招待論文

オンチップ並列プロセッサ Merlot の論理設計環境

松 下 智[†]

我々は，組み込み応用向けに 1 GIPS 1 W をターゲットとしたオンチップ並列型マイクロプロセッサ Merlot を試作した．Merlot では，設計環境を新規に構築したが，その際，設計法の改善に取り組み，RTL 入力から，版管理，検証，遅延評価まで有機的に結合した設計環境を構築することができた．RTL に設計情報が集中する環境を目指し，verilog のフロントエンド記述を定義して冗長性を除去し，module 間の接続を自動化することで，記述量の 65%を削減した．版管理では CVS を用い分散拠点での開発を実現した．論理検証では RTL 更新をうけて退行テストを自動走行し，デバッグに有効な情報を自動的に集計した．遅延最適化では RTL タイミングを採用し，設計早期から短 TAT で実施可能にした．同時に配線混雑度の影響を除外した論理見直しが可能になった．本論文では，Merlot の設計環境の着目点を，設計の再現性，制御性/収束性，機能拡張性，工程管理性に整理し，それらに着目しながら，設計法としての有効性を示す．

The Logic Design Environment of Chip Multiprocessor Merlot

Satoshi Matsushita[†]

We have designed a Chip Multiprocessor Merlot which performs one GIPS at one watt for embedded applications. Our design environment realizes tightly integrated flow with terse RTL entry, version management with CVS, automatic regression test environment, and timing refinement with RTL timing. With verilog frontend processors and automatic signal hook-up tool, 65% of RTL description was reduced. With RTL timing, we could focus on logic refinement by isolating delay slack caused by P&R. For establishing Merlot's design environment, we focused on repeatability, controllability, convergence, and expandability of LSI design. We paid attention to the tool development for design progress management as well. In this paper, I describe the logical design environment of Merlot.

1. はじめに

LSI の微細化によって，システム全体規模の論理が LSI 化できるようになり，System On Silicon への移行がさけばれて久しい．一方で，設計 TAT（Turn Around Time）の増加，設計コストの増加が顕著になってきており,「何を作るか」だけではなく「いかに作るか」に対する解決が求められている[1)]．近年，多くの CAD ベンダが提唱している論理合成・配置配線の統合化ツールは，こうした要求の高まりに対応している[2)]．

我々は，オンチップ 4 並列プロセッサ Merlot（メルロー）を設計試作した．Merlot の設計に際し，設計環境を新規に構築し，前述の設計要求への解決を見出すべく，様々な取り組みを行った[3)]．本論文では，はじめにプロセッサ Merlot の概要を示し，ついで設計法の観点から課題を整理したのち，Merlot の設計検証環境の有効性を示す．

2. プロセッサ Merlot の概要

制御並列（Control Parallel）型の Chip Multiprocessor（CMP）は，複数命令ストリームを陽に導入することでスケジューリング・ウインドウサイズを拡張し，局所並列性の抽出に限界が見え始めたスーパスカラ方式を超える，大きな並列度の抽出を狙ったものである．次期マイクロプロセッサ方式の有力候補として Multiscalar[4)]，Superthreaded[5)]，Hydra[6)]等の様々な方式が提案され，活発に研究されている．

我々は，Muscat[7)]として，動的スケジューリング不要な順序つきスレッドの概念を導入し，コンパイラによる自動並列スレッド生成の方式を提案した．この詳

† NEC シリコンシステム研究所
Silicon Systems Research Labs., NEC Corp.

表 1 プロセッサ Merlot の諸元
Table 1 Merlot specifications.

Technology	0.15 μm CMOS, 5-Metal
Supply Voltage	1.2–1.8 V (Internal), 3.3 V (I/O)
Clock Speed	125 MHz (at 1.3 V)
Number of TRs.	14 M (Logic 6 M, Memory 8 M)
Area	10.5 mm × 10.5 mm (110 mm^2)
Number of Pins	300 (Signal), 500 (Total)
Power	1 W (at 1.3 V)
Performance	1 GIPS (at 1.3 V)
I-Cache	64 KB (4 Set Assoc., 32 B Line)
D-Cache	64 KB (8 Banks, 32 B Line)
Inst. Issue	2-issue (in-order) × 4 PE
Data	32 bit, 16 b×2 Media, 8 b×4 Media
Nonblocking cache	3 load/ifetch miss per PE
Bus Interface	SDRAM: 64 bit+Ecc (1 or 2 Ch.)
	PCI2.1: 32 bit, 33 MHz
	Up to 8 outstanding requests

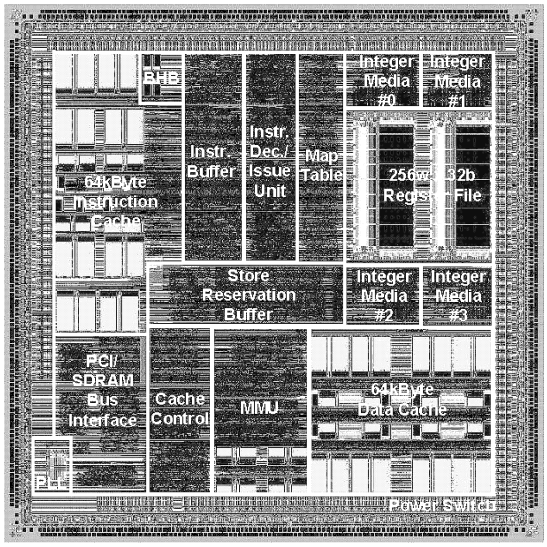

図 1 Merlot Die Plot
Fig. 1 Merlot Die Plot.

