Vol.2012-SLDM-155 No.4
Vol.2012-EMB-24 No.4
2012/3/2

RDR アーキテクチャを対象とした 部分 2 重化フォールトセキュア高位合成手法

田中 翔[†1] 柳澤政生[†2] 戸川 望[†1]

半導体の微細化技術の向上に伴い，ソフトエラーによる信頼性低下が問題となっている．そのため，LSI にエラー検出機能を組み込むフォールトセキュア設計の必要性が高まっている．一方，微細化技術の向上によりゲート遅延より配線遅延が支配的となったため，高位合成段階で配線遅延を予測する必要が生じている．配線長が不定である従来のレジスタ集中型アーキテクチャに対し，レジスタをチップ内に均等に配置することで配線長を一定とする RDR アーキテクチャが提案されている．本稿では RDR アーキテクチャを対象とした，部分 2 重化によるフォールトセキュア高位合成手法を提案する．提案手法では入力 CDFG の演算ノードを一部 2 重化することで，レイテンシ制約内で信頼性を最大化する．RDR アーキテクチャで生じる空き領域をフォールトセキュア設計に利用することで面積効率を向上させると同時に，2 重化可能な演算ノード数を増加させる．続いて，挿入比較ノード数を最小化するスケジューリング・バインディングを行うことで余分な演算器動作を抑制し，信頼性向上を図る．計算機実験により提案手法は，フォールトセキュア設計を利用しない手法と比して最大 57% 信頼性を向上させるフォールトセキュア高位合成が可能であることを確認した．

Partial Redundant Fault Secure High Level Synthesis for RDR Architecture

Sho Tanaka,[†1] Masao Yanagisawa[†2]
and Nozomu Togawa[†1]

As device feature size decreases, the reliability improvement against soft errors becomes quite necessary. A fault-secure system, in which concurrent error detection is realized, is one of the solutions to this problem. On the other hand, the average interconnect delay exceeds the gate delay which leads to the timing closure problem. By using *regular-distributed-register architecture (RDR architecture)*, we can estimate interconnection delays very accurately and influence of their interconnect can be much reduced even in the behavioral level. In this paper, we propose a partial redundant fault-secure high-level synthesis algorithm for an RDR architecture. In fault-secure high-level synthesis, a re-computation CDFG a part of normal-computation CDFG must be scheduled and bound to functional units. Firstly, our algorithm re-uses vacant areas on RDR islands to allocate new function units additionally for the re-computation CDFG.Secondly, we propose a scheduling algorithm which minimize the number of insert comparator nodes. We show the effectiveness of the proposed algorithm through experimental results. Our algorithm reduces the soft error rate by an average of 57% compared with the non fault-secure approach.

1. はじめに

半導体の微細化技術の向上により，LSI の面積削減，演算速度の向上，低消費電力化を実現することが期待できる[2]．しかし，微細化によりトランジスタの閾値電圧が低下し，ソフトエラーと呼ばれる一時的なビット反転エラーが起きる可能性が高くなる[3),15),16)]．ソフトエラーによる信頼性低下（フォールト）を解決する方法として，LSI にエラー検出機能を組み込むフォールトセキュア設計[1),7),12)–14),18),19),21)]が不可欠である．メモリ回路におけるソフトエラーは古くから発生率が高く問題となっておりエラー訂正符号（ECC）などの低コストで有効な対策が提案されている[11),17)]．一方，論理回路におけるソフトエラーはマスキング効果[9),10)]によりソフトエラーの影響が隠されることや，回路構造が複雑であることから対策方法が確立されていない．論理回路におけるソフトエラーは配線幅が小さくなるにつれ発生率が高くなるため，今後論理回路におけるソフトエラー対策が必須となることは明らかである．以降，本稿では論理回路におけるソフトエラー対策に着目する．さらに，半導体の微細化技術の向上に伴い大規模化，複雑化している回路を短期間で誤りなく設計するための技術として高位レベル合成が注目されており，フォールトセキュア設計も高位レベルで取り入れることが望ましい．

高位レベルフォールトセキュア設計は同演算を 2 回繰り返し，結果を比較することでソフトエラーを検出する．本来計算すべき演算を通常計算，エラー検出用の演算を再計算と呼ぶ．多数決を用いないため，これ自体ではエラーの訂正ができない．しかし，エラー発生時に演算を再実行する処理等を組み込むことでエラー訂正も実現可能である．同演算を 2 回繰り返し，結果を比較する手法に関する報告に，以下のものがある．

