平成 20 年度
実績報告

「ディペンダブル VLSI システムの基盤技術」
平成 19 年度採択研究代表者

坂井　修一

東京大学大学院情報理工学系研究科　教授

研究題目

1. 研究実施の概要

本研究課題は、検証技術とディペンダブルアーキテクチャ技術の２つを核としてこれにテスト技術・回路技術を加え、それぞれを新規に研究開発するとともに、これら諸技術の協調・融合によって、個々の技術では達成できないディペンダビリティを VLSI 上に実現する技術を研究開発する。このように、個々の技術を磨きながら各技術の協調融合によってかつてなかったディペンダビリティを実現することは、JST 研究開発戦略センターの戦略イニシャティブ「情報化社会の安全と信頼を担保する情報技術体系の構築 ―ニュー・ディペンダビリティを求めて―」にある「従来のようにフォールトを予防する努力だけでは不十分であり、フォールトの存在を前提とする情報化社会のデザインあるいは情報システムの設計方法論が必要である。たとえ一部の要素にフォールトが発生したとしてもシステム全体としては期待どおりの良質で確かなサービスを提供し続けるディペンダブルな情報システムを実現する新しい情報技術の体系が求められる」という記述の実現をめざしたものである。

2 年度目である平成 20 年度は、形式的検証方式・テスト方式の検討とツール群の開発、回路要素技術・プロセッサアーキテクチャ要素技術の評価、メニーコア用ディペンダブル高機能ルータの検討などを行った。

【形式的検証】

- 等価性検証および実行時間見積り手法のツール開発および評価を行った。数百行程度の設計例題の等価性を数秒間で検証することに成功した。また、シミュレーションと組合せて用いることにより、実行時間を全体の実行時間の 3%程度の範囲の中に見積もること可能であることが確認できた。今後は、主に企業から提供された設計例題を用いたツールの評価と改善を行う予定である。
- 従来から論理関数の充足性判定手法（SAT 手法）や２分決定グラフ（BDD）を利用した検証手法が研究されてきているが、扱う式が複雑なりすぎるため、掛算を中心とした算術演算回路の検証には相対的に向いていない。そこで、多項式を一般的に取り扱

う決定グラフである Horner Decision Diagram（HED）を拡張することで、掛算を含む複雑な算術式間の検証を行なう手法を開発した。また、算術式と論理回路間の対応を効率的に取る手法を新規に開発し、仕様レベルから論理回路に至る算術演算回路の設計を支援する手法を開発した。

- 現在の効率的なモデル検査手法の1つに、コンパイルドシミュレーションを基にした準形式的限定モデル検査がある。我々はそのアルゴリズムの一部をハードウェアで実装し、各々の処理に対してハードウェア・ソフトウェア協調実行に向けた最適化を行うことで高速化したモデル検査を提案した。例題を用いた実験によって実際に高速に検証を行い、提案手法の有効性を確認した。また検証指向FPGAの提案を行い、設計・試作を行った。
- 2つの高位設計記述間の内部変数の等価点を前提として、静的な依存関係や制御フローに基づいて定義された等価性規則をボトムアップに適用することで等価性を示すルールベース等価性検証を提案した。この手法では変数の名前によって内部等価点を求めるため，変数名の変更などにより対応が取れない場合には等価性を証明できない。そこで、ランダムシミュレーションにより内部等価点を推定することによって、より正確な検証を実現する手法を提案した。また、例題を用いた計算機実験では提案手法が現実的な最適化を適用した場合の等価性を高速に判定することが可能であることを示した。
- SpecC記述を対象とした、具体値および記号値を用いたシミュレーションによる検証・デバッグ環境を構築し、その上で具体シミュレーション(深さ優先探索)と記号シミュレーション(幅優先探索)を切り替えながら使用して探索するユーザ駆動幅優先探索手法を提案した。ケーススタディでは、400行程度の記述を用いて、100万回のランダムシミュレーションや6サイクル程度の記号シミュレーションでは発見できなかったアサーション違反を、本手法により発見できることを示した。今後は、より実用的で大規模な記述に対応するために、さらなる探索手法を考案する予定である。

【回路・プロセッサアーキテクチャ】

●タイミング故障耐性を持つクロッキング方式

前年度までの提案手法には、ショートパスを通った速い信号によってクリティカルパスを通った遅い信号が上書きされるという問題点があった。本年度は、ショートパスとクリティカルパスを物理的に分けることでこの問題を解決する提案を行った。今後はより詳細な回路の設計を行う。

●タイミング故障耐性を持つプロセッサ構成方式

タイミング故障耐性を持つスーパスカラプロセッサを FPGA 上に実装して、故障を注入することにより、タイミング故障を検出し回復が可能であることを確認した。今後は、実用化に必要なより詳細なデータの収集を行っていく。

●永久故障耐性を持つ FPGA アーキテクチャ

前年度に作成したテストベッド上の FPGA に提案手法を実装中である。このテストベッドは、ユーザロジックを実装するものと、故障の検出と回復を行う固定機能を担う

ものの 2 種類の FPGA を持つ。本年度は、後者の FPGA の設計がほぼ完了し、後者の FPGA から前者の FPGA が再構成可能であることを確認した。今後数カ月以内に実装を完了し、提案手法の動作の確認を行う。

【マルチコア用高機能ルータ】

平成 20 年度は、ディペンダビリティを支援する高機能ルータアーキテクチャの基本部および方式評価のためのメニーコアシミュレータの開発を行った。

ディペンダビリティ支援ルータのアーキテクチャに関しては、冗長実行を支援するためのパケットの複製、同一性検出、マージの機能を実現する基本方式を開発し、その動作確認を行った。

高機能ルータアーキテクチャを評価するためのメニーコアプロセッサアーキテクチャを構築した。また、このアーキテクチャの動作をクロックレベルで模倣するマルチコアプロセッサシミュレータの基本モジュールの実装を行った。

研究計画の一部を前倒しして、32 ビットの RISC プロセッサを 1 チップとして、また、基本機能をもつルータ 4 個を 1 チップとして試作（CMOS 0.18um）を行った。

2. 研究実施内容（文中にある参照番号は 4.(1)に対応する）

2.1 ディペンダブルアーキテクチャグループ

ディペンダブルアーキテクチャグループでは、回路技術とアーキテクチャ技術によって VLSI の信頼性を飛躍的に向上させる技術の研究開発を行っている。

平成 20 年度は、第一に、ディペンダブル回路として、タイミング故障耐性を持つクロッキング方式の提案を行った。

本研究で昨年度に提案した遅延補償 FF は、信号の遷移を表すパルスを用い、フリップフロップ（以下、FF と略）に入力されるクロックとデータの信号遷移タイミングを評価したもの（図１）であり、これによって既存研究の Razor、Canary FF の問題点（ショートパス問題、マージンの大きさの問題など）が解決された。また、最終的な結果に影響を及ぼさないようなタイミング故障を許容することにより、タイミング制約を緩和することができた。遅延保証 FF によって、電源揺らぎや、製造時のプロセスばらつきによって発生する予測しがたいタイミング故障を回路動作時に自動的に回避することが可能となった。また、動作環境に合わせて電源電圧を調整できることは省電力化効果ももつことになる。さらに、遅延補償 FF は回路のタイミング設計自由度を高く保つことができる。本年度は、遅延補償 FF の回路規模評価、実現性検証を進めるとともに、欧文誌での発表[1]などを行った。

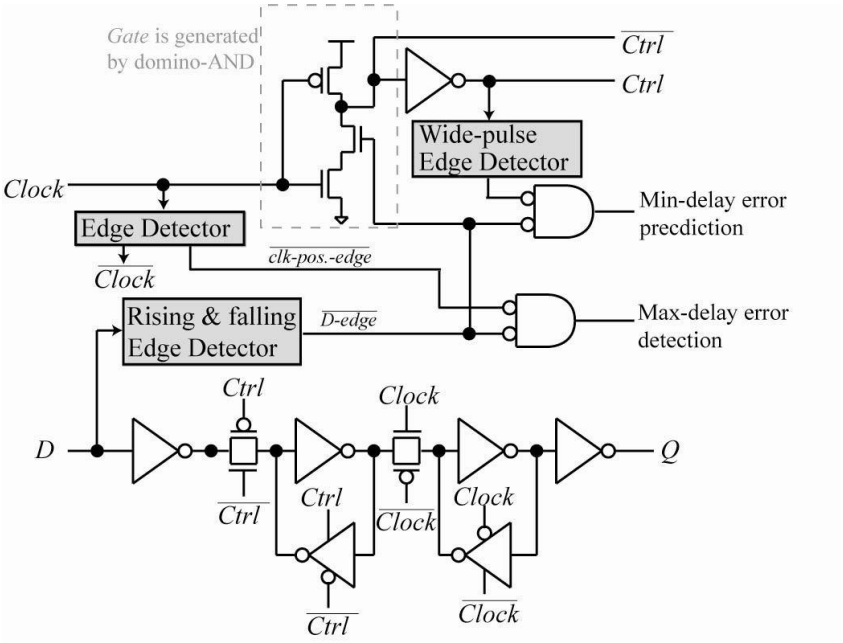


図１．タイミング故障を動的に検知・修正する FF

本年度は、この遅延補償 FF の概念を更に拡張した耐タイミング故障クロッキング方式を提案した。遅延補償 FF では、ショートパスを通った速い信号によって現在の信号が上書きされてしまう問題がある。そのため、ショートパスとクリティカルパスを物理的に分離することによって、この問題の解決を図る。

