設 計 自 動 化 74－8
（1995．3．8）

RT レベルの HDL 記述を用いたプロセッサ設計の比較

赤星 博輝　安浦 寛人
九州大学 大学院 総合理工学研究科 情報システム学専攻
〒816 春日市春日公園 6—1
Email: {akaboshi, yasuura }@is.kyushu-u.ac.jp

あらまし
論理/レイアウト合成の自動化に伴いハードウェア記述言語 (Hardware Description Language:HDL) による設計が一般化してきた．小規模なものに対しては，完全に自動設計を行うことが可能となっている．プロセッサのような大規模なものを自動合成する際の問題点を明らかにするために，RT レベルの HDL を用いてプロセッサの構成要素およびプロセッサの設計を行い，自動合成ツールによって生成された回路の比較，検討を行った．

和文キーワード：CAD, プロセッサ設計，HDL，自動合成

Design Comparision of Hardware Description Languages in RT Level

Hiroki AKABOSHI and Hiroto YASUURA

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6-1 Kasuga-koen, Kasuga-shi, Fukuoka 816 Japan

E-mail:{akaboshi,yasuura}@is.kyushu-u.ac.jp

Abstract
Progress of logic/layout synthesis makes it possible to design circuits by Hardware Description Languages(HDLs). When a designed circuit is small, it is synthesized automatically from HDL description. In this paper, to make it clear what kinds of problems are there in designing a large circuit looks like a processor, we design a processor and some components of it by HDLs in RT level and evaluate circuits synthesized by a logic/layout synthesis tool.

英文 Key Words: CAD, Processor Design, HDL, Synthesis

1 はじめに

最近のハードウェア/ソフトウェア協調設計と言われる分野では，あるシステムに対する要求からハードウェアとソフトウェアの最適な分割を自動で行う研究が行われている．ハードウェア/ソフトウェア協調設計では，プロセッサとして既存のプロセッサを利用する場合と，カスタマイズ可能なプロセッサを利用する2つに分けられる．既存のプロセッサを利用する場合には，必要なハードウェアをコ・プロセッサとしてインプリメントを行う．プロセッサをカスタマイズする場合には，必要とされるハードウェアの命令セット・アーキテクチャを変更したり，必要ならばコ・プロセッサを付加を行う．我々は，プロセッサをカスタマイズするためのハードウェア/ソフトウェア協調設計支援環境の研究を行っている．

プロセッサをカスタマイズするためには，プロセッサを自由にカスタマイズできる柔軟な設計環境が必要である．現段階では，ハードウェア記述言語(HDL)を用いて設計を行うことで，論理/レイアウト合成ツールを用いてハードウェアにすることが可能である．今後，HDLの処理系によって，自由にカスタマイズしたプロセッサを作成することが可能となると考えられる．一方，汎用プロセッサは，手設計などにより，High_Performanceなものが設計される．そのために，汎用プロセッサとHDLによって自動合成されたプロセッサを比べると，論理/レイアウト設計の質に差が出ると考えられる．

そこで，我々は，HDLを用いてプロセッサの構成要素およびプロセッサの設計を行い，論理合成およびレイアウト合成を行った．その結果から，生成された回路について考察を行った．本稿では，2章で論理合成ツールを用いて設計されたプロセッサについて述べ，3章で実験結果を示し，4章で考察を行う．

2 論理合成を用いたプロセッサ設計

2.1 設計例

商用プロセッサの場合，設計の詳細が不明なものが多いが，RTレベルのHDL記述からの論理合成ツールを使用して設計したプロセッサとしては，V810[1][2]などがある．また，大学レベルでも，KUE_CHIP[3], KUE_CHIP 2, などがCADツールを用いて設計され，QP_DLX[4]はHDL記述から論理合成ツールを用いて設計されている．

2.1.1 V810

V810はNECで開発された32ビットの*RISC*プロセッサで，0.8*um*CMOS2層アルミプロセスを使用し，24万トランジスタを$59.29mm^2$のシリコンチップ上に集積している．動作周波数は，基本的には電源電圧5Vで25MIIzで動作する．汎用レジスタ32本，32ビット単精度の浮動小数点演算，5段のパイプライン，1KBの命令キャッシュを持つ．次のようにプログラムをブロックに分割し，表1のように一部をCADツールを用いて自動設計している．ランダム・ロジックの部分をFDLというRTレベルのHDLで記述を行い，論理/レイアウト合成を行っている．チップの約10%の2.5万トランジスタの部分に，論理/レイアウト合成を利用している．

