平成 21 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

アーキテクチャと形式的検証の協調による超ディペンダブル VLSI

§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。

3年度目である平成21年度は、形式的検証・テストの研究とツール群の開発・基本評価、回路要素技術・プロセッサアーキテクチャ技術の開発・評価、メニーコア用ディペンダブル高機能ルータの基本設計などを行い、さらにグループ間連携として、ディペンダブルアーキテクチャの形式的検証を始めた。

【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
  前々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。前年度は、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。本年度は、この方式の回路の具体化・詳細化を行った。今後は詳細な評価を行う。

- 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。本年度は、より詳細な評価を行うべく、より現実的なスーパスカラプロセッサの設計を開始した。今後は、完成したスーパスカラプロセッサ上に実装した回路により、実効性の高い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
  前年度に引き続き、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。前年度は、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。本年度は、前年度に引き続き、詳細設計を行った。永久故障耐性は Triple Modular Redundancy (TMR) により実現される。FPGA 上に実装されたプロセッサが自身の TMR 状態を回復するためには、TMR を構成する故障したモジュールの FF の値を、スペア・モジュールに無時間的にコピーする必要がある。本年度は、このための方法を提案した。今後数カ月以内に実装を完了し、提案手法の具体的な動作の確認を行う。

【形式的検証】

- 研究開発している等価性検証ツール FLEC は既に企業などへ提供し評価を依頼している。その結果、具体的要望として、ループ処理部分の効率的な検証手法の追加(新規開発)、SystemC フロントエンドの追加が特に挙がっている。ループ処理に関しては、従来手法では扱えないようなイタレーション間に複雑な因果関係がある場合にも適用できる手法を考案・実装・評価し、従来比 10 倍以上の高速化を達成した。また、帰納法と融合した新規検証手法についても検討を進めており、机上では効率よく扱える範囲が飛躍的に向上することが確認されており、実装を進めている。また、SystemCフロントエンドについては、処理する範囲や処理フローを明確化するとともに、開発を進めている。第1バージョンは平成22年度上期中に完成し、評価を開始する予定である。
- 従来比10倍以上の高速化とともに、より広い範囲で効率よく等価性検証を行うため、従来のトップダウンな類似性解析に基づく等価性検証手法とは別に、ボトムアップな類似性解析に基づく等価性検証手法を考案し、実装・評価した。2 つの類似する設計記述間の等価性検証は、扱う記述上の差異によって、トップダウン解析が有効な場合とボトムアップ解析が有効な場合がある。今回、ボトムアップ解析機能を新規追加したことにより、等価性検証における(多様な設計記述に対応できるという意味での)ロバスト性が飛躍的に向上した。ボトムアップ解析が有効な設計記述の場合には、数千行規模の設計記述に対して、従来手法では数十分から数時間であったものが数十秒で処理できるなど、従来比 100 倍以上の高速化を達成している。

- 等価性検証ツールの評価を世界的に行うために、藤田研究グループと同様に等価性検証ツールの研究を開始している Oxford 大学と共同で、等価性検証のベンチマーク作成を行っている。専用のウェブページを作って、そこにベンチマークを集めるとともに、等価性検証技術やツールに関する技術紹介や情報交換を行う予定である。さらに、Oxford 大学 Ong 教授の Observation equivalence に関する研究成果、Kroening 教授の SystemC に対する高速シミュレーションと処理技術との融合を図る共同研究を来年度から開始することになっており、詳細について検討した。すでに両者で技術の評価を済ませ、研究スケジュールの調整を行っている。
- 高性能あるいは高効率（低消費電力）計算に要求される、応用に特化したカスタム算術演算回路最適化・形式的検証技術の開発を行っている。仕様がビット幅を指定した多項式として与えられ、それに特化した回路を自動生成できる。Modulo 等価性を考慮した多項式最適化（掛算数最小化）や等価性検証技術、従来全く利用できなかった大規模外部ドントケアを利用した算術回路自動生成技術などを新規に考案・実装・評価した。結果として、回路規模・遅延において従来比 30%から 40%の縮小を達成している。さらに、カスタム算術演算のパイプライン処理において、タイミングエラーが発生した場合に自動的にリカバリーする新規回路機構を考案した。これにより、パイプライン回路をオーバクロッキングで正しく動作させることが可能となる。FPGA ボード上に実装し、いくつかの回路で評価したところ、20%から 30%の高速化を観測している。さらに来年度、ドイツ・カイザースラウテルン大学 Kunz 教授の研究成果との融合を目指し、当方の研究成果をカイザースラウテルン大学のツールの環境で評価してもらっている。これまでに、当方の研究成果であるワード変数を利用した決定グラフ HED を利用することで、処理可能規模が大きく向上することが確認されている。来年度は、さらに共同研究へと発展させる予定である。
- 従来から研究開発しているハードウェア・ソフトウェア協調実行に基づくモデル検査手法の大規模設計への適用を図るため、従来のシミュレーションベース検証を行う際の検証カバレッジ向上手法への応用を検討している。これにより同数のシミュレーションパターン数で検証カバレッジを大幅に改良できる。FPGA ベースのエミュレータなどへの応用も期待でき、今後検討を進め、評価・改良していく予定である。
- 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・デバッグ環境の評価を進めている。等価性検証ツールと融合し、上の検証カバレッジ向上手法と併用することで、より広範囲の入力についての検証を可能とする環境構築を目指している。
- JAXAの元で研究開発が進められてきたシステムレベル設計支援ツールElegantツール内の等価性検証ツールVenusを正式にJAXAから移管された。今後、当グループのFLECツールとの融合を進め、Elegant環境でも利用可能としていく予定である。
- アーキテクチャアルゴリズムを形式的に検証する手法とそのツール化を進めている。現在までにRazorフリップフロップなどでタイミングエラーを検出し、エラーから自動的に回復するアーキテクチャアルゴリズムが提案され、シミュレーションなどでその有用性が示されている。しか

し例えばマイクロプロセッサ企業からみると、エラー回復のアーキテクチャアルゴリズムが複雑であるため、設計の正しさの保証が困難であり、実利用を躊躇している状況である。検証が不十分なため、どうしても保守的な設計に成らざるを得ず、結果的にマイクロプロセッサの性能を損ねてしまう可能性がある。そこで検証対象をマイクロプロセッサとし、各種アーキテクチャアルゴリズムを形式的に検証する手法とその検証環境（ツール化）について研究している。今年度は、単純パイプラインプロセッサ、スーパスカラプロセッサ、out-of-orderプロセッサに対し、タイミングエラー回復アーキテクチャアルゴリズムの形式的検証を行い、数秒から数時間（対象プロセッサの複雑度による）で検証できること、並びに、検証手法を拡張し、アーキテクチャアルゴリズムの最適化にも適用できることを確認した。

- 製造後のチップに残った設計バグを検出するpost-siliconデバッグについては、チップの実際のエラートレースから効率よくデバッグを行うため、チップトレースに対応する実行をC言語などの上位設計で再現する手法と、大規模SoCのデバッグを容易化するためチップの内部情報をバッファにトレースする手法に関して研究を行っている。前者については、チップトレースとC言語シミュレーションの対応を取る手法やそれを利用したC言語上でのデバッグ手法の提案と評価を行い、その有効性を実証した。また後者については、SoC内の各コア間の通信をハードウェアで検出・バッファする手法を提案し、その有効性を例題で実証した。これらの技術に関しては、実際の設計現場の状況を考慮して研究を行う必要性があるため、興味を示している企業と共同研究を行うための基礎的なディスカッションを行った。なお、この研究、及び前記の研究の一部は、東大VDECとスタンフォード大学のMitra教授とで共同で行っている研究に含まれる。さらに、製造後の自動バグ修正機構に関しては、部分的にLUTを自動挿入し、バグ発生時にそれを再プログラムすることで、バグまたは電気的エラーを自動的に修正する手法の基本アルゴリズムを考案し、基礎実験を行い、有効性を確認した。

【マルチコア用高機能ルータ】

- 高機能ルータアーキテクチャ
  前年度に取り組んだ高機能ルータアーキテクチャの基本部をベースとして、ディペンダビリティのレベルを変更できる高機能ルータアーキテクチャを開発した。
- ソフトウェアシミュレータの改良
  前年度に開発したソフトウェア実装によるシミュレータ SimMc の改良をおこなった。主な改良点は、メインメモリを持つノードの実装および 8 ビットルータ・モデルの追加である。
- FPGA システムの整備
  アーキテクチャ評価のための FPGA システムの整備をおこなった。前年度構築した ScalableCore システムを改良し、45 枚の FPGA カードを接続するシステムで安定して動作することを確認した。また、汎用の PC と比較して十分高速なシミュレーションが可能となるという試算を得た。

§2. 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、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)　「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ならびに回路最適化手法
- タイミングエラーリカバリー機能を持つパイプライン演算回路
- マイクロプロセッサアーキテクチャアルゴリズムの形式的検証手法
- 大規模設計記述の等価性検証のためのトップダウン並びにボトムアップ検証手法
- シミュレーション(エミュレーション)ベース検証における検証カバレッジ向上手法
- C ベース言語による上位設計記述の理解・デバッグのための解析技術
- Post-silicon デバッグ手法並びに、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

(3)　「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. 研究実施内容

(文中に番号がある場合は§4(4－1)に対応する)

3.1　「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 21 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

式の改良と予備評価を行った。

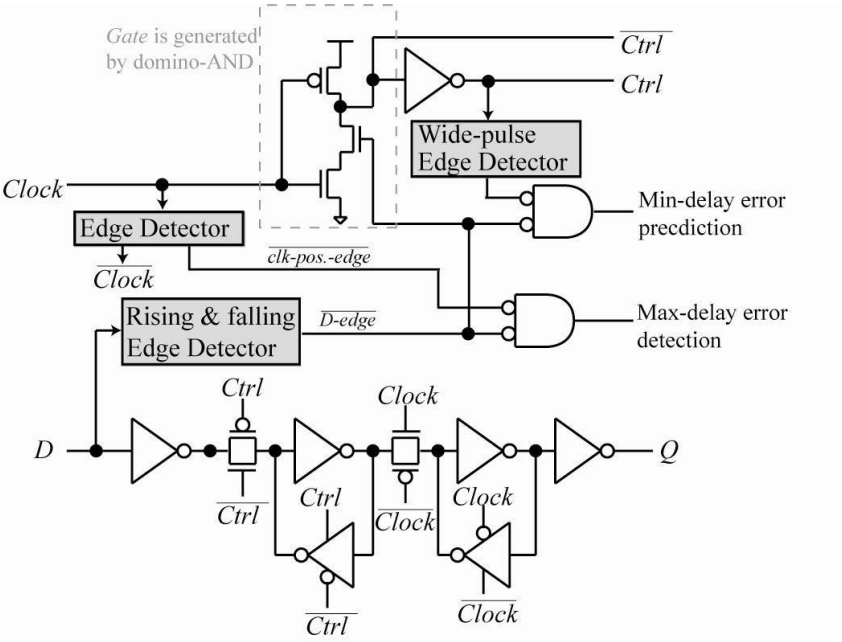

図 3.1.1 タイミング故障を動的に検知・修正する FF

平成 20 年度には、図 3.1.1 に示す遅延補償 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、ショートパスに遅延素子を挿入することによって、この問題の解決を図った。