図 2 に、遅延補償 FF を用いた回路の動作を示す。同図中、縦軸は時間を、横軸は回路内の信号の伝達方向を示す。同図には、3 つのエッジトリガ FF に対応する 3 本の縦線が引いてある。あるステージを通過する信号はクロックエッジにおいて FF を出発し、次のクロックエッジまでに次の FF に到着しなければならない。遅延補償 FF は、同図中の赤矢印で示したように、次のクロックエッジまでに間に合わなかった信号をも通過させることで制約を緩和するものである。

しかしこの方式では、遅れてきた前のフェーズの信号（同図中赤矢印）と早く到着した次のフェーズの信号（水色矢印）が交差することがある。その場合、ロジックの出力は「混じ」って、意味のないものになる。

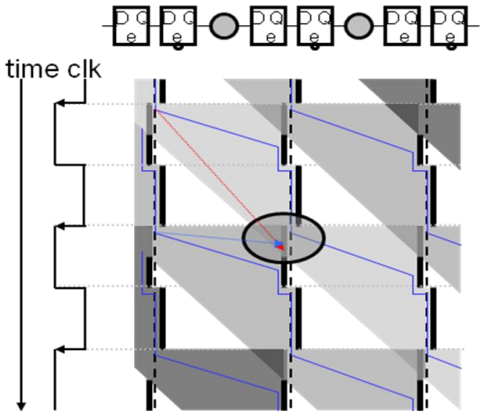


図２．耐タイミング故障クロッキング

ショートパスとクリティカルパスを物理的に分離するのは、この現象を回避するためである。物理的に分離されていれば、ショートパスを通過してきた赤矢印とクリティカルパスを通過してきた水色矢印が時間的に交差していても、「混じる」ことはない。

なお、遅延を挿入するなどして、すべての信号を遅らせるような方法では、所望の効果は得られないことに注意されたい。提案手法では、クリティカルパスを通ってきた信号の遅れは、次のステージでショートパスを通過することによって保証される。

本年度はまず、この方式を実現するための要件を調べ上げた。さらに、それを実現するための回路構成を開発した（図３）。

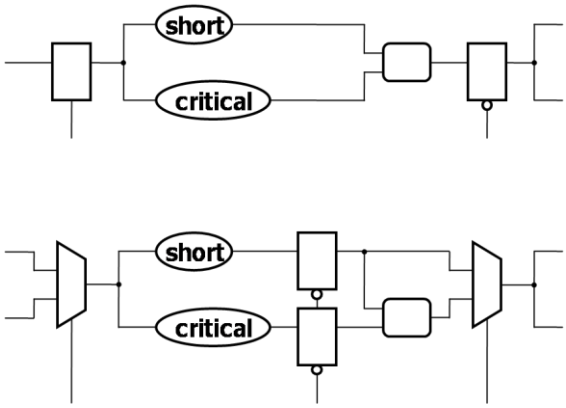


図３．耐タイミング故障クロッキング

第二に、前年度提案したタイミング故障耐性を持つプロセッサアーキテクチャに関する提案を詳細化し、検証した。本アーキテクチャでは、適切なパイプラインステージ間に、タイミング故障検出機構を入れた FF を配する。次に、各段におけるタイミング故障の情報を伝播させるネットワークを組み込む（図４）。さらに、レジスタファイルとプログラムカウンタはエラーから確実に保護されるようにし、エラーがある場合にはマシンステートが更新されないようにする（図 ５）。最後に、検出されたタイミング故障からプロセッサ動作を正常な状態に回復するためにリセットをかける。

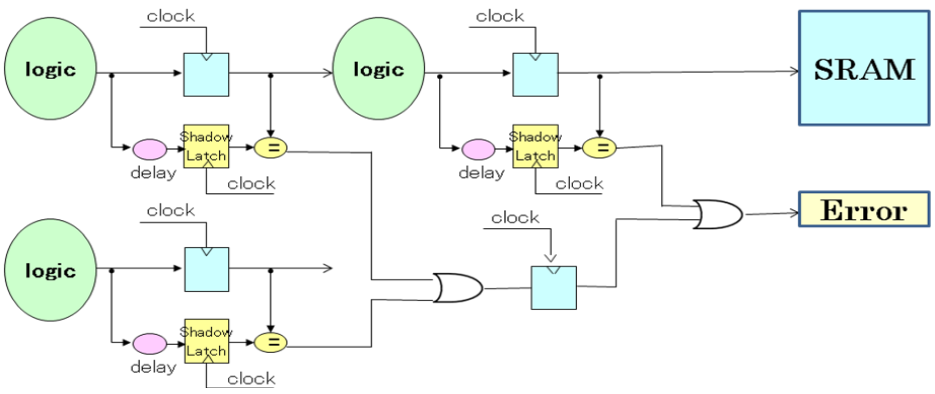


図４． タイミング故障の検出と通知ネットワーク

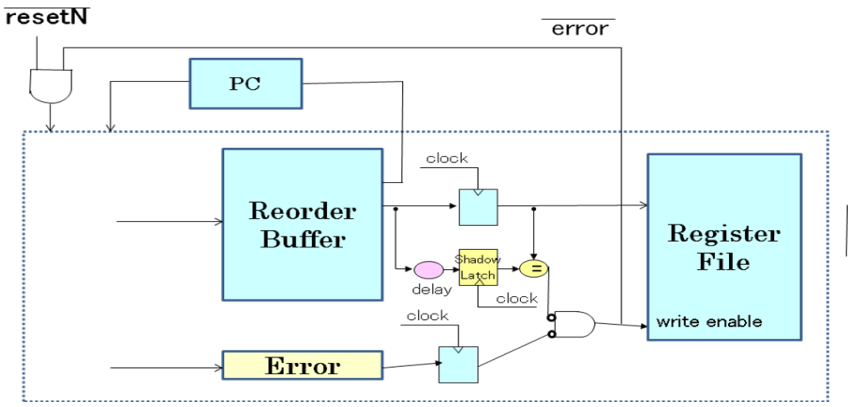


図５． レジスタファイルとプログラムカウンタの保護

この機構を、前年度に引き続き、テストベッド（図 6 上）上で FPGA 回路を改良することで、所期の動作が正確にできることを検証した。具体的には、2-way のスーパスカラプロセッサを設計し、提案手法を組み込んだ。最終的には対象となるプロセッサの設計情報に自動的に提案手法を組み込む方法を実現する予定であるが､現時点では手動で追加した。電源電圧、動作周波数を変更すると、動作不能範囲では、タイミング故障の検出とそれに伴うリセットを繰り返す。この状態から電源電圧、動作周波数を動作可能範囲に戻すと、再び実行を継続することが確認された。

