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

ECOLE NATIONALE POLYTECHNIQUE
DER Génie Electrique et Informatique
Département d'Electronique

Projet de fin d'études

pour l'obtention du diplôme d'Ingénieur d'Etat
en Electronique

Thème

FIABILITE DES SYSTEMES VLSI : TEST, TESTABILITE ET VERIFICATION DES MACHINES SEQUENTIELLES

Proposé et Dirigé par :

M r A.FARAH
et
M r H.BOUSBIA-SALAH

Présenté par :

M r BELAIFA SALAH SALIM

Promotion :

97 / 98

Résumé :

Ce travail comporte deux parties principales :

- la première partie a pour but de donner les fondements de base de la fiabilité , test , et vérification des circuits intégrés en particulier les systèmes VLSI .
- La deuxième consiste à réaliser un logiciel de simulation des méthodes de tests et de vérifications .

l'objectif essentiel est de vérifier la fonctionnalité d'un circuit intégré en un temps optimal (gain en temps) , en utilisant différentes méthodes et techniques de vérifications .

Abstract :

this work includes two main parts :

- The first part goal is to give the basic foundation of reliability , test and verification of integrated circuits , VLSI systems in particular .
- The second part consists in achieving a software simulation of tests and verifications methods .

The main objective is to verify the functionality of an integrated circuit in an optimal time , by using differents methods and technics of verification .

ملخص :

يتضمن هذا العمل جزئين رئيسيين :

- هدف الجزء الأول هو إعطاء مبادئ أساسية للفعالية ، الإختبار ، والتحقق من الدارات الإندماجية (CI) و خاصة أنظمة VLSI .
- بينما هدف الجزء الثاني هو إعداد برنامج يحوي طرق إختبار و تحقق .

الهدف الأساسي هو التحقق من عمل الدارات الإندماجية في زمن أمثل (ربح في الوقت) بإستعمال مختلف طرق و تقنيات التحقق .

Dédicaces

A la mémoire de mon très cher et brave père

A ma très chère mère

A mes soeurs et frères

A tous les amis et ceux qui me sont chers

Salim

Remerciements

Je commence par remercier ***M r Farah*** *et* ***M r Bousbia-Salah*** *pour leurs précieux conseils qui m'ont été bénéfiques dans mon travail.*

Que messieurs ***Mehenni*** *et* ***Sadoun*** *respectivement président et membre du jury trouvent ici mes remerciements pour avoir accepté de suivre et de juger mon travail.*

J'exprime également mes remerciements à toutes les personnes qui m'ont aidé et encouragé pendant les moments difficiles en particulier :

M r Amouri Mounir
et mes amis ***Riadh*** *et* ***Khalef***

Sommaire

Chapitre III : Vérification des circuits séquentiels

Chapitre IV :Simulation

Lexique

LEXIQUE

ASIC (application specific integrated circuit) *:désigne les circuits développées pour des applications particulières par opposition aux circuits standards.*

ATPG(automatic test pattern generation) *:logiciel permettant la génération automatique des séquences de test logique d'un ensemble décrit dans un fichier informatique*

AUTOTEST *: méthode de test dans laquelle ,moyennant l'addition de circuits spécifiques dans l'ensemble à tester, celui-ci est capable de se tester en autonome sans l'emploi d'un testeur ,on dit aussi qu'il est testable par lui même*

BILBO*(built in block observer) : circuit d'autotest intégré dans un VLSI*

CAO *: conception assistée par ordinateur*

CUSTOM CHIP *:circuit à la demande entièrement spécifique*

CIRCUIT OUVERT *:état d'une connexion qui ,pour une raison accidentelle, est maintenue ouverte*

COLLAGE*(à 1 ou 0) [stuck at 1 or 0] : liaison en défaut maintenue en permanence à 1 ou 0 .*

COURT-CIRCUIT *: liaison accidentelle entre deux connections qui devraient etre indépendantes.*

COUVERTURE DE TEST*(d'un sous-ensemble) :pourcentage de défauts de ce sous-ensemble qui peuvent etre détectés par une séquence de tests déterminée.*

DA *(design automation) : développement assisté*

DEFAUT *: anomalie physique d'un ensemble électronique pouvant entraîner une panne ou un fonctionnement altéré*

EDIF *(Electronic Design Interchange Format) : format d'échange (en cours d'adoption dans le monde entier)*

EXHAUSTIVITE D'UN TEST *: valeur calculée par simulation de défauts de la couverture de test, dont c'est une valeur théorique approchée*

FSM *: Finite state machnie*

LSFR (linear feedback shift register) :désigne un registre à décalage possédant des liaisons logiques entre sa sortie et certaines entrée de ses bascules lui conférant des propriétés utilisées en autotest

MODELES DE FAUTES : représentation abstraite des défauts les plus probables d'un ensemble électronique ,utilisée pour la préparation des tests de cet ensemble

PANNE :manifestation d'un défaut sur le fonctionnement d'un ensemble. le mot panne est en général utilisé dans le cas des pannes catastrophiques (arrêt complet)

ROUTAGE :définition des interconnexions à l'intérieur d'une puce ou d'une carte imprimée

SEQUENCE DE TEST : ensemble des vecteurs de test utilisés dans un ordre déterminé pour tester un sous-ensemble électronique

SIMULATION :analyse du fonctionnement d'un ensemble de circuits à l'aide d'un logiciel de CAO opérant sur une description abstraite de ces circuits
La simulation peut être électrique,logique, temporelle, fonctionnelle, thermique,.....

SIMULATION DE FAUTES(ou défauts) :simulation d'un ensemble de circuits dans lequel a été introduit un ou des défauts. Cette simulation de fautes sert à mesurer la qualité de séquences de tests.

SIGNATURE :compactage des signaux logiques observés en sorties d'un ensemble à tester et caractéristique du bon fonctionnement de cet ensemble. Un défaut modifie cette signature.

STIMULI :signaux appliquées à l'entrée d'un ensemble électronique pour le tester

STG : State transition graph

STRUCTURELLE :description d'un ensemble électronique qui présente les interconnexions entre les différents circuits utilisés (par opposition à comportemental)

TEST : mot très général qui désigne une technique, une discipline, une action, mais aussi le contenu de ce qui est appliqué pour tester un circuit ou un ensemble de circuits

TEST EN LIGNE : test d'un ensemble pendant son fonctionnement sans perturber celui-ci

TESTABILITE :aptitude d'un ensemble électronique à être testé

***VHDL** (VHSIC hardware description language)*

***VHSIC**(very high speed integrated circuit) : programme du DOD (ministère de la défense des Etats-Unis)*

***VLSI** (very large scale integration) :circuits intégrés contenant quelques dizaines de milliers de composants au minimum*

Introduction Générale

INTRODUCTION GENERALE

Il serait inutile de construire des ensembles électroniques complexes utilisant les technologies les plus avancées si l'on ne dispose pas d'un moyen de contrôler leur fonctionnement. La complexité d'un circuit VLSI est telle qu'un test manuel est devenu quasiment impossible. Les testeurs de VLSI sont devenus eux-mêmes des systèmes informatiques complexes et coûteux. La technologie des circuits intégrés passe progressivement de l'intégration à large échelle à l'intégration à très large échelle (de LSI à VLSI). Cette augmentation de complexité offre certains avantages : diminution du prix, augmentation de la fiabilité, réduction des contraintes d'interconnexion etc..... Par contre, la détection d'un circuit défaillant devient de plus en plus difficile et coûteuse à chaque étape de fabrication des composants, des modules et des systèmes. Ce problème doit être pris en charge dés la conception des circuits intégrés et des systèmes. Il constitue un défi redoutable à cause de l'impossibilité de procéder à un test exhaustif le plus efficace possible et dans une durée raisonnable; c'est un critère économique. Cela montre, d'une façon impérative, la nécessité de trouver des formules nouvelles de coexistence entre le fabricant des circuits intégrés, utilisateurs des circuits intégrés, fabricants d'équipements de test et chercheurs dans le domaine du test.

Dans le cadre de notre projet de fin d'études, nous avons décomposé notre travail en quatre parties essentielles :

La première partie comprend une étude sur la fiabilité des systèmes où d'abord l'aspect théorique (logique et mathématique) de la fiabilité a été traité, ensuite la fiabilité des systèmes électroniques a été présentée avec quelques données statistiques sur la fiabilité des circuits intégrés .

La deuxième partie traite du domaine du test et de la testabilité des circuits intégrés. Des algorithmes et des approches de tests y sont présentés .

La troisième partie traite de la vérification des circuits séquentiels, compte tenu de l'importance de la phase de conception d'un système et la facilité de modifier la structure ou l'architecture du système, durant cette phase; il est préférable de détecter une erreur et la corriger que dans une phase ultérieure.

Avec l'utilisation des outils de CAO, on peut vérifier un circuit donné pendant la conception pour le présenter par la suite à la fabrication sans défauts.

La quatrième partie est une simulation des méthodes de test et de vérification et une comparaison de ces différentes méthodes.

Chapitre I :
Fiabilité des systèmes

CHAPITRE I : FIABILITE DES SYSTEMES

I.1. INTRODUCTION :

Les premières études sur la fiabilité en électronique datent des années de la dernière guerre mondiale. Elles avaient été effectuées pour améliorer le comportement des équipements militaires de communication et de navigation. Elles se sont ensuite étendues aux équipements civils.

On peut définir la fiabilité ***R(t)*** d'un produit (reliability en anglais)comme étant la probabilité pour qu'il fonctionne avec des performances acceptables pendant une durée donnée dans des conditions d'emploi données.

$$R(t) = Ns(t)/N = \frac{\textit{Nombre d'éléments restant en survie}}{\textit{Nombre d'éléments à l'origine}}$$

Une autre définition de la fiabilité a été proposée par ***la commission électrotechnique internationale :***<<aptitude d'un dispositif à remplir une fonction requise ,dans des conditions données, pendent une durée donnée >>, cette notion de fiabilité est essentielle dans toute l'industrie moderne. Il faut aussi la rapprocher de la ***garantie*** offerte par les constructeurs pour toute sorte de produits : voitures, réfrigérateurs, postes TV etc.... Dans de tels cas, le fabriquant aura conçu les organes de son produit pour que la probabilité d'une panne durant la période de garantie soit la plus faible possible car il est dans l'obligation de remplacer gratuitement l'organe défectueux. Si la période de garantie est plus longue, le prix de vente du produit sera plus important car le constructeur sait que le ***vieillissement*** augmente le risque de panne.

Il exige alors que des périodes de révisions régulières soient instituées pour déceler à l'avance des signes de défaillances(maintenance préventive).

L'inconvénient, est que la révision d'une machine implique son immobilisation, ce qui n'est jamais souhaitable car cela revient très cher. il a donc toujours un compromis à réaliser entre fiabilité et période de révision (dans notre cas il faut réaliser un compromis entre la fiabilité des systèmes VLSI et le test).

A partir de la définition de la fiabilité elle permet de mesurer tout l'intérêt que l'on peut avoir à évaluer la fiabilité d'un dispositif :

1- Au niveau de la ***conception***, il est important de disposer de méthodes permettant de ***prédire la fiabilité*** pour répondre à un cahier des charges, choisir des solutions homogènes, réaliser un objectif au côut minimum (une meilleur fiabilité accroît les coûts de production mais permet des économies en exploitation).

2- Au niveau de la ***fabrication,*** il sera nécessaire de vérifier que les composants fabriqués ou employés répondent bien aux caractéristiques retenues lors de la conception.

3- Au niveau de ***l'exploitation,*** il est intéressant à plusieurs titres de contrôler les prédictions en matière de fiabilité :

- Vérifier que le système se comporte normalement et sinon trouver la cause des anomalies.
- Améliorer la connaissance des données de fiabilité. de plus les contraintes de fiabilité peuvent avoir une incidence sur l'exploitation du matériel par l'introduction de certaines procédures de tests ou par la nécessité de gérer un stock de pièces de sécurité.

Pendent longtemps le jugement de l'ingénieur, appuyé sur l'expérience, a suffi au niveau de la conception et de l'exploitation.

Nous pensons que ce jugement est toujours essentiel mais qu'il est utile qu'il puisse s'appuyer sur un ensemble de méthodes d'évaluation quantitatives et qualitatives de la fiabilité en raison de l'accroissement de la complexité et risques potentiels des systèmes (centrales nucléaires, ordinateurs, etc...).

Un examen plus détaillé de la définition de la commission électrotechnique internationales permet de mettre en évidence les notions importantes suivantes :

La notion de ***dispositif devant remplir une mission dans des conditions données***, nous appellerons systèmes les différents dispositifs dont on désire évaluer la fiabilité. Un système se définit comme un ensemble d'éléments en interaction. les éléments constituants le système sont susceptibles d'avoir des défaillances (c'est a dire ne plus être en mesure de remplir leur mission). La défaillance d'un élément peut être ***soudaine*** ou ***progressive, partielle ou totale.*** Les types de défaillance sont classés en modes de défaillances : en marche, à l'arrêt, à la sollicitation,... etc. Ces mêmes éléments peuvent également être ***réparables ou non réparables.*** Un élément sera dit réparable s'il est possible de lui restituer ses qualités primitives après défaillance sans qu'il soit nécessaire d'arrêter le fonctionnement du système. Un système est également susceptible de remplir plusieurs missions; lorsque sa mission est d'empêcher qu'une défaillance dont les conséquences sont catastrophiques ne se produise pas, La fiabilité est également appelée sécurité.

La notion de probabilité ; L'évaluation de la fiabilité d'un système nécessite donc des calculs de probabilités.

La notion de durée de la mission. La fiabilité apparaît ainsi comme une fonction du temps.

La notion de condition d'exploitation. On doit entendre par condition d'exploitation non seulement l'ensemble de l'environnement physique dans lequel le système doit remplir sa mission (température, degré d'humidité, pression ,etc...) mais également les modes de fonctionnements et la maintenance du système. Le système peut en effet être en marche permanente ou bien se trouver normalement à l'arrêt et être sollicité dans des conditions données. Dans le premier cas les défaillances des éléments sont généralement détectées immédiatement, ce qui n'est pas le cas dans la deuxième hypothèse.

En résumé ,nous pouvons donner une expression mathématique de la fiabilité ***R(t)*** d'un système ***S*** devant accomplir une mission dans des conditions données **:**
R(t) = probabilité (***S*** non défaillant sur [*0, t*]). Il en résulte que *R(t)* est une fonction non croissante variant de 1 à 0 sur [0,∞].

La fiabilité n'est pas la seule probabilité intéressante liée au fonctionnement d'un système .nous nous intéresserons également aux deux probabilités suivantes :

- la disponibilité ***A(t)*** qui est la probabilité pour que le système *S* soit non défaillant à l'instant *t*. on remarquera que dans le cas de systèmes non réparables, la définition de *A(t)* est équivalente à celle de la fiabilité, ce qui explique les confusions fréquentes entre fiabilité et disponibilité.

$$A(t) = \text{probabilité (} S \text{ non défaillant à l'instant } t)$$

- La maintenabilité ***M(t)*** qui est le complément à 1 de la probabilité pour que le système ne soit pas réparé sur l'intervalle [0, *t*] sachant qu'il est défaillant à l'instant t=0.

$$M(t) = 1\text{-probabilité (} S \text{ non réparé sur } [0, t] \text{)}$$

Cette notion ne concerne que les systèmes réparables. *M(t)* est une fonction non décroissante variant de 0 à 1 sur [0,+∞].

I.2. RAPPELES ET DEFINITIONS :

La probabilité *F(t)* pour qu'un équipement tombe en panne durant une certaine période (probabilité de non survie) est évidemment $F(t) = 1\text{-}R(t)$ (en général, on prend $t=0$ pour le début de la période de test).

Une fonction importante s'obtient à partir de $F(t)$: c'est la probabilité du nombre de pièces défaillantes pendant une durée Δt :

$$f(t) = \Delta F(t)/\Delta t = R(t-\Delta t)-R(t)/\Delta t = -\Delta R(t)/\Delta t$$

C'est donc un signe prés, la dérivée de $R(t)$.

I.2.1. Taux de défaillances Z(t) :

C'est la probabilité de panne pendant une période Δt :

$$Z(t)= f(t)/R(t)$$

Avec $$f(t) = - dR(t)/dt$$

On aura : $$z(t)dt =- dR/R$$

ce qu'en écrit : $$I= -\int_0^{\infty} z(u)\, du = Ln\, R(t)$$

d'ou $$R(t)= exp[\int(-z(u)\, du)\,]$$

L'intégrale I s'appelle « risque cumulés » (cummulatives hazard).

La figure (**I.1**) représente la fonction $z(t)$: on l'appelle courbe en baignoire, au vu de son apparence. On y distingue trois zones :

La notion de MTBF est surtout applicable aux équipements qui peuvent être réparés (par remplacement du composant défectueux).

I.2.3. Principales lois de probabilités utilisées en fiabilité :

Nous présentons ici quelques propriétés des principales lois utilisées en fiabilité .

I.2.3.1. Lois discrètes :

I.2.3.1.1. Loi binominale :

Considérons un expérience dont l'ensemble des événements se réduit à {A,$\overline{A}$, ∅, Ω}. Soient p la probabilité de réalisation de A et (1-p) celle de $\overline{A}$.

La variable aléatoire discrète X représentant le nombre de réalisation de l'événement A au cours de n expériences est distribuée suivant une lois binominale de paramètres (p,n) telle que :

$$P\,(X=k) = n!\; p^k\,(1-p)^{n-k}/\,k!\;(n-k)!$$
$$0 \le k \le n$$
$$0 \le p \le 1$$

On en déduit la fonction de répartition :

$$F\,(k)=P(x \le k)=\sum^{i=n}_{i=0}\; n!\; p^k\;(1-p)^{n-k}\,/\;i!\;(n-i)!$$

puis :

$$E[X]= np$$
$$\sigma^2[X] = np\,(1-p)$$

L'expression de $P(k)$ renferme une série de termes du type p^k ; elle est valable à condition qu'il n'y ait pas de corrélation entre les essais (les essais sont donc indépendants). Pour utiliser l'expression , il faut toujours connaître le nombre d'essais n ainsi que la probabilité p de l'événement ; ce qui n'est pas toujours le cas.

I.2.3.1.2.Lois de poisson :

La loi de poisson est une loi à paramètre positif m définit par :

$$P(X=k)= e^{-m}\,m^k/k!$$

Cette distribution de poisson est en fait une approximation de la distribution binominale dans le cas ou m=np ; l'erreur commise est presque insignifiante lorsque n>20 et p<0.05.

La fonction de répartition :

$$F(k)=\sum^{i=k}_{i=0} e^{-m} m^i /i! = 1- \Gamma(k+1,m)/k!$$

avec : $$\Gamma(k+1,m)=\int_0^m t^k e^{-t} dt$$

d'ou : $$E[X]=m$$

$$\sigma^2[X]=m .$$

Exemple :

Un appareil possède une fiabilité de 0.98 pour une journée de fonctionnement. Quelle est la probabilité pour qu'il est utilisé pendant un mois (30 jours) ,il tombe en panne deux jours ?

Réponse :

La probabilité de panne est $p=1-0.98=0.02$.

alors $p(2)= 30!. 0.02^2 .0.98^{28} / 2!.28! =0.0988$ *(distribution binominale)*

Si on effectue le calcul par application de la distribution de poisson ,on trouvera avec $m=np=0.6$, $p(2)= 0.0987$.

on voit bien que l'erreur commise est minime .

I.2.3.2. Lois continues :

Le tableau ci-dessous regroupe les principales caractéristiques des lois utilisée en fiabilité ,Y(t), représentant la fonction définie par :

$$Y(t)=0 \quad t<0$$
$$Y(t)=1 \quad t\geq 0$$

lois	densité de probabilité	moyenne	variance
exponentielle	$\lambda e^{-\lambda t} Y(t)$ $\lambda>0$	$1/\lambda$	$1/\lambda^2$
normale	$\exp(-0.5(t-m/\sigma)^2)/\sigma(2\Pi)^{1/2}$, $\sigma>0$	m	σ^2
Lognormale	$\exp(-0.5(\underline{\log(t-a)})^2) / tb(2\Pi)^{1/2}$ b^2 $b>0$	$\exp(a+0.5b^2)$	$e^{2a+b}(e^b-1)$
De weibull	$\underline{\beta}\ (t-\gamma/\eta)^{\beta-1}\exp(-(t-\gamma/\eta)^{\beta})Y(t-\gamma)$ η $\eta>0$ $\beta>0$	$\eta\Gamma(1+1/\beta)+\gamma$	α
Gamma	$\underline{1}\ \lambda^{\beta} t^{\beta-1} e^{-\lambda t} Y(t)$ $\alpha>0$ $\Gamma(\beta)$ $\beta>0$	β/λ	β/λ^2
Du khi deux	$\frac{t^{(\nu-2)/2} \exp(-t/2)\ Y(t)}{2^{\nu/2}\Gamma(\nu/2)}$ ν entier >0	ν	2ν

où $\alpha = \eta^2 [\Gamma(2/\beta+1) - \Gamma^2(1+1/\beta)]$

Tableau ***(I.1) :Les principales caractéristiques des lois utilisée en fiabilité***

I.3.REPRESENTATION DE LA LOGIQUE D'UN SYSTEME :

I.3.1.Introduction :

Le premier problème rencontré par l'ingénieur fiabiliste dans l'étude de la fiabilité ou de la disponibilité d'un système est la description de ce système.

On recherchera la représentation la plus simple sans tenir compte de la méthode de calcul de la fiabilité et de la disponibilité.

Considérons donc un système de n éléments en interaction et devant remplir une fonction donnée. Plaçons-nous dans le cas ou chaque élément n'a qu'un nombre fini d'états.

On aura en général les deux cas suivants :

cas 1. L'élément est normalement en marche dans le système et il a deux états possibles : état de marche et état de panne.

cas 2. L'élément est normalement à l'arrêt dans le système et ne démarre qu'en secours d'un élément principal .

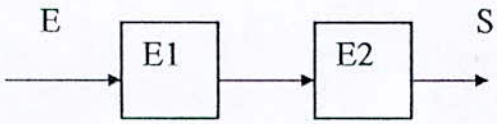
Il a alors quatre états possibles : en marche ,en réparation, à l'arrêt en bon état , à l'arrêt en panne .pour tenir compte de l'action extérieure sur le système , cataclysmes naturels, erreurs humaines,etc.... on introduit un certain nombre d'événements que l'on considère comme des éléments du système .ces éléments d'un type particulier prendront aussi en général un nombre fini d'états. finalement ,comme chaque élément admet un nombre fini d'états, le système général admet un nombre fini d'états.
or le nombre d'états du système croit exponentiellement avec le nombre des éléments. En effet, si chaque élément admet 2 états,le système admet 2^n états.

représenter la logique d'un système c'est représenter l'ensemble des états de fonctionnement et de non-fonctionnement du système et les liaisons entre ces différents états.

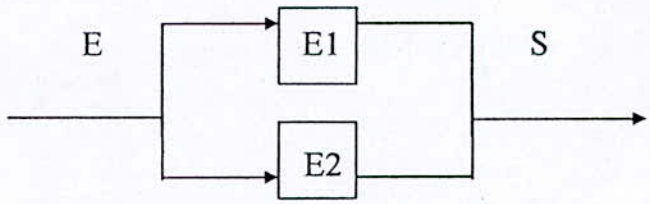
I.3.2.Diagramme de fiabilité :

C'est la représentation la plus naturelle de la logique de fonctionnement d'un système car elle est souvent proche du schéma fonctionnel du système.

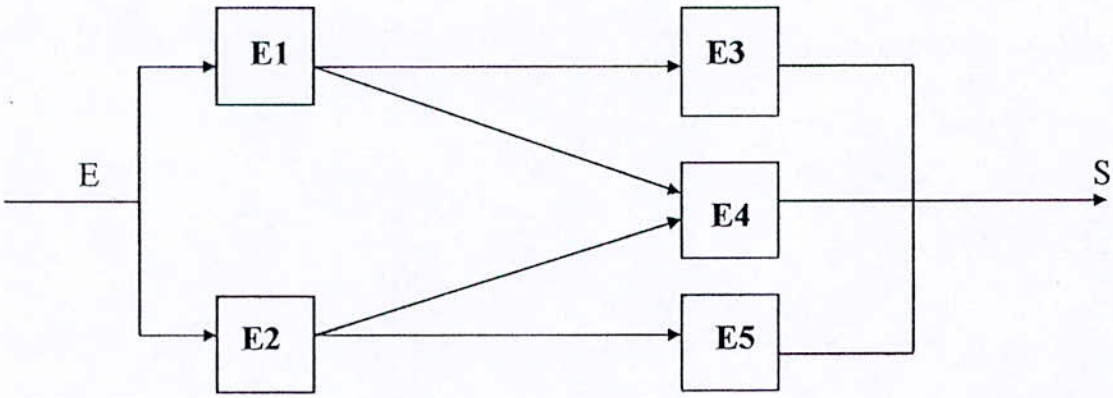
Dans cette représentation ,les blocs représentant des éléments (matériels ou événements) ou des fonctions dont la défaillance entraîne la défaillance du système sont placés en série , ceux dont la défaillance ne provoque la défaillance du système qu'en combinaison avec d'autre blocs sont disposés en parallèle sur ces derniers .



Figure(**I.2**) : ***Diagramme série***



Figure(**I.3**) : ***Diagramme parallèle***



Figure(**I.4**) : ***Diagramme complexe***

Le diagramme de fiabilité est donc un graphe sans circuit admettant une entrée et une sortie dont les sommets (appelés blocs) représentent les éléments du système et dont les arcs traduisent les relations entre différents éléments .

Le système fonctionne s'il existe un chemin de succès (successful path) entre l'entrée et la sortie du diagramme de fiabilité (reliability block diagram). La liste des chemins de succès permet donc de représenter l'ensemble des états de marche du système .

Dans l'exemple de la figure(I.4), il faut que les deux éléments d'un des chemins de succès E1 E3 , E1 E4 , E2 E4 , E2 E5 fonctionnent pour que le système fonctionne .

I.3.3. Arbre de défaillance :

Une des représentation de plus en plus utilisée de la logique d'un système est l'arbre de Défaillance (<< fault tree >>) connu aussi sous les noms d'arbre des défauts, d'arbre des causes ou d'arbre des fautes.

