計算機アーキテクチャ　96－4
（1992. 10. 22）

ＳＰＡＲＣアーキテクチャに基づく スーパスケーラマイクロプロセッサ”ＦＬＡＲＥ”の性能評価

岡本　光正[*1]　的場　司[*1]　山上　宣彦[*2]　山田　朝彦[*3]　須藤　英彦[*1]

*1（株）東芝　青梅工場　ＡＳＩＣ技術部
*2（株）東芝　情報処理・機器技術研究所　開発第８部
*3（株）東芝　情報処理・機器技術研究所　開発第７部

〒１９８　東京都青梅市末広町二丁目九番地

あらまし

ＳＰＡＲＣアーキテクチャ（Ｖ７）仕様の２命令スーパースケーラマイクロプロセッサ”ＦＬＡＲＥ”を開発した。ＦＬＡＲＥは命令，データに分離された４ＫＢ毎の１次キャッシュメモリを内蔵している。また、ＳＰＡＲＣレファレンスMMU仕様のメモリ管理機構も内蔵している。さらに、浮動小数点演算チップ、１２８ＫＢの２次キャッシュメモリをそれぞれ専用のインターフェイスで接続することによりシステム性能を向上させることができる。ＦＬＡＲＥはHS1.0 μＣＭＯＳスタンダードセルの半導体技術を用いて開発された。本稿ではＦＬＡＲＥの仕様、開発手法、２命令スーパスケーラアーキテクチャの評価及びスーパスケーラを意識したコードオプティマイザについて述べる。

和文キーワード　ＳＰＡＲＣ、ＲＩＳＣ、スーパスケーラ、マイクロプロセッサ、コードオプティマイザ

Performance of the Super scaler Microprocessor "FLARE" based on SPARC architecture

[*1]Kousei Okamot, [*1]Tsukasa Matoba, [*2]Nobuhiko Yamagami,
[*3]Asahiko Yamada & [*1]Hidehiko Sudou

*1 ASIC engineering Dept, OME works, TOSHIBA
*2 Engineering workstation development Dept, Infomation systems Lab, TOSHIBA
*3 Advanced computer architecture Dept, Information systems Lab, TOSHIBA

2−9 suehiro−cho, ome−shi, Tokyo, JAPAN 198

Abstract

We have developped two instructions super−scaler microprocessor "FLARE" based on SPARC architecture(V7). FLARE has the internal cache memories which are separated 4KB instruction cache and 4KB data cache. Also, FLARE has the memory management mechanism specified with SPARC reference MMU. Moreover, FALRE can be coupled with Floating unit and the secondry 128KB cache memory directry in oder to enhance the system performance. FLARE emploied HS1.0 μ −CMOS standard cell semiconductor technology. This paper describes the FLARE outline, development method, evaluation of two instruction super−scaler architecture and the code optimizer.

英文 key words　SPARC,　RISC,　super scaler,　microprocessor,　code optimizer

1．はじめに

近年、高性能ＲＩＳＣマイクロプロセッサの出現により、ワークステーションは急激に性能を向上させている。ＲＩＳＣプロセッサは動作周波数を上げることにより性能向上を図ってきたが、高速信号の取扱いや発熱などの実装上の問題、あるいは半導体設計の困難さ（開発期間の長期化）などの問題が発生してきた。

ＦＬＡＲＥはＳＰＡＲＣアーキテクチャＶ７に基づき、スーパスケーラ（２命令並列実行）アーキテクチャを採用することにより高速化を目指すとともに、スタンダードセル方式で実現することによって開発期間の短縮を図った。

本稿ではＦＬＡＲＥの仕様、開発フロー、スーパスケーラアーキテクチャの評価及びコードオプティマイザについて報告する。

2．ＦＬＡＲＥアーキテクチャ

2．1．ＦＬＡＲＥ概略仕様

ＦＬＡＲＥはＳＰＡＲＣアーキテクチャＶ７の命令セットをサポートする。またチップ上にＳＰＡＲＣレファレンスMMU仕様のMMUと命令，データ各々に分離された４ＫＢ毎の１次キャッシュメモリを持つ。MMU内には命令用に３２エントリーのＴＬＢ、データ用に１６エントリーＴＬＢがある。

