Ministère de l'Enseignement Supèrieur et de la Rechèrche Scientifique

12/95

Ecole Nationale Polytechnique

Département d'ELECTRONIQUE

PROJET DE FIN D'ETUDES

Intitulé:

SIMULATION EN VHDL et GENERATION DE TEST: APPLICATION à UN MULTIPLIEUR

Proposé et Dirigé par:

M^r A.FARAH

Etudié par:

LAIDI Kamel

et

BOUBAKIR Châabane

PROMOTION : Juin 1995

Ministère de l'Enseignement Supèrieur et de la Rechèrche Scientifique

Ecole Nationale Polytechnique

Département d ' ELECTRONIQUE

PROJET DE FIN D'ETUDES

Intitulé:

SIMULATION EN VHDL et GENERATION DE TEST: APPLICATION à UN MULTIPLIEUR

Proposé et Dirigé par:

M' A.FARAH

Etudié par:

LAIDI Kamel

et

BOUBAKIR Châabane

PROMOTION : Juin 1995

DEDICACES

A ma chère mère et mon cher père

à tous mes frères et soeurs

à toute ma famille

à tous mes amis

à tous les etudiants résidents à BOURAOUI AMMAR.

je dédie ce travail .

Kamel

je dédie ce modèste travail:

à ma chère mère et mon cher père

à mes frères: Messaoud, Ahcène, à mes soeurs

à mes neveux, à toute ma famille, en particulier mon oncle Ikhlef Salah

à tous mes amis

à tous qui ont de près ou de loin assisté à ma formation.

Châabane

Remerciements

Nous tenons à remercier vivements:

Monsieur A.FARAH , notre promoteur qui a bien voulu nous diriger et pour avoir mis à notre disposition les moyens nécessaires dûrant toute la période de l'élaboration de ce travail.

Monsieur M.TOUNSI , P.G à l'ENP ,laboratoireTDS , pour son aide morale et documentaire.

Monsieur A.NASRI, Magister attaché de rechèrche à l'ENP,laboratoire TDS, qui nous a aidé dûrant toute sa présence au laboratoire .

Monsieur TAGZOUT, ingenieur attaché de rechèrche au CDTA qui nous a donné le premier coup de main pour commencer notre travail.

Monsieur H.BOUSBIA-SALAH, maitre assistant à l'ENP pour son aide documentaire.

ملخص

يتناول هذا العمل موضوعين اساسيين:

الموضوع الاول يتطرق الى كيفية محاكاة الدارات المنطقية بواسطة لغة الحاسوب المسماة **vhdl** المصممة لوصف الدارات المتكاملة ذات الدرجة العالية ، ثم تطبيقها باعداد برنامج لدارة ضرب للاعداد الطبيعية .

الموضوع الثاني يتطرق الى مختلف المناهج المستعملة في اختبار الدارات للتحقق من الدور المنوط بها ، ثم توليد اختبار متسلسل لدارة الضرب و تطبيقه عليها.

ABSTRACT:

This work contains two main parts:

The first treat the simulation manner of the logic circuits using the hardware description language VHDL for the VLSI circuits, an application of this language to elaborate a program to the entire numbers multiplier.

The second treat the test and testability of the logic circuits, the various approaches to check their fonctionnalités, at last, we have applied an hiérarchical test génération approach for the multiplier.

RESUME:

Ce travail contient deux parties principales:

La première traite la manière de la simulation en VHDL des circuits logiques par le langage de description de matèriel VLSI nommé VHDL, ainsi q'une application de ce langage pour élaborer un programme d'un multiplieur des nombres entiers.

La deuxième traite le test et la testabilité des circuits intégrés, les différentes approches de vérification de leurs fonctionnalités, puis une application d'une approche hiérarchique de génération de test pour le multiplieur simulé.

SOMMAIRE

INTRODUCTION GENERALE

La conception des circuits à très grande échelle d'intégration est un domaine qui a subi une grande évolution . Un des facteur qui alimente cette évolution est la croissance de plus en plus du nombre de transistors par unité de surface . En effet, même dans les dernières années seulement on est passé de quelques milliers de transistors par unité de surface à quelques millions de transistors sur la même surface . Par conséquent, les outils de conception et de vérification ont subi un grand développement pour chaque phase de fabrication car chacune de ces phases constitue une source d'erreurs des outils utilisés pour remidier à ce problème est l'apparition de plusieurs langages de simulation et de vérification avec différents niveaux d'abstraction des circuits VLSI (very large scale integration). parmi ces langages on trouve le VHDL qui est d'un niveau haut d'abstraction .

La simulation et la vérification de la conception utilisant VHDL est devenue extrèment populaire dans les domaines industriels informatiques. La raison principale est que VHDL possède les meilleurs outils de conception pour la modélisation comportementale du matériel .
Utilisant VHDL, les conceptions peuvent être décrites soit en mode descendant ou en mode ascendant à travers des niveaux d'abstractions variants.

Une fois le modèle comportemental VHDL du circuit est développé, les données de test sont générées pour la simulation du modèle par l'observation de la sortie de la simulation, donc la fonctionnalité du modèle peut être vérifiée. Traditionnellement, le développement du test pour les modèles comportementaux est une tâche pour l'ingénieur de conception ou le développeur du modèle à construire les tests. C'est une tâche de manoeuvre intensive et très consommante de temps. De plus , est une approche qui produit des ensembles de test qui ne satisfaient pas une définition complete et formelle .Les tests cités attirent le point de vue de l'écrivain sur quel modèle doit le faire. Comme résultat des fonctions de base sont laissées intestées.

Dans ce cadre (simulation en VHDL et vérification de la fonctionnalité de la simulation), nous avons elaboré ce travail qui contient 4 chapitres :

CHAPITRE 1 :

Ce chapitre traite la modèlisation des circuits logiques, il contient la description de l'environement VHDL, les différents niveaux d'abstraction , et la manière de définir une entité (entrée/sortie), l'architecture (structure interne); les fonctions et les processus; enfin des programmes pour des circuits simples (AND, OR, XOR, ADD1, BASRS, BASCJK, REGISTRE, MULTIPLEXEUR ,COMPTEUR) , avec les résultats de simulation.

CHAPITRE 2 :

Ce chapitre traite le test et la testabilité des circuits intégrés (difficulté de test et sa nécessité), les différents algorithmes utilisés pour la génération de test, la testabilité des circuits: les différents approches de mesures de la testabilité (SCOAP, COP , ...), puis la conception pour la testabilité (la conception en vue de test) .

CHAPITRE 3 :

Est une explication détaillée d'une approche hiérarchique pour la génération de test (HBTG), les différents étapes à suivre pour procéder à cette approche: (PMG, chemins sensibilisés , implémentation de l'algorithme).

CHAPITRE 4 :

Simulation d'un multiplieur des nombres entiers en se basant sur le principe de SHIFT and ADD ce qui nécessite des registres à décalage et des additionneurs, ainsi l'application de la HBTG pour générer les vecteurs de test et les appliquer au multiplieur simulé pour vérifier sa fonctionnalité.

Chapitre I

MODELISATION EN VHDL DES CIRCUITS LOGIQUES

I-1 INTRODUCTION :

VHDL est un langage conçu pour la description de materiel pour les circuits à très grande echelle d'intégration (VLSI), le mot abregé VHDL est venu de VHSIC HARDWARE DESCRIPTION LANGUAGE et VHSIC pour VERY HIGH SPEED INTEGRATED CIRCUITS. Il sert à décrire des systèmes matériels digitaux à un niveau haut d'abstraction [1].

En 1981, le département de la défense des USA VHSIC lança un programme de recherche pour la mise au point d'un langage standard et intermédiaire entre les différents HARDWARE DESIGN LANGUAGEs existants qui soit capable de concevoir des systèmes électroniques de diverses complexités.

L'histoire du développement du VHDL débuta au mois d'Août 1983, et aboutit en 1987 à la standardisation du langage par l'établissement de la norme IEEE-1076.

Avant la mise au point du VHDL plusieurs langages HDLs, existaient déjà par exemple : (IDL:interactive design language ;TIDH : texas instrument description language; CTN : computer design language etc ...), contrairement aux autres, le VHDL se caractérise par ses capacités très variées, sa forme intermédiaire et sa totale indépendance de l'environement sur lequel il est lié.

Le VHDL possède plusieurs avantages:

- D'un point de vue commercial: VHDL est un standard IEEE , c'est un standard reconnu par tous les vendeurs d'outils de CAO. De plus la position du departement de la défence amercain sur l'utilisation du VHDL est loin de laisser indifférents les industriels. Des descriptions VHDL, normalisées aussi comme standard militaire, de niveaux comportementales et structurelles sont requises obligatoirement pour les circuits intégrés fabriqués à la demande des mulitaires .

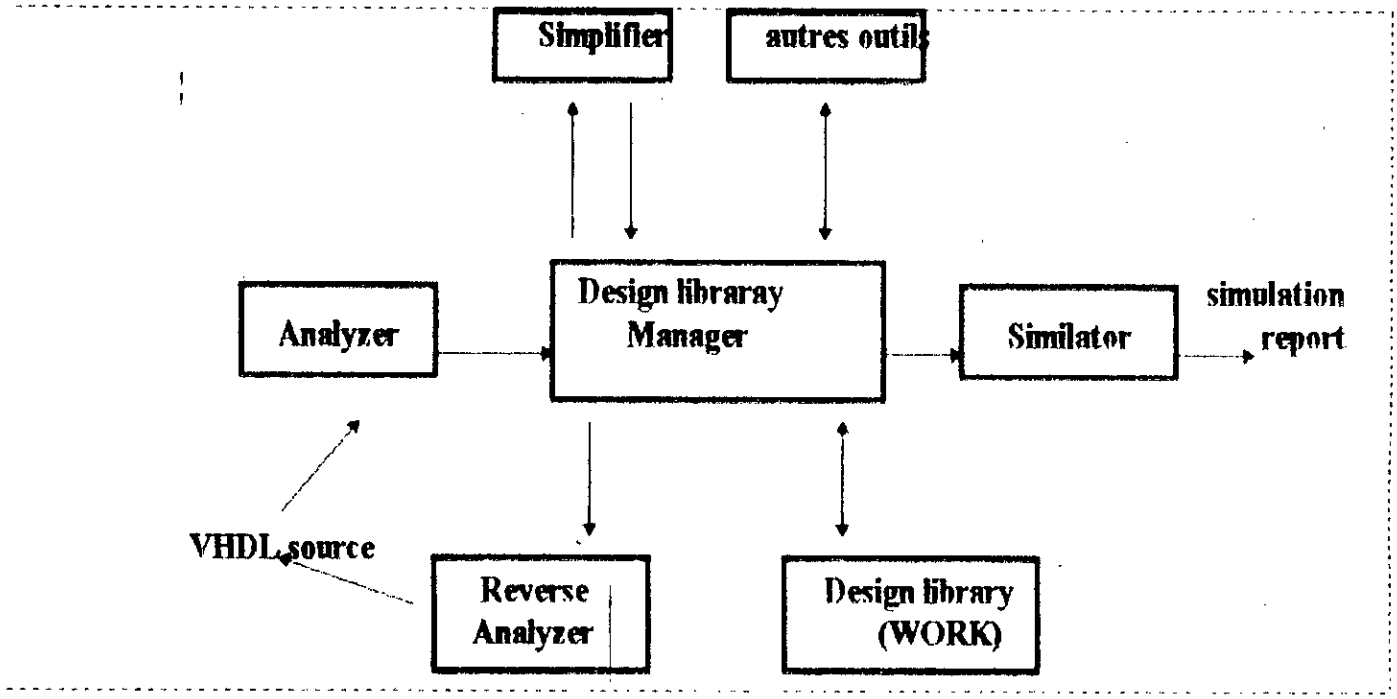
-D'un point de vue technique: Le VHDL est un langage modèrne puissant et général. Il est caracterisé par son excellente lisibilité , sa haute modularité , sa securité d'emplois et la fiabilité de sa description .

Ces caractéristiques découlent directement de notions modernes de langages de programmation, telles que: unités de compilation séparées, typage fort ou généricité.

-D'un point de vue économique: tous les langages modernes (dont le VHDL) s'efforcent de repérer le plus grand nombre d'erreurs dès la compilation, car il est important de savoir qu'une erreur trouvée dans le cycle de conception coûte moins chère.

I-2 - CONCEPTS DE VHDL

I-2-1 ENVIRONNEMENT VHDL D'INTERMETRICS:



Cet environnement proposé par la firme "INTERMETRICS" permet :

-D'écrire le comportement d'un circuit intégré dans la syntaxe VHDL à différents niveaux d'abstraction , du niveau portes au niveau comportemental .

- Le simulateur pour s'assurer d'une bonne description des circuits , il est composé de plusieurs blocs chacun d'eux joue un rôle bien spécifique .

* ANALYZER(analyseur):traduit un texte d'un format VHDL en IVAN (INTERMEDIATE VHDL ATTRIBUETED NOTATION). Il détecte les erreurs statiques (sans l'évolution du temps) .

* DESIGN LIBRARY : (bibliothèque) stocke les contextes de format IVAN et les classe.

*DESIGN LIBRARY MANAGER : (gestionnaire de la bibliothèque de la conception) c'est un logiciel utilisé comme interface entre la bibliothèque et les autres outils.

* SIMULATOR: (simulateur) il transforme le code IVAN en langage ADA ou en langage C, puis il le compile et l'execute.

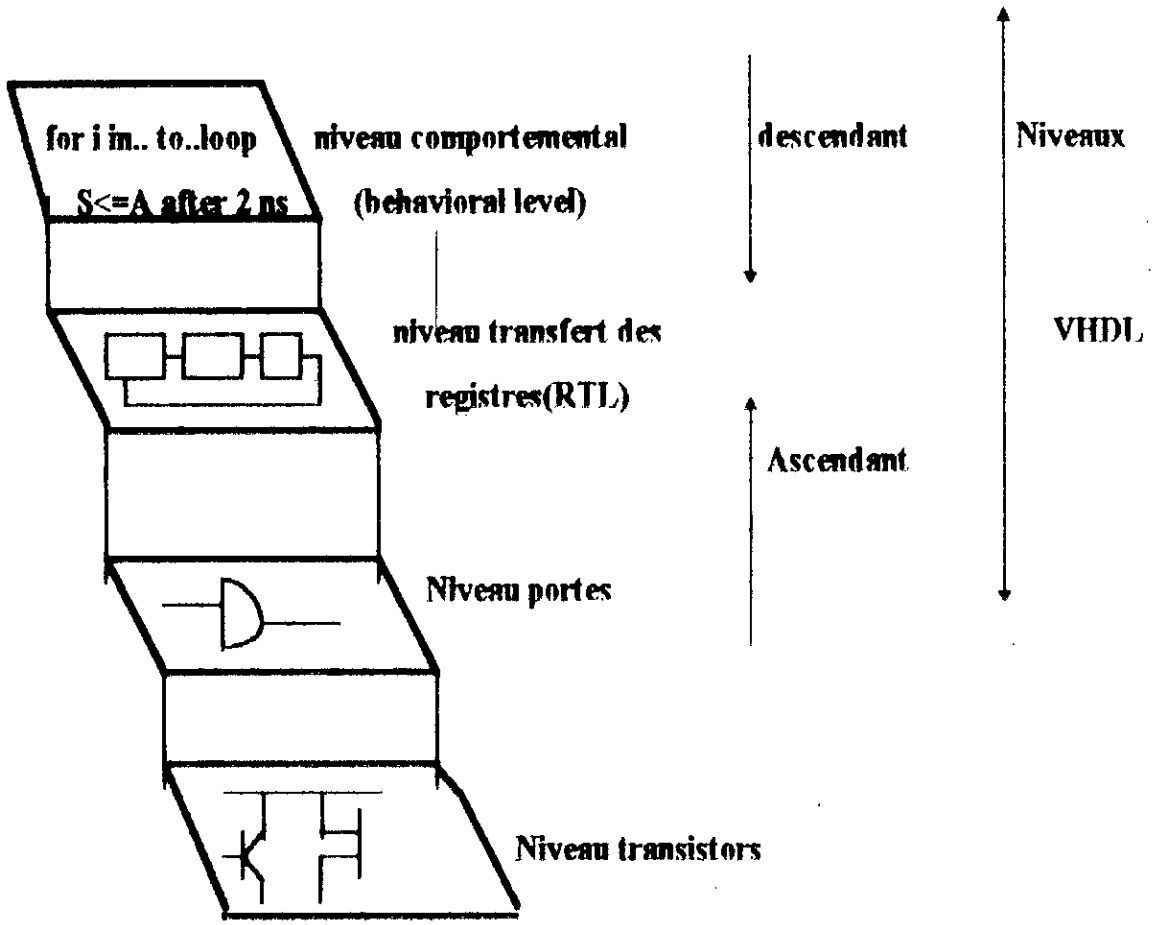
* REPORT GENERATOR (générateur de sortie): il formate les résultats

* REVERSE ANALYZER (analyseur inverseur): il traduit le code IVAN en VHDL.

