ISSN 2186-5647

7-62

コントロール／データフローグラフを用いた動作合成システム PICTHY の評価

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 circuits : 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，動作合成ツール次第で出力される回路の面積や性能に差が生じる．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，新しく言語を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実行する[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].

この問題に対し，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入出力できる機能を持った動作合成システム PICTHY（Provide Input C lan-guage and Testability interface High-level sYnt-hesis）を提案した[8]．PICTHY では中間言語として ADD（Assignment Decision Diagram）[9]を用いている．ADD はコントローラとデータパスが一体となったグラフであり，テスト環境生成[10]に関数研究に適している．しかしながら，時刻の概念がないためライフタイム[2]の抽出などが困難となり，途中結果を出力できる機能の活用が困難である．

本論文では，ADD の代わりにコントローラ／データフローグラフ（Control/Data-Flow Graph : CDFG）[2]を用いた動作合成システム PICTHY 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４つのフェーズにわけられる[1]（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，グラフ生成後，

Evaluation of Behavioral Synthesis System PICTHY Using Control/Data-Flow Graph

Hideaki ISHII and Toshinori HOSOKAWA

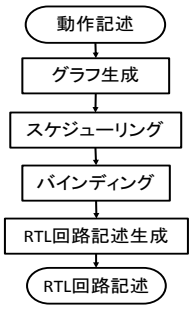

図 1. 動作合成

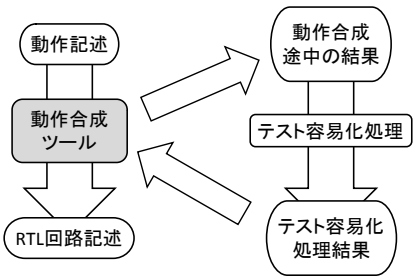

図 2. システム構成図

スケジューリング後，バインディング後のグラフを入出力できる（図 2）.

3. 動作合成システム PICTHY のアルゴリズム

入力となる動作記述は一部を除き ANSI-C99[11] 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例を図 3 に示す.

3-1. 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，CDFG を用いる方法を利用する．CDFG によるグラフ生成は，動作記述からコントロールフローグラフ（Control Flow Graph : CFG）とデータフローグラフ（Data Flow Graph : DFG）の 2 種類にわけて生成する.

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 文などのループ処理から構成されるグラフである（図 4）．ノード間の接続は処理の順番を表し，上から下へ処理をする.

```
int main(int a, int b, int c, int d)
{
    int p, q, r, z;
    p = a + b - c;
    q = a + d;

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

    z = r - q;
    return z;
}
```
図 3. 動作記述例

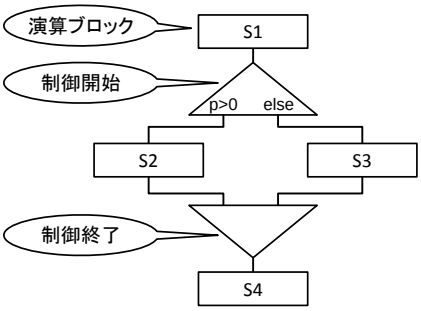

図 4. CFG

連続した計算式は演算ブロックとして表す．図 3 の記述では，p=a+b-c と q=a+d は連続した計算式となり演算ブロック 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*c は S3，z=r-q は S4 としてまとめられる.

if 文や while 文は制御処理とし，制御開始と制御終了で表す．if 文や while 文の条件分岐は制御開始内で保持され，条件分岐先の演算ブロックにそれぞれ接続される．図 3 の記述では p>0 の真偽で分岐するため，p>0 の先に S2 が接続され，else の先に S3 が接続される.

