設 計 自 動 化 66－5
（1993. 3. 12）

テンプレート化論理合成手法

遠藤 真　高原 厚

NTT LSI 研究所

RTL 記述からの論理合成は、言語に表現される部品と動作のモデルをセルやマクロの実部品のネットワークにマッピングする過程としてとらえることができる。従来は合成過程がシステムに組み込まれ、変更が困難であったため、多様な設計手法に対応するには人手が介入することが多かった。

本手法では、合成過程をテンプレート化することにより、記述と合成回路の対応を制御し、カスタマイズ可能とした。殊にレジスタ種別やその周辺回路構成の変更に効果的である。本論文では合成過程のモデル化、テンプレート記述形式などについて述べる。

LOGIC SYNTHESIS USING TEMPLATES

Makoto Endo　Atsushi Takahara

NTT LSI Laboratories
3-1, Morinosato Wakamiya, Atsugi-Shi, Kanagawa 243-01, Japan

A logic synthesis system consists of mapping procedures which maps behavioral facilities into networks of actual cells and modules. Conventional synthesis systems do not allow the designer to customize those mapping procedures.

This document introduces a new synthesis system, using a template mechanism to customize synthesis procedures. Designers can modify synthesized circuit structures by selecting or modifying templates.

We present (1) a new methodology for writing logic synthesis procedures and (2) the synthesis templates description language used in writing the procedures.

1 はじめに

LSIの論理設計工数を削減するために、設計の上位化が求められている。設計記述の抽象度を上げ、設計効率の向上を目指すとともに、LSI製造テクノロジーから独立な設計をすることにより、設計資産の流用を促進するためである。

現在、FSMを含むRTL記述のシミュレータ及び合成システムが実用に供されている。しかしながら実際の設計では、RTL言語を用いながら、従来通りテクノロジーに依存した部品間の結線(ネットワーク)を記述していることも多い。たとえば、レジスタを含むデータ転送回路、状態遷移回路、テスト用回路、演算回路などの記述がゲートレベルで記述される。これには、(1)多様な設計手法や回路構成法に合成ツールが対応していない、(2)合成回路では、所望のパフォーマンスが得られない、などの理由がある。

本論理合成手法は、設計手法や回路構成に応じたテンプレートによって合成過程を制御して所望の合成回路を得るとともに、高性能のマクロセルや既存回路、ソフトマクロを容易に回路に組み込めるようにするものである。設計者はテンプレートを選択あるいはカスタマイズすることにより、所望の構成、性能の回路を得ることができる。

本論文では、論理合成のテンプレート化にあたって検討したRTLデータ、合成過程モデルと、可読性の高いテンプレート記述形式について報告する。従来合成システムに組み込まれていた合成過程をテンプレートによってオープン化することにより、柔軟性、拡張性の高いシステムとすることができる。さらに設計者は、必要に応じて自らテンプレートを記述することにより、所望の回路を合成することができる。

2 合成データモデル

RTL記述からの論理合成は、言語に表現される**部品**と動作のモデルを、セルやマクロの実部品のネットワーク構造にマッピングする過程として捉えることができる。

RTL記述と合成回路の対応を明確にするため、RTLデータ構造、**部品**(レジスタ等)と**RTL動作文**をモデル化する。

2.1 RTLデータ構造

RTLデータ構造は、**モジュール**と呼ぶ回路のまとまりを基本にしている。以下に概略を示す。

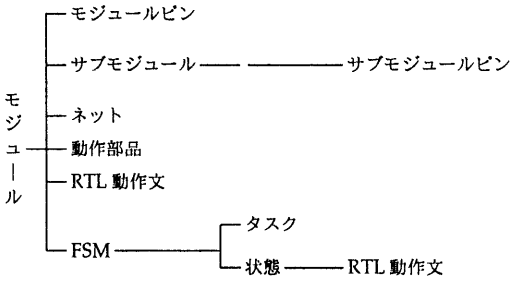


図1: RTLデータ構造

FSMの状態の配下にあるRTL動作文は、その状態がアクティブの時に実行されるもので、モジュール直下にあるRTL動作文とは区別される。

2.2 部品

部品には、以下のような**動作部品(ファシリティ)**と**構造部品**がある。

表1: 部品モデル

動作部品(ファシリティ)	構造部品
レジスタ	モジュール
ラッチ	サブモジュール
RAM	モジュールピン
ROM	サブモジュールピン
ターミナル	ネット
定数	
状態レジスタ	
タスクレジスタ	