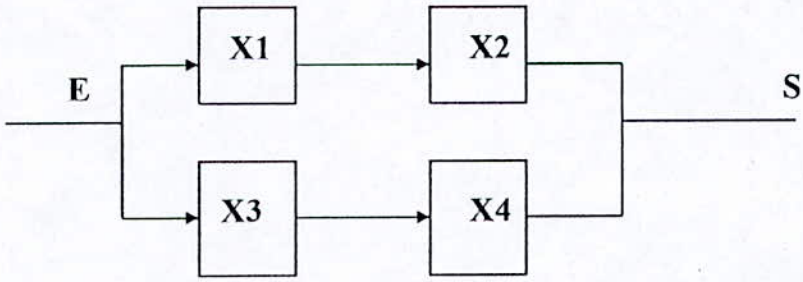
L'arbre de défaillance représentera graphiquement les combinaisons d'événements qui conduisent à la réalisation de cet événement indésirable .il sera formé de niveaux successifs tels que chaque événement soit généré à partir des événements du niveau inférieur par l'intermédiaire de divers opérateurs (ou porte) logiques.

successifs tels que chaque événement soit généré à partir des événements du niveau inférieur par l'intermédiaire de divers opérateurs (ou porte) logiques.

Ce processus déductif est poursuivi jusqu'à ce qu'on arrive à des événements de base, indépendants entre eux et probabilisables (même si l'estimation des probabilité est entachée d'incertitude).

Ces événements de base peuvent être des pannes ,des erreurs humaines, des conditions extérieures, etc...

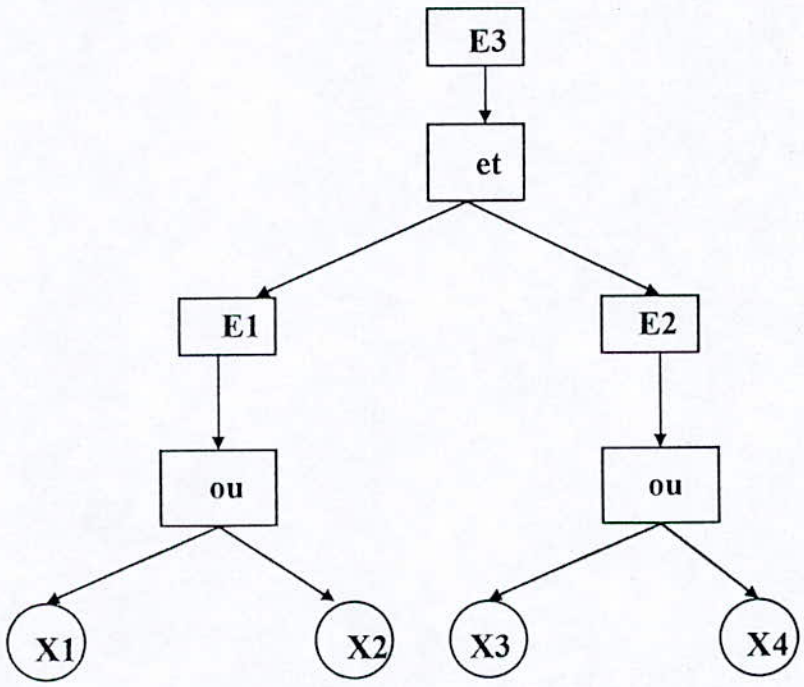
Considérons, par exemple, le système représenté par le diagramme de fiabilité suivant :



Figure(**I. 5**) : ***Exemple d'un diagramme de fiabilité***

Le système est en panne si les deux files sont en panne .

Un arbre de défaillance de ce système sera :



Figure(**I.6**) : ***Arbre de défaillance***

ou

E3 est la défaillance totale

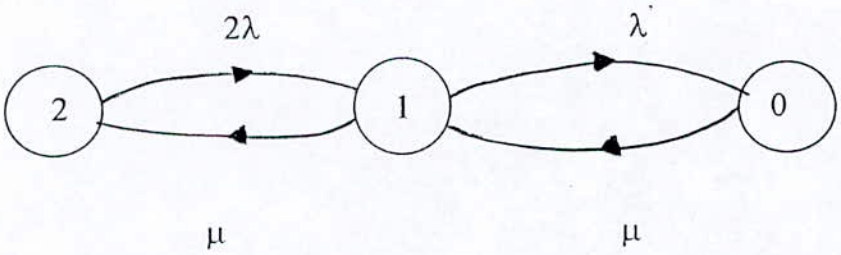
E1 est la première défaillance partielle

E2 est la deuxième défaillance partielle.

I.3.4. Graphe des états ou graphe de markov :

Pour tenir compte des dépendances entre les différents éléments d'un système, on construira un graphe dont les sommets correspondront aux différents états du système (si chaque élément a deux états : marche et panne et si le système à n éléments, le nombre maximum des états est 2^n) et dont les arcs correspondront aux transitions entre états. Sur ce graphe, chaque arc (i,j) est value par le taux de transition de l'états i à l'état j. Donnons-en un exemple très simple.

On considère un système formé de deux éléments identiques en parallèle. Lorsque les deux éléments fonctionnent ,ils ont chacun un taux de défaillance λ lorsque l'un des deux éléments tombe en panne, l'autre admet alors un taux de défaillance plus grand $\lambda' > \lambda$. Il n'y a de plus qu'un réparateur et le taux de réparation est μ . En notant 2 l'état du système ou les deux éléments fonctionnent, 1 l'état du système ou un seul élément fonctionne et 0 l'état du système ou les deux éléments sont en panne, on a le graphe des états suivant :



Figure(1.7) : ***Graphe des états***

Cette représentation permet de tenir compte de la dépendance statistique entre ces deux éléments.

Faisant quelques remarques sur la manière d'obtenir les taux de transitions entre les états .si la probabilité de passer de l'état i à l'état j entre les instants t et t+dt est $(\lambda_{ij}dt)$,alors λ_{ij} est le taux de transition entre les états i et j .

Lorsque les taux de transition entre les états sont constants, le système est markovien et le graphe des états sera souvent appelé le graphe de *markov.*

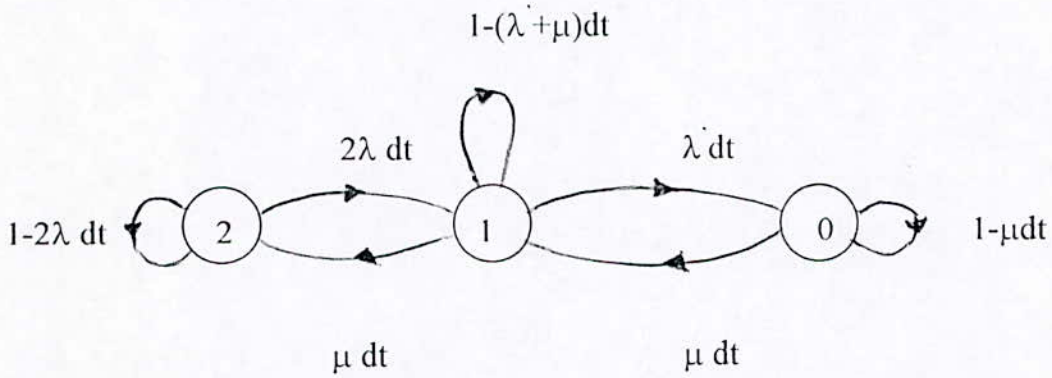
le taux de transition entre 2 et 0 est nul puisque la probabilité de passer de l'état 2 à l'état 0 entre t et t+dt est $(2\lambda\, dt)\,(\lambda' dt)=2\lambda\lambda'(dt)^2$.

Pratiquement cette représentation sera beaucoup plus lourde que les représentations précédentes. Elle ne sera pas donc utilisée que pour des sous-systèmes du système général à l'intérieur desquels il existe de fortes liaisons entre les éléments .

remarquons cependant que cette représentation sera très importante dans les calculs de fiabilité.

Remarque :

Un certain nombre d'auteurs donnent une représentation un peu différente des taux de transitions ; ils représentent sur chaque arc (i,j) la probabilité de passer de l'état i à l'état j entre les instants t et t+dt. Le graphe des états précédent devient alors dans cette représentation le graphe des états suivant :



Figure(I.7) : Graphe des états

I.4. FIABILITE DES SYSTEMES EN ELECTRONIQUE :

I.4.1. Détermination des taux de défaillances :

Les taux de défaillances en électronique sont constants.

Il existe des documents fournissant ces taux . Tels la norme de l'armée américaine « MIL-HDBK-217 » ou la norme MIL-STD-883.

Ces taux ont été déterminés après des séries de tests selon des procédures bien définies, telle la procédure proposée par le groupe agrée (american advisory group in reliability of electronic equipement) et adoptée pour la définition des taux de défaillances pour de nombreux équipements militaires et civils embarqués. la norme US MIL-HDBK-217 donne le taux de défaillances des composants pour 11 types d'environnement .

Les composants sont classés par type :

- composant passifs (résistances, selfs,.....)
  diodes , transistors ,.....
- circuits intégrés
- connexions (soudure, wrapping,.....)
  connecteurs
- autres

Le taux de défaillances est déterminé suivant le type de composant. Les contraintes d'emploi (température, lieu d'utilisation, etc.....).

Il est donné généralement sous la forme :

$$\lambda = \lambda_B \pi_E \pi_\Lambda \pi_Q \pi_e$$

le nombre de facteurs dépend du type de composant :

- λ_B est le taux de base , qui dépend de la température.
- π_E est le facteur d'environnement .
- π_Λ caractérise la tolérance du composant.
- π_Q dépend des tests du fabriquant, de l'encapsulation,...
- π_e dépend de la puissance de l'élément.

Le tableau ci-dessous donne quelques taux de défaillance pour des composants électroniques dans une ambiance à 40^0 et 70^0.

On remarque que $\lambda 70^0 > \lambda 40^0$, car on sait que si la température augmente, la défaillance augmente d'où $MTBF(40^0) > MTBF(70^0)$; finalement $\lambda 70^0 < \lambda 40^0$.

- Taux de défaillance de quelques familles de composants -		
Composants	**Taux de défaillance à 40^0C** en 10^{-9} par heure	**Rapport du taux de défaillance à70^0C et du taux de défaillance à 40^0C**
transistors		
au silicium (NPN)	7.5	1.3
au silicium (PNP)	11	1.4
à effet de champ	1.5	1.4
circuits intégrés		
bipolaires (DTL , TTL) 20 transistors	16	2.2
bipolaires (DTL ,TTL) 2000 transistors	900	2.9
MOS 20 transistors	30	6
MOS 2000 transistors	3000	7
Linéaire à 20 transistors	40	11
Mémoires à 1024 éléments binaires à lecteur seule ,TTL	340	3
à 1024 éléments binaires à lecture seule ,MOS.....	1100	7
à 1024 éléments binaires ,à accès direct ,TCC.....	600	3
à 1024 éléments binaires, à accès direct MOS.....	2000	7

Tableau(**I.2**) : ***Taux de défaillance quelque familles des composants***

Dans le cas des circuits intégrés par exemple, la loi s'écrit :

$$\lambda = \pi_e \pi_q \pi_l \pi_b$$

avec :

π_e : dépend de l'environnement dans lequel est utilisé le composant
π_q : représente la qualité du composant (1 à 300)
π_l : représente le facteur d'apprentissage pour le fabriquant (un composant fabriqué depuis longtemps est plus fiable qu'un nouveau composant).
$\pi_b = c_l \ \pi_t \pi_v \pi_e(c'+c'')$ avec ici :

c_l : dépend du nombre de circuits logiques
π_t : dépend de la température de fonctionnement
π_v : dépend de la tension de fonctionnement
c': dépend de la complexité du composant
c'' : dépend du nombre d'interconnexions vers l'extérieur et du type de boîtier

Le tableau ci-après présente à titre d'exemple des défauts trouvés dans un lot de circuits intégrés **LSI** après une certaine durée de fonctionnement (d'après YU . OVECHKIN) :

Cause de la panne	Contribution à la panne (%)	λ % 10^{-6} h^{-1}
Défaut dans les conducteurs et les liaisons	*33*	*0.970*
Défaut de métallisation	*26*	*0.765*
Effet de surface	*7*	*0.206*
Défaut de masquage	*18*	*0.529*
Défaut d'assemblage	*10*	*0.294*
Autres défauts	*6*	*0.176*
Total	*100*	*2.940*

Tableau(I.3) **:Défauts dans des circuits intégrés**

I.4.2.Calcul du MTBF d'un système électronique :

I.4.2.1.Méthode de calcul :

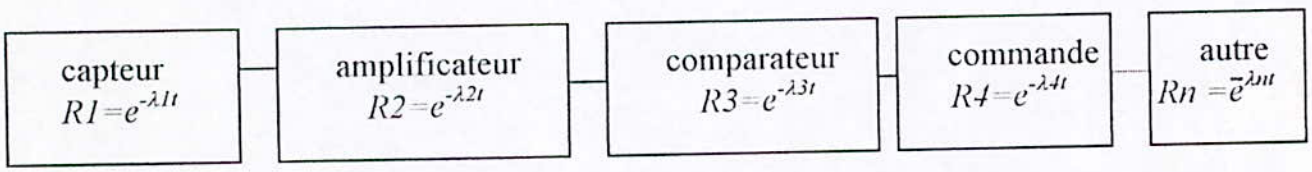
On procède de la façon suivante :

- on découpe le schéma électronique en blocs de fonctions élémentaires.
- on établit la nomenclature des composants pour chaque bloc.
- on indique le taux de défaillances pour chaque composant.
- on établit la somme des taux de défaillances pour l'ensemble des blocs.

On détermine enfin la probabilité de bon fonctionnement R(t) et le MTBF.

Exemple :

Quelle est la fiabilité du système dont le schéma synoptique est donné par la figure ci-dessous ?



Il est clair que si l'un quelconque des éléments de la chaine ne fonctionne pas, c'est tout le système qui s'arrêterait.

On a alors $R_{total}(t) = R1.R2.R3.....Rn$

et $\lambda_{total} = \lambda_1+\lambda_2+\lambda_3+.....+\lambda_n$

d'ou $M_{total} = 1/\lambda_{total}$

I.4.2.2.Augmentation du MTBF :

Il arrive souvent que la fiabilité d'un système électronique soit jugée trop faible . par exemple, une probabilité de bon fonctionnement de 80% au bout de quelques heures peut être (à juste titre) considérée comme trop faible s'il s'agit d'un appareil de surveillance d'organes dans une centrale nucléaire ou d'un appareil utilisé dans le bloc opératoire d'un hôpital etc...

Ne pouvant ni augmenter la fiabilité des composants ni la fréquence des périodes de révision, la seule solution consiste à doubler (ou tripler ou même d'avantage encore) les fonctions vitales d'un systèmes (c'est la redondance des fonctions).Ainsi, si l'une des fonctions tombe en panne, une seconde (ou une troisième ou un autre encore) assurera la relève.

des fonctions tombe en panne, une seconde (ou une troisième ou un autre encore) assurera la relève.

Prenons par exemple une redondance d'ordre 2 (*Figure*(**I.8**)):
la probabilité de non survie de la première fonction est $(1-R_1)$
d'ou pour l'ensemble (1-R1)(1-R2); la fiabilité est donc :

$$R=1-(1-R1)(1-R2)=R1+R2-R1R2$$

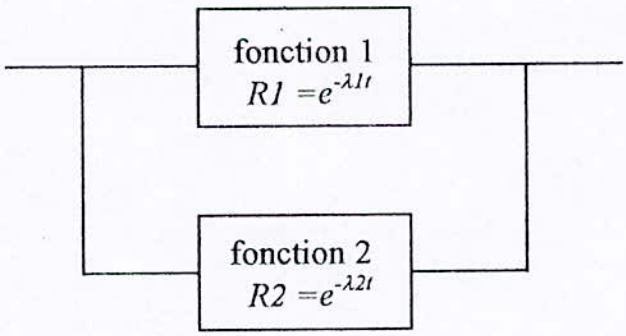


Figure (*1.8*) : ***Redondance de fonctions***

On aura dans le cas d'une redondance de système identiques :

$$R(t)=1-(1-R1)^2$$

d'ou $R(t)=1-(1-e^{-\lambda 1t})^2= 2e^{-\lambda 1t} - e^{-2\lambda 1t} =e^{-\lambda t}$

D'autre part on sait que $1\geq e^{-\lambda 1t} \rightarrow 2-e^{-\lambda 1t} \geq 1 \rightarrow (2-e^{-\lambda 1t})\, e^{-\lambda 1t} \geq e^{-\lambda 1t}$

On aura $2e^{-\lambda 1t} - e^{-2\lambda 1t} \geq e^{-\lambda 1t}$

Mais d'autre part $2e^{-\lambda 1t} - e^{-2\lambda 1t} =e^{-\lambda t}$.

En conclusion :

$e^{-\lambda t} \geq e^{-\lambda 1t} \rightarrow \lambda \leq \lambda 1 \rightarrow 1/\lambda \geq 1/\lambda 1 \rightarrow$ *MTBF≥MTBF1(augmentation du MTBF)*

*si la redondance est d'ordre n, on aura $R(t)=1-(1-R_1)^n$

Chapitre II :
Test et testabilité des circuits VLSI

TEST ET TESTABILITE DES CIRCUITS VLSI

II.1. INTRODUCTION :

Le test est un des aspects les plus importants de la conception et de la production des VLSI en particulier les ASICs. On distingue deux types de test différents :

- le test ***fonctionnel,*** pour vérifier le bon fonctionnement du circuit, autrement dit le respect du cahier des charges .
- le test en fin de ***fabrication***, pour vérifier que chaque puce ne contient pas de défauts.

Ces tests sont développés a partir de la simulation de CI durant sa conception.

Les logiciels de simulation font partie de la panoplie des outils disponibles dans la plupart des produits de CAO. Cette simulation peut être réalisée immédiatement après la saisie du circuit, auquel cas, chaque sous-ensemble est simulé séparément. Dans la plupart des systèmes , une simulation de charge peut être effectuée après l'implantation physique, de manière à prendre en compte les capacités réelles dues aux interconnexions. Ceci permet d'avoir une bonne idée du comportement de CI , dans un contexte opérationnel, une fois fabriqué.

Ces simulations sont conditionnées par des ensembles de vecteurs de test, qui sont appliqués successivement aux entrées du circuit pendant que l'on observe l'état des sorties et éventuellement l'état de certains noeuds internes. Le développement des vecteurs de test fonctionnel ne pose pas de problème. Ils sont essentiellement formés de configurations d'entrées typiques du fonctionnement opérationnel de CI plus quelques cas particuliers, afin de tester son comportement aux limites .

Il est beaucoup plus délicat de développer les vecteurs de test permettant d'identifier les défauts de fabrication, on parle également de vecteurs de test ***structurels*** dans la mesure ou ils reposent sur les propriétés structurelles de CI. Des vecteurs de test structurel non adaptés peuvent entraîner une non-détection de défauts de fabrication dans une puce , avec des conséquences potentiellement graves. Il est important de prendre en compte la testabilité de CI dès sa conception et d'y incorporer une stratégie de ***conception en vue de test*** ,encore appelée DFT " DESIGN FOR TESTABILITE "

II.2. LES FACTEURS PERMETTANT DE DETERMINER UNE STRATEGIE DE TEST :

Un aspect essentiel dans la gestion d'un projet de conception d'ASIC est de déterminer le niveau de couverture de test visé et de spécifier à quelles étapes les tests doivent être appliqués. Il ne suffit pas de dire qu'un composant doit être testé intégralement, une véritable stratégie de test doit être incorporée dans l'ensemble du cycle de conception. Le fabricant de CI va appliquer après fabrication, les vecteurs de tests fonctionnel et structurels développés par le constructeur.

Plus ces tests seront complets, plus le nombre de composants rejetés ultérieurement au cours du test du système final sera faible.

Cependant, trouver un ensemble de vecteurs de test qui permette une couverture de test complète (à supposer que ce soit théoriquement possible),peut prendre plusieurs fois le temps nécessaire au développement d'un ensemble de vecteurs que l'on pourrait juger raisonnablement suffisant. La stratégie de développement des vecteurs de test doit prendre en compte l'ensemble des facteurs intervenant dans le coût et donc notamment le temps de conception, le temps d'utilisation des ressources informatiques,la fabrication , le test des tranches, l'encapsulation, le test des circuits encapsulés , le test des cartes , le test du système .

II.3.LES EQUIPEMENTS DE TEST :

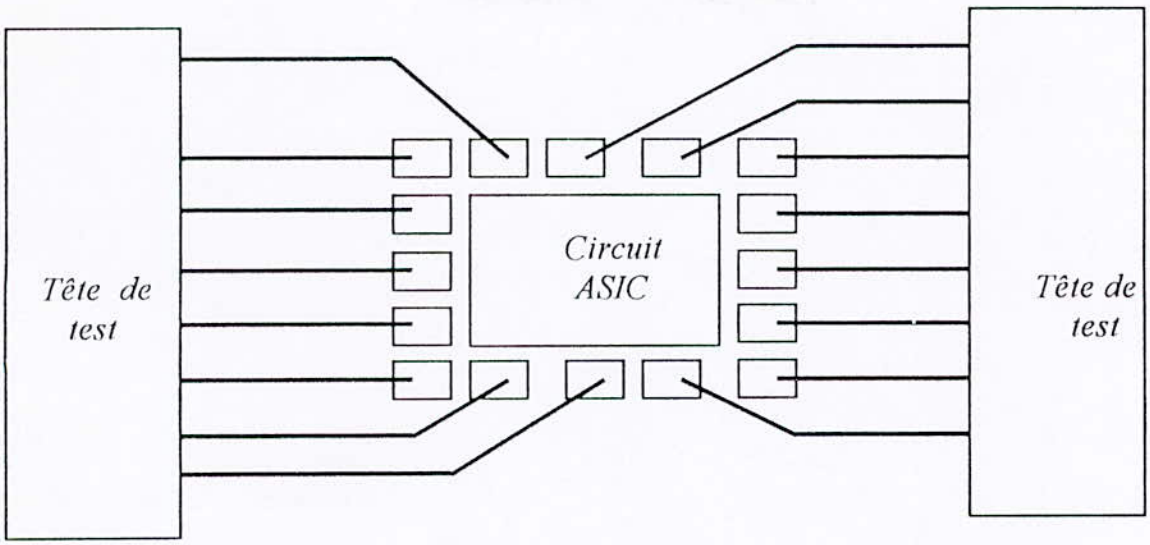
Les équipements de test utilisés par les fabricants d'ASICs possèdent un ensemble de sondes que l'on pose sur les plots du circuit. Ils permettent des tests **statiques**, des tests de **vitesse**, et un certain nombre de tests **électriques**.

Les vecteurs de test structurels développés pour les tests statiques doivent prendre en compte ce cycle de test. Ils comportent un ensemble de vecteurs d'entrée, à appliquer à intervalles réguliers sur l'ensemble des entrées au même instant ,et des valeurs attendues aux sorties du circuit , juste avant l'envoi du vecteur d'entrée suivant. Tout ceci ne prend pas compte le fonctionnement réel de l'ASIC qui peut avoir des entrées asynchrones ou fonctionner à des vitesses largement supérieures à celles des tests statiques.

En plus des tests statiques ,la plupart des fabricants d'ASIC proposent des tests de vitesse , au cours desquels ,l'ASIC fonctionne à pleine fréquence, avec des données de tests typiques. Ces tests mettent en évidence les problèmes dus aux désynchronisations d'horloge , et à d'autres phénomènes temporels.

Le dernier type de test est un ensemble de tests électriques, incluant une mesure des principaux paramètres électriques et des courants de fuite afin de vérifier l'intégrité physique du circuit.

Toutes les puces qui passent les tests sont encapsulées et ceux-ci sont renouvelés à partir des broches du boîtier pour mettre en évidence les défauts d'encapsulation.



*Figure(II.1) :**Equipement de test d'ASIC***

II.4. INITIALISATION :

Il est important qu'un équipement de test doit initialiser un ASIC dans un état bien défini, car les tests doivent se faire à partir d'une valeur connue dans chaque noeud. Comme il n'est pas possible de synchroniser le testeur à partir de l'ASIC, il faut employer d'autres méthodes. C'est une des raisons pour lesquelles on inclut généralement une entrée globale de remise à zéro dans les ASICs, de telle sorte que l'on puisse forcer les noeuds à un état connu sur le premier front d'horloge, ou au moins après un faible nombre de cycles.

Dans certains cas , il n'est pas possible d'initialiser l'ensemble du circuit sur le premier front d'horloge ;c'est notamment le cas par exemple si le circuit contient une RAM, ou ne contient pas de remise à zéro globale par économie de silicium. Pour mener à bien l'initialisation, on utilise le **nettoyage des états inconnus.**

Par exemple un long registre à décalage sans remise à zéro globale peut être nettoyé en utilisant le premier étage ,et en décalant petit-à petit cet état dans l'ensemble du registre. Les compteurs peuvent également servir à maintenir un état connu sur les entrées pendant que l'on nettoie le reste du circuit.

II.5. EVOLUTION DE LA COMPLEXITE DES CIRCUITS INTEGRES VLSI :

De nombreuses publications ont décrit l'accroissement de la complexité des VLSI et les conséquences sur la testabilité de machines contenant un million de transistors ou plus . Nous sommes passés d'une époque ou le test d'un circuit intégré était une tache subalterne à une époque ou il est devenu un souci majeur, car il est maintenant imposable de tester un VLSI en vérifiant son fonctionnement dans tous les cas possibles.

II.6. LE ROLE GRANDISSANT DE LA CAO(CONCEPTION ASSISTEE PAR ORDINATEUR) :

La manière traditionnelle pour tester un circuit logique numérique est de générer des vecteurs de test. Il s'agit d'appliquer une série de combinaison au calcul. Le problème du test des VLSI est qu'il est impossible d'appliquer toutes les combinaisons binaires dénombrables aux bornes du circuits. **Certaines combinaisons doivent être sélectionnées ; cette sélection constitue le problème fondamental du test.**

On admet généralement que, pour obtenir un test exhaustif, il faut trois vecteurs de test pour un transistor implanté dans le silicium. Il va sans dire que pour les circuits VLSI actuels la création des vecteurs de test ne peut plus être faite manuellement, le rôle de la conception assistée par ordinateur devient tout à fait fondamental. La testabilité et la génération automatique des vecteurs de test sont devenues aussi importantes et aussi coûteuses que le développement du VLSI proprement dit.

Le développement d'un VLSI comprenant 200 000 transistors peut coûter 6 mois à 1 an de travail de conception pour une personne et autant pour la création , la validation des tests et la conception des systèmes de test intégrés dans le VLSI (self test). Pour cette raison, tous les outils de conception de VLSI comprennent des simulateurs et générateurs de test. On voit se développer aujourd'hui des stations de travail qui prennent en charge tous les aspects ,depuis l'étude logique jusqu'à la validation finale de la fonctionnalité du VLSI.