(1) 演算処理を完全に 2 重化する手法
(2) 演算処理を部分的に 2 重化する手法

(1) に関する報告として[1)] や[7)] がある．[1)] や[7)] はコントロールデータフローグラフ（CDFG）を入力とし，通常計算 CDFG と共に，再計算 CDFG をスケジューリング・バインディングすることに基づく．[1)] は通常計算 CDFG と再計算 CDFG に対し FDS（Force-directed scheduling）により必要演算器数を最小化することを目的としている．しかし，再計算 CDFG に変更を加えないため，面積や時間のオーバーヘッドが大きくなる．[7)] は再計算 CDFG のエッジを切断し，部分的に通常計算 CDFG の計算結果を用いることにより小さいコントロールステップ数でのスケジューリング・バインディングを目的としている．完全 2 重化はエラー検出能力が高いが，時間オーバーヘッドが 1.5 倍程度と大きい

†1 早稲田大学大学院基幹理工学研究科情報理工学専攻
Dept. of Computer Science and Engineering, Waseda University.
†2 早稲田大学大学院基幹理工学研究科電子光システム学専攻
Dept. of Electronic and Photonic Systems, Waseda University.

Vol.2012-SLDM-155 No.4
Vol.2012-EMB-24 No.4
2012/3/2

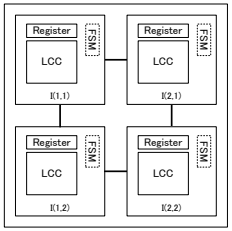

図 1 RDR アーキテクチャ.

問題がある[19]．(2) に関する報告として[8] がある.[8] では時間オーバーヘッドを許さないとき，資源制約と部分 2 重化によるエラー出力確率の関係について報告している．部分的に 2 重化することで資源制約下においても時間オーバーヘッドなく信頼性を向上できる．以降，本稿では (2) によるフォールトセキュア設計に着目する．

半導体の微細化技術の向上に伴うもう 1 つの問題として，配線による遅延がゲート遅延に対し相対的に増加していることが挙げられる．今後もこの傾向は継続すると予想されるため，高位合成の段階においても，配線遅延を考慮する必要がある．配線遅延を考慮した高位合成として，RDR（Regular Distributed Register）アーキテクチャ[4)–6),20)] を対象とした高位合成手法が報告されている．このアーキテクチャはチップを均一の大きさの島に分割する．各島は LCC（Local Computational Cluster）と呼ばれる演算器群と，ローカルレジスタ，FSM を持つ．RDR アーキテクチャの例を図 1 に示す．演算器とレジスタを隣接して配置するため，演算器とレジスタ間の配線長が短くなり配線遅延が小さくなる．島同士の距離が一定であるため高位レベルでの配線遅延の予測が容易である．

RDR アーキテクチャはチップを一定の大きさに分割するので，面積の空き領域が発生する可能性がある．この空き領域をフォールトセキュア設計の再計算に利用することで，面積の増加がなくフォールトセキュア設計が期待できる．しかし，RDR アーキテクチャを対象とした部分 2 重化によるフォールトセキュア設計の報告はまだない．

以上の議論のもと，本稿では RDR アーキテクチャを対象とした部分 2 重化によるフォールトセキュア高位合成手法を提案する．まず，RDR アーキテクチャで生じる空き領域をフォールトセキュア設計に利用することで，2 重化可能な演算ノード数を増加させる．次に，再計算 CDFG のスケジューリング・バインディングは挿入比較ノード数を最小化することにより余分な演算器動作を抑制し，信頼性向上を図る．

提案手法を計算機上に実装し，従来手法と比較した．実験の結果，提案手法はフォールトセキュア設計を利用しない手法と比して最大 57% コントロールステップ数を削減するフォールトセキュア高位合成が可能であることを確認した．

2. 問題の定式化

通常計算 CDFG $G = (V, E)$ は有向グラフで表現される．ノード集合 V は演算ノードの集合と，分岐制御を表すフォーク・ジョインノードの集合からなる．エッジ集合 E は，データフローエッジ集合と，コントロールフローエッジ集合からなる．通常計算 CDFG G の複製を再計算 CDFG と呼び，$G_R = (V_R, E_R)$ とする．

通常計算 CDFG のスケジューリング・バインディングをもとに再計算 CDFG を次の (1)〜(6) を満足するようにスケジューリング・バインディングする[1),7),19)]．(1)〜(5) の条件を全て満足することをフォールトセキュア条件と呼ぶ.

(1) 通常計算 CDFG のスケジューリングとバインディングは変更しない.
(2) 通常計算の結果と再計算の結果を比較する.
(3) 再計算 CDFG のデータフローエッジ $e \in E_d$ を切断することが可能である.
(4) 入力エッジを切断された再計算 CDFG の演算ノード $n' \in V_R$ の入力に，n' に対応する通常計算 CDFG の演算ノード $n \in V$ への入力を利用してもよい.
(5) 切断された再計算 CDFG のデータフローエッジエッジ $e' \in E'_d$ に対し，対応する通常計算 CDFG のデータフローエッジエッジ $e \in E_d$ と比較する.