* SIMPLIFIER (simplifieur): il permet de simplifier les hiérarchies.[2]

I-2-2 **LE VHDL**

Le VHDL est un langage puissant à tel point que tous les compilateurs n'implémentent pas toutes ses fonctionnalités, nous présenterons dans ce qui suit les grandes lignes qui permettent de decrire un circuit logique. La description en VHDL peut être en mode ascendant ou en mode descendant, qu'on peut la schématiser par le schéma suivant:



LES DIFFERENTS NIVEAUX D'ABSTRACTION

La description d'un circuit en VHDL revêt deux aspects :

un aspect externe :entité(ENTITY)

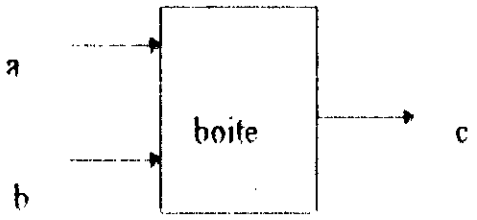
un aspect interne :ARCHITECTURE

A-L'ENTITE (ENTITY) : est la vision extèrne d'un circuit, on schématise le circuit par une boite noire ayant des signaux en entrée et d'autres en sortie, sa déclaration se fait comme suite:

```
ENTITY nom du circuit IS
 PORT (declaration des ports :TYPE de ports;... ) ;
  END nom du circuit ;
```

EXEMPLE

```
ENTITY boite IS
    PORT(a,b:IN BIT ;
        c: OUT BIT);
END boite ;
```



De plus, on peut déclarer une entité sans les signaux d'entrée /sortie qui sert a tester le fonctionnement de modeles en decrivant les signaux à l'intérieur de l'architecture

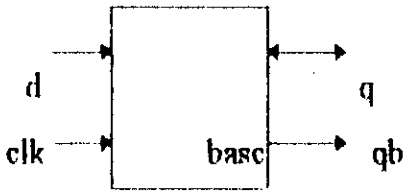
```
RNTITY test IS

END;
```

Un port d'entrée / sortie peut être de la classe

- input : IN
- output : OUT
- input output : INOUT

```
ENTITY basc IS
PORT(d,clk :INOUT BIT ;qb :OUT BIT);
END basc ;
```



on peut decrire aussi des entités paramètrées comme :

```
ENTITY générique IS
GENERIC(n:NATURAL:=10);
PORT(a,b:IN BIT_VECTOR(n-1 DOWNTO 0);
  c: OUT BIT;...);
  END générique;
```

On peut aussi écrire dans la déclaration d'entité des assertions (vérification) comme :

```
ENTITY méchant IS
  PORT(a,b :IN INTEGER;c : OUT BOOLEAN);
  BEGIN
    ASSERT{vérifier que a et b < valmax }
    REPORT"(limite depassée!)"
    SEVERITY ERROR;
  END méchant ;
```

La déclaration d'entité la plus complexe

```
  ENTITY complexe IS
   GENERIC(....) ;
   PORT(....) ;
   { déclaration de types ,de signaux ,de constantes pas de  variables } ;
   { déclaration de procédures , de fonctions }
   { corps de procédures , de fonctions }
    BEGIN
      { programme avec des instructions passives :
        c.a.d qui lisent mais ne mettent pas
        a jour les signaux }
    END complexe ;
```

B- **ARCHITECTURE** :

Ce sont les descriptions internes des circuits qui peuvent être:

- comportementales ,qui peuvent être algorithmiques ou DATAFLOW
  (un flux de données)
- structurelles
- mixtes

et cela par :

ARCHITECTURE type de descriptions OF nom de l'entité IS

{ partie déclarative }

BEGIN

{ partie descriptive } ;

END type de descriptions ;

Une description structurelle est une description, dans laquelle l'entité est décrite en termes des composants qui la constituent. Dans ce type de description il faut donc, d'une part spécifier les composants utilisés dans les circuits et d'autres part la manière dont ces composants sont connectés.

Dans la description comportementale "data flow", on tente de modéliser le circuit par un ensemble d'équations . On décrit donc chaque sortie par une équation en fonction des entées .

sortie(j) = f(entrèe(i)) f: est une fonction logique ou arithmétique .

La description algorithmique est une modélisation de haut niveau, ou le contenu de l'entité est décrit de façon algorithmique, en utilisant les structures de contrôle classiques des langages de programmation tels que ADA,PASCAL à savoir les déclarations séquentielles suivantes :

- la structure d'alternative :

IF condition THEN

instruction séquentielle ELSIF condition THEN

instruction séquentielle ELSE

instruction séquentielle ENDIF ;

- la structure d'aiguillage

CASE identificateur **IS**

WHEN choix1 => instruction sequentielle1 ;

WHEN choix2 => instruction sequentielle2 ;

WHEN others => instruction sequentielle3 ;

END CASE ;

others: designe tout ce qui ne coincide pas avec les choix exprimés auparavant .

- l'instruction d'affectation de forme grammaticale

< identificateur > < operateur-affectation > < expression > ;

ou :

Opérateur affectation est un élément de l'ensemble { := pour les variables, <= pour les signaux }

La description mixte est un mélange des deux descriptions structurelle et comportementale .

D'une manière générale la description d'une entité est de forme implantée ci-dessous :

ENTITY identificateur **IS**

entête de l'entité

{ partie déclaration :

- Déclaration de constantes

- Déclaration de signaux }

END identificateur ;

ARCHITECTURE type_description **OF** nom d'entité **IS**

Déclaration { déclaration de constantes, de signaux, de composants }

BEGIN

{ DECLARATIONS CONCURRENTES }

END type-description ;

Dans le corps de l'architecture, on declare seulement les instructions concurrentes .Pour cela les déclarations séquentielles ne peuvent figurer que dans un processus annoncé par le mot clé **PROCESS** .

```
PROCESS
 BEGIN
 { instructions séquentielles }
 END PROCESS ;
```

Reste a noter qu'un process ne travaille qu'a des instants particuliers du temps simulé entre temps il dort .Il faut explicitement dire a un process quand il doit se reveiller .

C- **FONCTIONS ET PACKAGES**

Le package est un regroupement de fonctions, de procedures et de déclaration de constantes, types, fichiers, de signaux, composants mais pas de déclaration de variables.

EXEMPLE :

```
PACKAGE digital IS
  CONST ligne : INTEGER;
  SUBTYPE : patientier IS NATURAL RANGE( 0 TO ligne);
  TYPE mefichs IS FILE OF patientier
  FILE  monfichier:mefichs IS IN"données.dat" ;
  FONCTIONS F1(s1,s2:BIT) RETURN BIT ;
  PROCEDURE maproc(VARIABLE a: INOUT BIT ,VARIABLE b: OUT BIT);


COMPONENTand_gate PORT(a,b:IN BIT; c: OUT BIT);
    END COMPONENT ;
     END PACKAGE ;
  PACKAGE BODY digital IS
   CONSTANT delais : TIME :=1 ns ;
```

```
PROCEDURE maproc(a:IN INTEGER ;b:OUT INTEGER) IS
    BEGIN
      .....
      .....
END digital;
```

On plus des packages qu'on peut definir nous même pour recuperer un effort de programmation en plusieurs endroit, il faut seulement indiquer le chemin

EXEMPLE :USE WORK.digital.ALL, il a deux packages standards pour le langage:

STANDARD et TEXTIO .

Le package STANDARD contient : la déclaration des types et des fonctions standards comme BOOLEAN;INTEGER,BIT,CHARACTER,.....

Le package TEXTIO gere les entrées / sorties, qui contient un ensemble de définitions, de fonctions READ et WRITE qui font ce type de travail, mais il faut écrire une entête pour faire reference à TEXTIO : USE STD.TEXTIO.ALL ;[3].[4]

Pour écrire et mettre en marche un programme en VHDL les étapes suivantes sont nécessaires :

** Creer un fichier nom.VHD dans un éditeur .

** Compiler ce programme par l'instruction :

VCOM nom.VHD et faire corriger les erreurs affichées.

** Declarer un fichier de données par un editeur :

ED nom.VEC par exemple(l'extension au choix) .

** La simulation par VSIM en choisissant :

* l'unité de temps de simulation (fs,ps,ns...)

* la bibliothèque de travail WORK par défaut

* l'entité désirée et sa description

(PRIMARY : nom d'entité

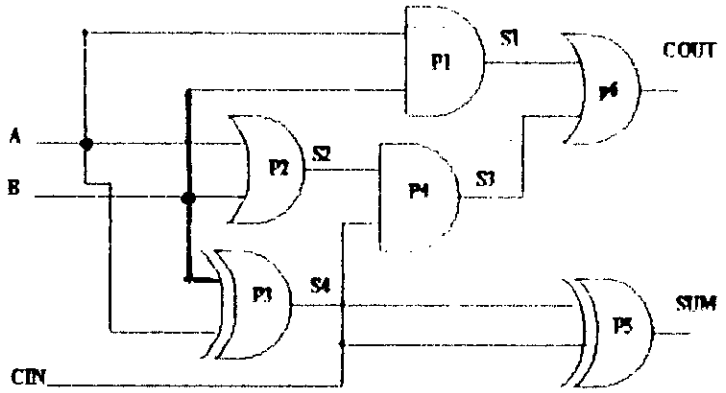
SECONDARY : type d'architecture);

* l'exécution par l'instruction des commandes VSIM: DO nom.VEC

Les résultats apparaissent dans une fenêtre appelée " LISTEFILE ". [5]

I-3 EXEMPLES

a-additionneur 1bit



ADDER 1BIT

********** PROGRAMME **********

```
--l'entité and_gate
entity and_gate is
    port(a:in bit;b:in bit;c:out bit);
    end and_gate;
architecture behavior of and_gate is
   begin
    process(a,b)
   begin
   c <= a and b after 3 ns;
   end process;
    end behavior;

--l'entité or_gate
entity  or_gate is
    port(a:in bit;b:in bit;c:out bit);
    end  or_gate;
```

```
architecture behavior of or_gate is
    begin
    process(a,b)
    begin
    c <= a or b after 3 ns;
    end process;
    end behavior;
--l'entité xor_gate
entity x_or1 is
    port (a,b:in bit;c:out bit);
    end x_or1;
    architecture behavior of x_or1 is
     begin
         process(a,b)
     begin
         if a = b then
       c <= '0' after 3 ns;
     else c <= '1' after 3 ns;
     end if;
      end process;
      end behavior;
entity additi1 is
     port(a,b,cin:in bit;cout,sum:out bit);
     end additi1;
architecture struct of additi1 is
     component andgate port(a,b:in bit;c:out bit);end   component;
     component orgate port(a,b:in bit;c:out bit);  end compenonent;
     component xorgate port(a,b:in bit;c:out bit);   endcomponent;
    signals1,s2,s3,s4:bit;
     for all:andgate use entity work.and_gate(behavior);
     for all:orgate use entity work.or_gate(behavior);
     for all:xorgate use entity work.x_or1(behavior);
```

```
begin
    p1:andgate port map(a,b,s1);
    p2:orgate port map(a,b,s2);
    p3:xorgate port map(a,b,s4);
    p4:andgate port map(s2,cin,s3);
    p5:xorgate port map(s4,cin,sum);
    p6:orgate port map(s1,s3,cout);
end struct;
```

*******fichier de données*******

```
force cin 0 10,0 20,0 30,0 40,1 50,1 60,1 70,1 80
force a 0 10,0 20,1 30,1 40,0 50,0 60,1 70,1 80
force b 0 10,1 20,0 30,1 40,0 50,1 60,0 70,1 80
list cin a b s1 s2 s4 s3 sum  cout
run
```

********fichier de resultats*******

ns	cin	a	b	s1	s2	s4	s3	sum	cout
0	0	0	0	0	0	0	0	0	0
20	0	0	1	0	0	0	0	0	0
23	0	0	1	0	1	1	0	0	0
26	0	0	1	0	1	1	0	1	0
30	0	1	0	0	1	1	0	1	0
40	0	1	1	0	1	1	0	1	0
43	0	1	1	1	1	0	0	1	0
46	0	1	1	1	1	0	0	0	1
50	1	0	0	1	1	0	0	0	1
53	1	0	0	0	0	0	1	1	1
56	1	0	0	0	0	0	0	1	1
59	1	0	0	0	0	0	0	1	0
60	1	0	1	0	0	0	0	1	0
63	1	0	1	0	1	1	0	1	0
66	1	0	1	0	1	1	1	0	0
69	1	0	1	0	1	1	1	0	1
86	1	1	1	1	1	0	1	1	1
70	1	1	0	0	1	1	1	0	1
80	1	1	1	0	1	1	1	0	1
83	1	1	1	1	1	0	1	0	1

b- **multiplixeur 8 ——→ 1**

```
****programme******
entity mux is
port(d0,d1,d2,d3,d4,d5,d6,d7:in bit;
   a,b,c,g :in bit ; y, w: out bit);
    end mux;
    architecture behavior of mux is
     signal s:bit_vector(3 downto 1);
     begin
      s(1) <= a;s(2) <= b;s(3) <= c;
      process(s)
       begin
         if(g='1') then          y <= '0' ;          w <= '1' ;
       else      case s is
          when"000" => y <= d0 ; w <= not d0;
          when"001" => y <= d1 ; w <= not d1;
          when"010" => y <= d2 ; w <= not d2;
          when"011" => y <= d3 ; w <= not d3;
          when"100" => y <= d4 ; w <= not d4;
          when"101" => y <= d5 ; w <= not d5;
          when"110" => y <= d6 ; w <= not d6;
          when"111" => y <= d7 ; w <= not d7;
         end case;
      end if ;      end process ;
    end behavior;
*************fichier de resultats*****************
```

ns	g	c	b	a	d0	d1	d2	d3	d4	d5	d6	d7	y	w
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	1	0	1
10	0	0	0	1	0	1	0	0	1	0	1	0	0	1
10	0	0	0	1	0	1	0	0	1	0	1	0	1	0
20	0	0	1	0	0	0	0	0	1	0	1	1	1	0
20	0	0	1	0	0	0	0	0	1	0	1	1	0	1
30	0	0	1	1	0	1	1	1	1	0	1	0	0	1
30	0	0	1	1	0	1	1	1	1	0	1	0	1	0
40	0	1	0	0	1	0	0	1	1	1	1	0	1	0
50	0	1	0	1	1	1	0	1	1	0	1	0	1	0
50	0	1	0	1	1	1	0	1	1	0	1	0	0	1
60	0	1	1	0	1	0	1	1	1	0	0	0	0	1
70	0	1	1	1	1	1	1	1	1	0	0	1	0	1
70	0	1	1	1	1	1	1	1	1	0	0	1	1	0

c- **bascule RS**

*******programme********

```
package xbit is
type mon_xbit is('1','0','x','z');
end xbit;
use work.xbit.all;
entity rs is
port(r,s:in bit;q:out mon_xbit);
end rs;
architecture behavior of rs is
 signal t:bit_vector (1 to 2);
 begin
 t(1) <= r;t(2) <= s;
  process(t)
  begin
  case t is
  when"00" => q <='0'after 0 ns;
  when"01" => q <='1'after 5 ns;
  when"10" => q <='0'after 3 ns;
  when"11" => q <='x'after 5 ns;
  end case; end process; end behavior;
```

**********resultats**********

ns	r	s	t	q
0	0	0	00	1
0	0	0	00	0
10	1	0	00	0
10	1	0	10	0
20	0	1	10	0
20	0	1	01	0
25	0	1	01	1
30	1	0	01	1
30	1	0	10	1
33	1	0	10	0
40	1	1	10	0
40	1	1	11	0
45	1	1	11	x
50	0	1	11	x
50	0	1	01	x
55	0	1	01	1

d- **bascule JK**

*******programme************

```
entity bascule_jk is
 port(j,k,clk:in bit;q: inout bit);
end bascule_jk ;
architecture behavior of bascule_jk is
signal s : bit_vector(1 to 2);
 begin
 s(1) <= j; s(2) <= k;
process
 begin
wait until not clk'stable and clk = '1' ;
  case s is
    when "00" => q <= q   ;
    when "01" => q <= '0'   ;
    when "10" => q <= '1'   ;
    when "11" => q <= not(q)  ;
  end case ;
end process;
end behavior;
```

*********resultats**********

ns	clk	j	k	q
0	0	0	0	0
15	1	0	0	0
30	0	0	0	0
45	1	0	0	0
50	1	0	1	0
60	0	0	1	0
75	1	0	1	0
90	0	0	1	0
100	0	1	0	0
105	1	1	0	0
105	1	1	0	1
120	0	1	0	1
135	1	1	0	1
150	1	1	1	1
150	0	1	1	1
165	1	1	1	1
165	1	1	1	0

e-**Registre à 8 bits**

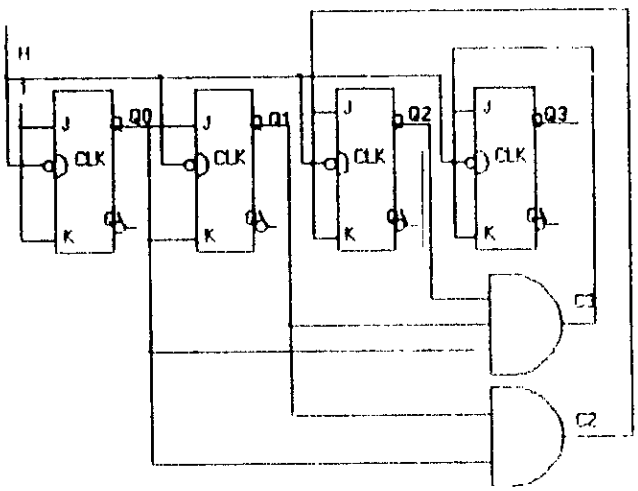
```
package zbit is
 type unresol_zbit is ('1','0','x','z');
 type unresol_zbit_vector is array( natural range <>) of unresol_zbit;
end zbit;
use work.zbit.all;
entity reg8 is
port(do:out unresol_zbit_vector(0 to 7);
   nds2,ds1:in bit;di:in unresol_zbit_vector(0 to 7);
   strb:in bit);
   end reg8;
   architecture behavior of reg8 is
   signal enbl:bit; signal s:unresol_zbit_vector(0 to 7);
   signal reg:unresol_zbit_vector(0 to 7);
   begin
  output_4:process(enbl,reg)
     begin
       if( enbl='1')then
       s <= reg;
       do <= s;
       else
       do<="zzzzzzzz";
       end if;
       end process output_4;
   enable_9:process(nds2,ds1)
       begin
          enbl<=ds1 and not nds2;
    end process enable_9;

  preg_14: process(strb)
       begin
        if(strb='1')
        then reg<=di;
       end if;
   end process preg_14;
       end behavior ;
```

