

COINSを用いた
高位合成器の実装

東京工業大学
理学部
情報科学科

五十嵐巳玲一
(0501699)

平成20年度卒業論文

指導教官 佐々 政孝 教授

平成21年2月5日

目次

第1章	はじめに	3
1.1	背景	3
1.2	研究動機	4
1.3	概要	4
第2章	COINS について	5
2.1	COINS の構成	5
2.2	HIR	7
第3章	準備	12
3.1	VHDL	12
3.1.1	回路エントリ手法の変移	12
3.1.2	HDL とは	12
3.1.3	ハードウェア記述言語での回路設計	14
3.1.4	VHDL の文法	16
3.2	SPARK	18
3.2.1	SPARK 出力コードの例	18
第4章	実装	22
4.1	目標とする仕様	22
4.2	実装のアプローチ	22
4.2.1	hir2c の概要	22
4.3	実装の詳細	23
第5章	評価	27
5.1	テストプログラムとその方法	27
5.2	結果	27
5.3	総評	32
第6章	関連研究	34
第7章	終わりに	35

目 次

1.1	動作記述のイメージ	3
1.2	RTL 記述のイメージ	3
2.1	COINS の構成	5
2.2	HIR の論理構造	8
2.3	入力プログラム	9
2.4	HIR 出力の例	10
3.1	回路エントリ手法の変移と主なデバイス	13
3.2	LSI 開発の設計フロー	14
3.3	エンティティとアーキテクチャのイメージ	16
3.4	SPARK の構成	18
3.5	入力プログラム 1	18
4.1	モデルイメージ	23
4.2	main 関数がない場合の例	25
4.3	追加された変数の例	26
4.4	while 文の例	26
5.1	テストプログラム	27
5.2	テストプログラム	28
5.3	論理合成後	29
5.4	図 (5.3) の内部 1	29
5.5	図 (5.3) の内部 2	30
5.6	図 (5.3) の内部 (簡略化した場合)	31
5.7	case 文を含む入力	32
5.8	図 (5.7) での SPARK 出力 (一部省略)	33

第1章 はじめに

1.1 背景

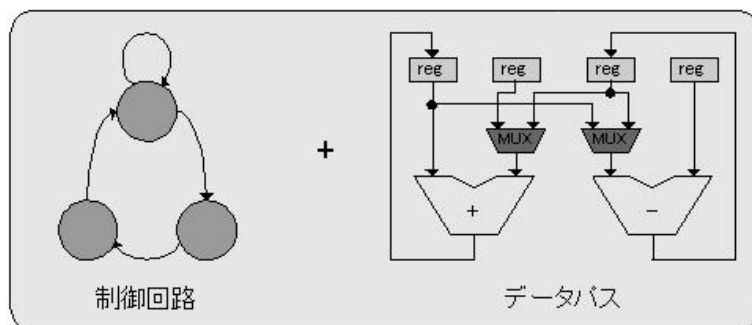
ハードウェア記述言語¹は回路の動作や構造を記述する言語であり、回路設計で従来のように回路図を書く必要がなく作業効率が大幅に向上した。しかし、LSIの集積度のさらなる増加により設計能力の限界が生じてきている。

この問題を解決するために考え出されたのが、高位合成である。高位合成とは、ある処理を実行する専用ハードウェアを設計する際に、対象となる処理のアルゴリズムを直接的に記述した「動作記述」から、レジスタやクロックによる同期などハードウェアに特有の概念を意識した「RTL²記述」を自動的に合成する処理を示す。

高位合成の出現によって作業効率がさらに向上し、これからの回路設計に欠かせない技術である。(図 1.1)(図 1.2)

```
...  
x = a + b;  
y = b - c;  
s = x + y + d;  
t = x - c;  
...
```

図 1.1: 動作記述のイメージ



実際には、VHDLやVerilogなどの言語で記述される

図 1.2: RTL 記述のイメージ

¹以下、HDL

²Register Transfer Level

1.2 研究動機

COINS [1] とは、新しいコンパイラ方式を容易に実験、評価できるような共通インフラストラクチャーである。COINS を用いることによって、新言語のコンパイラを作成したり、最適化等の評価ができる。

高位合成をテーマとした研究をするにあたって COINS を使ってフリーソフトの高位合成器を作れないかと考えた。高位合成器は実現してはいるものの未だ研究段階である。学生や研究者が簡単に拡張、実験ができる COINS に高位合成器を実装することで高位合成の研究が盛んになることを願い本研究を進めることにする。

1.3 概要

本研究は COINS へ高位合成器を実装する第一段階として、COINS の hir2c³機能を変更して、C コードから VHDL コードを出力するツールを実験的に作成する。このとき出力する VHDL は動作記述とする。本来の高位合成器では RTL 記述であるべきだが、RTL 記述への対応は別の研究で行うこととする。

実装した機能は以下である。

- 型は int 型と論理型のみ
- 四則演算と代入文
- if 文、while 文
- 簡単な関数呼び出し

以下 2 章で本研究で使用した COINS とその HIR について説明し、3 章ではモデルとなる高位合成器である SPARK についてとターゲットとなる HDL である VHDL について、4 章、5 章では実装とその評価について記述する。

³hir を C 風に変換するツール

第2章 COINS について

本研究のひな型となる処理系は COINS (COmpiler INfraStructure) を使用する。以下、COINS について説明する。

2.1 COINS の構成

一般にコンパイラは、フロントエンド (front end) とバックエンド (back end) から構成される。フロントエンドは、原始プログラム (source program) を中間コード (intermediate code) と呼ばれる内部形式に変換する。バックエンドは、中間コードを計算機の機械コード (machine code) に変換する。フロントエンドはさらに、字句解析器 (lexical analyzer)、構文解析器 (syntax analyzer)、意味解析器 (semantic analyzer) に分けられる。バックエンドは、最適化器 (optimizer) とコード生成器 (code generator) に分けられる。これら各部分は、コンパイラのフェーズと呼ばれる。COINS の構成は (図 2.1) を見てもらいたい。

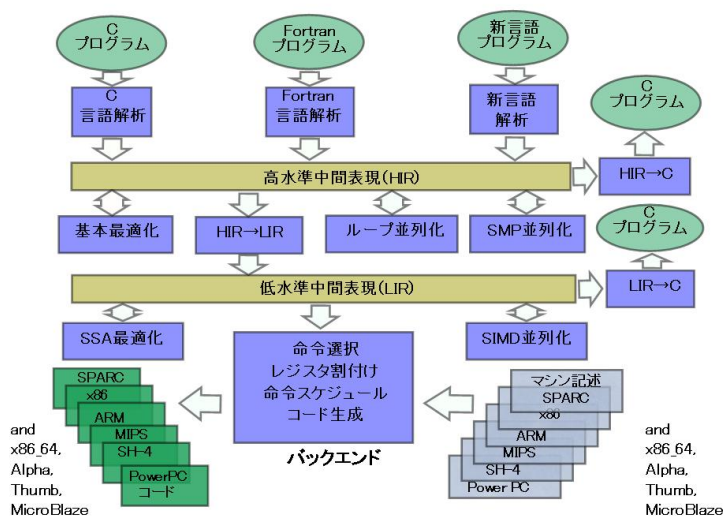


図 2.1: COINS の構成

また、特徴としては

- 高水準中間表現 (HIR) と低水準中間表現 (LIR) の二つの中間表現をもつ

ソース言語のレベルに近い高水準中間表現 (High-Level Intermediate Representation、HIR) と、機械語のレベルに近い低水準中間表現 (Low-level intermediate representation、LIR) の2つがあり、ソースレベル、マシンレベルどちらの処理も記述するのが比較的容易である。

- すべて、Java で書かれ新規に開発されている

メモリ管理に気を使わずにすむ、エラーチェックの機能による信頼性の向上、Java プラットフォームが独立である、等の理由で Java での開発となっている。また、他の高級言語に比べて実行速度が遅いことは懸念であるが、昨今のコンパイラ記述の進歩とハードウェア性能の向上とにより、問題は少ないと考える。