RDR アーキテクチャは，自然数 N，M に対しチップを $N \times M$ の 2 元配列の島に分割する．$1 \leq x \leq N$，$1 \leq y \leq M$ として，$I(x, y)$ を座標 (x, y) 上の島とする．各島は正方形とする．演算器 fu はいずれかの島 $I(x, y)$ に配置され，遅延時間 d_{fu} を持つ．すべての島 $I(x, y)$ はそれぞれローカルレジスタ $R(I(x, y))$ を持つ．2 つの島 $i_1 = I(x_1, y_1)$，$i_2 = I(x_2, y_2)$ 間の距離は一定である．配線遅延は距離の 2 乗に比例するため，配線遅延係数を C_d としたとき i_1，i_2 間の配線遅延時間を $D_c(i_i, i_2) = C_d \times (|x_1 - x_2| + |y_1 - y_2|)^2$ とする．$i_1 = I(x_1, y_1)$ に配置された演算器を fu_1 とする．1 コントロールステップで fu_1 が演算を実行した結果を島 $i_2 = I(x_2, y_2)$ に配置されたローカルレジスタ $R(i_2)$ に格納することを考える．クロック周期を T_{clk} とする．$d_{fu_1} \leq T_{clk}$ とする．$T_{clk} \geq D_c(i_1, i_2) + d_{fu_1}$ のとき，fu_1 の演算実行と同じコントロールステップで $R(i_2)$ へデータを格納する．一方，$T_{clk} < D_c(i_1, i_2) + d_{fu_1}$ のとき，まず 1 コントロールステップ目では fu_1 の実行結果を島 i_1 のローカルレジスタ $R(i_1)$ に格納する．次コントロールステップ以降，$\lceil \frac{D_c(i_i, i_2)}{T_{clk}} \rceil$ コントロールステップかけてレジスタ $R(i_1)$ からレジスタ $R(i_2)$ へデータを転送する．$d_{fu_1} > T_{clk}$ の場合も同様に，配線遅延時間が大きい場合にはデータ転送のためだけにコントロールステップを使用する．

演算器 fu のコストを c_{fu} とする．RDR アーキテクチャの全ての島は，配置可能な演算器の容量制約 C を持つ．島 $i = I(x, y)$ に配置されている演算器の集合を $F(i)$ とする．すなわち，任意の島 i について $C \geq \sum_{fu \in F(i)} c_{fu}$ が成立する必要がある．換言すれば，ある演算器 fu_i について，$c_{fu_i} \geq C - \sum_{fu \in F(i)} c_{fu}$ であれば新しく演算器 fu_i を島 i に配置可能である．

面積 A の RDR アーキテクチャは演算器 A_{fu}，マルチプレクサ A_{mux}，レジスタ A_{reg}，コントローラ A_{cont}，空き領域 A_{vacant} から構成され，$A = A_{fu} + A_{mux} + A_{reg} + A_{cont} + A_{vacant}$ と表わすことができる．

部分 2 重化フォールトセキュア設計におけるレイテンシ制約を L とする．ここで，RDR アーキテクチャの動作中にチップ内のいずれかの地点に 1 回宇宙放射線粒子が衝突したと仮定する．このとき生じたソフトエラーが演算結果に出力される確率を求める．構成要素のうち，レジスタ A_{reg} は ECC[11] によりソフトエラー耐性が施されているものとする．空き領域 A_{vacant} は動作していないため，宇宙放射線粒子が衝突してもソフトエラーとして出力されない．

一方，演算器 A_{fu}，マルチプレクサ A_{mux}，コントローラ A_{cont} は動作中に宇宙放射線粒子が衝突するとソフトエラーとして出力される．スケジューリング・バインディングされている演算ノード $n \in V$ に対し，n がバインディングされている演算器を fu_n，面積を $a(fu_n)$，演算器が配置されている島の座標 (x, y) を $i_{fu_n} = (x, y)$ とする．このとき演算器 A_{fu} の動作領域を $active(A_{fu}) = \sum_{n \in V} a(fu_n)$ とする．演算器 fu_n のソフトエラー出力確率はコスト c_{fu_n} に比例する．演算器 fu_n に接続されているマルチプレクサを $stage(fu_n)$ とする．マルチプレクサの面積を a_{mux} とする．このときマルチプレクサ A_{mux} の動作領域を $active(A_{mux}) = a_{mux} \sum_{n \in V} stage(fu_n)$ とする．

以上より，RDR アーキテクチャにおけるソフトエラー出力確率を以下に定義する．

Vol.2012-SLDM-155 No.4
Vol.2012-EMB-24 No.4
2012/3/2

$$P_{error} = \frac{active(A_{fu}) + active(A_{mux}) + A_{cont} \times L}{A \times L}$$

