ISSN 2186-5647

データパスの単一サイクル間欠故障に対する誤り検出のための演算器バインディング法の評価

日大生産工(学部)　○増田 哲也　日大生産工(院)　西間木 淳　日大生産工　細川利典

1. はじめに

近年，半導体集積技術の発達に伴い，設計される大規模集積回路(Large Scale Integrated circuits：LSI)の高機能化，高集積化が急速に進展している．これにより，LSI の設計コストの増大が問題視されている．従来，LSI 設計において，ハードウェア記述言語(Hardware Description Language：HDL)でレジスタ転送レベル(Register Transfer Level：RTL)の回路を記述し，論理合成ツールを用いることでネットリストを生成する方法が主流であった．しかしながら RTL 設計では，回路の構造情報を設計者が詳細に定義する必要があるため，LSI の複雑度が増加するにつれて設計が困難となる[1]．この問題を解決するための手法として，RTL と比べて高い抽象度で記述される動作レベルの回路記述から，その動作を実現する RTL 回路記述を自動生成する動作合成[1][2]が提案されている．動作記述には回路で実現したい機能動作のみを記述すればよいため， RTL 回路記述に対して記述量が少なく，設計生産性に優れる[1].

また，近年の LSI システムは，高性能，高集積化により様々な用途で使用されており，LSI システムの信頼性の実現・確保が重要な課題である．LSI の信頼性を損なう原因として，ハードエラーとソフトエラー[3][4]がある．ハードエラーは LSI 内部の物理的欠陥が原因で起こる永続的な誤りであり，ソフトエラーは地上まで到達する二次宇宙線が LSI のメモリセルに衝突した際に，メモリセルに蓄えられた電荷量の値を乱すことが原因で一時的に発生する誤りである．通常，LSI は製造後，良品と不良品を選別するためのテストを行い，良品と判定されたもののみを市場へ出荷する．しかしながら，ソフトエラーによる LSI の一時的な誤りは，正常と判定され出荷された LSI に対しても発生する可能性があるため，通常の LSI テストによってソフトエラーに対する信頼性を確保することが困難である．

本論文では，ソフトエラーに対する高信頼システムを実現するために，データパス多重化によるオンラインテスタビリティの向上を目的とし，さらに動作記述等の高位の情報を利用することで，小面積，高性能なシステムを効率よく設計する高位合成手法を実装し，評価する．

本論文では演算器に起こるソフトエラーを対象とした 1 サイクル単一ソフトエラーに耐性をもつオンラインテスト可能なデータパス合成法とその評価について述べる．ただし，本論文ではソフトエラー誤り検出可能なデータパスをソフトエラー耐性を持ったデータパスと呼ぶ．

2. 動作合成

動作合成とは，回路の動作記述からその動作を実現する RTL 回路を生成する技術である[1]．図 1 に動作合成の流れを示す．

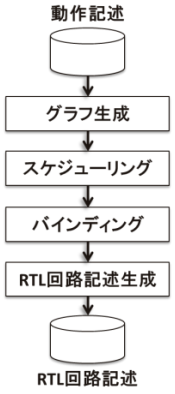

図 1. 動作合成の流れ

2-1. グラフ生成

グラフ生成では入力として与えられた動作記述より，動作の流れを表したコントロールデータフローグラフ(Control Data Flow Graph：CDFG)[1]を生成する．動作記述例を図 2 に表す. CDFG の表現方法は複数あるが，コントロールフローグラフ(Control Flow Graph：CFG)[1]とデータフローグラフ(Data Flow Graph：DFG)[1]の 2 種類に分けて表現する方法を利用する．

```
Int main(int a,int b,int c,int d,int e){
    int  p,q,r,s;
    p = a + b – c;
    q = a+d+e;

    if(p > 0){
        r = a * b;
    }
    else{
        r = a* d;
    }

    s = r +q+p;
    return s;
}
```

図 2. 動作記述例

An Evaluation of a Functional Unit Binding Method to Detect Error of a Single Cycle Transient Fault for Data Path Circuits
Tetsuya MASUDA, Jun NISHIMAKI and Toshinori HOSOKAWA

