「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

H22 年度
実績報告

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。

４年度目である平成 22 年度は、形式的検証・テストの最適化研究とツール群のさらなる開発・評価および産業界への展開、回路要素技術・プロセッサアーキテクチャ技術のさらなる開発・評価・最適化、メニーコア用ディペンダブル高機能ルータの方式改良などを行い、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】

- 過渡故障・永久故障耐性を持つ FPGA アーキテクチャ
  永久故障耐性は Triple Modular Redundancy（TMR）により実現される。本年度は、ロジックブロック内部に予備のロジックエレメントをいくつ持つと、面積当たりの信頼性が最も向上するか、モンテカルロシミュレーションによって評価した。その結果、ロジックブロック内にスペアのロジックエレメントを置いても、それに見合うだけの信頼性は得られないことが分かった。すなわち、ロジックブロック内のロジックエレメント数は 3（もしくは 3 の倍数）として、そのうちの 1 個が故障した場合にはロジックブロックごと置き換える

方が、バランスがよいとの知見を得た。

- タイミング故障耐性を持つクロッキング方式
  本年度は､最も単純な例として､リプルキャリーアダーを用いた64-bitカウンタに対して、提案のクロッキング方式を適用し、評価した。FPGAのシミュレータを用いて評価した結果、ほぼ2倍のクロック速度を達成できることを確認した。リプルキャリーアダーは、クリティカルパスとショートパスの差が大きく、最小遅延制約を満たすために多くの遅延を挿入する必要がある。その結果、配置配線ツールが最適な結果を実現することができず、最大遅延が増加した。今後は、実チップ上で動作を確認するとともに、最小遅延制約を満たすための遅延の数を最小化する方法について検討する。
- タイミング故障耐性を持つスーパスカラプロセッサ構成方式
  本年度は、タイミング故障体制を持つスーパスカラプロセッサを構成するための定式化を行うとともに、タイミング故障を検出するデータキャッシュの試作のための設計を行った。来年度は、得られた設計データに基づき、TSMC 40nm プロセスにおいてチップ試作を行う。

【形式的検証】

- 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、(1) NEC グループと共同で NEC 社の動作合成ツール CyberWorkBench による動作合成前後での等価性検証評価実験の準備、(2) ループ部分の等価性検証を効率的に行う手法の評価、(3) SystemC フロントエンドの開発、(4) 検証ツールを実用化する際の問題点・ニーズの市場調査、を行った。CyberWorkBench の例題を FLEC で検証するために設計記述の変換方法の検討を行い、FLEC の内部表現である ExSDG を CyberWorkBench 例題から構築することができた。ループ間の等価性検証では、実際のループ最適化に関する複数の例題に対して手法を適用し、短時間に等価性が検証できることを示した。SystemC フロントエンド開発では、完成したフロントエンドを用いて、SystemC 例題に対して等価性検証を始めとする FLEC の機能が適用できることを確認した。関連して、Oxford 大学と SystemC 設計記述の検証・合成のベンチマークの整備も引き続き進めている。また、実用化に向けた市場調査を通して、等価性指定の簡易化・タイミングを考慮した階層的な等価性検証・アルゴリズム記述と動作合成用記述の間の等価性検証などが産業界で強いニーズがあることが分かり、今後、FLEC ツールにおける実現を検討する。
- 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に関しては、引き続き詳細について検討している。英国側の研究準備が整った段階で開始する予定である。

- 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に関しては、Bremen 大学 Drechsler 教授のグループと共同で、SMT（Satisfiable Modulo Theory）ソルバーを利用することで、従来の最適化をより強力に行う技術を新規に開発し、従来比さらに 10-20%の高速化、面積縮小化を実現した。我々の技術を利用しない場合（従来手法）と比較すると遅延/面積ともに 1/2 以下に縮小化できている。現在、FLEC（等価性検証環境）との統合を検討している。
- 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。特に、形式的な解析手法を利用し、設計記述中の状態遷移を網羅的にカバーするようなシミュレーションパターンの自動生成を行う手法を Verona 大学のグループと共同で新規に開発した。ターゲットとなる（カバレッジをあげたい）状態あるいは状態遷移に初期状態から到達するシミュレーションパターンの生成に関し、記号シミュレーション結果を利用して生成する手法を各種ベンチマークに適用したところ、従来手法では一定以上カバレッジがあげられなかった例題すべてについて、100%のカバレッジを達成することができる。このシミュレーションベースの検証手法についても、FLEC（等価性検証環境）との統合を検討している。
- FPGAなどを利用して、検証／シミュレーション／解析／最適化の10倍以上の高速化を行う手法およびその実現システムについて検討している。設計対象としては、C言語ベース記述、RTL／ゲートレベル記述、アナログ・ミッスクシグナル設計の全て（そしてそれらを拡張した各種論理式・関数の解析を網羅的な解析）を統一的に高速にシミュレーション（あるいはエミュレーション）する環境の構築、またそれを利用したシミュレーション結果の解析によるアサーション・プロパティの自動抽出とそれらの自動検証を行う技術とその環境の構築を進めている。FLECなどを用いることで、一定規模の回路ブロック（あるいはIP）の検証は可能となるが、将来のSoCなど大規模チップでは、それらのIPが多数利用されるため、それらの間の通信やインターフェイスの検証が大きな課題となる。本手法はその検証を統一的に行なおうとするものである。
- 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。昨年度までに、アーキテクチャの検証を行うための基礎的な手法の開発を終えており、今年度は、その手法を実際にいくつかのエラーリカバリ方式に対して適用した。例えば、エラーが発生した際にリオーダバッファに残っている命令をやり直すリカバリ手法、エラーが発生した際にリセットを行うリカバリ手法などをモデル化し検証を行った。検証の結果、これらのエラーリカバリ方式が、元のプロセッサと同じ結果を出すことが証明できた。また、この検証を用いて、対象のプロセッサアーキテクチャがどの程度のエラーであればリカバリ可能であるかどうかの評価も行う手法の研究を今年度後半から開始した。これにより、プロセッサアーキテクチャのリカバリ性能を保証することができる。
- 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、エラーを再現するために最小限の入力パタンを求めるダイナミックスライシング回路と呼ばれるオンチップ機構を提案し、

例題で評価を行った。また、ポストシリコンで製造修正を可能とする技術として、プログラマブル制御回路を用いた回路とLUTを挿入して部分的にプログラム可能にした回路を提案し、例題を用いることで各方式の有効性を実証した。今後、FLEC（等価性検証環境）との統合を検討していく。

【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に、TMR(Triple Modular Redundancy)の機能を提供する方式を開発した。また、開発中の FPGA システムに高機能ルータアーキテクチャを部分的に実装し、正しく動作することを確認した。
- ソフトウェアシミュレータの機能拡張
  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 の改良および機能拡張をおこなった。主な改良点はキャッシュ階層の追加によるメモリシステムの詳細化である。
- FPGA システムの改良および整備
  アーキテクチャ評価のための FPGA システムの改良と整備をおこなった。前年度構築した ScalableCore システムを改良し、100 枚の FPGA カードを接続する大規模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評価結果を得た。
- タスク配置手法の検討
  改良を施したソフトウェアシミュレータを用いて検討しているタスク配置手法を評価し、平均 4.5%の性能向上を達成することを明らかにした。

【検証技術実証】

- 本年度より CREST での研究活動を開始した。形式的検証グループの研究成果の産業界での実用化を目指し、NEC の高位設計環境である CyberWorkBench(CWB)のデータを利用するなどして、設計現場の観点から評価・実証を行なっている。当初のターゲットとして等価性検証ツール(FLEC)の実証から活動を開始した。本年度は主に、CWB を用いた実回路設計に FLEC を組み込んだ利用シナリオの検討、接続インターフェイスの検討、実証実験、の 3 つの項目について活動を行なった。
- CWB を用いた実回路設計に FLEC を組み込んだ利用シナリオの検討では、FLEC の導入効果が高いと考えられるシナリオを検討した。その結果、検証済の動作レベル IP から異なる製品向けの RTL を合成する場合と、検証済の動作レベル IP を動作レベルでチューニングする場合を想定することとした。
- 接続インターフェイスの検討では、CWB の入出力データを FLEC に入力可能となるようデータ形式の変換方法について検討した。CWB への入出力データを FLEC が扱う SpecC 言語に変換する方針で検討を行い、変換方法を明確にするとともにインターフェイスの試作も行った。

- 実証実験では、CWB と共に提供されている動作レベル IP 群から、暗号回路、浮動小数点演算回路を選択して実証を行なった。CWB の入出力データを接続インターフェイスの検討でまとめた変換方針の通り SpecC 記述へ変換した。この SpecC 記述は、FLEC の内部データ構造である AST または SDG まで変換できた。

§2. 研究実施体制

(1)東京大学　坂井グループ

①研究分担グループ長:坂井　修一 （東京大学大学院情報理工学系研究科、教授)(研究代表者)

②研究項目

・永久故障耐性を持つ FPGA 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

・タイミング故障耐性を持つプロセッサアーキテクチャ

(2)東京大学　藤田グループ

①研究分担グループ長:藤田 昌宏 （東京大学大規模集積システム設計教育研究センター、教授)(主たる共同研究者)

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および評価

・ポストシリコンデバッグ技術

・プログラマブル制御回路を用いたポストシリコン設計修正技術

・部分的プログラム可能回路を用いた耐故障性向上技術

・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

・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

・形式的解析手法を応用した検証カバレッジ向上手法

・SoC・組込みシステム全体に対するシミュレーション（エミュレーション）ベース検証における検証手法

(3)東京工業大学　吉瀬グループ

①研究分担グループ長:吉瀬　謙二 （東京工業大学大学院情報理工学研究科、准教授)(主たる共同研究者)

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

(4)NEC　若林グループ

①研究分担グループ長:若林　一敏(日本電気株式会社組込みシステムソリューション事業部システム IP コア研究所、主管研究員)(主たる共同研究者)

②研究項目

・形式的等価性検証技術の実用化に向けた実証実験とその評価

§3. 研究実施内容

(文中に番号がある場合は(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。

第一に、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。

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 3.1.1、3.1.2)。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が、科学的探求や資源開発などの点で、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、コスト的に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ようになる。

平成 20 年度から提案を検証するためのテストベッド（図 3.1.3）の FPGA 回路設計を行ってきた。具体的には、図 3.1.1 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。

平成 22 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。

また、本年度は、ロジックブロック内部に予備のロジックエレメントをいくつ持つと、面積当たりの信頼性が最も向上するか、モンテカルロシミュレーションによって評価した。

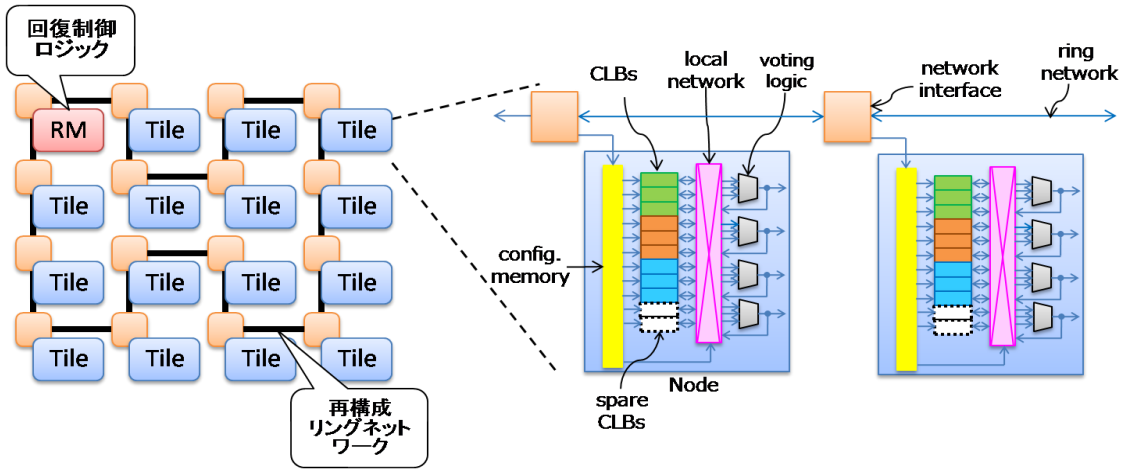

図 3.1.1 耐永久故障アーキテクチャ

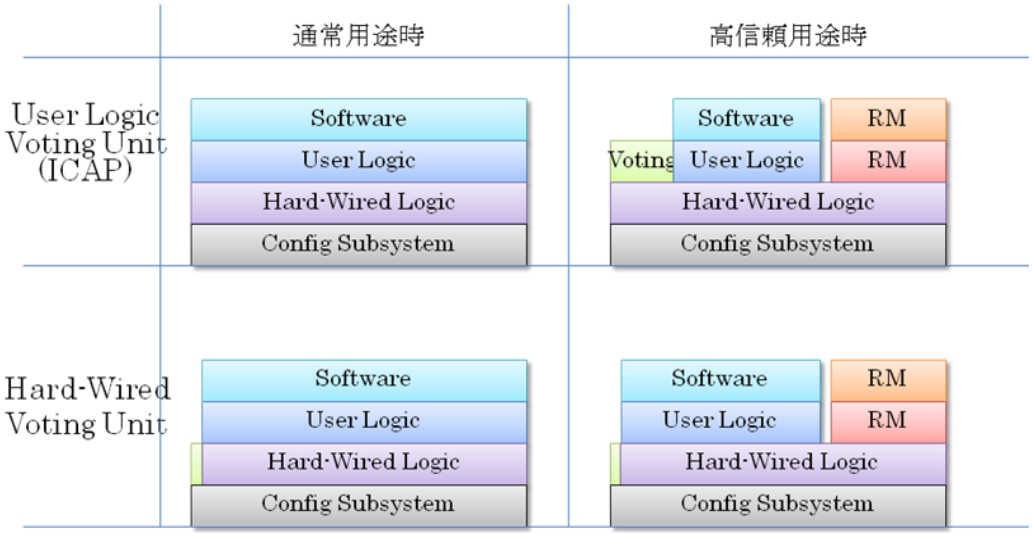

図 3.1.2 通常用途時と高信頼用途時の使い分け

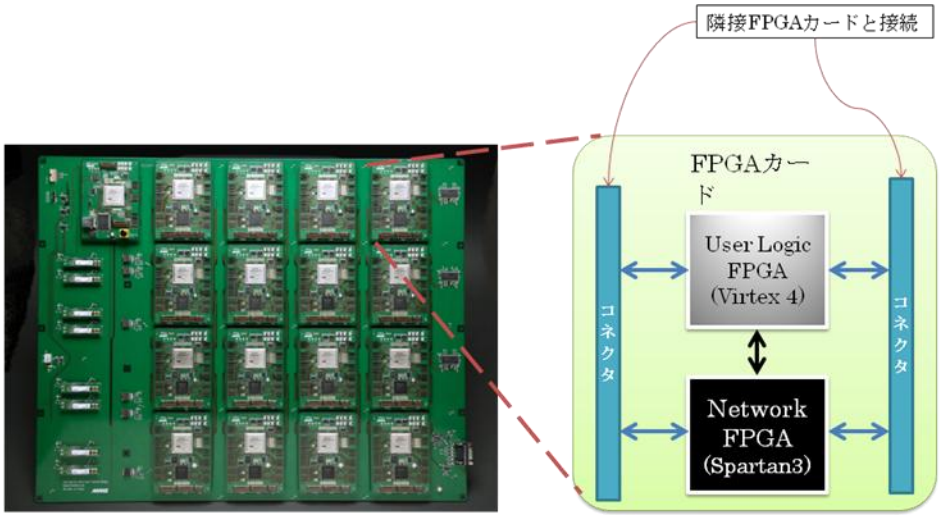

図 3.1.3　耐永久故障 FPGA 用テストベッドボード

図 3.1.4 に、その結果を示す。横軸は三重化しない場合の面積を 1 とした場合の相対面積を、縦軸は同じく三重化しない場合を 1 とした相対寿命を示す。N は、ロジックブロック内のロジックエレメント数である。たとえば N=4 とした場合には、4 個のロジックエレメントのうち 3 個を用いて 3 つ組を構成し、そのうちの 1 個が故障した場合には、まず残りの 3 個で 3 つ組を構成することを試みる。更にもう 1 個が故障した場合には、このロジックブロックを捨て、スペアのロジックブロック上に 3 つ組を構成する。

