Design for Testability
Seminar zur Projektgruppe Systementwurf

Claas Vieler
red@uni-paderborn.de

Paderborn den 4.11.94

Inhaltsverzeichnis

1 Einleitung

Unter "Design for Testability" faßt man die Methoden und Konzepte zusammen, die die Testbarkeit einer Schaltung erhöhen oder überhaupt erst ermöglichen. Daß das Testen heutzutage eine sehr wichtige Rolle in der Entwicklung und Herstellung digitaler Schaltungen hat, liegt vor allem in der immer stärker wachsenden Dichte integrierter Schaltungen. Formale Korrektheitsbeweise z.B. sind deswegen in der Praxis nicht mehr anwendbar. Mit dem Test versucht man sicherzustellen, daß eine Schaltung (einzelner Chip oder ganze Platine) nicht von der vorgegebenen Spezifikation abweicht. Es wird im weiteren also nur auf "defekte Hardware" (Gatter) getestet, Entwurfs- oder Designfehler können mit Tests natürlich nicht aufgedeckt werden.

1.1 Fehlermodell

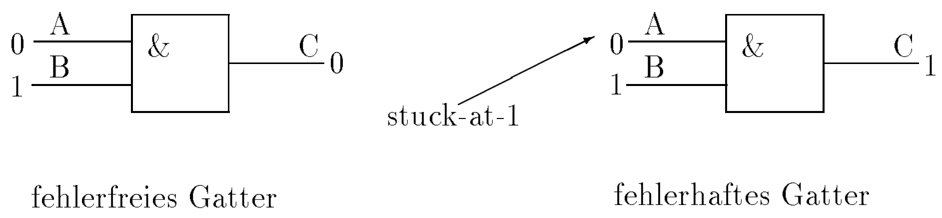


Abbildung 1: Fehlerfreies und fehlerhaftes AND-Gatter

Um mögliche Fehler zu erkennen, benutzt man das Haftfehlermodell. Man geht davon aus, daß eine Leitung fest auf einem der beiden logischen Werte 0 (stuck-at-0) oder (stuck-at-1) liegt. Abbildung 1 zeigt ein logisches AND mit zwei Eingängen A und B. Das fehlerfreie AND (good machine) liefert für die Belegung A=0, B=1 am Ausgang C eine Null ,während das fehlerbehaftete Gatter (fault machine) unabhängig von der Belegung von A immer eine Eins als Ausgangswert hat. Eine Eingangsbelegung mit der man diesen Fehler testen könnte wäre also A=0,B=1, da der Ausgangswert zwischen dem korrekten und dem fehlerhaften Gatter unterschiedlich ist.
Haftfehler entstehen meist durch Kurzschlüsse mit Masse oder Versorgungsspannung sowie durch defekte Leiterbahnen. Betrachtet man die Anzahl der möglichen Fehler bei einer Schaltung mit N Leitungen, so existieren insgesamt 2 mal N Fehler. Wenn man nun mehrere Fehler gleichzeitig zuläßt, erhöht sich diese Zahl auf $3^n - 1$. In der Praxis sind diese *Mehrfachfehler* wegen dem exponentiellen Wachstum nicht mehr abzudecken, daher beschränkt man sich auf *Einfachfehler*. Das Einfachfehlermodell ist im allgemeinen für die praktischen Tests ausreichend. In einigen seltenen Fällen kann es allerdings vorkommen, daß sich mehrere Stuck-At-Fehler auslöschen, so daß sie nicht mehr erkannt werden können (hart-to-detect-faults).

1.2 Grundlegende Konzepte des Testens

Um nun das AND-Gatter aus Abbildung 1 zu testen muß man gewisse Eingangsbelegungen, im weiteren Testmuster genannt, anlegen, das Ergebnis "abgreifen" und mit den Ausgaben eines fehlerfreien Gatters vergleichen. Sind beide Ausgangswerte gleich, so ist das getestete Gatter fehlerfrei bzgl. des getesteten Stuck-at-Fehlers. Diese Belegungen mit den man fehlerhafte Gatter erkennen kann heißen *fehlerinjizierende Belegungen*. Die beiden wichtigsten Prinzipien zum Testen eines Objekts sind also die *Steuerbarkeit der Eingänge (Controbality)* und die *Beobachtbarkeit der Ausgänge (Observability)*. Für ein einzelnes Gatter sind diese fehleraufdeckenden Testmuster sehr einfach zu konstruieren. Ist das zu testende Objekt aber in einer komplexeren Schaltung eingebettet, so müssen die primären Eingänge so belegt werden, daß an den Eingängen des Testgatters eine fehlerinjizierende Belegung anliegt. Zusätzlich muß der Wert des Gatterausgangs an einem primären Ausgang zu erkennen sein, man nennt dies *fehlerleitend*. Das Problem des Tests einer Schaltung kann auch so betrachtet werden, daß man versucht, durch die Schaltung Wege zu finden, durch welche die Änderung eines Eingangssignales in Form einer Signaländerung an einem Ausgang beobachtet werden kann. Einen solchen Pfad nennt man einen *kritischen Signalweg (sensitized path)*. Mit einem Test, der die Eingänge so belegt, daß ein *kritischer Signalweg* entsteht, testet man die Leitungen, die sich auf diesem Pfad befinden und im fehlerfreien Fall Null sind, auf stuck-at-1, und alle Knoten, die auf Eins liegen sollen auf stuck-at-0. Um eine Schaltung vollständig zu testen, muß man alle Leitungen auf alle Fehler (stuck-at-0, stuck-at-1) testen.

