

修士論文

題目

高効率動き検出に対応するSIMD  
併用型汎用データパス構成法の  
研究

指導教員

近藤 利夫 教授

平成 26 年度

三重大学大学院 工学研究科 情報工学専攻  
計算機アーキテクチャ研究室

渡邊 敬太 (413M530)

## 内容梗概

2016年にスーパーハイビジョンの試験放送が予定されるなど、近年動画画像の高精細化が進んでいる。この高精細化に伴い、動画画像符号化の圧縮率向上に対する要求が高まっている。2013年には、この動画画像符号化効率の要求を満たすため、ITU-TによってH.264/AVCの2倍の圧縮率を備えたH.265/HEVCが標準化されているが、処理の複雑性の増加により、H.264と比較し符号化速度は低減している。動画画像符号化処理の高速化は、処理の大半を占めている動き探索処理の高速化が必要不可欠である。ARMやx86などの従来の汎用プロセッサにおいて、動き探索処理の高速化に向けたデータパスおよび専用命令の改良は行われているものの、高速化の程度は十分であるとはいえない。そこで、この問題を解決するために当研究室では、従来のSIMDデータパスの高速化に必要な最低限の機能を追加した動き探索処理用の汎用志向のデータパスを提案しその有効性を評価した。

付加機能は大きく分けて、SAD演算に用いられる総和用5段ステージ、一時的にSAD値を保持するFIFO、最小SAD値を抽出する比較用レジスタ、コード量を削減するレジスタローテーション機能の4つである。

提案データパスの動き探索に対する性能を評価するために、ダイヤモンドパタンとSUCパタン、遠隔点探索でのSAD演算に要する実行ステップを用いる。また、論理回路設計と、それに対する論理合成を行うことで、ハードウェア規模を算出し、約3.20%のハードウェア規模の増加で、従来手法と比べて近傍点の探索で約3.83～4.83倍、遠隔点の探索で平均4.38倍高速化できることを示した。

# Abstract

In 2016, the test broadcast of Ultra High Definition Television is scheduled, and the high definition of video image has made progress in recent years. By this high definition, the demand for the compression ratio improvement of video encoding is increasing. In 2013, the video coding standard H.265 being twice as efficient as H.264 was recommended by ITU-T for satisfying the high coding efficiency of video. However, its encoding speed becomes at least several times slower than H.264 by increasing of computation complexity. Improving the motion estimation (ME) is the most important key point for speeding up, because ME processing occupies most of the encode processing. Although conventional general-purpose processors such as ARM and x86 processor have enhanced data paths and instructions for ME to reduce the processing time, these are insufficient. Hence, in order to solve this problem, in this paper, the author proposed a general-purpose-oriented data path that adds minimum function for speeding up to the previous SIMD data path for the ME and evaluated its effectivity.

There are four types of additional function: The sum total five-stage stage for SAD operation, the FIFO for temporarily storing the SAD, the comparison register for extracting the minimum SAD and the register rotation feature for reducing the code quantity.

In order to compare the performance for motion estimation of the proposed data path, we evaluate the number of steps for SAD operation in diamond, SUC pattern and non-dense pattern consisting of scattered points. In addition, I designed the circuit structure and calculated the hardware scale by logic synthesis. As a result, we speed up about 3.83-4.83 times in square pattern and SUC pattern and about 4.88 times in non-dense pattern consisting of scattered points compared to previous method with increasing the hardware scale about 3.20%.

# 目次

1	まえがき	1
1.1	研究背景	1
1.2	研究目的	1
1.3	各章の概要	2
2	動き検出処理と高速化の実態	4
2.1	動き検出の概要	4
2.2	H.265/HEVC の特徴	5
2.3	高速処理法	6
2.3.1	ダイヤモンドサーチ	6
2.3.2	スクエアサーチ	6
2.3.3	SUCサーチ	7
3	従来の SIMD 拡張命令とその問題点	8
3.1	動き探索命令の概要	8
3.2	MPSADBW 命令の問題	9
3.3	動き探索処理の高速化に対する要求条件	9
4	SIMD 併用型汎用データパスの提案	10
4.1	SAD 演算を含む動き探索処理全体の高速化	10
4.2	近傍探索点間の参照画像の再利用率向上	10
4.3	参照画像読み込み時に対するオーバーヘッドの低減	10
4.4	提案データパス構成	11
4.4.1	静的複数命令発行構成	11
4.4.2	非整列ブロックアクセス対応構成	12
4.4.3	再利用容易化構成	14
4.5	提案命令セット構成	15
4.5.1	SHUFFLE 命令	15
4.5.2	BB_SAD 命令	16
4.5.3	SAD_SUM 命令	16
4.5.4	MINIMUM 命令	17
4.5.5	MOVE 命令	17
4.5.6	addi 命令	18
4.5.7	lw 命令	18

4.5.8	sw 命令	19
5	提案データパスの設計	20
5.1	回路構成	20
5.1.1	レジスタ構成	22
5.1.2	アドレス修飾機構	22
5.1.3	シャッフルユニット	23
5.1.4	総和用 5 段ステージ	23
5.1.5	FIFO レジスタ	24
5.1.6	最小値抽出用比較レジスタ	25
5.1.7	レジスタローテーション	26
5.2	制御線	27
6	比較用専用志向データパスの構成	28
6.1	参照画像格納用専用レジスタ	28
6.2	MSPSADBW 命令	28
6.3	探索点抽出ユニット	28
6.4	専用志向データパスの特徴・まとめ	29
7	評価	30
8	あとがき	36
	謝辞	37
	参考文献	37
A	シミュレーションによる動作検証	39
A.1	ModelSim の起動	39
A.2	波形表示する信号線の選択	40
A.3	シミュレーション	40
B	CPU の動作テスト	41
C	MIPS 命令セットの詳細	42
C.1	add 命令	42
C.2	sub 命令	42
C.3	and 命令	42

C.4	or 命令	43
C.5	xor 命令	43
C.6	sllv 命令	43
C.7	srlv 命令	44
C.8	beq 命令	44
C.9	bne 命令	44

## 図 目 次

2.1	SAD 値計算例 . . . . .	4
2.2	H.265 で用いられる可変ブロックサイズ . . . . .	5
2.3	一般的な動き探索パタン . . . . .	7
2.4	<i>SUC</i> パタン . . . . .	8
3.5	MPSADBW 命令の概要 . . . . .	9
4.6	提案データパスで用いる 64 ビット 2 語命令の構成 . . . . .	12
4.7	非整列ブロックアクセス . . . . .	13
4.8	シャッフルユニットによるデータの並び替え . . . . .	14
4.9	レジスタローテーション . . . . .	15
5.10	回路全体図 . . . . .	21
5.11	提案データパスにおけるレジスタ構成 . . . . .	22
5.12	シャッフルユニット . . . . .	23
5.13	総和用 5 段ステージ . . . . .	24
5.14	FIFO レジスタ . . . . .	25
5.15	最小値抽出用比較レジスタ . . . . .	26
5.16	rrb デコーダ . . . . .	27
6.17	探索点抽出ユニット . . . . .	29
7.18	処理ルーチン . . . . .	31
7.19	動き探索の走査順序 . . . . .	31
7.20	高速化率 . . . . .	34
1.21	ModelSim ボタン . . . . .	39
1.22	ModelSim ウィンドウ . . . . .	39
1.23	波形表示ボタン . . . . .	40
1.24	シミュレーション開始ボタン . . . . .	40

## 表 目 次

5.1	命令と制御線の対応関係 . . . . .	27
7.2	ダイヤモンドパターンを用いた動き探索処理の実行総ステップ数 . . . . .	32
7.3	<i>SUC</i> パターンを用いた動き探索処理の実行総ステップ数 . . . . .	32
7.4	遠隔の探索点 1 点に対する動き探索処理の実行総ステップ数 . . . . .	33
7.5	参照画像データの総転送量 (Byte) . . . . .	35
7.6	ハードウェア規模 . . . . .	35



# 1 まえがき

## 1.1 研究背景

近年，ハイビジョン (HD : High Definition) の 16 倍の画素数を有するスーパーハイビジョン (UHD : Ultra High Definition) の試験放送が 2016 年に予定されるなど，動画像の高精細化が進む中，動画像の通信や蓄積に不可欠となっている動画像符号化の圧縮率向上の要求が高まっている．これに応えるために，現在に至るまで MPEG と VCEG によって，動画像圧縮規格が順次策定されおり，2013 年には H.264/AVC[1] 以来 10 年ぶりの新規格となる H.265/HEVC[2] が標準化された．この H.265 は現在広く利用されている H.264 と比較し，2 倍もの圧縮性能の改善に成功しているものの，可変ブロックサイズの多様化など複雑性の増大により符号化の処理量が大幅に増加してしまっている．このため H.264 と同様の符号化速度を得ようとする処理能力の大幅向上が要求される．動画像符号化処理を重要なアプリケーションの 1 つとする PC においても当然この要求に応えられる性能を備える必要がある．特に演算量の大半を占める動き探索の高速処理機能は必要不可欠である．しかし，PC に組み込まれている x86 プロセッサには， $16 \times 16$  以上のブロック用の PSADBW 命令と水平方向に連続する 8 探索点用の MPSADBW 命令の 2 種類の SAD 演算命令を備えているだけで，現在主流となっている高効率の追跡型の動き探索への適合性は低く，スーパーハイビジョンクラスの高速化要求に応えられるレベルには達していない．

## 1.2 研究目的

このような状況の中，近年，動き探索処理に適した演算器を組み込む高性能ハードウェアの設計の研究が継続的に行われてきたものの，汎用プロセッサ向けの研究は数少ない [3]．これに対し，当研究室では汎用プロセッサへの適用を目指して 2 通りの構成を提案してきた．その一方は動き探索に特化した専用的な演算器を組み込むことで，探索パターンに応じて，参照画像格納用専用レジスタに格納された複数の探索点に対してライン単位で SAD 演算を並列実行することができる専用志向のデータパスである．もう一方はこれまでの汎用プロセッサが備える SIMD 処理機構を最大限活かす構成をとるブロック単位の SAD 演算の並列実行に対応する汎用志向のデータパスである．専用志向のデータパスは近傍点での探

索に対し非常に有効であるものの、これまでの汎用のデータパスとハードウェアを共用できる割合が小さく汎用プロセッサの性能/コスト比を低下させる可能性がある。これに対し、汎用志向のデータパスでは、探索点が密な領域の高速化率は専用志向に比べ高めにくいことが予想させるものの、これまでの SIMD 処理機構にわずかなハードウェアを追加するだけで構成できる可能性がある上に、追加ハードウェアの一部は汎用的な処理能力向上に役立つ可能性がある。

そこで、本論文では、汎用プロセッサの動き検出能力の大幅向上を目指し、従来の汎用プロセッサとの適合性の高い高効率動き検出対応の SIMD 併用型汎用データパス構成法の確立とその命令セット制定を目指す。具体的には、従来の汎用データパスに対し最低限のハードウェア規模の追加で構成される SIMD 併用型の汎用志向データパス構成とその命令セットを提案し、論理設計まで行うことで、わずかなハードウェア規模増で専用志向データパスと遜色のない高速化率が達成されることを明らかにする。

### 1.3 各章の概要

本論文では、動き探索処理の高速化に向けて、従来の汎用プロセッサとの適合性の高い高効率動き検出対応の SIMD 併用型汎用データパスを提案する。はじめに動き探索処理の高速化に対する課題・要求条件について述べ、次にその要求条件を満たすデータパスの構成法および命令セットの提案・設計を行う。その後、性能評価を行い、実験結果から提案するデータパスの有効性を示す。

以下に各章の概要を述べる。

第2章「動き検出処理と高速化の実態」では、動き検出の概要と最新の動画圧縮規格 H.265 の特徴について述べる。また、各動き検出アルゴリズムの特徴についてそれぞれ述べる。