データパス	マニュアル
ROM,RAM	マニュアル
クロックドライバ	マニュアル
制御部	自動
命令デコード部	自動/マニュアル
端子部	マニュアル
ブロック間配線	マニュアル/自動

表 1: V810での設計

2.1.2 QP_DLX

QP_DLXは，九州大学で設計された32ビットのRISCプロセッサで，0.8*um*CMOSスタンダードセルで実現されている．動作周波数は，シミュレーションから10MIIz程度と予測されている．ハードウェア記述言語としてSFLを用い，論理合成ツールとして*PARTHENON*，レイアウト・ツールとして*Compass Design Automation System*を用いて設計された．トランジスタ数は12万トランジスタ，チップ面積は$66.24mm^2$である．レジスタ・ファイルはモジュール・ジェネレータを利用したが，その他の部分については全てPARTHENONとCOMPASSを用いて設計を行っている．

2.2 ハードウェア/ソフトウェア協調設計

汎用のプロセッサを設計する場合には，スピードや面積的な制約が厳しい場合には，手設計を行うこ

とで解決することができる．現在行われているハードウェア/ソフトウェア協調設計は，短期間で設計を行うことを目標としているために，できれば自動合成または可能な限り少ない労力で設計を行う必要がある．

用途を限定しているために，特殊な演算などがインプリメントされることが多く，ライブラリにある演算器だけでは不十分であることが考えられる．そのために，HDL などによって必要とされる演算器を設計する必要がある．

このような制約のために，HDL による設計から自動合成ツールを用いる方法が検討されているが，現段階では，HDL による設計から自動合成ツールを行う場合，まだ明らかになっていない部分がある．自動合成される回路の質や，設計者がどのような情報を与えると良いかなどについて調べるために，いくつかの実験を行った．

3 実験

以下の点から実験を行った．

1. **データパス・ライブラリとの比較**
   データパス・ライブラリと比較することで，HDL を用いた設計がどの程度実用的なものかについて比較を行った．
2. **人手による情報を与えることでのレイアウトの変化**
   人手によって，縦/横比などを与えたり，端子位置の指定を行うことで，チップ面積に与える影響を調べた．
3. **複数モジュールの設計**
   実際のプロセッサ設計では，複数のモジュール毎に設計が行われる．複数のモジュールを組み合わせて設計する場合の注意点などを調べた．
4. **32 ビットのマイクロプロセッサ**
   すべて，HDL で記述した32 ビットのプロセッサを実際にレイアウトまで行うことで，今後検討すべき項目を明らかにする．

今回の実験は，HDL としては VHDL を用いた．論理合成ツールとしては *Compass Tool*の *ASIC Synthesizer*を，レイアウト合成ツールとしては同じ *Compass Tool*の *Chip Compiler*を使用して，自動合成を行った．設計には，VLSI Technology 社の 0.8*um* CMOS ライブラリ VSC450 を用いた．そして，データパス・ライブラリとしては，VLSI Technology 社の 0.8-Micron Datapath Libray(VCC4DP3) を利用した．

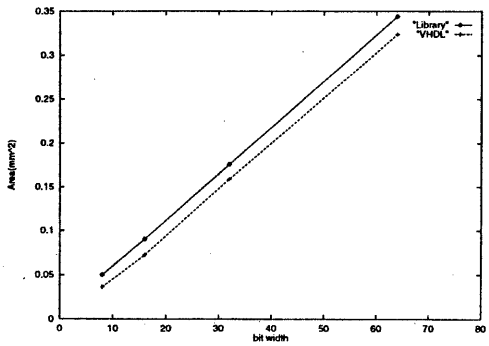
3.1 データパス・ライブラリとの比較

HDL による設計を行った場合と，データパス・ライブラリを使用した場合に，どの程度の差がでるかについて実験を行った．32 ビットの Ripple Carry 加算器と 2read 1write 32 ビットのレジスタ・ファイルを実際に設計し 論理/レイアウト合成を行った．また，同時に同機能のデータパス・ライブラリをレイアウトし，その大きさを比較した．