平成 21 年度には、新たに入力ばらつき(入力の違いによる、出力確定時刻のばらつき)に特に着目して、提案手法を改良した。クリティカルパスの出力を予測する回路を付加することによって、出力確定時刻はより早い時刻とすることができ、タイミング故障発生率を小さくできる。

半導体微細化に伴う製造ばらつき増大によって、タイミング故障を起こさないようなマージンが非常に大きくなり、性能向上を達成することが難しくなってきている。しかし実は、製造ばらつきが遅延に与える影響は、入力ばらつきのそれに比べればとても小さい。例えば 64bit 加算器の場合(図 3.1.2 左)、製造ばらつきによる遅延の変動は±30ps 程度であるのに対して、入力ばらつきの変動(桁上がりなしとありの差)は 100ps 以上と大きい。

さらに出力確定時刻の分布(図 3.1.2 右)を見ると、出力が変化しない場合が半分であり、変化する場合でも大抵の場合、信号は早く確定し、タイミング故障を発生しうる遅いケースは稀にしか起こらないことが読み取れる。すると、Razor のようなタイミング故障検出技術と後述する回復技術(本グループの第二の提案)を組み合わせて、稀にタイミング故障が発生することを、故障回復機構を用意することで許容してやれば、動作周波数を引き上げたり電源電圧を上げ下げしたりして、プロセッサの性能を向上させることができる。

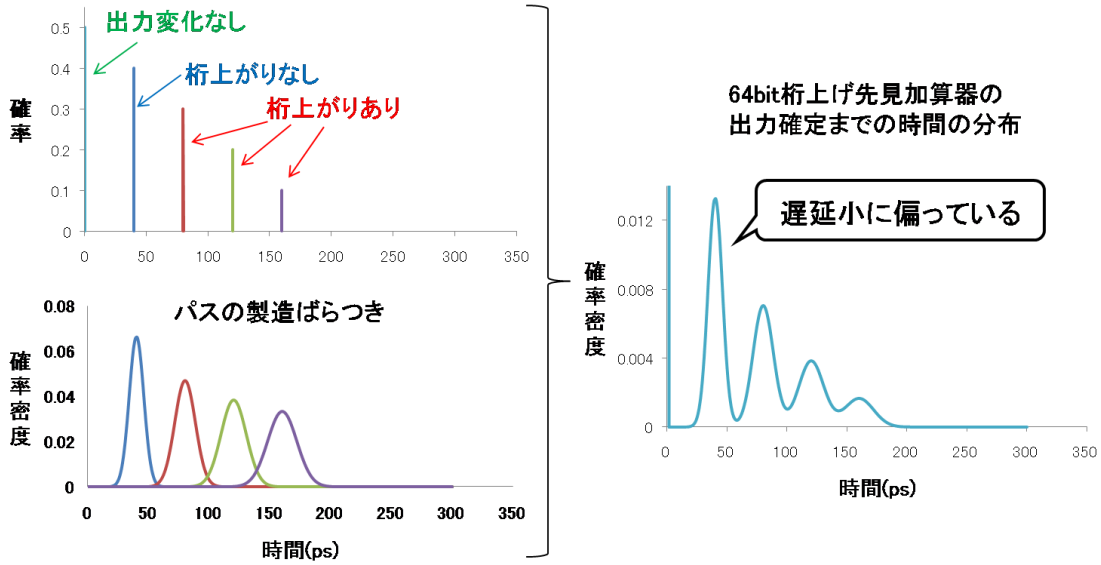

図 3.1.2　64bit 加算器の出力確定までの時間の分布

図 3.1.3 に、提案するクロッキング方式[32, 34] の回路図を示す。提案方式では、クリティカルパスの出力を早期に予測する回路（Pred）を付加して、クリティカルパスが活性化する確率、ひいてはタイミング故障の発生確率をさらに低く抑えるような工夫を施した。

図 3.1.4 は、縦軸が時間、横軸が回路内の信号の伝達方向を示している。入力の違いによってロジックの出力確定する時間が異なる様子を、多数の線で重ねて表している。大部分の線は傾きが緩やか、つまり早期に出力が確定しているが、稀に遅いケースもある。いくつかのステージでクリティカルパスが活性化して「時間の借金」が累積しても、予測が成功すれば借金を大幅に返済できる。万が一、借金が嵩んで破綻しても、すなわちタイミング故障が発生しても、前述のように検出・回復することができる。

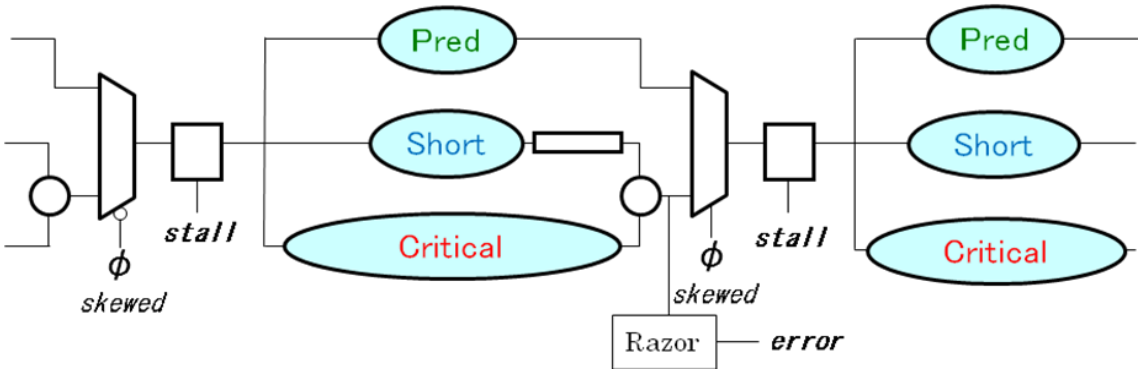

図 3.1.3　タイミング故障耐性を持つクロッキング方式

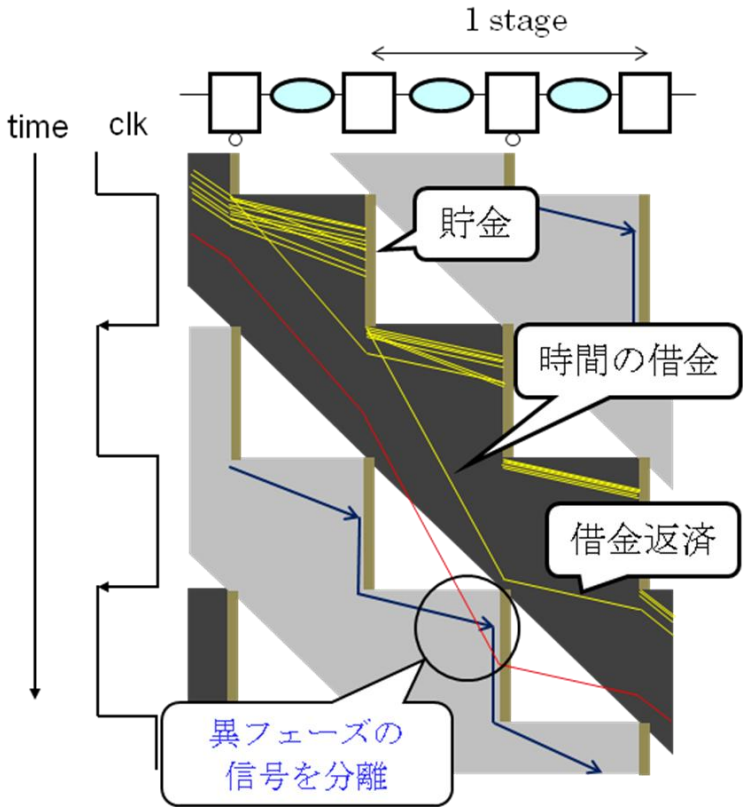

図 3.1.4 タイム・ボローイングの様子

従来の設計手法は「ロジックの遅延＝最長パスの遅延」と固定的に扱っていたため、製造ばらつきの増加に対応できなかった。提案手法はロジックの遅延は（静的にも動的にも）大きく変動する（ばらつく）ことを前提としており、このばらつきを逆に利用して柔軟に対処する手法となっているため、タイミング故障耐性と性能向上を両立できる。

前年度はまず、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。本年度は、この方式の効果を更に高めるため、回路の出力を予測する予測回路を追加する方式についての検討を行った。今後は後述のタイミング故障回復手法と併せて、図 3.1.7 下の新テストベッドに実装・評価する予定である。

第二に、平成 20 年度に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、テストベッド製作・回路詳細化によって検証した。

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 3.1.5）。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 3.1.6）。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。

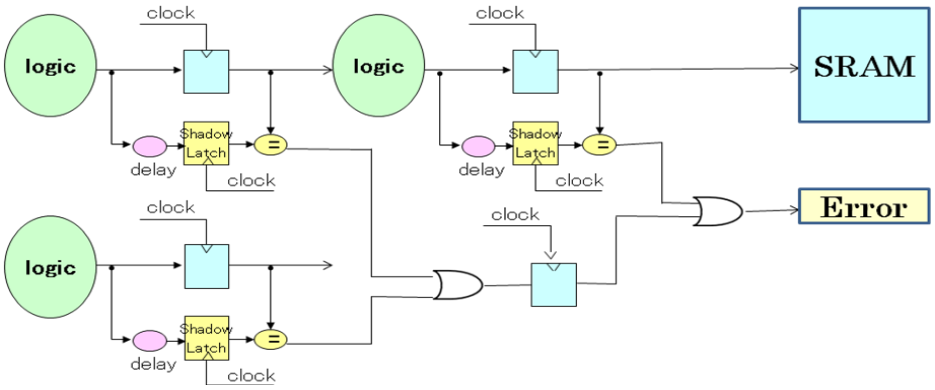

図 3.1.5. タイミング故障の検出と通知ネットワーク

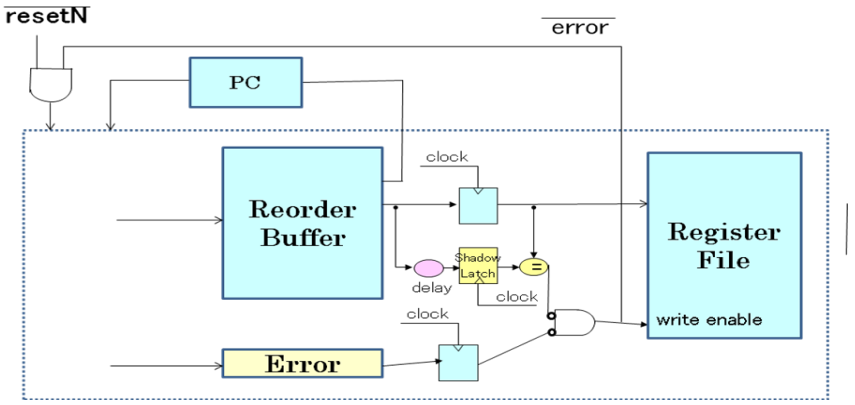

図 3.1.6. レジスタファイルとプログラムカウンタの保護

この機構を、前年度に引き続き、テストベッド（図 3.1.7 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ている。具体的には、平成 20 年度 2-way 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テストを行ったが、平成 21 年度は、より現実的なプロセッサにおける回路の評価のため、新たに 4-way スーパスカラプロセッサの開発を開始した。本開発は、当初の全体計画には入っていなかったが、本研究の真の実用化のためには必要なプロセスである。この結果として、次に述べるように、4-way スーパスカラプロセッサの FPGA への本格実装は、平成 22 年度にかけて引き続き行い、これを詳細に評価・検証することとした。