**** **** resultats**********

ns	ds1	nds2	enbl	strb	di	do
0	0	0	0	0	11111111	11111111
0	1	0	0	1	00000000	zzzzzzzz
0	1	0	1	1	00000000	zzzzzzzz
0	1	0	1	1	00000000	11111111
10	0	0	1	1	11111111	11111111
10	0	0	0	1	11111111	11111111
10	0	0	0	1	11111111	zzzzzzzz
20	1	0	0	0	11110000	zzzzzzzz
20	1	0	1	0	11110000	zzzzzzzz
20	1	0	1	0	11110000	00000000
30	1	1	1	1	00001111	00000000
30	1	1	0	1	00001111	00000000
30	1	1	0	1	00001111	zzzzzzzz

f- compteur synchrone modulo 16



COMPTEUR SYNCHRONE A 4 ETAGES

```
****programme******
entity compteur is
port ( h : in bit ;
     q0,q1,q2,q3 : inout bit );
end compteur ;
architecture structurelle of compteur is
component bascjkgate port (j,k,clk : in bit; q : inout bit); end component;
component andgate port ( a,b :in bit;c: out bit );end component ;
component and3gate port (a,b,c :in bit;d: out bit); end component ;
signal c2 , c3 ,j1 , k1 : bit ;
signal s : bit_vector(1 to 4);
for all : andgate use entity work.and_gate(behavior) ;
for all : and3gate use entity work.and3_gate(behavior) ;
for all : bascjkgate use entity work.bascule_jk(behavior) ;
begin
   j1 <= '1' ; k1 <= '1' ;
   s(1) <= q3; s(2) <= q2 ; s(3) <= q1 ; s(4) <= q0 ;
e1 : bascjkgate port map (j1 ,k1,h,q0 ) ;
e2 : bascjkgate port map ( q0,q0,h,q1) ;
e3 : bascjkgate port map ( c2,c2,h,q2) ;
e4 : bascjkgate port map ( c3,c3,h,q3) ;
e5 : andgate port map ( q0,q1,c2 ) ;
e6 : and3gate port map ( q0,q1,q2,c3 ) ;
end structurelle ;
```

********Resultats*********

```
  ns h  q3 q2 q1 q0
   0 0  0  0  0  0
  15 1  0  0  0  0
  15 1  0  0  0  1
  15 1  0  0  0  1
  30 0  0  0  0  1
  45 1  0  0  0  1
  45 1  0  0  1  0
  45 1  0  0  1  0
  60 0  0  0  1  0
  75 1  0  0  1  0
  75 1  0  0  1  1
  75 1  0  0  1  1
  90 0  0  0  1  1
 105 1  0  0  1  1
 105 1  0  1  0  0
 105 1  0  1  0  0
 120 0  0  1  0  0
 135 1  0  1  0  0
```

135 1 0 1 0 1 4
135 1 0 1 0 1 5
150 0 0 1 0 1 5
165 1 0 1 0 1 5
165 1 0 1 1 0 5
165 1 0 1 1 0 6
180 0 0 1 1 0 6
195 1 0 1 1 0 6
195 1 0 1 1 1 6
195 1 0 1 1 1 7
210 0 0 1 1 1 7
225 1 0 1 1 1 7
225 1 1 0 0 0 7
225 1 1 0 0 0 8
240 0 1 0 0 0 8
255 1 1 0 0 0 8
255 1 1 0 0 1 8
255 1 1 0 0 1 9
270 0 1 0 0 1 9
285 1 1 0 0 1 9
285 1 1 0 1 0 9
285 1 1 0 1 0 10
300 0 1 0 1 0 10
315 1 1 0 1 0 10
315 1 1 0 1 1 10
315 1 1 0 1 1 11
330 0 1 0 1 1 11
345 1 1 0 1 1 11
345 1 1 1 0 0 11
345 1 1 1 0 0 12
360 0 1 1 0 0 12
375 1 1 1 0 0 12
375 1 1 1 0 1 12
375 1 1 1 0 1 13
390 0 1 1 0 1 13
405 1 1 1 0 1 13
405 1 1 1 1 0 13
405 1 1 1 1 0 14
420 0 1 1 1 0 14
435 1 1 1 1 0 14
435 1 1 1 1 1 14
435 1 1 1 1 1 15
450 0 1 1 1 1 15
465 1 1 1 1 1 15
465 1 0 0 0 0 15
465 1 0 0 0 0 0

CHAPITRE II

Test et Testabilité des Circuits Intégrés

II-1 INTRODUCTION

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 des 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 systèmes, ce problème doit être pris en charge dés la conception des circuits intégrés et des systèmes. Déjà imparfaitement maîtrisé au stade des circuits LSI, il constitue pour l'avenir un défi redoutable à cause de l'impossibilité de proceder à un test exhaustif le plus effecace possible et dans une durée justifie l'exécution du test à la fréquence la plus élevée possible compatible avec le produit; donc c'est un critère économique. Cela montre d'une façon impérative la nécessité de trouver des formules nouvelles de coexistance entre le fabricant des circuits intégrés, utilisateurs des circuits intégrés, fabricants d'équipements de test et chercheurs dans le domaine du test.

Dans ce cadre ont été développé plusieurs méthodes et outils de test permettant de satisfaire les besoins des producteurs et des clients.

II-2 GENERALITES

II-2-1 MESURES DE LA CONFIANCE DANS UN TEST

Quand un ensemble de circuits est testé le résultat peut dépendre de nombreux paramètres la séquence de test, l'évaluateur de test (observation directe de la réponse du circuit ou analyse de signature), des fautes qui peuvent affecter le circuit, du rendement de production, de la fiabilité de l'équipement de test, etc ... .

Appelons W le nombre de circuits identiques que le dispositif de test doit ou devra tester. Parmi ces W circuits il y'on a G qui sont correctes et D qui sont défectueux . Le dispositif de test a pour objectif de séparer les G circuits qui sont bons des autres.

Naturellement cette séparation n'est pas parfaite on suppose que tous les circuits bons sont reconnus bons, ce qui généralement le cas en pratique. Parmi les D circuits défectueux il y'on a Df qui seront refusés parce qu'on aura un mauvais fonctionnement et Dp qui passerons le test sans détection d'erreur.

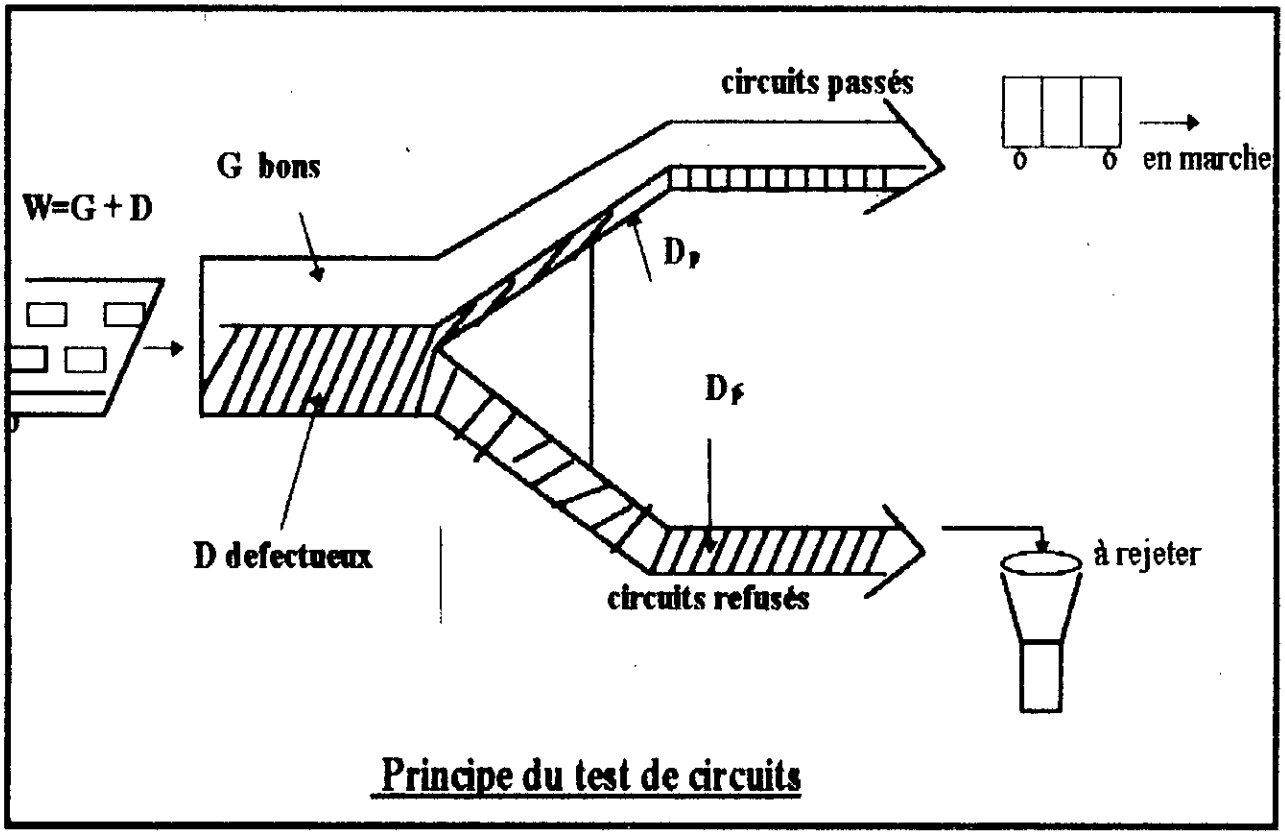
On a donc: $W = D + G$ et $D = Dp + Df$.

A l'essai du test on a $G + Dp$ circuits qui ont été testés et considérés comme bons pour l'utilisateur. Le résultat du test est d'autant meilleur que le nombre **Dp** est réduit . On suppose que chaque circuit parmi les D est affecté d'une faute de l'ensemble dit ensemble de fautes prescrit: $F = \{ f1,f2,f3,...,fm \}$

Les mesures de confiance sont applicables quelque soit la séquence de test, quelle soit aléatoire, pseudo-aléatoire, ou déterministe. La première mesure est la probabilité de couverture complète : $Pc = Pr[Dp = 0]$ et la probabilité d'échapper [escape probability]: $Pesc = 1 - Pc$.

Dans le cas général on considérera la couverture de fautes espérée, notée **Pe** comme la moyenne arithmétiques du nombre de fautes testées .

La probabilité minimale de test, notée **Pm**,est la probabilité minimale de détecter un circuit Défectueux sachant qu'il est défectueux. c'est donc la probabilité de détecter la faute la plus difficile à tester dans l'ensemble **F** . La couverture des circuits défectueux, notée **Pa**, est la probabilité qu'un circuit défectueux soit détecté . Les quatre mesures qui ont été citées permettent de caractériser la confiance que l'on peut avoir dans la méthode de test. [6]



Principe du test de circuits

II-2-2 NECESSITE ECONOMIQUE DU TEST

Plusieurs coûts sont à considérer lors du choix du test:

- Le coût de l'équipement de test et de personnel nécessaire pour appliquer les procédures de test
- Le coût de l'équipement complémentaire spécifique pour chaque circuit à tester afin de faciliter le test.
- Le coût équivalent au temps passé pour mettre au point les séquences de test et pour analyser les réponses obtenues.
- Les coûts de conception et de développements eventuels nécessaires à rendre le circuit plus facilement testable c.à.d pour améliorer la commandabilité et l'observabilité du circuit. (voir section testabilité)

Il a été prouvé économiquement, qu'il est plus avantageux de détecter les défaillances le plutôt possible. En effet, si la détection d'une défaillance au niveau puce induit un coût de 1U (unité), la découvrir une fois le circuit mis sous boitier induit un coût de 10U, ce coût est de 100U, si la défaillance est détectée au niveau de la carte imprimée, de 1000U à l'étape de l'intégration du système, et de 10000U si le système s'avère défaillant chez le client.[7]

II-3 VERIFICATION ET TEST DES CIRCUITS

Les phases de vérification survenant au cours des étapes de conception, de fabrication et de la vie d'un composant peuvent être résumées par le tableau suivant :[8]

phase de vérification	fin de conception	fin de fabrication	maintenance
type de défaillance	erreur de conception	défaut de fabrication	erreurde conception Résiduelle
dossiers disponibles	dossiers de conception	dossiers techniques	manuel usager
Equipement de test	élaboré, simulation	élaboré	variable
temps	plusieurs mois	Très court	variable

II-3-0 VERIFICATION DE LA CONCEPTION

Vu l'importance de la phase de conception concernant la fonctionnalité 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 dans cette phase que dans une phase ultérieure.

Avec le développement des outils de CAO, il est devenu plus facile de concevoir des systèmes même a très large échelle, ainsi de dimimuer les coûts résultants des erreurs de conception classique.

II-3-1 MODELLES DE PANNES:

Il est nécessaire de faire des distinctions entre une défectuosité, un défaut et une erreur .

* Un défaut: dans un composant est une imperfection physique qui peut entrainer un mauvais fonctionnement du composant.

* Une panne: est la manifestation logique d'un défaut (coller à zéro ou a un d'une connexion...);elle peut être permanente ou intermittente.

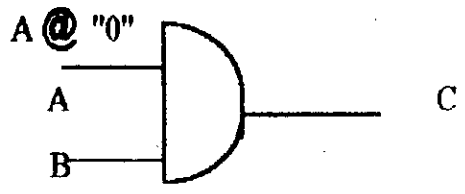
* une erreur: est un mauvais fonctionnement du composant dû a une panne.

Etant donnée la diversité des types de défectuosités possibles il serait avantageux de mettre au point un modèle logique qui les engloberait toutes. Ainsi, la production des vecteurs de test, pour l'ensemble des défectuosités possibles, constitue une tâche ardue, par contre, un modèle logique adéquat permetterait de produire des vecteurs de test sans connaitre les détails qui peuvent varier, selon les technologies.

a- MODELE BLOQUE CLASSIQUE (STUCK-AT)

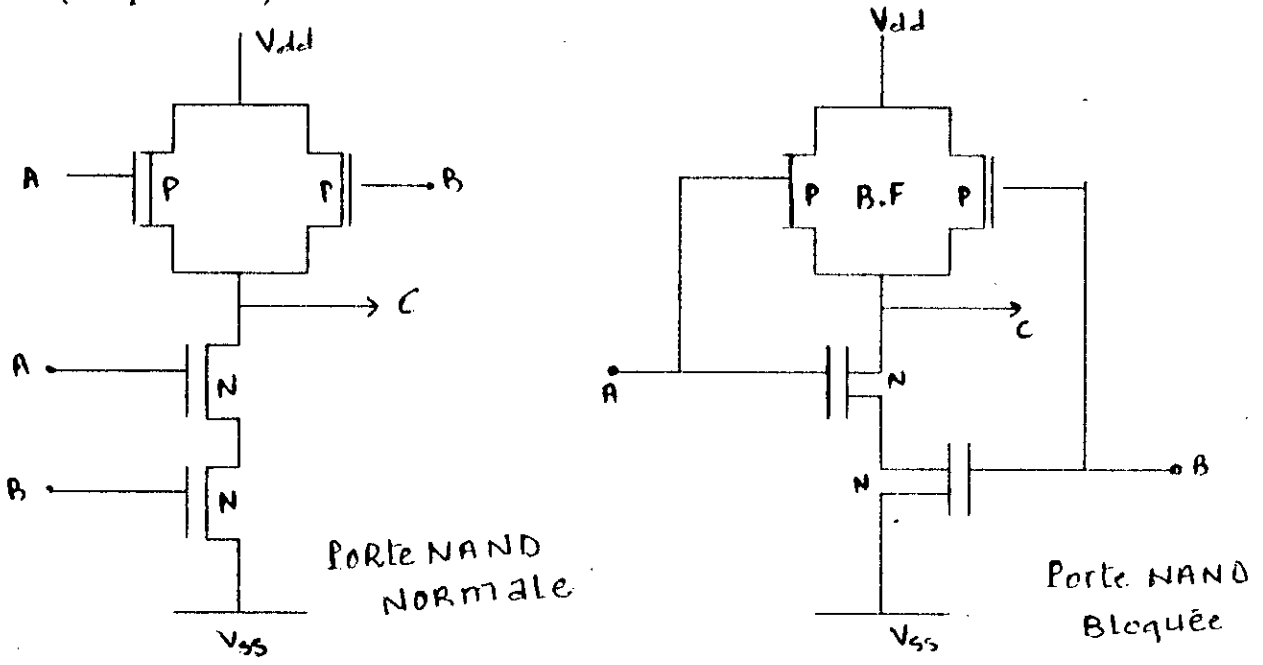
C'est le modèle de défaut logique, le plus utilisé. Avec ce modèle, on suppose que toutes les défectuosités se manifestent logiquement comme une entrée ou comme une sortie bloquée soit à '0' soit à '1'.

Dans la porte AND illustrée ci -contre , A est bloquée à '0' , la sortie C vaut '0' même pour AB = 11 ,qui constitue, donc un vecteur de test de la panne.



b- **MODELE BLOQUE OUVERT-BLOQUE FERME**

Ce modèle est particulièrement utile, lorsque la structure logique réalisée est l'une des différentes formes de logique, qu'il est possible d'obtenir avec des transistors MOS. Ce modèle considère que chacun des transistors peut être bloqué soit en conduction (bloqué fermé), soit en mode inactif (bloqué ouvert).



PORTE NAND ELEMENTAIRE AVEC UN TRANSISTOR BLOQUE-FERME(B-F)

c- **MODELE COURCIRCUIT(BRIDGING FAULTS)**:

L'apparition des courtcircuits entre deux structures dans une même couche ou en couches différentes est très fréquente. Très souvent, un courtcircuit provoque un OU-cablé ou un ET-cablé

II-3-2 SIMULATION DE PANNES

La simulation de pannes est une partie importante dans le processus de génération de test. Elle sert à vérifier la validité du test généré, à réduire le nombre de vecteurs de test et à calculer le taux de couverture de pannes de la séquence de test . Elle est aussi utilisée pour la construction de dictionnaires des signatures, destinés à détecter les défauts logiques, et donner la liste des défauts non détectés. Il existe trois principales méthodes de simulation de pannes.

a- **LA SIMULATION DE PANNES SERIE** :

Cette méthode effectue autant de simulation qu'il y'a de pannes contenues dans le circuit. Si pour un circuit ayant N défauts de collage, si on ne considère pas le cas de pannes multiples. Il faudrait faire 2N simulations, car il y'a 2 types de collage à '0' et à '1'. Cette algorithme n'est pas très utilisé, car nécessitant des temps de calcul prohibitifs.

b-**LA SIMULATION DE PANNES PARALLELE** :

Avec cette méthode un certain nombre de copies du circuit sont simulées simultanément. L'une d'elles, est le circuit correct, chacune des autres copies est un circuit avec un défaut particulier.