3.1.1 Ripple Carry 加算器

Ripple Carry 加算器のレイアウト結果を，図 1 に示す．設定はすべてデフォルトの状態で，論理/レイアウト合成を行った．8,16,32,64 ビットのすべての場合について，VHDL を用いた設計記述をレイアウトした場合の方が面積が小さくなった．

Ripple Carry 加算器は，1 次元構造のためにレイアウトが容易な点，また，比較的ゲート数が少ないために レイアウト・ツールが配置し易かったためと考えられる．小規模な回路の場合，HDL の設計でも，比較的高品質なデータパスを作成可能であると考えられる．

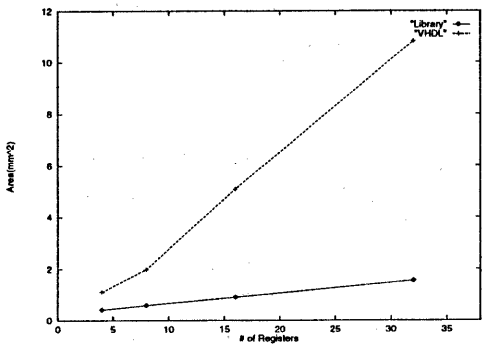


Bit Width	4	8	16	32	64
# of Gates	20	40	80	160	320

図 1: Ripple Carry Adder

3.1.2 レジスタ・ファイル

2read 1write の 32 ビット・レジスタファイルのレイアウト結果を，図 2に示す．VHDL による設計の場合には，D フリップ・フロップで実現しているために，データパス・ライブラリに比べて大きくなる．データパス・ライブラリの場合，データブックから 面積は (面積) $= K_1 \times (N + 0.6)$ で 表される (N はレジスタ数，K_1は定数)．この結果から，少数のレジスタしか必要ない場合は HDL の記述からレジスタ・ファイルを生成することもできるが，多くのレジスタを必要とする場合にはデータパス・ライブラリを用いる必要があることがわかる．



# of Registers	4	8	16	32
# of Gates	880	927	1898	3591

図 2: レジスタ・ファイル

3.2 レイアウト方法による違い

レイアウト時に，人手によってなんらかのパラメータが与えられることで，レイアウト結果に影響がどの程度与えられるかについて調べた．32 ビットの Ripple Carry 加算器を，人手によってレイアウトの指定を行うことで，どの程度チップ面積が変化するか調べた．変化させた項目は，チップの横/縦比と，入出力端子の位置指定をするしないの 2 項目である．入出力端子位置は，図 3にあるように，モジュールの横方向に端子を出す場合 (PW)，モジュールの縦方向に端子を出す場合 (PL) の 2 種類について行った．

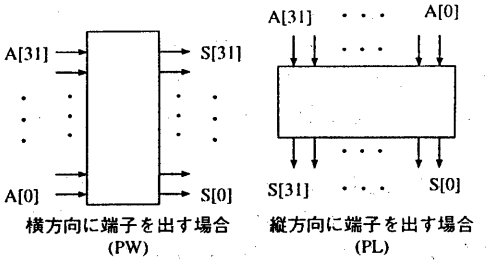


図 3: 端子位置

図 4に実験結果を示す．まず，横/縦比を 1:1 でレイアウトを行った場合，$0.159mm^2$であった．その後，Ripple Carry 加算器は 1 次元構造であるために，横/縦比だけを変えていった．横/縦比を小さくした場合には最初大きくなるが，1:16 では，最小の $0.158mm^2$ となる．

1:1 の場合，端子位置を指定すると，指定しない場合に比べて，チップ面積が大きくなっている．原因としては，設計者が与えるデータが必ずしも有効ではなく，レイアウト・ツールによっては余分な情報となってしまうことが考えられる．横/縦比が 1 以下のものは，特にチップ面積が大きくなっている．これは，図5に示すように，配線領域が横方向であったために，横方向に端子を出すためには，配線領域に配線するために，縦方向に端子を出す場合に比べて チップ面積を多く必要とするためである．