ＦＬＡＲＥは２レベルの階層キャッシュメモリシステムを採用しており、外部に接続される１２８ＫＢのキャッシュメモリチップを直接制御する。

浮動小数点演算は外部に接続されたＦＰＵにより処理される。ＦＰＵは４段の演算パイプライで構成され、このパイプラインはＦＬＡＲＥ内で制御される。

また、ＦＬＡＲＥはシステムに用いられるＥＷＳ標準バス（Ｍｂｕｓ）を直接チップから出力する。

ＦＬＡＲＥはＨＳ１．０μＣＭＯＳスタンダードセル方式のセミカスタム半導体技術を用いており、３３ＭＨｚの動作周波数を実現した。

表１にＦＬＡＲＥ概略仕様、図１に概略ブロック図を示す。

表１．ＦＬＡＲＥ概略仕様

命令セット	ＳＰＡＲＣ（Ｖ７）互換
ﾊﾟｲﾌﾟﾗｲﾝ段数	４段
同時実行命令数	２命令
内蔵ｷｬｯｼｭﾒﾓﾘ	命令 (4KB) ＋データ (4KB)
内蔵MMU	ＳＰＡＲＣﾚﾌｧﾚﾝｽMMU仕様
ＦＰＵ	外付け (WTL4064)
２次ｷｬｯｼｭﾒﾓﾘ	Ｔ９４９０Ｅ×２ (128KB)
チップバス	Ｍｂｕｓ
使用半導体技術	ＨＳ１．０μＣＭＯＳ　ｽﾀﾝﾀﾞｰﾄﾞｾﾙ
動作周波数	３３ＭＨｚ

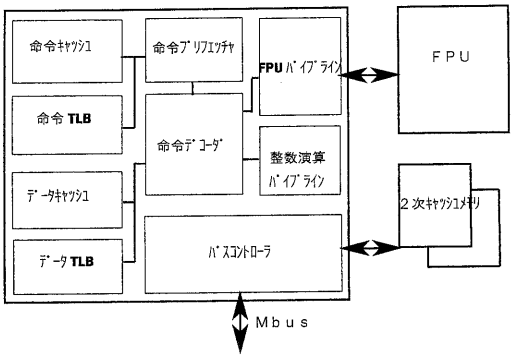

図１．ＦＬＡＲＥ概略ブロック図

2．2．パイプラインアーキテクチャ

ＦＬＡＲＥは命令フェッチ（ＩＦ）、命令デコード（ＤＥ）、実行（ＥＸ）、データ書き込み（ＷＲ）の４段のパイプラインで構成される。また、各ステージでは２命令の処理が同時に実行される。

ＩＦステージでは命令フェッチアドレスを用いて命令キャッシュ、命令ＴＬＢが検索される。命令キャッシュにヒットした場合は命令バッファに２つの命令が読み込まれる。命令キャッシュに命令が無い（ミス）場合は命令キャッシュの更新シーケンスに入り、更新が終了するまでＩＦステージの状態が続く。

ＤＥステージでは２つの命令が同時にデコードされる。２つの命令が同時実行可能な組み合わせであり、かつレジスタの競合が発生しない場合に２つの命令が同時にＥＸステージに移される。同時実行条件を満足しない場合は先行命令のみＥＸステージに移される。また、同時にレジスタソースオペランドが読み出される。Load/Store命令のオペランドアドレスもこのステージで算出される。分岐命令がデコードされた場合はＤＥステージで分岐アドレスが算出され、ＩＦステージに入る。

ＥＸステージでは命令で指定される演算が行われる。また、Load命令ではデータキャッシュ、データＴＬＢの検索が行われる。

ＷＲステージでは演算結果、Load命令オペランドがレジスタファイルに格納される。

図２にパイプラインの各ステージの機能、図３に分岐命令実行時のパイプライン動作を示す。

ＩＦ	ＤＥ	ＥＸ	ＷＲ
命令ｷｬｯｼｭ検索 命令 TLB 検索	命令ﾃﾞｺｰﾄﾞ 分岐ｱﾄﾞﾚｽ算出 ｵﾍﾟﾗﾝﾄﾞｱﾄﾞﾚｽ算出	整数演算実行 ﾃﾞｰﾀｷｬｯｼｭ検索 ﾃﾞｰﾀ TLB 検索	レジスタ書き込み