2 Testmustergenerierung

2.1 D-Algorithmus

Der sogenannte *D-Alorithmus* bietet eine konstruktive Möglichkeit zur Erzeugung von Eingangsbelegungen. Er benutzt das Prinzip der *kritischen Pfade* und ist das grundlegende Verfahren zur Bestimmung von Testmustern für kombinatorische Schaltungen. Der Algorithmus erzeugt für jeden möglich Haftfehler einer Schaltung ein Testmuster, bei N Leitungen also $2N$ Testmuster. Für jeden Fehler werden die drei folgenden Schritte durchlaufen:

1. Auf der zu testenden Leitung wird ein Wert erzeugt, der dem angenommenen Fehler entgegengesetzt ist. Ist dies nur durch genau eine Belegung der Eingänge des Gatters möglich, so spricht man von einer *Implikation*. Es werden nun alle Rückwärtsimplikationen zu den primären Eingängen bestimmt. Dabei auftretende nichtdefinierte Variablen werden in einer Schlange gespeichert.

2. Nun muß eine fehlerleitende Belegung gefunden werden, so daß der evtl. Fehler an einem der Ausgänge sichtbar wird. Vorwärtsimplikationen sind ebenfalls in einer Schlange zu speichern und später abzuarbeiten. Erzwungene Rückwärtsimplikationen sind sofort durchzuführen und die undefinierten Eingangsvariablen sind ständig zu aktualisieren.

3. Nun muß ein Testmuster so erzeugt werden, daß die Bedingungen für 1. und 2. erfüllt sind. Es müssen dazu alle undefinierten Variablen belegt werden, bis alle primären Eingänge eindeutig bestimmt sind. Bei Widersprüchen in der Belegung ist ein Backtracking durchzuführen. Dieses wird auch Konsistenzoperation genannt.

Um den Ablauf des D-Algorithmus deutlich zu machen folgt nun ein kleines Beispiel.

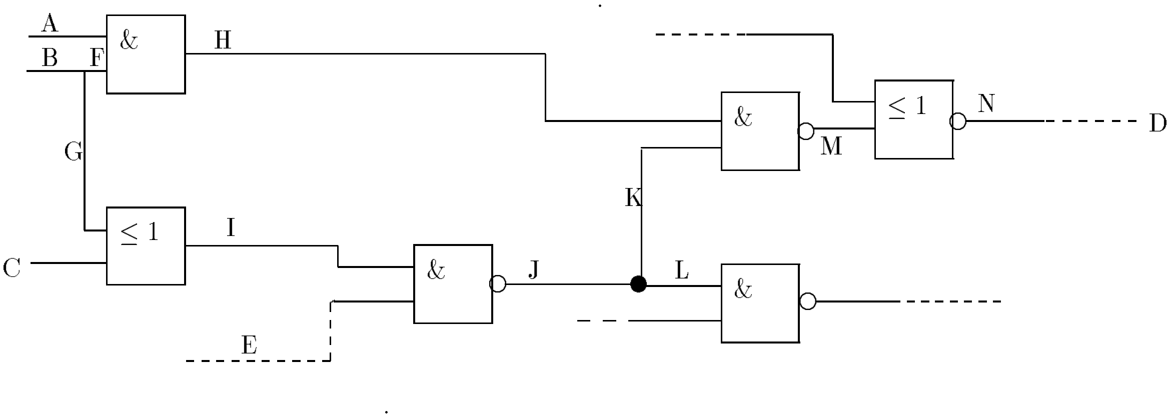


Abbildung 2: Vorwärts und Rückwärtsimplikation

In dem Beispiel in Abbildung 2. wird ein stuck-at-1-Fehler auf Leitung N angenommen. Es wird also M=1 gesetzt, um auf N eine Null zu "injizieren". Dies bewirkt die Rückwärtimplikation H=1 und K=1. Aus H=1 folgt A=1 und F=1. Aus F=1 folgt B=1, allerdings speist B auch die Verbindung G, so daß G=1 für eine Vorwärtsprüfung vorzusehen ist.
Aus K=1 folgt J=1 und L=1, welches ebenfalls für eine Vorwärtsprüfung vorzusehen ist. J=1 führt zu keiner Implikation, wird daher als undefiniert in die Liste eingetragen. Alle Rückwärtprüfungen sind nun abgearbeitet, so daß nun Vorwärtprüfungen gestartet werden können. Aus G=1 folgt nun I=1. Da nun ein Eingang des NAND-Gliedes auf Eins steht, und der Ausgang J=1 vorgegeben ist, folgt daraus E=0. Mit E würden jetzt weitere Rückwärtprüfungen verbunden sein (kann aber wegen fehlender Schaltung hier nicht durchgeführt werden). J=1 ist nun definiert und kann aus der Liste der undefinierten Leitungen entfernt werden. Als letzte muß man noch L=1 nach vorne prüfen.
Der Algorithmus ist in der Praxis sehr komplex, und dieses Beispiel zeigt nur eine kleinen Ausschnitt. Für eine ausführliche Darstellung sind [8] und [9] zu empfehlen.

2.2 weitere Algorithmen & Optimierungen

Der D-Algorithmus löst generell das Problem der Testmustergenerierung für kombinatorische Schaltungen, allerdings treten hierbei Effizienzprobleme auf. Eine Verbesserung des D-Algorithmus ist der *PODEM-Algorithmus (path oriented decision making)*. Er beschränkt die Wertzuweisung ausschließlich auf primäre Eingänge, so daß sich auch das Backtracking nur auf diese beziehen kann. Ein weitere Methode ist der sogenannte *LASAR-Alorithmus (logical automated stimulus and response)*. Dieser berechnet möglichst lange kritische Pfade von einem Eingang zu einem primären Ausgang. Man versucht dann diesen Pfad in die anderen Berechungen einzubeziehen. Eine weitere Möglichkeit zu optimieren ist das Zusammenfassen von Fehlern mit gleichen Auswirkungen, z.B. stuck-at-0-Fehler auf einem Eingang eines AND-Gatters und ein stuck-at-0-Fehler auf dem Ausgang des gleichen Gatters.

3 Ad-hoc Ansätze

Ad-Hoc bedeutet, das man diese Methoden auf eine schon erstelltes Objekt (Schaltung,Platine, Chip) anwenden kann, im Gegensatz zu den strukturierten Ansätzen, die man schon bei Entwurf einer Schaltung anwenden muß. Ad-Hoc Ansätze sind also ganz spezifisch auf eine bestimmte Schaltung zugeschnitten.