第3章「従来の SIMD 拡張命令とその問題点」では、SAD 演算に用いられる MPSADBW 命令の問題点を指摘し、提案データパスに求められる動き探索処理の高速化に対する要求条件を示す。

第4章「SIMD 併用型汎用データパスの提案」と第5章「提案データパスの設計」では、第3章で記された高速化に対する要求条件を満たすデータパスの構成法と命令セットを提案し、その回路構成について述べる。

第6章「比較用専用志向データパスの構成」では、比較対象用に用い

る当研究室で提案する専用志向データパスの構成法と SAD 演算命令として考案する MSPSADBW(Multiple Sparse point PSADBW) 命令について述べる．

第 7 章「評価」では，提案する高効率動き検出対応の SIMD 併用型汎用データパスの優位性を明らかにするため，動き探索処理に必要な実行ステップ数とデータの総転送量から高速化率とデータの再利用性を評価する．また，設計した提案データパスのハードウェア規模を論理合成によって見積もり，評価結果と合わせてその優位性を示す．

第 8 章「あとがき」では，本論文の内容を総括し，今後の課題を検討する．

## 2 動き検出処理と高速化の実態

### 2.1 動き検出の概要

動き探索処理は，符号化対象画像の位置やブロックサイズを変更しながら，参照画像とブロックマッチングを行うことで，最も類似度の高い領域を求め，対象物体の動き検出をする手法である．ブロックマッチングでは，符号化対象画像と参照画像の類似度を調べるために，差分絶対値和 (SAD) の演算が用いられる．SAD 値が小さければ小さいほど，2つのブロック間の類似度は大きく，完全に一致している場合には値は0になる．符号化対象画像ブロックを X，参照画像ブロックを Y とした  $4 \times 4$  のブロックサイズにおける SAD 値を求めるための計算例を図 2.1 に示す．

符号化対象画像ブロック X

123	47	88	86
124	34	71	88
122	33	80	76
103	45	74	52

参照画像ブロック Y

128	54	91	95
121	36	75	99
122	34	86	93
111	66	85	87

$|X - Y|$

差分値

5	7	3	9
3	2	4	11
0	1	6	17
8	21	11	25

$\Sigma |X - Y|$

133  
SAD値

画素値

図 2.1: SAD 値計算例

動き探索処理は符号化処理の大半を占めており，その高速化には，中核の SAD 演算処理の高速化が必要不可欠である．一方，SAD 演算を含む動画画像処理は，一般的に動画画像内の全ての画素データに対して同処理を行うことが多いため，一つの命令を同時に複数のデータに適用し処理する SIMD 演算により，高効率化・高速化が可能になる．

## 2.2 H.265/HEVC の特徴

現在に至るまで，様々な動画圧縮規格が策定されており，2013 年には H.265/HEVC が国際標準化された．ブロックマッチングでは，複数の可変ブロックサイズを用いることで，動画内の形状や動きに適したブロックを選択し，符号化効率を向上させることが可能である．H.265 ではこの可変ブロックサイズが多種多様になっており，H.264 の最大  $16 \times 16$  の 7 種類だったブロックサイズから，さらに図 2.2 に示すような非対称な形を含め最大  $64 \times 64$  の 24 種類のブロックサイズから選択することが可能となり，高い圧縮性能を実現している．しかしながら，ブロックサイズの多様化により処理の複雑性が増加し，処理が増える要因ともなっている．

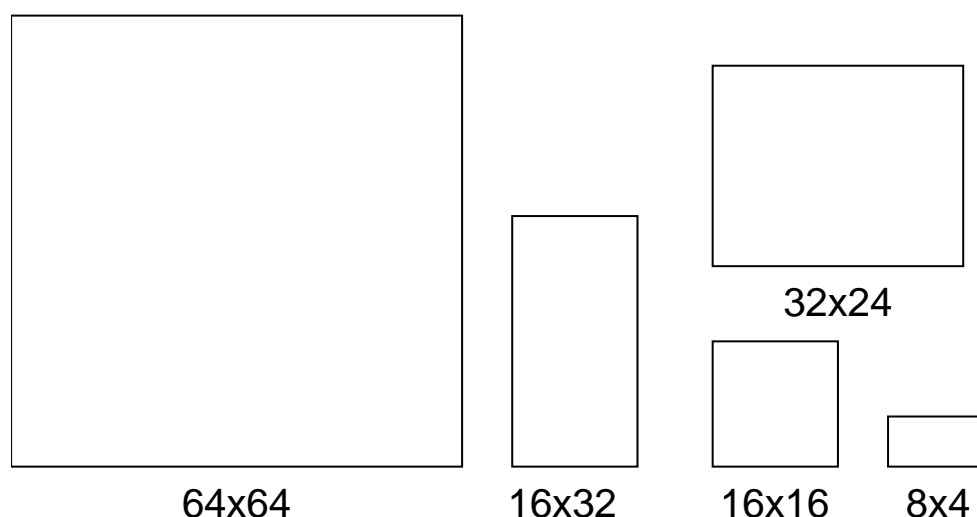


図 2.2: H.265 で用いられる可変ブロックサイズ

このため，多様なブロックサイズの全てに対して効率的に SAD 値が算出されることが必要になる．この要求に対し，各サイズのブロックを  $8 \times 4$  サイズのサブブロックに等分割し，32 並列で個別に求める個々の SAD 値を足し合わせることでブロック全体の SAD 値を求めるようにすれば， $4 \times 8$  以外のサイズの全てのサブブロックの SAD 値が算出できるようになるのに加え，大ブロックが包含する小ブロックの SAD 値も同時に求められるようになる．従って，各サイズのブロックを  $8 \times 4$  サイズのサブブロックに分け，それぞれを効率的に処理する SAD 演算機能の実現が必要といえる．

## 2.3 高速処理法

動き検出アルゴリズムは，並列化は容易であるものの処理量が非常に大きなハードウェア向き探索法と，並列化は容易ではないものの動き探索の効率化には非常に有効なソフトウェア向き探索法の2通りに分けることができる．ハードウェア向き探索法の代表としてフルサーチが挙げられる．フルサーチは，一定の走査範囲を左上から右下まで1画素単位でずらしながら探索する手法であり，単純なアルゴリズムでハードウェアでの並列処理が容易であるものの，膨大な量の演算が必要である．そのため，現在，ソフトウェア向きの動き探索法としては，高効率な追跡型の動き探索法が広く利用されている．追跡型の動き探索は，探索パターン（一回分の探索範囲）を必要に応じて移動させながら探索を繰り返すことで，最も類似度の高いブロックを検出しようとする探索法である．しかし，次の探索点を逐次的に決定せざるを得ないため，ハードウェアでの高効率な並列処理は容易でない問題がある．この追跡型の探索法を利用した代表的なソフトウェア処理向きの動き検出アルゴリズムに，H.264 参照ソフトウェア JM に搭載されている UMHexagonSearch[4] や，EPZS，HEVC 参照ソフトウェアに搭載されている TZsearch[5] などがある．これらのアルゴリズムは複数の追跡型動き探索処理を組み合わせることで，探索点数の大幅な低減による高速化を実現している．

以下にこれらの核となっている追跡型動き探索処理を示す．

### 2.3.1 ダイヤモンドサーチ

ダイヤモンドサーチ [6] は探索中心と上下左右の4個のブロックのブロックマッチングを行う探索手法である．また，ダイヤモンドサーチには図 2.2(a) に示す探索中心 (画素 0) とその周囲 4 点 (画素 1) を探索するスモールダイヤモンドパターン [7] と，探索点の距離を 2 の累乗で拡大していく拡大型ダイヤモンドパターンがあり，TZsearch など で用いられている．

### 2.3.2 スクエアサーチ

スクエアサーチは探索中心とその周囲 8 個のブロックのブロックマッチングを行う探索手法である．スクエアサーチで用いられるスクエアパターンを図 2.2(b) に示す．スクエアパターンは，ダイヤモンドパターンと比較し

データの再利用面で優れており，データ読み込み時のオーバーヘッドを低減することができる．

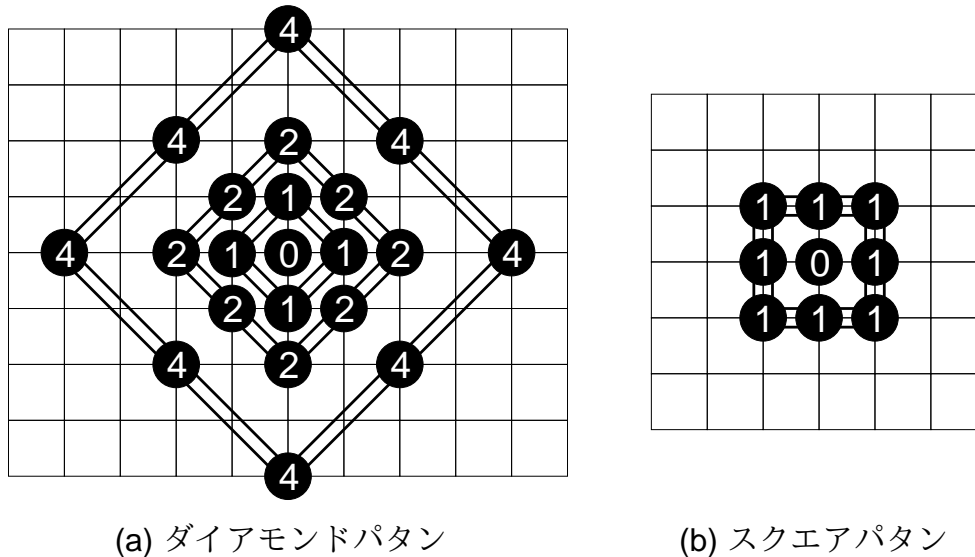


図 2.3: 一般的な動き探索パターン

### 2.3.3 *SUC* サーチ

*SUC*(Small Unsymmetric Cross) サーチは，当研究室で提案している拡大型ダイヤモンドパターンに代わる動き検出アルゴリズムである．拡大型ダイヤモンドパターンなどの高精度な動き検出はその探索範囲の広さから探索点が疎らでデータの再利用性が低く，並列化も容易ではない．また，スクエアパターンは探索範囲が狭く探索点が少ないものの，探索精度が低くローカルミニマムに陥る可能性がある．これらの問題に対し，図 2.4 に示す *SUC* サーチは *SUC* パターンを用いることで，探索精度の維持とデータの再利用性の両立を実現したものである．*SUC* パターンは，一般的な動画像では垂直方向の動きと比較し水平方向の動きが多い特徴を利用し，斜め方向の 4 点を省いた拡張ダイヤモンドパターンに対し，水平方向の  $\pm 4$  の点を加えた 11 点を探索点とした固定探索パターンである．

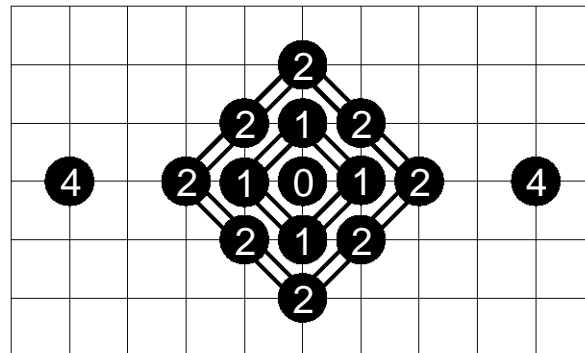


図 2.4: SUC パタン

### 3 従来の SIMD 拡張命令とその問題点

#### 3.1 動き探索命令の概要

現在，既存の汎用プロセッサでは SIMD(Single Instruction Multiple Data：単一命令/複数データ) 演算が用いられている．SIMD 演算は 1 命令で複数のデータに対して処理をおこなうことができる演算方式であり，画像処理や動き探索処理などの同一の演算を繰り返す処理の高速化に役立つ．

