EMAX5SPC-0001
Ver.0.9: May. 6 2015
Ver.1.6: Sep. 1 2016
Ver.1.7: Sep. 21 2016
Ver.1.8: Dec. 9 2016

EMAX5/L2 and EMAX5/ZYNQ64 Architecture Handbook
– Energy-aware Multimode Accelerator eXtension –

Nara Institute of Science and Technology
Computing Architecture Laboratory
Accelerator Group

Table of contents

List of Figures

List of Tables

Chapter 1

EMAX5/L2

1.1 Overview

一般に，計算アクセラレータは，単独で OS や複雑なプログラムを走行させることが困難であるため，実用的システムの構築には汎用 CPU との連携が欠かせない．汎用 CPU に接続する際には，キャッシュメモリとの連携が重要であり，キャッシュラインをそのまま利用できるショートベクトル（SIMD）方式が一般に採用される．より大規模なアクセラレータが必要な場合，同様に SIMD を基本とする演算機構を二次キャッシュに接続する．さらにロングベクトル方式を採用して大規模化する場合，主記憶に接続したり，主記憶と DMA 転送が可能な別のメモリ空間に接続する．

さて，グラフ処理に代表される非定型処理では，均一なデータ幅を確保できないため，SIMD の効果は限定的である．かわりに，データ依存関係が多いことを利用すれば，CGRA の効果が期待できる．ただし，CGRA は，複数の内蔵メモリブロックの各々にアドレス範囲を設定し，各々を連続参照する利用が一般的であり，アドレスによりブロック位置が一意に決まる従来型キャッシュと共存させることは困難である．また，内蔵メモリに収容できない大規模グラフ処理のためには，CGRA と主記憶とのシームレスな接続および一体的なパイプライン化が重要であるものの，毎サイクル一斉演算を行う CGRA と長大な主記憶レイテンシの共存も困難である．このような困難から，グラフ処理の高速化を図るアクセラレータと汎用 CPU とは，一般に，主記憶を介して接続する方法しかなく，アクセラレータを使用する度に，キャッシュフラッシュなどのオーバヘッドが発生する．

そこで，本プロジェクトでは，CGRA 内蔵メモリブロックを (1)SIMD バッファ; (2) ステンシルメモリ; (3) キャッシュ; (4) 主記憶レイテンシ隠蔽バッファ; の全てに利用できるアクセラレータ（以下，EMAX5/L2）を開発することにした．

1.2 EMAX2との差異

図 1.1 は，EMAX2 をベースとする従来型のシステム構成である．EMAX2 の CGRA 内蔵メモリブロックは，汎用 CPU の主記憶とは別空間であり，プログラマは，全ての内蔵メモリに関して，主記憶との間の明示的なデータ転送を意識する必要がある．一方，図 1.2 のように，キャッシュメモリを CGRA 内蔵メモリブロックに配置できれば，様々なオーバヘッドを削減できる．EMAX5/L2 において新たに導入される機能は，全面的なキャッシュフラッシュを必要としない，汎用 CPU とのデータ共有機能である．図 1.3 に 3 つの機能要件を示す．

機能要件 1 は，複数ブロックからの，連続複数ワード毎サイクル同時読み出しである．左端の例では，連続領域 A が 1 つのブロックに対応する．アドレスは，毎サイクル 4 ワード分増加し，毎サイクル 4 ワードのデータを同時に読み出す．中央の例は，複数ブロックを連結して容量を増加させた構成である．さらに，右端の例は，複数ブロックから構成される 1 つの段を複数連結した構成である．右端の構成の場合，段数分の異なるアドレスを供給して，段数分の複数ワード毎サイクル読み出しが可能である．本構成は，SIMD やグラフ処理の並列実行により高性能化を図るのに適している．なお，SIMD の場合は 4 ワードが配列の連

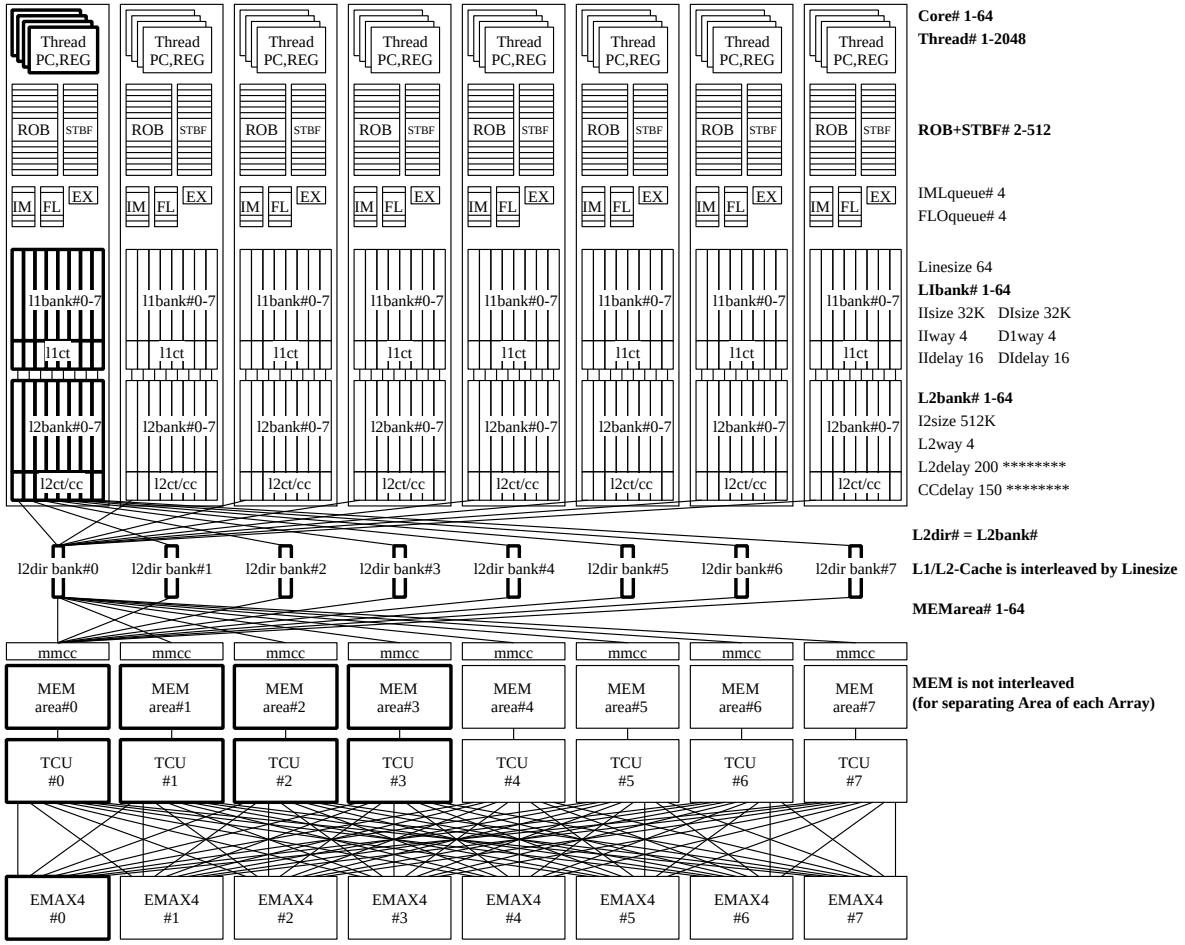


Figure.1.1: Overview of EMAX2 system

続 4 要素に対応しており，読み出しデータは後続の SIMD 命令により一斉演算されるのに対し，グラフ処理の場合は 4 ワードが構造体配列の各メンバに対応しており，読み出しデータの扱いは各ワードによって異なる．すなわち後続命令が条件判断等を伴う複雑な命令となる点が SIMD とは異なる．まとめると，次のデータ供給能力がある．

SIMD 計算時の能力
アドレス数: 全段数
各データ幅: 4 ワード

機能要件 2 は，複数ブロックからの，連続単一ワード毎サイクル同時読み出しである．左端の例では，連続領域 C が 1 つのブロックに対応する．アドレスは，毎サイクル 1 ワード分増加し，毎サイクル 1 ワードのデータを読み出す．中央の例は，複数ブロックを配置して，同時に読み出し可能な空間数を増加させた構成である．機能要件 1 では連続アドレスから複数ワードを同時に読み出すのに対して，機能要件 2 では異なる連続アドレスから各々 1 ワードを同時に読み出す点が異なる．右端の例は，複数ブロックから構成される 1 つの段を複数連結した構成である．右端の構成の場合，段内ブロック数×段数分の異なるアドレスを供給して，全ブロック数分の単一ワード毎サイクル読み出しが可能である．本構成は，次数の高いステンシル計算の並列実行により高性能化を図るのに適している．まとめると，次のデータ供給能力がある．

ステンシル計算時の能力
アドレス数: 全ブロック数 (段数× 4)
各データ幅: 1 ワード

機能要件 3 は，全ブロックのパイプライン探索および該当行の読み出し･伝搬である．左端の例では，連続領域 A が 1 つのブロックに対応する．連続領域は複数のキャッシュラインを包含する構成であり，キャッ

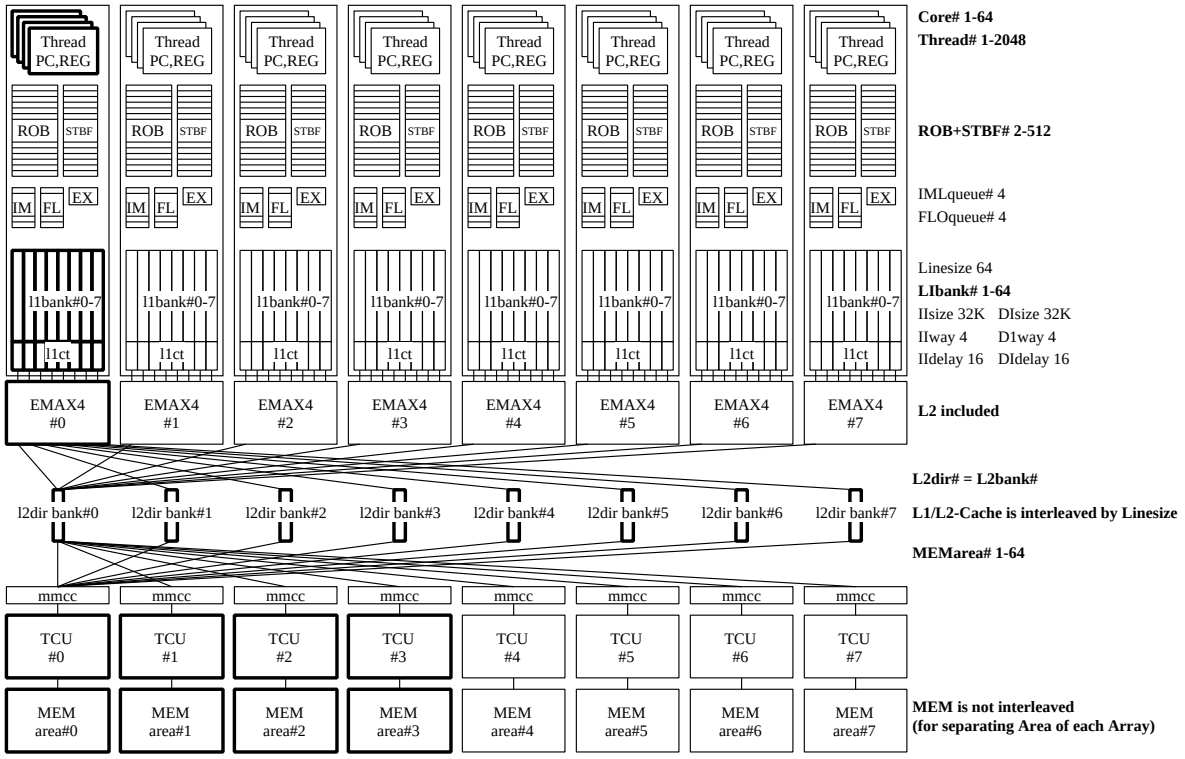


Figure.1.2: Overview of EMAX5/L2 system

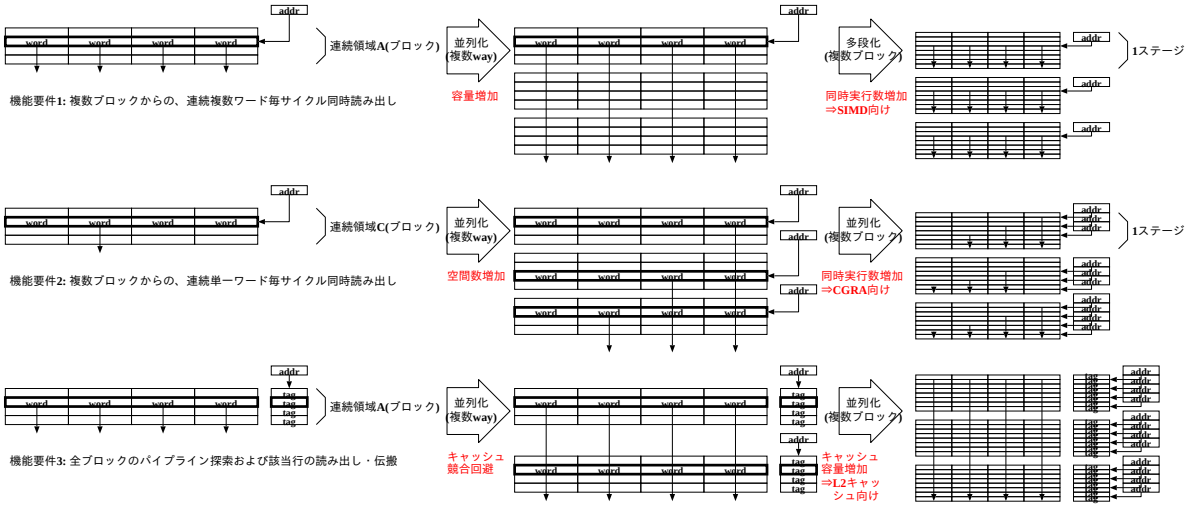


Figure.1.3: Basic function of EMAX5/L2

シュラインの先頭に該当するアドレスが供給される．機能要件 1 および機能要件 2 とは異なり，アドレスの一部によりキャッシュラインを特定した後，キャッシュライン毎に設けられた個別 tag 情報とアドレスの上位ビットを比較し，内容が一致したキャッシュラインのみを読み出す．中央の例は，一般的なキャッシュと同様に，キャッシュラインの衝突を抑制するための複数 way 構成である．全ての way を同時に探索し，個別 tag が一致したキャッシュラインのみを読み出す．右端の例は，複数ブロック（複数 way）から構成される 1 つの段を複数連結した構成である．右端の構成の場合，段内ブロック数×段数分の空間を 4way セットアソシアティブのキャッシュとして使用できる．まとめると，次のデータ供給能力がある．

キャッシュ利用時の能力
アドレス数: 1(パイプライン動作)
各データ幅: 4 ワード (パイプライン動作)

EMAX2 と EMAX5/L2 の主要な差異は，アクセラレータに適するメモリ構成である機能要件 1 および 2 と，キャッシュに適するメモリ構成である機能要件 3 を 1 つのハードウェア上で効率良く一度に実現する機能の有無にある．

1.3 Basic structure

性能および機能を落すことなく，機能要件 1，2，3 の全てを実現するためには，段位置の管理方法が重要である．機能要件 1 および 2 では，あらかじめ，ソフトウェアが指定した段に必要なデータを配置し，段内に閉じたメモリ空間に対して読み書きできればよい．しかし，機能要件 3 では，アドレスの一部を用いて，メモリブロックの位置を特定し，さらにアドレスの別の一部を用いて，メモリブロック内の相対位置を特定し，当該キャッシュラインが参照アドレスと正しく対応しているかを個別 tag 比較により検証する必要がある．フルアソシアティブ構成とすれば，任意の段位置にキャッシュラインを対応付けることが論理的には可能であるものの，EMAX5/L2 が内蔵するメモリの総量は二次キャッシュ程度以上であるため，キャッシュライン数の多さから，従来型フルアソシアティブ方式の採用は現実的ではない．そこで，機能要件 1 および 2 のために，ソフトウェアがメモリブロック位置を明示的に指定可能とするとともに，機能要件 3 のために，パイプライン処理による実質的なフルアソシアティブキャッシュを実現可能とする．汎用 CPU がデータの前処理を行い，CGRA 型アクセラレータにデータを渡す場合，また，CGRA 型アクセラレータの処理結果を汎用 CPU が引き続き利用する場合に必要となるデータ共有を広範囲のキャッシュフラッシュを必要とすることなく可能とすることにより，処理の高速化が可能となる．