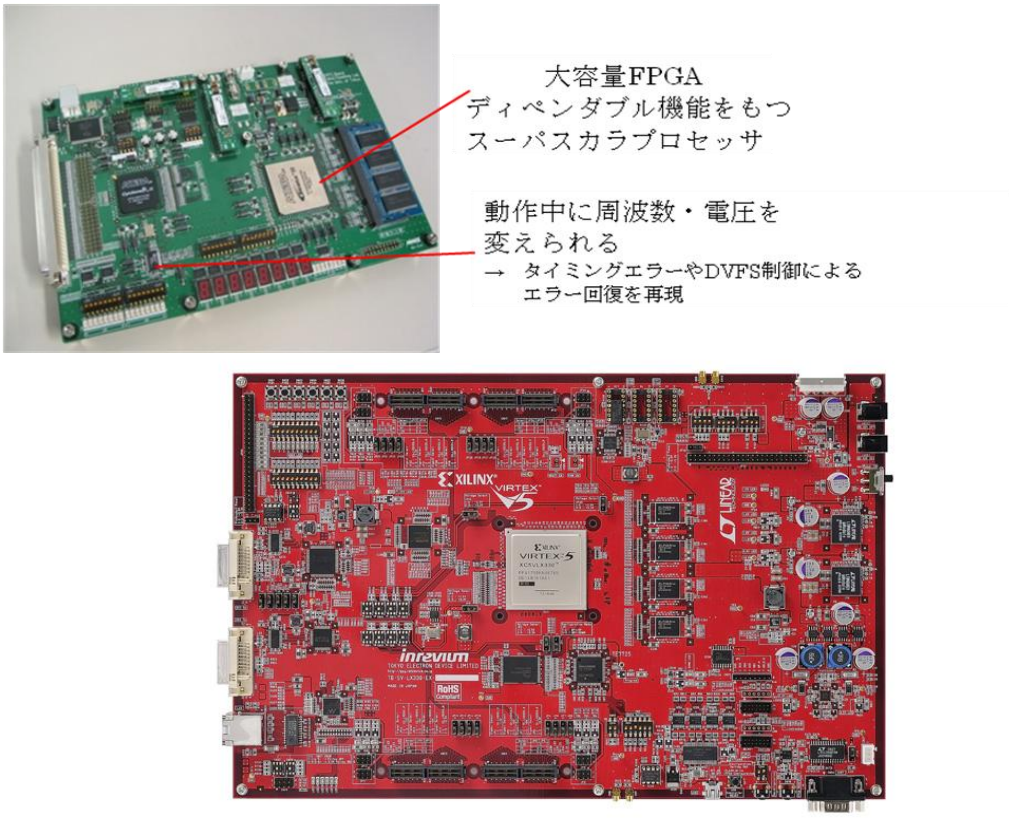


図６．テストベッド（上：従来品、下：新規開発品のベースとなる評価ボード）

今回の実装では、スーパスカラプロセッサのすべての FF に検出回路を実装したため、あらゆる個所に発生したタイミング故障を検出することができるものの、回路規模の増大は避けられない。今後は、クリティカルパスにのみ検出回路を挿入するとともに、前述した耐タイミング故障クロッキング方式との併用について検討する予定である。

また今年度は、より大規模なテストベッドの開発を行った。図 6 上のテストベッドは、予算の都合上十分な容量の FPGA を搭載することができなかった。新たに開発したテストベッドは、図 6 下に示す評価ボードをベースにしたもので、Xilinx 社 Virtex-5 の最大容量の FPGA を搭載しており、より大規模なプロセッサに対する検証が可能となる。来年度以降は、主にこのテストベッドを用いて研究を進める予定である。

第三に、永久故障耐性を持つ FPGA アーキテクチャに関しての提案を行った。提案の基本的なアイディアは、故障に対応して FPGA の動的再構成を行う再構成マネージャを、ハードワイアードな固定機能として FPGA に埋め込むのではなく、ソフトコアのプロセッサとして同じ FPGA 上に構成することである（図 7）。このことによって、通常の FPGA に対する追加回路を最小限に抑えることができる。永久故障耐性を必要とする市場は、宇宙などごく限られたものである。このような限られた市場のために専用に LSI 開発することは、最早現実的ではない。通常の FPGA に対する追加回路を最小限にすることによって、オプション機能として耐永久故障性を持つ FPGA として普通に市販することができる。

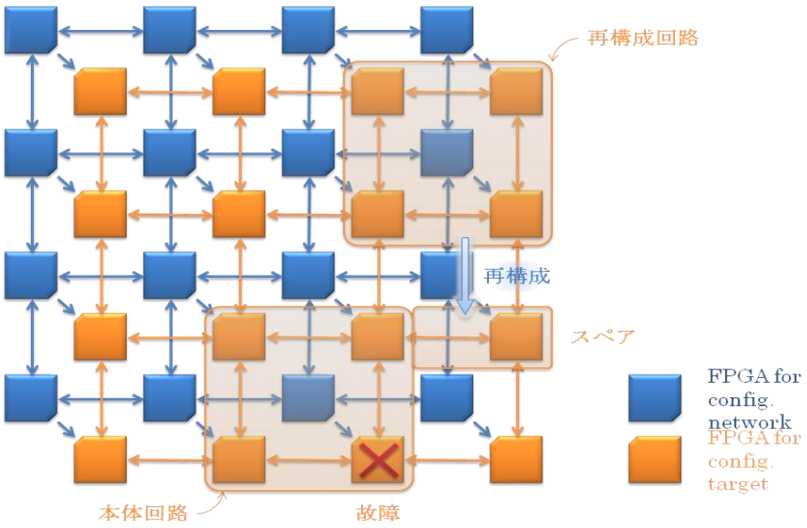


図 7．耐永久故障アーキテクチャ

提案を検証するためのテストベッド（図 8）の FPGA 回路設計を行った。具体的には、図 7 において FPGA for config network と示した固定回路部分にあたる。これは、故障を検出し、その発生を再構成マネージャに伝え、再構成マネージャからの指示により対象部分回路を再構成するリング状のネットワークである。

本年度は、この FPGA の設計がほぼ完了し、この FPGA からユーザロジック FPGA が再構成可能であることを確認した。図 9 に、そのネットワークのインタフェース部のブロック図を示す。平成 21 年度は本ボードを用いた詳細提案と評価が課題となる。

図８．　耐永久故障用テストベッドボード

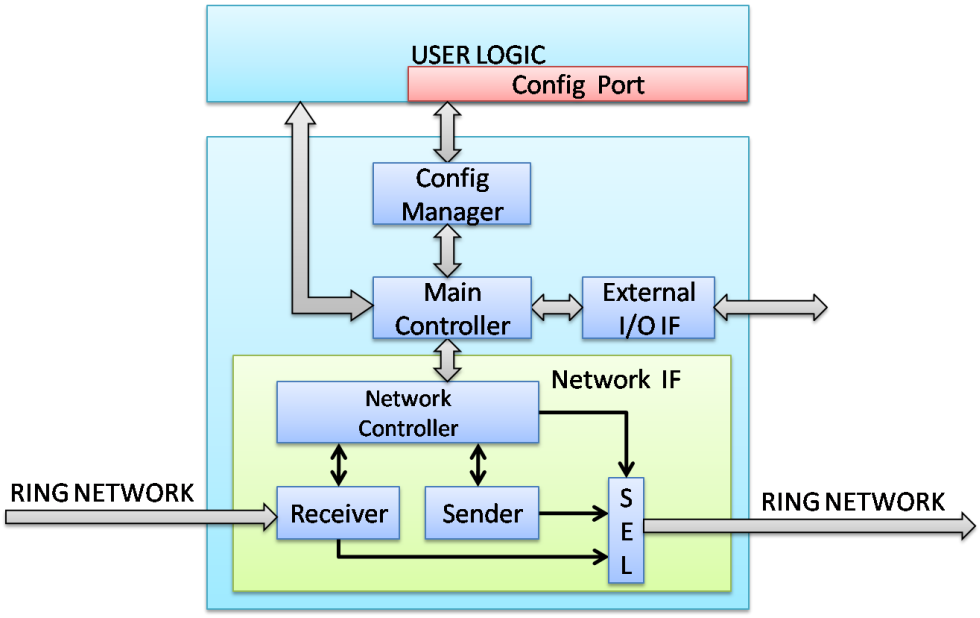


図９．　耐永久故障 FPGA アーキテクチャのネットワーク I/F のブロック図

2.2　ディペンダビリティ支援ルータグループ

ディペンダビリティ支援ルータグループでは、ディペンダブルVLSIアーキテクチャのうちで、通信系、特にルータの高機能化を核として、VLSIの信頼性を飛躍的に向上させる技術の研究開発を行っている。

平成20年度は、第一に、ディペンダビリティを支援する高機能ルータアーキテクチャの基本方式の開発を行った。具体的には、多数のコアの利用と高機能ルータによってメニーコアプロセッサのディペンダビリティ向上および速度向上を目指す仕組みとして、SmartCoreシステム(Smart many-core system with redundant cores and multifunction routers)を提案した。