近年では，様々な汎用プロセッサが動き探索処理と相性の良い SIMD 命令セットとして SAD 演算用の命令を備えている．例えば ARM プロセッサは，128 ビット幅のベクトルレジスタと SAD 演算器およびその演算結果の累積を行う VABA 命令を備えている [8]．また，x86 プロセッサは拡張命令セットの一種である SIMD 命令セットとして，Advanced Vector Extensions(AVX)[9] を搭載している．これは SSE2 で追加された  $8 \times 1$  画素幅の符号化対象画像と  $8 \times 1$  画素幅の参照画像との SAD 演算を処理する PSADBW 命令と，SSE4.1 で追加された  $4 \times 1$  画素幅の符号化対象画像に対し， $4 \times 1$  画素幅の参照画像を水平方向に 1 画素ずつシフトした 8 座標の SAD 演算を並列に処理する MPSADBW 命令の両方を備えている．いずれも AVX，AVX2[10] として 128 ビット XMM レジスタに対する操作から，256 ビット幅の YMM レジスタへの操作へと拡張され，並列度を倍増させることで演算性能の向上が図られている．



## 3.2 MPSADBW 命令の問題

MPSADBW 命令は図 3.5 のように， $4 \times 1$  画素幅の参照画像を水平方向に 1 画素ずつシフトした画素 A ~ H の計 8 座標の SAD 演算を並列に実行し，計 32 並列の SAD 演算結果をレジスタ A ~ H に格納する命令である．

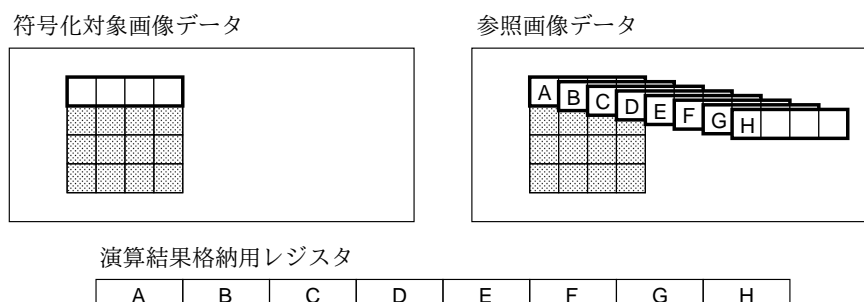


図 3.5: MPSADBW 命令の概要

この命令を用いることで動き探索処理に対し，PSADBW 命令と比較し数倍以上高速化を見込むことができる．しかしながら MPSADBW 命令では水平方向に連続する探索点に対してのみ高並列演算が可能である．このためソフトウェア志向のアルゴリズムに広く用いられている追跡探索用の探索パターンに対する適合性が低く，動き探索処理の高速化には対応できない．また，探索点ごとに参照画像データの読み込みを行う必要があるため，データ再利用性も低く，高効率の動き探索処理の実現は困難である．

## 3.3 動き探索処理の高速化に対する要求条件

汎用プロセッサにおいても動き探索処理の高速化に向けて，専用命令をサポートしているものの，高速化の程度は十分であるとはいえない．これは，高効率の動き検出実現に向けた以下の要求を満たせていないからである．

- SAD 演算を含む動き探索処理全体の高速化
- 水平，垂直の両方向に広がる探索パターンの構成点に対する SAD 演算の効率化
- 参照画像読み込み時に対するオーバーヘッドの低減

## 4 SIMD 併用型汎用データパスの提案

最小限のハードウェアコストで最大限の動き探索効率化の実現を目指し、SIMD 併用型の汎用的なデータパスに、最低限の動き探索専用ハードウェアを追加するだけで、効果的な動き探索専用の命令を用いることで高速・高効率な動き探索処理を実現する。

### 4.1 SAD 演算を含む動き探索処理全体の高速化

動き探索処理を高速化させる方法として、動き探索処理全体の並列実行度を高める必要がある。動き探索処理は SAD 演算とそのブロックサイズ分の集計のための加算が処理のほとんどを占めており、これらの処理は一般的に SIMD 演算を用いることで高速化が可能である。しかしながら、データパスの構成上 SAD 演算と加算処理は並列実行することができない。そこで、処理量が特に多いと考えられる SAD 演算を SIMD 処理用のベクトル演算パスで処理するのと並行して、ブロックサイズ分の集計のための加算などの一部の処理がスカラ処理用のスカラ演算パスで処理されるようにする。

### 4.2 近傍探索点間の参照画像の再利用率向上

SAD 演算命令 MPSADBW では、参照画像の再利用が、水平方向に連続する探索点間に限られていた。これは従来の汎用プロセッサがデータを 1 次元的に扱っており、ダイヤモンドパターンやスクエアパターンなどの 2 次元の探索パターンの探索に対応できないためである。そこで、提案するデータパスではブロック単位でのアクセスが可能となるようにキャッシュとレジスタファイルを構成することで、動き探索処理時のデータ処理の高効率化を実現する。

### 4.3 参照画像読み込み時に対するオーバーヘッドの低減

MPSADBW 命令を用いた SAD 演算は、探索点ごとに随時参照画像データの読み込みを行う必要があり、データの再利用性が低い問題がある。このために、重複部分が大半であるにも関わらず参照画像を繰り返し読み込まねばならず、その読み込みのオーバーヘッドが動き探索処理の高速

化の弊害となっている．そこで，一度レジスタに読み込んだデータを複数回再利用できる構成を採り，データ読み込みのオーバーヘッド低減をはかる．

## 4.4 提案データパス構成

本研究では，上記の要求を満たす以下のデータパス構成を提案する．

### 4.4.1 静的複数命令発行構成

提案データパスでは SIMD 演算とスカラ演算の並列処理を可能にするために複数命令発行方式 [11] を用いる．複数命令を発行するハードウェア構成の方法は大別して 2 つあり，ロード，ストア，演算などの命令複数個からなる命令語を構成し，それをコンパイラを用いることで依存関係を回避しつつ一度に実行する静的複数命令発行構成と，複数の命令を同時にフェッチし，同時に実行する動的複数命令発行構成に分けることができる．静的複数命令発行構成は一度発行した命令は並列性を分析する必要がなく，動的複数命令発行構成と比較しハードウェアコストの削減や動作の高速化を期待することができるものの，性能を引き出すためには，命令の依存関係をなくし命令発行率を向上させることができる複雑かつ高性能なコンパイル性能が要求される．

当研究では設計容易化を優先し，1 クロック・サイクル当たりスカラ演算命令とベクトル演算命令の 2 つの命令からなる命令語列を発行する最も単純な静的複数命令発行構成を用いることで，SIMD 演算とスカラ演算の並列処理可能なデータパスを提案する．また，スカラ部とベクトル部の命令語をそれぞれ 32 ビットの固定長にすることで，図 4.6 のような上位 32 ビットにスカラ演算用命令語，下位 32 ビットにベクトル演算用命令語を割り付ける 64 ビット 2 語命令に対応する．

提案データパスは SIMD 処理による高速化が必要不可欠な SAD 演算をベクトル演算パスで処理し，ブロックサイズ分の計数に用いる加算などの一部の処理をスカラ演算パスで処理することで，動き探索処理を並列に処理することができる．また，SAD 演算結果をベクトル・スカラレジスタファイル間で移動する転送オーバーヘッドが生じないようにするため，SAD 演算結果が得られ次第，それを必要とされるまで保持する FIFO レジスタをスカラ演算パス側に設ける．この構成により，SAD 演算結果が

得られるたびに，スカラ演算パスで実行中の他処理を止めることがなくなる上に，スカラ演算パスが空き次第，次の集計演算が開始できるようになり，並列処理効率を大きく高めることができる．

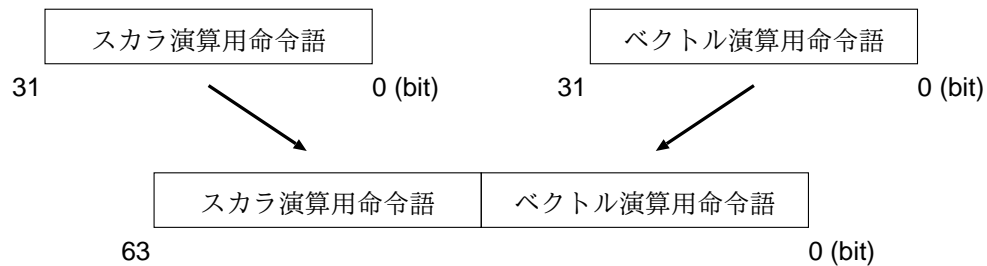


図 4.6: 提案データパスで用いる 64 ビット 2 語命令の構成

#### 4.4.2 非整列ブロックアクセス対応構成

2次元ブロックデータ単位の処理をおこなうために，提案データパスは当研究室で提案しているライン・ブロック両単位のデータアクセスが可能なキャッシュ[12]を用いる．この提案キャッシュにより，動き探索の処理単位である  $8 \times 4$  サイズのサブブロックデータをレジスタへ 1 サイクルで読み込むことができるようになる．また，データの再利用性を高めるため，隣接するブロックが格納されたレジスタファイル間の境界線をまたいだ水平・垂直両方向の非整列ブロックアクセスを可能とするアドレス修飾付きレジスタファイルとシャッフルユニットからなる機構を付加している．図 4.7 に参照画像データがレジスタファイル間の境界線をまたぎ，ブロック単位でデータアクセスを行う 2 次元配列図と 1 次元配列図を示す．

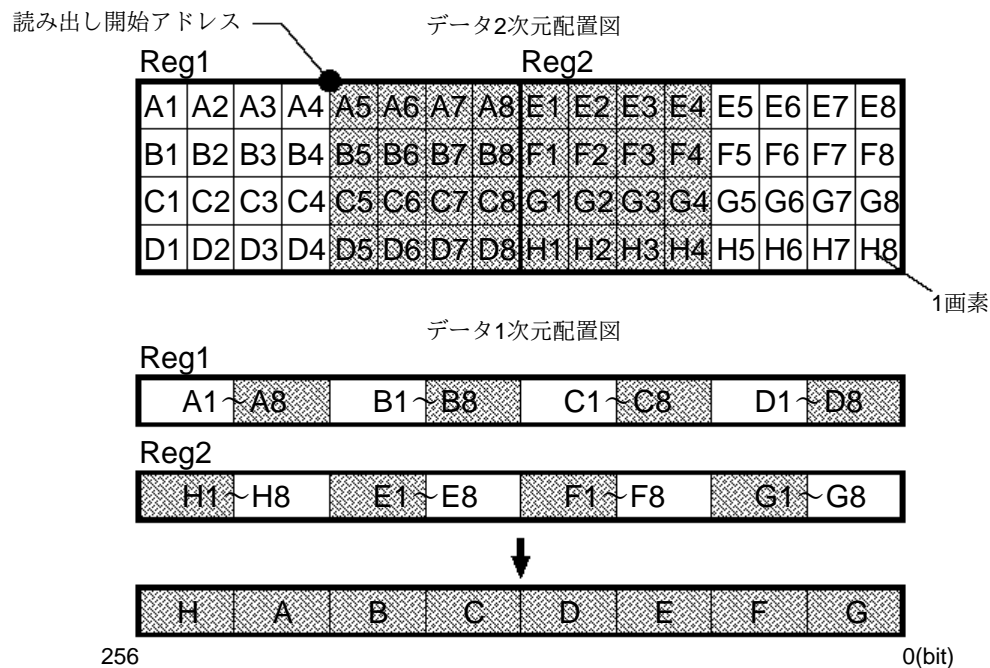


図 4.7: 非整列ブロックアクセス

#### 4.4.2.1 レジスタファイルアドレス修飾

ベクトルレジスタへタイル単位で参照画像を読み込む際にはキャッシュメモリの8バイト境界の整列化制約を受けるため、ベクトルレジスタへの任意位置の参照ブロックの読み込みは行えない。そこで、提案データパスが備えるベクトルレジスタは、データ読み出し時に開始アドレスに対してアドレス修飾を行えるようにすることで、水平・垂直両方向の非整列ブロックアクセスを実現する。この際、読み出されたデータはアドレス修飾を行った分だけ画素単位でずれるため、レジスタファイルの出力はデータの並び替え(ローテートシフト)を行うシャッフルユニットを通す必要がある。

