7-17

テスト容易化のためのインタフェースを備えた 動作合成システム PICTHY の開発

日大生産工(院)　○石井 英明　　日大生産工　　細川 利典

1. はじめに

近年，半導体技術の進歩による大規模集積回路（Large Scale Integrated circuit : LSI）の大規模化に伴い，回路が複雑化してきている．従来，LSI の設計において，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）の記述をし，論理合成ツールを用いることでネットリストを生成している[1]．しかしながら，従来の手順では設計生産性の面から設計が困難となってきている．そこで RTL よりさらに抽象度の高い動作レベルでの設計を可能にする技術として動作合成[1]が着目されている．

動作合成の入力である動作記述は RTL より抽象度が高いため，RTL の記述に比べ記述量が少なく設計生産性に優れる[2]．反面，どの演算操作をどの演算器に割当てるか，どの変数をどのレジスタに割当てるかなど，詳細な設定は動作合成ツールが行うため，アルゴリズム次第で出力される回路の面積や性能に差が生じてしまう．しかしながら，動作合成によって生成される回路の面積や性能を最適化する方法がいくつも提案され，動作合成による設計は実用的なレベルであると報告されている[3]．市販ツールもいくつか登場し，C 言語記述を入力とした CyberWorkBench[3]，System-C 言語記述を入力とした Cynthesizer[4]などがある．これらの動作合成ツールは，入力言語として既存の言語を使用している．しかしながら，System-C 言語は，ソフトウェア開発者にとってあまり馴染みのない言語であり，言語習得が必要となる．また馴染みのある C 言語を使用している場合でも，ハードウェア用に拡張されているため，新しく習得する必要がある．

動作合成や論理合成を用いて設計，製造した LSI を出荷するにあたり，不良品を市場に出さないよう LSI のテストをする必要がある．通常，LSI のテストはテスト容易化設計としてフルスキャン設計[5]を施した回路に対し，自動テストパターン生成ツール（Automatic Test Pattern Generator : ATPG）を用いてテストパターンを生成しテストをする[6]．フルスキャン設計を施しているため高い故障検出効率を得るテストパターンが生成できるが，LSI の記憶素子にスキャンチェインを挿入することによる面積オーバーヘッドや，テスト長の増大が問題となっている．そのためなるべくスキャン設計を用いないでテスタビリティを向上させるために，動作合成内でテスト容易化を考慮する研究が報告されている[7]．

本論文では，言語習得を必要としない C 言語を入力とし，テスト容易化の研究のために動作合成の各ステップで処理結果を出力できる機能を持った動作合成システム PICTHY（Provide Input C language and Testability interface High-level sYnthesis）について述べる．

2. 動作合成システム PICTHY

動作合成は動作記述を入力とし，RTL 回路記述を生成する．入力から出力までの過程を大きくわけると，グラフ生成，スケジューリング，バインディング，RTL 回路記述生成の四つのフェーズにわけられる[8]（図 1）．PICTHY も同様に，動作記述を入力とし，RTL 回路記述を生成する．また，グラフ生成とスケジューリング後のグラフを出力できる（図 2）．

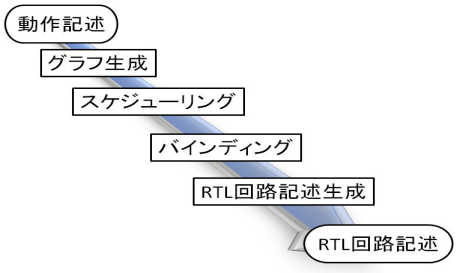

図 1. 動作合成

Development of behavioral synthesis system PICTHY with interfaces for testability

Hideaki ISHII and Toshinori HOSOKAWA

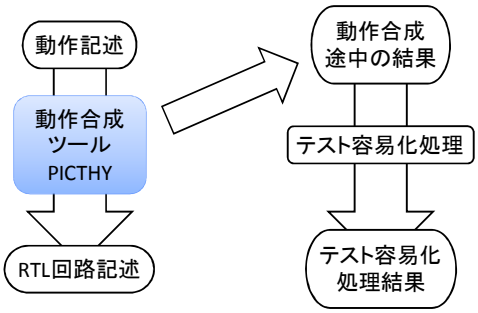

図 2. システム構成図

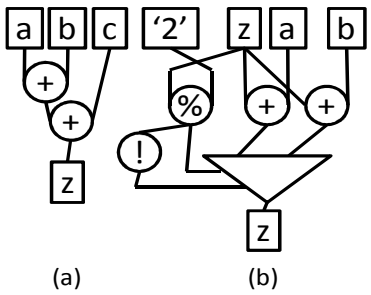

図 4. マルチ ADD

3. 動作合成システム PICTHY のアルゴリズム

基本的な流れは 2 章で述べたような四つのフェーズで進むが，C 言語にはコンパイルの第一段階としてプリプロセッサというプログラムを使ってコンパイル前のソースファイルにさまざまな操作（include や define 等）を施す処理がある[6]．このプリプロセッサを対応させるために，グラフ生成の前へプリプロセッサ処理のフェーズを付け加えた五つのフェーズを C 言語からの動作合成の流れとする．

