ドントケア抽出技術を用いたLFSコーディングの効率化

日大生産工(院) ○宇佐美 龍哉
日大生産工 細川 利典

１．はじめに

近年の大規模集積回路(LSI)のテストにおいては，従来挙げられてきた故障の検出率に関する問題だけでなく，そのテストにかかるコストの削減が重要である．LSI のテスト時のコストに直接影響してくる要素として，一般的にテスト実行時間，テストに必要なテストパターンのデータ，テスト時にかかる消費電力などが挙げられる．
しかし最近では，前述のコストの問題だけでなく，LSI の高レベルな微細化，集積化により，設計上の制約が生まれている．その制約により満足いくようなテストが行えない事態が発生している．

本稿ではこのような問題のうち，回路の入力ビット幅を減少させ，テストを効率よく行えるようにすることを考える．

具体的には LFSR の論理を用いて入力するテスト系列を圧縮，符号化し，回路内に付加した展開器によって，本来回路に対して印加したいテスト系列を入力する手法を提案し，さらにより小さなビット幅での入力を可能にするため、改善する手法について検討する．

２．入力ビット幅の問題と解決方法

LSIの設計手法の一つにSystem-on-Chip(SoC)がある．SoCとは，1つの半導体チップ上に必要とされる機能を集積する集積回路・LSIの設計手法である．それぞれの機能は，コアという単位でチップ上に配置されており，図1のようなイメージで表される．

SoC のテストにおいては，各コアの入力に，単体テストのパターンを入力して，テストを行う．しかし，コアの入出力が SoC 自体の外部入出力の数を上回ってしまうと，完全なテストが困難となってしまう．図 1 の例だと，B,D,E のコアが直接外部入力による単体テストが困難である．

解決する方法に，コアの入出力にスキャンフリップフロップを配置し，任意の入力を与えるようにするバウンダリースキャンという手法がある．しかし，これには回路規模が膨大になればなるほどテスト実行時間が増大してしまうなどの問題がある．

本稿で提案する LFS コーディング(1 は，ビット幅減少のための難易度が回路規模に依存せず，テストパターン中のケアビットにのみ依存し，ケアビット数＋20 程度の入力で符号化が可能である．そのため非常に有用であると考えられる．一般的にテストパターン中のケアビットはそう多くは存在しないため，LFS コーディングは非常に効果があると考えられる．

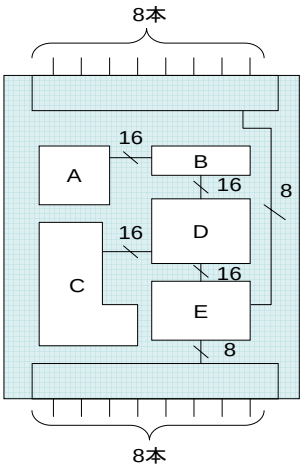

図 1:SoC の構成例

３．LFS 符号化の論理

本章では，提案する LFS 符号化の論理について述べる．LFSとはLinear Feedback Shiftの略で，線形フィードバックシフトレジスタの持つ論理を利用している．具体的に述べると，N ビットの LFSR に M ビットのシフトチェインが接続されている回路モデルを考える． 図 2 には LFS コーディングに使用するLFSR論理モデルの例を示す．この図では 4 ビット LFSR と 6 ビットシフトチェインで構成されている．

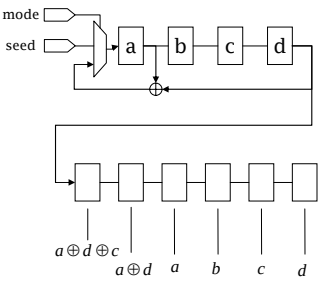

図 2:LFSR 論理モデル

An Efficient LFS Coding Method Using Don’t Care Inputs Identification Technique

Tatsuya USAMI , Toshinori HOSOKAWA

シフトチェインの下に記されている式は，LFSR の初期値を(a,b,c,d)とした，6 時刻後の各シフトチェインの論理関数である．この論理関数は，符号化するべきパターンの連立方程式である．したがってこの連立方程式を解くことで， LFS コーディングのための符号化が可能となる．

