (cost_matrix, 2^m, 2^n); // Remplissage du fichier des codes

    Tant que (il y'a des données à transmettre)
     Faire
        ancien_vect := int2bin (vold, m, p1);
        Mettre la donnée courante dans Cod;
        Vnew := numéro du code affecté a la donnée courante dans la table
                d'affectation;
         //Charger le tampon dans le cas où les données dont on a besoin ne
         // s'y trouvent pas
        Charger_tampon () ;
        nouv_vect := int2bin (ENERGI_MAT [vold] [vnew].j, m, p1) ;
        S1 +:= ENERGI_MAT [vold] [vnew].e1; // S1 : énergie dynamique
        S2 +:= ENERGI_MAT [vold] [vnew].e2; // S2 : énergie parasite
        S +:= S1 + S2;// énergie totale
        vold := bin2int (nouv_vect, m);
     Fait

    R1.intrinseq := S1;
    R1.parasit := S2;
    Write (R1, size of (RESULTAT), 1, result);
     //Sauvegarde du résultat dans le fichier des résultats
    Close (cost_matrix);
    Close (result);    //fermeture des fichiers ouverts
Fait
 Close (info_cod);
FIN
```

3.4.2. Description de structures de données:

La structure Nœud :

Elle à la même structure que celle de la méthode 1.

La structure RECORD :

Cette structure contient les informations concernant l'affectation des codes (voir **Fig.3.7**).

```
struct RECORD {
                int d_valid;
                int f_valid;
                int *AFFECT_TAB;
               };
```

Fig.3.7. La structure RECORD

Où :

- **d_valid :** contient le numéro de la donnée à partir de laquelle on utilise la table de cet enregistrement pour le décodage.
- **f_valid :** contient le numéro de la donnée à partir de laquelle on n'utilise plus cette table pour le décodage. On utilise alors la table de l'enregistrement suivant.
- **AFFECT_TAB [ ] :** contient la table à utiliser pour le décodage.

Les variables :

En plus de celles de la méthode 1, nous avons utilisé la variable suivante :

float *prob : un tableau de taille 2^n contenant la probabilité de chaque occurrence de code tel que la case **prob (i)** contient la probabilité du mot **wi**.

Exemple :

Si les données avant la codification sont codées sur 3 bits, et le nombre des occurrences du mot 010 dans la liste des codes qu'on veut transmettre est 5, alors **prob** (2) = 5 / nombre des codes à transmettre.

3.4.3. Description des fichiers utilisés:

Le fichier cost_matrix : c'est le même que celui utilisé dans la méthode 1.

Le fichier resultat : c'est le même que celui utilisé dans la méthode 1.

Le fichier INFO_COD :

Ce fichier contient les informations concernant l'affectation des codes et est utilisé pour le décodage après la réception des données (voir **Fig.3.8**).

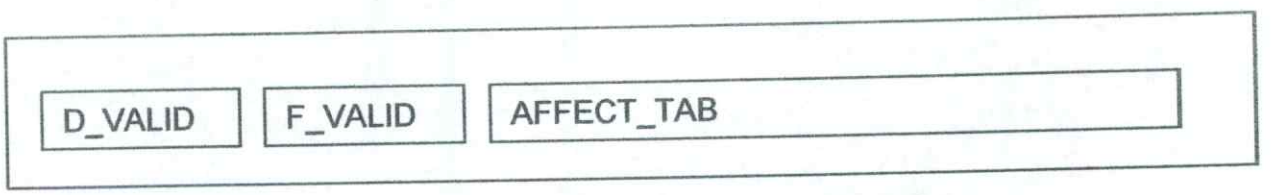


Fig.3.8. Format du fichier INFO_COD

Où :

- **D_VALID :** contient le numéro de la donnée à partir de laquelle on utilise la table AFFECT_TAB de cet enregistrement dans le décodage.
- **F_VALID :** contient le numéro de la donnée à partir de laquelle on utilise la table AFFECT_TAB de l'enregistrement suivant dans le décodage.
- **AFFECT_TAB :** contient la table d'affectation des codes.

Note :

Pour cette méthode on prend d_valid = 0 et f_valid = DATA_NBR car l'affectation des codes est la même pendant toute la période de transmission (probabilité constante).

3.4.4. Les fonctions de cette méthode :

int2bin (int elt, int Taille, int *pp) : c'est la même que celle de la méthode 1.

bin2int (char *T, int Taille) : c'est la même que celle de la méthode 1.

Facteur1 (int x, int y, int Taille) : la même que celle de la méthode 1.

Facteur2 (int x, int y, int Taille, int Lambda) : la même que celle de la méthode 1.

Remplir (FILE *str, int T1, int T2) : la même que celle de la méthode 1.

La fonction Trier : la même que celle de la méthode 1.

La fonction affecter () :

Cette fonction permet d'affecter les codes selon les probabilités des données: le code qui consomme le moins d'énergie est affecté a la donnée qui a la plus grande probabilité.

Algorithme

```
Algorithme Affecter ;
   Var y : integer ;
       x : real ;
Début
      Pour i:= 0 jusqu'à 2^n
         Faire
            R.AFFECT_TAB[i]:= i;
         Fait

      Pour  i:= 0 jusqu'à 2^n-1
         Faire
            Pour j:= i + 1 jusqu'à 2^n
               Faire
                  Si (prob [i] < prob [j]) Alors
                        y := R.AFFECT_TAB [i];
                        R.AFFECT_TAB [i] := R.AFFECT_TAB [j];
                        R.AFFECT_TAB [j] := y;
                        x := prob [i]; prob [i] := prob [j]; prob [j] := x;
                  FSi
               Fait
```

Fait

FIN

Exemple : Nous avons utilisé le même exemple que celui de la méthode 1. Les résultats ont été alors les suivants :

```
Donnez n      : 2
Donnez lambda: 3
*****************************************************************
*état courant  *donnée   *état prochain *dynamique     *parasit
*****************************************************************
*000           * 11      * 000          * 0.00 CL      * 0 CL
*000           * 10      * 111          * 1.50 CL      * 0 CL
*111           * 01      * 110          * 0.50 CL      * 3 CL
*110           * 11      * 110          * 0.00 CL      * 0 CL
*110           * 01      * 100          * 0.50 CL      * 3 CL
*100           * 11      * 100          * 0.00 CL      * 0 CL
*100           * 10      * 000          * 0.50 CL      * 0 CL
*000           * 00      * 100          * 0.50 CL      * 3 CL
*100           * 01      * 101          * 0.50 CL      * 3 CL
*101           * 01      * 001          * 0.50 CL      * 0 CL
*001           * 10      * 000          * 0.50 CL      * 0 CL
*000           * 11      * 000          * 0.00 CL      * 0 CL
*000           * 10      * 111          * 1.50 CL      * 0 CL
*111           * 11      * 111          * 0.00 CL      * 0 CL
*111           * 11      * 111          * 0.00 CL      * 0 CL
*111           * 10      * 000          * 1.50 CL      * 0 CL
*000           * 00      * 100          * 0.50 CL      * 3 CL
*100           * 10      * 000          * 0.50 CL      * 0 CL
*000           * 00      * 100          * 0.50 CL      * 3 CL
*100           * 00      * 111          * 1.00 CL      * 0 CL
*111           * 11      * 111          * 0.00 CL      * 0 CL
*111           * 00      * 011          * 0.50 CL      * 3 CL
*011           * 11      * 011          * 0.00 CL      * 0 CL
```

*011	* 01	* 001	* 0.50 CL	* 3 CL
*001	* 10	* 000	* 0.50 CL	* 0 CL
*000	* 10	* 111	* 1.50 CL	* 0 CL
*111	* 10	* 000	* 1.50 CL	* 0 CL
*000	* 11	* 000	* 0.00 CL	* 0 CL
*000	* 11	* 000	* 0.00 CL	* 0 CL
*000	* 11	* 000	* 0.00 CL	* 0 CL

-RESULTAT FINAL:

-L'énergie ch/dech: 15.00 CL

-L'énergie parasite: 24 CL

-TOTAL 39.00 CL

3.5. Méthode 3 : Codification Avec Probabilité Variable

Description de la méthode :

Dans cette méthode on reçoit des données sur n bits, on les code sur m bits (m>n) puis on les envoie selon la procédure suivante:

1. Affecter la même probabilité pour tous les codes ;
2. Envoyer N données;
3. Déterminer les probabilités de manière que le code qui engendre le moins d'énergie et la plus forte probabilité sont affectés à la donnée qui a été la plus transmise à l'étape précédente;
4. S'il y'a des données à envoyer, aller a 2, sinon aller a 5 ;
5. Fin.

3.5.1. L'algorithme de la méthode