構造部品間の関係は図1に示した。ネットは、構造部品の接続を表すものであり、動作部品を参照することはない。一方、次節に述べるRTL動作文は動作部品と、ネットを除く構造部品を参照する。

なお、表1のうち状態レジスタとタスクレジスタは、後に述べるFSMモジュール生成時に用いる。

2.3 RTL動作文

RTL動作文は、合成可能で単純なRTL動作を定義したもので、クロック付き条件代入文や関数参照文等の文がある。UDL/I[1]、SFL[2]記述は全て、RTL動作文に変換可能である。市販の合成システムが合成可能なVHDL, Verilog HDL記述も同様に、RTL動作文に変換可能である。

UDL/Iの「コアサブセット」は意味定義のために簡潔な形式で定められており、全て合成可能であるが、レジスタとターミナルに対する動作の情報に全て還元されており、FSMの状態・タスク等の情報が失われている。また、FSMの状態割り当てが、1状態1レジスタに固定されているなど、合成のためには簡略化され過ぎている。そこで、合成用のサブセットとして、RTL記述を単純化すると共に、合成の自由度を確保するようにRTL動作文を定めた。

RTL動作文の種類と対応するUDL/I記述例は以下の通り：

- 単純代入文 *A* := *B*;
- 連結代入文 *A* := *B*!!*C*!!*D*!!...;
- 関数参照文 *A* := function(*B*,*C*,*D*,...);
- 条件代入文
  IF *cond* THEN
  A := *B* OFFSTATE *offstate* END_IF;
- クロック付き代入文
  AT RISE(*clock*) DO *A* := *B* END_DO;
- クロック付き条件代入文
  AT RISE(*clock*) DO IF *cond* THEN
  A := *B* END_IF END_DO;
- メモリ参照文 (*A* := *MEM* < *address* >;
- レジスタ・リセット文
  IF *cond* THEN RESET(*REG*1,*REG*2,...);
- レジスタ・プリセット文
  IF *cond* THEN PRESET(*REG*1,*REG*2,...);
- オートマトン状態遷移文
  AT RISE(*clock*) DO IF *cond* THEN
  *state*1- > *state*2 END_IF END_DO;
- オートマトン割り込み文
  IF *cond* THEN - > *state* END_IF;
- タスク消滅文
  AT RISE(*clock*) DO IF *cond* THEN
  FINISH(*task*1,*task*2,...) END_IF END_DO;
- タスク転移文
  AT RISE(*clock*) DO IF *cond* THEN
  task - - > *task* END_IF END_DO;
- タスク発生文
  AT RISE_HIGH(*clock*) DO IF *cond* THEN
  state- >> *task* END_IF END_DO;
- タスク・リセット文
  IF *cond* THEN
  RESET(*task*1,*task*2,...) END_IF;

RTL動作文には、同一資源に対する複数の代入(信号の衝突)の解決のため、UDL/Iのoffstate値の概念を導入している。相違点は単純代入文や関数参照文にもoffstate値があることである。全てのRTL動作文が単純代入文とブーリアン関数参照文に還元されるマッピングの過程を通じて、「信号の衝突」を正しく扱えるようにするためである。

3 マッピングモデル

マッピングは、いわゆるテクノロジーマッピングと、テクノロジー独立なマッピングに分けられる。本合成手法では、テンプレートによってマッピング過程をきめ細かく制御するために、各マッピングを以下のようにモデル化した。

テクノロジー独立マッピング

- ファシリティマッピング
- FSMマッピング
- 関数マッピング
- RTL動作文マッピング
- FSMモジュール生成
- 関数モジュール生成

テクノロジー依存マッピング

- メモリマッピング
- 定数マッピング
- 組合せ回路マッピング

次節以下、各マッピングについて述べる。

なお、最適化はそれぞれのマッピングのレベルで実行されるが、本論文では触れない。

3.1 ファシリティ・マッピング

表1に示した部品のうち、ターミナル・定数を除くファシリティを信号の入出力ポートをもつテクノロジー独立な仮想モジュールにマッピングする。RTL動作文の動作モデルに必要なピンと、ユーザが設けたいピンをテンプレートに定義する。各ピンには動作モデルに対応した属性を付与し、これをRTL動作文マッピングで用いる。