3.1 Testpunkte

Die einfachste Möglichkeit, die Testbarkeit zu erhöhen ist das Einfügen von Testpunkten, an denen man Werte "abgreifen" bzw. "eingeben" kann. Dies wird meistens mit einem zusätzlichen Pin ermöglicht, der aus der Platine herrausschaut. Oder auch mehrere Pins gleichzeitig (bed of nails) sind möglich, allerdings nur wenn die zu testende Platine nur einseitig bestückt ist und nicht in einem IC integriert ist.

3.2 Degating

Im Gegensatz zu den vorherigen Methoden, mit denen einzelne Knoten zur Beobachtung und Steuerung herausgeführt werden, trennt das sogenannte *Degating* mehrere Module voneinander. Dies wird durch zusätzliche Logik (degating logic) erreicht. Dadurch sind die einzelnen Module unabhängig testbar und dies senkt gleichzeitig den Aufwand für die Testmustergenerierung. Der prinzipielle Aufbau des *Degatings* wird in Abb. 3 gezeigt.

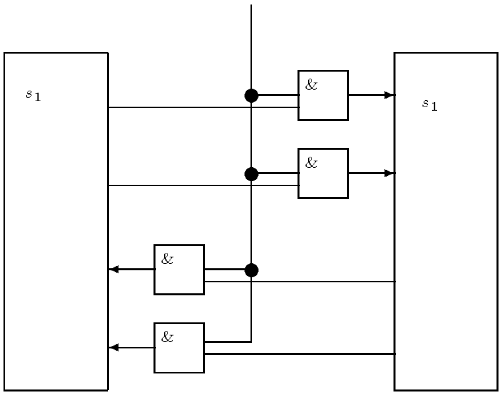


Abbildung 3: Partionieren durch *Degating*

Die Steuerleitung ist im normalen Betrieb auf eins gesetzt und wird nur beim Testen getoggelt um die Module zu separieren.

4 Strukturierte Ansätze

Die folgenden Kapitel beschreiben Methoden, die man schon beim Entwurf einer Schaltung anwenden bzw. einbeziehen muß. Die systematischen Designkonzepte bauen fast alle auf dem Konzept auf, daß wenn alle Register, Speicher u.ä. einer Schaltung kontrolliert werden können (Controllability, Observability), sich die Testmustergenerierung und Fehlersimulation nur noch auf kombinatorische Netzwerke bezieht.

4.1 Scan Path

Der *Scan-Path (Prüfbus)* ist ein in der Praxis sehr häufig eingesetztes Verfahren zur Verbesserung der Testbarkeit sequentieller Schaltungen, es kann auch für Partitionierungszwecke verwendet werden. Der *Scan-Path* ist ein serieller Pfad

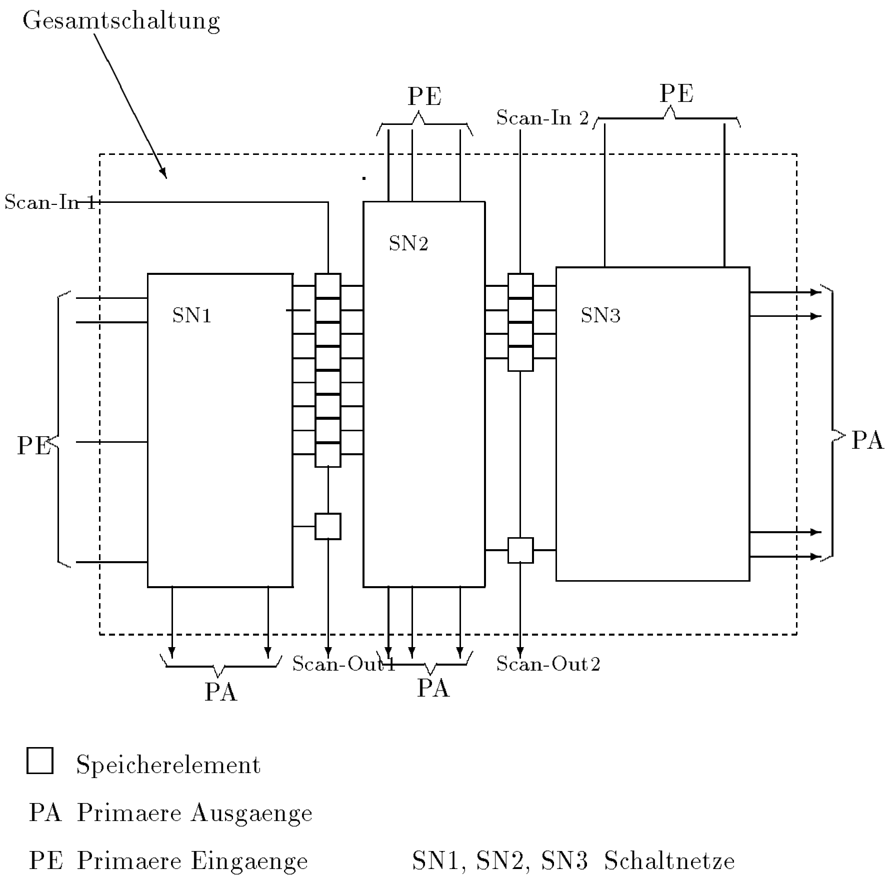


Abbildung 4: Allg. Aufbau des *Scan-Path*

auf internen Knoten einer Schaltung, der alle Speicherelemente, sowohl Flipflops als auch Register umfaßt, und durch zusätzliche Logik zu einem Schieberegister zusammengeschaltet ist. Das Schieberegister wird jedoch nur beim Testen aktiv, im normalen Betrieb sind alle Elemente des Scan-Path physikalisch getrennt. Durch den Scan-Path können nun alle Speicherelemente beobachtet und kontrolliert werden. Im Schiebebetrieb können über die seriellen Verbindungen der Schieberegister Daten in die internen Speicherelemente geladen werden, bzw. die Zustände der Speicherelemente ausgelesen werden. Durch diesen Scan-Path entsteht eine Schaltungsstruktur, die aus einzelnen Modulen besteht, die durch Elemente des Schieberegisters untereinander verbunden sind. Die Ein- und Ausgänge des oder der Scan-Pfade sind von außerhalb der Schaltung zu erreichen, sie können genauso angesprochen werden wie die primären Ein- und Ausgänge der einzelnen Module. In den einzelnen Modulen sind keine weiteren Speicherelemente vorhanden, so daß sich die Testproblematik auf den Test reiner Schaltnetze reduziert. Es werden nun alle Module nacheinander getestet. Der Test eine Moduls läuft folgendermaßen ab:

1. Die primären Eingänge der zu testenden Schaltung werden mit Testmustern belegt. In den Prüfbus werden die Eingangsdaten "eingeschoben", dabei wird vom normalen Betrieb in den Schiebebetrieb umgeschaltet.