```
Algorithme Méthode_Avec_Codification_Et_Probabilité_variable
Var
    Cod, data: char*; // tableaux contenant les codes à envoyer
    ancien_vect, nouv_vect : integer*; // contiennent les différents états de bus
    vnew, vold : integer; // contiennent les équivalents des états du bus
    Nbr, S2 : integer init 0;
    S, S1 : real init 0;
    R : RECORD ; // contient les informations de codage
    R1 : RESULTAT ; // contient le résultat

Début
Ecrire ("donnez n :", n); // nombre de bits avant la codification
Ecrire ("donnez lambda :", lambda); // facteur technologique
    // Génération de codes aléatoires à transmettre
    Pour i := 0 jusqu'à DATA_NBR
    Faire
        Nbr := rand (); // générer un entier
        Cod := int2bin (Nbr, n, p2); // conversion de l'entier en binaire
        Mettre le contenu de Cod1 dans Data ;
    Fait
Open (result, L/E) ; // Ouverture du fichier résultat en mode L/E

Pour m := n + 1 jusqu'à n + 10
    //m : nombre de bits sur lesquels on encode les données on le fait varier de
    //n+1 jusqu'à n+10
Faire
    S := 0; S1 := 0;  S2 := 0;    // Initialement, l'énergie consommée est nulle
    vold := 0;  //  L'état initial du bus est (000…0)
    Open (cost_matrix, L/E) ; // Ouverture du fichier des codes en mode L/E
    Open (info_cod, L/E) ; // Ouverture du fichier info_cod en mode L/E
    Remplir (cost_matrix, 2^m, 2^n); // Création de tous les codes possibles
```

```
Tant que (il y'a des données à envoyer)
Faire
    ancien_vect := int2bin (vold, m, p0);
    Mettre la donnée courante dans Cod;

    Si (Nbr des mots envoyés est égal à N) Alors
        Affecter les codes selon les probabilités;
        Mettre les informations d'affectation dans R;
        Write (R, size of (RECORD), 1, info_cod);
        Réinitialiser Nbr des mots envoyés à ZERO;
    FSi

    Déterminer wi, le nombre de fois que chaque code a été envoyé ;
    vnew := numéro du code affecté a la donnée courante dans la table
    d'affectation;
    // Charger le tampon si la donné demandée ne s'y trouve pas
    Charger_tampon () ;
    nouv_vect := int2bin (ENERGI_MAT [vold] [vnew].j, m, p1) ;
    S1 + := ENERGI_MAT [vold] [vnew].e1;
    S2 + := ENERGI_MAT [vold] [vnew].e2;
    S := S1 + S2;
    vold := bin2int (nouv_vect, m);
Fait

R1.intrinseq := S1 ;
R1.parasit := S2 ;
Write (R1, size of (RESULTAT), 1, result);
Close (cost_matrix); // fermeture des fichiers ouverts
Close (info_cod);
Fait
Close (result);
FIN
```

3.5.2. Description des structures de données:

Elles sont les mêmes que celles de la méthode 2.

3.5.3. Description des fichiers utilisés:

Ces fichiers sont les mêmes que ceux de la méthode 2.

3.5.4. Les fonctions de cette méthode :

Ce sont les mêmes que celles de la méthode 2.

Exemple : Nous avons utilisé le même exemple que celui des méthodes 1 et 2.

La ligne discrète (-------) indique le changement de la table du codage / décodage qui est déterminée en fonction des nouvelles probabilités.