例えばレジスタに対する動作モデルと対応する仮想モジュールのピンの属性は表2の通りである。

表 2: レジスタ動作モデルと仮想モジュールのピン属性

動作	ピン属性
クロック	CLK
データ転送条件	COND
データ入力	ASGN
データ出力	REF
非同期リセット	RESET
非同期プリセット	PRESET

ファシリティ・マッピングでは、次に示すテクノロジー独立な仮想モジュールをデフォルトとして定義している。

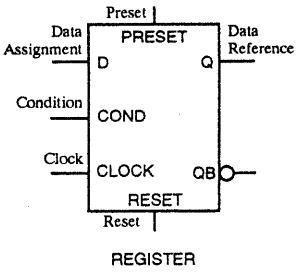


図 2: ファシリティマッピング例 (レジスタ)

ユーザは、例えばテスト設計手法に応じ、テンプレートを変更してスキャン用のピン等を仮想モジュールに付加することができる。

3.2 FSM マッピング

FSM を仮想モジュールにマッピングする。FSM の動作モデルと対応する仮想モジュールのピン属性とデータ幅は以下の通りである。

表 3: FSM 動作モデルと仮想モジュールのピン属性

動作	ピン属性	データ幅
データ転送クロック	DATA_CLOCK	1
状態遷移クロック	STATE_CLOCK	1
非同期リセット	RESET	1
状態参照	STATE_REF	状態数
WAIT 条件	WAIT	状態数
非同期状態割込み	STATE_INTERRUPT	状態数
タスク参照	TASK_REF	タスク数
タスク発生	TASK_GEN	タスク数
タスク消滅	TASK_TERM	タスク数

本マッピングにおいて FSM 配下の RTL 動作文は、動作条件、クロックを付加してモジュール配下に移される。例えば、レジスタ reg へのデータ転送

```
reg := .in ;
```

は、状態、WAIT 条件、タスクを転送条件とし、FSM のデータ転送クロックをクロックとするクロック付き条件代入文

```
IF sta & wait & task1 THEN
  AT RISE(.clkd) DO
    reg := .in ; END_DO END_IF;
```

となる。

モジュール配下にあるオートマトン割り込み文、タスク発生文等の RTL 動作文のマッピングは、後に述べる RTL 動作文マッピングで実行される。

状態・タスクが、レジスタ等にマッピングされるのは、状態最適化、コード割り当ての後の、FSM モジュール生成においてである。

3.3 関数マッピング

RTL 言語の用意するシステム関数、ユーザ定義関数及び合成用に設けた内部関数を仮想モジュールにマッピングする。引数に応じた入力ピンと出力ピンを以下のようにテンプレートに定義する。

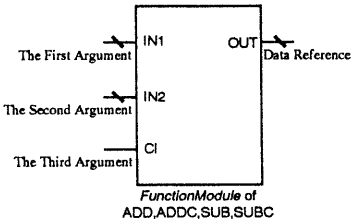


図 3: 関数マッピング例 (加減算器)

内部関数には、条件信号によりデータをスイッチするPASS関数等がある。PASS関数は、条件文をマッピングするときに現れる。

ブーリアン関数は対象外である。

3.4 RTL動作文マッピング

全てのRTL動作文は、単純代入文とブーリアン関数参照文、内部関数参照文にマッピングされる。ステートメント中のファシリティは、参照される動作に対応する仮想モジュールのピンに置き換えられる。

クロック付き条件代入文

```
IF .cond THEN AT RISE(.clk) DO
      reg := .data ; END_DO ; END_IF;
```

をマッピングすると、表2のレジスタの仮想モジュールに従って、次のようなレジスタregの各ピンとの接続を表す単純代入文にマッピングされる。

```
reg.CLK   := .clk ;
reg.D     := .data ;
reg.COND  := .cond ;
```

3.5 FSMモジュール生成

FSMの仮想モジュールを、FSMとタスクの機能を実現するレジスタとそれらの周辺回路にマッピングする。

状態割り当てルーチンと連係して、状態 / タスクを状態 / タスクレジスタに割り当てる。また、FSMのリセット、状態遷移、割り込み回路を生成する。

3.6 関数モジュール生成

関数マッピングが関数モジュールとピンにマッピングするのに対し、関数の機能を実現するモジュール内部の回路を生成する。

回路をブール式を用いて記述できる。テンプレート記述をビット幅可変にしておくことにより、モジュール・コンパイラあるいはソフトマクロの記述として用いることができる。

回路にセルライブラリ中のスタンダードセルやマクロセルを含めることもできる。この場合にはテクノロジー依存になるが、?? 節に述べる**総称名**を用いることにより、ライブラリ依存性を低減できる。