$$\left.\begin{aligned} a \oplus d \oplus c &= 0 \\ a &= 0 \\ b &= 1 \\ d &= 1 \end{aligned}\right\} \cdots\cdots\cdots (1)$$

$$\because a, b, c, d \in \{0,1\}$$

例を挙げると，(0,X,0,1,X,1)というテストパターンを入力するための図 2 中の a,b,c,d を求めたい．そのためには連立方程式(1)を解くことで求められる．これにより求められる a,b,c,d，すなわち符号化されたテストパターンは(a,b,c,d)＝(0,1,1,1)である．

また，LFSR はそのレジスタ数，シフトチェインが可変長でなく，シフトする時刻数がシフトチェイン数と等しい場合，組み合わせ回路により等価な論理を持つ回路を作成することが可能である．このことにより生成した図 2 と等価な論理を持つ回路を図 3 に示す．

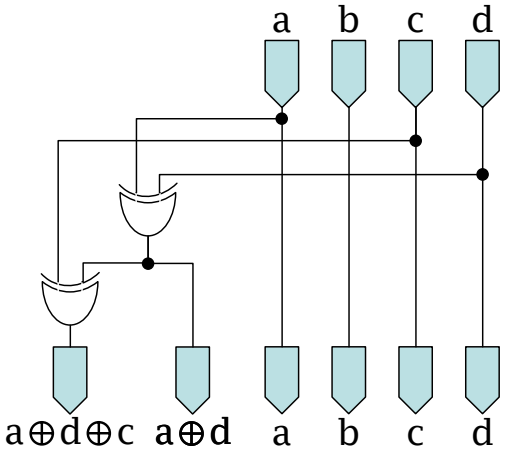

図. 3:LFS 符号復号器

本手法では，図.3 のような回路を構成し，展開器として使用することで，回路の入力ビット幅調整の技術として利用する．

４．LFS 符号化のアルゴリズム

４．１．LFS 符号化実施の上での問題点

本章では，実際に LFS 符号化の実施方法について述べる．LFS 符号化は 2 章でも述べた通り，より小さい回路の入力ビット幅でテストするためのものである．そのために連立方程式を解くわけであるが，その入力パターンと復号器の構成によっては連立方程式が解なしとなる場合が存在する．

例として，図 2,図 3 の論理によって，(1X0X11)というパターンを符号化することを考える．このパターンの符号化には連立方程式(2)の解を求めればよい．

$$\left.\begin{aligned} a \oplus d \oplus c &= 1 \\ a &= 0 \\ b &= 1 \\ d &= 1 \end{aligned}\right\} \cdots\cdots\cdots (2)$$

$$\because a, b, c, d \in \{0,1\}$$

しかしこの連立方程式の解を求めようとすると，解は存在しないことが解る．このような場合，モデルとなる LFSR のビット数，つまり復号器の入力ビット幅を大きくする必要がある．しかしながら，SoC 中のコアのテストを考えた場合，SoC の外部入力数は限られているため，SoC の外部入力数/コアの入力数の圧縮が可能であることが前提である．