```
- Donnez n      : 2
- Donnez lambda: 3
*****************************************************************
*état courant  *donnée   *état prochain *dynamique     *parasit
*****************************************************************
*000           * 11      * 001          * 0.50 CL      * 3 CL
*001           * 10      * 111          * 1.00 CL      * 0 CL
*111           * 01      * 000          * 1.50 CL      * 0 CL
*000           * 11      * 001          * 0.50 CL      * 3 CL
*001           * 01      * 000          * 0.50 CL      * 0 CL
*000           * 11      * 001          * 0.50 CL      * 3 CL
*001           * 10      * 111          * 1.00 CL      * 0 CL
*111           * 00      * 111          * 0.00 CL      * 0 CL
*111           * 01      * 000          * 1.50 CL      * 0 CL
*000           * 01      * 111          * 1.50 CL      * 0 CL
-----------------------------------------------------------------
*111           * 10      * 011          * 0.50 CL      * 3 CL
*011           * 11      * 111          * 0.50 CL      * 0 CL
*111           * 10      * 011          * 0.50 CL      * 3 CL
*011           * 11      * 111          * 0.50 CL      * 0 CL
*111           * 11      * 000          * 1.50 CL      * 0 CL
```

*000	* 10	* 100	* 0.50 CL	* 3 CL
*100	* 00	* 101	* 0.50 CL	* 3 CL
*101	* 10	* 111	* 0.50 CL	* 0 CL
*111	* 00	* 110	* 0.50 CL	* 3 CL
*110	* 00	* 100	* 0.50 CL	* 3 CL
*100	* 11	* 111	* 1.00 CL	* 0 CL
*111	* 00	* 000	* 1.50 CL	* 0 CL
*000	* 11	* 100	* 0.50 CL	* 3 CL
*100	* 01	* 101	* 0.50 CL	* 3 CL
*101	* 10	* 101	* 0.00 CL	* 0 CL
*101	* 10	* 101	* 0.00 CL	* 0 CL
*101	* 10	* 101	* 0.00 CL	* 0 CL
*101	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 11	* 011	* 0.50 CL	* 3 CL
*011	* 11	* 000	* 1.00 CL	* 0 CL

- RESULTAT FINAL:

- L'énergie de ch/dech: 20.00 CL

- L'énergie parasite: 36 CL

- TOTAL 56.00 CL

3.6. Méthode 4 : Notre méthode

Description de la méthode :

Le principe de cette méthode est le suivant :

1. Réception de N données sur n bits ;
2. Sauvegarde des données dans un tampon et codage ;
3. Mise à jour du module de décodage ;
4. Envoi des données codées sur m bits ;
5. Décodage des données ;

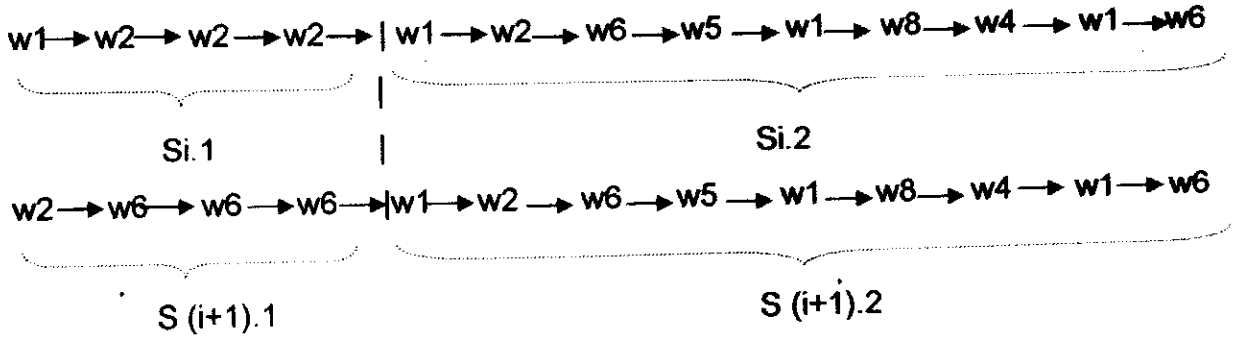
6. Réception des données sur n bits ;
7. S'il y'a des données à transmettre aller a 1 sinon aller a 8 ;
8. Fin.

En variant l'état initial du bus (avant de commencer la transmission des données, nous initialisons le bus à un état parmi les 2^m états possibles). Pour chacun de ces états initiaux, l'énergie consommée est déterminée tout en gardant trace de la plus faible. Cette dernière déterminera le codage et le décodage appropriés, c'est-à-dire ceux qui seront utilisés pour la transmission des données.

Notons que le flot de données à envoyer engendre une succession d'états déterminés selon le codage utilisé. Il est alors possible que la même suite d'états est rencontrée après un certain nombre d'itérations quand bien même deux états initiaux quelconques sont différents. Pour des fins de rapidité de traitement, cette observation est exploitée pour éviter de refaire les mêmes calculs pour des suites d'états identiques : Quand on est à l'itération i et que la donnée à envoyer est Dj, si on se trouve dans le même état que celui de l'itération i-1, alors au lieu de refaire tous les calculs, nous exploitons les résultats précédents comme il est indiqué dans le prochain paragraphe.

Soit **Si** la somme totale d'énergie consommée a l'itération **i**, et **S(i+1)** celle consommée a l'itération **(i+1)**, Si on note **Si.1**, **S(i+1).1** les sommes des énergies consommées dans les étapes où le traitement est différent et **Si.2**, **S(i+1).2** celles consommées où le traitement est identique, alors on fait le calcul comme il est indiqué dans l'exemple suivant :

Exemple1 :



S(i+1) = Si – Si.1 + S(i+1).1 . Ainsi, ceci nous évite de recalculer S(i+1).2

3.6.1. L'algorithme de la méthode

```
Algorithme Méthode_Exacte
Var
    Cod, data: char*; // contiennent les codes à envoyer
    ncien_vect, nouv_vect : integer*; // contiennent les différents états du bus
    vnew, vold : integer; // contiennent les équivalents des états du bus
    Nbr, S2 : integer init 0;
    S, S1 : real init 0;
    R  : RECORD ; // contient les informations de codage/décodage.
    N  : node; // contient le résultat

Début
Ecrire ("donnez n :", n); // nombre de bits avant codification
Ecrire ("donnez lambda :", lambda); // facteur technologique

    // Génération des codes aléatoires à transmettre
 Pour i := 0 jusqu'à DATA_NBR
 Faire
    Nbr := rand (); // génération aléatoire d'un nombre entier
    Cod := int2bin (Nbr, n, p2);  //conversion du nombre entier en binaire
    Mettre le contenu de Cod1 dans Data ;
 Fait

 Pour m := n + 1 jusqu'à n + 10
 // m : étant le nombre de bits utilisés pour le codage des données. Il varie de n+1
 // jusqu'à n+10
 Faire
    Min_Cout := + ∞ ;
    Open (cost_matrix, L/E) ; // Ouverture du fichier des codes en mode L/E
    Open (result, L/E) ; // Ouverture du fichier résultat en mode L/E
```

Open (info_cod, L/E) ; // Ouverture du fichier info_cod en mode **L/E**

Remplir (cost_matrix, 2^m, 2^n); // Remplissage du fichier des codes

Pour w := 0 **jusqu'à** 2^m

// w : état initial du bus avant l'envoi des données

Faire

// Initialisation de l'état de bus par un des 2^m états possibles

vold := w;

S := 0; S1 := 0; S2 := 0; // initialement, l'énergie consommée est nulle

Tant que (il y'a des données à transmettre)

Faire

1- **Si** (donnée à envoyer) **Alors** Réception de **N** données **Sinon** Aller a 15 ;

2- Sauvegarde des données dans un tampon et codage;

3- M.A.J du module de décodage;

4- Mettre les informations de décodage dans R;

5- **Write** (R, **size of** (RECORD), 1, info_cod);

6- ancien_vect := **int2bin** (vold, m, p1); // mettre état courant //dans vold

7- vnew := numéro du code affecté a la donnée courante dans la table d'affectation;

//Charger le tampon si la donnée demandée n'y est pas

8- **Charger_tampon** () ;

9- nouv_vect := **int2bin** (ENERGI_MAT [vold] [vnew].j, m, p1);

// Mettre état suivant dans nouv_vect

10- S1+ := ENERGI_MAT [vold] [vnew].e1;

11- S2 + := ENERGI_MAT [vold] [vnew].e2;

12- S + := S1 + S2;

13- **Si** (Transition déjà traitée) **Alors**

// Faire le calcul comme il est indiqué dans l'exemple 1

S := S0 – S0.1 + S1.1 ;

//S : énergie engendrée par l'état courant

//S0 : énergie engendrée par l'état précédent

```
                    //S1.1 : la partie où le traitement diffère de l'état
                    //précédent
                    Choisir un autre état initial;

                Sinon
                    vold := bin2int (nouv_vect, m);
                    //changement de l'état courant: Etat courant = Etat suivant
                FSi
            14- Si buffer non vide Aller à 6 sinon aller à 1;
            15- Sortir de la boucle ;
        Fait

        Si (S < Min_Cout) Alors
            Min_Cout := S ;
            N.e1 := S1 ;
            N.e2 := S2 ;
            N.ind_min := w ;
            Write (N, size of (node), 1, result);
        FSi

        Sauvegarder les états de cette itération;
    Fait

    Close (cost_matrix); // fermeture des fichiers ouverts
    Close (info_cod);
    Close (result);
Fait
FIN
```

3.6.2. Description des structures de données:

La structure Nœud : C'est la même que celle de la méthode 1.

La structure Node : Elle contient le résultat de l'exécution, sa structure est représentée dans **Fig.3.9.**