細化を続け，Merlot（mp98 ver1）と呼ぶ試作チップを作成した[8)~9)]．Merlot の諸元を**表 1** に，Die Plot を**図 1** に示す．Merlot では，在来のソースコードを並列化コンパイラによって，マルチスレッド制御命令を含んだ並列オブジェクトコードに自動変換し，4PE（Processing Element）で並列に実行する．各 PE は Media（SIMD）命令拡張した 2 命令 in-order 同時発行のスーパスカラ型 RISC プロセッサである[10)]．Merlot では，各 PE 間で命令/データキャッシュを共有した密結合型の CMP とすることで，スレッドの投入を高速化する．さらに投機的なスレッド実行機構，およびに並列化にともなうデータハザード（Write-After-Read, Read-After-Write，Write-After-Write ハザード）の検出および解消ハードウェアを実装している[8),10)]．これらハードウェアサポートは，並列化に際し発生していたデータ一貫性保証のためのコンパイラ処理を軽減し，自動並列化による在来逐次コードの十分な性能向上を可能にした．

表 2 Merlot の設計複雑さ
Table 2 Design difficulty in Merlot.

1. オンチップ 4 並列プロセッサ
- 命令 Sim との比較困難，RT Simulation 時間増大
- 検証パターン作成困難
2. オンチップ電源スイッチ
- 多電源系，貫通電流防止ゲート挿入必要
3. 周辺論理内蔵
- SDRAM（可変分周比），PCI（非同期）：多クロック
- システム検証要
4. デバッグ支援
- Partial Scan，Spare Cell（FIB 対応）
- Test Bus（PCI バスから内部メモリや RegFile を覗く）
- テスト用 On-Chip Clock 制御
5. シグナルインテグリティ対策

Merlot では，並列処理による加速を動作クロックの低下に当てる．クロックの低下にほぼ比例して動作電圧を下げることが可能なため，P（電力）$= CV^2F$（C は容量，V は電源電圧，F はトグル周波数）の式が示す大幅な低電力性が達成される．さらに，組み込み MPU では停止時間の低電力化が必要である．加えて，微細化にともなうリーク電流の増大，特に 0.1 μm 世代ではゲートリークが無視できなくなる点を考慮して，オンチップ電源スイッチを搭載した．

シミュレーション結果では，ソースコードに指示行を追加してはいるが，コンパイラが自動並列化した音声認識コードにおいて，340 MIPS，IPC=2.72（1 PE に対する加速率 3），消費電力 1.1 W の性能見積りを得ている[8)]．

Merlot には，64 bit × 2 チャンネルの SDRAM インタフェースと，PCI インタフェースを内蔵し，組み込みシステム構築の容易化を図っている．この際，SDRAM 制御回路とコア・パイプラインの双方の特性を考慮した制御設計によって，平均の cache miss overhead が 100 ns 弱という 2 次キャッシュに匹敵する遅延で SDRAM アクセスを実現している[10)]．

Merlot では，カスタム設計を Cache Data Array，Cache Tag Anrray，Register File，電源 Switch，PLL，I/O pad 等に適用した．設計の観点からの Merlot の複雑度を，**表 2** にまとめる．

3. 設計環境への要求

具体的な設計法を紹介する前に，設計ツールに対す

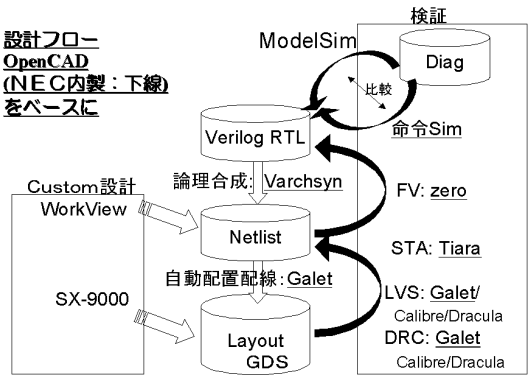

図 2 設計フロー
Fig. 2 Design flow.