DFG とは，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 5）．DFG の上側が入力となり，下側が出力となる．つまり複数の変数や定数が入力となり，1 つの変数に収束され出力される．間にある演算操作は，左入力に対

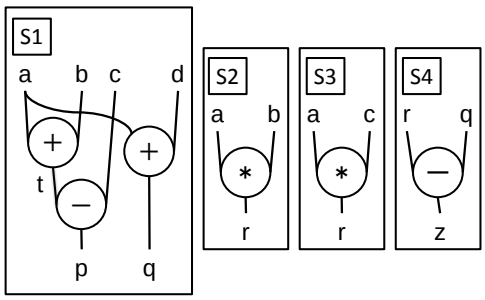

図 5. DFG

して右入力で演算操作を実行し，結果を下に出力することを意味する. 例えば演算ブロック S1 の q=a+d の場合，演算操作の左入力に a が接続され，右入力に d が接続され，下に q が接続される．同様に全演算ブロック内の計算式に対してグラフ化する．

S1 の加算操作と減算操作を繋ぐ部分のように，2 つ以上の項を持つ計算式では，演算操作同士が接続される部分が存在する．これは，一時的に値を保持する必要を意味するため，中間変数とみなし，新たな変数として設定する．

3-2. 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．演算ブロック S2～S4 では，演算操作がそれぞれ 1 つずつしか使われていないため, 1 時刻で処理を終えることができ，時刻が一意に決定される．一方，演算ブロック S1 では, p=a+b-c の演算をするのに 2 時刻分, q=a+d の演算をするのに 1 時刻分必要となる．この場合，演算が早く終わる q=a+d の加算演算を処理する時刻を他方の p=a+b-c の演算に必要な時刻内で自由に設定できる．

基本的なスケジューリング方法として ASAP（As Soon As Possible）スケジューリングと ALAP（As Late As Possible）スケジューリングがある[2]．ASAP スケジューリングとは，各演算操作を可能な限り早い時刻に割り当てるスケジューリングであり，q=a+d の加算操作を 1 時刻目に割り当てる方法である（図 6（a））．ALAP スケジューリングとは各演算操作を可能な限り遅い時刻に割り当てるスケジューリングであり，q=a+d の加算操作を 2 時刻目に割り当てる方法である（図 6（b））． 以降の処理は ALAP スケジューリングを選択した場合を例に説明する．

3-3. 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り当てるフェーズである．演算器を割り当てる処理を演算器バインディング，レジスタを割り当てる処理をレジスタバインディングと呼ぶ

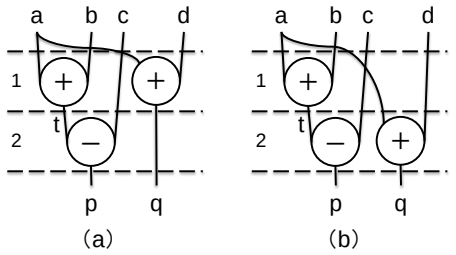

図 6. スケジューリング
（a）ASAP （b）ALAP

[2]. 両バインディングとも, ライフタイムを取得し，ライフタイムが重ならない演算操作同士，変数同士を同一リソースに割り当てる（図 7（a））．ライフタイムとは変数や演算操作が使用している時刻のことである．演算ブロックが異なる場合，演算器バインディングに関しては，処理する時刻も異なるため共有可能となる．しかしながら，レジスタバインディングでは，演算ブロックを越えて値を保持する場合があるため，全体を考慮する必要がある．例えば S1 の出力で使用する変数 p をレジスタ R1, 変数 q をレジスタ R2 に割り当てた場合について考える．変数 p は S1 後の分岐判定として使用し, 以後使用していない．よって S2 や S3 の時点でレジスタ R1 は未使用となり共有可能となる．一方, 変数 q は S4 で入力として使用されているため，S4 まで値を保持する必要がある．よって S2 や S3 の時点でレジスタ R2 は使用中となり，共有不可能となる．以上のことより，各演算ブロックの入出力を考慮しながらライフタイムを取得し，レジスタバインディング行う．

代表的なバインディング方法として LEA（Left-Edge Algorithm）がある．LEA とは面積最小化バインディングの 1 つであり，変数や演算操作を出現時刻順でソートし，出現順に共有可能な部分へ割り当てる方法である（図 7（b））．S1 の場合，レジスタ R1 に a，t，p，レジスタ R2 に b，q，レジスタ R3 に c，レジスタ R4 に d がそれぞれ割り当てられる．

3-4. 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行うフェーズである．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．生成した RTL 回路全ての情報を基に，Verilog-HDL[12]記述として RTL 回路記述を出力する．