2. Die Schaltung wird in den Normalbetrieb umgeschaltet, und ein Takt ausgeführt.

3. Der Prüfbus wird (durch Umschalten in den Testbetrieb) ausgelesen. Zusammen mit den primären Ausgängen wird das Ergebnis von einer unabhängigen Einheit, evtl. Software verglichen mit den Testdaten einer fehlerfreihen Schaltung.

Man kann beim "rausschieben" der Ergebnisse gleichzeitig schon neue Eingaben auf den Prüfbus legen. Durch Zusammenschalten aller Speicherelemente einer Schaltung in ein oder mehrere Schieberegisterketten werden diese zu von außen erreichbaren Knoten. Der Vorteil dieser Technik besteht darin, daß wenige Zusatzanschlüsse benötigt werden, allerdings wird zusätzliche Logik gebraucht, um die Speicherelemente der Schieberegister untereinander zu verbinden. Man kann evtl. auch noch zusätzliche Speicherelemente einfügen, um sehr schlecht testbare Knoten besser zu erreichen. Je nach Art der Speicherelemente braucht man zusätzliche Takte, es wird mind. noch eine Steuerleitung benötigt, die von Testbetrieb auf Normalbetrieb umschaltet. Der prinzipielle Aufbau eines Scan-Path mit drei Teilschaltungen und zwei Pfaden ist in Abbildung 4. dargestellt.

4.2 Level Sensitive Scan Design (LSSD)

Das *LSSD (Level Sensitive Scan Design)* ist eine Weiterentwicklung des Prüfbusses. Bei dieser Entwurfsmethode wird eine laufzeitunabhängige und synchrone Arbeitsweise der Schaltung vorrausgesetzt, daher der Name *Level Sensitive Design*. Eine Schaltung heißt *laufzeitunabhängig*, wenn sie bei Änderung der Eingangssignale unabhängig von Anstiegs- und Abstiegszeiten ein definiertes Verhalten zeigt, wie z.B. ein zustandsgesteuertes Vorspeicherflipflop. Das LSSD-Prinzip ist in Abbildung 5. dargestellt. Man hat hier den Scan-Path, der durch die Schaltung läuft mit Vorspeicher-Flipflops realisiert. Die Ausgänge jedes Flipflops, das aus jeweils zwei Teilschaltungen L_1 und L_2 besteht, sind mit den Eingängen des Nächsten verbunden. A und B sind die Takte, die den Scan-Path steuern, sie müssen sich überlappen. C ist der Takt, der im Normalbetrieb die Schaltung steuert. Die Entkopplung des Ein- und Ausgangs eines Speicherelementes ist wichtig, da die alten Daten des Flipflops als Testmuster für den nächsten Durchlauf benutzt werden. Die Ergebnisse werden dann sofort wieder in den L_1-Flipflops gespeichert. Dadurch, daß sich alle Speicher und Register durch den Scan-Path steuern und kontrollieren lassen, vereinfacht sich das Testproblem wieder auf ein kombinatorisches Netzwerk. Durch das Entkoppeln von Ein- und Ausgängen und die Laufzeitunabhängigkeit kann man sogar "Rückkopplungen" zulassen. Der Testablauf ist fast analog zu dem des Scan-Path. Über den Prüfbus werden Testmuster "eingeschoben" , genau ein Takt im Normalbetrieb ausgeführt und danach alle Daten wieder "rausgeschoben" und ausgewertet.

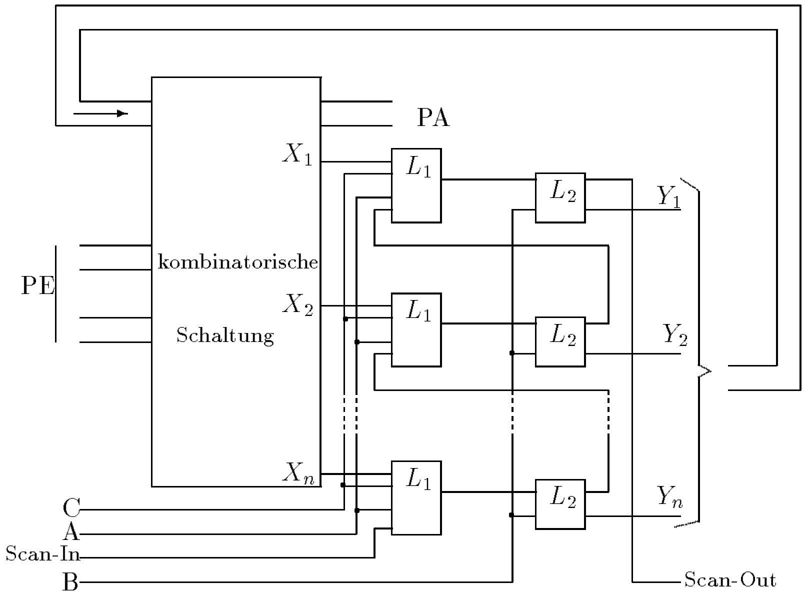


Abbildung 5: *LSSD*-Prinzip

Vorteile sind genau wie beim Prüfbus wenige zusätzliche externe Anschlüsse sowie wenig Zusatzlogik. Man muß jedoch beachten, daß das hinein- und herausschieben der Testdaten jeweils einen Takt pro Bit des Schieberegisters benötigt.