る一般的な要求について整理しておく．

従来，DA ツールは，遅延やエリアといった最終結果のチューニングに論点がいきがちであった．しかるに，設計複雑化と設計競争の激化によって，設計 TAT も考慮した総合的な性能評価へシフトしつつある．我々がツール環境構築で着目した点を以下に示す．

バグの混入を防ぐため人手介入を抑え，1. **再現性**を高める．目標性能にすみやかに近づけるためには，ツールの 2. **収束性**と 3. **制御性**の向上が重要である．また，プロセスや回路技術での特色を生かして製品差別化をするためには，ツールの 4. **拡張性**も重要である．管理者は，つねに性能と納期のトレードオフとして判断を下していくことになる．LSI の製造にますますコストと時間がかかるようになるにつれ，より的確な管理判断が要求されるようになってきている．ユニットごとの遅延達成度等，管理判断に必要な 5. **管理支援情報**を的確に提供できることも，今後ますます重要となる機能である．

4. Merlot の設計フロー

Merlot では，自社製の ASIC 設計環境を主に採用した．ツール間を接続するスクリプト群を新規整備し，時にはツール自体を改造し，設計環境を構築した．設計フローの概要を**図 2** に示す．設計フロー中での各ツールの役割については，文献 11) が詳しい．

5. RTL 入力フロントエンド

設計仕様の管理については，情報の更新が容易である web ドキュメント化を徹底した．一方で，ハードウェア記述に仕様情報が集中するように，verilog HDL を拡張した RTL（Register Transfer Level）記述（以下では，vf 記述と呼ぶ）を行い，変換ツール vfront, vhookup によって verilog HDL へと変換した．

```
vfront port(.module(MTmaptable))
  input [4:0]  ED0DMLr0F; {arr=500, load=1.0} //
vfront endport

vfront ff (.scan, .clk(Phi1))   // vfront引数部
  BM0NewCmdID<= BictMergeFnd ? MergedBictC:NextBictC;
  BM0WBCmdID <= NextWBBictC;
vfront endff

vfront nmux (.out(C0BiuCmd))
  //以下, 選択信号（one Hot）:: 被選択信号の形式
  BM0C0BictCmdAck :: BM0C0BictCmd;
  devErrAck       :: BIU_CC_ER;
  default         :: BIU_CC_IDLE;
vfront endnmux

vfront   decoder (.out(BM0C1RflDoneV), .sel(C1RflID),
         .ena(BM0C1RflDone), .fill)   // vfront引数部
  // binary decoderは自動生成。
  // 例外部だけ陽に記述すればよい
vfront enddecoder

特徴）
1. バスのbit幅は信号定義部から得る
2. 右辺には式が書ける。
3. 式内のビット幅の完全一致チェックを行う
4. scan FF, ClockGating FF, やbinary decoderは
   ツールが変換する。変換法は、perlライブラリとして記述する。
5. アサーション/マルチセレクト チェック等の自動挿入
```

図 3 vf 記述例
Fig. 3 vf description sample.

5.1 verilog フロントエンド：vfront

vf 記述は，vfront によって verilog 記述に変換される．vf 記述では，verilog が本来シミュレーション向け言語である点に着目し，機能記述として有害もしくは冗長である表現の使用は禁じた．さらに，表記を統一することは，RTL の可読性を上げ，保守性の側面からも重要である．port 記述，FF，nmux（デコーダなしマルチプレクサ），decoder の記述例を**図 3** に示す．vf 記述では，キーワード vfront で囲んだ部分のみを処理するので，その外部には通常の verilog 記述をすることができる．本実装により，vfront にはパーザが不要になり，設計者自身が片手間にツール更新できるほど実装を軽くすることができた．

通常の verilog において，function 文や if 文を用いたマルチプレクサを記述した場合，不要なプライオリティエンコーダやラッチを生成する可能性がある．このため，RTL 記述では，これらの文の使用を制限し，代わりに図 3 にある Mux 記述等を導入した．port 記述では，繰返しを除去するとともに，8 章の RTL timing 用の遅延記述を含んでいる．

また，vfront の実装では，変換処理部を変換ライブラリ群として独立させ，**拡張性**に配慮した．ライブラリ関数の修正で，合成手法を一度に変更することや，vfront の引数部に書かれた合成への個別指示の解釈を変更することが可能である（**制御性**に対する配

```
vfront xperl
 foreach $reg (qw(Sdcr Sdmr Sdrr)) { # perlのループ記述
     ($regU = $reg) =~ tr/a-z/A-Z/;  # perl記述: uppercaseに
* wire ${reg}Sel = (BM0D1DevPadr[4:2] == BDADR_${regU});
* reg [31:0]   ${reg}Reg;  // この辺はverilog記述
+ vfront xperl
+  for (\$b = 3; \$b >= 0; \$b--) { # perl記述の入れ子
++    vfront ff (.clk(Phi1),.ena(BM0D1DevWeV[\$b] &
++                               BM0D1BdevWe & ${reg}Sel),
++          .arst(GSxRealReset))
++     ${reg}Reg[\$b * 8 + 7: \$b * 8]
++      <= BM0D1DevWrData[\$b * 8 + 7: \$b * 8];
++    vfront endff
+ }
+ vfront endxperl
 }
vfront endxperl
```

図 4 vf 記述での perl による繰返し例
Fig. 4 Perl description in vf format.

