7-47

PICTHY を用いた AES の設計

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，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]．

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）を提案した[8]．しかしながら，実際に RTL 設計したものを論理合成した設計との比較を行っていない．

本論文では．暗号規格の一つである AES（Advanced Encryption Standard）[9]を回路実装し，PICTHY を用いた動作合成と，従来の設計手順である RTL 設計による論理合成との差について調査する．

2. AES のアルゴリズム

AES の暗号化アルゴリズムは，128 ビットの入力データに対し，ShiftRows，SubBytes，MixColumns，AddRoundKey の処理を順に，直列に並べて繰り返し適用するものである．繰り返し数は鍵長により決定される．本稿で使用する鍵長は 128 ビットのため，繰り返えし数は 12 回である．各処理では，入力データを 8 ビット単位に区切った 4×4 の行列として扱う．

3. 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

Design of AES Using PICTHY

Hideaki ISHII and Toshinori HOSOKAWA

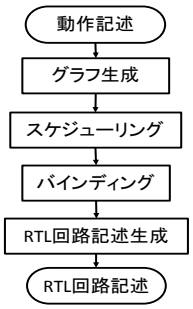

図 1. 動作合成

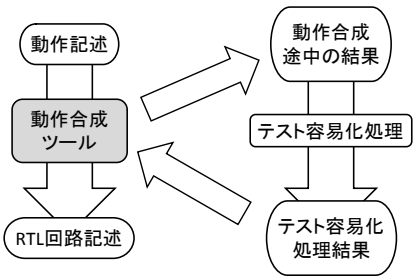

図 2. システム構成図

```
void SubBytes(){
    int i;

    for(i=0; i<16; i++){
        St [i] = Sbox[St[i]];
    }
}
```
図 3. 動作記述例

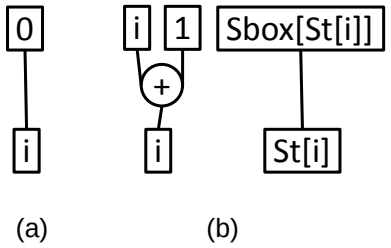

図 4. ステートごとの ADD

グ，RTL 回路記述生成の 4 つのフェーズにわけられる[10]（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入出力できる（図 2）．

4. 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 AES の動作記述例として，SubBytes を図 3 に示す．ここで ST[i]は外部変数として宣言されているものとする．

4-1. 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，ADD（Assignment Decision Diagram）[12]を用いる方法を利用する．

動作記述から ADD に変換するためには，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 for 文が 1 つである．そこで，for 文の直前までを State1，for 文内を State2 とする．for 文の初期化式については，ループへ入る前に処理する必要がある．そこで直前のステートである State1 として考える．

ステートごとに ADD を生成したものを図 4 に示す．図 4(a)が State1 の ADD，図 4(b)が State2 の ADD である．上部の□は入力変数または定数，下部の□は出力変数，○は演算操作を表す．入力変数や出力変数で配列が使用されている場合，RAM を利用することを示す．C 言語記述での i++は，i に 1 を足しこむことを示す．そこで，入力変数 i と定数 1 を加算操作に接続し，出力変数 i へと接続することで i++を示す ADD となる．

ステートごとの 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 i<16 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．ステートごとの 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，各ステート状態を分岐としてステートごとの ADD に ADN(Assignment Decision Node)[12]

表 1. 状態遷移表

現状態	分岐	次状態
State1	—	State2
State2	i<16	State2
	$\overline{i<16}$	State1

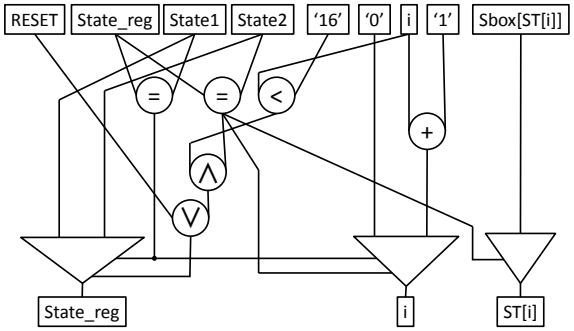

図 5. ADD 生成

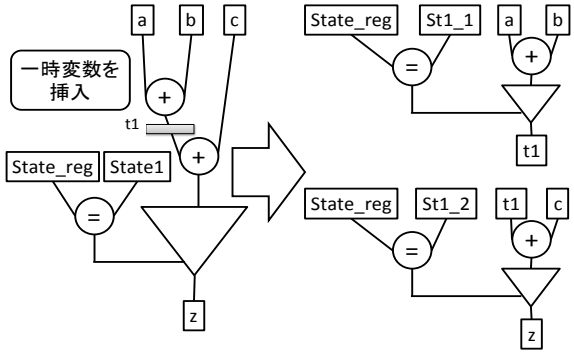

図 6. リニアステートインサーション

を用いて接続している. 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.