#### 4.4.2.2 シャッフルユニット

シャッフルユニットは画素(バイト)単位でデータをローテートシフトするバレルシフト機能を備えることで、 $8 \times 4$ のサブブロックのデータの正しい並びで出力されるようにする。バレルシフトを用いたデータの並び替えの様子を図4.8に示す。

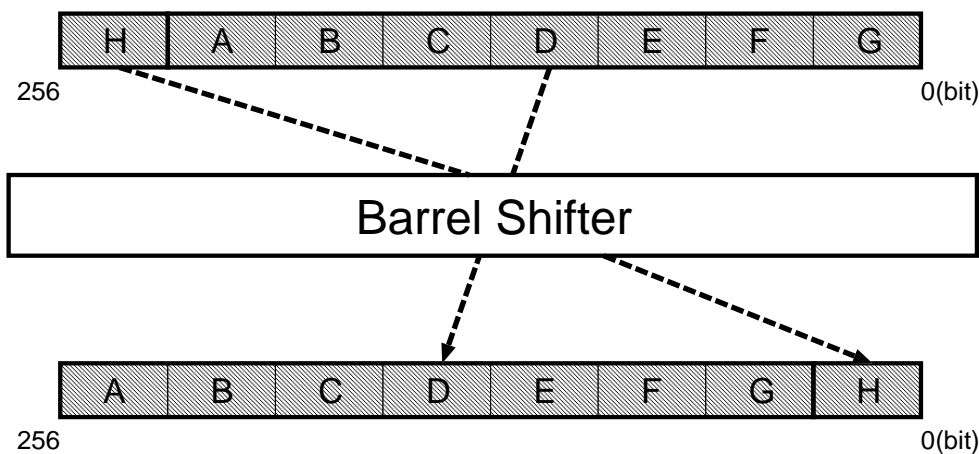


図 4.8: シャッフルユニットによるデータの並び替え

#### 4.4.3 再利用容易化構成

探索に用いる参照画像の再利用は，レジスタの保持する  $8 \times 4$  のサブブロックサイズの参照画像をマッチング対象のサブブロック位置の移動に応じて，順次入れ替えることで実現される．この際，移動に応じて参照画像データを格納するレジスタ番号を指定し直すと，ループ形式に収まらないためコードが複雑化したり，分岐のオーバーヘッドが増加する．そこで，提案データパスにはレジスタローテーション機能を付加して，この問題の解決をはかる．

レジスタローテーションは，スカラーレジスタの一部を  $rrb$ (レジスタ・リネーム・ベース) レジスタとして流用することで，次の走査時に利用する参照画像を含むベクトルレジスタ名を一括して変更できるようにする．図 4.9 に SUC パタンでの探索に十分な  $3 \times 3$  の計 9 個のレジスタを用いた縦方向の走査時のレジスタローテーションの実行を示す．各ベクトルレジスタ  $Reg0 \sim 8$  には参照画像がそれぞれ  $8 \times 4$  画素ずつ格納されており，SAD 演算を行うための参照画像はベクトルレジスタである  $Reg0 \sim 5$  へ格納されている． $Reg6 \sim 8$  には次の縦方向の走査時の SAD 演算に必要な参照画像が格納されている．また  $rrb$  レジスタへ  $Reg0 \sim 8$  に対して 3 を入力することでベクトルレジスタ番号が 3 ずつ変更され， $Reg0 \sim 5$  に次の SAD 演算に必要な参照画像が格納されている状態となる．

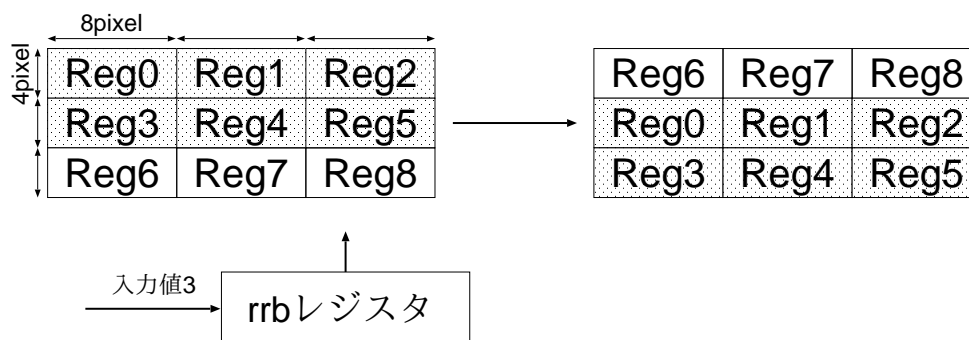


図 4.9: レジスタローテーション

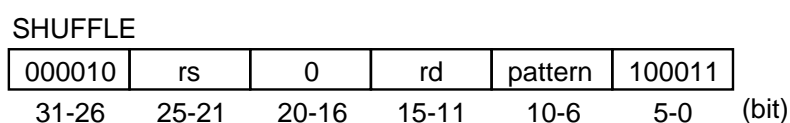
## 4.5 提案命令セット構成

考案する命令セットは，従来の基本的な MIPS スカラ命令をサポートしているものの，ハードウェア規模低減のため動き探索処理の実行に必要な最小限のベクトル命令に限定する．また汎用性と設計容易化を考慮し，各命令の命令長は 32 ビットの固定長とする．

制定した命令セットを以下に示す．

### 4.5.1 SHUFFLE 命令

SHUFFLE reg1, reg2, pattern



SHUFFLE 命令は，整列化制約の解除やダイヤモンドサーチなどの動き探索処理で用いられる探索パタンに応じて，アドレス修飾されたデータの並び替えをおこなう命令である．pattern フィールドに並び替えのパタンを入力することで，rs フィールドで指定されたデータを並び替え，その演算結果を rd フィールドで指定したレジスタに格納することが可能である．また，参照画像を入力することで，演算結果を 2 サイクル後の BB\_SAD 命令に用いることが可能である．

#### 4.5.2 BB\_SAD 命令

BB\_SAD reg1, reg2, reg3, pattern

BB_SAD					
000001	rs	rt	rd	pattern	100001
31-26	25-21	20-16	15-11	10-6	5-0 (bit)

BB\_SAD(Block Based SAD) 命令は、動き探索処理で用いられる  $8 \times 4$  ブロックサイズの差分絶対値和を並列に処理することできる命令である。従来の水平方向に演算が限られた MPSADB\_W 命令と比較し、効率的な 2 次元ブロックデータ単位での処理が可能である。rt フィールドで符号化対象画像を指定することで、SHUFFLE 命令によって並び替えられた参照画像との演算結果を rd フィールドで指定したレジスタに格納、または総和用 5 段ステージに送られ SAD 演算結果として FIFO レジスタに格納される。

rs フィールドに参照画像を入力することで、SHUFFLE 命令と同様、pattern フィールドに入力されたパタンに応じて、演算結果を 2 サイクル後の BB\_SAD 命令に用いることが可能である。この機能により、SHUFFLE 命令を挟むことなく連続して SAD 演算を実行可能となる。

#### 4.5.3 SAD\_SUM 命令

SAD\_SUM reg1, reg2

SAD_SUM					
010001	0	rt	rd	0	100000
31-26	25-21	20-16	15-11	10-6	5-0 (bit)

SAD\_SUM 命令は FIFO レジスタに格納された SAD 演算結果を取り出し加算する命令である。rt フィールドで指定した各探索点の SAD 値を加算することで、任意のブロックサイズの SAD 値を生成することができる。また、演算結果は rd フィールドに格納されると同時に最小値抽出用比較レジスタに送られ、最小 SAD 値を求めることができる。FIFO レジスタに格納されたデータはスカラ演算パスが空きしだいこの命令を発行し用いることで、SAD 演算、加算処理、最小値抽出を効率よく並列処理可能である。



#### 4.5.4 MINIMUM 命令

MINIMUM reg1

MINIMUM					
010010	0	0	rd	0	0
31-26	25-21	20-16	15-11	10-6	5-0 (bit)

最小値抽出 (MINIMUM) 命令は、最小値抽出用比較レジスタに格納される最小 SAD 値を rd フィールドで指定したレジスタに格納する命令である。この命令を用いることで、動き探索における最も類似の高いブロック位置を検出する。また、この命令実行時に最小値抽出用比較レジスタ内のデータはリセットされる。

#### 4.5.5 MOVE 命令

MOVE reg1, reg2, select

MOVE					
000111	rs	0	rd	select	0
31-26	25-21	20-16	15-11	10-6	5-0 (bit)

MOVE 命令はスカラレジスタからベクトルレジスタへデータを移行、またはベクトルレジスタからスカラレジスタへデータを移行する命令である。select フィールドによってベクトルレジスタに対する読み出し位置・格納位置を 64 ビット単位で選択することができる。

#### 4.5.6 addi 命令

addi reg1, reg2, constant

addi

001000	rs	rt	constant
31-26	25-21	20-16	15-0

addi 命令は rs フィールドで指定したレジスタの値と constant フィールドに入力された 16 ビットの値を加算し、結果を rt フィールドで指定したレジスタに格納する命令である。この命令を用いることで、rrb レジスタの値を書き換えレジスタローテーション機能を実現する。

#### 4.5.7 lw 命令

lw reg1, address(reg2)

lw

000001	rs	0	address
63-58	57-53	52-48	47-32

lw (Line)

100011	0	rt	0
31-26	25-21	20-16	15-0

lw (Tile)

100111	0	rt	0
31-26	25-21	20-16	15-0

ベクトル演算用の lw 命令は、スカラ演算パスでアドレス計算を行う。rs フィールドで指定したレジスタの値と address フィールドに入力された 16 ビットの値から生成したアドレスを参照することで、メモリから 256 ビットのデータを rt フィールドで指定したレジスタに格納する。また、31-26bit の op フィールドを変更することで、ライン・ブロック両単位のデータアクセスが可能である。

#### 4.5.8 sw 命令

sw reg1, address(reg2)

sw

000001	rs	0	address
63-58	57-53	52-48	47-32

sw (Line)

101011	0	rt	0
31-26	25-21	20-16	15-0

sw (Tile)

101111	0	rt	0
31-26	25-21	20-16	15-0

lw 命令と同様ベクトル演算用の sw 命令は，スカラ演算パスでアドレス計算を行う．rs フィールドで指定したレジスタの値と address フィールドに入力された 16 ビットの値から生成したアドレスを参照し，rt フィールドで指定したレジスタのデータをメモリに格納する．また，31-26bit の op フィールドを変更することで，ライン・ブロック両単位のデータアクセスが可能である．

## 5 提案データパスの設計

### 5.1 回路構成

提案する汎用志向データパスは、データパスを6ステージに分割したパイプライン処理方式をとる。図 5.10 に設計した全体の回路構成を示す。各パイプライン・ステージは以下の構成をとる