このグラフから、ロジックブロック内にスペアのロジックエレメントを置いても、それに見合うだけの信頼性の向上は得られないことが分かる。すなわち、ロジックブロック内のロ

ジックエレメント数は 3（もしくは 3 の倍数）として、そのうちの 1 個が故障した場合にはロジックブロックごと置き換える方が、バランスがよい。

来年度以降、この結果に基づいて基本設計の修正を行う必要がある。

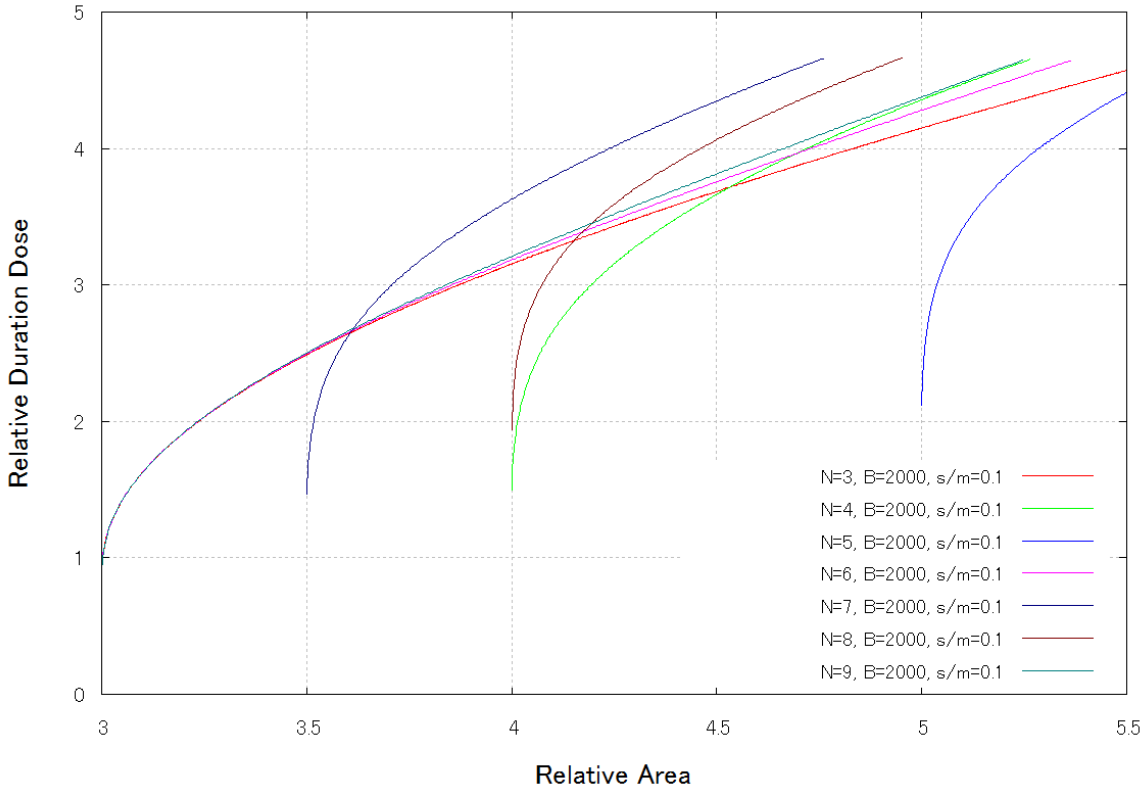

図 3.1.4　面積と信頼性

第二に、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。このクロッキング方式は、タイミング故障体制を持つ上に、クロック速度を 2 倍に高めることができる。

平成 20 年度から、図 3.1.5 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、実現方式の検討を進めてきた。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 年度からは、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、提案手法を改良してきた。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。

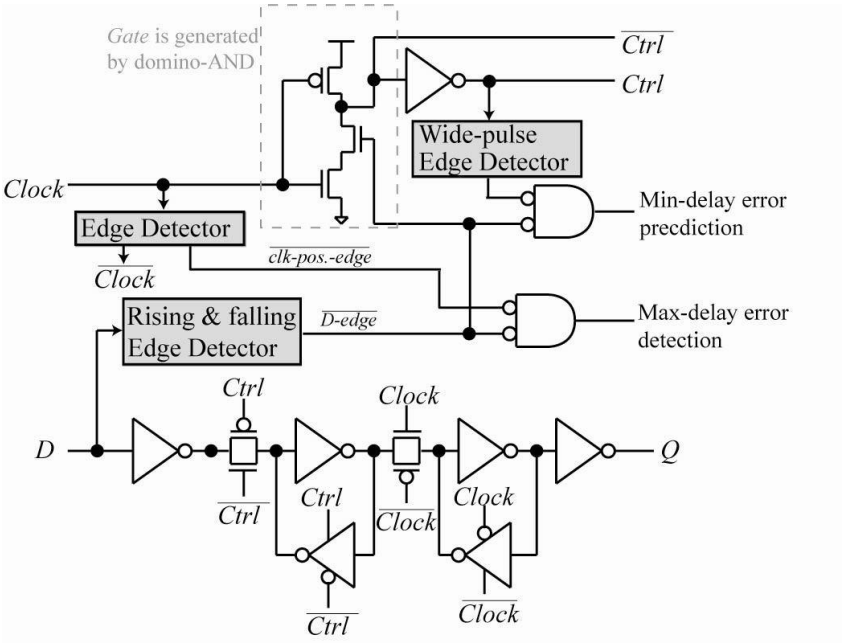

図 3.1.5 タイミング故障を動的に検知・修正する FF

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 3.1.6 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 3.1.6 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 3.1.7 に、提案するクロッキング方式の回路図を示す。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 3.1.8 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。

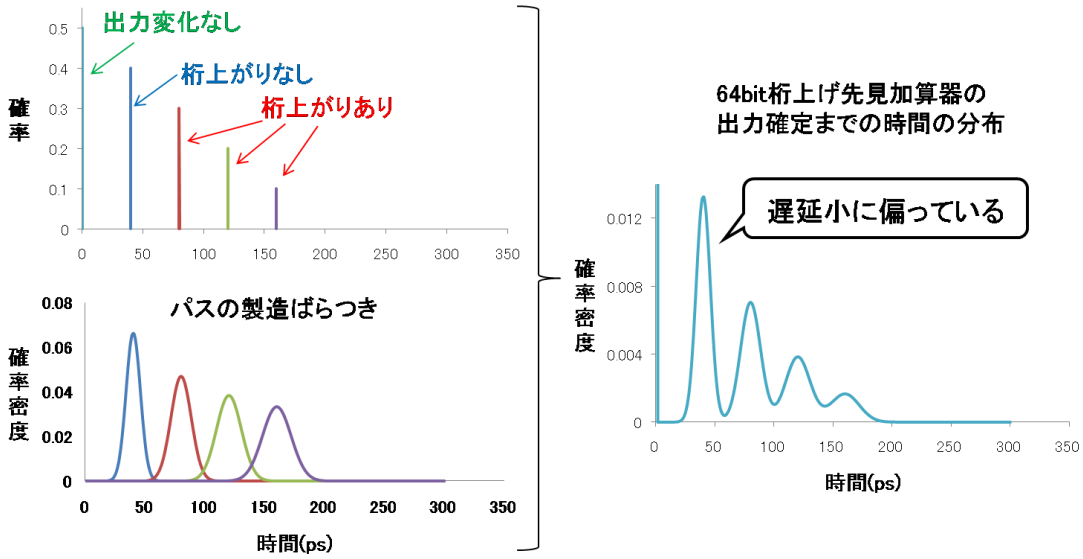

図 3.1.6 64bit 加算器の出力確定までの時間の分布

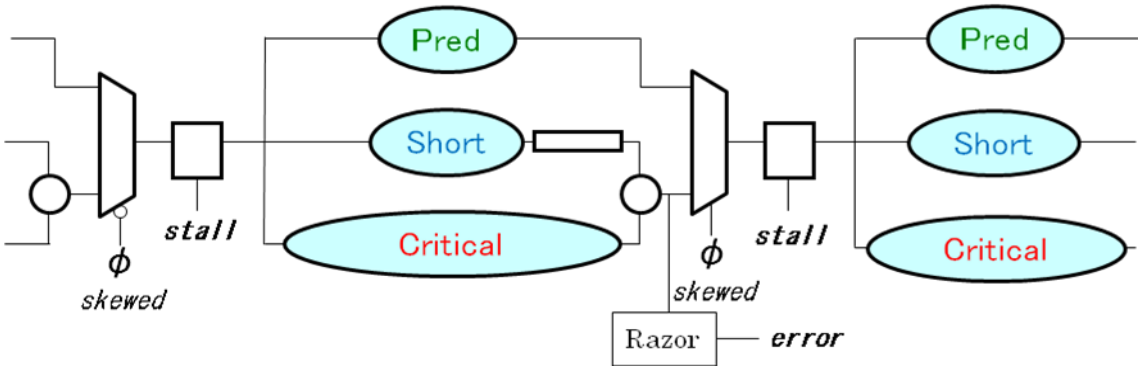

図 3.1.7　タイミング故障耐性を持つクロッキング方式

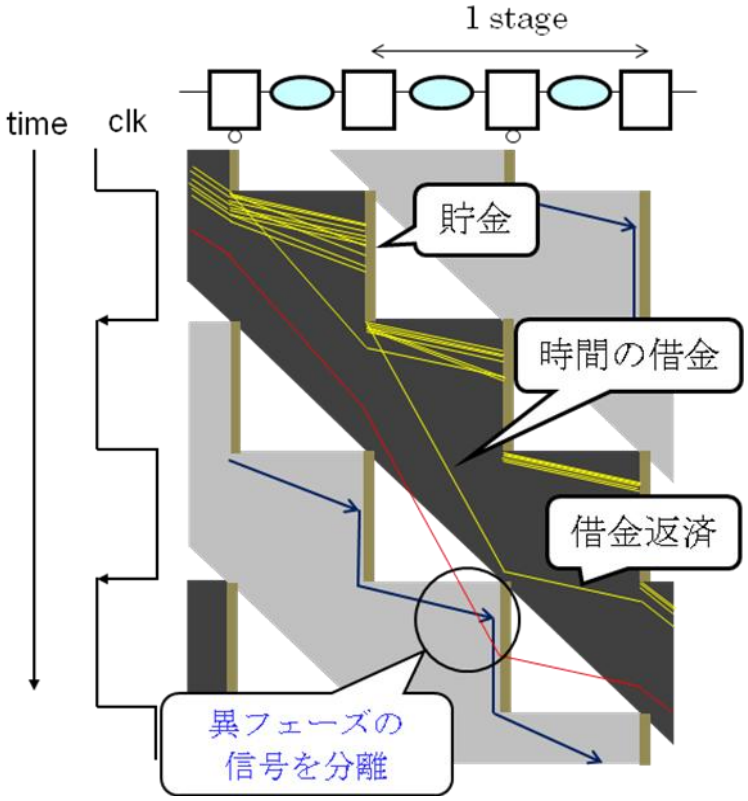

図 3.1.8　タイム･ボローイングの様子

第三に、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 3.1.9）。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 3.1.10）。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パイプラインに初期化をかける。

この機構を、前年度に引き続き、テストベッド（図 3.1.11 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 年度から本年度には、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行っている。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 23 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。

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 Razor II 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 In-Order 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い。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。

本年度は更に、タイミング故障を検出するデータキャッシュの試作のための設計を行った。このデータキャッシュは、①本研究室で以前から提案していた SRAM の読み出し時のタイミング・フォールトを検出する機構、②書き込みの途中で書き込みを中止する機構、③ワードラインデコーダにおけるタイミング・フォールトを検出する機構などを持つ。来年度は、得られた設計データに基づき、TSMC 40nm プロセスにおいてチップ試作を行い、実チップにおいてこれらの機構が正しく動作することを確認する。