II.7. CONCEPTION DES VLSI POUR LE TEST ET LE DIAGNOSTIQUE :

II.7.1.Methodes :

Concevoir un VLSI testable suppose la mise en place d'une véritable stratégie de test dès la conception de l'architecture même du circuit. La méthode consiste à établir des règles de testabilité qui seront des règles de conception à part entière.

Ces règles dépendent de nombreux facteurs tels que les outils CAO pour la génération des tests et les moyens de test. La complexité des VLSI est devenue telle qu'il n'est généralement pas possible de tester le circuit avec une méthode unique.
On peut envisager la mise en place d'une stratégie de test de la manière suivante:

a) **prise en compte de l'organisation du circuit** pour réaliser un découpage en unités testables ;
b) pour chaque unité : **définition de la méthode de test** qui sera appliquée, par exemple :
- génération de tests déterministes,
- développement d'algorithmes de tests pour une zone mémoire,
- implantation de modules d'autotest ;

c) **analyse de l'exhaustivité des tests** obtenue pour chaque unité avant la fin de la conception du circuit ;

d) **retour sur la conception du circuit** si nécessaire pour améliorer la contrôlabilité et l'observabilité , par exemple :
introduire un chemin d'accès par registre série (scan path)
introduire un multiplexeur pour pouvoir exécuter un algorithme de test au niveau d'une mémoire interne,
ajouter des moyens d'initialisations des registres, compteurs ,mémoires,
modifier une logique de commande pour pouvoir prendre le contrôle de générateurs ou de séquences.

II.7.2.Génération des tests :

Nous allons traiter ici les différents types de tests envisageables et évoquer la manière de les créer.

II.7.2.1. Test fonctionnel :

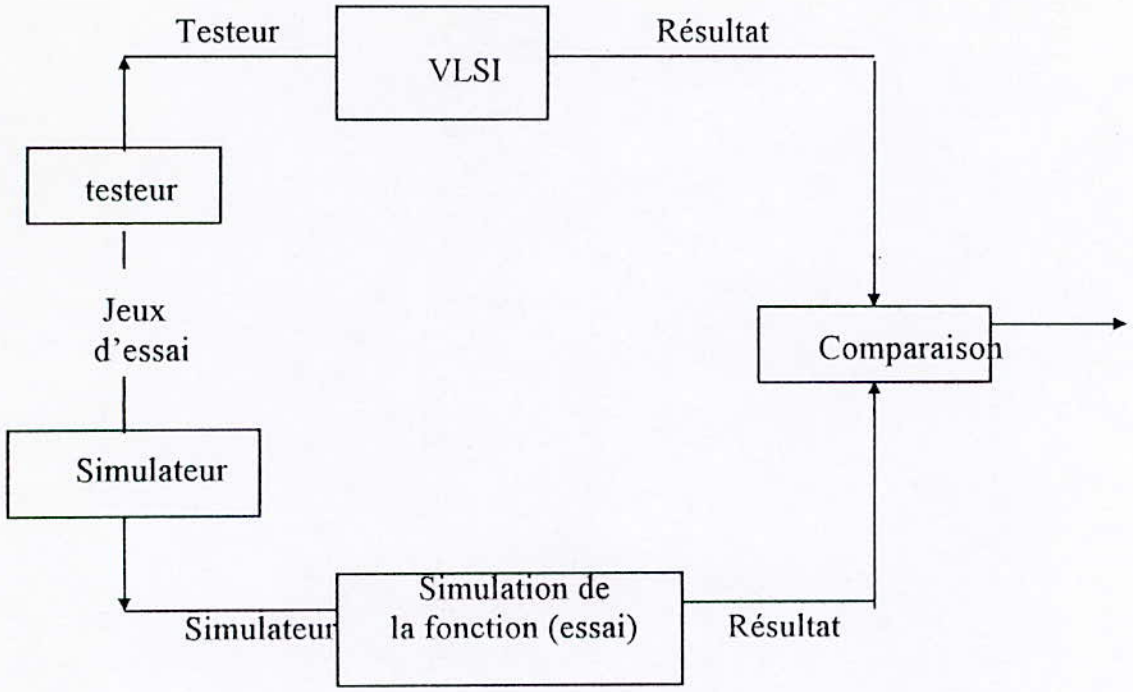
On peut dire que le test fonctionnel permet de répondre à la question : est-ce que le circuit fait ce que je veux qu'il fasse ?

Ce type de test n'est pas systématiquement mis en place au niveau du VLSI lui même. Bon nombre de VLSI ont été conçus sans passer par cette étape. le plus souvent, il faut attendre que le VLSI soit implanté dans le dispositif complet pour vérifier qu'il répond bien à son objectif.

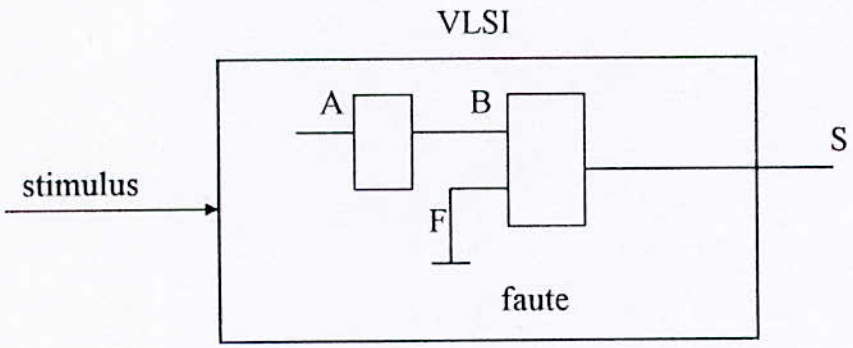
Cependant, pour réduire les temps de développement et de mise au point des VLSI, le test fonctionnel prend une place grandissante. L'essor des stations de mise au point liées directement aux outils de simulation CAO facilite le développement de ces tests et leur mise en œuvre sur les testeurs de VLSI.

Le principe consiste à écrire des jeux d'essais voir figure(*II.2*).Pour faire ces simulations, on utilise des programmes de simulation logique ou des simulateurs câblés (simulateurs hardware) spécialisés dans ce type de fonction. Dans un premier temps, les résultats de la simulation servent à vérifier que le circuit n'a plus de faute de conception avant de décider sa mise en fabrication.

Dans un deuxième temps, ces vecteurs issus de la simulation sont utilisés pour tester la fonctionnalité du circuit. L'exhaustivité des tests dépend de la qualité des jeux d'essais. En principe, il faut vérifier que toutes les fonctions du circuit sont correctement exécutées. A ce niveau ,il y a une difficulté pour estimer le taux de couverture de ce type de test. Il existe cependant des simulateurs de fautes qui permettre de vérifier l'exhaustivité des tests fonctionnels. Le principe consiste à introduire une faute dans le circuit. Le simulateur est actionné par les jeux d'essai précédent .il vérifie alors que la faute introduite est bien détectée.



*Figure(III.2) : **Test fonctionnel***



Le programme de génération :

1) suppose une faute en F

2) génère le stimulus pour ouvrir le chemin A B S

3) génère le test de la sortie S

F=0 ⟶ S=0

F=1 ⟶ S=1

Si la faute supposée en F existe réellement . une des combinaisons sera en erreur

*Figure(II.3) : **Test structurel***

II.7.2.2. Test structurel :

Le test structurel permet de répondre à la question est-ce que toutes les parties du circuit sont mises en jeu dans le test ?.

Pour générer ce test, on fait le plus souvent appel à des programmes de génération automatique (figure(*II.3*). Ces programmes partent d'un modèle de fautes du types collage à 1 ou 0. La technique consiste à déterminer le stimulus d'entrée qui fait se propager vers les sorties du circuit l'effet du défauts envisagés. pour cela, le programme analyse la structure du circuit (les équations logiques).Il peut arriver que ce type de programme ne trouve pas les vecteurs de test pour détecter toutes les fautes. On parle alors de couverture de test ou exhaustivité de test inférieure à 100%.

L'exhaustivité est le rapport du nombre de fautes détectables au nombre total de fautes.

Avec ce type de programme, il n'est généralement pas possible de traiter les circuits séquentiels ou mémorisants. En effet , l'aspect fonctionnel du circuit étant ignoré du programme ,il lui est impossible de créer une séquence (généralement complexe) pour positionner les éléments mémorisants. Cette limitation peut réduire considérablement l'efficacité des programmes de génération automatique.

II.7.2.3. Test des circuits séquentiels et des mémoires :

Pour tester les circuits séquentiels et plus particulièrement les mémoires incluses dans le VLSI ,il faut générer des tests d'un type particulier.

Il y a plusieurs raisons qui justifient ce besoin :

a) la logique n'est plus de type combinatoire pur ;

b) les modèles de fautes du type collage à 1 ou 0 ne s'appliquent plus ; une mémoire est un ensemble de cellules organisées de manière matricielle. cette organisation conduit à une forte densité de connexions rangées en lignes et colonnes . il devient hautement probable que les collages habituels se traduisent par **des courts-circuits** entre fils qui véhiculent l'état détectable que l'on a pris la précaution d'écrire des contenus opposés dans des cellules adjacentes. Pour cette raison, on fait appel, pour les tests mémoires, à des algorithmes de test qui portent des noms comme damier, maching, etc....

Le concepteur devra porter une attention particulière à l'accessibilité de la mémoire pour être sûr que ce type de test est exécutable à partir des bornes du VLSI.

II.7.2.4. Moyens mis en œuvre dans les VLSI :

On a vu que les techniques de génération de test avaient des limites. Pour contourner cette difficulté, on est amené à introduire la **testabilité** dans le silicium lui-même. Nous allons examiner quelques méthodes pratiquées aujourd'hui.

II.7.2.4.1.Chemin d'accès par registre à décalage (scan path) :

Le principe du registre à décalage comme organe d'accès et d'observation fut introduit vers 1964 pour le test des ordinateurs IBM 360.

A cette époque ,ce concept était plutôt utilisé pour tester des cartes de circuits logiques numériques. il n'est donc pas étonnant de voir cette technique s'appliquer au test des circuits intégrés , dès lors que la complexité des VLSI rejoint des celle cartes.

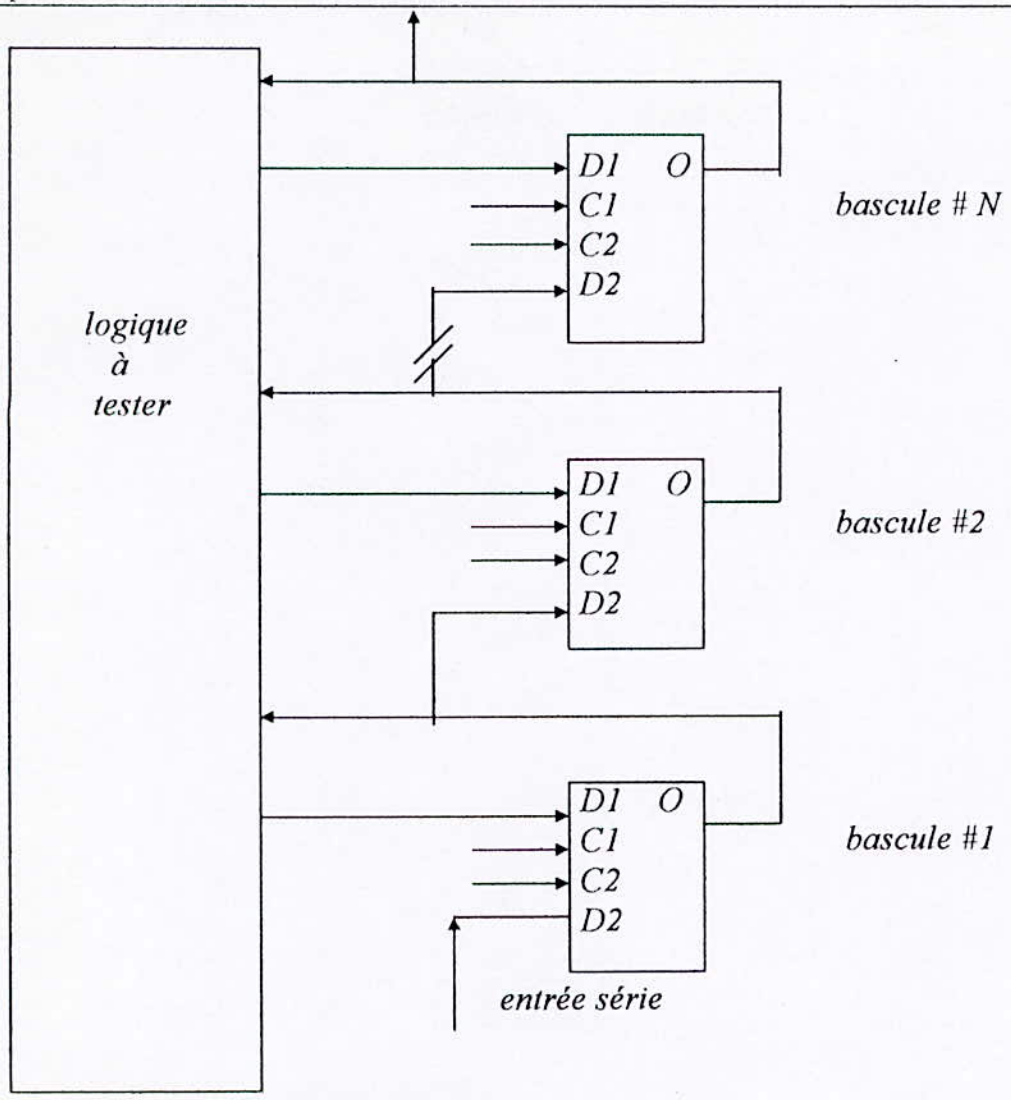
La figure (**II.4**) illustre le principe élémentaire mis en œuvre. La logique à tester est connectée à des bascules #1,#2,.....#N. Chaque point du VLSI à observer est relié à une borne de type D1 qui est chargée dans la bascule par l'horloge de type C1.chaque point du VLSI à commander est réuni à la sortie 0 d'une bascule.

Toute sortie 0 d'une bascule est connectée vers l'entrée de type D2 de la bascule de rang suivant. D2 est chargée dans la bascule par l'horloge de type
C2. L'ensemble des bascules #1 à # N constitue donc un registre à décalage changeable à partir de l'entrée série de la bascule #1 et observable à partir de la sortie série de la bascule # N.

Cette fonction conduit à un accroissement de la taille du VLSI qui varie entre 4 et 20% .la difficulté majeure est de conserver la performance dynamique du circuit. Il faudra en outre veiller à l'application pratique de ces tests sur les testeurs de VLSI. Surtout si les registres à décalage sont de grande longueur .
l'utilisation de la technique du scan path connaît de très nombreuses variantes .il est impossible de les citer toutes ici.
l'utilisation de la technique du scan path connaît de très nombreuses variantes .il est impossible de les citer toutes ici.

-LSSD de IBM : La plus connue est la technique du LSSD de IBM (level sensitive scan design). Elle fut introduire en 1977 par eichelberger et williams .le LSSD est basé sur l'utilisation de bascule RS maitre-esclave utilisant deux horloges décalées dans le temps .le terme level sensitive se réfère au type de logique utilisé .il s'agit de bascules sensibles à une transition.

-TEST SERIE (scan test) : très voisine dans le principe , la méthode du scan test demande une seule horloge .pour cette raison ,elle est plus facile à réaliser au niveau système. Les bascules sont du type maitre-esclave sensibles à une transition. la fonction maitre-esclave est réalisée par l'utilisation des deux fronts d'horloges .un signal de contrôle active les bascules en mode normal ou décalage.



C1 horloge fonctionnelle
C2 horloge décalage série
D1 donnée fonctionnelle
D2 donnée série
O sortie (output)

Figure(II.4) : ***Concept de base du registre d'accès série pour le test des VLSI.***

II.7.2.4.2. Autotest et bilbo :

L'autotest tel qu'il apparaît aujourd'hui dans la construction dite BILBO (built in block obesrver) peut être considéré comme une exploitation de la structure registre à décalage que nous avons vu précédemment.

Le registre à décalage est utilisé pour réaliser deux fonctions différentes :

- une fonction génération de stimulus.
- une fonction contrôle.

a) Fonction génération de stimulus :on utilise un registre à décalage rebouclé sur lui-même pour construire un générateur de séquences pseudo-aléatoires.
La figure (**II.5**) montre comment fonctionne un tel générateur.

b) Fonction contrôle : on utilise la technique de l'**analyse de signature.** L'analyse de signature, basée sur la division polynomiale, permet les résultats de test . Si le test série génère une séquence de m bits

$a_{m-1}, a_{m-2}, a_{m-3}, \ldots\ldots\ldots, a_1, a_0 \quad a \in [0,1]$

on fait correspondre le polynôme de degré m-1

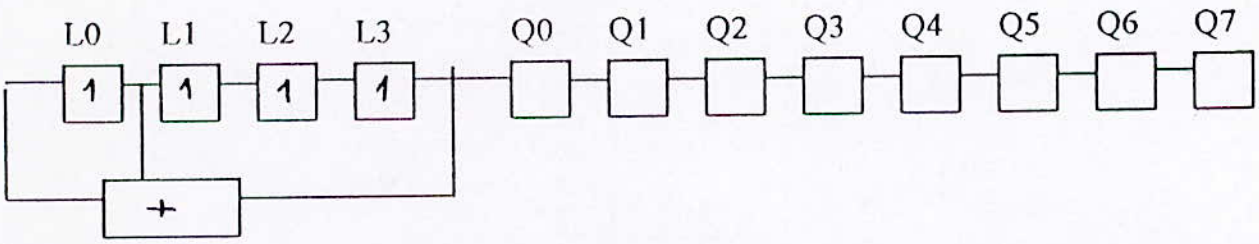
$p(x) = a_{m-1}\, x^{m-1} + a_{m-2}\, x^{m-2} + \ldots\ldots\ldots + a_1 x + a_o$

la division de p(x) par un polynome D(x) donne un quotient et un reste :

$$p(x) = Q(x).D(x) + R(x)$$

R(x) est la **signature du polynome** p(x) divisé par le diviseur D(x).Le reste R(x) sera d'un degré inférieur à p(x). La figure (**II.6**) donne un exemple de diviseur réalisé par un registre à décalage rebouclé par des OU exclusifs. Les rebouclages définissent le polynome diviseur .

la figure (**II.7**) donne le principe retenu pour la construction d'un BILBO. la partie supérieure du BILBO a trois modes de fonctionnement : fonctionnement normal du circuit , initialisation, générateur de séquences pseudo- aléatoires. La partie inférieure du BILBO fonctionne en analyse de signature.

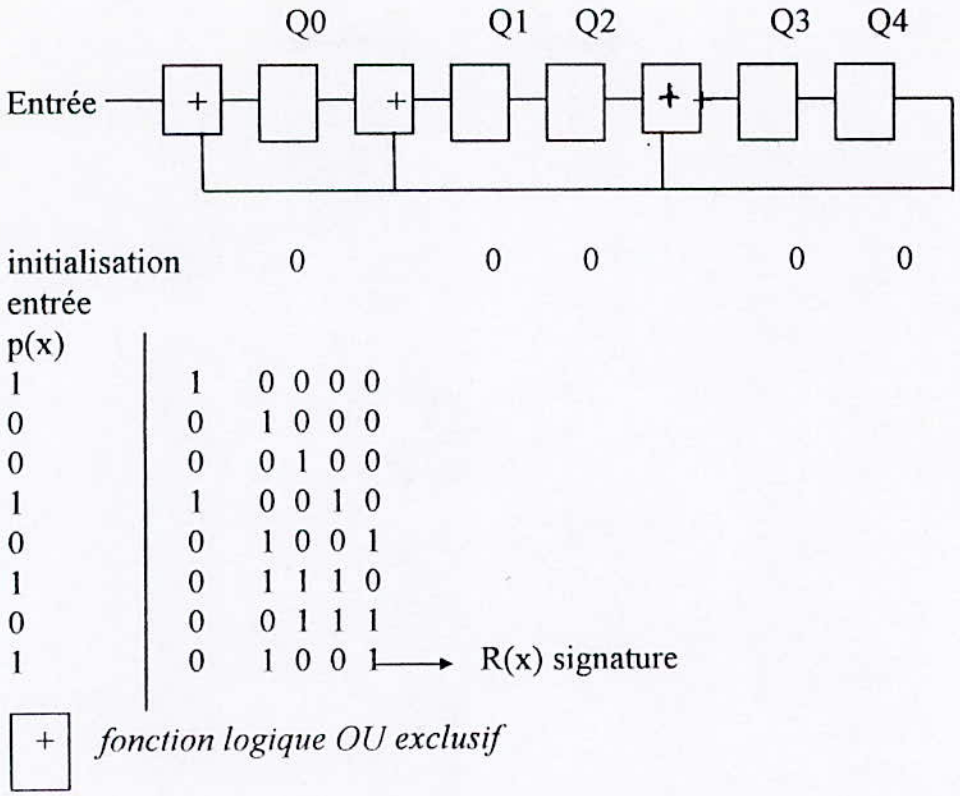


1	1	1	1	1							
0	1	1	1	1	1						
1	0	1	1	1	1	1					
0	1	0	1	1	1	1	1				
1	0	1	0	1	1	1	1	1			
1	1	0	1	0	1	1	1	1	1		
0	1	1	0	1	0	1	1	1	1	1	
0	0	1	1	0	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1
0	0	1	0	0	1	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	1	0	1
1	0	0	0	1	0	0	1	1	0	1	0
1	1	0	0	0	1	0	0	1	1	0	1
1	1	1	0	0	0	1	0	0	1	1	0
1	1	1	1	0	0	0	1	0	0	1	1
0	1	1	1	1	0	0	0	1	0	0	1

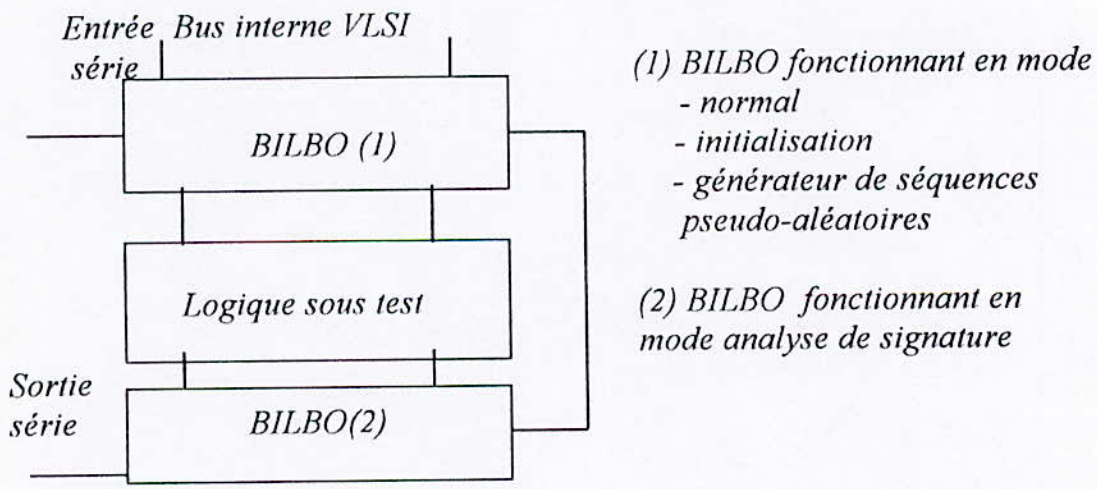
pour être sur que le générateur fonctionne , il faut au moins une bascule initialisée à 1.

+ *fonction logique OU exclusif*

fig(II.5) : Génération de séquence de test pseudo-aléatoire utilisant un registre à décalage (LSFR linear feedback shift register).



Figure(II.6) : Analyse de signature.

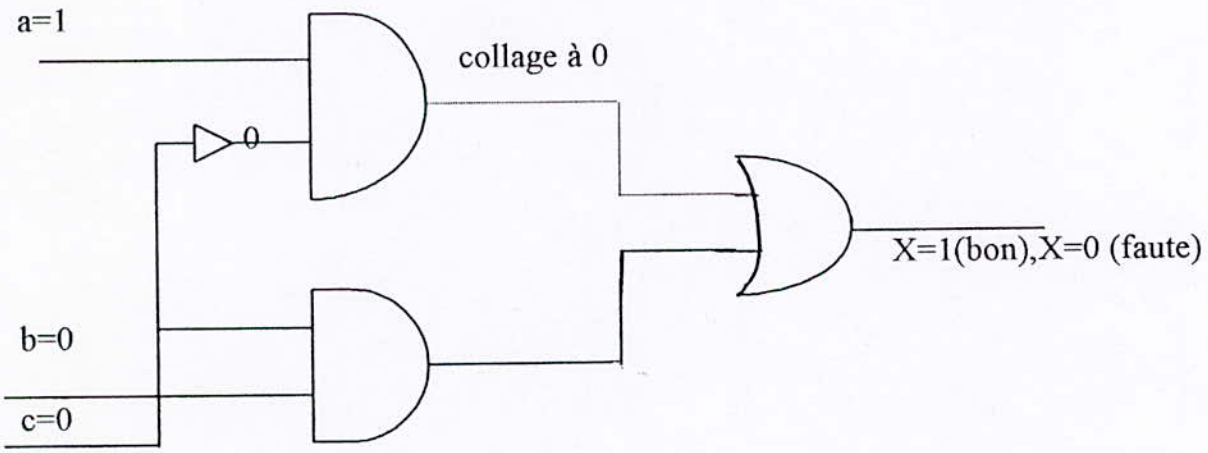


Figure(II.7) : Autotest technique de BILBO

Le modèle de fautes sur lequel repose les équipements de test et les stratégies de génération des vecteurs est celui des collages. C'est une simplification très grossière, mais suffisante des défauts pouvant arriver dans un circuit CMOS. On distingue deux types de collages :

- le collage à 0, pour lequel un noeud reste à l'état bas même si l'on cherche à le faire passer à l'état haut.
- le collage à 1 , pour lequel un noeud reste à l'état haut même si l'on cherche à le faire passer à l'état bas.

La plupart des modèles de fautes supposent qu'une seule faute arriver dans un circuit à un instant donné. (les conséquences d'un collage unique sont déjà difficiles à appréhender , la prise en compte de collages multiples entraînerait une explosion combinatoire).ceci implique qu'à partir d'un circuit de n noeuds , on obtiendra 2n circuits différents ,chacun avec une faute de collage unique.