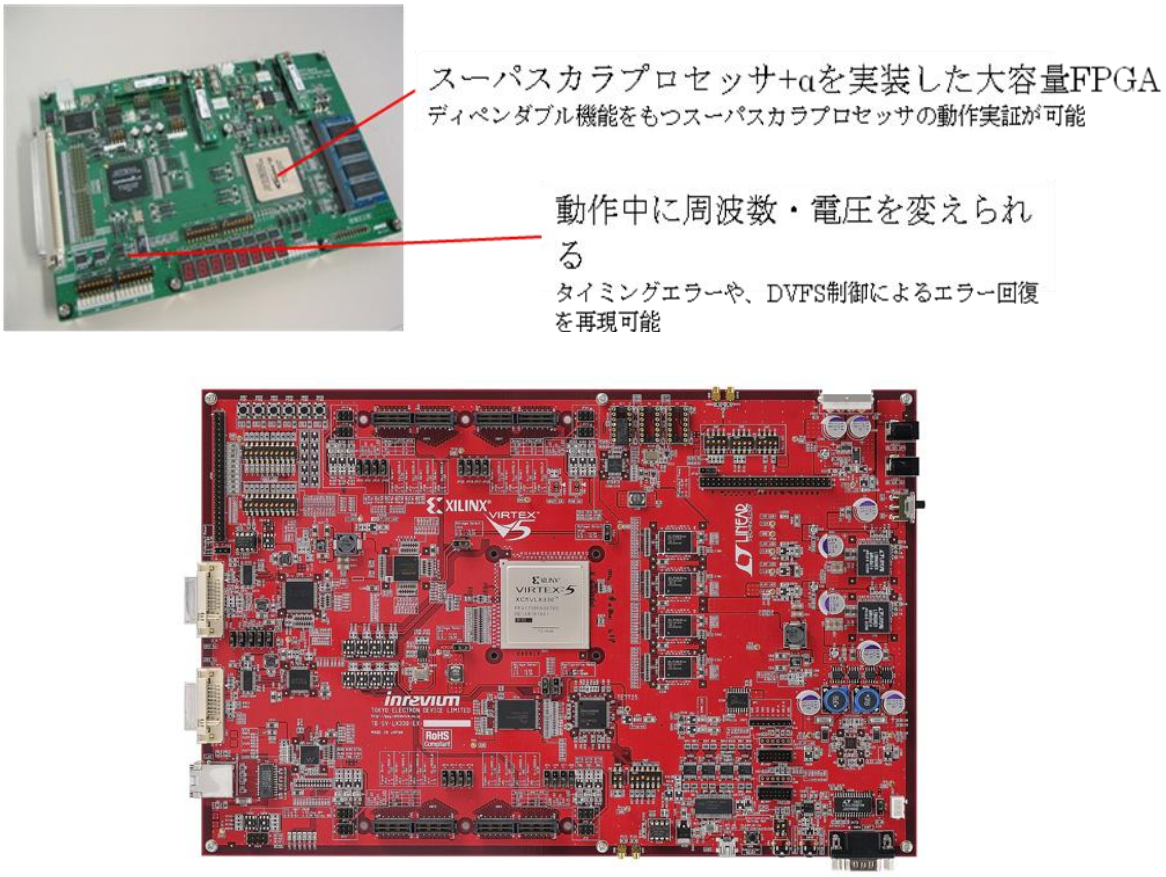

図 3.1.7. テストベッド（上：従来品、下：新規開発品のベースとなる評価ボード）

既存のタイミング故障回復手法は、制御系パス上でのタイミング故障を回復できず不完全である上に、メモリの故障やエラーには ECC で対処すれば十分であると考えられていた。実際、Razor チームが本年度 Razor2 と名付けて発表した論文（2006 年度発表の改良版）では、回復方法の改善がアピールポイントの一つであったのに関わらず、制御系パス上のタイミング故障はやはり意識されておらず、さらに前回に引き続き InOrder なシンプルなプロセッサへ実装したためにロード・ストアキューの問題にも気づいていなかった。タイミング故障耐性が重要になってくる中で、本研究のような包括的・一般的な手法を提案する意義は非常に大きい。

平成 22 年度には、以上述べた高度なスーパスカラプロセッサの詳細設計を完了し、図 3.1.7 上の新テストベッドへの実装をさらに進める。現実のプロセッサ上での実用化に必要なより詳細な評価を進め、方式の有効性の検証を行う。現実的な性能を持つプロセッサ上への実装をすることで、前述のような手法の問題点を発見・改善し、本研究が実戦的な成果を生むと期待している。論理動作や遅延だけでなく、回路規模・コスト・電力を評価し、商用化の可能性について詳細に検討する。

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実装・改良を行った[33]。

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 3.1.8、3.1.9)。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・深海など限られたものである(その重要性は言を俟たない)。このような限られた市場のために専用に LSI を開発することは、現実的とは言えない。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。

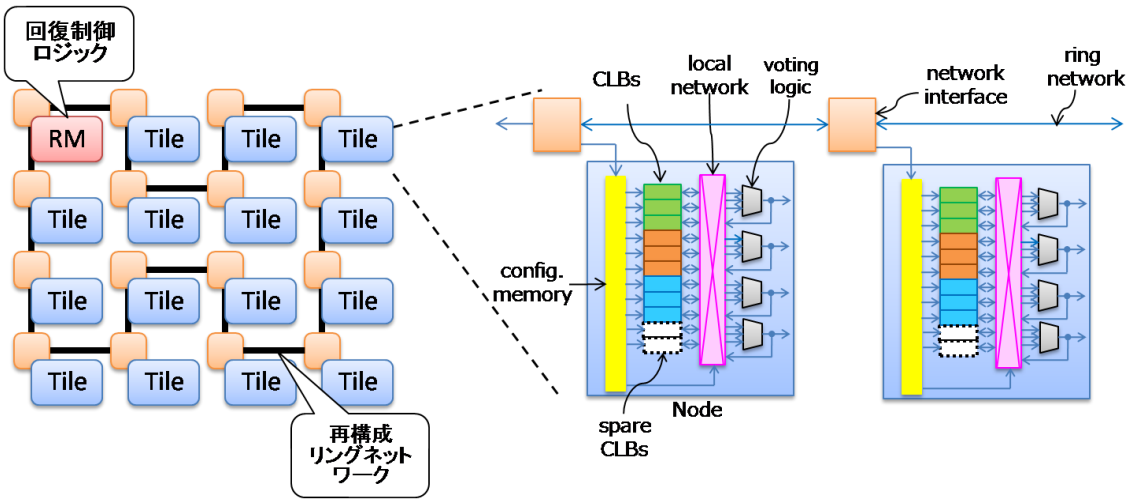

図 3.1.8 耐永久故障アーキテクチャ

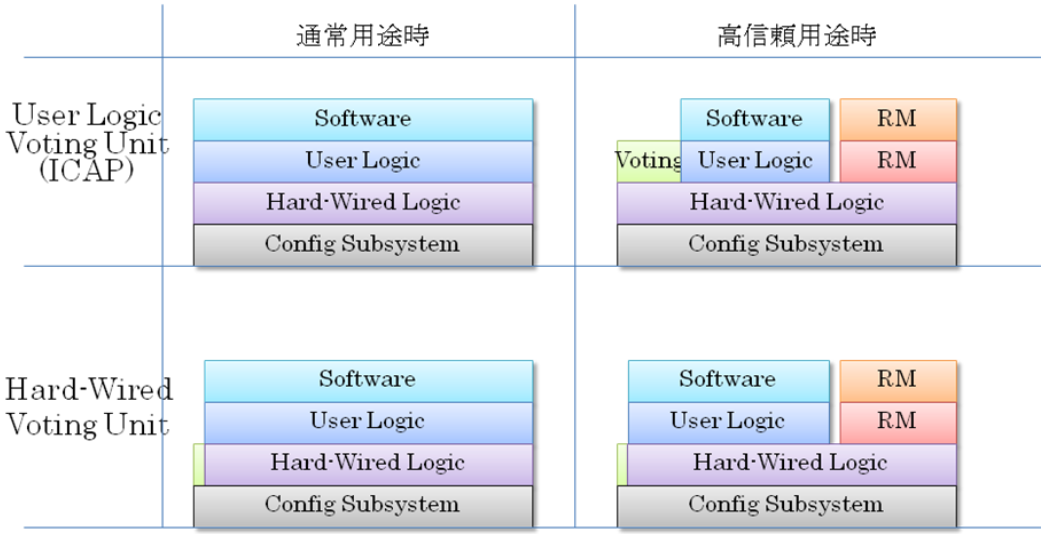

図 3.1.9 通常用途時と高信頼用途時の使い分け

平成 20 年度には、提案を検証するためのテストベッド（図 3.1.10）の FPGA 回路設計を行った。具体的には、図 3.1.8 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。

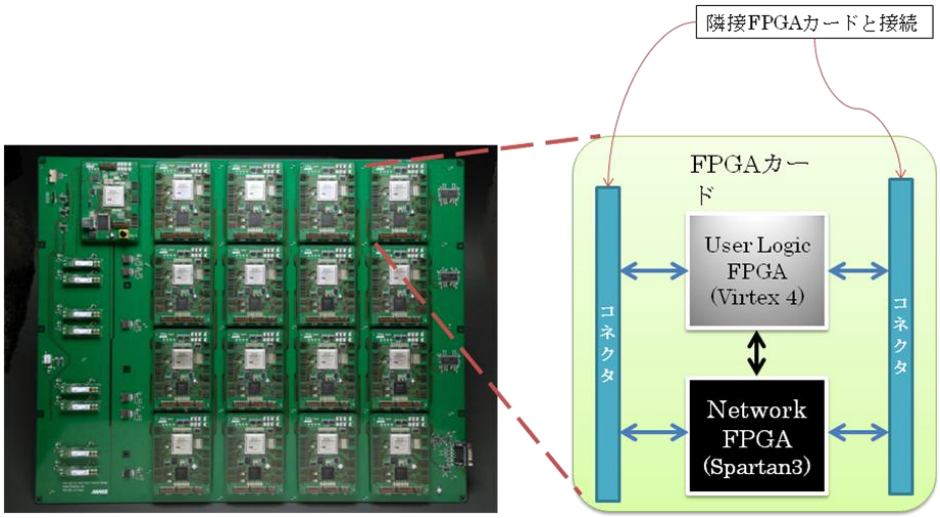

図 3.1.10　耐永久故障 FPGA 用テストベッドボード

平成 21 年度には前年度に引き続き、提案手法の実装を進めた。そしてその際浮上した問題について、改良を施した。

永久故障耐性は Triple Modular Redundancy（TMR）により実現される。3重化されたモジュールのうち一つが故障しても、正常動作を継続でき、さらに故障したロジックは FPGA 上の別の箇所に動的に再構築して回復する。TMR 状態を回復するためには、TMR を構成する故障したモジュールの状態、すなわち、フリップフロップ（FF）の値を、スペア・モジュールに無時間的にコピーする必要がある。以前は特別な事をしなくても、再構築後に残りの2つのモジュールと同様の入力を与えて動作させれば、自然と FF の値がスペア・モジュールにコピーされると考えていた。しかし予備評価を通じて、書き込み不可に設定された FF にはコピーされない問題が浮上した。また、書き込み制御を行うために FF に付いている小さな自己ループは、TMR で投票されずに FF に書き込まれてしまうため、正確な値であることを保証できない。