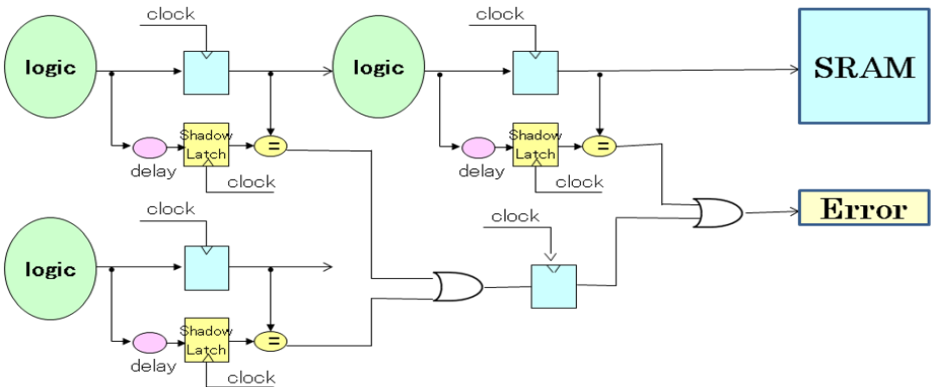

図 3.1.9　タイミング故障の検出と通知ネットワーク

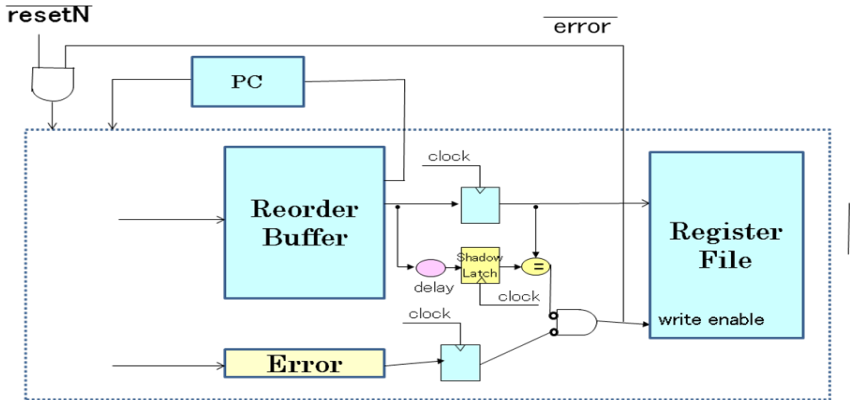

図 3.1.10　レジスタファイルとプログラムカウンタの保護

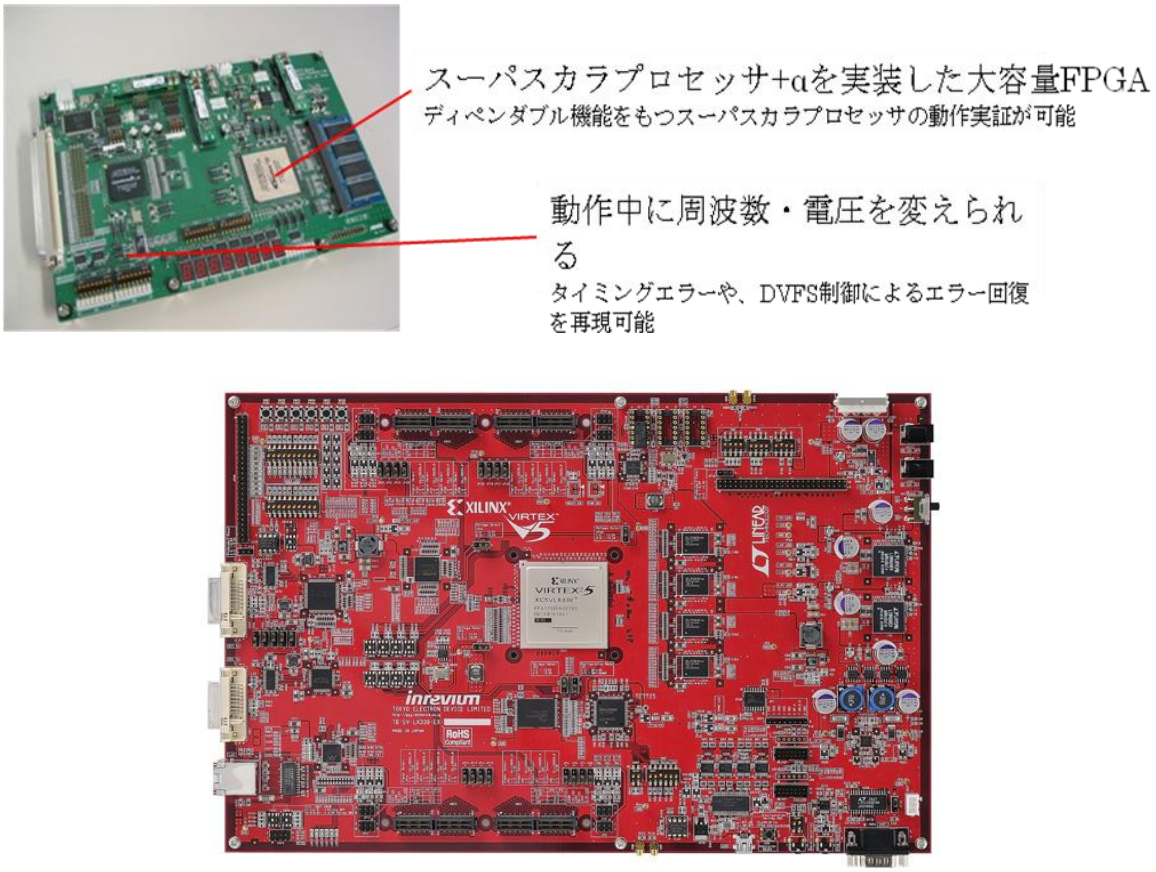

図 3.1.11 テストベッド（上：従来品、下：新規開発品のベースとなる評価ボード）

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および評価

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、(1) NEC グループと共同で NEC 社の動作合成ツール CyberWorkBench による動作合成前後での等価性検証評価実験の準備、(2) ループ部分の等価性検証を効率的に行う手法の評価、(3) SystemC フロントエンドの開発、(4) 検証ツールを実用化する際の問題点・ニーズの市場調査、を行った。

NEC グループとの協調の結果として、設計および検証の全体フロー（全体像）は図 3.2.1 に示す形になると考えている。まず、シミュレーションや性能評価目的に作成された C ベース設計に対して、高位合成ツールが効果的に機能するような記述の書き換えや変換が適用される（図 3.2.1 の C ベース設計記述 1 から n)。これらの間の機能的な等価性検証がまず適用される。その後、高位合成ツールによる C ベース設計から RTL 設計が自動生成される。この高位合成の前後の等価性検証は、高位合成ツール自体が非常に複雑なプログラムであるためバグがある

可能性があること、また、合成結果を人手で最適化する場合もあることなどから、非常に重要である。また、高位合成時にカスタマイズされた算術演算回路を利用することでより高性能なハードウェアを生成できるようになるため、従来行ってきた算術演算回路の合成／検証手法も高位合成時に利用可能にする。さらに、高位合成時あるいは、その後で、従来から研究を行っているポストシリコンデバッグ技術が適用される。

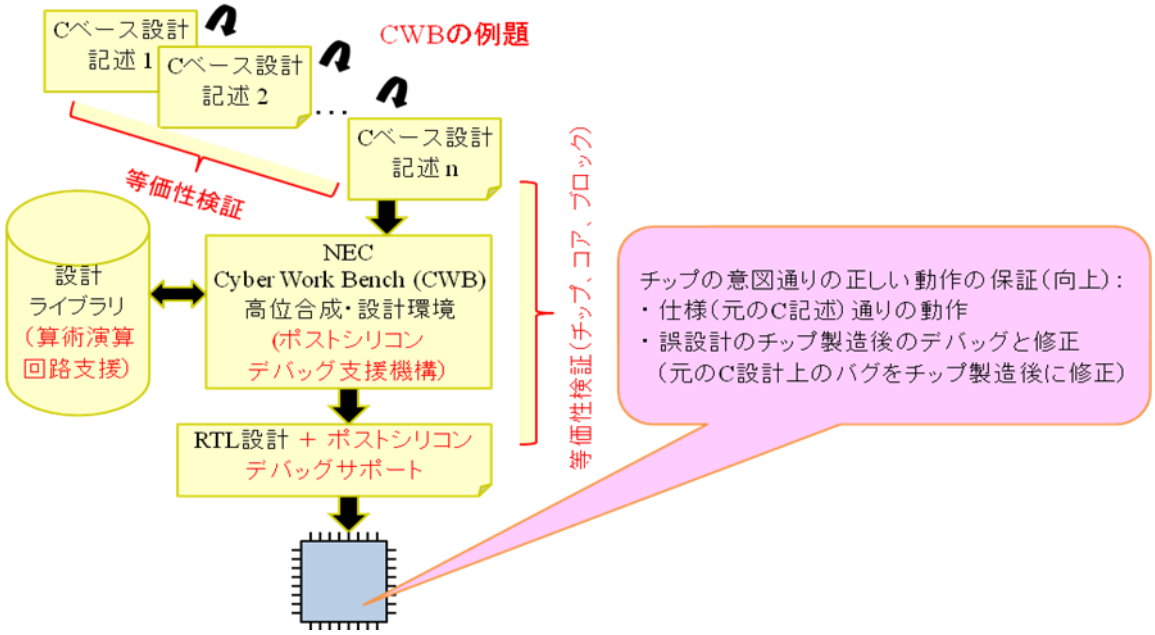

図 3.2.1　NEC との協調を踏まえた、設計および検証の全体フローの概要

等価性検証ツールの内部構成図を図 3.2.2 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。

今年度第 4 四半期より、NEC グループと共同で、動作合成ツール CyberWorkBench による動作合成前後での等価性検証評価を開始している。CyberWorkBench と藤田研究グループで開発している等価性検証ツール FLEC では入力可能な言語が異なるため、その間の変換方法の検討を行った。当面、DES 暗号化処理、浮動点小数の加算、乗算の 3 つの例題を対象に FLEC で一通りの等価性検証ができることを目指している。入力言語の変換方法が確立できた時点で、変換処理のプログラム実装を行うことも検討している。CyberWorkBench に入力される設計記述を扱えるようになることにより、より多くの例題に対して評価が可能となり、FLEC ツールの問題点洗い出しなどが効率的にできると期待される。

ループ部分の等価性を効率的に検証する手法については、昨年度の研究を通して得られた手法を、今年度は、実際のループ最適化に対して評価を行った。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。実際のループ最適化例題に対して、短時間（数秒以内）で等価性を検証できることが示された。

等価性検証ツール FLEC の SystemC 言語フロントエンドについては、昨年度後半に策定した仕様に従って、ツール開発を行った。本フロントエンドでは、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。外部企業と共同で開発を行い、主に藤田研究グループで仕様策定、テストケース整備を行い、実際のプログラミングは外部企業が担当した。この SystemC フロントエンドを利用して、ある範囲の SystemC 設計記述に対して、FLEC が持っている等価性検証を初めとする機能が適用できることを実際に確認した。

実用化へ向けた取り組みとして、今年度後半に等価性検証ツールの市場調査を実施した。この調査では、現在の FLEC ツールの持つ様々な機能の中で、どのような機能が産業界でニーズがあるか、どの程度の機能的追加や利便性向上があれば実用化できるか、を中心に聞き取り調査を行った。その結果、ほぼ全ての企業が市販の等価性検証ツールの性能に満足しておらず、より大規模な設計記述が扱える等価性検証ツールを希望していることが分かった。そのためには、タイミングを考慮して等価性検証問題を分割する機能の追加が必要と考えられる。また、現在のツールでは等価性指定が煩雑になるため、与えられた設計記述から成立つと思われる等価性を推定する手法や動作合成ツールから内部等価点情報を得ることが必要との意見もあった。これらのフィードバックを考慮して、今後の機能追加を検討しているところである。中でも、タイミングを考慮した等価性検証の分割の実現は、より大規模な設計記述を検証する際には不可欠な機能であるため、優先的に開発を進めていく予定である。また、市場調査時に行ったアンケートや、訪問先企業とのディスカッションでは、等価性検証のみではなく、高位設計の各種チェック機能（一般に Static 検証とも呼ばれ、軽い動作の形式的な検証を指す）や、ポストシリコンデバッグ技術との融合により、ツールの魅力が大幅に増すとの指摘が、ほぼ全社から聞かれた。また、高位検討とアナログ検証の融合に関する要望も強かった。これらの技術は、等価性検証技術と並行して研究を進めてきたものであり、是非ツール化することを検討していきたい。

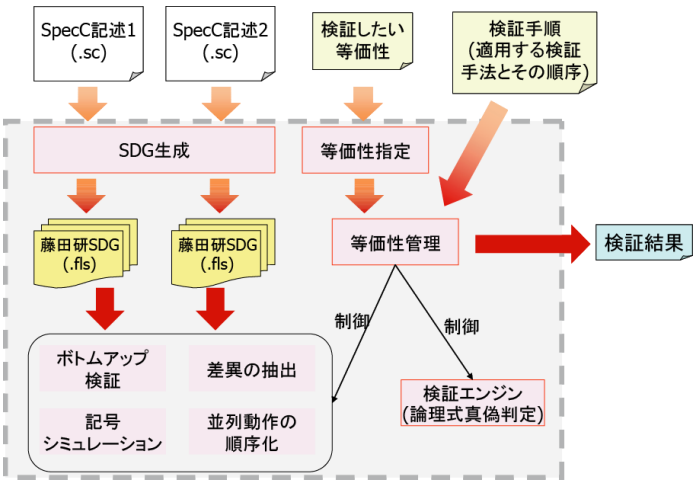

図 3.2.2 等価性検証ツールの構成

- ポストシリコンデバッグ技術

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。

Post-silicon においてもシミュレーションを行うことは可能ではあるが、エラーを再現するためのサイクル数が非常に多く、またそのために多くの入力パタンを記憶しておく必要があり、現実的ではない。そこで、エラーを再現するための最小限の入力パタンを求めるオンチップ機構であるダイナミックスライシング回路(Dynamic Slicing Circuit, DSC)を提案した。DSC は図 3.2.3 に示すような回路であり、信号間の依存関係を示す d-tag(Dependency Tag)と呼ばれる信号を用いることでエラーに影響する入力値を求めることが可能である。いくつかの例題を用いた実験では、提案回路は平均 4%の面積オーバヘッドで入力パタン数を 10 分の 1 に削減することが可能であることを確認した。[11]

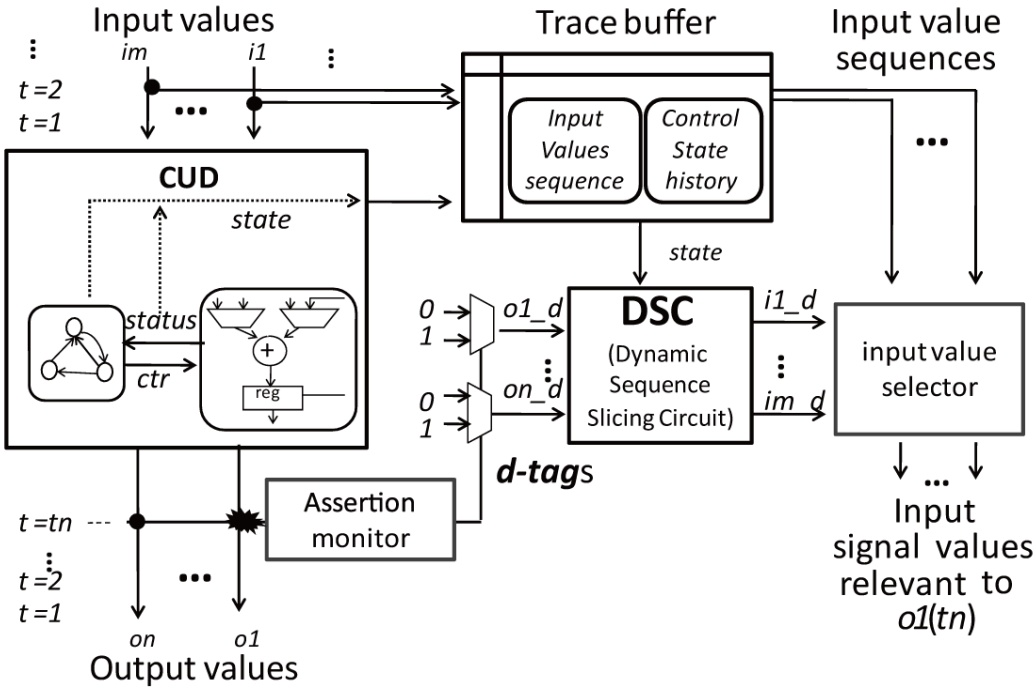

図 3.2.3　ダイナミックスライシング回路(Dynamic Slicing Circuit, DSC)