図２．パイプラインステージの機能

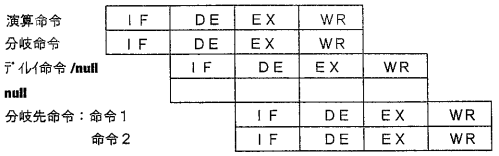

図３．分岐命令実行時のパイプライン動作

浮動小数点演算命令はＤＥステージ以降に専用パイプラインで処理される。ＤＥステージでデコードされた浮動小数点演算命令は、次に浮動小数点レジスタ競合のチェック及び演算制御信号ステージ（ＦＤＥ）に入る。その後、浮動小数点レジスタ読みだしステージ（ＲＲ）、演算ステージ１（Ｃ１）、演算ステージ２（Ｃ２）、演算結果書き込みステージ（ＷＲ）の４段のＦＰＵ内のパイプラインで処理される。

図４に浮動小数点演算のパイプラインを示す。

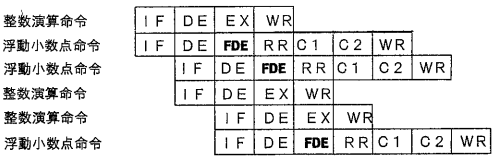

図４．浮動小数点演算パイプライン

ＦＬＡＲＥはパイプラインの実行効率を上げるため、ステージ間のデータバイパスを持っている。しかしながら、以下の場合にはＤＥステージからＥＸステージへの命令の発行が禁止される。

・Load/Store 命令のアドレスレジスタがＥＸステージの命令のディスティネーションレジスタと一致する場合。
・分岐命令のインデックスレジスタがＥＸステージの命令のディスティネーションレジスタと一致する場合。
・IN,OUT,LOCAL レジスタを使用する命令であり、ＥＸステージでＳＡＶＥあるいはＲＥＳＴＯＲＥ命令が実行されている場合。
・Load 命令であり、ＥＸステージで Store 命令を実行している場合。
・浮動小数点分岐命令であり、ＲＷステージで浮動小数点比較命令が実行されている場合。

２．３．並列実行命令の組み合わせ

ＦＬＡＲＥは整数演算命令・分岐命令・Load/Store 命令・浮動小数点演算命令の内の任意の２命令を同時に実行する。

表２に詳細な２命令の組み合わせを示す。表中のシンボルは以下のとうり。

LOAD　：LD,LDUB,LDUH,LDSB,LDSH

STORE　：ST,STB,STH,STD,LDD

ALU　：SETHI,ADD,AND,OR,XOR,SUB,ANDN,ORN,XORN,
ADDcc,ANDcc,ORcc,XORcc,SUBcc,ANDNcc,ORNcc,
XORcc,ADDXcc,SLL,SRL,SRA,RDY,RDPSR,RDWIN,
RDTBR

FLDST　：LDF,LDDF,LDFSR,STF,STDF,STFSR

FPop　：FPop1,FPop2

SINGLE　：TADDcc,TSUBcc,TADDccTV,TSUBccTV,WRY,
WRPSR,WRWIM,WRTBR,Ticc,IFLUSH,
SAVE,RESTORE,MULScc

表２．同時実行可能命令の組み合わせ

次命令 / 先行命令	Ｌｏａｄ	Ｓｔｏｒｅ	ＡＬＵ	FLDST	ＦＰｏｐ	Bicc, CALL	JMPL, RETT	ＦＢｆｃｃ	SINGLE
Ｌｏａｄ	不可	不可	可	不可	可	可	可	可	不可
Ｓｔｏｒｅ	不可	不可	不可	不可	可	可	不可	可	不可
ＡＬＵ	可	可	不可	可	可	可	可	可	不可
FLDST	不可	不可	可	不可	不可	可	可	可	不可
ＦＰｏｐ	可	可	可	不可	不可	可	可	不可	不可
Bicc,CALL	可	可	可	可	可	不可	不可	不可	不可
JMPL,RETT	可	不可	可	可	可	不可	不可	不可	不可
ＦＢｆｃｃ	可	可	可	可	可	不可	不可	不可	不可
SINGLE	不可	不可	不可	不可	不可	不可	不可	不可	不可