*Figure(**II.8**) : **Circuit avec faute de collage***

La plupart des vecteurs de test (mais pas tous) vont détecter les fautes de collages dans un circuit, car le plus souvent , la sortie du circuit défectueux sera différente de celle du circuit correct (figure (**II.8**)).Certains vecteurs de test vont d'ailleurs identifier plusieurs circuits défectueux parmi les 2n possibles. Tout ensemble de vecteurs de test va donc identifier un certain nombre (compris entre 0 et 2n)de circuits défectueux .la couverture de test d'un ensemble de vecteurs de test est définie comme suit :

$$\textit{Couverture de test} = \frac{\textit{Nombre de circuits avec faute de collage unique identifiés}}{2n}$$

L'idéal est de développer un ensemble de vecteurs de test structurel qui donne un taux de couverture de 100%. On peut théoriquement y arriver sur un certain nombre de catégories de circuits idéaux, mais pratiquement c'est une tache très difficile et qui risque de prendre un temps inacceptable. La méthode consiste à effectuer une simulation en charge du circuit avec l'ensemble des vecteurs de test, tout d'abord pour le circuit sans faute, puis tour à tour pour chacun des circuits possédant une des fautes de collages .s'il s'avère que l'ensemble de vecteurs de test ne donne pas une couverture de test suffisante ,on doit itérer ces opérations avec un ensemble plus grand. Lorsqu'un ASIC comprend plusieurs milliers de noeuds , chaque simulation de faute peut prendre plusieurs heures ,ce n'est pas réaliste. Les méthodes de conception d'ASICs prenant en compte la testabilité ,visent à limiter les besoins d'une telle simulation.

II.9. Contrôlabilité et observabilité :

Pour tester un noeud dans un ASIC, on doit pouvoir le forcer à un état connu : c'est le problème de la contrôlabilité, et mesurer son état réel (c'est alors le problème de l'observabilité).Il est facile d'imposer une valeur aux entrées primaires d'un circuit, et de mesurer l'état des sorties primaires, mais plus on considérera des noeuds profonds , plus il deviendra difficile de le contrôler et de l'observer. la contrôlabilité d'un noeud est facile à analyser . C'est à partir des entrées primaires qu'il faut mettre tous les noeuds du circuit dans un état connu.

Pour observer l'état d'un noeud du circuit sur une sortie primaire, on utilise un chemin entre ce noeud et la sortie, en général le plus court ou le plus évident.on doit sensibiliser ce chemin ,de telle sorte que tout changement d'état de ce noeud se reflète sur la sortie. Tous les autres noeuds du chemin sensibilisé doivent être forcés à une valeur en respectant les règles suivantes :

- pour sensibiliser un chemin à travers une porte ET, les autres entrées de la porte doivent etre à 1.
- pour sensibiliser un chemin à travers une porte OU ,les autres entrées de la porte doivent etre à 0 .

Il faut ensuite **justifier** ces états, en déterminant la valeur des entrées primaires correspondantes, tout en respectant les contraintes de contrôlabilité. On obtient alors le vecteur de test d'une faute donnée sur un noeud donné. la méthode se complique si le circuit contient des boucles de rétroaction simples ou indirectes. Théoriquement ,on peut itérer ce processus pour chacune de 2n fautes de collages d'un circuit. dans la plupart des cas, de nombreuses fautes sont couvertes par les mêmes vecteurs. En choisissant soigneusement les vecteurs, on peut (théoriquement) arriver à couvrir l'ensemble des fautes avec un petit nombre de vecteurs de test structurel.

cependant, cette démarche analytique se révèle bien souvent trop coûteuse (en terme de temps de préparation comme de test).

II.10. Les approches de test :

La génération de vecteurs de test est le processus qui détermine l'ensemble des vecteurs qui serviront de stimuli pour le circuit dans le but de vérifier son bon fonctionnement ,et doit satisfaire au moins aux critères suivants :

la taille de l'ensemble doit être raisonnable.

le coût de production ne doit pas être élevé.

les vecteurs doivent détecter le plus grand nombre de pannes possibles.

II.10.1. Approche exhaustive :

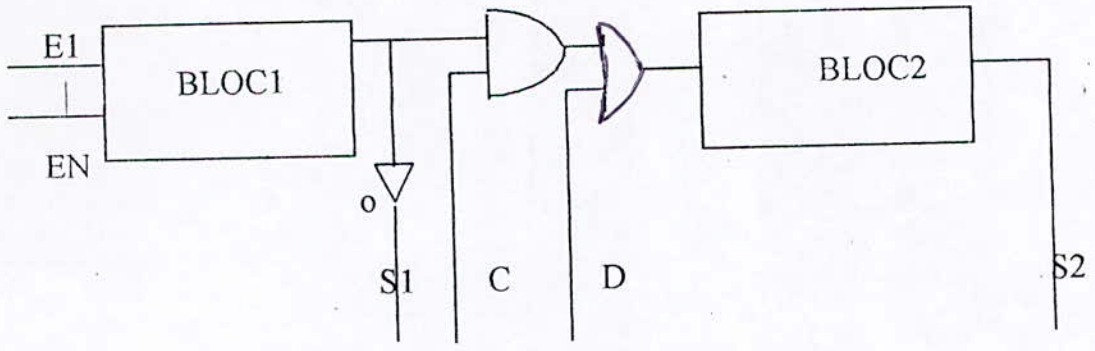
En effet ,appliquer un test exhaustif pour un circuit de n entrées, par exemple :un réseau combinatoire logique simple sans mémoire de stockage exige 2^n vecteurs d'entrée pour tester le réseau exhaustivement. Si n= 20 donc environ un million de vecteurs de test sont nécessaires, pour faire un test exhaustif. similairement ,pour un réseau séquentiel avec une mémoire de stockage en total 2^s combinaisons des états des circuits de stockage doivent être vérifier pour

Un test exhaustif de la mémoire, le réseau séquentiel et combinatoire combiné qui à n entrées et s circuits de stockage internes théoriquement nécessite 2^{n+s} vecteurs de test pour tester toutes les combinaisons possibles du réseau. c'est un nombre extrément grand (pour un simple circuit).prenant par exemple un accumulateur de 16 bits qui additionne ou soustrait un nombre d'entrée de 16 bits qui est stocké et qui à 19 entrées (16 bits plus 3 de commande) et 16 circuits de stockage, ce qui exige $2^{(16+19)}$ = 34359738368 vecteurs de test, avec une horloge de 10 mhz le test prend plus d'une heure pour un seul composant. Si pour un circuit de 64 broches (ces du microprocesseur 68000)on a $2^{(64)}$états, ce qui nécessite environ 109 heures, par le biais d'une horloge de 10 mhz (pour un seul composant).

II.10.2. Approche pseudo-exhaustive :

Un contraire au test exhaustif, le test pseudo-exhaustif est réalisable et intéressant. Si chaque sortie du circuit **ne dépend que d'une sous-ensemble d'entrée**, ou si on peut **modifier le circuit au cours de la conception,** on pourra avoir un ensemble de blocs (chacun étant testable en un nombre de cycles égal à 2n) tel que n est le nombre maximal des entrées d'un bloc avec n<< N nombre d'entrées dans le circuit.
La solution consiste à partitionner un gros circuit en blocs testable séparément.

Pour cela un rajout d'un certain nombre de portes est nécessaire, afin de rendre certains blocs transparents (garantir l'indépendance des blocs)ou translucides(contrôler et observer un bloc quelconque).Une des méthodes d'implantation de cette approche est la technique d'isolation (DEGATING), illustrée par le schéma suivant :



Figure(II.9) : La technique d'isolation

L'entrée C est utilisée pour empêcher la sortie du module 1d'arriver au module2,permettant aussi le contrôle du module 2 par D ;le module 2 ne dépendra désormais que d'une seule entrée au lieu des N entrées du module 1. Cette approche présente un taux de couverture de pannes excellent, mais cependant elle souffre de certains inconvénients majeurs dont, la difficulté d'automatiser le processus de partitionnement du circuit et le choix des portes supplémentaires.

II.10.3. Approche déterministe :

II.10.3.1. Principe de la méthode :

Quelque soit l'algorithme choisi, la détection se fait en deux étapes. la première étape consiste à sensibiliser le chemin comportant la panne à une sortie primaire puis générer un vecteur pour la panne testable. Il y'a une contrainte de temps à respecter car il peut arriver qu'une panne soit non testable et l'algorithme ne la reconnaisse pas a priori. donc ,si on fixe un quantum ,après plusieurs tentatives de sensibiliser un chemin comportant la panne sans pour autant y parvenir, la panne est déclarée non testable .la deuxième étape, consiste à simuler le circuit pour détecter toutes les pannes qui peuvent l'être avec le même vecteur généré.

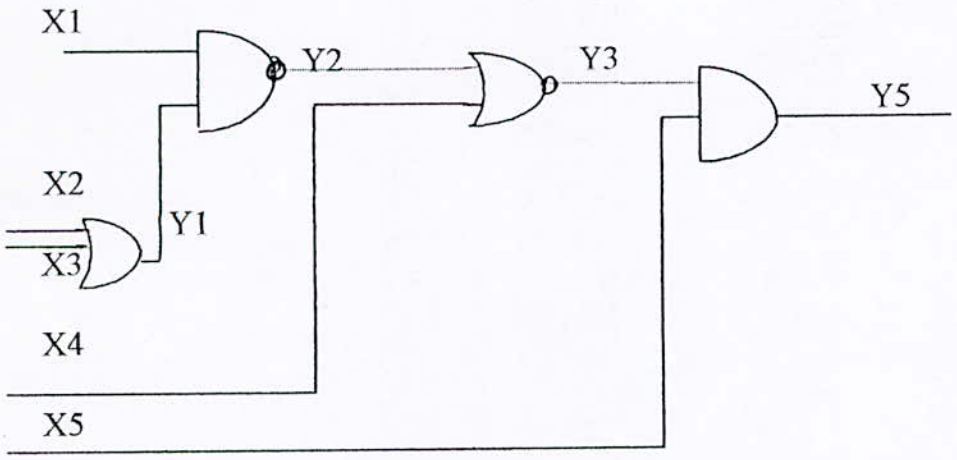
II.10.3.2. Algorithme D :

C'est le premier algorithme utilisé pour la génération des vecteurs. il est à la base de plusieurs techniques largement répandues actuellement. il procède comme suit :
partant du noeud portant la panne, il propage la valeur, permettant de tester la panne, aux sorties primaires. Il avance de niveau en affectant des valeurs arbitraires aux noeuds permettant de sensibiliser tous les chemins possibles.
Puis il justifie son choix en revenant jusqu'aux entrées primaires .

Il peut arriver que certaines valeurs désensibilisent le chemin qui propageait la panne vers les sorties primaires. Alors l'algorithme doit changer la dernière valeur arbitraire affectée par son opposée, mais souvent le mauvais choix a été fait loin en arrière. ainsi l'algorithme D et particulièrement inefficace pour certaines classes de circuits .

exemple :

Soit le réseau combinatoire suivant :



Figure(II.10) :Exemple d'un circuit combinatoire

Supposant que la ligne Y2 est bloquée à 1, pour faire propager cette panne on attribue la valeur logique 0 à Y2 (valeur opposée) et on force X4 à 0 pour avoir Y3 à 1, pour avoir Y5 à 1 X5 doit être à 1. Pour justifier notre choix en revenant aux entrées primaires, en commençant le retour en arrière à partir de Y2(Y2=0 doneY1X1=11) ainsi pour que Y1=1 implique X2X3=1X,ou X2X3=X1X(X soit 0 ou 1) ;donc le vecteur X5X4X3X2X1=10X11 est un vecteur pour tester la panne **stuck at 1** de la ligne Y2.

II.10.3.3. Podem (path oriented decision making) :

C'est une amélioration de l'algorithme précédent. Il tente de minimiser les retours arrières en n'affectant des valeurs qu'aux entrées. Donc s'il y'a un mauvais choix les retours arrières se font au niveau des entrées. Il est toujours possible qu'il est des valeurs contradictoires si un même noeud influence plus d'une porte ou plus d'une entrée d'une même porte. Si aucune combinaison n'est un test pour la panne, cette dernière est déclarée non testable. PODEM est 2 à 3 fois plus rapide que l'algorithme D pour les circuits classiques .

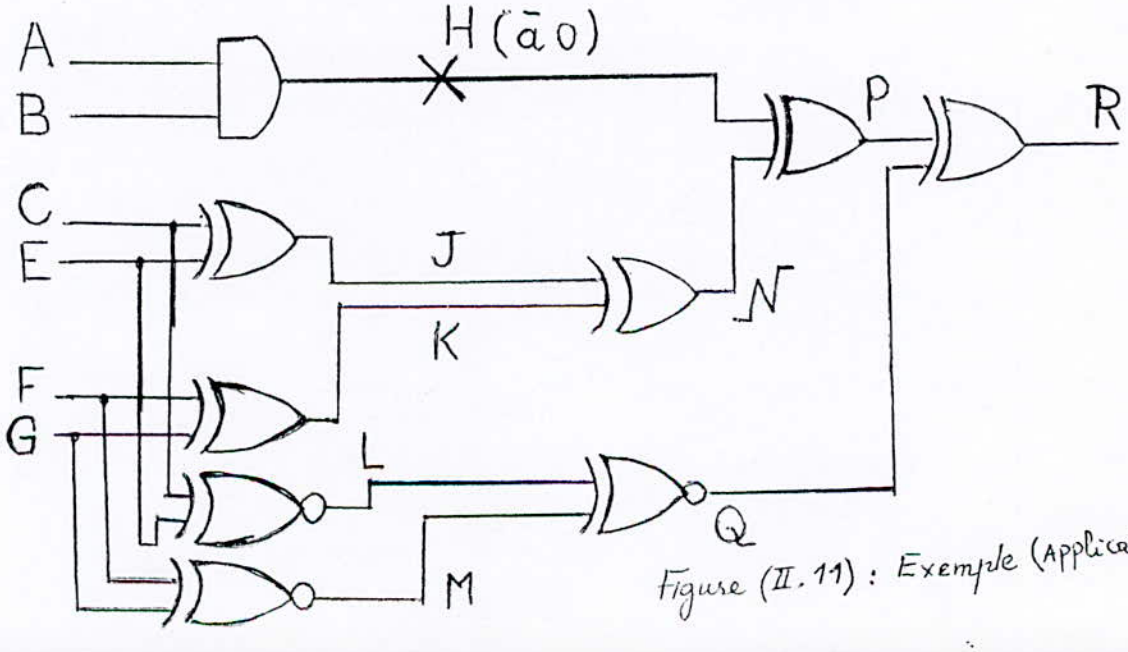


Figure (II.11) : Exemple (Application Podem)

On considère la figure(*II.11*), pour la faute H bloqué à 0 supposant qu'on assigne A=1. Nous déterminons l'implication pour toutes les entrées primaires. A=1 ne cause pas d'implication ,donc on choisit B=1 ce qui implique H=D , pour propager la valeur D de H à P nous assignons C=1,E=0 d'ou J=1 et L=$\overline{0}$. Pour F=0 et G=0 implique K=0, M=1, N=1, Q=0. De plus ,H=D et N=1 implique P=$\overline{D}$ avec Q=0 d'ou R= $\overline{D}$.

II.10.4. Approche pseudo-aléatoire :

Dans cette méthode, contrairement à l'approche déterministe, le coût de production des vecteurs de test est réduit. ce type de test consiste en l'application d'un nombre N de vecteurs aléatoirement choisis.

Le nombre de vecteurs appliqué n'est pas connu. Grâce à des méthodes heuristiques de calcul de testabilité, on peut prédire adéquatement la couverture de pannes. La technique conventionnelle pour générer des vecteurs pseudo-aléatoires est l'utilisation d'un registre à décalage à rétro-action linéaire (LFSR : linear feed back register).

II.11.Méthodes de test :

Ce paragraphe présente certaines techniques de test couramment utilisées dans les conceptions des circuits intégrés.

II.11.1. Après conception :

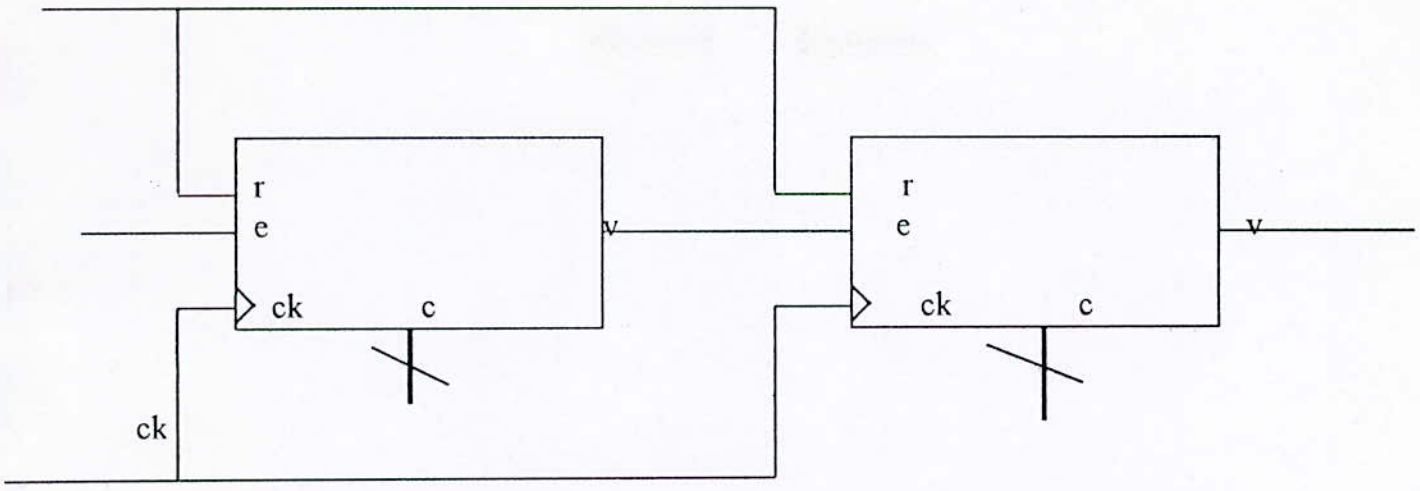
Cette technique consiste à prendre un circuit tel quel, et à développer un ensemble de vecteurs de test offrant le meilleur taux de couverture que l'on puise obtenir dans le temps dont on dispose. on utilise alors les méthodes de sensibilisation de chemins et de justification ,ou toute autre technique similaire, pour développer les vecteurs de test statiques. Si l'on dispose d'un simulateur de fautes dans les outils de conception, on peut l'utiliser pour vérifier que les vecteurs de test couvrent un nombre suffisant de fautes de collage. C'est en général la démarche la plus longue et la moins efficace pour le test.

II.11.2. Méthodes ad-hoc :

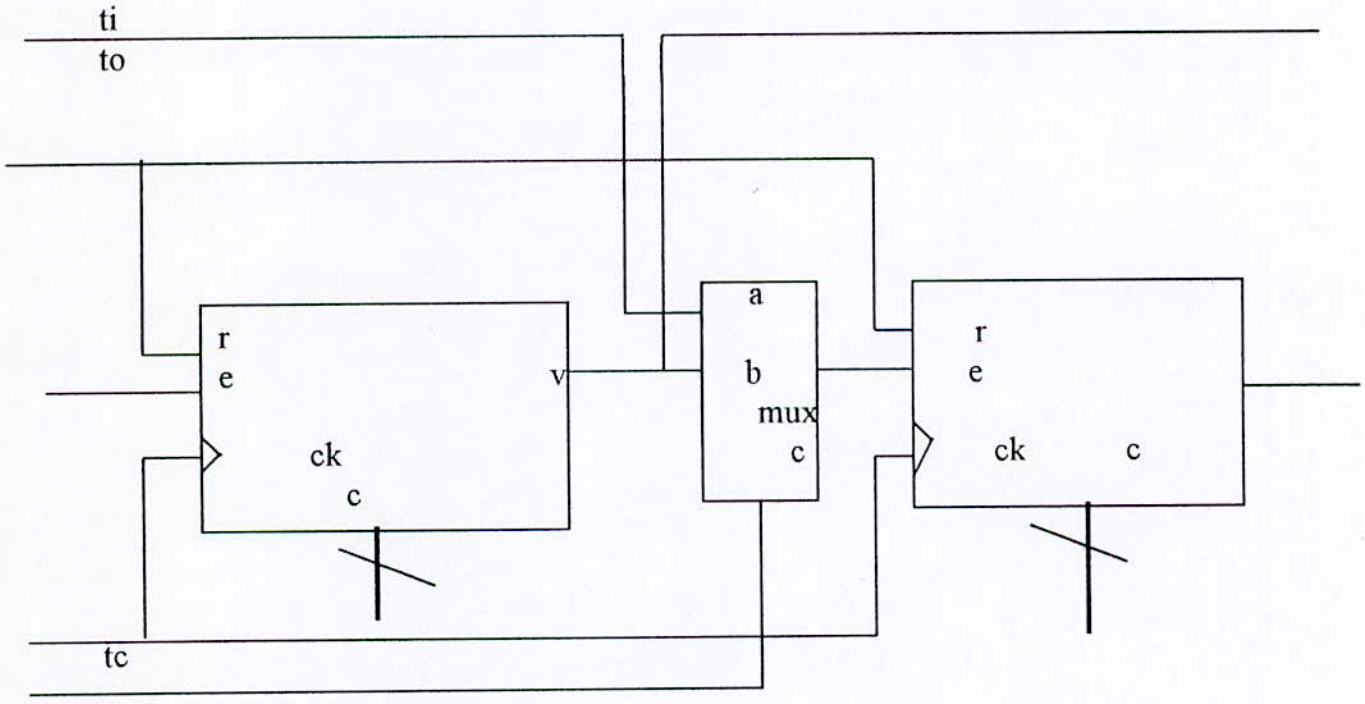
Les méthodes de conception en vue du test ad-hoc cherchent à identifier les noeuds obscurs d'un circuit (c'est-à-dire les noeuds difficiles à observer ou à contrôler) et à insérer la circuiterie nécessaire pour les relier directement aux entrées et aux sorties primaires. Pour les identifier ou s'en remettre à certaines règles empiriques. Les plus utiles sont :

- séparer les compteurs chaînés par des points de test.
- découpler les boucles de rétroaction.

Les compteurs chaînés sont fréquents dans les circuits digitaux. Toutefois, si on insère un point de test entre les compteurs (figure*(II.13)*), en reliant la sortie du premier à une sortie de test et en multiplexant une entrée de test vers le second, on améliore, donc, la contrôlabilité du second compteur et l'observabilité du premier.



Figure(II.12) :Compteur chaînés

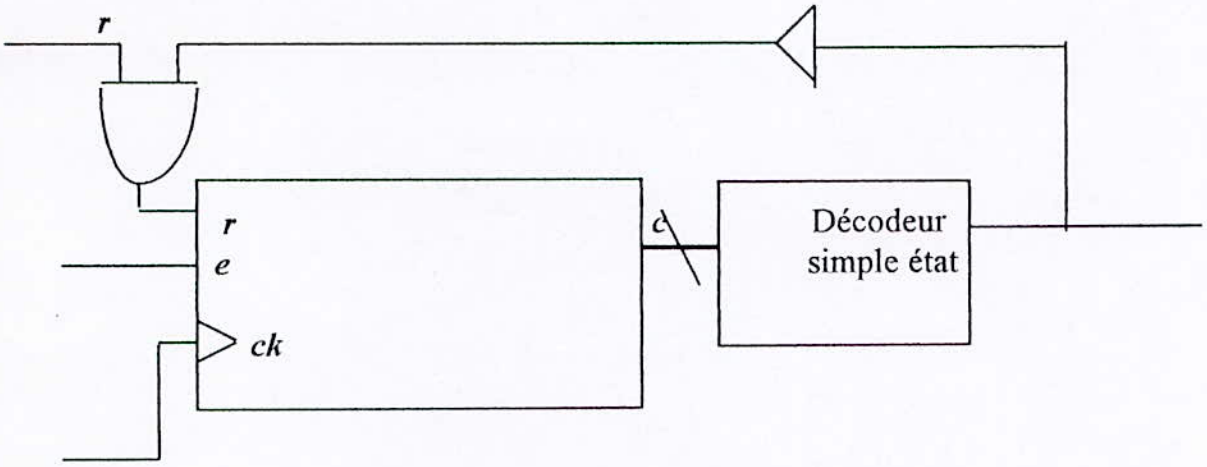


Figure(II.13) : Point de test au sein de compteurs chainés

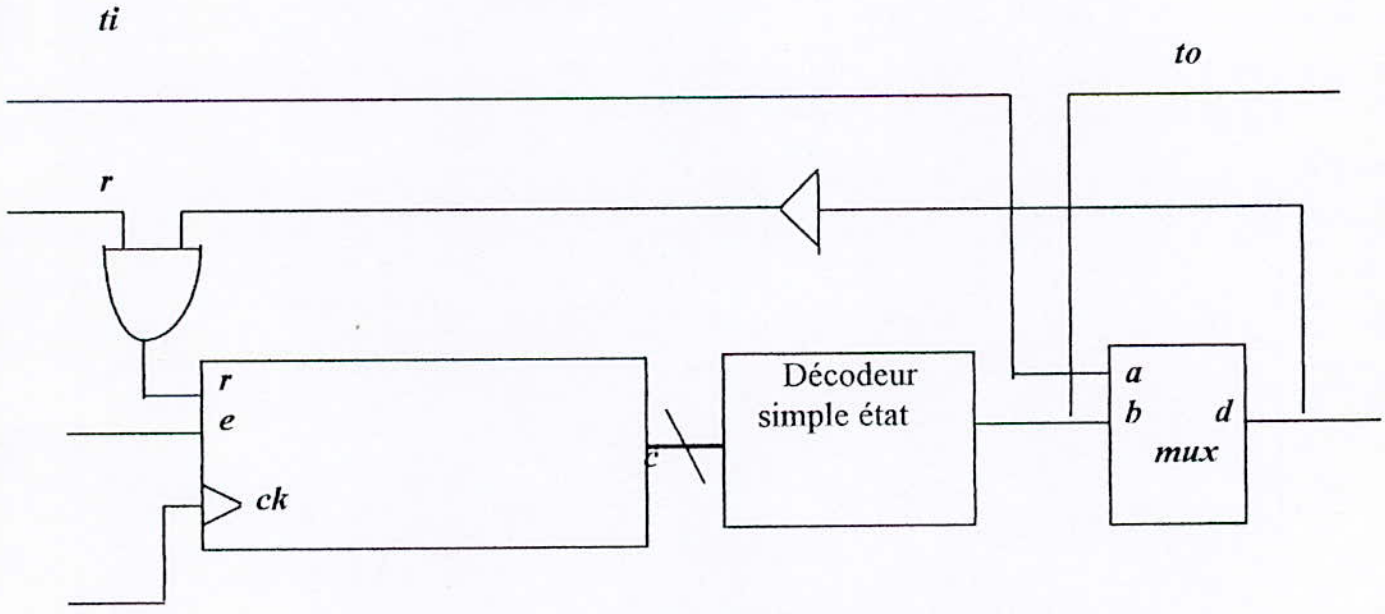
On peut appliquer le même principe à un compteur trop long ,que l'on peut voir comme deux compteurs plus courts chaînés. Le meilleur endroit pour insérer le point de test est au milieu du compteur.