● プログラマブル制御回路を用いたポストシリコン設計修正技術

製造後の設計修正は FPGA のようなプログラマブル素子を用いることで可能である。しかしながら FPGA は専用ハードウェアに比べて非常に効率が悪く、そのため携帯機器といった電力制約の厳しい機器には適用が難しい。近年普及が進む高位記述を用いた設計手法では設計修正も高位記述上で行われるため、演算器内部の変更が必要であることは少なく、ほとんどの場合制御の変更だけで対処可能である。そこで我々は制御回路のみをプログラマブルにすることで効率を失うことなく製造後設計修正を可能とする設計手法を提案した。ビットレベルの設計変更が可能な部分的プログラム可能回路(次節で説明)と組み合わせることで様々な設計変更に対応することができる。

一般的なプログラマブル制御回路の実装方式はメモリを用いた水平型マイクロコード方式である。しかしながらこの方式は大規模なメモリを必要とするため電力効率が悪い。そこで我々は効率の良いプログラマブル制御回路として、命令レジスタファイル(Instruction Register File, IRF)を用いたプログラマブル制御回路を提案した。実際の設計を用いた実験では、水平型マイクロコード方式に比べて提案手法では約 60%の電力削減が可能であることを確認し、提案方式の有効性を実証した。

● 部分的プログラム可能回路を用いた耐故障性向上技術

昨年度、我々は回路の一部分の機能を変更可能にすることで耐故障性を向上する部分的プログラム可能回路(Partially Programmable Circuit, PPC)を提案した。PPCは図3.2.4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。

昨年度はPPCの初期検討のみであったが、本年度は耐故障性を最大化するようなPPCを合成する手法の提案およびベンチマーク例題を用いたPPCの有効性評価を行った。我々は提案手法をABCという論理合成ツール上に実装し、このツールを用いて例題をPPCに合成し耐故障性の評価を行った。27個のベンチマーク例題を用いた実験では、提案手法で合成したPPCは平均で15%の冗長配線の追加で回路全体の78%の配線を耐故障にすることが可能であることを確認し、その有効性を実証した。

PPCは耐故障性向上のみでなく、製造後における設計修正に応用することも可能である。このようなPPCを上記のポストシリコンデバッグ技術と組み合わせることで、図3.2.5に示すように製造のやり直しを行うことなく設計誤りを修正できるようになり、VLSIのディペンダビリティを大幅に向上可能であると考えている。今後は設計修正に適したPPCの合成手法を検討していく。

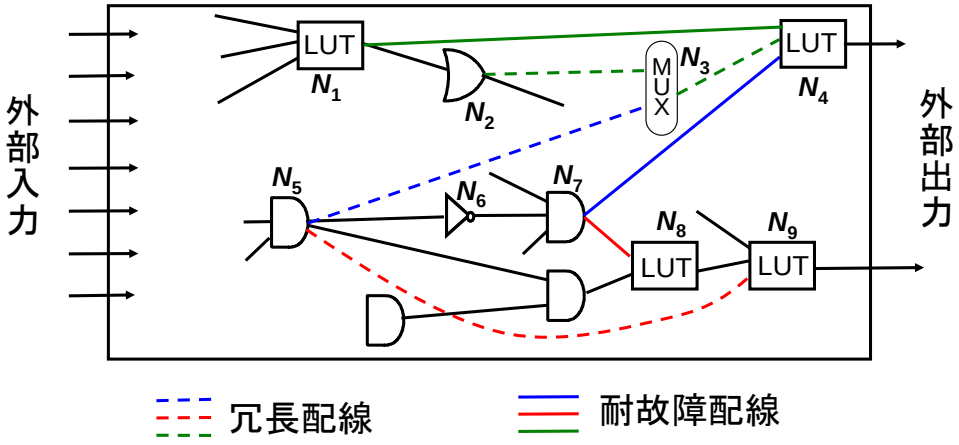

図 3.2.4 部分的プログラム可能回路(Partially Programmable Circuit, PPC)

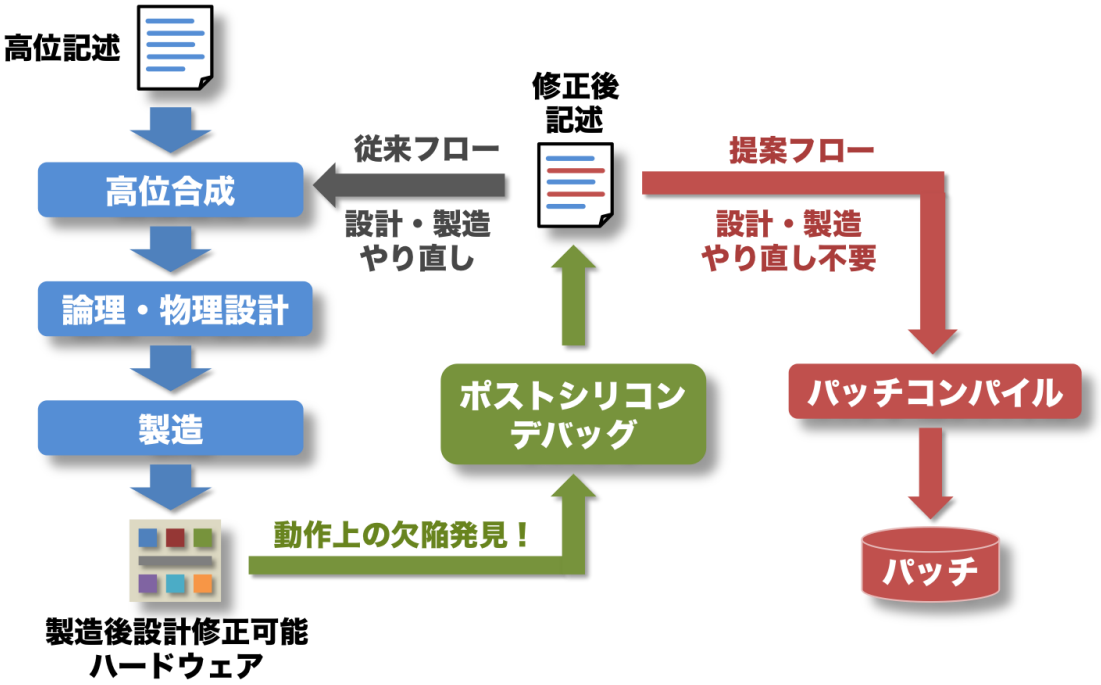

図 3.2.5 設計修正可能ハードウェアを用いたポストシリコン自動設計修正フロー

- 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

現在のプロセッサアーキテクチャは様々な最先端の技術が取り入れられており、その検証とデバッグは非常に難しくなっている。本研究では、マイクロプロセッサのアーキテクチャを形式的検証・デバッグ・最適化するための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。本研究で扱う手法は、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プロセッサなど、様々な種類のプロセッサを対象としている。手法では、検証対象のプロセッサアーキテクチャと、既に正しいことが証明されている非パイプラインの単純なプロセッサアーキテクチャの間で等価性を検証することにより、検証対象の正しさを検証する（図3.2.6）。

昨年度までに、アーキテクチャの検証を行うための基礎的な手法の開発を終えており、今年度は、その手法を実際にいくつかのエラーリカバリ方式に対して適用した。例えば、エラーが発生した際にリオーダバッファに残っている命令をやり直すリカバリ手法、エラーが発生した際にリセットを行うリカバリ手法などをモデル化し検証を行った。検証の結果、これらのエラーリカバリ方式が、元のプロセッサと同じ結果を出すことが証明できた。また、この検証を用

いて、対象のプロセッサアーキテクチャがどの程度のエラーであればリカバリ可能であるかどうかの評価も行う手法の研究を今年度後半から開始した。これにより、プロセッサアーキテクチャのリカバリ性能を保証することができる。

また、アーキテクチャ検証・デバッグ・最適化フローを提案している（図3.2.7）。アーキテクチャモデル中にマルチプレクサを挿入することにより（図3.2.8）、プロセッサアーキテクチャのデバッグを行う手法を提案・評価した[10,14]。この手法では、検証の結果として反例が得られた場合に、その反例を正しいトレースにするために修正すべき箇所を知ることができる。これにより、修正すべき箇所の候補を1桁以上減らすことができ(表3.2.1)、デバッグ効率の向上が期待できる。元の単純なプロセッサアーキテクチャと検証対象のプロセッサアーキテクチャが、同じ命令列に対して等価な結果を出すことが証明された場合、その等価性を維持したまあ検証対象のアーキテクチャを最適化（簡単化）することにより、検証時間を大幅に減らすことができることを示した（表3.2.2）。

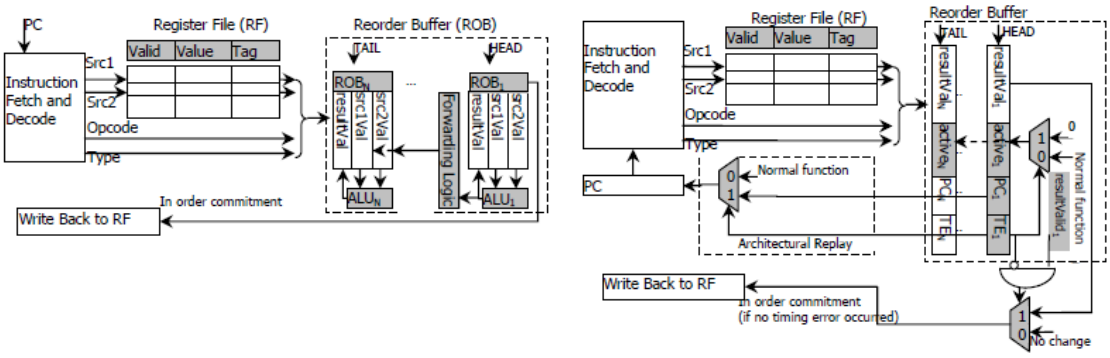

図3.2.6 Out-of-orderプロセッサ（左）とタイミングエラーリカバリ機能が追加されたもの（右）

表 3.2.1　5 命令発行 out-of-order プロセッサのデバッグ結果

Method/Bug		**#active control signals for debug**	**Reduction in active control signals**	**Time (sec)**
Traditional Debug Method	Bug1	151	N/A	5.5
	Bug2	110	N/A	17.3
Our Debug Method	Bug1	5	**30x**	12.8
	Bug2	10	**11x**	37.3
After Fixing All Bugs				639.8

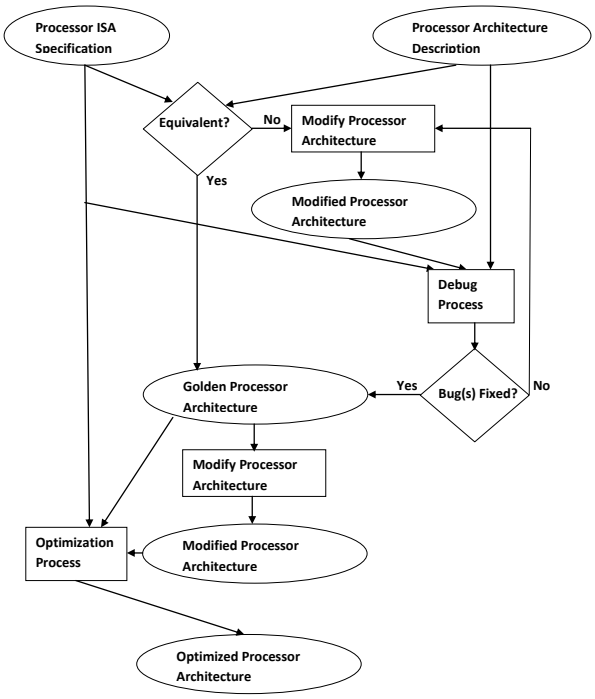

図 3.2.7 検証・デバッグ・最適化の全体の流れ

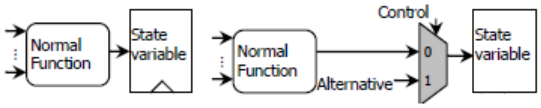

図 3.2.8 アーキテクチャモデル中へのマルチプレクサ挿入

表 3.2.2 N 同時命令発行 out-of-order プロセッサにおける最適化

Processor	N	Time (sec)
Without timing error recovery mechanism	5	0.5
	6	1.0
	9	100.7
With timing error recovery , before optimization	5	639.8
	6	Timeout
	9	Timeout
With timing error recovery, after optimization	5	3.3
	6	10.8
	9	2975.6
Average reduction in verification time		**~180x**

- 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 等価性を考慮した多項式レベルの最適化手法を考案・実装・評価してきたが、Bremen 大学 Drechsler 教授のグループと共同で、constraint optimization 問題として定式化し直し，SMT（Satisfiable Modulo Theory）ソルバーを用いて問題を解くことで考慮した constraints に対する最適化を得ることができ、従来の最適化をより強力に行う技術を新規に開発し、表 3.2.3 に示すように、従来比さらに 10%程度の高速化、面積縮小化を実現した[8]。処理時間も最大数十秒程度で最適化が得られている。我々の技術を利用しない場合（従来手法）と比較すると遅延/面積ともに 1/2 以下に縮小化できている。現在、この分野で利用されているベンチマークは、表 3.2.3 に示すものであるが、High Performance Computing の分野では、数千の多項式の同時最適化問題を扱わなければ成らない場合もあることが分かっており、現在そのような例題への適用を検討中である。現在は、ソフトウェアとして実装されている問題であるが、この研究成果はソフトウェア実装においても、多項式の研鑽という意味では、ハードウェア実装と同様に計算高速化を実現できる。この検討と行うと共に、FPGA などを利用した一部ハードウェア化によるさらなる高速化も検討していきたい。また同時に、FLEC（等価性検証環境）との統合も検討している。

表 3.2.3 SMT ソルバーを利用したより厳密な最適化結果

Benchmark	M/V/D/n	Our previous results			New proposal		
		Time (s)	Delay (ns)	Area	Time (s)	Delay (ns)	Area
ANTI	7/1/6/16	<0.01	38.78	100574	15.4	34.83	88761
DIRU	8/2/4/16	3.1	23.17	55631	57.9	20.57	48501
MVCS	9/2/3/16	2.6	27.49	111801	12.7	23.89	95147
PFLT	6/1/5/16	<0.01	39.38	76069	15.5	34.87	65819
PSK	9/2/4/16	4	29.24	141923	24.2	35.24	143277
QUAD	5/2/2/16	<0.01	21.64	48289	3.8	18.14	45090
SG2	9/2/3/16	<0.01	29.21	149496	6.4	25.58	125578
average savings						7.72%	10.47%