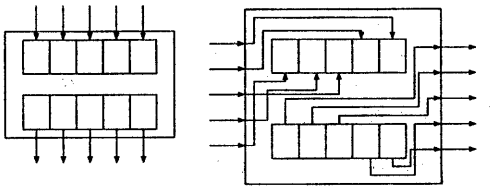


図 5: 端子の配線への影響

3.3 複数モジュールでの設計

実際の設計では各モジュール毎に論理/レイアウト設計を行い，その後，各モジュール間の配線を行うことが行われている．今回は，加算器とレジスタ・ファイルを組み合わせた簡単な回路で実験を行った．32 ビットのレジスタを 4 本持ち，次に示す 2 種類の命令を持ち，演算結果は常に出力される．

1. $REG_d = REG_1 + REG_2$

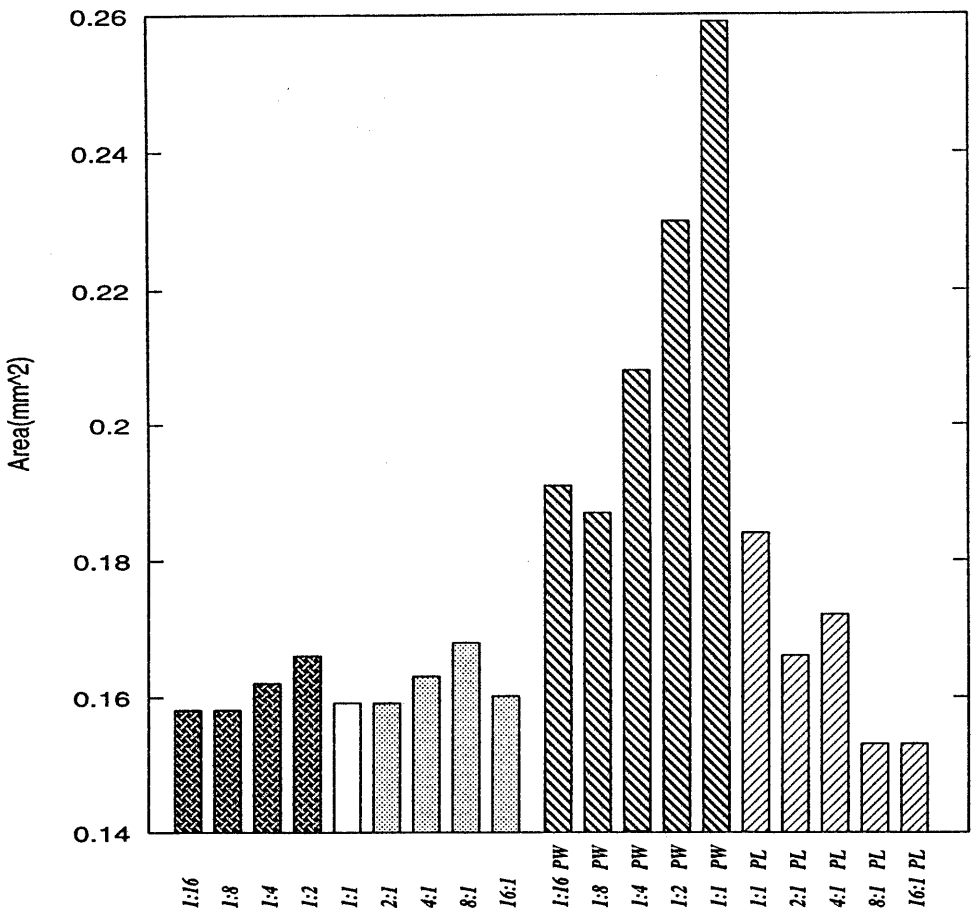


図 4: チップ面積の変化

2. $REG_d = REG_1 + 1$

最初に，Carry Look-Ahead 加算器とレジスタ・ファイルの VHDL 記述を作成し，最上位モジュールで，それらの接続，および，命令により加算器への入力を次のように切替えるように設計を行った．

```
if ( prg(6) = '0' ) THEN
  port_A <= reg_Rs1;
  port_B <= "000....00001"; --- 0X00000001
else
  port_A <= reg_Rs1;
  port_B <= reg_Rs2;
end if;
```

加算器とレジスタ・ファイルの論理合成を行い，各モジュールの端子位置を指定してレイアウトを行った結果，チップ面積は $2.60mm^2$，レイアウト結果を図 6に示す．モジュール間の配線が特定部に集中し，無駄な領域がチップの数カ所に見られるため，レイアウト結果から解析を行った．