慮）．これは，partial scan FF の処理や enable つき FF を clock gating 用多 bit セルに置換する際等に活用された．

さらに，vf 記述においては，スクリプト言語 perl [12] を RTL 記述中に埋め込み，制御構造として利用できるようにして，単純複製等の作業軽減とミスの軽減を狙った．この例を**図 4** に示す．埋め込まれた複数の perl 記述部どうしは，同一の perl インタプリタで解釈され副作用を継承する．これにより，前方の perl 部で定義した関数の再利用等が可能となり，記述性は大きく向上した．ループ中での verilog 記述の印字とループイタレーションの関係には，次の 3 形式を指定できるように拡張を行った．これにより，perl に不慣れな設計者への導入を容易にし，単純なループ構造だけで，様々な繰返し構造を単純化できるようにしてある．括弧内に示した 1 文字の印字指定子を第 1 カラム目に置くことで，残り行が所望の印字制御をともなって verilog に挿入される．

（−） ループ先頭で一度だけ印字される．

（∗） 異なったイタレーションの行が連続行として印字される．

（+） ソース上連続した行ブロックが連続行として印字される．

（−）と（∗）を用いることで，同一式内での繰返しが記述でき，（+）によって式単位の繰返しが記述できる．また，perl 記述の入れ子（外側から評価される）も可能にしており，バイトアクセス可能な 32 bit レジスタ 3 本の記述は，図 4 のように簡単に書くことができる．vfront で狙った perl での制御構造記述の有用性は RTL に閉じたものではなく，テストパターン（論理検証用命令列）の作成でも広く活用された．

表 3 Merlot の RTL 記述量
Table 3 RTL lines in Merlot.

	行数	word 数	文字数
◆ verilog にて直接記述 — 4 Files			
記述量	3,102	8,769	84,438
◆ vhookup にて自動接続 — 53 Files			
記述量	3,397	9,540	111,622
verilog 変換後	35,779	89,403	1,496,160
記述比率	9.5%	11%	7.4%
◆ vfront での記述変換 — 168 Files			
記述量	64,252	208,709	2,168,927
verilog 変換後	155,082	569,928	5,614,290
記述比率	41%	37%	39%
総合記述比率	37%	34%	33%

5.2 自動結線ツール：vhookup

自動結線ツール vhookup の運用に先立ち，厳格な信号名規則を定義した．そこでは，ユニット名 2 文字をまず登録し，グローバル信号にはドライブするユニット名 2 文字を先頭につけるよう義務づけた．そのうえで，設計者単位でグローバル信号名を一意に定めることで，グローバル信号がチップで一意になることを保障した☆．vhookup は，階層設計に対応し，同一信号名を結線し，入出力の対応のない信号をその階層の port として上位へ送る☆☆．これにより，スペルミスした信号は，トップレベルからの port 信号にみえるため，信号名のタイプミスの検出が容易になった．さらに vhookup では，複数インスタンスに対する信号名への suffix 自動拡張についても，簡単に指定できるようになっている．この際，reset，clock 等グローバル信号は suffix 拡張から除外しないとならないが，これらに信号名定義規則で疑似ユニット名を与えることで対応できている．vhookup は，RTL から直接ユニット間ネットリストを作成する機能も有しており，8 章で述べる RTL タイミングツールの一部としても利用された．

5.3 RTL 入力フロントエンドの効果

vfront，vhookup での記述削減効果として，Merlot の RTL 記述量を**表 3** に示す．本環境により，ユニットごとにコーディングされた部分 RTL を結合し，フルチップ検証を開始するまで，3 人日で済んだ．

6. RTL 版管理環境

RTL 版管理では，Public Domain Software である CVS [13] を拡張し，排他制御機構を強化したものを用

☆ 信号名定義規則により，設計者間で調停を行うことなく信号名の一意性を保証できる．さらには，のちのちのバグやタイミングトレースを容易にする点でも非常に有効である．

☆☆ 無論，対応ある信号を上位に取り出す指定も設けた．

いた．本ツールでは，ファイル単位に設計担当者がほぼ排他的に定まることを前提に，重複作業の除去と一貫性管理がなされる．分散拠点開発にも対応し，多地区のネットワークに分散した 10 名を超える設計者が同時に論理修正を行い独立にテストしたのち，メインデータベースへ矛盾なく統合（リリースと呼ぶ）できる環境が構築できた．リリースに際するデータベースのロック時間は 10 分強であり，毎日十数回のリリースが現実的に行えた．ソース中に挿入されるのは，版番号だけであり，改版ログは別途管理される．これにより，ソース中のログ情報が不必要に巨大化することも避けられる．データベースは差分管理であり，1,400 回を超えるリリースの後でも，150 MB 強に収まっている．グラフィカルツールに比べてテキストベースの環境は，共同作業や版管理において適合性が良い☆．