Cet algorithme est plus rapide que celui de la simulation serie; puisque plusieurs pannes sont simulées en même temps. Néanmoins, pour cette méthode, la simulation doit continuer jusqu'à ce que chaque panne simulée en parallèle soit détectée. Un autre inconvénient de cette méthode est de ne pas prendre en compte les délais de propagation.[9]

c- **LA SIMULATION DE PANNES CONCURRENTE**

Elle est de loin la plus puissante des trois méthodes; basée elle aussi, sur le concept de "GOOD and FAULTY MACHINE". La bonne machine sera le circuit sans pannes. Les mauvaises machines seront les parties du circuit qui ne se comporte pas comme le bon circuit, et ce sous l'effet d'une panne. Ainsi pour la simulation, il y'aura autant de machines avec pannes que de pannes à insérer. On commence par simuler le bon circuit et quand les pannes insérées font diverger les résultats, on copie les circuits. L'algorithme concurrent est difficilement implémentable, par contre il est plus rapide et nécessite moins d'espace que les deux algorithmes précédents.[9]

II-4 GENERATION DE VECTEURS DE TEST

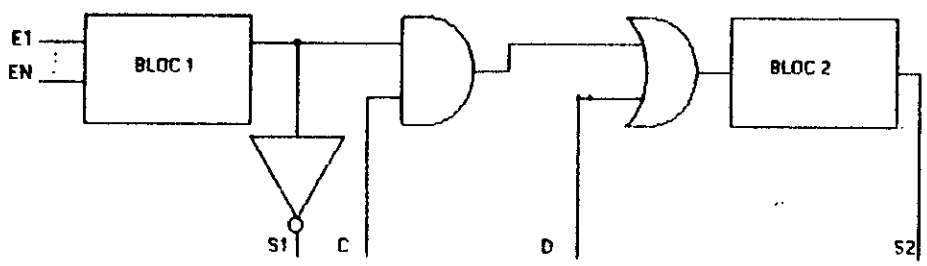
La génération de vecteurs de test est le processus qui détermine l'ensemble des vecteurs qui serviront de stimuli pour le circuit dans le but de vérifier son bon fonctionnement, et doit satisfaire au moins aux critères suivants:

- 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.

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 vérifiés pour un test exhaustif de la mémoire, le réseau séquentiel et combinatoire combiné qui à N entrées binaires 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 extrêment 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 (cas 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).Il existe plusieurs approches pour la génération de vecteurs de test. L'essentiel de ces approches sont :

II-4-1 APPROCHE PSEUDO-EXHAUSTIVE :

Au 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 aucours 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



L'entrée C est utilisée pour empecher la sortie du module 1 d'arriver au module 2, permettant aussi le controle du module 2 par D; le module 2 ne dépendera désormais que d'une seule entrée au lieu des N entrées du module 1. Cette approche présente un taux de couverture de pannes exellent,mais cependant elle souffre de certains inconvénients majeurs dont, la difficulté d'automatiser le processus de partionnement du circuit et le choix des portes supplémentaires.[7]

II-4-2 **APPROCHE DETERMINISTE** :

Cette technique résout le problème de la technique précédente par la génération algorithmique du nombre exact de vecteurs nécessaires pour la détection de toutes les pannes testables dans le circuit.

a-PRINCIPE DE LA METHODE :

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 declarée non testable. La deuxième étape, consiste à simuler le circuit pour détecter toutes les pannes qui peuvent l'être avec le même vecteur généré.

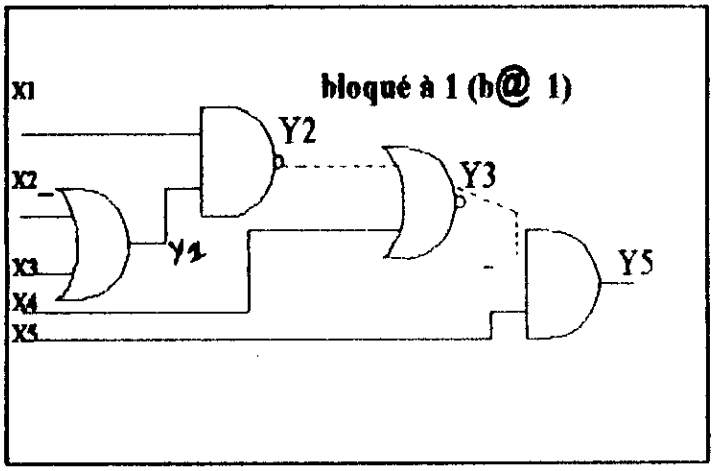
b- ALGORITHME D :

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 au 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 très loin en arrière. Ainsi l'algorithme D et particulièrement inefficace pour certaines classes de circuits : Citant par exemple les circuits détecteurs d'erreurs pour lesquels un temps prohibitif est consommé ou alors la couverture de panne est médiocre si une limite de temps est fixée .

EXEMPLE :

Soit le réseau combinatoire suivant:



supposant que la ligne Y2 est bloquée a '1',

pour faire propager cette panne on attribue

la valeur logique '0' a Y2 (valeur opposée)

et on force X4 a '0' pour avoir Y3 a '1' , pour avoir

Y5 = '1' X5 doit être a '1'. Pour justifier notre

choix en revenant aux entrées primaires, en

Commençant le retour en arrière à partir

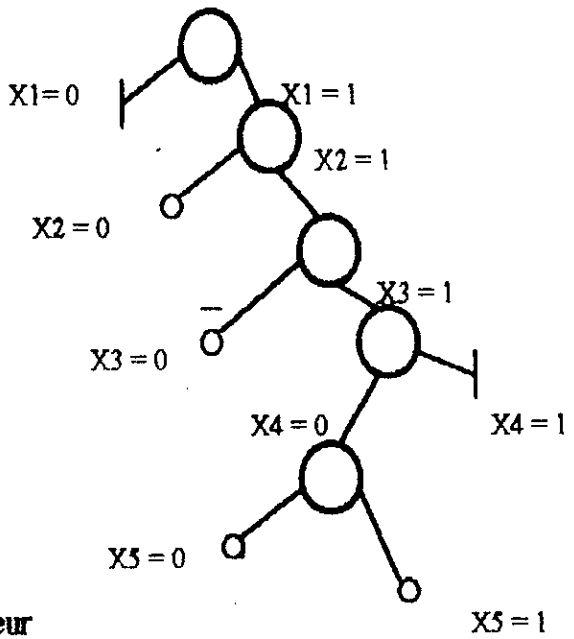
de Y2 (Y2 = '0' => Y1 = '1' et X1 = '1')

ainsi pour que Y1 = '1' => X2 = '1' et X3 = 'X',

ou X2 = 'X' et X3 = '1' (X soit '0',soit '1');donc

le vecteur X5 X4 X3 X2 X1 = 1 0 X 1 1 est un vecteur

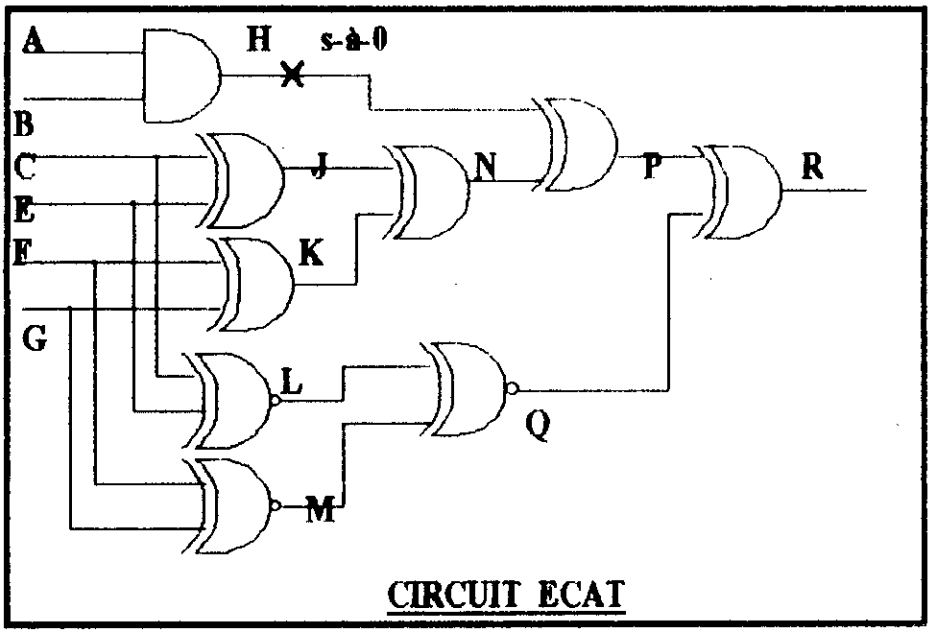
pour tester la panne "stuck at '1' de la ligne Y2) . [7]



c- 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 y'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 alors qu'il est 30 fois plus rapide que les détecteurs d'erreurs.

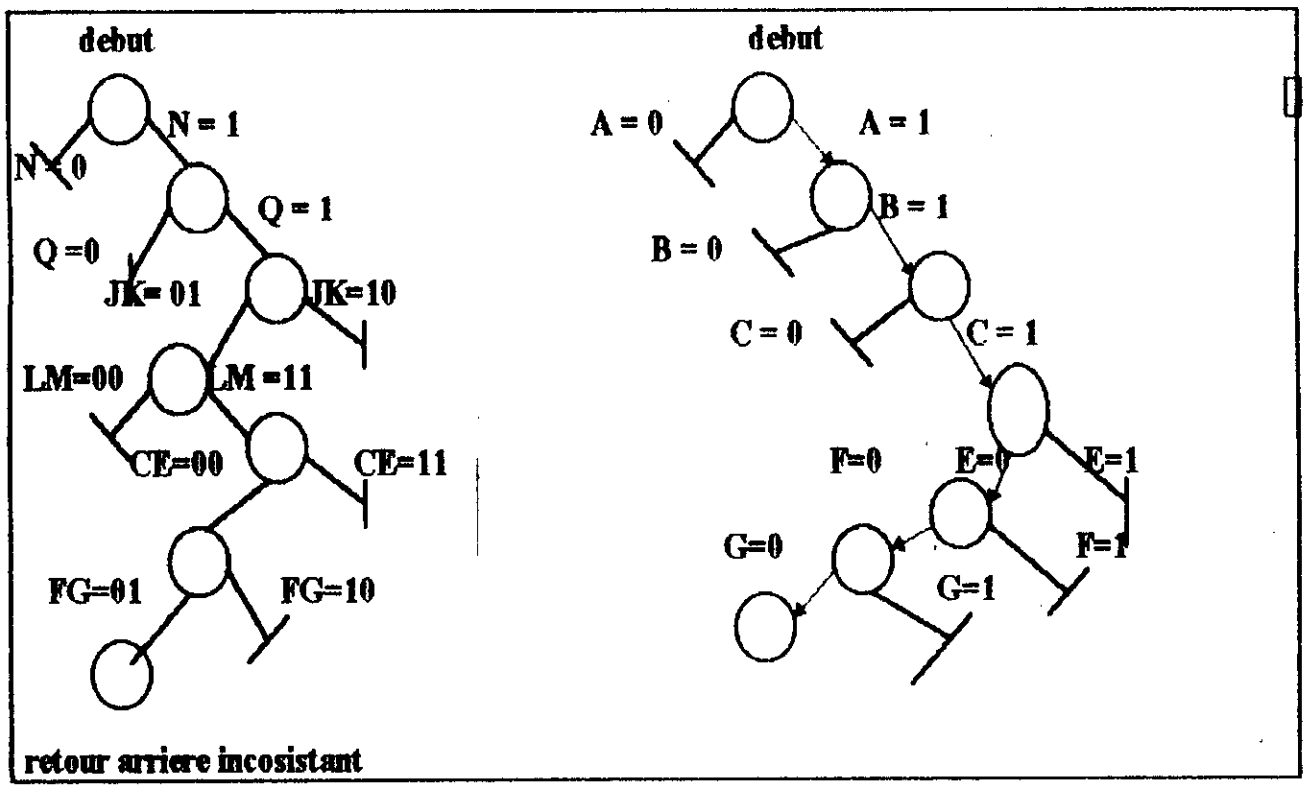
EXEMPLE:



CIRCUIT ECAT

On considère la figure ci-dessous, qui montre un circuit translateur et correcteur d'erreur (ECAT).Pour la faute H bloqué à '0' (H s-a-0) supposant qu'on assigne A=1. Nous detèrminons l'implication pour toutes les entrées primaires. depuis A = 1 ne cause pas d'implication, donc on choisit B = 1 ce qui => H = D, pour propager la valeur D de H à P nous assignons C = 1, E = 0 => J = 1 et L = 0. Pour F = 0 et G = 0 => K =0, M = 1, N = 1, Q = 0. De plus, H = D et N = 1 => P = $\overline{D}$ avec Q = 0 => R = $\overline{D}$:

est une test complète pour la faute bloquée à '0' . [10]



(ALGORITHME D\) (PODEM\)

d- **FAN(FANOUT -ORIENTED ALGORITHME)**

Pour minimiser le temps de CPU(central processing unit) consommé et les retours arrières, cette technique essaie de découvrir le plutôt possible l'apparition des contradictions ou la non-existence de solutions. Son principe est qu'on assigne le nombre maximal possible de valeurs au début pour minimiser le choix arbitraire qui peut être erroné. Si elle s'avère erronée elle sera changée par son opposée. Ainsi aucun temps ne sera consommé inutilement. Grace à la mesure de testabilité, une version modifiée de FAN a été utilisée. Cette dernière guide le choix aléatoire dans la version originale . Grace à de telle stratégie, FAN a pour vu qu'il pouvait être beaucoup plus rapide que la technique PODEM et plus précis.

e- **L'ALGORITHME HBTG (HIERARCHICAL BEHAVIORAL TEST GENERATOR)**

L'algorithme HBTG :générateur de test comportemental hiérarchique qui est une version modifiée de l'algorithme D, il est développé pour développer systématiquement les tests pour les modèles comportementaux VHDL [11]. En généal il est basé sur deux étapes principales :

- La première est de construire les chemins sensibles du module VHDL en se basant sur le modèle graphique du processus (PMG:PROCESS MODELE GRAPHICAL), ce modèle qui est une modèlisation graphique du circuit.
- La deuxième est de générer les vecteurs de test pour les modules individuels et les stocker dans des fichiers, ces modules interconnectent entre eux pour former l'entité entière du circuit. En se basant sur certains critères durant la génération. L'algorithme HBTG utilise ces deux bases de données pour générer les vecteurs de test pour l'entité globale. Cet algorithme sera examiné plus en détail dans le chapitre III et appliqué pour notre circuit spécifié (MULTIPLIEUR).

II-4-3 L'APPROCHE PSEUDO-ALEATOIRE

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 à l'avance et doit être déterminé de telle sorte qu'il permette d'atteindre la couverture de pannes désirée. Grace à des méthodes heuristiques de calcul de testabilité, on peut prédire adéquatement de la couverture de pannes. La technique conventionnelle pour générer des vecteurs pseudo-aléatoires et l'utilisation d'un registre à décalage a rétro-action linéaire(LFSR:linear feed back shift register).[7]

II-4-4 TESTS FONCTIONNELS:

La complexité de génération de vecteurs de test associés aux méthodes algorithmiques, basées sur une description structurelle,et le fait que souvent on doit générer la séquence de vecteurs de test, sans la connaissance des détails d'implémentation du circuit au niveau portes, on sait que certains concepteurs optent pour les tests fonctionnels.

En effet parfois la seule information disponible est le catalogue des circuits intégrés qui donne les différents modes de fonctionnement et décrit l'architecture générale du circuit. Le concepteur peut ainsi partionner l'unité sous test en plusieurs modèles tels que registres, multiplexeurs, UALs, ROMs, RAMs. Chaque module peut être traité comme boite noire réalisant une entée/sortie spécifique. Ces modules peuvent être testés pour des défaillances fonctionnelles. La considération explicite des pannes affectant les lignes n'est pas nécessaire.

Pour générer des séquences de test raisonnables, à moindre coût et avoir une bonne couverture de pannes(>90/100) le concepteur doit:

1- Vérifier qu'un bloc fonctionnel ne réalise pas des fonctions supplémentaires en plus de celles pour lesquelles il a été conçu.

2- Vérifier que des informations structurelles, en plus des informations fonctionnelles permetteront de générer des séquences de test courtes et efficaces.[12]

II-5 TESTABILITE

La testabilité est une notion très difficile à définir au sens propre du terme, la testabilité doit traduire la facilité de test que présentera un circuit intégré. Or les différents types de test, les phases d'applications du test, les différents équipements de test, ne conduisent pas aux même appréciations de testabilité. En pratique les études de testabilité ne sont faites très souvent à la lueur d'une expérience de test précise. Une méthode de test ayant été choisie et analysée pour un modèle de pannes donnée, on cherche alors à mesurer de façon prédicative la difficulté d'appliquer cette méthode au circuit considéré et les performances obtenues. Les mesures de testabilité sont des techniques heuristiques, basées sur le concept de contrôlabilité, observabilité, testabilité. Nous présentons dans les paragraphes suivantes les plus utilisées.[9]

II-5-1 SCOAP (SCANDIA CONTROLABILITY AND OBSERVABILITY ANALYSIS PROGRAM)

SCOAP est une des premières mesures de testabilité apparue en 1979, elle permet d'analyser aussi bien les réseaux séquentiels que les réseaux combinatoires. Ainsi pour chaque noeud six valeurs sont calculées. Notant qu'un noeud combinatoire est une entrée primaire ou une sortie d'une porte, tandis qu'un noeud séquentiel est la sortie d'un élément mémoire.

-CC0(N) (respectivement CC1(N)), contrôlabilité combinatoire à '0' (resp à '1') du noeud N, est le nombre minimum de noeuds combinatoires qu'il faut forcer pour amener le noeud N à '0' (resp à '1')

- CO(N), observabilité combinatoire du noeud N, est le nombre de noeuds combinatoires qui doivent être forcés pour qu'un défaut sur le noeud N puisse être propagé du noeud vers une sortie primaire du circuit. De même, on définit les mesures SC0, SC1, SO qui sont les contrôlabilités et les observabilités séquentielles du noeud N ; elles ne tiennent compte que des noeuds séquentiels.

Le calcul est basé sur les règles suivantes:

- Si N est une entrée primaire du circuit, alors :

CC0(N) = CC1(N) = 1.

SC0(N) = SC1(N) = 0

si N est une sortie primaire du circuit, alors : **CO(N) = 0**

On définit une profondeur combinatoire(resp séquentiel)par module. Elle vaut '1' quand le module est combinatoire(resp séquentiel)'0' sinon.