4-2. 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．ADD では，入力変数または定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．例えば z=a+b+c のような二項以上の演算式がある場合，1 時刻で実行することとなる．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[12]. 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．

通常，二項以上の式を含むため，リニアステートインサーションによるスケジューリングが必要となる．しかしながら，図 3 に示す通り SubBytes 内には二項以上の式がない．そこでスケジューリングができず，結果が一意に決まる．また，他の処理に関しても同様に，結果が一意に決まる．

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．

4-3. 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，リソース使用の開始時刻をリソース開始時刻，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 Sbox[ST[i]]が RAM からの入力となるため，外部入力変数となり，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 ST[i]が RAM への出力となるため，外部変数出力となり，状態遷移は State1→State2 である．リソース終了時刻を時刻 2+1 となる時刻 3 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．i のリソース終了時刻を例に考える. 定数 0 を入力とする i は ADN の選択条件から現状態が State1 のときの操作だとわかり，時刻 1 が候補となる．一方，加算操作の入力となる i は，ADN の選択条件から現状態が State2 のときの操作だとわかり，時刻 2 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，ライフタイムが重ならない変数を，同一変数として割り当てる．ここで，RAM を使用する変数に関しては，RAM 専用のレジスタとして割当てる．演算器についても同様にライフライムを求め，ライフタイムが重ならない演算操作を，同一の演算器に割り当てる．SubBytes 内で使用する変数は RAM を除くと i のみ，演算操作は加算操作 1 つとなるため，どちらも一意に決まる．

以上でバインディングの処理は終わりとなる．

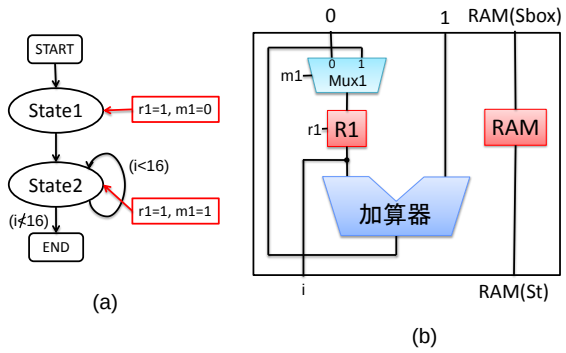

図 7. RTL 回路記述生成
(a)コントローラ (b)データパス

4-4. 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 7 に示す．State2 で実行される処理 i++を例に説明する．次の状態に遷移する過程で r1=1，m1=1 がコントローラからデータパスに対し信号を送る．R1 は変数 i を割り当てたレジスタである．よって Mux1 は m1=1 で右を選択することで，i+1 の結果を i に代入することができる．

以上で SubBytes に対する RTL 回路が生成されたが，他の処理も同様にグラフ生成，スケジューリング，バインディングを行い，RTL 回路を生成する．生成した RTL 回路全ての情報を基に，Verilog-HDL [13]記述として RTL 回路記述を出力する．

5. 実験結果

AES を C 言語記述による動作合成，Verilog-HDL による論理合成で生成した．各結果をまとめたものを表 2 に示す. 合成方法は設計に使用した合成方法，入力言語は入力記述として使用したプログラミング言語，ファイルサイズは入力ファイルの総バイト数，作業日数は合成までにかかった日数を示す．

表 2 より，動作合成のほうが，ファイルサイズが 1/7 以下，作業日数が 1/4 以下であることがわかる．これは，動作合成のほうが作業者の負担が少ないことを意味し，生産性が向上していることがわかる．

表 2. 動作合成と論理合成の差

合成方法	入力言語	ファイルサイズ	作業日数
動作合成	C言語	6104	14
論理合成	Verilog-HDL	45590	60

6. おわりに

本論文では暗号規格の一つである AES を回路実装し，PICTHY を用いた動作合成と，従来の設計手順である RTL 設計による論理合成との差について調査した．

今後の課題は，評価対象とする回路を増やし正確なデータを集めること，アルゴリズムの違いによる動作合成と論理合成のメリット・デメリットの解析などが挙げられる．

「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) 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) J.Daemen and V.Rijmen, “AES proposal:Rijndael”, Web サイト：http://csrc.nist.gov/archive/aes/iinde.html, 2001

10) 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

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

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

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