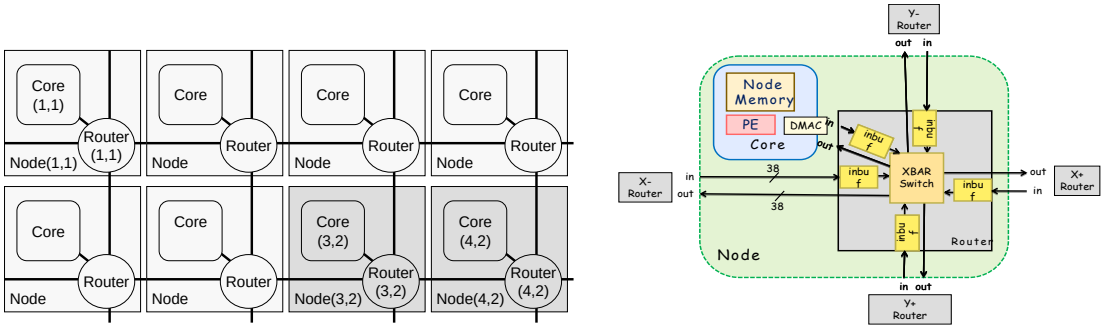


図10．Smart Coreシステムによるディペンダビリティ向上（左）。右はノードの構成

Smart Coreシステムにおけるディペンダビリティ支援アーキテクチャの概要について述べる（図10）．図では、コアのことをCore、ルータのことをRouter、CoreとRouterを含む領域をNodeと記述している。この図では、Node(3,2)のディペンダビリティ向上のため、同等の機能を持つNode(4,2)を冗長実行のために割り当てる場合の様子である。また、Core(4,2)をソースとする通信が特定の高機能ルータ、ここではRouter(3,2)、を経由するように経路を固定する。Core(3,2)が受け取るべきパケットは、高機能ルータRouter(3,2) が複製し、パケットの系列を同じ順番でCore(3,2)およびCore(4,2)の2つのコアに送信する。2つのコアが送信したパケットは、経路が固定されているため、必ずRouter(3,2)を経由する。このルータにおいて、それらのパケットの内容が一致することを検査してエラーが発生していないことを確かめた後に、1つのパケットとして適切な宛先へと送信する。2つのコアが送信したパケットが一致していない場合にはエラーが検出され、適切な処置が施される。このように、本方式では、高機能ルータを中心にしてコアの単位で冗長実行をおこない、かつ送受信パケットのレベルでエラーを検出する。このため、コアの実装の詳細に依存することなく、様々なハードウェア構成に対してディペンダビリティを向上させることができるという利点を持つ。

本研究における高機能ルータの詳細な評価はクロックレベルのソフトウェアシミュレータを用いる評価と、FPGAシステムを用いる大規模な評価の２つの方向性を適切に利用して進めている。

第二に、ソフトウェアシミュレータに関して、高機能ルータアーキテクチャを評価するためのシンプルなメニーコアプロセッサアーキテクチャを構築し、このアーキテクチャの動作をクロックレベルで模倣するマルチコアプロセッサシミュレータSimMcの基本モジュールの実装を行った。マルチコアプロセッサシミュレータSimMcを用いて、ベースとなるマルチコアプロセッサの検討をおこなった。姫野ベンチマーク(himeno benchmark)、equation solver kernel、行列積(matrix multiply)を並列化し、コア数を増加させて測定した速度向上の結果を図11にまとめる。図11右は、このシミュレーションに要したシミュレーション時間

から計算したシミュレーション速度である。

図11の結果から、姫野ベンチマークおよびequation solver kernelのようにボトルネックの生じにくいベンチマークプログラムについてはコア数の増加に伴って性能が向上する、行列積については一定のところで性能が飽和する、という想定通りの結果が得られることを確認した。一方で、このようなソフトウェアシミュレータの評価では、64コア構成において1秒間に8K cycleしかシミュレーションできずに非常に低速である。シミュレータの最適化によりある程度の高速化を期待することができるが、高機能ルータアーキテクチャの詳細評価においては、FPGAシステムによる高速化が好ましいことを再確認した。

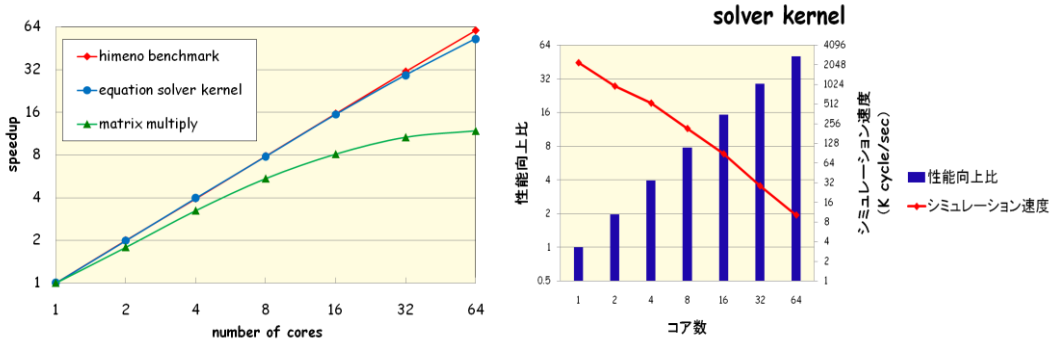


図11. コア数を増加させて測定した速度向上（左）とシミュレーション速度（右）

第三に、本研究における高機能ルータの詳細な評価のためのFPGAシステムの開発を進めた。多数のFPGAを搭載する従来の評価用システムでは、評価対象のアーキテクチャに応じてスケーラブルに構成を変更することが困難であった。この欠点を解決し、高機能ルータの評価を容易にするために、必要に応じてメッシュ接続されたFPGAボードの枚数を増加することができるスケーラブルな構成を可能とするボードの設計・開発をおこなった。

2.3 形式的検証グループ

- 上位設計記述の等価性検証および設計理解のための環境開発

形式的検証グループでは、形式的検証の適用による VLSI システム設計の信頼性向上を目指して研究を進めており、特に、上位設計において適用可能な形式的検証技術の研究開発を対象としている。その中でも、設計記述を詳細化・最適化していく際に、等価性を検証することは、設計誤りの検出に有効である。本年度は、設計例題による等価性検証ツールの評価・改善、上位設計記述の静的な実行時間見積もり手法に関する研究を行った。

等価性検証ツールの内部構成図を図 16 に示す。設計記述は、初めに拡張システム依存グラフ(ExSDG: Extended System Dependence Graph)に変換され、指定された等価性を設計記述が満たすかどうかを形式的に検証する。加えて、検証ツールは、設計解析・典型的な設計誤りの発見、デバッグ支援、などの検証・デバッグに有用な機能を追加することが可能な構成となっている。これは、本研究で用いる ExSDG に、設計解析・検証に必要な抽象構文木や依存関係の情報が全て含まれているためである。今年度は、主に、設計例題を用い

た等価性検証ツールの評価と改善を行った。以降で述べる例題の多くは実際に電機メーカ企業から提供されたものである。現在、検証ツールは C++言語で 9 万行程度のプログラムとして実装されている。

検証ツールにおける設計記述の内部表現である ExSDG を生成する部分については、45000 行の設計記述から 50 分で ExSDG を生成することができた。これは、十分に大規模な設計記述から ExSDG を生成することが可能であると言うことができる結果である。等価性検証ツールで設計例題を検証したところ、数百行程度の設計の等価性を数秒で検証することができた。また、実際の設計において多く含まれるループを扱うことができるように、記号シミュレーション時にループを指定回数だけ展開して検証を行う機能を追加した。これにより、検証できる設計記述が大きく広がった。実際に、企業から提供されたループを含む設計例題の検証も正しく行うことができるようになった。設計例題を用いた評価から、条件分岐数やループの繰り返し回数が多い検証例題では、現在の手法では短時間で検証を行うことができないことが分かった。今後は、部分的な等価性を証明し、それらを組み合わせることによって全体の等価性を証明する手法を導入する必要があると考えている。