この問題を解決するために、FF への値コピー用の特別なネットワークを設けることを検討している。書き込み不可に設定されているFFにも、再構築時にはこのネットワークを通じて強制的に値をコピーする。また、FF の小ループは粗いルールで生成し、故障の発生しないように製造する。これ以外の小ループは、TMR を通るように遠回りをして配置することになるが、FF 以外でそのような小ループは稀であるから、この制約は問題ないと考えている。当初の全体計画では、平成 21 年度に耐永久故障機構の入った FPGA 用テストベッドを実装するはずであったが、本ネットワークの導入のための研究により、本格実装は平成 22 年度となった。すなわち、平成 22 年度は、ここで新たに提案した手法の実用的な実験を進め、正確な動作検証を行う。

3.2 「形式的検証」グループ

- 上位設計記述に対する形式的等価性検証ツールの開発

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、実際に企業での評価を通して得られたフィードバックに基づき、2 つのループの等価性を効率的に検証する手法の提案と SystemC 言語フロントエンド開発の検討を行った。

等価性検証ツールの内部構成図を図 3.2.1 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[13,14,17,20]。

今年度の企業による評価は、企業側で用意された約 20 個の例題(等価性を検証すべき約 20 組の SpecC 言語設計記述)について行われた。例題は、実設計フローにおいて生じる等価性検証を行う必要性がある設計変更・最適化を想定して作られたものである。提供された例題に対する評価結果を表 3.2.1 に示す。表からも分かるように、検証することができたかどうかや検証時間の長さと、設計規模や条件分岐数は必ずしも強く相関しているわけではない。これは、小さい例題や単純な例題であっても、現在の検証ツールで検証することができない要素が含まれる場合、検証結果を得ることができないためである。特に、modulo 演算や除算がある場合や、動的に繰返し回数が決まるループ実行がある場合については、現在の手法では検証することができない。一般的に、比較する2つの設計記述の計算上の差異が少ない場合には、両者を同じデータパス上で実行するようにマッピングし直すことで、比較的大きな設計記述に対する等価性検証を効率よく行えることを示した[2]。評価を通して、1) 2 つのループの等価性を効率的に検証する手法が必要である、2) 実際の上位設計の多くは SystemC 言語で記述されるため SystemC 言語フロントエンドが必要である、の 2 点の強い要求が企業側からあった。これ以外にも、順序動作の等価性指定は必ずしも自明ではないため、何らかの計算機支援が望まれるという要望もあったが、研究スケジュールの関係上、今年度は pot-silicon デバッグの研究内で基礎検討のみ行った。

今年度は、ループの等価性をデータグラフ上で効率的に検証する手法の提案と実装を行った。提案手法では、ループ内でアクセスされる配列のインデックスの関係をエッジに付加した特別なデータフローグラフを用いて、任意の出力配列を計算するために必要な部分とそのときのループ繰返し回数を導出し、必要部分のみを記号シミュレーションして効率的に検証を行うことができる。提案手法では、従来手法では繰返し回数が多い場合に現実的な実行時間で終了できなかった例題であっても、数秒以内に検証を行うことができることを確認した。

また、SystemC 言語フロントエンドについては、その仕様を策定した。その際、実用的に良く使用される SystemC 言語のサブセットを定義し、その範囲の記述を SpecC 言語に変換することにより、

本研究で開発している検証ツールに入力することができるようにした。外部の企業と共同で開発を進めており、来年度前半には、SystemC 言語記述から現在の等価性検証ツールで受け付けることができる SpecC 言語記述へ変換するフロントエンドソフトウェアを使用可能になる予定である。

本等価性検証ツールは、後述する JAXA の元で開発されたシステムレベル設計ツール Elegant 内の検証ツール Venus を正式に移管を受けた。これは、将来、本等価性検証ツールとの融合を図っていることになっている。またこれを通じて、Elegant ツール利用者に対する本等価性検証ツールの評価も進めていくことになっている。現在、車関係企業に対するツールの説明や評価について打合せを行っている。

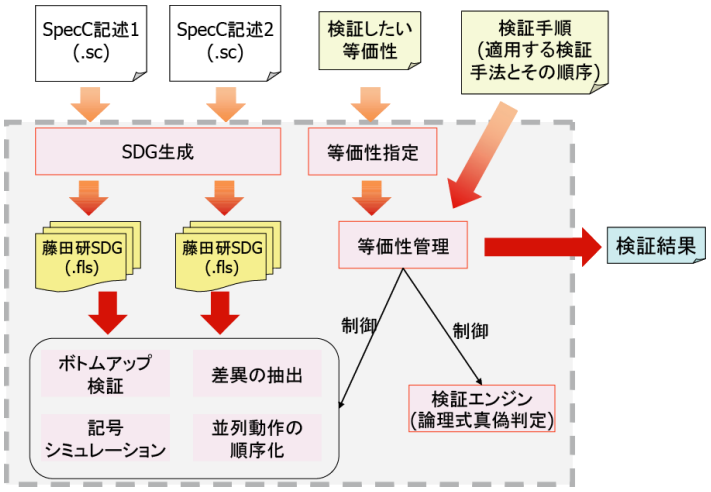

図 3.2.1 等価性検証ツールの構成

表 3.2.1 等価性検証ツールの評価結果

実験名	例題内容	記述1		記述2		検証時間	検証結果
		行数	分岐数	行数	分岐数		
K1	IDCT	246	16	564	164	> 24 hour	終了せず
K2	ループ処理	38	2	29	1	2.6 sec	等価と証明できず
K3	入力数値の平均化	21	0	26	1	0.1 sec	等価と証明できず
K4	モード制御付き算術演算	33	4	52	4	0.1 sec	等価
D1	モード制御付き算術演算	23	1	33	1	0.2 sec	等価
D3	モード制御付き算術演算	24	1	28	2	24 sec	等価と証明できず
D4	配列演算	28	1	34	1	0.3 sec	等価
D5	算術演算	23	0	65	0	0.1 sec	等価と証明できず
D6	算術演算	18	0	29	3	0.1 sec	等価と証明できず
D8	算術演算	26	1	34	5	0.2 sec	等価
D9	配列要素の総和	80	1	43	1	0.4 sec	等価と証明できず
D10	2入力の加算	17	0	24	1	0.1 sec	等価
H1	ループ処理	19	0	34	0	0.6 sec	等価と証明できず
IDCT1	逆離散コサイン変換	651	0	658	0	0.5 sec	等価
IDCT1	逆離散コサイン変換	651	0	662	0	1.5 sec	等価
IDCT1	逆離散コサイン変換	651	0	738	0	2.1 sec	等価
ELLIP1	楕円フィルタ	48	0	79	0	0.1 sec	等価
ELLIP1	楕円フィルタ	48	0	81	0	0.1 sec	等価
ELLIP1	楕円フィルタ	48	0	127	0	0.1 sec	等価

- 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。前年度は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。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。

本年度は当研究グループで研究開発を行っている等価性検証ツールFLEC上へ本提案手法を実装し評価を行った。本実装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。また等価性規則をExSDG上のパターンとみなすことで、ルールベース等価性検証を単純なグラフマッチング問題に帰着できるため、効率的に検証を行うことが可能である。またボトムアップ検証において最大の課題である初期等価点の発見手法として、ランダムシミュレーションに基づく発見手法を実装した。これはシミュレータにはSpecCリファレンスコンパイラを使用しており、自動的に初期等価点を発見することができる。ボトムアップ解析が有効な設計記述の場合には、数千行規模の設計記述

に対して、従来手法では数十分から数時間であったものが数十秒で処理できるなど、従来比100倍以上の高速化を達成している [21]。

また提案手法をFLEC上に実装したことにより、トップダウン手法とボトムアップ手法を組み合わせることが可能になり、(多様な設計記述に対応できるという意味で)よりロバストな検証が可能になると考えられる。これらについては、今後検討を進め、評価・改良していく予定である。

- 等価性検証ベンチマークの作成

等価性検証ツールの評価を世界的に行うために、当研究グループと同様に等価性検証ツールの研究を開始している Oxford 大学と共同で、等価性検証ベンチマークの作成を行っている。専用のウェブページを作り、そこにベンチマークを集めるとともに、等価性検証技術やツールに関する技術紹介や情報交換を行う予定である。現在、本研究で開発を行っている検証ツールの評価・テスト用に作成した約 50 個の小規模例題の集合をベンチマークとして提供する準備を行っている。また、等価性検証の性能を評価できる規模の大きな例題の作成を検討しており、選定を進めている。なお、Oxford 大学でも企業1社と共同で SystemC 設計記述に対する形式的手法とそのツール化の研究を行っており、研究情報や企業での評価結果についても適宜可能な範囲で情報交換を行うことになっている。これらの作業並びに、ウェブページを共同で管理運営することで、C 言語ハードウェア設計検証に関する情報の世界的中心になることを目指す。企業で実際に使ってもらえるか否かは、それあるいはそれに関係したものが世界的に有名か否かにも大きく影響を受けるため、このような努力は非常に重要であると考えている。

- 高性能カスタム算術演算回路の自動生成・形式的検証・最適化

高性能あるいは高効率(低消費電力)計算に要求される、応用に特化したカスタム算術演算回路最適化・形式的検証技術[1,3]の開発を行っている。上位の仕様記述から、ビット幅なども考慮してハードウェア実装に必要な多項式を自動生成・最適化し、それを実現する算術演算回路を外部・内部から発生するドントケアを考慮して最適化する手法[5,12]並びに、それらの処理の正しさの形式的検証を行うツールについて、研究開発している。すでに Modulo 等価性を考慮した等価性検証技術を開発していたが、今年度はそれを発展させ、多項式レベルの最適化手法を考案・実装・評価した[6,9]。多項式をハードウェアで計算する場合には、掛算の回数を減らすことが重要であるが、共通式の単純な括りだしなど、従来手法ではほとんど最適化できなかった。本研究では、多項式を効率よく計算機上で操作できる HED グラフを利用することで、試行錯誤や式の大幅な変換を伴う最適化手法を開発している。実装・評価したところ、従来手法と比較し、掛算の回数を数十%削減できることが分かった。また、最適化された多項式からハードウェア(回路)を自動生成することで、従来もっとも広く利用されている手法と比較し、回路面積・遅延の両面で 40%から 50%の縮小化を達成している。また、従来全く利用できなかった大規模外部ドントケアを利用した算術回路自動生成技術などを新規に考案・実装・評価した。結果として、回路規模・遅延で従来比30%から 40%の縮小化を達成している。これらの技術を統合すると、算術演算回路向けの上位仕