```
struct node{
        float e1;
        int e2;
        int ind_min;
        };
```

Fig.3.9. Format de la structure Node

Où :

- **E1** (resp. **E2**) **:** indique la somme de l'énergie **intrinsèque** (resp. **parasite**) consommée.
- **ind_min :** indique la valeur équivalente de l'état initial du bus qui correspond au résultat obtenu.

La structure Noeud1 :

Cette structure sert à la sauvegarde des résultats de chaque itération. On a donc un tableau de taille égale au nombre de données qu'on veut transmettre. Les informations concernant les quantités d'énergie consommée de la première donnée envoyée ainsi que celles des états courant et prochain du bus sont sauvegardées dans la première case du tableau, celles de la deuxième donnée dans la deuxième case du ce tableau, et ainsi de suite. Quand on envoie la première donnée, on la compare avec la première case du tableau, quand on envoie la deuxième on la compare avec la deuxième et ainsi de suite. Ceci est expliqué ci-après :

Première itération : on envoie la première donnée et on remplit tous les champs de la première case du tableau par les informations concernant la première donnée envoyée. Puis, on envoie la deuxième donnée et on remplit la deuxième case par les informations la concernant. L'énergie cumulée pour la première (resp. pour la deuxième donnée) est rangée dans le champ *e1cum* (respectivement *e2cum*). Le même procédé est utilisé pour toutes les autres données.

Deuxième itération : on envoie la i^{eme} donnée et on compare l'état prochain du bus qui sera atteint avec celui qui est stocké à la i^{eme} case du tableau. Si on

trouve que c'est le même état, alors on exploite les résultats de l'itération précédente stockée dans le tableau comme il est indiqué dans l'exemple1 précédent. Sinon, on modifie les informations du tableau comme il est indiqué dans la première itération.

On réitère les actions de la deuxième itération pour toutes les autres itérations qui restent.

La forme de la structure noeud1 est représentée dans **Fig.3.10**

```
struct noeud1 {
        Noeud n;
        float e1cum;
        int e2cum ;
        };
```

Fig.3.10. Format de la structure Noeud1

Où :

- **n** : indique une donnée de type nœud;
- **e1cum** : l'énergie **intrinsèque cumulée**;
- **e2cum** : l'énergie **parasite cumulée**;

La structure RECORD : C'est la même que celle de la méthode 3.

3.6.3. Description des fichiers utilisés:

Ce sont les mêmes que ceux du Méthode 3.

3.6.4. Les fonctions de cette méthode :

Elles sont les mêmes que celles de la méthode 3.

Exemple: Soit l'exemple utilisé pour les méthodes précédentes.

- Donnez n : 2
- Donnez lambda: 3

- w0= 000

*état courant	*donnée	*état prochain	*dynamique	*parasite
*000	* 11	* 111	* 1.50 CL	* 0 CL
*111	* 10	* 011	* 0.50 CL	* 3 CL
*011	* 01	* 011	* 0.00 CL	* 0 CL
*011	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 01	* 111	* 0.00 CL	* 0 CL
*111	* 11	* 000	* 1.50 CL	* 0 CL
*000	* 10	* 100	* 0.50 CL	* 3 CL
*100	* 00	* 101	* 0.50 CL	* 3 CL
*101	* 01	* 101	* 0.00 CL	* 0 CL
*101	* 01	* 101	* 0.00 CL	* 0 CL
*101	* 10	* 101	* 0.00 CL	* 0 CL
*101	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 11	* 011	* 0.50 CL	* 3 CL
*011	* 11	* 000	* 1.00 CL	* 0 CL
*000	* 10	* 000	* 0.00 CL	* 0 CL
*000	* 00	* 111	* 1.50 CL	* 0 CL
*111	* 10	* 111	* 0.00 CL	* 0 CL
*111	* 00	* 000	* 1.50 CL	* 0 CL
*000	* 00	* 111	* 1.50 CL	* 0 CL
*111	* 11	* 111	* 0.00 CL	* 0 CL
*111	* 00	* 011	* 0.50 CL	* 3 CL
*011	* 11	* 011	* 0.00 CL	* 0 CL
*011	* 01	* 001	* 0.50 CL	* 3 CL
*001	* 10	* 000	* 0.50 CL	* 0 CL
*000	* 10	* 111	* 1.50 CL	* 0 CL

*111	* 10	* 000	* 1.50 CL	* 0 CL
*000	* 11	* 000	* 0.00 CL	* 0 CL
*000	* 11	* 000	* 0.00 CL	* 0 CL
*000	* 11	* 000	* 0.00 CL	* 0 CL

-L'énergie ch/dech: 16.00 CL

-L'énergie parasite: 18 CL

-Total--> 34.00 CL

- w0= 001

*état courant	*donnée	*état prochain	*dynamique	*parasite
*001	* 11	* 000	* 0.50 CL	* 0 CL