CFG とは，演算処理のまとまりであるブロック処理，if 文などの条件分岐，while 分などのループ処理から構成されるグラフである(図 3)．図 2 の記述では p=a+b-c と q=a+d+e は連続した計算式となり 1 つの演算ブロック S1 としてまとめられる．同様に if 文内の r=a*b は S2，else 文内の r=a*d は S3，s=r+q+p は S4 としてまとめられる．

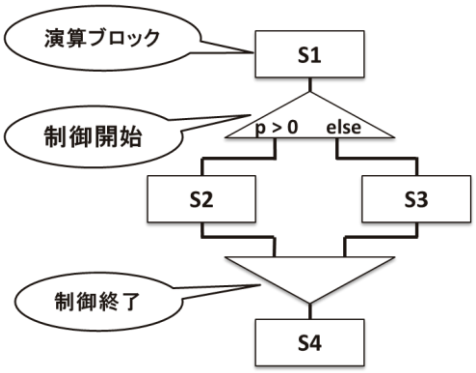

図 3. if 文の CFG 例

DFG とは入力変数，出力変数，内部変数，演算操作から構成され，各演算ブロック内の計算式に対し，変数と演算操作の関係を表したグラフである(図 4)．

2-2. スケジューリング

スケジューリングとは生成された CDFG に対して各演算操作をどの時刻で実行するかを決定する処理である．代表的なスケジューリングアルゴリズムとして，割当て可能な演算操作をできるだけ早い時刻に割当てる ASAP(As Soon As Possible)や，割当て可能な演算操作をできるだけ遅い時刻に割当てる ALAP(As Late As Possible)がある[1]．ここで，スケジューリング済みの DFG を SDFG(Scheduled Data Flow Graph)と呼ぶ．図 4 に示す演算ブロック S1 の DFG のスケジューリング例を図 5 に示す．

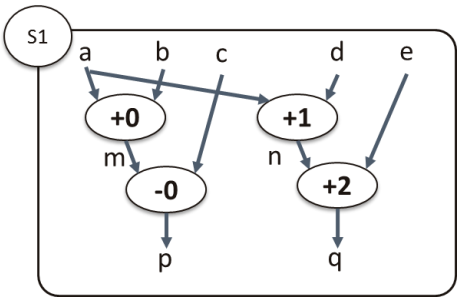

図 4. 演算ブロック S1 の DFG

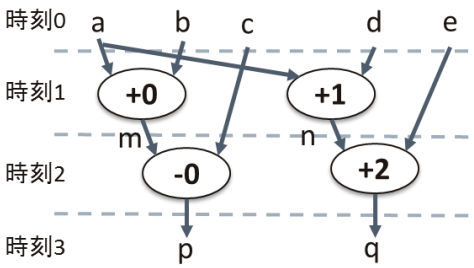

図 5. SDFG 例

2-3 バインディング

バインディングとは SDFG に存在する演算操作および変数に具体的なハードウェア資源(演算器，レジスタ)を割当てる操作である．面積の小さな RTL 回路を合成するためには，演算操作間，変数間で適切な資源の共有が必要となる．バインディングは演算操作に演算器を割当てる演算器バインディングと変数にレジスタを割当てるレジスタバインディングの 2 つのバインディングから成る．

2-3-1 演算器バインディング

演算器バインディングでは SDFG 内の各演算操作に対し，その演算を実行する演算器を割当てる．本論文で提案するソフトエラー耐性をもつデータパス合成処理内で実行される演算器バインディングでは，実行時刻の異なる同種類の演算操作に対して，貪欲に同一演算器を割当てるアルゴリズムを採用している．図 5 に示す演算ブロック S1 の SDFG に対して演算器バインディングを実行した結果を図 6 に示す．

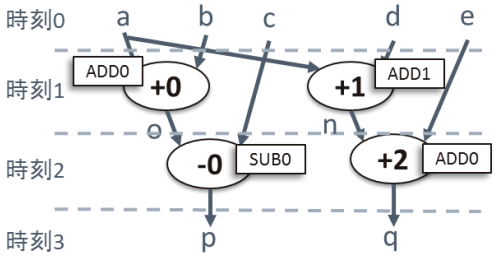

図 6. 演算器バインディング例

2-3-2 レジスタバインディング

レジスタバインディングでは SDFG に存在する各変数にレジスタを割当てる．演算器バインディングと同様，レジスタは同一時刻内で複数の変数を同時に記憶することは不可能である．したがってレジスタ数最小化を指向する場合，各変数のライフタイム[1]を解析する必要がある．ライフタイムとは変数の値の保持が必要となる時刻から，必要がなくなる時刻までの時間をいう．ライフタイムが重ならない変数間でレジスタは共有可能となる．その代表的なアルゴリズムとして，レフトエッジアルゴリズム(Left Edge Algorithm：LEA)[7]が存在する．