2 重化したノードで処理される演算と，付随するマルチプレクサはエラー検出機能によりソフトエラーが発生しても出力されることがない．つまり，2 重化したノードの演算と，付随するマルチプレクサ動作領域から除外することができる．

以上の定義のもと，RDR アーキテクチャを対象としたフォールトセキュアスケジューリング・バインディング問題を次のように定義する．

定義 1. *RDR* アーキテクチャを対象とした部分 *2* 重化フォールトセキュアスケジューリング・バインディング問題とは，*RDR* アーキテクチャの島数 $N \times M$ と演算器の配置，クロック周期，レイテンシ制約，通常計算 *CDFG* とそのスケジューリング・バインディングが与えられたとき，フォールトセキュア条件を満足しかつソフトエラー出力確率を最小化するよう再計算 *CDFG* をスケジューリング・バインディングすることである．*RDR* アーキテクチャ上の既配置の演算器の配置は変更しないものとする．ただし，容量制約を満足すれば演算器を追加してもよいものとする． □

3. RDR アーキテクチャを対象とした部分 2 重化フォールトセキュア高位合成アルゴリズム

RDR アーキテクチャを対象としたフォールトセキュアスケジューリング・バインディング問題の効率的な解法は，(a)2 重化する演算ノードのスケジューリング・バインディング，(b) 空き領域への演算器配置，の 2 つを考える必要がある．なお，説明を簡単にするため本節では入力は DFG のみを考えるが，入力を CDFG としても同様にフォールトセキュアスケジューリング・バインディング問題を解くことが可能である．そこで，次の方針が挙げられる．

(1) (a)2 重化する演算ノードをスケジューリング・バインディング，次に (b) 空き領域へ演算器を配置する．
(2) (a)2 重化する演算ノードのスケジューリング・バインディングと同時に (b) 空き領域へ演算器を配置する．
(3) (b) 空き領域へ演算器を配置し，次に (a)2 重化する演算ノードをスケジューリング・バインディングする．

(1) は，2 重化する演算ノードをスケジューリング・バインディングする際，(b) で配置される演算器を予測しながらスケジューリング・バインディングする必要があるため最適化が困難である．(2) は局所的な改善は見込めるが，グローバルな改善ができない可能性がある．(3) は空き領域に演算器をあらかじめ配置しておくことで配置情報を把握でき，それに基づくスケジューリング・バインディングが可能である．

以上より (3) の方針をとることにする．(3) の方針に基づき，図 2 に RDR アーキテクチャを対象とするフォールトセキュア高位合成手法を提案する．提案手法は初期化 (Step1)，空き領域への演算器の配置 (Step2)，2 重化する演算ノードのスケジューリング・バインディング (Step3) の 3 ステップからなる．

3.1 初期化 (Step1)

Step1 では Step2 と Step3 の処理を容易化するために，初期値としてフォールトセキュアの 1 つの解を暫定的に求める．Step1 は次の方針をとる．

(1.1) 通常計算 CDFG から再計算 CDFG を複製する．
(1.2) 再計算 CDFG を暫定的にスケジューリング・バインディングする
(1.3) RDR アーキテクチャの島それぞれに比較器を 1 つ配置する．

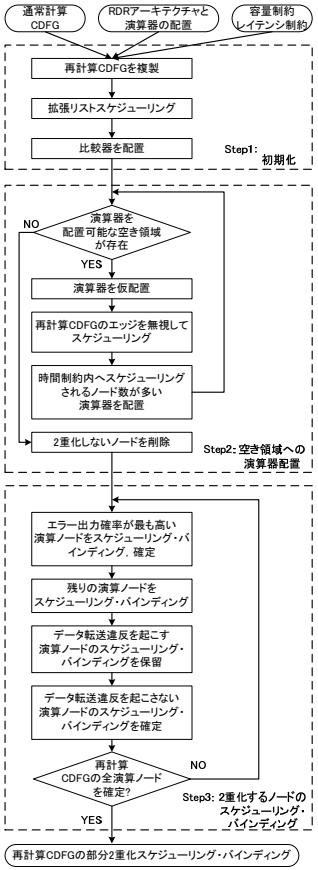

図 2 合成フロー．

(1.1) では通常計算 CDFG $G = (V, E)$ から再計算 CDFG $G_R = (V_R, E_R)$ を複製する．

(1.2) にて再計算 CDFG をフォールトセキュア条件を満足するよう暫定的にスケジューリング・バインディングする．ここで，演算ノードのスケジューリング・バインディングは拡張リストスケジューリング[19] を利用する．拡張リストスケジューリングは，リストスケジューリング手法に対して，RDR アーキテクチャ上に配置された演算器のバインディングを取り入れたものである．(1.3) では RDR アーキテクチャの各島にエラー検出用の比較器を 1 つずつ配置する．