*000	* 10	* 100	* 0.50 CL	* 3 CL
*100	* 01	* 100	* 0.00 CL	* 0 CL
*100	* 11	* 000	* 0.50 CL	* 0 CL
*000	* 01	* 000	* 0.00 CL	* 0 CL
*000	* 11	* 111	* 1.50 CL	* 0 CL
*111	* 10	* 011	* 0.50 CL	* 3 CL
*011	* 00	* 001	* 0.50 CL	* 3 CL
*001	* 01	* 001	* 0.00 CL	* 0 CL
*001	* 01	* 001	* 0.00 CL	* 0 CL
*001	* 10	* 001	* 0.00 CL	* 0 CL
*001	* 11	* 111	* 1.00 CL	* 0 CL
*111	* 10	* 111	* 0.00 CL	* 0 CL

Traitement exceptionnel ... (même suite d'états rencontrée)

-L'energie ch/dech : 15.50 CL

-L'energie parasite: 18 CL

-Total--> 33.50 CL

- w0= 010

*état courant	*donnée	*état prochain	*dynamique	*parasite
*010	* 11	* 011	* 0.50 CL	* 0 CL
*011	* 10	* 000	* 1.00 CL	* 0 CL
*000	* 01	* 000	* 0.00 CL	* 0 CL
*000	* 11	* 111	* 1.50 CL	* 0 CL
*111	* 01	* 111	* 0.00 CL	* 0 CL
*111	* 11	* 000	* 1.50 CL	* 0 CL
*000	* 10	* 100	* 0.50 CL	* 3 CL
*100	* 00	* 101	* 0.50 CL	* 3 CL
*101	* 01	* 101	* 0.00 CL	* 0 CL
*101	* 01	* 101	* 0.00 CL	* 0 CL
*101	* 10	* 101	* 0.00 CL	* 0 CL
*101	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 10	* 111	* 0.00 CL	* 0 CL

Traitement exceptionnel ... (même suite d'états rencontrée)

-L'energie ch/dech : 16.50 CL

-L'énergie parasite: 15 CL

-Total--> 31.50 CL

- w0= 011

*état courant	*donnée	*état prochain	*dynamique	*parasite
*011	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 10	* 011	* 0.50 CL	* 3 CL
*011	* 01	* 011	* 0.00 CL	* 0 CL

*011	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 01	* 111	* 0.00 CL	* 0 CL

Traitement exceptionnel ... (même suite d'états rencontrée)

-L'energie ch/dech: 15.0 CL

-L'energie parasite: 18 CL

-Total--> 33.00 CL

- w0= 100

*état courant	*donnée	*état prochain	*dynamique	*parasite
*100	* 11	* 000	* 0.50 CL	* 0 CL
*000	* 10	* 100	* 0.50 CL	* 3 CL
*100	* 01	* 100	* 0.00 CL	* 0 CL
*100	* 11	* 000	* 0.50 CL	* 0 CL
*000	* 01	* 000	* 0.00 CL	* 0 CL
*000	* 11	* 111	* 1.50 CL	* 0 CL
*111	* 10	* 011	* 0.50 CL	* 3 CL
*011	* 00	* 001	* 0.50 CL	* 3 CL
*001	* 01	* 001	* 0.00 CL	* 0 CL
*001	* 01	* 001	* 0.00 CL	* 0 CL
*001	* 10	* 001	* 0.00 CL	* 0 CL
*001	* 11	* 111	* 1.00 CL	* 0 CL
*111	* 10	* 111	* 0.00 CL	* 0 CL

Traitement exceptionnel ... (même suite d'états rencontrée)

-L'energie ch/dech: 15.50 CL

-L'energie parasite: 18 CL

-Total--> 33.50 CL

- w0= 101

*état courant	*donnée	*état prochain	*dynamique	*parasite
*101	* 11	* 100	* 0.50 CL	* 0 CL
*100	* 10	* 111	* 1.00 CL	* 0 CL
*111	* 01	* 111	* 0.00 CL	* 0 CL
*111	* 11	* 000	* 1.50 CL	* 0 CL
*000	* 01	* 000	* 0.00 CL	* 0 CL

Traitement exceptionnel ... (même suite d'états rencontrée)

-L'energie ch/dech : 17.00 CL

-L'energie parasite: 15 CL

-Total--> 32.00 CL

- w0= 110

*état courant	*donnée	*état prochain	*dynamique	*parasite
*110	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 10	* 011	* 0.50 CL	* 3 CL
*011	* 01	* 011	* 0.00 CL	* 0 CL
*011	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 01	* 111	* 0.00 CL	* 0 CL
*111	* 11	* 000	* 1.50 CL	* 0 CL
*000	* 10	* 100	* 0.50 CL	* 3 CL
*100	* 00	* 101	* 0.50 CL	* 3 CL
*101	* 01	* 101	* 0.00 CL	* 0 CL
*101	* 01	* 101	* 0.00 CL	* 0 CL
*101	* 10	* 101	* 0.00 CL	* 0 CL
*101	* 11	* 111	* 0.50 CL	* 0 CL
*111	* 10	* 111	* 0.00 CL	* 0 CL

Traitement exceptionnel ... (même suite d'états rencontrée)

-L'énergie ch/dech: 15.00 CL

-L'energie parasite: 18 CL

-Total--> 33.00 CL

- w0= 111

*état courant	*donnée	*état prochain	*dynamique	*parasite
*111	* 11	* 000	* 1.50 CL	* 0 CL
*000	* 10	* 100	* 0.50 CL	* 3 CL
*100	* 01	* 100	* 0.00 CL	* 0 CL
*100	* 11	* 000	* 0.50 CL	* 0 CL
*000	* 01	* 000	* 0.00 CL	* 0 CL
*000	* 11	* 111	* 1.50 CL	* 0 CL
*111	* 10	* 011	* 0.50 CL	* 3 CL
*011	* 00	* 001	* 0.50 CL	* 3 CL
*001	* 01	* 001	* 0.00 CL	* 0 CL
*001	* 01	* 001	* 0.00 CL	* 0 CL
*001	* 10	* 001	* 0.00 CL	* 0 CL
*001	* 11	* 111	* 1.00 CL	* 0 CL
*111	* 10	* 111	* 0.00 CL	* 0 CL

Traitement exceptionnel ... (même suite d'états rencontrée)

-L'energie ch/dech : 16.50 CL

-L'energie parasite: 18 CL

-Total--> 34.50 CL