様レベルからの自動合成ツールを実現することができる。制御回路の自動合成については、従来からカリフォルニア大学バークレー校の SIS や ABC に代表されるツールが有名であり広く利用されてきたが、それらのツールは、算術演算回路に対してはほとんど無力であり、算術演算回路はライブラリにある既設計回路をそのまま利用することになる。したがって、必要十分なビット幅の演算を行うようなカステム算術演算回路の自動生成はできなかった。現在、FPGAなどを利用したカスタムメードの高性能計算（カスタムスーパーコンピュータ）が銀行や資源探索などの企業で利用されるようになっている。このような設計には高度に最適化されたカスタム算術演算回路が必須であり、当研究成果によるツールを実現することで、より高性能カスタムスーパーコンピュータをより短期間に正しく設計できるようになると考えられる。次年度以降は、この種の応用についても研究していきたい。特に、複雑な算術演算の組合せをパイプライン化することで効率よく実装する手法に関しての研究も進めており、回路最適化をある程度以上行う際に必須となるレイアウト（配置・配線結果）を考慮した合成手法を考案し評価している[4]。

さらに、カスタム算術演算などをパイプライン処理する場合に、タイミングエラーが発生した場合に自動的にリカバーする新規回路機構を考案した[15]。この回路構成は図 3.2.2 に示されるもので、従来のタイミングエラー検出フリップフロップ Razor にさらにもう1つフリップフロップを付け加え、タイミングエラー発生時もパイプラインの前段からくるデータを取りこぼすことなくパイプライン処理を続行できるようにしたものである。この回路を利用することで、タイミングエラー発生時もパイプライン処理を続行できるため、意図的にタイミングエラーが発生するまでクロックを加速する overclocking を行っても正しく動作させることが可能となる。簡単な実証を行うため、提案手法を FPGA ボード上に実装し、いくつかの回路で評価した。図 3.2.3 に示すような算術演算（多項式の計算）を6段パイプラインで実現する回路を生成し、回路面積と性能を評価した結果を表 3.2.2 に示す。LUT 数では Razor フリップフロップを利用したタイミングエラー検出のみの回路をほぼ同等であり、クロック周期は 30%高速化でき、それがそのまま処理時間の高速化になっている。実用的な回路として、インターネットなどのパケット通信から予め指定された string とマッチするパケットを検出する回路（string matching の回路）を FPGA で実現したものに対して適用したところ、同様に30%程度の高速化を達成している。次年度以降は、マイクロプロセッサへの適用などを行い、手法の評価・改良を行っていく。なお、この overclocking 手法は、上のカスタム算術演算回路自動生成手法と組合せることで、カスタムスーパーコンピュータ計算をさらに 30%程度高速ができると期待される。

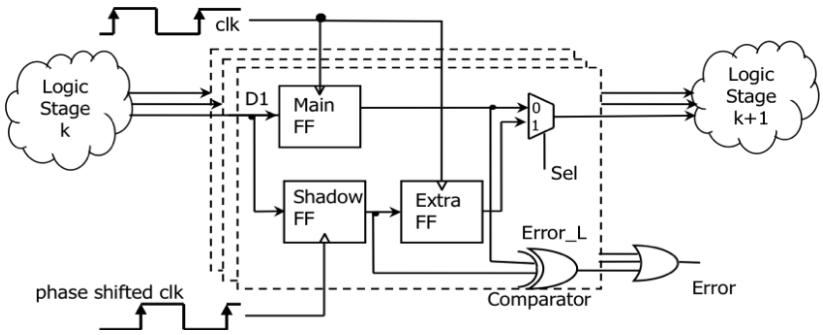

図 3.2.2　提案するタイミングエラー検出と修正を自動で行う回路

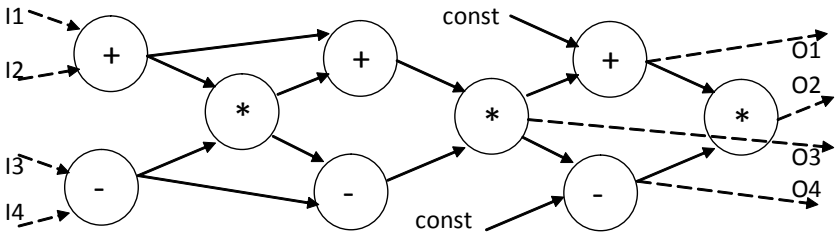

図 3.2.3　例題の回路（多項式計算）

表 3.2.2　overclocking の実験結果

Circuit	**# FFs**	**# LUTs**
1- Original (w/o error detection/correction)	212	362
2- Razor (w/ Razor error detection)	309	439
3- Our Basic circuit	549	838
4- Optimized circuit	411	457

Clock Frequency	# timing errors	Timing error rate	# cycles	Operation time (us)
100 MHz	0	0%	1030	10.30
110 MHz	3	0.05%	1031	9.37
120 MHz	121	2%	1036	8.63
130 MHz	808	13%	1036	7.97
140 MHz	Does not work correctly.			

- 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。図3.2.4に示すように、この手法は明示的に全ての入力パタ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。

前年度は、この手続きの一部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を提案した。しかしながら大規模な設計で検証カバレッジ100%を達成することは本質的に計算困難な問題であり、このため大規模な設計へそのまま適用することは難しい。本年度は現実的な設計への適用を目指し、従来のシミュレーションベース検証を行う際の検証カバレッジを向上する手法を提案した[11]。シミュレーションベース検証では、(条件付き)ランダムで(あるいは、人手で)入力パターンを生成し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。提案手法ではパタン毎にスキップキューブを計算し考慮することで、一回のシミュレーションによって多数の入力パタンのシミュレーションと同等の効果を得られる。従来の準形式的手法ではパタン生成に二分決定グラフ(BDD)を用いていたため、大規模設計への適用が困難であった。提案手法では、パタン生成問題を充足可能性判定(SAT)問題に帰着し効率のよいSATソルバを用いることで、大規模回路への適用を可能にしている。また、本提案手法はFPGAベースのエミュレータなどへの応用も期待できる[11]。つまり、FPGAエミュレータに本検証手法を実装することで、従来に比べ、飛躍的に検証カバレッジが向上することが期待できる。また、前年度に提案・試作を行った検証指向FPGAであるVDD-FPGAを組み合わせることで、高品質なシミュレーションパターンを高速に生成することも可能となる。これらについては、今後検討を進め、評価・改良していく予定である。

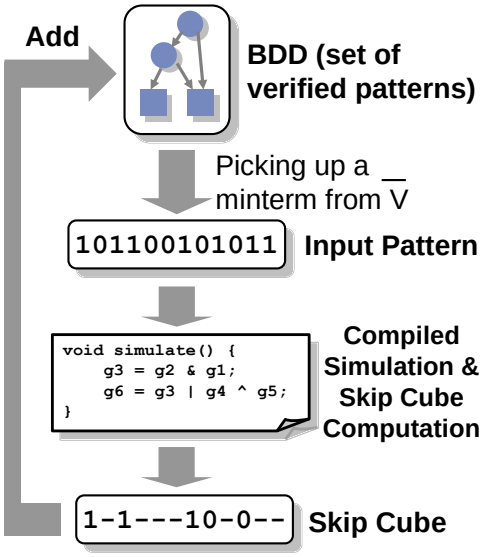

図 3.2.4 準形式的検証手法の流れ

● 具体値/記号値混合シミュレーションを利用した検証カバレッジの向上

本研究では、前年度までに具体値によるシミュレーションと記号シミュレーションを合わせて適用することにより、より早期に設計誤りを発見する手法を提案した。この手法では、ユーザ駆動で具体値シミュレーション(与えられた1つの入力パタンのみに対するシミュレーション)によって設計のある状態まで行き、その状態から記号シミュレーション(可能性のある全ての入力パタンに対する網羅的なシミュレーション)を行い、探索した範囲内に設計誤りがあるかどうかを調べる(図 3.2.5)。既に、前年度に等価性検証ツール FLEC 上に実装を行い、具体値シミュレーションと記号シミュレーションを行う環境ができている。今年度は、さらにツール実装を進め、その評価を行った。予備的な実験では、企業から提供された FIR フィルタとエレベータ制御の例題において、提案手法を用いて設計誤りを発見することができた。FIR フィルタの例題は SpecC 言語記述で約 170 行であり、設計誤りを発見するまで 61 サイクルを要した。一方、エレベータ制御の例題は SpecC 言語記述で約 3000 行であり、設計誤りを発見するまでに要したサイクル数は 20 サイクルであった。これらの結果から、現在のツールでは、数千行規模の例題において数十サイクル分の深さにある設計誤りを発見することができると言える。

ユーザが探索範囲を指定する際に、前項の「HW/SW 協調動作による検証高速化手法」と組合せることにより、深さ優先探索の際の効率を大幅に向上することができる。これは1つのパスをシミュレーションすれば、スキップキューブを利用して同様の結果となるシミュレーションパターンの集合を自動的に計算でき、次回のユーザ指定の探索範囲から削除できるためである。今後は、このような各種検証手法の融合のよる処理の効率化も図っていく予定である。

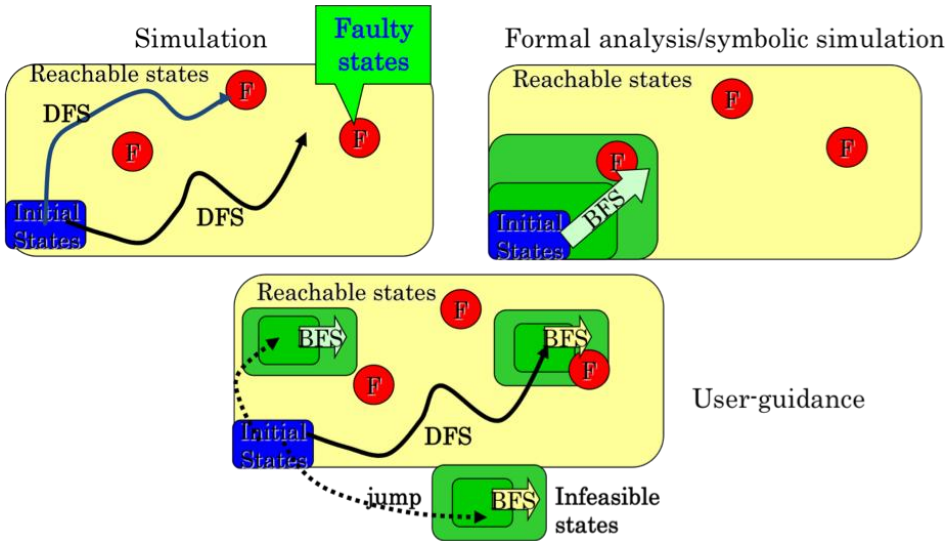

図 3.2.5 ユーザ駆動幅優先アプローチ

● ELEGANT ツールとの統合の検討

JAXA が開発したシステムレベル設計支援ツール ELEGANT では、仕様レベルで記述された SpecC 言語設計記述に対して、アーキテクチャ決定・モジュール間接続関係の決定・通信プロトコ