本論文ではソフトエラーの影響が演算器にのみに発生すると仮定している．しかしながらソフトエラーがレジスタに影響を及ぼすと仮定した場合，レジスタバインディングについても演算器バインディングと同様にソフトエラー耐性を指向したバインディングを考えることができるが，本論文では対象外とする．

3. 演算器 1 サイクル単一ソフトエラー耐性を指向したデータパス合成法

本章では,ソフトエラー耐性を持つデータパス合成法について述べる．ここで，ソフトエラーとは演算器にのみ発生し，多サイクルに渡って影響は及ばないものとする．さらにソフトエラーによって誤りが生じる期間内ではソフトエラーは複数の演算器では発生しないと仮定する．

3-1. データパス二重化システム

本論文では 1 サイクル単一ソフトエラーに耐性をもつ

データパス合成法は，誤り検出能力をもたせるため，データパス二重化システム[5][6]を基本とする．図 7 に二重化システムの例を示す．

動作記述より生成された DFG に対して，DFG に現れる演算操作と同じ処理を実行する演算操作を新たに追加し，演算器およびレジスタバインディングを行うことで，図 7 に示したような二重化されたデータパスを生成できる．ソフトエラーは同時に複数の演算器に発生せず，かつエラー期間は 1 サイクルのみと仮定しているため，片方のデータパス内の演算器にソフトエラーが発生したとしても，二重化したもう一方の同一演算は正常な動作が保証される．従ってデータパスを二重化し，二つのシステムの出力を比較することで，誤りを検出することができる．しかしながら，単純な DFG の二重化によって生成される RTL 回路の演算器数は，二重化前の RTL 回路の演算器数の 2 倍となり，さらに元の DFG の出力と重複した DFG の出力を比較し，誤りを検出するための比較器の挿入も必要となるため，100%以上のハードウェアオーバーヘッドが発生する．図 7 に図 6 に示す演算ブロック S1 のバインディング済 SDFG に対して単純な二重化を実行した例を示す． 図 7 では二重化前の DFG を org.，二重化した DFG を dup.とし，dup.は org.と同じ動作を実行する．

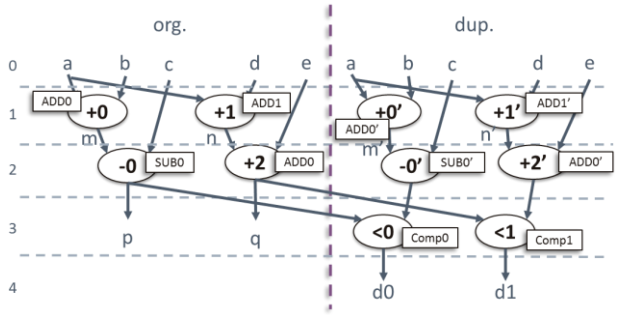

図 7. 単純な DFG 二重化例

3-2. アイドル時間を考慮したデータパス二重化

本節にて，図 7 で示したデータパス二重化に伴うハードウェアオーバーヘッド増加の問題点を解決し，かつソフトエラーに耐性をもたせるためのデータパス二重化手法の代替案として，演算器アイドル時間を考慮した DFG 二重化法を提案する．この手法では二重化前の CDFG に対する演算器バインディング情報を基に，各演算ブロックの DFG の各時刻において演算操作が割当てられていない演算器情報を取得し，元の演算操作に対して新たに追加されたソフトエラー耐性用の演算操作に，アイドル状態となっている演算器のみを使用してバインディングを実行することで二重化を実現する．図 6 に示す演算ブロック S1 のバインディング済み SDFG に対して，各時刻におけるアイドル状態となる演算器情報を図 8 に示す．

