並列乗算器のテストに関する研究

鬼頭 信貴

要旨

VLSI 製造技術および設計技術の進展により，VLSI チップ上に集積される回路がますます大規模化している．VLSI チップのテストに要するコストが増大しており，テストの容易化は重要となっている．本論文は，データパス回路において多く用いられる並列乗算器のテストについて，その研究の成果をまとめたものである．本論文では，これまで知られていなかった，乗算器のビット幅に依存せず定数個のテストパターンでテストできる (C テスト可能) 高速なツリー型乗算器の構成法を示す．さらに，C テスト可能な種々の乗算器を構成する手法を示す．また，上述の構成法で対象外とした乗算器を含め，桁上げ保存加算器で構成したあらゆる乗算器が，若干の回路の付加により，回路の段数に比例する個数のパターンでテストできる (レベルテスト可能) ことを明らかにする．本論文で得られた乗算器のテストに対する知見が，今後，様々の算術演算回路のためのテスト容易化手法を確立するための基礎となると期待される．また，回路設計において，本論文で示した乗算器や乗算器の設計法を用いることで，VLSI のコストダウンの実現に貢献できるものと期待される．

第 1 章では，研究の背景，及び，本論文の構成と各章の概要を示す．VLSI のテストコストの低減において，VLSI を構成する回路のテスト容易化やテストに関する性質を明らかにすることの重要性を説明する．そして，乗算器がデータパス回路で用いられ，乗算器のテスト容易化がチップ全体のテスト容易化につながることを説明する．乗算器のテスト容易化について従来の研究を挙げ，研究の現状を整理し，本論文の研究の位置付けを示す．

第 2 章では，準備として並列乗算器と本研究で用いる故障モデルを説明する．並列乗算器は部分積生成部，部分積加算部，最終加算部の 3 つの部分で構成されることを

説明し，各部の構成を説明する．本論文で扱う故障モデルである，単一セル機能故障についても説明する．単一セル機能故障の仮定においては，回路をセルを用いて構成し，回路中の高々一つのセルが機能故障を起こすと考える．そして，セルの故障を検出できるようにテスト集合を構成する．この故障モデルがセルのゲートレベルでの実現に依存しないことを説明し，ゲートレベルの故障モデルと比較して高品質なテストが可能であることを説明する．

第 3 章では，C テスト可能な 4-2 加算木を用いた乗算器を示す．これまで，4-2 加算木を用いた乗算器をはじめとするツリー型乗算器について，C テスト可能な構成は知られていなかった．4-2 加算木のテストのために交互反転パターンとよぶパターンを提案する．テスト容易な 4-2 加算木の構成法と，構成した 4-2 加算木の規則性をいかした再帰的なパターンの設計手法を示す．部分積生成部へ若干の回路を追加することにより，部分積生成部で交互反転パターンを生成できることを示す．提案するパターンの設計法により，乗算器のビット幅に依存せず，14 個のパターンで 4-2 加算木と部分積生成部のテストができることを明らかにする．最終加算器として既存のテスト容易な桁上げ伝搬加算器を構成を用いることで乗算器を構成できることを示す．

第 4 章では，種々の部分積加算器を構成可能なテスト容易な乗算器の構成法を示す．提案法では 3 種類の加算器のブロックを組み合わせて部分積加算部を構成する．ブロックの組み合わせ方により，規則正しい回路構造で小面積な配列型乗算器や，高速なツリー型の乗算器を設計できることを示す．提案法により，要求される性能に合わせたテスト容易な乗算器の設計が可能であることを示す．乗算器が演算数のビット幅に関わりなく 14 個のパターンでテストできることを示す．

第 5 章では，第 4 章の手法の対象外になる Wallace 木を含め，CSA で構成した任意の構造の部分積加算部が，テストのための若干の回路の追加でレベルテスト可能となることを示す．まず，部分積加算部の入出力間の CSA の段数をレベル数 L としたとき，任意の構造の部分積加算部が部分積生成部を含めて高々$6L + 5$ 個のパターンでテストできることを示す．さらに，CSA 間の接続に制約を加えると，高々$2L + 9$ 個

のパターンでテストできることを示す．従来研究では，Wallace 木など，4-2 加算木よりレベル数が小さな部分積加算部について，オペランドのサイズとテストに必要なパターン数との間の関係が示されていなかった．本章は，これらの部分積加算部について，オペランドのサイズとテストに必要なパターン数との間の関係をはじめて明らかにする．

第 6 章では結論を述べる．本論文の研究の成果をまとめ，研究を通して得た並列乗算器のテストに対する知見が，乗算以外の算術演算回路のテスト容易化や，遅延故障等の機能故障を越えた様々な故障に対するテスト手法確立の基礎となると結論付ける．課題，展望についても述べる．

目 次

図 目 次

表 目 次

第1章　緒論

1.1　研究の背景

VLSI 製造技術および設計技術の進展により，VLSI チップ上に集積される回路がますます大規模化し，そのテストに要するコストが増大している．たとえば，国際半導体技術ロードマップ (International Technology Roadmap for Semiconductors)[1] では，一つの SoC(System on a Chip) のテストに必要な時間が 2012 年には 2007 年の 5 倍以上になると予測している．VLSI のテストのコストを低減するためには，VLSI を構成する回路のテスト容易化やテストに関する性質の解明が重要である．

乗算は信号処理やコンピュータグラフィックス，科学技術計算などで頻出する演算である．乗算を高速に行うために，マイクロプロセッサや特定用途向け集積回路のデータパス回路には乗算器が組み込まれている．本研究では並列乗算器のテストについて考える．現在では，コンピュータグラフィックスにおける演算や科学技術計算の高速化のために多数の乗算器を一つのチップに搭載することがあり，乗算器のテストを容易化することはチップ全体のテスト容易化につながると考えられる．

本研究では故障モデルとして単一セル機能故障を仮定する．単一セル機能故障の仮定では，回路をセルを用いて構成する．回路中の高々一つのセルが機能故障を起こすと考え，セルの機能故障を検出可能なテストパターンを構成する．この故障モデルは，セルのゲートレベルでの実現に依存しない．そして，ゲートレベルの故障モデルと比較して高品質のテストが可能である．本研究では，全加算器等をセルとして扱う．

並列乗算器は一般に部分積生成部，部分積加算部，最終加算部の 3 つの部分で構成される．部分積生成部では乗数と被乗数から部分積を生成する．部分積加算部では桁

上げ保存加算器 (Carry Save Adder, CSA) を用いて桁上げ伝搬なしで部分積を加え合わせ，桁上げ保存形で加算結果を得る．最終加算部は桁上げ伝搬加算器からなり，部分積加算部で得られる桁上げ保存形の桁上げと中間和を加え合わせて最終結果を出力する．

乗算器には，その主要な構成要素である部分積加算部の構造により，配列型乗算器や Wallace 乗算器や 4-2 加算木を用いた乗算器等のさまざまな種類がある．配列型乗算器では直列に接続した CSA を部分積加算部として用いる．Wallace 乗算器や 4-2 加算木を用いた乗算器では，ツリー状に接続した CSA を部分積加算部として用いる．このため，これらの乗算器はツリー型乗算器と呼ばれる．部分積加算部の入出力間の CSA の段数の最大数をレベル数 L とすると，同じビット幅においてツリー型乗算器のレベル数は配列型乗算器のレベル数よりも小さく，高速に乗算ができる．特に，Wallace 乗算器では L が最小になる．

これまで，テスト容易な乗算器の研究が様々行われている．文献 [2, 3, 4, 5] では，ビット幅に依存せず定数個のパターンでテスト可能 (C テスト可能) な配列型乗算器を示している．文献 [6, 7] では，レベル数 L に比例する数のパターンでテスト可能 (レベルテスト可能) な 4-2 加算木を用いた乗算器を示している．しかし，4-2 加算木を用いた乗算器をはじめとするツリー型乗算器について C テスト可能な設計が提案されておらず，高速乗算器として広く用いられている Wallace 乗算器についてはビット幅とテストに必要なパターン数との間の関係が知られていなかった．

1.2 本論文の構成と各章の概要

本論文では，はじめに，これまで知られていなかった，C テスト可能なツリー型乗算器の構成法を示す．次に，C テスト可能な種々の乗算器を構成する手法を示す．そして，Wallace 乗算器等の上述の構成法で対象外となる乗算器も含め，CSA で構成したあらゆる乗算器が若干の回路の付加によりレベルテスト可能であることを明らかにする．

第 2 章では，準備として並列乗算器と本論文で用いる故障モデルの説明を行う．並列乗算器が部分積生成部，部分積加算部，最終加算部の 3 つの部分で構成されることを説明し，各部の構成を説明する．本論文で扱う故障モデルである，単一セル機能故障についても説明する．

第 3 章では，C テスト可能な 4-2 加算木を用いた乗算器を示す．これまで C テスト可能なツリー型乗算器は知られていなかった．4-2 加算木のテストのために交互反転パターンとよぶことにする新たなパターンを導入する．テスト容易な 4-2 加算木の構成法と，構成した 4-2 加算木の規則性をいかした再帰的なパターンの設計手法を示す．部分積生成部へ若干の回路を追加することにより，部分積生成部で交互反転パターンを生成できることを示す．部分積生成部と部分積加算部はオペランドのサイズに依存せずに常に 14 個のパターンでテストできる．最終加算器として，既存の C テスト可能な桁上げ伝搬加算器を用いることで，C テスト可能な乗算器を構成できることを示す．

第 4 章では，部分積加算部を 3 種類の加算器のブロックを組み合わせて構成できる，C テスト可能な種々の乗算器を構成する手法を示す．ブロックの組み合わせ方により，規則正しい回路構造で小面積だが高速とはいえない配列型乗算器や，高速だが回路構造が複雑で面積がやや大きいツリー型の乗算器を設計できる．従来の乗算器のテスト容易化に関する研究は，手法が対象とする乗算器の構成に強く依存するため，同じ手法を他のタイプの乗算器に適用することはできない．そのため，要求性能が変化し，部分積加算の構造を他のタイプに変更する場合，同じ手法では対応できなかった．提案法により，要求される性能に合わせたテスト容易な乗算器を設計できる．演算数のビット幅に関わりなく，乗算器全体を 14 個のパターンでテストできることを示す．

第 5 章では，第 4 章の手法の対象外になる Wallace 木を含め，CSA で構成したあらゆる構造の部分積加算部が，若干の回路の追加によりレベルテスト可能になることを示す．まず，任意の構造の部分積加算部が部分積生成部を含めて高々$6L + 5$ 個のパターンでテストできることを示す．さらに，CSA 間の接続に制約を加えると，高々

$2L + 9$ 個のパターンでテストできることを示す．これまで，Wallace 木など，4-2 加算木よりレベル数が小さな部分積加算部について，オペランドのサイズとテストに必要なパターン数との間の関係が示されていなかった．本章は，これらの部分積加算部について，オペランドのサイズとテストに必要なパターン数との間の関係をはじめて明らかにした．

第 6 章では結論を述べる．本論文の研究の成果をまとめ，研究を通して得た知見と課題，展望を述べる．

第2章　準備

2.1　並列乗算器

本論文では，符号なし2進並列乗算器を対象とする．被乗数，乗数ともにNビットとし，被乗数を$X = (x_{N-1}...x_0)_2$，乗数を$Y = (y_{N-1}...y_0)_2$とする．一般的な並列乗算器は図2.1のように部分積生成部，部分積加算部，最終加算部の3つの部分から構成される．部分積生成部は，被乗数と乗数から部分積$P_j = X \cdot y_j \cdot 2^j \quad (0 \leq j < N)$を生成する．部分積加算部は，桁上げ保存加算器(CSA)を用いて桁上げ伝搬なしで部分積を加え合わせ，桁上げ保存形で結果を得る．最終加算部は，部分積加算部で得られる桁上げ保存形の桁上げと中間和を足し合わせ，最終結果を出力する．部分積加算部の例を図2.2に示す．部分積加算部の入力は，部分積$P_0, \ldots, P_{N-1}$である．

CSAは図2.3(a)のように全加算器(FA)で構成される．CSAは3つの2進数を足し合わせ，加算結果は桁上げ保存形で桁上げと中間和の2つの2進数で出力する．出力は両端を除いて各桁で2ビットずつ得られる．

乗算器は部分積加算部の構造により，配列型乗算器やWallace乗算器，4-2加算木を用いた乗算器等のさまざまな種類がある．図2.4に配列型乗算器の部分積加算部を示す．配列型乗算器では直列に接続したCSAを部分積加算部として用いる．NビットWallace乗算器では，まずN個の部分積を$\lfloor \frac{N}{3} \rfloor$個のCSAを用いて3つずつ並列に加え合わせ，およそ$\frac{2N}{3}$個の中間結果を得る．次に，これらの中間結果をおよそ$\frac{2N}{9}$個のCSAを用いて3つずつ並列に加え合わせる．このように中間結果を3つずつ並列に加え合わせる計算を繰り返す．図2.2はWallace乗算器の部分積加算部の例である．4-2加算木を用いた乗算器では，図2.5のように，CSA 2段からなる4-2加算器を2分

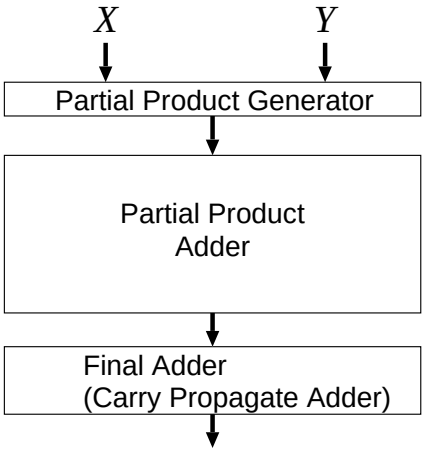


図 2.1: 並列乗算器の構成

木状に接続した 4-2 加算木を部分積加算部として用いる．Wallace 乗算器や 4-2 加算木を用いた乗算器などは，その部分積加算部の構造から，ツリー型乗算器と呼ばれている．

部分積加算部の入出力間の CSA の段数の最大数をレベル数 L とすると，N ビット配列型乗算器では $L = N - 2$，Wallace 乗算器では $L \approx \log_{\frac{3}{2}} N$，4-2 加算木を用いた乗算器では $L = 2\lceil \log_2(\frac{N}{2}) \rceil$ となる．L は Wallace 乗算器で最小となり，最も高速に部分積の加算ができる．

VLSI の設計においては，レイアウトのしやすさや遅延時間を考慮して，要求に合う乗算器を選択する．配列型乗算器はレイアウトが容易だが，CSA の段数が多くなるため高速とは言えず，Wallace 乗算器はレイアウトが複雑となるが，高速である．4-2 加算木を用いる乗算器は，Wallace 乗算器と比較すると遅延時間で若干劣るが，レイアウトは Wallace 乗算器より容易である．

2.2 故障モデル

VLSI のテストは，欠陥のある VLSI チップを出荷前に取り除くために行われる．論理回路のテストは，基本的に，回路に入力パターンを印加し，出力を観測し，想定し

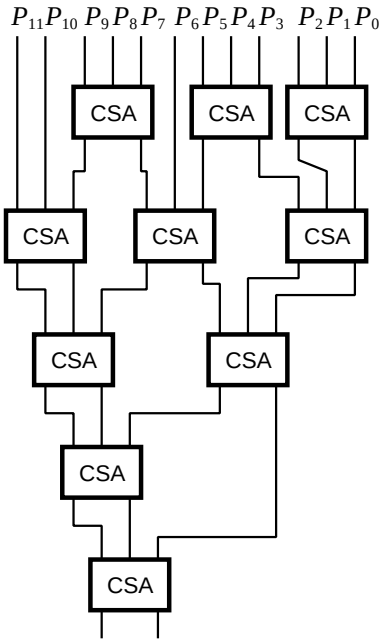


図 2.2: 部分積加算部の例

た出力を得られるか確認することで行われる．

テストを考える際には，欠陥の振る舞い (故障とよぶ) をモデル化した故障モデルを設定する．そして，設定した故障モデル上で故障を検出できるようにテストパターンの構成を考える．注目する故障やモデル化のしかたにより様々な故障モデルが存在する．VLSI チップに対してテストパターンを構成する際には，様々な故障モデルを組み合わせることができる．欠陥のある VLSI チップを検出するために必要な故障モ

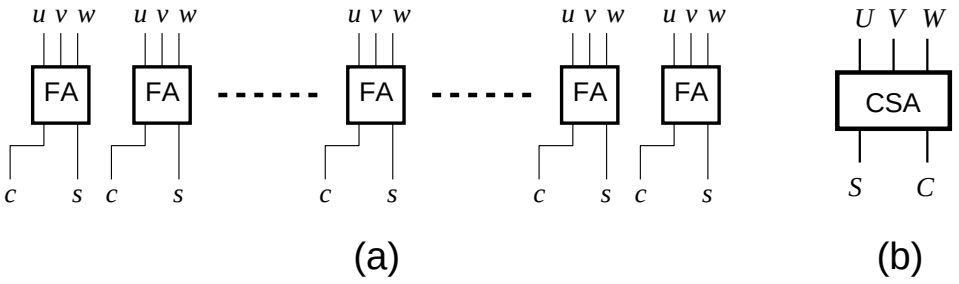


図 2.3: 桁上げ保存加算器

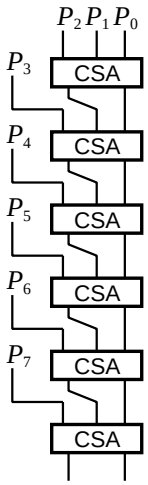


図 2.4: 8 ビット配列型乗算器の部分積加算部

デルを経済性などを考慮した上でいくつか選択し，これらの各故障モデルのもとでテストのためのパターンを構成する．

本論文では，故障モデルとして単一セル機能故障を仮定する．FA 等をセルとして扱い，これらのセルを用いて回路を構成する．単一セル機能故障は論理回路の故障モデルとしては基本的なものであり，文献 [5, 6, 8, 9, 10, 11] 等で用いられている．単一セル機能故障とは，次の条件を満たす故障である．