4.3 BISTs (Build-In Self-Tests)

Der Vorteil von eingebauten Selbsttests ist offensichtlich. Man benötigt keine externe Einheit, die die Testmuster bereitstellt und nach der Fehlersimulation die Ergebnisse analysiert. Diese Daten müssen nicht mehr durch die verschiedenen Layer (Ebenen) durchgereicht werden, dadurch werden weniger externe Anschlüsse benötigt, das spart Platz und Zeit. Man muß der Schaltung nur noch durch ein Signal mitteilen, daß sie sich selber testen soll. Das Ergebnis lautet dann "Schaltung OK (GO)" oder "Schaltung nicht OK (NO GO)". Im Allgemeinen kann der Fehler nicht genau lokalisiert werden.

4.3.1 Allgemeiner Aufbau eines hierachischen BIST

Die meisten Schaltungen sind hierachisch aufgebaut, so auch der häufig benutzte und verbreitete hierachische BIST. Der allgemeine Aufbau eines hierachischen BIST ist in Abbildung 6. zu sehen. Der höchste Testmanager erhält das Testsignal und stößt seine "Söhne" zu einem Selbstest an. Er erhält ein *go/no-go Signal*. Der hierachisch unterste Testmanager testet die Schaltung (Circuit under test) durch. Er steuert den Testmustergenerator (pattern generator), der die Daten auf eine bestimmte Weise bereitstellt und erhält von der Einheit, die die Testdaten auswertet (response analyser), das Ergebnis der Tests. *Pattern generation* und *response analyse* wird in den nächsten Kapiteln genauer beschrieben.

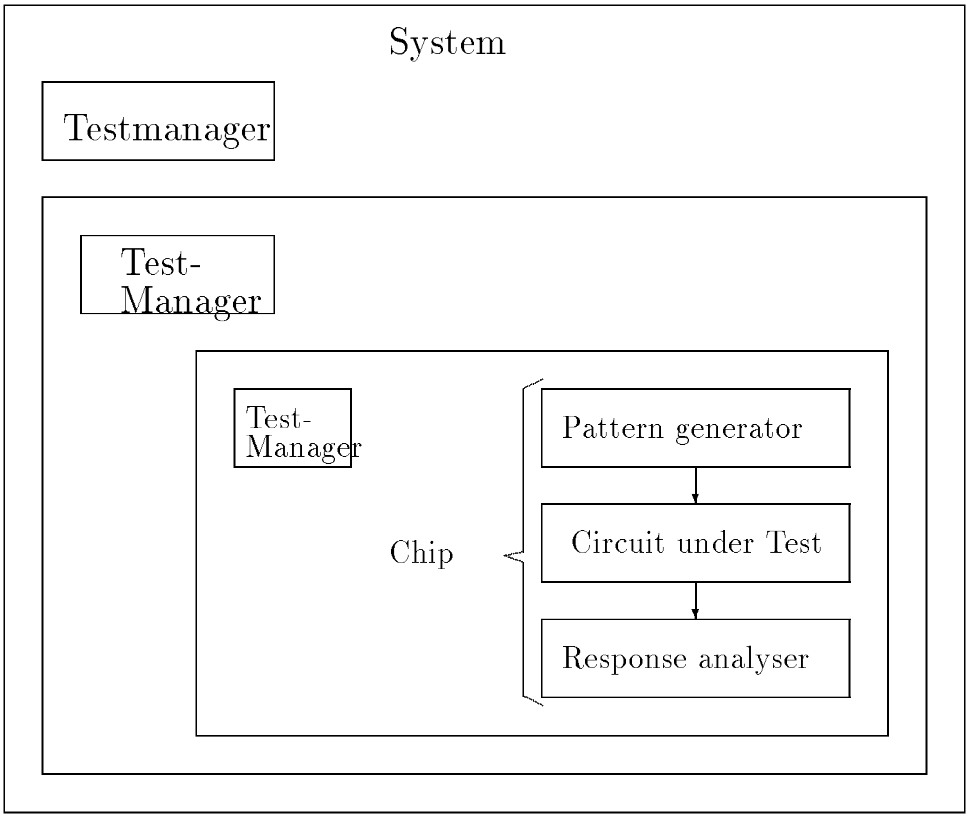


Abbildung 6: Allg. Aufbau eines hierachischen *BIST*

Durch den hierachischen Aufbau läßt sich ein Fehler sehr gut erkennen. Ist eine Schaltung fehlerhaft, so gibt der aktuelle Testmanager dieses Ergebnis eine Stufe in der Hierachie nach oben. Der oberste Testmanager kann dann genau bestimmen welcher Baustein bzw. Chip fehlerhaft ist. Dieser kann dann z.B. bei modularer Fertigung sehr einfach ausgetauscht werden.

4.3.2 Bereitstellung von Testmustern

Das bereitstellen von Testmustern (*pattern generation*) kann auf vielfältige Weise geschehen, man unterscheidet zwischen den verschiedenen Arten von Testmustern:

- *Stored Pattern*
  Hier werden die Testmuster einfach in einem Read-Only-Memory (ROM) gespeichert, das ist zwar die einfachste Möglichkeit, benötigt aber bei großer minimaler Testmenge sehr große ROMs, was wiederum sehr viel Platz auf der Platine benötigt.

- *Exhaustive or pseudoexhaustive Pattern*
  Bei dieser Methode werden einfach alle Eingabemöglichkeiten angelegt und getestet. Dabei erreicht man eine sehr hohe Fehlerabdeckung, allerdings sind hier Grenzen gesetzt durch die extrem hohe Anzahl von Testmustern. Bei mehr als 25 Eingängen ist dies sehr unpraktikabel, dann muß man das sogenannte *pseudoexhaustive Pattern Generation* benutzen, das die Schaltung vorher partitioniert bzw. aufteilt. Dabei können sich evtl. Segmente überlappen. Erzeugt werden solche Testmuster üblicherweise mit Zählern.

