SING チップ記述書

牧野淳一郎

October 2, 2006

Abstract

本稿は SING (Sing Is Not GRAPE) チップの記述を与える。形式的な仕様ではなく、何をするか、どういうものかという記述になる。

SING の基本的概念は、1 チップ上に比較的単純なベクトルプロセッサを非常に多数集積するアーキテクチャである。単純にベクトルプロセッサを集積すると、システムの性能はオフチップメモリへの通信バンド幅で決まってしまい、多数のプロセッサを集積しても性能向上は得られない。SING では、計算負荷の高いアプリケーションでは多くの場合に適切なコーディングやアルゴリズムの変換により必要メモリバンド幅を劇的に下げることができることに注目し、オフチップメモリバンド幅が比較的小さいままで非常に多数のプロセッサを 1 チップに集積する。プロセッサは演算器とレジスタファイルを持ち、チップ内の制御プロセッサから放送される命令を SIMD 的に実行する。制御プロセッサはまた全プロセッサのレジスタにデータを放送したり、個別のプロセッサのレジスタを読み書きすることができる。また、プロセッサ内に reduction tree をもたせることで、プロセッサにまたがった縮約演算を高速に実行する。

Contents

1 更新履歴 **5**
1.1 2004/8/4 . . . 5
1.2 2004/8/16 . . . 5
1.3 2004/8/23 . . . 5
1.4 2004/8/25 . . . 5
1.5 2004/8/30 . . . 5
1.6 2004/9/17 . . . 5
1.7 2004/10/12 . . . 6
1.8 2004/11/2 . . . 6
1.9 2004/11/29 . . . 6
1.10 2005/2/3 . . . 6
1.11 2005/2/13 . . . 6
1.12 2005/3/5 . . . 6
1.13 2005/3/18 . . . 6
1.14 2005/3/23 . . . 6
1.15 2005/4/4 . . . 7
1.16 2005/4/5 . . . 7
1.17 2005/4/7 . . . 7
1.18 2004/4/13 . . . 7
1.19 2005/5/9 . . . 7
1.20 2005/5/26 . . . 7
1.21 2005/6/2 . . . 7
1.22 2005/6/15 . . . 7
1.23 2005/7/27 . . . 8
1.24 2005/7/29 . . . 8
1.25 2005/7/30 . . . 8
1.26 2006/10/2 . . . 8

2 TODO **8**

3 はじめに **8**

4 チップの基本的構成 **9**
4.1 基本概念 . . . 9
4.1.1 PE . . . 10
4.1.2 放送ブロック . . . 10
4.1.3 結果縮約ネットワーク . . . 11
4.1.4 ベクトルアドレス生成シーケンサ . . . 11
4.1.5 注意 . . . 11
4.2 外部インターフェースと動作モード . . . 11
4.2.1 命令ストリームの詳細 . . . 12
4.2.2 入力データポート . . . 12
4.2.3 出力ポート . . . 12
4.2.3.1 ブロックマスク . . . 13
4.2.4 エラー検出・訂正 . . . 13

5 ハードウェア概要 **13**
5.1 動作クロック . . . 14
5.2 命令ストリーム入力ユニット (ISU) . . . 14
5.3 データ入力ユニット (DIU) . . . 16
5.4 放送ブロック (BB) . . . 18
5.4.1 放送メモリ (BM) . . . 19
5.4.2 PE . . . 20
5.4.2.1 浮動小数点フォーマット . . . 21
5.4.2.2 浮動小数点乗算器の構成 . . . 22

5.4.2.3 浮動小数点加減算器 . . . 24
5.4.2.4 固定小数点 ALU . . . 24
5.4.2.5 条件付き命令 . . . 26
5.4.2.6 T レジスタ . . . 26
5.4.2.7 レジスタファイル (GRF) . . . 29
5.4.2.8 ローカルメモリのアドレスモード . . . 31
5.4.2.9 ローカルメモリの動作 . . . 31
5.4.2.10 ローカルメモリの回路構成 . . . 31
5.4.2.11 アドレス生成ユニットの物理的実現 . . . 32
5.4.2.12 PE の実行ステージ . . . 32
5.5 結果縮約ネットワーク (RRN) . . . 33
5.5.1 RU 命令語 . . . 34
5.5.2 ネットワークトポロジ . . . 34
5.6 出力データストリーム処理ユニット (DOU) . . . 35

6 PE 演算の詳細 35
6.1 浮動小数点データフォーマット . . . 36
6.2 乗算器 . . . 37
6.2.1 仮数の切り出し . . . 37
6.2.2 仮数の丸め . . . 38
6.2.3 仮数乗算 . . . 38
6.2.4 仮数の後処理 . . . 38
6.2.4.1 正規化する場合 . . . 38
6.2.4.2 正規しない場合 . . . 38
6.2.4.3 丸め . . . 38
6.2.5 指数の処理 . . . 39
6.3 加減算器 . . . 39
6.4 整数 ALU . . . 40
6.4.1 シフト命令の動作詳細 . . . 41

7 命令セット定義 41
7.1 PE 命令 . . . 41
7.1.1 ループ長 (LL) . . . 42
7.1.2 M レジスタ制御 (MRC) . . . 42
7.1.3 T レジスタ制御 . . . 42
7.1.4 アドレスに関する規約 . . . 43
7.1.5 レジスタファイル制御 (RFC) . . . 43
7.1.6 ローカルメモリ制御 (LMC) . . . 43
7.1.7 FMUL 制御 (FMC) . . . 44
7.1.8 FADDSUB 制御 . . . 44
7.1.9 IALU 制御 . . . 45
7.1.10 FADDSUB/ALU 出力セレクト . . . 45
7.1.11 BM 制御 (BMC) . . . 45
7.2 命令コードの長さ . . . 45
7.3 IDP/BM 命令 . . . 46
7.4 RRN 命令 . . . 47

8 チップ I/O 定義 48
8.1 クロック入力 . . . 48
8.2 分周クロック出力 . . . 48
8.3 リセット入力 . . . 48
8.4 IDP モード指定 . . . 48
8.5 エラーステータス出力 . . . 48
8.6 ISP . . . 49
8.7 IDP . . . 49
8.8 ODP . . . 49
8.9 チップ状態表示 . . . 49

9 メモ **50**
9.1 統計情報等の収集について 50
9.2 IALU 即値 50
9.3 IDP ストローブをどうするか？ 50

10 名前 **51**

11 メモ **51**
11.1 IDP 同期とかについて 51
11.1.1 2005/7/7 51

12 略語定義 **51**

1 更新履歴

1.1 2004/8/4

更新履歴をつけ始める。PE の各パーツについて一応書いた。次は RRN と DOU で、それが終わったら命令マイクロコードの記述とシミュレータ作成。それがすめばほぼ終わりだ。本当かな？

レジスタファイルは 32 語とする。 IBM のライブラリのレジスタファイルでは 16 を 32 にしてもあんまり大きくならないため。

1.2 2004/8/16

制御コードを含めてとりあえず全部書いたつもりになった。いろいろ足りないはずなのでこれから全体をチェックしなおすこと。

1.3 2004/8/23

文書名、チップ名を SING で統一する (VPM は使わないことにする)

浮動小数点加減算から ADJUSTB を取る。元々の機能はこういうもの:

ADJUSTB	B の指数を強制的に A に合わせる

これを使いたいような場合、A に積算していて A は非正規化数であるので、まともに計算できるためには A の指数のほうが大きい必要がある。で、この時には別にこの指定がなくても B の指数は A に合うので、この指定は必要ではない。

B が正規化数で本当に指数が A より大きい時には、疑似固定小数点積算としてはどうせまともに計算できてないけど B に指数を合わせてしまって普通の浮動小数点加算をしてくれたほうが多分ありがたい。

1.4 2004/8/25

ステータス出力に関する記述を追加。

RRN 命令にフィールドを追加。

ODPOE	ODP から出力する
SREGEN	ステータスレジスタに書き込む

1.5 2004/8/30

「PE 演算の詳細」の章を作成

PE ALU の演算結果フラグの説明を変更

チップのステータス出力の項を追加

ローカルメモリのアドレスモードにコンスタントストライドを追加

1.6 2004/9/17

アドレスに関して、レジスタファイル、ローカルメモリは全て 36 ビットワードでアドレスするという規定にした。これに応じてレジスタアドレス等の値を変更した。

RRN 命令で PE 番号 を指定することになっていた。これは BB を指定するのでそのように修正した。

RRN 命令に転送語数がなかったので追加した。

RRN ツリーのトポロジと、ノード番号と RRNUNIT ポート番号の対応を定義した。

T-reg のブロック図修正。アドレスがなぜか 4 ビットになっていたのを 2 ビットにする。

T-reg からは論理的にデータ用とアドレス用の 2 種類の出力を出すことにしたので、Treg 制御の LMADR は不要になった。これに関係する記述を削除。

1.7 2004/10/12

BMC に ISEL フィールドがあったのを削除。これは FADD/IALU の A ポートと同じものを使う。

1.8 2004/11/2

BM 論理アドレスが 12 ビットだった (間違い) を 13 ビットに修正。

1.9 2004/11/29

LM ADRI の仕様を明確化。 ADRI=1 の時に短語でも長語でも連続アクセスになる。

1.10 2005/2/3

浮動小数点乗算器の仕様変更作業を開始。主な変更は、 25x25 だった乗算器を 50x25 にしたことである。
入力フラグの変更は SHIFT25A をとって代わりに ROUND50A をつけただけ。

1.11 2005/2/13

浮動小数点乗算器の、入力が無限大とか 0 の時の動作を規定。無限大を優先する。その次に 0 がくる。

1.12 2005/3/5

BB の OV が BMR で制御されることに関する説明を追加。

1.13 2005/3/18

- 即値ストア命令、 logical not 命令を追加
- IALU フラグ出力の説明を更新。全ての命令でフラグが出ることを記述
- 命令コードのビット長が間違っていたのを訂正。合計 108 ビットになる。

1.14 2005/3/23

- IALU シフタの動作でシフト量が 71 を超える時の記述を追加した。

1.15 2005/4/4

- 乗算器で指数処理の説明がおかしかったのを修正した。デフォルトで引く量は 0x3ff ではなく 0x3fe である。

1.16 2005/4/5

- 乗算器で指数処理の説明がおかしかったのを修正した。演算結果の仮数が 0 になる場合には指数は無条件に 0 になるのが正しいが、この場合の記述がなかった。

1.17 2005/4/7

- 整数 ALU で最大値やフラグの処理がおかしかったものを修正した。

1.18 2004/4/13

- fmul から faddsub への直結フィードバックパスをつけた。7.1.8 節と図 10

1.19 2005/5/9

T レジスタ shorstop 動作がマスクレジスタを無視することを明記した。

1.20 2005/5/26

以下はシミュレータ、アセンブラ、アセンブラ記述で未実装

- PE-¿ BM のデータパスがこれまでは FADD/IALU A ポート入力と共通であったものを GRF B 出力ポート固定とした。
- IDP パケットヘッダに BB 全部にシーケンシャルにデータを送るかどうかを指定するフラグをつけた。
- 入出力仕様を LVDS ではなく HSTL を使うものとした。さらに、 IDP ポートについては 18 ビットや 36 ビット毎にしかクロックがこないものを可能にした。これらは QDR SRAM を直接サポートするためである。
- 上のために入力モードを指定するピンをつけた。 IDPMODE[0,1] である。

1.21 2005/6/2

8 節を修正した

- ISPHASE, IDPHASE 信号を追加した
- PHASEERR 信号を追加した
- リセット信号で入力シンクロナイザもリセットされることにした

1.22 2005/6/15

IDP 入力仕様を修正した

- DS0/1 とあったものを DDR の DS 1 本にした

1.23 2005/7/27

IDP/RRN の語数指定、 0 の時に 256 語と解釈する仕様にした。

1.24 2005/7/29

即値の T-register への書き込みについて曖昧であったものを修正した。

1.25 2005/7/30

ref(sect:idp-bm-inst) のブロックマスクレジスタの記述を修正した。自分だけではなく全 BB の情報を持つ。

1.26 2006/10/2

フラグ設定が、演算結果が正の場合にセットされるとなっていたが実際のチップは非負の場合であったので記述を修正した。

2 TODO

入出力のピン配置等の規定を現状に合わせて修正する必要あり。 ODP はピン数が変わっている。また、 IDP, ODP についてはパリティの規定が曖昧である。

また、 loopback について記述がない。

また、リセットについても追加必要。

(2005/8/14)

3 はじめに

SING は、基本的には 1 チップに非常に多数の演算器を集積し、限られた範囲の問題に対してではあるかもしれないがそれらを有効に使うためのアーキテクチャである。 そのような方向を目指すアーキテクチャは、IBM, Sun, Intel, AMD 等の目指す On-chip Multiprocessor の他、再構成可能プロセッサ、SIMD 演算ユニット等様々なものがあるが、 SING では SIMD 動作する多数の (単純な) ベクトルプロセッサを多数集積するというアプローチを取る。

SIMD プロセッサの場合、大きな問題は、1 チップに多数のプロセッサを集積すると、プロセッサ数とその速度の比例して外部メモリへのバンド幅を増やす必要があることである。現在の半導体技術では、チップ間接続のバンド幅と 1 チップで実現可能な演算性能のギャップは極めて大きくなっており、これが SIMD プロセッサが使われなくなった理由であるといえる。

SING では、この問題を「外部メモリへのバンド幅を増やさない」という単純な方法で「解決」する。重力多体問題や、密行列の演算のような、データ量に対して演算量が多く、 n 個のデータに対して n^2 の演算、あるいは n^2 個のデータに対して n^3 の演算をする、しかも演算を並列処理可能な応用では、プロセッサ側にはレジスタファイル程度のものだけをおき、そこに局所的に必要なデータを保持し、計算中は外部メモリから全プロセッサにデータをブロードキャストしながら演算を行うというアプローチが可能である。これが SING の基本的なアイデアである。

実際には、例えば 1024 個のプロセッサが全て同じデータを外部メモリから受け取るのも使い勝手が悪いので、プロセッサを適当な数、例えば 32 個のグループに分けて、グループ毎に共有メモリをもたせる。同じグループのプ