- 形式的解析手法を応用した検証カバレッジ向上手法

本研究では、これ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 3.2.9）。今年度は、Verona 大学と共同で、記号シミュレーションの履歴を活用しながら、検証カバレッジを向上する新規手法を考案し、実装・評価した。表 3.2.4 に示すように、乱数、固定値、記号シミュレーションいずれでも高い検証カバレッジを達成できなかったベンチマークに対しても、非常に高いカバレッジを達成できている。

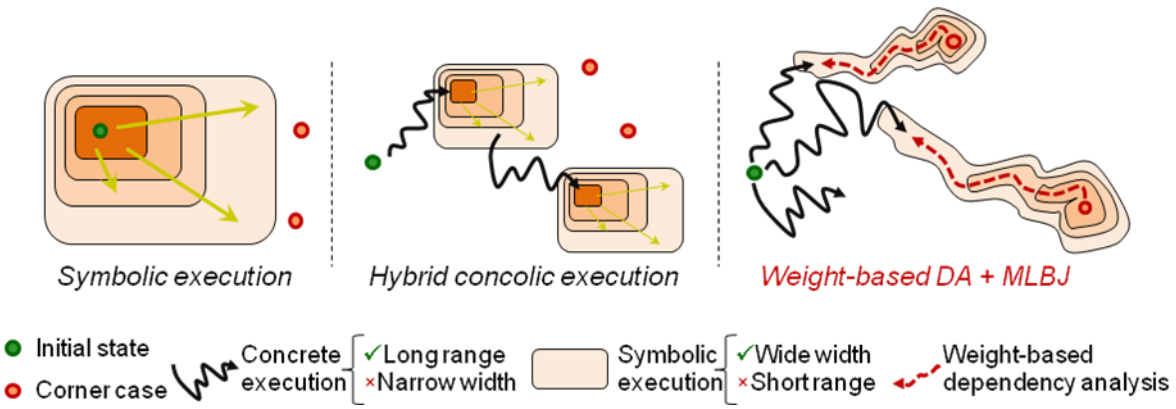

図 3.2.9　記号シミュレーションと固定値シミュレーションの統合による効率化

表 3.2.4　網羅的シミュレーションパターン生成結果

	乱数		固定値		記号 sim		新提案手法	
benchmark	TC%	ET	TC%	ET	TC%	ET	TC%	ET
Ciitp	96.1	6.8	96.1	18.7	100	180.3	100	23.8
Inres	89.4	24.1	92.7	18.4	100	186.8	100	24
Lift	11.1	0.7	81.4	63.5	100	1224.5	100	13.9
Ifss	32.3	25.6	55.8	59.1	100	1024.1	100	12.7
Atm	12.5	0.2	71.1	76	100	934.1	100	51
Thermox	33.8	8	47	170.5	47	3402.3	60.2	37.6
Filter	62.8	0.8	62.8	2.7	68.5	3503.1	100	30.5
Elevator	69.3	30.2	69.3	190.9	1.4	4914.1	81.5	4459.6

Execution time, ET is in seconds

- SoC・組込みシステム全体に対するシミュレーション（エミュレーション）ベース検証における検証手法

SoCや組込みシステムのような大規模システムでは、シミュレーションあるいはエミュレーションを如何に効率的に適用していくかが重要となる。上で説明した図3.2.1に示す設計・検証フローを適用することで、個々のブロック（あるいはIPコア）に対する検証・デバッグ支援を行うことができる。それを踏まえて、SoCチップあるいは組込みシステム全体に対する検証は図3.2.10に示すように、下記のようになると考えられる。

（ステップ１）個々のブロックレベル（数十万ゲート程度）の形式的検証
（ステップ２）チップ（システム）全体のシミュレーション（エミュレーション）ベース検証
（ステップ３）アナログ部も含めてシステム全体の正しさの検証

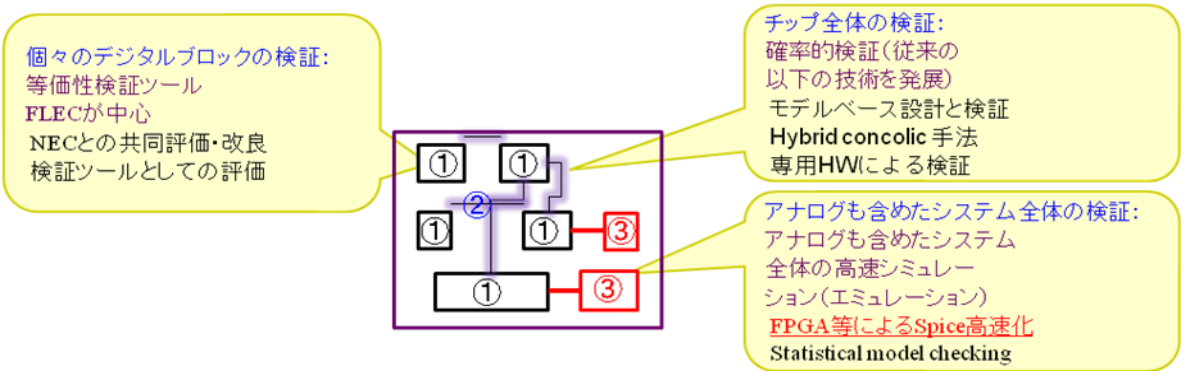

図3.2.10　チップ全体の検証の流れ

ステップ１により個々のブロックが正しくても、チップ（システム）の動作はその組合せとなるため、特に通信やインターフェイス関連の不良が起こり得る。したがって、ステップ２が如何に網羅的あるいは、検証カバレッジが適正であるかが重要となる。部分的には形式的手法がチップやシステム全体でも適用されるが、検証は基本的にシミュレーションあるいはFPGA

などによるエミュレーションによる行われることになる。これには、確率論的・統計論的に検証カバレッジを把握することが重要となるとともに、シミュレーションやエミュレーション時にバグや不具合の的確に判定する必要があり、アサーションベース検証と呼ばれる手法が重要となる。アサーションを用意しておき、シミュレーションやエミュレーション時に動的にチェックする。このため、如何にバグや不具合を効率的に検出できるアサーションを用意するかが主な研究テーマの１つであり、シミュレーションやエミュレーション結果から自動的にアサーションを抽出する手法を検討している。

また、高品質のアサーションを作成するには、設計の仕様レベルからの抽出も重要である。そこで、Verona大学と共同です。図3.2.11に示すフローでアサーションを生成し、改良していく手法について、研究を進めている。現在までに基本アルゴリズムを作成し、その評価から、いくつかの例題に対して、有益かつカバレッジの高いアサーションが生成できることが分かっている。さらに、与えられたアサーションの有効性／有益性の評価技術についても研究を開始した。アサーションはシミュレーション／エミュレーション時に評価されることになるが、バグや不具合の検出に有効でないアサーションは、単純にシミュレーションやエミュレーションの速度を低下させるだけで、結果として検証効率は悪くなる。そこで、個々のアサーションの有効性／有益性を効率的に評価し、選択する手法について研究している。

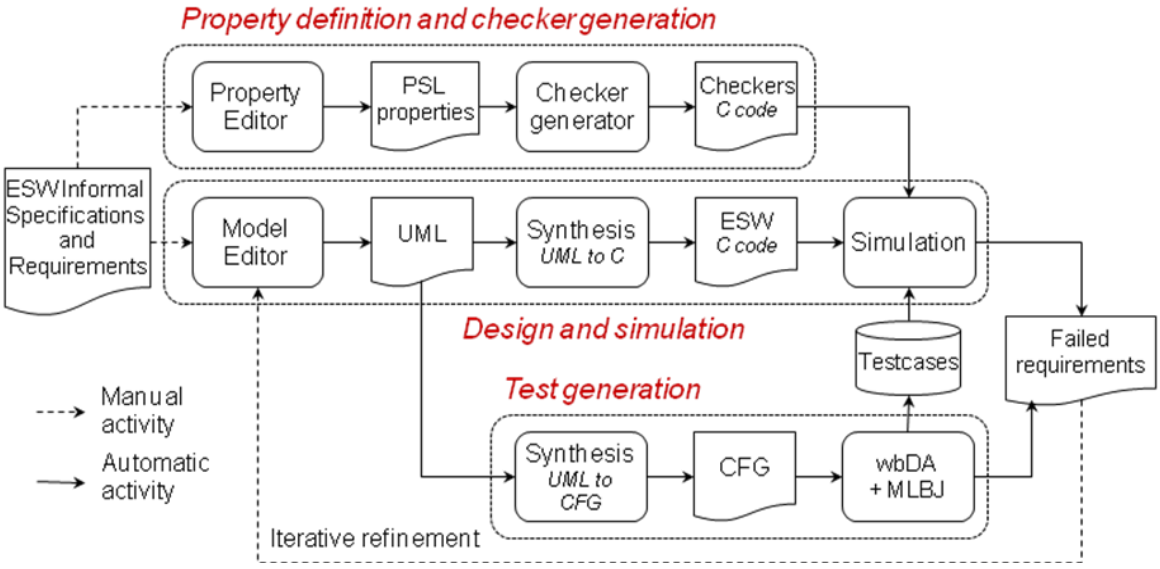

図3.2.11　仕様からアサーション生成の流れ

また、シミュレーションの超高速化、つまり、FPGAなどによるエミュレーションも重要であり、アナログ部も含めた超高速シミュレーション環境を構築する必要がある。現在図3.2.12に示すアーキテクチャのエミュレーション環境について検討している。デジタル部だけでなく、アナログ部についても、通常のプロセッサ上での実行と比較し、数十倍の高速化の実現を目標としている。従来のFPGAベースのエミュレーションと異なり、設計をそのままFPGAで実現するのではなく、設計をデータと見なし、そのデータをパイプライン的に処理することで、デジ

タル・アナログ両方を同じように扱え、SoCやシステム全体、さらにその上で動作する基本ソフトウェアも含めたエミュレーションが可能となる。これを利用し、さらに図3.2.13に示すように、形式的な手法も導入することで、全体の検証カバレッジの向上を目指していく。今年度は、主に検討を行ったが、来年度以降、試行的な実装を通して評価を行いながら、改良や新規手法の提案を行うことを予定している。

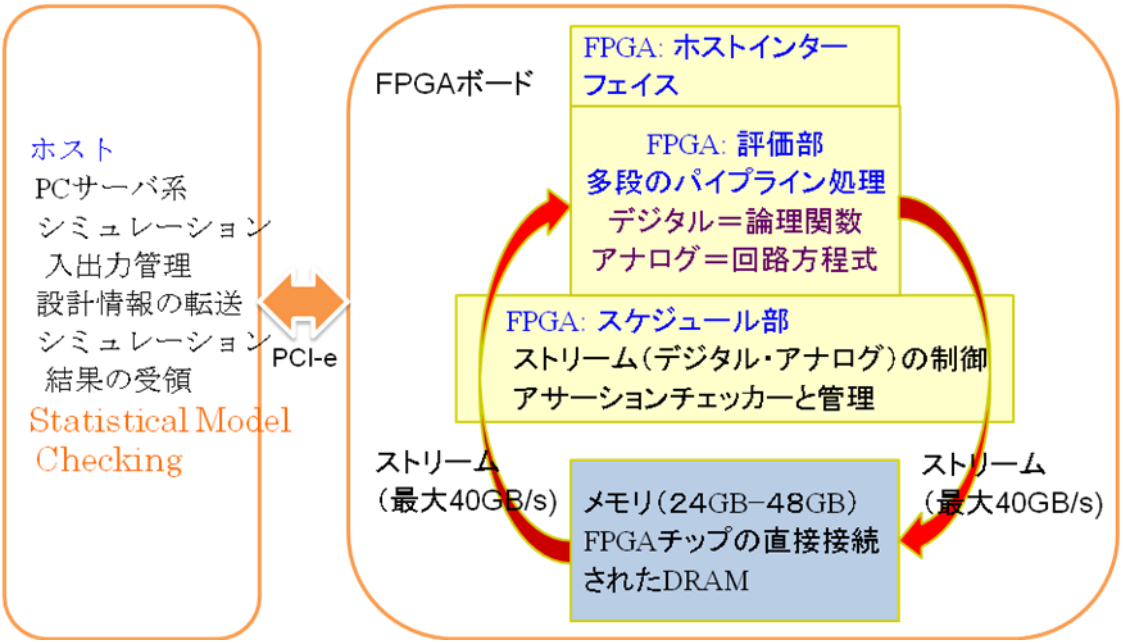

図3.2.12　デジタル・アナログ全体を超高速にシミュレーションするアーキテクチャ

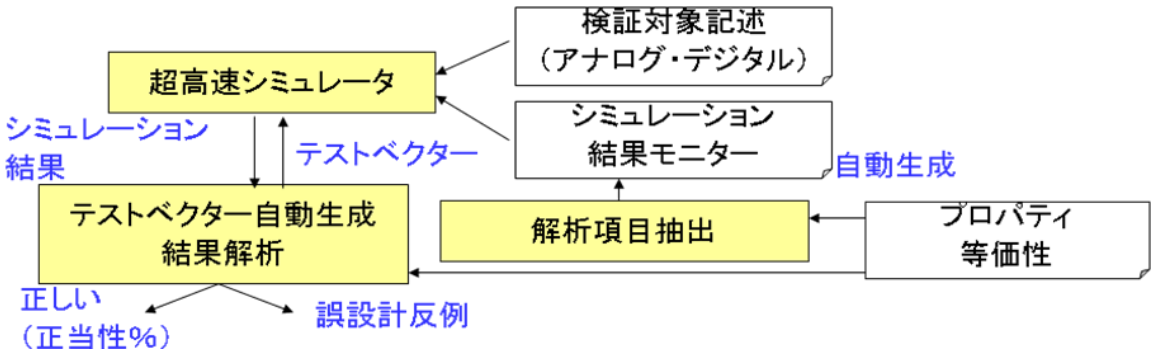

図3.2.13　超高速シミュレータと形式的検証手法の組合せ

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 22 年度は、第一に、これまでに開発してきた DMR（Dual Modular Redundancy）を実現する高機能ルータをベースに TMR(Triple Modular Redundancy)を実現する手法を開発した。

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。図 3.3.1 左の例では、Node（3, 2） と Node（4, 2）を用いて 2 重実行をおこなうことでDMR（Dual Modular Redundancy）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。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。これら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。このために、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。