3.7 メモリ・マッピング

仮想モジュールにマッピングされたメモリファシリティ (レジスタ, ラッチ,RAM,ROM) をライブラリ中の実セルを用いた回路にマッピングする。

3.7.1 セルの選択

例えばレジスタのマッピングでは、クロック付き代入文、クロック付き条件代入文のクロック種別と、レジスタ・リセット文、レジスタ・プリセット文による参照の有無によって、セルが選択される。

クロックの種別には、RISE、FALL、HIGH、LOWがある。RISEであれば、ポジティブ・エッジトリガのレジスタが選択される。さらにレジスタ・リセット文で参照されていれば、リセットピン付きのセルにマッピングされる。HIGH、LOWであれば、レベル・センシティブなラッチが選択される。

レジスタの型の選択、例えばD型かJK型かは、それぞれにマッピングするテンプレートを選択することによって行なう。

3.7.2 内部回路

レジスタの仮想モジュールのピンが、実レジスタセルのピンに直接対応するわけではない。レジスタの動作モデルは、データ転送条件が成り立たない時に以前の値を保持するようになっているが、実セルがデータ保持できるとは限らない。また、ピンの極性も異なる。そこで、ファシリティの動作モデルに合うように、仮想モジュールの内部回路を、実セルと周辺回路を用いて生成する必要がある。

周辺回路を、実セルではなくブール式を用いて記述し、仮想モジュールの外側の回路と合わせて最適化の対象とすることができる。

図4、5にレジスタをDフリップフロップとJKフリップフロップを用いた回路にマッピングする例を示す。

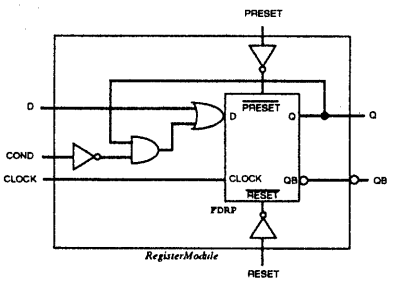


図 4: メモリマッピング例 (D-FF)

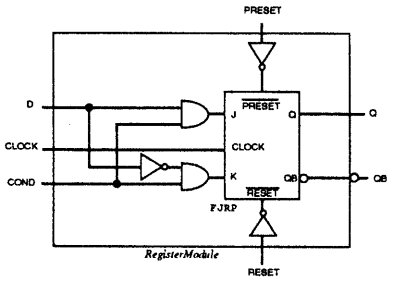


図 5: メモリマッピング例 (JK-FF)

3.8 定数マッピング

定数マッピングは、定数ファシリティの処理を行なう。0,1 の論理出力ピンを持つセルのピンにマッピングする方法と、電源ピン等のモジュールの外部ピンにマッピングする方法がある。

3.9 組合せ回路マッピング

組合せ回路マッピングは、従来のテクノロジーマッピングである。FSM モジュール生成、関数モジュール生成、メモリマッピング後に、レジスタ等の実セルの周辺回路と合わせて組合せ回路を切り出し、ライブラリ中のセルにマッピングする。

4 マッピング・テンプレート記述

前章で述べた合成過程の各段階を記述するマッピング・テンプレート記述形式を検討し、C++に準じた記述言語を提案する。テンプレート記述言語は、C++の operator overloading 機能を用いて記述できるようにしたため、C++コンパイラによって容易に言語処理系を実現できる。

テンプレートでは基本的に、仮想モジュールのピン定義、RTL 動作文要素定義、回路 (構成部品間の結線) を表現する。その他に、ピンや部品の属性を問い合わせたり、ライブラリ中のセルを探索する関数がある。

4.1 総称名

テンプレートのテクノロジー依存性 (ライブラリ依存性) を低減させるため、同一機能の実セル及びピンに対して共通な「総称名」を与え、これを用いてテンプレートを記述する。

したがってセル・ライブラリには、実セル名と「総称名」を登録する。同一機能で面積や遅延の異なるものでも同一の総称名を持つが、テクノロジーマッピングでは、それらから性能を基に選択する。

以下にセルとピンの総称名の例を示す。

表 4: 総称名の例 (セル)

総称名	機能
BFI	Inverter
BFZB	Tristate/Buffer
FDRP	D-FF with $\overline{\text{Reset}}$ and $\overline{\text{Preset}}$
FJRP	JK-FF with $\overline{\text{Reset}}$ and $\overline{\text{Preset}}$
LD	D-Latch($\overline{\text{Hold}}$)

表 5: 総称名の例 (ピン)