- *Pseudorandom Pattern*
  Pseudorandom Pattern sind wie der Name schon sagt zufällig erzeugte Muster, die Wahrscheinlichkeit von Null und Eins ist genau gleich. Die

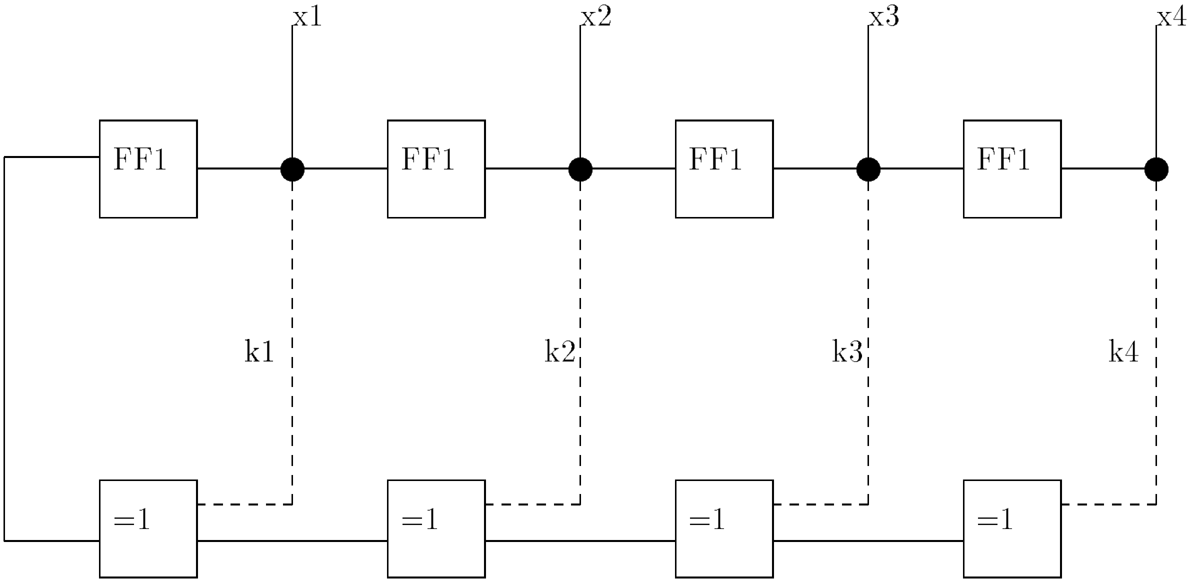


Abbildung 7: Pseudozufallszahlengenerator

Bereitstellung dieses Muster erfolgt mit einem sogenannten Pseudozufallsgenerator (PZG). Die von einem PZG erzeugten Datenfolgen haben den

Charakter von Zufallsfolgen, die sich allerdings nach einer gewissen Zeit wiederholen. Die Abbildung 7. zeigt einen PZG, der aus einem Schieberegister besteht, das mit Antivalenzgliedern rückgekoppelt ist. Wenn man die k_i als Koeffizienten eines Polynoms betrachtet, so kann man das Verhalten des PZG durch algebraische Methoden beschreiben. Diese Sichtweise wird auch später bei der Signaturanalyse benutzt. Durch die geschickte Wahl der Anfangsbelegung und der Koeffizienten kann man die erzeugten Zahlen und damit die Fehler, die dadurch erkannt werden beeinflussen.

- *Weighted pseudorandom Pattern*
  Bei den gewichteten Pseudorandom Pattern ist die Wahrscheinlichkeit von Nullen und Einsen unterschiedlich. Mit diesen Mustern lassen sich die sog. *Hart-to-detect-Faults* sehr gut und effektiv erkennen. Hierbei müssen die Wahrscheinlichkeiten aber zusätzlich in einem ROM gespeichert werden.

4.3.3 Auswerten der Testdaten (response analysis)

Da beim Analysieren einer Schaltung gegebenenfalls sehr große Datenmengen anfallen, versucht man diese durch Kompaktifizierung zu verkleinern. Bei der Kompaktifizierung werden die Daten reduziert, es gehen also Informationen verloren. Die Ausgaben eines CUT werden durch eine Funktion kompaktifiziert, und mit den ebenfalls reduzierten Daten einer fehlerfreien Schaltung verglichen. Sind diese Sequenzen beim fehlerhaften und beim fehlerfreien Schaltkreis gleich, so kann der Fehler nicht erkannt werden. Man nennt dies *Aliasing*. Die gebräuchlichsten Kompaktifizierungsfunktionen sind im folgenden aufgeführt:

- *transition count*
  Es wird die Anzahl der übergänge von Null und Eins und umgekehrt gezählt.
- *syndrom or 1's counting function*
  Es werden nur die Anzahl der Einsen gezählt.
- *signature analyse*
  Siehe unten.

Es wird leicht klar, daß die Fehlererkennung bei den ersten beiden Funktionen nicht besonders hoch liegt. Hinzu kommt noch, daß die Datenreduktion je nach auftretenden Bitstrings sehr gering sein kann. Die wichtigste und in der Praxis meistgenutzte Funktion ist daher die Signaturanalyse, die im nächsten Kapitel beschrieben wird.

4.3.4 Signatur-Analyse

Die Signaturanalyse ist die wichtigste und effizienteste Methode Daten zu kompaktifizieren. Die Berechnung erfolgt durch ein sogenanntes Signaturanalyseregister. Diese Schaltung dividiert den Datenstrom D durch ein Polynom p(x) (mod 2). Nur der Divisionsrest ist hier von Bedeutung, das eigentliche Ergebnis wird nicht weiter beachtet. Das Polynom p(x) ist durch die Rückkopplungen