```
-RESULTAT FINAL :

     -L'energie ch/dech: 16.50 CL

     -L'energie parasite: 15 CL

     -TOTAL--> 31.50 CL

     -L'indice de w0: w2
```

3.7. Conclusion :

Dans ce chapitre, nous avions présenté les techniques de base utilisées pour réduire la consommation de la puissance dissipée lors du transfert de données sur un bus. Nous avions programmé ces techniques pour des fins de comparaison à notre technique, celle qui a été présentée en dernier dans ce chapitre. Les résultats de cette comparaison sont abordés dans le chapitre qui suit.


CHAPITRE 4

4.1. Introduction :

La programmation des techniques de base pour la réduction de l'énergie dissipée sur un bus ainsi que celle de notre technique a été effectuée sur la plateforme suivante :

- PC équipé d'un Pentium ***2.4GHZ***, de ***256MO*** de RAM et de ***40GO*** de Disque Dur.
- Système d'exploitation ***Linux*** [15].
- Programmation en **C++** [16].

4.2. Résultats obtenus :

Les résultats obtenus sont ceux indiqués dans la table 4.1. Nous avions également programmé la méthode qui n'utilise aucune codification afin de pouvoir montrer l'intérêt et l'impact de la codification sur la dissipation d'énergie d'un bus.

La table 4.2 indique les résultats de la comparaison obtenus. Ces résultats seront discutés ultérieurement. Au préalable, nous discuterons de quelques détails techniques qui sont en rapport direct avec la taille des données codées.

La taille du fichier cost_matrix :

C'est le fichier des codes. Il contient $2^n \times 2^m$ enregistrements où chaque enregistrement contient (04) quatre champs (un champ chacun pour l'état courant, pour l'état prochain, pour l'énergie dynamique et pour l'énergie parasite). Le champ de l'énergie dynamique est de type réel tandis que les autres sont de type entier.

- La taille d'un entier (**int**) sous Linux est de (04) quatre octets.
- La taille d'un réel (**float**) sous Linux est de (04) quatre octets.

La taille d'un enregistrement = la somme des tailles de ces champs.

$= 4 + 3 \times 4 = 16$ octets

La taille du fichier = la somme des tailles de ces enregistrements

= la taille d'un enregistrement $\times (2^n \times 2^m)$.

$= 16 \times (2^n \times 2^m)$

Exemples :

Considérons quelques exemples où les valeurs de n et de m varient, puis calculons la taille du fichier **cost_matrix** selon en fonction de ces valeurs.

Pour n = 1 et m = 2→ la taille du fichier égale $16 \times (2^1 \times 2^2) = 16 \times 8 = 128$ *octets.*

Pour n = 1 et m = 3→ la taille du fichier égale $16 \times (2^1 \times 2^3) = 16 \times 16 = 256$ *octets.*

Pour n = 4 et m = 5→ la taille du fichier égale $16 \times (2^4 \times 2^5) = 16 \times 512 = 8$ *K octets.*

Pour n = 8 et m = 9→ la taille du fichier égal $16 \times (2^8 \times 2^9) = 2$ *M octets.*

Pour n = 8 et m = 10 → la taille du fichier égal $16 \times (2^8 \times 2^{10}) = 4$ *M octets.*

Pour n = 8 et m = 11 → la taille du fichier égal $16 \times (2^8 \times 2^{11}) = 8$ *M octets.*

Pour n = 8 et m = 17 → la taille du fichier égal $16 \times (2^8 \times 2^{17}) = 512$ *M octets.*

Et ainsi de suite ....

Ces exemples montrent clairement que pour des valeurs conséquentes de n et m, la taille du fichier **cost_matrix** devient importante. Afin de tenir compte des caractéristiques de notre PC (tailles de la mémoire principale et de la mémoire secondaire, fréquence du processeur), nous avions fixé n à 8 bits et fait varier m de n+1 à n+9 bits. Les résultats correspondant à l'énergie consommée pour chacune des quatre techniques présentées dans le chapitre précédent sont ceux indiqués dans la table 4.1 ci-après :

Table 4.1.Energie dissipée pour chacune des quatre techniques avec n = 8 bits et m variant de n+1 bits jusqu'à n+9 bits.

Méthode →	Méthode sans probabilité			Méthode avec probabilité fixe			Méthode avec probabilité variable			Notre méthode		
Energie consommée /CL	E1	E2	E1+E2	E1	E2	E1+E2	E1	E2	E1+E2	E1	E2	E1+E2
Nbr lignes après codification /bits ↓												
9	306.5	1221	1527.5	280.5	759	1039.5	303.5	1191	1494.5	179.0	315	494.0
10	307.5	1053	1360.5	295.5	708	1003.5	324.5	1038	1362.5	180.5	318	498.5
11	338.5	954	1292.5	323.0	663	986.0	350.0	921	1271.0	151.0	330	481.0
12	373.0	915	1288.0	317.5	654	971.5	352.0	900	1252.0	150.5	321	471.5
13	372.0	876	1248.0	301.5	642	943.5	355.0	846	1201.0	145.5	327	472.5