例 1. 図 3(a) の RDR アーキテクチャに対し，(c) のようにスケジューリング・バインディングされた通常計算 DFG が与えられたとする．RDR アーキテクチャの配線遅延は，隣接する島へのデータ転送に 1 コントロールス

Vol.2012-SLDM-155 No.4
Vol.2012-EMB-24 No.4
2012/3/2

表 1 演算器のコスト，面積，遅延．

	コスト	面積 [μm^2]	遅延 [ns]
加算器 (ADD)	1	282	1.32
減算器 (SUB)	1	316	1.33
乗算器 (MUL)	2	4661	2.7

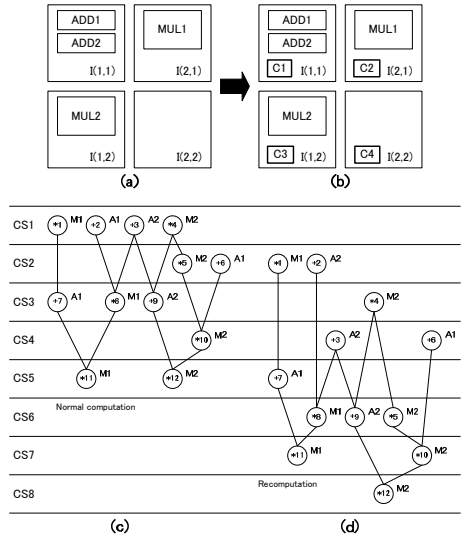

図 3 初期化ステップ．

テップ必要とする．演算器のコストを表 1 に示す．1 つの島の容量制約 C を 2 とする．演算器の遅延は全て 1 コントロールステップとする．レイテンシ制約を $L = 6$ とする．(1.1) より，通常計算 DFG と同じ DFG を再計算 DFG として複製する．(1.2) より，暫定的に再計算 DFG をスケジューリング・バインディングすると図 3(d) が得られる．(1.3) より図 3(b) のように全ての島に 1 つずつ比較器 “C1”，“C2”，“C3”，“C4”を配置する． □

3.2 空き領域への演算器配置 (Step2)

Step2 では RDR アーキテクチャの空き領域に再計算 CDFG 用の演算器を配置する．Step2 の方針を次に示す．

(2.1) 空き領域に演算器を仮配置する．
(2.2) 再計算 CDFG の演算ノードに対し，入力を全て通常計算 CDFG から流用してスケジューリング・バインディングする．
(2.3) レイテンシ制約内にスケジューリングされる演算ノード数が多い演算器を新たに配置する．
(2.4) レイテンシ制約内にスケジューリングされない演算ノードを削除する．

この方針より，より多くの演算ノードを 2 重化でき，信頼性の向上が期待できる．

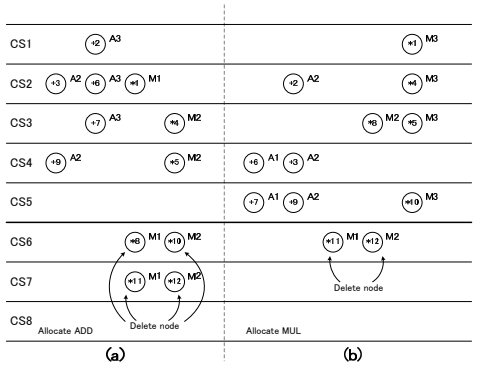

図 4 (a) 加算器と (b) 乗算器を仮配置したときのスケジューリング・バインディング．

(2.1) は空き領域に対し，配置可能な演算器 fu を仮配置する．(2.2) では再計算 CDFG の演算ノードに対し，入力を全て通常計算 CDFG から流用してスケジューリング・バインディングする．まず再計算 CDFG の演算ノード n' に対し，対応する通常計算 CDFG の演算ノードを n とする．演算ノード n に入力を与えているエッジの集合を $E_{in}(n)$ とする．n' に入力を与えるエッジを $E_{in}(n')$ からではなく，$E_{in}(n)$ を流用しスケジューリング・バインディングする．(2.3) では通常計算 CDFG のレイテンシ制約 L に対し，$L-1$ より遅いコントロールステップにスケジューリングされている再計算 CDFG の演算ノード n' の集合を S_R を求める．n' がバインディングされている演算器を fu' とする．fu' のコストを $c_{fu'}$ とし，S_R に対しコストの和 $S_{fu} = \sum_{n' \in S_R} c_{fu'}$ をとる．S_{fu} の最小値を与える演算器 fu を新たに配置する．演算器を配置できる空き領域が無くなるまで (2.1) から (2.3) を繰り返す．