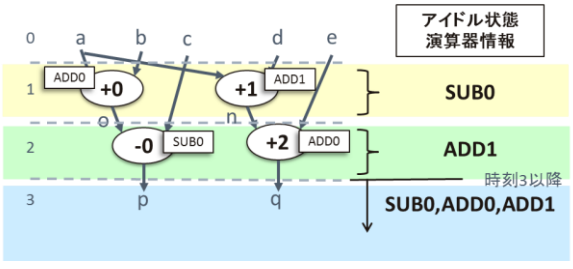

図 8. アイドル状態演算器情報取得例

図 8 の DFG の時刻数は 4 であり，DFG 内で割当てられた演算器は ADD0,ADD1,SUB0 の 3 つである．それらの情報から各時刻でアイドル状態となる演算器情報を取得することができる．

時刻 1 においては演算操作”+0”,”+1”に対してそれぞれ ADD0,ADD1 の演算器が割当てられているため，時刻 1 においてアイドル状態にある演算器は SUB1 である．同様に時刻 2 においてアイドル状態にある演算器は ADD1 である．ここで，演算操作”+0”または”+1”に対する二重化を，演算器 ADD1 を使用して実行する．このように各演算操作に対して追加する同種類の演算操作を，アイドル状態となる演算器が存在するコントロールステップに追加することで，オリジナルのバインディング済み SDFG に対して新たなハードウェア資源を挿入することなく，DFG を二重化し，ソフトエラーに耐性をもったデータパスが合成可能となる．

3-4. データパス重複化アルゴリズム

本節ではソフトエラー耐性付与のためのデータパス重複化アルゴリズムについて述べる．図 9 にアルゴリズムの全体フローを示す．

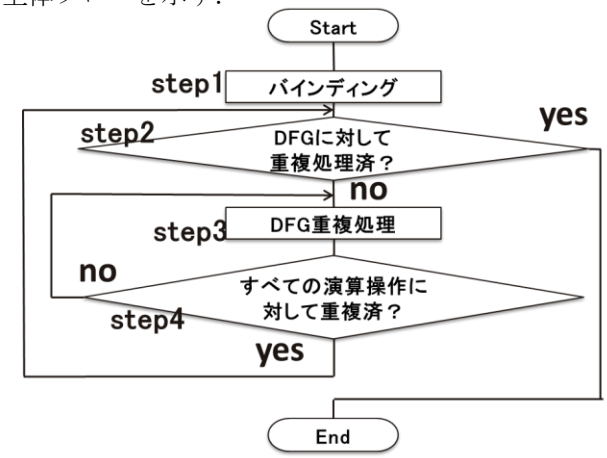

図 9. データパス重複化全体フローチャート

入力となる回路記述が CDFG の場合，CFG に現れるすべての演算ブロックの DFG に対してそれぞれ演算器アイドル時間を考慮した演算操作重複処理を実行することで，CDFG で表現される回路のあらゆる動作において，ソフトエラーに耐性をもったオンラインテスト可能な回路の RTL 記述を合成する．図 2 の動作記述より生成された図 4 の CDFG の演算ブロック S1 に対して，本論文で評価するデータパス二重化演算器バインディングを実行した結果が図 10 である．

演算ブロック S1 の SDFG に対し，演算器バインディング実行後の SDFG を図 6 に示す．図 6 より S1 は加算器 2 つ(ADD0,ADD1)，減算器 1 つ(SUB0)で構成されることがわかる．演算器バインディング後に DFG の各時刻における演算器アイドル状態情報を取得できる．

データパス二重化のため，DFG 内に現れるすべての演算操作に対して同種類の演算操作を追加する必要がある．しかしながら，追加する演算操作は、その演算を実行する演算器がアイドル状態となるコントロールステップにのみ挿入できるものとする．このように二重化をアイドル状態演算器のみを使用して実行することで，新たな演算器を挿入することなく，動作記述からソフトエラーに耐性をもったデータパスを表現する CDFG を生成でき