Un exemple classique de boucle de rétroaction est présent dans un compteur de modulo différent d'une puissance de deux (figure (*II.14*)). La ligne de rétroaction est difficile à contrôler car elle dépend de la sortie finale du compteur.

Dans ce cas ,le compteur est à la fois difficile à contrôler et à l'observer parce que la rétroaction fait partie de son contrôle. La solution consiste à découpler la ligne de rétroaction en insérant un point de test (figure(*II.15*)). En mode de test, on peut remettre le compteur à zéro à la demande ,ou autoriser un cycle de comptage complet afin d'exciter tout les noeuds internes.



Figure(II.14) :Compteur avec boucle de rétroaction



Figure(II.15): Découplage de la boucle de rétroaction

Chapitre III :
Vérification des circuits séquentiels

CHAPITRE III : VERIFICATION DES CIRCUITS SEQUENTIELS

III.1. INTRODUCTION :

Le problème de vérifier l'équivalence entre deux implémentations de circuits séquentiels et le problème de la génération de test pour chaque circuit est intimement lié. Le problème de la vérification peut être formuler comme un problème de décision ou pour deux descriptions données d'un circuit la question qui se pose est si les deux descriptions ont le même fonctionnement. Il est devenu essentiel de vérifier efficacement que les descriptions originales et les descriptions optimisées représentent la même machine, c'est a dire le processus de synthèse n'est pas introduit avec des erreurs dans le circuit.

La vérification est exigée à toute les scènes de processus de DESIGN et les différentes techniques de vérification sont utilisées dans différentes scènes de vérification.

La plus importante phase de processus de design est l'optimisation de la logique séquentiel. a ce niveau différentes techniques (par exemple l'optimisation **sous don't care**)sont utilisées pour transformer la description de niveau logique du circuit a une description plus optimale. l'application principale pour ces techniques est de vérifier la convenance d'optimisation et les outils de synthèse. L'approche la plus efficace de vérification des circuits combinatoires utilisant le diagramme de décision binaire BDD, par lequel le problème de vérification peut se convertir a un graphe. L'approche **simulation énumération** est utilisée de façon que la table de vérité d'un circuit soit énumérée et simultanément simulée dans l'autre circuit. Le temps exigé pour la vérification peut augmenter exponentiellement avec le nombre d'entrées du circuit.

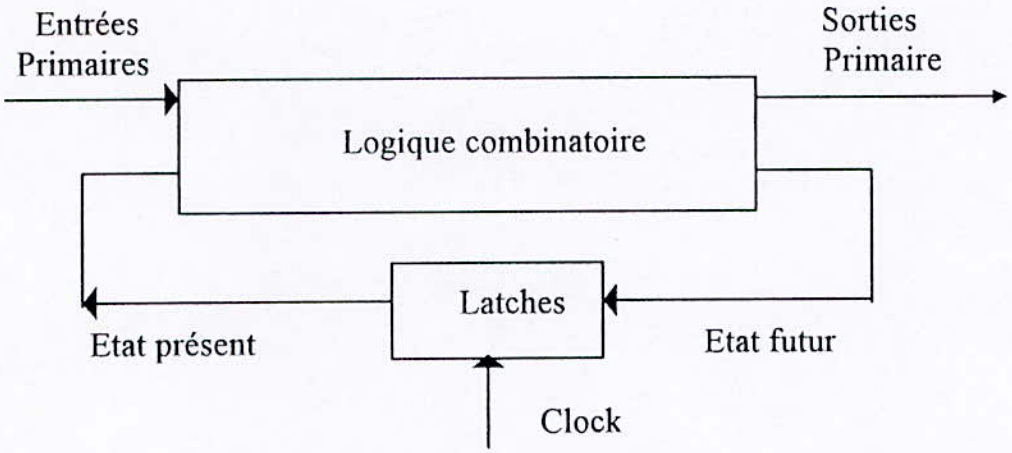
Cependant si on utilise **l'énumération implicite**, le nombre d'états concatenés pour les simulées peut être considérablement réduit.

Une autre approche de la vérification séquentielle est dite **simulation exhaustive** ou **chaque chemin** dans le graphe des états (STG) des machines est simulé, la réduction du nombre de chemin est énumérée et simulée.

Le problème de la vérification de la logique séquentielle est plus compliqué que le problème de la vérification de la logique combinatoire.

III.2. RAPPEL :

Une machine séquentielle est représentée en général selon la figure (**III.1**) :



Figure(III.1) : ***Une machine séquentielle***

Une telle machine est définie comme une fonction M tel que :

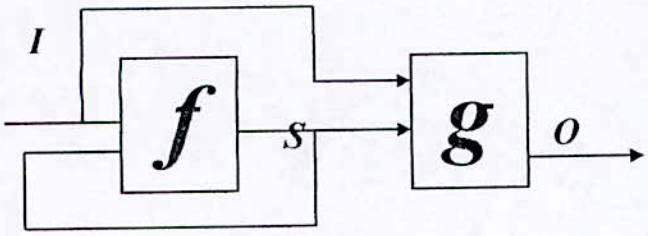
$M=(I, O, S, S^0, f, g)$.

I : ensemble des entrées .
O : ensemble des sorties.
S : ensembles des états.
S^0 :ensembles des états initiaux.
f : la fonction de l'état futur $S \times I \longrightarrow S$.

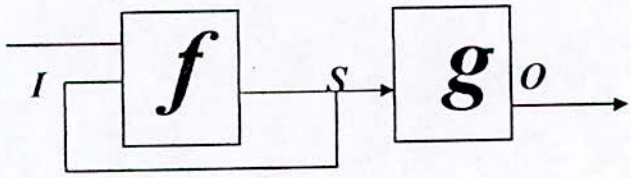
$g : S \times I \longrightarrow O$ fonction de sortie de la machine de mealy.

g : la fonction de O :

$g : S \longrightarrow O$ fonction de sortie de la machine de moore.

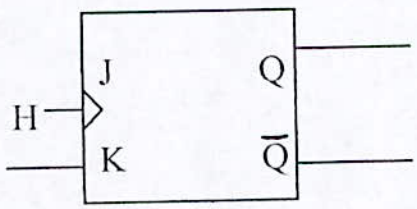


Figure(III.1) : Machine de Mealy



Figure(III.2) :Machine de Moore

Exemple d'application :
comment trouver un STG (state transition graph) pour la bascule JK ?



- l'équation de la bascule : $Q^+ = J\overline{Q} + \overline{R}Q$

- table de vérité de la bascule :

J	K	Q^+	$\overline{Q^+}$
0	0	Q	$\overline{Q}$
0	1	0	1
1	0	1	0
1	1	$\overline{Q}$	Q

- table des excitations ou de transition :

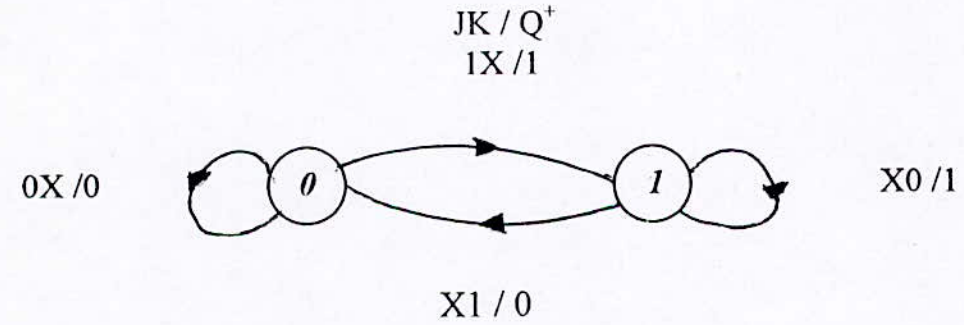
Q	Q^+	J	K	
0	0	0	X	S_0
0	1	1	X	T_1
1	0	X	1	T_0
1	1	X	0	S1

ou S_i : statique à i (0 ou 1)
Ti : transition vers i (0 ou 1)

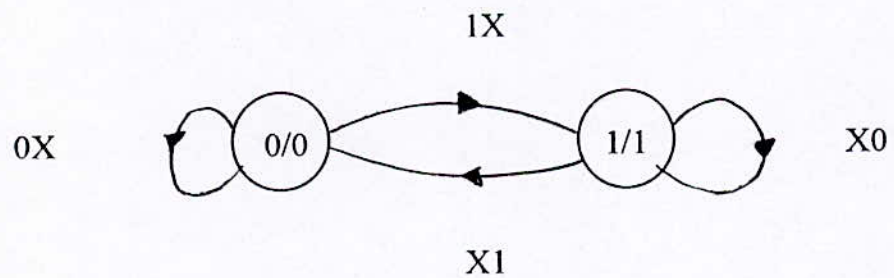
I	**S**	**S^+**	**O**
J K	Q	Q^+	Q^+
0 X	0	0	0
1 X	0	1	1
X 1	1	0	0
X 0	1	1	1

*Tableau(III.1) : **Table des états***

d'ou le graphe des états est le suivant :



*Figure(III.3) : **Modèle de Mealy***

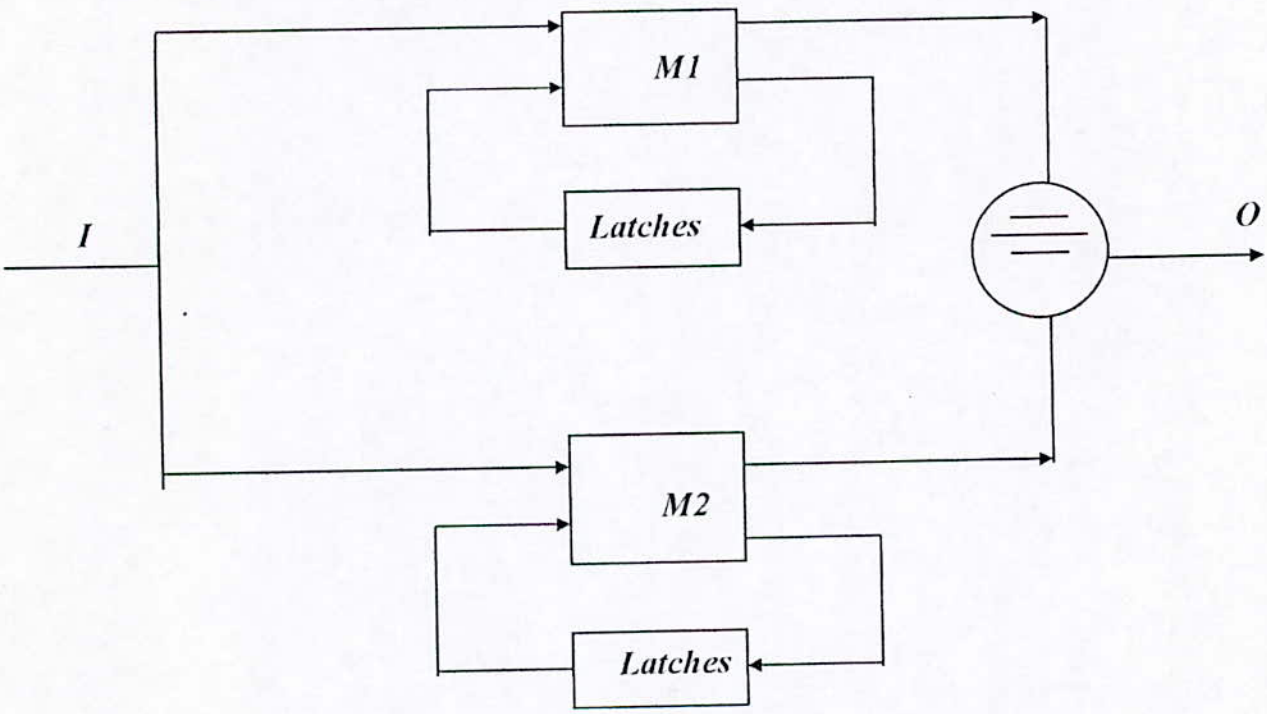


*Figure(III.4) :**Modèle de Moore***

III.3. PRELIMINAIRE :

Pour montrer que deux circuits ne sont pas équivalents, il est important de trouver la séquence primaire d'entrées appliquée aux deux machines nous donne comme ré résultat séquences différentes de sorties. Si une telle séquence n'existe pas, les machines sont dites **équivalentes**. Avant d'essayer de déterminer l'équivalence, la correspondance au moins entre un état dans chaque machine est exigée. Par conséquent chaque machine est supposée avoir un état initial.

Soient **M**1 et **M**2 deux machines à état fini ; le produit des deux machines est défini comme étant une seule machine obtenue par connections des machines en parallèle comme l'indique la figure (**III. 5**).



Figure(III.5) : Machine-produit

Les entrées des deux machines sont connectées, la machine produit a une seule sortie obtenue par connexion des sorties de deux machines par des portes **XNOR**,puis connectant les sorties du **XNOR** à des portes **AND**. Si la sortie est toujours vraie (c-a-d 1),alors on dit que les deux machines **M**1 et **M**2 sont équivalentes. Le principe d'utilisation de la machine produit est de vérifier ou établir l'équivalence des deux états.

L'énumération du STG pour une machine est le processus d'obtenir une description STG de la machine. Pour une telle description, chaque état est représenté par un code symbolique unique. Prenons un vecteur d'entrée et un état donné, on peut déterminer la sortie et l'état futur.

En ce qui concerne la transverse STG pour la machine, chaque état ne peut pas avoir un code symbolique unique et le prochain état pour chaque combinaison d'entrée ne peut pas être déterminé. Cependant, tous les états sont visités pour vérifier si une certaine propriété est vraie pour tous les états.

III.4. METHODE TRANSVERSE :

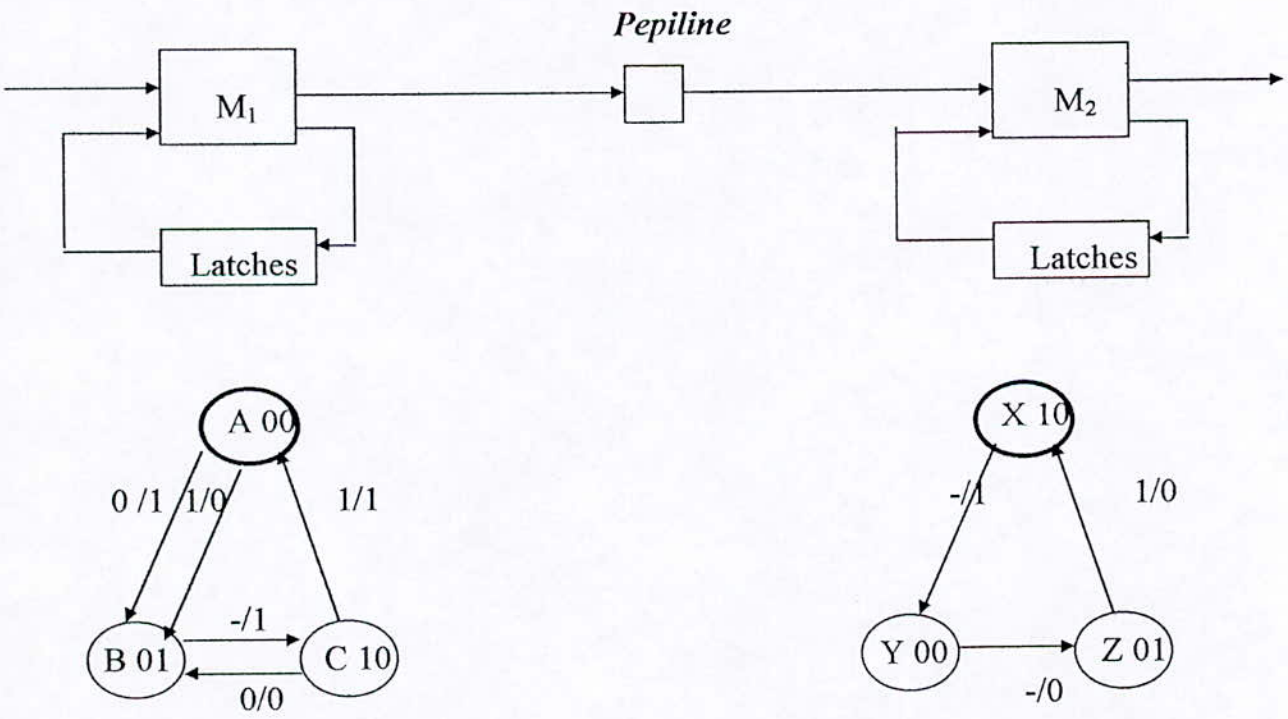
Il existe des algorithmes qui exigent un **stockage implicite** du STG de la machine séquentielle pour vérifier l'équivalence. Ces algorithmes peuvent être utilisés pour les circuits avec 10 latches à partir de la description du STG des grands circuits.Seulement, ils exigent une capacité de stockage énorme de la mémoire.

Les algorithmes transverses symboliques STG sont développés pour permettre de traverser le STG de la machine et de vérifier si certaines propriétés sont vraies pour tout les états validés de la machine. Un tel algorithme (transverse) peut être utiliser pour la vérification par construction en traversant le STG de la machine produit.

Ces algorithmes sont mieux implémentés on utilisant le diagramme de décision binaire (**BDD**).Cependant pour certains circuits, BDD_s sont si grands qu'ils ne peuvent pas être construits.

III.4.1. STG transverse implicite :

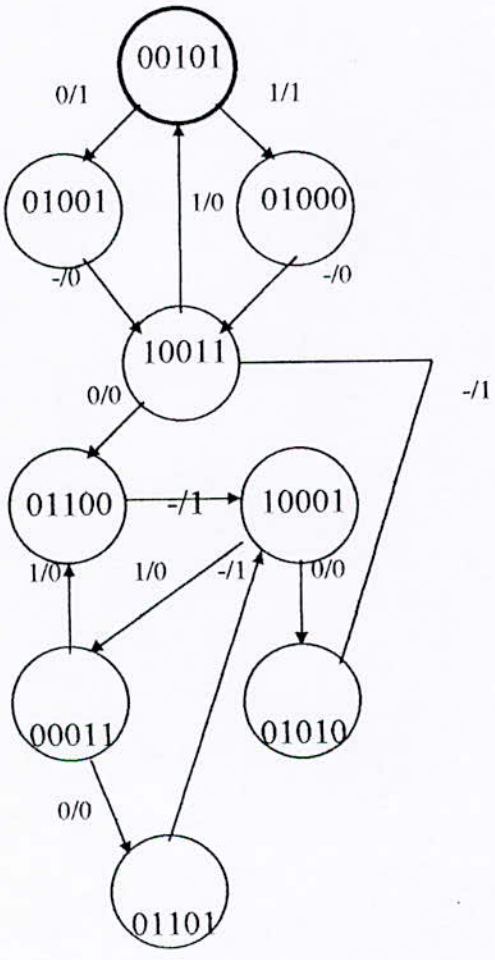
On considère une interconnexion des machines selon la figure (**III. 6**) :



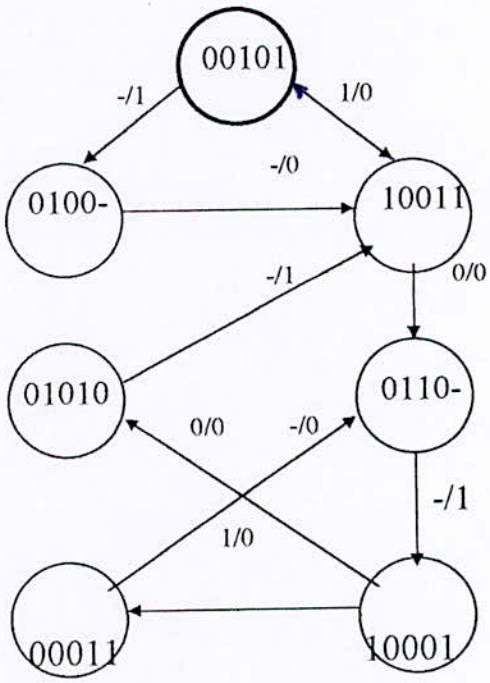
*Figure(III.6) : **Deux machines en cascade***

Les STGs des machines **M1** et **M2** sont montrées au dessous de chaque machine, l'état **A** dans **M1**et l'état **X** dans **M2** sont des états initiaux, la sortie de **M1** est connectée a l'entrée de **M2** a travers une pepiline-latch. Avec l'état initial de pepiline-latch est a 1.

Les pepiline-latch sont introduites pour conserver et fixer les contraintes et souvent n'ajoutent pas beaucoup de complexité au comportement séquentiel du circuit. Si cette interconnexion de machines est considérée comme étant une seule machine, alors pour l'approche d'énumération on aura seulement **9** états et **13** arêtes (edges). Comme l'indique la figure (**III.7**).



*Figure(III.7) : **Enumération explicite***



*Figure(III.8) :**Enumération implicite***

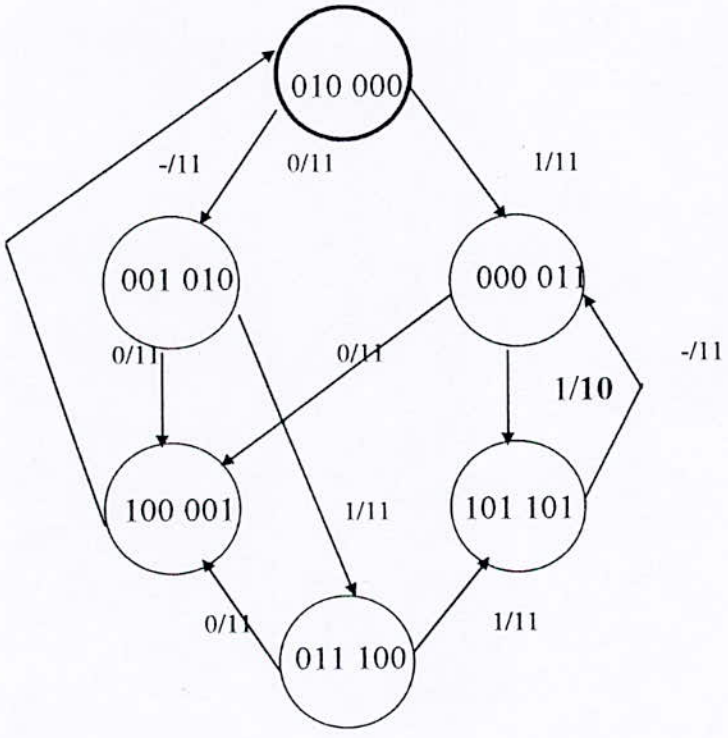
Chaque état dans cette machine a un code de 5 bits , ou les deux premiers bits correspondent a l'état de **M1** et les deux bits qui suit correspond a l'état de **M2**. Le dernier bit correspond a l'état du pipeline-latch.

Les états pairs {01001,01000} et {01101,01100} qui différent seulement dans l'état du pipeline latch sont équivalent . et on peut les concatenés dans un seule état 0100- et 1110-

Pendant le processus d'énumération, si pour les bits d'états ont peut avoir le don't-care alors on obtient un STG avec 7 états et 9 arêtes (edges), (voir la figure (**III.8**)), (on utilisant l'algorithme transverse pour ce circuit),on dit que le STG est réduit et le temps d'énumération est aussi réduit.

La procédure principale de vérification est montrée suivant la figure(**III.9**) ;l'état initial de la machine produit est la concaténation des états initiaux des deux machines.

Pendant le **transverse** tous les états sont visités, et si un état donne a la sortie un **0** alors les machines sont différentes sinon elles sont équivalentes.



Figure(III.9):Le STG d'une machine produit

III.4.2. Méthodes transverse symbolique FSM :

III.4.2.1. Introduction :

Les algorithmes de STG transverse pour une machine à états finis ont des applications dans les domaines de synthèses, test et vérification. Les méthodes traditionnelles qui énumèrent l'espace d'état de la machine (explicitement) ne peuvent pas manier la machine avec un grand nombre d'états.

Les approches présentées dans ce chapitre essaient de combiner l'énumération implicite des entrées avec l'énumération implicite d'espace d'état Malheureusement,En général, les états peuvent se combiner si et seulement si ils sont équivalents.

La méthode d'énumération implique une recherche d'une depth-first (recherche en profondeur)du STG .

Coudert [1]est le premier a réaliser quelques (BDDs) pourraient être utilisés pour représenter l'ensemble des états. Il propose des algorithmes pour calculer la rangée d'une fonction ; cela a mené la formulation d'un algorithme qui traverse le STG de la machine. Dans cet algorithme, l'entrée et l'espace d'état sont énumérés implicitement.

Ces algorithmes sont souvent rapportés à des algorithmes symboliques transverses parce qu'ils impliquent des opérations sur l'ensemble des états. Ces opérations peuvent être exécuter en utilisant efficacement les(BDDs). L'efficacité de ces algorithmes dépend de l'efficacité des procédures manipulant les BDDs.

Les algorithmes symboliques transverses travaillent pour des circuits qui ne sont pas séquentiellement ''profonds''. Tous les états dans un circuit peuvent être atteints dans un petit nombre de cycles d'horloges à partir de l'état initial. Les circuits qui sont séquentiellement ''très profonds'', ou un état particulier peut être atteint après un grand nombre de cycles d'horloges à partir de l'état initial , leurs algorithmes peuvent prendre un temps long.

Par conséquent, la technique transverse depth-first pour les machines séquentielles permet de traverser le compteur FSM (par exemple) en $O(n)$,ou n est le nombre de bits dans le compteur présenté. D'autre approches traversant le STG exigent 2^n pas.

III.4.2.2.Preliminaire :

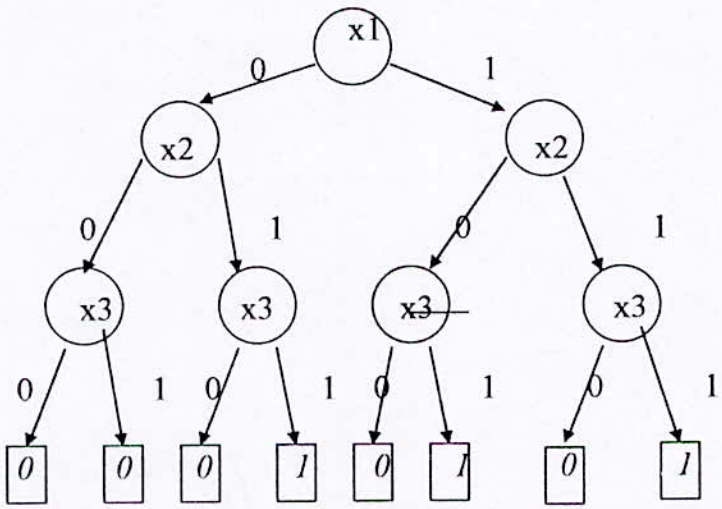
III.4.2.2.1. Diagramme de décision binaire(BDD) :

Des variétés de méthodes sont développées pour représenter les fonctions booléennes, comme les représentations classiques (table de vérité, table de karnaugh, méthodes heuristiques, BDD,RBDD).