また、上位設計記述の理解を深めるための手法として、静的な実行時間見積もり手法の研究を行った。この手法では、与えられた実行時間付きの設計記述に対して、最大実行時間を静的に見積もる手法である。設計を進める際に、モジュールやブロックごとに実行時間を挿入し性能をシミュレーションによって測定することは多く行われている。この場合、得られる見積もり結果は、実際の最大実行時間より短いものである。一方、我々が提案している静的な手法では、実際の最大実行時間より必ず長い見積もり結果が得られる。そのため、シミュレーションによって見積もった最大実行時間と提案手法によって見積もったものの間に、実際の最大実行時間が存在するため、両者の範囲を小さくすることによって、より正確な見積もりが可能である。提案する見積もり手法は、前述の ExSDG 上で実行することができる。提案手法による実行時間の見積もりの様子を表したのが図 17 である。この例では、実行時間は waitfor 文によって与えられている。提案手法では、実行パスごとに waitfor で記述された実行時間を足し合わせていき、その中で最大のものを見積もり値とする。見積もり手法を実施中に、偽パス（プログラム中でどのような入力に対しても実行されないパス）であると判断された場合には、そのパスは見積もり対象から外される。図の例では、p1→p3 と p2→p4 のパスが偽パスにあたる。提案手法では、この偽パスの検出を自動的に行う。一般的に、設計記述中の実行パス数は膨大になるため、全ての実行パスの実行時間の最大を取る手法は現実的ではない。そこで、提案手法では、実行パス数がある一定数以上になった時点で、見積もりを行う範囲を分割することにより、実行パス数の増大を防ぐ工夫を導入している。

提案した静的な実行時間見積もり手法の評価として、SpecC 言語で記述されたエレベータ制御システムを例題として見積もりを行った。設計記述の規模は約 2600 行である。このエレベータ制御システムのシミュレーションによって観測された最大実行時間は 148 であった。提案する静的見積もり手法で得られた最大実行時間は 152 であった。そのため、実際の最大実行時間は 152 と 148 の間にあることが分かった。見積もり領域の分割を行わない場合には、実行パス数が大きくなりすぎたため 30000 秒の間に見積もり結果を得ること

ができなかった。このことからも、実行パス数が一定数に達した場合に、見積もり領域を分割する提案手法は非常に有効であることが分かった。今後は、より大規模な設計例題を用いた評価を行う予定である。

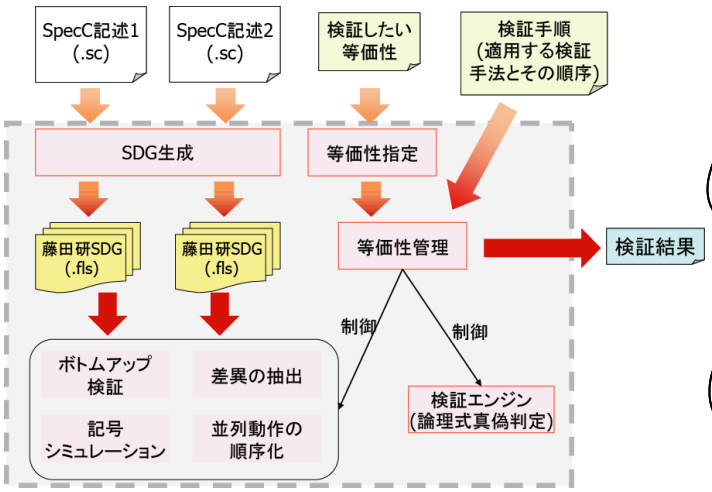


図 16. 等価性検証ツールの構成

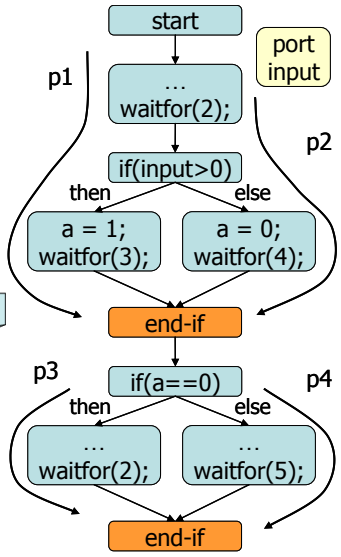


図 17. 実行時間見積り手法

- 算術回路検証

従来、形式的論理検証を行なう基本的な解析手法として、論理式や論理回路を対象として、2 値変数からなる論理関数を効率的に計算機で処理する手法として、２分決定グラフを論理関数の充足性判定手法（SAT 手法）が研究されてきた。最近の進歩により、特に SAT 手法を利用すると数百万変数からなる論理式を数時間程度の処理時間で解析できるようになってきている。しかし、これらの論理関数処理手法は、掛算に代表される算術演算回路には不向きであることも分かっている。例えば、32 ビット同士の掛算を行なう掛算器の検証を BDD や SAT 手法で行なうことは難しい。一方、最近の半導体技術の進歩により、信号処理や画像処理の分野では、大規模な算術式のハードウェア実装も行なわれるようになっている。このため、複雑・大規模な算術式の解析を効率的に行うことで、算術演算回路を形式的に検証できる技術の要求が高まっている。また、算術演算回路設計を上位から支援するためには、従来の 2 値変数による論理関数を利用した表現ではなく、計算に利用するビット幅に対応する変数（一般に 2 値変数は論理変数、ビット幅を集めた変数はワード変数と呼ばれる）で算術式を表現し、解析することが重要である。整数変数の場合、ワード変数としては 1 変数であるが、それを論理変数に展開すると 32 変数となり、算術式を論理変数に展開すると、解析しなければならない変数の数が膨大となってしまい、効率的な検証は不可能になる。従来から、整数変数に基づく決定グラフが提案されてきたが、計算機上での効率的な処理手法がなく、また、整数変数を単に整数と扱っていたため、オーバーフローなど有限のビット幅を考慮した解析ができなかった。そこで、整数変数からなる算術多項式を 2 分決定グラフの形で表現する Horner Decision Diagram（HED）を新規に考

案した。HED の例を図 18 に示す。

合わせて、HED 表現向けの有限ビット幅を考慮した多項式の Modulo 等価性判定手法を開発した。今、a^2-b^2 と a^2+7b^2 は a, b が通常の整数変数である場合には、明らかに不等価であるが、ビット幅を 3 ビットとすると、$8b^2$ は 3 ビット幅では必ずオーバーフローするので、$a^2+b^2-8b^2=a^2-b^2$ となり、等価となる。このような Modulo 等価性を効率よく判定し、任意の多項式間の等価性を検証する手法を Modular-HED として実装した。信号処理や画像処理におけるベンチマークで評価したところ、従来の BDD や SAT 手法では、数時間かかっても処理できなかったものを数ミリ秒程度で完全に検証できることを示している。

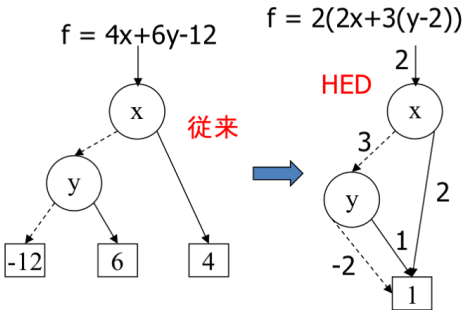


図 18　従来の多項式のための決定グラフと提案する決定グラフ（HED）

また、論理回路レベルの算術演算回路を効率的かつ形式的に検証する手法を新規に開発した[3]。VLSI などハードウェアで複雑な算術演算回路を実現する際には、掛算器と加算器が基本的な回路構成要素となる。掛算器は一般に、まず部分的を計算し、それらを何らかの手順で足し合わせることで計算している。ここで、部分積は 2 進数の場合論理積で計算できるので、回路の大部分は基本的に加算を行っていることになる。そこで、半加算器（half adder）を基本とした各種掛算器の表現法を新規に考案した。従来から、図 19 に示すように、与えられた掛算回路から、半加算器を自動的に抽出し、抽出された半加算器から構成されるネットワークが、掛算の標準的な回路（仕様）と同一であるか否かを判定することで、効率的に掛算器を検証する手法が提案されているが、特定の掛算器しか検証できないという問題点があった。これは、部分積を加算していく際に、特にキャリー伝搬部分の回路が、個々の加算器によって大きく異なり、抽出した半加算器ネットワークと仕様との間で対応がつかなくなることも多いためである。そこで、部分積を加算していく際にキャリー計算回路を別に表現することで、多様な加算器に対応手法を新規に考案した。図 20 に示すように、部分積同士の加算を行う際に、キャリー信号の入出力も同時に定義した標準形を利用する。そして、キャリー信号回路の解析は、従来の SAT 手法などを適用して別途行うようにしている。特定の部分積を計算するキャリー回路自体は小さいため、従来手法でも効率よく検証できる。このようにすることで、キャリー回路部分の違いを吸収できるため、多様な掛算器を効率よく検証できるようになっている。実験の結果、Booth 掛算器など各種掛算器に対し、64 ビット同士の掛算を行うような大規模回路でも、数分で検証できている。また、誤設計がある場合には、仕様との対応がどのように取れないかを示すことも可能である。