7. 論理検証

MPU 設計において論理検証の網羅率向上は，重要かつ困難な課題である．実チップでの適用例がいくつか報告されている[14)~15)]．疑似ランダムによる命令列を主体に assertion checker を組み合わせた報告[14)]もあるが，RTL のタイミングを少し変更しただけで，そのテストパターンがバグ条件を発生させられなくなることが多々発生し，退行テスト（regression test）としては問題がある．RTL デバッグのしやすさ，効率の高さ等を加味し，MPU 設計で一般的に行われる綿密なテストプランに基づいた手設計のテスト命令列[16)]と，疑似ランダムを組み合わせた．疑似ランダムにおいては，文献 15) 同様，様々な狙いで 11 カテゴリを作成し，網羅率と検証効率の向上を狙った．さらに実アプリ断片と OS のカーネルコードを作成し検証パターンとして用いた．アプリケーションおよび OS の移植を容易にするため，RTL/命令シミュレータ上のアプリケーションと支援 OS 上の system call をつなぐ stub の機構を構築した．これは RTL シミュレータおよび命令シミュレータ共用として構築し，用途に応じて何通りか構築した☆☆．

版管理機構を監視して，改版があった晩に自動的に最新の full chip RTL model を checkout，コンパイルして，退行テストを行う環境を構築した．市販のシミュレータ（Modelsim）と分散実行ツール（LSF）を組み合わせ，6 台の Work Station にて，一晩で 400 本以上のテストパターン（diag）を流し，退行テストとしている（**表 4**）．走行結果は，自動集計され web にグラフ化される**図 5**．さらに各テストの IPC（Instruction per Cycle）の履歴もグラフ化される．これは，品質のトレースおよび性能バグ混入の発見に貢献した．

表 4 退行テストの内容（一晩分）
Table 4 Components of nightly regression test.

direct テスト（ベンチマーク 2 本，アプリ 4 本，OS Kernel 4 本を含む）	310 本（1 M クロック分）
ランダムテスト（毎日更新）	65 本（500 K クロック分）
システムテスト（PCI，SDRAM モデルをテスト）	53 本（150 K クロック分）

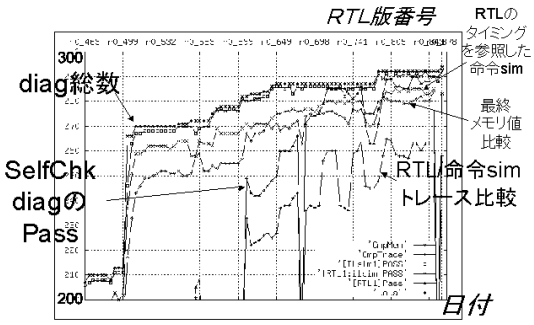

図 5 退行テスト履歴グラフ
Fig. 5 Regression history graph.

Merlot では，スレッド並列処理状況下においても命令シミュレータを RTL と同期して動作させ比較する方式を開発した．これにより，並列動作時においても RTL と命令シミュレータの比較が可能になる．退行テストの結果として不一点でのトレース情報をログに残し，デバッグ効率向上に多大な貢献をした．

自動走行と結果の自動集計すなわち論理品質のグラフ化（図 5）は，論理品質向上に貢献し，我々にも重要な示唆を与えた．設計者には，適切な自己プランニングを行うための整理された情報が必要であり，ツールによる**管理支援**情報が有益であるという点である．

8. RTL タイミングツールによる遅延改善

8.1 ね ら い

従来，遅延改善では，配置配線後からスタティック・タイミング・アナライザ（STA）を用いてフィードバックを行っていた．これでは，遅延修正作業の TAT が悪化するばかりか，遅延問題を把握する時期が遅れ，論理変更による遅延改善はスケジュール上大きな後戻りとなりかねない．加えて，論理設計上の問題と，配線迂回等配置配線で発生した問題が混同され，的確な

☆ もちろん，図形による表現も有効な場合がある．たとえば，遅延理解では非常に有効である．

☆☆ C ライブラリレベルで fake する実現性重視のタイプ，PCI バスを経由したホストへの I/O 処理までターゲットマシンモデル上で動作させる RTL 検証（性能測定向けのもの）等．

表 5 RTL タイミングを構成するスクリプト群
Table 5 Scripts for RTL timing environment.