- 最適化の機能が充実している

2水準の中間表現それぞれについて最適化を考えることができる。最適化は、SSA¹最適化に関して10種類ほど、またループ並列化、インライン展開、ループ展開なども備わっており、それぞれに関して性能が評価できる。

- リターゲットブルなコード生成系をもつ

COINS でのバックエンドの処理はすべて、LIR から LIR への変換としてマシンに依存しない形で表現できる。つまり、すべてのバックエンドにおける最適化結果が LIR で表現され、対象となるマシンが変わったとしても気にしないで開発を進められるということである。

- 並列マシンのための並列化の機能を持つ

並列コンパイラとしては基本的な機能を有す。ループの do-all 方の並列化、SMP (symmetric multi-processor) 用の並列化として粗粒度並列化の基本的な機能を持っている。

- SIMD 並列化の機能を持つ

SIMD (single-instruction multiple-data) 命令とは、例えば、64 ビットのレジスタを 8 ビット × 8 のベクトルレジスタのように使って並列に実行してしまう命令などである。技術的に大変難しいものであるため、実用的に使用できる水準ではないが、実装してある。

が、あげられる。また、一般的なコンパイラの最適化や処理過程に関しては文献 [4]、COINS を用いたコンパイラの設計等は [2] を参照されたい。

¹Static Single Assignment form 静的単一代入

2.2 HIR

COINS の中間表現である HIR では HIR から C 言語への逆変換ツールが備わっている。本研究ではこのツールを参考にして実装を施した。

高水準中間表現は、入力言語の論理構造を表現できるとともに、多くの言語に共通する表現でなければならない。プログラミング言語の具体形式は、言語ごとに大きく異なる。しかし、現行の手続き型言語を見ると、演算式、代入文、if 文、反復文、整数型、浮動小数点型、配列、構造体など、共通する概念を持つ。具体形式にとらわれずに、それらの処理操作と処理対象を抽象化してゆくと、多くの言語に共通する論理構造を持つ中間表現を設定することができる。

そのようにして定めた中間表現が本コンパイラ・インフラストラクチャの高水準中間表現 HIR (High-level Intermediate Representation) である。

HIR では、処理操作はオペレータとそのオペランドとして表現される。オペランドとしては処理操作の結果も許されるので、オペレータとそのオペランドを線で結んでゆくと、処理操作は木構造として表現できる。処理対象はスカラー量と配列、構造体、共用体として表現できる。

HIR の特徴として

- 現存の多くの手続き型言語に対応化
- 記述水準は C 言語に近く、HIR から C への逆変換が可能
- 式、代入文、条件文、ループ文など、高級言語の諸機能とほぼ 1 対 1 に対応するファクトリーメソッドを備え、容易に HIR を作成できる。
- HIR の走査、変換を行う各種メソッドの完備
- 記号表 (symbol table) に関する各種メソッド完備、シンボルの登録、探索、シンボルテーブルのネストの作成
- すべてを知らなくても使用できるように、基本的なインターフェースを用意 (HIR0.java、Sym0.java)

これらによりフロントエンドの作成が容易となる。

HIR の論理構造

HIR の論理構造を図 (2.2) に示す。

HIR // High-level Intermediate Representation.

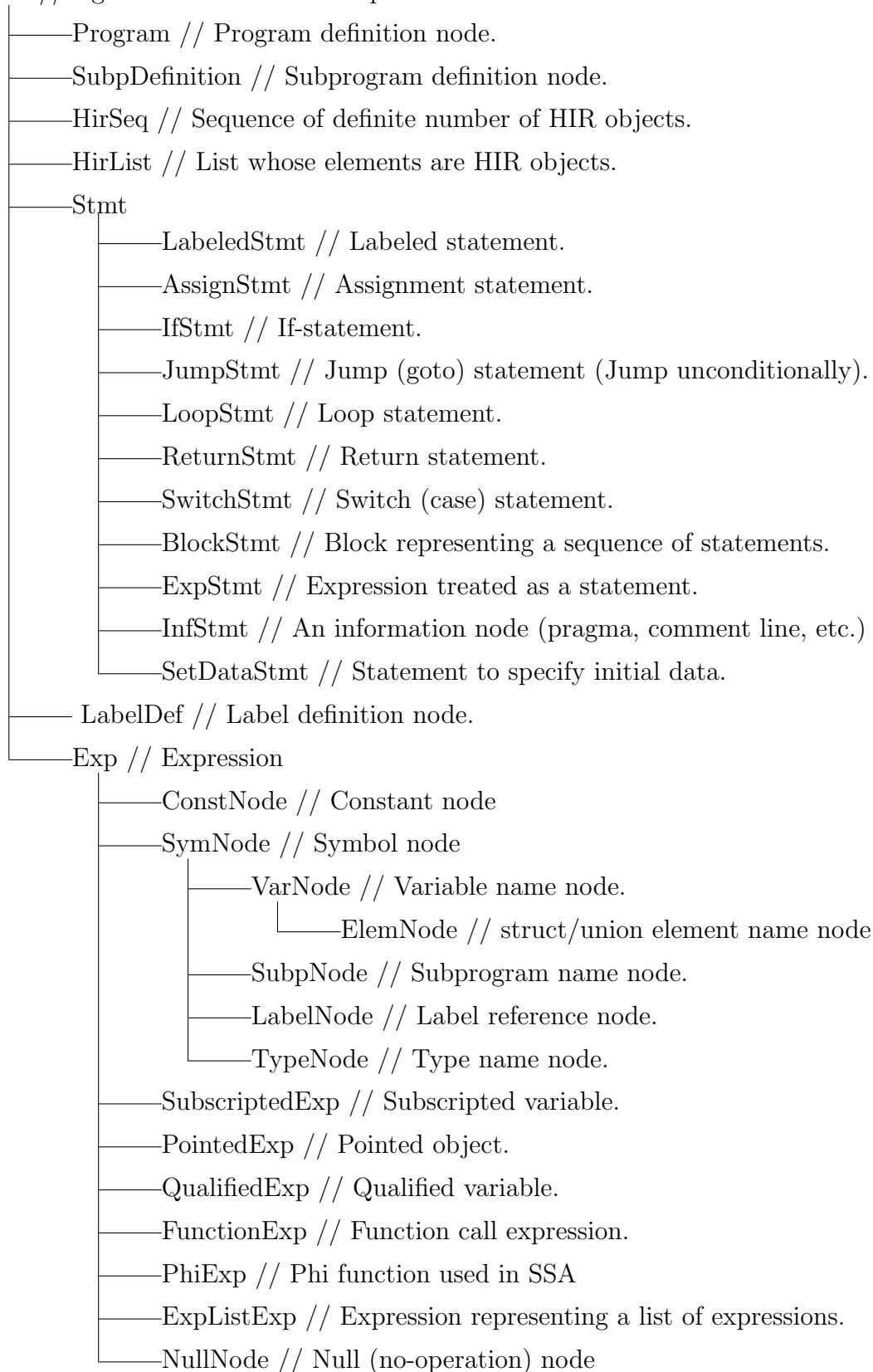


図 2.2: HIR の論理構造

具体表現

HIR をテキストで表すとき, 節は

(オペレータ 型 第1子 第2子 … 第n子)

葉は

< 種別 記号 型 >

と表す.