- テスト対象回路で故障するのは高々1 つのセルのみである．
- 故障したセルの出力は現在の入力のみで決まる．
- 故障したセルでは，正常なセルと出力値が異なるような入力が存在する．

単一セル機能故障を検出するため，次の 2 つの条件を満たすようにテストパターンを設計する．

- テスト対象回路を構成する全てのセルの各々に全入力パターンを入力できる．つまり，各セルについて，セルが m 入力のとき 2^m 通りのパターンを全て入力できる．

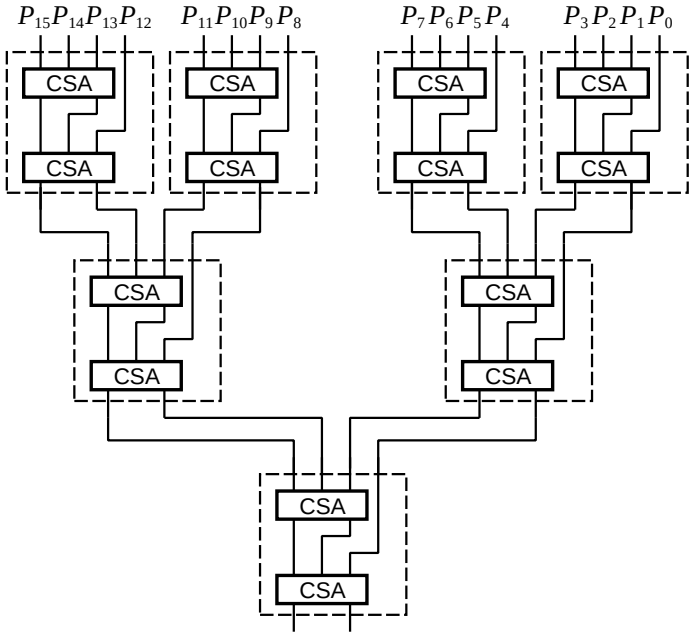


図 2.5: 16 ビット乗算器のための 4-2 加算木

- テスト対象回路内のセル故障の影響が回路出力まで伝搬し，回路出力で観察できる．

単一セル機能故障では，セルのレベルでテストを考えるため，セルのゲートレベルの実現方法に依存しない．また，ゲートレベルで構成された各セルに対し網羅的に全入力パターンを入力するため，ゲートレベルの故障モデルよりも高品質のテストができる．

回路が C テスト可能であるとは，回路がそのオペランドのサイズに依存せず定数個のパターンでテストできることである．また，回路がレベルテスト可能であるとは，回路が回路の段数に比例する数のパターンでテストできることである．

第3章　4-2加算木を用いたテスト容易化乗算器

3.1　はじめに

本章では，C テスト可能な 4-2 加算木を用いた乗算器を提案する．これまで，文献 [2, 3, 4, 5] 等で C テスト可能な配列型乗算器は提案されているものの，高速なツリー型乗算器について C テスト可能な設計はこれまで提案されていなかった．文献 [6, 7] の 4-2 加算木を用いた乗算器では，4-2 加算木の回路の段数に比例するパターンがテストに必要であった．

まず，テスト容易な 4-2 加算木の構成法と，構成した 4-2 加算木の規則性をいかした再帰的なパターンの設計手法を示す．4-2 加算木のテストのために，交互反転パターンとよぶ新たなパターンを導入する．4-2 加算木のテストのための交互反転パターンを生成可能な部分積生成部の構成も示す．提案するパターンの設計法により，乗算器のビット幅に依存せず 14 個のパターンで 4-2 加算木と部分積生成部のテストができる．

最終加算器として用いられる桁上げ伝搬加算器については，文献 [11] 等でテスト容易な構成が提案されている．図 3.1 のように部分積加算部と最終加算部の間にマルチプレクサを挿入することで，部分積加算部と切り離してテスト可能な構成にできる．C テスト可能な加算器を最終加算器に用いることで，乗算器全体が C テスト可能になる．

以降，次節で準備を行い，3.3 節で C テスト可能な 4-2 加算木とテストパターンの再帰的生成手法を示す．3.4 節では部分積生成部の構成法を示し，部分積生成部と 4-2 加算木が C テスト可能であることを示す．3.5 節ではテスト容易化によるオーバーヘッ

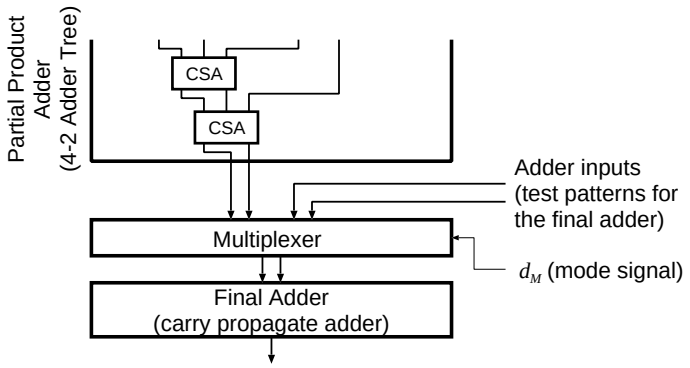


図 3.1: 最終加算器の設計

ドを見積もる．

3.2 準備

本章では，簡単のために乗算器のビット N を 4 以上の 2 の冪とする．

4-2 加算木を用いた乗算器では，4-2 加算木を部分積加算部として用いる．4-2 加算木は，4-2 加算器を用いて構成される．32 ビット乗算器における 4-2 加算木の例を図 3.2 に示す．本章では，4-2 加算木の各 4-2 加算器を $adder(l, m)$ と表す．l が 4-2 加算器が属するレベル，m がレベル内での 4-2 加算器の番号を示す．

木の葉の 4-2 加算器をレベルを 1 とし，根の 4-2 加算器のレベルを $L = \log_2 N - 1$ とする．レベル 1 の 4-2 加算器について，部分積 P_{4m}, P_{4m+1}, P_{4m+2}, P_{4m+3} を入力するものを $adder(1, m)$ とする．また，レベル $l(\geq 2)$ の 4-2 加算器について，$adder(l-1, 2m)$ と $adder(l-1, 2m+1)$ の出力を加算するものを $adder(l, m)$ とする．

4-2 加算器は 4 つの 2 進数を足し合わせ，加算結果は桁上げ保存形で桁上げと中間和の 2 つの 2 進数で出力する．出力は両端を除いて各桁で 2 ビットずつ得られる．本稿では，図 3.3(a) のように 4-2 加算器を 2 つの CSA で構成する．4-2 加算器の各ビット位置で，図 3.3(b) のようにビット加算は 2 つの FA が行う．この 2 つの FA で構成される回路を加算ブロックとよぶことにする．本章では，4-2 加算ブロックの入力に

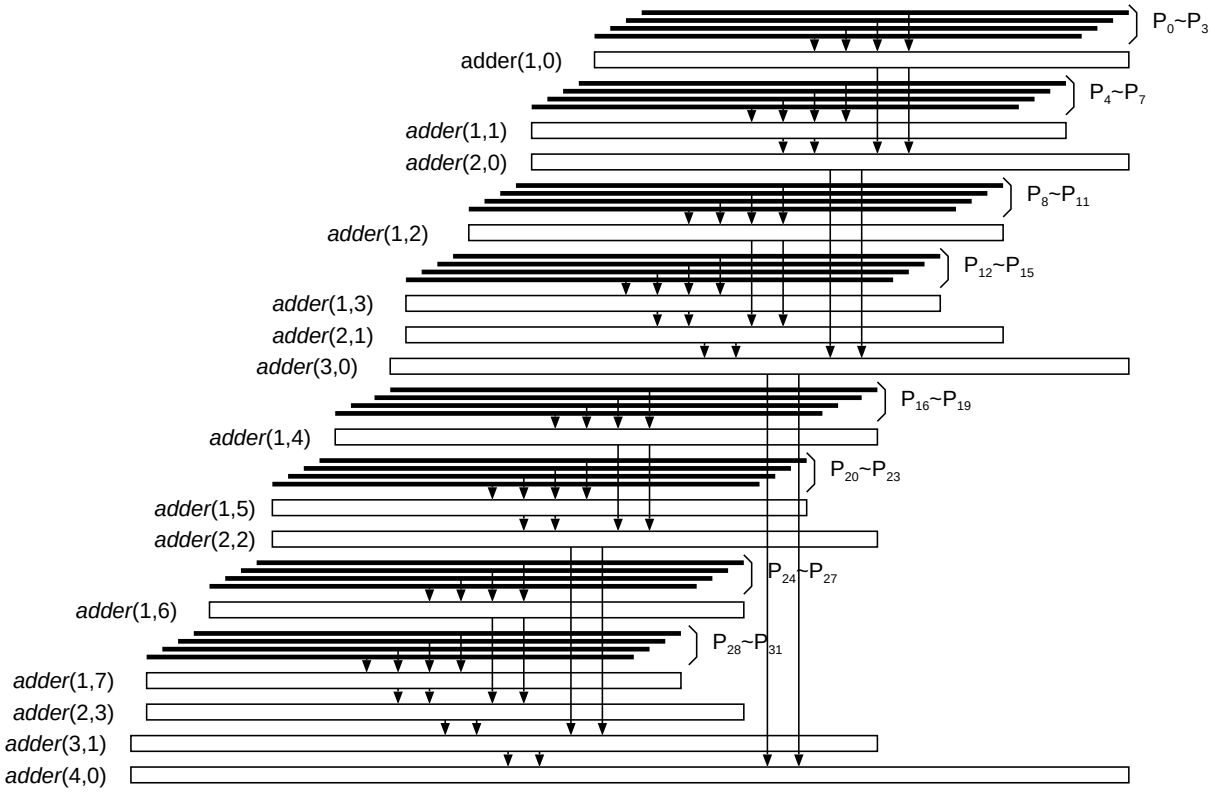


図 3.2: 32 ビット乗算器における 4-2 加算木

a, b, c, w，出力に e, f, u と名前付けする．加算ブロックの出力 u は一つ上のビット位置にある加算ブロックの入力 w と接続する．加算ブロック内部の信号線に v と名前付けする．

図 3.4 に 4-2 加算木の中の 4-2 加算器の構成を示す．図 3.4(a), (b), (c) はそれぞれ，レベル 1 の 4-2 加算器，レベル 2 の 4-2 加算器，3 以上のレベルの 4-2 加算器の構成を示す．図において，入力ビットの重みが 2^k のビット位置を位置 (position) k で表している．以後，位置 k の加算ブロックを加算ブロック k とよぶ．

各 4-2 加算器の両端のビット位置において，入力ビットは 4 つ未満となる．これらのビット位置では，4-2 加算ブロックの代わりに FA や半加算器 (HA) でビット加算を行う．

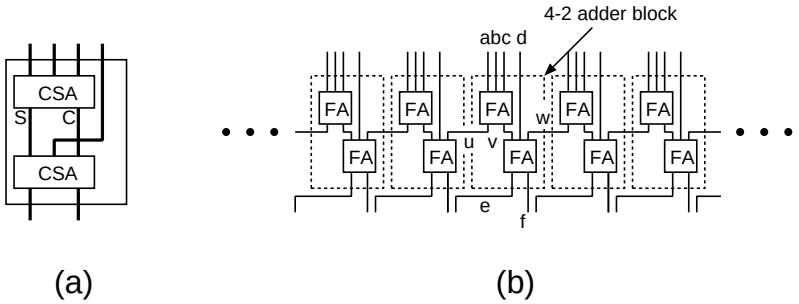


図 3.3: 4-2 加算器

3.3 4-2 加算木の構成とテスト設計

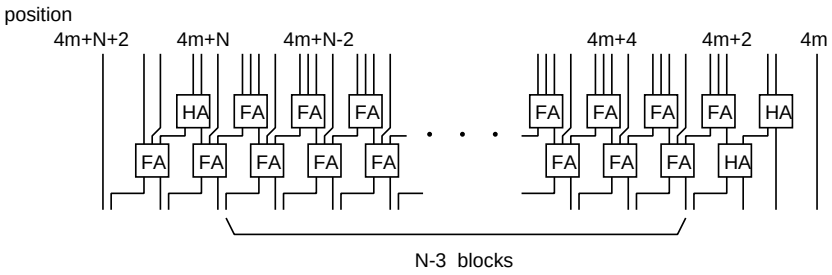
3.2 節で示したように，4-2 加算木において，$adder(l,m)(l \geq 2)$ は $adder(l-1,2m)$ と $adder(l-1,2m+1)$ の出力を入力とする．まず，4-2 加算器をテスト容易化するために，4-2 加算器間の接続を決める．ここでは，4-2 加算器の両端を除いた中央部分，つまり入力ビットが 4 つある $2^{l+1}m+2^l+2l-1$ から $2^{l+1}m+2^l+N-2$ までのビット位置を考える．それ以外のビット位置については次節で考える．

本稿では，図 3.5 に示すように，$adder(l,m)$ のブロック k の入力 a, b, c, d をそれぞれ，$adder(l-1,2m)$ のブロック k の出力 f，$adder(l-1,2m)$ のブロック $k-1$ の出力 e，$adder(l-1,2m+1)$ のブロック k の出力 f，$adder(l-1,2m+1)$ のブロック $k-1$ の出力 e に接続する．

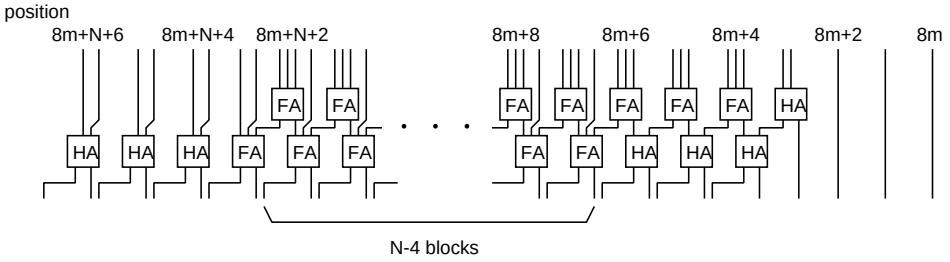
4-2 加算木のテストのために交互反転パターン (alternately inverted pattern) とよぶことにする新たな入力パターンを導入する．4-2 加算器への交互反転パターンは $(\alpha,\beta,\gamma,\delta)_{ai}$ (α, β, γ, $\delta \in \{0,1\}$) で表す．

交互反転パターン $(\alpha,\beta,\gamma,\delta)_{ai}$ は，各加算ブロック k の a, b, c, d に，k が偶数ならそれぞれ α, β, γ, δ を，k が奇数なら $\bar{\alpha}$, $\bar{\beta}$, $\bar{\gamma}$, $\bar{\delta}$ を入力する．$\bar{\alpha}$ は，α のビット反転を表す．

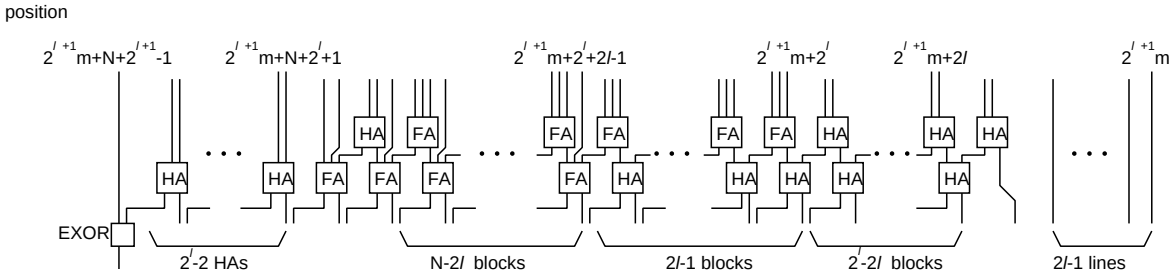
4-2 加算器への交互反転パターンは 16 種類存在する．これらのパターンの $(\alpha\beta\gamma\delta)$ を 2 進数とみて，$ai_0,\ldots,ai_{15}$ と名前付けする．例えば，$ai_5=(0,1,0,1)_{ai}$ である．4-2



(a) レベル 1 の 4-2 加算器: $adder(1, m)$



(b) レベル 2 の 4-2 加算器: $adder(2, m)$



(c) レベル l の 4-2 加算器 ($l \geq 3$): $adder(l, m)$

図 3.4: 4-2 加算木の中の 4-2 加算器

加算木のテストには 10 個の交互反転パターン (ai_0, ai_{15}, ai_1, ai_{14}, ai_3, ai_{12}, ai_5, ai_{10}, ai_7, ai_8) を用いる．

表 3.1 に，10 個の交互反転パターンにより 4-2 加算ブロックに入力される信号値を示す．表より，10 個の交互反転パターンにより，加算ブロック内の 2 つの FA (a, b, c を入力とする FA，v, d, w を入力とする FA) のどちらにも 8 つの全ての入力パターンを入力できる．FA が実現する関数が自己双対関数であるため，表において，4-2 加算器の出力も交互反転となっている．