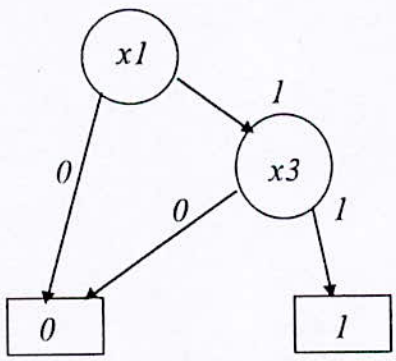
Le diagramme de décision binaire est proposé par lee *[1]* ; un diagramme de décision binaire ou BDD peut être représenté par un arbre (binaire) ou par un graphe direct acyclique (pas de boucle). Voir exemple figure*(III.10)*.

Comme L'BDD n'est pas canonique (c-à-d des différentes représentations de la fonction), il est important de choisir la représentation de la fonction la plus réduite, on parle ici de RBDD (Réduction du diagramme de décision binaire).

Exemple :
f(x1,x2,x3)=(x1+x2) . x3
On va calculer le BDD et le RBDD de cette fonction.



Figure(III.10) :Le BDD de la fonction f



Figure(III.11) :Le RBDD de la fonction f

III.4.2.2.2. propriétés :
soit f(x1,x2,....xn) une fonction booleanne a n variables ,ou {x1,x2,.....,xn}le support cette la fonction.

Le cofacteur de f(x1,x2,.....,xn) selon la variable xi est : f_{xi} = f(x1,....,**1**,..,xn), de même le **cofacteur** de f(x1,x2,.....xn) selon xi est :$\overline{f_{xi}}$ = f(x1,.....,**0**,....,xn), c'est à dire :

$$f_{xi}= f/x_i=1$$

$$f\bar{x}_i=f/\bar{x}_i=0$$

Théorème : (expansion de shannon)

Si f une fonction a n variables alors :

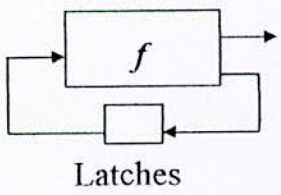
$$f(x1,.....xn)=xi \cdot f_{xi} + xi \cdot \overline{f_{xi}}$$

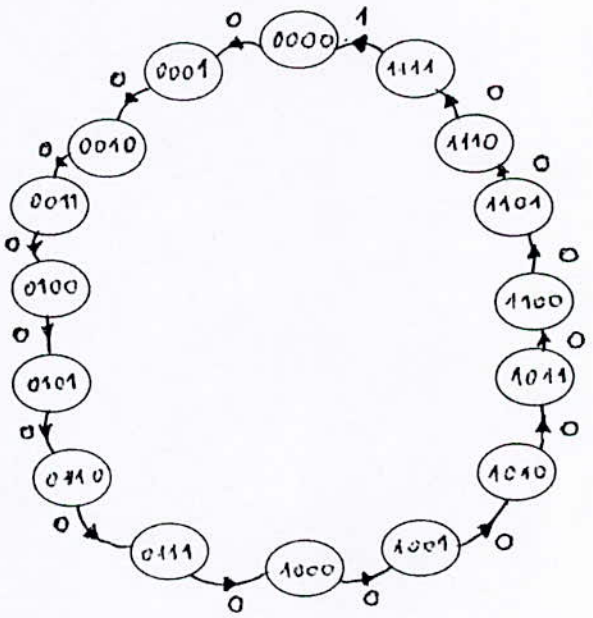
$$f(x1,.....xn)= (xi + f_{xi}) \cdot (xi + \overline{f_{xi}})$$

D'ou on peut exprimer n'importe quelle fonction sous forme de somme de produit de littéraux appelles(**mintermes**)ou sous forme de produit de somme appelles (**maxtermes**).

III.2.3.Chainage géométrique DEPTH- FIRST(Recherche en profondeur) :

Dans cette section on va présenter le DEPTH-FIRST par un exemple (figure(*III.12*)) on présente ici le graphe des états (**STG**) d'un compteur modulo 16.

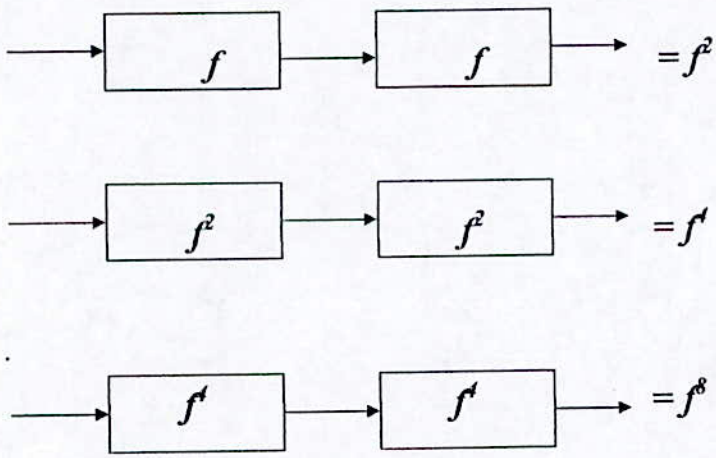




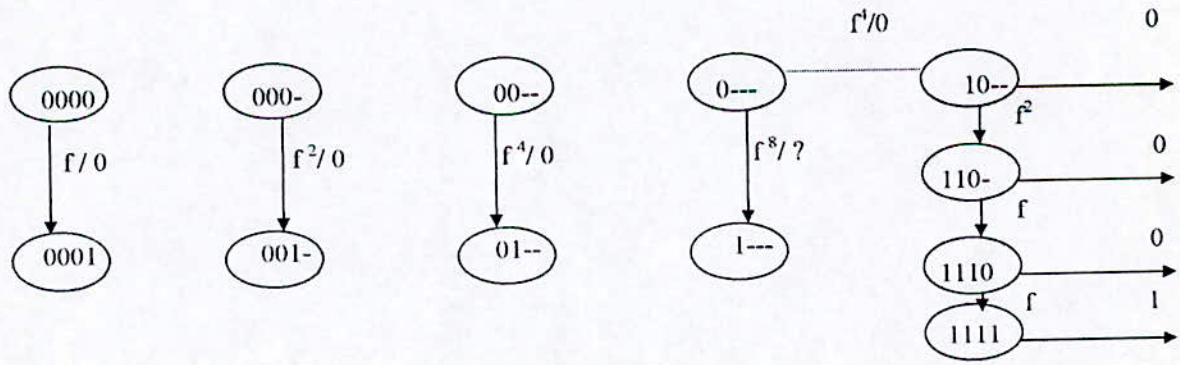
Figure(III.12) : Compteur autonome de 4 bits

L'état futur du compteur est obtenu à partir de l'état présent et on appliquant la fonction d'incrementation *f*. (ici pour représenter *f* on utilise le RBDD pour avoir une représentation la plus optimale).

L'état futur du compteur est obtenu à partir de l'état présent et on appliquant la fonction d'incrementation *f*. (ici pour représenter *f* on utilise le RBDD pour avoir une représentation la plus optimale).



Figure(III.13): L'obtention des fonctions géométrique



Figure(III.14): Exemple du transverse d'un compteur à 4 bits

Chapitre IV
Simulation

CHAPITRE IV: SIMULATION

IV.1. INTRODUCTION :

Dans ce chapitre on présentera des algorithmes de test et de vérification, ensuite on les programmera en utilisant le langage C++ de BORLAND (version 4.5),et enfin on présentera les résultats de simulation avec une discussion .

Le BORLAND C++ avec **Application Framwork** dispose :

- d'un **EDI** (Environnement de Développement Intégrés) fonctionnant sous Windows et permettant de développer des applications pour Windows . Borland C++ avec application framwork permet également de développer des applications pour DOS et, de plus, il dispose d'un compilateur **en ligne de commande,**

- de la bibliothèque de classes **Object Windows**, la quelle facilite énormément le développement d'applications Windows ; vous y trouvez des classes toutes prêtes vous permettant de manipuler des fenêtres, des boites de dialogue, des contrôles (boutons poussoir ,boutons radio ,case à cocher...),etc. Ces classes disposent des facilités communes à toutes les objets du type correspondant ; par exemple, les classes fenêtres comporteront déjà tout ce qui est nécessaire à la gestion de leur déplacement ,de leur changement de taille, leur réduction en icône...

- d'un utilitaire **Ressource Workshop** qui permet de développer visuellement les boites de dialogue et les menus déroulants.

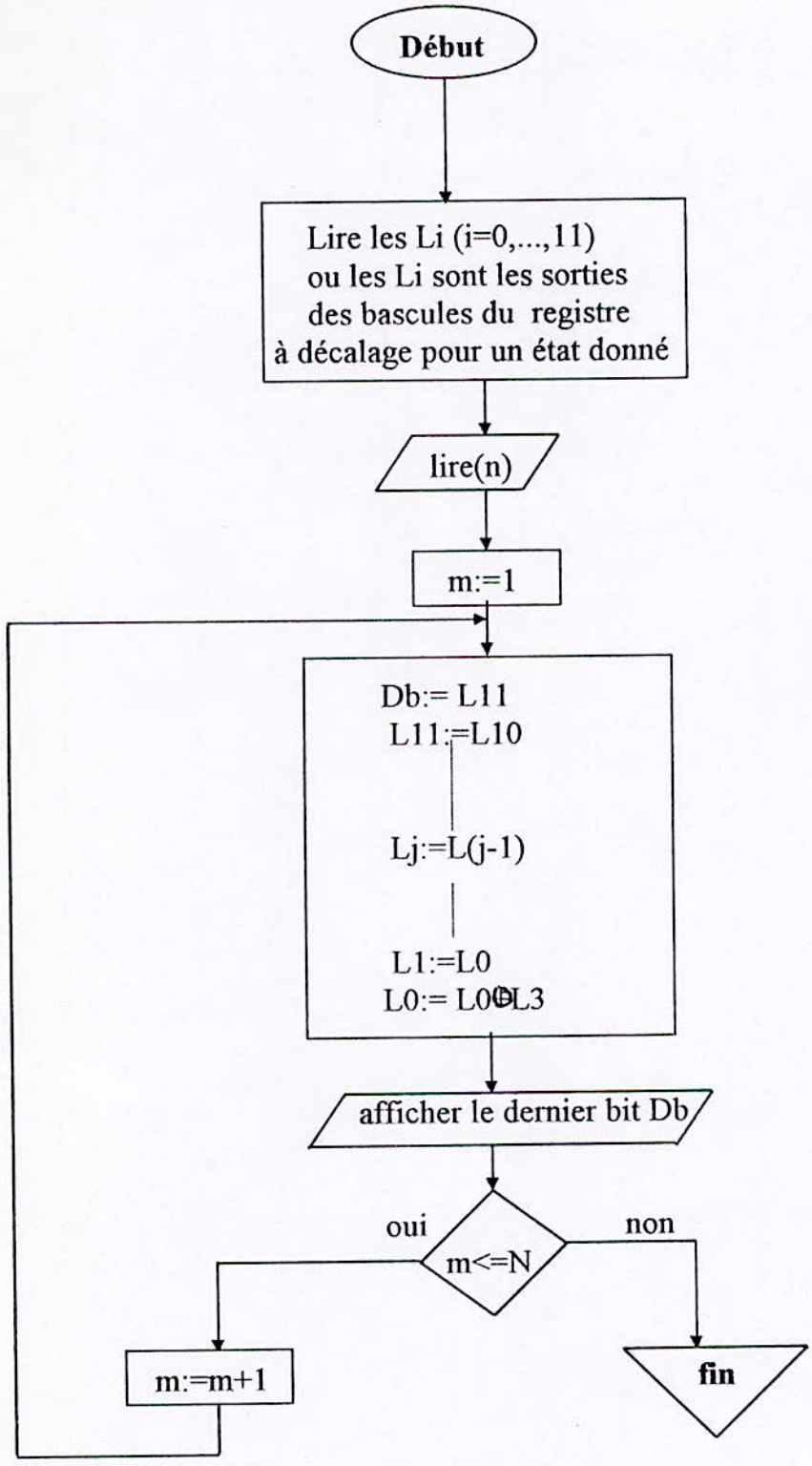
La figure(**IV.1**)montre l'interface de notre logiciel de simulation.

Figure(IV.1) : ***Interface du logiciel de test et de vérification des VLSI***

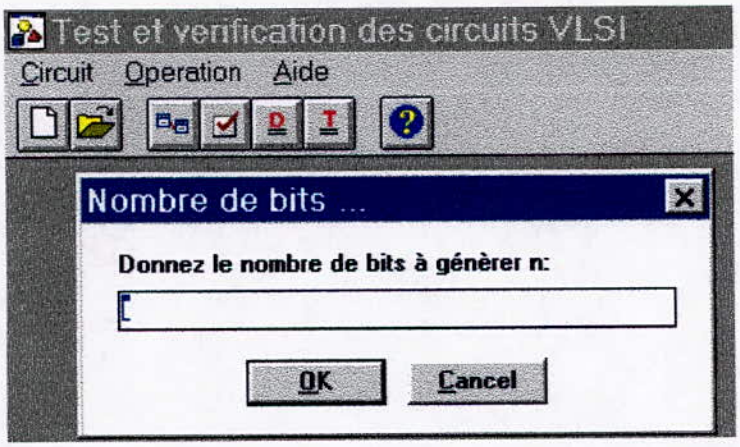
IV.2. SIMULATION DES METHODES DE TEST :

IV.2.1.Organigramme de génération de test pseudo-aléatoire utilisant un registre à décalage :

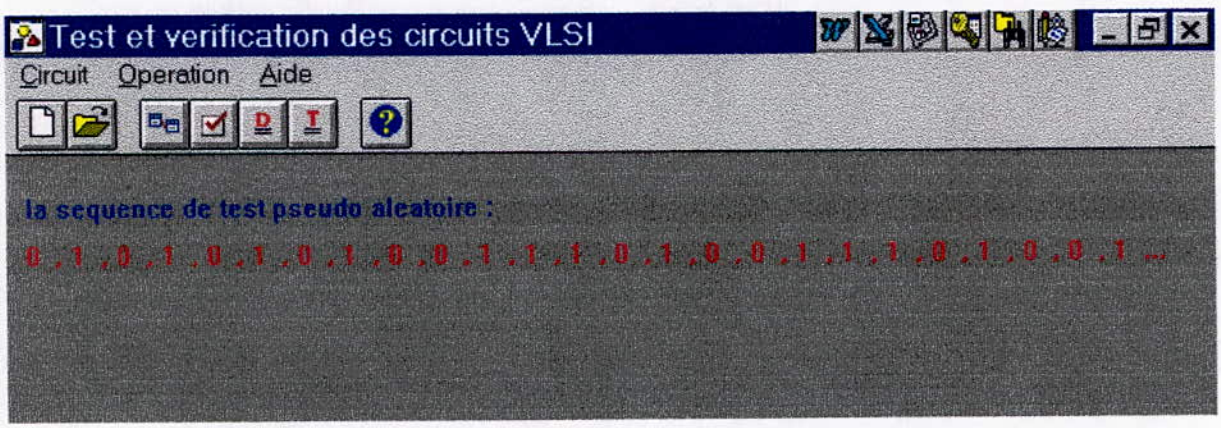
C'est la description de la figure (***II.5***) du chapitre II (page 33). L'organigramme de génération de séquences pseudo-aléatoires utilisant un registre à décalage, est montré par la figure (*IV.2)* suivante.



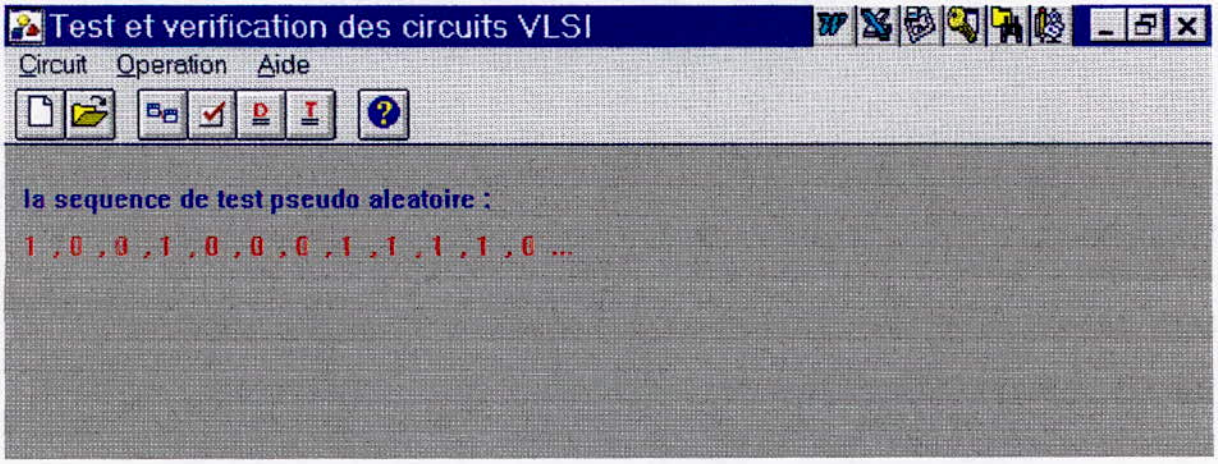
Figure(IV.2) : ***Organigramme de génération de test pseudo-aléatoire utilisant un registre à décalage (LSFR)***



*Figure(IV.3): **Saisie le nombre de bits n***

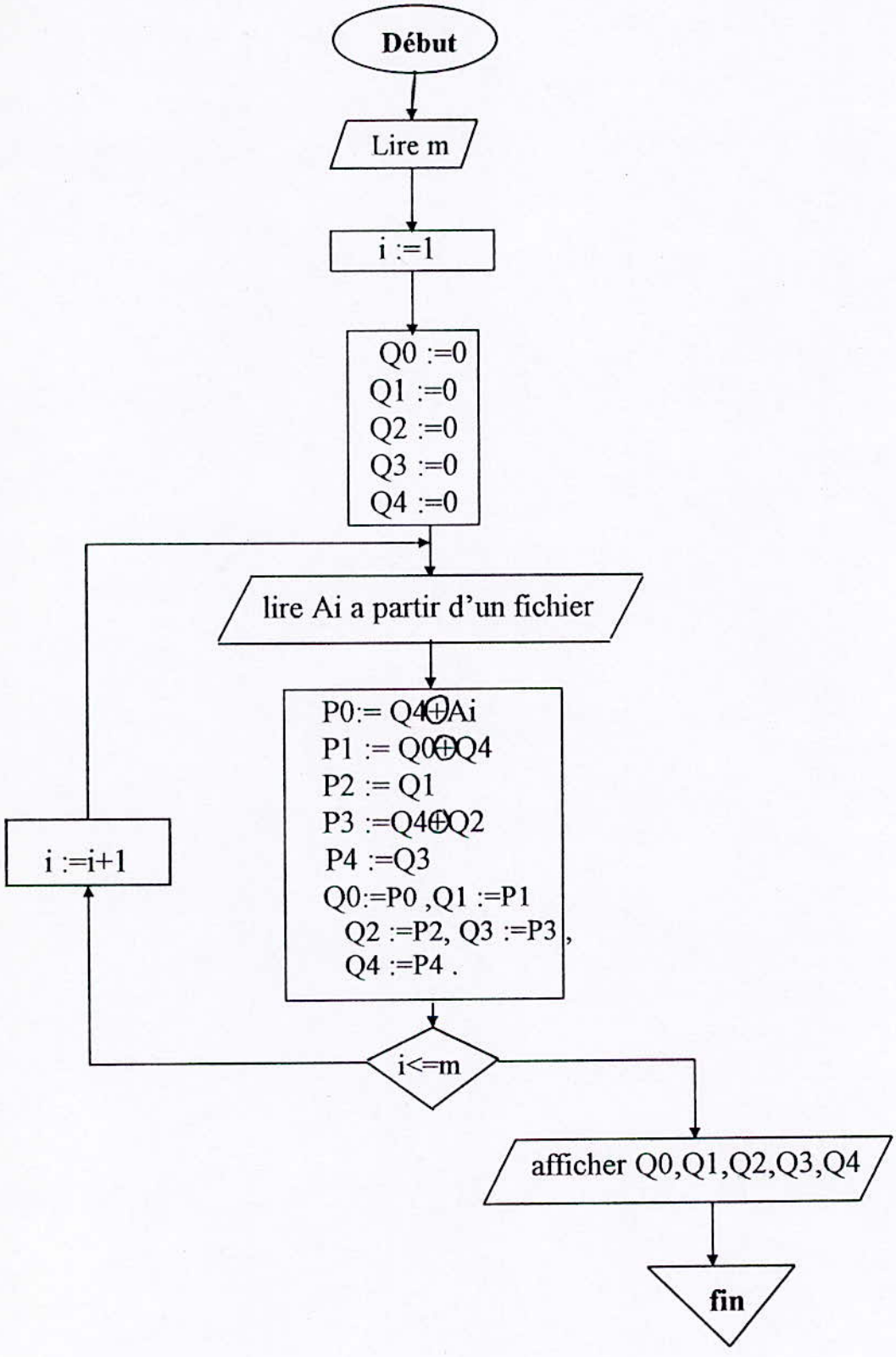


*Figure(IV.4) :**Génération de séquence pseudo-aleatoire (ici n =25)***



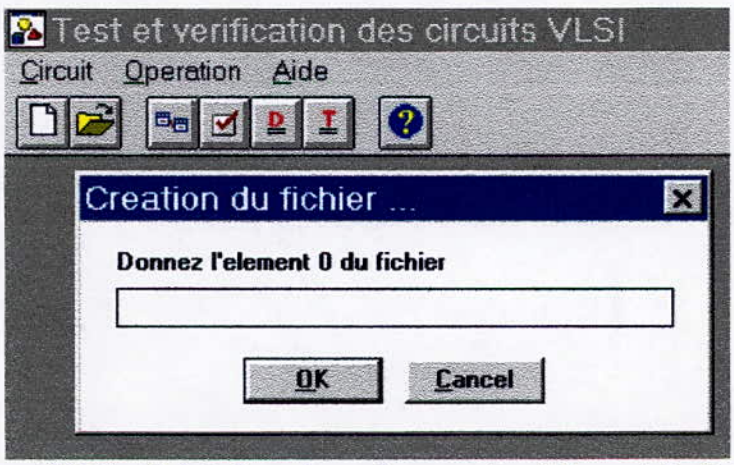
Figure(IV.5) :Génération de séquence pseudo-aleatoire (ici n =12)

IV.2.2.Organigramme d'analyse de signature : Soit un circuit intégré à n entrées. Si on arrive à générer m ($m<=2^n$) bits en série, (voir figure **(II. 4)** page 31), on peut trouver la signature (sign1) pour cette séquence de m bits, en utilisant l'organigramme de la figure *(***IV.6***)*. Si cette séquence change d'un seul bit, alors il faut que la signature (sign2) change aussi. Un avantage de l'utilisation d'analyse de signature est la détection des erreurs pendant un temps réduit (gain en temps) par rapport à la comparaison directe.