最上位モジュールで命令毎に加算器への入力を切替えているために，レジスタ・ファイルから最上位モジュール，そして，最上位モジュールから加算器といったパスが生成された．そのために，図 6の左から CLA と Register の間に配線領域が多く必要となった．

このレイアウト結果から，モジュールの配置のみを変更することで，改善を試みた．レジスタ・ファイルから最上位モジュールそして加算器のパスが必要なために，最上位モジュールをレジスタ・ファイルと加算器の間においてレイアウトを行った．その

結果を図7に示す．チップ面積は，$2.35mm^2$となり，図6より約10%面積が減った．

レイアウト結果から機能分割を変更し，再設計を行い，論理/レイアウト合成を行った．加算器への入力データを選択するために，レジスタから最上位モジュール そして 最上位モジュールから加算器へ配線する部分が配線領域として大きな部分を占めていた．最上位モジュールで加算器への入力を行っていた部分を変更し，最上位モジュールは各モジュールに対する制御のみを行うようにし，CLA 加算器にインクリメントと加算を行う機能を組み込んだ．その結果を図8に示す．最上位モジュールは小さくなり，端子位置を指定しているために，加算器とレジスタ・ファイル間の配線領域を小さくすることができた．チップ面積は$1.89mm^2$となり，図6の結果に比べ27%，図8の結果に比べても20%減少している．

HDL で設計する場合には，設計を動作中心に考える傾向にあり，ハードウェアを意識しない傾向にある．このために，質の高い設計を行うためには，常にレイアウトを意識して設計する必要がある．

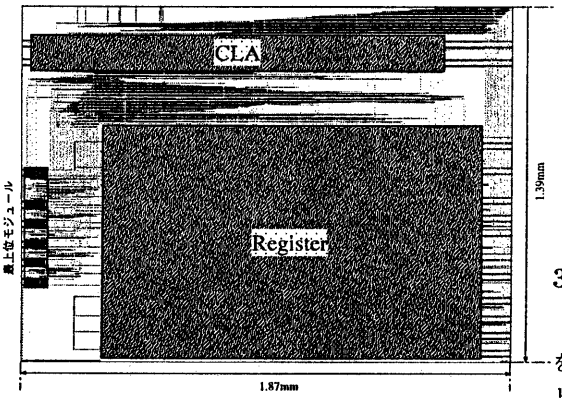


図 6: 最初の機能分割で合成した場合

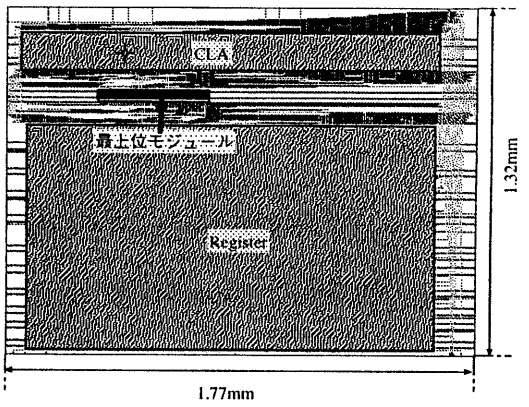


図 7: レイアウトのみ修正した場合

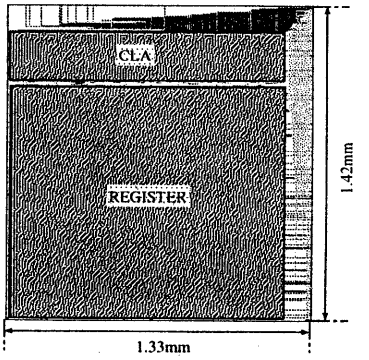


図 8: 再設計を行った場合

3.4 32 ビット・プロセッサ

32 個の RISC 命令を持つ 32 ビット・プロセッサをハードウェア記述言語で設計し，論理/レイアウト合成を行った．データパス・ライブラリは使用せずに，すべて HDL から合成を行った．5 段のパイプライン (IF, ID, EXE, MEM, WB)，16 本の汎用レジスタ (2read 1write) を持ち，フォワーディングなどのデータ・ハザード処理は行っていない．

3.4.1 設計結果

VHDL での記述では，演算器 (ALU) に 584 行，レジスタ・ファイルに 158 行，パイプラインのモジュールに 450 行，全体が 98 行の 1290 行で記述した．