4. 出力フォーマット

PICTHY の機能として，グラフ生成後，スケジューリング後，バインディング後の結果を出力する機

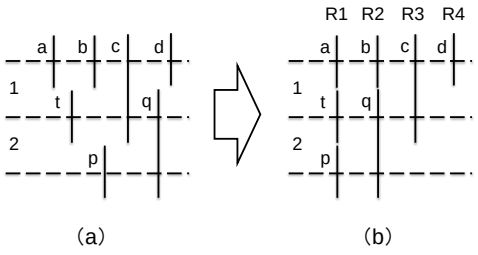

図 7. レジスタバインディング
（a）ライフタイム （b）LEA

能を持つ．ここではスケジューリング後の結果の出力を例に説明する．出力フォーマットは CFG, DFG, 外部入力，外部出力で構成される（図 8）．

CFG は[FSM]の項目で表し，状態名，分岐数，条件，分岐先からなる．状態名は演算ブロックの名前を表し，例だと S1 や S2 となる．分岐数は if 文などの分岐数を表し，以降の条件と分岐先の数と一致する．条件は分岐先に進む条件を表す．分岐先は条件を満たすときの遷移先となる演算ブロック名を表す．

DFG は[状態名]の項目で表し，識別子名，識別子の種類，時刻，左接続先，右接続先からなる．演算ブロックの数だけ[状態名]の項目が存在する．識別子名は変数名や演算操作を表す．なお，識別子には固有の ID 番号が付けられている．識別子の種類は識別子が変数なのか演算操作なのかを表す．時刻はスケジューリングされた時刻を表す．左接続先は演算操作の左入力先となる識別子を表す．右接続先は演算操作の右入力先となる識別子を表す．

外部入力は[INPUT]の項目で表し，識別子名からなる．外部入力となる変数が 1 行ずつ記される．

外部出力は[OUTPUT]の項目で表し，識別子名からなる．外部出力となる変数が 1 行ずつ記される．

これらの情報を 1 つのファイルとして出力した例が図 9 である．図 9 は図 4～6 の出力例である．

```
[FSM]
状態名  分岐数  条件  分岐先  条件  分岐先  ･･･
          :

[状態名]
識別子名[Id]  識別子の種類  時刻  左接続先 [id]  右接続先 [id]
          :

[状態名]
識別子名[Id]  識別子の種類  時刻  左接続先 [id]  右接続先 [id]
          :

[INPUT]
識別子名
          :

[OUTPUT]
識別子名
          :
```

図 8．出力フォーマット

```
[FSM]
S1  2  x>0  S2  else  S3
S2  1  (null)  S4
S3  1  (null)  S4
S4  0  (null)  (null)

[S1]
x[0]  IDENT  -1  -[1]  (null)
-[1]  '-'  2  t[2]  c[3]
t[2]  IDENT  1.2  +[4]  (null)
c[3]  IDENT  0  (null)  (null)
+[4]  '+'  1  a[5]  b[6]
a[5]  IDENT  0  (null)  (null)
b[6]  IDENT  0  (null)  (null)
y[7]  IDENT  -1  +[8]  (null)
+[8]  '+'  2  a[4]  d[9]
d[9]  IDENT  0  (null)  (null)

[S2]
z[0]  IDENT  -1  *[1]  (null)
*[1]  '*'  1  a[2]  b[3]
a[2]  IDENT  0  (null)  (null)
b[3]  IDENT  0  (null)  (null)

[S3]
z[0]  IDENT  -1  *[1]  (null)
*[1]  '*'  1  a[2]  c[3]
a[2]  IDENT  0  (null)  (null)
c[3]  IDENT  0  (null)  (null)

[S4]
r[0]  IDENT  -1  -[1]  (null)
-[1]  '-'  1  z[2]  y[3]
z[2]  IDENT  0  (null)  (null)
y[3]  IDENT  0  (null)  (null)

[INPUT]
a
b
c
d

[OUTPUT]
r
```

図 9．出力ファイル例

6. おわりに

本論文では，CDFG を用いた動作合成システム PICTHY について述べた．

今後の課題は，出力ファイルを実際に利用し，テスト容易化の研究に利用できるか否かの確認や，PICTHY 自身にテスト容易化の機能を付加することが挙げられる．

「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) 石井英明，細川利典，テスト容易化のための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

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

10)A.Gosh, S.Devadas, and A.R.Newton, Test generation and verification for highly sequential circuits, IEEE Trans. Comput. Aided Des., vol. 10 no.5 pp.652-667, May 1991

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

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