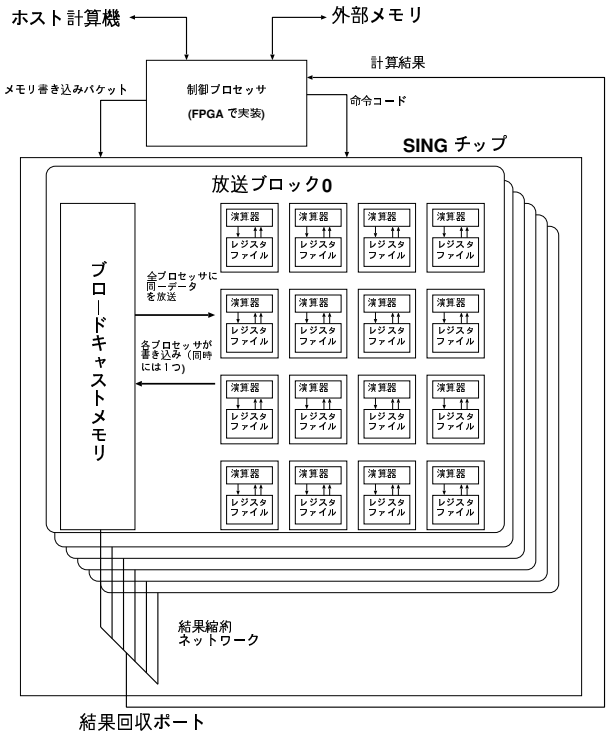


Figure 1: SING の基本構成

ロセッサは同じメモリからデータを受け取る。但し、それぞれの共有メモリには、やはり外部メモリから書き込むことになる。

この詳細は図 1 に示す。グループ間にまたがって結果を合計 (縮約) することも可能にする。開発する SING プロセッサチップ自体は、なるべく内部に制御回路やシーケンサ等をもたない構造とし、各プロセッサへの命令を直接外部の制御プロセッサから投入する。このプロセッサは FPGA で実現する。この場合、FPGA からプロセッサチップへの命令ストリームのために必要なバンド幅が性能ボトルネックになる可能性があることと、パイプラインレイテンシのための命令スケジューリングの手間を省くために基本命令はベクトル命令とし、4-8 個程度の短いベクトルを 1 命令でパイプライン処理する。

全体システムは、このチップ+FPGA+メモリが数個のったボードを普通の PC に 1-2 枚挿したシステムのクラスタとして実現する。いかにしてその上でアプリケーションを並列化するかといった問題はここでは扱わない。

以下、3 章ではチップの基本的な構成についてまとめる。4 章ではチップ各部の詳細な記述を与える。5 章では、PE の演算器について、その動作の詳細をまとめる。 6 章では命令セットを定義する。7 章ではチップの I/O の詳細を定義する。 8 章以下はおまけである。

4 チップの基本的構成

4.1 基本概念

図 1 に SING の基本概念を示す。

一つの SING チップは、「放送ブロック」が多数集積され、一つの放送ブロックは多数の演算要素 (Processor Element, PE) からなるという階層構造を持つ。チップ単体では完結した計算機ではなく、外付けの制御プロセッサから命令ストリームを受け取る。命令ストリームは基本的には水平型マイクロコードであるが、命令のために必要な通

信バンド幅を減らすためにベクトル命令とする。また、同じく制御プロセッサを通して外部メモリ、ホスト計算機と通信する。

以下、下の階層から順に概略を述べる。

4.1.1 PE

PE は、浮動小数点/固定小数点/論理演算の演算器、アドレス生成ユニット (テーブルルックアップのための間接アドレッシング機能をもつ)、ローカルメモリからなる。 PE は自分のローカルメモリにあるデータを使って演算する他、自分の上位にある放送メモリからデータを受け取って、選択的に (ある PE だけが)、あるいは非選択的に (全 PE が) ローカルメモリに書くことができる。

PE の動作は SIMD であり、全 PE が (マスクレジスタによってディスエイブルされなければ) 同一動作をする。これは古典的な SIMD 超並列計算機と変わらない。

4.1.2 放送ブロック

PE が複数 (今回の実装では 32 個集まって) 放送ブロックを構成する。放送ブロックは、放送メモリと PE からなる。放送メモリは、同一ブロック内の PE と以下の 3 通りの方法で通信する。

1. データを放送する (PE から見ると普通のロード命令)
2. 1 つの PE にデータを転送する (PE の ID 番号を条件にする条件実行ロード命令)
3. 1 つの PE がデータを書き込む (PE の ID 番号をパラメタにとるストア命令)

PE からのアクセスとは独立 (並行) に、チップ外の制御プロセッサは放送メモリにランダムアクセス、あるいは全放送ブロックに同一データを放送することができる。放送メモリのデータを制御プロセッサが読み出すのは後述の結果縮約ネットワークを通してになる。

このように、外部から放送ブロックへの書き込みと、放送ブロック内での PE への書き込みの両方に、放送モードとランダムアクセス (非放送) モードを設けることで、 外部 (制御プロセッサ) 側からみると

- 全 PE への同一データ放送
- ある放送ブロック内の PE への放送
- 放送ブロックにまたがった、同じブロック内 ID を持つ PE への放送
- 個別 PE への非放送書き込み

の 4 種類の書き込みが可能になる。これらを使い分けることで、

- 従来の GRAPE のエミュレーション
- 行列演算
- その他、データ量に対して演算量が多い数値計算

を実行する。

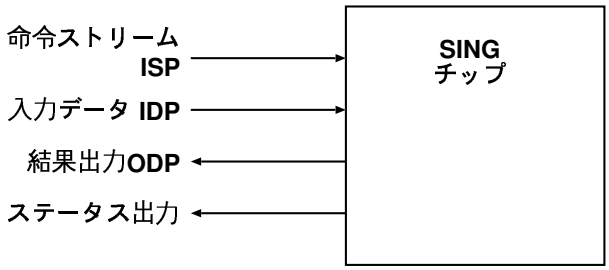


Figure 2: チップの外部インターフェースの概念

4.1.3 結果縮約ネットワーク

SING チップが計算結果を制御プロセッサに返すのは、結果縮約ネットワークを通してである。 結果縮約ネットワークは、各放送ブロックからの出力を、加算、論理演算等の縮約演算をしながら返すツリー型のネットワークである。基本的な動作モードは、PE がストア命令で放送メモリに書いたデータを縮約しながら読み出して出力するというものである。

4.1.4 ベクトルアドレス生成シーケンサ

図には示してないが、ベクトルから実際にレジスタファイル等を制御する信号を生成する簡単なシーケンサが存在する。チップ内のデータ転送を減らすためにこのシーケンサは放送ユニット毎に設け、制御プロセッサから放送ユニットまではベクトル命令のままで伝達されるものとする。これらの動作の詳細は、詳細記述においてはレジスタファイル等の記述のところでまとめて述べる。演算器の動作制御はベクトル命令で、同じ命令の実行中は同じ制御信号をラッチしているだけなので特に述べることはない。

4.1.5 注意

ハードウェアにはインターロックや命令が実行可能かどうかの判断をするロジックは一切実装しない。指定したベクトル長が短い場合や、命令の種類によっては連続する命令間で干渉が起こることがありえるが、これはアセンブラなり人間なりが必要なダミーサイクルを挿入して実行可能コードを生成する。

4.2 外部インターフェースと動作モード

図 2 にチップの外部インターフェースの概念を示す。

命令ストリーム、入力データはそれぞれ専用の入力ポート (ISP, IDP) から供給され、結果出力も専用ポート (ODP) からなされる。

IDP, ODP はパケットベースの単純なプロトコルを持つ。 HyperTransport(HT) の write request だけを実装するようなものを考えて欲しい。

命令ストリームは固定長の命令を常時たれながしなので、プロトコルといっても命令ワードをシリアライズして送るというくらいである。

物理的な接続にはシングルエンドでソースシンクロナスなクロックが使える、例えば HSTL か SSTL を使う。

当初はこれは LVDS を使うことを想定していたが、大袈裟過ぎる気がするのと IDP ポートにメモリを直結できるようにしたいので SSTL か HSTL にする。具体的にメモリとしてつける可能性があるのは QDR SRAM やパイプラインデュアルポート SRAM なので、 HSTL が適当と考えられる。

ステータス出力は、エラーステータス、チップの動作状態、診断出力レジスタ等を出力する。これは非同期出力であり、CMOS レベルの出力である。

4.2.1 命令ストリームの詳細

命令は SIMD であるとしても基本的には 1 サイクルに 1 命令である。命令長は 2004/8/17 現在では 100 ビット強なので、 110 ビットとする。これを供給するために必要な転送バンド幅は、例えば 1 GHz 動作だと 14GB/s になってちょっと大き過ぎる。パリティを付けるともっと増える。

これは、実際にはベクトル命令が基本ということで、4 サイクルに 1 命令供給すればいいことにして 4 GB/s 程度、 1GHz のクロックで 32 ビット並列転送とする。

4.2.2 入力データポート

IDP からの入力は BM だけに書く。ローカルメモリに直接書けるようにしてもいいが、 BM と LM の転送命令はどうせあるので重複するからである。

書き込みは、

1. 放送
2. 単一の BM への書き込み
3. 複数の、しかし全部ではない BM への書き込み

の 3 モードがありえる。 GRAPE-6 の場合と同様に、BM の番号指定とマスク指定を組み合わせたアドレス指定とすることで、単純な 2 のべきの数のサブグループは指定できるようにする。

さらに、放送ではなく、全 BB にそれぞれ違うデータを書くことも可能にする。

BM は PE へのデータ放送と IDP からの書き込みが並列に動作できるものとする。これによって多くの場面でデータ転送と計算をオーバーラップ動作させて実効性能を上げることができる。

このため、 BM はデュアルポートメモリを使って実装する。

4.2.3 出力ポート

出力ポートは BM のデータを縮約しながら、あるいはランダムアクセスで指定した放送ブロックの BM のものを、出力する。

出力指定コマンドは IDP から投入する。コマンド自体は PE のコマンドと類似になるが、実行シーケンサは全く別のものとなる。

命令で指定される必要があるものは

- 縮約かどうか
- 縮約でなければ BM の ID
- 縮約の時には縮約命令の種類
  - ブロック浮動小数点加算
  - 一般の浮動小数点加算。
  - 最大値
  - 最小値
- 開始アドレス
- 語数 (ベクトル長)

ストライドとかは不要。縮約演算でも一般の浮動小数点加算を許す。論理演算とかは、and, or は付ける。語長に関しては、面倒なので長語のみとする。入力が短語の時は下半分が 0 拡張される。

4.2.3.1 ブロックマスク

特に縮約を伴う出力の時に、全てのブロックの結果を合計したくはない場合がある。縮約に参加しないブロックは、縮約ネットワークに output valid を出さない。縮約ネットワークの ALU は、入力の一方が output valid でなければ他方の入力をそのまま出力に出し、両方 output valid でなければ出力も output valid を出さない。

ブロックマスクは、単純に IDP からブロックマスクレジスタに書き込みを行うことで設定する。つまり、 IDP からは、BM への書き込みの他、放送ブロックに属するレジスタへの書き込みを行う。

4.2.4 エラー検出・訂正

入出力ではエラー検出・訂正をしたいような気がする。現在の GRAPE-6 の場合、誤動作は

- 入力データのパリティエラー
- HIB 上での FIFO の良くわからない誤動作。ノイズでデータが消える/ゴミが入る等。

である。設計段階でもっとも心配した、メモリとプロセッサチップの間のデータ転送でのエラーは、折角 ECC をつけたのにほとんど起きていない。この経験からすると、ECC を付けるのはなんとなくむなしい。

しかし、全体システムとしてみた時の速度は大幅に上がるので、デバイスのエラーレートが変わらなくてもシステムとしてのエラーレートは大幅に上昇する。

とはいえ、現在メモリのパリティエラーは実際上「全く」起きていない。また、PLL の同期エラーとかだとどうせ ECC では回復できない。従って、基本的には単純なバイトパリティとする。

エラー検出はいいとして、起こったことをどうやってホストに返すかということが問題である。

GRAPE-6 の場合は返すデータ形式が決まっていたので、その中にエラーステータスをあらかじめ組み込んでいたわけだが、 SING の場合はそういう決まったフォーマットがあるわけではない。

まあ、制御用の FPGA があるわけなので、もっとも簡単には単純に線を出せばよい。ステータスについては物理的な線と読み出し可能なレジスタの両方を設ける。

エラーステータスは、とりあえず

- IDP パリティエラー
- ISU パリティエラー

くらい。

これは、エラークリア命令 (というものを定義する) によってのみクリアされる。

入出力については例えば HT でもやっているように CRC を付けるという考え方もあるが、今回の SING チップでは採用しない。面倒なのと、 ECC を考える場合にはレイテンシが増えるからである。

5 ハードウェア概要

チップ全体は以下の要素からなる。

- 命令ストリーム入力ユニットと放送ネットワーク (ISU, instraction stream unit)
- データ入力ユニットと放送ネットワーク (DIU, data input unit)
- 放送ブロック (BB, broadcast block)

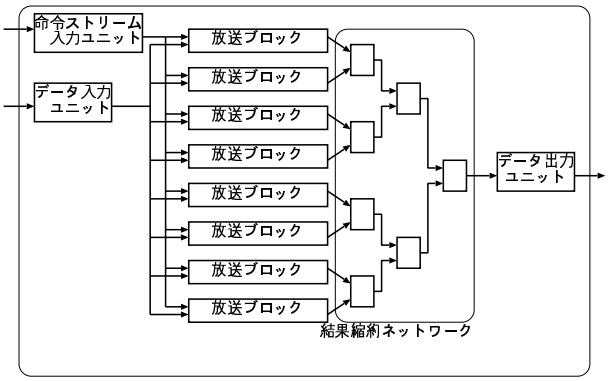


Figure 3: SING の基本構成

- 結果縮約ネットワーク (RRN result reduction network)
- データ出力処理ユニット (DOU, data output unit)

図 3 にブロック間の関係を示す。

ISU, DIU についた放送ネットワークはそれぞれの出力を全 BB に配るだけの単純なものである。ここは適当なパイプライン遅延 (全 BB で同じ) があってもかまわない。以下、ISU から順に内容を述べる。

5.1 動作クロック

ブロック間の関係を述べる前に、動作クロックについてまとめておく。PE 自体の動作クロックの目標は 700MHz から 1GHz 程度である。この、PE の動作クロックのことを以下基本クロックまたはベースクロックと呼ぶ。

I/O の動作クロックは基本的はベースクロックと同じとする。これは、差動信号 (LVDS 等) を使うことを考えるならこの程度の動作速度はそれほど難しいわけではないからである。

また、一つの BB の中は全てベースクロックに同期して動くものとする。これは放送メモリも含めてそうであるとする。物理的なサイズが小さいのでそれほど大変ではないであろう。