全トランジスタ数は64548個，フリップフロップが901個，その他のゲートが3663ゲートである．ネットリストを展開してレイアウトを行った結果，面積は$11.6mm^2$となった．それに対して，ネットリストを展開せずにモジュール毎のレイアウトを行った結果，面積は$16.1mm^2$となった．展開してレイアウトを行った方が，図9(a)のように均等にレイアウトが行われるためであると考えられる．それに対して，モジュール毎にレイアウトを行い最後にモジュール間の配線を行う場合は，図9(b)のように個別のモジュールのレイアウトの密度が高くなる．しかし，モジュールの大きさなどにより，レイアウトに使われない領域が発生する可能性がある．そのために，面積が大きくなったと考えられる．

3.4.2 合成系による差

同じアーキテクチャをSFLというハードウェア記述言語を用いて設計を行い，PARTHENONという論理合成ツールを用いて合成を行い，*Compass Tool*を用いてレイアウトを行った．VHDLの場合と同じライブラリ(VSC450)を用いてレイアウトを行った．

ネットリストを展開してレイアウトを行った場合，面積が$22.43mm^2$となった．VHDLの設計に比べて，約2倍の面積を必要とした．SFLで設計した場合の，トランジスタ数は75616個，Dフリップフロップは856個，その他のゲートが7539個であった．

SFL記述の場合，トランジスタ数はVHDLの場合に比べ約1.2倍であるが，ゲート数は約1.8倍である．同じ機能を複数のゲートでインプリメントしているために，配線領域が増加していると考えられる．

4 考察

4.1 データパス，モジュール設計

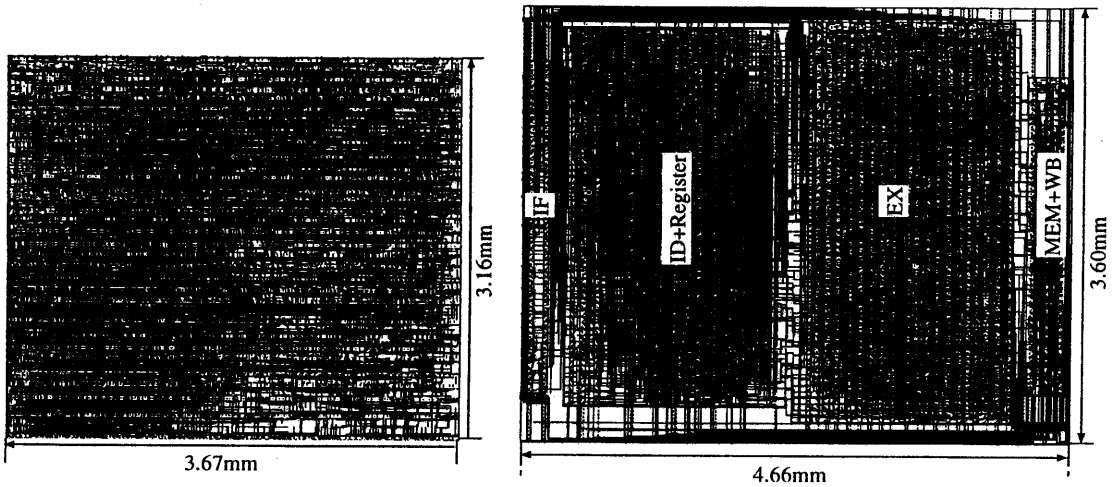
通常データパス系の設計ではライブラリを使用するものであるが，今回の実験では，HDLの設計でも簡単な回路であれば，かなり良い回路を生成できるということがわかった．今後，もう少し大きな例で実験していく必要があると考えられる．ハードウェア/ソフトウェア協調設計では，用途に特化した命令セットを組み込む場合などがあり，HDLによる設計が可能であれば柔軟な設計が可能となる．そのためにも，今後，実験を行っていく必要がある．

レイアウトを行う時に設計者が端子位置や縦/横比などを与えることは，現状のツールでは必ずしも良い結果をもたらすわけではない．モジュール毎に設計を行う場合，端子位置や縦/横比などは重要な要素である．設計する場合には，HDL記述を行う段階から，レイアウト・イメージを考えて設計する必要がある．また，CADツールに対する要求としては，このような指定に対応できる論理/レイアウト合成ツールが望まれる．