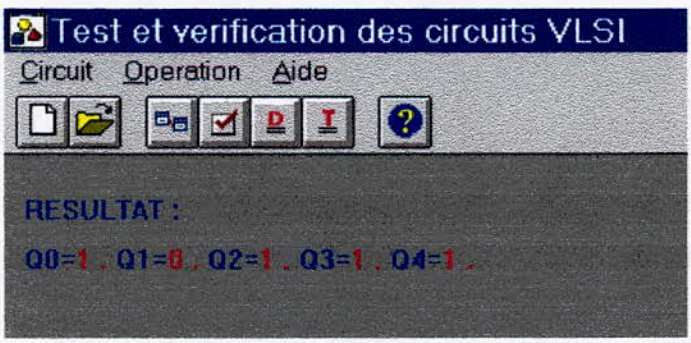


Figure(IV.6): ***Organigramme d'analyse de signature***

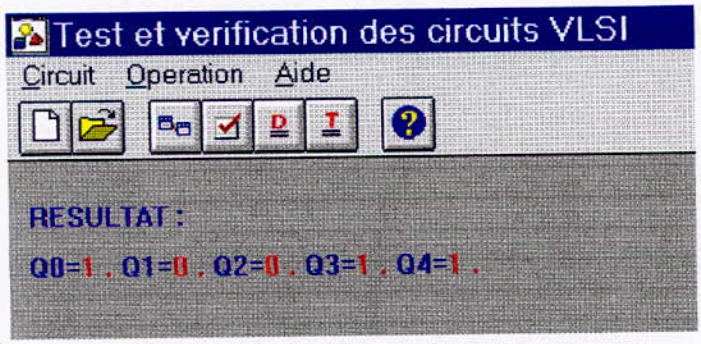
*Figure(IV.7):**Saisir le nombre de bits générer en série par un circuit intégré***



*Figure(IV.8): **Saisir la valeur du premier élément du fichier***



*Figure(IV.9) :**résultat de signature pour la séquence(11111000001111100000111110010**0**)***

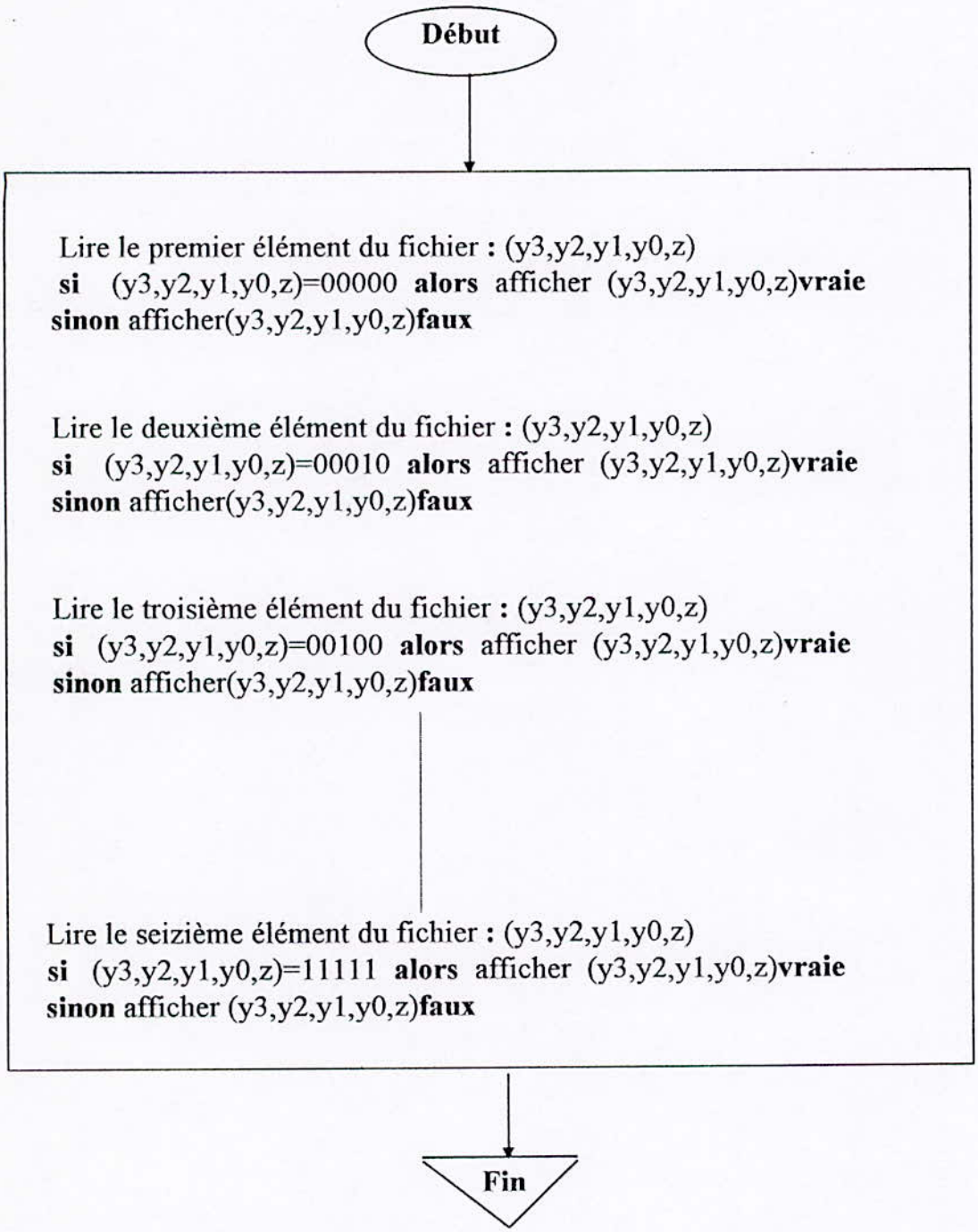


Figure(IV.10) ***:résultat de signature pour la séquence(11111000001111100001111100000)***

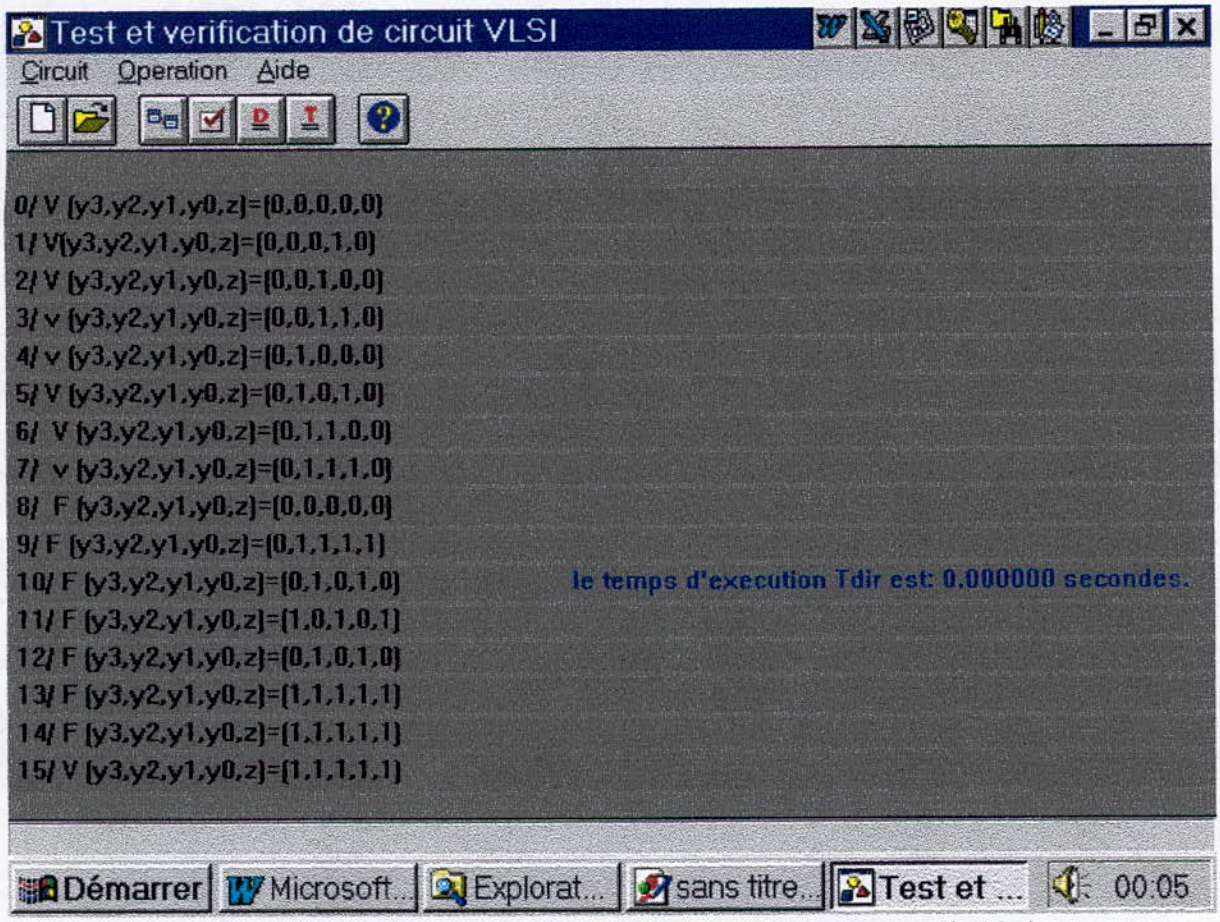
IV.3. SIMULATION DES METHODES DE VERIFICATION :

Soit un fichier de données d'un circuits intégré (dans notre exemple un compteur de 4 bits); ces données sont des données pratiques ou des données d'un modèle à vérifier, par exemple un concepteur d'un circuit intégré doit vérifier son fonctionnement en générant des données de fonctionnement réel puis on les compare à un modèle logique ou mathématique, s'il existe une différence on détecte des erreurs , d'ou le concepteur doit corriger l'erreur pendant sa conception.

IV.3.1. Méthode directe : Cette méthode est basée sur la comparaison terme à terme. Elle sera illustrée par un exemple d'un compteur de 4 bits avec une sortie z d'indication. L'organigramme de cette méthode est présenté par la figure ***(IV.11).***



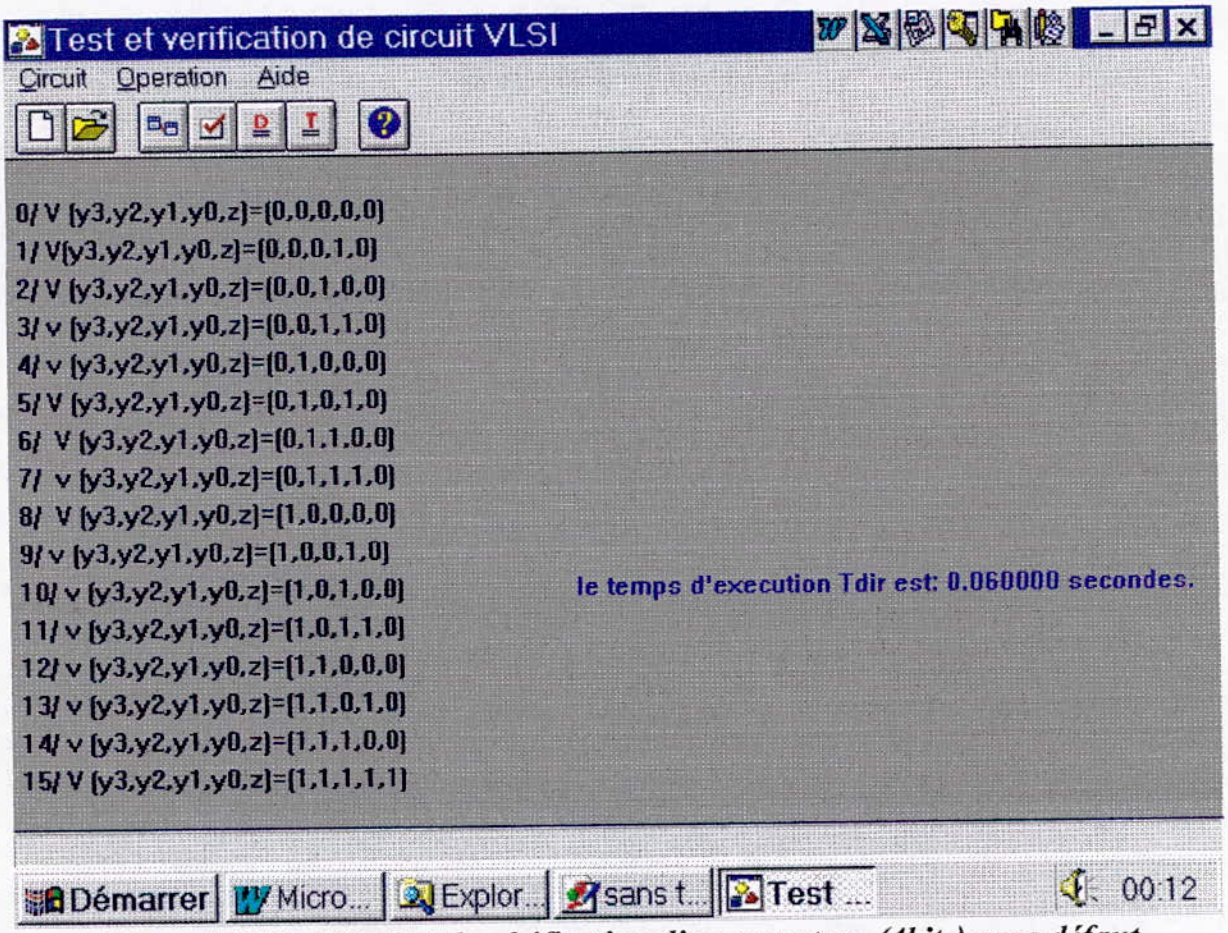
*Figure(IV.11): **Organigramme de la méthode directe de vérification d'un compteur à 4 bits***



*Figure(IV.12) :**Résultat de vérification du compteur (4 bits)***

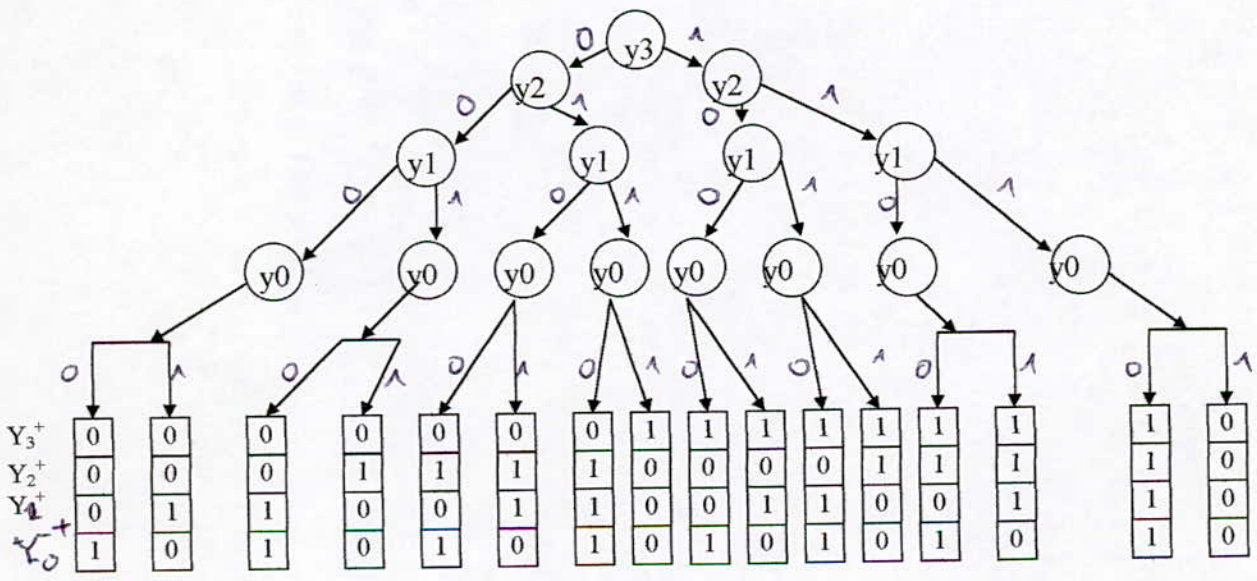
F : Fausse séquence

V : Vraie séquence

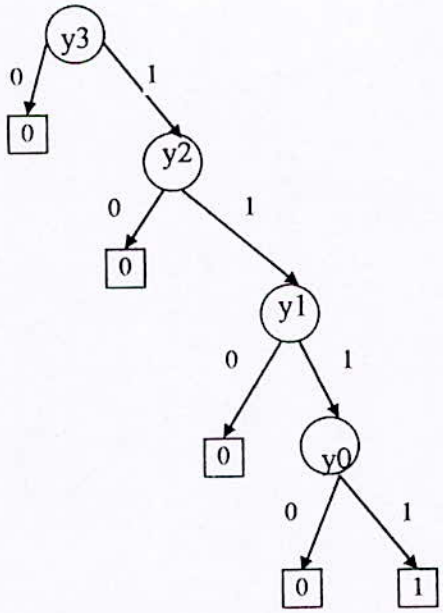


*Figure(IV.13) :**Résultat de vérification d'un compteur (4bits) sans défaut***

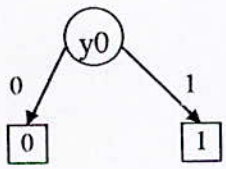
IV.3.2. Méthode transverse: Pour ce cas, on se limite pour la représentation de la fonction f (fonction d'incrementation) la plus optimale, en utilisant la technique de RBDD.



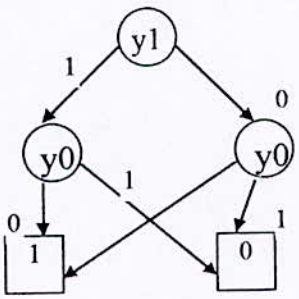
*Figure(IV.14) :**BDD de la fonction d'incrementation f***



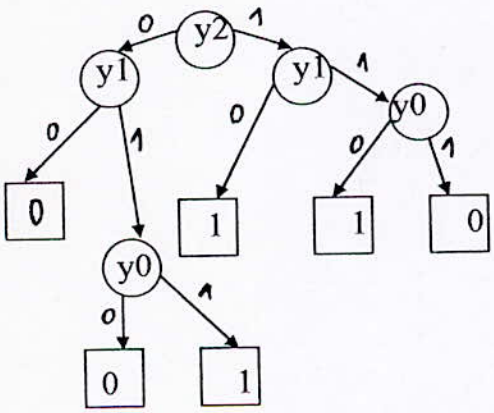
Figure(IV.15) :RBDD de Z



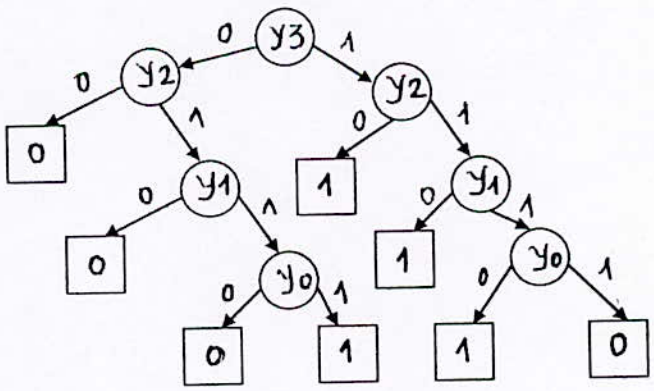
Figure(IV.16) :RBDD de $y0^+$



Figure(IV.17) :RBDD de $y1^+$



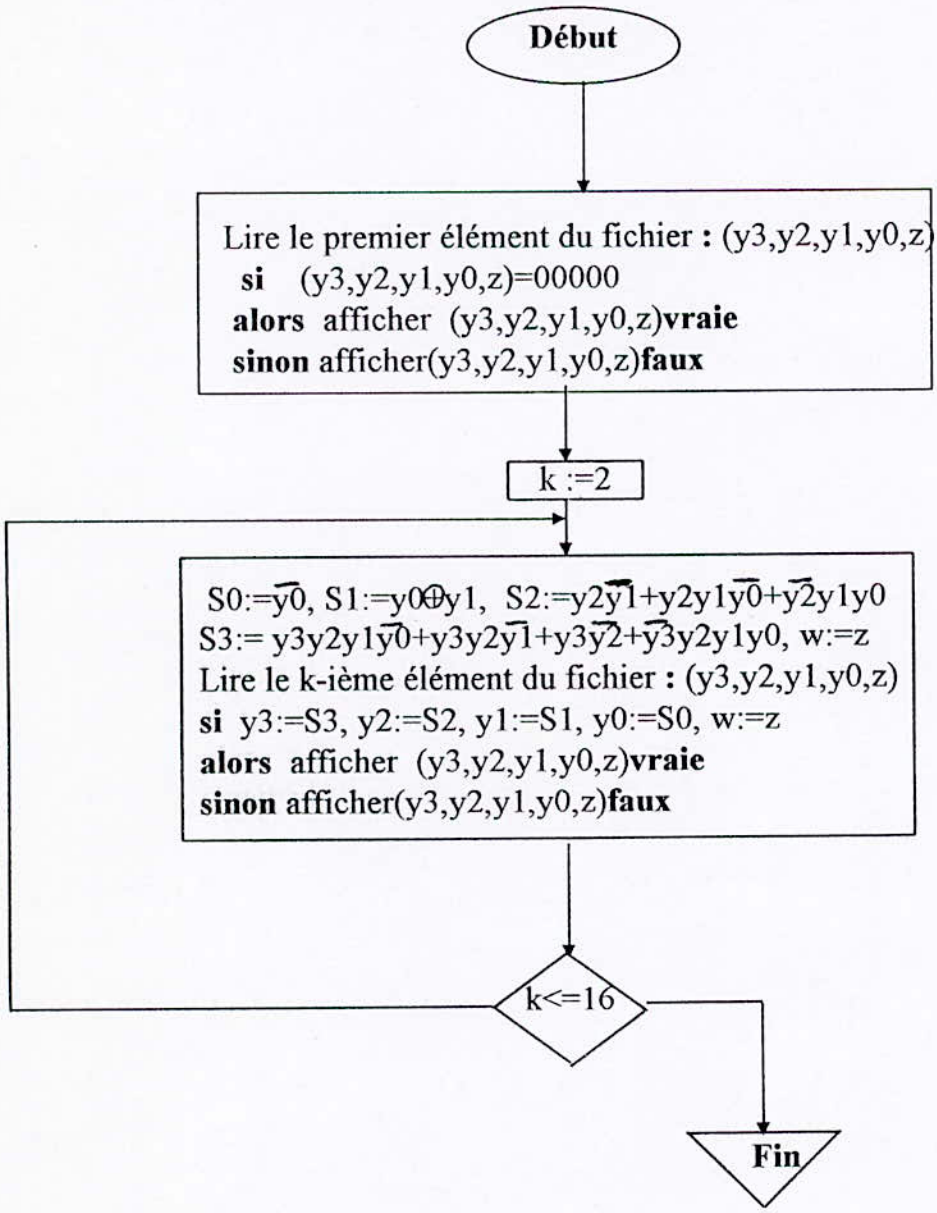
Figure(IV.18):RBDD de $y2^+$



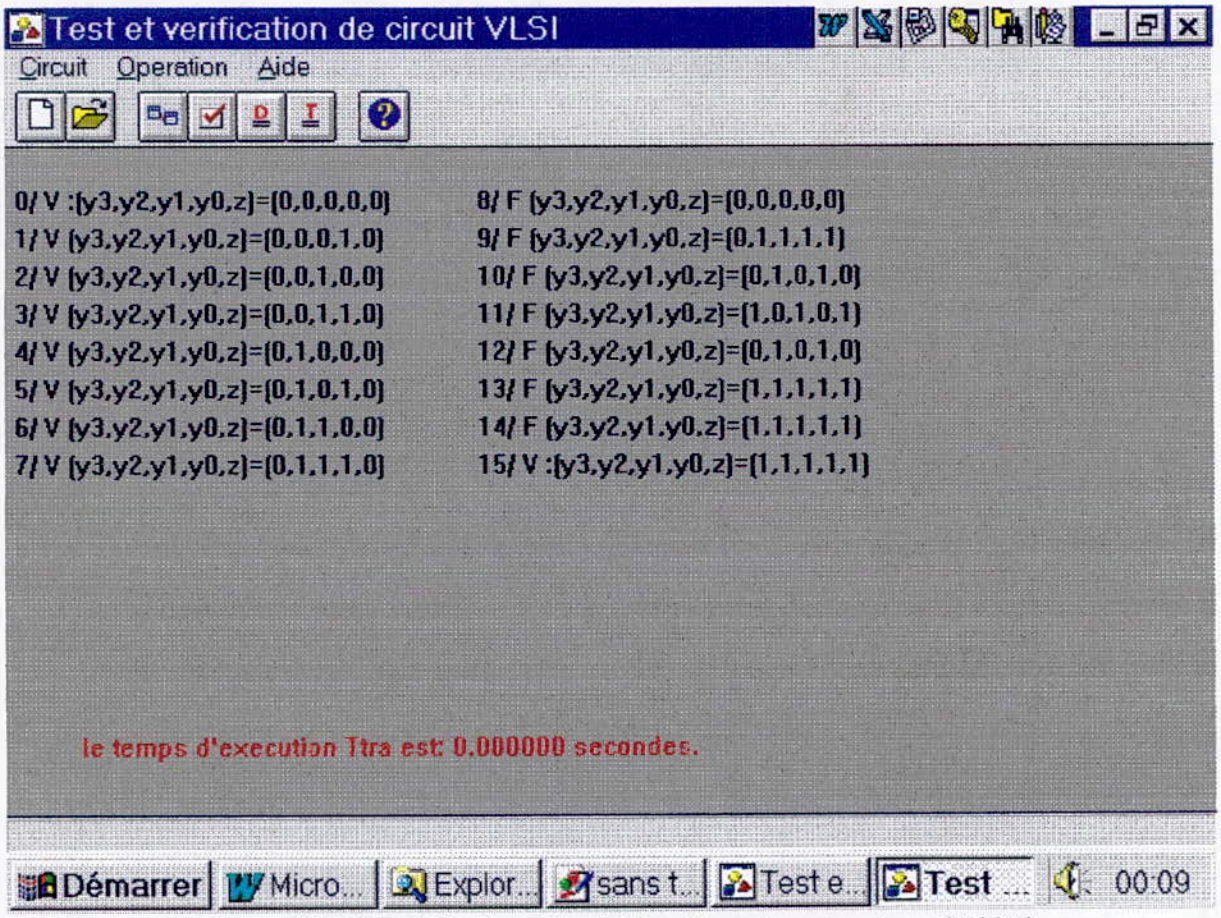
Figure(IV.19) :RBDD de $y3^+$

d'ou $y3^+y2^+y1^+y0^+z$ sont données par :

$y3^+ = y3y2y1\bar{y}0 + y3y2\bar{y}1 + y3\bar{y}2 + \bar{y}3y2y1y0$
$y2^+ = y2\bar{y}1 + y2y1\bar{y}0 + \bar{y}2y1y0$
$y1^+ = y0 \oplus y1$
$y0^+ = \bar{y}0$
$z = y3y2y1y0.$
d'où l'organigramme de la figure (*IV.20*) présente la méthode transverse.



*Figure(IV.20): **Organigramme de la méthode transverse de vérification d'un compteur à 4 bits***



Figure(IV.21) ***:Résultat de vérification du compteur (4 bits)***
F : Fausse séquence
V : Vraie séquence

IV.3.3. Comparaison:

D'après les résultats présentés dans le tableau (*IV.1*) par le micro ordinateur pentium 100MHZ ,on remarque que les deux méthodes présentent un temps d'exécution légèrement différent pour un faible nombre de circuits (compteurs).

Le temps d'exécution de la méthode directe est un peu plus réduit que la méthode transverse; cette différence va augmenter pour les grands circuits comme les VLSI, mais le concepteur veut utiliser la méthode transverse car elle est facile à implémenter pour la programmer (insérer seulement les formules de RBDD, avec une boucle for)tandis que la méthode directe prend du temps d'implémentation ; car il faut répéter les test plusieurs fois(20.000 fois pour un VLSI).
Conclusion : il faux faire un compromis de choix entre les deux méthodes.

Nombre de circuits	Fichier 1	Fichier2
700	T dir=7.47 seconde T tran =7.74 seconde	Tdir=7.64 seconde T tran=7.91 seconde
	T dir=7.58 seconde T tran =7.80 seconde	Tdir = 7.47 seconde Ttran=7.91 seconde
	T dir=7.47 seconde T tarn=7.91 seconde	Tdir=7.47 seconde Ttran=7.58 seconde
2500	T dir=27.08 seconde T tran =7.14 seconde	Tdir=27.73seconde T tran=28.56 seconde
	T dir=26.04 seconde T tran =27.02 seconde	Tdir = 26.91 seconde Ttran=27.85 seconde
	T dir=26.41 seconde T tarn=26.97 seconde	Tdir= 26.86 seconde Ttran=27.79 seconde
4000	T dir=41.69 seconde T tran =43.56 seconde	Tdir=42.62seconde T tran=44.16seconde
	T dir=41.69 seconde T tran =43.20 seconde	Tdir = 41.69 seconde Ttran=43.12seconde
	T dir=41.58 seconde T tarn=43.72 seconde	Tdir= 43.01 seconde Ttran=44.71 seconde

Tableau(IV.1) *:Comparaison du temps d'exécution pour les deux méthodes*

Conclusion

CONCLUSION

Le travail présenté dans ce mémoire porte sur la fiabilité, le test et la vérification des circuits intégrés, en particulier les circuits VLSI. Il englobe deux grandes parties :
Une partie théorique et une partie simulation :

Lors de l'élaboration de l'étude théorique, nous avons traité les fondements de base et la philosophie de la fiabilité, test et vérification :

- Le traitement des tests consiste en un test fonctionnel ayant ses propres méthodes et un test structurel ainsi que la testabilité.