14	355.5	846	1201.0	298.0	639	937.0	357.5	843	1200.5	135.0	333	468.0
15	370.0	852	1222.0	291.5	642	933.5	354.5	816	1170.5	136.5	330	466.0
16	358.5	819	1177.5	302.0	639	941.0	334.5	801	1135.5	128.5	324	452.5
17	374.0	789	1163.0	301.5	612	913.5	339.5	789	1128.5	129.5	330	459.5

Nous avions également exécuté, sur le même exemple, la technique n'utilisant aucune codification. Les résultats obtenus sont alors les suivants :

- L'énergie intrinsèque: 355.0 CL

- L'énergie parasite: 1818 CL

- TOTAL : 2173.0 CL

La comparaison des quatre techniques par rapport à la méthode qui n'opte pas de codification montre que le codage apporte un gain en termes d'énergie. Ces gains sont déterminés par l'expression

$$100 * (\text{énergie\_méthode\_sans codification} - \text{énergie\_méthode\_considérée}) / \min(\text{énergie\_méthode\_considérée}, \text{énergie\_méthode\_sans codification})$$

et sont indiqués dans la table 4.2. Cette table montre qu'en général, les méthodes optant pour une codification donnent de meilleurs résultats que la technique n'utilisant aucun codage. Plus précisément, le gain obtenu en terme d'énergie totale par rapport à la méthode sans codification varie de

- 42.26% à 86.84% pour la méthode n'utilisant pas de probabilités
- 109.04% à 137.87% pour la méthode utilisant des probabilités fixes
- 45.40% à 92.59% pour la méthode utilisant des probabilités variables
- 335.91% à 380.22% pour notre méthode

Ces résultats montrent que l'énergie parasite est plus importante que l'énergie dynamique, d'où l'intérêt à lui apporter pour les technologies submicroniques.

Les résultats montrent aussi qu'en général, le gain augmente avec la taille utilisée pour le codage (réduction de l'énergie parasite) jusqu'à une certaine taille où le gain commence à diminuer à cause de l'augmentation de l'énergie dynamique qui commence à l' "emporter" sur l'énergie parasite. Ces résultats montrent aussi l'avantage certain de notre technique par rapport aux autres techniques basées sur

Table 4.2. Résultats de comparaison (pourcentage de Gain d'énergie) entre les différentes méthodes et la méthode sans codification.

Méthode →	Méthode sans probabilité			Méthode avec probabilité fixe			Méthode avec probabilité variable			La méthode Exacte		
Pourcentage de gain d'énergie / Nbr lignes après codification /bits ↓	E1	E2	E1+E2	E1	E2	E1+E2	E1	E2	E1+E2	E1	E2	E1+E2
9	+ 15.82	+ 48.89	+ 42.26	+ 26.56	+ 139.52	+ 109.04	+ 16.97	+ 52.64	+ 45.40	+ 98.32	+ 477.14	+ 339.88
10	+ 15.44	+ 72.64	+ 59.78	+ 20.13	+ 156.78	+ 116.54	+ 09.40	+ 75.14	+ 59.48	+ 96.67	+ 471.70	+ 336.91
11	+ 04.87	+ 90.56	+ 68.12	+ 09.91	+ 174.21	+ 120.38	+ 01.43	+ 97.39	+ 70.97	+ 135.10	+ 460.91	+ 351.77
12	- 05.07	+ 98.69	+ 68.71	+ 11.81	+ 177.96	+ 123.67	+ 00.85	+ 102.00	+ 73.56	+ 135.88	+ 466.35	+ 360.87
13	- 04.78	+ 107.53	+ 74.12	+ 17.74	+ 183.17	+ 130.31	+ 00.00	+ 114.89	+ 80.93	+ 143.98	+ 455.96	+ 359.89
14	- 00.14	+ 114.89	+ 80.93	+ 19.12	+ 184.50	+ 131.91	- 00.70	+ 115.66	+ 81.01	+ 162.96	+ 445.94	+ 364.32
15	- 04.22	+ 113.38	+ 77.82	+ 21.78	+ 183.17	+ 132.78	+ 00.14	+ 122.79	+ 85.65	+ 160.07	+ 450.91	+ 366.31
16	- 00.98	+ 121.98	+ 84.54	+ 17.55	+ 184.50	+ 130.92	+ 06.12	+ 126.96	+ 91.37	+ 178.26	+ 461.11	+ 380.22
17	- 05.35	+ 131.42	+ 86.84	+ 17.74	+ 197.06	+ 137.87	+ 04.56	+ 130.42	+ 92.59	+ 174.13	+ 458.91	+ 372.90

la même méthodologie de transfert de données sur un bus, ce qui dénote notre grande contribution pour la problématique posée.

Les résultats obtenus sont aussi illustrés par les graphes des Figures 4.1, 4.2 et 4.3 qui correspondent respectivement à l'énergie dynamique, l'énergie parasite et l'énergie totale. Pour le flot de données utilisé et pour une taille des données réelles égale à 8 bits, la plus faible énergie totale a été obtenue avec les longueurs de codes suivantes :

- 17 bits pour la méthode n'utilisant pas de probabilités
- 17 bits pour la méthode utilisant des probabilités fixes
- 17 bits pour la méthode utilisant des probabilités variables
- 16 bits pour notre méthode

Enfin, notons que ces graphes montrent nettement l'avantage de notre technique par rapport aux techniques restantes.

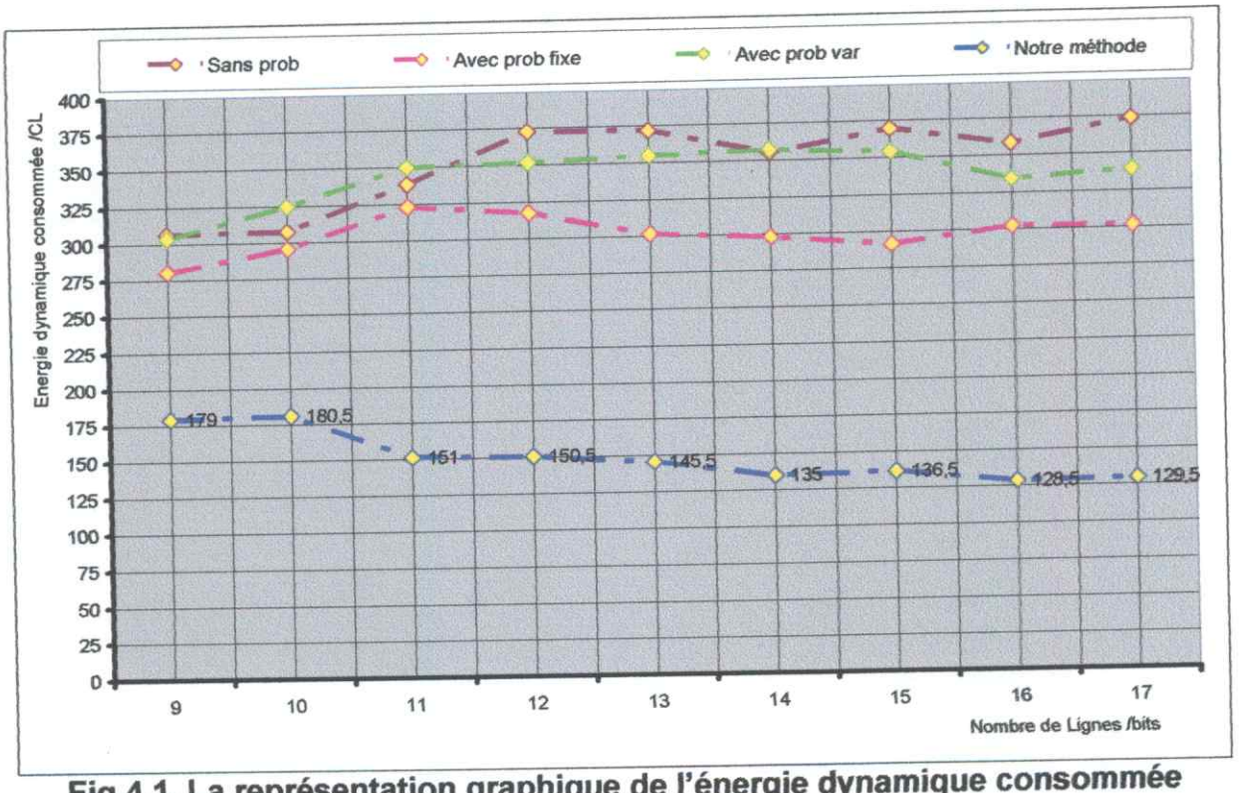


Fig.4.1. La représentation graphique de l'énergie dynamique consommée

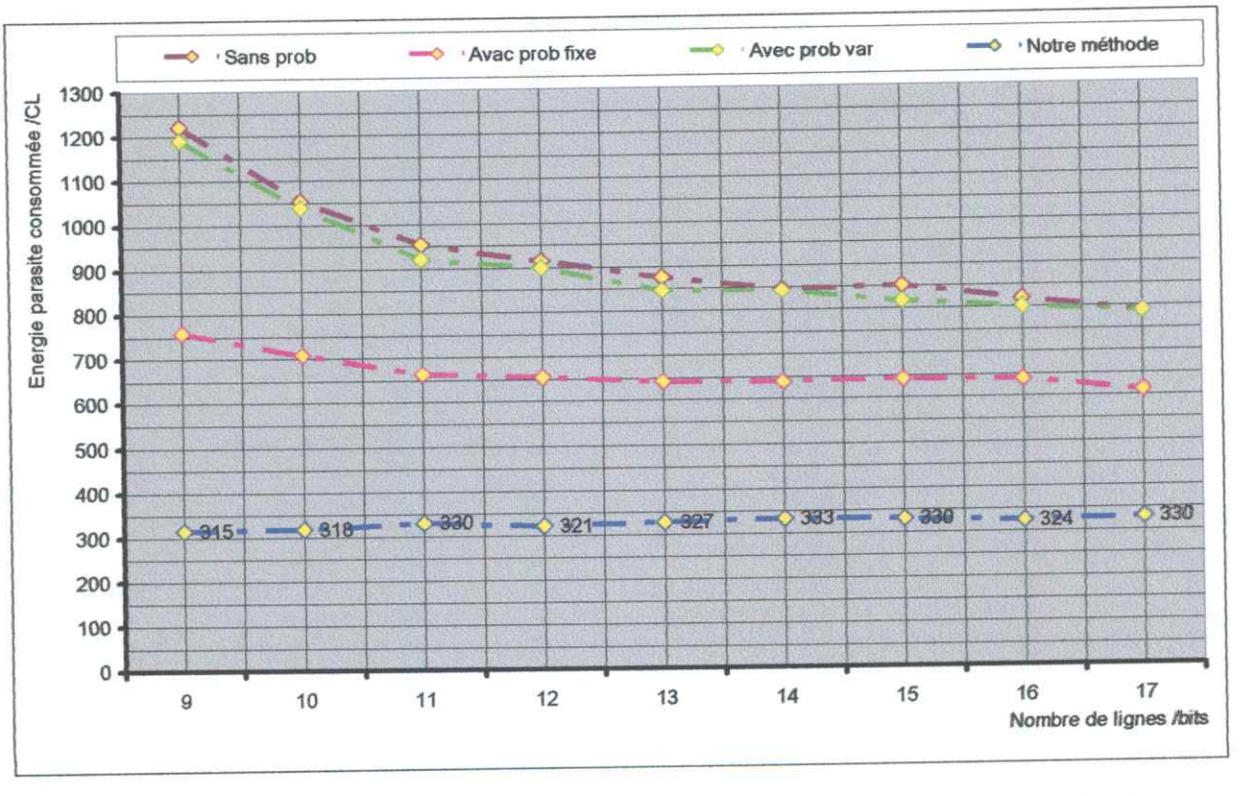


Fig.4.2. La représentation graphique de l'énergie parasite consommée

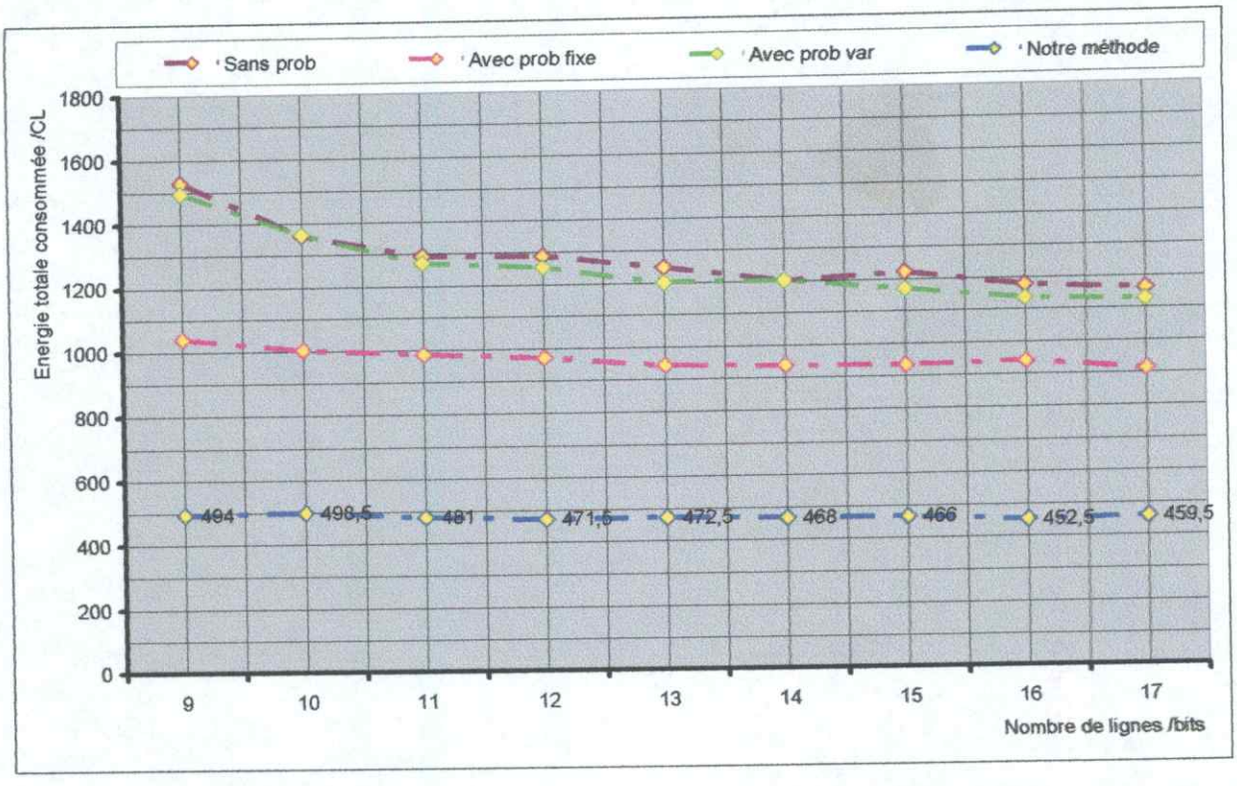