入力となる動作記述は一部を除き ANSI-C99[9]に準拠する．ANSI-C99 と異なる部分は main 関数の仮引数部分と戻り値の意味である．通常 main 関数の仮引数はコマンドライン引数の文字列を格納する変数と，引数の数を格納する変数の 2 つで構成される．動作合成においてはコマンドライン引数の変わりに外部入力が必要となる．そこで main 関数の仮引数を外部入力とみなし，仮引数の型の決まりは特にないものとする．また外部入力同様，外部出力も必要となるため，戻り値を外部出力とみなす．しかしながら，ANSI-C99 の仕様では外部出力が最大 1 つとなるため，複数の外部出力が必要な場合は，外部変数を用いることで対応する．以上の条件を満たした動作記述の例を図 3 に示す．この動作記述を入力として各フェーズを説明する．

3-1. プリプロセッサ処理

プリプロセッサ処理とは，C 言語のプリプロセッサの処理をするフェーズである．PICTHY では，プリプロセッサ処理は，他のファイルを埋め込む命令#include のみ対応する．その他については無視し，全てコメントとして扱う．

```
int main(int a, int b, int c)
{
    int i, z;
    z = a + b + c;
    while(z < 100){
        if(z%2){  z += a; }
        else{ z += b; }
    }
    return z;
}
```

図 3. 動作記述例

3-2. グラフ生成

グラフ生成とは，動作記述をグラフで表現するフェーズである．グラフの表現方法は複数あるが，ADD（Assignment Decision Diagram）[10]を用いる方法を利用する．動作記述から ADD に変換するには，まずループごとにステート分割をする必要がある．図 3 の動作記述内のループは while 文が 1 つである．そこで，while 文の直前までを State1，while 文内を State2 とする．ステートごとに ADD を生成したものを図 4 に示す．説明の都合上，同ステートに存在する ADD をまとめてマルチ ADD とする．図 4(a)が State1 の ADD，図 4(b)が State2 の ADD である．上部の□は入力変数，下部の□は出力変数，○は演算操作，▽は ADN(Assignment Decision Node)[10]と呼ばれ制御を表す．図 4(b)を例に説明する．z を 2 で割った余りが 1，つまり z が奇数のときは z+a の結果が z に代入され，余りが 0，つまり z が偶数のときは z+b の結果が z に代入されることを示している．マルチ ADD を生成すると同時に，状態遷移表（表 1）を生成する．State1 は if 文による分岐がないので，常に State2 へと遷移する．一方 State2 は z<100 が真の間ループをし，偽の場合ループを抜ける．よって真の間は State2 から State2 へと遷移し，偽の場合はループを抜けて終了となるため，初期状態である State1 へと戻る遷移となる．マルチ ADD と状態遷移表の 2 つの情報から，1 つの ADD を生成したものを図 5 に示す．状態初期化用の RESET 変数と，現状態を示す State_reg 変数を自動で付加し，各ス

表 1. 状態遷移表

現状態	分岐	次状態
State1	—	State2
State2	z<100	State2
	$\overline{z<100}$	State1

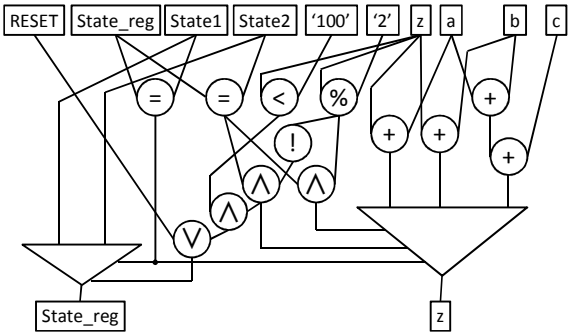

図 5. ADD 生成

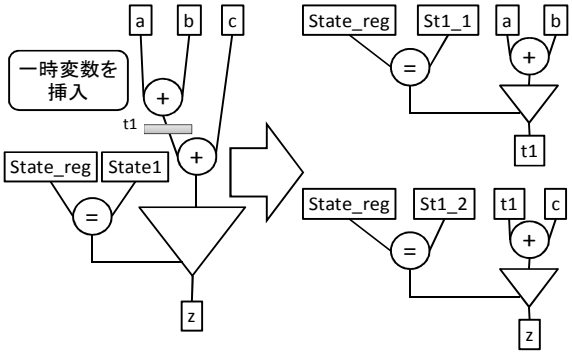

図 6. リニアステートインサーション

テート状態を分岐としてマルチ ADD に接続している. この ADD はテスト容易化用のインタフェースとしてグラフ生成処理が完了した直後に出力可能である.