#### ①命令フェッチ

PC 中のアドレスを用いて命令メモリから命令を読み出す。

#### ②命令デコードとレジスタ・フェッチ/命令デコードとアドレスデコード

命令はスカラ演算命令とベクトル演算命令に分けられ、それぞれを各命令デコーダでデコードする。

スカラレジスタでのデータの読み出しを行い、ベクトルレジスタ用のアドレスは `rrb` デコーダにより生成される

#### ③実行/アドレス修飾とレジスタ・フェッチ

スカラ演算パスで命令を実行する。

ベクトルレジスタでの修飾アドレスとデータの読み出しを行う。

#### ④アドレス・デコード/実行

アドレスのデコードをする。

ベクトル演算パスで命令を実行する。

#### ⑤メモリアクセス

メモリに対しデータの読み出し・書き込みを行う。

#### ⑥ライトバック

データをスカラレジスタとベクトルレジスタに書き込む。

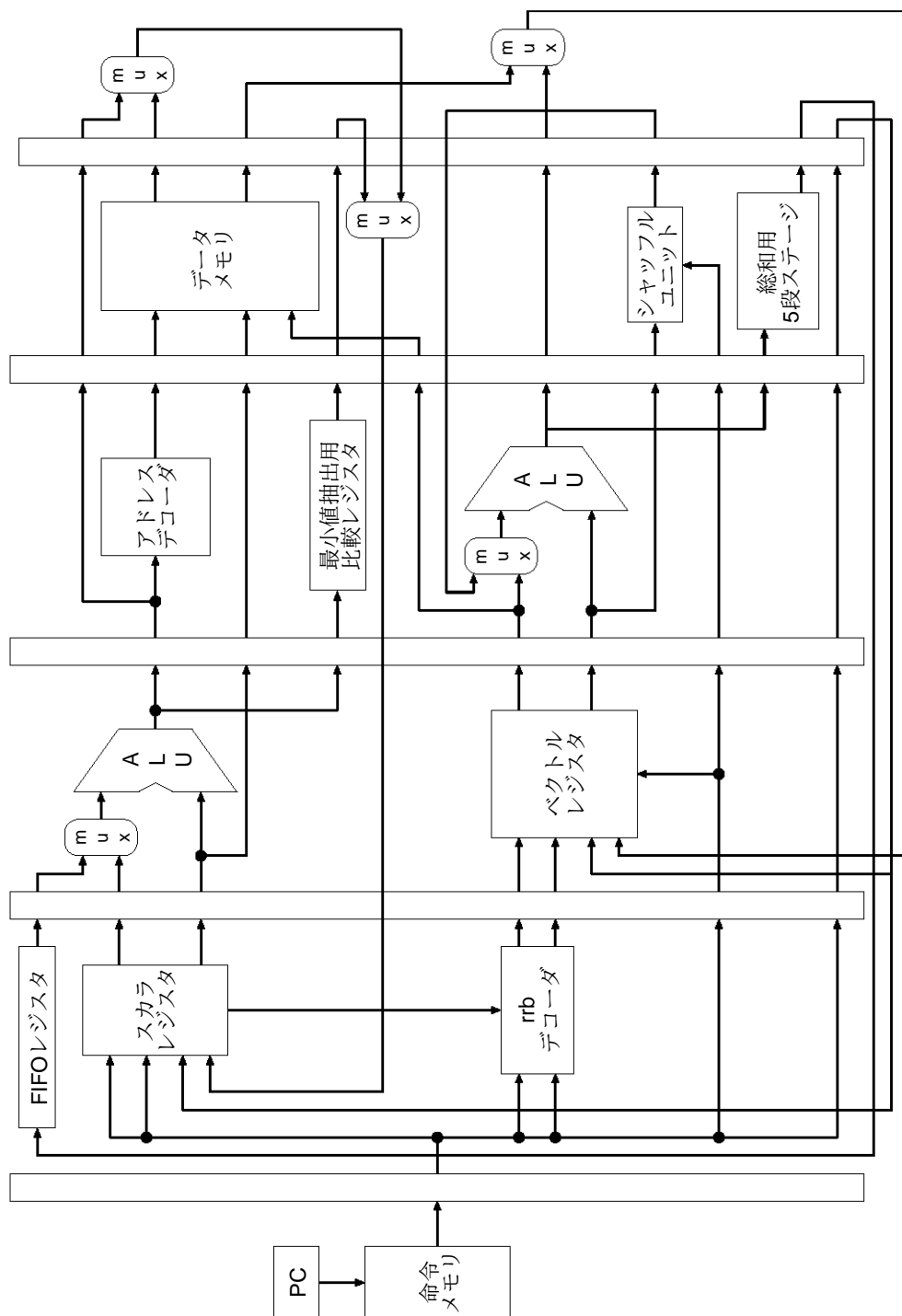


図 5.10: 回路全体図

### 5.1.1 レジスタ構成

提案データパスが備える図 5.11 に示すレジスタファイルは，汎用性を高めるために従来の x86 プロセッサに合わせた構成である．ス칼ラレジスタ 64 ビットを 32 本とベクトルレジスタ 256 ビットを 16 本備えており，ベクトルレジスタはデータ読み出し時に開始アドレスに対してアドレス修飾を行うことで水平・垂直両方向の非整列ブロックアクセスに対応している．またス칼ラレジスタの r0 はレジスタローテーションで用いる rrb レジスタの機能を割り付けられている．

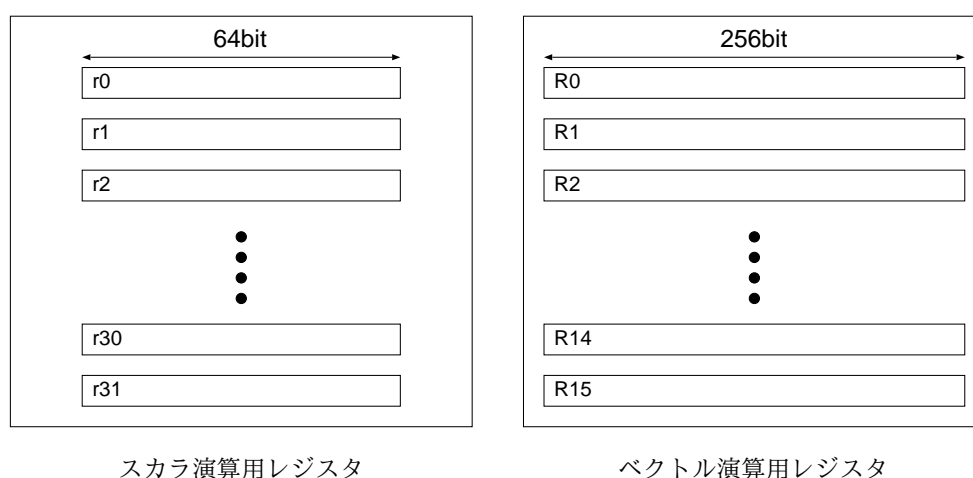


図 5.11: 提案データパスにおけるレジスタ構成

### 5.1.2 アドレス修飾機構

ブロック単位での処理を行う際，レジスタファイルのアドレス修飾機構で Pattern Signal から探索パターンなどに対応したアドレス修飾をおこなうことで，水平・垂直両方向の非整列ブロックアクセスを実現することができる．ただし，非整列ブロックアクセスによって読み出されたデータは SHUFFLE 命令を用いて，シャッフルユニットで正しいデータの順に並び替えを行う必要がある．

### 5.1.3 シャッフルユニット

非整列ブロックアクセスによって読み出されたデータは，図に示すシャッフルユニットによって正規の順に並び替えることができる．シャッフルユニットには画素単位でデータをシフトするバレルシフタが組み込まれており，SHUFFLE 命令や BB\_SAD 命令発行時に入力される Pattern Signal に対し，データの並び替えを行う．また，BB\_SAD 命令実行時に Shuffle Select 信号が 0 から 1 に切り替わるため，事前に並び替えたデータを演算に用いることが可能である．

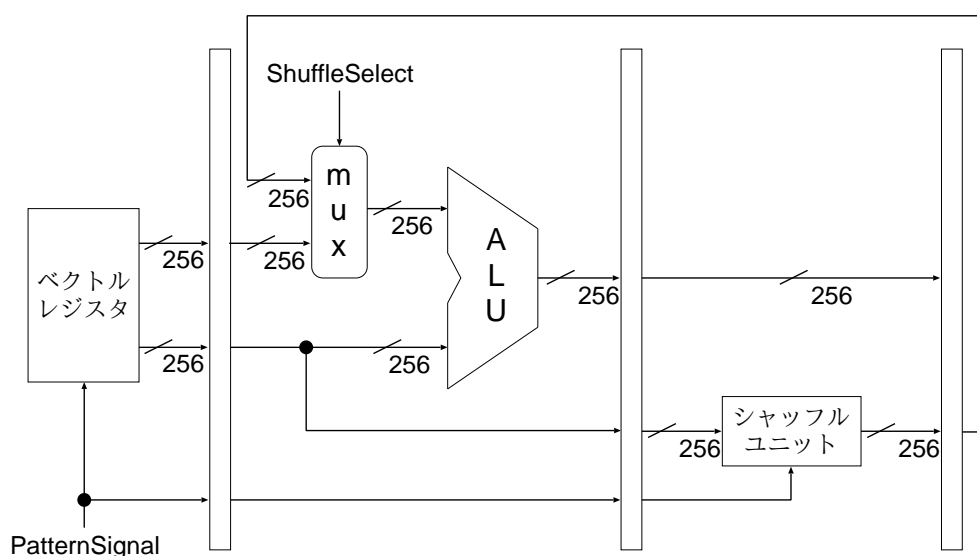


図 5.12: シャッフルユニット

### 5.1.4 総和用 5 段ステージ

総和用 5 段ステージでは，BB\_SAD 命令によって処理された 1 画素単位の差分絶対値の総和をとることができる．総和用 5 段ステージは 5 つのステージからなっており，各ステージで隣接するデータを足し合わせていくことで差分絶対値の総和をとり，SAD 値を生成する．また，BB\_SAD 命令実行時に Sum Req 信号が 0 から 1 に切り替わり，自動で総和演算を実行するため，スカラ演算パスとベクトル演算パスとは別に独立して処理をおこない並列化が可能である．総和用 5 段ステージの構成を図 5.13 に示す．

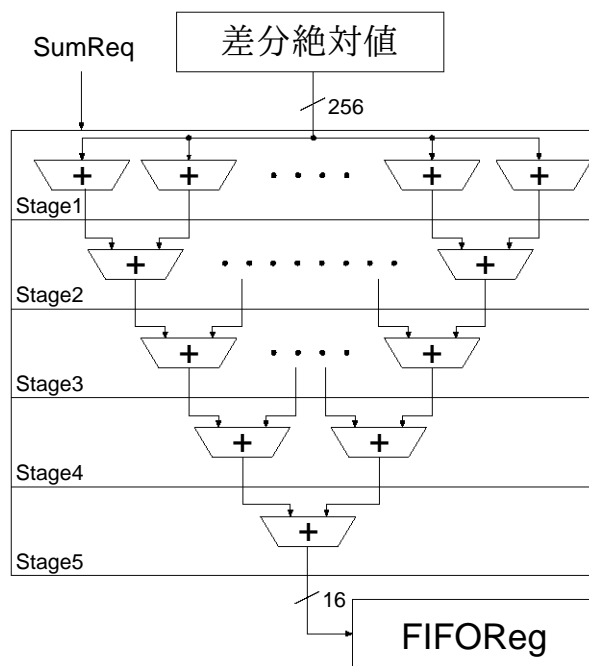


図 5.13: 総和用 5 段ステージ

#### 5.1.5 FIFO レジスタ

SAD 演算結果をレジスタファイル間で転送するオーバーヘッドを生じないようにするため、総和用 5 段ステージから SAD 演算結果が得られ次第保持する FIFO レジスタを設ける。図 5.14 に示す FIFO レジスタは、総和用 5 段ステージから出力されたデータを 16 個まで保持することが可能である。SAD 値が出力されると Write Req 信号が 0 から 1 に切り替わり、FIFO レジスタにデータが格納される。また、SAD\_SUM 命令実行時に Read Req 信号と FIFO Select 信号が 0 から 1 に切り替わり、FIFO レジスタからスカラ演算パスヘデータが読み出される。SAD\_SUM 命令の演算結果はスカラレジスタ、または最小値抽出用比較レジスタに格納される。