る．生成された二重化済 CDFG 情報から，RTL 記述を合成する．

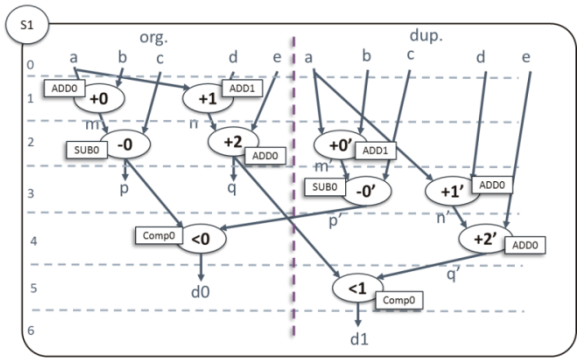

図 10. 重複化済 DFG

4. 実験結果

本論文で提案した手法を用いて，DFG ベースである ex2，DCT，DFCT の 3 個の回路および，CDFG ベースである kim，maha，sehwa の 3 個の回路，合計 6 つの回路の演算操作を重複化し，PICTHY[8]を用いて合成された RTL 回路の性能を評価した．評価項目としては論理合成後の二重化前，単純な二重化後，アイドル状態演算器を利用した二重化後，それぞれの回路面積と，遅延，およびオンラインテスタビリティを対象とした．表 1 に各回路の論理合成後の面積を示す．表 2 に各回路のレイテンシを示す．表 3 に各回路のオンラインテスタビリティを示す．

表 1 より，すべての実験回路において，演算器数を 2 倍にした二重化に対しアイドル状態演算器のみを使用した二重化が小面積を実現できることがわかる．しかしながら遅延においてはすべての実験回路においてアイドル状態演算器のみを使用した二重化が最も長い遅延が発生するという結果であった．十分なアイドル状態の演算器が確保できない回路の場合，二重化によるレイテンシが増大しやすい傾向にあることがわかる．また，ソフトエラーの検出に関しては二つの二重化手法共に 100%の結果が得られた．

表 1. 面積

回路名	面積(Area)		
	重複なし	二重化(double)	二重化(idle)
DCT	6559	12813	12148
DFCT	28678	56371	40498
ex2	25132	49422	27946
Maha	7702	15525	13653
Sehwa	8711	16981	15413
Kim	10940	21216	20501

表 2. 遅延

回路名	遅延(Latency)		
	重複なし	二重化(double)	二重化(idle)
DCT	6	8	11
DFCT	7	10	13
ex2	6	7	9
Maha	18	23	28
Sehwa	17	23	26
Kim	15	18	20

表 3. オンラインテスタビリティ

回路名	オンラインテスタビリティ		
	重複なし	二重化(double)	二重化(idle)
DCT	0%	100%	100%
DFCT	0%	100%	100%
ex2	0%	100%	100%
Maha	0%	100%	100%
Sehwa	0%	100%	100%
Kim	0%	100%	100%

5. おわりに

本論文では，ソフトエラーによる誤りを検出するためのデータパス二重化システムを基本とした 1 サイクル単一ソフトエラーに耐性をもったオンラインテスト可能なデータパス高位合成法を評価した．今後の課題として，多サイクルに渡るソフトエラーに対して耐性をもった回路の合成や誤り訂正機能の追加，およびレジスタに影響が及ぶと仮定したソフトエラーに対して耐性をもたせるレジスタバインディング法の提案，さらにはスケジューリング法の提案．

参考文献

[1] Daniel D. Gajski, Nikil D. Dutt, Allen C-H Wu, and Steve Y-L Lin: HIGH-LEVEL SYNTHESIS Introduction to Chip and System Design, Kluwer Academic Publisher, 1992.

[2] M.C.McFarland, A.C.Parker, R.Camposano: The high-level synthesis of digital systems, Proc. IEEE, 1990.

[3] 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.1476-1485, 2004.

[4] S. Tosun, N. Mansouri, M. Kandemir and Y. Xie, “Reliability- Centric High-Level Synthesis,” Proc. Design, Automation and Test in Europe Conf. (DATE), Vol.2, No.2, 1258-1263, 2005.

[5] Tomoo INOUE,Hayato HENMI,Yuki YOSHIKAWA,Hideyuki ICHIHARA : High-Level Synthesis for Multi-Cycle Transient Fault Tolerant Datapaths, 2011.

[6] Petros OICONOMAKOS,Mark ZWOLINSKI,Bashir M. AL-HASHIMI : Versatile High-level Synthesis of Self-checking Datapaths Using an On-line Testability Metric, 2003.

[7] F.J.Kurdahi and A.C.Parker, REAL:A program for register allocation, In Proc. Design Sutomation Conf.. pp210-215, 1987.

[8] 石井英明,細川利典, テスト容易化インタフェースを設けた動作合成システム PICTHY の開発，第 62 回 FTC 研究会,2010.