(2.4) では，演算器の配置後もレイテンシ制約内へスケジューリングできない演算ノードを削除する．

例 2. 図 3(b) の空き領域に演算器を配置することを考える．島 $I(2,2)$ が空き領域となっており，容量制約から加算器，乗算器が配置可能である．まず，(2.1) 加算器を仮に配置する．次に (2.2) でスケジューリング・バインディングし，結果を図 4(a) に示す．このとき $L-1=5$ より遅いコントロールステップにスケジューリングされている演算ノードは $S_R =$ “$*8$”, “$*10$”, “$*11$”, “$*12$” である．このとき $S_+ = 8$ となる．

同様に乗算器を配置したときのスケジューリング・バインディング結果を図 4(b) に示す．このとき $S_R =$ “$*11$”, “$*12$” となり，$S_* = 4$ となる．$S_* < S_+$ なので乗算器を新たに配置する．容量制約より他に演算器を配置することはできないため，空き領域への演算器配置ステップを終了する．最後に，CS5 以内にスケジューリングされなかった演算ノード “$*11$”, “$*12$” を削除する． □

3.3 部分 2 重化スケジューリング・バインディング (Step3)

再計算 CDFG の一部のエッジを切断することで，より早いコントロールステップへスケジューリング・バインディングが可能になる．しかし切断したエッジに対し比較ノードを挿入する必要があるため，比較ノードにより信頼性が低下する可能性がある．そこで，挿入する比較ノード数が最小になるようスケジューリング・バインディングし，信頼性低下を防ぐ．部分 2 重化スケジューリング・バインディングの方針を次に示す．

(3.1) エラー出力確率が最も高い演算ノードをスケジューリング・バインディングする．

Vol.2012-SLDM-155 No.4
Vol.2012-EMB-24 No.4
2012/3/2

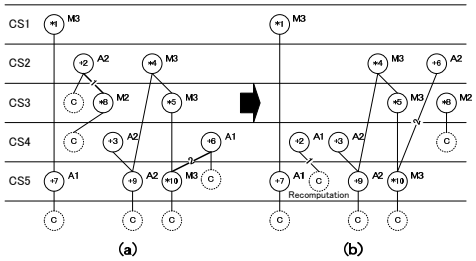

図 5 再計算 CDFG の部分 2 重化スケジューリング・バインディング.

(3.2) エッジ切断数が最小になるようその他の演算ノードをスケジューリング・バインディングする.
(3.3) 比較ノードを挿入，スケジューリング・バインディングする.

(3.1) でエラー出力確率が最も高い演算ノードのスケジューリング・バインディングを基準として確定し，その他の演算ノードに焦点を絞った上でデータ転送違反が無いようスケジューリング・バインディングを繰り返す．その結果，冗長なエッジ切断を回避することが期待できる．

(3.1) はまず再計算 CDFG の演算ノード n' に対し，バインディングされている演算器 fu' のコスト $c_{fu'}$ が大きいノードから順にスケジューリング・バインディングする．(3.1) の処理を次に示す．n' に対応する通常計算 CDFG の演算ノードを n とする．演算ノード n に入力を与えているエッジの集合を $E_{in}(n)$ とする．n' に入力を与えるエッジを $E_{in}(n')$ からではなく，$E_{in}(n)$ を利用したとき，n' がスケジューリング・バインディング可能となる最小のコントロールステップを $s(n')$ とする．$cs(n') < s(n')$ のとき，n' を $s(n')$ にスケジューリング・バインディングする．このスケジューリング・バインディングを確定し，今後変更しないものとする．

(3.2) では，(3.1) でスケジューリング・バインディングされていない演算ノード n' に対し，通常計算 CDFG のエッジ $E_{in}(n)$ を利用せず，$E_{in}(n')$ を利用してスケジューリング・バインディングする．n' に入力を与える 1 つのエッジを $e' \in E_{in}(n')$ としたとき，n' の入力が e' からではデータ転送が間に合わない場合，n' のスケジューリング・バインディングを保留する．データ転送違反が発生しないとき，n' のスケジューリング・バインディングを確定する．全ての演算ノードが確定するまで，(3.1), (3.2) を繰り返す.

(3.3) ではフォールトセキュア条件を満足するよう比較ノードを挿入，スケジューリング・バインディングする．比較ノードのスケジューリングに競合が起きレイテンシ制約違反を起こす場合，次の処理を行う.

(3.3.1) 競合する比較ノードに接続されている演算ノードを削除する.
(3.3.2) 比較ノードを挿入し直したとき，必要比較ノード数をノード削除前と比べる.
(3.3.3) 必要比較ノード数が最も少なくなる演算ノードを削除する．同数の場合，ノード番号が小さい演算ノードを削除する.

例 3. RDR アーキテクチャ上における演算器の配置をもとに，再計算 CDFG をスケジューリング・バインディングする．まず，最もエラー出力確率が高い乗算ノード “$*1$”, “$*4$”, “$*5$”, “$*8$”, “$*10$” をスケジューリング・バインディングする．次に加算ノードをスケジューリング・バインディングする．結果を図 5(a) に示す．このとき演算ノード “+2”，“+6” にデータ転送違反が発生している．そこでまず，演算ノード “+2” のスケジューリング・バインディングを保留する.