ルの決定を行いながら、設計を段階的に詳細化し、最終的に動作合成ツールの入力となり得る SpecC 言語設計記述を得ることができる。ELEGANT では、この詳細化における等価性を保証するために、Venus と呼ばれる形式的等価性検証ツールが提供されている。藤田研究グループで研究開発を進めてきた FLEC とこの JAXA が開発した Venus は互いに上位設計記述の等価性検証を目的としており、両者の長所を合わせることによって、より広範囲の設計記述や等価性を検証することができると考えられる。今年度、藤田研究グループは Venus を JAXA より正式に移管を受けている。今後、藤田研究グループの FLEC との融合を進め、ELEGANT 環境でも利用可能としていく予定である。既に Venus ツールの基本的な使い方や例題評価は終えている。現在、移管された Venus ツールのソースコードをレビューし、どの部分について FLEC との融合が可能であるかの検討を進めている。

Venus では、検証対象となっている入出力信号の遷移について、次の 4 種類の等価性を検証することができる。

1) 同期などのイベントが起こった全てのタイミングにおいて、注目している 2 つの変数値が等価である場合のみに、その 2 つの変数が等価である
2) 注目している変数値が変化した場合のみに着目し、その遷移が等価であれば 2 つの変数は等価である
3) 時間経過が起こった全てのタイミングで、2 つの変数が等価であれば、等価である
4) 時間経過が起こり、かつ、変数に代入があったときの変数の遷移が等価であれば、2 つの変数は等価である

1)と 2)は時間経過を含まない設計記述に対する等価性であり、3)と 4)は時間経過を含む設計記述に対する等価性である。現在の FLEC では、そのような等価性をサポートしていないため、ありえる拡張の 1 つとして、より広範囲の等価性を検証できるように、両者で検証できる数学的な定義としての等価性を統合していくことが考えられる。

また、Venus では記号シミュレーションを適用する前に、ランダムシミュレーションを行い、内部等価点候補を探す作業を行っている。これは、等価性検証の問題を分割して高速化を図るために有効な手段であり、FLEC においても導入を検討している。ただし、その際に、時間経過を含む設計記述における等価性検証問題の分割は研究テーマとして、来年度以降の研究が必要であると考えられる。

- アーキテクチャ検証

アーキテクチャアルゴリズムを形式的検証に検証する手法とそのツール化を進めている。現在までにRazorフリップフロップなどでタイミングエラーを検出し、エラーから自動的に回復するアーキテクチャアルゴリズムが提案され、シミュレーションなどでその有用性が示されている。しかし例えばマイクロプロセッサ企業からみると、エラー回復のアーキテクチャアルゴリズムが複雑であるため設計の正しさの保証が困難であり、実利用を躊躇している状況である。検証が不十分なため、どうしても安全サイドの保守的な設計に成らざるを得ず、結果的にマイクロプロセッサの性能を損ねてし

まう可能性がある。そこで対象をマイクロプロセッサとし、各種アーキテクチャアルゴリズムを形式的に検証する手法とその検証環境（ツール化）について研究している。今年度は、単純パイプラインプロセッサ、スーパスカラプロセッサ、out-of-orderプロセッサに対し、タイミングエラー回復アーキテクチャアルゴリズムの形式的検証を行い、数秒から数時間（対象プロセッサの複雑度による）で検証できること、並びに、検証手法を拡張し、アーキテクチャアルゴリズムの最適化にも適用できることを確認した[16]。

- Post-silicon デバッグ支援手法

現在、多くの VLSI 設計では、設計段階で完全に設計誤りを検出・除去することができないため、チップ製造後にも設計デバッグを行う必要がある。これは Post-silicon デバッグと呼ばれている。Post-silicon デバッグではチップ実行結果の入出力シーケンスを解析して設計誤りを特定・修正するため、シミュレーションによって全ての内部信号値を見ることができる製造前のデバッグに比べて、観測性が低く、より困難な作業であると言える。本研究では、上位設計記述を利用した効率的な Post-silicon デバッグ手法[10]とトランザクションレベルにおけるデバッグ支援手法[7,8]に取り組んでいる。

提案した上位設計を利用した効率的な Post-silicon デバッグ手法では、チップ実行で得られた入出力シーケンスを抽象化することによって上位設計の入出力シーケンスを生成し、上位設計記述中で設計誤りを効率的に特定する。本研究で提案しているデバッグ支援フレームワークを図 3.2.6 に示す。このフレームワークでは、与えられたチップのエラーパタンから上位設計記述においてシミュレーション可能な抽象化されたエラーパタンを生成する。この際に、図の”Mapping”において、上位設計とRTL 設計の間で対応から得られるポートとタイミングの対応を抽出する。図 3.3.7 は上位設計とRTL 設計の間の入出力パタンのマッピングの例である。マッピングは、入出力ポートのマッピングペアの集合 $Mp = \{pairP(i) \mid pairP(i) := (psA(i), psB(i))\}$ (Mp: ポートマップ，$psA(i) \subseteq$ 設計 A のポートの集合、$psB(i) \subseteq$ 設計 B のポートの集合)と、Mp の各ポートペア $pairP(i)$ 上のポートにおけるタイミングの対応 Mt $(i) = \{ pairT(j) \mid pairT(j) := (\langle t\_psA(i)\rangle, \langle t\_psB(i)\rangle) \}$ ($t\_psA(i)$, $t\_psB(i)$: $psA(i)$と $psB(i)$におけるタイミング)から成る。図 3.3.7 の例では、Mp = {$pair(1)$ = (<a1>, <b1, b2, b3>) }、$pair(1)$= (<a1>, <b1, b2, b3>)の各ポート上の対応すべきタイミングのマッピングは $Mt(1)$={ (<2>, <0,1,2>), (<5>, <2,3,4>), (<8>, <2,3,4>), …}である。これまで、スループットとレイテンシが一定の繰返し動作や簡単なハンドシェーク通信について、このマッピングを自動生成する手法を提案した。現在は、様々な設計におけるマッピングを数学的に表現する方法の検討と、その指定されたマッピングに従って、上位設計における入力パタン自動生成手法の検討を行っている。

上位設計とRTL 間の入出力トレース間のマッピングがとれた後は、そのトレースを上位に適用し、シミュレーションを基本とした提案手法でデバッグに有用な情報を求める。図 3.2.6 の”Error-relevant Element Extraction”では、エラーに関連がある部分を依存関係に着目して抽出する。このとき、上位設計における実行とレースから依存関係を表すグラフ D-SDG（Dynamic System Dependency Graph）が作られる。D-SDG はトレース上の各実行時における文をノードとし

て持ち、ノード間のデータ依存・制御依存をエッジとして持つグラフである。D-SDG 上の依存関係をたどることによって、エラーが観測される出力変数に影響を与えるステートメントとそのタイミングが求められる。複数のエラートレースから得られる D-SDG の共通部分が、設計誤りを含む文の集合となる。さらに、設計誤り箇所を小さい範囲で特定するため、現在、前述の共通部分に含まれる文を、誤りである可能性の高さによって順序付けるヒューリスティックを検討している。

今年度行った予備的な実験では、SpecC 言語で記述された Elevator Controller 設計(239 ステートメント)に対して、設計誤り位置の特定を行った。この例題は、D-SDG による依存解析と共通部分抽出により、誤りの可能性がある部分を全体の 2.9 パーセント(7 ステートメント)に絞ることができた。

実際のチップのエラートレースが巨大となることも珍しくない。例えば、マイクロプロセッサや SoC では、動作を開始し、数分から数時間経って初めて動作がおかしいことに気づく場合も多い。このような場合には、チップトレースからチップの実行をシミュレーションで初期状態から辿って再現することは、原理的に不可能であり、チップ内の動作を何らかの方法でモニターし、必要に応じてバッファに格納しておくような、ハードウェアサポートが post-silicon デバッグでは不可欠となる。従来から設計者が指定した内部信号線を設計の指定のタイミングや条件によるバッファへ単純に格納するような手法を利用されているが、デバッグに有効な情報が必ずしも得られていないという問題がある。設計バグだけでなく、電気的なエラー(間欠故障やソフトエラーを含む)も対象とする場合には、現状ではデバッグに数日から数週間かかっていると言われている。また、誤った出力からエラーの伝搬をバックワードに解析する手法も、現状では状態ごとに行っており、数百サイクル以上辿っていくことは現実には難しい。そこで、本研究では、チップのエラートレースを用いて C 言語レベルでデバッグを効率的に行えるように、エラーの伝搬を状態ごとではなく、C 言語設計記述で扱われるトランザクションレベルで解析する手法の確立を目指している。1つのトランザクションは多数のサイクルからなるので、まずトランザクション単位でバグの解析を行い、その後、ターゲットとなったトランザクション内部の解析を行うという階層的なアプローチを取ることで、デバッグ作業の大幅な効率化を図る。これを実現するため、図 3.2.8 に示すように、チップ内の各コア(ブロック)間の通信をハードウェアでモニターしバッファする回路を提案している。ターゲット SoC の通信に利用されているプロトコルの定義から自動的に各通信の起動と終了を検出する回路とそれをバッファする回路を生成し、それを設計に追加する。チップのエラーが発生した場合には、通信状況を格納したバッファをダンプしその内容を解析することで、どのような通信シーケンスが実行されたかを確認し、それを C 言語レベルのトランザクション処理の流れの対応を取ることでデバッグ作業を行っていく。今年度は、通信を検出しバッファする回路を付加した設計を実際に FPGA 上で実行し、デバッグ作業の実験を行った。通信の検出のための付加ハードウェアは、数千ゲートとバッファであり、通信の頻度を考えると、従来の各サイクルごとにバッファするものと比較すると非常にコンパクトになっている。現在、バッファされた情報から C 言語レベルのトランザクション設計との対応を自動的に取る手法の実装を進めている。当研究では、実際の設計現場の状況の把握が非常に重要であるため、マイクロプロセッサ／SoC 企業と共同研究に関し、情報交換を行っている。簡単な SoC で

の評価をこちらでまず行い、その結果に基づいて、具体的な共同研究作業の詳細を決めることになっている。

また、論理的な設計バグではなく、電気的なエラーを対象とする場合には、チップの動作は設計記述とは異なっているはずなので、その差をハードウェアサポートのもとで自動的に検出する技術について、東大 VDEC 客員研究員でもあるスタンフォード大学 Mitra 教授と共同研究を進めている。