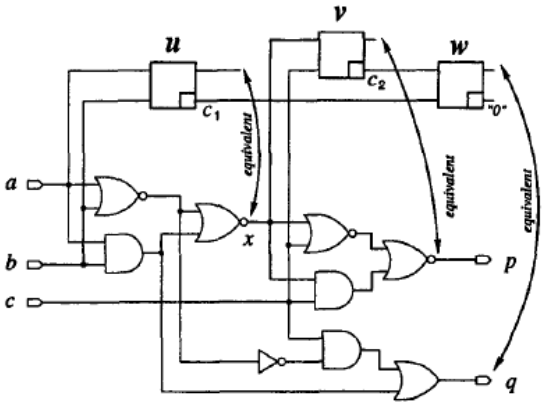


図 19　論理回路から半加算器を抽出することで掛算器の検証を行う

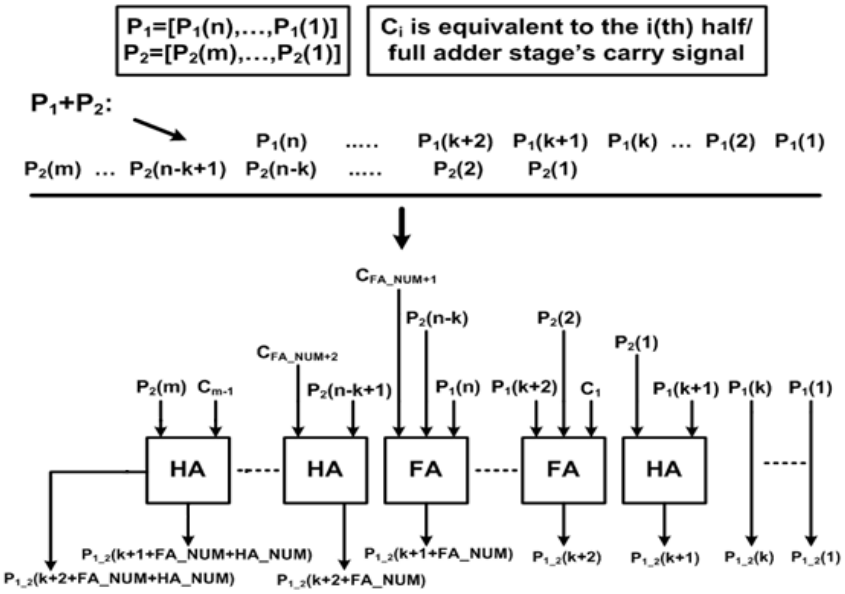


図 20　部分積を加算していく際にキャリー計算回路を別に表現することで、多様な加算器に対応

- HW/SW 協調動作による検証高速化手法

現在の効率的なモデル検査手法の一つに、コンパイルドシミュレーションを基づく準形式的限定モデル検査がある。この手法は明示的に全ての入力パターンに対してプロパティを満たすかどうかを、シミュレーションによって検査するという手法を取っている。単純に全入力パターンをシミュレーションすることは現実的に難しいが、この手法ではシミュレーションと同時に回路を解析することで、シミュレーションの必要が無い冗長な入力パターンを動的に割り出す。それを次回以降の実行から除外することでシミュレーション回数を大幅に減らす。

図21にこのモデル検査のフローを示す。最初にコンパイルドシミュレーション用のプログラムを作成する。これには検証対象回路、プロパティ、検証対象回路に対応する解析用回路が含まれる。この回路の出力によって、検証対象回路が入力に対してプロパティを満たすかどうかを判別することができる。次にシミュレーションに対する入力を選択する。既にシミュレーションを行ったもしくはシミュレーションをする必要が無いと判断された入力パターンの集合を論理関数として保持するため、この関数に含まれない入力パターンが選択される。次に選択された入力パターンに対してシミュレーションを行う。この手法では、シミュレーションを行うとその結果を計算すると同時に、それと全く同じ結果を導く入力パターンの部分集合を求めることができる。結果が同じになるとわかっている、すなわちシミュレーションを行う必要がない入力パターンの集合をスキップキューブと呼び、これらは次回以降の入力パターンの選択候補から除外される。

我々は図19のように、回路シミュレーションおよびスキップキューブ計算をハードウェア化しハードウェア・ソフトウェア協調実行することによって準形式的検証を高速化する手法（図22）を提案した。また提案手法の有効性を示すため、計算機とFPGAによる実験を行った。実験ではシミュレーション部分とスキップキューブの計算部分をFPGAに実装し、計算機上のソフトウェアと実際に協調実行した。どの例題に対してもハードウェア・ソフトウェア協調実行による高速化は有効であり、平均で6.7倍の高速化が見られた。これにより本研究の提案手法による準形式的検証の高速化の効果が実験によって実証された。

上記のハードウェア・ソフトウェア協調実行による検証手法は検証アルゴリズムを実行する時間を削減することが可能であるが、対象回路をハードウェア化するために必要な時間は含まれていない。提案手法では検証対象回路を解析することでスキップキューブ計算回路を生成し、これらを FPGA に搭載する。一般的に設計記述から FPGA 向けの構成データをコンパイルするために必要な時間は大規模な回路であれば数時間から数日であり、全体の検証時間の大部分を占めることになってしまう。このコンパイル時間を削減するため、あらかじめ FPGA 内部にスキップキューブ計算回路を組み込んだ検証指向 FPGA である VDD-FPGA を提案した。この VDD-FPGA は汎用 FPGA と同じアーキテクチャであるため、CAD ツールや構成データは同じものが使用可能である。汎用 FPGA と同様に再構成を行い回路を動作させることで、入出力信号とともにスキップキューブも出力される。ROHM 0.18um 5-Metal CMOS プロセスを用いて VDD-FPGA を設計し、VDEC を通じて試作を行った。試作チップの面積は $2.5\times2.5\mathrm{mm}^2$ である。図 23 に VDD-FPGA のチップ写真を、図 24 に VDD-FPGA の構成単位であるタイルのマスクレイアウトを示す。