ところで，グラフ処理のためには，機能要件 1 に加えて，主記憶参照時の遅延時間隠蔽のために，各ブロックを利用できる必要がある．この遅延時間吸収機構は，機能要件 4 とする．

図 1.4 に基本構成（5 段分）を示し，続いて，各機能要件に対応する，(1)SIMD 計算；(2) ステンシル計算；(3) 大容量キャッシュ；(4) グラフ処理；の 4 種類の計算における挙動を説明する．本実装の特徴は，4 種類の参照方法それぞれに対して最適化が可能であるとともに，切替えのデータ移動が最小限で済む点にある．図 1.4 は，汎用 CPU がキャッシュミスした際に EMAX5/L2 内キャッシュ（L2 キャッシュ）の参照を要求するインタフェース（L2addr）を最上部に備え，さらに L2 キャッシュにも存在しない場合にさらに下位の記憶装置へ参照を要求するインタフェースを最下部に備える，

さて，汎用演算器およびメモリブロック（1R+1W）からなる基本ユニットは，Read 専用と Write 専用の 2 つのアドレス生成ユニットを備えており，Read と Write を同時に行うことができる．メモリブロックでは，右から左方向へアドレスが増加する順にワード（例では 8 バイト）が配置され，連続 8 ワードにより 1 キャッシュラインが構成されている．512 組のキャッシュラインから構成される場合，容量は 32KB となる．

各段は 4 つの基本ユニットから構成され，右から順に way0，way1，…，way3 を構成する．各 way に対して主記憶が接続されており，異なる way に対して同時に主記憶間転送が可能である．

メモリブロックからの読み出し方法は 2 通りあり，4 ワードを同時読み出して基本ユニットの出口に配置したラッチに格納するか，または，1 ワードを読み出して前述のラッチの 1 つに格納すると同時に同一段の FIFO に書き込むかのいずれかを選択する．前者が機能要件 1 に対応し，後者が機能要件 2 に対応する．

1.4 実装例と動作

1.4.1 SIMD 計算時

図 1.5 に SIMD 計算時の動作を示す．第 1 段では，way2 に配列 B，同 way1 に配列 C，同 way0 に配列 D が写像され，3 つの 4 倍幅ロード命令が連続実行される．4 組の B,C,D 要素は，第 2 段における積和演算に入力され，way3 のメモリブロックに格納される．同時に，次の連続実行に備えて，第 2 段の way2，1，0 には主記憶からプリフェッチをするとともに，第 1 段の way3 に格納された前回の実行結果を主記憶へ書き戻す．本実行モデルでは，命令写像を 1 段ずつ下方にシフトし，プリフェッチしたデータを次段で使用する．メモリブロックの容量制約により連続実行が分断される欠点があるが，実行中に，メモリブロックへの

書き込みと読み出しの速度差を調整する必要がない．

一方，第４段と第５段に，命令写像のシフトが不要な使用方法を示す．主記憶から第４段のメモリブロックに供給しつつ，同メモリから読み出しを行う．また，第４段のメモリブロックに演算結果を格納しつつ，同メモリから主記憶へ書き戻す．各 way に接続される主記憶バスが全て稼働する理想的状態である．メモリブロックへの書き込みと読み出しの速度差を調整する必要があるものの，連続実行が分断される欠点がない．

1.4.2 ステンシル計算時

図 1.6 にステンシル計算時の動作を示す．第１，２，３段の各 way0 のメモリブロックから読み出した各１ワードデータが同一段の FIFO に送信され，段毎に同時に 6 ワードのデータを次段の演算器に供給する．ステンシル計算の場合，メモリブロックを最大限に再利用するために，前述した命令写像のシフト機能を併用する．

1.4.3 キャッシュ機能時

図 1.7 にキャッシュ機能時の動作を示す．汎用 CPU により L2addr に格納された L2 キャッシュ参照要求からアドレスを連続的に取り出し，way3-0 のアドレスバスを使用して tag 比較を行う．L2addr は下方へ伝搬され，全段をパイプライン的に参照する．Tag が一致した way からは，４ワード分のデータを読み出して下方へ伝搬する．なお，各段においては，一般的なセットアソシアティブキャッシュと同様，全ての way は同時動作する．HIT した場合，キャッシュライン全体の読み出しが必要であるため，一度にキャッシュライン全体を読み出すだけのバス幅がない場合，複数サイクルを用いて読み出す．例示の場合は２サイクル連続で読み出しを行うため，L2addr の受け付けは２サイクルに１回となる．また，L2addr を受け付けてから当該キャッシュに HIT した場合のレイテンシは，段数分となる．

なお，L1 キャッシュは L2 に対して inclusive なので，L1 キャッシュのフラッシュ動作は，必ず L2 キャッシュにヒットする．このため，L1 キャッシュ全フラッシュに要する時間は一定である．

1.4.4 グラフ計算時

最後に，図 1.8 にグラフ計算時の動作を示す．Way0 に接続された主記憶から隣接頂点データが第２段のメモリブロックに供給されている．なお，前述のように，必ずしも命令写像をシフトする必要がないグラフ計算の場合，第１段に直接供給しつつ，同時に隣接頂点データを読み出してもよい．第２段において条件判断および主記憶直接参照要求（MMR）を行う．前段から第２段を経由して後続段にデータを伝搬させる場合，MMR のレイテンシに合わせた遅延挿入が必要である．このために，第２段のメモリブロックを FIFO として使用し，レイテンシを吸収している．前段から供給されるデータと，主記憶から読み込んだデータのタイミングを揃えて第３段に供給している．また，第３段からは，さらに主記憶へトランザクションを発行している．本例では，隣接頂点データの読み出しに way0，MMR に way1，トランザクションに way2 の主記憶バスを各々使用しており，主記憶参照のトラフィックが互いに干渉しないようスケジューリングできている．