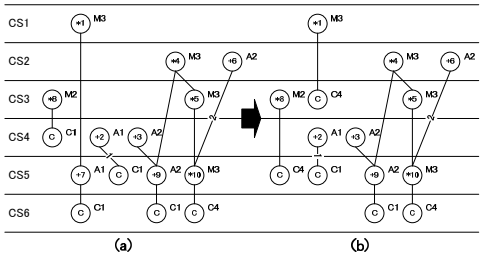

図 6 提案手法の出力.

次に演算ノード “+6” に着目する．保留した演算ノード “+2” により空いた CS2 にスケジューリング・バインディングすると，データ転送違反が解消されるため，CS2 にスケジューリング・バインディングを確定する．“+3”, “+7”, “*9” に関してはデータ転送違反がないためスケジューリング・バインディングを確定する．確定されていない演算ノード “+2” に対し，(3.1) より CS4 にスケジューリング・バインディングを確定する．結果，図 5(3) のスケジューリング・バインディングが得られる.

比較ノードを挿入すると，図 6(a) より CS6 で比較器 “C1” の競合が起きる．そこで，“C1” に接続されている演算ノード “+7” と “+9” に着目する．“+7”，“+9” 共に，削除したと仮定した場合比較ノード数の増減がないため，(3.3.3) より “+7” を削除する．提案手法の結果を図 6(b) に示す．演算ノードのスケジューリング・バインディングの変更により必要比較ノード数を削減できる.

□

4. 計算機実験結果

提案手法を C++言語を用いて計算機上に実装した．計算機実験環境は，CPU が Intel Xeon 3.0GHz, メモリ容量が 4GB である．対象アプリケーションとして 7 次 FIR フィルタ (ノード数 75)，DCT(ノード数 48)，EWF3(ノード数 102) を用いた．実験で用いた演算器のコスト，遅延を表 1 に示す．各演算器は 16bit 幅と仮定し, クロック周期を 3ns とする．配線遅延係数を $C_d = 1ns$ とする．通常計算 CDFG のスケジューリング・バインディングと，RDR アーキテクチャ上の演算器の初期配置は[4] の手法を用いて得た.

提案手法とフォールトセキュア設計を行わない手法についてソフトエラー出力確率を比較した．実験結果を表 2 に示す．表 2 の 1 列目に入力として与えられた CDFG，2 列目に RDR アーキテクチャの島数，3 列目に容量制約を表す．例えば，dct の CDFG に対して島数 2×3 のとき，この RDR アーキテクチャは 6 つの島を持つ．各島の容量制約は 2 であるため，表 1 より 1 つの島には乗算器が 1 つまたはその他の演算器を 2 つ配置可能である．表 2 の 4 列目から 6 列目はそれぞれ実行した手法，必要コントロールステップ数，島の面積を表す．7 列目から 9 列目はそれぞれ，コントローラの面積，マルチプレクサの面積，レジスタの面積を示している．10 列目から 12 列目はそれぞれ，動作しているFU，動作しているマルチプレクサ，ソフトエラー出力確率を表わす．11 行目にプログラムの実行時間を示す．実験結果より提案手法は，フォールトセキュアなしの手法と比して平均 35%，最大 57% 信頼性が向上した.

5. おわりに

本稿では RDR アーキテクチャを対象とした部分 2 重化フォールトセキュア高位合成手法を提案し

表 2 Experimental Results.

Input CDFG	#Islands $N \times M$	Capacity constraint	Algorithm	CS	Island Area[μm^2]	Controller Area[μm^2]	MUX Area[μm^2]	Reg Area[μm^2]	Active FU	Active MUX	Soft Error Rate[%]	CPU time [sec]
dct	2×3	2	Normal CDFG	13	8737	2056	7280	7488	78736	26880	16.47	107.9
			Ours	14	10539	3431	13664	12096	6699	7840	7.07	129.9
ewf3	2×2	2	Normal CDFG	53	10297	2524	6272	3456	133860	72800	9.14	74.8
			Ours	54	16604	5321	21728	8064	7329	6272	8.02	99.1
fir	2×3	2	Normal CDFG	24	7089	2464	4032	7776	157048	36624	11.66	109.4
			Ours	25	11398	5547	22064	17856	20032	3584	7.41	135.6

た．通常計算 CDFG を部分的に 2 重化することで，時間オーバーヘッドなく信頼性を向上させている．計算機実験により提案手法は，フォールトセキュア設計を利用しない手法と比して最大 57% 信頼性が向上することを確認した．今後の課題として，マルチプレクサ数の増大を抑制するフォールトセキュア設計が挙げられる．