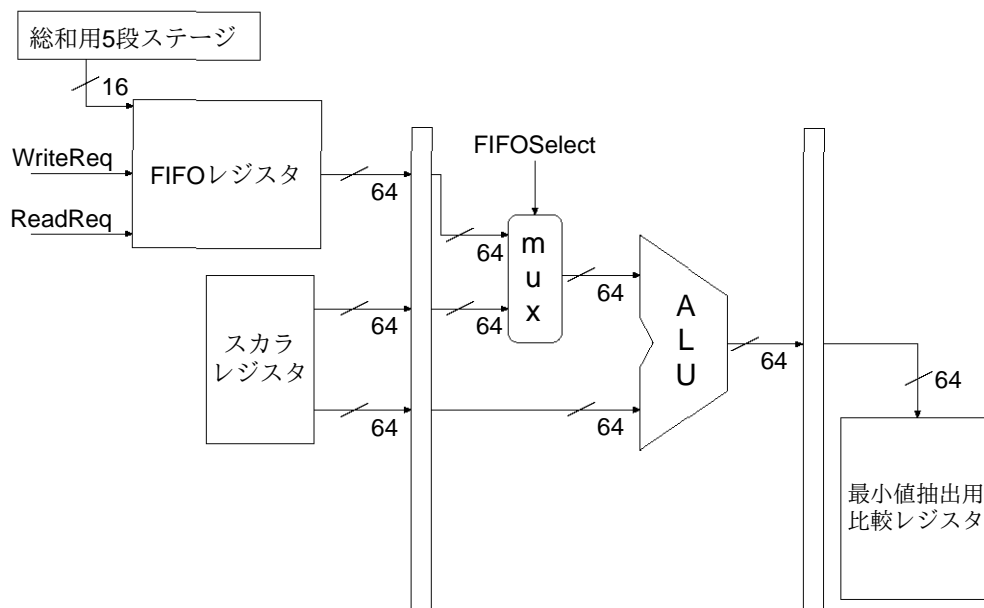


図 5.14: FIFO レジスタ

#### 5.1.6 最小値抽出用比較レジスタ

図 5.15 に示す最小値抽出用比較レジスタでは，SAD\_SUM 命令によって求められたブロックサイズごとの SAD 値を比較し，最小 SAD 値のみを保持するレジスタである．SAD\_SUM 命令実行時に Minimum Req 信号が 0 から 1 に切り替わり，レジスタに格納されたデータとの比較を行い，SAD 値が小さいデータの方が保持される．また，MINIMUM 命令実行時に Minimum Result 信号と Minimum Select が 0 から 1 に切り替わり，最小 SAD 値がスカラレジスタに格納されると同時に最小値抽出用比較レジスタ内のデータはリセットされる．比較処理は自動で行われるため，スカラ演算パスとベクトル演算パスとは別に独立して処理をおこない並列化が可能である．

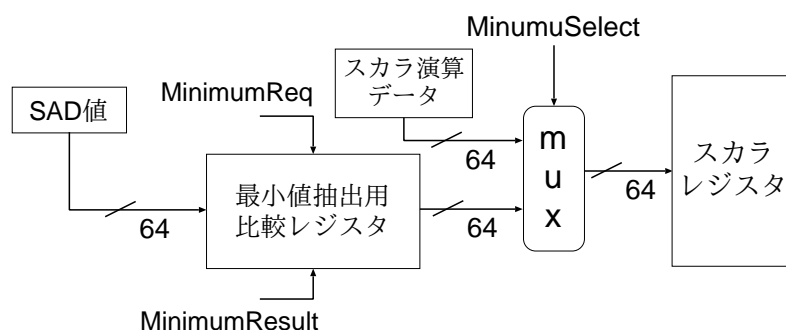


図 5.15: 最小値抽出用比較レジスタ

### 5.1.7 レジスタローテーション

動き探索におけるコード量の大幅な削減を実現するために、レジスタ番号の指定による複雑化を低減させ、データの再利用性を高めるレジスタローテーション機能を追加する。図 5.15 に示す `rrb` デコードは `rrb` レジスタから入力された値とプログラマから指定されたアドレスを用いて新たなアドレスを生成する。スカラレジスタの `r0` を `rrb` レジスタとして割り付けることで、レジスタローテーションを行うための専用命令の追加を必要とせず、`rrb` レジスタ内のデータを書き換えることができる。また、`r0` に 0 を格納することでレジスタローテーションをリセットすることができる。

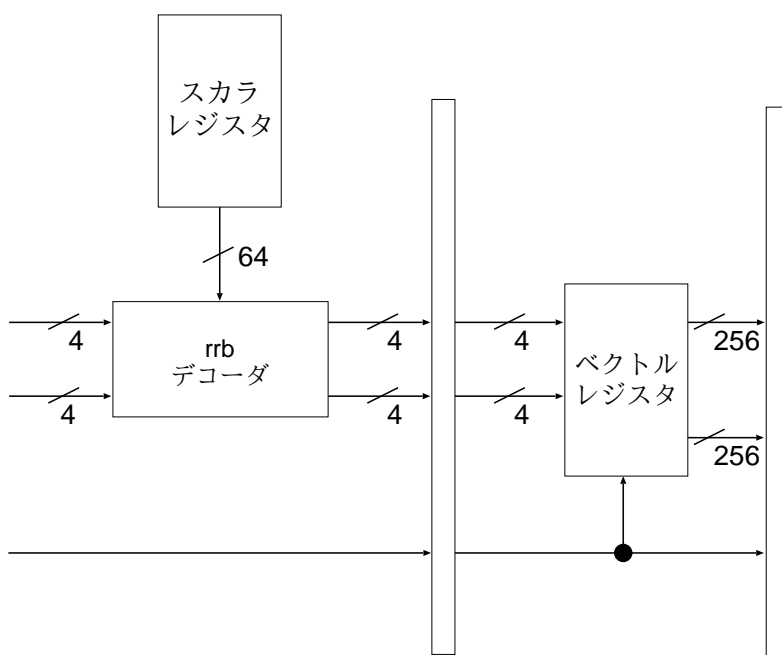


図 5.16: rrb デコーダ

## 5.2 制御線

上記のデータパスにおける制御線の設定は Write Req を除いて命令の命令操作コード・フィールドから命令デコーダによって決定される．表 5.1 に各命令と制御線の対応関係を示す．x は命令がその制御線に関与していないことを表す．

表 5.1: 命令と制御線の対応関係

命令名	Shuffle Select	Sum Req	Read Req	FIFO Select	Minimum Req	Minimum Result	Minimum Select
SHUFFLE	0	0	0	x	x	x	x
BB_SAD	1	1	0	x	x	x	x
SAD_SUM	x	x	1	1	1	0	0
MINIMUM	x	x	x	0	0	1	1

## 6 比較用専用志向データパスの構成

比較対象用に用いる提案する専用志向データパスの構成と命令セットを以下に示す。

### 6.1 参照画像格納用専用レジスタ

専用志向データパスは従来の汎用プロセッサが備えるスカラー演算用レジスタとベクトル演算用レジスタとは別に動き探索処理でのみ用いる 16x1 画素幅の参照画像格納用専用レジスタを備えている。参照画像格納用専用レジスタは合計 5 本用意されており、符号化対象画像 1 ラインに対して、探索中心のラインから上下プラスマイナス 2 画素分の探索範囲を扱うことができる。また、専用レジスタは FIFO 構造になっており、新しい参照画像データが読み込まれるたびに画像データを順送りする機能を持つ。

### 6.2 MSPSADBW 命令

MSPSADBW(Multiple Sparse point PSADBW) 命令は、4x1 画素幅の符号化対象画像 1 ラインと参照画像格納用専用レジスタ内に格納された 16x1 画素幅の参照画像 5 ライン内の複数の探索点に対する SAD 演算をライン単位で同時に並列処理する命令である。命令実行時には専用レジスタに新しい参照画像データを入力し順送りすることで、命令の連続実行を可能にし、データの再利用性を高めている。

### 6.3 探索点抽出ユニット

専用志向データパスは MSPSADBW 命令実行時に SAD 演算器に入力する参照画像データを制御する探索点抽出ユニットを備えている。探索点抽出ユニットは探索パターン信号を用いることで、ダイヤモンドパターンやスクエアパターンなどの探索パターンに応じて、自由に参照画像格納用専用レジスタ内のデータの読み出し位置を変更することが可能である。探索パターンとしてダイヤモンドサーチを用いた際の MSPSADBW 命令実行時の読み出し位置を 6.17 図に示す。

4×1 画素幅の符号化対象画像に対し，探索点の探索中心となる画素 c6～c9 の 1 点と上下左右 1 画素ずらした画素 b6～b9，c5～c8，c7～c10，d6～d9 の 4 点分の 4×1 画素幅の参照画像を読み出している．

## 6.4 専用志向データパスの特徴・まとめ

専用レジスタ内の参照画素データを順送りし再利用率を高めつつ，複数の SAD 演算器を用いての高速な動き探索が可能という利点がある．しかしながら，動き探索の高速化率は参照画像格納用専用レジスタに格納された探索点の数に大きく依存するため，従来の汎用プロセッサと比較し，近傍点の動き探索アルゴリズムに対して非常に有効であるものの，拡大型ダイヤモンドサーチなどで用いられる遠隔点の探索に対しての効果はほとんど得られないと考えられる．また，動き探索処理専用のレジスタやデータパスを用いるため，汎用志向データパスと比べ処理の汎用性は劣る．

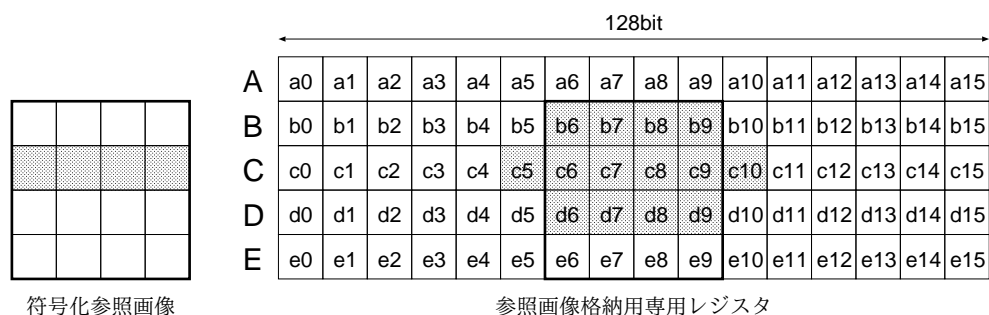


図 6.17: 探索点抽出ユニット

## 7 評価

提案する高効率動き検出対応の SIMD 併用型汎用データパスの優位性を明らかにするため、与えられた符号化対象ブロックに対する動き探索処理に必要なステップ数・データの再利用率およびハードウェアオーバーヘッドを以下の条件の下に算出した。

- ① SAD 演算では、従来手法の MPSADB\_W 命令、提案データパスの BB\_SAD 命令、MSPSADB\_W 命令をそれぞれ用いる
- ② 各提案データパスの動き探索処理は図 7.18 の処理ルーチンにより実行する
- ③ 各提案データパスの構成を最大限に活かすために、汎用志向データパスは走査に図 7.19(a) の  $8 \times 4$  単位のブロックマッチングを縦方向の走査を水平方向にずらしながら折り返すスネークスキャン [13] で走査し、専用志向データパスは  $4 \times 4$  単位のブロックマッチングを、図 7.19(b) の列方向の走査を繰り返すことで実行する
- ④ 汎用志向データパスでは SAD 演算時の BB\_SAD 命令と可変ブロック生成の加算処理は並列に実行可能とする
- ⑤ 専用志向データパスには最大 11 点の探索点に対する SAD 演算を並列に処理するために十分な 44 個の SAD 演算器を設けている

表 7.2～表 7.4 に H.265 で適用されている最小  $8 \times 4$  から最大  $64 \times 64$  のブロックサイズに対し、近傍点探索であるダイヤモンドパターンと *SUC* パターン、拡大型ダイヤモンドパターンなどに用いられる他の探索点との距離が 32 画素以上離れた遠隔の探索点 1 個に対する動き探索処理の総実行ステップ数と 1 画素当たりの平均実行ステップ数の算出結果を示す。