交互反転パターンを 4-2 加算木に入力すると，4-2 加算器の出力が交互反転となり，

表 3.1: 交互反転パターン入力時の加算ブロック内の信号値

	even-numbered adder block						odd-numbered adder block					
pattern	a,b,c,d	w	v	e	f	u	a,b,c,d	w	v	e	f	u
ai_0	0,0,0,0	1	0	0	1	0	1,1,1,1	0	1	1	0	1
ai_{15}	1,1,1,1	0	1	1	0	1	0,0,0,0	1	0	0	1	0
ai_1	0,0,0,1	1	0	1	0	0	1,1,1,0	0	1	0	1	1
ai_{14}	1,1,1,0	0	1	0	1	1	0,0,0,1	1	0	1	0	0
ai_3	0,0,1,1	1	1	1	1	0	1,1,0,0	0	0	0	0	1
ai_{12}	1,1,0,0	0	0	0	0	1	0,0,1,1	1	1	1	1	0
ai_5	0,1,0,1	1	1	1	1	0	1,0,1,0	0	0	0	0	1
ai_{10}	1,0,1,0	0	0	0	0	1	0,1,0,1	1	1	1	1	0
ai_7	0,1,1,1	0	0	0	1	1	1,0,0,0	1	1	1	0	0
ai_8	1,0,0,0	1	1	1	0	0	0,1,1,1	0	0	0	1	1

4-2 加算木を効率的にテストできる．2 つの 4-2 加算器 $adder(l-1,2m)$ と $adder(l-1,2m+1)$ に適切な交互反転パターンを入力することで，$adder(l,m)$ にも交互反転パターンを入力できる．例えば，$adder(l-1,2m)$ と $adder(l-1,2m+1)$ にそれぞれ ai_1，ai_8 を入力すると，$adder(l,m)$ に ai_0 を入力できる．

表 3.2 に，4-2 加算木において隣接する 4-2 加算器間の入力パターンの関係を示す．この関係は加算木のどのレベルについても成り立つ．表より，3 つの 4-2 加算器で入力パターン集合が等しいことが分かる．したがって，レベル数に依存することなく，10 個のパターンを用いることで 4-2 加算木をテストできる．

4-2 加算木のテストパターン集合は表 3.2 の関係を再帰的に用いることで構成できる．構成する 10 個のテストパターンのそれぞれは，4-2 加算木の根の 4-2 加算器に 10 個の交互反転パターンを一つずつ入力する．

例として，レベル数が 4 の 4-2 加算木を考え，根の 4-2 加算木に ai_0 を入力するパター

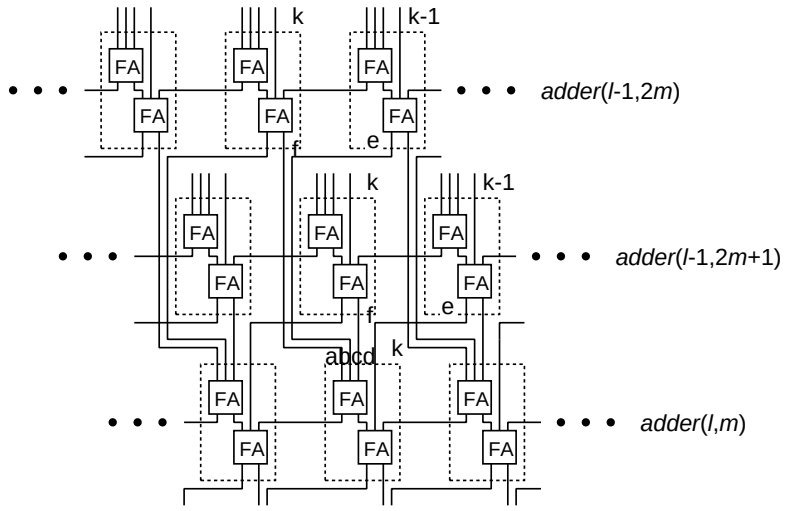


図 3.5: 4-2 加算器間の接続

ンの構成を考える．この 4-2 加算木の $adder(3,0)$ と $adder(3,1)$ にそれぞれ ai_1 と ai_8 を入力することで，$adder(4,0)$ に ai_0 を入力できる．また，$adder(2,0)$，$adder(2,1)$，$adder(2,2)$，$adder(2,3)$ のそれぞれに ai_8，ai_{12}，ai_3，ai_1 を入力することで，ai_1 を $adder(3,0)$ に，ai_8 を $adder(3,1)$ に入力できる．最後に，$adder(1,0)$ ... $adder(1,7)$ に入力するテストパターン $(ai_3,\ ai_1,\ ai_0,\ ai_{15},\ ai_{15},\ ai_0,\ ai_8,\ ai_{12})$ が得られる．

3.4 C テスト可能な 4-2 加算木と部分積生成部

3.4.1 4-2 加算木の構成

3.2 節に示したように，4-2 加算木の各 4-2 加算器において，両端の複数のビット位置の入力ビット数は 4 未満である．図 3.4 に示したように，これらのビット位置では 4-2 加算器の代わりに FA や HA を用いたビット加算が行われる．前節で示した再帰的なテスト設計法を用いるために，4-2 加算器のこれらの両端のビット位置についても 4-2 加算ブロックを用いる．

図 3.6 に，変更したレベル 1 の 4-2 加算器を示す．変更した部分を太線で示す．S セルと C セルは FA を縮退させたもので，それぞれ FA の和と桁上げのみを計算する．

表 3.2: 4-2 加算器における 4-2 加算器の入力パターン間の関係

$adder(l-1,2m)$	$adder(l-1,2m+1)$	$adder(l,m)$
ai_1	ai_8	ai_0
ai_{14}	ai_7	ai_{15}
ai_8	ai_{12}	ai_1
ai_7	ai_3	ai_{14}
ai_{15}	ai_0	ai_3
ai_0	ai_{15}	ai_{12}
ai_{10}	ai_{10}	ai_5
ai_5	ai_5	ai_{10}
ai_3	ai_1	ai_8
ai_{12}	ai_{14}	ai_7

$adder(1,0)$ の下位桁は少し構造が簡単になり，図 3.6(a) の破線で囲んだ部分を (b) のように置き換えられる．これは，この部分の出力が直接，最終加算器に接続するためである．

図中の ◦ で示す両端の入力端子には追加の入力ビットが必要である．これについては後で議論する．

図 3.7 に変更したレベル 2 の 4-2 加算器を示す．$adder(2,0)$ の下位桁は (b) のように簡単になる．図 3.8 に変更したレベル $l(l \geq 3)$ の 4-2 加算器を示す．$adder(l,0)$ の下位桁は (b) のように簡単になる．$adder(L,0)$ つまり加算木の根は，出力が最終加算器に直接接続するため，上位桁を変更する必要がなく (c) のようになる．

$adder(l,m)$ $(l \geq 2)$ の入力端子は，前節で述べたように，$adder(l-1,2m)$ と $adder(l-1,2m+1)$ の出力端子と接続する．すると，図 3.7 と図 3.8 で，◦ で示す両端部の入力端子が未接続のまま残る．($adder(3,m)$ では，最上位 (ビット位置が $16m+N+15$) の端子 c と図 3.8 で • で示すビット位置 $16m+N+7$ の端子 a はどの出力端子とも接

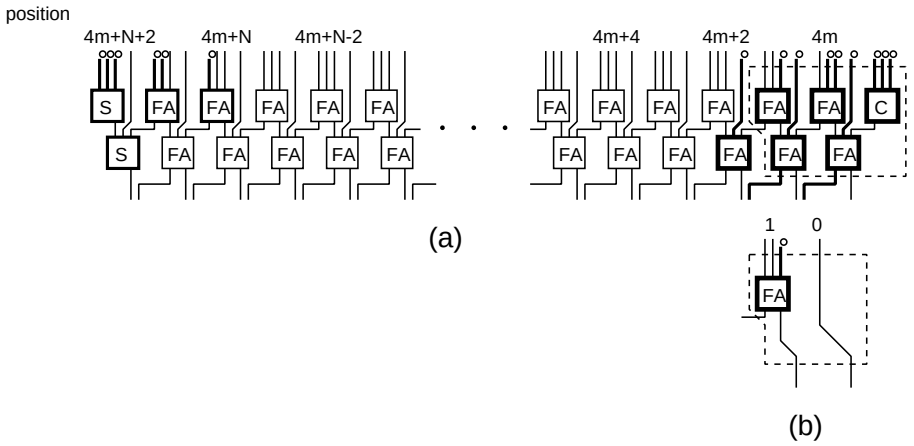


図 3.6: 変更したレベル 1 の 4-2 加算器: (a) $adder(1, m)$, (b) $adder(1, 0)$ の最下位部分

続していない．) これらの端子のために，レベル 1 の加算器の場合と同様に，追加の入力ビットが必要である．

追加する各入力ビットは通常時 0 になり，テスト時に適切な値になる必要がある．各 4-2 加算器には，テスト時に交互反転パターンが入力されるので，中央部の加算ブロックの対応する端子の入力ビットを分配し (あるいは，その否定をとり)，追加する入力ビットとして用いることができる．$adder(l, m)$ において，入力端子 a,b への追加入力ビットとして，ビット位置 $N + 2^{l+1}m - 1$ の入力ビットを用い，入力端子 c,d への追加入力ビットとして，ビット位置 $2^{l+1}m + N + 2^l - 1$ の入力ビットを用いる．これらの入力ビットは，前段の加算器の中央部の最上位からとっている．$2^{l+1}m + N - 1$ と $2^{l+1}m + N + 2^l - 1$ はともに奇数なので，偶数のビット位置については分配した信号を反転し，それを入力とする．

テスト時に，追加の入力ビットとその反転を生成するために，図 3.9 の EX セルを導入する．テストのための追加入力 t を導入し，通常時は 0 とすることにすると，EX セルは，入力ビットと信号 t の論理積と入力ビットの反転と信号 t の論理積をとることで実現できる．

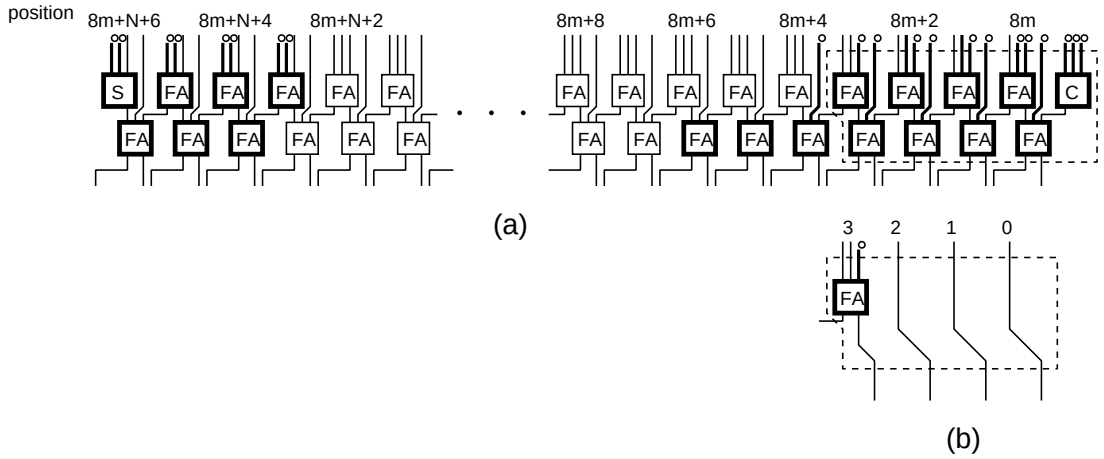


図 3.7: 変更したレベル 2 の 4-2 加算器: (a) $adder(2, m)$, (b) $adder(2, 0)$ の最下位部分

3.4.2 部分積生成部の構成

本項では部分積生成部の構成と，部分積生成部と 4-2 加算器の接続について考える．

部分積生成部は，各部分積を生成する N 個の部分積生成回路からなる．部分積 $P_j = X \cdot y_j \cdot 2^j$ を生成する部分積生成回路を $PPG(j)$ で表す．各部分積生成回路は N 個の PG セルから構成され，各 PG セルは乗数のビットと被乗数のビットの論理積を計算する．$PPG(j)$ のビット位置 k の PG セルは x_{k-j} と y_j の論理積を計算する．

部分積生成回路 $PPG(4m)$, $PPG(4m+1)$, $PPG(4m+2)$, $PPG(4m+3)$ $(0 \leq m \leq \frac{N}{4} - 1)$ のビット位置 k の出力端子は，$adder(1, m)$ の加算ブロック k の入力端子 a, b, c, d にそれぞれ接続する．

部分積生成部でテスト時に交互反転パターンを生成するため，$PPG(j)$ の PG セルを，ビット位置が偶数のものと奇数のもので 2 つのグループに分ける．テスト時に，ビット位置が偶数のものには y_j，奇数のものには y_j の反転を入力する．テスト時に，y_j の反転を生成するために，y_j と外部入力 t の EXOR を計算する CI(Controlled Inverter) セルを導入する．

図 3.10 に，変更した偶数の j と奇数の j に対する $PPG(j)$ の構成をそれぞれ示す．

交互反転パターン $(\alpha, \beta, \gamma, \delta)_{ai}$ $(\alpha, \beta, \gamma, \delta \in \{0, 1\})$ を $adder(1, m)$ に入力するには，

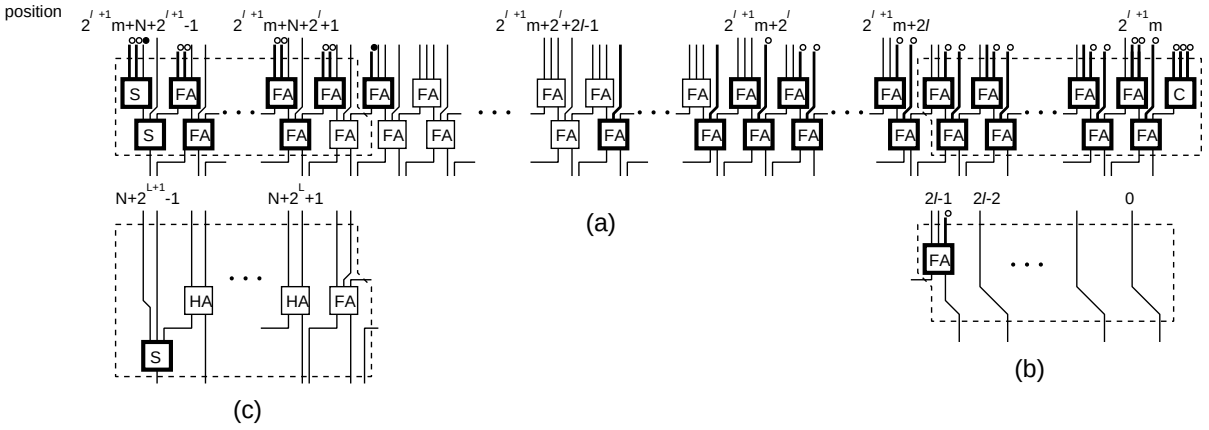


図 3.8: 変更したレベル l の 4-2 加算器 ($l \geq 3$): (a) $adder(l, m)$, (b) $adder(l, 0)$ 最下位部分, (c) $adder(L, 0)$ の最上位部分

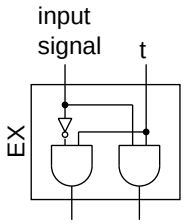


図 3.9: EX セル

X を $(11\cdots1)_2$ (all 1), y_{4m}, y_{4m+1}, y_{4m+2}, y_{4m+3} をそれぞれ α, β, γ, δ にする.

例として, 32 ビット乗算器の中の 4 レベルの 4-2 加算木を考える. PPG を通して $adder(4, 0)$ に ai_0 を入力するには, $X = (1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111)_2$, $Y = (0011\ 0001\ 0000\ 1111\ 1111\ 0000\ 1000\ 1100)_2$, $t = 1$ とする.

3.4.3 テスト設計と故障伝搬

単一セル機能故障の仮定におけるテストでは, 回路の各セルについて, 全ての入力パターンを入力する. 上述の 10 個の交互反転パターンを用いることで, 4-2 加算木の FA, C セル, S セルには全ての入力パターンを入力できる. 部分積生成部の PG セル, CI セルと 4-2 加算木の EX セルにすべての入力パターンを入力するために, 以下の 4 つの入力パターンを用いる.

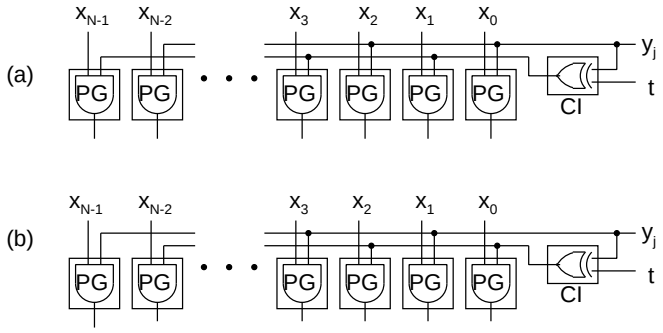


図 3.10: 変更した部分積生成回路 ((a) j が偶数の場合, (b) j が奇数の場合)

1. $X = (00\cdots0)_2$ (all 0), $Y = (00\cdots0)_2$ (all 0), $t = 0$

2. $X = (00\cdots0)_2$ (all 0), $Y = (11\cdots1)_2$ (all 1), $t = 0$

3. $X = (11\cdots1)_2$ (all 1), $Y = (00\cdots0)_2$ (all 0), $t = 0$

4. $X = (11\cdots1)_2$ (all 1), $Y = (11\cdots1)_2$ (all 1), $t = 0$

表 3.3 に 32 ビット乗算器の 4-2 加算木と部分積生成部のためのテスト集合を示す. ここで，セルの故障の影響が乗算器の出力で観察できることを示す.