謝　　辞

本研究の一部は，科学研究費補助金 (課題番号 22300019) ならびに KDDI 財団調査研究助成による．

参 考 文 献

1) A. Antola, V. Piuri, and M. Sami, “High-level synthesis of data paths with concurrent error detection,” in *Proceedings of IEEE Int. Symp. Defect Fault Tolerance VLSI Systems.*, pp. 292–300, 1998.
2) C. Constantinescu, “Impact of deep submicron technology on dependability of VLSI circuits,” in *Proceedings of Dependable Systems and Networks*, pp. 205–209, 2002.
3) D. Alexandrescu, L. Anghel, and M. Nicolaidis, “New methods for evaluating the impact of single event transients in VDSM ICs,” in *Proc. of Defect and Fault Tolerance in VLSI Systems*, pp. 99–107, 2002.
4) J. Cong, Y. Fan, X. Yang, and Z. Zhang, “Architecture and synthesis for multi-cycle communication,” in *Proceedings of 2003 International Symposium on Physical Design*, pp. 190–196, Apr. 2003.
5) J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, “Architectural synthesis integrated with global placement for multi-cycle communication,” in *Proceedings of International Conference on Computer Aided Design*, pp. 536–543, Nov. 2003.
6) J. Cong, Y. Fan, and Z. Zhang, “Architecture-level synthesis for automatic interconnect pipelining,” in *Proceedings of Design Automation Conference*, pp. 602–607, Jun. 2004.
7) K. Wu and R. Karri, “Fault secure datapath synthesis using hybrid time and hardware redundancy,” *IEEE Trans. computer-aided design of integrated circuits and systems*, vol.23, no.10, pp. 1476–1484, Oct. 2004.
8) K. Mohanram and N. A. Touba, “Cost-effective approach for reducing soft error failure rate in logic circuits,” in *Proc. Intl. Test Conference*, pp. 893–901, 2003.
9) M. J. B. Diaz, J. J. Chico, A. J. Acosta, M. Valencia, and J. L. Huertas, “Logical modelling of delay degradation effect in static CMOS gates,” *IEEE Proc-Circuits, Devices and Systems*, vol.147, pp107–117, April 2000.
10) P. Shicakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, “Modeling the effect of technology trends on the soft error rate of combinational logic,” in *Proceedings International Conference on Dependable System and Networks*, pp. 389–398, 2002.
11) R. Baumann, “ The impact of technology scaling on soft-error rate performance and limits to the efficacy of error correction, ” in *Proceedings Int’l Electron Devices Meeting*, pp. 329–332 ,2002.
12) R. Karri and A. Orailoglu, “High-level synthesis of fault-secure microarchitectures,” in *Proc. of 30th international Design Automation Conference*, pp. 429–433, 1993.
13) R. Karri and A. Orailoglu, “Time-constrained scheduling during high-level synthesis of fault-secure VLSI digital signal processors,” *IEEE Trans. on Reliability.*, vol.45, pp. 404–412, Sept. 1996.
14) R. Karri and B. Iyer, “Introspection: A register transfer level technique for cocurrent error detection and diagnosis in data dominated designs,” *ACM Trans. on Design Automation of Electronic Systems.*, Vol. 6, No. 4, pp. 501–515, Oct. 2001.
15) R. R. Rao, K. Chopra, D. Blaauw, and D. Sylvester, “An efficient static algorithm for computing the soft error rates of combinational circuits,” in *Proc. DATE Conf.* , pp. 164–169, Mar. 2006.
16) S. Buchner, M. Baze, D. Brown, D. McMorrow, and J. Melinger, “Comparison of error rates in combinational and sequential logic,” *IEEE Trans. Nuclear Science*, vol.44, pp. 2209–2216, Dec. 1997.
17) S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, “Robust system design with built-In soft-error resilience,” *Computer*, vol. 38, no.2, pp. 43–52, Feb. 2005.
18) S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, and Yuan Xie, “Reliability-centric high-level synthesis,” in *Proceedings of the conference on Design, Automation and Test in Europe*, vol.2, pp. 1258–1263, 2005.
19) S. Tanaka, M. Yanagisawa, T. Ohtsuki, and N. Togawa, “A fault-secure high-level synthesis algorithm for RDR architectures,” *IPSJ Trans. on System LSI Design Methodology.*, Vol. 4, pp.150–165, Aug. 2011.
20) W. S. Huang, Y. R. Hong, J. D. Huang, and Y. S. Huang, “A multicycle communication architecture and synthesis flow for global interconnect resource sharing,” in *Proc. of Asia and South Pacific Design Automation Conference*, pp. 16–21, Jan. 2008.
21) Y. Liu and K. Wu, “Towards cool and reliable digital systems:RT level CED techniques with runtime adaptability,” *Conference on IEEE International Computer Design*, pp.528–533, 2010.