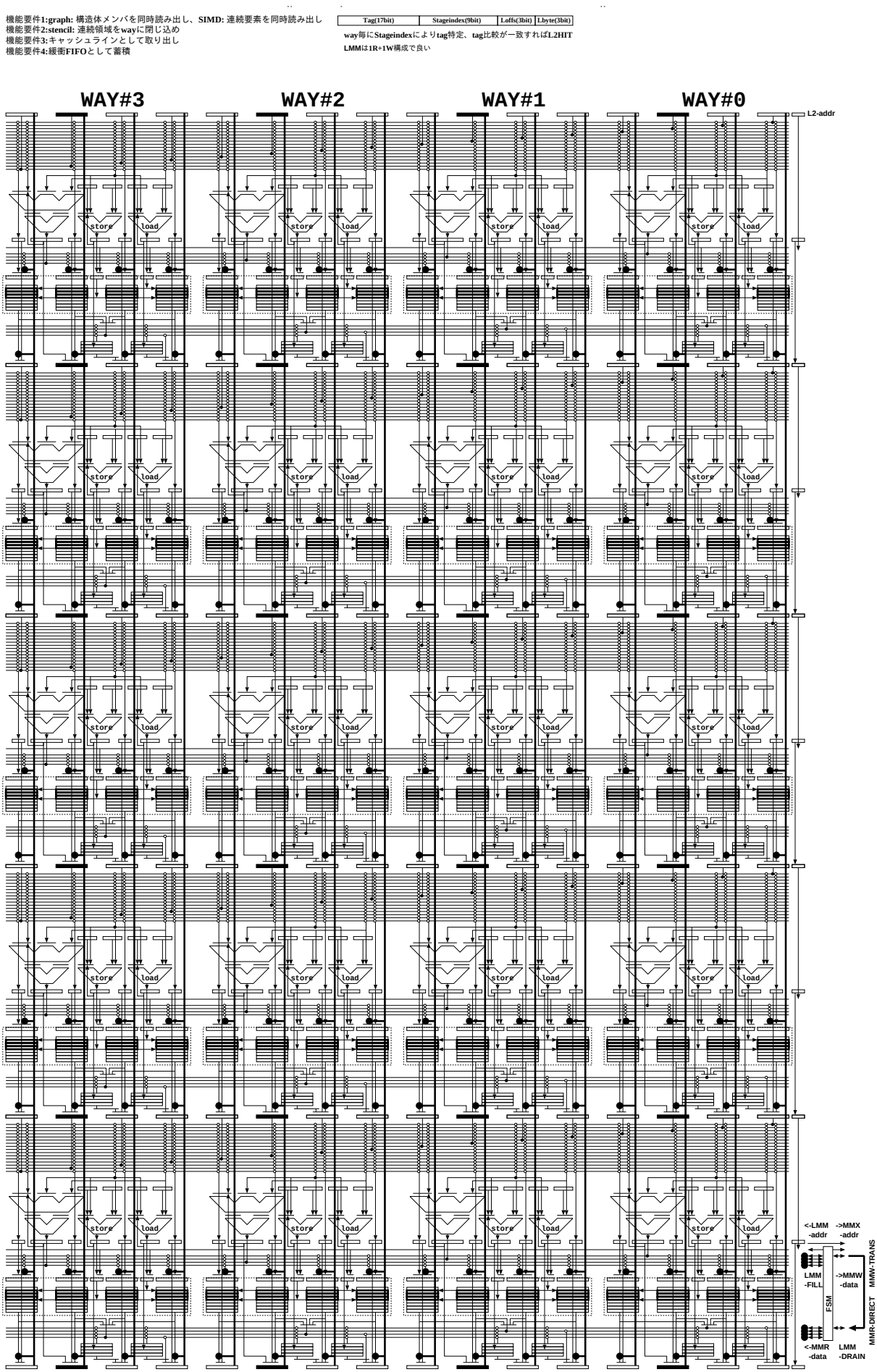


Figure.1.4: Basic structure of EMAX5/L2

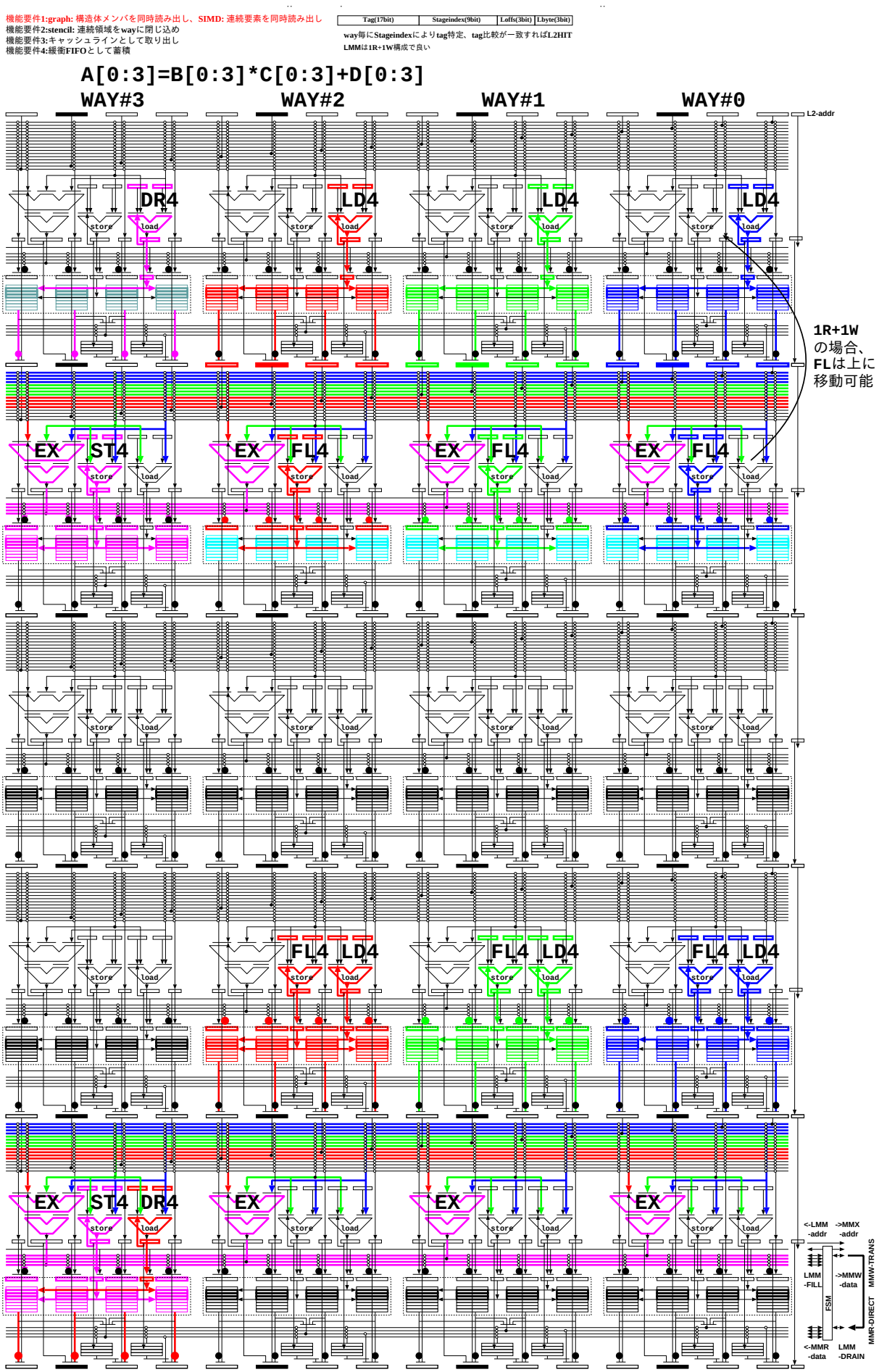


Figure.1.5: Mapping of parallel SIMD

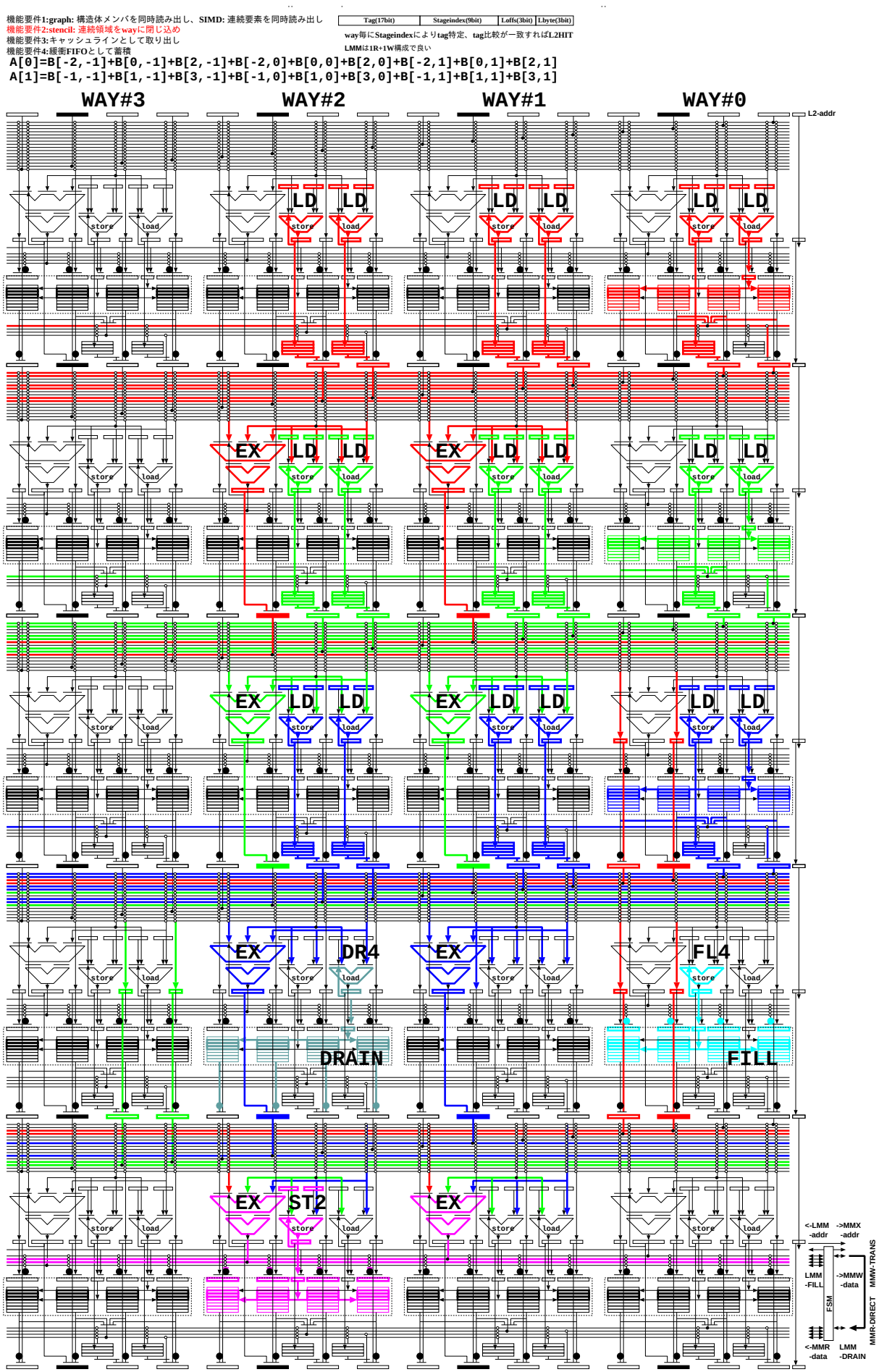


Figure.1.6: Mapping of stencil calculation

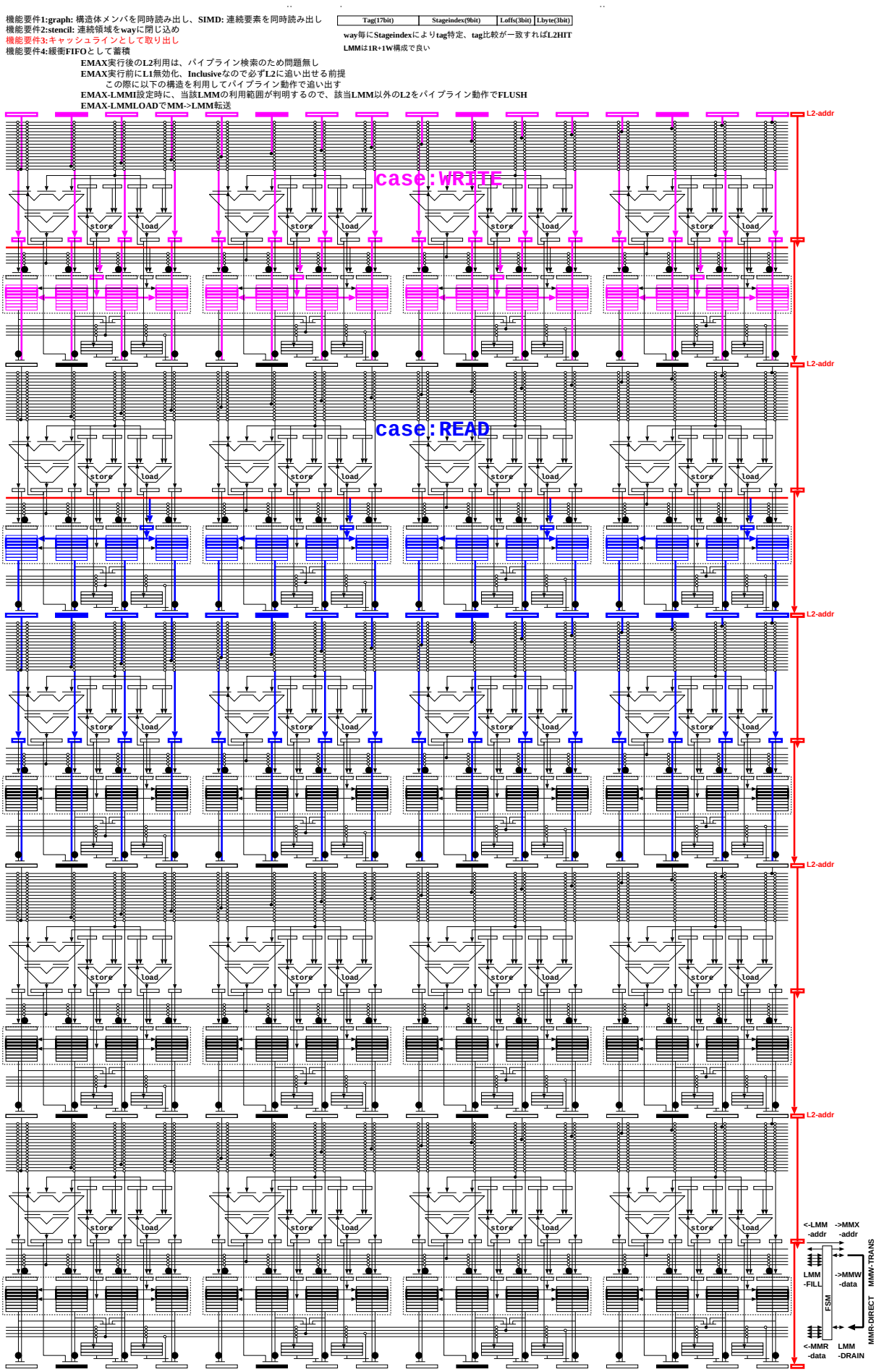


Figure.1.7: Mapping of L2 cache

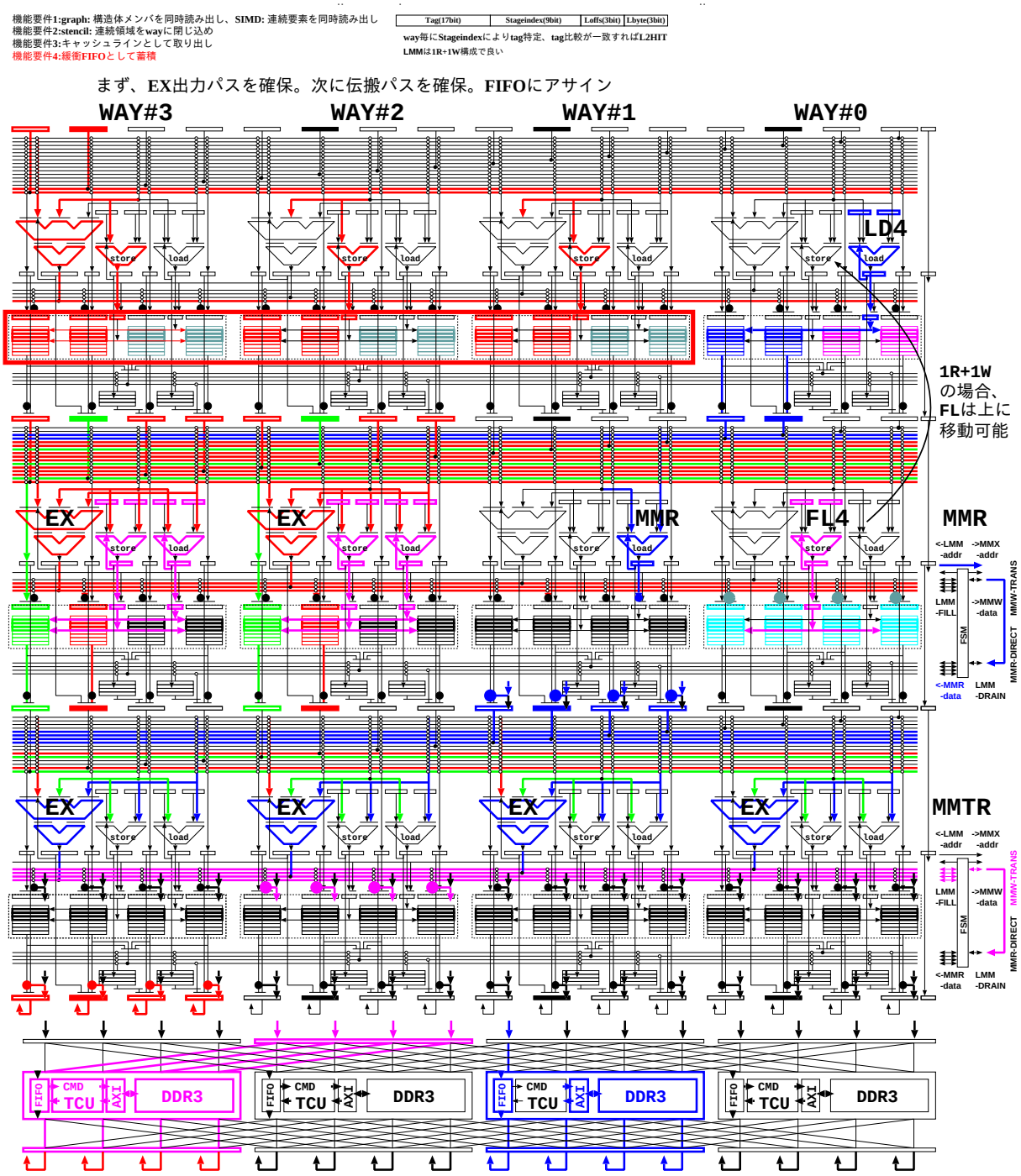


Figure.1.8: Mapping of graph processing

Chapter 2

EMAX5/ZYNQ Hardware

2.1 History of CGRA

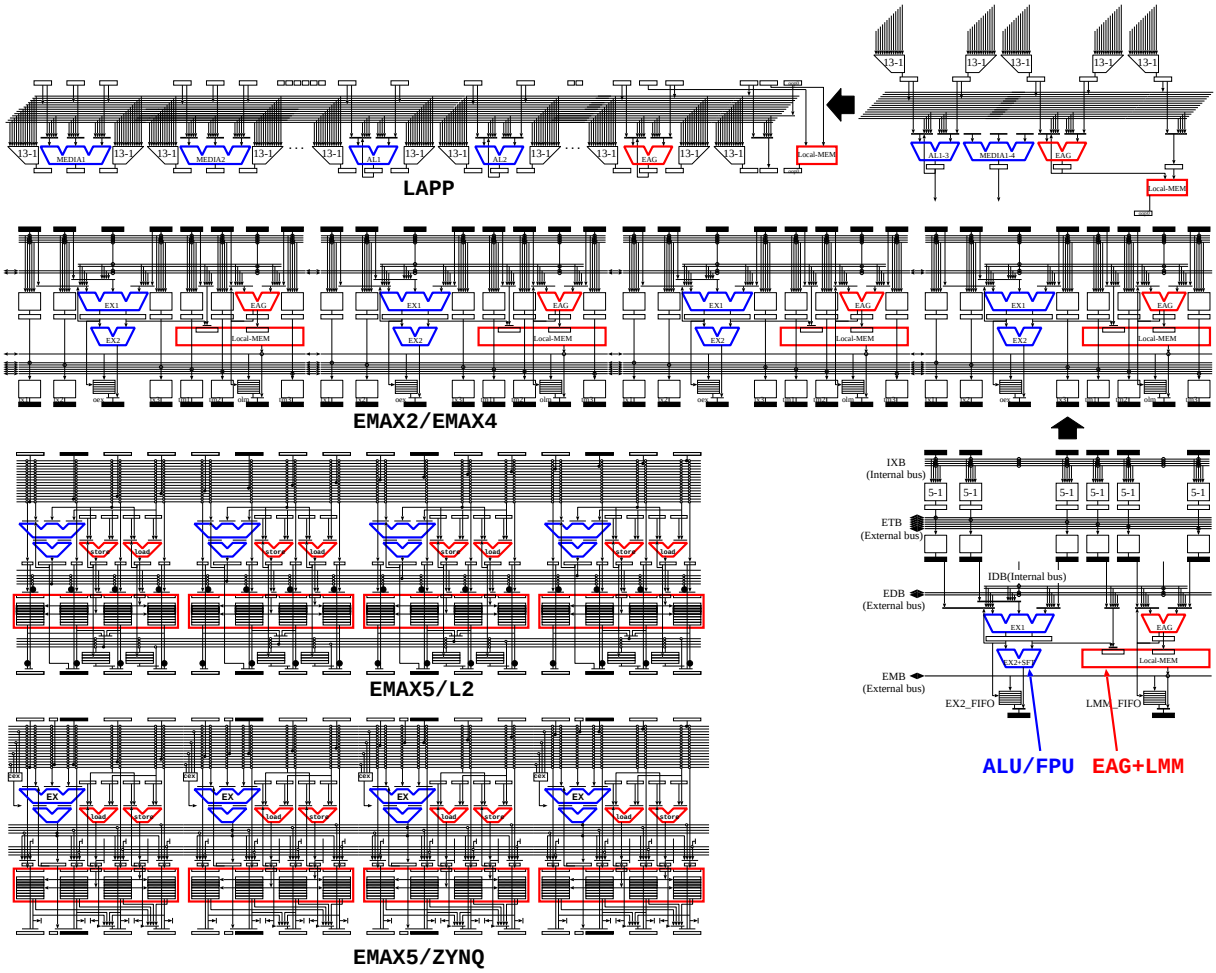


Figure.2.1: History of CGRA

図 2.1 に，当グループが開発してきた CGRA の 1 段分の構造を示す．LAPP は VLIW 互換 CGRA であったものの，段毎の LMM 容量は極めて小さく，次数（1 辺の長さ）が大きなステンシル計算には不向きであった．EMAX2 では，LMM の容量を増やすとともに，各演算器との関連付けを強化し，多くのステンシル計算を写像できるようにした．ただし，段間のレジスタ数が大幅に増加したため，レジスタから演算器出力までに 4 段を要した．EMAX4 では，さらにグラフ処理のためのトランザクション機構を追加した．EMAX5/L2 では，SIMD 演算機構を備えるとともに，外部メモリバスから LMM へのスループットを向上

させ，ALU と LMM の入力レジスタを共用することで段間のレジスタ数を半減し，レジスタから演算器出力までを 2 段とした．EMAX5/ZYNQ では，FIFO 初期化のオーバヘッドを削減するために FIFO を削除し，かわりに，1 つの外部メモリバスから複数 way の LMM へ同時に書き込むバスを追加した．これにより，複数の外部メモリバスを使用して多数の LMM を初期化することが可能となった．