CI セルの故障の影響は，部分積生成回路の $\frac{N}{2}$ 個の PG セルに伝搬し，$X = (00\cdots0)_2$ (all 0) 以外のとき，部分積を誤った値とする．CI セルには，$X = (11\cdots1)_2$ (all 1) のとき全ての入力パターンを入力するため，CI セルの故障の影響は部分積の誤りとして 4-2 加算器に伝播する．これは最終加算器まで伝搬し，乗算器の出力で観察できる．

他の種類のセル (PG セル, FA, C セル, S セル, EX セル) の故障は部分積や 4-2 加算器の出力を誤った値とする．$t = 0$ のとき，この故障の影響は最終加算器まで伝播し，乗算器の出力で観察できる．$t = 1$ で，4-2 加算器の EX セルに故障の影響が伝搬しない場合，影響は最終加算器に伝搬する．故障の影響が EX セルに伝搬する場合，その影響は 4-2 加算器の両端の複数の箇所にも伝搬する．a,b 入力のための EX セルに故障の影響が伝搬したとき，その影響は 4-2 加算器の上位と下位の部分に伝搬し，誤りが相殺して消えることはない．c,d 入力のための EX セルに故障の影響が伝搬したとき，その影響は 4-2 加算器の下位の部分に伝搬する．ここで，$adder(l, m)$ の EX セルに故

表 3.3: 32 ビット乗算器内の 4-2 加算木と部分積生成部のためのテスト集合

X	$Y(=(y_{31}\dots y_0)_2)$	t
all 1	0011 0001 0000 1111 1111 0000 1000 1100	1
all 1	1100 1110 1111 0000 0000 1111 0111 0011	1
all 1	1110 0111 0001 1000 0011 0001 0000 1111	1
all 1	0001 1000 1110 0111 1100 1110 1111 0000	1
all 1	1000 1100 0011 0001 0111 0011 1100 1110	1
all 1	0111 0011 1100 1110 1000 1100 0011 0001	1
all 1	0101 0101 0101 0101 0101 0101 0101 0101	1
all 1	1010 1010 1010 1010 1010 1010 1010 1010	1
all 1	0000 1111 0111 0011 1110 0111 0001 1000	1
all 1	1111 0000 1000 1100 0001 1000 1110 0111	1
all 0	0000 0000 0000 0000 0000 0000 0000 0000	0
all 0	1111 1111 1111 1111 1111 1111 1111 1111	0
all 1	0000 0000 0000 0000 0000 0000 0000 0000	0
all 1	1111 1111 1111 1111 1111 1111 1111 1111	0

障の影響が伝搬したときを考える．EX セルはビット位置 $2^{l+1}m+N+2^l-1$ にある．故障の影響は各加算器で高々2 桁分だけ伝搬するので，誤りは $2^{(2^{l+1}m+N+2^l-2-2l)}$ よりも大きい．一方，EX セルが生成する値は $2^{(2^{l+1}m+2^l+1)}$ より小さいため，打ち消し合うことはない．したがって，故障の影響は最終加算器まで伝搬する．

3.5 評価

部分積生成部と 4-2 加算木のハードウエアオーバーヘッドを 2 入力 NAND ゲート換算で評価した．例えば，CMOS テクノロジで，2 入力 EXOR ゲートと 2 入力 NAND

表 3.4: 各セルの等価ゲート数

Cell	Gate equivalents
FA	8.5
HA	4.0
C	3.5
S	5.0
PG	1.5
CI	2.5
EX	3.0

ゲートはそれぞれ 10 トランジスタ，4 トランジスタで実現できる．そこで，2 入力 EXOR ゲートで実現する CI セルを 2.5 ゲート相当と換算する．他のセルについて，トランジスタ数から等価ゲート数を求めたものを表 3.4 に示す．

ハードウエアオーバーヘッドの見積りを表 3.5 に示す．表 3.5 において，“Original 4-2 adder tree with PPGs” は元となる 4-2 加算木と部分積加算部の等価ゲート数を示す．“Proposed 4-2 adder tree with PPGs” は提案した 4-2 加算木と部分積加算部の等価ゲート数を示す．表 3.5 から，N が大きくなるとハードウエアオーバーヘッドの割合が小さくなることが分かる．64 ビット乗算器でのハードウエアオーバーヘッドは約 15%となる．

遅延時間のオーバーヘッドについて考える．部分積を PG セルのみで生成する元の部分積生成部と比較し，本稿の部分積生成部では CI セル (EXOR ゲート) 一つ分だけ遅延時間が長い．

4-2 加算木において，通常時，元の 4-2 加算木と提案した 4-2 加算木で信号が経由するセルの段数が等しい．そのため，4-2 加算木において遅延時間はほぼ等しい．したがって，4-2 加算木と部分積生成部について，遅延時間のオーバーヘッドは CI セル一つ分である．

表 3.5: 4-2 加算木と部分積生成部のハードウエアオーバーヘッド

N	C-testable 4-2 Adder Tree with PPGs (#of gates)	Original 4-2 Adder Tree with PPGs (#of gates)	Overhead
16	2996.0	2346.0	27.7 %
32	12254.0	10103.0	21.3 %
64	47315.0	41336.0	14.5 %
128	181184.0	165985.0	9.2 %

3.6 まとめ

高速乗算器のための，C テスト可能な 4-2 加算器と部分積生成部を示した．提案した 4-2 加算木は再帰的構造をもち，再帰的なパターンの生成手法によりテスト集合を求める．テストパターンには，交互反転パターンとよぶ特別なパターンを用いる．[12, 9, 10, 13, 11] 等のテスト容易な加算器を最終加算器として用いることでテスト容易な高速乗算器を構成できる．

本章では N を 2 の冪としたが，任意の N についても提案法は適用できる．N が 2 の冪ではない偶数の場合には，4-2 加算木は完全 2 分木とはならないものの，N が 2 の冪の場合と同様にテストパターンを構成できる．N が奇数の場合には，$N+1$ ビットの乗算器を構成する．このとき，X,Y のそれぞれ 1 ビット分がテストのための追加の入力信号になる．

第4章　種々の部分積加算構造をもつテスト容易な乗算器の設計手法

4.1　はじめに

本章では，さまざまな部分積加算部を構成可能なテスト容易な乗算器の設計手法を示す．提案手法では，部分積加算部を3種類の加算器のブロックを組み合わせて構成する．ブロックの組み合わせ方により，規則正しい回路構造で小面積だが高速とはいえない配列型乗算器や，高速だが回路構造が複雑で面積がやや大きいツリー型の乗算器を設計できる．これにより，要求される性能に合わせたテスト容易な乗算器を設計できる．乗算器は，全加算器等をセルとする単一セル機能故障の仮定において，演算数のビット幅に関わりなく14個のパターンでテストできる．

これまでにCテスト可能な配列型乗算器の設計に関する研究が行われている ([2, 4, 5] 等)．文献 [7, 14] ではツリー型乗算器のひとつである4-2加算木を用いた乗算器のテスト手法を述べている．しかしこれらの手法は，対象とする乗算器の構成に強く依存するため，同じ手法を他のタイプの乗算器に適用することはできない．そのため，要求性能が変化し，部分積加算の構造を他のタイプに変更する場合，同じ手法では対応できなかった．

以降，次節で準備を行い，4.3節でCテスト可能な部分積加算部の構成法とテスト手法を示す．4.4節では乗算器全体の設計法と，そのテスト方法を示す．4.5節で本章のまとめを行う．

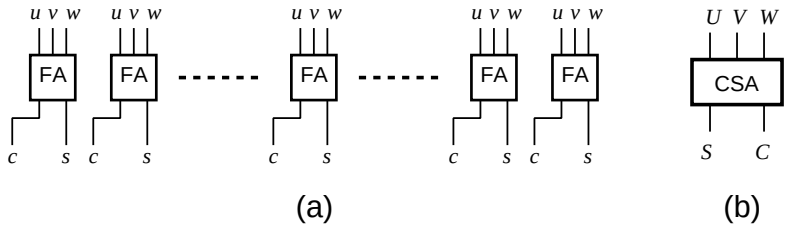


図 4.1: 桁上げ保存加算器

4.2 準備

本章では，CSA を構成する FA に，図 4.1(a) のように入力端子に u, v, w，出力端子に c, s と名前付けする．図 4.1(b) のように，CSA の入力に U, V, W，出力に S, C と名前付けする．U, V, W, S, C はそれぞれ CSA を構成する FA の u, v, w, s, c 端子からなる．CSA のなかの各 FA には入力ビットの重みに応じて番号をつける．乗算器の中で入力ビットの重みが 2^k の FA を番号 $k(k \geq 0)$ とする．FA の入力端子について，重みが 2^k の入力端子 u を u^k と表記する．

CSA を組み合わせて設計する部分積加算部には図 4.2 のように出力側をレベル 0 とし，入力側に向かってレベル付けする．CSA の属するレベルは，その CSA の出力側の CSA が属するレベルの最大値よりひとつ大きい値とする．部分積加算部をツリーとしてみることにし，レベル 0 の CSA を根，他の CSA の出力が入力とならない CSA を葉とよぶ．各レベルにおいて，各 CSA には区別のために 0 から順に重なりなく任意の順序でラベル付けをする．

4.3 C テスト可能な部分積加算部の構成法とテスト設計

4.3.1 部分積加算部の構成法

部分積加算部を CSA で構成した 3 種類のブロックを組み合わせることにより構成する．ブロックの組み合わせかたにより配列型乗算器の部分積加算部や 4-2 加算木 [15],

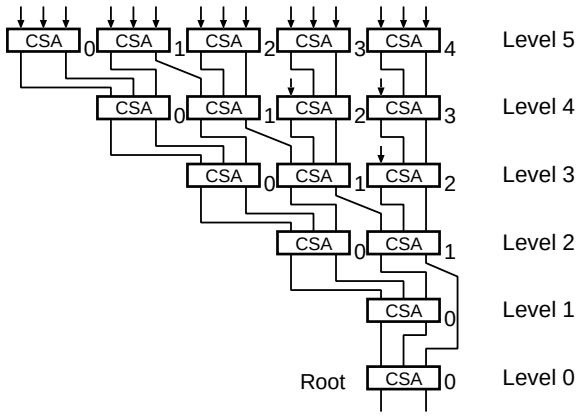


図 4.2: 部分積加算部へのレベル付け

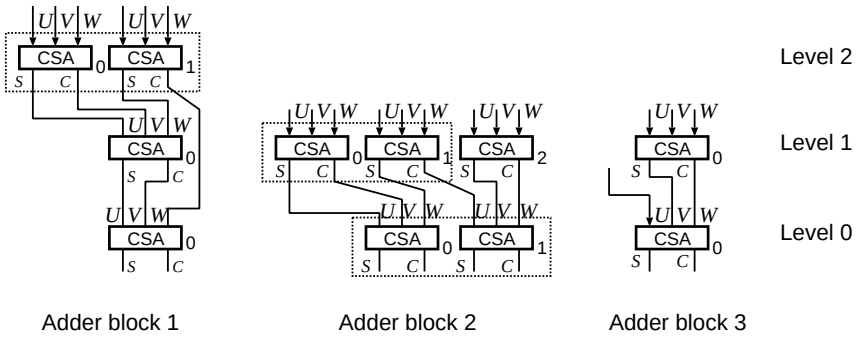


図 4.3: 加算器ブロック

Overturned-Stairs 木 [16]，バランス木 [17] などを構成できる．

3 種類のブロックを図 4.3 に示す．それぞれ加算器ブロック 1，加算器ブロック 2，加算器ブロック 3 とする．各加算器ブロックについて，レベル付けと各レベルでの CSA へのラベル付けをしている．図において CSA のラベルを CSA の右下に記している．

部分積加算部は根から葉に向かって加算器ブロックを接続することで構成する．部分積加算部の初期構成を加算器ブロック 1 あるいは加算器ブロック 3 とし，各時点の部分積加算部の葉に加算器ブロックを接続していくことで目的の部分積加算部を得る．加算器ブロックを接続する際には，加算器ブロックの出力部にある CSA をその時点の部分積加算部の葉と併合することで接続する．

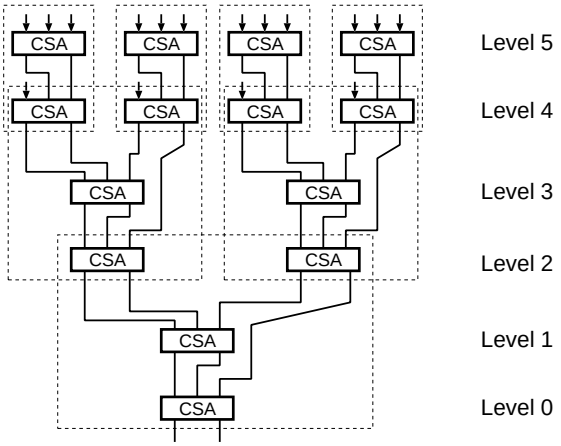


図 4.4: 部分積加算部の設計例 (4-2 加算木)

ただし加算器ブロック 2 の接続については，その時点の部分積加算部の葉の CSA のうち，図 4.3 で破線で示した加算器ブロック 1 のレベル 2 の CSA0,1 あるいは加算器ブロック 2 のレベル 1 の CSA0,1 に，同じく破線で示した加算器ブロック 2 のレベル 0 の CSA0,1 をそれぞれ併合することで接続し，これ以外の箇所に加算器ブロック 2 を接続しない．

加算器ブロック 1 と加算器ブロック 3 を接続して部分積加算部を設計した例を図 4.4 に示す．破線で囲んだ部分が各加算器ブロックを表している．図では，部分積加算部のレベル 2 において加算器ブロック 1 の出力を加算器ブロック 1 に接続し，レベル 4 において加算器ブロック 3 の出力を加算器ブロック 1 に接続している．

加算器ブロック 2 の接続についての制約により，図 4.5 のような，加算器ブロックの出力が別の加算器ブロックの入力において再収斂する部分積加算部は構成されない．図 4.5 では，加算器ブロック 2 の出力が 2 つの加算器ブロック 3 に枝分かれして入力し，加算器ブロック 1 において再収斂している．

本節の構成法を用いることで，ツリー型の高速な乗算器の部分積加算部に用いられる 4-2 加算木は図 4.4 のように構成できる．加算器ブロックの出力が再収斂することのある Wallace 木は設計できないものの，バランス木や図 4.2 のような Overturned-Stairs

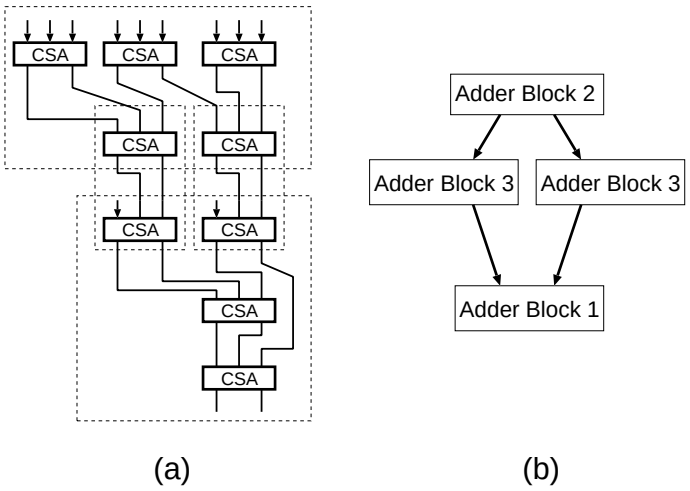


図 4.5: 加算器ブロック出力が再収斂する接続禁止の例 ((a) 構成，(b) 加算器ブロックレベルでの接続)

木等の様々な部分積加算部を設計できる．

4.3.2 加算器ブロックのテストのためのパターン集合

3 つの加算器ブロックについて，個々のブロックをテストするための入力パターン集合を示す．これらの加算器ブロックのテストのためのパターンから，部分積加算部全体のテストのためのパターンを設計する．本章でも部分積加算部のテストに交互反転パターンを用いる．

加算器ブロック 1 と 2 の入力部，そして加算器ブロック 3 のレベル 1 に位置する入力部への交互反転パターンを $(\alpha,\beta,\gamma)_{ai}$ $(\alpha,\ \beta,\ \gamma\in\{0,1\})$ で表す．交互反転パターン $(\alpha,\beta,\gamma)_{ai}$ は，CSA の FA j の u,v,w に，j が偶数なら α,β,γ を，j が奇数なら $\overline{\alpha},\overline{\beta},\overline{\gamma}$ をそれぞれ入力する．

加算器ブロック 3 のレベル 0 のビット入力端子 u への交互反転パターンを $(\alpha)_{ai}(\alpha\in\{0,1\})$ で表す．交互反転パターン $(\alpha)_{ai}$ は，CSA の FA j の u に，j が偶数なら α を，j が奇数なら $\overline{\alpha}$ を入力する．