3．開発手法

3．1．開発フロー

ＦＬＡＲＥの開発ではＣ言語によるパイプラインモデル（ＦＳＩＭ）と機能記述言語（Ｈ２ＤＬ）によるハードウェア機能モデル及びテストプログラムを同時に作成した。ＦＳＩＭはＳＰＡＲＣアーキテクチャモデルあるいはＳＰＡＲＣワークステーションを用いてテストプログラムを実行させることにより、その記述の正当性を検証した。その後、ＦＳＩＭとハードウェア機能モデルとの動作を比較することにより、ハードウェア機能モデルの正当性を検証した。

ハードウェア機能モデルの正当性が確認されると論理合成、一部タイミングの厳しい論理回路は人手により論理回路を作成した。作成された論理回路はハードウェア機能モデルの機能シミュレーション結果から自動的に作成される論理シミュレーションテストデータを用いて論理シミュレーションを行った。

図５に開発フローを示す。また、表３にモデル記述量及びテストデータ量の設計データ量をを示す。

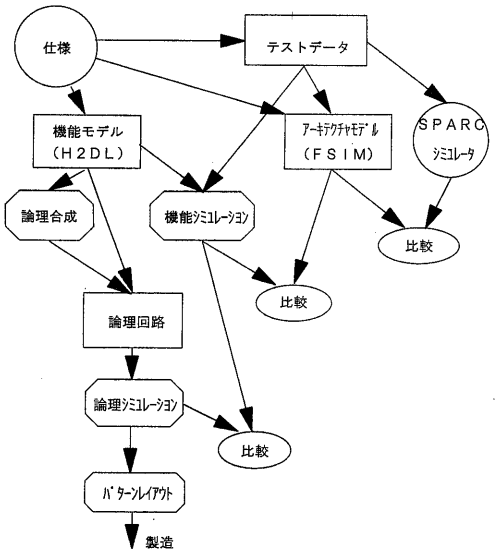

図５．ＦＬＡＲＥ開発フロー

表３．設計データ量

設計データ	データ量
テストデータ	約 1,000,000step
ＦＳＩＭ（Ｃ言語）	１９、０００行
機能記述（Ｈ２ＤＬ）	１９、０００行
論理記述（ＴＤＬ）	２４、０００行

3．2．ＬＳＩレイアウト

ＬＳＩのレイアウトはチップサイズ、動作周波数に影響が大きい。ＦＬＡＲＥはキャッシュメモリ、レジスタファイルといった大きなメガセル持ち、また加算器，ＡＬＵ等のマクロセルも持っている。これらのマクロセル間を効率よく配線するためにグルー回路は５０のブロックに分けて階層設計した。

またレイアウトにあたっては、ＴＬＢ，整数演算データパス，浮動小数点パイプラインレジスタは繰り返しの多い規則的なパターンのため、人手により標準セルを用いてレイアウトした。

最終的にはメガセル，マクロセル，人手レイアウトブロックを配置し、その間に３つのブロックに分けた自動レイアウトブロックを作成し、さらにその間のブロック間自動レイアウトを行った。

ＦＬＡＲＥ内のレイアウトブロックを表４に示す。

表４．ＦＬＡＲＥレイアウトブロック

使用ﾒｶﾞｾﾙ	RAM, 多ﾎﾟｰﾄRAM,ALU,adder, ｼﾌﾀ
人手ﾚｲｱｳﾄﾌﾞﾛｯｸ	CAM,IUﾃﾞｰﾀﾊﾟｽ,FPUﾊﾟｲﾌﾟﾗｲﾝ
自動ﾚｲｱｳﾄﾌﾞﾛｯｸ	５Ｋ，１０Ｋ，１５Ｋﾌﾞﾛｯｸ
総ﾄﾗﾝｼﾞｽﾀ数	約１Ｍﾄﾗﾝｼﾞｽﾀ