2.2 Overview

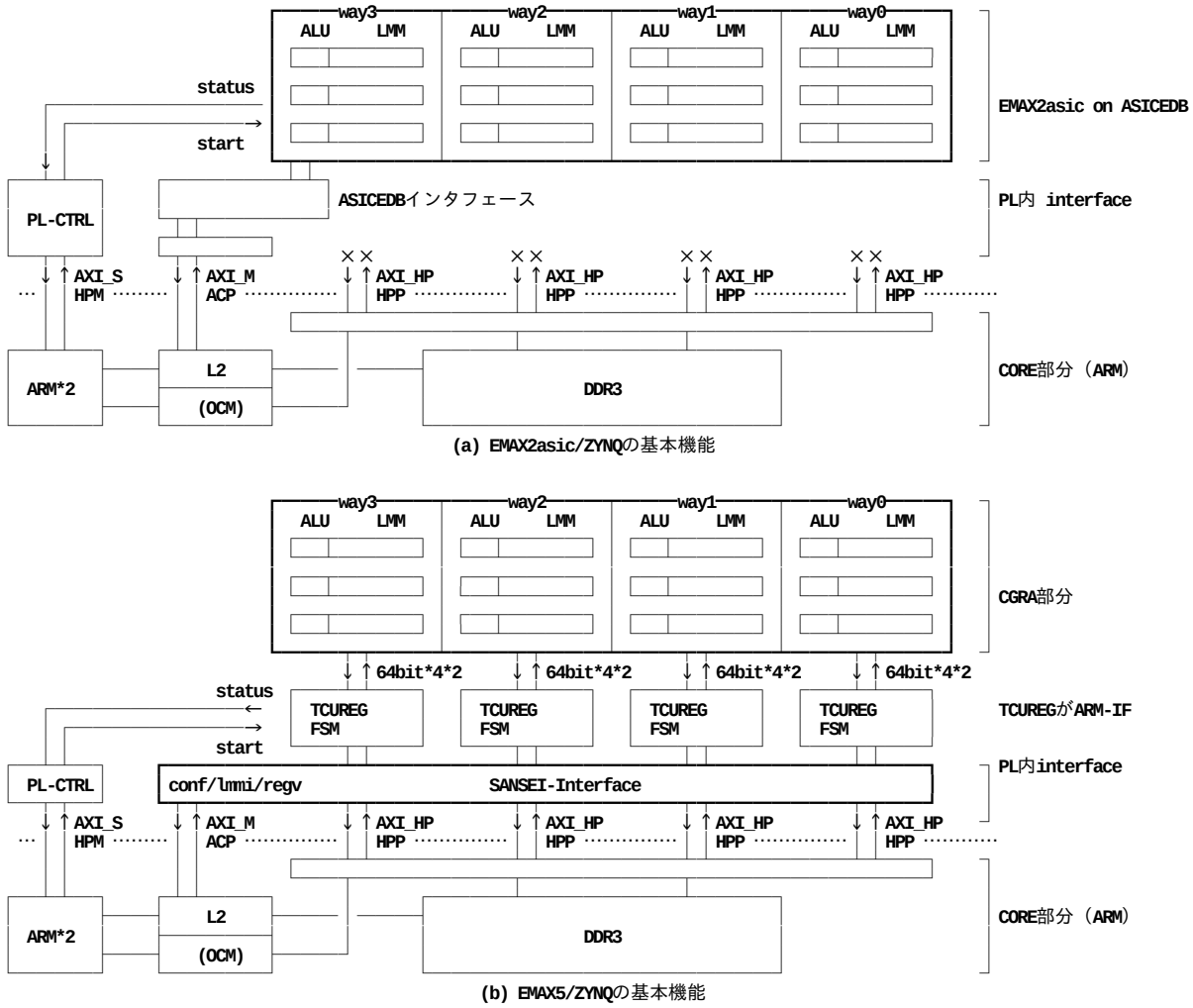


Figure.2.2: Overview of EMAX2asic/ZYNQ and EMAX5/ZYNQ

EMAX5/L2 が高速化対象とする SIMD 計算，ステンシル計算，グラフ計算のうち，SIMD 計算およびステンシル計算は，結果の書き込み先アドレスが連続である．ショートベクトルの場合，書き込み量が L2 キャッシュ容量を超えないものの，ARM が後続処理を引き継ぐことは少ないため，L2 キャッシュへ書き込むメリットは少ない．このため，HPP を使用してメモリ間ネットワークを構築する実行モデルが適している．一方，グラフ計算は，結果の書き込み先アドレスが不連続である．隣接頂点の属性変更など，書き込み量が L2 キャッシュの容量を超える場合でも，ARM が後続処理を引き継ぐことが多く，L2 キャッシュへ書き込んでおく効果が高いと考えられる．このため，ARM により直前に更新される可能性のあるデータおよび出力データは ACP を使用し，頻繁には更新しないデータの読み込みには HPP を使用してメモリ間ネットワークを構築する実行モデルが適している．

EMAX5/ZYNQ は，EMAX5/L2 の基本思想を ZYNQ プラットフォームの HPM（High Performance Master Port），ACP（Cache Coherent Port），および，HPP（High Performance Port）を利用して実装するアーキテクチャである．ただし，EMAX5/L2 が L2 キャッシュ機能をメモリブロックに内蔵するのに

対し，EMAX5/ZYNQ は ACP ポートを利用して ARM の L2 キャッシュを参照する．キャッシュ機能の差異を除くと，EMAX5/ZYNQ の PL（Programmable Logic）部に実装される CGRA 部分は，EMAX5/L2 と同様の構成である．EMAX5/ZYNQ の各 way は，64bit*4 双方向のメモリインタフェースを有し，way 毎の TCUREG（Transaction Control Unit Register）および HPP と接続される．TCU は ARM のソフトウェアとして実装されており，トランザクションに必要な命令列，入力データ，出力データの読み書きは L2 キャッシュを経由して行われる．4 組の HPP は，OCM（On Chip Memory）および DDR3 に接続されているものの，EMAX5/ZYNQ では OCM は使用しない．EMAX5/ZYNQ の起動および終結確認は，ARM から HPM を経由して行われる．

EMAX2asic/ZYNQ（図 2.2(a)）から EMAX4/bsim への変更点は以下の通りである．

- 複数条件の組合せによる条件付実行の追加
- トランザクション機能の追加
- 各段と外部メモリの間のデータバスを 64bit*1 から 32bit*4 に拡張

さらに，EMAX4/bsim から EMAX5/ZYNQ（図 2.2(b)）への変更点は以下の通りである．

- 単精度浮動小数点演算を 32bit*2 演算へ SIMD 化
- 32bit 整数演算を 32bit*2 整数演算へ SIMD 化
- 8/16bit メディア演算を 4/2 倍幅 SIMD から 8/4 倍幅 SIMD へ増強
- 各ユニットのローカルメモリ（LMM）バス幅を 32bit*1 から 64bit*4 に拡張し，LMM-外部メモリ間のスループットを 8 倍に拡張
- 各段と外部メモリの接続を 32bit*4*1 チャネルから 64bit*4*4 チャネルに拡張し，各段-外部メモリ間のスループットを 8 倍に拡張（conf/regv/lmmi 初期化オーバヘッドを 1/8 に削減）
- 外部メモリ直接参照のための Dual-Port-LMM の採用および CGRA 基本構成の変更
- 最大 3 ユニットの LMM から 3 入力 SIMD へデータ供給しつつ LMM に出力できるユニット間バスネットワーク
- ユニットあたりの伝搬レジスタ数を 8 から 4 に削減し，代わりに伝搬レジスタから演算器へはクロスバスイッチを配置
- LMM の出力を水平方向に伝搬する FIFO を削除し，代わりに外部メモリバスから同一段の LMM へブロードキャストする機構を装備（FIFO の初期化が不要）

2.3 UNIT 構成

CGRA に対応するコードは，図 2.3 に示す基本ユニットおよびユニット間ネットワークに関連付けられる．演算機構（EX）と LMM アドレス計算機構（load，store）は独立しているものの，段間レジスタは共有している．EX の入力は前段の出力レジスタ，出力は（A）同一行に属する LMM 固定カラムの指定アドレス位置,（B）同一ユニット内 LMM 全カラムの指定アドレス位置,（C）同一ユニット内出力レジスタのいずれかに接続される．LMM アドレス計算機構は，2 組のアドレス生成器から構成される．アドレス生成器の入力は（E）前段の出力レジスタ（ランダム参照時）,（F）連続アドレス初期値およびオフセットレジスタ（単調参照時）のいずれかに接続される．なお（E）および（F）では専ら主記憶アドレスを使用し，さらに（F）において LMM をバッファとして使用する場合は，バッファの先頭および終端位置を指定するインデックス（主記憶アドレスではない）を使用する．アドレス生成器以外の LMM の入力は（G）前段の出力レジスタ,（I）同一行の演算器出力から LMM 固定カラム,（J）同一ユニットの演算器出力から LMM 全カラム,（M）HPP のいずれかに接続される．また，LMM の出力は（K）カラム毎出力レジスタ,（L）カラム集約後出力レジスタ,（N）HPP のいずれかに接続される．

図 2.3(a) から (j) は，接続の具体例である．LDRQ（図 2.3(a) : load-64bit*4+prefetch）は，blk（0:ブロッキング無し，1:16 回連続参照毎に先頭ポインタ配列を進めるブロッキング，2:32 回連続参照毎に先頭ポインタ配列を進めるブロッキング，3:64 回連続参照毎に先頭ポインタ配列を進めるブロッキング）および len（64bit 単位のバースト長）に従い，主記憶と CGRA の動作速度差を吸収するために，次の実行に必要な主記憶からのデータを LMM にバッファリングしつつ，ロード済のデータを演算器に供給する．接続

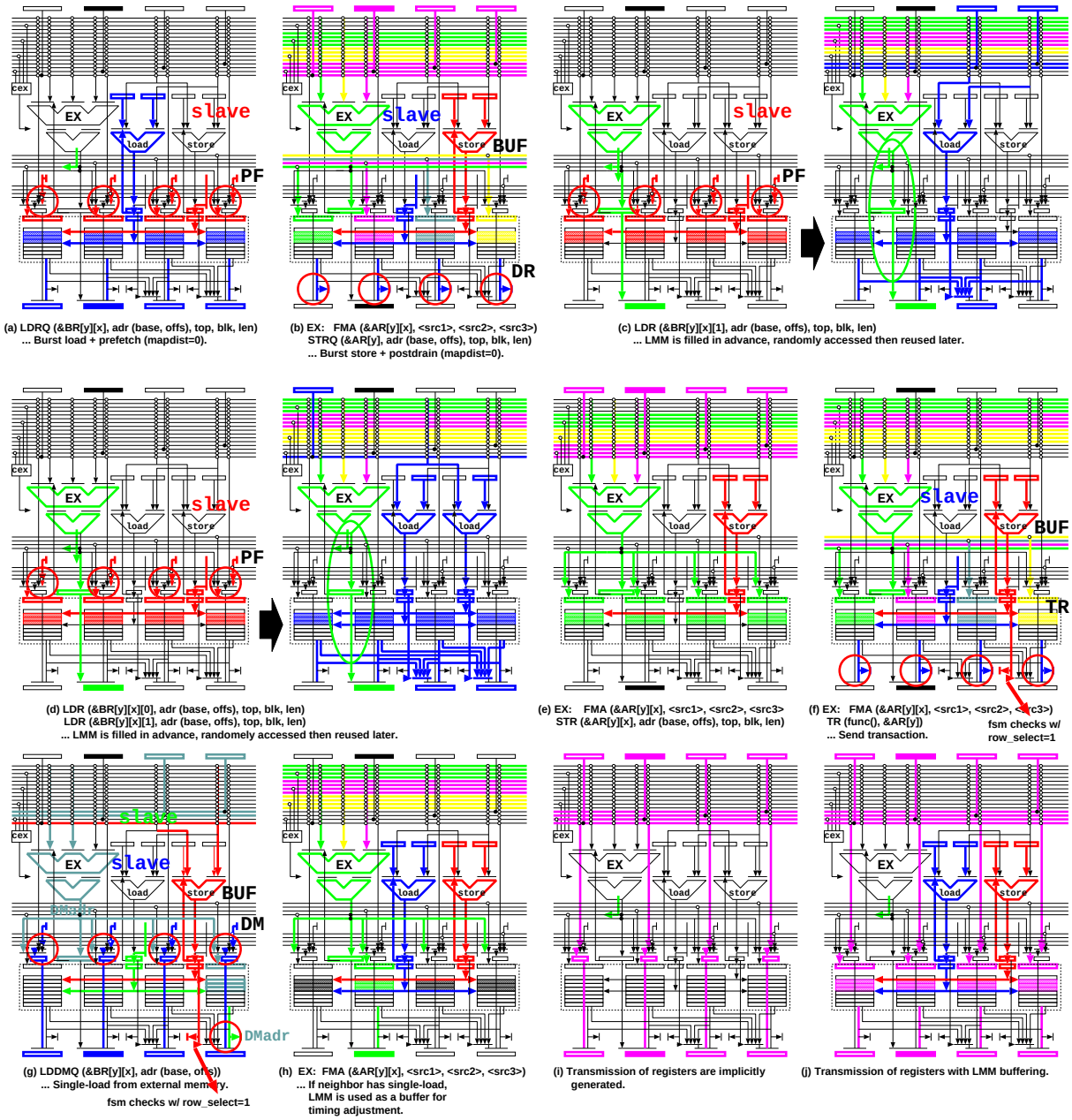


Figure.2.3: Basic structure and function of CGRA

パターンは（F，M，K）となる．主記憶から LMM への書き込み時，後述の FSM が master，ユニットが slave となる．

LDR（load-64bit+prefetch）は，blk および len に従い，主記憶と CGRA の動作速度差を吸収するために，次の実行に必要な主記憶からのデータを LMM にバッファリングしつつ，ロード済のデータを演算器に供給する．接続パターンは（F，M，L）となる．主記憶から LMM への書き込み時，後述の FSM が master，ユニットが slave となる．

LDDMQ（同図 (g)：load-64bit*4-direct-mem）は，外部メモリからの直接ロードに関連付けられる．接続パターンは（E）となる．主記憶の読み出し時，後述の FSM が slave，ユニットが master となる．主記憶アドレス計算は EX1 に写像され，アドレスは LMM の word0 にキューイングされる．LMM 書き込み用には EA0 を使用，FSM は row_select=1 の当該 AXRA（EA0 と同じ値）を監視し，新規アドレスの登録を検知して LMM から主記憶アドレスを取り出し，AXI を使用して主記憶を読み出し，LMWD にデータを送出する．UNIT では，LMWD → TR → BR を経由して 4 ワードを次 stage に送出する．

STRQ（同図 (b)：store-64bit*4+postdrain）は，主記憶と CGRA の動作速度差を吸収するために，演

算器からのデータを LMM にバッファリングしつつ，前回の実行結果を主記憶に転送する．接続パターンは（A，F，I，N）となる．LMM から主記憶への書き込み時，後述の FSM が master，ユニットが slave となる．STR（store-64bit+postdrain）も同様である．store-64bit*4-direct-mem に相当する機能は用意されておらず，後述のトランザクション TR（同図 (f)）により代用する．

LDR（同図 (c)：load-64bit-random）は，LMM からのロードに先立ち，top，blk，len により指定された範囲が予め LMM に格納される．その後，adr() によるランダムアドレス指定に基づき LMM が参照される．接続パターンは（F，M）→（E，L）となる．LDR （同図 (d)：load-64bit-random）のうち，top，blk，len の指定がないものは，同一ユニットに前述の ldr が指定されていることを前提に，LMM を同時に参照する．接続パターンは同様に（F，M）→（E，L）となる．主記憶から LMM への書き込み時，後述の FSM が master，ユニットが slave となる．

STR（同図 (e)：store-64bit-random）は，blk および len の指定に従い，LMM に 64bit が連続的に書き込まれる．その後，LMM から主記憶に len により指定された量のデータがバースト転送される．接続パターンは（B，F，J，N）となる．

TR（同図 (f)：transaction-64bit*4）は，64bit*4 を TCUREG へ転送する際に使用する．ARM に実装される TCU 機能は，64bit*4 に含まれるアドレス情報を使用して，主記憶の該当アドレスに対して更新処理を行う．64bit*4 全てを主記憶に書き込むことはできないため，厳密には，64bit*4 のシングルストアに相当する機能はない．接続パターンは（A, F, I, N）となる．LMM から TCUREG への書き込み時，後述の FSM が slave，ユニットが master となる．演算器出力 4word を LMM にキューイングし，LMM 書き込み用に EA0 を使用する．FSM は row_select=1 の当該 AXRA（EA0 と同じ値）を監視し，TCU の空きと新規アドレスの登録を検知して LMM から 4word を取り出し，TCUREG に格納し，ARM に通知する．ARM は TCUREG の内容を元にトランザクションを実行し，終了したら TCU を空きに戻す．

ところで，以上の基本構成のうち，同図 (g) の写像に際しては，主記憶遅延に対する考慮が必要である．プログラミング時に特殊な記述は必要としないものの，写像時には，同図 (g) と同一行に含まれるユニットでは，同図 (h) のように LMM を利用した遅延同期機構が活性化される．接続パターンは（B，F，I，K）となる．同様に，レジスタ間伝搬が必要である場合，空きレジスタを使用して，同図 (i)（遅延同期無）や同図 (j)（遅延同期有）のように写像される．