1.	概略フロアプラン入力用スクリプト
2.	RTL からネットリストへの変換用スクリプト（vhookup）
3.	fplan への配線遅延評価パラメータ計算スクリプト
4.	遅延バジェット計算，遅延 DB 生成，遅延レポートツール
5.	遅延 DB からの論理合成制約生成スクリプト
	- 設計緩和，shrink の効果の反映スクリプト

遅延対策が講じにくくなる．

これらの問題は近年汎用 DA ツールでも取り上げられている．たとえば Magma[2)]では Gain-Based Synthesis として配置配線での理想的な遅延最適化を前提[17)]に RTL を sign off し，ワンパスフローを実現する．ここでは，後工程で理想に近い遅延最適化を達成するため，階層のないフラットな配置配線と，それを前提とした合成を行うことになる．一方，合成は階層をともなって行ったうえで，フラットにして配置配線をする手法[18)]や，合成と配置配線をともに階層を維持して行う手法[19)]が報告されている．文献 18) の手法では，論理合成時と配置配線時のセル配置の前提が異なるので，遅延問題を論理に反映するのは容易ではない．我々は文献 19) 同様，RTL タイミングと呼ばれる手法を採用した．RTL タイミングでは，文献 2) と異なり，既存のツール体系をさほど改造することなく埋め込めるばかりか，トップレベルのブラックボックスに見える cache，register file 等のカスタム設計ユニットとの融合性も良い．

RTL タイミングとの比較で考えると，文献 19) では，グローバル配線遅延を簡易見積りで行っているが，我々は，最終配線と同様のアルゴリズムで概略配線[20)]およびリピータ挿入[21)]を行い配線遅延を高精度に評価している．さらに，RTL 記述に遅延バジェット記述を埋め込むことで，情報の一元化をするとともに，データベース化した遅延バジェッティングの結果から，違反量を論理合成制約に自動再配分する制約生成環境，遅延レポーティングを統合したツール体系を構築した．以下，詳細に報告する．

8.2 RTL タイミングの構成

タイミング・バジェッティングは，論理合成の結果得られる遅延レポートとグローバルワイヤ遅延評価ツール fplan の結果を集計して得られる．RTL タイミングを構成するスクリプト群を**表 5** に示す．できうる限りスクリプトで構成しているため**拡張性**が高い．

論理合成では，論理量から推測した仮想配線長を用いるため，大面積の設計ユニットの遅延評価は誤差が大きい．一方で，あまりに小さな設計ユニットに適用すると，概略フロアプランの作成や配置変更に手間がかかる．この対策として，**図 6** に示すように，小さなモジュールに分割して設計したものを，いくつかまとめた単位（visible module）を，グローバルワイヤによる接続の単位とする．visible module 間をつなぐネットリストの作成は，RTL 結合ツール vhookup が行う．visible module のサイズを最大 1 mm 角程度にとることで，仮想配線長にともなう誤差を抑えた．さらに visible module の介在により，ユニット構成の変更が，vhookup 定義ファイルでのグルーピング情報の変更だけで行え，レイアウト試行が容易化になった．

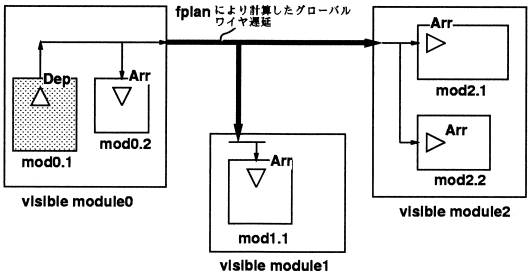

図 6 visible モジュールを導入した遅延計算
Fig. 6 Delay calculation with visible module.

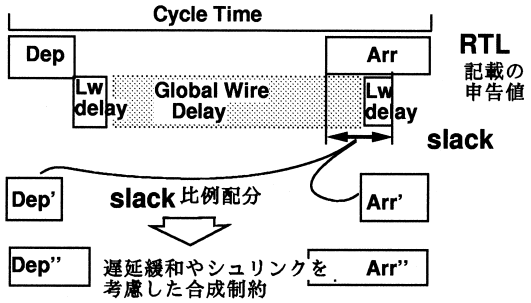

図 7 合成制約への Slack 再配分
Fig. 7 Slack feedback to synthesis.

配線遅延計算では，定義ファイルの変更で，遅延緩和やプロセスシュリンクの加味等が容易に実現できる（**拡張性**）．また，概略配線を用いるため，詳細配線の混雑による局所的な迂回配線の影響は反映されないが，これは逆に，遅延計算の再現性を高め，RTL レベルで大局的な遅延改善を実施するために好ましい（**制御性**，**収束性**の向上）．

遅延評価と，遅延 slack（違反量）の論理合成制約へのフィードバックの概念を**図 7** に示す．ここでは，visible module 内での遅延（図 7 での Lw delay）を規定値とする．合成制約の生成時には，RTL 上の遅延記述子の違いによって，遅延緩和の抑制や slack 配分の抑制が制御できる．後者は，遅延制約が固定的なカスタムブロックの動作記述部等で活用された．初期

従来手法STA:

実施可能時期:	設計後期
1. 合成	36hr (6 x ultra 300M)
2. ECO （NL 差し替え）	0.25hr
3. 電源配線	4hr
3+. 電源配線 part2	15hr (-12hr: trim wire)
4. 配置	4hr
5'. 配置禁止付き CTS	18hr (- 6hr: trim wire)
6'. リピータ挿入	5hr
7. 配線	24hr
8. Standard Delay Format 生成	> 24hr
9. STA (FF間検証)	24hr
合計	> 1week

RTLタイミング:

RTL統合直後から	
1. vhookup, etc.	5 min
2. global遅延計算.	15 min
3. global違反chk	5 min
4. 制約違反chk	5 min
通常 合計TAT	30 min
5. 合成制約生成	10 min
6. 論理合成	36hr(部分合成可能)

図 8 RTL タイミングによる TAT 改善
Fig. 8 TAT improvement with RTL timing.