4．アーキテクチャ評価

ＦＳＩＭを用いて、dhrystone ベンチマークテストプログラムの実行解析を行い、ＦＬＡＲＥアーキテクチャの評価を行った。

表５に１クロックあたりの命令実行数を示す。２命令同時実行を行ったサイクルは２１％である。表６に実行命令の種類を示す。表６から解るように、dhrystone では整数演算の頻度が高く、２つの整数演算を同時に実行できないＦＬＡＲＥのアーキテクチャでは２命令の同時実行頻度が少なくなっている。

また、表７に null サイクル発生の要因を示す。レジスタ更新待ちの頻度が最も高く、５２％である。これはＳＰＡＲＣ標準コンパイラがＦＬＡＲＥのアーキテクチャを考慮していないために発生している。また、分岐後の命令バッファへの命令読み込みに時間がかかるため命令を発行できない場合が３５％ある。分岐による null サイクルの発生を減少させるには分岐予測機構が有効であるであると思われる。

表５．１サイクルあたりの実行命令数頻度 (dhrystone)

実行命令数　0	２０％
実行命令数　1	５９％
実行命令数　2	２１％

表６．実行命令種別の出現頻度

整数演算命令	５７％
分岐命令	２０％
Load/Store 命令	２０％
単独実行命令	３％

表７．null サイクル発生要因頻度

レジスタ更新待ち	５２％
分岐後の命令待ち	３５％
Store 後の Load 命令	７％
ｳｨﾝﾄﾞｳ切り換え待ち	６％

5．FLARE用オプティマイザ

5．1．オプティマイザ概要

スーパスカラ・プロセサFLAREのCPU内並列化による効果を引き出すための言語処理系としては、アセンブル・テキスト変換プログラムを作成した。アセンブル・テキスト変換プログラムは、SPARC用のコンパイラが生成したアセンブル・テキストを読み込み、並列実行度を上げるような命令の並べ替え（命令スケジューリング）を行ない、アセンブル・テキストを出力する。実行に当たっては、アセンブル・テキスト変換プログラムの出力をSPARC用のアセンブラによりオブジェクト・コードに変換して実行させる。

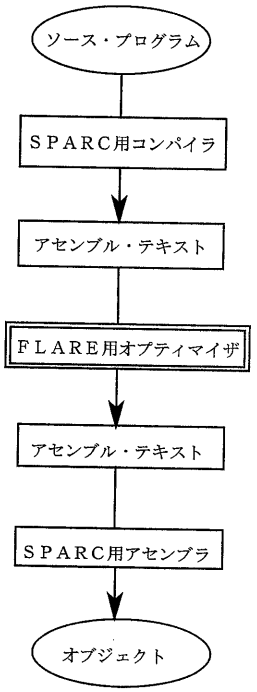

図6．オプティマイザの位置づけ

アセンブル・テキスト変換プログラムは、アセンブラ・テキストを読み込んで内部形式に変換し、その内部形式を基本ブロックに分割し、基本ブロック内の各命令間の依存関係を求めてDAG (Directed Acyclic Graph) を作成し、その依存関係をもとに命令スケジューリングを行ない、内部形式をアセンブル・テキストに変換する。

アセンブル・テキスト変換プログラムの役割は、命令スケジューリングにより並列実行命令を増加させ実行効率を向上させることであるが、その際パイプラインのインタロックを考慮することが不可欠である。すなわち、本変換プログラムの中心である命令スケジューラは、並列実行を考慮したパイプライン・スケジューラである。

5．2．命令スケジューラ

以下、命令スケジューラについて説明する。なお、命令スケジューリングの本質的な部分は基本ブロック内のスケジューリングにあると考え、今回は命令の並べ替えを基本ブロック内に限定した。

命令スケジューラは、一般に知られているパイプライン・スケジューラの技法に基づいて作成した。第1のフェイズで、予め決めた各々の命令固有の重みを基にスケジューリングの対象となるプログラムの命令依存DAGの各命令に重みを付ける。第2のフェイズで、命令に付けられた重みとプロセッサのハードウェア・リソースの状況に従い命令のスケジューリングを行なう。