図 2.4 に各ユニットの詳細構造を示す．特に，前述の (a) および (g) を含む行では，(a) および (g) により生じるユニット間のタイミングのずれを同一行内で吸収するために，隣接 LMM 間の協調動作が必要である．連続動作の長さを LMM がタイミングのずれを必ず吸収できる範囲内とすることにより，同一行内で吸収できなくなった際に必要となる，前段に対して動作を一時停止させるインタフェースやバッファを省略できる構成としている．

2.3.1 FSM

図 2.5 に FSM の構成を示す．FSM は，TCUREG を内蔵し，CGRA の外部インタフェースと主記憶インタフェース（AXI）を相互接続するブロックである．CGRA 内の LMM に対して master として読み書きするとともに，AXI に対しても master として動作する．CGRA からのトランザクション要求（TCUreq）は，FSM 内の TCUREG に送信される．

2.3.2 TCU

TCU に対応するコードは，図 2.6 の構成に関連付けられる．TCU の第 1 引数は，使用するトランザクションに対応する関数の先頭アドレスであり，一連の命令列が主記憶から取り出されて ARM により実行される．異なる関数を使用する複数のトランザクションを同時に実行でき，マルチスレッディング機構により主記憶遅延が隠蔽される．なお，TCU がプログラムの高速化に寄与するためには，TCU を主記憶に近い記憶階層に配置することが望ましいものの，小規模モデルでは，TCU を省略し，FIFO を経由して ARM にトランザクションを送信し，ソフトウェアにより処理する実装も可能である．

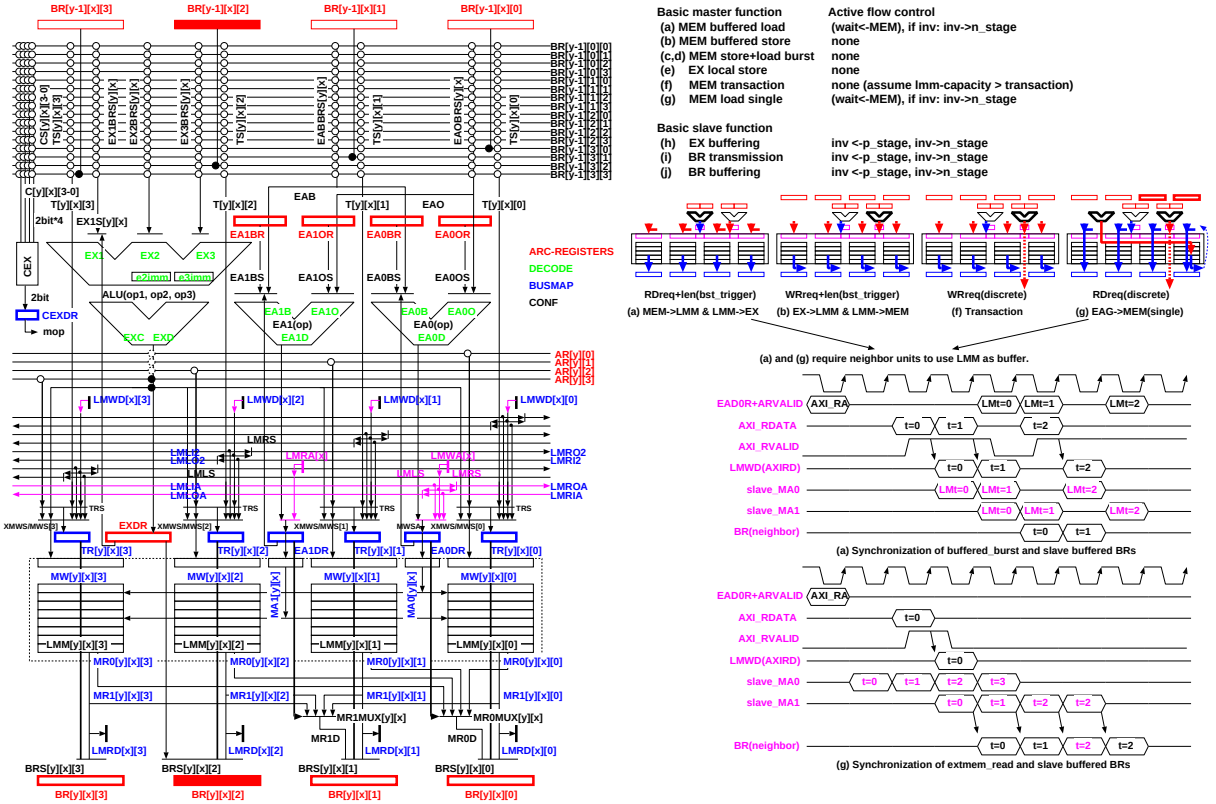


Figure.2.4: Detailed structure of CGRA

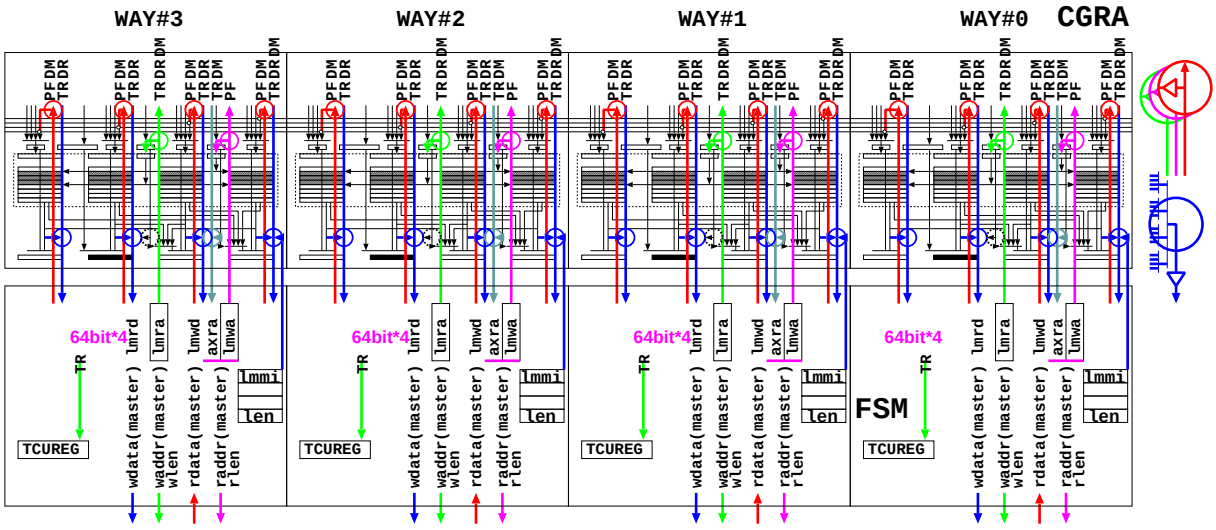


Figure.2.5: Basic structure of FSM

2.4 ZYNQ-EMAX5 物理インタフェース

図 2.7 に，FSM と ZYNQ のインタフェースを示す．各 way の AXI-master(Read) インタフェースは物理アドレスを使用し，ACP/HPP 切替えビット（m_acphpp）に応じて，ACP または HPP へ動的に接続される．0 の場合 ACP へ接続され，専ら conf，lmmi および regv 参照に使用する．1 の場合 HPP へ接続され，専ら LMM 写像領域の参照に使用する．ACP の場合，全 way からのリクエストは先着順にキューイングされ，HPP の場合，way 毎の HPP にキューイングされる．AXI-master(Write) インタフェースは，常に way 毎の HPP に接続される．EMAX のリセット/起動/状態確認には，AXI-slave(HPM) を使用する．

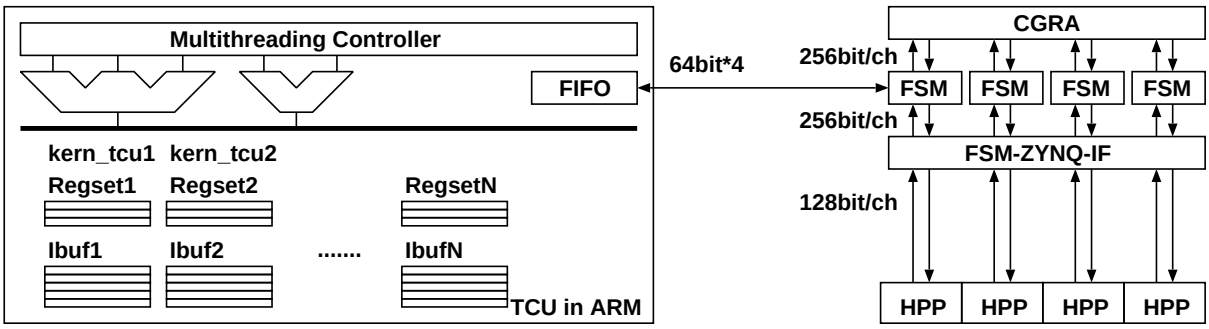


Figure.2.6: Basic structure of TCU

FSM Master (Read) *4 out wire m_acphpp → in wire m_arready ← out wire m_arvalid → out wire [63:0] m_araddr → out wire [7:0] m_arlen → in wire m_rvalid ← in wire [255:0] m_rdata ← in wire m_rlast ← out wire m_rready → ※m_acphppの値によりACP/HPP自動切替え (stack-frameの場合,ACPへqueueing) m_acphpp=0:ACP+物理addr(lmmi,regv) m_acphpp=1:HPP+物理addr(conf,data)	4 : 1	**ZYNQ HPP:Master (Read) *4** ← → → → [7:0]*2 m_arlen ← ← [127:0] m_rdata ← → **ZYNQ ACP:Master (Read) *1** ← → → → [7:0]*2 m_arlen ← ← [127:0] m_rdata ← →
FSM Master (Write) *4 in wire m_awready ← out wire m_awvalid → out wire [63:0] m_awaddr → out wire [7:0] m_awlen → out wire [31:0] m_wstrb → out wire m_wvalid → out wire [255:0] m_wdata → out wire m_wlast → in wire m_wready ←		**ZYNQ HPP:Master (Write)** ← → → → [7:0]*2 m_awlen → [15:0] m_wstrb → → [127:0] m_wdata → ←
FSM Slave (Read) out reg s_arready → in wire s_arvalid ← in wire [6:0] s_araddr ← out reg s_rvalid → out reg [63:0] s_rdata → in wire s_rready ←		**ZYNQ HPM:Slave (Read)** → ← ← → → [63:0] m_rdata ←
FSM Slave (Write) out reg s_awready → in wire s_awvalid ← in wire [6:0] s_awaddr ← in wire [7:0] s_wstrb ← in wire s_wvalid ← in wire [63:0] s_wdata ← out reg s_wready →		**ZYNQ HPM:Slave (Write)** → ← ← ← [7:0] s_wstrb ← ← [63:0] s_rdata →

Figure.2.7: ZYNQ-EMAX5 物理インタフェース

2.5 ZYNQ-EMAX5論理インタフェース

2.5.1 Configuration data

The binary data of the instructions is not directly sent to each unit. Each unit requires lower level information representing input signals for all selectors as shown in Figure 2.8, so that no complicated instruction decoder is required on each unit. Such binary translation from the instructions to the configuration data is performed by EMAX5 compiler (conv-c2b).

2.5.2 LMM information

Unlike the configuration data, LMM information should be transmited to EMAX5 every time before starting execution. LMM information shown in Figure 2.9 are calculated by EMAX5 compiler and transmitted to EMAX5 with other source data. For concurrent execution of EMAX in multi-threading, lmmi should be located in different area.

2.5.3 Initial values of registers

Unlike the configuration data, the initial values of registers should be transmited to EMAX5 every time before starting execution. The initial values of registers shown in Figure 2.10 are calculated by EMAX5 compiler and transmitted to EMAX5 with other data. For concurrent execution of EMAX in multi-threading, regv should be located in different area.

2.6 初期化および起動手順

アプリケーション起動時に行うべき EMAX 初期化手順 sysinit() は次の通りである．最初に呼び出すべき emax5_open() では，EMAX_DEVNAME を open() し，3 つの物理アドレス領域を仮想アドレスへ写像する．EMAX のリセット/起動/状態確認に使用する制御レジスタは，物理アドレス（REG_BASE_PHYS）を mmap() により仮想アドレス（emax_info.reg_mmap）に写像して使用する（4KB 境界，CPU キャッシュオフ）．EMAX 処理対象データ（LMM 格納データ）は，DDR4 物理上位 2GB（HPP_BASE_PHYS）を mmap() により仮想アドレス（emax_info.hpp_mmap）に写像して使用する（4KB 境界，CPU キャッシュオフ）．conf，lmmi，regv の各領域は，ioctl() により取得した物理アドレス（emax_info.acp_phys）を mmap() により仮想アドレス（emax_info.acp_mmap）に写像して使用する（4KB 境界，CPU キャッシュオン）．なお，emax5_open() は，EXI_S_ADDR_RESET に 1 を書き込むことにより EMAX をリセットする．

```
sysinit();
{
  if (emax5_open() == NULL) {
    error;
  acp_conf = (Ull*)(emax_info.acp_mmap); /* 8KB * 256sets */
  acp_lmmi = (Ull*)(emax_info.acp_mmap + 0x200000);
  acp_regv = (Ull*)(emax_info.acp_mmap + 0x304000);
}
```

```
struct conf { /* final configuration info. for EMAX5-CGRA */
  struct cdw0 { /* select EXE-in */
    Ull  op1    :  6; /* alu_opcd */
    Ull  op2    :  3; /* logical_opcd */
    Ull  op3    :  3; /* sft_opcd */
    Ull  ex1brs :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    Ull  ex1s   :  1; /* 0:ex1brs, 1:exdr(self-loop) */
    Ull  ex1exp :  2; /* 0:--, 1:B5410, 2:B7632, 3:H3210 */
    Ull  ex2brs :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    Ull  ex2exp :  2; /* 0:--, 1:B5410, 2:B7632, 3:H3210 */
    Ull  ex3brs :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    Ull  ex3exp :  2; /* 0:--, 1:B5410, 2:B7632, 3:H3210 */
    Ull  e2is   :  2; /* 0:e2imm, 1:ex2, 2:ex3 */
#define E3IMMBITS  6
    Ull  e3imm  : E3IMMBITS;
    Ull  e3is   :  1; /* 0:e3imm, 1:ex3 */
    Ull  dmy00  : 24;
  } cdw0;

  struct cdw1 { /* select CEX-in and EAG-in */
    Ull  cs0    :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    Ull  cs1    :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    Ull  cs2    :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    Ull  cs3    :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    Ull  cex_tab: 16; /* c3.c2.c1.c0 の組合せ (cop=NOP の場合,ffff) */
                      /* 1111,1110,1101,1100,....,0001,0000 の各々に 0/1 を割り当てた 16bit を指定 */
    Ull  ea0op  :  5; /* mem_opcd */
    Ull  ea0bs  :  2; /* 0:ea0br, 1:ea0dr(ea0br+self-loop), 2:eabbrs, 3:ea0dr(eabbrs+self-loop) */
    Ull  ea0os  :  1; /* 0:ea0or, 1:eaobrs */
    Ull  ea0msk :  4; /* 15:64bit, 13:word1, 12:word0, 11-8:half3-0, 7-0:byte7-0 of offset */
    Ull  ea1op  :  5; /* mem_opcd */
    Ull  ea1bs  :  2; /* 0:ea1br, 1:ea1dr(ea1br+self-loop), 2:eabbrs, 3:ea1dr(self-loop) */
    Ull  ea1os  :  1; /* 0:ea1or, 1:eaobrs */
    Ull  ea1msk :  4; /* 15:64bit, 13:word1, 12:word0, 11-8:half3-0, 7-0:byte7-0 of offset */
    Ull  eabbrs :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
    Ull  eaobrs :  4; /* 0:br0_0, 1:br0_1, ... 15:3_3 */
  } cdw1;

  struct cdw2 { /* select TR/BR-in */
    Ull  lmls   :  1; /* 0:lmwad, 1:lmri */
    Ull  lmrs   :  1; /* 0:lmwad, 1:lmli */
    Ull  ts0    :  4; /* 0:br0_0, 1:br0_1, ... 15:br3_3 */
    Ull  ts1    :  4; /* 0:br0_0, 1:br0_1, ... 15:br3_3 */
    Ull  ts2    :  4; /* 0:br0_0, 1:br0_1, ... 15:br3_3 */
    Ull  ts3    :  4; /* 0:br0_0, 1:br0_1, ... 15:br3_3 */
    Ull  trs0   :  2; /* 0:exdr0, 1:lmwd0, 2:ts0 */
    Ull  trs1   :  2; /* 0:exdr1, 1:lmwd1, 2:ts1 */
    Ull  trs2   :  2; /* 0:exdr2, 1:lmwd2, 2:ts2 */
    Ull  trs3   :  2; /* 0:exdr3, 1:lmwd3, 2:ts3 */
    Ull  mwsa   :  2; /* 0:off, 1:lmlia,2:lmwa,3:lmria  *//* for STATUS_EXEC+lmp */
    Ull  mws0   :  3; /* 0:exdr,1:exdr0,2:ts0, 3:lmli0, 4:lmwd0, 5:lmri0 */
    Ull  mws1   :  3; /* 0:exdr,1:exdr1,2:ts1, 3:lmli1, 4:lmwd1, 5:lmri1 */
    Ull  mws2   :  3; /* 0:exdr,1:exdr2,2:ts2, 3:lmli2, 4:lmwd2, 5:lmri2 */
    Ull  mws3   :  3; /* 0:exdr,1:exdr3,2:ts3, 3:lmli3, 4:lmwd3, 5:lmri3 */
    Ull  brs0   :  2; /* 0:off, 1:mr10, 2:tr0, 3:mr0  */
    Ull  brs1   :  2; /* 0:off, 1:mr11, 2:tr1, 3:mr1  */
    Ull  brs2   :  2; /* 0:off, 1:mr12, 2:tr2, 3:exdr */
    Ull  brs3   :  2; /* 0:off, 1:mr13, 2:tr3         */
    Ull  xmws   :  2; /* 0:off, 1:lmli, 2:lmwd,3:lmri *//* for STATUS_LOAD */
    Ull  mapdist:  6;
    Ull  dmy20  :  8;
  } cdw2;

  struct cdw3 { /* e2 immediate */
    Ull  e2imm  : 64;
  } cdw3;
} conf[EMAX_WIDTH][EMAX_DEPTH];
  /* 4dwords/unit costs 1cycle/unit: 4-parallel conf costs 1cycle/stage */
```