以下に HIR の例を示す。

— 入力 —

```
int fact( int p)
/* fact0.c: Factorial function */
{
  if (p <= 1)
    return 1;
  else
    return p * fact(p - 1);
}
```

図 2.3: 入力プログラム

```

(prog      1
  <null 0 void>
  <nullNode 2>
  (subpDef 3 void
    <subp   4 <SUBP < int > false false int> fact>
  <null 0 void>
  (labeldSt 5 void
    (list 6
      <labelDef 7 _lab1>)
    (block   8 void file fact0.c line 2
      (if     9 void file fact0.c line 4
        (cmpLe 10 bool _XId1
          <var   11 int p _XId2>
          <const 12 int 1>)
        (labeldSt 13 int
          (list 14
            <labelDef 15 _lab3>)
          (return 16 int file fact0.c line 5
            <const 17 int 1>)))
        (labeldSt 18 int
          (list 19
            <labelDef 20 _lab4>)
          (return 21 int file fact0.c line 7
            (mult 22 int _XId3
              <var   23 int p _XId2>
              (call 24 int _XId4
                (addr 25 <PTR <SUBP < int > false false int>> _XId5
                  <subp 26 <SUBP < int > false false int> fact>)
                (list 27
                  (sub 28 int _XId6
                    <var   29 int p _XId2>
                    <const 30 int 1>))))))))
          (labeldSt 31 void
            (list 32
              <labelDef 33 _lab5>)
            <null 0 void>))))))

```

図 2.4: HIR 出力の例

また、HIR レベルでは以下の機能が備わる。

- 基本最適化

制御フローグラフの作成、データフローの解析、別名解析、定数畳み込み・伝播、共通部分式削除・無用命令削除、ループ展開・インライン展開

- 並列化

ループ解析・並列化、粗粒度並列化

- C 出力

OpenMp 出力

HIR についての詳しい仕様や HIR での最適化については文献 [3, 11] 等を参照されたい

第3章 準備

本研究に必要な知識として、VHDL とその文法、SPARK の出力、について説明する。

3.1 VHDL

本研究で出力コードとなる VHDL [12] について説明したいと思う。

3.1.1 回路エントリ手法の変移

回路設計は当初は紙に手書きで行っていたが、設計量の増加と技術の発展とともに新たな試みがなされてきた。紙面上の設計から計算機ベースの設計へと変わり CAD ツールの発達によって飛躍的に効率が上がる。しかし、回路の集積度の増加により更なる作業効率の増加が要求され、抽象度のより高いプログラミング言語での回路設計が登場する。その言語がハードウェア記述言語 (Hardware Description Language:HDL) である。

また、HDL での設計も限界に来ていると言われ、次世代の回路設計は HDL よりも中小度の高い高級言語、例えば C 言語などによる設計も研究されている。その際、抽象度の高いプログラミング言語から HDL へと変換するツールが高位合成器であり、本研究のテーマとなるものである。図 3.1 に回路エントリ手法の変移と主なデバイスについて示す。

3.1.2 HDL とは

HDL は電子回路の経時的振舞いと空間的構造を表現する。ソフトウェアのプログラミング言語と比較すると、HDL の文法や意味論には、ハードウェアの基本的属性である時間や並行性の明確な記法を含んでいる。主に普及している言語としては、VHDL と Verilog-HDL がある。比較すると、

- VHDL

曖昧な定義は許されず、定義に対して厳密さを要求される。そのため、定義などが省略不可能なため記述量は多い。

- Verilog-HDL

プログラム記述に近く、曖昧な表現も許される。よって宣言、定義等が省略可能なため記述量は少ない。

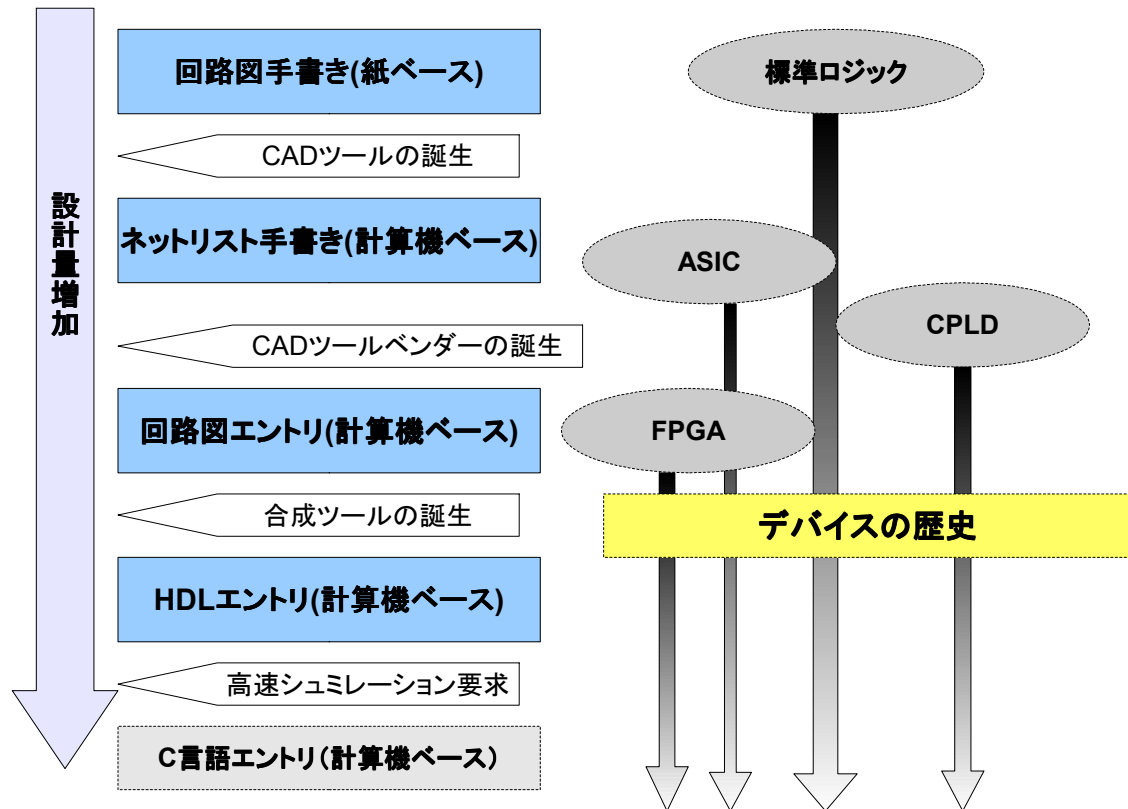


図 3.1: 回路エントリ手法の変移と主なデバイス

この二つについては国際基準 (IEEE) である。

また、HDL で記述できる抽象度のレベルは高い順に

- behavioral
 - 一般的に論理合成は不可ではあるが簡単なプログラムにおいては可能であり、本研究の評価においてそれを実証済み。大規模なプロジェクトにおいての上流工程において作業の効率を求められる際に使用したり、高速シミュレーションのために記述する。
- RTL(Register Transfer Level)
 - 論理合成可能で、デバイス製造データ作成のために記述。
- Logic
 - 回路図・ゲートレベルの記述