II-5-1-1 CALCUL DES VALEURS SCOAP

SCOAP initialise d'abord les noeuds des entrées et sorties primaires avec les valeurs Précédentes, puis affecte des valeurs d'observabilité infinies aux noeuds autres que les sorties primaires. Pour une porte logique à sortie S et à entrée Ek, on a

CC1(S) = F(CC1(Ei) , CC0(Ej)) + 1

CO(Ej) = CO(S) + F(CC1(Ej)) + 1 I<>j

Sachant qu'un noeud collé à '0'(resp à '1') doit être controle à '1'(resp à '0'). Donc pour mesurer la testabilité d'un noeud collé à '0'(resp à '1') il faut estimer la quantité CC1 + CO (resp CC0 + CO).

Plus les valeurs de la mesure de testabilité sont élevées, plus le noeud est difficile à tester.

II-5-1-2 **INCONVINIENTS:**

SCOAP est la technique heuristique de la testabilité la plus connue, c'est l'une des premières a avoir vu le jour. Néanmoins elle souffre de certains inconvénients, en effet la corrélation entre les valeurs SCOAP et la difficulté à tester un noeud, bien qu'elle existe, n'est pas excellente.

SCOAP à tendance à éxagerer les valeurs associées à des noeuds facilement testables et inversement.

Elle s'avère inefficace pour guider l'élaboration d'une méthode automatique pour l'ajout des points de test car elle prédit rarement les résistantes au test aléatoire.

II-5-2 COP(CONTROLABILITY AND OBSERVABILITY PROGRAM)

COP opte pour une approche probabiliste du problème de vérification. La testabilité d'un noeud est la probabilité qu'un vecteur choisi aléatoirement détecte la panne portée par le noeud.

COP évalue la probabilité qu'un noeud prenne la valeur "1" ou"0", et la probabilité que cette valeur se reflete sur au moins une sortie primaire lorsqu'un vecteur choisi au hasard est appliqué aux entrées.

COP associe à chaque noeud cinq valeurs(soit N un noeud dans le circuit et n le nombre total de noeuds dans le circuit).

- **C1(N):** Contrôlabilité à '1' du noeud N.

$$C1(N) = \frac{\text{nombre de vecteurs donnant '1'}}{2^n}$$

- **C0(N):** Contrôlabilité à '0' du noeud N.

$$CO(N) = \frac{\text{nombre de vecteurs donnant '0'}}{2^n}$$

- **O(N):** observabilité pure de N.

$$O(N) = \frac{(\text{nbre de '0'} + \text{nbre de '1'}) \text{ sur N observables sur une sortie}}{2^n}$$

- **Pd0(N):** probabilité pour la détection d'un noeud collé à '0'

$$Pd0(N) = CO(N) \times O(N)$$

- **Pd1(N):** probabilité pour la détection d'un noeud collé à '1'

$$Pd1(N) = C1(N) \times O(N)$$

Ce traitement COP se fait en deux phases; La première phase consiste à initialiser lesContrôlabilités des entrées primaires à 0.5 (avoir '1' ou '0' sur une entrée primaire sont deux événements equiprobables), puis calculer les controlabités des noeuds internes en se basant sur la bibliothèque de fonctions définies selon le type de portes.

La deuxième phase consiste à initialiser les observabilités des entrées primaires à '1'(événement certain d'observer une sortie primaire), puis calculer les observabilités des autres noeuds en remontant dans le circuit vers les entrées primaires . Il est à noter que les nombres COP restent largement utilisées du fait qu'ils presentent un bon compromis precision-temps de calcul.

Un exemple pour une porte OR qui a deux entrées A, B et C comme sortie.

C1(C) = 1 - (1 - C1(A)) x (1 - C1(B)) .

O(A) = (1 - C1(B)) x O(C) .

Cette métrique, tout comme SCOAP d'ailleur, se base sur l'indépendance des signaux aux entrées d'une porte ce qui n'est pas réaliste. Par conséquent des problèmes surgisent quant un circuit possede des sortances reconvergentes, les valeurs sur les noeuds qui reconvergent à l'entrée d'une porte ne sont plus indépendantes.

II-5-3 STAFAN(STATISTICAL FAULT ANALYSIS)

comme la métrique COP, STAPAN effectue un calcul probabiliste des contrôlabilités et observabilité. Cette méthode est venue pour remidier aux insuffisances de COP quant au traitement des circuits séquentiels. STAFAN resoud le problème de rétroaction (feedback) dans les circuits séquentiels, en éliminant celle ci afin de rendre le réseau combinatoire. Les sorties connectées aux lignes coupées sont reliées a une copie du circuit et l'opération est itérée jusqu'a la stabilisation des valeurs de contrôlabilité et d'observabilité initialisées au préalable à des valeurs nulles.

Durant la simulation du bon circuit les paramètres suivants sont calculés :

C1(N) ,C0(N) : Contrôlabilité a '1' ou à '0' du noeud N ,

O1(N) ,O0(N) : observabilité à '1' ou à '0' du noeud N ,

CYs : Contrôlabilité de la sortie ,

CYe : moyenne des contrôlabilités des entrées et

$$CTF = 1 - \frac{(|N(0) - N(1)|)}{(N(0) + N(1))}$$

avec N(0) (resp N(1)) le nombre de vecteurs d'entrée pour lequel la sortie est à '0' (resp à '1')

CTF : facteur de transfert de contrôlabilité d'un composant. Le concept de CTF est utilisé pour expliquer la possibilité de diminution de l'information de contrôle au moment ou elle propagée à travers le circuit.Le CTF d'un composant doit représenter l'aptitude de contrôler la sortie d'un composant par l'application des valeurs d'entrée. L'observabilité d'une entrée est exprimée :

(OY)e = (OTF) x (OY)s

(OY)s : est la moyenne d'observabilité des sorties pour un module à plusieurs sortances.

(OTF) : facteur de transfert d'observabilité d'un composant . L'OTF d'un composant doit représenter la facilité de propagation de la valeur de faute à travers le composant.[10]

La mesure d'observabilité (OY) sur chaque noeud varie entre '0' et '1'. (OY)e est assignée à chacune des entées des composants. Le facteur OTF est détèrminé comme suit :

$$OTF = \sum_{i=0}^{N} P_i FD_i$$

avec Fdi la portion de vecteurs qui sensibilise un chemin depuis l'entrée i jusqu'a la sortie du module par les vecteurs d'entées appliqués. Et Pi : la probabilité que l'entrée i contient un défaut. L'OTF et la CTF d'un circuit séquentiel se calcule de la même manière. Des règles permettant de résoudre le problème des sortances a plusieurs destinations ont été proposées par STEPHENSON et GRASON :

1- Pour de bonnes contrôlabilités, il faudrait, assigner à chaque destination la contrôlabilité de la sortie divisée par (1+LOG10+D); D: etant le nombre de destination.

2- $OY_s = 1 - \prod_{i=1}^{d}(1 - OY_i)$ qui est l'observabilité de la sortance et OYi est l'observabilité de chaque destination. Par contre aux autres méthodes, STAFAN effectue une simulation du circuit en plus du temps de traitement itératif du circuit séquentiel consomment beaucoup de temps CPU. Un autre inconvénient est celui de l'espace nécessaire au stockage.[13]

II-5-4 CAMELOT (COMPUTER AIDED MESURE FOR LOGIC TESTABILITY)

La testabilité d'un noeud est définie comme suit :

TY(N) = CY(N) x OY(N) avec CY(N) : Contrôlabilité du noeud N

OY(N) : observabilite du noeud N

La contrôlabilité d'un noeud est calculée en propageant les valeurs de contrôlabilité des entrées primaires, et ce grâce aux facteurs de transfert de contrôlabilité CTF définit la mesure précédente. L'observabilité d'un noeud est déterminée grâce au facteur de transfert d'observabilité (OTF) propre à chaque porte. CAMELOT nécessite le stockage des facteurs de transfert et des tables de vérité dans le cas du calcul de l'OTF pour des éléments mémoire. Une table de vérité dressée pour un élément mémoire contient, en plus des combinaisons des entrées et de l'état interne de la bascule, l'effet de chaque combinaison sur la sortie en l'abscence de défauts, les défauts détectés et les états invalides. Les contrôlabilités des entrées primaires sont initialisées à '1', les contrôlabilités des autres noeuds sont calculées comme suit :

CY (noeud en sortie) = CTF x f (CYs(ei) .

Si on a un dispositif à N entrées (E1 ,...,En) et à sortie S,alors :

OY(Ei - S) = OTF x g (CYs(Ej)) j <> i ,

Pour calculer OTF deux facteurs nécessaires :

N(SP : Ei ⟶ S) = nombre de vecteurs sensibilisant le chemin .

N(IP : Ei ⟶ S) = nombre de vecteurs desensibilisant le chemin .

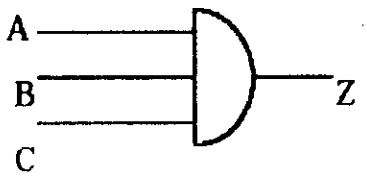
$$OTF = \frac{[N(SP: \rightarrow S)]}{[N(SP: \rightarrow S) + (IP: E_i \rightarrow S)]}$$

EXEMPLE

N(SP : A ⟶ Z) =1

N(IP: A ⟶ Z) = 7

OTF (A ⟶ Z) = 0,125



Pour une porte AND à 3 entrées A,B,C et à sortie z La fonction f est calculée de la même manière que g .

$$f(CYs(E_i)) = \frac{[\sum CY(\text{entrées asynch}) + \prod CY(\text{horloge}) + \prod CY(\text{hori}) \times \sum CY(\text{entrées synch})]}{1+1}$$

Une autre mesure de testabilité est définie par CAMELOT, c'est la testabilité moyenne d'un circuit donnée par:

$$TYm(\text{circuit}) = \frac{\sum TY(N_i)}{n}$$

II-7 CONCEPTION POUR LA TESTABILITE :

La conception pour la testabilité consiste à imposer des règles de conception pour réduire les problèmes du test ou bien modifier le circuit déjà conçu pour le rendre facilement testable, ce qui nécessite de la circuiterie et des E/S additionnelles. Il existe plusieurs approches pour la conception en vue de test :

II-7-1 LES METHODES NON STRUCTUREES :(AD-HOC)

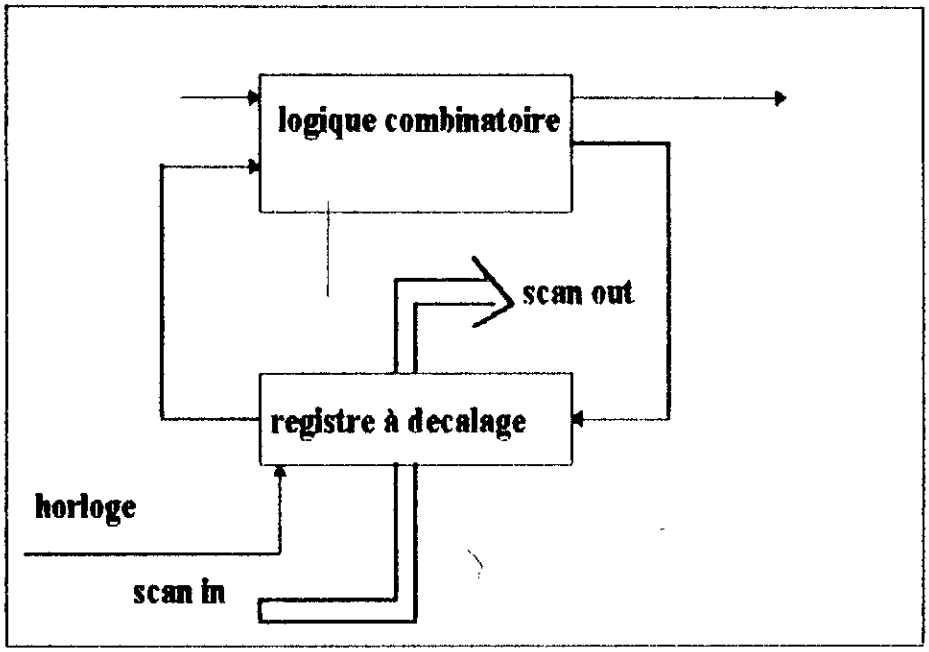
Le test étant pris en compte très tard dans le cycle de conception. Ces méthodes cherchent à identifier les noeuds difficiles à observer et/ou à controler d'un circuit pour cela on peut s'aider de programmes d'analyses de testabilité pour les identifier ou à défaut s'en remettre à certains regles empiriques, dont les plus utiles sont :

- Elimination des redondances.
- Elimination des boucles de rétroaction externes.
- Initialisation de l'état du circuit. on utilise généralement des entrées asynchrones PRESET et CLEAR
- Insertion de points de test.
- partitionnement. [12]

II-7-2 LES APPROCHES STRUCTUREES:

Pour les circuits VLSI, les approches AD-HOC ne sont plus suffisantes, les approches structurées sont venues pour éliminer le problème complexe du test des circuits séquentiels, et ce en structurant les éléments mémoires pour éviter les rétroactions incontrôlées et assurer la contrôlabilité et l'observabilité des éléments mémoires.

Les approches structurées sont basées essentiellement sur le concept de chaine de balayage ou "SCAN".Le principe est d'incorporer au circuit un second mode d'opérations (MODE TEST). Dans lequel les bascules sont chainées de façon que l'état du circuit soit facilement observé ou controlé.



METHODE DE SCAN

Les étapes de test du circuit sont:

1- test des bascule à travers le "Scan path",

2- controle des états internes,

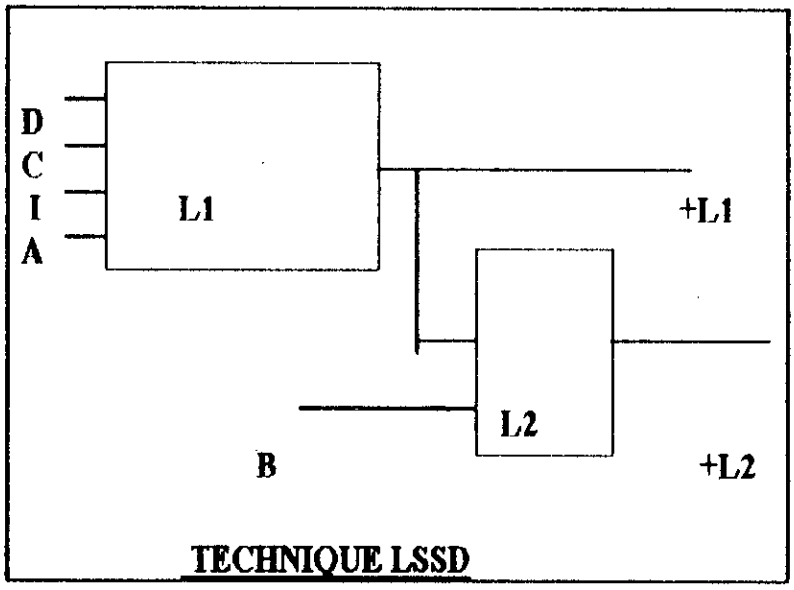
3- test des parties combinatoires à l'aide d'un registre à décalage, on place un vecteur aux entrées du circuit combinatoire, et on charge les sorties de ce bloc dans le registre ce qui permettera de les observer en effectuant le décalage. Les approches structurées les plus connues sont:

a-LSSD(LEVEL SENSITIVE SCAN DESIGN):

L'approche LSSD est la technique à base de bascules la plus utilisée, qui se décompose en deux parties distinctes:

La partie sensibilté aux niveaux (level sensitive) qui permet d'éliminer les effets des aléas et des sensibilités paramétriques (température, tolérance, etc...).

La partie conception avec chaine de balayage (SCAN DESIGN) est la partie la plus importante de LSSD. Son but est de permettre d'imposer des valeurs connues à tous les noeuds internes, et en suite d'observer les résultats sur les sorties. En pratique cela signifie le remplacement de tout élément de mémorisation par le dispositif SRL (shift register latch) de la figure suivante:



TECHNIQUE LSSD

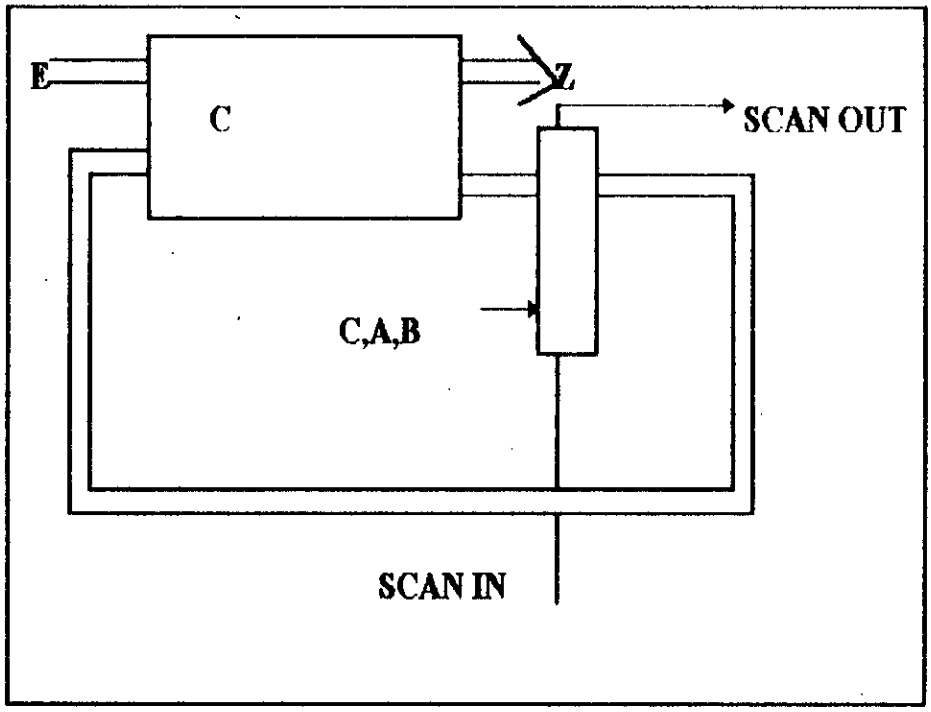
D : entrée des données normale de la bascule L1

C : signal d'horloge de L1

I : données que l'on veut forcer dans le point de mémorisation pour le test

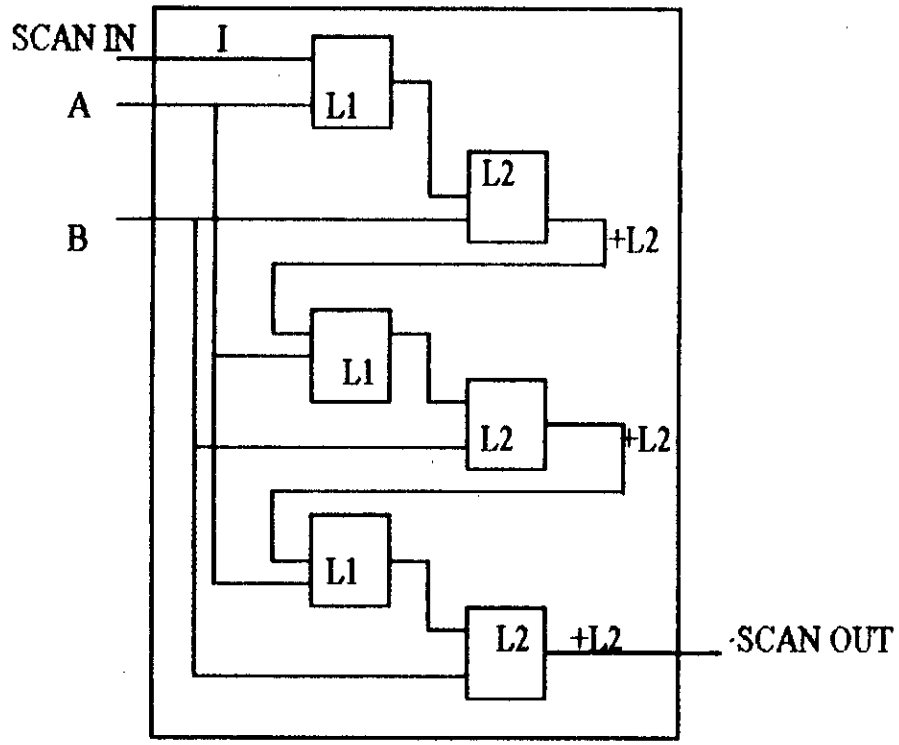
A,B : commandes de décalage permettant de faire circuler le mot d'état à des fins de test.