表 6 RTL タイミングと STA 結果の比較（相対値）
Table 6 Comparison between STA results and RTL timing (relative).

作業フェイズ	遅延相対値
1. RTL タイミング提示値	0.98
2. 初期詳細配線後（未配 1 万）	1.64
3. 電源引剥し試行 1 （未配 500）	1.15
4. 電源引剥し試行 2 （未配 0）	2.18
5. Repeater 挿入フロー改善後（未配 1 万）	1.04
6. 最終レイアウト	1.00

導入を容易にするため，RTL 上にタイミング指定がない場合をデフォルト値として扱う．また，片側の遅延制約だけ指定されている場合は反対側はバジェット計算して制約を作成する．

8.3 RTL タイミングの効果

図 8 に遅延フィードバックに要する TAT の改善を示す．STA では 1 週間以上かかった TAT が，30 分程度に短縮された．

表 6 に STA との精度比較を示す．Merlot 設計において，RTL タイミングの結果は，配置配線最適化後の最終遅延値の STA 結果に対し 2%の誤差にとどまっている．表 6 の 2〜5 では，迂回配線等，配置配線問題で遅延が悪化しているが，これらを除去できている点でも，RTL タイミングの有益性が示されている．

遅延バジェッティングの結果は，**図 9**☆に示すように設計ユニットごとの改善の時系列として web にグラフ化される．合成ユニットごとの合成時制約違反状況や，合成の結果得られた slack 値にグローバルワイヤ遅延を加味した予測最大遅延パスなど複数のグラフを自動生成し，バランスの良い遅延対策を講じるためのプランニングに活用した（**管理支援機能**）．これは，同時に作業者の達成感向上にも寄与した．

☆ 図 9 では false path が完全に除去しきれていない．

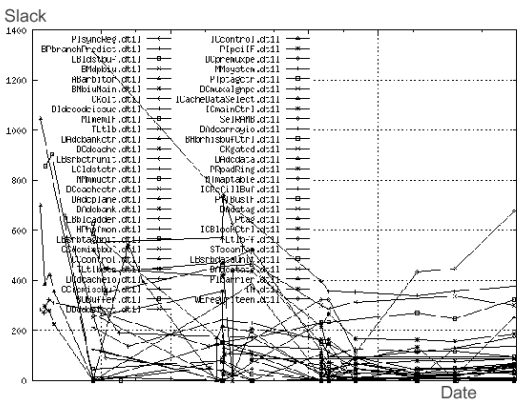

図 9 遅延改善曲線
Fig. 9 Slack improvement curve.

9. おわりに

本論文では，オンチップ並列プロセッサ Merlot の試作において構築した LSI 設計環境を解説した．本論文では，設計環境の着目点を，設計の再現性，収束性，制御性，機能拡張性，工程管理性に整理し，それら課題への取り組みの観点でまとめた．

Merlot では，設計者が，自らツールを理解し，自らコーディングするよう心がけた．これにより，ツールのたゆまぬ進化が実現し，短期間で RTL 入力から，版管理，検証，遅延評価まで有機的に結合した環境を構築することができた．本設計を通じ，設計物やツールとして何を作るかという What 的着想に加え，どのように物を作るかといった How 的着想に基づく設計法の体系化が必要であると感じ，本論文を執筆した．今後，RTL を中心とした上流系にとどまらず，ネットリスト，レイアウトといったバックエンド系も検討すると同時に，新規ツールの設計事例にも積極的に耳を傾け，体系だてていきたい

これまで，How 的着想は，ともすると Know-How として隠すことにもなりかねなかったが，設計コスト増大と競争の激化で，産業界の水平型分業化が進むなか意識改革が必要である．かつては秘中の秘であった LSI プロセス情報すら交換される時代になりつつある．設計手法においても，仲間作りをし情報交換することで，進歩の先端に立ったものだけが生き残れると感じている．

謝辞 実設計作業を進めつつ，Merlot の論理設計環境構築に尽力いただいた方々を以下に紹介し，感謝の意を表します．

情報通信メディア研究所中村祐一氏（CAD 運用全般），岡本匠氏（RTL タイミング計算ツール fplan），シ

リコンシステム研究所 mp98 チーム枝廣正人氏（RTL タイミングのデータ変換系），鳥居淳氏（RTL タイミング運用），伊藤義行氏（版管理環境），大澤拓氏（RTL 命令シミュレータ比較，退行テストツール，自動 web 化環境），NEC 情報システムズ加藤哲氏（vhookup ツール），鈴木研司氏（ランダム diag ツール），池野晃久氏（ダイレクト diag 作成，命令シミュレータ）．