チップ全体に渡る 3 つのネットワークについては、それぞれ以下のようにする

- ISU の放送ネットワーク: 命令自体は 4 サイクルに 1 語のレートで入ってくるので、放送ネットワークの入力の前で命令語の組み立てを行ってネットワーク自体は 1/4 のクロックレート (これを以下分周クロックと呼ぶ) で動作するものとする。これによりグローバルなネットワークに必然的な長い配線での遅延に対して設計上の余裕をもたせる。
- DOU の縮約ネットワーク: これも ISU 放送ネットワークと同様に 分周クロックで動作させる。
- DIU の入力データ放送ネットワーク: 可能ならばこれも遅くしたいが、高い実効性能を実現するためにはここを余り遅くはできない。これについては 72 ビットのデータをそのままベースクロックで配る必要がある。この部分がチップの物理設計としてはもっとも大変なところになる可能性が高い。パイプライン遅延はあってもかまわないので放送ネットワークに初めから適当な段数のパイプラインレジスタをいれておくことが必要であろう。

5.2 命令ストリーム入力ユニット (ISU)

ISU の機能は単純であり、入力ポートから入ってきた命令語を組み立てて全放送ブロックに送るだけである。

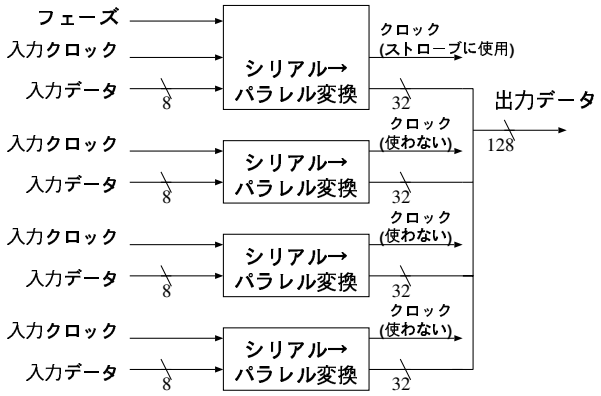


Figure 4: ISU の構成

最大の問題は、物理的な信号レベルに何を使うかである。通信は FPGA と行うので、 FPGA で容易に使えるものである必要がある。Xilinx の場合でも、 Altera の場合でも、 HSTL をサポートするので、物理層はこれを使うことにする。

上で述べたようにデータレートは遅いわけだが、チップ外からの入力のところでは逆にクロックを上げて信号線数を減らすことにしたい。線の数が多いのは大変だからである。具体的には、命令語 110 ビット (パリティつけて 124 ビット) として、これをクロックを 4 倍してベースクロックと同じくレートにして幅の細いパラレル転送にする。伝送自体には DDR SRAM で使われている、クロック上下でサンプルし、そのために 2 相クロックが供給されるインターフェースを使う。さらにクロックの半分の周期のフェーズを示す信号をつけることで命令ワードの最初を指定する。

ここでは PLL を使って LVDS を使った ChannelLink (NS DS90CR21x のような) と同じく分周したクロックでシリアル化したデータの順番を指定することにする。

図 4 に ISU の構成を示す。HT と同様にデータ 8 本毎にクロックがあるものとするので、入力のシリアル・パラレル変換ブロックは 8 → 32 のものが 4 ついる。これを合わせて 124 ビットの命令語になる。クロック当りのデータ線は 16 本程度までは問題ないかもしれないが、 HT に合わせておくのが無難と考えられる。

基本的なタイミングチャートは例えば以下のようになる。ここで入力で ”Dxy” は x 番目のワードの y 番目のフラグメントを意味している。 Dx0 から Dx3 がまとまって Dx として出力される。

```
clkin    ____====____====____====____====____====____====____====____====____====____
phase  ________==========_________==========_________==========_________==========_________
din            <D00><D01><D02><D03><D10><D11><D12><D13><D20><D21><D22><D23>
clkout   ________==========_________==========_________==========_________==========_________
dout                                    <--------D0--------><--------D1-------->
```

clkin と din, phase、および clkout と dout のタイミングの仕様の詳細については別に指定する。

なお、パリティはバイトパリティだが、シリアル化 (高クロック化) したバイト毎にパリティを入れるのではなく、組上がった 128 ビットワードの最後の 14 ビットをパリティとする。つまり、パリティビットは図 4 の下側のモジュールの、4 サイクルの最後に来るワードの下 14 ビットとする。バイトパリティにしないことでクロックずれ等も検出可能にする。

ソースシンクロナスな信号の場合、チップのベースクロックとソースのクロックの関係が問題だが、 ここでは出力クロックは分周クロックであり十分遅く、またこのクロックは (位相はともかく) チップのベースクロックを分周したものとみなせるので、 シリアル・パラレル変換ユニットの出力クロックをストローブとして使って以下はベースクロックで処理する。

もうひとつの方法は、ブロードキャストブロックまでこの入力クロックのまま配って、ブロードキャストブロック

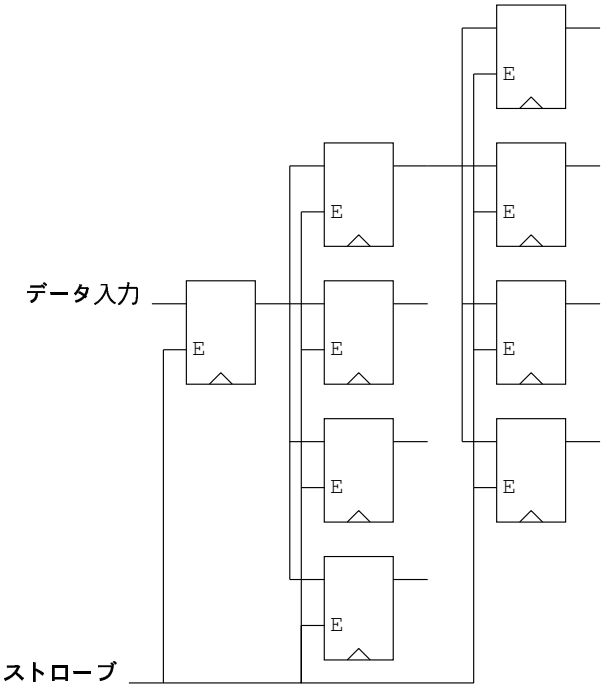


Figure 5: ISU 放送ネットワークの構成。DFF のファンアウトは 4 を仮定し、2 段目は一番上のもの以外省略した。

で初めてベースクロックで動作する回路で受けることである。どちらでもいいが、この場合多相のクロックがチップ全体に供給されるのでなんか嬉しくない気がする。ストローブがチップ全体に配られるのは同じともいえるが、これのスキューは厳しくないし、そもそも途中でパイプラインレジスタを入れることもできる。従って、放送ネットワークの DFF にはベースクロックを供給するものとする。

図 5 に ISU 用の放送ネットワークの概念図を示す。これは単純なファンアウトを取るだけのネットワークで、適当にパイプラインレジスタをいれて配線遅延があってもレイアウト可能にする。ストローブについても 4 段づつパイプラインレジスタをいれることもできる。遅延の段数には特に設計上の制限はない。もちろん、全放送ブロックで段数が同じである必要はある。厳密にいえばおそらく必要はないが、そういう場合の動作を考えるのが面倒なので段数が同じであることにする。

5.3 データ入力ユニット (DIU)

データ入力ユニットは、バンド幅が大きいという以外は ISU とほとんど同じものであるが、バンド幅が大きい分タイミングは厳しくなる。データ入力は 72 ビットあるので、図 5 のような構成ではブロックが少なくとも 8 個必要になる。ここも ECC にする意味はあんまりない気がするのでパリティだけ。

パリティは 9 ビット毎のパリティとすることでデータ総数を 80 本とする。

なお、18 ビット幅や 36 ビット幅のメモリデバイスを接続可能にするため、シリアル・パラレル変換ユニットは

- 9 ビット入力のもの 9 個
- 18 ビット入力のもの 4 個と 9 ビット入力のもの 1 個
- 36 ビット入力のもの 2 個と 9 ビット入力のもの 1 個

の 3 種類の構成をサポートする必要がある。これは入力モードピンで指定する。

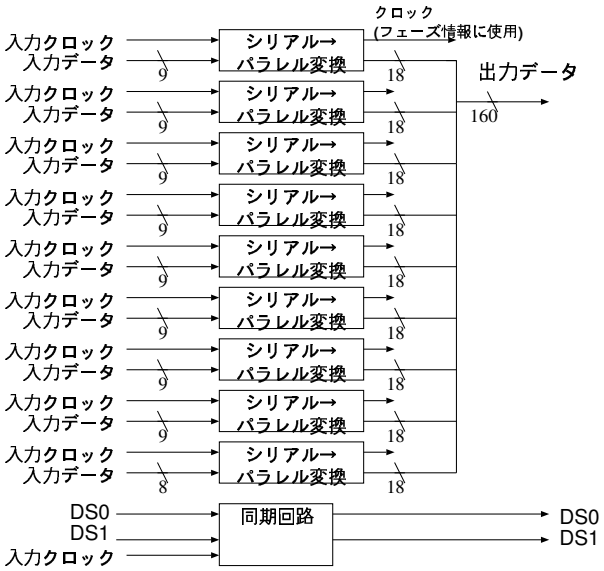


Figure 6: DIU の構成

もしも 1GHz で動いたとすると、そのレートで 9 個の位相があっていないかもしれないクロックでくるデータをなんとかするのはとても大変な気がするので、入力クロックは 1/2 分周したものを使って、一旦デシリアライザでビット幅を 144 に広げてベースクロックの 1/2 のクロックで受ける。これをもう一度ベースクロック同期の 72 ビット幅信号に戻して放送ブロックに配る。

これに伴い、入力クロック自体も 1/2 分周したものとし、 DDR で送る。これによりデータの順番を区別する。

ストローブは DDR で送るのではなくクロック立ち上がりでそのデータと次の立ち下がりでくるデータの 2 本分のストローブを入れる。このため、ストローブは専用クロックを持つ。

問題はチップ上のどこで 160 → 80 の変換を行うかであるが、1GHz のバスラインがチップを這いまわるよりは 160 ビットでも 500 MHz の線のほうが楽であるとするならこの変換は各 BB で行う。途中 DFF をいれてもかまわないので線が少ないほうがいいかもしれない。変換のロジック自体は大した大きさではないので、1 つでも各 BB にあってもかまわない。変換は論理的には、分周クロックでサンプルしたデータを分周クロックを制御入力とするマルチプレクサで切換えるだけである。

入力信号は、クロック、データ+パリティ 80 本、ストローブである。同じクロックで動作できるシリアル・パラレル変換ユニットのビット数によってクロック本数が変わる。 出力は放送ブロックのところではベースクロックに同期したデータとストローブということになる。

DIU の構成を図 6 に示す。基本的に 9 → 18 の変換を行うブロックを並べただけである。最後のビット (最後のブロックの MSB) をデータストローブと解釈し、これがでているものは有効なデータであるとする。以下にタイムチャートを示す。

```
clkin    ____====____====____====____====____====____====____====____====____====
din      ---<D0><D1><D2><D3>--------<D4><D5><D6>-----------------
ds0in    ==\_____________/=======\_________/=================
ds1in    ==\_____________/=======\_____/=====================
clkout   ____====____====____====____====____====____====____====____====____====
dout     ---------------<--D01--><--D23-->--------<--D45--><--D6x-->-----
ds0      ===============\_____________/========\_____________/=====
ds1      ===============\_____________/========\_______/==============
```

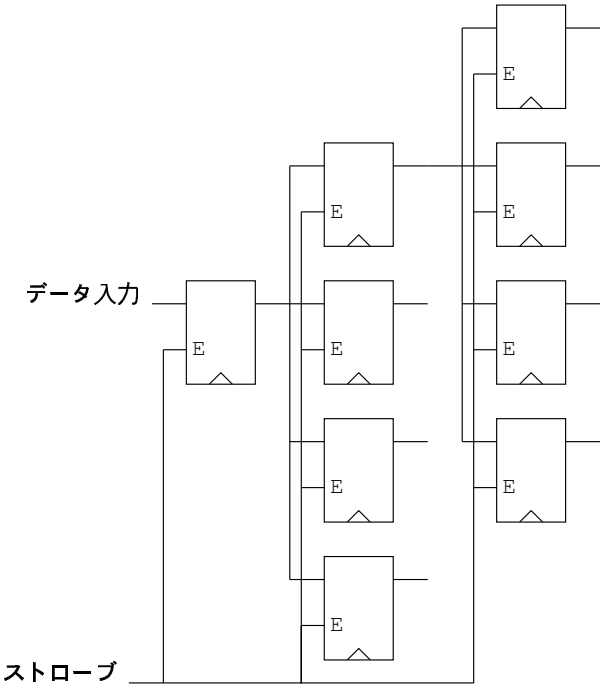


Figure 7: DIU 放送ネットワークの構成。DFF のファンアウトは 4 を仮定し、2 段目は一番上のもの以外省略した。

これでわかるように、ストローブは 72 ビットワード毎に 2 語付加され、入力が出力にコピーされる。つまり、倍幅になった時に 2 本ともに意味がある。また、DIU のレベルではデータとストローブのみが存在し、コマンドやパケットといった概念は存在しない。

実際の転送は 72 ビットワード 1 語のヘッダとその後のデータからなるパケットベースになるが、

- パケットの途中で DS がネゲートされてウェイトサイクルが入りえる。
- 2 つのパケットは連続して (ウェイトなしで) 到着することがありえる。

ということに (放送ブロックの実装では) 注意する必要がある。 DIU のレベルではこれは考える必要はない。

DIU からの出力データも、ISU からのデータと全く同様に全放送ブロックに放送される。

図 7 に DIU 用の放送ネットワークの概念図を示す。これは単純なファンアウトを取るだけのネットワークで、適当にパイプラインレジスタをいれて配線遅延があってもレイアウト可能にする。遅延の段数には特に設計上の制限はない。もちろん、全放送ブロックで段数が同じでないと話が面倒になる。特に、 RRN へのコマンドがここから入るので、段数が同じでないと RRN が正しい動作をしない。

5.4 放送ブロック (BB)

図 8 に BB の基本構成を示す。BB は以下の要素からなる

- 放送メモリ (BM)
- プロセッサエレメント (PE)
- 命令デコーダ (IDU)