(Koeffizienten) gegeben. Analog zu dem PZG kann man hier wieder Methoden aus der Algebra anwenden, um das Verhalten der Schaltung zu bestimmen. Das grundlegende Konzept kann man unter [4] LFSR-Theory (linear feedback shift register) nachlesen, genauere und weiterreichende Analysen und Methoden sind unter [7] zu finden. Der Division läuft folgendermaßen ab: Der Datenstrom D wird in das Register "eingeschoben". Der Quotient, also das eigentliche Ergebnis wird am letzten Flipflop "rausgeschoben", aber nicht mehr weiter betrachtet. Der Divisionsrest R der Division steht jetzt im Register und kann jetzt ausgelesen und mit der Signatur einer fehlerfreien Schaltung verglichen werden.
Bei diesem Verfahren werden alle Fehler erkannt, wenn die Sequenzlänge klei-

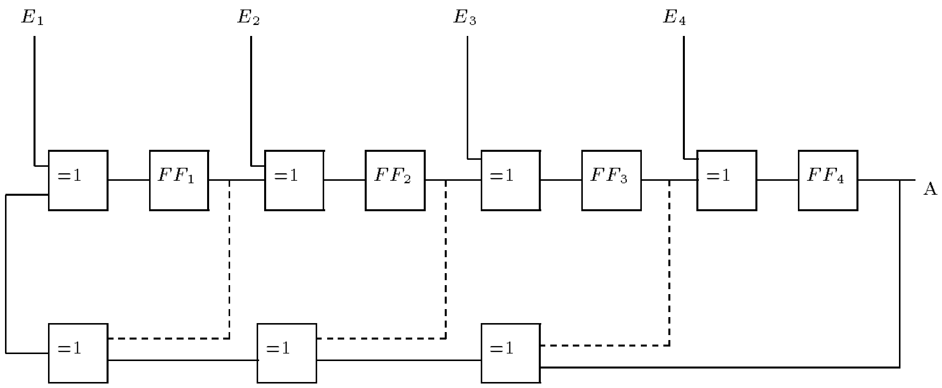


Abbildung 8: paralelles *Signaturanalyseregister*

ner als die Länge des SR ist. Dies ist sofort einzusehen, da in diesem Fall keine Datenreduktion stattfindet. Für längere Bitstrings hängt die Fehlererkennungsrate nur von der Breite des SR ab: $F \approx 1 - 2^{-n}$ wobei n die Bitbreite des SR. Bei einem 16-Bit SR liegt diese Rate bei ca. 99,9985 %.
Mit einem parallelen Signaturanalyseregister kann man nun gleichzeitig mehrere Datenströme gleichzeitig "signieren". Ein solches paralleles SR ist in Abbildung 8. zu sehen, durch die Eingänge E_i werden die Datenströme D_i eingeschoben. Der Informationsverlust, der durch die Zusammenfassung der Signaturen entsteht wird aber wegen der hohen Fehlererkennungswahrscheinlichkeit und der Verringerung des Aufwandes in Kauf genommen. Die Fehlererkennung ist auch im parallelen Fall nur abhängig von der Bitbreite des Registers.

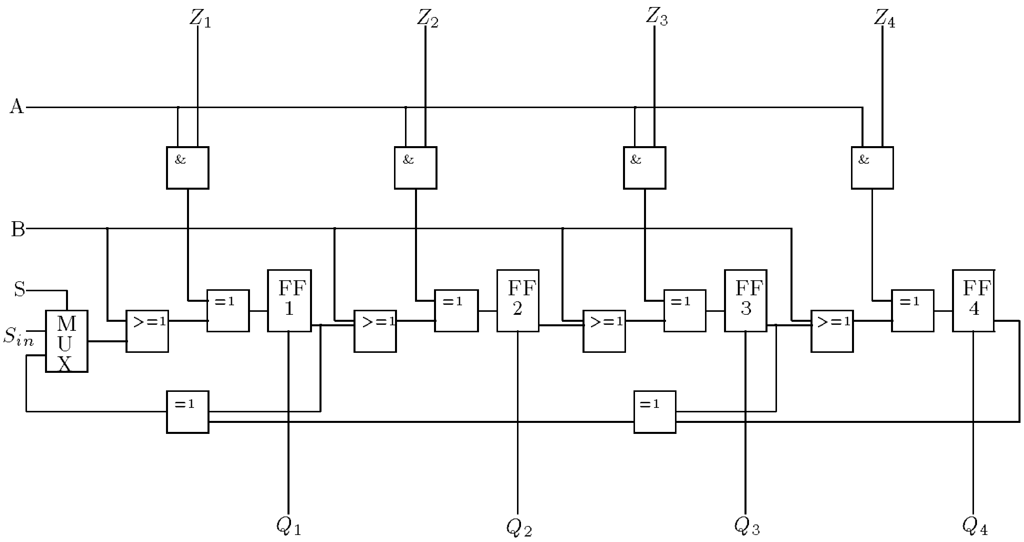
4.3.5 Build-in Logic Block Observation (BILBO)

Der BILBO ist nun das erste Beispiel für ein BIST, in dem viele Methoden aus den vorherigen Kapiteln eingehen. Der BILBO testet nur die reine Funktionsfähigkeit einer Schaltung (go/no-go). Ein solches BILBO-Register (Abbildung 9.) mit vier Bit ist nicht nur als paralleles Register sondern auch noch als Pseudozufallszahlengenerator bzw. Signaturanalyseregister einzusetzen, außerdem verfügt es zusätzlich über einen Scan-Path. Die Funktion wird über die Leitungen A und B sowie die Selektvariable S des Multiplexers gesteuert. Die einzelnen Betriebsarten des BILBO werden im folgenden dargestellt.

- *Normalbetrieb*
  Durch A=B=1 werden die an den Eingängen Z_i liegenden Werte auf die Flipflops geschaltet. Die seriellen Verbindungen der FF untereinander sind unterbrochen und die Werte der FF liegen an den Ausgängen Q_i an.

- *Schieberegister*
  A=0, B=0: Die Eingänge Z_i sind abgetrennt, die seriellen Verbindungen der Flipflops untereinander sind aktiv. Die Steuervariable S wird auf Eins gesetzt, so daß die Eingaben für das erste FF über den Scan-Path-Eingang S_{in} kommen. Der Ausgang des vierten Fliplflops wird zum Ausgang des Scan-Pfades.