D'une façon globale, le circuit séquentiel est rempli par un circuit dont les états de mémorisation sont complètement commandables et observables par décalage, selon le principe de la figure suivante:



PRINCIPE DE LSSD

L'enfilage et le décalage de l'information à travers les points mémoire se fait selon le schéma d'interconnexion suivant :



INTERCONNEXION DE SRLs SUR CIRCUIT

En mode test les points de mémorisation étant montés en registre à décalage, ces éléments seront des points "MAITRE-ESCLAVE". Un signal A change le maitre, puis un signal B change l'esclave avec le contenu du maitre. La donnée de test est envoyée sur l'entrée i. En fonctionnement normal, la donnée D est rentrée sous le signal de l'horloge C dans le point maitre: cette donnée D est directement accessible en L1 ; elle peut également transiter par le point esclave. Dans ce cas ,l'horloge C est remplacée par un couple de signaux C1 ou C2 ou C1 charge le maitre et C2 charge l'esclave. Pour réduire le nombre de signaux de controle, C2 peut être remplacée par le signal B qui est utilisé en mode test.

b- SCAN PATH:

Le principe du SCAN PATH est similaire à celui utilisé dans la technique LSSD où les éléments de mémorisation utilisés sont du type bascule D. Une horloge C2 synchronisée , en mode test, l'aquisition de la donnée de test dans un point mémoire maitre; le complément de C2 fait passer du point maitre au point esclave. En mode normal un signal d'horloge C1 et son complément, font transiter la donnée normale.

Dans cette technique, deux signaux d'horloge seulement sont utilisés, contrairement à la Technique LSSD où 4 signaux indépendants peuvent être implémentés. Les points de mémorisations fonctionnent toujours en mode MAITRE-ESCLAVE.

c- SCAN/SET LOGIC :

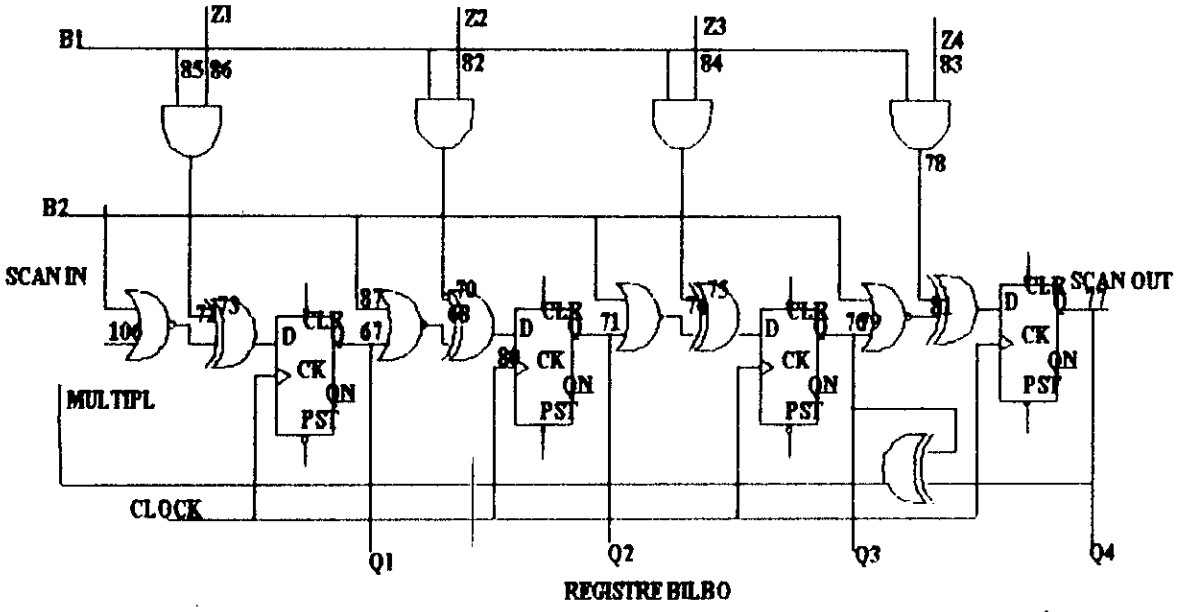
Le concept de base de cette technique est similaire aux deux précédentes approches, consiste à utiliser des registres à décalage, pour écrire et lire des données. Mais ces registres ne sont pas intégrés au chemin de données et tous les éléments de mémorisation ne sont pas nécessairement commondables et observables à travers les registres à décalage . Le controle et l'observation des points de mémorisation sont réduits à un sous ensemble de ces points.

d- RANDOM ACCESS SCAN :

Cette approche a les mêmes objectifs que les techniques LSSD et SCAN PATH, à savoir la commondabilité et l'observabilité de l'ensemble des points mémoire internes. La différence essentielle consiste en la non utilisation de registres à décalage. Les points de mémorisations contrôlables et observables sont regroupés en plan mémoire deux dimension 2D . Chaque point de mémorisation est alors repéré par son adresse (x,y). En fonctionnement normal, il reçoit une donnée D synchronisée par un signal d'horloge CLK est affiche une sortie Q . En mode test , une entrée de test donnée, synchronisée sur une horloge de test, controle le contenu du point de mémorisation sélectionné par (x,y) et affiche une sortie de test .

e- BUILT-IN LOGIC BLOC OBSERVATION (BILBO) :

Cette technique conjugue les concepts de LSSD et SCAN PATH (possibilité de séparer le réseau en parties combinatoires et séquentiels) et les concepts d'analyse de signature.



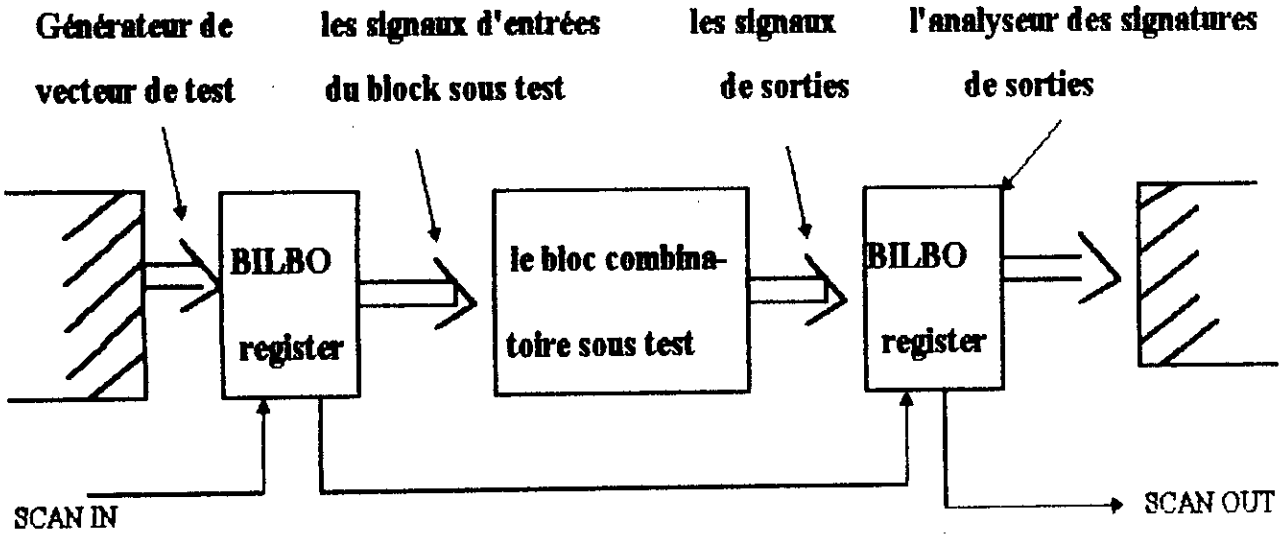
REGISTRE BILBO forme Générale

Un tel registre peut fonctionner en trois modes selon la valeur de commande (B1,B2) :

- si B1B2 = 11 => mémorisation parallèle ;

- si B1B2 = 00 => le registre peut être chargé en série à partir du plot de " SCAN IN " et observé en sortie sur le plot de "SCAN OUT"

- si B1B2 = 10 => le registre est monté en diviseur polynomiale avec entrées parallèles, il peut alors servire soit de registre d'analyse soit de générateur de données aléatoires. Un circuit combinatoire placé entre deux registres de ce type, dans le dernier mode, peut être testé avec des opérandes générés par le premier, et observer par une analyse de signature qui est effectuée dans le deuxième registre .



Le concept de base du test d'un bloc avec le registre BILBO

CHAPITRE III

Technique hiérarchique de la génération de test pour les modules VHDL

III-1 INTRODUCTION :

Cette technique de génération de test prend un avantage de l'hierarchie complète en particulier dans les modèles VHDL. La description graphique des modèles VHDL (PMG : process model graph) est une représentation directe de la division de la fonctionnalité avec un modele VHDL. L'algorithme appelé HBTG(hierarchical behavioral test generator) utilise le PMG comme base pour la génération de test. Pour chaque processus dans le PMG les tests sont précalculés et introduits comme base de données.

Utilisant ces tests prémitifs, le HBTG construit hiérarchiquement les tests pour l'entité entière.

III-2 METHODOLOGIE DE LA GENERATION DE TEST :

III-2-1 UTILISATION DU PMG

Le modèle VHDL du circuit VLSI peut être développé en partageant la fonction du circuit à des sous- fonctions. Chacune de ces sous-fonctions peut être représentée par un processus .La représentation graphique d'un modèle VHDL, le PMG, est fréquement utilisé pour représenter le modèle VHDL. Le PMG est un graphe dont ses noeuds représentent les processus dans le modèle VHDL et ses ponts (liaisons) représentent les signaux entre processus. Les ports sensibilisés sont construits pleins dans la zone interne du pont, donc, nous pouvons dire que le modèle VHDL est actuellement un ensemble des processus communicant entre eux .

III-2-2 APPROCHE DE GENERATION DE TEST :

On utilise une approche hiérarchique pour la génération de test. On sait que le stockage des processus primitifs accélère le processus du développement du modèle. Dans l'approche présentée, l'information de test pour les modules individuels du circuit est précalculée et stockée dans des fichiers. La séquence de test pour l'entité entière est donc construite de ces tests primitifs hiérarchiquement .

Les tests precalculés sont seulement l'information disponible pour l'algorithme concernant la fonctionnalité du processus dans le PMG. Qoique, ils fournissent l'information suffisante pour des opérations différentes dans le processus de génération de test. Dans plusieurs techniques traditionnelles de génération de test, des modèles de faute spécifique sont utilisés pour représenter les fautes dans le circuit par exemple : S.A.V est un modèle de faute ordinairement utilisé pour représenter les fautes logiques dans la génération de test au niveau portes. .Dans une approche de niveau superieur comme HBTG des modèles de fautes connues "STUCK-THEN", "STUCK-ELSE", controle d'instruction, etc..., sont utilisées.

Cette approche n'adopte pas des modèles de fautes spécifiques. Au lieu de générer une séquence de test pour détecter une faute, HBTG construit une séquence de test qui exerce le module Complètement. Le but est de générer un pattern de test efficace pour tester les différentes opérations du modèle. Dans le but que le test soit efficace, la séquence de test doit être la plus courte possible et doit vérifier de nombreuses opérations possibles du modèle. Sachant les sémantiques du modèle VHDL, le processus à l'intérieur du corps de l'architecture du modèle comportemental sera exécuté seulement s'il y'a un événement(changement de la valeur du signal) au moins dans un de ses ports d'entrée sensibilisé. par conséquent le test qui génère les événements sur les ports sensibilisés est meilleur que celui qui ne génère pas dont le but est d'exercer le modèle, une contrainte majeure utilisée par le HBTG est que la séquence de test doit activer le plus possible des ports sensibilisés dans le modèle. Dans ce contexte, activer un port revient à créer un événement sur ce port qui génère un front montant ou descendant sur un signal à bit unique ou associer des bit vecteurs différents à un bus multi-bits. Dans ce cadre il est préférable de donner quelque définitions sont fréquemment utilisées dans l'approche.

DEFINITION 1:

Une séquence de test effectue pour le modèle VHDL est la séquence de test qui active tous les ports d'entrée sensibilisés du modèle au moins une fois dans le processus de génération de test, les chemins sensibilsés sont construits à travers le PMG.

DEFINITON 2:

Un chemin sensibilisé est un chemin direct qui se commence au port d'entrée primaire sensibilisé (PI) et se termine au port de sortie primaire (PO) avec des ports intermédiaires le long du chemin composé autant que possible des ports sensibilisés.

II-2-3 **FORMAT DES TESTS PRECALCULES**:

Des exigences de base pour les tests précalculés sont listés ci-dessous :

1- Pour chaque processus, l'entrée qui est une entrée primaire pour l'entité, le test avec un événement sur le signal doit être inclu, le meilleur test doit générer un événement sur la sortie du processus également.

2- Pour un signal interne qui est un signal d'entrée sensibilisée pour tous les processus, les tests qui peuvent générer un événement aussi une valeur constante sur le signal doivent être inclus.

3- Pour un signal qui n'est pas un signal d'entrée sensibilisé pour n'importe quel processus dans le modèle, seulement le test qui génère une valeur constante sur le signal est nécessaire.

4- Pour n'importe quelle trame de temps, seulement le signal d'entrée doit avoir un événement dans le fichier de données. Les tests ayant des événements sur les signaux sensibilisés garantissent qu'il y'a au moins un test pour activer chacun des ports sensibilisés, aussi l'exigence que dans une trame de temps seulement un signal d'entrée doit avoir un événement dans le fichier des données du test est important .

II-3 ALGORITHME DE GENERATION DE TEST

Comme il est discuté dans la section 2 un algorithme appelé HBTG était développé systématiquement pour développer les modèles VHDL, l'algorithme est une version modifiée de l'algorithme D .[11]

II-3-1 **L'ALGORITHME HBTG**

L'algorithme inclu deux parties : La première partie, est la partie d'initialisation, les chemins sensibilisés sont construits à travers le PMG comme il est expliqué dans (2-2). La deuxième partie est la partie de génération de test, les chemins sensibilisés sont activés un par un jusqu'a ou tous les chemins sont activés. L'activation d'un chemin commence par l'activation du premier port le long du chemin.

La valeur du signal est donc propagée à la sortie (PO) sur le chemin, ça par l'activation des autres ports sensibilisés le long du chemin . Une fois le PO est atteinte, la justification commence à justifier les valeurs du signal interne spécifié durant le processus d'activation directe vers les entrées primaires. Au même temps l'implication est performante pour calculer les valeurs de la sortie du processus lorsque ses entrées sont spécifiées. Des termes utilisés dans le HBTG sont définis ci-dessous :

DEFINITION 1:

a FRONT SIGNAL: c'est le signal sur le chemin sélectionné qu'a était associé la valeur le plus récemment .

DEFINITION 2:

a PUT(process under test): est le processus sous test. Pendant la propagation, le PUT est le processus sur le chemin sélectionné qui a le front signal comme un de ces ports d'entrée. Durant la justification, le put est le processus dans le chemin sélectionner qui a le signal à être justifié aussi comme son port de sortie .

DEFINITION 3:

Le PORTACT d'un port : est le nombre de fois que la séquence de test a généré un événement sur le port. Il est signifiant seulement pour les ports d'entrée sensibilisés. Les données des tests précalculés pour chaque module sont stockés dans des fichiers. L'algorithme HBTG sélectionne les séquences de test appropriées à partir de ces fichiers primitifs et les ordonne durant le processus de génération de test, différents tests sont sélectionnés. Les règles suivantes sont suivies lors de la sélection de test :

1- Pour l'activation d'un port sensibilisé dans le chemin sélectionné le test avec un événement dans ce port est à sélectionner. Le meilleur test doit avoir aussi un événement sur le prochain port dans le chemin(le port de sortie du put) .

2- Pour la propagation d'un événement dans le chemin, le test ayant le même événement sur le port est désiré. Encore, le test avec un événement sur le prochain port du chemin est préférable.

3-Pour la propagation de la valeur constante du signal des ports d'entrée non sensibilisé, le test avec un événement dans un autre port d'entrée sensibilisé du put est besoin .

4- pour la justification de la valeur du signal, le test qui peut générer la même valeur sur le signal est exigé .