専用志向データパス

```

lw (Tile)
lw (Tile)
lw (Tile)
lw (Tile)
lw (Tile)
lw (Tile)
L1:
lw (Tile)
lw (Tile)
lw (Tile)
SHUFFLE
SHUFFLE
BB_SAD & SAD_SUM
BB_SAD & SAD_SUM

    * * *

BB_SAD & SAD_SUM
addi rrbReg + 3
L1に戻る(条件1)
SHUFFLE
SHUFFLE
BB_SAD & SAD_SUM
BB_SAD & SAD_SUM

    * * *

BB_SAD & SAD_SUM
addi rrbReg + 2
lw (Tile)
lw (Tile)
L1に戻る(条件2)
SAD_SUM
SAD_SUM

    * * *

SAD_SUM
終了

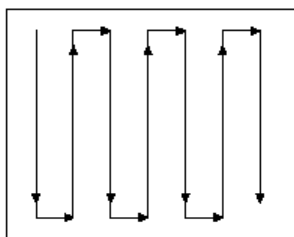
```

```

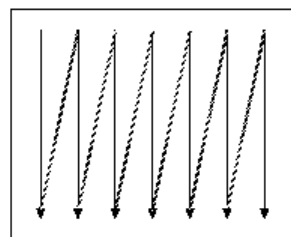
L1:
    lw
    lw
    lw
    lw
L2:
    MSPSADBW
    MSPSADBW
    MSPSADBW
    MSPSADBW
    L2に戻る(条件1)
    L1に戻る(条件2)
    終了

```

図 7.18: 処理ルーチン



(a)汎用志向データパス



(b)専用志向データパス

図 7.19: 動き探索の走査順序

表 7.2: ダイヤモンドパターンを用いた動き探索処理の実行総ステップ数

	ダイヤモンドパターン		
	従来手法	汎用志向データパス	専用志向データパス
8×4	51(1.594)	22(0.688)	16(0.500)
8×8	101(1.578)	32(0.500)	24(0.375)
8×16	201(1.570)	54(0.422)	40(0.313)
16×8	201(1.570)	53(0.414)	48(0.375)
16×16	401(1.566)	97(0.379)	80(0.313)
16×32	801(1.564)	185(0.361)	144(0.281)
32×16	801(1.564)	183(0.357)	160(0.313)
32×32	1601(1.563)	359(0.351)	288(0.281)
32×64	3201(1.563)	711(0.347)	544(0.266)
64×32	3201(1.563)	707(0.345)	576(0.281)
64×64	6401(1.563)	1411(0.344)	1088(0.266)

表 7.3: SUCパターンを用いた動き探索処理の実行総ステップ数

	SUCパターン		
	従来手法	汎用志向データパス	専用志向データパス
8×4	99(3.094)	34(1.063)	16(0.500)
8×8	197(3.078)	50(0.781)	24(0.375)
8×16	393(3.070)	84(0.656)	40(0.313)
16×8	393(3.070)	83(0.648)	48(0.375)
16×16	785(3.066)	151(0.590)	80(0.313)
16×32	1569(3.064)	287(0.561)	144(0.281)
32×16	1569(3.064)	285(0.557)	160(0.313)
32×32	3137(3.063)	557(0.540)	288(0.281)
32×64	6273(3.063)	1101(0.538)	544(0.266)
64×32	6273(3.063)	1097(0.536)	576(0.281)
64×64	12545(3.063)	2137(0.522)	1088(0.266)



表 7.4: 遠隔の探索点 1 点に対する動き探索処理の実行総ステップ数

	遠隔点探索		
	従来手法	汎用志向データパス	専用志向データパス
8×4	27(0.844)	10(0.313)	16(0.500)
8×8	53(0.828)	15(0.234)	24(0.375)
8×16	105(0.820)	25(0.195)	40(0.313)
16×8	105(0.820)	23(0.180)	48(0.375)
16×16	209(0.816)	45(0.176)	80(0.313)
16×32	417(0.814)	85(0.166)	144(0.281)
32×16	417(0.814)	81(0.158)	160(0.313)
32×32	833(0.813)	165(0.161)	288(0.281)
32×64	1665(0.813)	325(0.158)	544(0.266)
64×32	1665(0.813)	325(0.158)	576(0.281)
64×64	3329(0.813)	645(0.157)	1088(0.266)

算出結果から、汎用志向と専用志向の両方が従来手法と比べて動き探索処理の演算量を低減できていることがわかる。また、近傍点探索では専用志向の方が汎用志向よりも高速に動き探索を実行できていることがわかる。特に探索点を近傍に限定する *SUC* パタンでは実行ステップ数の差が大きくなっている。これは参照画像格納用レジスタに格納されたデータを複数同時処理できる専用志向データパスが、*SUC* パタンのような探索点が近傍にある探索パタンに対し非常に有効であるためである。しかし、遠隔点 1 点の探索では、汎用志向の方が専用志向よりも効果的であることがわかる。これは、専用志向の SAD 演算の並列度がわずか 4 であったのに対し、ブロック単位探索では並列度を 32 のまま実行できたためである。

また括弧内に示す平均処理ステップ数はブロックサイズが大きくなるほど小さくなり、探索効率が向上している傾向が見て取れるが、汎用志向の方が平均処理ステップ数の低減度が大きいいため、探索の折り返し数の多い大ブロック探索で効率が良くなることがわかる。

各手法を用いたブロックサイズの平均ステップ数を探索手法ごとに比較したものを表 7.20 に示す。動き探索処理演算量は従来手法の演算量を 1 として正規化したときの値である。

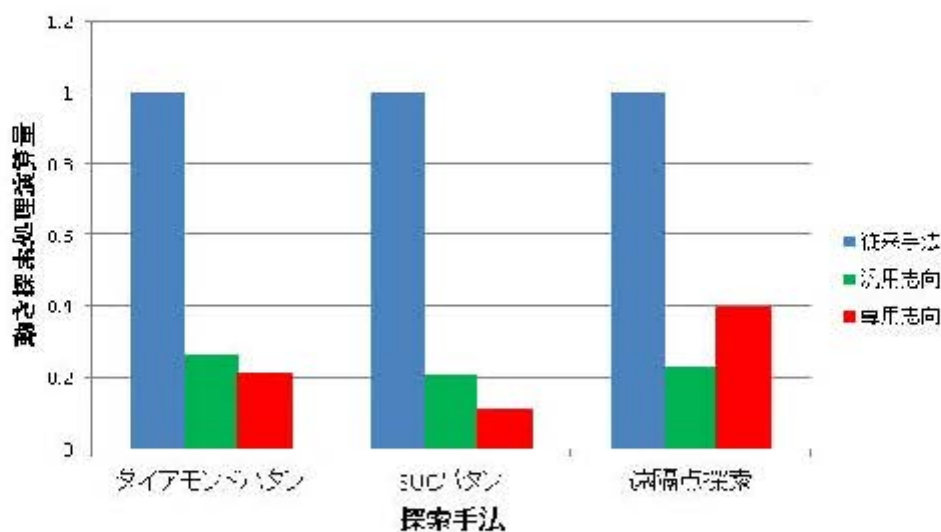


図 7.20: 高速化率

この結果から，汎用志向は従来手法と比較し近傍点の探索では約 3.83 ~ 4.83 倍の高速化が達成されていることがわかる．これは専用志向の約 4.84 ~ 9.48 倍の高速化率と比べ劣るものの，遠隔点の探索では専用志向の約 2.53 倍を上回る約 4.38 倍の高速化率が達成されている．

次に，参照画像データの再利用率を評価するために，ダイヤモンドパターンを用いた際のデータの総転送量を評価した結果を表 7.5 に示す．ここで，総転送量は図より LOAD 命令実行数に各データパスの一度のメモリ読み込み量 (32Byte, ただし MS SAD 命令実行時は 16Byte) の積算で求めたものである．

表より，汎用志向の方がデータの総転送量が少ないことがわかる．これは走査の折り返し法としてスネークスキャンを用いることで，専用志向と比較して参照画像の同じ領域の画素データを再度読み込む回数が大幅に削減されるためである．

また論理合成を行い，設計した汎用志向データパスのハードウェア規模を 2 入力の NAND 換算ゲート数として算出した結果を表 7.6 に示す．評価には動き探索処理専用 x86 プロセッサに追加したデータパスである最小値抽出用比較レジスタ，FIFO レジスタ，総和用 5 段ステージ，rrb デコーダおよび汎用志向データパス全体のハードウェア規模を用いる．

表 7.5: 参照画像データの総転送量 (Byte)

	汎用志向データパス	専用志向データパス
8×4	192	256
8×8	352	384
8×16	544	640
16×8	512	768
16×16	896	1280
16×32	1664	2304
32×16	1600	2560
32×32	3136	4608
32×64	6208	8704
64×32	6080	9216
64×64	12224	17408

表 7.6: ハードウェア規模

	NAND ゲート (gate)
最小値抽出用比較レジスタ	853
FIFO レジスタ	3024
総和用 5 段ステージ	1598
rrb デコーダ	158
追加データパス	5633
全体規模	176082

この結果より，当研究で提案する動き探索処理用のデータパスは，ハードウェア全体対し，約 3.20% のハードウェア規模増で実現できることが明らかになった．また，設計したデータパスに対し，x86 プロセッサ等はさらに豊富な機能を備えており，ハードウェアの全体規模は大きくなるため，追加したデータパスの割合はいっそう小さくなると考えられる．このため，提案データパスは従来手法に対する動き探索処理の高速化率と比べ，汎用性を重視しつつ，最低限のハードウェア規模の追加で十分に高速化されることを明らかにしたといえる．

## 8 あとがき

本論文は、動画像符号化の処理量の大半を占める動き探索処理の高速化を図るために従来の汎用プロセッサとの適合性の高い高効率動き検出対応の SIMD 併用型汎用データパスとその命令セットを提案し、その設計を行った。最小限のハードウェアコストで最大限の動き探索効率化を実現するために、SIMD 併用型の汎用的なデータパスへの動き探索専用ハードウェアの追加を最低限に留めた。その結果、追加したデータパスのハードウェア規模が全体の約 3.20%でありながら、近傍点の探索で約 3.83 ~ 4.83 倍、遠隔点の探索で約 4.38 倍高速化できることを示した。これは当研究室が提案してきた動き探索処理専用の演算器を用いた専用志向の SIMD データパス構成での約 4.84 ~ 9.48 倍、約 2.53 倍の高速化率と比較しても遜色のない探索性能といえる。

今後の課題として、比較用として専用志向データパスの設計を行い、その回路構成のハードウェア規模について検証を行う必要がある。また、動き探索処理全体に要する実行サイクル数を求め、2 つの提案データパスの優劣を明らかにして行く必要がある。

## 謝辞

本研究を遂行するにあたり，日頃から御指導，御助言を頂きました近藤利夫教授，佐々木敬泰助教に感謝いたします．また，研究に協力していただいた計算機アーキテクチャ研究室の方々に感謝の意を表します．

## 参考文献

- [1] ITU-R Recommendation H.264 “Advanced video coding for generic audiovisual services, ” May 2003.
- [2] 大久保，鈴木，高村，中條，H.265/HEVC 教科書，ISBN-978-4-8443-3468-2，2013 年 10 月.
- [3] Sung Dea Kim, Myung Hoon Sunwoo, “MESIP: A Configurable and Data Reusable Motion Estimation Specific Instruction-Set Processor ” IEEE Trans. on Circuit Systems., pp.1767-1780, Oct. 2013.
- [4] Zhibo Chen, et al. “Fast integer-pel and fractional-pel motion estimation for H. 264/AVC ” Journal of Visual Communication and Image Representation 17.2, pp.264-290, April. 2006.
- [5] Tang XL, Dai SK, Cai CH, “An analysis of TZ search algorithm in JMVC, ”IEEE Proc. ICGCS., pp.516–520, June 2010.
- [6] O. Ndili and T. Ogunfunmi, “Hardware-oriented modified diamond search for motion estimation in H.264/AVC, ” IEEE Proc. ICIP., pp.749–752, Sep. 2010.
- [7] Z. Chen, P. Zhou and Y. He, “Fast integer PEL and fractional PEL motion estimation for JVT, ”JVT-F017, 6th meeting, pp.5–13, Dec. 2002.
- [8] ARM ソフトウェア開発ツール RealView Development Suite Version 4.0 RealView Compilation Tools アセンブラガイド [Online]: <http://infocenter.arm.com/help/index.jsp>
- [9] Intel, Corp. (2010 年 4 月). Intel Advanced Vector Extentions プログラミング・リファレンス [Online]: <http://www.intel.co.jp/>

- [10] Intel, Corp. (2011, June.) Intel Advanced Vector Extensions Programming Reference [Online]: <http://www.intel.co.jp/>
- [11] David A. Patterson, John L. Hennessy(著), 成田 光彰(訳), コンピュータの構成と設計 ハードウェアとソフトウェアのインタフェース(下) 第3版, ISBN978-4-8222-8267-7, 2006年3月.
- [12] 王, 深澤, 近藤, 佐々木, “タイル・ライン両アクセス対応キャッシュメモリの提案,”電子情報通信学会技術研究報告(集積回路研究会), 2014年10月.
- [13] X. Wen, O. C. Au, J. Xu, L. Fang, R. Cha and J. Li, “ An analysis of TZ search algorithm in JMVC, ” IEEE Trans. on Circuit Syst. Video Technol., Vol.21, No.2, pp.206–219, Feb. 2011.