総称名	機能	セル名
Q	Data Output	FDxx,FJxx,
QB	$\overline{\text{DataOutput}}$	LDxx
CLOCK	Clock	FDxx,FJxx
D	Data Input	FDxx,LDxx
J	J Input	FJxx
K	K Input	
RESET	$\overline{\text{Reset}}$	xxRx
PRESET	$\overline{\text{Preset}}$	xxPx
HOLD	$\overline{\text{Hold}}$	LDxx

4.2 仮想モジュールのピン定義

ファシリティ、FSM、関数マッピングにおける仮想モジュールのピンのピン名、信号種別、ピン属性をテンプレートに定義する。ピンの属性は、表 2、3に挙げたものの他に、関数の引数の位置を示す、"ARG1","ARG2" 等がある。

図 2のレジスタのファシリティ・マッピングにおけるテンプレートのピン定義例を示す。

```
Pins("D",      INPUT, ASGN);
Pins("PRESET",INPUT, PRESET);
Pins("RESET", INPUT, RESET);
Pins("COND",   INPUT, COND);
Pins("CLOCK", CLOCK, CLOCK);
Pins("Q","QB",OUTPUT,REF);
```

図 6: ピン定義例 (レジスタ)

ここで、"Q","QB" は相補の関係になっているピンを表す。

4.3 RTL 動作文マッピング定義

RTL 動作文の要素を表 6に示す。これらの要素を接続した合成回路をテンプレートに記述する。

表 6: RTL 動作文の要素と参照関数

要素	関数
ソース	SOURCE()
デスティネーション	DESTINATION()
クロック	CLOCK()
条件	CONDITION()
アドレス	ADDRESS()

3.4節で示したクロック付き条件代入文のマッピングは、仮想モジュールのピン定義で指定されたピン属性を用いて、次のように定義される。

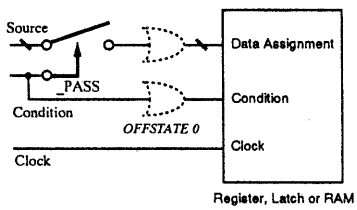


図 7: クロック付き条件代入文のマッピング回路例

```
1 DESTINATION(ASGN) =
2  _PASS(SOURCE(),CONDITION(),OFFSTATE());
3 DESTINATION(COND) = CONDITION();
4 DESTINATION(CLK)  = CLOCK();
```

接続を表す代入文のデスティネーション側 (左辺) のファシリティの仮想モジュールは、複数の入力ピンを持つので引数にピンの属性を記述するが、ソース側 (右辺) は全てデータ出力 (参照) 属性 "REF" のピンが来るので属性が省略されている。

図 7テンプレート記述例の 1、2 行目は、PASS スイッチのデータ入力に RTL 動作文のソースを、制御入力に文の条件を接続し、スイッチの出力をデスティネーションの仮想モジュールピンのうち、"ASGN" 属性のピンに接続することを表している。この時、OFFSTATE 値は、元の RTL 動作文の値が継承される。

4.4 合成回路定義

FSM モジュール生成、関数モジュール生成、メモリマッピングにおいて、仮想モジュール内部の回路を定義する。ライブラリ中のセル等、サブモジュールのピン間の接続をブーリアン演算子を用いて記述できる。

演算子 = は、単純代入文の生成を表す。演算子 |、&、 ^、 ~、 |=、 &=、 ^= は、それぞれブーリアン関数代入文の OR, AND, XOR, NOT, OR-reduction, AND-reduction, XOR-reduction を表す。

サブモジュールの生成、サブモジュールピン、モジュールピンの参照は、関数 NewSUBMODULE(), SUBMODULE(サブモジュール).PIN(ピン名), XPIN(ピン名) によって行なう。

また、for 文によって、ビット幅可変の回路を定義することができる。

以下に、メモリマッピングと関数モジュール生成の定義例を挙げる。

4.4.1 メモリマッピング例

以下に、図 4のレジスタのメモリマッピング例を示す。

```
1 SUBMODULE* reg = NewSUBMODULE("FDRP");
2 SUBMODULE(reg).PIN("CLOCK") =  XPIN("CLOCK");
3 SUBMODULE(reg).PIN("D") = XPIN("D") |
4   (~XPIN("COND")&SUBMODULE(reg).PIN("Q"));
5 SUBMODULE(reg).PIN("PRESET") = ~XPIN("PRESET");
6 SUBMODULE(reg).PIN("RESET")  = ~XPIN("RESET");
7 XPIN("Q")       = SUBMODULE(reg).PIN("Q");
8 XPIN("QB")      = SUBMODULE(reg).PIN("QB");
```