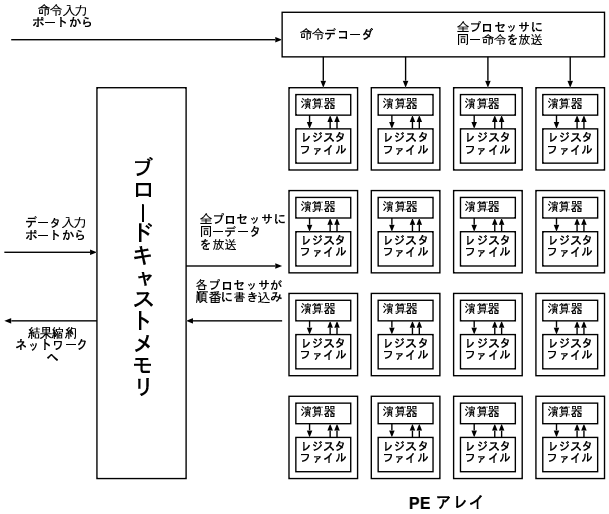


Figure 8: 放送ブロックの基本構成

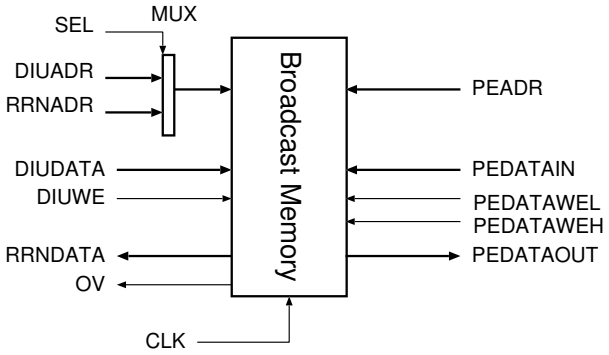


Figure 9: 放送メモリ。

外部との接続は、 ISU は命令デコーダと、DIU, RRN は BM とつながる。PE は直接に外部につながってはいない。

5.4.1 放送メモリ (BM)

放送メモリは 72 ビット 1024 語の完全デュアルポートメモリ (2 アドレスでそれぞれ独立に read/write が指定できる) である。32 個しかないのでチップ全体で 2.3Mbit であり、まあ、 大した大きさではないと考えていいと思われる。

図 9 に BM の構成を示す。ポートはグローバル側とローカル側と呼ぶことにする。グローバル側は DIU からの書き込みと RRN への出力である。ローカル側はプロセッサエレメントへの放送または書き込みである。BM 自体は単純なメモリで特別な機能は何もない。

動作は、とりあえず

- 書き込みデータはアドレスと同じくサイクルで入る。同時に WE もアサートされる。
- 読出しデータはアドレスの 2 クロック後に出る。

という想定で行う。

PE 側からの書き込みは 36 ビット短語単位でも可能にする必要があるので、こちらの WE は 2 本 付けられる必要がある。

5.4.2 PE

各プロセッサエレメント (PE) は

- ALU/FPU
- 汎用レジスタファイル (GRF) 2 read 1 write ポート、 32 ロングワード
- ローカルメモリ (LM) 256 ロングワード
- T レジスタ (補助レジスタ,TREG)。 1R1W

からなり、それぞれのメモリ要素は放送メモリから、または外部メモリポートから書き込み可能である。

外部メモリに書き込めるのは GRF B ポートから読み出したデータのみとする。外部メモリの読み出しは、ランダム読み出し、及び縮約を伴う読み出しが可能である。

LM は基本的に SRAM マクロを使って実装される。動作速度、読み出し遅延等は何が使えるかに依存するが、動作モードは BM と同じく

- 書き込みデータはアドレスと同期
- 読出しデータはアドレスをラッチしたクロックの次の立ち上がりの後ででる

ということを想定する。

GRF をどうやって実装するかは半導体メーカーのライブラリで何が使えるかによるが、 IBM の場合だと非同期読み出し可能なレジスタファイルのライブラリがあり、ワード数 (深さ) が小さい場合にはメモリより小さいのでこれを使うことになる。

TREG は (多分) 物理的に DFF のマトリックスとセレクタで実装される。TREG は GRF に対して補助的な役割を果たす。

図 10 に PE の論理構造を示す。

演算器は浮動小数点乗算器、浮動小数点加算器と、整数 ALU の 3 種を持つ。それらへの入力は基本的には上の T レジスタ、汎用レジスタ、ローカルメモリの 3 つのメモリ要素の出力である。メモリ要素への書き込みは、演算要素の出力または共有メモリからのものである。また、メモリ要素の出力を共有メモリに出すこともできる。この時には 1 つの PE だけが選択される。

演算ユニットは 3 個あるが、今回の実装ではそのうち浮動小数点加算器 (FMUL) は独立した入出力をもたせる。浮動小数点加減算器 (FADD) と整数 ALU (IALU) は入力、出力ともに共有する。つまり、FMUL-FADD, FMUL-IALU は並列動作可能であるが FADD-IALU はどちらかの出力しかストアできない。

また、BM への出力は GRF B ポートからである。

結局、マルチプレクサ A は 4 入力 1 出力の独立なマルチプレクサ 4 個からなる。但し、 FADDSUB/IALU の A 入力については、固定 ID であるプロセッサ番号、放送ブロック番号も入力として選択できるものとする。さらに、乗算器からの直接フィードバックも入力にできる。従って、このマルチプレクサだけは 7 入力となる。マルチプレクサ B も同様に 3 入力 1 出力の独立なマルチプレクサ 3 個である。これらのマルチプレクサの動作はマイクロコードによって指定される。

なお、LM については T レジスタの値をアドレスにすることを可能にする。これにより間接アクセスを実現する。

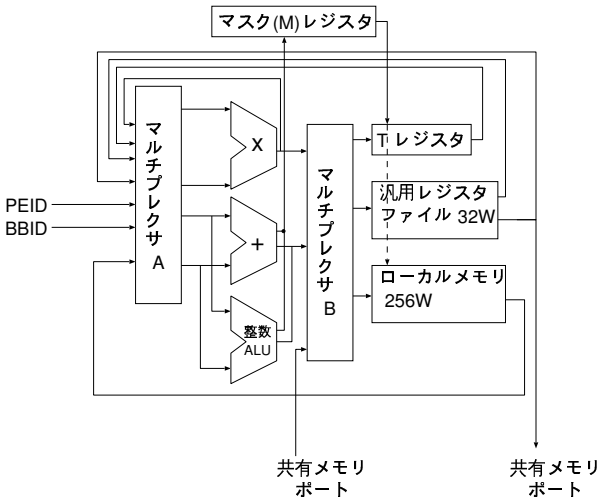


Figure 10: PE の論理的構造

5.4.2.1 浮動小数点フォーマット

浮動小数点データフォーマットは以下の通りとする。

- 短語 36 ビット、長語 72 ビット
- 仮数 24/60 ビット unbiased forced-1 rounding
- 指数 11 ビット
- 符号ビット

指数形式は IEEE-754 と同じ。 短語 で 0x3FF000000 が 1 になるもの。

0 は指数 0

無限大は指数 all 1 (短語で 0x7FFFFFFFF, 0xFFFFFFFFF)

無限大と 0 の掛け算の結果は 0 とする。こういった演算例外についてはアプリケーション側で面倒をみる必要がある。面倒なので NaN はサポートしない。

丸めは bias-corrected force-1 rounding である。これは、 演算結果の MSB の下のビットの一つでも 0 でないビットがあれば、 MSB を 1 にする方式である。

元々 MSB が 1 であれば、これは実効的に切り捨てである。これに対して元は MSB が 0 であった場合は切り上げになる。というわけでほぼ四捨五入になっていることは理解されよう。

しかし、MSB の下が全て 0 であった場合にも MSB を 1 にすると、丸めた後の値の平均が丸めた前と同じにならない。 5 ビットの数の下 4 ビットを丸める場合を考えると、可能な値は 00000b から 11111b (ここでは b がつくのは 2 進数を表すとする) の 32 通りで、平均は 15.5 である。ところが丸めた後が 10000b であるとするともちろん丸めたあとの平均は 16 になってしまう。このバイアスを補正するためには 00000b は 00000b のまま、つまり、下位ビットが all 0 の時には MSB をそのままにする (10000b はもちろんそのままである) ようにしておけばよい。

この丸め方式は、 IEEE 754 で採用されている round-to-nearest-even 方式に比べて

- 圧倒的に実装が簡単である

という利点と

- 精度 1 ビット損をする

という欠点がある。

今回は、1 ビットの損よりも、実装の単純さから来る開発期間の短縮と、回路そのものが単純になることによる高速動作の可能性を期待する。

なお、語形式として、非正規化数をサポートする。これの解釈は、「hidden bit を補わない」というもの。

何故こんなものを使うかというと、

1. 積算、総和の時に、積算順序によらないで結果が同じになるようにする。
2. 倍精度乗算の時に使う。

ためである。

5.4.2.2 浮動小数点乗算器の構成

今回の実装では、浮動小数点演算器が持つ乗算器は 50×25 のものとする。これにより、短語乗算は 1 サイクル 1 結果を出す (乗算器の半分は無駄になる)。また、倍精度乗算は部分積 2 つに分けて、掛け算と掛け算+加算の組合せで実行する。

浮動小数点乗算器では以下の手順で乗算を行う。

1. 仮数の切り出しを行う。パラメタの指定に従って、25 ビットまたは 50 ビットシフトする。また、正規化表現 (ケチ表現) でシフトしない場合には MSB に 1 を補う。同時に指数も仮数のシフトに応じて調整する。つまり、25 または 50 を引く。アンダーフローしたら 0 にする。
2. 仮数を 25 ビットまたは 50 ビットに丸める。これはパラメタの指定に従って、指定があれば 25 ビットに丸める。何も指定がない時には A ポート側は 50 ビットに丸める。
3. 仮数の乗算を行う。入力は A ポート 50 ビット、B ポート 25 ビットで、結果は 75 ビットになる。
4. 結果を正規化する場合には、最上位が 0 なら 1 ビットシフトして、別に計算していた指数から 1 を引く。
5. 結果を正規化しない場合には、無条件に 1 ビット右シフトして、別に計算していた指数に 1 を足す。
6. シフトした結果の最上位を切りとった、残りが新しい仮数になる。結果を短語にしまう場合には 24 ビットに、長語にしまう場合には 60 ビットに丸めを行う。
7. 指数はバイアス表現なので足してからバイアスである 0x3fe を引く。で、上の仮数による調整を行ったあとで、オーバーフローしていたら 0x7ff、アンダーフローしていたら 0 にする。但し、入力の指数のどちらかが 0x7ff の時にはそのままとし、そうでなくてどちらかが 0 であれば 0 とする。また、仮数が 0 になった場合も指数を 0 とする。但し、この場合も入力の指数のどちらかが 0x7ff の時にはそのまま 0x7ff である。

ブロック図を 11 に示す。

乗算器入力ポートの制御線は A と B で異なる。A は

- シフト量 1 ビット (50 ビット)
- 丸めモード 2 ビット (25 または 50 ビット)
- 正規化入力かどうか 1 ビット

であり、B 入力ポートに以下の制御線が必要ということになる

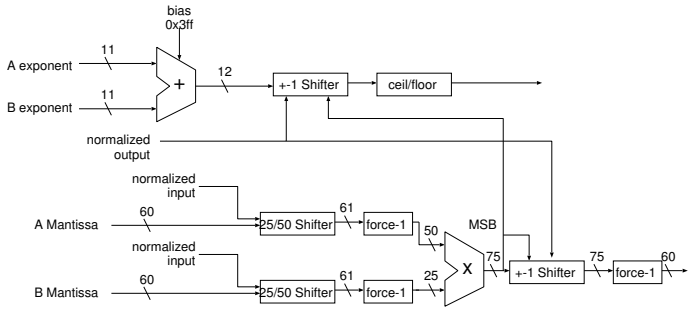


Figure 11: 浮動小数点乗算器

- シフト量 2 ビット (25 または 50 ビット)
- 丸めモード 1 ビット
- 正規化入力かどうか 1 ビット

それぞれ 4 ビットづつである。さらに出力の丸めモードでもう 1 ビット。合計 9 ビット。
つまり、制御線は以下のようになる:

SHIFT50A	ポート A の仮数を 50 ビット上にシフト
ROUND50A	ポート A の仮数を (シフト後)50 ビットに丸める
ROUNDA	ポート A の仮数を (シフト後)25 ビットに丸める
NORMALA	ポート A の入力を正規化数とみなす
SHIFT25B	ポート B の仮数を 25 ビット上にシフト
SHIFT50B	ポート B の仮数を 50 ビット上にシフト
ROUNDB	ポート B の仮数を (シフト後)25 ビットに丸める
NORMALB	ポート B の入力を正規化数とみなす
ROUND	出力を丸める
NORMALO	出力を正規化する

パイプラインレジスタは必要な動作速度を実現できるような段数ということだが、現在の想定では入力ラッチを除いて最大 3 段である (内部 2 段ということ)。図では force-1 ユニットへの制御線が入ってないが、それぞれ独立に命令のパラメタで制御される。
入力のシフト、MSB 1 の追加、丸めの動作がわかりにくいが、整理すると以下のようになる。
1 を補う前の出力は (上シフトしていない場合)24 ビットである。つまり

- 短語で仮数 24 ビットの場合、 force-1 しても結果は変わらない
- 長語で仮数 60 ビットの場合、 force-1 は下 36 ビットを見て 24 ビット目をいじる

この 2 ケースの場合、この段階では出力は 24 ビットでその上に 25 ビット目の 1 を付ける

- 長語で 25 ビットシフトする場合、a[59:0] とあるものの a[35:11] を切り出す。丸めでみるのは a[10:0]
- 長語で 50 ビットシフトする場合、a[59:0] とあるものの a[10:0] を切り出す。丸めはおこらない。

これらの場合、25 ビット目までがシフタの出力となってでてくる。

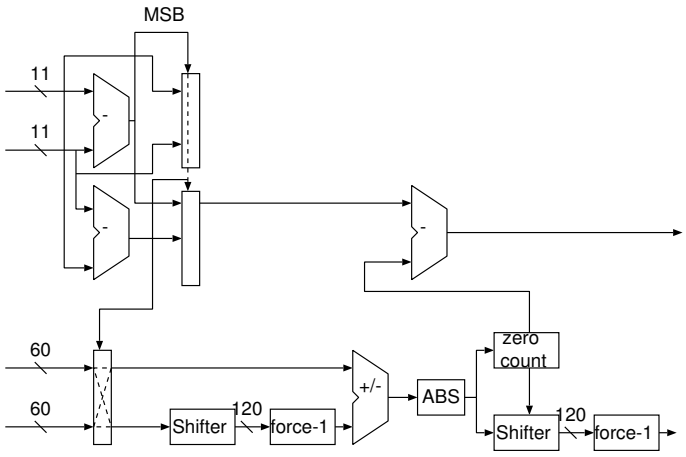


Figure 12: 浮動小数点加減算器

5.4.2.3 浮動小数点加減算器

加減算器のほうも乗算器と同様に非正規化数を扱うことができる必要がある。また、演算は長語で行うが、出力は短語に丸めることが可能でなければならない。このため、以下のような制御線がある。