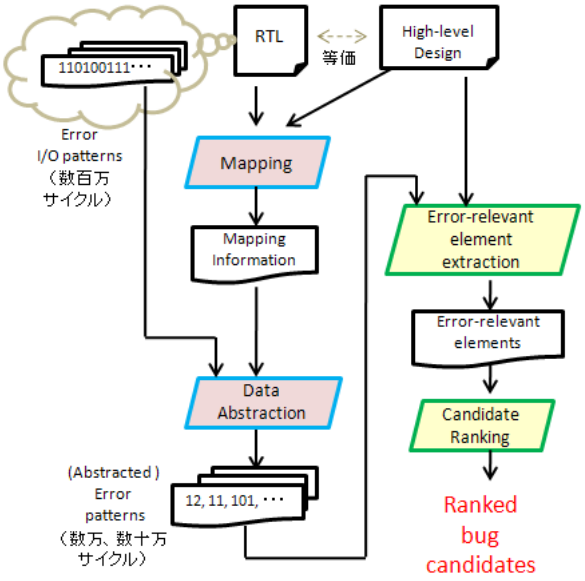

図 3.2.6 製造後デバッグ支援フレームワーク

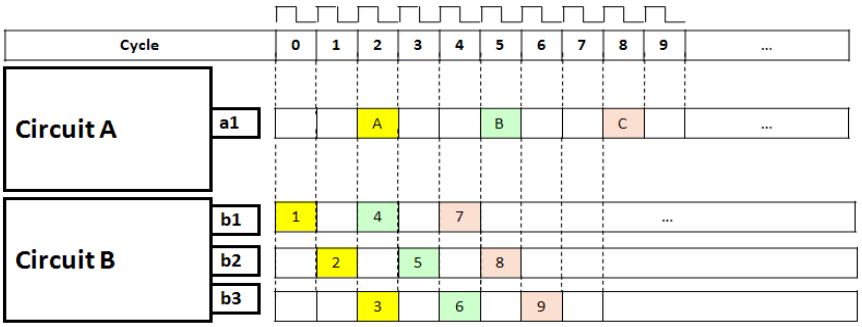

図 3.2.7 二つの設計記述 A と B のマッピング抽出の例

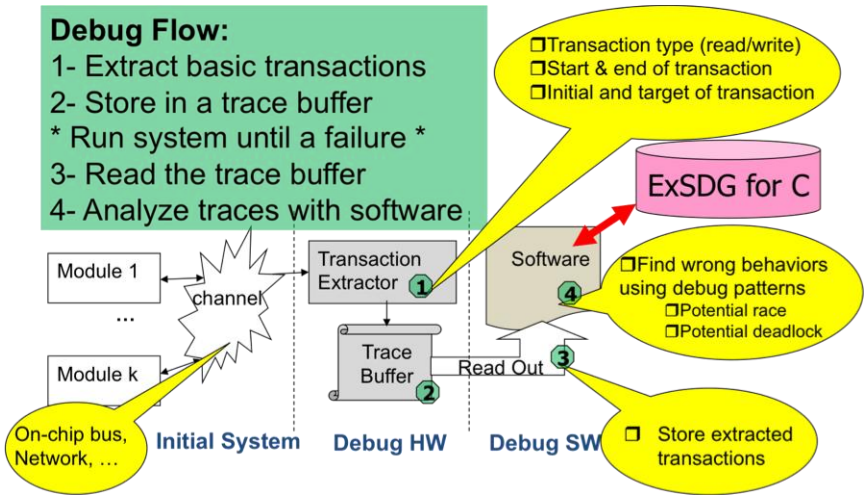

図 3.2.8 通信状況のハードウェアによる検出に基づく効率的なデバッグの流れ

- 部分的プログラム可能回路を用いた製造歩留まり向上手法

半導体技術の向上に伴い、製造歩留まりの低下が問題になってきている。回路レベルにおける製造歩留まり向上手法としては、DMRなどといった多重化に基づく手法やFPGAに代表される再構成可能素子を用いる手法などが知られている。しかしながら、一般的に製造故障は非常に局所的で1~2箇所である場合が多い。上記の手法はこれらの小規模の故障の修正に対して大規模な面積や性能のオーバーヘッドを要求している。

我々はこの問題を解決するため、部分的プログラム可能回路(Partially Programmable Circuit, PPC)を提案した。PPCは図3.2.9に示すように通常の論理ゲートのみからなる回路の一部をルックアップテーブル(LUT)に置き換えた回路である。PPC内のLUTにはいくつかの冗長な配線が接続されており、故障が無い場合にはこれらの配線は使用しない。もし、ある配線が故障した場合には、冗長配線を使用してLUTを再プログラミングすることでその故障配線を冗長化することで故障を修正する。ここで冗長化が可能な配線を耐故障配線と呼ぶ。PPC合成問題はなるべく少数のLUTによって回路全体の耐故障配線の割合を最大化するような回路を合成することを目的とする。

本年度は与えられた設計記述からPPCを合成する手法および与えられた配線故障を修正するLUTプログラムの生成手法の2手法の初期的な検討を行った。また、これらの手法をABCという論理合成ツール上に実装し初期実験を行った。初期実験では、単純な方法でLUTを挿入した場合においても回路全体の配線を最大50%耐故障にすることが可能であった。現在はLUT挿入方法を改良することで耐故障配線の割合をさらに向上する手法の研究を進めている。

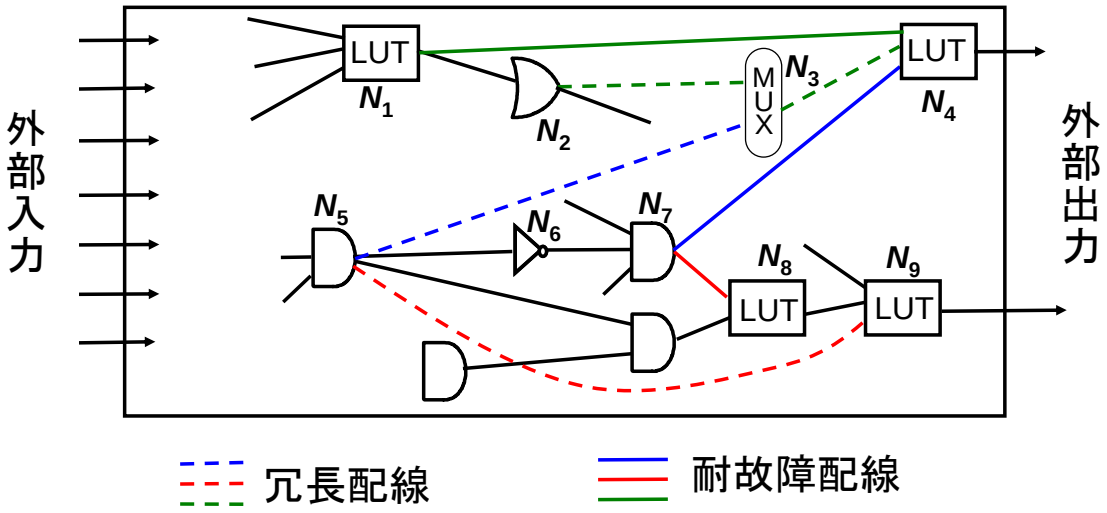

図 3.2.9 部分的プログラム可能回路(Partially Programmable Circuit, PPC)

以上、3.2で述べて各技術の実用化を図るため、米国のEDAツール企業3社と技術移管やツール利用に関する技術的な打合せを数度行った。特に形式的検証技術に関しては、今後の共同研究を行う上でのテーマの詳細も検討した。来年度、互いの人的交流も開始する予定になっている。

3.3 「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21年度は、第一に、ディペンダビリティのレベルを変更できる高機能ルータアーキテクチャの開発 [26]を行った。

SmartCore(Smart many-core system with redundant cores and multifunction routers)システムは、高機能ルータを用いたパケットレベルでの信頼性向上を提供する。SmartCoreシステムにおける高機能ルータアーキテクチャの概要を述べる。図3.3.1左の例では、Node(3, 2)とNode(4, 2)を用いて2重実行をおこない、DMR(Dual Modular Redundant)を実現する。高機能ルータにおいてNode(3, 2)とNode(4, 2) が送信するパケットを比較することでエラーを検出し、信頼性の向上を達成する。

開発した高機能ルータアーキテクチャ(図3.3.1右)について述べる。ここでは、アプリケーションを実行するために本来必要とされるノードをマスターノード、多重実行のために追加されたノードをミラーノードと呼ぶことにする。マスターノードとミラーノードで多重実行をおこない、信頼性の向上を達成するために、パケットの複製、パケットの送信先変更、パケットの比較という3つの機能を高機能ルータで実現する。

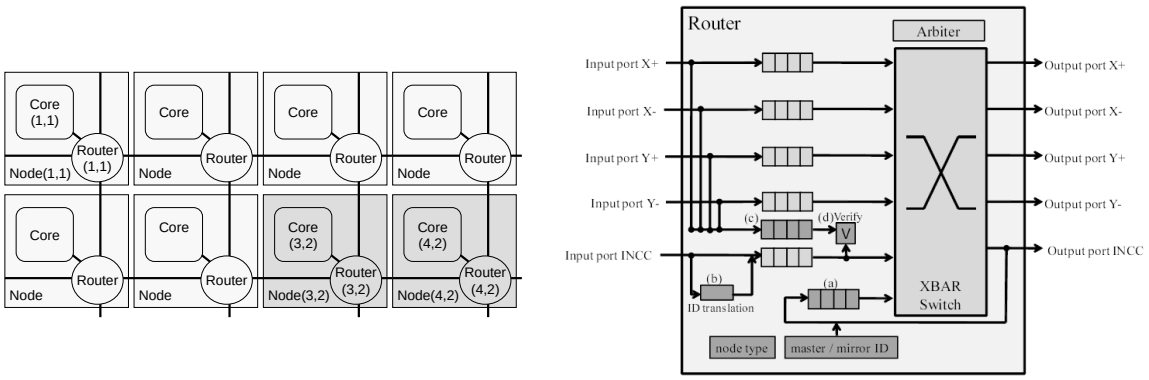

図3.3.1 SmartCoreシステムによるディペンダビリティ向上(左)。
右はルータの内部構造

図3.3.1右に示す高機能ルータには、自身がマスターノードかミラーノードかを示す情報を持つ。また、マスターノードであれば対応するミラーノードのID、ミラーノードであれば対応するマスターノードのIDの情報を持つ。これらの情報は、システムソフトウェアによってミラーノードの導入時に設定される。

パケットの複製のためにバッファ(図中(a))を追加する。自身がマスターノードである場合はINCC(Inter-Node Communication Controller, ノード間の通信コントローラ)にパケットを出力するとともに、送信先をミラーノードへ変更した複製パケットをこのバッファに格納する。

パケットの送信先を変更するために、ID変換テーブル(図中(b))を追加する。INCCから送信されるパケットの送信先情報を監視し、変換テーブルを参照することによりパケットの送信先を変更する。自身がミラーノードの場合、INCCから送信されるパケットの送信先はマスターノードに変更される。このテーブルの情報はシステムソフトウェアによって設定される。