- La vérification des machines séquentielles a été caractérisée par une présentation des méthodes de vérification, notamment la méthode transverse qui est une technique basée sur l'énumération implicite (donc un gain en temps pour le stockage) et sur des techniques d'optimisation de la structure de la machine séquentielle (RBDD, par exemple).

La partie simulation s'est caractérisée par :

Pour les tests : l'élaboration d'algorithmes de génération de séquences pseudo aléatoires et d'analyse de signatures;
Pour la vérification : traitement d'un exemple de compteur à quatre (04) bits (modulo 16), par le développement de deux algorithmes direct et transverse, avec une comparaison entre les deux algorithmes. Il s'avère que les deux méthodes présentent un temps d'exécution légèrement différent pour un faible nombre de circuits (compteurs). Toutefois, pour un nombre de circuits plus important, l'écart est plus significatif. Ceci a été confirmé par un exemple de simulation de 700 et 4000 circuits identiques de compteurs de quatre (04) bits. Ainsi, pour les circuits VLSI (200 000 circuits), le gain en temps (différence entre le temps d'exécution de la méthode transverse et le temps d'exécution de la méthode directe) est plus important, mais l'avantage de la méthode transverse réside dans la facilité de son implémentation dans le programme par rapport à la méthode directe qui prend du temps, d'où la nécessité d'un compromis entre les deux méthodes pour le choix.

Bien que beaucoup d'efforts restent à fournir pour aboutir à un véritable système informatique (Hard et Soft) orienté pour le test, nous pensons que nous avons préparé le terrain pour le lancement d'un nombre assez important de projets qui coopèrent en parallèle pour la réalisation du produit final désiré qui est un système informatique orienté pour le test. Pour cela, deux points essentiels devraient être développés :

Partie Software : développement des techniques et des méthodes de test et de vérification, afin de trouver un maximum de programme de gestion pour le test des circuits intégrés.

Partie Hardware : étude et réalisation des cartes Entrées/Sorties, pour PC et compatibles pour le test des circuits intégrés.

Bibliographie

[1] Abhijit Ghosh, Svinivas Devadas, A.Richard Newton
Sequential Logic Testing and Verification: ***(Kluwer AcademicPublishers 1992)***

[2] Alain pagès et Michel Goudrau
fiabilité des systèmes

[3] A.Farah et H.Bousbia
New synthesis method for finite state machines
(Journal of technologie,Ecole nationale polytechnique d'Alger 1995, page1)

[4] C.piguet , A.stauffer, J.zahnd
conception des circuits ASIC : numériques CMOS

[5] Turbo C, Gerard Leblanc

[6] Borland C++ Programmation Windows
Gerard Leblanc

[7] Henri Félix
test des ensembles à base de VLSI ***(technique de l'ingénieur 1996)***

[8] Kwang Iing cheng , Vishwari D .A grawal
Unified methods for VLSI simulation and verification and test generation
(Kluwer Academic Publishers 1993)

[9] Laidi Kamel et Boubakir Chaabane
Simulation en VHDL et génération de test :application à un multiplieur
(thèse d'ingénieur :juin 1995)

[10] Michel Moreau
test des ***VLSI*** *numérique* ***(technique de l'ingénieur 1996)***

[11] M .Haddadi
technologies et techniques de conception : cours et exercices

[12] Paul blanquart et Jean claude roucin
Fiabilité ***(technique de l'ingénieur 1996)***

[13] S.L. Hurst
VLSI :Testing and testability considerations :an overview(***I.E.E.E 1990)***

[14] Un défi : le test des circuits intégrés VLSI :
(Ecole Polytechnique de Lausanne 1983)

Annexe

SALIM.CPP

```
#include<owl\applicat.h>
#include<owl\dc.h>
#include<owl\dialog.h>
#include<OWL\opensave.h>
#include<owl\inputdia.h>
#include<OWL\buttonga.h>
#include<owl\controlb.h>
#include<owl\framewin.h>
#include<owl\decframe.h>
#include<owl\messageb.h>
#include<dos.h>
#include"salim.h"
#include<stdio.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <time.h>

int n;
int e[30],d2[30],f[30],o[30],a[30];
int l[]={0,0,1,1,0,1,0,1,1,1,1,1};
int q[]={0,0,0,0,0};
int q1[]={0,0,0,0,0};
char f_test[40];
clock_t debut,fin;

FILE *hf,*p;

struct sortie
{
int e[10];
int f[10];
}sortief;

struct comp4
{
int y3;
int y2;
int y1;
int y0;
int z;
}fich;
//*****************************************************************************
void direct_compteur_4bits();
void trans_compteur_4bits();
void test_pseudo_aleat();
void analyse_signature();
//*****************************************************************************
        int pos_de_i,pos_de_j;
        char NomFichier[80];
        TDib *pdib;
        TPalette *ppal;
        TDibDC  *pdibdc;
        BYTE  image[200][200];
```

```
//*************************
class Tbd_about:public TDialog
{
 TDib    *image;
 TPalette *imagepal;
 void    EvPaint();
 void SetupWindow();
 public:
 Tbd_about(TWindow *parent,LPSTR titre):TDialog(parent,titre)
 {
  image=new TDib(*GetApplication(),im_salim);
  imagepal=new TPalette(*image);
  SetBkgndColor(TColor::LtGray);
 }
 ~Tbd_about(){delete image;delete imagepal;}
DECLARE_RESPONSE_TABLE(Tbd_about);
};
DEFINE_RESPONSE_TABLE1(Tbd_about,TDialog)
 EV_WM_PAINT,
END_RESPONSE_TABLE;

void Tbd_about::EvPaint()
{
 TPaintDC dc(*this);
 TDialog::DefaultProcessing();
 dc.SelectObject(*imagepal);
 dc.RealizePalette();
 TRect rc(1,1,image->Width(),image->Height());
 dc.SetDIBitsToDevice(rc,TPoint(0,0),*image);
}

void Tbd_about::SetupWindow()
{
 TDialog::SetupWindow();
 TRect r=GetWindowRect();
 MoveWindow(200,110,r.Width(),r.Height(),FALSE);
 Invalidate();
}

//*****************************************************************************

class Tfenapp:public TWindow
{
  void open();
  void creer();
  void test_pseudo_aleat();
  void analyse_signature();
  void direct_compteur_4bits();
  void trans_compteur_4bits();
  void quiter();
  void help();
  void Paint(TDC&,BOOL,TRect&);
public:
  Tfenapp():TWindow(0,0,0)
  {
        SetBkgndColor(TColor::Gray);
  }
```

```
 DECLARE_RESPONSE_TABLE(Tfenapp);
};
 DEFINE_RESPONSE_TABLE1(Tfenapp,TWindow)
        EV_COMMAND(id_ouv,open),
        EV_COMMAND(id_cre,creer),
        EV_COMMAND(id_psd,test_pseudo_aleat),
        EV_COMMAND(id_ans,analyse_signature),
        EV_COMMAND(id_dc4,direct_compteur_4bits),
        EV_COMMAND(id_tc4,trans_compteur_4bits),
        EV_COMMAND(id_hlp,help),
        EV_COMMAND(id_qui,quiter),
 END_RESPONSE_TABLE;

///*************************************************************************

class Tapp:public TApplication
{
 public:
        TDecoratedFrame *fenetreapp;
        TMessageBar *messagebar;
        TControlBar *bouttonbar;
        void InitMainWindow();
        Tapp() : TApplication(){}
};
void Tapp::InitMainWindow()
 {
          fenetreapp =new TDecoratedFrame(NULL,"Test et verification de circuit VLSI",new
Tfenapp,TRUE);
        messagebar=new TMessageBar(fenetreapp);
        fenetreapp->Insert(*messagebar,TDecoratedFrame::Bottom);
        bouttonbar=new TControlBar(fenetreapp);

        TButtonGadget *bt1,*bt2,*bt3,*bt4,*bt5,*bt6,*bt7;

        bt1=new TButtonGadget(im_moi,id_cre,TButtonGadget::Command,TRUE);
        bt2=new TButtonGadget(im_ouv,id_ouv,TButtonGadget::Command,TRUE);
        bouttonbar->Insert(*bt1);
        bouttonbar->Insert(*bt2);
        bouttonbar->Insert(*new TSeparatorGadget(12));

        bt3=new TButtonGadget(im_nou,id_ans,TButtonGadget::Command,TRUE);
        bt4=new TButtonGadget(im_cou,id_dc4,TButtonGadget::Command,TRUE);
        bt5=new TButtonGadget(im_efa,id_tc4,TButtonGadget::Command,TRUE);
        bt6=new TButtonGadget(im_bru,id_psd,TButtonGadget::Command,TRUE);
        bouttonbar->Insert(*bt6);
        bouttonbar->Insert(*bt3);
        bouttonbar->Insert(*bt4);
        bouttonbar->Insert(*bt5);
        bouttonbar->Insert(*new TSeparatorGadget(12));
        bt7=new TButtonGadget(im_help,id_hlp,TButtonGadget::Command,TRUE);
        bouttonbar->Insert(*bt7);
        fenetreapp->Insert(*bouttonbar);
         bouttonbar->TGadgetWindow::SetHintMode(2);
        MainWindow=fenetreapp;
        nCmdShow=SW_SHOWMAXIMIZED;
        MainWindow->Attr.AccelTable="racourci";
        MainWindow->AssignMenu("choixaction");
        MainWindow->EnableKBHandler();
```

```
        EnableCtl3d();EnableCtl3dAutosubclass(TRUE);//EnableCTl3d(TRUE);}
//*************************************************************************

void Tfenapp::creer()
{

}
///************************************************************************
void Tfenapp::help()
{
Tbd_about(this,"diag_about").Execute();
}

//*************************************************************************

 void Tfenapp::quiter()
 {
        int i;
        i=TWindow::MessageBox("VOULEZ VOUZ VRAIMENT QUITER\n\t L'application",
        "Testeur VLSI",MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL);
        if(i==IDYES) TWindow::CloseWindow();
 }

///************************************************************************
void Tfenapp::open()
{
        char Filter[]="fibmp *.*|*.";
        static TFileOpenDialog::TData data(OFN_FILEMUSTEXIST,Filter);
        TFileOpenDialog ofd(this,data);
        if(ofd.Execute()==IDOK)
         {
         }
}
//*************************************************************************/
void Tfenapp::Paint(TDC& dc,BOOL, TRect&)
{
}
//*************************************************************************

int OwlMain(int,char *[])
{
 return Tapp().Run();
}
//*************************************************************************


void Tfenapp::test_pseudo_aleat()
{
int h=1,c1=1,c2=0,e;
TClientDC dc(*this);
dc.SetBkMode(TRANSPARENT);
char buf[30];
int res=TInputDialog(this,"Nombre de bits ...","Donnez le nombre de bits à génèrer n:",
                                                        buf,sizeof buf).Execute();
if(res==IDOK )
{
 n=atoi(buf);
 char f[25],g[30];
```

```
for(e=0;e<12;e++)
{
sprintf(f,"Saisie de L%d ...",e);
sprintf(g,"Donnez la valeur de L%d:",e);
res=TInputDialog(this,f,g,buf,sizeof buf).Execute();
if(res==IDOK ) l[e]=atoi(buf);
else break;
}

dc.SetTextColor(TColor::LtBlue);
dc.TextOut(10,20,"Resultat d21:");
dc.MoveTo(10,45);
dc.SetTextAlign(1);
dc.SetTextColor(TColor::LtRed);

char b[20];

while(h<=n)
{
for(int i=11;i>0;i--)
{
d2[1]=l[11];
l[i]=l[i-1];
}
l[0]=l[0]*(1-l[3])+l[3]*(1-l[0]);

sprintf(b,"%d ",d2[1]);
dc.TextOut(150,5,b);
if(h<n) dc.TextOut(150,5," , ");
else   dc.TextOut(150,5," ...");

h++;
}

}

}

//---------------------------------------------------------------------------

void Tfenapp::analyse_signature()
{
int i,l,v;

TClientDC dc(*this);
dc.SetBkMode(TRANSPARENT);
char buf[30];
int res=TInputDialog(this,"Nombre de bits...","Donnez le nombre de bits à génèrer n:",
                                                    buf,sizeof buf).Execute();
if(res==IDOK )
{
n=atoi(buf);

int nat;
nat=TWindow::MessageBox("Si vous voulez tester un fichier existant cliquer sur Oui\n si vous voulez creer un
nouveau fichier pour le test cliquer Non",
"Testeur VLSI",MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL);
```

```
if(nat==IDYES)
{
 char Filter[]="fibmp *.*|*.*";
 static TFileOpenDialog::TData data(OFN_FILEMUSTEXIST,Filter);
 TFileOpenDialog ofd(this,data);
 if(ofd.Execute()==IDOK) strcpy(f_test,data.FileName);
}

if(nat==IDNO)
{
int re;
char el[40],bu[3];

for(int y=0;y<n;y++)
{
sprintf(el,"Donnez l'element %d du fichier",y);
re=TInputDialog(this,"Creation du fichier ...",el,bu,sizeof bu).Execute();
if(re==IDOK ) a[y]=atoi(bu);
}

char Filter[]="fibmp *.*|*.*";
static TFileSaveDialog::TData data(OFN_FILEMUSTEXIST,Filter);
TFileSaveDialog ofd(this,data);
if(ofd.Execute()==IDOK)
{
 strcpy(f_test,data.FileName);
 p=fopen(f_test,"w+b");
 for(y=0;y<n;y++) fwrite(&a[y],sizeof(a[y]),1,p);
 fclose(p);
}
}

 q[0]=0;q[1]=0;q[2]=0;q[3]=0;q[4]=0;
 p=fopen(f_test,"rb");
 for(i=0;i<n;i++)
 {
 fread(&a[i],sizeof(a[i]),1,p);
 q1[0]=(1-q[4])* a[i]+q[4]*(1-a[i]);
 q1[1]=(1-q[0])*q[4]+q[0]*(1-q[4]);
 q1[2]=q[1];
 q1[3]=(1-q[4])*q[2]+q[4]*(1-q[2]);
 q1[4]=q[3];
 q[0]=q1[0];q[1]=q1[1];q[2]=q1[2];q[3]=q1[3];q[4]=q1[4];
 }

 char b[10];

 dc.SetTextColor(TColor::LtBlue);
 dc.TextOut(10,20,"RESULTAT : ");

 dc.MoveTo(10,45);
 dc.SetTextAlign(1);

 for(i=0;i<5;i++)
 {
 dc.SetTextColor(TColor::LtBlue);
 sprintf(b,"Q%d=",i);
 dc.TextOut(150,5,b);
```

```
dc.SetTextColor(TColor::LtRed);
sprintf(b,"%d . ",q[i]);
dc.TextOut(150,5,b);
}
fclose(p);
}
}

//---------------------------------------------------------------------------
void Tfenapp::direct_compteur_4bits()
{
int nbz0=0,nbz1=0;
int i,y3,y2,y1,y0,z,code[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

TClientDC dc(*this);
dc.SetBkMode(TRANSPARENT);
char b[60];


//fich_compteur();
int nat;
nat=TWindow::MessageBox("Si vous voulez tester un fichier existant cliquer sur Oui\n si vous voulez creer un
nouveau fichier pour le test cliquer Non",
"Testeur VLSI",MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL);

if(nat==IDYES)
{
 char Filter[]=" *.*|*.";
 static TFileOpenDialog::TData data(OFN_FILEMUSTEXIST,Filter);
 TFileOpenDialog ofd(this,data);
 if(ofd.Execute()==IDOK) strcpy(f_test,data.FileName);
}

if(nat==IDNO)
{
int re;
char el[40],bu[3];

char Filter[]="fibmp *.*|*.*";
static TFileSaveDialog::TData data(OFN_FILEMUSTEXIST,Filter);
TFileSaveDialog ofd(this,data);
if(ofd.Execute()==IDOK)
{
 strcpy(f_test,data.FileName);
 p=fopen(f_test,"w+b");

for(int y=0;y<16;y++)
{
sprintf(el,"creation de la sequence %d ...)",y);
re=TInputDialog(this,el,"Donnez la valeur de y3 :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.y3=atoi(bu);
re=TInputDialog(this,el,"Donnez la valeur de y2 :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.y2=atoi(bu);
re=TInputDialog(this,el,"Donnez la valeur de y1 :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.y1=atoi(bu);
re=TInputDialog(this,el,"Donnez la valeur de y0 :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.y0=atoi(bu);
re=TInputDialog(this,el,"Donnez la valeur de z :",bu,sizeof bu).Execute();
```

```
if(re==IDOK ) fich.z=atoi(bu);
fwrite(&fich,sizeof(fich),1,p);
}

fclose(p);
}
}

debut = clock();
hf = fopen(f_test,"r+b");
int posx=5,posy=0;
for(int k;k<=700;k++){
  posx=5;posy=0;
fread(&fich,sizeof(fich),1,hf);
y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z;posy+=20;
//if(i==9){posx=250;posy=20;}
 i=0;
 if(y3==0 && y2==0 && y1==0 && y0==0 && z==0)
          {sprintf(b,"%d/ V (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",i,y3,y2,y1,y0,z);
           dc.TextOut(posx,posy,b); fread(&fich,sizeof(fich),1,hf);
           y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z;posy+=20;i=1;}
  else{sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",i,y3,y2,y1,y0,z);
           dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
           y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z;posy+=20;i=1;}



if(y3==0 && y2==0 && y1==0 && y0==1 && z==0){
          sprintf(b,"%d/ V(y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",i,y3,y2,y1,y0,z);
          dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
           y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z;posy+=20;i=2; }
          else{sprintf(b,"%d/ F(y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",i,y3,y2,y1,y0,z);
          dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z;posy+=20;i=2;}



if(y3==0 && y2==0 && y1==1 && y0==0 && z==0) {
          sprintf(b,"%d/ V (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
          dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
          y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=3;}
else {sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
          dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z;posy+=20; i=3;}



if(y3==0&&y2==0&&y1==1&&y0==1&&z==0){
          sprintf(b,"%d/ v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
          dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
          y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=4;}
else{

sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
          dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
          y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=4;}
```

```
if(y3==0&&y2==1&&y1==0&&y0==0&&z==0){
        sprintf(b,"%d/ v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=5;}
else{
        sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=5;}



if(y3==0&&y2==1&&y1==0&&y0==1&&z==0){
        sprintf(b,"%d/ V (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=6;}
else{
        sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=6;}



if(y3==0&&y2==1&&y1==1&&y0==0&&z==0){
        sprintf(b,"%d/  V (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=7;}
else{
sprintf(b,"%d/  F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",i,y3,y2,y1,y0,z);
 dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
  y3=fich.y3;y2=fich.y2 ; y1=fich.y1;y0=fich.y0 ;z=fich.z;posy+=20;i=7;}



if(y3==0&&y2==1&&y1==1&&y0==1&&z==0){
        sprintf(b,"%d/  v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=8;}
else{
        sprintf(b,"%d/  F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=8;}



if(y3==1&&y2==0&&y1==0&&y0==0&&z==0){
        sprintf(b,"%d/  V (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=9;}
else{
        sprintf(b,"%d/  F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
```

```
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=9;}

if(y3==1&&y2==0&&y1==0&&y0==1&&z==0){
        sprintf(b,"%d/ v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=10;}
else{
        sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=10;}



if(y3==1&&y2==0&&y1==1&&y0==0&&z==0){
        sprintf(b,"%d/ v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=11;}
else{
        sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=11;}



if(y3==1&&y2==0&&y1==1&&y0==1&&z==0){
        sprintf(b,"%d/ v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=12;}
else{
        sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=12;}



if(y3==1&&y2==1&&y1==0&&y0==0&&z==0){
        sprintf(b,"%d/ v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=13;}
else{
        sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=13;}



if(y3==1&&y2==1&&y1==0&&y0==1&&z==0){
        sprintf(b,"%d/ v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=14;}
else{
        sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=14;}
```

```
{
int re;
char el[40],bu[3];

char Filter[]="fibmp *.*|*.*";
static TFileSaveDialog::TData data(OFN_FILEMUSTEXIST,Filter);
TFileSaveDialog ofd(this,data);
if(ofd.Execute()==IDOK)
{
 strcpy(f_test,data.FileName);
 p=fopen(f_test,"w+b");

for(int y=0;y<16;y++)
{
sprintf(el,"creation de la sequence  %d ...)",y);
re=TInputDialog(this,el,"Donnez la valeur de y3 :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.y3=atoi(bu);
re=TInputDialog(this,el,"Donnez la valeur de y2 :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.y2=atoi(bu);
re=TInputDialog(this,el,"Donnez la valeur de y1 :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.y1=atoi(bu);
re=TInputDialog(this,el,"Donnez la valeur de y0 :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.y0=atoi(bu);
re=TInputDialog(this,el,"Donnez la valeur de z :",bu,sizeof bu).Execute();
if(re==IDOK ) fich.z=atoi(bu);
fwrite(&fich,sizeof(fich),1,p);
}

fclose(p);
}
}

debut = clock();
hf = fopen(f_test,"r+b");
int posx=5,posy=0;
for(int k; k<=700;k++){
posx=5;posy=0;
fread(&fich,sizeof(fich),1,hf);
y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z;
posy+=20;

if(y3==0&&y2==0&&y1==0&&y0==0&&z==0)
        sprintf(b,"%d/ V :(y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",0,y3,y2,y1,y0,z);
        else sprintf(b,"%d/ F :(y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",0 ,y3,y2,y1,y0,z);
         dc.TextOut(posx,posy,b);
         for(int m=1;m<15;m++)
         {
                s3=y3*y2*y1*(1-y0)+y3*y2*(1-y1)+y3*(1-y2)+(1-y3)*y2*y1*y0;
                s2=(1-y2)*y1*y0+y2*(1-y1)+y2*y1*(1-y0);
                s1=y0*(1-y1)+y1*(1-y0);
                s0=(1-s0);
                fread(&fich,sizeof(fich),1,hf);
                y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z;
                 posy+=20;if(m==8){posx=250;posy=20;}
                 if(y3==s3 && y2==s2 && y1==s1 && y0==s0 && z==0)

          sprintf(b,"%d/ V (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",m,y3,y2,y1,y0,z);
                  else
```

```
if(y3==1&&y2==1&&y1==1&&y0==0&&z==0){
        sprintf(b,"%d/ v (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=15;}
else{
        sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);fread(&fich,sizeof(fich),1,hf);
        y3=fich.y3;y2=fich.y2;y1=fich.y1;y0=fich.y0;z=fich.z; posy+=20;i=15;}



if(y3==1&&y2==1&&y1==1&&y0==1&&z==1){
        sprintf(b,"%d/ V (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d) ",i,y3,y2,y1,y0,z);
        dc.TextOut(posx,posy,b);
 }
 else{
 sprintf(b,"%d/V (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",i,y3,y2,y1,y0,z);
        dc.TextOut (posx,posy,b);
 }
         }

fclose(hf);
fin = clock();
sprintf(b,"le temps d'execution Tdir est: %f secondes.",(fin - debut)/CLK_TCK);
dc.SetTextColor(TColor::LtBlue);
dc.TextOut(300,posy-100,b);

}
//---------------------------------------------------------------------------

void Tfenapp::trans_compteur_4bits()

{
int nbz0=0,nbz1=0;
int y3,y2,y1,y0,z, code[]={0,0,0,0,0,0,0,0,0,0,0,0};
int i,s3,s2,s1,s0,w;
TClientDC dc(*this);
dc.SetBkMode(TRANSPARENT);
char b[30];

int nat;
nat=TWindow::MessageBox("Si vous voulez tester un fichier existant cliquer sur Oui\n si vous voulez creer un
nouveau fichier pour le test cliquer Non",
"Testeur VLSI",MB_YESNO|MB_ICONQUESTION|MB_SYSTEMMODAL);

if(nat==IDYES)
{
 char Filter[]=" *.*|*.";
 static TFileOpenDialog::TData data(OFN_FILEMUSTEXIST,Filter);
 TFileOpenDialog ofd(this,data);
 if(ofd.Execute()==IDOK) strcpy(f_test,data.FileName);
 }

 if(nat==IDNO)
```

```
            sprintf(b,"%d/ F (y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",m,y3,y2,y1,y0,z);
                    dc.TextOut(posx,posy,b);
            }
            posy+=20;
 if(y3==1&&y2==1&&y1==1&&y0==1&&z==1)
          sprintf(b,"%d/ V :(y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",15,y3,y2,y1,y0,z);
          else sprintf(b,"%d/ F :(y3,y2,y1,y0,z)=(%d,%d,%d,%d,%d)",15 ,y3,y2,y1,y0,z);
          dc.TextOut(posx,posy,b);
}
fclose(hf);
fin = clock();
sprintf(b,"le temps d'execution Ttra est: %f secondes.",(fin - debut) /CLK_TCK);
dc.SetTextColor(TColor::LtRed);
dc.TextOut(40,posy+150,b);
}
```

Salim.h

```
#define CM_KEY312      312
#define BITMAP_13      13
#define im_salim2011
#define im_nouv 2000
#define id_nouv  300000
#define BITMAP_9       9
#define nou_im      1
#define im_ouv     2
#define im_cou     30102
#define im_bru     7
#define im_moi     8
#define im_help    2010
#define im_hlp       2002

#define  id_psd     31
#define  id_ans     32
#define  id_dc4     33
#define  id_tc4     34

#define  im_sav     101
#define  im_nou      106
#define  im_efa     107
#define  im_pen      108
#define  id_cou     109
#define  id_nfch     110
#define  id_nres     111
#define  id_nbas     112

#define  id_efa     113
#define  id_alpha    114

#define  id_taux     115
#define  id_ermin    116

#define  id_bru     39

#define  id_nob     117
#define  id_der     118
#define  id_par     119
#define  id_svb     120
#define  id_ovb     121
#define  id_nos     122
#define  id_nbe     124
#define  id_scv     125
#define  id_sch     126
#define  id_apr     127

#define  id_trec    220
#define  id_cre     221
#define  id_inv     222
#define  id_seg     224
#define  id_niv     226
#define  id_sta     227
```

```
#define id_zoom    228
#define id_moi     229

#define id_ouv     311
#define id_sav     313
#define id_sav_as  314
#define id_qui     315
#define id_app     317
#define id_mod     318
#define id_nou     319
#define id_nouar   320
#define id_rec     321
#define id_hlp     322
#define id_bou     323
#define id_bin     324
#define id_enr     336
#define id_abo     337
#define im_about   2010
```