NORMALA	ポート A の入力を正規化数とみなす
NORMALB	ポート B の入力を正規化数とみなす
SIGNB	B の符号を反転する (減算)
ROUND	出力を 25 ビットに丸める
NORMALO	出力を正規化する

図 12 に加減算器の主要な回路を示す。非正規化数/正規化数の処理の回路、符号の取り扱いの回路等は現在のところ省略されている。

演算結果フラグ (後述の M レジスタに格納される) は、演算結果が正であることを示す (符号ビットを反転) ものとする。

5.4.2.4 固定小数点 ALU

これは普通の固定小数点 ALU であり、以下の演算をサポートする。語長は ALU は面倒なので 72 ビットのみ。36 ビット 2 演算とかいう SSE とか MMX みたいなことは考えない。

- 2 の補数表示での加減算
- ビット毎の論理演算 AND OR NOT XOR。 ビット毎でないものは命令レベルではサポートしない。
- シフト演算。論理シフト、バレルシフトの両方。符号拡張もする。

オペレーションは

命令コード	オペレーション
0x00	A+B

```
0x01        A-B
0x02        A+1
0x03        A-1
0x04        not A
0x05        A and B
0x06        A or B
0x07        A xor B
0x08        max (A,B)
0x09        min (A,B)
0x0A        A
0x0B        B
0x0C        A lshiftl B
0x0D        A lshiftr B
0x0E        A bshiftl B
0x0F        A bshiftr B
0x10        logical not A
0x1F        immidiate
```

これらは 5 ビットの命令コードで指定する。l[b]shiftl[r] は、論理 (バレル) 左 (右) シフトである。算術演算、論理シフトについては符号ありと符号なしの両方の演算を可能にする。符号あり演算は 2 の補数である。

Max, min は符号なし、ありの両方でちゃんと動くようにする。符号なしのモードでは正の浮動小数点数も同様に扱うことができる (正規化されていれば)。但し、符号が変わるものについては扱えない。符号がある浮動小数点数の選択はフラグと条件付き実行で行う。 Max, min をもたせる理由は、 RRN のところでは必要であり、 RRN の ALU と PE の ALU を同じものにしたいからである。

logical not は A 入力が 0 であれば 1 (長語境界と短語境界の両方の LSB を 1)、0 でなければ 0 にする命令である。これにより、ビット演算ではない論理式評価を可能にする。単純に論理積を取るためには多少面倒な操作が必要だが、まあ、できないよりはましであろう。

immidiate (即値) 命令は、入力ポートはつかわないで命令フィールドの一部のビット値からそのまま短語の値を生成して出力する。この時に、レジスタ書き込みフィールドはそのまま使うので、データを以下のフィールドに分解して入れる

レジスタ読み出し制御フィールド	16 ビット
FMUL 制御	14 ビット
FADDSUB	10 ビット
合計	40 ビット

FADDSUB 制御フィールドの下 4 ビットは使わない。

整数乗算命令はもたない。浮動小数点の仮数の処理には定数倍等が必要だが、浮動小数点演算ユニットを使って処理する。

メモリ、レジスタから短語がくる時には MSB 側に入る。LSB は 0 で埋める必要がある。バレルシフトはこの時短語に対して正しい結果になることを要求しない。

演算結果フラグについては、 演算が A-B, A+B, A+1, A-1 の時、 結果が非負ならフラグがセットされる。それ以外の時は結果が all 0 ならフラグがセットされるということにする。符号なしモードでの減算では、 MSB の上へのキャリを見て結果の符号を判断する必要がある。

IALU の制御コードは以下のようになる。

```
IALUOP[0:4] ALU 自体の命令コード
UNSIGNED    符号なし演算を指定 (1 の時に)
```

5.4.2.5 条件付き命令

SIMD プロセッサであり局所的な (PE によって違う動作になる) ジャンプ命令は存在できないので、条件付き実行命令によってジャンプでできることをする必要がある。

基本的に全命令は条件付きにすることが可能であり、条件付きの時にはマスクレジスタ (M レジスタ) の値によって結果をストアするかどうかを判定する。

M レジスタをセットする命令が必要である。これは、通常なら

- 浮動小数点大小比較命令
- 浮動小数点ゼロ比較命令
- 固定小数点大小比較命令
- 固定小数点ゼロ比較命令

の 4 種はあるところ。浮動小数点ゼロ比較はあんまり使わないし整数演算で実現可能なので実装しない。大小比較は普通に s1 が大きかった時にフラグをセットする。これは専用の比較命令を持つわけではなく、 fadd, fsub 命令の結果が非負であった時にフラグをセットする。

固定小数点比較は前節で既に述べた。

条件のネストに対応するためには M レジスタが複数必要なので、これは複数もたせて命令のオペランドで制御 M レジスタと結果 M レジスタを指定する。ベースとして、 M0 レジスタは常に 1 (真) を返すとしておく。

M レジスタを読み出す命令は特に設けない。あらかじめ 0 クリアしたところに M レジスタで制御して 1 をしまうことで読み出しに変えることができるからである。書き込みはもちろん条件判定の結果を書くので問題ない。

M レジスタ関係の制御線は以下のようになる

```
IMR[0:2] 書き込み制御を  M レジスタのどれから取るか。 0 は ALL 1
OMR[0:2] 演算器の出力を M レジスタのどれに書くか。 0 は 書かない
IFSEL    FADD と IALU のどちらのフラグ出力を取るか
```

5.4.2.6 T レジスタ

T レジスタの動作は単純である。演算中は

- アドレスの初期値は 0 が与えられ、ベクトル演算中は必ず 1 づつ増える。
- 読み出しは常に行われる。
- 書き込みは命令フラグによる。

T レジスタは短語・長語を区別しないで、短語命令であれば MSB 側だけが使われる。

ブロックとしての T レジスタは以下の入出力を持つ

入力:

```
DIN[0:71]  72 ビットデータ入力
WADDR[0:3] 書き込みアドレス
MASK       書き込みマスク。0 なら書かない
WE         ライトイネーブル。
RADR[0:3]  読み出しアドレス
SHORTSTOP  書き込み (入力) データをそのまま出力に回す。
```

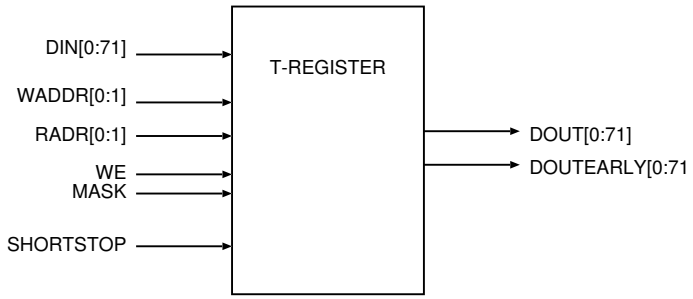


Figure 13: T レジスタの入出力定義

出力

```
DOUT[0:71]        72 ビットデータ出力
DOUTEARLY[0:71]   72 ビット出力。メモリアドレス用
```

図 13 に T レジスタの基本的回路を示す。
ベクトル動作時のアドレス生成は以下のような感じ。

- スタートでカウンタ 0 クリア、WE をロード。
- 書き込みアドレス、WE は必要なだけ遅延させる
- 命令のベクトル長を過ぎたらカウンタ止まる。WE も上げる。

つまり、ベクトル長 4 の時の write 動作はこんな感じ

```
plase     0     1     2     3     0     1     2     3
clock  ___===___===___===___===___===___===___===___===___===___===___===
load   =\__/====================\__/====================
upc    =0=><=0==><=1==><=2==><=3==><=0==><=1==><=2==><=3==>
dnc    =0=><=3==><=2==><=1==><=0==><=3==><=2==><=1==><=0==>
wein   =\__/================================================
weout  ====\____________________/==========================
```

つまり、ベクトル長 2 とか 3 ではこうなる:

```
plase     0     1     2     3     0     1     2     3
clock  ___===___===___===___===___===___===___===___===___===___===___===
load   =\__/====================\__/====================
upc    =0=><=0==><=1==><=0==><=0==><=0==><=1==><=2==><=0==>
dnc    =0=><=1==><=0==><=0==><=0==><=2==><=1==><=0==><=0==>
wein   =\__/====================\__/====================
weout  ====\___________/============\__________________/====
```

これだけ。回路図は図 14 みたいなもの。load が clear より優先になる動作をするものとする。この図では書き込みアドレス用のパイプラインレジスタは省略している。

図 15 に T レジスタの基本的回路構成を示す。

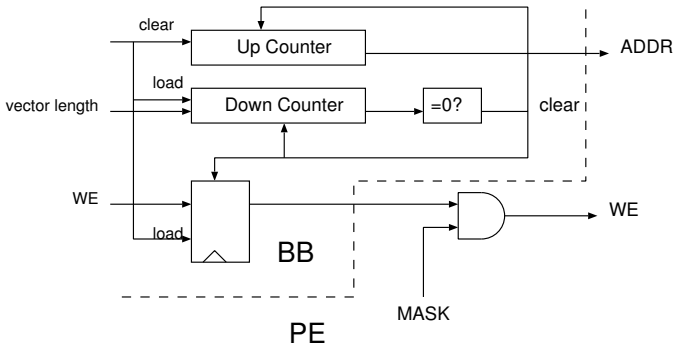


Figure 14: T レジスタのアドレス生成回路。

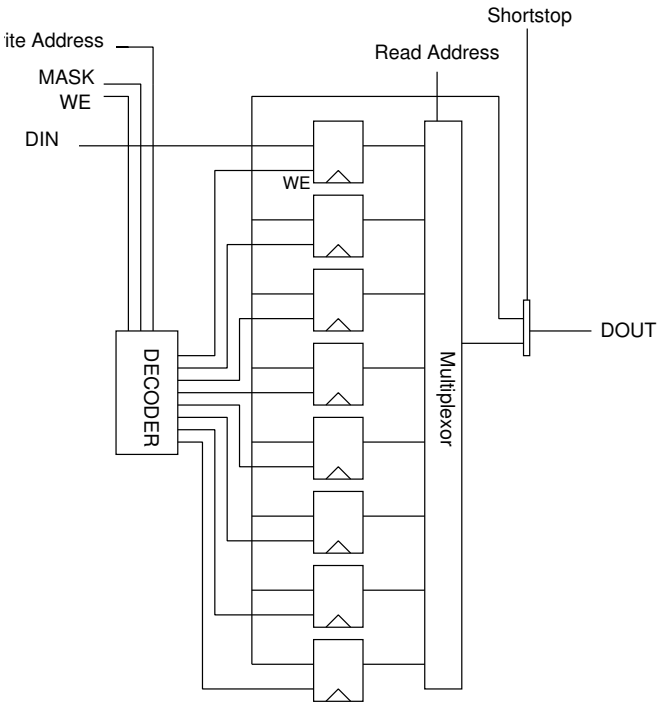


Figure 15: T レジスタの基本的回路構成

5.4.2.7 レジスタファイル (GRF)

レジスタファイルは長語 1 語を短語 2 語としてアクセス可能にする。

読み出しポートの制御は

- アドレス
- 短語か長語か

だけ。アドレス生成回路としては、

- ロード可能なカウンタ
- カウンタの動作を制御するレジスタ。クロック毎の増分を 0,1,2 と変える。このカウンタの LSB 以外がレジスタを構成するメモリのアドレスになる。このレジスタはカウンタと同時にロードされる。
- 短語の時に出力の選択を行うマルチプレクサの制御。カウンタ LSB を使う。

と、これだけ。

書き込みポートも本質的には同じである。違いは、指定された M レジスタの値を見て write enable を出すのと、短語の時に書き込み制御がちょっとややこしいことくらいである。

IBM のライブラリを使う場合、 GRF はマルチポートメモリではなくマルチポートレジスタファイルを使うほうがサイズが小さい。この時、読み出しは非同期である (出力がクロックをまたずに、アドレスが変わった時からの遅延でてくる) ことに注意する必要がある。つまり、ローカルメモリに比べて遅延が少ない。SING ではこの事実を有効に使うわけではなく、パイプラインレジスタをいれてレジスタとメモリの動作タイミングを合わせる。これは T レジスタと同様である。

IBM のレジスタファイルやメモリライブラリでは、ビット毎に書き込み制御ができるので、72 ビット幅のレジスタファイルを一つもたせておけば短語、長語の両方に対応できる。

1 つのポートの読み出し制御には論理的には以下の信号が必要になる。

- 長語アドレス
- 短語か長語か
- 短語の時には上か下か。

短語の時には下のワードは 0 固定にするとして、

- 長語アドレス。レジスタファイルのアドレスになる
- 上か下か (アドレスの LSB) で上位・下位を選ぶ

だけで十分である。

制御回路の概要を図 16 に示す。短語の時に下を 0 固定にする回路は省略されているので注意。

図でわかるように、SELA、SELB だけが制御されるがこれには単純にアドレスカウンタの LSB が入る。

書き込みは

- 長語アドレス
- 短語か長語か
- 短語の時には上か下か。

の全てが必要である。

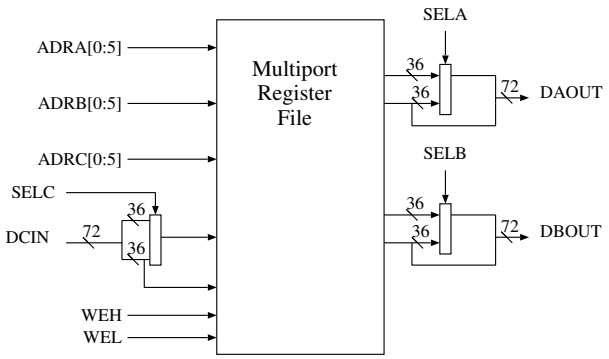


Figure 16: GRF の回路構成。カウンタ等は別に示す。

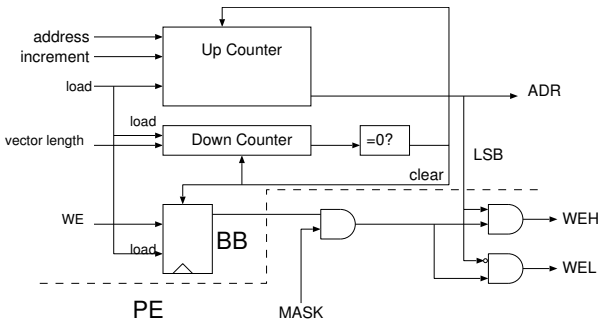


Figure 17: GRF のアドレス生成回路。アドレス生成の Up カウンタは同じものを 3 個持つ。また、ループカウントのダウンカウンタは T レジスタ制御と共通である。