パケットの比較のために、ミラーノードから送信されたパケットを格納するためのバッファ(図中(c))を追加する。ミラーノードにおいてパケットの送信先をマスターノードに変更する際に、ミラーノードからのパケットであることを示す識別子を付加する。マスターノードでは、その識別子によりミラーノードからのパケットを判別する。ミラーノードからのパケットの場合、追加したバッファにそのパケットを格納する。マスターノードのINCCから送信されるパケットは、入力バッファにおいてミラーノードからのパケットを待ち合わせる。マスターノードのパケットは、ミラーノードからのパケットがバッファに格納されると、比較器(図中(d))で順次パケットを比較してエラーを検出する。

提案した高機能ルータアーキテクチャを用いて多重実行をおこなう場合の性能低下をソフトウェアシミュレータにより評価した。NAS Parallel Benchmarkを用いたシミュレーションにより、パケットの待ち合わせによるアプリケーション性能の変化を評価し、SmartCoreシステムによる多重実行の性能低下は最大で4.1%と許容範囲であることを示した。

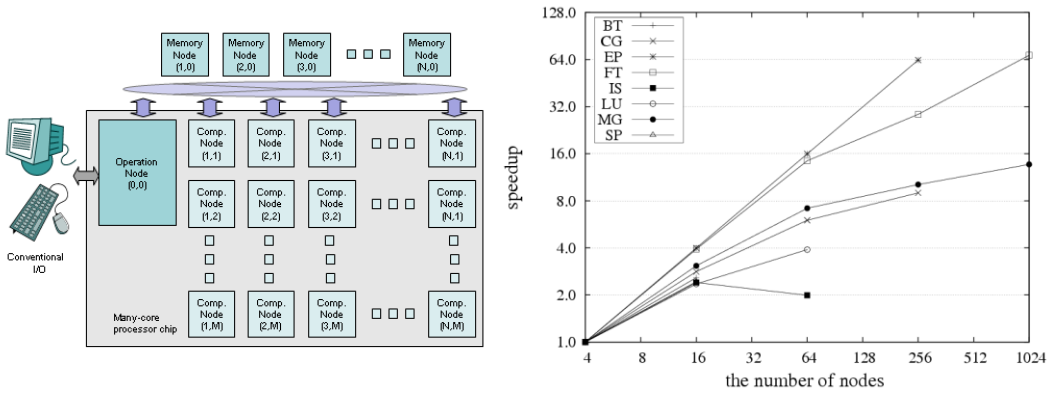

図3.3.2 M-Coreアーキテクチャ(左)とシミュレーション速度(右)

本研究における高機能ルータの評価は、クロックレベルのソフトウェアシミュレータ[27]を用いる評価と、FPGAシステムを用いる大規模な評価の2つの方式を適切に利用して進めていく。

第二に、ソフトウェアシミュレータの改良をおこなった[45]。我々は、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M-Coreを定義しており、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を構築している。本年度はSimMcの改良を行った。主な改良点は、メインメモリを持つノードの実装および8ビットルータ・モデルの追加である。図3.3.2左にM-Coreの構成を示す。マルチコアプロセッサシミュレータSimMcを用いて、M-Coreアーキテクチャの性能を評価した。NPB(NAS Parallel Benchmark) を用いて、コア数を増加させて測定した速度向上の結果を図3.3.2右にまとめる。ノード数が1024と非常に大規模なメニーコアプロセッサの評価が可能となっていることがわかる。

第三に、本研究における高機能ルータの詳細評価のためのFPGAシステムの整備[31, 37, 40, 52, 55]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ScalableCoreシステムを提案している。本年度はこのシステムの改良をおこなった。

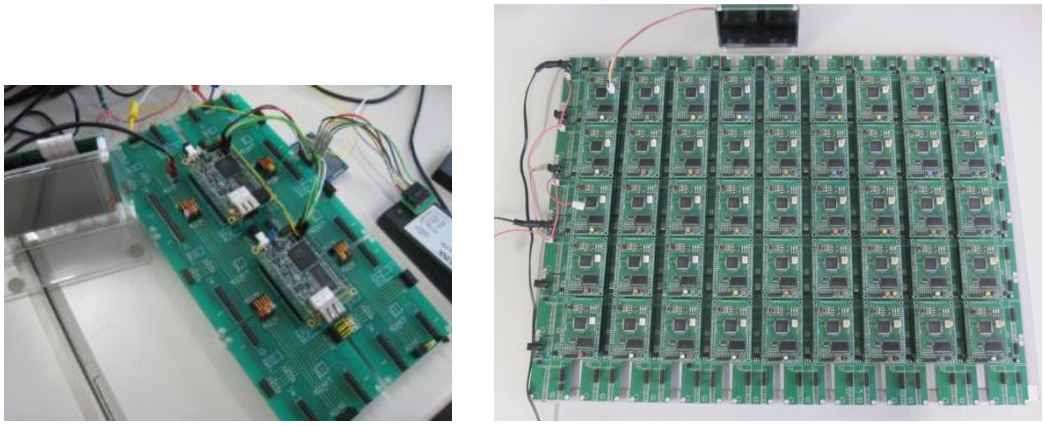

図 3.3.3 前年度設計した ScalableCore ボード 6 枚と FPGA ボード2枚によるシステム(左)と今年度設計した ScalableCore ボード 60 枚と FPGA ボード 45 枚によるシステム(右)

前年度に設計した ScalableCore ボードを 6 枚と FPGA ボード2枚によるシステムの様子を図 3.3.3 の左に示す。FPGA ボードに搭載される FPGA にルータを実装し、FPGA ボード間でパケットの送受信ができることを確認していた。

今年度はハードウェアの全面的な見直しをおこない、安定かつ高速に動作するシステムを構築した。今年度に設計した FPGA ボード 16 枚により構成されるシステムを用いて評価したところ、一般的な計算機を用いたソフトウェアシミュレータと比較して、メニーコアプロセッサのサイクルレベルの評価を 2～3 倍程度高速化できることを確認した。また、今年度に設計した ScalableCore ボード 60 枚と FPGA ボード 45 枚によるシステム(図 3.3.3 右)を構築し、安定して動作することを確認した。

§4. 成果発表等

(4－1) 原著論文発表

- 論文詳細情報

1. O. Sarbishei, M. Tabandeh, B. Alizadeh, and M. Fujita, “A Formal Approach for Debugging Arithmetic Circuits,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol.28, No.5, pp.742-754, May 2009, doi:10.1109/TCAD.2009.2013998.
2. T. Nishihara, T. Matsumoto, and M. Fujita, “Word-Level Equivalence Checking in Bit-Level Accuracy with Identical Datapath,” IEICE Trans. on Information and Systems, Vol.E92-D, No.5, pp.972-984, May 2009, doi:10.1587/transinf.E92.D.972.
3. B. Alizadeh and M. Fujita, “A Unified Framework for Equivalence Verification of Datapath Oriented Applications,” IEICE Trans. on Information and Systems, Vol.E92-D, No.5,

pp.985-994, May 2009, doi:10.1587/transinf.E92.D.985.
4. S. Gao, H. Yoshida, K. Seto, S. Komatsu, and M. Fujita, “Interconnect-aware pipeline synthesis for array based architectures,” IEICE Trans. on Fundamentals, Vol.E92-A, No.6, pp.1464-1475, June 2009, doi:10.1587/transfun.E92.A.1464.
5. O. Sarbishei, M. Tabandeh, B. Alizadeh, and M. Fujita, “High-level Optimization of Integer Multipliers over a Finite Bit-Width with Verification Capabilities,” Proc. of 7th International Conference on Formal Methods and Models for Codesign, pp.56-65, July 2009, doi:10.1109/MEMCOD.2009.5185378.
6. O. Sarbishei, B. Alizadeh, and M. Fujita, “Polynomial Datapath Optimization using Partitioning and Compensation Heuristics,” Proc. of the 46th Annual Design Automation Conference, pp.931-936, July 2009, doi:10.1145/1629911.1630151.
7. A.M. Gharehbaghi and M. Fujita, “Transaction-Based Debugging of System-on-Chips with Patterns,” Proc. of 27th IEEE International Conference on Computer Design, pp.186-192, Oct. 2009, doi:10.1109/ICCD.2009.5413157.
8. A.M. Gharehbaghi and M. Fujita, “On-Chip Transaction Level Debug Support for System-on-Chips,” Proc. of International SoC Design Conference, pp.124-127, Nov. 2009, doi:10.1109/SOCDC.2009.5423891.
9. B. Alizadeh and M. Fujita, “Improved Heuristics for Finite Word-Length Polynomial Datapath Optimization,” Proc. of International Conference on Computer-Aided Design, pp.169-174, Nov. 2009, doi:10.1145/1687399.1687536.
10. Y. Lee, T. Nishihara, T. Matsumoto, and M. Fujita, “A Post-Silicon Debug Support Using High-level Design Description,” Proc. of the 18th Asian Test Symposium, pp.141-147, Nov. 2009, doi:10.1109/ATS.2009.28.
11. H. Yoshida, S. Morishita, and M. Fujita, “Demonstration of Hardware Accelerated Formal Verification,” Proc. of International Conference on Field-Programmable Technology, pp.380-383, Dec. 2009, doi:10.1109/FPT.2009.5377610.
12. B. Alizadeh and M. Fujita, “Optimization of Modular Multiplication on FPGA using Don’t Care Conditions,” Proc. of International Conference on Field-Programmable Technology, pp.399-402, Dec. 2009, doi:10.1109/FPT.2009.5377687.
13. B. Alizadeh and M. Fujita, “Guided Gate-level ATPG for Sequential Circuits using a High-level Test Generation Approach,” Proc. of 15th Asia and South Pacific Design Automation Conference, pp.425-430, Jan. 2010, doi:10.1109/ASPDAC.2010.5419843.
14. T. Matsumoto, T. Nishihara, and M. Fujita, “Performance Estimation with Automatic False-Path Detection for System-Level Designs,” IPSJ Transactions on System LSI Design Methodology, Vol.3, pp.69-80, Feb. 2010, doi:10.2197/ipsjtsldm.3.69.
15. A.M. Gharehbaghi, B. Alizadeh, and M. Fujita, “Aggressive Overclocking Support using a

Novel Timing Error Recovery Technique on FPGAs," Proc. of Eighteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, page 288, Feb. 2010, doi:10.1145/1723112.1723178.

16. B. Alizadeh, A. M. Gharehbaghi, and M. Fujita, "Pipelined Microprocessors Optimization and Debugging," Proc. of 6th International Symposium on Applied Reconfigurable Computing, pp.435-444, March 2010, doi:10.1007/978-3-642-12133-3.
17. Masahiro Fujita, Hideo Tanida, Fei Gao, Tasuku Nishihara, Takeshi Matsumoto, "Synthesis and Formal Verification of On-Chip Protocol Transducers through Decomposed Specification," 11th International Symposium on Quality Electronic Design, pp515-524, March 2010.

(4－2) 知財出願

CREST 研究期間累積件数(国内 2 件)