とあるが、以上のすべての抽象度において記述可能である。

本研究では、モデルとなる高位合成器 (SPARK) がフリーで手に入れられたので、VHDL を対象の言語とした。

3.1.3 ハードウェア記述言語での回路設計

本節では、HDL での開発について説明する。

設計フロー

VHDL での LSI 設計について図 (3.2) に示す。この図は LSI 開発の一般的な例であり、実際は多くの後戻り工程があるが、ここでは省略する。

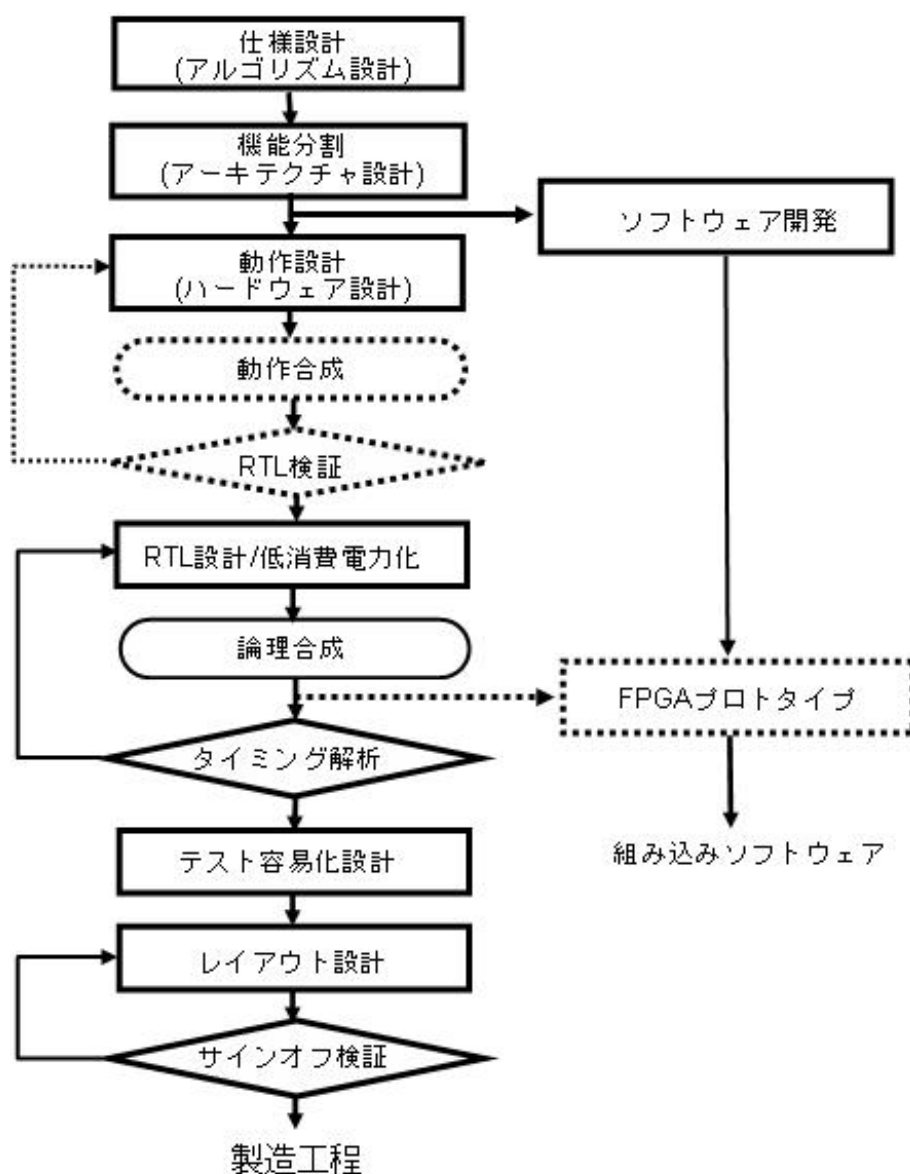


図 3.2: LSI 開発の設計フロー

- アルゴリズム開発 (仕様設計)

アルゴリズム開発は製品をどのような仕様で開発するかを決める工程である。必要なデータや情報をどのようなアルゴリズムや手順に従って処理するかを検討する。次の工程を意図して、効率的で品質のよいアルゴリズムを開発することが重要である。

- アーキテクチャ設計

この工程ではアルゴリズムをどのような構造で実現するかを検討する。アーキテクチャの実現には、CPU や DSP 等の汎用プロセッサや汎用 LSI を用いて組み込みプログラムを開発するソフトウェア開発と、アルゴリズムに対する専用 LSI を開発するハードウェアによる実現方法がある。ソフトウェアによる実現は、ハードウェア実現化に比べて実行処理に時間がかかるが、プログラムの変更が容易であり、製品化した後でも機能の変更が容易。一方、ハードウェアによる実現は、専用 LSI や専用ボードを開発するため高い開発コストがかかるが、高速処理可能である。

- RTL 設計

Verilog HDL または VHDL を用いて回路を記述するのが一般的である。回路内部のすべての動作をクロックサイクルに同期して記述する必要があり、記述に時間がかかり、シミュレーションにも時間がかかるが、論理合成後のゲート回路は RTL で直接的に制御できるメリットがある。

- 低消費電力設計

低消費電力が要求される LSI の場合に、RTL 設計もしくはゲートレベルの段階で回路構造を検討する。

- 論理合成

RTL 記述から論理合成ツールを用いて、ゲート回路を自動的に生成する。論理合成は、HDL 記述からクロック周期、消費電力等の設計規約条件にしたがって自動的に論理ゲート回路を最適化し、論理ゲート回路を生成する。

- タイミング解析

論理ゲートは固有な遅延値をもち、論理ゲートの遅延と配線による遅延を計算することでゲート回路の遅延値が得られる。論理合成されたゲート回路が設計規約条件やセットアップ時間、ホールド時間などのタイミング条件を満足しているかどうかを確認する。

- テスト容易化設計

LSI の製造時には配線の、ショート、オープン、トランジスタ不良などの製造不良が発生する可能性がある。そのテストを容易にするのがこの工程である。

- レイアウト設計

ゲート回路に対して LSI の製造に必要なマスクを作成するためのレイアウト設計を行う。フロアプランではレイアウト設計を実行する前に LSI のチップ上で、どの回路ブロックをどこに配線するかの大枠を決める。

- サインオフ設計

レイアウト設計終了後、配線上を検証環境に戻して、LSI が機能やタイミング仕様を満足しているかを最終的に検証する。

- 製造工程

LSI を製造するために必要なマスクを作成し、製造工程に入る。完成品をテストし出荷となる。

高位合成器を導入した設計の例

図 (3.2) の点線で囲んであるブロックの、動作設計、動作合成と RTL 検証の部分に注目いただきたい。工程に動作合成 (高位合成) を導入する場合は、まず高位合成可能な記述において動作設計をする。その後、高位合成ツールで高位合成を行う。この工程でできた RTL を RTL 設計後の工程でそのまま使える。HDL で設計する工程での労力をかなり削減できるだけでなく、昨今では消費電力を考慮した高位合成器も実現している。これからの LSI 開発はこの工程を加えたものになる予想ができる。

3.1.4 VHDL の文法

VHDL の基本構成は以下のようにになっている。

1. 使用するライブラリ宣言 (ライブラリ)
2. 外部インターフェイス (エンティティ)
3. 内部動作 (アーキテクチャ)

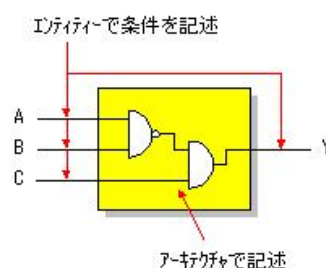


図 3.3: エンティティとアーキテクチャのイメージ

ライブラリ

ライブラリとは基本的なデザインデータの集まりで、パッケージ宣言、エンティティ宣言、アーキテクチャ宣言などが含まれる。その種類であるが、ユーザが記述した VHDL が保存され、ライブラリ宣言が不要な WORK、VHDL の標準仕様のライブラリでライブラリ宣言が同様に不要である STD、外付けのライブラリで、ライブラリ宣言が必要な IEEE、ほかにベンダが提供するライブラリがある。また、VHDL では、パッケージと呼ばれる記述を用いて資源の共用を計ることができ、ベンダが提供するものとユーザが作るオリジナルなものがある。

```
--ライブラリ宣言--
library IEEE;
--パッケージ--
use IEEE.std_logic_1164.all; --IEEEの標準のパッケージ
use IEEE.std_logic_arith.all; --算術演算用のパッケージ
```

エンティティ

外部とのインターフェイスを記述する。

```
ENTITY エンティティ名 IS
port(
--ポート名 方向 データタイプ
x: IN std_logic;
y: IN std_logic;
z: out std_logic);
END エンティティ名;
```

以上のように記述する。エンティティではポート名(信号の名前)、方向(インプットかアウトプットか)、データタイプ(std_logic など)

アーキテクチャ

内部の動作や、内部の接続を記述する。

```
ARCHITECTURE アーキテクチャ名 OF エンティティ名 IS
内部信号の宣言
begin
z <= x and y;
end アーキテクチャ名;
```

以上のように記述する。アーキテクチャ内では内部信号があれば宣言し、内部の動作を記述する。エンティティが箱と例えると、アーキテクチャは箱の中身といえるだろう。

3.2 SPARK

SPARK [6, 5] はANSICを入力としVHDLを出力するツールである。本研究ではSPARKの出力コードをモデルとする。図3.4にSPARKの構成を示す。