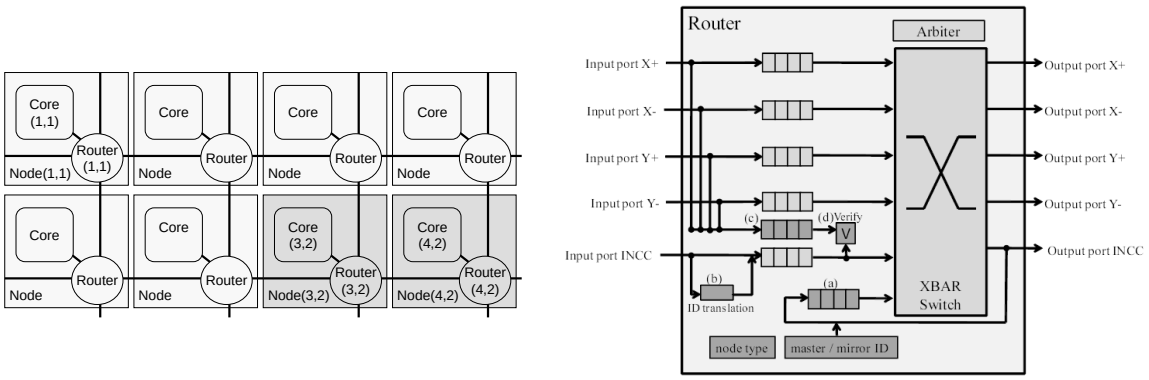

図 3.3.1 SmartCore システムを用いた DMR （左）と高機能ルータの構成（右）

開発した高機能ルータアーキテクチャの構成を図 3.3.1 右に示す。高機能ルータは、自身がマスターノードであるかミラーノードである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、ミラーノードの導入時に、システムソフトウェアによって設定される。

DMRを実現する高機能ルータは、自身のノードが受信するパケットを複製し、他の1ノードに送信する機能と、他の1ノードから送信されたパケットと自身のノードから送信されたパケットの2つのパケットを比較してエラーを検出するハードウェアを搭載する。TMRを実現する高機能ルータは、3パケットを比較する機能を持たせるのではなく、2パケットを比較するDMRの手法を拡張することで、そのTMRを実現する。

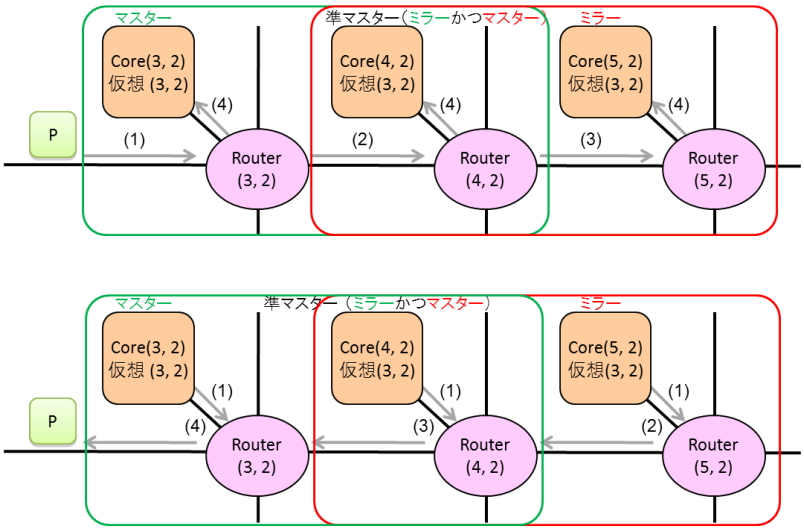

図 3.3.2 SmartCore システムを用いたTMR（上）とエラー検出（下）

図 3.3.2 に SmartCore システムを用いたTMR手法の概要を示す。ここでは、Node（3, 2），Node（4, 2），Node（5, 2）の3ノードを冗長実行のために割り当てる。これらのノードに同じ実行ファイルをロードし、仮想 IDを Node（3, 2）に設定する。

3つのノードを区別するために、DMRで設定したマスターノード、ミラーノードに加えて、準マスターノードという概念を導入する。準マスターノードとはマスターノード，ミラーノードの両方の役割をもつノードである。図 3.3.2 では、Node（3, 2）と Node（4, 2）および Node（4, 2）と Node（5, 2）がマスターとミラーに関係にある。Node（4, 2）は Node（3, 2）のミラーかつ Node（5, 2）のマスターであり、これを準マスターノードとする。

図 3.3.2 上は、3ノードが同じパケットを受信する仕組みを表している。 図の(1)～(4)の動作を順番に説明する。まず、Node（3, 2）のルータに受信するパケットが届く。次に、Node（3, 2）のルータはマスターなので、パケットを複製しミラーである Node（4, 2）に送信する。次に、Node（4, 2）のルータはマスターなので，Node（3, 2）から送信された自ノード宛てのパケットを複製し，Node（5, 2）に送信する。そして、各ノードのコアが同じパケットを受信する。

図 3.3.2 下は、3つのノードが送信するパケットを比較し、多数決によって正しいパケットを選択する様子を表している。高機能ルータは2つのパケットを比較する機能をもつ。そのためTMRを実現するために 2パケットの比較を準マスターノードとマスターノードにおいておこない、トーナメントでエラーを検出する。それぞれのノードの比較結果から、単一故障を想定する場合に多数決を実現する。

図 3.3.2 下におけるパケット送信の流れを述べる。各コアからパケットが送信される（1）. Node（5, 2）のルータはミラーなのでパケットの送信先をマスターである Node（4, 2）に変更し送信する（2）。Node（4, 2）のルータは準マスターなので、Node（5, 2）と自ノードのコアからのパケットを待ち合わせ、比較する。また，パケットの送信先をNode（3, 2）に変更し、比較結果をパケットに付与し送信する（3）。Node（3, 2）のルータはマスターなので、Node（4,

2）と自ノードのコアからのパケットを待ち合わせ、比較する。準マスターにおける比較結果を考慮し、正しいパケットを送信する（4)。

表 3.3.1 単一故障を想定した場合の故障箇所

(3, 2)と(4, 2)の比較，手順(3)	(4, 2)と(5, 2)の比較，手順(2)	エラーが発生したコア
OK	OK	なし
OK	NG	(5, 2)
NG	OK	(3, 2)
NG	NG	(4, 2)

表 3.3.1に、Node（4, 2）における比較結果と Node（3, 2）における比較結果から判明する故障箇所および Node（3, 2）が送信するべき正しいパケットの送信元をまとめる。表における手順（2）および手順（3）は，図 3.3.2 下の手順に対応している。準マスターノードにおける比較結果とマスターノードにおける比較結果から故障箇所が判明し、マスターノードは正しいパケットを送信することができる。これによりTMRを実現する。

これら方式検討に加えて、開発中のFPGAシステムに高機能ルータアーキテクチャを部分的に実装し、正しく動作することを確認した。

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。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のソフトウェアシミュレータ SimMc を構築している。

本年度は、ソフトウェアシミュレータ SimMc の改良および機能拡張をおこなった。主な改良点はキャッシュ階層の追加によるメモリシステムの詳細化である。構築したマルチコアプロセッサ M-Core の構成を図3.3.3に示す。この構成におけるソフトウェアシミュレータを SimMc 2.0 と定義している。

マルチコアプロセッサシミュレータ SimMc 2.0 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 3.3.4 左にまとめる。

また、前年度に開発を行っていた SimMc1.0 とのシミュレーション速度の比較を図 3.3.4 右にまとめる。メニーコアのアーキテクチャを詳細化したため、シミュレーション速度の低下が懸念されたが、最適化を施すことで、逆に僅かながらシミュレーション速度の改善を達成できている。

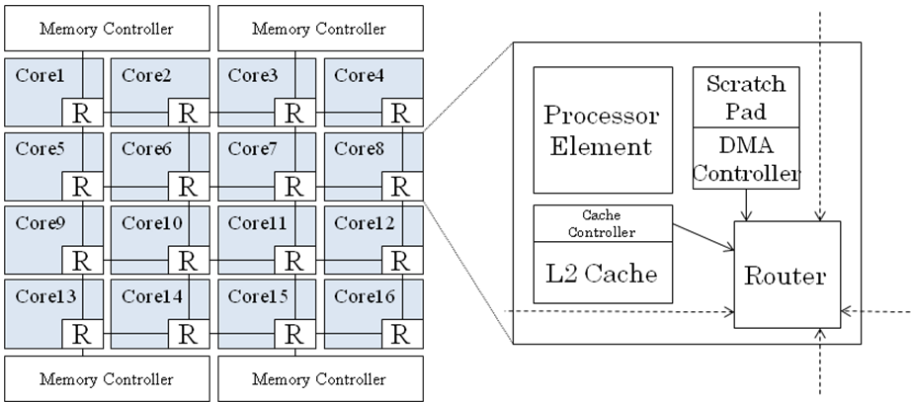

図3.3.3 キャッシュを追加したM-Coreアーキテクチャ

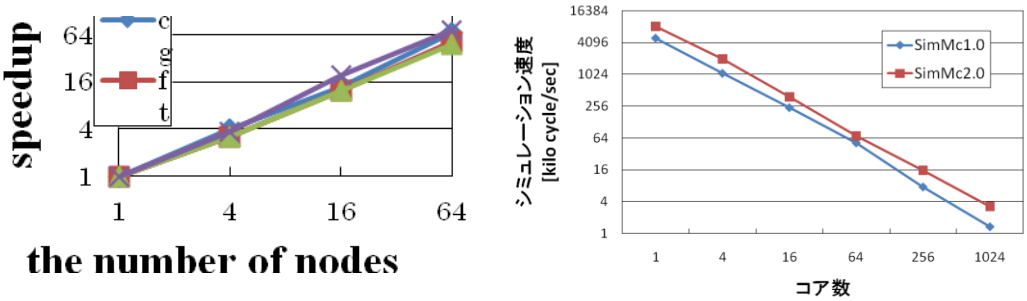

図3.3.4 M-Coreアーキテクチャの台数効果（左）とシミュレーション速度（右）

第三に、本研究における高機能ルータの詳細評価のための FPGA システムの開発及び整備を進めた。

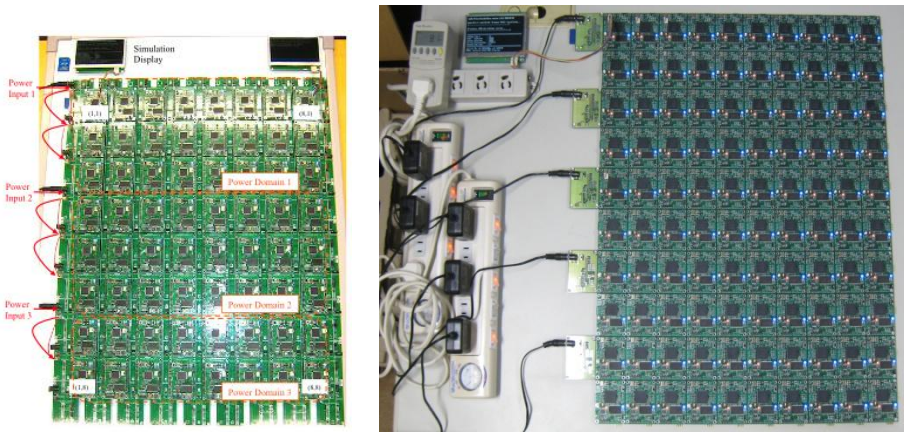

図 3.3.5 前年度設計した ScalableCore ボードと ScalableCore ユニット 64 枚によるシステム（左）と今年度設計した ScalableCore ユニット 100 枚によるシステム（右）

多数の FPGA を搭載する従来の評価用 FPGA 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[12]している。本年度はこのシステムの改良およびシステム構築をおこなった。

今年度、新規に設計した FPGA ボード（ScalableCore ユニット）を 64 枚接続してシステムを構築して評価したところ、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約 14 倍に高速化できることを確認した。また、今年度は新たに設計した ScalableCore ユニット 100 枚によるシステム（図 3.3.5 右）を構築し、ソフトウェアシミュレータと比較して約 25 倍の高速化が達成できることを確認した。

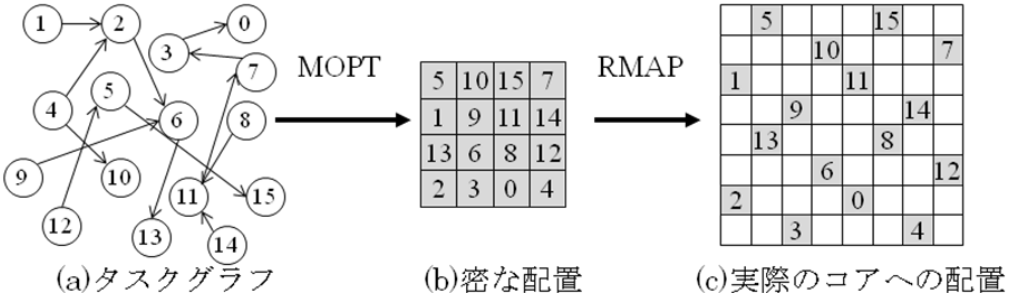

図 3.3.6 提案するタスク配置手法の流れ

第四に、メニーコアプロセッサのためのスケジューラを検討した。このスケジューラはタスクの配置を考慮して高速化を達成する。

提案する配置手法では、図 3.3.6 に示す 2 段階のプロセスを採用する。図 3.3.6(a) は並列化されたアプリケーションのタスクを表す。提案手法ではタスクグラフに最適化手法 MOPT を適用し、密な配置を求める(図(b))。その後、実際に使用するプロセッサのコア数を考慮し，配置手法 RMAP によって配置を行う(図(c))。

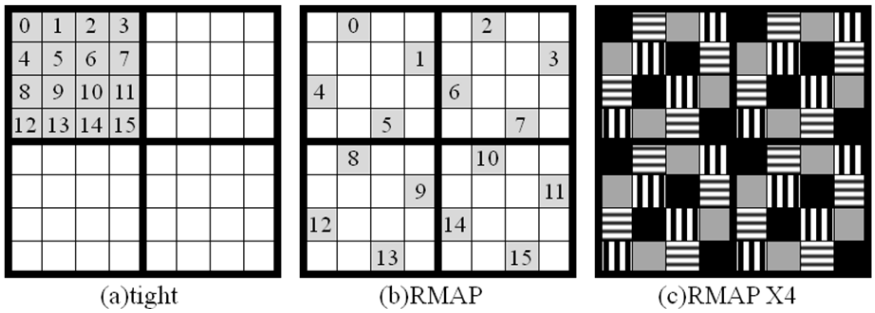

図 3.3.7 64 ノードに RMAP を適応した配置

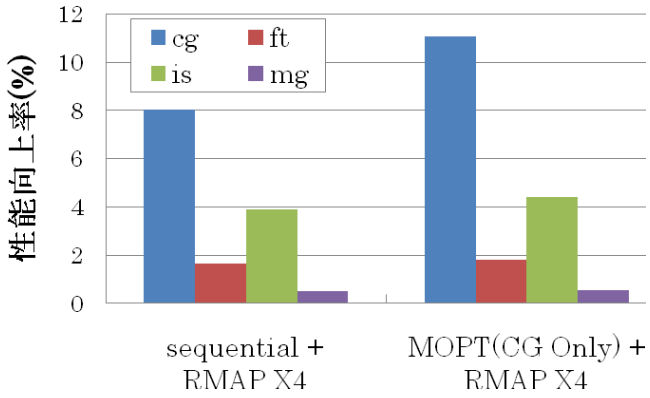

図 3.3.8 密な配置(sequential + tight)と比較したときの RMAP X4 の性能向上率

RMAP を使用して、64 ノードのメニーコアプロセッサに 16 個のタスクを配置する例を図 3.3.7 に示す。これらのタスク配置手法を図 3.3.7(a)に示す密な配置(tight)と比較した結果を図 3.3.8 に示す。MOPT と RMAP を組み合わせる提案手法は、既存手法の密な配置と比較して、平均 4.5%の性能向上を達成する。