図 4.6(a),(b) に加算器ブロック 3 に交互反転パターンを入力した例を示す．図では

表 4.1: 加算器ブロック 1 への入力パターン集合

	Inputs for Level 2			Level 1	Level 0
		CSA 0	CSA 1	CSA 0	CSA 0
t_1^1	p_a	p_2: $(010)_{ai}$	p_5: $(101)_{ai}$	p_6: $(110)_{ai}$	p_0: $(000)_{ai}$
$t_{\overline{1}}^1$	$p_{\overline{a}}$	p_5: $(101)_{ai}$	p_2: $(010)_{ai}$	p_1: $(001)_{ai}$	p_7: $(111)_{ai}$
t_2^1	p_b	p_6: $(110)_{ai}$	p_0: $(000)_{ai}$	p_0: $(000)_{ai}$	p_3: $(011)_{ai}$
$t_{\overline{2}}^1$	$p_{\overline{b}}$	p_1: $(001)_{ai}$	p_7: $(111)_{ai}$	p_7: $(111)_{ai}$	p_4: $(100)_{ai}$
t_3^1	p_c	p_0: $(000)_{ai}$	p_4: $(100)_{ai}$	p_3: $(011)_{ai}$	p_1: $(001)_{ai}$
$t_{\overline{3}}^1$	$p_{\overline{c}}$	p_7: $(111)_{ai}$	p_3: $(011)_{ai}$	p_4: $(100)_{ai}$	p_6: $(110)_{ai}$
t_4^1	p_d	p_3: $(011)_{ai}$	p_6: $(110)_{ai}$	p_0: $(000)_{ai}$	p_2: $(010)_{ai}$
$t_{\overline{4}}^1$	$p_{\overline{d}}$	p_4: $(100)_{ai}$	p_1: $(001)_{ai}$	p_7: $(111)_{ai}$	p_5: $(101)_{ai}$
t_5^1	p_e	p_0: $(000)_{ai}$	p_0: $(000)_{ai}$	p_2: $(010)_{ai}$	p_7: $(111)_{ai}$
$t_{\overline{5}}^1$	$p_{\overline{e}}$	p_7: $(111)_{ai}$	p_7: $(111)_{ai}$	p_5: $(101)_{ai}$	p_0: $(000)_{ai}$

加算器ブロックのレベル 1 の CSA に $(010)_{ai}$，レベル 0 の入力端子に $(1)_{ai}$ を入力している．図 4.6(a) の FA において，FA の番号が偶数のものを網かけで示している．

FA は入力のパターンが反転すると出力のパターンも反転する性質をもつ．そのため，各加算器ブロックにおいて入力部の CSA に交互反転パターンを入力すると，入力部の次のレベルの CSA においても交互反転パターンが入力される．したがって，加算器ブロックの入力部の CSA に交互反転パターンを入力すると，加算器ブロック内の全ての CSA に交互反転パターンが入力される．なお，CSA の両端のビット位置の扱いについては 4.4.2 で議論する．

加算器ブロック 1 のテストのために，交互反転パターンを用いた表 4.1 の t_1^1 から $t_{\overline{5}}^1$ の 10 個の入力パターンを用いる．各パターンの上付き文字は加算器ブロックを表し，下付き文字はパターンの番号を表している．表 4.1 では，レベル 2 の CSA に入力する入力パターンと，そのときレベル 1，レベル 0 の CSA に入力されるパターンを示

表 4.2: 加算器ブロック 2 への入力パターン集合

	Inputs for level 1			Level 0		
	CSA0	CSA1	CSA2	CSA0	CSA1	
t_1^2	p_e : p_0	p_0	p_0	p_a : p_2	p_5	
$t_{\overline{1}}^2$	$p_{\overline{e}}$: p_7	p_7	p_7	$p_{\overline{a}}$: p_5	p_2	
t_2^2	p_a : p_2	p_5	p_5	p_b : p_6	p_0	
$t_{\overline{2}}^2$	$p_{\overline{a}}$: p_5	p_2	p_2	$p_{\overline{b}}$: p_1	p_7	
t_3^2	p_b : p_6	p_0	p_6	p_c : p_0	p_4	
$t_{\overline{3}}^2$	$p_{\overline{b}}$: p_1	p_7	p_1	$p_{\overline{c}}$: p_7	p_3	
t_4^2	p_c : p_0	p_4	p_7	p_d : p_3	p_6	
$t_{\overline{4}}^2$	$p_{\overline{c}}$: p_7	p_3	p_0	$p_{\overline{d}}$: p_4	p_1	
t_5^2	p_d : p_3	p_6	p_3	p_e : p_0	p_0	
$t_{\overline{5}}^2$	$p_{\overline{d}}$: p_4	p_1	p_4	$p_{\overline{e}}$: p_7	p_7	

している．表に示すように交互反転パターン $(000)_{ai}, \ldots, (111)_{ai}$ にそれぞれ $p_0, \ldots, p_7$ と名前付けする．さらに，表 4.1 においてレベル 2 の CSA0 と CSA1 に入力されるパターンの組に $p_a, \ldots, p_{\overline{e}}$ と名前付けする．

同様に，加算器ブロック 2 のテストのために表 4.2 の t_1^2 から $t_{\overline{5}}^2$ の 10 個のパターンを用い，加算器ブロック 3 には表 4.3 の t_1^3 から $t_{\overline{5}}^3$ の 10 個のパターンを用いる．以降，特に特定のブロックに依存しない説明においては，これらのパターンを記す際，ブロック名を表す肩の数字を省き，$t_1, \ldots, t_{\overline{5}}$ と表す．

表 4.1, 表 4.2, 表 4.3 から，3 つの各ブロックについて，ブロックに対応するパターン集合のパターンを全て入力すると，ブロック内の全ての CSA に p_0, p_7 がそれぞれ 2 回，p_1 から p_6 までがそれぞれ 1 回ずつ入力される．このとき，CSA に含まれる各 FA には 000 から 111 までが全て入力される．また CSA の性質から，FA が故障すると演算結果は本来の値とは異なる値となるため，乗算器の出力において故障の影響を

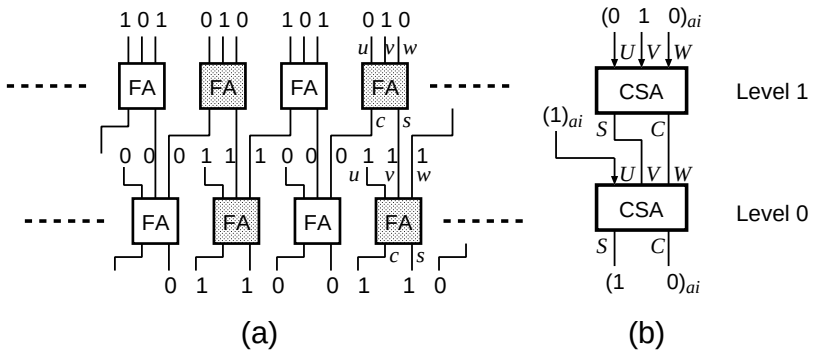


図 4.6: 交互反転パターンの入力例

観測できる．このことから，3 つの表の t_1 から $t_{\bar{5}}$ は加算器ブロックのテストのための入力パターンとなっている．

4.3.3 部分積加算部のためのテスト設計

加算器ブロックに，前節で示した入力パターンを入力するとき，加算器ブロック内の全ての CSA には交互反転パターン p_0,p_7 がそれぞれ 2 回，p_1 から p_6 までがそれぞれ 1 度ずつ入力される．つまり，加算器ブロックの入力部の CSA と，出力部 (レベル 0) の CSA の両者で，p_0 から p_7 までのそれぞれのパターンについての出現回数が同じである．

また，加算器ブロック 1 について前節で示した入力パターンを入力するとき，入力部の CSA0, CSA1 に $p_a, \ldots, p_{\bar{e}}$ のパターンが 1 度ずつ入力される．同様に，加算器ブロック 2 についても，前節で示した入力パターンを入力するとき，入力部 (レベル 1) と出力部 (レベル 0) の CSA0, CSA1 に $p_a, \ldots, p_{\bar{e}}$ のパターンが 1 度ずつ入力される．

これらの性質から，4.3.1 に示した設計法で部分積加算部を構成したとき，10 パターンで部分積加算部を構成する全ての加算器ブロックに，各加算器ブロックに対応する t_1 から $t_{\bar{5}}$ のパターンをすべて入力できる．

部分積生成部のテストのためのパターン集合は，根の加算器ブロックに t_1 から $t_{\bar{5}}$ のパターンを入力する 10 個の入力パターンから構成する．入力パターンは，部分積加

表 4.3: 加算器ブロック 3 への入力パターン集合

	Level 1	Level 0	
	Inputs for CSA0	Input for u	CSA0
t_1^3	p_0	$(1)_{ai}$	p_5
$t_{\overline{1}}^3$	p_7	$(0)_{ai}$	p_2
t_2^3	p_2	$(1)_{ai}$	p_7
$t_{\overline{2}}^3$	p_5	$(0)_{ai}$	p_0
t_3^3	p_6	$(1)_{ai}$	p_4
$t_{\overline{3}}^3$	p_1	$(0)_{ai}$	p_3
t_4^3	p_0	$(0)_{ai}$	p_1
$t_{\overline{4}}^3$	p_7	$(1)_{ai}$	p_6
t_5^3	p_3	$(0)_{ai}$	p_0
$t_{\overline{5}}^3$	p_4	$(1)_{ai}$	p_7

算部の根の位置にある加算器ブロックから葉に向かって CSA の入力パターンを順に求めてゆくことで構成する．つまり，ある加算器ブロックの入力部の CSA にパターン p を入力する必要がある場合，その CSA の接続元の加算器ブロックの出力部の CSA の入力が p となるように，t_1 から $t_{\overline{5}}$ の中から接続元の加算器ブロックへのパターンを選択する．このとき，4.3.1 の制約から加算器ブロックの出力が再収斂を起こす経路はないので，求まったパターンで不整合は起きない．

図 4.7 の配列型乗算器の部分積加算部を例に，テストのための入力パターンの構成法を示す．配列型乗算器の部分積加算部は加算器ブロック 3 を繰り返し接続することで構成できる．部分積加算部の根の加算器ブロックに，t_1^3 から $t_{\overline{5}}^3$ までの全てを入力するようにパターンの集合を構成する．図では，根の加算器ブロックに t_1^3 を入力する場合を示している．根の加算器ブロックに t_1^3 を入力する場合，図 4.7 のレベル 1 の CSA には $(000)_{ai}(=p_0)$ を入力する必要があり，表 4.3 より，加算器ブロックの出力部（レ

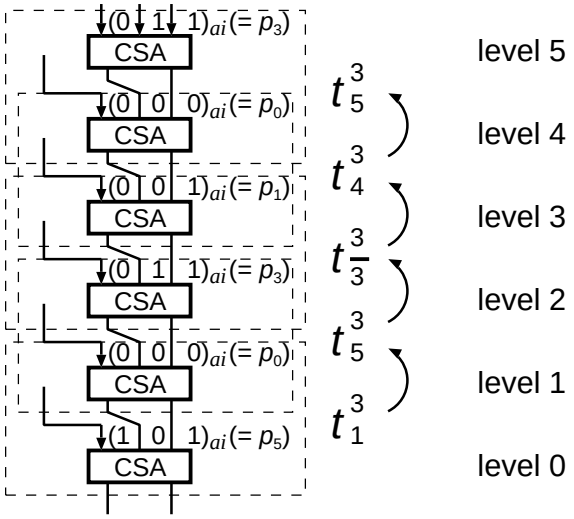


図 4.7: 配列型乗算器の部分積加算部の入力パターン設計の例

ベル 0）の CSA のパターンが p_0 となるものを，図 4.7 のレベル 1,2 の加算器ブロックへの入力割当とする．t^3_2 又は t^3_5 を入力することで図 4.7 のレベル 1 の CSA に p_0 を入力できる．この例では t^3_5 を用いている．図 4.7 のレベル 0,1 の加算器ブロックに t^3_4 を入力する際にも，レベル 1 の CSA に p_0 を入力するが，その際には t^3_2 を用いる．

残った部分についても，入力側に向かって入力パターンを求められ，部分積加算部の入力パターンが決まる．図 4.7 のレベル 4 の CSA にも p_0 を入力するが，このときレベル 4,5 の加算器ブロックには t^3_2 あるいは t^3_5 どちらを用いてもよく，例では t^3_5 を用いている．この場合，レベル 4 の CSA に p_0 を入力する別の入力パターンを構成する際にはレベル 4,5 の加算器ブロックに t^3_2 を用いる．

4.4 Cテスト可能な乗算器の設計手法

前節で示した部分積加算部設計手法を用いた，乗算器の構成法とテスト方法を示す．本章で提案する乗算器の構成を図 4.8 に示す．図中の d_1, d_2, r, r_0, r_1, r_2 は本節で導入するテストのための外部入力である．

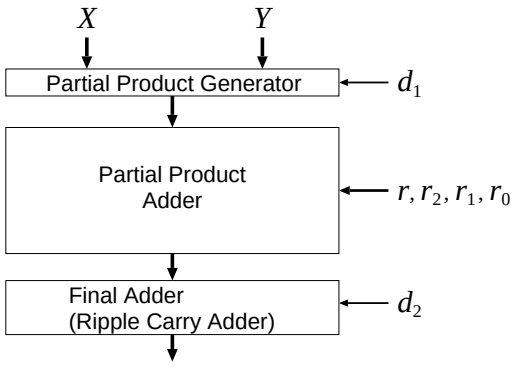


図 4.8: 乗算器の構成

4.4.1 部分積生成部の構成

部分積生成部として，交互反転パターンを生成するために，前章と同様の部分積生成部を用いる．奇数の j に対する部分積 P_j の生成回路を図 4.9(a) に，偶数の j に対する部分積 P_j の生成回路を図 4.9(b) に示す．部分積生成回路は外部入力信号 d_1 を導入し，前章と同様に CI セルと PG セルで構成する．部分積生成部は，$0 \leq i < N$, $0 \leq j < N$ について次の式のように部分積のビットを生成する．

$$pp_{ij} = \begin{cases} x_i \cdot y_j & (i+j \ \text{ is even}) \\ x_i \cdot (y_j \oplus d_1) & (i+j \ \text{ is odd}) \end{cases} \tag{4.1}$$

部分積加算部の葉の各 CSA には 3 つの部分積を入力する．CSA に P_g, P_{g+1}, P_{g+2} を入力するとき，CSA の入力ビット幅は $N+2$ ビットで，最小の入力重みは 2^g になる．このとき，CSA 内部の各 FA(入力重み 2^f, $g \leq f < g+N+2$) と部分積生成部を次のように接続する．

$$\begin{aligned} u^f &\leftarrow pp_{(f-g)g} \\ v^f &\leftarrow pp_{(f-g-1)(g+1)} \\ w^f &\leftarrow pp_{(f-g-2)(g+2)} \end{aligned} \tag{4.2}$$

ただし，$i < 0$ あるいは $i \geq N$ についての pp_{ij} は 4.4.2 で述べる入力補完ユニットに接続する．外部入力信号 d_1 には通常時に 0 を入力し，テスト時に 1 とする．

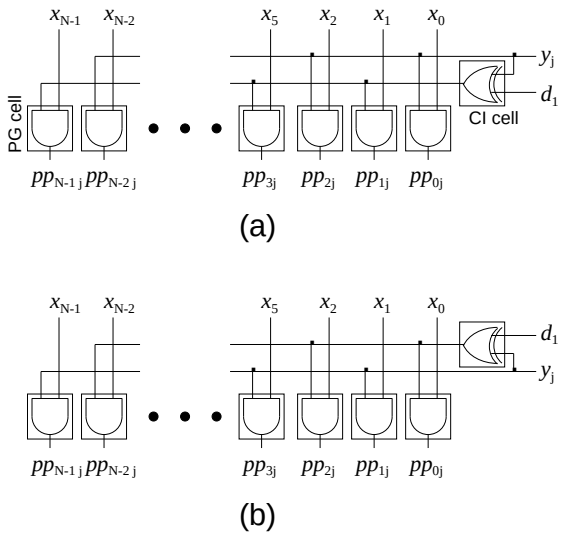


図 4.9: 部分積 P_j の生成回路 ((a) j が偶数の場合, (b) j が奇数の場合)

$d_1 = 1, X = 11\ldots1$ のとき，FA の入力ビットパターンは式 (4.1)，式 (4.2) より次の式で表される．

$$(u^f, v^f, w^f) = \begin{cases} (y_g, y_{g+1}, y_{g+2}) & (f \text{ is even}) \\ (\overline{y_g}, \overline{y_{g+1}}, \overline{y_{g+2}}) & (f \text{ is odd}) \end{cases}$$

上の式より，$d_1 = 1, X = 11\ldots1$ としたとき，部分積 P_g, P_{g+1}, P_{g+2} を入力とする CSA に y_g,y_{g+1},y_{g+2} の値に応じた交互反転パターンが入力されることが分かる．

加算器ブロック 3 のレベル 0 に位置する CSA の入力についても同様で，P_g を入力とすると，y_g の値に応じた交互反転パターンを入力できる．したがって，4.3.3 で求めた部分積加算部のテストのための入力パターンは部分積生成部を通して全て入力できる．

4.4.2 部分積加算部の構成

各部分積は，ビット幅が等しく，最小の入力重みがそれぞれ異なっている．そのため，部分積を入力とする CSA の両端のいくつかの桁では入力ビットが 2 つ，あるいは 1 つになる．これらの部分では入力が交互反転のパターンとならない．また，部分