Fig.4.3. La représentation graphique de l'énergie totale consommée

4.3. Conclusion:

Dans ce chapitre, nous avions présenté les résultats obtenus avec le même flot de données pour différentes techniques de base que nous avions programmées ainsi que pour notre propre technique. Ces résultats ont montré que les techniques basées sur une méthodologie de codification engendrent de plus faibles consommations d'énergie par rapport à celle transmettant directement des données sur un bus. Ces résultats ont aussi montré que l'énergie due aux capacités parasites n'est pas négligeable pour les technologies submicroniques, ce qui nous oblige à en tenir compte. Enfin, parmi toutes les méthodes présentées, notre technique a été celle qui a donné les meilleurs résultats, ce qui dénote notre contribution pour la problématique posée.

CONCLUSION GENERALE

Ce mémoire a traité du problème de consommation d'énergie lors du transfert de données sur un bus. Nous avions jugé utile de donner quelques rappels et définitions nécessaires pour une lecture aisée du mémoire. Ceci a été suivi par la présentation des principales méthodes basées sur la codification, puis par celle de notre propre technique. Le dernier chapitre a été consacré aux résultats obtenus où une comparaison a été faite. Il a été montré que l'énergie due aux capacités parasites n'est pas négligeable pour les technologies submicroniques, et qu'il faudrait donc en tenir compte. Nous avions aussi vu que les techniques basées sur la codification engendrent une dissipation d'énergie totale (dynamique et parasite) moins importante que celle obtenue avec les méthodes n'optant pas pour une codification. Enfin, les résultats obtenus pour le même flot de données ont montré clairement que notre méthode est plus avantageuse par rapport aux techniques restantes, ce qui dénote que notre contribution à la problématique posée est appréciable.

Ce thème nous a été bénéfique dans la mesure où il nous a permis de mettre en pratique certaines de nos connaissances acquises durant notre cursus universitaire, et d'acquérir de nouvelles connaissances ayant trait à la conception des systèmes intégrés.


Bibliography

[1] Weste & Eshraghian "Principles of CMOS VLSI Design: A Systems Perspective" Addison Wesley, 1993

[2] C. Mead and L. Conway "Introduction aux Systèmes VLSI" Inter Editions

[3] M. R. Garey & D. S. Johnson "Computers and intractability: A Guide to the Theory of NP-Completeness" Freeman, San Francisco

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

[5] A. Mahdoum "SPOT: A Tool for Estimating the Maximal Switching Power Dissipation of CMOS Circuits and Data Paths" SASIMI'97 1-2 Dec.97, Osaka, Japan

[6] 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, Hôtel Sheraton, Alger

[7] A. Mahdoum "SPOT : A Tool for Estimating the Maximal and the Average Switching Power Dissipation of CMOS Circuits " Designer's Forum Proceedings, DATE'02, 4-8 Mars 2002, Palais des Congrès, Paris, France

[8] A. Mahdoum, A. Boutammine, D. Touahri, 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

[9] A. Mahdoum, M. L. Berrandjia "FREEZER2: Un Outil à Base d'un Algorithme Génétique pour une Aide à la Conception de Circuits Digitaux à Faible Consommation de Puissance" IEEE/FTFC'07, 21-23 Mai 2007, Paris, France

[10] A. Mahdoum, N. Badache, H. Bessalah "A Low-Power Scheduling Tool for SOC Designs" IEEE/IWSSIP'05, 22-24 Septembre 2005, Chalkida, Greece

[11] A. Mahdoum, N. Badache, H. Bessalah "An Efficient Assignment of Voltages and Optional Cycles for Maximizing Rewards in Real-Time Systems with Energy Constraints" JOLPE, Vol.2, N°2, August 2006, American Scientific Publishers

[12] P. P. Sotiriadis, A. P. Chandrakasan "A Bus Energy Model for Deep Submicron Technology" IEEE Transactions On Very Large Scale Integration, Vol.10, N°3, June 2002

[13] P. P. Sotiriadis, A. P. Chandrakasan "Bus Energy Reduction by Transition Pattern Coding Using a Detailed Deep Submicrometer Bus Model" IEEE Transactions On Circuits And Systems, Vol.50, N°10, October 2003

[14] A. R. Brahmbhatt, J. Zhang, Q. Wu, Q. Qiu "Low-Power Bus Encoding Using an Adaptive Hybrid Algorithm" DAC'06, 24-28 July 2006, San Francisco, California, USA

[15] Neil Matthew, Richard Stones "Programmation Linux" Editions SEYROLLES 2000

[16] J.R.NUBARD "Programmer en c++" 2eme édition