3-3. スケジューリング

スケジューリングとは，各演算操作をどの時刻に実行するかを決定するフェーズである．ADD では，入力変数から出力変数までを 1 時刻で実行することを意味する．これは，図 5 の右端，つまり図 4(a)に該当する部分 z=a+b+c を 1 時刻で実行することを意味する．ここで仮に 1 時刻 1 演算にしたい場合を考える． 1 時刻 1 演算にしたい場合，式を二項式に分解することで実現できる．例えば z=a+b+c の場合，一時変数 t1 を用いて t1=a+b，z=t1+c と分解しても z の結果は同じとなる．ADD において同様の効果を与える方法としてリニアステートインサーションがある[10]. 図 6 では a+b の後に一時変数 t1 を挿入し，二つのステートに分割する様子を示している．分割後, つまりスケジューリング後の ADD が図 7 となり，テスト容易化用のインタフェースとしてスケジューリング済み ADD を出力可能である.

3-4. バインディング

バインディングとは，各演算操作や変数に演算器やレジスタを割当てるフェーズである．演算器を割当てる処理を演算器バインディング，レジスタを割当てる処理をレジスタバインディングと呼ぶ[2]．両バインディングとも，ADN から状態遷移の情報を得

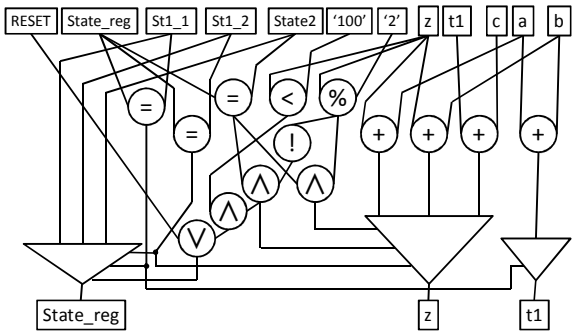

図 7. スケジューリング済み ADD

て，リソース使用の開始時刻と終了時刻を決定する．以降，リソース使用のリソース開始時刻，リソース使用の終了時刻をリソース終了時刻とする.

まずはレジスタついて考える．外部入力となる変数は，処理の開始に値が代入される．そこでリソース開始時刻を時刻 1 に設定する．例では a，b，c が外部入力変数となるため，それぞれのリソース開始時刻を時刻 1 に設定する．反対に外部出力となる変数は．処理の最後に値を出力する必要がある．そこでリソース終了時刻を状態遷移の最も遅い時刻+1 として設定する．+1 となる理由は，状態遷移直後に値が失わないようにするためである．例では z が外部変数出力となり，状態遷移は St1_1→St1_2→State2 である．リソース終了時刻を時刻 3+1 となる時刻 4 に設定する．残りの部分は状態遷移の情報から，必要となる時刻を計算する. a のリソース終了時刻を例に考える. a は一番左の加算操作と一番右の加算操作の入力となる．一番左の加算操作は現状態が State2 のときの操作だとわかり，時刻 3 が候補となる．一方，一番右の加算操作は現状態が St1_1 のときの操作だとわかり，時刻 1 が候補となる．リソース終了時刻は最後に必要となる時刻となるため，時刻が遅い時刻 3 が選ばれる．同様の操作で全ての変数のリソース時刻を設定しライフタイム[2]を求めたものを図 8 に示す．z と t1 はライフタイムが重ならないため共有化し R1 に，c を R2 に，a を R3 に，b を R4 に割当てた．R1，R2，R3，R4 はそれぞれレジスタである.