Figure.2.8: Low-level configuration data

```
volatile struct lmmi { /* final FSM configuration for EMAX5-CGRA */
  Ull v    : 1; /* valid */
  Ull rw   : 1; /* 0:load(mem->lmm), 1:store(lmm->mem) */
  Ull f    : 1; /* load: 0:reuse LMM as possible, 1:force read */
                /* store:0:none,                 1:force read */
  Ull p    : 1; /* 0:normal, 1:prefetch/drain */
  Ull bcas : 4; /* column-bitmap for lmm broadcasting: slave の row_lmm_en に使用 */
  Ull copy : 1; /* lmmi 既 load 検査に使用. bcas_master は使用しない */
  Ull blk  : 2; /* 0:inf, 1:16, 2:32, 3:64 width*block=page(burst)_size(bytes) */
  Ull dmy  : 5; /* dummy */
  Ull len  :16; /* dwords of current stream (dwords) */
  Ull ofs  :32; /* lmp/lmd offset for f=0,p=1,mapdist=0 */
  Ull top  :64; /* top of current stream / TCU function() */
} lmmi[EMAX_WIDTH][EMAX_DEPTH];
/* 2dwords/unit costs 0.5cycle/unit: 4-parallel conf costs 0.5cycle/stage */

● lmmi 指示ルール (copy from conv-c2b/emac5.c)                    lmmi-loc  v  top  blk  len  rw  f  p
LD w/ f=0,ptop==NL gen cur(lmr) and reuse LMM. (lmr/EMAX4)           curr  1  top  blk  len   0  0  0
LD w/ f=1,ptop==NL gen cur(lmf) and !reuse LMM. (lmf/EMAX4)          curr  1  top  blk  len   0  1  0
LD w/ f=0,ptop!=NL gen cur(lmr) and next(lmp). mdist!=0              curr  1  top  blk  len   0  0  0
                                                                  c+dist  1 ptop  blk  len   0  0  1
LD w/ f=0,ptop!=NL gen cur(lmr) and next(lmp). mdist==0 ofs=ptop-top curr  1  top  blk  len   0  0  1
                                                                 p=1 の場合,pref-addr は常に lmmi.top+ofs
LDDMQ set f=1,p=1 in lmmc automatically                             curr  1  top  -    -     0  1  1
*************************************************************************************************
ST w/ f=0,ptop==NL gen cur(lmw) and reuse+wback LMM. (lmw/EMAX4)     curr  1  top  blk  len   1  0  0
ST w/ f=1,ptop==NL gen cur(lmx) and !reuse+wback LMM. (lmx/EMAX4)    curr  1  top  blk  len   1  1  0
ST w/ f=0,ptop!=NL gen cur(lmw) and prev(lmd). mapdist!=0            curr  1  top  blk  len   1  0  0
                                                                  c-dist  1 ptop  blk  len   1  0  1
ST w/ f=0,ptop!=NL gen cur(lmw) and prev(lmd). mdist==0 ofs=ptop-top curr  1  top  blk  len   1  0  1
                                                                p=1 の場合,drain-addr は常に lmmi.top+ofs
TR set f=1,p=1 in lmmc automatically                                curr  1  top  -    -     1  1  1

LDR(f=0||p=0) mapdist!=0 lmmo.top（演算済）は前回までに LOAD 済
  STATUS_LOAD 時, (lmmc==lmr かつ lmmo!=lmmc) の場合または, lmmc==lmf/lmx の場合, LOAD
  STATUS_EXEC 時, (lmmc==lmp かつ lmmo!=lmmc) の場合, LOAD （アドレス一致を厳密に検査）

LDR(f=0&&p=1) mapdist==0 lmmo.top（演算済）と lmmo.top+ofs（演算未）は前回までに LOAD 済
  LDR top=A, len=L, f=1, ptop=(A+) 符号付 ofs
  STATUS_LOAD 時, lmmo.top+ofs と lmmc.top を比較. 不一致なら lmmc.top を load.
  STATUS_EXEC 時,                                    lmmc.top+ofs を load. (lmp と同じ)

STR(f=0||p=0) mapdist!=0 lmmo.top（更新済 dt=1）は前回までに STORE 済. ただし書き込みが無い dirty=0 は drain 不
要とし区別
  STATUS_DRAIN 時,(lmmc!=lmd かつ lmmo==lmw) の場合または, lmmo==lmx の場合, dirty なら lmmc.top を drain.
  STATUS_EXEC 時, (lmmc==lmd)    (lmmo!=lmmc の場合 drain 箇所不一致だが)    dirty なら lmmc.top を drain.

STR(f=0&&p=1) mapdist==0 lmmo.top（更新済）と lmmo.top+ofs（追出済）は前回までに STORE 済で dirty=1
  STR top=A, len=L, f=1, ptop=(A+) 符号付 ofs
  STATUS_DRAIN 時, lmmo.top!=lmmc.top+ofs かつ dirty なら lmmo.top を drain.
  STATUS_EXEC 時,                              dirty なら lmmc.top+ofs を drain. (lmd と同じ)
                                               ただし, drain 後でも dirty=1 は維持
```

Figure.2.9: LMM information

```
volatile struct regv { /* final register values for EMAX5-CGRA */
  Ull ea0br; /* EA0 for write Base   addr */
  Ull ea0or; /* EA0 for write Offset addr */
  Ull ea1br; /* EA1 for read  Base   addr */
  Ull ea1or; /* EA1 for read  Offset addr */
  Ull br[UNIT_WIDTH]; /* output of unit */
} regv[EMAX_WIDTH][EMAX_DEPTH];
  /* 8dwords/unit costs 2cycle/unit: 4-parallel conf costs 2cycle/stage */
```

Figure.2.10: Initial values of registers

```
volatile struct emax_info {
  Ull  reg_phys;     // kern-phys
  Ull  reg_vadr;     // not used
  Ull  reg_mmap;     // user-virt Contiguous 256K register space
  Ull  hpp_phys;     // kern-phys
  Ull  hpp_vadr;     // not used
  Ull  hpp_mmap;     // user-virt Contiguous 2GB space in high-2GB space
  Ull  acp_phys;     // kern-phys
  Ull  acp_vadr;     // not used
  Ull  acp_mmap;     // user-virt Contiguous 4MB space in low-2GB space
             // 0x1fffff-0x000000: for conf[ 8KB] * 256sets
             // 0x2fffff-0x200000: for lmmi[ 4KB] * 1set
              // 0x3fffff-0x300000: for regv[16KB] * 1set
  int driver_use_1;
  int driver_use_2;
} emax_info;

#define ACP_CONF_MAX 256
volatile Ull  acp_conf; /* acp_mmap+0x000000+8KB*i (L1miss*1 回のため ACP で OK) */
volatile Ull  acp_lmmi; /* acp_mmap+0x200000        L1=32KB */
volatile Ull  acp_regv; /* acp_mmap+0x304000        L1=32KB */
volatile struct {
  int v;   /* 0:acp_conf is empty, 1:copied from text to acp_conf */
  Ull top; /* conf_address in text-segment */
} acp_conf_tab[ACP_CONF_MAX];

#define EMAX_DEVNAME "/dev/emax5_zynq_drv"
int emax_fd;

struct emax_ioctlbuf {
  Ull phys_addr;
  Ull virt_addr;
} emax_ioctlbuf;

#define REG_BASE_PHYS    0x0000000080000000LL ... fixed
#define REG_BASE_SIZE    0x0000000000040000LL ... 256KB
#define HPP_BASE_PHYS    0x0000000800000000LL ... fixed
#define HPP_BASE_SIZE    0x0000000080000000LL ... 2GB
#define ACP_BASE_SIZE    0x0000000000400000LL ... 4MB

#define EXI_S_ADDR_RESET 0x0000000000030000LL
#define EXI_S_ADDR_CONF  0x0000000000030010LL
#define EXI_S_ADDR_LMMI  0x0000000000030018LL
#define EXI_S_ADDR_REGV  0x0000000000030020LL
#define EXI_S_ADDR_OFFS  0x0000000000030028LL
#define EXI_S_ADDR_MODE  0x0000000000030030LL
#define EXI_S_ADDR_STAT  0x0000000000030040LL

emax5_open()
/* HPM を経由する制御レジスタにリセット送出 */
/* HPP を経由する画像メモリを仮想空間に写像 */
/* ACP を経由する conf/lmmi/regv 空間を仮想空間に写像 */
{
  if ((emax_fd = open(EMAX_DEVNAME, O_RDWR|O_SYNC )) == -1) {
    printf("emax_open(): Invalid EMAX_DEVNAME: '%s'\n", EMAX_DEVNAME);
    return (NULL);
  }

  // mmap(cache-off) 4KB aligned
  emax_info.reg_phys = REG_BASE_PHYS;
  emax_info.reg_mmap = (Ull)mmap(0,REG_BASE_SIZE,PROT_READ|PROT_WRITE,MAP_SHARED,emax_fd,REG_BASE_PHYS);
  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_MODE)  = 0LL; // ★★★ CLEAR COMMAND
  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_RESET) = 1LL; // ★★★ RESET EMAX5

  // mmap(cache-off) 4KB aligned
  emax_info.hpp_phys = HPP_BASE_PHYS;
  emax_info.hpp_mmap = (Ull)mmap(0,HPP_BASE_SIZE,PROT_READ|PROT_WRITE,MAP_SHARED,emax_fd,HPP_BASE_PHYS);

  // mmap(cache-on)  4KB aligned
  emax_ioctlbuf.phys_addr = 0x0; // phys
  emax_ioctlbuf.virt_addr = 0x0; // virt
  ioctl(emax_fd, EMAX_GET_ACPMEM, &emax_ioctlbuf);//in driver:virt=(Ull*)kmalloc(ACP_MEM_SIZE(<=4MB),GFP_KERNEL);
  emax_info.acp_phys = emax_ioctlbuf.phys_addr;   //in driver:phys=(Ull)virt_to_phys(buf);
  emax_info.acp_mmap = (Ull)mmap(0,ACP_BASE_SIZE,PROT_READ|PROT_WRITE,MAP_SHARED,emax_fd,emax_ioctlbuf.phys_addr);

  return (emax_fd);
}
```

EMAX の起動手順 emax5_start() は次の通りである．引数 conf，lmmi，regv はアプリケーションの仮想アドレスである．起動と状態監視に使用する制御レジスタを表 2.1 に示す．制御レジスタのうち，CONF，LMMI，REGV には物理アドレスを使用するため，emax_info 内の情報を使用してアドレス変換を行う．LMM 格納データ参照に必要な仮想→物理アドレス変換のためのオフセット情報 OFFS も同様である．EMAX が lmmi 内の仮想アドレスを用いて HPP にアドレスを送出する際，OFFS が加算され，物理アドレスとして HPP に送出される．また，STAT レジスタにより EMAX の動作完了を確認する．

EMAX コンパイラは，前述の conf，lmmi および regv を acp_mmap 領域（R/W）に配置するコードを生成する．EMAX が conf，lmmi および regv を取得する際は ACP を経由して L2 キャッシュを参照し，LMM 格納データを取得する際は HPP を経由して直接主記憶を参照する．すなわち，図 2.2(b) には，各 way 毎に，領域別に ACP または HPP のいずれかを選択するポート指定ビット（m_acphpp）が含まれる．

```
emax5_start(Ull conf, Ull lmmi, Ull regv)
{
  volatile Ull emax_status;

  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_CONF) = emax_info.acp_phys+(conf-emax_info.acp_mmap);
  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_LMMI) = emax_info.acp_phys+(lmmi-emax_info.acp_mmap);
  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_REGV) = emax_info.acp_phys+(regv-emax_info.acp_mmap);
  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_OFFS) = emax_info.hpp_phys - emax_info.hpp_mmap;
  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_MODE) = 1LL;
  do {
    emax_status = *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_STAT);
  } while (emax_status != STATUS_IDLE);
}

emax5_drain_dirty_lmm()
{
  volatile Ull emax_status;

  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_OFFS) = emax_info.hpp_phys - emax_info.hpp_mmap;
  *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_MODE) = 2LL;
  do {
    emax_status = *(Ull*)(emax_info.reg_mmap+EXI_S_ADDR_STAT);
  } while (emax_status != STATUS_IDLE);
}
```