４．２．簡単な符号化手法

図 4 に符号化手法実施の最も簡単なフローを示す．

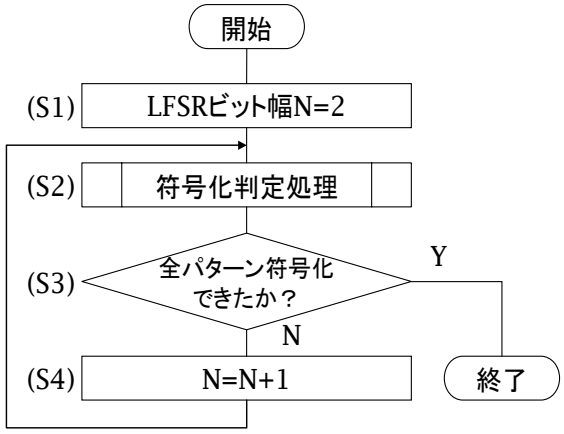

図 4:符号化実施手法例

(S1)で LFSR のビット幅，つまり符号器の入力ビットを 2 に初期化する．

(S2)で，指定されたビット数での符号化が可能かどうかを調べ，符号化する．

(S3)で符号化が完了したかを調べ，符号化でき

ていなければ(S4)に進み，ビット幅を1増加させ，再度符号化判定処理を行う．

次節では(S2)の部分，符号化判定処理について詳細に述べる．

４．３．符号化判定処理

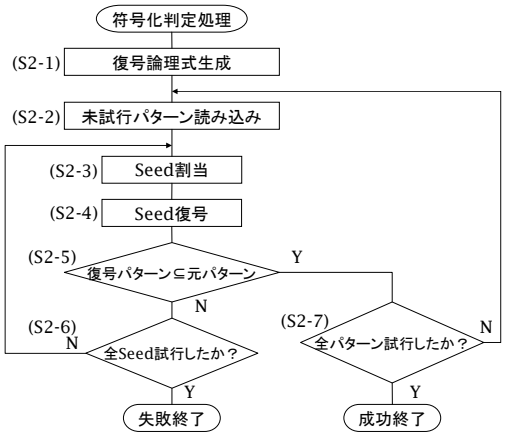

図 5:符号化判定処理の詳細フロー

図 5 に符号化判定処理の詳細なフローを示す．まず，(S2-1)において指定されたビット数の LFS 復号論理式を生成する．(S2-2)では，符号化するパターンを 1 つ読み込んでいる．そして(S2-3)，(S2-4)，(S2-5)で LFSR のシード，つまり符号化パターンを割り当て，その割り当てパターンが適切かどうかを調べている．

割り当てたパターンが適切でなければ(S2-6)へ進み，また異なるLFSRシードを割り当てる処理へ戻る．また，割り当てるシードがもう無い場合は失敗終了として符号化判定処理を抜ける．

シードとして割り当てたパターンが適切であれば，(S2-7)へ進み，次のテストパターンに対しても同じ処理を適用する．試行するパターンが無くなれば，成功終了として符号化判定処理を抜ける．

ただし，ここまで述べた手法だけでは，あまり効率の良いビット幅の縮小結果は得られるとは考えにくい．なぜなら，高い縮小効率を求めようとすればするほど，LFSR モデルで表現するところのシフトチェインの入力側ビットの論理関数が複雑になってしまうためである．さらに論理回路のテストパターンは一般的に非常にパターン，数量共に多くなり，複雑な論理関数になればなるほど，連立方程式の解を得ることが不可能になってしまう．次章では，その改善方法について述べる．

５．効率的な LFS 符号化

５．１．効率化のための戦略

前章で述べたとおり，LFSR モデルのシフトチェインの入力側のビットの復号論理の検討が難しい．そこで難しくさせている要因の一つである，元となるテストパターンの中で，複雑な論理関数が割り当てられる部分に着目する．

一般に論理回路のテストパターンは 0 もしくは 1 により表記されているが，故障を検出する際に必要でない論理値というものが存在する．この値をドントケアと呼び，X で表される．ドントケアは通常テスト生成時にランダムな値により 0 か 1 どちらかに設定されている．このドントケアが複雑な論理関数の部分に多く割り当てられれば，符号化時の失敗終了する可能性が減少すると考えられる．

本章ではビット割り当てを変更し，パターン中にドントケアの多い入力が複雑な論理関数のビットに割り当てられるようにする手法と，テストパターン中の対象部分にドントケアを抽出し，テストパターンを改良する方法を提案する．