モジュール設計を行う場合には，データパスが再短径路で結べるようなモジュール分割を行う必要がある．すでにHDLで設計した演算器などを使用する場合，モジュールの分割がうまく行かないといった問題が発生すると考えられる．3.3節では，Carry LookAheadの加算器を再利用したため，レイアウト結果を見ると，データパスが引き回されてしまった．設計の再利用を行うことは設計期間の短縮に効果があるが，回路の質を高めるためには設計の修正，あるいは再設計を行う必要があると考えられる．

4.2 プロセッサの設計

市販されているプロセッサで$0.8um$CMOSのテクノロジを使用しているものについて調べると，i486は120万トランジスタを$80.24mm^2$，PA7100LCは130万のトランジスタを$210mm^2$，V810は24万トランジスタを$59.29mm^2$上に集積している．これを$1mm^2$の集積度で比べてみると，i486は約15000 ($/mm^2$)，PA7100LCは約6200 ($/mm^2$)，V810は約4047 ($/mm^2$)である．3.4節での実験結果は，約5600 ($/mm^2$)と約4000 ($/mm^2$)である．集積度だけからするとi486などには及ばないが，その他のものと比べると大差ない．レジスタ・ファイルをデータパス・ライブラリを使えば，レジスタ・ファイルだけで約$4mm^2$面積を小さくできる．$4mm^2$小さくなった場合の集積度は，約8500 ($/mm^2$)と約5300 ($/mm^2$)となる．集積度という点からはHDLを用いた設計でも，十分な設計が可能であると考えられる．プロセッサの場合は，面積だけでなく，クロック周波数や消費電力といった項目も重要である．今後は，さらに解析を行い，クロック周波数などを含めて評価を行う必要がある．また，モジュール分割を行う場合には，設計者によって各モジュールの端子位置，各モジュールの配置位置を指定することにより，かなり質の高い設計を行うことができる．



(a)展開してレイアウト　　(b)モジュール毎にレイアウト

図 9: レイアウト結果

5 おわりに

本稿では，HDL を使ってプロセッサの構成要素および実際のプロセッサの設計を行い，面積を中心に考察を行った．HDL を用いた設計の問題点としては，HDL による設計は柔軟性に富んだ設計が可能であるため，実際のハードウェアとしてインプリメントする場合には問題が生じる可能性がある．この問題点を解決するためには，ハードウェアのイメージを常に持って設計を行う必要がある．

現段階の，HDL とその処理系は，小さなものであればかなり質の高い回路を生成することができる．また，32 ビットの RISC プロセッサの設計も，集積度という点からは十分可能であると考えられる．しかし，今回の実験では，周波数などに関する解析が不十分である．今後は，周波数などについての解析を進め，HDL の記述方法やモジュールの分割方法に関してさらに実験を行っていきたい．

謝辞

Compass Tool など提供頂くソリトン システムズの皆様，PARTHENON など提供頂く NTT データの皆様に感謝致します．レイアウト設計の協力を頂いた中川 智水氏，SFL による設計をして頂いた山家 陽氏，さまざまな御助言を頂いた宮嶋 浩志氏，吉井 卓氏に感謝します．

参考文献

[1] 針谷尚夫, 楠田昌弘, 小嶋仲吾, 森山昌俊, 家永隆, 矢野陽一. “低消費電力·低動作電圧の 32 ビットマイクロプロセッサ V810”. 情処研報 ARC-92-6, pp. 41-48, 10 1992.

[2] “鈴木 宏明 and 鈴木 千佳 and 木村 晃子 and 佐藤 庄一 郎 and 井手 秀一 and 坂中 康秀”. 32-ビットマイクロプロセッサ v810 の設計手法. 情処研報 VLD-92-89, pp. 51-58, 1 1993.

[3] 神原弘之, 安浦寛人. “計算機教育用マイクロコンピュータの開発とその応用 — 集積回路技術を利用した情報工学実験—”. 情報処理学会誌, Vol. 83, No. 2, pp. 118-127, Feb. 1992.

[4] Tomomi Nakagawa, et al. “Designing Education Microprocessor QP-DLX with Full Synthesis”. In *Second Asian Pacific Conference on Hardware Description Languages*, pp. 143-146, Oct. 1994.