図 8: メモリマッピング例 (レジスタ)

第 1 行では NewSUBMODULE() によって、ライブラリからリセット・プリセット付き D 型 FF(総称名 "FDRP") を選択し、サブモジュール (reg) として生成している。
第 2 行では、サブモジュール (reg) のピン (総称名 "CLOCK") に、マッピングするレジスタの仮想モジュールの外部ピン (ピン名 "CLOCK") を接続している。

4.4.2 関数モジュール生成例

以下に、図3に示した関数仮想モジュールのリップルキャリ加算器による実現例を示す。これは1ビットの全加算器をサブモジュールとして用い、for文によって関数のビット幅に応じた回路を生成している。

```
XPIN("CO")  =  (XPIN("IN1") & XPIN("IN2"))
            |  (XPIN("IN2") & XPIN("CI"))
            |  (XPIN("CI")  & XPIN("IN1"));
XPIN("OUT") =
   (XPIN("IN1") & XPIN("IN2") & XPIN("CI"))
 | ((XPIN("IN1") | XPIN("IN2") | XPIN("CI"))
 & ~XPIN("CO"));
```

図 9: 関数モジュール生成例 (全加算器)

```
int N = BitWidth();
SUBMODULE* fadd = NewSUBMODULE(N,FADD);
SUBMODULE(fadd).PIN("IN1")  =  XPIN("IN1");
SUBMODULE(fadd).PIN("IN2")  =  XPIN("IN2");
SUBMODULE(fadd,0).PIN("CI") =  XPIN("CI");
SUBMODULE("OUT")  =
        SUBMODULE(fadd).PIN("OUT");
for ( int i=1; i<N; i++ ){
    SUBMODULE(fadd,i).PIN("CI") =
        SUBMODULE(fadd,i-1).PIN("CO");};
```

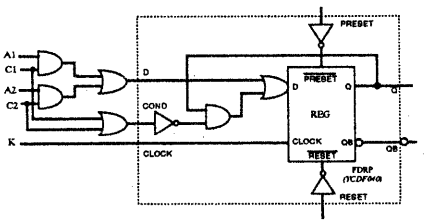
図 10: 関数モジュール生成例 (リップルキャリ加算器)

4.5 合成例

マッピング過程のモデリングとテンプレート記述は、各マッピングにおいて、できるだけ局所的に処理できるように注意深く定義している。

ここでは、次のクロック付き条件代入文を例にマッピング過程を追う。

```
AT RISE(K) DO IF C1 THEN
  REG := A1 OFFSTATE 0 ; END_IF; END_DO;
AT RISE(K) DO IF C2 THEN
  REG := A2 OFFSTATE 0 ; END_IF; END_DO;
```



1. ファシリティ・マッピング：レジスタ REG を仮想モジュール (破線) にマッピング
2. RTL 動作文マッピング：2つの動作文を REG との接続にマッピング (破線の外)
3. 関数モジュール生成：動作文の OFFSTATE が 0 なので、PASS 関数を AND で、信号の衝突は OR で実現
4. メモリマッピング：仮想モジュール内部回路を D-FF を用いて実現 (破線内)

論理ゲートは、後の段階で最適化され、組合せ回路マッピングされる。

5 まとめ

多様な設計手法に対応するため、合成過程をオープンにしたテンプレート化合成手法を提案した。本手法の特長は、以下の通り。

- 可読性の高いテンプレート記述言語により、設計手法に応じた合成回路構成のカスタマイズが容易。
- 合成システムのメンテナンスが容易
- テンプレート記述によりモジュール・コンパイラを実現
- RTL 言語の関数記述を用いて、容易に高パフォーマンスな演算器 (ハード / ソフトマクロ) を利用可能
- 総称名の導入によるテンプレートのライブラリ依存性の低減
- RTL 言語の変更・拡張に対して柔軟に対処可能

なお、本論文で述べた合成手法に基づいた論理合成システム SERAPHIM の基本部分は、SYNTHESISKIT として、日本電子工業振興協会 UDL/I 標準化委員会に開示する予定である。

参考文献

[1] UDL/I 標準化委員会. UDL/I 言語仕様, 1.0m edition, May 1992.

[2] 中村行宏, 小栗清, 野村亮. RTL 動作記述言語. 電子情報通信学会論文誌, No. 10, pp. 1570–1593, 1989.