５．２．ビット割り当て変更処理を用いた LFS コーディングの効率化

本節では，展開器の構成上依存する入力の多い出力ビットにテストパターン中のドントケアの多いビットを割り当てる手法について紹介する．

ビット割り当て変更処理は図 4 のフロー中の符号化試行処理の直前に挿入することで，効果を得ることができる．以下にビット割り当て変更処理のフローを示す．

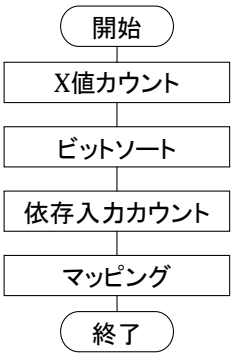

図. 6:ビット割り当て変更処理

まず，テストパターン中のドントケアの数を，入力ごとにカウントする．そして，その数順にテストパターンをビットスライスごとにソートする．依存する入

力数を，展開器の出力ごとに数える．最後に依存する入力の多いものから，ドントケアの多いビットにマッピングする．

マッピング時のイメージを図 7 に示す．見ての通り，テストパターンを並べ替え，依存関係の強い出力をドントケアの多いビットスライスを持つテスト対象回路の入力につなぎ変えているものである．

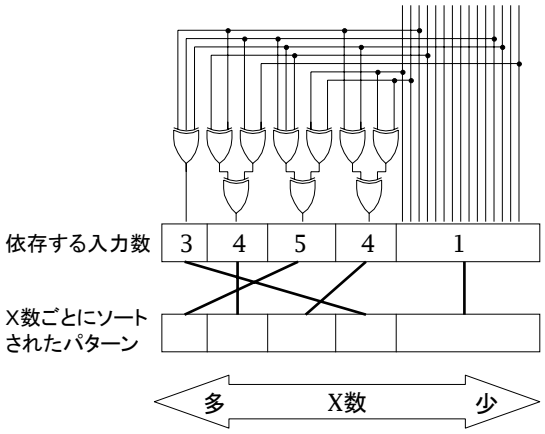

図 7:ビット割り当て変更後のイメージ

５．３．ドントケア抽出を用いた LFS コーディングの効率化

本節では LFS 符号化の論理関数が複雑になるテストパターン中のビットに対して，ドントケア抽出技術(2 を用いることで，テストパターンを改良し，符号化成功率を向上させようとする手法である．

一般的なテストパターン中には故障の検出に必要の無い論理値が潜在的にかなり存在する．そのため，ある程度の数のドントケアを任意の場所に抽出することが可能である．

ドントケア抽出処理は，すべての符号化処理の前に行い，LFSR モデルにおけるシフトチェインの入力側に近いビットにできる限りドントケアが抽出されるように行う．

図 8 に特定ビットに対するドントケア抽出処理(3 を行った前後のテストパターンの一例を示す．

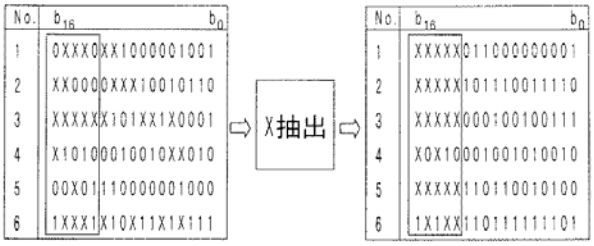

図 8:ドントケア抽出例

６．おわりに

本稿では LFS 符号化技術の効率化について検討し，2 種類の効率化技術について述べた．

今後は提案した手法を実装し，圧縮効率，ビット幅調整の効用，面積のオーバーヘッド等を他の技術と比較，検討することを考える．

参考文献

(1) 欧州特許番号：EP0481097 “Method and apparatus for testing a VLSI device” International Business Machines Corporation, 2003.

(2) S.Kajihara and K.miyase, “On Identifying Don’t Care Inputs of Test Patterns for Combinational Circuits, “ Proc. Of Int’l Conf. of CAD, pp. 364-369, 2001.

(3) K.Miyase et al., ”Don’t-Care Identification on Specific Bits of Test Patterns”, ICCD,2002.

(4) 藤原秀雄 著 “ディジタルシステムの設計とテスト” 工学図書 刊 2004 年 ISBN-4769204590