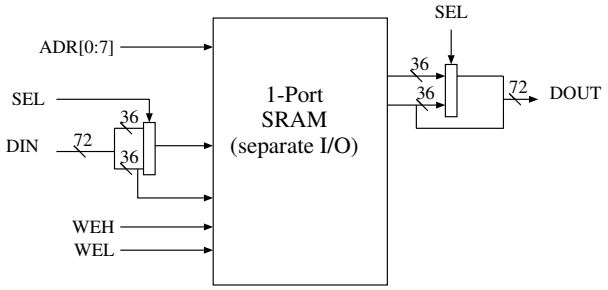


Figure 18: LM の回路構成。カウンタ等は別に示す。

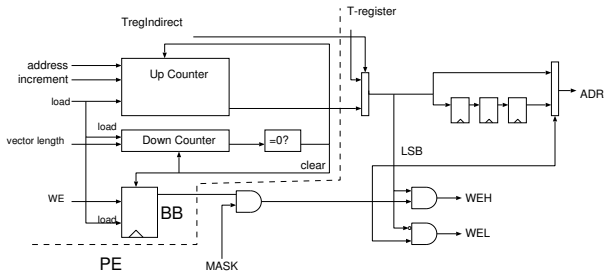


Figure 19: LM のアドレス生成回路。 GRF のアドレス生成回路と、アドレスカウンタが 1 つしかないこと以外は同一である。

5.4.2.8 ローカルメモリのアドレスモード

ローカルメモリのアドレスモードには、直接アドレスと T レジスタ間接がある。T レジスタ間接は、ベクトル演算時に T レジスタの値がそのままローカルメモリアドレスとなるものであり、アドレス修飾 (オフセットを足したりとかする) の機能はもたない。

実行ステージ的には、間接アドレスによる読み出しでは T レジスタの出力選択 (ここはマルチプレクサだけ、つまり組合せ論理でパイプライン遅延は入らないものとする) の分だけゲート遅延が増えるので、それがクリティカルパスにならないような配慮がレイアウト時に必要になる。

5.4.2.9 ローカルメモリの動作

ローカルメモリは 1 ポートなので、書き込み中に読み出しは行えない。これは、つまり、ある命令が LM への書き込みであった時にそれの次の 2 命令は LM からの読出しを行うことができないということである。これは 1 ポートメモリを使う限り避け難い。

5.4.2.10 ローカルメモリの回路構成

図 18 と図 19 にローカルメモリの回路構成を示す。

制御回路等は GRF のそれとほとんど同じである。一つ大きく違うのは、読み出し、書き込みで同じアドレスラインを使うので書き込みか読み出しかでアドレスの遅延が異なることである。このため、書き込みの時にはそれに対応する遅延したアドレスを選択するような回路にする。 T レジスタや GRF でも書き込みアドレスは同様に遅延させる必要がある (回路図からは省略されているので注意)。

5.4.2.11 アドレス生成ユニットの物理的実現

同じ BB 内の PE は、 T レジスタ間接アクセスの場合を除いてレジスタ・メモリにアクセスするアドレスは同じである。また、アドレス計算にフィードバック等があるわけではない。従って、アドレス生成は BB 毎に一つのユニットで行ない、生成したアドレスや書き込み制御信号を各 PE のメモリユニットに直接配るものとする。ここの配線は短いがファンアウトは大きいので、場合によってはパイプラインレジスタを挿入する。

5.4.2.12 PE の実行ステージ

少なくともローカルメモリに対しては T レジスタの中身をアドレスにできる必要がある。で、 read/write 両方必要である。

タイミング的には write のほうが簡単で、遅延を合わせるだけ。

LM の read 動作はそもそもこんな感じのはず

```
           0         1         2
clock _____=====_____=====_____=====_____=====_____=====
addr  -<===A0===><===A1===><===A2===><===A3===>><===A4===>
dout  --------------------<===D0===><===D1===><===D2===>
```

read アドレスに入れるには、動作ステージのどこに入れるか考える必要がある。

ステージ 0 : 命令が来て、アドレスカウンタとかにロードされる。
この時には T レジスタのアドレスカウンタもロードされる。
ステージ 1 : アドレスカウンタから出力でる。
T レジスタの出力セレクタもデータがいってる。

というわけで、遅延が増えるのはここだけ。このセレクタと、アドレスカウンタ自体の出力とのセレクタのゲート 4-8 段分くらい。結構馬鹿にならないけど、メモリアドレスはどうせこの後でラッチするからあんまり関係ない。

ステージ 2 : メモリアドレスラッチされる。
ステージ 3 : メモリ出力でる。
ステージ 4 : 演算器 1 段目
ステージ 5 : 演算器 2 段目
ステージ 6 : 演算器 3 段目
ステージ 7 : メモリ書き込み

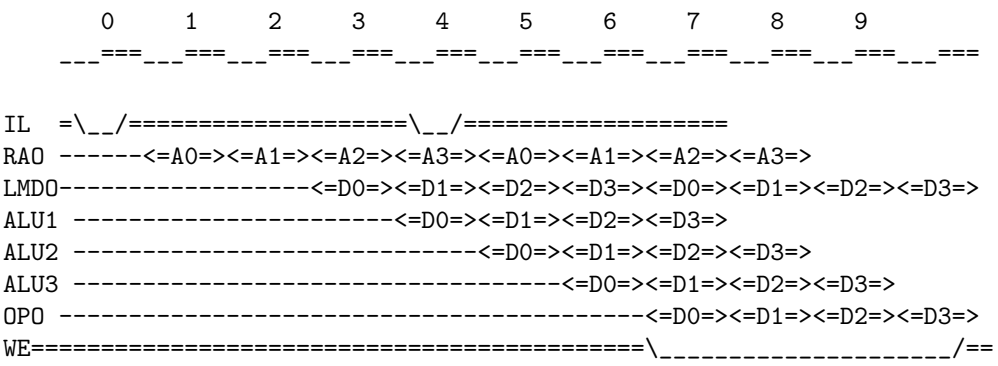
となり、パイプラインは 8 段になる。演算器の中間ステージこんなにいらない気もするけど、入力前のマルチプレクサとか出力のマルチプレクサとかあるのでこれくらいとったほうが後が楽になると思われる。つまり、

ステージ 4 : 入力マルチプレクサ出力をラッチ (演算器から見ると入力ラッチ)
ステージ 5 : 演算器内部ラッチ 1 段目
ステージ 6 : 演算器内部ラッチ 2 段目
ステージ 7 : 演算器出力ラッチ (出力マルチプレクサの入力)

という感じになることを一応想定する。

```
命令:              IL
read アドレス出力: RAO
LM データ出力:     LMDO
演算器 データ出力: OPO
```

とすると、



問題点:

T レジスタの使い回しがこれでは (このままでは) できない。つまり、前の命令の結果が T レジスタに入るのは 7 (4n+3) サイクル目なのに、T レジスタの出力は 2 (4n+2) サイクル目には必要で間に合わない。

もっとも、これは T レジスタに対してシャドウレジスタとして演算器の出力レジスタをわりあてて、これらを明示的に指定すればよい。これは、T レジスタのところで shortsop と呼んだものである。このパスの時に遅延が大きすぎないかという問題がある。ゲート段数はしれてるけど配線がちょっと長いかも。これはレイアウト時に検討が必要かもしれない。

なお、T レジスタの出力をアドレスとして使う時にはデータとして使う時より 2 サイクル早く結果が必要なので、シャドウがあっても全然間に合わない。これは、「アドレスとしては直前の演算結果は使わ (え) ない」という「仕様」であるということにする。

5.5 結果縮約ネットワーク (RRN)

RRN は基本的に PE の浮動小数点ユニットと ALU と同じものである縮約ユニット (RU) からなるツリー構造を持つ。動作モードとして、縮約モードとランダムアクセスモードの 2 つが必要である。縮約モードでは 2 つの入力に対して命令語で指定された演算がなされる。 ランダムアクセスモードでは、指定した BB からの入力がそのまま伝わる。これは、入力に input valid (IV)、出力に output valid (OV) の線をつけて、 OV は 2 つの IV の OR を取ることにすればよい。入力のどちらかの IV がネゲートされていたら、出力はもう片方の入力をそのまま出す。

各放送ブロックには簡単なシーケンサがあり、放送メモリから指定されたアドレスのデータを読んで自分につながった RU に出力する。縮約モードでない時にはコマンドパラメータの BB 番号と自分の BBID を比較し、一致していた時にだけ OV を出す。縮約モードの時にはブロックマスクレジスタの値でマスクされてなければ OV を出す。

一つの RU は以下の入力を持つ。

- クロック。ベースクロック
- クロックイネーブル。分周クロックから作られる。

これらは一つだけ。以下は上の RU から伝わるので 2 組ある。

- 命令語。RU の動作を指定する。
- IV (Input valid)

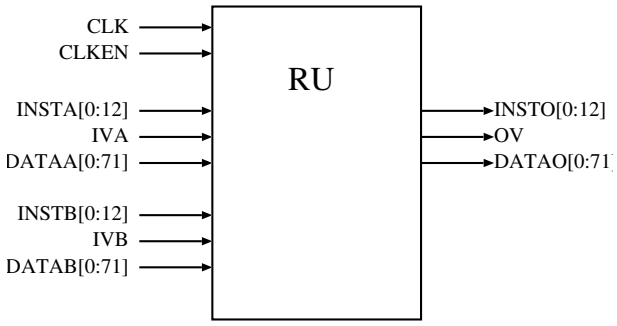


Figure 20: RRN の演算ユニット一つの入出力。

- データ 72 ビット。

命令語はもしも 2 つが違ったら (ハードウェア的に起きえないはずだけど)、 ID の小さいほうを優先としておく。
出力はクロック以外の入力と同じく

- 命令語。RU の動作を指定する。
- OV (Output valid)
- データ 72 ビット。

である。 IV/OV はストローブの役割も果たす。図 20 に RU 一つの入出力信号を示す。
命令語の詳細は次に定める。

5.5.1 RU 命令語

各 RU は以下の命令語を受け取り、それに従って動作する。

FSEL	出力選択。1 なら FADDSUB の出力を取る。 0 なら IALU
NORMALA	ポート A の入力を正規化数とみなす
NORMALB	ポート B の入力を正規化数とみなす
SIGNB	B の符号を反転する (減算)
ROUND	出力を 25 ビットに丸める
NORMALO	出力を正規化する
IALUOP[0:4]	IALU の命令コード
UNSIGNED	符号なし演算を指定 (1 の時に)

5.5.2 ネットワークトポロジ

RRN はバイナリツリーの構造を持つことにする。ツリー構造は BB の ID の LSB の側から順番に加算される、つまり図 ref(fig:rrn-id) に示すような構成になる。

ここで、 RRN はツリーの各ノードにはいるが、BB 番号の小さいほう (あるいは、小さいほうに由来するほう) が RRN のポート A、大きいほうがポート B にはいるものとする。

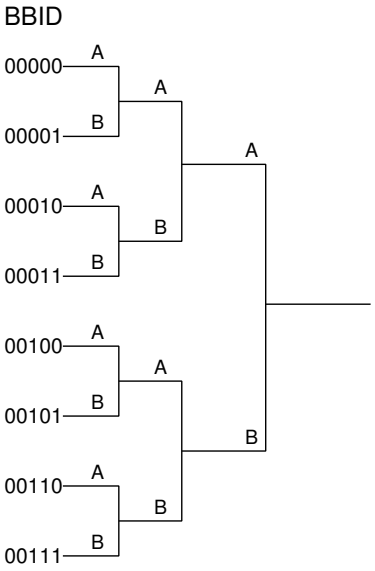


Figure 21: RRN ツリーの構成。BB 8 個分のみ示す。

Figure 22: DOU の入出力。

5.6 出力データストリーム処理ユニット (DOU)

DOU の機能は単純であり、72(パリティつけて 81) ビットの入力データ、分周クロック、OV 信号からベースクロックで動くシリアライズした出力を作るだけである。

出力のデータ線数は 21、クロックの本数は 3 本とする。パリティは付けるが ECC はなしですます。図 22 に DOU の入出力を示す。

なお、真っ当な ODP ポートの他に、ステータスレジスタとして 8 ビットの内部レジスタを設けて、出力の LSB 8 ビットをストアできるようにする。このステータスレジスタはチップの出力ピンを直接ドライブし、LED (そのままはつかないと思うが) などで診断情報の表示に使う。結果をステータスレジスタに出すか出力ポートに出すかの指定は

ODPOE	ODP から出力する
SREGEN	ステータスレジスタに書き込む

というフィールドを RRN 命令語に追加することで行う。これらは RU にいく必要はないので、必要な段数パイプラインレジスタで待ったあと直接 DOU に供給される。

6 PE 演算の詳細

前章では、 SING チップの各部について基本的な動作を記述した。ここでは特に演算器について、動作の詳細を解説する。

6.1 浮動小数点データフォーマット

浮動小数点データフォーマットは以下の通りとする。

- 短語 36 ビット、長語 72 ビット
- 仮数 24/60 ビット unbiased forced-1 rounding
- 指数 11 ビット
- 符号ビット

指数形式は IEEE-754 と同じ。 短語 で 0x3FF000000 が 1 になるもの。つまり、 IEEE-754 の規定のように、正規化された仮数を $1 \leq x < 2$ の範囲の数と解釈する場合には、指数のオフセットが 0x3FF ということになる。

但し、通常の IEEE 計算と違って、非正規化数というデータ型を持つものとする。通常の正規化数では、仮数は hidden bit を持つものと暗黙に解釈される。つまり、(16 進で)

```
指数 3FF
仮数 000000
```

という表現は、仮数の MSB の「上」に 1 があるもの、つまり、仮数が

```
1000000
```

であるものと解釈される。仮数の小数点はちょうどこの MSB とその上の hidden bit のあいだにあるので、

```
(1)000000 -> 1.0
(1)800000 -> 1.5
(1)FFFFFF -> 1.999999....
```

と解釈される。で、指数は固定小数点であり、符号ビットは 0 が正、1 が負である。例えば 0x400000000 は 2.0 0x7FF000000 は -1.0 になるわけである。

非正規化数と解釈する時には、この hidden bit を補わないで解釈する。つまり、

```
000000 -> 0.0
800000 -> 0.5
FFFFFF -> 0.999999....
```

と、そのまま解釈する。この解釈では、指数がいくつであっても仮数が 000000 なら表現する値は 0 である。正規化数か非正規化数かの指定は演算器のオペコードで行ない、データ自体は自分がどちらであるかを示すタグのようなものはもたない。

この非正規化数の基本的な用途は積算である。つまり、積算の最終結果の指数くらいの値を初めから足し込まれる変数にセットしておくことで、積算時の丸め誤差が積算順序に依存しないようにするのである。