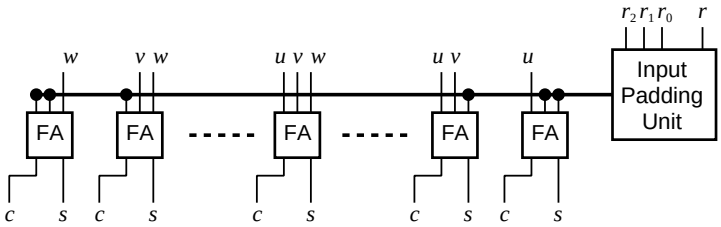


図 4.10: 桁上げ保存加算器と入力ビット補完器

積加算部の内部の CSA についても，CSA の両端の桁において入力ビットが 3 未満となる箇所がある．

このような入力ビットが 3 未満の桁について，交互反転パターンの規則性を保つために入力ビットを補完して入力し，CSA の各桁で入力ビットが必ず 3 つになるようにする．これらの入力ビットを生成するため，図 4.10 のように各 CSA に以下で説明する入力ビット補完器 (Input Padding Unit) を一つずつ接続する．入力ビット補完器は通常時には 0 を出力するため，通常時の演算結果に影響を与えない．

4.3.3 では，根の加算器ブロックに t_1 から $t_{\bar{5}}$ を入力するパターンをそれぞれ 1 つずつ設計し，テストのためのパターン集合を設計した．そのため設計したパターン集合において，部分積加算部の各 CSA の入力は，根の加算器ブロックの入力パターンと対応関係を持つ．つまり，パターンにより根の加算器ブロックに t_1 から $t_{\bar{5}}$ のどれが入力されるかが分かれば，各 CSA への入力が分かる．たとえば図 4.7 では，根の加算器ブロックにパターン t_1 が入力されると分かれば，レベル 2 の CSA の入力が $(0\ 1\ 1)_{ai}$ と分かる．そこで，外部入力線を用いて，入力パターンにより根のブロックに t_1 から $t_{\bar{5}}$ のどのパターンが入力されるかを与え，これらの外部信号線を用いて入力ビット補完器を設計する．

4 本の外部入力 r, r_0, r_1, r_2 を導入する．r は，テスト時に，部分積加算部の根の加算器ブロックへの入力パターンが $t_{\bar{z}} (z \in \{1,2,3,4,5\})$ になるテストパターンのときだけ 1 を入力する．一方，テスト時に部分積加算部の根の加算器ブロックへ t_z,$t_{\bar{z}}$ が入力される場合，r_0, r_1, r_2 には 2 進数 $(r_2\ r_1\ r_0)_2$ とみて z を入力する．つまり，根の加算器ブロックへの入力パターンが t_4 となるテストパターンでは $(r_2, r_1, r_0, r) = (1,0,0,0)$,

$t_{\bar{1}}$ となるテストパターンでは $(r_2, r_1, r_0, r) = (0, 0, 1, 1)$ とする．通常動作時には r や r_1, r_2, r_3 は 0 とする．

入力ビット補完器は PAD セルと CI セルを用いて構成する．PAD セルは 3 入力 1 出力のセルで，3 変数の論理関数を実現する．各 CSA の入力ビット補完器で PAD セルが実現する論理関数は異なり，設計時に静的に決める．PAD セルが実現する論理関数を PAD$(Z)(Z \subset \{0, 1, 2, \ldots, 7\})$ の形で記述する．セルの入力 n_0, n_1, n_2 を 2 進数 $(n_2\ n_1\ n_0)_2$ とみたとき，Z に含まれれば 1 を出力し，それ以外で 0 を出力することを意味する．

入力ビット補完器の 2 ビット分を図 4.11 に示す．根の加算器ブロックに $t_{z_i}(z_i \in Z_i \subset \{1, 2, 3, 4, 5\})$ が入力されるとき 1 となる信号は，外部入力信号 r_2, r_1, r_0 と PAD セル (PAD(Z_i)) を 1 つ用いて生成できる．根の加算器ブロックへの入力パターンが t_z から $t_{\bar{z}}$ になると，部分積加算部の内部の全ての FA の入力ビットは反転することから，PAD セルの出力を CI セルと r 信号を用いて反転させる．図中の各 PAD セルの機能は異なるため，PAD セルを PAD_0,PAD_1 と区別している．一つの入力ビット補完器は最大で 6 個の PAD セルを含む．

なお，PAD セルや CI セルの故障の影響は部分積加算部に伝搬し，乗算器の出力でその影響は観測できる．PAD セルに対する網羅パターンの入力については 4.4.4 で述べる．

4.4.3 最終加算部の構成

最終加算部として図 4.12 の順次桁上げ加算器 (RCA) を用いる．RCA の各ビットの入力ビット端子に a, b と名前付けする．RCA の FA には最下位より 0 から順番に番号をつけ，m 個目の FA に対応するビット位置の a, b 入力端子を a_m, b_m で表す．FA の入力端子には i_1, i_2, i_3 と名前づけする．a, b 端子と奇数番目の FA の i_2, i_1 端子の間には CI セルを 1 つずつ挟む．テストのために外部入力 d_2 を導入し，CI セルの制御に用いる．通常時，d_2 の入力は 0 とする．

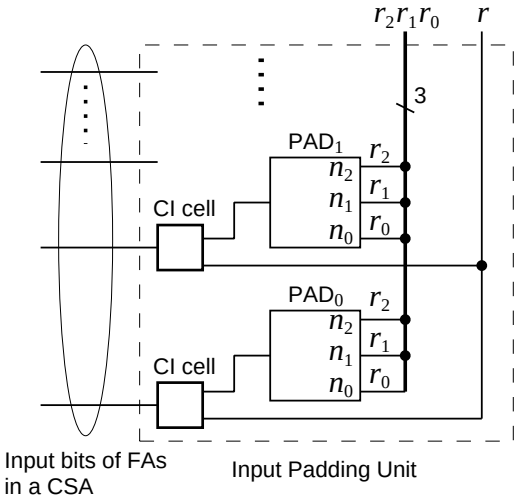


図 4.11: 入力ビット補完器の構成

最終加算部の入力は部分積加算部の根の CSA の出力である．4.3.3 で示した部分積加算部のテストのためのパターンでは，部分積加算部内の全ての CSA に交互反転パターン p_0, p_7 がそれぞれ 2 回，p_1 から p_6 までがそれぞれ 1 回入力される．この性質を利用し，部分積加算部のテストと同時に最終加算部もテストする．

CSA に交互反転パターン p_0, p_7 をそれぞれ 2 回，p_1 から p_6 までをそれぞれ 1 回入力すると，出力も交互反転パターンで得られ，CSA の各 FA は出力端子 c, s に $(c, s) = (0, 0), (1, 1)$ をそれぞれ 2 回，$(0, 1)$,$(1, 0)$ をそれぞれ 3 回出力する．そのため，最終加算部の各ビット位置で入力端子 a, b には $(a, b) = (0, 0), (1, 1)$ が 3 回ずつ，$(0, 1)$,$(1, 0)$ がそれぞれ 2 回ずつ入力され，最終加算部の入力パターンも交互反転のパターンとなっている．最終加算部のテストのための入力パターンを表 4.4 に示す．

表 4.4 では，1 列目に最終加算部の最下位のビット位置での入力ビットパターンを示している．入力パターンは交互反転のパターンなので，最下位のビット位置の入力ビットパターンで最終加算部全体のパターンが一意に決まる．2 列目に外部入力 d_2 に入力する値を示している．3 列目は最終加算部の最下位の 0 個目の位置にある FA の入力を示し，4 列目では 2 個目以降の偶数個目の FA の入力，5 列目では奇数個目の FA の入力を示している．

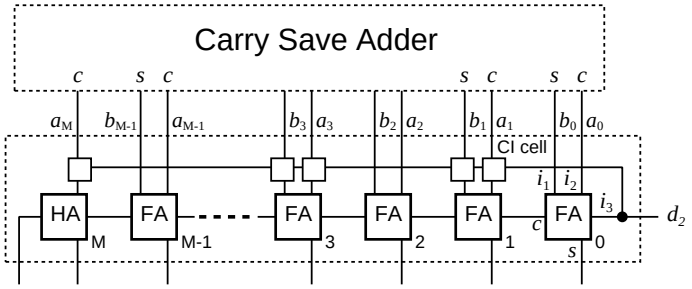


図 4.12: RCA を用いた最終加算部の構成

表より，最終加算部の全ての FA，CI セルに網羅的に全てのパターンを入力できることが確認できる．また，FA や CI セルが故障すると乗算器の出力が正しい値とならないため，最終加算部を構成するセルの故障は最終加算部の出力より確認できる．

ここでは，最終加算部として遅延時間の大きい RCA を用いたが，第 3 章の図 3.1 で示しように部分積加算部と最終加算器の間にマルチプレクサを挿入し，[10, 11] 等のテストが容易で高速な桁上げ伝播加算器を最終加算部として用いることも可能である．

4.4.4 C テスト可能な乗算器のテスト集合

提案手法により設計した乗算器において，部分積加算部のテストに交互反転パターンを用いる．4.4.1 で示した部分積生成部を用いることで部分積生成部で交互反転パターンを生成でき，4.3.3 で示した 10 個の交互反転パターンにより，部分積加算部をテストできる．また最終加算部として RCA を用いるとき，4.4.3 より，最終加算部と部分積加算部は同時にテストできる．

部分積加算部のためのテストパターン入力時に，外部入力信号 r_0, r_1, r_2 には $(r_2\, r_1\, r_0)_2$ が 1 から 5 までのみが入力される．そのため，入力ビット補完器内の PAD セルには入力されないパターンがある．また，外部入力信号 d_1 と X はそれぞれ 1,11 . . . 1 となっており，部分積生成部の PG セルや CI セルには入力されないパターンがある．

表 4.4: 最終加算部への入力パターン集合

	input values for the a_0b_0	d_2	Input values for $i_1i_2i_3$ of a FA		
			0-th	even numbered	odd numbered
t_{A1}	00	1	001	000	000
t_{A2}	00	0	000	001	110
t_{A3}	00	0	000	001	110
t_{A4}	01	0	010	010	100
t_{A5}	01	1	011	011	011
t_{A6}	10	0	100	100	010
t_{A7}	10	1	101	101	101
t_{A8}	11	1	111	111	111
t_{A9}	11	1	111	111	111
t_{A10}	11	0	110	110	001

そこで，以下の 4 つのパターンを追加する．“*” の箇所はドントケアを意味する．

$$
\begin{aligned}
&(X, Y, d_1, d_2, r, r_2, r_1, r_0) \\
&= (000\ldots0, 000\ldots0, 1, *, *, 0, 0, 0), \\
&\quad (000\ldots0, 111\ldots1, 1, *, *, 1, 1, 0), \\
&\quad (111\ldots1, 000\ldots0, 0, *, *, 1, 1, 1), \\
&\quad (111\ldots1, 111\ldots1, 0, *, *, *, *, *)
\end{aligned}
$$

この 4 つのパターンを先に述べた 10 個テストパターンと共に用いることで，部分積生成部の PG セルや CI セル，PAD セルに全ての入力組合せを入力できる．また，部分積生成部の PG セルや CI セルの故障の影響は乗算器の出力で観測できる．

部分積加算部と最終加算部のための 10 個のテストパターンと，上に示した 4 個の

テストパターンを用いることで乗算器全体のテストができる．テストパターンの数は演算のビット幅に依存せず 14 個のため，提案乗算器は C テスト可能である．このとき，テストのために必要な追加の外部入力は d_1, d_2, r, r_2, r_1, r_0 の 6 つである．

4.5 まとめ

乗算器の部分積加算部を 3 種類の加算器ブロックの組み合わせで設計し，ブロックの組み合わせかたにより様々なタイプの乗算器を設計できる C テスト可能な乗算器の設計手法を示した．提案手法で設計した乗算器は，最終加算部として順次桁上げ加算器を用いた場合，14 個のテストパターンでテストできる．

提案手法を用いることで，面積やスピードなどの要求に対して，性能に合わせたテスト容易な乗算器を設計できる．

第5章　CSAで構成した任意の構造の部分積加算部のレベルテスト可能性

5.1　はじめに

本章では，第 4 章の手法で対象外であった Wallace 木を含め，桁上げ保存加算器 (CSA) で構成された任意の構造の部分積加算部をもつ並列乗算器のテストについて考える．

文献 [2, 3, 4, 5] 等では，配列型乗算器の C テスト可能な設計を示している．文献 [6, 7] では，4-2 加算木のレベルテスト可能な設計を示している．本論文の第 3 章と第 4 章では，4-2 加算木などの C テスト可能な設計を示している．これまでの研究では，Wallace 木など，4-2 加算木よりレベル数が小さな部分積加算部について，オペランドのサイズとテストに必要なパターン数との間の関係が示されていなかった．

本章では，単一セル機能故障の仮定において，部分積加算部が，テストのための若干の回路の追加で，レベルテスト可能にできることを示す．部分積加算部の入出力間の CSA の段数の最大値をレベル数 L としたとき，任意の構造の部分積加算部が部分積生成部を含めて高々$6L+5$ 個のパターンでテストできることを示す．さらに，CSA 間の接続に制約を加えると，高々$2L+9$ 個のパターンでテストできることを示す．

最終加算器として用いる桁上げ伝搬加算器としては，文献 [10, 11] 等のテスト容易な構成を用いる．第 3 章の図 3.1 に示したように部分積加算部と最終加算器の間にマルチプレクサを挿入することで，部分積加算部と切り離してテスト可能な構成にで

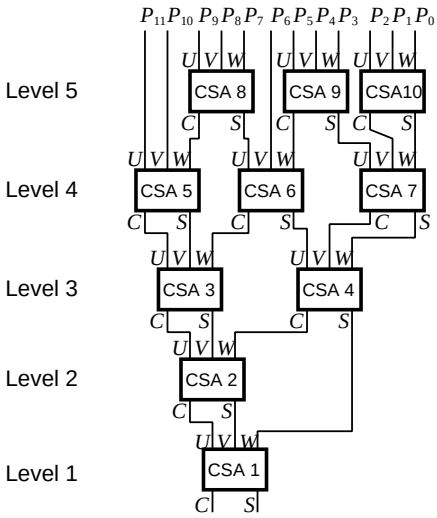


図 5.1: 部分積加算部の例

きる．

以降，次節でまず本章の準備を行う．5.3 節では任意の部分積加算部が部分積生成部を含めて高々$6L+5$ 個のパターンで，5.4 節では CSA 間の接続に制約を加えることにより，高々$2L+9$ 個のパターンでテストできることを，パターン集合の構成法を示すことで明らかにする．

5.2 準備

本章では，CSA を構成する各 FA について，図 5.2(a) のように入力端子に u,v,w と名前付けし，出力端子に c,s と名前付けする．また，CSA の入力にも図 5.2(b) に示すように U,V,W と名前付けし，出力に C,S と名前付けする．CSA の入出力 U,V,W,C,S はそれぞれ CSA の中の全ての FA の対応する端子 u,v,w,c,s からなる．

部分積加算部に図 5.1 のように出力側をレベル 1 として，入力に向かってレベル付けする．各 CSA の属するレベルは，その CSA の出力に接続する CSA の属するレベルの最大値より一つ大きな値とする．部分積加算部の最大のレベルがレベル数 L である．図 5.1 では，$L=5$ である．

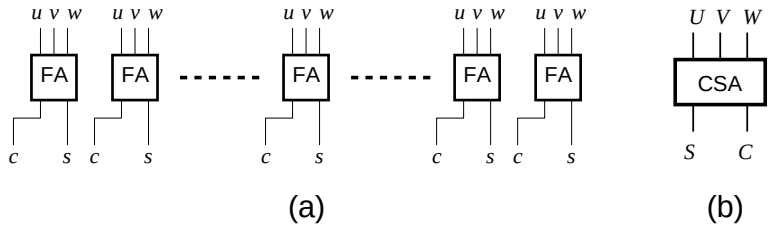


図 5.2: 桁上げ保存加算器

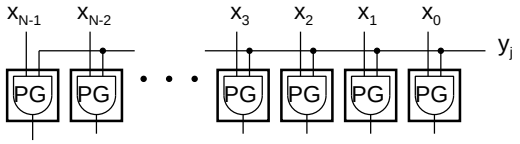


図 5.3: 部分積生成回路

5.3 任意の構成の部分積加算部のレベルテスト可能性

本章では，テストにおいて部分積加算部の各入力には部分積として $(00\ldots0)_2$ か $(11\ldots1)_2$ のどちらかのビットパターンを入力する．$(00\ldots0)_2$ と $(11\ldots1)_2$ のビットパターンをそれぞれを **0**, **1** で表す．

各部分積を生成する部分積生成回路を図 5.3 に示す．部分積生成回路は N 個の部分積生成セル (PG セル) で構成する．各 PG セルは AND ゲートで構成できる．被乗数を $X = (11\ldots1)_2$ とすると，部分積加算部へ入力される部分積 P_j のビットパターンは y_j が 0 か 1 かにより $(00\ldots0)_2$ か $(11\ldots1)_2$ となり，部分積加算部の各入力へ **0**, **1** を容易に入力できる．

部分積加算部の各入力を **0** か **1** にすると，部分積加算部の各 CSA の入力と出力も全て **0** か **1** となる．以後，CSA の入力パターンを CSA 内の FA の入力ビットを太字で記したもので表現する．例えば，CSA の中の FA に入力端子 u,v,w においてそれぞれ $0, 1, 1$ が入力されているとき，この CSA の入力パターンを **011** で表す．

部分積はビット幅が等しく，それぞれの部分積の最下位ビットの位置が異なるため，CSA の両端のビット位置において入力ビットが 2 つや 1 つになる箇所ができる．部