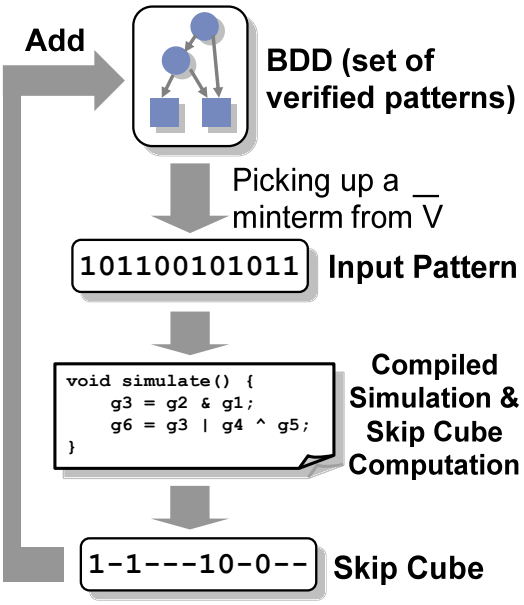


図 21　準形式的検証手法の流れ

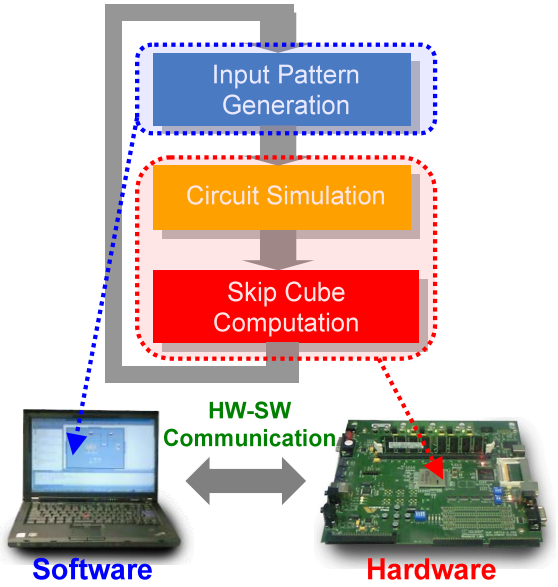


図 22　ハードウェア・ソフトウェア分割

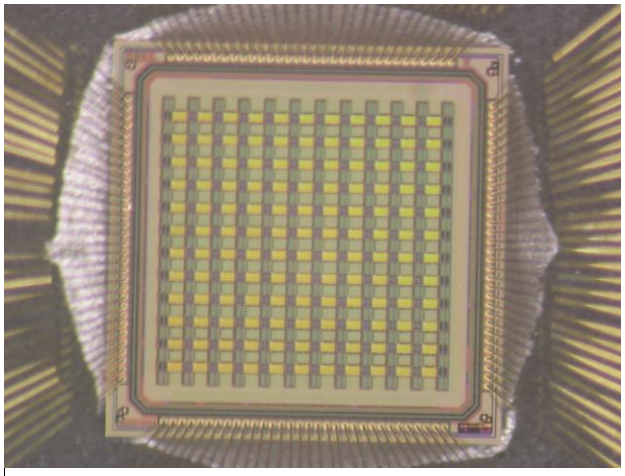


図 23 VDD-FPGA チップ写真

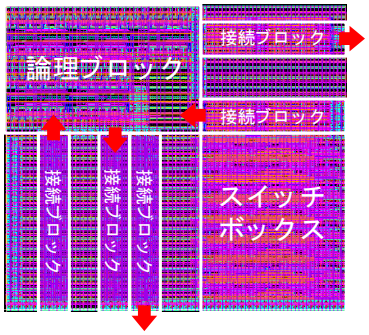


図 24 タイルのマスクレイアウ

● 高位設計記述向けルールベース等価性検証手法

現在よく用いられているシステムレベル設計手法では、設計記述に対して対話的な設計の変換や詳細化を支援するツールを提供している。この手法では、設計記述を主に人手で局所的かつ段階的に変更していくことで最終的な設計記述を作成する。よって、この各段階において変換前後の等価性を確認しながら設計を進めていくことによって設計誤りを防ぐことが可能となる。このような用途を目的とした高位記述の検証手法として、ルールベース等価性検証を提案した[3]。この手法では静的な依存関係やプログラムフローに基づいて定義された等価性規則をボトムアップに適用することで2 設計間の等価性を示す。等価性規則を部分グラフのパターンとして見なすことで、ルールベース等価性検証は2抽象構文木間のマッチング問題とみなすことができるため、ルールベース等価性検証手法は効率的

に検証を行うことが可能である。またこの手法では、設計記述における依存関係や制御フローを抽象構文木に追加した拡張システム依存グラフExSDGを用いることで、効率的な実装を実現している。

ルールベース等価性検証ではボトムアップに等価性を証明していくため、変数といった設計の基本構成要素の等価性をあらかじめ求めておく必要がある。どのようにしてこの初期等価点を発見するかが課題となる。[3]では、ボトムアップ証明の際の初期等価点をどう求めるかを明らかにしていない。また[3]における実装では名前に基づいて初期等価点を求めているが、変数名が変更された場合や変数の対応が発見できない場合などには検証は失敗してしまう。図25(a)-(d) はどれも機能的に等価であるが、名前に基づいた初期等価点ではこれらの等価性を証明することはできない．上記のような設計変更は頻繁に行われるものであり、これらの変更に対して検証ができないことは実用的に問題がある。本研究では、内部等価点をランダムシミュレーションによって推定することで上記のような場合においても検証を可能にする手法を提案した[7]。

例題として逆離散コサイン変換(IDCT) の最適化前後のSpecC 設計記述を用いた。最適化前の例題では各行・各列の処理を逐次的に実行しているが、最適化によりそれらを並列に実行するように記述が変更されており、また変数名も変化している。従来手法では変数の対応が取れないため検証不能となり、等価性の判定ができなかった。提案手法ではランダムシミュレーションにより内部等価点の推定を行ったため変数の対応を求めることが可能となり、2設計が等価との判定を行った。本実験では、ランダムシミュレーションを数サイクル行うことで変数の対応を求めることが可能であった。実行時間は数秒程度であり、また実行時間の大部分はシミュレータのコンパイル時間であった。シミュレータのコンパイル時間は一般的に設計規模に比例すること、またルールベース等価性検証問題もグラフマッチング問題と見なせることから、本手法は大規模設計に対しても適用可能であると思われる。

```
int ex1(int a, int b) {
    return a - b;
}
```

図 25(a) 元の設計

```
int ex2(int b, int a) {
    return b - a;
}
```

図 25(b) 変数名が変更された場合

```
int ex3(int c, int d) {
    return c - d;
}
```

図 25(c) 変数の対応が発見されない場合

```
int ex4(int a, int b) {
    int c = a - b;
    return c;
}
```

図 25(d) 中間変数など対応変数がない場合

- デバッグ支援

半導体集積技術の向上に伴い、VLSI システムの規模および複雑さが増大する傾向にあり、検証が困難になってきている。システムレベル設計方法論では、SystemC や SpecC などの C ライクな高位記述言語を用いるようになってきているが、RTL と比較すると、言語の扱いの複雑さのため、このような言語をサポートするツールはまだあまり整備されていない。一方で、検証の方法としては、従来のシミュレーションベースの方法と、記号的な方法があるが、前者は探索の深さは伸ばせるもののコーナーケースを活性化しにくい問題、後者は探索の幅が網羅的にとれるものの、記号式の爆発のために探索の深さに限界がある。言い換えると、両者はそれぞれ状態空間に対する深さ優先探索および幅優先探索を行っていると解釈できる。

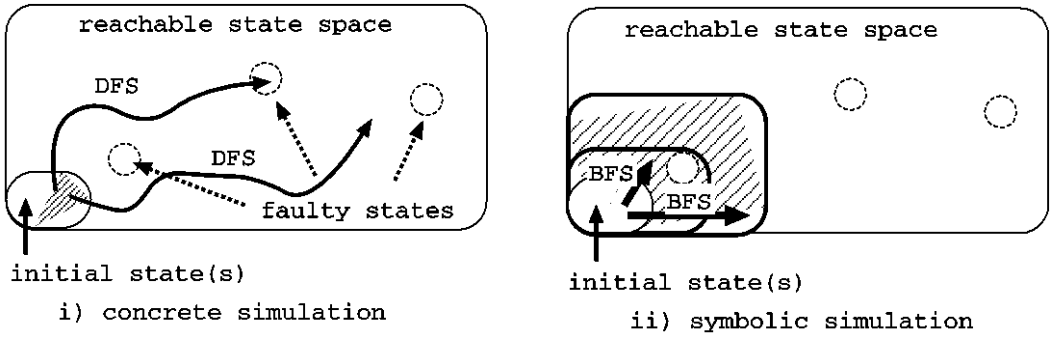
本研究では、C ライクな言語による設計記述を対象として、従来の具体値を用いたシミュレーションと記号シミュレーションとを組み合わせ、これらの問題点を克服するユーザ駆動幅優先探索アプローチ(図 26)を提案し、また、そのコンセプトの実証ための検証・デバッグ環境を提案した[5]。

提案するユーザ駆動幅優先探索では、まず具体値を用いたランダムシミュレーションにより、到達可能な状態を収集する。次に、ユーザは到達可能な状態から、興味深いと思われる状態を選び、そこを始点として記号シミュレーションにより幅優先探索を行う(もしくは、到達可能かどうかわからない状態を作って使用してもよい)。このアプローチにより、初期状態から遠い状態であっても、指定した始点からの網羅的な状態探索が可能となるが、この始点の状態をどうやって選ぶかが状態探索の可否の肝となり、現状、ユーザの設計に対する知識に依存している。

提案する検証・デバッグ環境のツールは、SpecC により記述された設計と、入力変数の宣言(FL_INPUT)および制約の定義(FL_ASSUME)、出力変数が満たすべきアサーションの条件式(FL_ASSERT)が与えられると、制約を満たしつつアサーション違反を起こすような入力値を(もし存在し、発見することができれば)出力する(図 27)。図 28 の例では、入力変数が x=y>=0 の条件を満たすときにアサーション違反が発生し、ツールは(x=0,y=0)などの入力パターンを生成する。構成は、図 29 のようになっており、設計記述から生成された抽象構文木(AST)を解釈するインタプリタ、具体シミュレーションを実現するための具体値評価エンジン、記号シミュレーションを実現するための記号値評価エンジン、記号シミュレーションによって生成された記号式を保管するマネージャ、および記号式の充足可能性(もしくは妥当性)を判定する SMT ソルバからなる。

提案手法を実証するためのケーススタディとして、抽象化されたエレベータコントローラの設計記述を用いた。3 階建て、1 基のエレベータを制御するものであり、9 個の入力変数および 7 個の出力変数があり、SpecC 記述で 400 行程度である。アサーションとしては、1) エレベータは 1 階から 3 階までのどこかに存在する、2) エレベータのドアはどこかのフロアに停止中でなければ開かない、といったものを定義した。まず、一つ目の従来手法として、具体値によるランダムシミュレーションを行った。初期状態から 10 サイクル(リセット用 2 サイクルを含む)までに限定し、100 万回のランダムシミュレーションを行ったが、アサーション違反は発見できなかった。二つ目の従来手法として、記号シミュレーシ