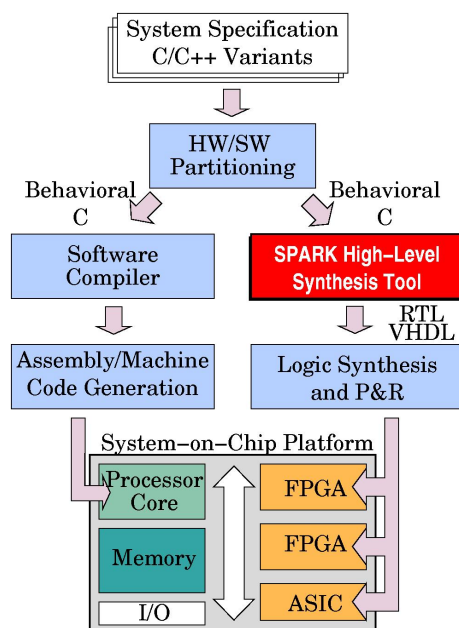


図 3.4: SPARK の構成

3.2.1 SPARK 出力コードの例

では、図3.5のCコードを入力とし実際にSPARKを実行する。コマンドラインは`SPARK -hvbfile.c`を使用する。このコマンドラインは何もオプションを指定せず動作記述のVHDLを出力する。

```
program
int x;
int add(int y){
int z;
z = x+y;
return z;
}
```

図 3.5: 入力プログラム 1

ライブラリ宣言

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

library work;
use work.spark_pkg.all;
```

ライブラリ宣言の部分は、IEEE のライブラリのデータタイプ、シグナルなどの基本的なもの、算術演算用のパッケージ記述と SPARK 独特のパッケージの宣言となる。SPARK のオリジナルパッケージではエンティティ、アーキテクチャ部分で使用する新しいデータタイプの定義をしている。このデータタイプは関数の引数、ローカル変数などで関数で使われる変数に対するものである。子の派生パッケージの詳しい記述はライブラリ宣言の前に出力されているがここでは省くものとする。このライブラリ宣言は、入力によって変化するものではない。

エンティティ

```
ENTITY add IS
port(
  y : IN wiredOrInt range -32768 to 32767 ;
  returnVar_ : OUT wiredOrInt range -32768 to 32767 ;

-- global variables are
  x : INOUT wiredOrInt range -32768 to 32767 ;
  CLOCK : IN std_logic ;
  RESET : IN std_logic ;
  done : OUT std_logic );
END add;
```

エンティティに部分の出力は以下ようになる。

- エンティティ名として関数の名前

もし入力プログラムに main 関数があれば main となる。main 関数が存在しない場合は、ソースの一番下に記述された関数となる。

- 関数の持つパラメタ

この例では、y がパラメタである。port 部分に入ってくる信号として記述される。データタイプが wiredOrInt となっているのはライブラリで宣言されたオリジナルのデータタイプである。

- return 文への対応

関数が return 文を求められる場合に、returnVar が宣言される。方向は出力でデータタイプは wiredOrInt となる。この信号に返り値を代入する。

- グローバル変数

プログラム内にグローバル変数があれば宣言する。方向は入力及び出力である。この例では x がグローバルである。

- CLOCK、RESET、done

この信号は入力コードによらず宣言される。CLOCK は動作を行うための信号で、この値が 1 であれば動作を行いそうでなければ何も行わない。RESET は初期化を行う信号、done は動作の完了をしめす。

アーキテクチャ

```
ARCHITECTURE behav OF add IS

PROCEDURE add (
  y : IN wiredOrInt range -32768 to 32767

-- global variables are
  x : INOUT wiredOrInt range -32768 to 32767 ;
) IS
  signal z : wiredOrInt range -32768 to 32767 ;

  BEGIN
    z <= (x + y);
    returnVar_ <= z;
  END add;
signal z : wiredOrInt range -32768 to 32767 ;

BEGIN
PROCESS
  BEGIN
    wait until CLOCK'event and CLOCK = '1';

    z <= (x + y);
    returnVar_ <= z;

  END PROCESS;
END behav;
```

アーキテクチャ部に宣言されるものは以下である。

- PROCEDURE 文

これは、プログラムの副関数に当たるものある。関数のパラメタ、大域変数が宣言され、内部の動作が記述される。

- ローカル変数

関数にローカル変数が使われていれば宣言する。このとき、もし IF 文や while 文など動作に比較が必要な記述があれば、比較に用いる変数も宣言する。この宣言は入力でも出力でもなく、データタイプは論理型である。

- 内部動作

PROCEDURE 文で記述された、または main 関数の動作を記述する。

第4章 実装

4.1 目標とする仕様

処理の流れは以下である。

1. C 言語で書かれたソースを COINS 上でコンパイルする。
2. コンパイルした C 言語は HIR で表現される。
3. HIR では各種の最適化があるが、今回の実装では最適化をかけずに VHDL に変換する。
4. HIR から VHDL に変換する。本研究ではこの部分を主に実装する。

この実装においてサポートする機能を確認しよう。

- int 型、boolean 型
- 四則演算、代入文
- if 文
- while 文
- 簡単な関数呼び出し

4.2 実装のアプローチ

4.2.1 hir2c の概要

hir2c は HIR 表現を C に逆変換する機能である。この機能によって、HIR 上で最適化した結果を C コードに反映することができる。本研究では、実装の雛形として hir2c のソースを使う。

hir2c を使用した時の処理の流れは、C 言語で書かれたプログラムを構文解析を通して、HIR にし、HIR での最適化を施し、C 言語に逆変換するものである。本来は、最適化を施した後のプログラムを評価する際に使用するツールであるが、変換する際の HIR における除法の扱いがしっかりと実装されていることと、HIR が C 言語に非常に近いものがあり、例えば変数の名前が変化せず保存されている、ことから実装が効率的に行えるであろうと予想できるので実装の雛型として hir2c を選ぶことにする。

hir2c のパッケージに記述されているクラスを次に示すである。

- AssociationList.java
HIR の持つ情報をノードととして結合などを行うクラス。
- Hir2C.java
変換の制御を行うクラス。このクラスにおいて、HirBaseToCImpl が呼び出されている。
- HirBaseToC.java
インターフェース。このクラスは HirBaseToCImpl に継承される。
- HirBaseToCImpl.java
変換の詳細が記述されている。
- KeyWords.java
逆変換後の C ソースの予約語等の定義。
- LabelRef.java
ラベルに関する処理を記載。
- PrintDef.java
出力に関するもの。出力コードのマクロ定義などが出力される。

4.3 実装の詳細

実装について、C 言語の構文解析と HIR に変換するまでは COINS の既存のものを用いる、そこで出来上がった HIR を既存の hir2c を雛型として VHDL 変換ツールを実装する。この実装のモデルイメージを図 4.1 に与える。

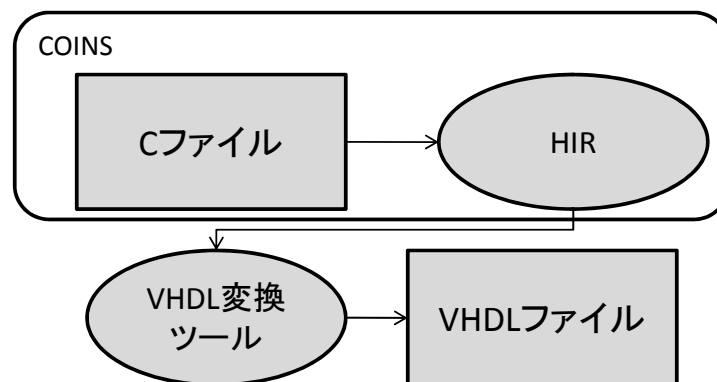


図 4.1: モデルイメージ

ライブラリ部分

ライブラリ部分ではソースによらずきまったコードを出力する。よって hir2c において、マクロ定義を出力する機能をもとに実装を行った。これは hir2c の PrintDef にあたり、ここからかじめ出力する String を定義するだけでよい。ちなみにこの PrintDef は HirBase2CImpl 内で最初に呼ばれる。

hir2c での出力

```
# define hir_t_short short
# define hir___ XOR(a,b) (a) ^ (b)
...
...
```

実装での出力

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
```

エンティティ部分

エンティティで記述するポートは、入力信号として主となる関数の持つパラメタ、ソースによらない CLOCK、RESET、出力信号として returnVar、done、入出力ともにグローバル変数、以上である。