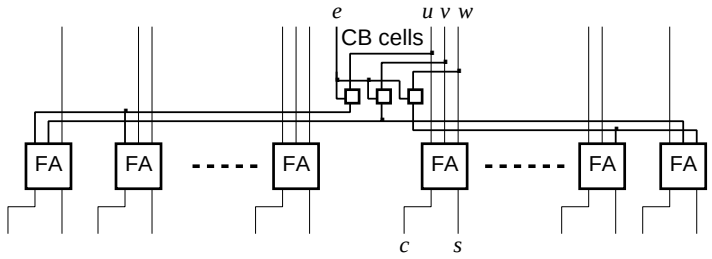


図 5.4: 入力ビットの補完をした CSA の例

分積加算部の内部の CSA についても入力ビットが 3 つにならないビット位置がある．これらのビット位置についてテストのために入力ビットを補完し，入力ビットパターンを CSA の他のビット位置と同じにする．このとき，CSA のこれらのビット位置でのビット加算にも FA を用いる．

入力ビットの補完は，CSA 内部の FA の入力ビットの配線を分岐させ，入力ビットが不足するビット位置に接続することで実現する．このとき，テスト時のみビットの補完を行うように，入力ビットの補完の配線に補完ビット制御セル (CB セル) を挿入する．CB セルを用いて入力ビットの補完をした CSA の例を図 5.4 に示す．外部入力信号 e を導入し，各 CB セルに接続することで，補完の制御ができる．通常時には $e = 0$ とし，部分積加算部のテスト時には $e = 1$ とする．CB セルは AND ゲートで実現できる．入力ビットの補完の配線は通常時 $(e = 0)$ に $X = Y = (11 \ldots 1)_2$ としたとき，FA の入力ビットが 111 となるビット位置から分岐させる．

部分積加算部のテストのために，部分積加算部の各 CSA に **000** から **111** までの 8 つのパターンの全てを入力できるテストの設計法を考える．

部分積加算部の全ての入力に **0** を入力すると，全ての CSA に **000** が入力される．同様に部分積加算部の全入力に **1** を入力すると，全ての CSA に **111** が入力される．したがって，2 つのパターンで全ての CSA に **000** と **111** が入力できる．

あるレベル l の全ての CSA に一斉にパターン p $(p \in \{\mathbf{001}, \mathbf{010}, \mathbf{011}, \mathbf{100}, \mathbf{101}, \mathbf{110}\})$ を入力することを考える．FA が実現する関数は全射なので，レベル $l + 1$ に属する CSA にレベル l に属する全ての CSA に p を入力するような入力パターンが必ず一つ

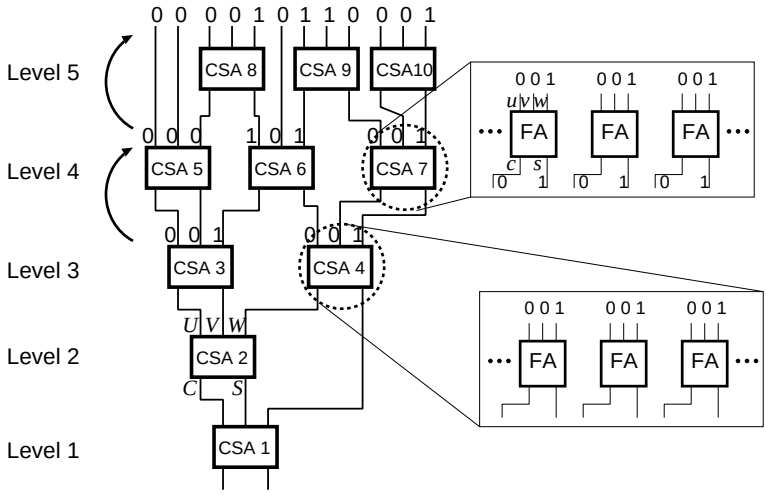


図 5.5: 部分積加算部への入力パターン構成例

は存在する．この性質を再帰的に用いると，レベル l に属する全ての CSA に一斉に p を入力する部分積加算部への入力パターンが得られる．

図 5.5 にパターンの設計例を示す．図では，レベル 3 に属する全ての CSA(CSA3, CSA4) に一斉に **001** を入力する入力パターンを構成する場合を示している．まず，CSA3 と CSA4 に **001** を入力するための，レベル 4 に属する CSA(CSA5, CSA6, CSA7) へのパターンを一つ求める．CSA5, CSA6, CSA7 へのパターンとして **000**, **101**, **001** や **000**, **110**, **010** などいくつかのパターンがあり，どれを選んでもよい．図では CSA5, CSA6, CSA7 へそれぞれ **000**, **101**, **001** を入力している．次に，レベル 5 に属する CSA8, CSA9, CSA10 に対するパターンを一つ求める．ここでは，それぞれに対してパターン **000**, **101**, **000** を用いる．このようにして，レベル 3 に属する全ての CSA に一斉に **001** を入力する部分積加算部への入力パターン $(P_{11}, \ldots, P_0) = (\mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1}, \mathbf{0}, \mathbf{1}, \mathbf{1}, \mathbf{0}, \mathbf{0}, \mathbf{0}, \mathbf{1})$ が得られる．このパターンを部分積加算部に入力するには被乗数と乗数を $X = (111111111111)_2$, $Y = (000010110001)_2$ とすればよい．

このように入力パターンを構成すると，一つの入力パターンで一つのレベルの全ての CSA に一斉に同じパターンを入力できる．部分積加算部の全ての CSA に 6 つのパターンを入力するために，各レベルに 6 つの入力パターンを生成すると，合計 $6L$ 個の入力パターンとなる．したがって，$6L + 2$ 個のパターンで部分積加算部をテストで

きる．この手法でパターンを構成すると，同じ入力パターンが重複して得られることがあるので，重複する場合は除去する．

PG セルと CB セルのテストについて考える．部分積加算部のテスト時に，X の各ビットは 1 で，$e = 1$ なので，PG セルと CB セルの一方の入力は常に 1 となる．PG セルと CB セルのために追加のパターンを用いる．$e = 0$ として，$X = Y = (00\ldots0)_2$ と $X = (00\ldots0)_2, Y = (11\ldots1)_2$ と $X = Y = (11\ldots1)_2$ の 3 つのパターンを用いる．1 つ目のパターンで PG セル，CB セルの両者について 00 を入力できる．2 つ目のパターンで PG セルに残りの入力パターンを入力できる．3 つ目のパターンで CB セルの補完ビット入力を 1 にできる．

部分積加算部において，故障した FA が重み 2^k のビット位置にあるとき，加算器の性質より，部分積加算部の出力値は真の値から $\pm 2^k$ または $\pm 2 \cdot 2^k$ または $\pm 3 \cdot 2^k$ だけ異なった値となる．したがって，故障した FA が存在するとその影響は乗算器の出力で観察できる．FA と同様に，故障した PG セルや CB セルの影響も乗算器の出力で観測できる．CB セルにより CSA 内の複数の FA に故障の影響が伝搬することがあるが，伝搬先は CSA の両端の最上位部と最下位部の複数のビットにわたるため，故障の影響が打ち消し合うことはない．

以上より，部分積加算部は部分積生成部を含めて高々$6L + 5$ 個のパターンでテストできる．これにより，任意の構成の部分積加算部がレベルテスト可能であることが明らかになった．

5.4　CSA 間接続に制約を加えた部分積加算部に対するテスト

5.4.1　CSA 間接続に制約を加えた部分積加算部

部分積加算部において,「各 CSA の 3 つの入力に，異なる 3 つの CSA の出力を接続しない」という制約を加える．図 5.6 に，この制約を満たす部分積加算部の例を示す．

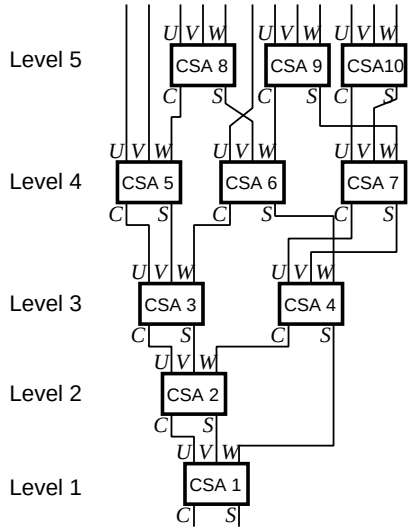


図 5.6: CSA 間接続に制約を加えた部分積加算部の例

以降，3 つの入力が全て部分積加算部の入力である CSA をタイプ 1，他の一つの CSA の出力 S,C の両方が入力する CSA をタイプ 2，これら以外の CSA をタイプ 3 とする．図 5.6 では，CSA8, CSA9, CSA10 はタイプ 1，CSA1, CSA2, CSA3, CSA4, CSA7 はタイプ 2，CSA5, CSA6 はタイプ 3 である．

テストの構成の説明を容易とするため，以下を仮定する．CSA の入力の対称性から，これらの仮定をしても一般性を失わない．

- 他の一つの CSA の出力 S,C の両方を入力するタイプ 2 の CSA では，これらの S,C はそれぞれ U, V に接続されている．
- タイプ 3 の CSA では，CSA の入力の 1 つが部分積加算部の入力であるときは，これは U に入力され，入力の 2 つが部分積加算部の入力であるときは，これらは U,V に接続されている．

タイプ 3 の CSA において，入力 U は必ず部分積加算部の入力である．図 5.6 は図 5.1 の CSA の入力の入れ換え (端子名の付け替え) により得られる．

1: P_0 から P_{N-1} が全て **0** の入力パターンと全て **1** の入力パターンを構成し，各 CSA g について $R(g)$ を $\{\mathbf{001}, \mathbf{010}, \mathbf{011}, \mathbf{100}, \mathbf{101}, \mathbf{110}\}$ とする

2: レベル L からレベル 1 へ各レベルに順に注目し，各レベルの CSA に以下の操作を行い 1 つの入力パターンを構成する．

   1. タイプ 1 の各 CSA について，U, V, W に入力する部分積をそれぞれ **1**, **0**, **0** とする
   2. W が部分積加算部の入力となっているタイプ 2 の各 CSA について，W に入力する部分積を **0** とする
   3. V が部分積加算部の入力となっているタイプ 3 の各 CSA について，V に入力する部分積を **0** とする
   4. タイプ 3 の各 CSA について，U に入力する部分積を，V,W の入力が **0**,**0** か **0**,**1** のとき **1**, それ以外のとき **0** とする

   得られた入力パターンの反転の入力パターンも構成し，各 CSA g について，$R(g)$ から 2 つの入力パターンにより入力されるパターンを除く

3: レベル L の全ての CSA に一斉に **010** を入力する入力パターンと，その反転の入力パターンの 2 つを構成し，レベル L の各 CSA g について，$R(g)$ から **010** と **101** を除く

4: **for** $l = L$ to 2 **do**

   レベル l と $l-1$ の CSA へ入力するパターンを次の手順で決め，5.3 節で述べた構成法で入力パターンを 1 つ構成する．

   1. レベル $l-1$ のタイプ 1 の各 CSA g に入力するパターンを，**001** とする
   2. パターンが決まっていないレベル $l-1$ の CSA g を一つ選択し，タイプ 2 なら $R(g)\backslash\{\mathbf{001}, \mathbf{110}\}$ の中の一方，タイプ 3 なら **001** を入力するパターンとする
   3. パターンが決まっていないレベル l の各 CSA g について，接続先のレベル $l-1$ の CSA のパターンが決まっていたら，矛盾しない出力となる入力を $R(g)$ より選び，入力するパターンとする
   4. パターンが決まっていないレベル $l-1$ の各 CSA g について，接続するレベル l の CSA のパターンが決まっていたら，CSA g がタイプ 2 なら $R(g)\backslash\{\mathbf{001}, \mathbf{110}\}$ から，タイプ 3 なら $\{\mathbf{001}, \mathbf{110}\}$ から，矛盾しないパターンを選び，入力するパターンとする
   5. (4) で新たにパターンが決まれば (3) へ戻る．(4) で新たにパターンが決まらず，レベル $l-1$ にパターンが決まっていない CSA があれば (2) へ戻る．

   得られた入力パターンの反転の入力パターンも構成し，レベル l, レベル $l-1$ の各 CSA g について，$R(g)$ から決定したパターンとその反転のパターンを除く

   end for

5: レベル 1 の CSA g に $R(g)$ の各パターンを入力するための入力パターンを，5.3 節で述べた構成法を用いて構成する

図 5.7: 入力パターン集合の構成手順

5.4.2 入力パターン集合の構成法

部分積加算部のテストのための入力パターン集合の構成アルゴリズムを図5.7に示す．アルゴリズムで構成した各入力パターンは，5.3 節で示した方法により，部分積生成部を通して部分積加算部に入力できる．

アルゴリズムでは，各 CSA g に対して，入力が済んでいないパターンの集合を表す $R(g)$ を用いる．

ステップ1では，全ての CSA に **000** と **111** を入力するための2つの入力パターンを構成する．ステップ1により，全ての CSA について $R(g)$ は $\{\mathbf{001}, \mathbf{010}, \mathbf{011}, \mathbf{100}, \mathbf{101}, \mathbf{110}, \mathbf{110}\}$ となる．

ステップ2では，部分積加算部の入力に直接接続する CSA に着目し，これらの入力を定めることにより，各 CSA に **100**, **010**, **011**, **101** のいずれかを入力する入力パターンを構成する．得られた入力パターンの反転の入力パターンも構成するため，ステップ2では合計2つのパターンを構成する．反転の入力パターンとは，各部分積を **0** と **1** とで反転した入力パターンである．FA が実現する関数は自己双対関数であり，FA の入力ビットパターンが反転すると出力ビットパターンも反転するため，部分積加算部の入力パターンを反転すると各 CSA に入力するパターンも反転する．

図5.8(a) にステップ2で構成した入力パターンを示す．図5.8(a) では，まずレベル5の CSA について，(1) より各 CSA の U,V,W に入力する部分積を **1**, **0**, **0** と決めている．次にレベル4の CSA について，(3) より CSA5 の V に入力する部分積を **0**，(4) より CSA5, CSA6 の U に入力する部分積をそれぞれ **1**, **0** とし，入力パターンを構成している．その反転の入力パターンが図5.8(b) である．

ステップ2で構成する2つの入力パターンにより，タイプ1の CSA には **100** とその反転の **011** を入力できる．タイプ2とタイプ3の CSA には，**100**, **010**, **101**, **011** のいずれかと，その反転のパターンの計2つを入力できる．したがって，各 CSA の $R(g)$ は $\{\mathbf{001}, \mathbf{010}, \mathbf{101}, \mathbf{110}\}$ か $\{\mathbf{001}, \mathbf{011}, \mathbf{100}, \mathbf{110}\}$ になる．レベル L の CSA は全てタイプ1であるため，$R(g)$ は前者となる．

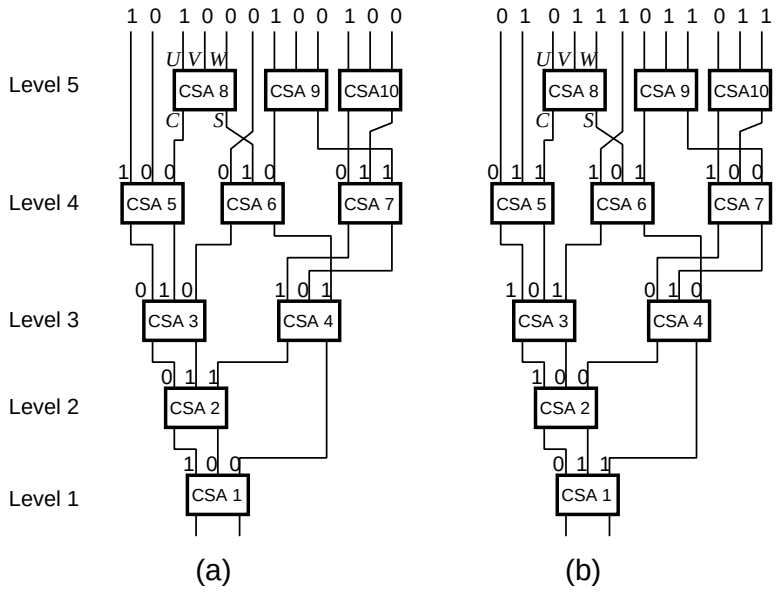


図 5.8: ステップ 2 による入力パターンの構成例

ステップ 3 ではレベル L の CSA に **101** を入力する入力パターンと，その反転の入力パターンを構成する．レベル L の CSA の $R(g)$ は $\{\mathbf{001}, \mathbf{110}\}$ になる．

ステップ 4 では，$l = L$ から 2 まで順に，隣接するレベル l と $l-1$ に注目してパターンを 2 つずつ生成していく．このとき，レベル l の各 CSA で R は空に，レベル $l-1$ の各 CSA で R の要素数が 2 となる．

図 5.9(a),(b) では，$l = 5$，つまりレベル 5 と 4 の CSA に注目した際のパターンの構成を示す．タイプ 3 の CSA を網かけで示している．図 5.9(a) では，(2) で CSA5 へのパターンを **001** に決めて，(3) で接続元の CSA8 の入力パターンを決めた状態を示している．CSA5 の W の入力は **1** なので，CSA8 の $R(8) = \{\mathbf{001}, \mathbf{110}\}$ から，CSA8 のパターンを **110** と決めている．図 5.9(b) は，図 5.9(a) からさらに (4) で CSA6，(3) で CSA9，(4) で CSA7, (3) で CSA10 と CSA のパターンを順に決めた後の状態を示している．

図 5.9(c) では，$l = 4$，つまりレベル 4 と 3 の CSA に注目した際のパターンの構成を示す．レベル 3 の CSA3, CSA4 の $R(g)$ はどちらも $\{\mathbf{001}, \mathbf{011}, \mathbf{100}, \mathbf{110}\}$，レベル 4 の CSA5, CSA6, CSA7 の $R(g)$ はそれぞれ，$\{\mathbf{010}, \mathbf{101}\}$, $\{\mathbf{011}, \mathbf{100}\}$, $\{\mathbf{001}, \mathbf{110}\}$ であ