3.4 検証技術実証グループ

形式検証実証グループでは、形式的検証グループの研究成果の実用化を目指して当年度の平成 23 年 1 月より本 CREST 研究活動に参加した。日本電気株式会社(NEC)の商用高位設計環境である CyberWorkBench(CWB)を用いて設計された実回路データを利用するなどして、形式的検証グループの研究成果を設計現場の観点から評価・実証を行い、産業界での実用化を目指すことを主な目的とする。実規模のデータに適用することは簡単ではないが、実用化できた場合にディペンダビリティへの寄与が大きいと考えられる。当初のターゲットとして等価性検証ツール(FLEC)の実証から活動を開始することとした。本活動の進捗に応じて、他の形式的検証技術や高位合成関連技術も導入候補とすることも検討している。

活動報告に先立ち CWB について簡単に説明する。CWB は、NEC から販売されている高位統合設計環境であり、図 3.4.1 にツールの全体像を示す。ANSI-C 言語および SystemC 言語を入力とする動作合成ツールを中心として各種検証ツールも内包し、SoC および FPGA の設計及び検証を C 言語で行うことを可能としている。社内において広く実製品の設計に用いられた実績を有している。また 2007 年から本格的に外販を開始しており、現在では数十社の顧客を有している。

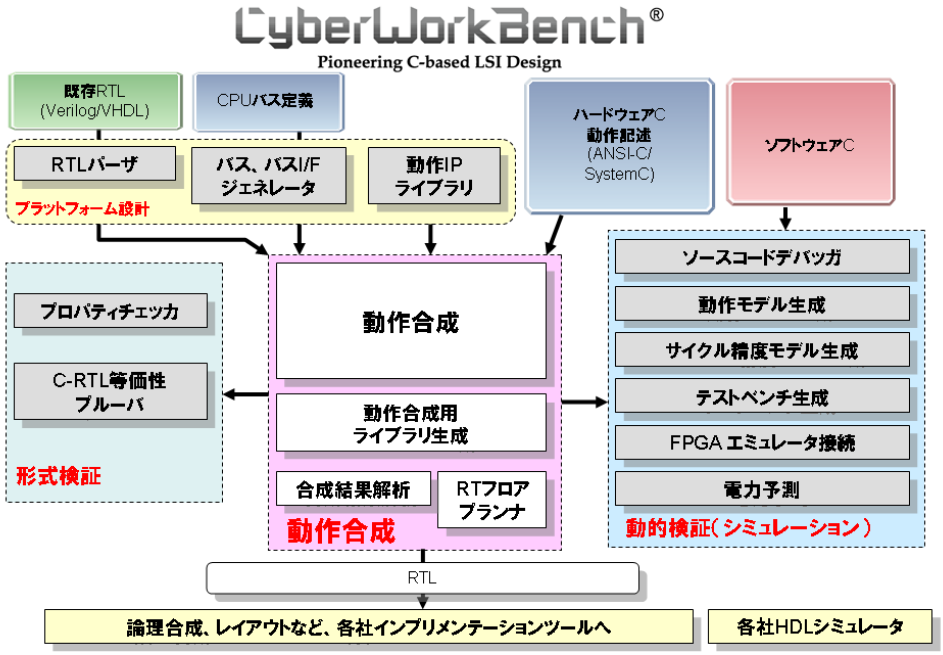

図 3.4.1 CyberWorkBench(CWB)概要

本年度は、主に 3 つの項目について活動を行なった。第一に、CWB を用いた実回路設計に FLEC を組み込んだ利用シナリオの検討を行なった。その結果、FLEC の導入効果が高いと考えられる 2 つのシナリオを想定することとした。1 つ目のシナリオは検証済の動作レベル IP から異なる製品向けの RTL を合成する場合、2 つ目のシナリオは検証済の動作レベル IP を動作レベルでチューニングする場合であり、図 3.4.2 に想定したシナリオ 2 つの概略を示す。

1 つ目のシナリオは、検証済の動作レベル IP から目的に応じた様々な RTL 回路を合成する場合に、FLEC を利用してそれぞれの回路が動作レベル IP と等価な動作をすることを保証するというシナリオである。形式的手法に基づく等価性検証技術によって動作レベル IP と RTL 回路の等価性を検証することで、シミュレーションに基づく検証と比較して網羅性の高い検証をより高速に行えることを期待している。動作レベル IP とは、IP コアのうち、動作記述の形で提供されるものであり、あらかじめロングラン検証などで徹底的に検証されている。

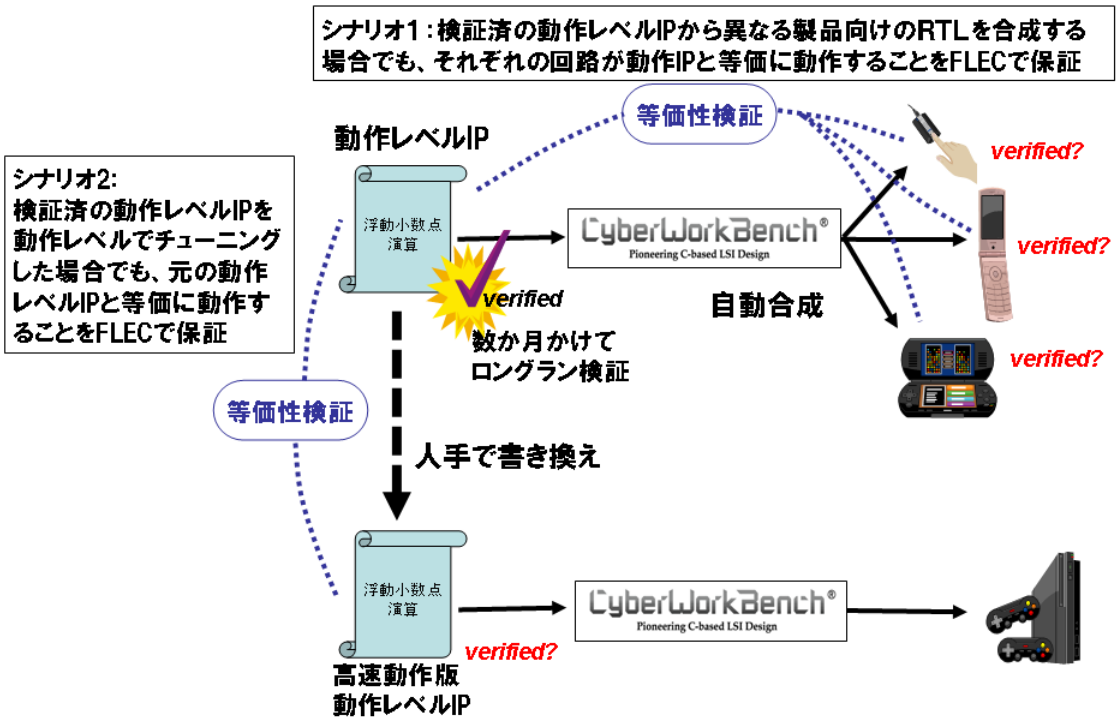

図 3.4.2 産業界における等価性検証技術の利用シナリオ

動作合成では、1 つの動作レベル IP から合成時の制約を変化させることによって目的に応じた速度や面積、消費電力の異なる RTL 回路を合成することができる。例えば、同じ動作記述から処理性能が要求される機器で用いられる回路が必要な場合には、演算器を多く用いるような制約を与えて面積や消費電力が大きいが高速に動作する RTL 回路を合成する。逆に、携帯機器などのように処理速度よりは消費電力や回路面積を重視する場合では、使用する演算器を少なくするような制約を与えて、処理速度は制限されるが小規模な回路を合成する。このような特徴から、検証済の動作レベル IP から異なる製品向けの RTL 回路を合成することが行なわれる。動作レベル IP が徹底的な検証によりその動作の正しさが保証されていても、これから合成した RTL 回路が同様に正しい動作をするとは限らない。これは、複雑な動作合成の過程にバグが存在する可能性があるためである。そこで、実製品として市場投入後に問題が発覚することがないよう、動作合成で得られた RTL 回路のそれぞれに対しても漏れのない検証が必要となる。しかし、現状の RTL 回路の検証に一般に用いられる RTL 回路の検証は、動作レベルでの検証に対して数倍から数百倍程度の時間を要するのが通常である。そのため、検証済動作レベル IP と同等の検証精度を得るためには数ヶ月からそれ以上の長時間検証が必要となり、それでも検証漏れの問題は避けられない。この問題に対応するのが 1 つ目のシナリオである。

2 つ目のシナリオは、検証済の動作レベルの IP を動作レベルで書換えてチューニングした場合でも、FLEC を利用して書換え前の動作レベル IP と等価な動作をすることを保証するというシナリオである。等価性検証技術を用いることにより、書換えが頻繁に行なわれる場合でも、書換え前後の動作記述同士の等価性を網羅的に効率よく行うことが出来るものと期待してい

る。

実回路設計においては、所望の性能を得るためのチューンアップを目的として検証済の動作レベル IP に対して変更を行うことがある。例えば、関数の切り分け方を変えたり、配列の構造を変えたりという書換えである。動作レベルでの検証は RTL 回路の検証と比べると高速ではあるが、それでも元の IP と同等の検証品質とするためには長い検証時間が必要となる。また、このような書換えが必要になった場合には、回路が所望の性能を得るまでに細かな書換えを頻繁に行なうことが多いため、できるだけ高速に漏れの無い検証を行いたい。この問題に対応するのが 2 つ目のシナリオである。

第二に、CWB のデータを FLEC で検証可能にするインターフェイスの検討を行なった。CWB は、ANSI-C 言語、ANSI-C 言語をハードウェアの記述を効率よく行なえるように入出力端子や変数へのビット幅指定、ビット演算などが記述できるよう拡張した BDL 言語、又は SystemC 言語を動作記述として扱い、Verilog HDL または VHDL を RTL 記述として扱う。FLEC は SpecC 言語のみを扱う。この違いがあるため、CWB のデータをそのままの形では FLEC へ入力できない。そこで図 3.4.3 に示すように、両ツール間のインターフェイスを検討することとした。FLEC で検証するためには、検証対象データと、検証したい検証仕様の 2 つを与える必要があるが、このうち本年度は CWB の入出力データを FLEC に入力可能となるようデータ形式の変換方法について検討した。

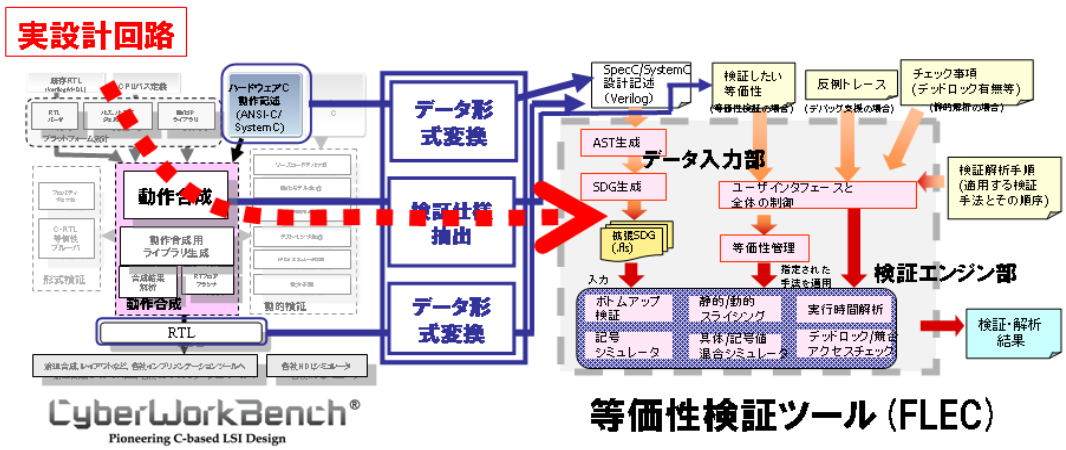

図 3.4.3 CWB-FLEC 間インターフェイス

FLEC は、データ入力部において SpecC 言語による記述からまず抽象構文木(Abstract Syntax Tree: AST)を生成し、続いて AST から拡張システム依存グラフ(System Dependence Graph: SDG)を生成した上で、SDG を入力に記号シミュレーションなど検証エンジン部の処理が行なわれる。そのため、データ形式の変換方針として、(1)CWB への入出力データを SpecC 言語に変換する方針、(2)CWB への入出力データを FLEC の内部表現に直接変換する方針、の 2 つが考えられた。検討の結果、実証実験では(1)の方針で検討を進めることとした。このように選択した理由

は以下の通りである。(2)の方針では、変換規則が複雑でインターフェイスの構築に時間がかかると考えられた。これに対して、動作記述は SpecC 言語と同様に C 言語をベースとするため言語仕様に共通部分も多い。RTL 記述についても言語の差は有るが変換規則などは FLEC の研究過程で概ね明確になっている。このことから(1)の方針では変換規則が単純でインターフェイスが短期で構築でき、早期に多くのデータを用いて実証実験を行なう上で有利と考えられたためである。

この方針で、CWB の入力となる BDL 言語による動作記述の SpecC 言語への変換方法の検討を行なった。その結果、(1)CWB の合成単位ごとに変換する(2)当該合成単位ごとに SpecC 言語の動作クラス(behavior)で表現する(3)動作記述の入出力端子は SpecC 言語のポートに変換する(4)ビット幅指定とビット演算を SpecC 言語のビットベクタ型に変換する、という変換方法を取ることで、動作記述について、概ね SpecC 言語への変換が可能であることを確認した。この変換規則はほぼ自動化が可能であり、実際に変換するインターフェイスも試作し、問題なく変換できることを確認した。ただし、ビット精度の扱いについて、BDL 言語と SpecC 言語で相違がある場合も分かったため、更に精査して変換規則に反映する予定である。