乗算器は通常非正規化数を扱う必要はないが、倍精度乗算のための部分積の計算では非正規化数が入力に現れる。仮数の途中を切り出してくるからである。このため、乗算器でも非正規化数を入出力ともにサポートする。通常の正規化数同士の乗算では、結果のシフトは 1 または 0 ビットビットだが、非正規化数の乗算ではもしも結果を正規化しようとするとフルサイズのシフタが必要になる。これは回路規模に響くので、入力のどちらかが非正規化数ならば出力も自動的に非正規数になるものとする。入力の両方が正規化数の場合に結果を非正規化数にすることは可能である。

0、無限大等は以下のように表現する。

0 は指数 0

無限大は指数 all 1 (短語で 0x7FFFFFFF, 0xFFFFFFFF)

無限大と 0 の掛け算の結果は 0 とする。

6.2 乗算器

浮動小数点乗算器では以下の手順で乗算を行う。

6.2.1 仮数の切り出し

短語の場合には無関係だが、長語の場合には部分積を求めるために仮数を切り出す。これは、以下のように行う。

60 bit の仮数が以下の形を持っていたとする。以下、数字一つが 2 進数 1 ビットを表す。0/1 でないのは位置を示しているからである。

```
012345678911234567892123456789312345678941234567895123456789
```

シフトしない場合には、これはとりあえずこのままである。但し、入力が正規化表現の時にはここで上に 1 を補う。そうでなければ 0 を補う。つまり、シフトしない時にはこのステージの出力は必ず 61 ビットとなる。

```
x012345678911234567892123456789312345678941234567895123456789
```

こんなふう。それから 25 ビットまたは 50 ビットのシフトを行う。つまり 25 ビットシフトなら

```
x012345678911234567892123456789312345678941234567895123456789
456789312345678941234567895123456789000000000000000000000000
                                    |                         |
                                    |------25 個 0------------|
```

50 ビットシフトなら

```
x012345678911234567892123456789312345678941234567895123456789
951234567890000000000000000000000000000000000000000000000000
           |                                                  |
           |---------------------- 50 個 0--------------------|
```

というふうになる。仮数を持ち上げた分指数は引く必要がある。これは単純に 25 または 50 を引き、負になったら 0 にする。

制御コードは以下のもの

```
SHIFT50A    ポート A の仮数を 50 ビット上にシフト
SHIFT25B    ポート B の仮数を 25 ビット上にシフト
SHIFT50B    ポート B の仮数を 50 ビット上にシフト
NORMALA     ポート A の入力を正規化数とみなす
NORMALB     ポート B の入力を正規化数とみなす
```

6.2.2 仮数の丸め

パラメータで指定があれば、仮数を 25 ビットで丸める。前のステージからの 61 ビット出力の 25 ビット目

```
x0123456789112345678921234567893123456789412345678951234567 89
                              |
                              ここ
```

を、

1. もしもそれ以下のビットが全て 0 ならばそのまま
2. それ以外の場合、つまりそれ以下のビットが 1 つでも 0 でなければ、この

位置を 1 にする。
制御コードは以下のもの

ROUNDA	ポート A の仮数を (シフト後)25 ビットに丸める
ROUND50A	ポート A の仮数を (シフト後)50 ビットに丸める
ROUNDB	ポート B の仮数を (シフト後)25 ビットに丸める

6.2.3 仮数乗算

仮数の乗算を行う。入力は 25 ビットで、結果は 50 ビットになる。

6.2.4 仮数の後処理

6.2.4.1 正規化する場合

正規化する場合には、MSB が 0 なら 1 ビット左論理シフトして、指数は結果の指数から 1 引く。

6.2.4.2 正規しない場合

正規化しない場合には、無条件に 1 ビット右論理シフトして、指数は結果の指数に 1 を足す。この時に、 LSB は切り捨てられるのではなく、下に延びる。出力の仮数は 60 ビットなのでこれが可能である。

6.2.4.3 丸め

結果を丸める場合には、上から 24 ビットに対して、入力の丸めと同様に、

1. もしもそれ以下のビットが全て 0 ならばそのまま
2. それ以外の場合、つまりそれ以下のビットが 1 つでも 0 でなければ、この

位置を 1 にする。
という処理を行う。これは、上のシフトを行った後で行う。制御コードは以下のもの

ROUND	出力を丸める
NORMALO	出力を正規化する

6.2.5 指数の処理

指数は入力の指数を足して、 0x3FE を引く。上の仮数による調整を行ったあとで、オーバーフローしていたら 0x7ff、アンダーフローしていたら 0 にする。但し、入力の指数のどちらかが 0x7ff の時にはそのままとし、そうでなくてどちらかが 0 であれば 0 とする。また、仮数が 0 になった場合も指数を 0 とする。ただし、この場合も入力の指数のどちらかが 0x7ff の時にはそのまま 0x7ff である。

6.3 加減算器

浮動小数点加減算は以下のような手順で行う。

２つの入力を in1, in2 と書くことにする。in1 の指数、符号と仮数をそれぞれ exp1, sign1, mantissa1 と書き、in2 についても同様とする。

最初に、 減算であれば sign2 を反転しておく。また、入力が正規化数であれば乗算の時と同様に仮数 MSB の上に 1 を補う。そうでなければ 0 を補う。

次に指数が小さいほうの仮数をシフトする。シフタは 1 つなので、実際には、

1. まず exp1 と exp2 のどちらが大きいか判定し、
2. 指数が大きいほうと小さいほうをの仮数を入れ替え
3. 指数が小さいほうの仮数を指数の差だけ右論理シフトする

という操作になる。仮数を m2[0:60] (m2[60] が MSB) というふうに書いた時に、シフト量 s = exp1 - exp2 を使って、シフト後の m2 は以下のように書ける

```
m2new[i]=0  if i > 60 - s
m2new[i]=m2[i+s] if 60-s >=i>0
m2new[0]=m2[s] if m2[j]=0 for all j<s
m2new[0]=1     if m2[j]=1 for any j<s
```

つまり、LSB の値は、シフトアウトしたものが全て 0 でなければ強制的に 1 にされる。

これを mantissa1 に加える。 sign1 と sign2 が違えば減算になる。この結果は桁上がりも含めて 63 ビットの符号つき整数である。これを、まず符号と 62 ビットの絶対値に分ける。結果が正であれば、出力の符号が入力 1(スワップされていれば 2) の符号、負であればそれをを反転したものになる。

プライオリティエンコーダで最上位の 1 の位置を決める。これが LSB なら 0、MSB なら 61 が出力になるものとする。この値を p とする。

出力を正規化しないというパラメータが指定されている場合、 p が 59 以下ならば実際にシフトの必要はない。この場合には仮数のシフトは発生せず (シフト量 s は 0 であり)、出力の指数は exp1 のままになる。但し、 p が 60 または 61 の場合、非正規化数になるように結果のシフトをする必要がある。この場合、結果を右論理シフトする。シフト量 s は p-59 であり、その分指数を増やす。

出力を正規化する場合、シフト量は p-60 (正なら右論理シフト、負なら左論理シフト) になる。

以上をまとめると、シフタの動作は入力を min[0:60], 出力を mout[0:60] シフト量を s(正の時右シフト) として以下のようになる

```
mout[i]=0        if i>60-s
mout[i]=min[i+s] if 60-s >=i>0
mout[i]=0        if 0< i< -s
mout[0]=min[s]   if m2[j]=0 for all 0<=j<s
mout[0]=1       if m2[j]=1 for any 0<=j<s
```

指数は常に exp1 に s を加えることになる。この結果、オーバーフローが起これば指数を 0x7ff に、アンダーフローが起これば指数を 0 にする。

さらに、結果を 25 ビットに丸めるというパラメータが指定されていれば、mout[0:35] が全て 0 でなければ mout[36] を 1 にする。同時に mout[0:35] は 0 にする。

最終的な結果の仮数は mout[0:59] になる。これからわかるように mout[60] は使われないので、シフタはこのビットを出力する必要はない。

上の演算手順から、以下のような制御線があることになる。

NORMALA	ポート A の入力を正規化数とみなす
NORMALB	ポート B の入力を正規化数とみなす
SIGNB	B の符号を反転する（減算）
ROUND	出力を 25 ビットに丸める
NORMALO	出力を正規化する

6.4 整数 ALU

これは演算手順というほどのものはない。命令コードは一応以下の割り当てにする。

opcode	operation
0x0	A+B
0x1	A-B
0x2	A+1
0x3	A-1
0x4	not A
0x5	A and B
0x6	A or B
0x7	A xor B
0x8	max (A,B)
0x9	min (A,B)
0xA	A
0xB	B
0xC	A lshiftl B
0xD	A lshiftr B
0xE	A bshiftl B
0xF	A bshiftr B
0x10	logical not A
0x1F	immidiate

なお、命令コード 4,5,6,7 の論理演算は全てビット毎である。

この他、 UNSIGNED 指定ビットがある。これは以下のオペレーションに影響する。

opcode	operation
0x8	max (A,B)
0x9	min (A,B)
0xC	A lshiftr B

すなわち、 大小比較は、 SIGNED の場合には 2 の補数として行う。UNSIGNED ならば絶対値の比較になる。実装は、73 ビットに符号拡張した減算を行って、MSB を見るのが簡単である。SIGNED の時には符号拡張し、UNSIGNED の時にはしない。

右論理シフトの場合には符号拡張が起こるので、シフト前に最上位ビットが 1 で unsigned でなかった時ににはシフトして空いたところには 1 が埋められる。それ以外の場合は論理シフトでは 0 が入る。

以下の命令では演算結果が正の時 (UNSIGNED の時には 73 ビットに拡張した MSB を見て) にフラグビットがセットされる。これらの命令では、演算結果には UNSIGNED かどうかは影響しないが、フラグには影響することになる。

opcode	operation
0x0	A+B
0x1	A-B
0x2	A+1
0x3	A-1

また、それ以外の命令では、結果が all 0 の時にフラグがセットされる。

これらは UNSIGNED ビットに影響されない。

6.4.1 シフト命令の動作詳細

バレルシフト、論理シフトのどちらも、シフト量が語長を超える場合の動作を規定しておく必要がある。

右シフトと左シフトで別命令なので、シフト量は常に絶対値として解釈される。シフト量の最大値は 71 なので、シフト量指定の 8 ビットより上は単に無視される。72 から 127 までのシフト量が指定された場合の動作はバレルシフトと論理シフトで異なり、

- バレルシフトの場合は 72 を引いたあまりだけシフトする
- 論理シフトの場合は 72 シフトする。つまり、 all 0 (右シフトで符号で 1 になった時には all 1) になる。

となる。

7 命令セット定義

命令は以下の種類に分かれる。

- PE 命令
- IDP/BM 命令
- RRN 命令

PE 命令は各 PE に放送される。これは ISU を通る。それ以外の命令は全て IDP のデータパケットとして投入される。IDP 命令は IDP を通って BM やそれ以外の PE 固有でないチップ内レジスタにアクセスする命令、 RRN 命令は RRN の動作を指定する命令である。

以下、 PE 命令、それ以外の順番で命令フォーマットと動作を記述する。

7.1 PE 命令

PE 命令は基本的に水平型マイクロコードであり、 PE の全てのユニットについてその動作を記述する。具体的には、命令自体の記述としてループ長 (4 を越えてはいけない) があり、それに各ユニットの制御コードが続く。

なお、ベクトル命令である関係上、制御コードの遅延については BB 内の制御ロジックで処理するものとする。つまり、投入される PE 命令は水平型マイクロコードといっても論理的に同じ命令のフィールドが記述され、メモリ読み出し、演算器制御、メモリ書き込みといった一連の動作に必要な遅延は BB のなかでハードウェア的に実現される。

なお、ループ長は、レジスタ、メモリ (マスクレジスタ含む) への書き込みを抑制するためにのみ用いられる。

PE 命令は以下のフィールドからなる。これらが MSB からつめて入る。以下のサブセクションで順番に詳細を述べる。最後に形式をまとめる。

- ループ長
- M レジスタ制御
- T レジスタ制御
- レジスタファイル制御
- ローカルメモリ制御
- FMUL 制御
- FADDSUB 制御
- IALU 制御
- BM 制御

7.1.1 ループ長 (LL)

LL フィールドはベクトル命令の繰り返し数を指定する。今回の実装ではフィールド長は 3 ビットで、最大値は 4 とする。それ以上を指定した時の動作は保証しない。0 の時には実効的の NOP (no operation) となる。

7.1.2 M レジスタ制御 (MRC)

MRC フィールドは以下の指定を行う。

```
IMR[0:2] 書き込み制御を  M レジスタのどれから取るか。 0 は ALL 1
OMR[0:2] 演算器の出力を M レジスタのどれに書くか。 0 は 書かない
IFSEL    FADD と IALU のどちらのフラグ出力を取るか
```

少しわかりにくいが、M レジスタの出力セレクトが IMR である。これによる書き込み指定は、M レジスタ自身を含めて全てのストレージ書き込みに適用される (他の指定との AND) になる。

合計 7 ビット。

7.1.3 T レジスタ制御

T レジスタの制御コードは以下の通り

```
WRITE       書き込むかどうか。 1 なら書き込む。
SHORTSTOP   書き込み (入力) データをそのまま出力に回す。
ISEL[0:1]   入力セレクト
            00: FMUL
            01: FADDSUB/IALU
            10: BM
```

合計 4 ビット。

MRC の項で述べたように、マスクレジスタは MRC で指定されたものが適用される。また、アドレス初期値は読み出し、書き込みともに必ず 0 なので指定の必要はない。

なお、 shorstop 動作の時にはマスクレジスタ指定は無視され、マスクレジスタの指定にかかわらず前の命令での T レジスタ入力ポートへの入力データがそのまま出力ポートにでる。

7.1.4 アドレスに関する規約

レジスタファイルのアドレス指定は全て短語アドレスで行う。長語アクセスの時には LSB は無視される (non-aligned access はサポートしない)。これはローカルメモリ、放送メモリでも同様である。

7.1.5 レジスタファイル制御 (RFC)

レジスタファイルの制御コードは以下の通り

WRITE	書き込むかどうか。 1 なら書き込む。
ISEL[0:1]	入力セレクト
	00: FMUL
	01: FADDSUB/IALU
	10: BM
WADR[0:5]	書き込みアドレス初期値
WADRI	書き込みアドレスをインクリメントする (1)/しない
WWL	書き込みワード長 1:長語、 0:短語
RADRA[0:5]	読み出し A アドレス初期値
RADRIA	読み出し A アドレスをインクリメントする (1)/しない
RWLA	読み出し A ワード長 1:長語、 0:短語
RADRB[0:5]	読み出し B アドレス初期値
RADRIB	読み出し B アドレスをインクリメントする (1)/しない
RWLB	読み出し B ワード長 1:長語、 0:短語