記述する変数の情報は HIR の持つ情報を使用する。HIR は木構造になっているため、その木をたどって行くことによって情報の収集が可能。またその情報を取り出すメソッドも COINS には準備されている。

アーキテクチャ部分

エンティティ部分と同様の手法で HIR の情報を使用する。この時副関数の記述にあたる、PROCEDURE 文のパラメタには、副関数の持つパラメタ以外にもグローバル変数を記述しなければならない。また、main 関数を持たないコードを入力とした場合には、記載された副関数のうち一番最後に記述した関数、つまり HIR の木構造の一番最後の関数を main 関数とみなし PROCESS 文に記述している。

```

program
int func1 (){
    ...
    return 0;
}
int func2(){
    ...
    return 0;
}
int func3(){ //この関数が main 関数とみなされる。
    ...
    return 0;
}

```

図 4.2: main 関数がない場合の例

四則演算、代入文について

四則演算 (+, -, *, /) についてはまず hir2c での出力の予約語を変更しなければならない。予約語を変更するには KeyWord.java 内を変更する。例えば、 $x + y$; というコードを hir2c に入力すると hir... ADD(x, y) というコードが出力される。そこで、KeyWord.java に定義された hir... ADD を + に変更する。その結果、 $+(x, y)$ が出力コードになる。加えて、生成規則を変えると、目的とするコードが生成されるようになる。

以上のように他の演算子も実装を行い、加えて比較演算子¹(<, ≤, =, ≥, >) と代入演算子 (< =) の予約語の登録も行う。

if 文について

if 文では入力コードに記述されない boolean 型の変数が目的コードでは記述される。その変数は If 文と else 文の数だけ現れる。HIR では if 文の数の情報は無いため、この情報は実装時に追加しなければならない。他の点は四則演算の実装と同じである。

また VHDL では if 文のみでの使用は、論理合成する際に不具合が起きるため推奨されておらず、必ず else 文と一緒に使用しなければならない。

¹VHDL では比較を表す“ == ”は“ = ”である。

```

if
  signal sT0_3 : wiredOrBoolean  ;//sT_03 が新たに追加された変数

BEGIN
  sT0_3 <= (x = 0); //(X==0) の真偽値を代入
  if sT0_3 then
    returnVar_ <= 0;
    else
    returnVar_ <= 1;
    end if;

END branch;

```

図 4.3: 追加された変数の例

while 文について

while 文も if 文と同様に比較を行う際の新たな変数を記述する。また、VHDL の while 構文はコード内に loop...end loop の記述が必要なのでその追加も行う。

```

while
BEGIN
  while sT0_10 <= (i=0);
if sT0_10 then
loop//while 構文はこの記述が追加される
  i <= (i+1);
  end loop//ループの終了
  returnVar_ <= 0;

END while1;

```

図 4.4: while 文の例

関数呼び出しについて

対象となる関数はパラメタを持たない関数とした。

第5章 評価

5.1 テストプログラムとその方法

テストについては COINS 付属のテストプログラムを変更して行った。またモデルとなる SPARK の出力と見比べるという方法を採用した。テストプログラムは、代入文、四則演算、if 文、while 文、関数呼び出しについてのものので簡単なプログラムを用いた。

5.2 結果

以下テストプログラムとして、四則演算を含むプログラムを例に、VHDL 変換と論理合成結果を図 (5.3) に、その内部構造を図 (5.4)、(5.5) に、入力の整数型のビットを下げた場合の論理合成結果を図 (5.6) に示す。

論理合成結果に関して使用したツールは Xilinx 社 [7] の ISE10.1 付属のものである。

```
test
int a, b,c ;
int main() {
int a, b,c,d,e,f;
a = 3;
b = a+5;
c = b-2;
d = c*e;
f = d/2;
return (0);
}
```

図 5.1: テストプログラム

```

vhdl
//インポート宣言省略
ENTITY main IS
port(
a : INOUT wiredOrInt range -858993460 to 858993460 ;
b : INOUT wiredOrInt range -858993460 to 858993460 ;
c : INOUT wiredOrInt range -858993460 to 858993460 ;
returnVar_ : OUT wiredOrInt range -858993460 to 858993460 ;
CLOCK : IN std_logic ;
RESET : IN std_logic ;
done : OUT std_logic );
END main;

ARCHITECTURE behav OF main IS

    signal d : wiredOrInt range -858993460 to 858993460 ;
    signal e : wiredOrInt range -858993460 to 858993460 ;
    signal f : wiredOrInt range -858993460 to 858993460 ;

BEGIN
    PROCESS
    BEGIN
        wait until CLOCK'event and CLOCK = '1';

        a <= 3;
        b <= (a+5);
        c <= (b-2);
        d <= (c*e);
        f <= (d/2);
        returnVar_ <= 0;

    END PROCESS;
END behav;