ョンを行ったが、図 30 のように、3,4 サイクル程度で記号式のサイズが爆発してしまい、リセットを含んでも 5,6 サイクル程度のシミュレーションしか実行できず、この範囲内ではアサーション違反を発見できなかった。そこで、ランダムシミュレーションの結果から、5,6 サイクル目によくあらわれる状態のひとつを選び、そこを始点として記号シミュレーションを実行したところ、さらに 4 サイクル後(初期状態から 9 サイクル目)でアサーション 1)が違反されることを発見した。これにより、従来のシミュレーションや記号シミュレーション単独では発見できなかった不具合をユーザ駆動幅優先探索手法により発見できたことを示した。



i) 深さ優先探索(DFS)アプローチ　　ii) 幅優先探索(BFS)アプローチ

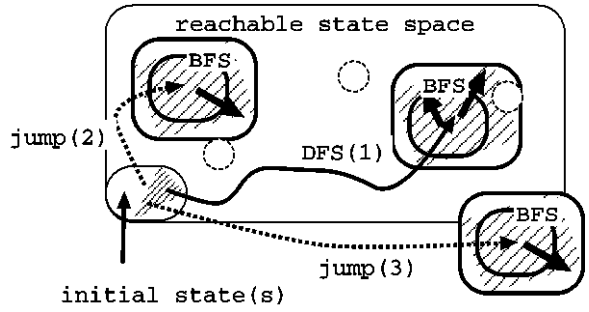


図 26 ユーザ駆動幅優先アプローチ

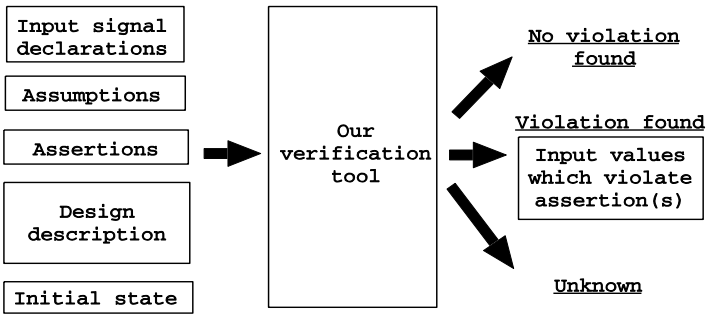


図 27 ツールの入出力

```
1: int func(int x, int y) {
2: int r = 0;
3: if (x - y > 0) // B1-T
4: r = x - y;
5: else // B1-F
6: r = y - x;
7: return r;
8: }
```

```
1: int x, y, z;
2: FL_INPUT(x);
3: FL_INPUT(y);
4: FL_ASSUME(x >= 0);
5: FL_ASSUME(y >= 0);
6: z = func(x, y);
7: FL_ASSERT(z > 0);
```

図 28 サンプルコードとドライバ

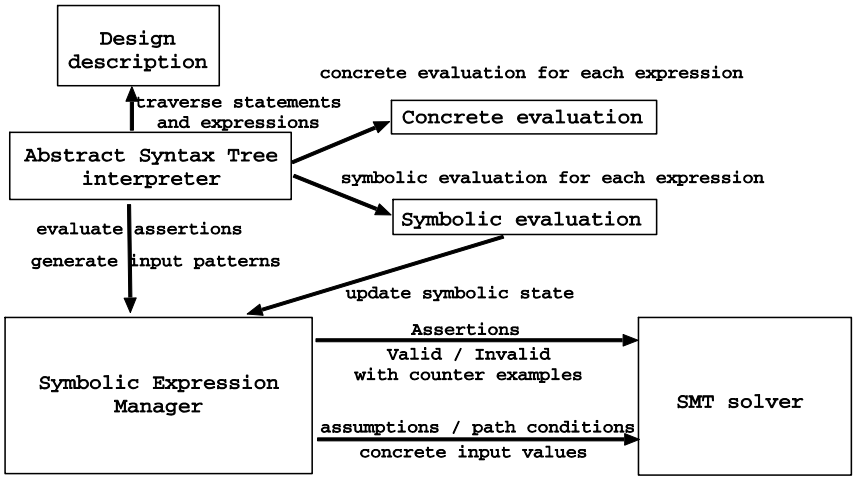


図 29 ツールの構成

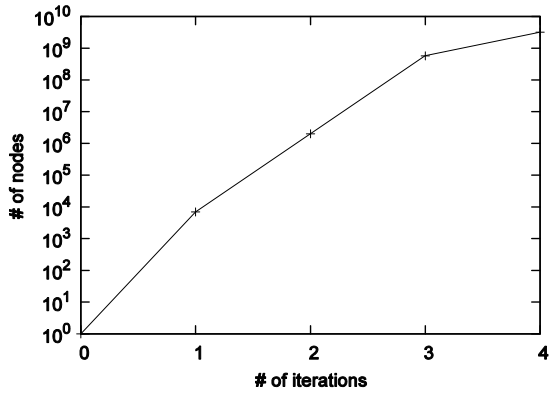


図 30 記号シミュレーションの結果

3. 研究実施体制

(1)ディペンダブルアーキテクチャグループ

①研究分担グループ長:坂井　修一(東京大学大学院、教授)

②研究項目

・タイミング故障耐性を持つクロッキング方式

・タイミング故障耐性を持つプロセッサアーキテクチャ

・永久故障耐性を持つ FPGA アーキテクチャ

(2)ディペンダビリティ支援ルータグループ

①研究分担グループ長：吉瀬 謙二（東京工業大学大学院、講師）

②研究項目

・高機能ルータアーキテクチャの研究開発および評価

・形式的検証を用いた高機能ルータアーキテクチャ技術の有用性の検討

・中規模高機能ルータのチップ試作による実現可能性の検討

(3)形式的検証グループ

(1)研究分担グループ長：藤田昌宏

（東京大学、教授）

②研究項目

・上位設計記述に対する形式的等価性検証ツール開発

・演算系回路に対する効率的な等価性検証手法

・大規模設計記述の等価性検証のためのボトムアップな検証手法

・C ベース言語による上位設計記述の理解・マニュアル作成のための解析技術

・プログラマブル素子の挿入による In-field 回路デバッグ技術

4. 研究成果の発表等

(1)論文発表（原著論文）

1. Kenichiro Hirose, Yasuo Manzawa, Masahiro Goshima and Shuichi Sakai, “Delay-Compensation Flip-Flop with In-situ Error Monitoring for Low-Power and Timing-Error-Tolerant Circuit Design”, IPAP (Institute of Pure and Applied Physics) Japanese Journal of Applied Physics), 10.1143/JJAP.47.2779, Apr. 2008.
2. Sarbishei, B. Alizadeh, and M. Fujita, “Arithmetic Circuits Verification without Looking for Internal Equivalences”, 6th ACM/IEEE International Conference on Formal Methods and Models for Co-Design, pp.7-16, Anaheim, USA, June 2008.
3. Subash Shankar and Masahiro Fujita, “Rule-Based Approaches for Equivalence Checking of Spec C Programs”, 6th ACM/IEEE International Conference on Formal Methods and Models for Co-Design, pp.39-48, Anaheim, USA, June 2008.
4. Shuichi Sakai, Masahiro Goshima, Hidetsugu Irie, “Ultra Dependable Processor”, IEICE Trans. on Electronics, Vol. E91-C, No. 9, pp. 1386—1393 (2008).
5. Yoshihisa Kojima, Tasuku Nishihara, Takeshi Matsumoto, and Masahiro Fujita, “An Interactive Verification and Debugging Environment by Concrete/Symbolic Simulations for System-level Designs”, The 17th Asian Test Symposium, pp.315-320, Sapporo, Japan, November 2008.
6. Goeschwin Fey, Satoshi Komatsu, Yasuo Furukawa and Masahiro Fujita, “Targeting

Leakage Constraints during ATPG", The 17th Asian Test Symposium, pp.225-230, Sapporo, Japan, November 2008.

7. Hiroaki Yoshida, Masahiro Fujita, "Improving the Accuracy of Rule-based Equivalence Checking of System-level Design Descriptions by Identifying Potential Internal Equivalences", IEEE International Symposium on Quality Electronic Design, San Jose, USA, March 2009.

(2)特許出願

平成 20 年度 国内特許出願件数：1 件（CREST 研究期間累積件数：2 件）