合計 27 ビット。

7.1.6 ローカルメモリ制御 (LMC)

ローカルメモリの制御コードは以下の通り

WRITE	書き込むかどうか。 1 なら書き込む。
ISEL[0:1]	入力セレクト
	00: FMUL
	01: FADDSUB/IALU
	10: BM
ADR[0:8]	アドレス初期値
ADRI[0:3]	アドレスインクリメントの値
TREGADR	アドレスを TREG から取る
WL	ワード長 1:長語、 0:短語

書き込みがある命令のあとの 2 命令は、読み出したデータは保証されない。ADRI は 1 の時に連続アクセスになる、つまり長語ならアドレス増分は ADRI で指定した値の 2 倍になるものとする。

これは 18 ビット。

7.1.7 FMUL 制御 (FMC)

浮動小数点乗算器の制御コードは以下の通り。

```
SHIFT50A    ポート A の仮数を 50 ビット上にシフト
ROUND50A    ポート A の仮数を (シフト後)50 ビットに丸める
ROUNDA      ポート A の仮数を (シフト後)25 ビットに丸める
NORMALA     ポート A の入力を正規化数とみなす
SHIFT25B    ポート B の仮数を 25 ビット上にシフト
SHIFT50B    ポート B の仮数を 50 ビット上にシフト
ROUNDB      ポート B の仮数を (シフト後)25 ビットに丸める
NORMALB     ポート B の入力を正規化数とみなす
ROUND       出力を丸める
NORMALO     出力を正規化する
ISELA[0:1] A ポート入力セレクト
           00: レジスタファイル A ポート
           01: レジスタファイル B ポート
           10: T レジスタ
           11: ローカルメモリ
ISELB[0:1] B ポート入力セレクト
           00: レジスタファイル A ポート
           01: レジスタファイル B ポート
           10: T レジスタ
           11: ローカルメモリ
```

14 ビット。

7.1.8 FADDSUB 制御

浮動小数点加減算器の制御コードは以下の通り。

```
NORMALA     ポート A の入力を正規化数とみなす
NORMALB     ポート B の入力を正規化数とみなす
SIGNB       B の符号を反転する (減算)
ROUND       出力を 25 ビットに丸める
NORMALO     出力を正規化する
ISELA[0:2]  A ポート入力セレクト
            000: レジスタファイル A ポート
            001: レジスタファイル B ポート
            010: T レジスタ
            011: ローカルメモリ
            100: PE 番号 (固定)
            101: BM 番号 (固定)
            110: 乗算器フィードバック
            111: 未定義
ISELB[0:1]  B ポート入力セレクト
            00: レジスタファイル A ポート
            01: レジスタファイル B ポート
            10: T レジスタ
            11: ローカルメモリ
```

10 ビット。

FADDSUB はフラグ出力を持つ。これは、演算結果が正である時にセットされる (符号ビットを反転したものがそのまま使われる)。

ISELA の乗算器フィードバックは直前の乗算器の演算結果がそのまま入る。

フィードバックパスの場合には、マスクやベクトル長とは無関係に必ず前の命令での乗算器の 4 サイクル分の出力結果がそのまま加算器の入力となる。

7.1.9 IALU 制御

整数 ALU の制御コードは以下の通り

IALUOP[0:4]	ALU 自体の命令コード
UNSIGNED	符号なし演算を指定 (1 の時に)

6 ビット。

フラグ生成は以下のルールに従う

演算が A-B, A+B, A+1, A-1 の時: 結果が正ならフラグがセットされる
それ以外の時: 結果が all 0 ならフラグがセットされる

7.1.10 FADDSUB/ALU 出力セレクト

FADDSUB と IALU の出力はどちらか一方だけがメモリ等のマルチプレクサに入るという仕様なので、まず手前で選ぶ必要がる。これをこのフィールドで指定する。

FSEL	出力選択。1 なら FADDSUB の出力を取る。 0 なら IALU

7.1.11 BM 制御 (BMC)

BM の制御コードは以下の通り

WRITE	書き込むかどうか。 1 なら書き込む。
ADR[0:10]	アドレス初期値
PEADR[0:4]	BM 書き込みの時にアクセスする PE 番号
WL	ワード長 1:長語、 0:短語

書き込みがある命令のあとの 2 命令は、読み出しデータは保証されない。

これは 18 ビット。

7.2 命令コードの長さ

とりあえず全てまとめると以下の通り。

ループ長	3
M レジスタ制御	7
T レジスタ制御	4

```
レジスタファイル制御 27
ローカルメモリ制御   18
FMUL 制御             14
FADDSUB 制御          10
IALU 制御              6
FADDSUB/IALU 選択      1
BM 制御               18
合計                 108
```

これを全てこのまま送り込むか、あまり同時に使いそうにない機能をまとめて命令ビットを減らすかという選択が必要だが、今回面倒なのでそのまま送り込む。

7.3 IDP/BM 命令

IDP 命令 (データパケット) は以下の形式を取る:

先頭ワード (72 ビットワード) はパラメータ指定等であり、その後に先頭ワードによって指定された語数のデータが続く。

先頭ワードの形式は以下の通り

```
LEN[0:7]    データ語数
ADDR[0:12] BM の先頭アドレス
BBN[0:4]    BB 番号の指定
BBNM[0:4]   番号のマスク
SEQ[0]      シーケンシャルライトフラグ
```

合計 32 ビット。

これを MSB からつめる。 下に 40 ビットくらいあまるけど未使用。

LEN は長語単位の語数指定であり、 0 は 256 語と解釈される。

BBN は実際に書く BB の番号を指定する。放送やマルチキャストを可能にするために、 BBNM で指定されたビット位置だけを比較し、BBNM が 0 であるビット位置は無視する。

SEQ は同一データを放送 (またはマルチキャスト) するか、 BB 毎に違うデータを送るかを制御する。 SEQ=0 の時は放送である。 SEQ=1 の時には LEN で指定された語数のデータをアクティブな (下の放送ブロックマスクレジスタがセットされていない) BB の数だけ送り、送られたものが順番に BB0 から書かれる。この時には BBN, BBNM は無視される。つまり、データパケットは以下のようになる

```
0      パケットヘッダ
1      BB0 のアドレス ADDR のデータ
2      BB0:ADDR+1
....
LEN    BB0:ADDR+LEN-1
LEN+1 BB1:ADDR
....
```

となり、パケットの総語数は (有効な BB の数) × LEN + 1 となる。

メモリは 1024 ワードであるので、

```
0x000-0x7ff     放送メモリ
```

```
0x800-0xfff     放送ブロック制御レジスタ
0x1000-0x17ff   RRN 制御コマンドレジスタ
0x1800-0x1fff   未定義
```

というメモリマップにしてアドレスの上位ビットでメモリとそれ以外を区別する。放送ブロック制御レジスタは以下の通り

```
0x800   ブロックマスクレジスタ (BMR)
0x802   エラーステータスクリア。データ長は 1 だが無視
```

ブロックマスクレジスタは LSB からの各ビットが BB0 から BBn (n は実装依存。今回は 16) の状態を制御する。対応するビットが 1 にセットされると、縮約モードの時には RRN 命令が無視され、出力の OV がでない。また、IDP SEQ モードの時に書き込みがスキップされる。

7.4 RRN 命令

RRN 命令は、 IDP データパケットのアドレス 0x1000 への書き込みとして指定される。命令のフィールドは以下の通り。

```
ADR[0:12]    アドレス初期値
N   [0:7]    語数
BBADR[0:4]   アクセスする BB 番号
REDUC        縮約モードかどうか。 1 なら縮約、 0 なら PE 選択。
WL           ワード長 1:長語、 0:短語
FSEL         出力選択。1 なら FADDSUB の出力を取る。 0 なら IALU
NORMALA      ポート A の入力を正規化数とみなす
NORMALB      ポート B の入力を正規化数とみなす
SIGNB        B の符号を反転する (減算)
ROUND        出力を 25 ビットに丸める
NORMALO      出力を正規化する
IALUOP[0:4]  ALU 自体の命令コード
UNSIGNED     符号なし演算を指定 (1 の時に)
ODPOE        ODP から出力する
SREGEN       ステータスレジスタに書き込む
```

これを MSB から詰め込む。

メモリは 1024 長語 (2048 短語) であるので、

```
0x000-0x7ff     放送メモリ
0x800-0xfff     放送ブロックステータスレジスタ
0x1000-0x1fff   未定義
```

となる。ステータスレジスタは

```
0x800 ブロックマスクレジスタ
0x802 エラーステータスレジスタ
      ビット位置   0: IDP パリティエラー
                   1: ISU パリティエラー
```

もうちょっとなんかいるかもしれない。

N は語数指定であり、 0 は 256 語と解釈される。

なお、非縮約モードが指定された時にはブロックマスクレジスタは無視される。

8 チップ I/O 定義

8.1 クロック入力

クロック CLK はベースクロック。

8.2 分周クロック出力

ISU 入力で 4 分周したクロック信号が入るが、これがそのままでるもの。 RRN の分周クロックはこれを基準にする。

8.3 リセット入力

リセット RST は、

- IDP のデータを受けるシーケンサ
- RRN のデータを出すシーケンサ
- エラーステータスレジスタ

をクリアする。さらに、入力シンクロナイザを再同期させる。

それ以外はそのまま。

8.4 IDP モード指定

IDP モード指定 IDPMODE0 IDPMODE1

IDPMODE0	IDPMODE1	意味
0	0	9 ビット毎にクロック
0	1	18 ビット毎にクロック (72-79 は 8 ビットにクロック)
1	0	36 ビット毎にクロック (72-79 は 8 ビットにクロック)
1	1	未使用

8.5 エラーステータス出力

チップのエラーステータス出力は、基本的は各 BB のエラーステータスの OR をとったものと、入力同期エラーの 2 種類がでる。定義されているものは

ISPERR	ISP の入力パリティエラー
IDPERR	IDP の入力パリティエラー
PHASERRR	ISP/IDP の入力フェーズエラー

8.6 ISP

```
ISCLK[0:3]     入力ソースクロック
ISDATA[0:31]   入力データ
ISPPHASE[0:3]  フェーズ信号
```

ISDATA を 4 クロック分くっつけた後のものを IS[0:127] と呼ぶ。これは 9 ビット毎に 1 ビットパリティを持つ。パリティは 10 ビット目、つまり IS[10n] になる。

4 サイクルのデータは、下からくるものとする。つまり、

```
Cycle 0  IS[0:31]
Cycle 1  IS[32:63]
Cycle 2  IS[64:95]
Cycle 3  IS[96:127]
```

の順番でくる。

8.7 IDP

```
IDCLK[0:8]    入力ソースクロック
IDDATA[0:79]  入力データ
IDPPHASE[0:8]  フェーズ信号
DS            データストローブ
DSCLK         データストローブクロック
```

8.8 ODP

```
ODCLK[0:2]    出力ソースクロック
ODDATA[0:20]  出力データ
```

8.9 チップ状態表示

チップが何をしているかが目でわかるものをいろいろつける。

- 入力
- 実行
- 出力

が基本なので、それぞれについて、「作動中」を示す出力ピンをつける。

```
IDPSTATUS IDP からデータパケットが来ている
ISPSTATUS ISP から NOP でない命令が来ている
DOPSTATUS DOP から データを出している
```

これは、非同期動作でかまわない。システムクロックと同期したりする必要は全くない。
この他に、各 PE のレジスタを直接表示する仕掛けをつける。具体的には、

```
SREGEN=1
```

で指定された出力命令で出力ポートに出る 72 ビットのうち LSB 8 ビットをそのまま表示する。

`PEOUT[0:7]` ステータス表示

これらは、非同期動作でかまわない。システムクロックと同期したりする必要は全くない。

9 メモ

9.1 統計情報等の収集について

ソフトウェアの効率評価等のためには、実行効率をハードウェア的に測定するような仕掛けが欲しいような気もする。具体的には、例えば以下のようなものである。

- 実行時 MASK が真であった回数
- 実行時 NOP でなかった回数

こういうのをハードウェアで常時モニタするのはそれほど面倒なわけではない。それ用のカウンタを準備して、そこから読めるようなパスをつけるだけだからである。
しかし、今回は、少しでも面倒なことは、それが

- (比較的単純なコードの) 性能向上に貢献するわけではない
- ソフトウェアでエミュレーションできる

場合にはつけないことにしたいという基本方針 (なんてものがいつのまに出来たのか知らないけど) に従って、つけない。
ソフトウェアエミュレーションは、要するに LM か GRF にこれらのカウンタをとっておいて、コンパイラかアセンブラが命令毎、あるいはマスクレジスタが変わる毎に必要な数を加算すればよい。

9.2 IALU 即値

2004/8/30 現在での命令セットでは、 IALU に即値をオペランドにとる命令がない。 +1 命令とシフト命令はあるから、 log N ステップで任意の数を置けるからこれでいいかな？繰り返し使う数はレジスタに置くしかないし。

9.3 IDP ストローブをどうするか？

2005/5/26

- FPGA からなんとか出す
- 止める

- 4 サイクルに 1 回 4 ビットとする
- 2 サイクルに 1 回 2 ビットとする

とりあえず 2 サイクル毎ということで。

10 名前

当初使っていた VPM という名前は今一つであるというのは確か。今まで提案された中でもっともインパクトのある名前は

SING, for SING is Not GRAPE

であるのはなかなか疑いをいれないところ。とりあえずこれを開発コードネーム (HARP とか相当) としては採用する。

同工異曲なものとしては King, Ming, Ring, Wing とかが可能だけど、まあ、このなかでは Sing が美しいですね。うむ。

もうちょっとましな名前募集中。

11 メモ

11.1 IDP 同期とかについて

11.1.1 2005/7/7

名村さんから:

ISP_CHAN_PHASE_ERROR 動作について

rx_logic が 4 サイクルに 1 度 data をだしてきます。このときに、データの出た次のサイクルに 1 パルス PHASE_EDGE という信号が出力されます。

各 channel からこの信号がでますので、これらを OR した信号をつくりこれが 1 から 0 になるタイミングをさがします。このタイミングがすべての channel から出てきたデータをうけとるタイミングになります。このときまでに、一度は rx_logic から PHASE_EDGE という信号が 1 になる必要があるので、これがない場合に Phase 間のずれが大きくなったと判断して、この信号が 1 になります。

12 略語定義

ALU Arithmetic and Logic Unit 算術・論理演算器

BM: broadcast memory 放送メモリ

HT HyperTransport

IDP Input Data Port

ISP Instruction Stream Port

ODP Output Data Port