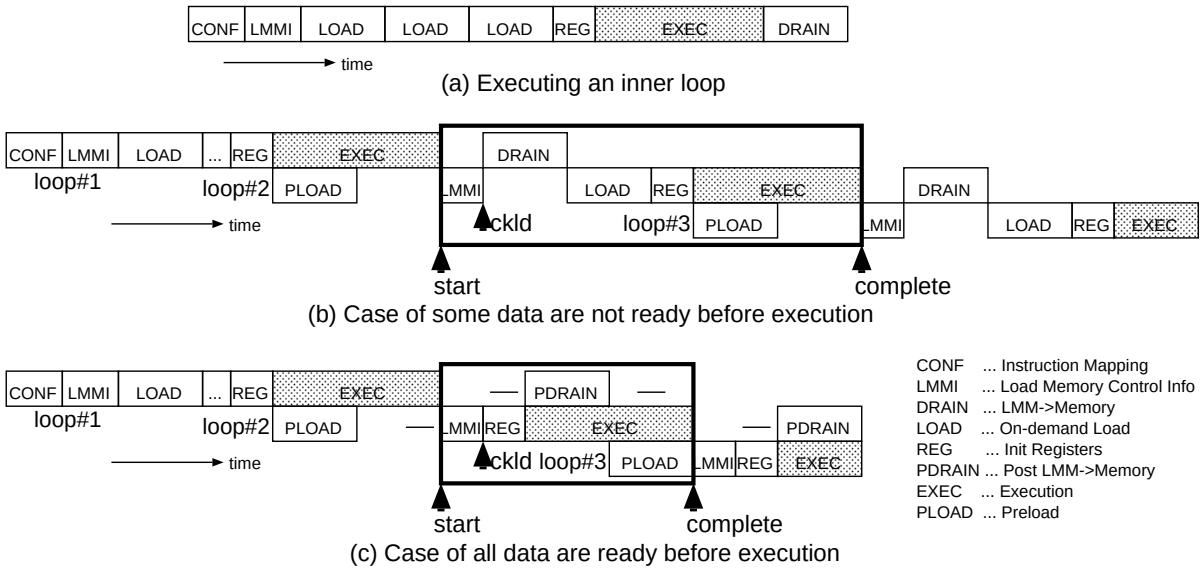


Figure.2.11: Overlapping of execution and long-burst transmission

図 2.11 に，EMAX の状態遷移を示す．EMAX 起動（start）後，必要な conf および lmmi を取得する．次に（ckld），既に LMM に格納されているデータの先頭アドレス，距離，長さ情報と，新たに演算入力として LMM に必要なデータの情報を比較し，不一致の LMM が 1 つでも存在する場合（図 2.11(b)），前回の動作により LMM に格納された演算結果を主記憶へ書き戻した後（DRAIN），主記憶から LMM への転

Table.2.1: 制御レジスタ

アドレス (16進 offset)	属性	内容
0x0000	W/O	1を書き込むとEMAXのハードウェアリセットを行う．リセットは直ちに完了し，リセット完了を待ち合わせる必要はない．
0x0010	W/O	conf先頭物理アドレスを指定する（段数16の場合は2KB，段数64の場合は8KB）．
0x0018	W/O	lmmi先頭物理アドレスを指定する（段数16の場合は1KB，段数64の場合は4KB）．
0x0020	W/O	regv先頭物理アドレスを指定する（段数16の場合は4KB，段数64の場合は16KB）．
0x0028	W/O	EMAX占有領域の先頭物理アドレス-先頭仮想アドレスを指定する．EMAXはLMM仮想アドレスに本オフセットを加算したアドレスをZYNQインタフェースに送出する．
0x0030	W/O	EMAX起動レジスタ．1を書き込むとEMAXを通常起動（emax_start）する．2を書き込むとdirty状態にあるLMMの内容を主記憶へ書き戻す動作（emax_drain_dirty_lmm）を開始する．内部状態はstatusレジスタの監視により取得できる．STATUS_IDLE以外の場合に本レジスタに1を書き込んだ際の動作は保証されない．
0x0040	R/O	Statusレジスタ．後述の内部状態を表示する．
0x0100	R/O	FSM#0 TCUADR． コンパイル時にTCU関数アドレスをFSM#0にセットする情報を生成．当面は1つ．
0x0108,10,18,20	R/O	FSM#0 TCUREG#0,1,2,3.
0x0140	R/O	FSM#1 TCUADR． コンパイル時にTCU関数アドレスをFSM#1にセットする情報を生成．当面は1つ．
0x0148,50,58,60	R/O	FSM#1 TCUREG#0,1,2,3.
0x0180	R/O	FSM#2 TCUADR． コンパイル時にTCU関数アドレスをFSM#2にセットする情報を生成．当面は1つ．
0x0188,90,98,a0	R/O	FSM#2 TCUREG#0,1,2,3.
0x01c0	R/O	FSM#3 TCUADR． コンパイル時にTCU関数アドレスをFSM#3にセットする情報を生成．当面は1つ．
0x01c8,d0,d8,e0	R/O	FSM#3 TCUREG#0,1,2,3.

送を行う（LOAD）．情報が一致する場合（図2.11(c)），LMMの再利用が可能であり，転送を省略する．演算に必要なレジスタ初期値を取得（REG）後，演算（EXEC）を開始する．同時に，前回の動作によりLMMに格納された演算結果が残っている場合，主記憶への書き戻しも開始する（PDRAIN）．また，プリフェッチ情報に基づき，主記憶からLMMへの転送（PLOAD）も開始する．EXEC，PDRAIN，および，PLOADの全てが終結した時点で，EMAX5の動作が完了する（complete）．図2.11(c)の太枠の状態では，PDRAINとPLOADのスループットがEXECの4倍あることを利用した演算と主記憶参照のオーバラップにより，高性能を発揮できる．

2.7 EMAXの内部状態および状態遷移

2.7.1 Idle (pe0_status=STATUS_IDLE(0))

EMAX起動レジスタに1が書き込まれた場合，confの値が前回と異なる時は，STATUS_CONFへ遷移する．confの値が前回と同じかつ命令写像のシフト量（mapdist）が0より大きい時，STATUS_SCONへ遷移する．confの値が前回と同じかつ命令写像のシフト量が0の時，STATUS_LMMIへ遷移する．一方，EMAX起動レジスタに2が書き込まれた場合，STATUS_DRAINへ遷移し，dirty状態にあるLMMの内

容を主記憶へ書き戻す動作（emax_drain_dirty_lmm）を開始する．

2.7.2 Unit configuration in progress (pe0_status=STATUS_CONF(1))

主記憶の text 領域から conf を読み出して各ユニットにセットする．conf は 1 ユニットあたり 64bit*4 ワード分であり，way 毎のスループットに等しい．従って，段数分のサイクルにより，全てのユニットへのセットが完了する．その後，STATUS_LMMI へ遷移する．

2.7.3 Unit configuration in progress (pe0_status=STATUS_SCON(2))

mapdist 段数分の命令シフトを行う．各ユニットは 1 クロック目に内部の conf 情報を自ユニットの出力レジスタ（BR[]）に書き出し，2 クロック目に前段の BR[] から自ユニット内部の conf 情報に取り込む．即ち，mapdist*2 サイクルにより，命令シフトが完了する．その後，STATUS_LMMI へ遷移す本機構により，主記憶から conf を読み出す方法に比べて，CGRA の命令写像を大幅に高速化できる．

2.7.4 LMM tag initialization in progress (pe0_status=STATUS_LMMI(3))

主記憶のスタックフレームから lmmi を読み出して FSM 内レジスタにセットする．lmmi は 1 ユニットあたり 64bit*2 ワード分であり，way 毎のスループットの半分である．従って，段数/2 のサイクルにより，全ての LMM 情報のセットが完了する．その後，STATUS_DRAIN へ遷移する．なお，旧 lmmi が 1 世代分保存される．

2.7.5 LMM drainage in progress (pe0_status=STATUS_LMM_DRAIN(4))

旧 lmmi と新 lmmi を比較し，EMAX 起動レジスタに 1 が書き込まれている場合，旧 lmmi が書き込み先かつ新 lmmi が PDRAIN 対象（lmd）でないか，または，旧 lmmi が強制 STORE 対象（lmx）の時，当該 LMM を主記憶へ追い出す．その後，STATUS_LOAD へ遷移する．EMAX 起動レジスタに 2 が書き込まれている場合，dirty 状態にある LMM を主記憶へ追い出す．その後，STATUS_TERM へ遷移する．

2.7.6 LMM loading in progress (pe0_status=STATUS_LMM_LOAD(5))

旧 lmmi と新 lmmi を比較し，新 lmmi に対応する領域が LMM に存在しない場合，または，強制 LOAD 対象（lmf，lmx）の場合，主記憶から当該 LMM へ書き込む．その後，STATUS_REGV へ遷移する．

2.7.7 Register initialization in progress (pe0_status=STATUS_REGV(6))

主記憶のスタックフレームから regv を読み出して各ユニットの EAG 入力レジスタおよび演算出力レジスタ（BR[]）にセットする．regv は 1 ユニットあたり 64bit*8 ワード分であり，way 毎のスループットの 2 倍である．従って，段数*2 のサイクルにより，全てのレジスタのセットが完了する．その後，STATUS_START へ遷移する．

2.7.8 Start execution (pe0_status=STATUS_START(7))

論理的な先頭段から演算を開始する．直ちに，STATUS_EXEC へ遷移する．

2.7.9 Execution in progress (pe0_status=STATUS_EXEC(8))

演算実行中の状態である．論理的な先頭段が生成した演算結果が順次次段へ送られ，最終的には，演算が写像された全てのユニットが動作する．所定のサイクル数の動作が完了すると，STATUS_TERM へ遷移する．なお，演算実行中に，LMM に対する主記憶からの書き込み（PLOAD）および LMM から主記憶への読み出し（PDRAIN）が行われる．

2.7.10 Termination in progress (pe0_status=STATUS_TERM(9))

EMAX の動作が完了した状態である．内部状態を初期化後，直ちに，STATUS_IDLE へ遷移する．

2.8 実装例と動作

2.8.1 SIMD 計算時

図 2.12 に SIMD 計算時の動作を示す．第 1 段では，way2 に配列 B，同 way1 に配列 C，同 way0 に配列 D が写像され，3 つの 4 倍幅ロード命令が連続実行される．4 組の B,C,D 要素は，第 2 段における積和演算に入力され，演算結果が way3 の LMM に格納される．同時に，次の連続実行に備えて，第 2 段の way2，1，0 には主記憶から B，C，D をプリフェッチするとともに，第 1 段の way3 に格納された前回の実行結果を主記憶へ書き戻す．本実行モデルでは，命令写像を 1 段ずつ下方にシフトし，プリフェッチしたデータを次段で使用する．LMM の容量制約により連続実行が分断される欠点があるが，実行中に，LMM への書き込みと読み出しの速度差を調整する必要がない．

一方，第 4 段と第 5 段に，命令写像のシフトが不要な使用方法を示す．主記憶から第 4 段の LMM に供給しつつ，同メモリから読み出しを行う．また，第 4 段の LMM に演算結果を格納しつつ，同メモリから主記憶へ書き戻す．各 way に接続される主記憶バスが全て稼働する理想的状態である．LMM への書き込みと読み出しの速度差を調整する必要があるものの，連続実行が分断される欠点がない．

なお，図中の oneshot は，外部からバースト転送開始アドレスを受け取り，後続アドレスはユニット内部のアドレス生成器が LMM に対して供給することを意味する．

2.8.2 基本ステンシル計算時

図 2.13 に基本ステンシル計算（後述する jacobi）時の動作を示す．ステンシル中心 [z][y][x] と [z][y][x+1] からのロードは第 2 行の way0，[z][y][x-1] は way1 に各々写像されており，[z][y-1][x] と [z][y+1][x] は各々第 1 行 way0，第 3 行 way0 に写像されている．新たな y に関して計算を開始する際は，[z][y-1][*]，[z][y][*] に前回の LMM を再利用でき，[z][y+1][*] が LMM に用意されていればよいため，前回の演算中に [z][y+2][*] を第 4 行にプリロードする．また，再利用ができない [z-1][y][*] と [z+1][y][*] は，第 2 段においてプリロードし，第 1 段にてロードする写像となっている．

2.8.3 離散ステンシル計算時

図 2.14 に離散ステンシル計算（後述する gdepth）時の動作を示す．3x3 の画素が 3x3 の位置に分散しており，中央の画素値計算に，9 行（3 行ずつ垂直方向に隣接）の画素行列からの計 42 画素の読み出しと 36 回の絶対値差分計算を必要とする．各段に，1 行分の配列データを読み込み，way2,way1,way0（背景紫色），way2,way0（背景黄色），または，way2,way0（背景水色）の 2 ポート LMM に保持することにより，同一行の多数箇所からの同時読み出しを実現する．各段の LMM は全て同一内容であり，いずれかの way を利用して外部メモリから読み出したデータを複数 LMM が水平方向のバスを共有して同時に受け取る．なお，離散ステンシルの場合，垂直方向の再利用が期待できないため，LMM から読み出す段の次段において必ず LMM へのプリフェッチが必要である．4way を利用すれば，4 段分の LMM に対する同時書き込みが可能である．この例では，第 1，3，5 段の LMM を演算に利用しつつ，way2 および way1 の外部メモリバスを利用して，第 2，4，6 段の複数 LMM へプリフェッチしている（図では第 6 段を省略）．第 2 段では，まず，way2,way1,way0 の外部メモリバスから LMM の格納開始アドレスを一度だけ受け取り（oneshot），以降はユニット内部のアドレス生成器を使用して連続アドレスを LMM に供給する．これにより，複数段へのアドレス供給の競合を回避している．way2,way1,way0 の各 LMM へは，way2 の外部メモリバスから供給されるデータを水平方向のバスを利用して供給する．

2.8.4 グラフ計算時

図 2.15 にグラフ計算（後述する tricount0）時の動作を示す．第 2 段においてプリフェッチした隣接頂点リストを第 1 段において LMM から読み出し，第 2 段に写像された LDDMQ が主記憶を直接参照して隣接頂点データを読み出す．第 2 段は主記憶参照の遅延を吸収するために，way2 において LMM を利用したバッファリング（タイミング合わせ）を行う．LDDMQ により主記憶から取得したデータは第 3 段に送られ，第 4 段から主記憶へのトランザクションが発行される．第 2 段へのプリフェッチは way0，第 2 段の LDDMQ は way1，第 4 段のトランザクションは way2 に接続されており，主記憶参照パスが分離されている．