次に演算器について考える．演算操作は全て ADN

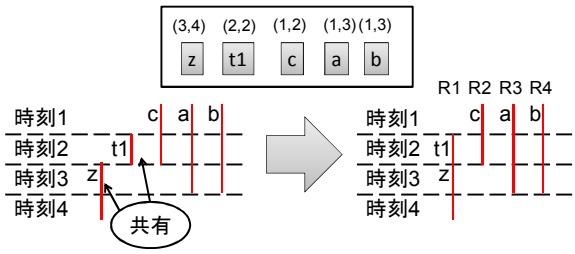

図 8. RTL 回路記述生成

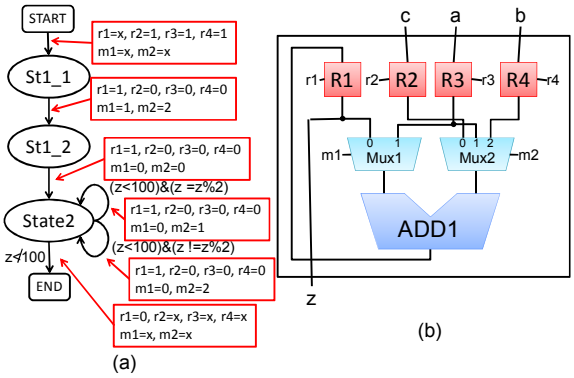

図 9. RTL 回路記述生成
(a)コントローラ　(b)データパス

から情報を得る必要がある．例えば図 7 の一番右の加算操作は，現状態 St1_1 のときの操作だとわかる．よってリソース開始時刻，終了時刻ともに時刻 1 に設定される．同様の操作で全ての演算操作のリソース時刻を求めると，左から順番に{3,3}，{3,3}，{2,2}，{1,1}となる．ここで括弧内の左がリソース開始時刻，右が終了時刻を表す．ここで時刻 3 に設定されている演算操作に着目すると，時刻が同じでも z%2 が真か偽かでわかれている．よって同時に処理が起こらない．このことより，全ての演算操作は同時に起こり得ないため，全て同じ加算器を割当てることとなる．以上でバインディングの処理は終わりとなる．

3-5. RTL 回路記述生成

RTL 回路記述生成とは，バインディングで割当てたレジスタや演算器間の結線をするフェーズである．制御情報をコントローラとして，処理内容をデータパスとして生成する．実際に生成した例を図 9 に示す. 時刻 1 で実行されるべき処理 t1=a+b を例に説明する．時刻 1 ということで St1_1 の処理となるが，次の状態に遷移する過程で r1=1，r2=0，r3=0，r4=0，m1=1，m2=2 がコントローラからデータパスに対し信号を送る．よって Mux1 は m1=1 で右を選択し，Mux2 は m2=2 で一番右を選択する．Mux1 の右は R3 となるが，これは変数 a を割当てたレジスタである．また，Mux2 の一番右は R4 であり変数 b を割当てたレジスタである．よって t1=a+b が時刻 1 で正常に動作することがわかる．この情報を基に Verilog-HDL[11]記述として RTL 回路記述を出力する．

4. おわりに

本論文では C 言語を入力とし，テスト容易化の研究のために動作合成の各ステップで処理結果を出力できる機能を持った動作合成システムについて述べた．現在は動作記述からグラフ生成，つまり ADD を生成する部分を実装中である．動作合成の本質と外れるため動作合成の流れから外してあるが，動作記述からグラフ生成の間に，フロントエンドとして字句解析や構文解析，意味解析と呼ばれる処理がある[12]．この構文解析の処理まで実装済みであり，意味解析をすることで ADD が生成されることとなる．

今後の課題は動作合成完成に努め，ADD の利点を生かしたスケジューリング方法やバインディング方法の提案などが挙げられる．

「参考文献」

1)　藤原秀雄，ディジタルシステムの設計とテスト，工学図書株式会社，2004

2)　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

3) Kazutoshi Wakabayashi, CyberWork¬Bench: Integrated Design Environment Based on C-based Behavior Synthesis and Verification, 2005

4) "Forte Design Systems" Web サイト：http://www.forteds.com/ 2003

5) M.Abramovici, M.A.Breuer and A.D. Friedman, Digital systems testing and test-able design, IEEE Press, 1995

6) H.Fujiwara, Logic Testing and Design for Testability, The MIT Press, 1985

7) Tien-Chien Lee, Wayne H.Wolf, Niraj K.Jha, John M.Acken, Behavioral Synthesis for Easy Teatability in Data Path Allocation, ICCD, 1992

8) Herbert Schildt, 株式会社トップスタジオ 訳：独習 C 第 3 版，株式会社翔泳社，1994-2004

9) American National Standards Institute: ANSI/ISO/IEC 9899-1999: Programming Languages -- C, 1999

10) V.Chaiyakul, D.D.Gajski, Assignment Decision Diagram for High-Level Synthesis, Technical Report #92-103, 1992

11) IEEE Standard 1076, Verilog Language Reference Manual, IEEE, 2001

12) 原田賢一，コンパイラ構成法，共立出版株式会社，1999