```

図 5.2: テストプログラム

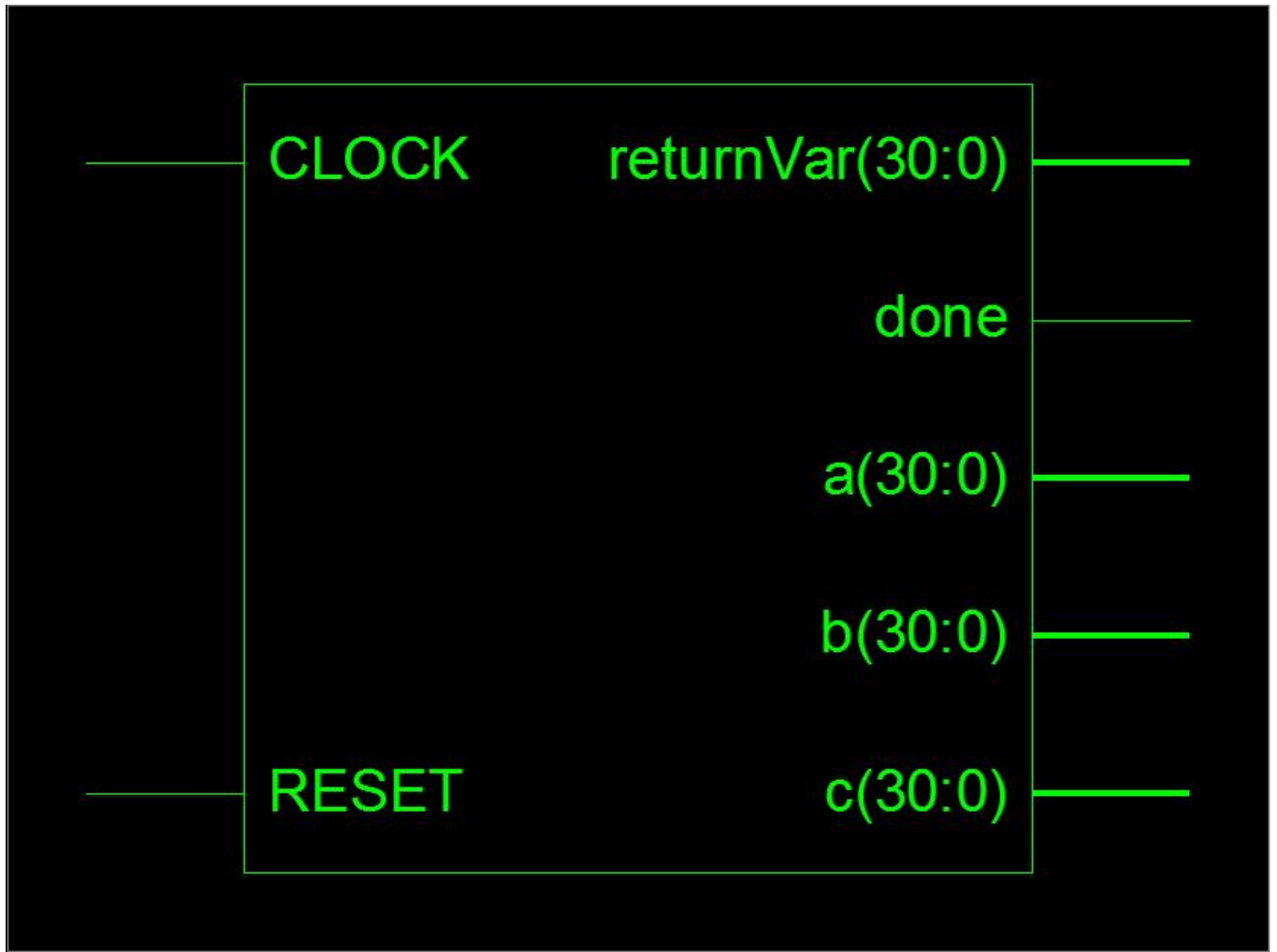


図 5.3: 論理合成後

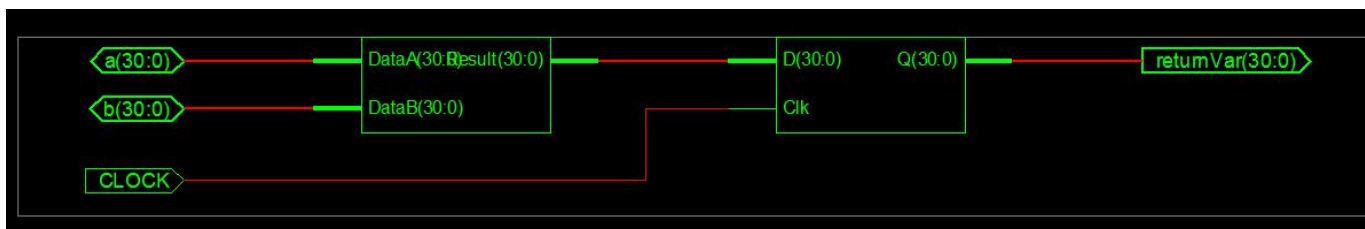


図 5.4: 図 (5.3) の内部 1

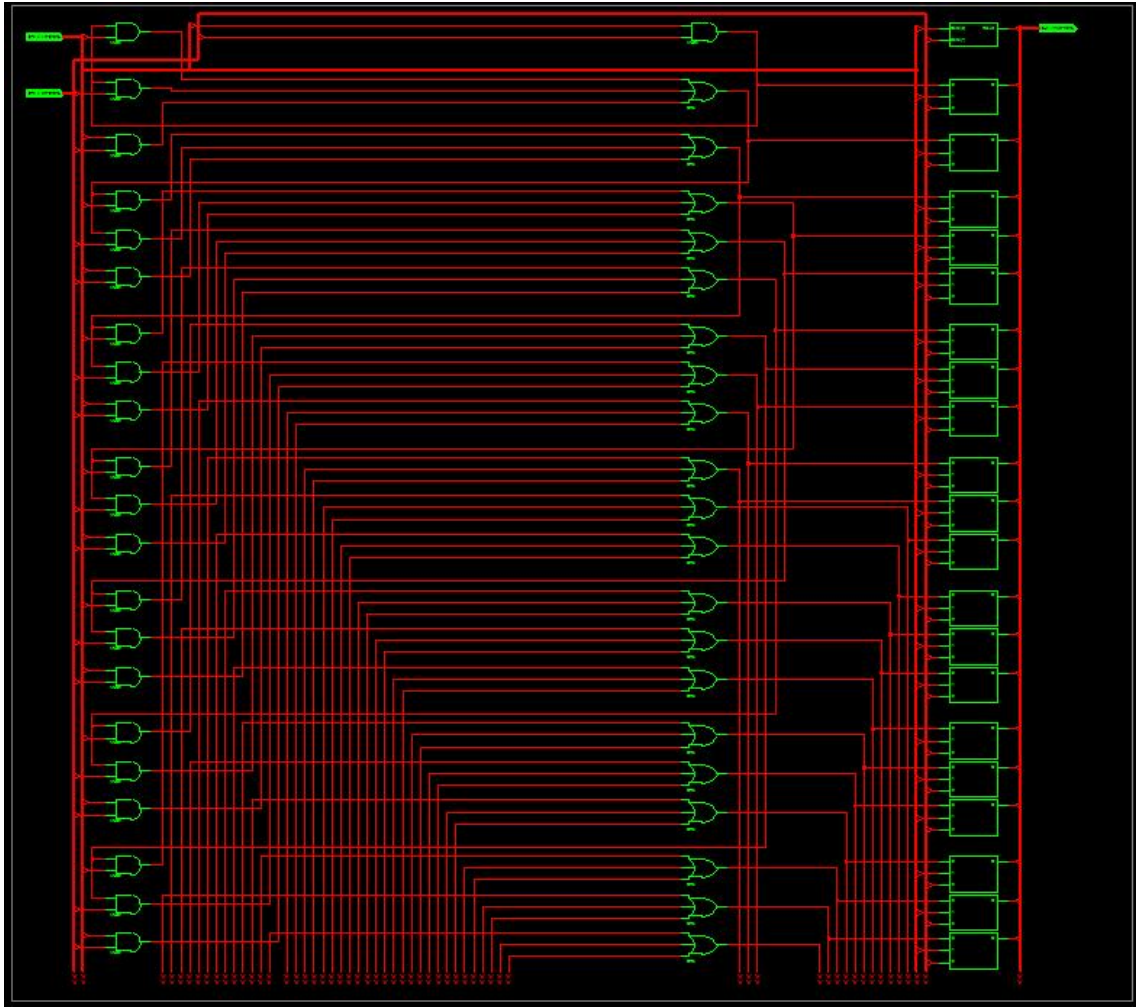


図 5.5: 図 (5.3) の内部 2

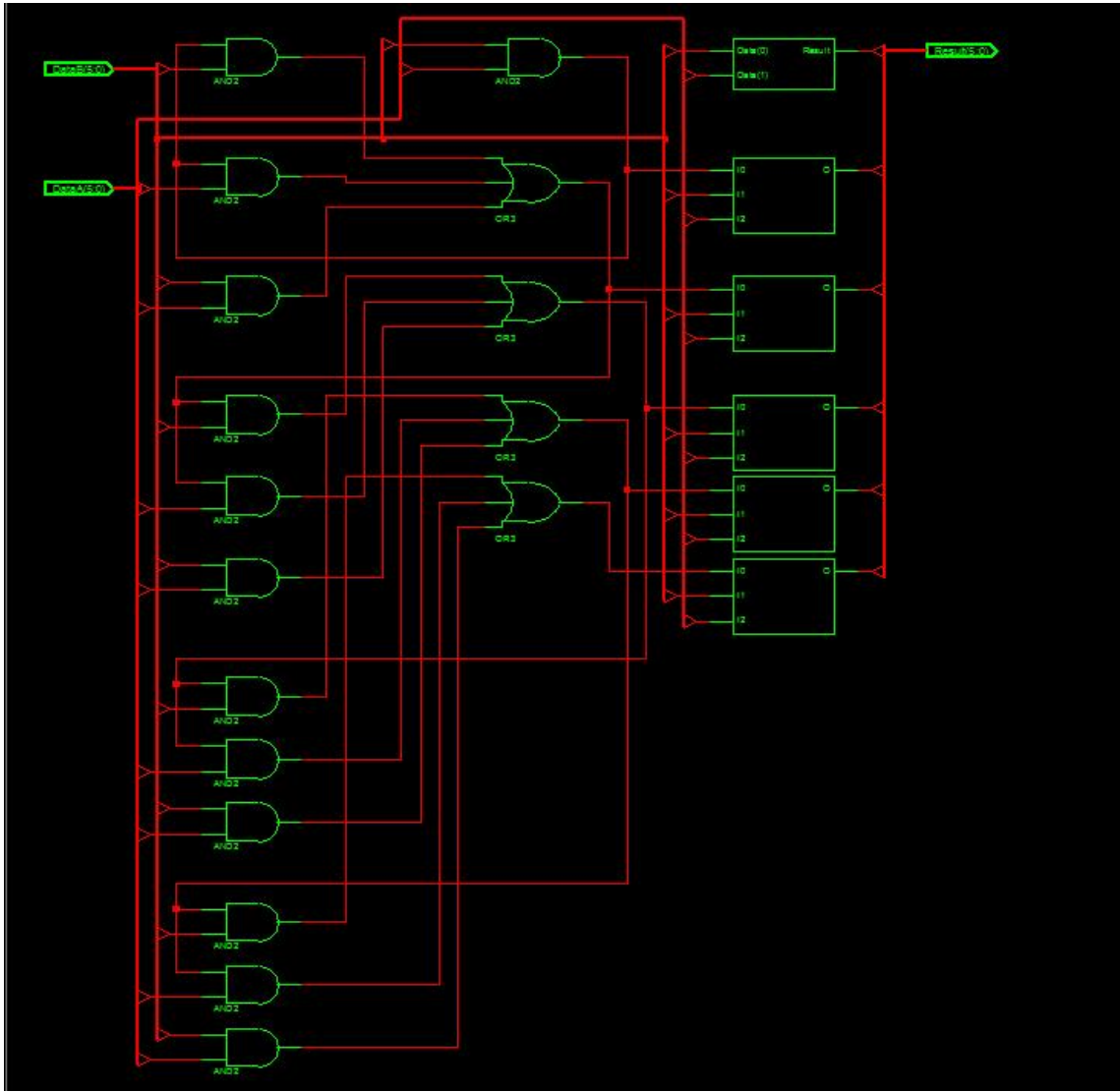


図 5.6: 図 (5.3) の内部 (簡略化した場合)

5.3 総評

当初の目標である SPARK をモデルとした、C2VHDL の変換ツールは達成できたと思う。ただ、SPARK 自体にバグがあり、VHDL の文法として成り立たない部分が多少あった。致命的なバグは、変数の最後に `;` が付加される場合があることが挙げられる。VHDL では変数の最後に `;` がついてしまうとコンパイルできない。また、本研究で使用したコマンドラインでは for 文が挿入されるとコンパイルできなくなってしまう。致命的なバグではないものの SPARK は case 文を if 文と認識してし出力している。この点は個人的に実装を難しくする要因であった。case 文での実装については後に記述する。

論理合成に関しても、今回実装した VHDL は RTL と呼べるものではないので程度が低くなってしまった。内部信号や変数に定数を代入するとうまく論理合成ができなかったのだが、これは記述レベルに問題があるかもしれない。今後の課題として、以上の問題の改善に加え、さらに多くの C 言語を認識できるように拡張する事、RTL での変換を目指す事、が挙げられる。

case 文の実装について

SPARK で case 文は図 (5.7)、(5.8) のように if 文として認識される。VHDL では case 文による配線配置と if 文による配線配置は違ってくるので、本当ならば case 文を if 文と認識して出力するべきではない。加えて、本研究の実装は hir2c を雛形としているため、HIR の情報にないものは実装しにくい物になってしまう。この例で言うと、if 文や case 文による分岐の個数と case 文の分岐評価の基準となる、整数の名前 (x) が HIR の情報にはないものである。正確に言うと、分岐の基準となる整数の名前は情報としてはあるのだが、探索するのが容易では無い。case という入力に対して if を対応させることは容易であるが。その後に来る整数 (この例では 0) を一度内部信号に真偽値を代入する作業が困難であった。難しい実装であることと、case 文を if 文と認識することに疑問を持ったことが、case 文の実装を行わなかった理由である。

case 文を含む入力

```
int x;
int sw(){
    switch(x){
        case 0:
            return 0;
        case 1:
            return 1;
            break;
    }
}
```

図 5.7: case 文を含む入力

出力

```
..  
..  
PROCEDURE sw (  
    x : INOUT wiredOrInt range -858993460 to 858993460 ;  
) IS  
    signal sT0_3 : wiredOrBoolean ;  
    signal sT1_3 : wiredOrBoolean ;  
  
BEGIN  
    sT0_3 <= (x = 1);  
    if sT0_3 then  
        returnVar_ <= 0;  
        else  
            sT1_3 <= (x = 2);  
            if sT1_3 then  
                returnVar_ <= 1;  
            end if;  
        end if;  
  
END sw;  
..  
..
```

図 5.8: 図 (5.7) での SPARK 出力 (一部省略)

第6章 関連研究

高位合成の実現に向けて