- *Pseudozufallszahlengenerator*
  A=B=0, so daß die Eingänge abgetrennt und die FF untereinander verbunden sind. Allerdings wird jetzt die Steuervariable des Multiplexers auf Null gesetzt, um die Rückkopplungen an den Eingang des ersten FF zu legen.



A	B	S	Funktion
1	1	-	Normalbetrieb als paralleles Register
0	0	1	Betrieb als Schieberegister
0	0	0	Betrieb als Pseudozufallsgenerator
1	0	0	Betrieb als Signaturanalyseregister

Abbildung 9: *BILBO*

- *Signaturanalyseregister*
  Das Signaturanalyseregister arbeitet genau wie der PZG, nur müssen die Datenströme an die Eingänge der FF gelangen. Daher muß Steuervariable A auf Eins gesetzt werden.

Der Einsatz eines BILBO setzt vorraus, daß die Schaltung genau wie beim Scan-Path in Speicherelemente und dazwischenliegende Schaltnetze aufgeteilt ist. Diese Prüfbusse werden dann durch jeweils ein BILBO-Register gebildet. Ist dies nicht möglich, so muß man Verbindungen auftrennen. Der allgemeine Aufbau einer Schaltung mit drei BILBOs ist in Abbildung 10. erläutert.
Der Test der Schaltung läuft so ab, daß jeweils ein BILBO als Pseudozufallszahlengenerator geschaltet wird und das zu Testende Schaltnetz mit Testmustern versorgt, während ein anderer BILBO als paralleles Signaturanalyseregister arbeitet. Man muß vor dem Testbeginn im ersten BILBO die Anfangsbelegung des Register über den Scan-Path einschieben. Am Ende die Signatur aus dem zweiten BILBO herrausschieben und analysieren.

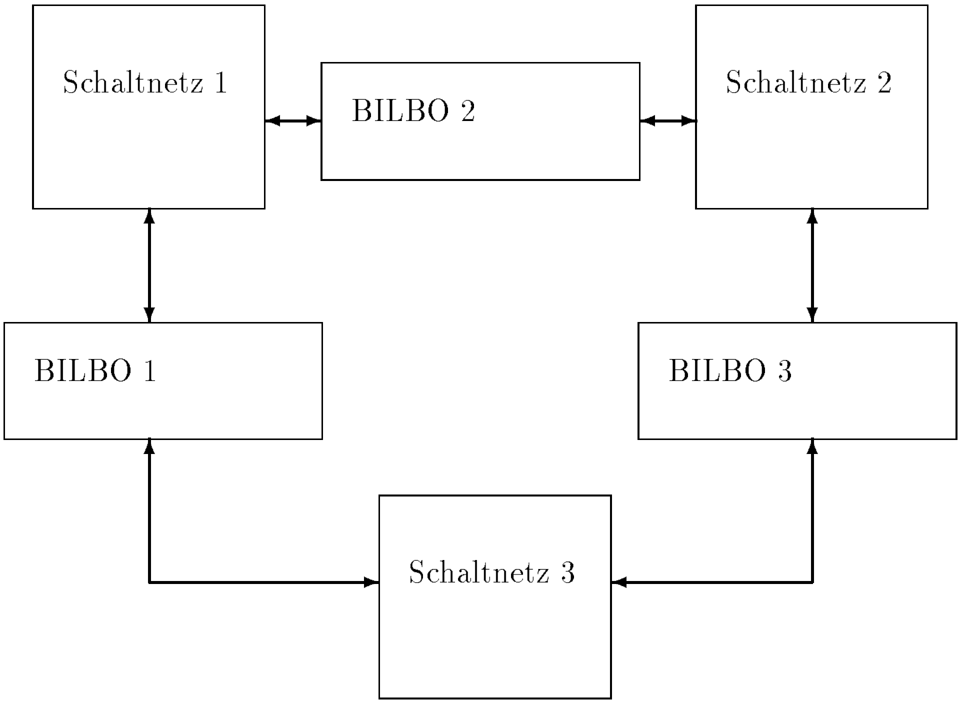


Abbildung 10: Schaltung mit drei *BILBO*-Registern

Der Reihe nach vertauschen die Register ihre Funktion, bis alle Teilnetze getestet sind. Ein Bilbo kann *nicht* gleichzeitig als Signaturanalyseregister und als Pseudozufallszahlengenerator arbeiten. Es ist also nicht möglich z.B. BILBO1 als PZG für Netz 1 und gleichzeitig als SR für Netz 3 zu betreiben, was von BILBO2 mit Testvektoren gespeist wird.
Es gibt Ansätze (circular BISTs) bei denen solche Kombinationen möglich sind, weitere Literatur findet man unter [5].

Literatur

[1] Thomas W. Williams and K.P. Parker. *Design for Testability – a survey. IEEE Transactions on Computers, C31(1):2 -15, January 1982.*

[2] H.Wojtkowiak. *Test und Testbarkeit digitaler Schaltungen. Leitfäden und Monographien der Informatik. B.G.Teubner Stuttgart, 1988.*

[3] F.J.Rammig. *Systematischer Entwurf digitaler Systeme. Leitfäden und Monographien der Informatik. B.G.Teubner Stuttgart, 1989.*

[4] Charles R.Kime and Kewal K.Saluja. *A tutorial on Build-In Self-Test. Part 1: Principles. IEEE Design and Test of Computers, March 1993.*

[5] Charles R.Kime and Kewal K.Saluja. *A tutorial on Build-In Self-Test. Part 2: Applications. IEEE Design and Test of Computers, June 1993.*

[6] S.W. Golomb. *Shift Register Sequences, Aegean Park Press, Laguna Hills, Calif, 1982*

[7] W.W.Peterson and E.J. Weldon. *Error-Correcting Codes, John Wiley & sons, New York, 1972*

[8] P.J. Roth. *Diagnosis od Automata Failures: A Calculus and a Method, IBM Journal, Juli 1966, pp.278-291*

[9] P.J.Roth, Bouricius, W.G.,Schneider, P.R.. *Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Cuircits, IEEE Transactions on Electronic Computers, Vol. EC-16, no,5,October*