Verilog HDL による RTL 記述の SpecC 言語への変換方法については、Verilog HDL から直接 SpecC 言語へ変換する方法と、Verilog HDL から CWB の機能を利用して一旦 BDL 言語に変換し、BDL 言語から SpecC 言語へ変換する方法、の 2 通りの変換方法を検討した。前者は、FLEC の研究過程で Verilog HDL から SpecC 言語への変換方法が検討されている。主要な変換規則として、(1)RTL 記述の各モジュールに対し、SpecC 言語上で init()、run_one_cycle()、main()の関数を用意する(2)ワイヤを wire bit 型に変換する(3)レジスタを bufferd 型変数へ変換する(4)assign 文を init 関数内の代入文に変換する(5)always 文を run_one_cycle 関数内の文に変換する、という規則を適用して Verilog HDL から SpecC 言語への変換が可能である。また、後者は、動作記述で検討した変換規則を適用した上で、ワイヤ相当の変数とレジスタに相当する変数を区別し、(1)ワイヤ相当の変数を wire 型に変換する(2)レジスタ相当の変数を buffered bit 型に変換する(3)ワイヤ相当の変数への代入を init 関数内に記述する(4)レジスタ相当の変数への代入を run_one_cycle 関数内に記述する、という規則を適用すればよい。後者のフローについても実際に変換するインターフェイスも試作し、問題なく変換できることを確認した。これらの変換規則の一部は表 3.4.1 の通りにまとめられる。また、ごく簡単な BDL 記述から SpecC 記述への変換例を図 3.4.4 に示す。動作記述であるため、表 3.4.1(a)の変換規則を用いることで記述が変換可能であることが分かる。

表 3.4.1 ANSI-C(BDL)言語から SpecC 言語への主な変換規則

(a)動作記述・RTL 記述両方で必要な変換規則

変換元の ANSI-C(BDL)記述	変換後の SpecC 記述
(1)CWB 合成単位のトップレベル関数	全体を動作クラス(behavior)で表現し、トップレベル関数を main()関数に変換
(2)入出力端子	ポートに変換
(3)ビット幅指定された変数	bit 型の変数に変換
(4)ビット演算	連節演算(@)、切り取り演算([*lb*,*rb*])に変換

(b) RTL 記述で必要な変換規則

変換元の ANSI-C(BDL)記述	RTL 記述で必要な規則
(1)ワイヤ相当の変数	wire 型の変数に変換
(2)レジスタ相当の変数	buffered 型の変数に変換
(3)ワイヤ相当の変数への代入	init 関数内に当該代入文を記述
(4)レジスタ相当の変数への代入	run_one_cycle 関数内に当該代入文を記述

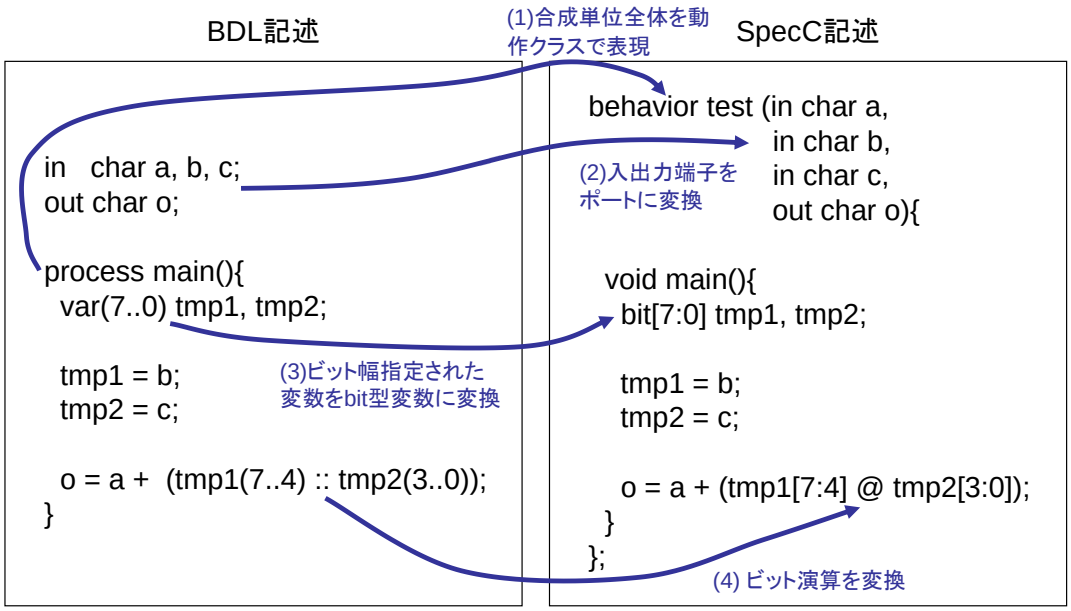

図 3.4.4 BDL 記述から RTL 記述への変換例

上記の変換規則で変換可能な記述であっても、現時点での FLEC のデータ入力部の制限で等価性検証ができない場合がある。具体的には、関数呼び出しを含む記述、ポインタを含む記述、

多次元配列を含む記述などを FLEC にそのまま入力とすることができない。しかし、本年度の実証実験では、検証エンジン部の性能、つまり検証できる回路規模や検証時間の評価を優先することとした。データ入力部の制限に抵触しないようにするには、人手で制限事項を含まないように書き換えるか、動作合成中の一部の処理を利用して関数展開・ポインタ解消するなどして回避することにした。

このようにして変換された記述を FLEC で等価性検証を行うためには、さらに、いくつかの入力が必要にある。その一つが、検証仕様である。検証仕様とは、たとえば、入力ポートや出力ポートの対応関係である。今回想定しているシナリオでは、これらの対応関係を与えることはそれほど難しくないと予想している。1 つ目シナリオの場合では、信号名や動作合成の内部情報から与えることができ、2 つ目シナリオの場合では、書換えを行なう設計者自身がこれらの情報を知っているのが通常であるからである。しかしながら、実用規模の回路を検証するためには、この入力ポートや出力ポートの対応関係のほかに、補助情報が必要になると予想している。補助情報とは、入出力以外の変数や信号の対応関係や、ループや条件分岐の存在箇所の対応関係などである。このような補助情報を与えることで、一度に検証する範囲を小さくし、検証エンジンの負荷を軽減することができる。しかしながら、このような補助情報は本プロジェクトの対象外となってしまっており、今後の検討課題とすることとした。

第三に、実証実験を行なった。評価対象データは、CWB と共に提供されている動作 IP 群から、暗号化の標準方式の一つである Data Encryption Standard (DES)暗号回路、及び浮動小数点演算回路(加減算回路、乗算回路の 2 つ)を選択した。これらデータの実証実験の後に別の暗号化の標準方式である Advanced Encryption Standard (AES)暗号回路の評価を行なうことも予定している。最初の実験のため比較的小規模かつ扱いやすいものを選んだ。記述行数を表 3.4.2 に示す。しかし、小規模といっても検証は必ずしも容易でない。例えば、浮動小数点演算器の場合、高速な動作シミュレーションを用いても 10^{11} パタン(=1000 億パタン)を検査するのに 6 か月程度の期間がかかる。低速なＲＴＬシミュレーションを行った場合には、さらに 6 から 7 倍の期間がかかる。しかも、このパタン数は全入力パタンのごく一部であり、網羅度としては 5.4×10^{-7}%程度に過ぎない。したがって、一度動作レベルでの徹底的な検証を行ったあとに、動作合成後の RTL に対する検証や、記述書き換え後の検証を、シミュレーションによって行うことは現実的ではない。そのため、FLEC により RTL 回路の等価性が短時間に検証できれば、回路の信頼性を高めるために有効であると考えられる。

実験形態としては、先に説明した等価性検証の 1 つ目の利用シナリオに相当する形態として、BDL 言語による動作記述と CWB による動作合成により生成した RTL 回路(Verilog HDL による出力)との間の等価性検証を行うことにした。また、2 つ目のシナリオに相当する形態として、動作レベル IP を動作が変わらない形で書き換え、これを元の記述と等価性検証を行うことにした。評価内容としては、実回路規模のデータの等価性検証においても実用的な検証時間やメモリ使用量で検証可能であるか、疑似エラー(2 つの回路が等価か否かに関わらず常に等価/非等価との検証結果を出力してしまうこと)の有無、などをはじめとして実際の設計現場での運用可

能性を確認することを想定している。問題があれば問題点を明確にして今後の改善につなげることとする。

表 3.4.2 実証実験対象データ規模(各記述の行数)

	動作記述(行数)	RTL 回路(行数)
DES 暗号回路	837	1389
浮動小数点演算回路(加減算)	546	2338
浮動小数点演算回路(乗算)	421	2286

実証実験の状況は DES に対して以下を確認したところである。動作記述を SpecC 言語へ前記の変換方針の通りに変換できることを確認した。一部にポインタを含む記述について人手でポインタのない形に書き換えた。関数呼び出しが多く、人手による解消が煩雑であったため、ツールによる関数展開を行なった。本実験ではエンジン部の性能評価が主目的のためこのような方針としたが、この変換過程は検証結果に影響があり得るので、実用化に向けて何らかの対策が必要と思われる。動作記述を変換した SpecC 記述は最終的なデータ形式であるシステム依存グラフ(SDG)への変換ができた。

RTL 記述についても、前記の変換方針の通りに変換を進め、SpecC 言語に変換することができた。また、RTL 記述を変換した SpecC 記述についても抽象構文木(AST)へ変換することができ、この後 SDG へ変換する前に RTL 展開という処理を通してタイムド動作記述(Timed Behavior Level: TBL)へ変換する過程である。引き続き対応関係の指定などの検証仕様の検討、記号シミュレーションといった等価性検証の各過程の処理を行い、実証を進める予定である。

以上、検証技術実証グループの活動内容をまとめる。最初の実証実験として等価性検証技術を対象とし、実設計での等価性検証技術の利用シナリオの検討、接続インターフェイスの検討、実データの評価を行なった。利用シナリオの検討では 2 つのシナリオを想定し、そのシナリオに基づいて実証を進めることとした。接続インターフェイスの検討では、入力データ形式の変換方法を中心に検討し、CWB の入出力データを FLEC に入力するインターフェイスの試作も行なった。暗号回路や浮動小数点演算回路をターゲットとした実証実験においては、主に動作記述に関して CWB の記述を FLEC の内部データ形式にまで変換でき、前記の変換方法の妥当性を確認した。また、現時点ではデータ入力部の制限で等価性検証の難しいデータの扱いや、検証仕様や補助情報の抽出方法など、今後更に検討が必要な項目も明らかになった。引き続き、これらのデータの等価性検証やその他の多くのデータを対象とする実データ評価を進め、検証技術の実用化に向けた実証を進めていく予定である。

§4. 成果発表等

(4－1) 原著論文発表

●論文詳細情報

1. B. Alizadeh, M. Mirzaei, and M. Fujita, “Coverage Driven High Level Test Generation using a Polynomial Model of Sequential Circuits,” IEEE Transactions on Computer-Aided-Design of Integrated Circuits and Systems, Vol. 29, No. 5, pp. 737-748, May 2010.(DOI:10.1109/TCAD.2010.2043571)
2. B. Alizadeh and M. Fujita, “Modular Data-path Optimization and Verification Based on Modular-HED,” IEEE Transactions on Computer-Aided-Design of Integrated Circuits and Systems, Vol. 29, No. 9, pp. 1422-1435, Sep. 2010. (DOI:10.1109/TCAD.2010.2059271)
3. H. Yoshida and M. Fujita, “Performance-Constrained Transistor Sizing for Different Cell Count Minimization,” IPSJ Journal of Information Processing, Vol. 18, pp. 252-262, Dec. 2010.(DOI:10.2197/ipsjjip.18.252)
4. H. Yoshida and M. Fujita, “Exact Minimum Factoring of Incompletely Specified Logic Functions via Quantified Boolean Satisfiability,” IPSJ Transactions on System LSI Design Methodology, Vol. 4, pp. 70-79, Feb. 2011.(DOI:10.2197/ipsjtsldm.4.70)
5. Y. Lee, T. Matsumoto, and M. Fujita, “An Automatic Method of Mapping I/O Sequences of Chip Execution onto High-level Design for Post- Silicon Debugging,” IEICE Transactions on Fundamentals of Fundamentals of Electronics, Communications and Computer Sciences. (in press)
6. Y. Lee, T. Matsumoto, and M. Fujita, “Generation of I/O Sequences for a High-level Design from Those in Post-silicon for Efficient Post-silicon Debugging,” Proc. of 28th IEEE International Conference on Computer Design, pp. 402-408, Oct. 2010. (DOI:10.1109/ICCD.2010.5647681)
7. R. Krishnamoorthy, K. Varadarajan, G. Garga, M. Alle, R. Narayan, S.K. Nandy, and M. Fujita, “Towards Minimizing Reconfiguration Overhead in Dynamically Reconfigurable Processors: REDEFINE as a case study,” Proc. of the 2010 International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES ’10), pp. 77-86, Oct. 2010.(DOI:10.1145/1878921.1878935)
8. F. Haedicke, B. Alizadeh, G. Fey, M. Fujita, and R. Drechsler, “A Polynomial Datapath Optimization using Constraint Solving and Formal Modeling,” Proc. of the 2010 International Conference on Computer-Aided Design, pp. 756-761, Nov. 2010. (DOI:10.1109/ICCAD.2010.5654279)
9. Ryota Shioya, Kazuo Horio, Masahiro Goshima, and Shuichi Sakai: Register Cache System not for Latency Reduction Purpose, IEEE Int'l Symp. on Microarchitecture (MICRO-43), pp. 301—312 (2010).(DOI: 10.1109/MICRO.2010.43)
10. B. Alizadeh and M. Fujita, “A Debugging Method for Repairing Post-Silicon Bugs of High Performance Processors in the Fields,” Proc. of the 2010 International Conference on Field-Programmable Technology, pp. 328-331, Dec. 2010.

(DOI:10.1109/FPT.2010.5681434)
11. Y. Lee, T. Matsumoto, and M. Fujita, “On-chip Dynamic Signal Sequence Slicing for Efficient Post-Silicon Debugging,” Proc. of Asia and South Pacific Design Automation Conference, pp. 719-724, Jan. 2011.(DOI:10.1109/ASPDAC.2011.5722280)
12. 高前田伸也, 佐藤真平, 藤枝直輝, 三好健文, 吉瀬謙二:メニーコアアーキテクチャの HW 評価環境 ScalableCore システム, 情報処理学会論文誌コンピューティングシステム, Vol.4, No.1, pp. 24-42 ,Feb. 2011.
13. A.M. Gharehbaghi, M. Fujita, “Global Transaction Ordering in Network-on-Chips for Post-Silicon Validation”, International Symposium on Quality Electronic Design (ISQED’11), pp. 284-289, Santa Clara, California, USA, Mar. 2011.
(DOI はまだ不明)
14. B. Alizadeh, and M. Fujita, “Debugging and Optimizing High Performance Superscalar Out-of-Order Processors Using Formal Verification Techniques”, International Symposium on Quality Electronic Design (ISQED), 2011, pp. 297-302.
(DOI はまだ不明)
15. R. Krishnamoorthy, K. Varadarajan, M. Fujita, S.K. Nandy, M. Alle, and R. Narayan, “Dataflow Graph Partitioning for Optimal Spatio-Temporal Computation on a Coarse Grained Reconfigurable Architecture,” Proc. 7th International Symposium on Applied Reconfigurable Computing (ARC 2011), pp. 125-132, March 2011.
(DOI:10.1007/978-3-642-19475-7_15)

(4－2)知財出願

①平成22年度特許出願件数(国内　1 件)
②CREST 研究期間累積件数(国内　4 件)