参 考 文 献

1) 桜井貴康：システム LSI 設計の現状と課題，情報処理学会論文誌，Vol.41，No.4，pp.834–842 (2000).
2) Gain-based Synthesis: Speeding RTL to Silicon (2000). http://www.magma-da.com/articles/GBS.pdf
3) 松下 智：1GIPS 1W オンチップ並列プロセッサ Merlot の設計検証，DA シンポジウム，pp.115–120, 情報処理学会 (2000).
4) Sohi, G., Breach, S. and Vijaykumar, T.N.: Multiscalar Processor, *Proc. 22nd Annual International Symposium on Computer Architecture*, pp.414–425 (1995).
5) Tsai, J., Huang, J. and Amlo, C.: The Superthreaded Processor Architecture, *IEEE Trans. Comput.*, Vol.48, No.9, pp.881–902 (1999).
6) Hammond, L., Hubbert, B., Siu, M., Prabhu, M., Chen, M. and Olukotun, K.: The Stanford Hydra CMP, *IEEE MICRO*, Vol.20, No.2, pp.71–84 (2000).
7) 鳥居，近藤，本村，池野，小長谷，西：オンチップ制御並列プロセッサ MUSCAT の提案，情報処理学会論文誌，Vol.39, No.6, pp.1622–1631 (1998).
8) Nishi, N., et al.: A 1GIPS 1W Single-Chip Tightly Coupled Four-Way Multiprocessor with Architecture Support for Multiple Control Flow Execution, *ISSCC 2000* (*WP25.5*), pp.418–419 (2000).
9) 最新の mp 98 情報 (2000). http://www.labs.nec.co.jp/MP98
10) Matsushita, S., et al.: Merlot: A Single-Chip Tightly Coupled Four-Way Multi-Thread Processor, *Cool Chips III*, pp.63–74 (2000).
11) 藤波，吉田，石塚，恵谷：ますます多様化する EDA ツール，大規模 LSI のトップダウン設計を支える，日経エレクトロニクス，No.673, pp.100–139 (1996).
12) Wall, L., et al.: *Programming Perl*, 3rd Edition, O'Reilly & Associates, http://www.ora.com/ (2000). 近藤（訳）：プログラミング Perl（2nd Edition），オライリージャパン (1997).
13) Concurrent Versions System. http://www.cvshome.org/
14) Kantrowitz, M. and Noack, L.: I'm Done Simulation; Now What? Verification Coverage Analysis and Correctness Checking of the DECchip 21164 Alpha microprocessor, *33rd DAC*, pp.325–330 (23.5), ACM (1996).
15) Hosseini, A., Mavroidis, D. and Konas, P.: Code Generation and Analysis for the Functional Verification of Microprocessors, *33rd DAC*, pp.305–310 (23.1), ACM (1996).
16) Monaco, J., Holloway, D. and Raina, R.: Functional Verification Methodology for the PowerPC 604 Microprocessor, *35th DAC*, pp.246–249, IEEE (1998).
17) Sutherland, I., Sproull, B. and Harris, D.: *Logical Effort: Designing Fast CMOS Circuits*, Morgan Kaufmann Publishers Inc., San Francisco, CA (1999).
18) Koehl, J., et al.: A Flat, Timing-Driven Design System for a High-Performance CMOS Processor Chipset, *Design Automation and Test in Europe*, pp.312–320, IEEE (1998).
19) Hattori, T., et al.: Design Methodology of a 200 MHz superscalar microprocessor: SH-4, *35th DAC*, pp.246–249, IEEE (1998).
20) 岡本，石川，藤田：線分再割当による概略配線経路改善アルゴリズム，DA シンポジウム，pp.67–71, 情報処理学会 (1994).
21) Okamoto, T. and Cong, J.: Buffered Steiner Tree Construction with Wire Sizing Interconnect Layout Optimization, *ICCAD*, pp.44–49, IEEE (1996).

(平成 13 年 1 月 15 日受付)
(平成 13 年 2 月 1 日採録)

松下 智（正会員）

昭和 36 年生．昭和 62 年東京大学大学院工学系研究科電子工学専攻修士課程修了．同年 NEC（株）入社．並列マシン Cenju のシステムソフトおよび並列アプリの研究開発ののち，並列マシン Cenju2 の開発指揮および，CPU ボードの設計を担当した．平成 5 年より平成 9 年まで米国に出向し，MIPS Tech. Inc. にて VR10000，VR12000 マイクロプロセッサの検証に従事した．平成 9 年より MP98 のサブリーダとして論理設計の管理および設計検証環境の構築を担当するかたわら，Merlot のバスインタフェースの論理設計を担当した．IEEE 会員．