5- Pour le processus d'implication, le test avec des valeurs sur tous les ports d'entrées du processus egales aux valeurs courantes des ports doivent être sélectionnés .

III-3-2 CRITERES POUR LA CONSTRUCTION DES CHEMINS SENSIBILISES:

Lors de la construction des chemins sensibilisés les critères suivantes doivent être respectés :

* Sélectionner le premier port le long du chemin.

- Si le processus a une seule sortie, on va au prochain port .

- Si le processus a plusieurs sorties, on doit choisir le chemin dont le nombre des chemins sensibilisés est minimal(le plus petit).

* Si plus d'un signal quittant le port, donc choisir le moins activé de tous .

* Sinon choisir le prochain port de destination .

III-3-3 IMPLEMENTATION DE L'ALGORITHME HBTG

```
DEBUT
  introduire les chemins sensibilisés;
    TANT QU'il y'a un chemin sensibilisé dans le PMG non activé,
              faire sélectionner le chemin;
              decider le put; activer le premier port;
              augmenter le portact du port;
          TANT QUE la sortie primaire n'est pas atteinte,
                    decider le put;
            SI le front signal est un événement, DONC
                SI le signal est un signal sensibilisé DONC
                  faire la propagation de l'événement,
                  augmenter le portact du port;
                SINON faire la propagation du signal
                        sensibilisé,augmenter le portact du port;
                FIN DE SI
                      faire l'implication;
              SINON faire la propagation du signal non sensibilisé ;
                      faire l'implication;
            FIN DE SI
          FIN DE TANT QUE;

          TANT QU'il y'a un processus avec une sortie
                connue et une entrée inconnue existe,faire sélectionner
                le chemin sensibilisé qui traverse OS s'il est possible;
            TANT QUE la sortie primaire sur le chemin n'est pas atteinte faire,
                  decider le PUT ;
                  faire la justification du signal jusqu'à l'entrée primaire ;
                  faire l'implication ; augmenter le portact du port;
            FIN de TANT QUE;
          FIN de TANT QUE;
    FIN de TANT QUE;
FIN .
```

CHAPITRE IV

Simulation d'un Multiplieur et application de la HBTG

IV-0 INTRODUCTION :

Il s'agit de simuler en VHDL un multiplieur est d'appliquer les vecteurs de test générés par le programme de la HBTG pour la vérification de la fonctionnalité du multiplieur .

Le multiplieur simulé est de (6 bits x 6 bits) donc le résultat est de 12 bits .

Pour la programmation de la HBTG nous avons élaboré en PASCAL un programme qui génère les vecteurs de test pour le multiplieur , à partir des tests précalculés pour les modules individuels (produit partiel , enable, décalage, addition). Ces tests ont été introduits dans des tableaux de chaines de caractères et le programme de la HBTG extrait les tests désirés et genère le test pour l'entité entière du multiplieur .

IV-1 SIMULATION DU MULTIPLIEUR:

Comme il est indiqué dans le schéma synoptique du multiplieur , nous avons procéder à un multiplieur de type "PARALLELE" pour minimiser le temps de propagation du "CARRY"ainsi de rendre facile le test de sa fonctionnalité: ce type de multiplieurs est basé sur le principe "SHIFT and ADD".[14]

Le multiplieur est constitué essentiellement d'un :

- Un réseau combinatoire de portes logiques AND réalisant le produit partiel .
- Un réseau séquentiel qui est des registres à différents pas de décalage .
- Des additionneurs (12bits + 12bits) réalisant l'addition des produits partiels décalés .

On note que les valeurs prises dans ce qui suit pour le temps de propagation des composants ainsi la fréquence de l'horloge sont des standards TTL .

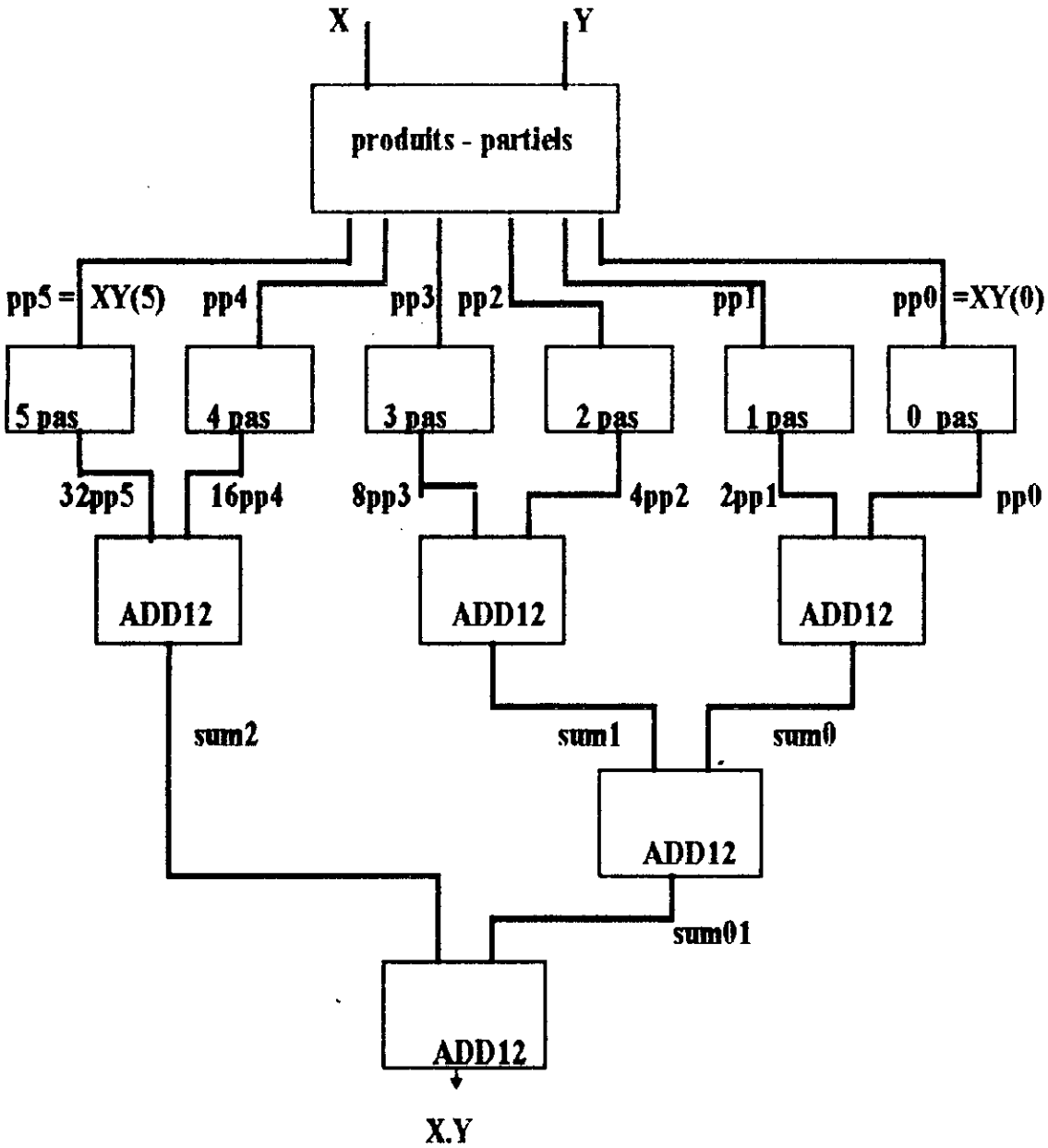
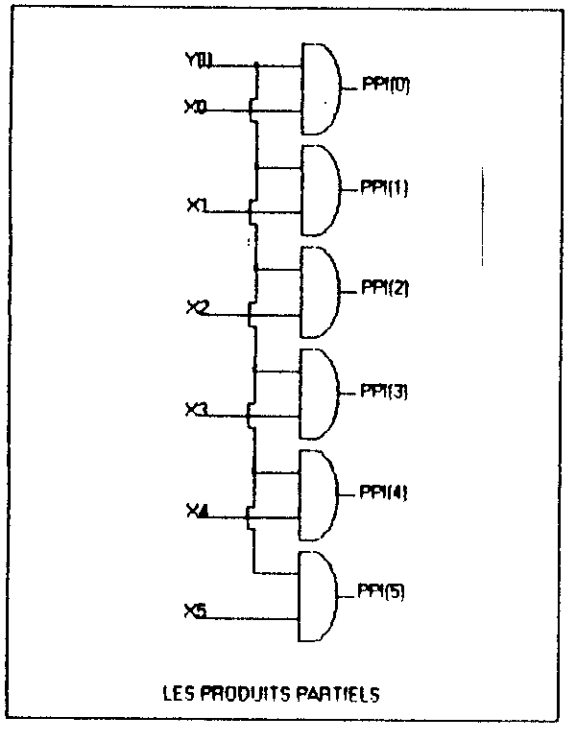


schéma synoptique du multiplieur

IV-1-1 LES PRODUITS PARTIELS :

La fonction produit partiel est réalisée par des portes AND simple comme il est indiqué dans la figure des produits partiels. Le temps nécessaire pour faire cette opération est de 3 ns.



LES PRODUITS PARTIELS

Donc pour chaque bit du multiplicande (Y) on fait la même opération avec tous les bits du multiplicant (X) ;ce qui exige six réseaux identiques à la figure ci-dessus. Le resultat sera 6 vecteurs PPi(0), PPi(1),..., PPi(5) .

Une fois cette opération est terminée , ces vecteurs constituent les entrées des registres à décalage .

IV-1-2 LA FONCTION DE DECALAGE :

La fonction de décalage est réalisée par six registres à différents pas de décalage et à des entrées séries '0' chacun . chaque registre est activé lorsque l'horloge passe de '0' à '1' (front montant) , ainsi il est validé par deux entrées; une entrée inversée , l'autre non inversée (G1.G2). En VHDL pour l'horloge le front montant est obtenu par :

WAIT UNTIL NOT CLOCK'STABLE AND CLOCK = '1' ;

Le temps consommé par chaque registre à décalage est proportionnel au nombre de pas de décalage , il est compris entre 4 ns et 22 ns.

Le tableau suivant résume le fonctionnement des registres à décalage:

	clock	$G1.\overline{G2}$	regin	regout
shifter0	⎍	⎍ 1	**pp0**	**pp0 after 6 ns**
shifter1	⎍	⎍ 1	**pp1**	**2pp1 after 8 ns**
shifter2	⎍	⎍ 1	**pp2**	**4pp2 after 10 ns**
shifter3	⎍	⎍ 1	**pp3**	**8pp4 after 12 ns**
shifter4	⎍	⎍ 1	**pp4**	**16pp4 after 14 ns**
shifter5	⎍	⎍ 1	**pp5**	**32pp5 after 16 ns**

FONCTIONNEMENT DES SHIFTERS

La période de l'horloge est donnée dans le fichier de données par :**FORCE CLOCK 0 0,1 15 -REPEAT 30**
Donc un décalage à gauche avec entrée série (LS = 0) donne une multiplication par 2^i du nombre décalé (i :represente le nombre de pas) .

IV-1-3 **L'ADDITION** :

Comme il est montré dans le schéma synoptique du multiplieur on a cinq additionneurs 12 bits :

PP0 + 2PP1 => SUM0 ;

4PP2 + 8PP3 => SUM1 ;

16PP4 + 32PP5 => SUM2 ;

SUM0 + SUM1 => SUM01 ;

SUM01 + SUM2 => PROD = X . Y ;

Pour la réalisation d'un additionneur (12 bits + 12 bits) , il faut 12 cellules d'un additionneur 1 bit , l'entité de l'additionneur 1 bit est prise comme une composante dans l'additionneur 12 bits ; réaliser un additionneur : revient à connecter des additionneurs 1 bit comme suit :

- La sortie "CARRY-OUT" de la cellule (i-1) est le "CARRY-IN" de la cellule (i) .
- Les sorties SUM(i) de chaque additionneur 1 bit sont les bits du résultats de l'addition .
-La cellule (0) a un "CARRY-IN = 0" .

Le schéma synoptique suivant montre le fonctionnement d'un additionneur 12 bits.

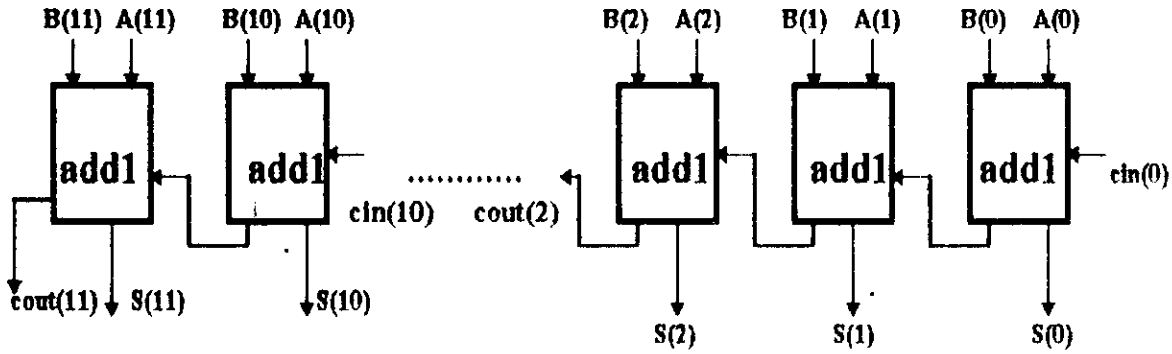


schéma synoptique de l'additionneur 12 bits

Chaque cellule "ADD1" est un réseau combinatoire avec des portes logiques AND, XOR ,OR avec un temps de propagation (3 ns, 5 ns, 4 ns) respectivement .(voir chap 1) .

Cette fonction de rassemblage entre les " FULL ADDER 1 bit " se fait en VHDL par l'instruction :

```
ENTITY GENERIC IS
   ...
   ...
FOR I = 0 TO 11 GENERATE
IF I = 0 ADD1 : PORT MAP ( X(0),Y(0),0,SUM(0),Cout(0))
IF I > 1 ADD1 : PORT MAP ( X(I),Y(I),Cin(I-1),SUM(I) ,
                    Cout(I))
END GENERATE ;
```

Pour chaque additionneur utilisé dans le multiplieur on utilise la même entité de "ADD12" , ce qui change la configuration seulement :

exemple :

```
C1 :ADD12 : PORT MAP ( REG0,REG1,SUM0,COUT(0) )
C2 :ADD12 : PORT MAP ( REG2,REG3,SUM1,COUT(1) )
.
.
C5 :ADD12 : PORT MAP ( SUM2,SUM01,PROD,COUT )
```

(voir programme en détaille).

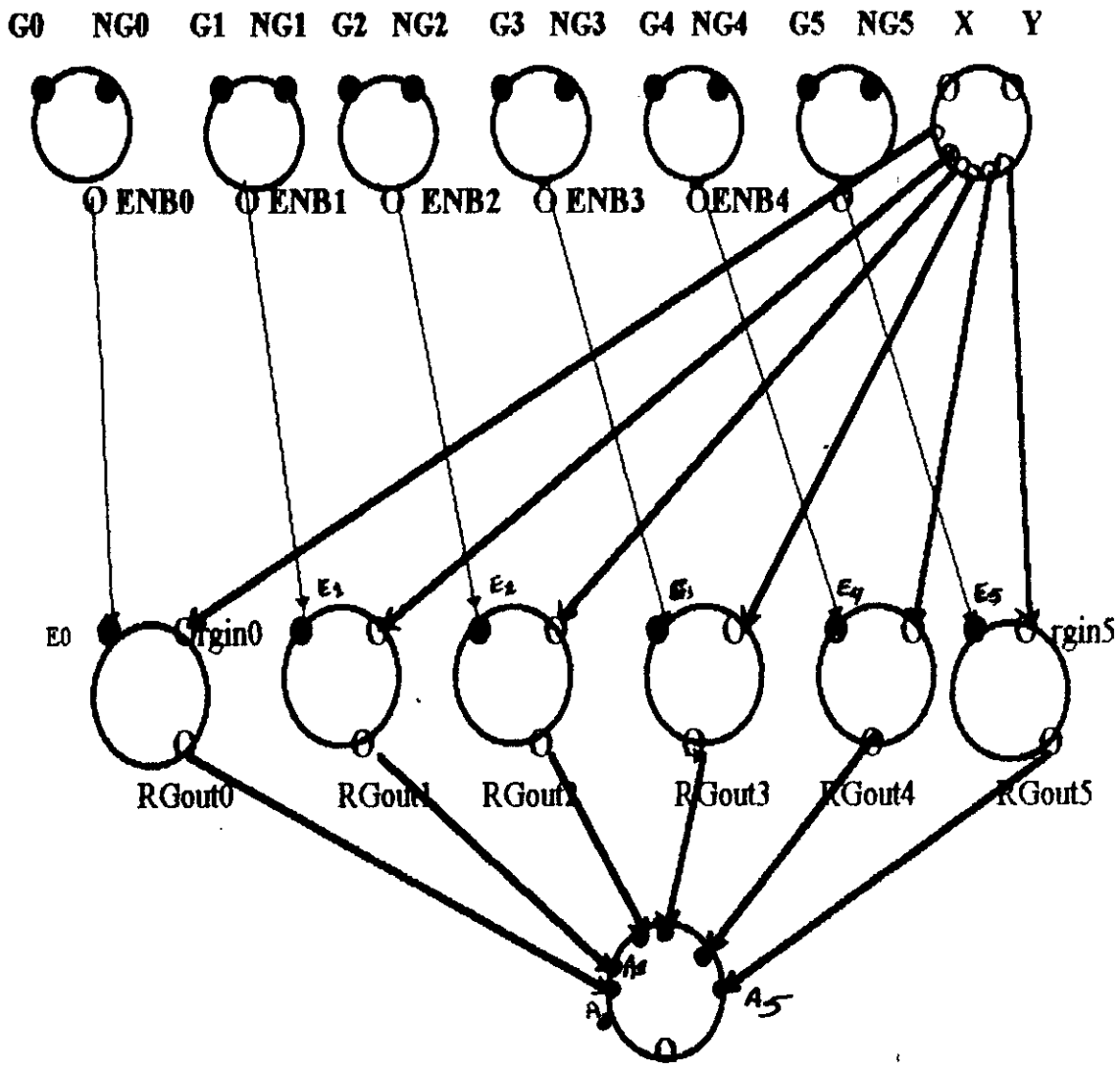
IV-2 APPLICATION DE LA HBTG :