5．2．1．重みの計算

第1のフェイズで計算される重みは、以下のように求める。

各命令固有の重みは、先行命令とインタロックを起こさずに実行された時に先行命令の実行に隠れない分のクロック数とする。これはほとんどの命令に対しては1である。DAGの各エッヂには、エッヂをはさむ2命令が継続して実行された場合のインタロックのサイクル数の重みを付ける。

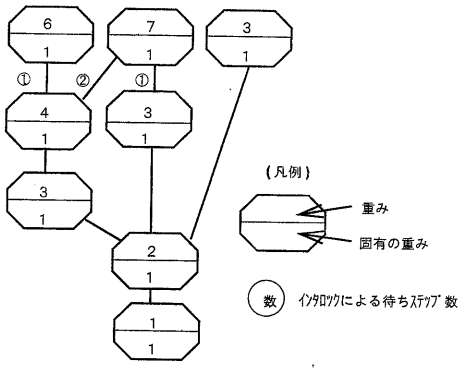

図7．Directer Acyclic Graph

DAG上の命令の重みは帰納的に求める。DAGの終端の命令の重みは命令固有の重みとする。終端命令でない命令の重みは、その命令に依存する命令それぞれについて、依存する命令の重みとはさまれるエッヂの重みの和の最大値とする。すなわち、命令の重みは、その命令から終端に至る経路に沿って継続実行したときのクロック数の最大をほぼ反映するものである。

5．2．2．命令スケジューリング

命令スケジューリングが参照する情報は静的情報と動的情報に分類できる。静的情報は、スケジューリング以前にDAGのみにより決まる情報であり、DAG上の各命令については重みのほか直接依存する命令の数、エッヂについてはインタロックの最大クロック数の情報がある。動的情報は、スケジューリングの過程で決まる情報

であり、既にスケジューリングされた命令により決まるハードウェア・リソースであるレジスタ、バス等の使用状況、命令の実行状態等の情報である。命令に関する静的情報としては、重みを直接依存する命令の数よりも重視する。

命令のスケジューリング方法は、基本的には

・後続命令とインタロックを起こす命令は後続のインタロックに応じて優先的に選択する
・浮動小数点演算については、特に後続命令とのインタロックを起こし易いので、特別に優　先的に選択する
・その時点で最後に選択されている命令と並列実行可能な命令は優先的に選択する
・既に選択された命令によりインタロックが生じる命令は選択しないようにする

であり、パイプライン・スケジューラの一般的な技法に従っている。具体的には、上に挙げた４点それぞれに対して決めた定数を静的情報である重みに加減し、いわば動的な重みを求めることにより、命令の選択を行なう。

５．３．効果

本命令スケジューラによる効果を見るためにいくつかのベンチマーク・プログラムについてシミュレータ上で実行、測定した。使用したベンチマーク・プログラムは、

dhrystone
livermore(#2,#3,#4,#5,#6,#8,#13)
linpack(87s,87d)

である。なお、シミュレータ上での実行を考慮して、プログラムを部分的に書き換えた。ループ回数については、dhrystone と livermore については１０回、linpack については各ループを約５分の１に短縮した。

コンパイラとアセンブラは、当社ＡＳシリーズ付属のものをそれぞれ使用した。コンパイラの最適化オプションについては、無し、レベル２、レベル３について測定を行なった。

以下の表における数値は、左からインタロック・サイクル数、並列実行サイクル数、及び実行サイクル数それぞれのスケジューリング版の非スケジューリング版に対する比率である。また、左端の数の０は最適化無し、２は最適化レベル２、３は最適化レベル３を表す。

表８．dhrystone 実行サイクル数比

dhrystone	ｲﾝﾀﾛｯｸ	並列	実行
0	0．813	1．070	0．952
2	0．787	0．983	0．969
3	0．813	0．982	0．974

表９．livemore　#2 実行サイクル数比

livemore#2	ｲﾝﾀﾛｯｸ	並列	実行
0	0．882	3．196	0．799
2	1．068	1．461	0．963
3	0．905	1．220	0．957

表１０．livemore　#3 実行サイクル数比

livemore#3	ｲﾝﾀﾛｯｸ	並列	実行
0	0．876	1．860	0．879
2	0．996	2．321	0．867
3	0．665	1．276	0．896