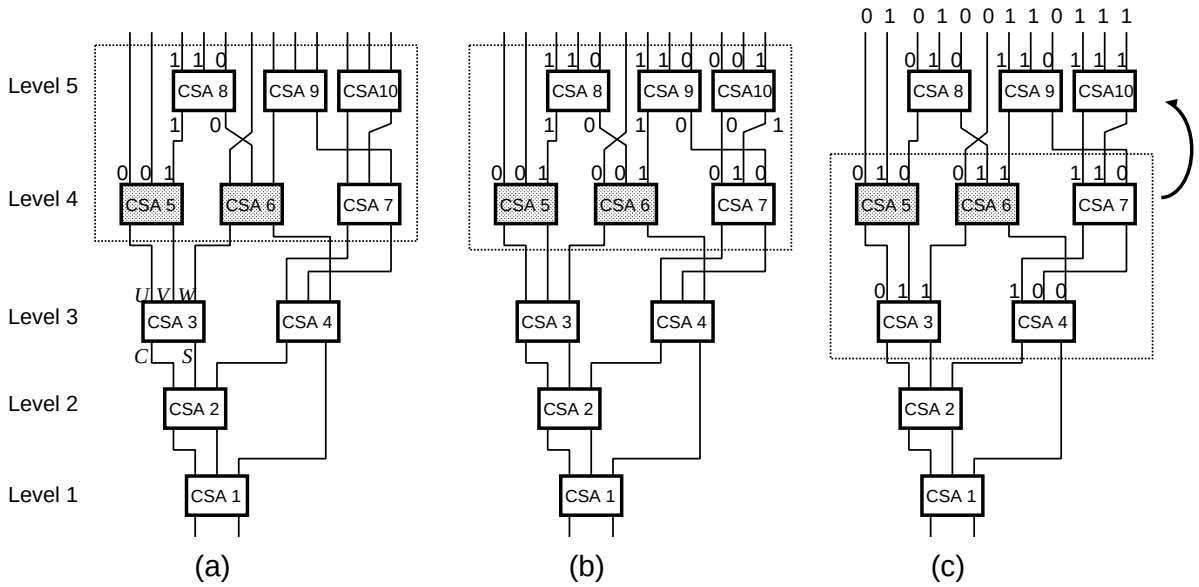


図 5.9: ステップ 5 による入力パターンの構成例 ((a),(b) レベル 5 と 4 へのパターンの構成, (c) レベル 4 と 3 へのパターンの構成)

る．このとき，レベル 3 の CSA3 に入力するパターンを (2) で **011** に決めると，CSA5, CSA6 のパターンが決まり，さらに CSA4，CSA7 の順にパターンが決まる．最後に，これらのパターンが入力されるように部分積加算部の入力パターンを求めている．

ステップ 4 ではレベル 1 の CSA に $R(g)$ の要素が 2 つ残る．ステップ 5 では 5.3 節で示したパターン構成法を用い，それぞれに対してパターンを構成する．

アルゴリズムでは，ステップ 1 からステップ 3 までで各 2 個，ステップ 4 で $2(L-1)$ 個，ステップ 5 で 2 個のパターンを生成する．したがって，部分積加算部のテストに合計で $2L+6$ 個のパターンを生成する．ステップ 4 で構成するパターンに重複があれば除去する．部分積生成部を含めた部分積加算部のテストに必要なパターン数は，5.3 節で示した部分積生成部と補完ビット制御セルのテストのための 3 パターンを加えた，高々$2L+9$ 個のパターンである．

5.4.3　入力パターン集合の最小性

CSA が全てタイプ 1 かタイプ 2 で，レベル $l(1 \leq l < L)$ の CSA が必ずレベル $l+1$ の CSA と接続する部分積加算部のテストには，5.4.2 で示した $2L+9$ 個のパターンが必要であることを示す．図 5.10 にそのような部分積加算部の例を示す．

このような部分積加算部では，レベル 1 の CSA の U, V 入力は，レベル 2 のある一つの CSA の C, S 出力と接続する．このレベル 2 の CSA の U, V 入力に接続する，レベル 3 のある一つの CSA があり，同様に繰り返すことでレベル L の CSA まで到達する．図 5.10 では，レベル 1 からレベル 5 にわたって，CSA1, CSA2, CSA3, CSA5, CSA8 の CSA で U, V 入力と S, C 出力が接続している．

これらの連続して接続する CSA のレベル 1 からレベル $L-1$ までに対して **001** と **110** を入力することを考える．レベル $l(1 \leq l < L)$ の CSA に **001** を入力するには，S,C で接続した CSA のうちレベル $l+1$ から L までの CSA の入力は全て **000** とする必要がある．そのため，これらの CSA の複数に同時に **001** を入力することはできない．したがって，レベル 1 からレベル $L-1$ までの CSA に **001** と **110** を入力するために $2(L-1)$ 個のパターンが必要である．

図 5.10 では，例としてレベル 2 の CSA2 に **001** を入力する場合の各 CSA の入力を示している．レベル 3 からレベル 5 までの CSA3, CSA5, CSA8 の各入力は全て **000** となる．

$2(L-1)$ 個のパターンでは U, V 入力と S, C 出力が接続した CSA のうちレベル L の CSA には **000** と **111** しか入力できず，レベル 1 の CSA には **000** と **111** を入力できない．レベル L の CSA に残りのパターンを入力する 6 個のパターンと，レベル 1 の CSA に **000** と **111** を入力する 2 個のパターンが必要となる．そのため，合計して $2L+6(=2(L-1)+6+2)$ 個のパターンが必ず必要である．このことから，上述の部分積加算部に **0**,**1** を入力してテストするには，少なくとも $2L+6$ 個の入力パターンが必要である．

以上より，部分積加算部と部分積生成部のテストに $2L+9$ 個のパターンが必要と

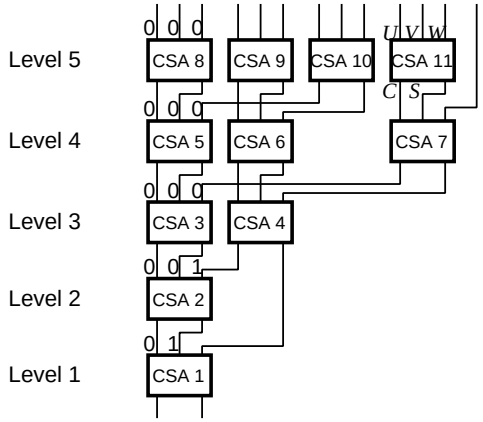


図 5.10: CSA への **001** の入力の例

なる.

5.5 まとめ

並列乗算器の主要な構成回路である部分積加算部について，若干の回路を付加により，レベルテスト可能となることを示した．まず，任意の構成の部分積加算部が部分積生成部を含めて高々$6L+5$ 個のパターンでテストできることを示し，さらに，部分積加算部の構成に制約を加えると高々$2L+9$ 個のパターンでテストできることを示した．これにより，Wallace 乗算器など，4-2 加算木より CSA の段数が小さな乗算器について，レベルテスト可能であることがはじめて明らかになった.

第6章　結論

本論文では，並列乗算器のテストに関する研究成果をまとめた．

第3章では，4-2加算器を用いたテスト容易な乗算器を示した．4-2加算器のテストのために，交互反転パターンとよぶ隣接するビット位置に互いに反転したビット入力を印加するパターンを提案した．交互反転パターンを用いることで，本稿で示した構成をもつ4-2加算器がCテスト可能で，10個のパターンでテストできることを示した．部分積生成部への若干の回路の追加により，部分積生成部で交互反転パターンが生成できることを示し，乗算器のビット幅によらず，部分積生成部と4-2加算器を合わせて14個のパターンでテストできることを示した．既存のテスト容易な桁上げ伝搬加算器を最終加算器として用いることでCテスト可能な乗算器を設計できる．これまで，配列型乗算器のCテスト可能な設計法は知られていたが，高速なツリー型乗算器のCテスト可能な設計法は知られていなかった．本提案により，Cテスト可能なツリー型乗算器を設計可能であることが明らかになった．提案乗算器を用いることで，高速乗算器を少ないパターンでテストすることが可能になる．

第4章では，さまざまな部分積加算部を構成可能なテスト容易な乗算器の設計手法を示した．提案手法では，乗算器の部分積加算部を3種類の加算器ブロックの組み合わせで設計し，ブロックの組み合わせかたにより様々なタイプの乗算器を設計できる．提案手法で設計した部分積加算部は，交互反転パターンを用いることで，ビット幅によらず10個のパターンでテストできることを示した．最終加算部として順次桁上げ加算器を用いた場合，乗算器はCテスト可能で，14個のテストパターンでテストできた．提案手法を用いることで，面積やスピードなどの要求に対して性能に合わせたテスト容易な乗算器の設計ができるため，テスト容易な乗算器の自動合成ツールの実

現が可能になる．現状の演算器 IP(Intellectual Property) ライブラリの乗算器の設計ではテストを考慮していないが，提案法によりテストを意識した乗算器の構成が可能になり，テスト時間の短縮など VLSI のテスト容易化に貢献すると考えられる．

第 5 章では，第 4 章の手法の対象外であった Wallace 木を含め，任意の構成の部分積加算部が若干の回路の追加によりレベルテスト可能になることを示した．まず，部分積加算部の CSA の段数を L とすると，任意の構成の部分積加算部が部分積生成部を含めて高々$6L + 5$ 個のパターンでテストできることを示した．さらに，部分積加算部の構成に制約を加えると高々$2L + 9$ 個のパターンでテストできることを示した．これにより，Wallace 乗算器など，4-2 加算木より CSA の段数が小さな乗算器について，部分積加算部がレベルテスト可能であることがはじめて明らかになった．ただし，C テスト可能性については判明しておらず，ビット幅が大きい乗算器を構成する際，Wallace 乗算器は 4-2 加算木を用いた乗算器よりもテストに必要なパターン数が多くなる．VLSI 設計において，レイアウトの容易さや遅延時間と同時に，テストの容易さを考慮し，乗算器を選択することが大切になると考えられる．

この研究から，どのような部分積加算構造をもつ乗算器についても，若干の回路の付加によりテストが容易にできることが明らかになった．また，第 3 章と第 4 章より，交互反転パターンが乗算器のテストに有用であることがわかった．乗算は四則演算のなかでも使用頻度が高く，乗算器は VLSI チップに搭載されることが多い．特に現在では，コンピュータグラフィックの演算や科学技術計算の高速化のために多数の乗算器を一つのチップに搭載した VLSI チップが実用されている．本論文で示した乗算器や本論文の手法で構成した乗算器を用いることで，チップ全体のテストの容易化ができると考えられる．

本研究で得た乗算器のテストに対する知見は，加算器や除算器などの乗算器以外の演算回路のテスト容易化法の確立や，遅延故障や stuck-open 故障などの機能故障を越えた様々な故障に対する演算回路のテスト方法の確立のための基礎となると期待される．

本論文では，種々の乗算器のテストを考えた．これらの乗算器を回路設計で用いる際には，そのなかから設計上の要求にあった乗算器を用いる必要がある．面積やスピード，テストの容易さなどの要求に対して乗算器を自動で合成するための手法の開発が今後の課題として挙げられる．また，複数の故障が同時に存在した場合に検出できるかどうかは判明しておらず，CSA の段数が 4-2 加算木より小さな乗算器について，C テスト可能であるかどうかは判明していない．これらの点についてさらなる研究がのぞまれる．

謝辞

本論文は，名古屋大学大学院情報科学研究科に在籍した期間に行われた研究をまとめたものです．本研究を遂行するにあたり，多くの方々の御指導，御支援を頂きました．ここに心からの感謝の意を表します．

とりわけ，指導教員である高木直史教授には日頃より研究についての御指導と御鞭撻を頂きました．また，様々な議論を通し，幅広い知識に基づく貴重な御意見を頂きました．深く感謝の意を表します．

名古屋大学大学院情報科学研究科の高浜盛雄教授には，本論文に対し的確な御助言と御示唆を頂き，本論文をまとめる上で大変有益なものとなりました．深く感謝の意を表します．

高木一義准教授には，日頃の論文執筆において的確な批評と御助言を頂きました．また，研究について議論をして頂きました．深く感謝の意を表します．

中村一博助教には，研究活動を進めるにあたり多くの御支援を頂きました．熊澤文雄博士，小畑幸嗣博士をはじめとする研究室の諸先輩方には，研究や研究に対する心構えについての多くの助言を頂きました．小林克希氏，川島裕崇氏をはじめとする研究室の皆様には日頃から様々な助言を頂きました．深く感謝の意を表します．

参考文献

[1] ITRS 2007: International technology roadmap for semiconductors 2007 edition test and test equipment, 2007.

[2] J. Shen, and F. Ferguson, "The design of easily testable VLSI array multipliers," IEEE Trans. on Computers, vol.33, pp.554-560, June 1984.

[3] A. Chatterjee, and J. Abraham, "Test generation, design-for-testability and built-in self-test for arithmetic units based on graph labeling," Journal of Electronic Testing, vol.2, pp.351-372, Nov. 1991.

[4] D. Gizopoulos, D. Nikolos, A. Paschalis, and C. Halatsis, "C-testable modified-booth multipliers," Journal of Electronic Testing, vol.8, pp.241-260, June 1996.

[5] K.O. Boateng, H. Takahashi, and Y. Takamatsu, "Design of C-testable modified-booth multipliers," IEICE Trans. on Inf. and Syst., vol.E83-D, no.10, pp.1868-1878, 2000.

[6] B. Becker, "An easily testable optimal-time VLSI-multiplier," Acta Informatica, vol.24, pp.363-380, 1987.

[7] P. Zeng, Z. Mao, Y. Ye, and Y. Deng, "Test pattern generation for column compression multiplier," Proc. Seventh Asian Test Symposium, pp.500-503, 1998.

[8] D. Gizopoulos, A. Paschalis, and Y. Zorian, "An effective built-in self-test scheme for parallel multipliers," IEEE Trans. on Computers, vol.48, pp.936-950, Sep. 1999.

[9] B. Becker, R. Drechsler, and P. Molitor, “On the generation of area-time optimal testable adders,” IEEE Trans. on CAD, vol.14, pp.1049-1066, Sep. 1995.

[10] R. Blanton, and J. Hayes, “Testability of convergent tree circuits,” IEEE Trans. on Computers, vol.45, pp.950-963, Aug. 1996.

[11] D. Gizopoulos, M. Psarakis, A. Paschalis, and Y. Zorian, “Easily testable cellular carry lookahead adders,” Journal of Electronic Testing, vol.19, pp.285-298, June 2003.

[12] B. Becker, “Efficient testing of optimal time adders,” IEEE Trans. on Computers, vol.37, pp.1113-1121, Sep. 1988.

[13] R. Blanton, and J. Hayes, “On the design of fast, easily testable ALU’s,” IEEE Trans. on Very Large Scale Integration (VLSI) Systems, vol.8, pp.220-223, Apr. 2000.

[14] 鬼頭信貴，花井健輔，高木直史，“4-2 加算木を用いたテスト容易な乗算器,” 信学技報，vol.106，no.549，pp.1-6，Mar. 2007.

[15] J. Vuillemin, “A very fast multiplication algorithm for VLSI implementation,” Integration the VLSI journal, vol.1, pp.39-52, 1983.

[16] Z.J. Mou, and F. Jutand, “Overturned-stairs adder trees and multiplier design,” IEEE Trans. on Computers, vol.41, pp.940-948, Aug. 1992.

[17] S. Pezaris, “A 40-ns 17-bit by 17-bit array multiplier,” IEEE Trans. on Computers, vol.20, pp.442-447, Apr. 1971.

発表論文

学術雑誌論文 (査読付き)

1. 鬼頭信貴, 高木直史, “種々の部分積加算構造を持つテスト容易な乗算器の設計手法,” 電子情報通信学会論文誌 (D), Vol.J91-D, no.10, pp.2478–2486, 2008 年 10 月

国際会議論文 (査読付き)

1. Nobutaka Kito and Naofumi Takagi, “Level-testability of multi-operand adders,” Proc. of the Seventeenth Asian Test Symposium, pp.257–260, Nov. 2008

学会での口頭発表 (査読なし)

1. 鬼頭信貴, 花井健輔, 高木直史, “4-2 加算器を用いた乗算器のためのテスト容易化設計,” 第 56 回 FTC 研究会, 2007 年 1 月
2. 鬼頭信貴, 花井健輔, 高木直史, “4-2 加算木を用いたテスト容易な乗算器,” 電子情報通信学会技術研究報告, VLD2006-140, pp.1–6, 2007 年 3 月
3. 鬼頭信貴, 高木直史, “C テスト可能な乗算器の設計手法,” 第 57 回 FTC 研究会, 2007 年 7 月
4. 鬼頭信貴, 高木直史, “種々の部分積加算構造に対応したテスト容易な乗算器の設計手法,” 電子情報通信学会技術研究報告, DC2007-38, pp.7–12, 2007 年 11 月

5. Nobutaka Kito and Naofumi Takagi, “Test generation for multi-operand adders consisting of full adders,” 電子情報通信学会技術研究報告, DC2008-14, pp.19–22, 2008 年 6 月

6. 鬼頭信貴, 高木直史, “要求性能を考慮したテスト容易な乗算器の自動合成,” 第 59 回 FTC 研究会, 2008 年 7 月