Comme nous l'avons vu dans le chapitre "3" ,la technique HBTG à deux phases essentielles :

- La phase d'initialisation :construction des chemins sensibilisés à partir du PMG pour le module (dans le cas du multiplieur).
- La phase de génération de test à partir des tests individuels précalculés pour chaque module et introduits comme base de données pour la HBTG (dans notre cas les modules sont : le processus des produits partiels ,les processus enables , les processus shifters ,le processus d'addition)

IV-2-1 REPRESENTATION GRAPHIQUE DES PROCESSUS VHDL DU MULTIPLIEUR (PMG):

Le graphe suivant montre le PMG du multiplieur:



OUT = X.Y

LE PMG DU MULTIPLIEUR

IV-2-2 CONSTRUCTION DES CHEMINS SENSIBILISES :

Un chemin sensibilisé se commence par un port sensible et se termine à la sortie << OUT >> . Les ports sensibles sont indiqués dans le PMG par les points pleins (O),d'ou les chemins construits sont :

SP00 = G0	SP10 = NG0	SP20 = G1	SP30 = NG1
SP01 = ENB0	SP11 = ENB0	SP21 = ENB1	SP31 = ENB1
SP02 = E0	SP12 = E0	SP22 = E1	SP32 = E1
SP03 = REG0	SP13 = REG0	SP23 = REG1	SP33 = REG1
SP04 = A0	SP14 = A0	SP24 = A1	SP34 = A1
SP05 = OUT	SP15 = OUT	SP25 = OUT	SP35 = OUT

SP40 = G2	SP50 = NG2	SP60 = G3	SP70 = NG3
SP41 = ENB2	SP51 = ENB2	SP61 = ENB3	SP71 = ENB3
SP42 = E2	SP52 = E2	SP62 = E3	SP72 = E3
SP43 = REG2	SP53 = REG2	SP63 = REG3	SP73 = REG3
SP44 = A2	SP54 = A2	SP64 = A3	SP74 = A3
SP45 = OUT	SP55 = OUT	SP65 = OUT	SP75 = OUT

SP80 = G4	SP90 = NG4	SP100 = G5	SP110 = NG5
SP81 = ENB4	SP91 = ENB4	SP101 = ENB5	SP111 = ENB5
SP82 = E4	SP92 = E4	SP102 = E5	SP112 = E5
SP83 = REG4	SP93 = REG4	SP103 = REG5	SP113 = REG5
SP84 = A4	SP94 = A4	SP104 = A5	SP114 = A5
SP85 = OUT	SP95 = OUT	SP105 = OUT	SP115 = OUT

IV-2-3 TESTS DES MODULES INDIVIDUELS :

Avant la simulation du multiplieur il était préférable de programmer chaque processus seul et vérifier sa fonctionnalité puis former un programme en VHDL du multiplieur qui utilise ces processus.

Les tests des modules individuels (tests précalculés) sont les suivants :

a- TEST DU PRODUIT PARTIEL :

	X	Y (i)	PPi
Di: représente une	Di	0	0
donnée quelconque	Di	1	Di
(vecteur de 6 bits)			

Avant de lister les tests des autres modules voici la signification des symboles utilisés :

'0' : Un zero constant dans un bus a bit-unique .

'1' : Un '1' constant dans un bus a bit-unique .

'R' : Une transition de '0' à '1' .

'F' : Une transition de '1' à '0' .

'X' : Une valeur inconnue ou n'est pas chargée .

'Z' : Haute impédance.

b- VECTEURS DE TEST DES PROCESSUS DE VALIDATION (ENABLE) :

On donne les vecteurs de tests pour un module (i) qui est valable pour les autres .

G(i)	NG(i)	ENB(i)
R	0	R
1	0	1
1	R	F
1	1	0
1	F	R
F	0	F
0	0	0

c- **TESTS DES SHIFTERS** :

SHIFTER0

E0	REGIN0	REGOUT0
0	PP0	Z
1	PP0	PP0
F	PP0	Z
R	PP0	PP0
X	PP0	X

SHIFTER1

E1	REGIN1	REGOUT1
0	PP1	Z
1	PP1	2PP1
F	PP1	Z
R	PP1	2PP1
X	PP1	X

SHIFTER2

E2	REGIN2	REGOUT2
0	PP2	Z
1	PP2	4PP2
F	PP2	Z
R	PP2	4PP2
X	PP2	X

SHIFTER3

E3	REGIN3	REGOUT3
0	PP3	Z
1	PP3	8PP3
F	PP3	Z
R	PP3	8PP3
X	PP3	X

SHIFTER4

E4	REGIN4	REGOUT4
0	PP4	Z
1	PP4	16PP4
R	PP4	16PP4
F	PP4	Z
x	PP4	X

SHIFTER5

E5	REGIN5	REGOUT5
0	PP5	Z
1	PP5	32PP5
R	PP5	32PP5
F	PP5	Z
X	PP5	X

c- **TEST DE L'ADDITIONNEUR** :

A0	A1	A2	A3	A4	A5	SUM
0	0	0	0	0	0	0
D1	0	0	0	0	0	D1
D1	D2	0	0	0	0	D1 + D2
D1	D2	D3	0	0	0	D1 + D2 + D3
D1	D2	D3	D4	0	0	D1 + D2 + D3 + D4
D1	D2	D3	D4	D5	0	D1 + D2 + D3 + D4 + D5
D1	D2	D3	D4	D5	D6	D1 + D2 + D3 + D4 + D5 + D6

Les vecteurs de tests obtenus pour verifier la fonctionnalité du multiplieur ainsi les trames de temps donnant l'ordre d'application sont données par le tableau ci-aprés.(Voir tableau des vecteurs générés par la HBTG)

Des exemples de multiplication qui justifient le bon fonctionnement du programme elaboré sont listés en Annexe.

CONCLUSION

Dans ce travail nous avons abordé deux sujets, qui sont la descripltion des circuits logiques par le VHDL et comment procéder hiérarchiquement pour générer un test qui exerce tout le module VHDL exhaustivement.

Le circuit spécifié est un multiplieur (6 bits x 6 bits); il est constitué de circuits séquentiels et de circuits combinatoires simples.

Ce multiplieur nous a permis d'utiliser les différents niveaux d'abstraction du langage (comportemental, structurel,dataflow...), pour decrire le fonctionnement des circuits logiques et de bien observer le déroulement des séquences en fonction du temps, qui est un phénomène réel que la physique y oblige..

Donc le grand intéret de ce travail est de mettre au point le fonctionnement d'un circuit matériel par un programme (logiciel) qui a donné de bons résultats, qui peuvent être justifiés en utilisant l'affichage soit en binaire soit en décimal (voir Annexe).

Concernant la partie de la génératin de test nous avons présenté les différents algorithmes utilisés , les méthodes suivies pour la mesure de la testabilité , ainsi la conception pour la testabilité (conception en vue du test) . Enfin, nous avons appliqué une approche hiérarchique pour générer les vecteurs de test et de les appliquer au multiplieur afin de vérifier sa fonctinnalité ; cette approche qui a réduit le nombre de vecteurs de test d'un nombre très grand , qui est de l'ordre de 2^{12} , pour trier toutes les combinaisons possibles et de faire justifier les résultats obtenus ; ce nombre a été réduit à 14 vecteurs, mais on tient compte de la génération des vecteurs de tests pour les modules individuels qui sont trés facile à obtenir par rapport à ceux de l'entité entière du multiplieur.

Par conséquent, cet outil (simulation en VHDL et vérification de la simulation) utilisant l'approche hiérarchique est de plus en plus important chaque fois l'intégration des circuits augmente , ce qui est le cas dans les centres industriels à fabrication en série des circuits intégrés VLSI.

Tableau des vecteurs de test obtenus par l'algorithme HBTG

trame	G0	NG0	E0	G1	NG1	E1	G2	NG2	E2	G3	NG3	E3	G4	NG4	E4	G5	NG5	E5	RG0	RG1	RG2	RG3	RG4	RG5	Prod
0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
1	R	0	R	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	pp0	X	X	X	X	X	X
2	1	F	R	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	pp0	X	X	X	X	X	X
3	1	0	1	R	0	R	X	X	X	X	X	X	X	X	X	X	X	X	pp0	2pp1	X	X	X	X	X
4	1	0	1	1	F	R	X	X	X	X	X	X	X	X	X	X	X	X	pp0	2pp1	X	X	X	X	X
5	1	0	1	1	0	1	R	0	R	X	X	X	X	X	X	X	X	X	pp0	2pp1	4pp2	X	X	X	X
6	1	0	1	1	0	1	1	F	R	X	X	X	X	X	X	X	X	X	pp0	2pp1	4pp2	X	X	X	X
7	1	0	1	1	0	1	1	0	1	R	0	R	X	X	X	X	X	X	pp0	2pp1	4pp2	8pp3	X	X	X
8	1	0	1	1	0	1	1	0	1	1	F	R	X	X	X	X	X	X	pp0	2pp1	4pp2	8pp3	X	X	X
9	1	0	1	1	0	1	1	0	1	1	0	1	R	0	R	X	X	X	pp0	2pp1	4pp2	8pp3	16pp4	X	X
10	1	0	1	1	0	1	1	0	1	1	0	1	1	F	R	X	X	X	pp0	2pp1	4pp2	8pp3	16pp4	X	X
11	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	R	0	R	pp0	2pp1	4pp2	8pp3	16pp4	32pp5	Sumpp
12	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	F	R	pp0	2pp1	4pp2	8pp3	16pp4	32pp5	Sumpp
13	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	pp0	2pp1	4pp2	8pp3	16pp4	32pp5	Sumpp

ANNEXE

Quelque résultats de multiplication

[35 x 31 ... linstant 0 ns, 35 x54 ... l'instant 100 ns,
20 x 43 ... l'instant 200 ns, 40 x 7 ...l'instant 300 ns
17 x 47 ... l'instant 400 ns]

ns	ck	x	y	pp0	pp1	pp2	pp3	pp4	pp5	reg0	reg1	reg2	reg3	reg4	reg5	prod
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	35	31	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	35	31	35	35	35	35	35	0	0	0	0	0	0	0	0
15	1	35	31	35	35	35	35	35	0	0	0	0	0	0	0	0
21	1	35	31	35	35	35	35	35	0	35	0	0	0	0	0	0
25	1	35	31	35	35	35	35	35	0	35	70	140	0	0	0	0
27	1	35	31	35	35	35	35	35	0	35	70	140	280	0	0	0
29	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	0
30	0	35	31	35	35	35	35	35	0	35	70	140	280	560	0	0
45	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	0
49	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	528
51	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	529
55	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	661
57	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	917
60	0	35	31	35	35	35	35	35	0	35	70	140	280	560	0	917
61	0	35	31	35	35	35	35	35	0	35	70	140	280	560	0	797
65	0	35	31	35	35	35	35	35	0	35	70	140	280	560	0	829
69	0	35	31	35	35	35	35	35	0	35	70	140	280	560	0	573
75	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	573
77	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	61
85	1	35	31	35	35	35	35	35	0	35	70	140	280	560	0	1085
90	0	35	31	35	35	35	35	35	0	35	70	140	280	560	0	1085
100	0	35	54	35	35	35	35	35	0	35	70	140	280	560	0	1085
103	0	35	54	0	35	35	0	35	35	35	70	140	280	560	0	1085
105	1	35	54	0	35	35	0	35	35	35	70	140	280	560	0	1085
111	1	35	54	0	35	35	0	35	35	0	70	140	280	560	0	1085
117	1	35	54	0	35	35	0	35	35	0	70	140	0	560	0	1085
120	0	35	54	0	35	35	0	35	35	0	70	140	0	560	0	1085
121	0	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	1085
135	1	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	1085
141	1	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	114
147	1	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	66

149	1	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	2114
150	0	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	2114
155	0	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	3106
163	0	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	1506
165	1	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	994
171	1	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	866
173	1	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	3938
195	1	35	54	0	35	35	0	35	35	0	70	140	0	560	1120	1890
200	1	20	43	0	35	35	0	35	35	0	70	140	0	560	1120	1890
203	1	20	43	20	20	0	20	0	20	0	70	140	0	560	1120	1890
210	0	20	43	20	20	0	20	0	20	0	70	140	0	560	1120	1890
225	1	20	43	20	20	0	20	0	20	0	70	140	0	560	1120	1890
231	1	20	43	20	20	0	20	0	20	20	70	140	0	560	1120	1890
233	1	20	43	20	20	0	20	0	20	20	40	140	0	560	1120	1890
235	1	20	43	20	20	0	20	0	20	20	40	0	0	560	1120	1890
237	1	20	43	20	20	0	20	0	20	20	40	0	160	560	1120	1890
239	1	20	43	20	20	0	20	0	20	20	40	0	160	0	1120	1890
240	0	20	43	20	20	0	20	0	20	20	40	0	160	0	1120	1890
241	0	20	43	20	20	0	20	0	20	20	40	0	160	0	640	1890
255	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	1890
259	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	1858
261	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	706
263	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	704
265	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	716
267	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	588
269	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	620
270	0	20	43	20	20	0	20	0	20	20	40	0	160	0	640	620
273	0	20	43	20	20	0	20	0	20	20	40	0	160	0	640	636
275	0	20	43	20	20	0	20	0	20	20	40	0	160	0	640	892
277	0	20	43	20	20	0	20	0	20	20	40	0	160	0	640	380
281	0	20	43	20	20	0	20	0	20	20	40	0	160	0	640	348
283	0	20	43	20	20	0	20	0	20	20	40	0	160	0	640	860
285	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	860
291	1	20	43	20	20	0	20	0	20	20	40	0	160	0	640	860
300	1	40	7	20	20	0	20	0	20	20	40	0	160	0	640	860
300	0	40	7	20	20	0	20	0	20	20	40	0	160	0	640	860
303	0	40	7	40	40	40	0	0	0	20	40	0	160	0	640	860
315	1	40	7	40	40	40	0	0	0	20	40	0	160	0	640	860
321	1	40	7	40	40	40	0	0	0	40	40	0	160	0	640	860
323	1	40	7	40	40	40	0	0	0	40	80	0	160	0	640	860
325	1	40	7	40	40	40	0	0	0	40	80	160	160	0	640	860

327	1	40	7	40	40	40	0	0	0	40	80	160	0	0	640	860	
330	0	40	7	40	40	40	0	0	0	40	80	160	0	0	640	860	
331	0	40	7	40	40	40	0	0	0	40	80	160	0	0	0	860	
345	1	40	7	40	40	40	0	0	0	40	80	160	0	0	0	860	
351	1	40	7	40	40	40	0	0	0	40	80	160	0	0	0	216	
353	1	40	7	40	40	40	0	0	0	40	80	160	0	0	0	152	
360	0	40	7	40	40	40	0	0	0	40	80	160	0	0	0	152	
361	0	40	7	40	40	40	0	0	0	40	80	160	0	0	0	24	
369	0	40	7	40	40	40	0	0	0	40	80	160	0	0	0	280	
375	1	40	7	40	40	40	0	0	0	40	80	160	0	0	0	280	
390	0	40	7	40	40	40	0	0	0	40	80	160	0	0	0	280	
400	0	17	47	40	40	40	0	0	0	40	80	160	0	0	0	280	
403	0	17	47	17	17	17	17	0	17	40	80	160	0	0	0	280	
405	1	17	47	17	17	17	17	0	17	40	80	160	0	0	0	280	
411	1	17	47	17	17	17	17	0	17	17	80	160	0	0	0	280	
413	1	17	47	17	17	17	17	0	17	17	34	160	0	0	0	280	
415	1	17	47	17	17	17	17	0	17	17	34	68	0	0	0	280	
417	1	17	47	17	17	17	17	0	17	17	34	68	136	0	0	280	
420	0	17	47	17	17	17	17	0	17	17	34	68	136	0	0	280	
421	0	17	47	17	17	17	17	0	17	17	34	68	136	0	544	280	
435	1	17	47	17	17	17	17	0	17	17	34	68	136	0	544	280	
441	1	17	47	17	17	17	17	0	17	17	34	68	136	0	544	785	
443	1	17	47	17	17	17	17	0	17	17	34	68	136	0	544	787	
445	1	17	47	17	17	17	17	0	17	17	34	68	136	0	544	791	
447	1	17	47	17	17	17	17	0	17	17	34	68	136	0	544	799	
450	0	17	47	17	17	17	17	0	17	17	34	68	136	0	544	799	
465	1	17	47	17	17	17	17	0	17	17	34	68	136	0	544	799	
480	0	17	47	17	17	17	17	0	17	17	34	68	136	0	544	799	
495	1	17	47	17	17	17	17	0	17	17	34	68	136	0	544	799	

BIBLIOGRAPHIE

[1] - R.Airau, J.M Berg,, V.Olive, J.Rouillard,"VHDL du langage à la modélisation" 1990.

[2] -" V-system/PC user's manual software "
version 1.2 (1990-1991)

[3] - Contenson, Cours d'une université française: "introduction au VHDL"

[4] - Contenson, "VHDL par la pratique "

[5] -" VHDL user's manual" -volume 1-Tutorial, Texas instuments
1985.

[6] - M.Jacomino, David, "Sur les mesures de la confiance dans un test"
Revue, TSI vol 8 Numéro 5,1989

[7] - Y.Savaria "Conception et verification des circuits VLSI"
Ecole Polytechnique de Montreal,1988.

[8] -"Un defi: le test des circuits intégrés VLSI"
Ecole polytechnique de Lausane,
les journées d'electronique 1983 .

[9] - C.Gauthron, "Testabilité des circuits intégrés CAE graphic "
micro systemes, 1989

[10] - H.Fujiwara, "Logic testing and design for testability"
The MIT PRESS MASSACHUSZTTS INSTITUTE OF TECHNOLOGY.

[11] - SANATR.RAO, B.YU PAN, JAMES.R "Hierarchical test generator
for the VHDL models" the Bradely Departement of Electrical
Engeenering Virginia polytechnic institute and state
university Blacksburg, IEEE 1993.

[12] - M.Koudil "FO+outil d'insertion automatique de point de test dans les description VHDL" these de magistere,Ecole Polytechnique de Montreal,1988

[13] - "STAFAN an alternative to fault simulation" proceding of the 21 st design automation conference Albuquerque, 1984.

[14] -J.Bernard, "Pratique des circuits logiques" Nouvelle Edition Dunod,