高位合成が新しい技術でありその実現に向けた研究は [10] を例として 10 年以上前から行われていた。CAD ツールと比較し、アーキテクチャ設計において、CAD を超える機能をもつものが要望されていたのである。高位合成器の仕組みはコンパイラのそれと似通ってはいるが、ハードとソフト、それぞれの視点からシステムの構築に向けていろいろな試みがなされていたようである。

高位合成過程における並列処理

再構成コンピューティングを利用した新しい並列処理パラダイム [9] について C プログラムをスレッドへ静的に並列化し、VHDL へ変換するという研究である。スレッドをハードウェア化することで、ハード的にも並列処理の性能が上がるというものである。

C 言語からの高位合成を用いたハードウェア最適化

これからのハードウェア設計において C 言語開発はなくてはならないものであろう。その設計においてソフトウェアの視点、ハードウェアの視点を考慮し設計にどのような差異が出るか実験したのが文献 [8] である。結果的には少々設計時に手間は生じる者の高位合成の結果ハードウェア寄りの考え方のほうが、最終的にはゲート数の減少や実行速度が向上したと報告されている。このことから全て C 言語による開発にするよりも、ケースバイケースで開発フローを考慮することが重要だと言えよう。

第7章 終わりに

本研究では COINS 上に C2VHDL の変換ツールを実装した。基本的な文法に対応した物が実装できた。今後の課題としては、更なる拡張を進め、基本的な文法だけ無くより多くの文法を VHDL に変換できるようにしたいと考える。高位合成の分野は 20 年ほど前から研究が進められ、近年においては商用のものが高いレベルで実現してきている。ただ、学生や研究者が気軽に手に入れられるものではない。オープンソースである COINS に実装することによって、または COINS のパッケージとして配布することによって、誰でも気軽に高位合成器を手に入れられるようになれば便利ではないかと考え本研究を始めた。まだまだ、残る課題は多いが、成し遂げたい目標である。

謝辞

本研究を進めるにあたり多大なる御指導ご鞭撻を頂いた、東京工業大学 数理・計算科学専攻教授の佐々政孝先生に深く感謝の意を表します。

また、佐々研究室の皆様、東京理科大学滝本宗弘先生と滝本研究室の皆様にはさまざまな面で助力を頂きました。あらためまして、ここに深くお礼申し上げます。

参考文献

- [1] COINS-Project. Coins homepage. <http://www.coins-project.org/>.
- [2] 中田育男. コンパイラの構成と最適化. 朝倉書店, 1999.
- [3] COINS Project. COINS プロジェクト HIR 仕様, 2002. <http://www.coins-project.org/COINSdoc/hir/hir-frame.html>.
- [4] 佐々政孝. プログラミング言語処理系. 岩波書店, 1989.
- [5] University of California. Spark homepage. <http://mes1.ucsd.edu/spark/>.
- [6] Gupta Sumit, Gupta Rajesh, Dutt Nikil, and Nicolau Alexandru. *SPARK:: A Parallelizing Approach to the High-Level*. Kluwer Academic Publishers, 2004.
- [7] Xilinx. Xilinx homepage. <http://japan.xilinx.com/>.
- [8] 諭井上, 毅近藤, 知諭泉. C 言語から vhdl への変換における並列処理. 情報処理, 20040730.
- [9] 浩正下尾, 彰山脇, 雅彦岩根, 正博福井. C 言語からの高位合成を用いたハードウェア最適化に関する一検討. 情報処理, 20051021.
- [10] 一敏若林. 「高位合成の実用化へむけて:必要な機能と応用例」～機能合成システム cyber による機能設計支援. 情報処理, pp. 171–172, 199301.
- [11] 坦渡邊, 哲朗藤瀬. 高水準中間表現 hir での最適化 (21 世紀のコンパイラ道しるべ…coins をベースにして 8). 情報処理, Vol. 47, No. 11, pp. 1263–1271, 20061115.
- [12] 秀明並木, 亘道永井. VHDL によるデジタル回路入門. 技術評論社.