## A シミュレーションによる動作検証

### A.1 ModelSim の起動

図 1.21 に示す ModelSim ボタンをクリックし，HDL シミュレータ ModelSim を起動する．ModelSim ボタンを押すと，Start ModelSim ウィンドウが現れる．ここでは特に設定する項目はないので OK ボタンをクリックする．設計ミスがなければ図 1.22 のようなウィンドウが現れる．

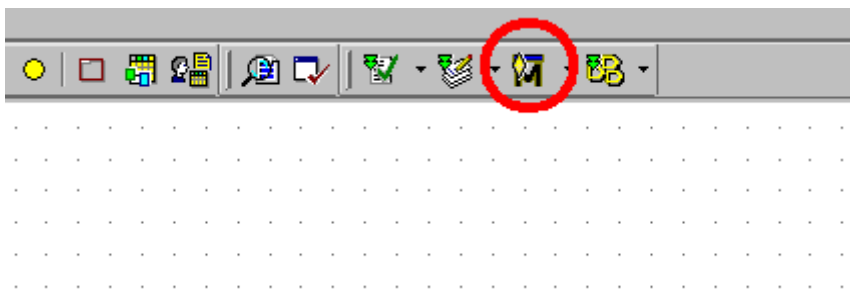


図 1.21: ModelSim ボタン

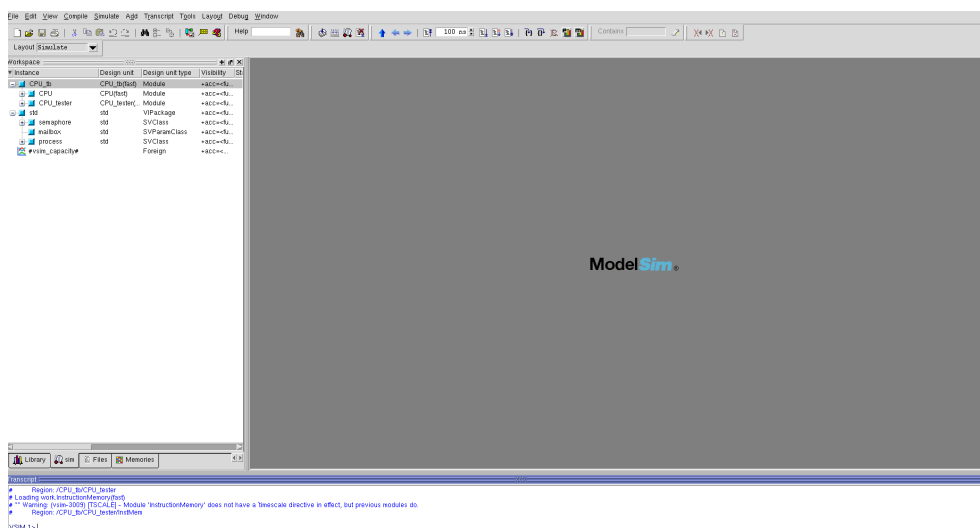


図 1.22: ModelSim ウィンドウ

## A.2 波形表示する信号線の選択

ブロック図設計ウィンドウから任意の信号線をクリックして選択し，図 1.23 に示す波形表示ボタンをクリックし，観測したい信号線を波形ビューアに追加する．



図 1.23: 波形表示ボタン

## A.3 シミュレーション

ModelSim，あるいは波形ビューア，ブロック図入力ウィンドウにある図 1.24 に示すシミュレーション開始ボタンをクリックし，シミュレーションを開始する．

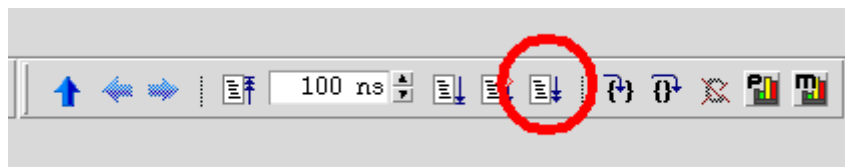


図 1.24: シミュレーション開始ボタン



## B CPUの動作テスト

CPUの動作テストを行うためには、設計者の環境に合わせて命令メモリの初期化部分を変更する必要がある。Design Unit の CPU\_tester をダブル・クリックすると、テキスト・エディタが起動し、テスト用の回路 (Verilog-HDL で記述されている) が表示される。初期化部分を変更する場合は initial begin から end までの間の任意の場所に、

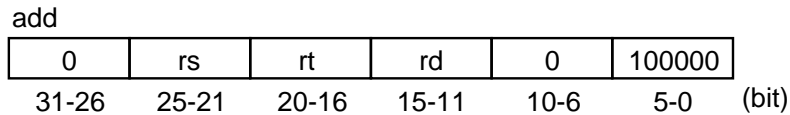
```
InstMem.mem_data[0] = 64'h2001123420015678;
```

のように記述する。この例では、スカラ演算用レジスタ\$1 に 1234、ベクトル演算用レジスタ\$1 に 5678 を格納している。メモリの初期化部分を変更後、ModlSim ボタンをクリックし、ModelSim を起動する。その後、観測したい信号線を波形ビューアに追加し、シミュレーションを開始する。

## C MIPS 命令セットの詳細

### C.1 add 命令

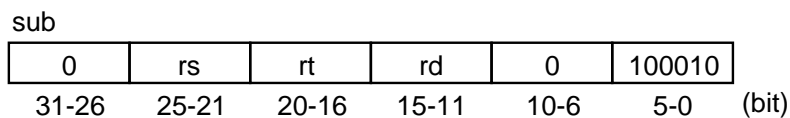
add reg1, reg2, reg3



rs フィールドで指定したレジスタの値と rt フィールドで指定したレジスタの値を加算し、結果を rd フィールドで指定したレジスタに格納する。

### C.2 sub 命令

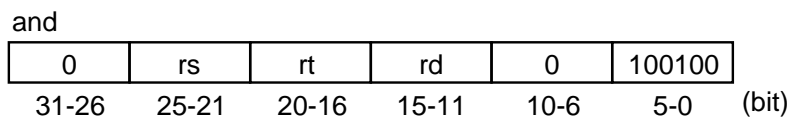
sub reg1, reg2, reg3



rs フィールドで指定したレジスタの値と rt フィールドで指定したレジスタの値を減算し、結果を rd フィールドで指定したレジスタに格納する。

### C.3 and 命令

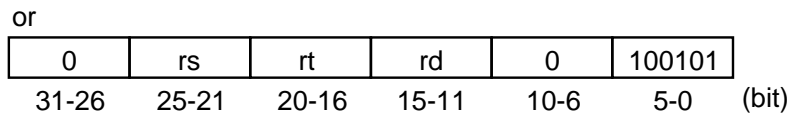
and reg1, reg2, reg3



rs フィールドで指定したレジスタの値と rt フィールドで指定したレジスタの値をビットごとに論理積をとり、結果を rd フィールドで指定したレジスタに格納する。

## C.4 or 命令

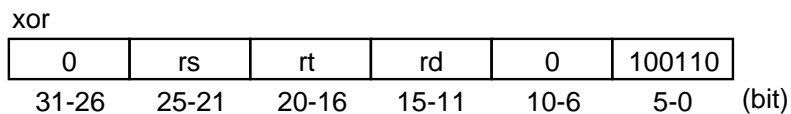
or reg1, reg2, reg3



rs フィールドで指定したレジスタの値と rt フィールドで指定したレジスタの値をビットごとに論理和をとり、結果を rd フィールドで指定したレジスタに格納する。

## C.5 xor 命令

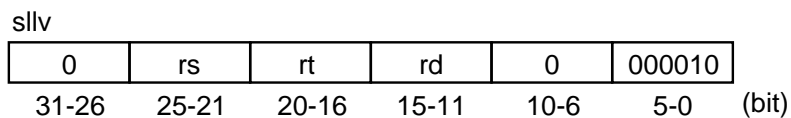
xor reg1, reg2, reg3



rs フィールドで指定したレジスタの値と rt フィールドで指定したレジスタの値をビットごとに排他的論理和をとり、結果を rd フィールドで指定したレジスタに格納する。

## C.6 sllv 命令

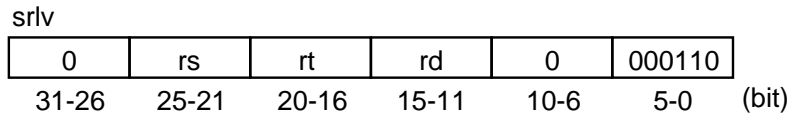
sllv reg1, reg2, reg3



rt フィールドで指定したレジスタの値を rs フィールドで指定したレジスタの値の数だけ左シフトし、結果を rd フィールドで指定したレジスタに格納する。

## C.7 srlv 命令

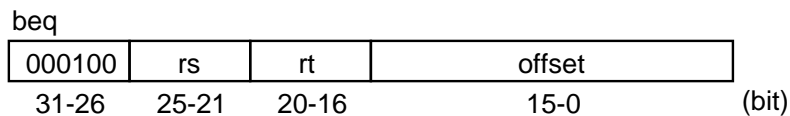
srlv reg1, reg2, reg3



rt フィールドで指定したレジスタの値を rs フィールドで指定したレジスタの値の数だけ右シフトし、結果を rd フィールドで指定したレジスタに格納する。

## C.8 beq 命令

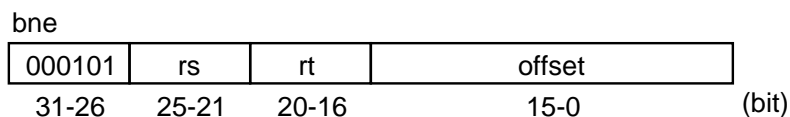
beq reg1, reg2, offset



もし、rs フィールドで指定したレジスタの値と rt フィールドで指定したレジスタの値が一致した場合は、現在のプログラムカウンタと offset から計算した分岐ターゲット・アドレスに分岐する。一致しなかった場合は、次の命令を実行する。

## C.9 bne 命令

bne reg1, reg2, offset



もし、rs フィールドで指定したレジスタの値と rt フィールドで指定したレジスタの値が一致しなかった場合は、現在のプログラムカウンタと

offset から計算した分岐ターゲット・アドレスに分岐する．一致した場合は，次の命令を実行する．