表１１．livemore　#4 実行サイクル数比

livemore#4	ｲﾝﾀﾛｯｸ	並列	実行
0	0．731	1．835	0．850
2	0．564	1．233	0．874
3	1．118	1．191	0．985

表１２．livemore　#5 実行サイクル数比

livemore#5	ｲﾝﾀﾛｯｸ	並列	実行
0	0．735	2．228	0．873
2	0．973	1．371	0．959
3	0．967	1．073	0．984

表１３．livemore　#6 実行サイクル数比

livemore#6	ｲﾝﾀﾛｯｸ	並列	実行
0	0．789	2．215	0．835
2	1．120	1．281	0．990
3	0．912	1．087	0．961

表１４．livemore　#8 実行サイクル数比

livemore#8	ｲﾝﾀﾛｯｸ	並列	実行
0	0．915	1．994	0．812
2	0．884	1．617	0．905
3	0．930	1．721	0．914

表１５．livemore　#13 実行サイクル数比

livemore#13	ｲﾝﾀﾛｯｸ	並列	実行
0	0．753	1．449	0．869
2	0．973	1．492	0．954
3	0．974	1．406	0．959

表１６．linpack87s 実行サイクル数比

linpack87s	ｲﾝﾀﾛｯｸ	並列	実行
0	0．６６６	1．３３２	0．８３８
2	0．８９０	1．２４３	0．９３２
3	0．８４９	1．０６０	0．９３１

表１７．linpack87d 実行サイクル数比

linpack87d	ｲﾝﾀﾛｯｸ	並列	実行
0	0．６５７	1．３９０	0．８３８
2	0．８９５	1．２３９	0．９３９
3	0．８９５	1．１１９	0．９４９

インタロックの減少は 43.6%(livermore#4 最適化レベル２) から −12.0%(livermore#6 最適化レベル２)、並列実行の増加は 219.6% (livermore#2 最適化無し) から −1.8%(dhrystone 最適化レベル３)、実行サイクル数の減少は 20.1%(livermore#2 最適化無し) から 1.0%(livermore#6 最適化レベル２) となっている。

ＦＬＡＲＥでは整数命令どうしの並列実行が限られているため、浮動小数点演算を含まないプログラムである dhrystone での命令スケジューラの効果は小さい。一方、livermore 以下の浮動小数点命令を含むプログラムにおいては、実行サイクル数で比較すると、最適化無しの場合で 15% から 20% 程度、最適化レベル２では 5% から 15% 程度、最適化レベル３では 5% から 10% 程度の改善効果が現れている。

しかしながら、実行サイクル数で見れば命令スケジューラによる性能の劣化は生じてないが、いくつかの例においてインタロックの増加、並列実行の減少が見られる。並列実行を増加させたために生じるインタロック、インタロックを回避するために減少する並列実行の存在がある。インタロックの減少の要求と並列実行の増加の要求は、このように対立する場合がある。この２つの要求のより良いバランス関係を求めるための改良が必要である。

６．まとめ

ＳＰＡＲＣアーキテクチャ（Ｖ７）に基づいた２命令スーパースケーラマイクロプロセッサ”ＦＬＡＲＥ”をＨＳ１．０μＣＭＯＳスタンダードセル方式で開発できた。しかしながら性能評価結果から、まだ以下の改良の余地が残されていることがわかった。

・実行性能向上にともなう命令の供給問題。
　分岐予測による分岐実行時の命令の先取り。
　命令キャッシュのヒット率の向上。
・並列実行度の向上。
　整数演算の並列実行化。
　３命令以上の並列実行化。

今後ハードウェアにはこのような改善をほどこしていく。

また、ＲＩＳＣの高速化にはコンパイラの改善が必要不可欠である。今回はコードオプティマイザにとどめたが、命令間のインタロックはコンパイラにさかのぼって改善して行かなければならない。

７．謝辞

最後に、ＦＬＡＲＥの開発にあたり、数々の半導体技術の支援とＬＳＩレイアウトのために惜しみない努力をいただきました（株）東芝半導体事業本部ＩＣセンター産業用ＬＳＩ開発部の技術者に感謝いたします。