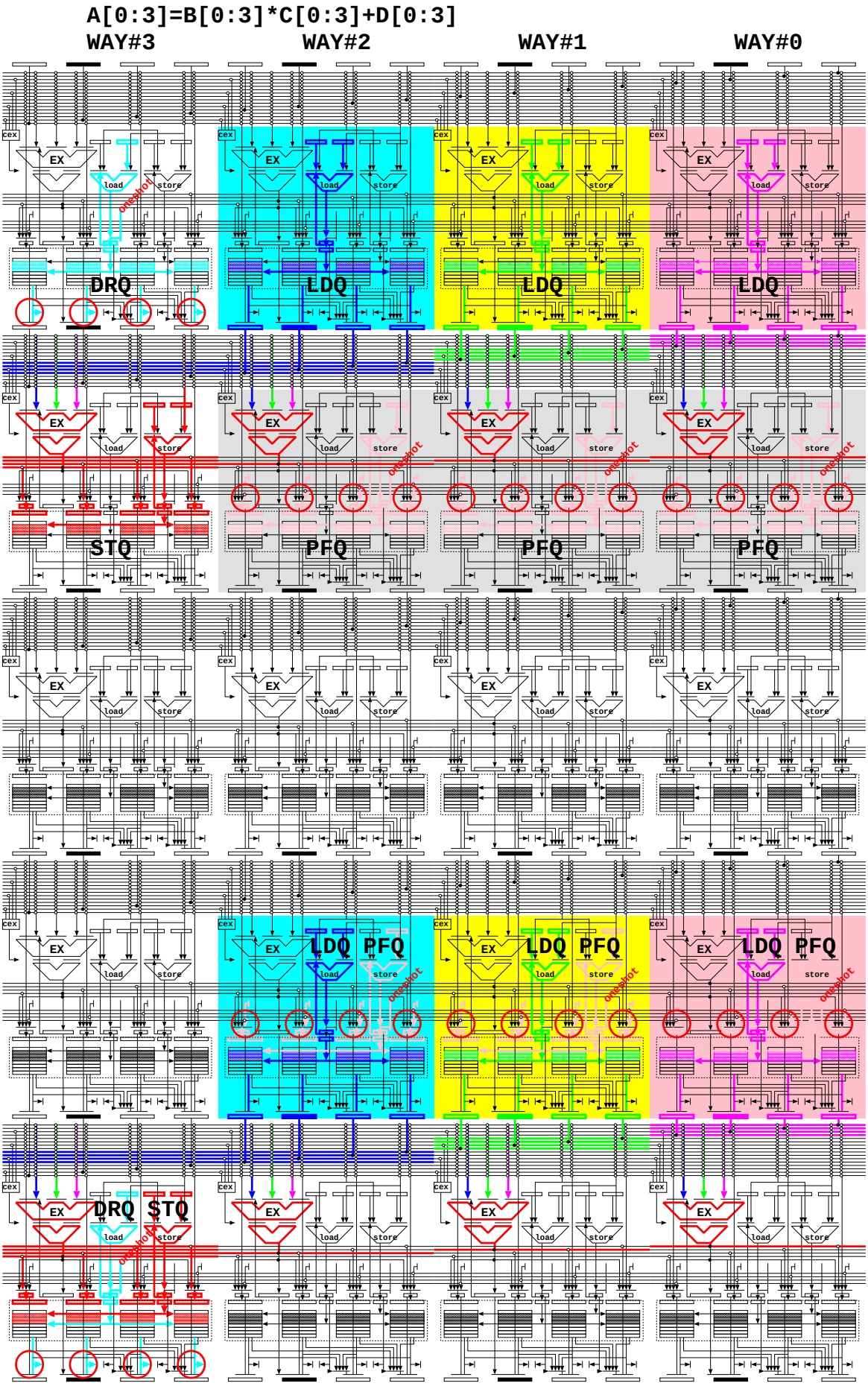


Figure.2.12: Mapping of parallel SIMD

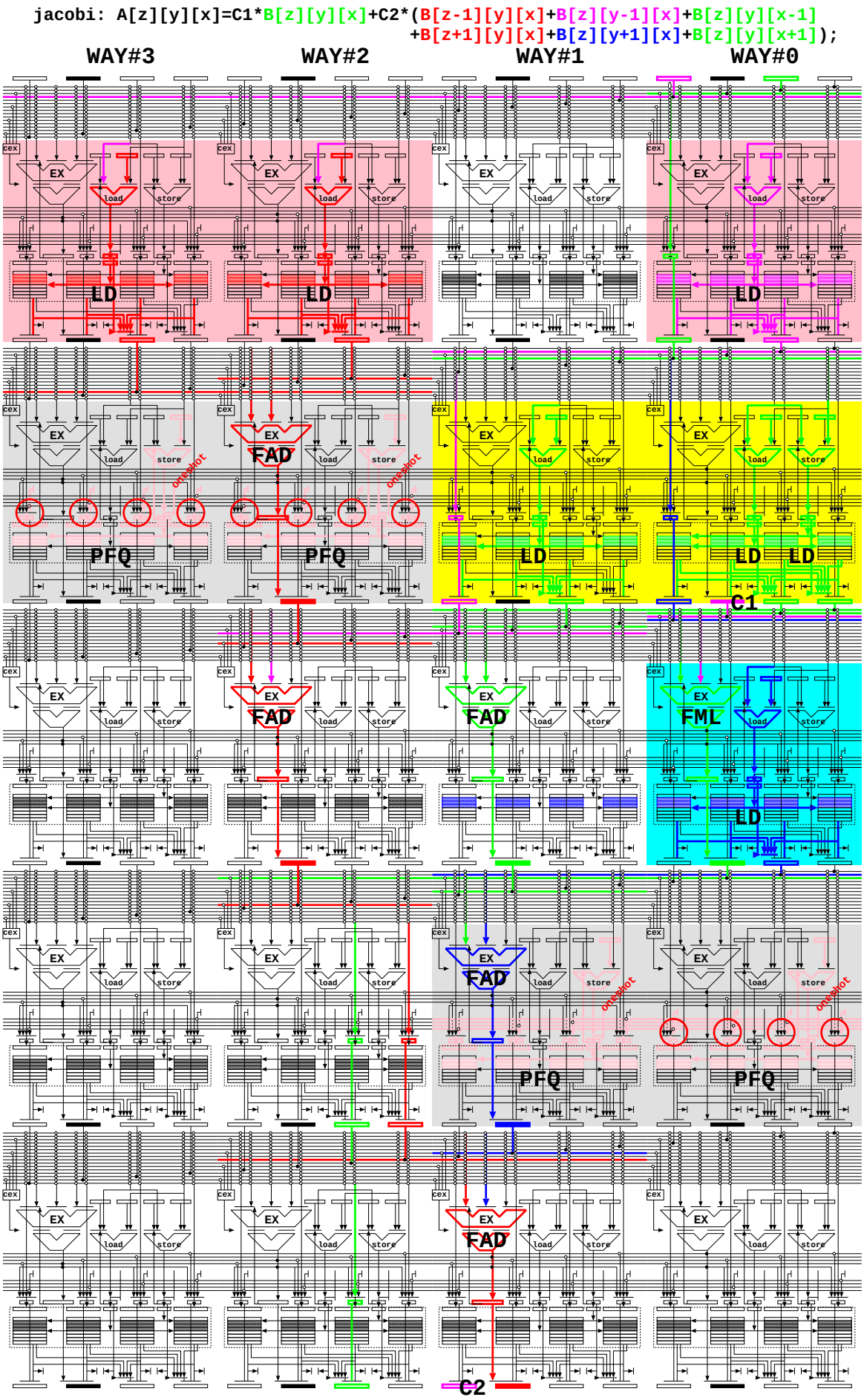


Figure.2.13: Mapping of basic stencil calculation

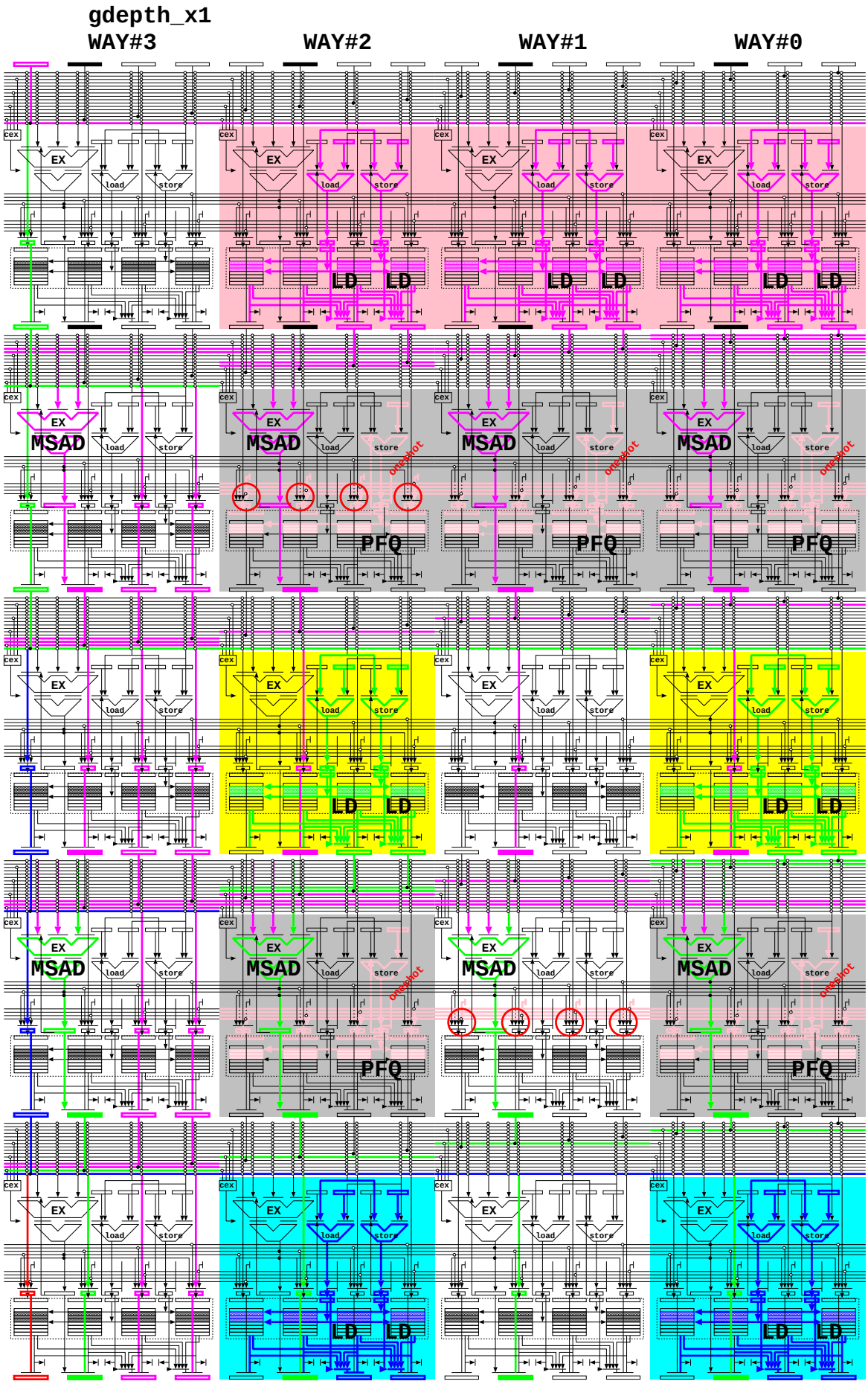


Figure.2.14: Mapping of sparse stencil calculation

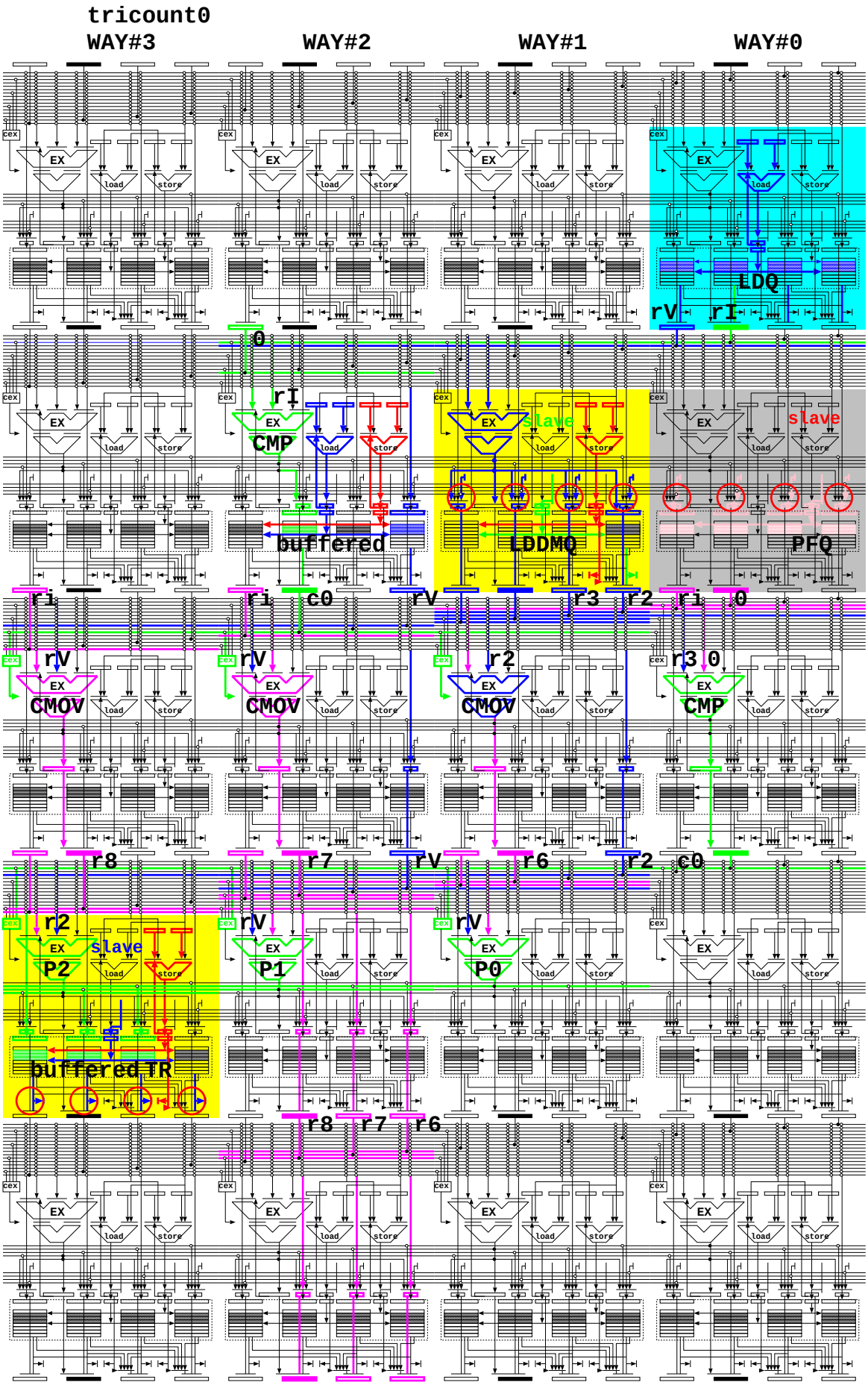


Figure.2.15: Mapping of graph calculation

Chapter 3

EMAX5/ZYNQ Software

3.1 Programming model

3.1.1 Combination of ARM, CGRA and transaction

一般のメニコアプログラムが，一般命令列，スレッド生成･待ち合わせ命令，および，排他･同期命令から構成される並列プログラミングモデルに基づくのに対し，EMAX5/ZYNQ のプログラムは（1）ARM+pthread 命令列；（2）CGRA 写像命令列；（3）Transaction 命令列；の 3 つから構成される機能分散プログラミングモデルに基づく．なお（2）および（3）において使用する命令列は，各々（1）において使用する ARM 命令のサブセットであり，（1）の命令列として逐次実行しても同じ結果を得ることができる．特に，サブセットからは，bsim における ARM デコーダが命令分解の対象とする LDM,STM,VLDM,VSTM が除かれる．（1）kernel_top()，（2）kernel_cgra()，および（3）kernel_tcu1() を連携させるプログラム記述例は以下の通りである．

```
#define XSIZE 1024
#define YSIZE 1024
double A[YSIZE][XSIZE];
double B[YSIZE][XSIZE];
double C[YSIZE][XSIZE];
double D[YSIZE][XSIZE];

main(argc, argv) int argc; char **argv;
{
  sysinit(); /* emax5_open(), mmap() and reset */
  while (read_data()) {
    do {
     kernel_top(&status);
    } while (status != STATUS_END);
    write_data();
  }
  return (0);
}
kernel_top(status) int *status; /* CPU always starts from kernel_top() */
{
  pre_processing;
  for (y=0; y<YSIZE; y++)
    kernel_cgra(y);
//EMAX5A drain_dirty_lmm
  return (status);
}
kernel_cgra(y) int y; /* If CGRA is not available, CPU executes. */
{
  struct { Ull data[4]; } *a = A[y];
  struct { Ull data[4]; } *b = B[y];
  struct { Ull data[4]; } *c = C[y];
  struct { Ull data[4]; } *c = D[y];
  Ull  AR[16][4];                      /* output of EX     in each unit */
  Ull  BR[16][4][4];                   /* output registers in each unit */
#define dst1 AR[1]                     /* address of AR[1][3:0]   */
#define dst2 AR[2]                     /* address of AR[2][3:0]   */
#define src1 BR[0][0]                  /* address of BR[0][0][3:0] */
#define src2 BR[0][1]                  /* address of BR[0][1][3:0] */
#define src3 BR[0][2]                  /* address of BR[0][2][3:0] */
  /* D = A * B + C */
//EMAX5A begin map_dist=0
  while (loop--) { /* mapped to while() on BR[15][0][0] */
    mo4(LDRQ, 1, src1, a++, 0, msk, A[y], XSIZE, 0, 0, NULL); /* burst */
    mo4(LDRQ, 1, src2, b++, 0, msk, B[y], XSIZE, 0, 0, NULL); /* burst */
    mo4(LDRQ, 1, src3, c++, 0, msk, C[y], XSIZE, 0, 0, NULL); /* burst */
    ex4(FMA, dst1, src1, src2, src3);
    mo4(STRQ, 1, dst1, d++, 0, msk, D[y], XSIZE, 0, 0, NULL); /* burst */
    ...
    mo4(TR, ...);                        /* adr, data1, data2, data3 */
  }
//EMAX5A end
}
kernel_tcu1(param) Ull *param; /* If TCU is not available, CPU executes. */
{
  if (param[1] > 0.0)
    *(double*)(param[0]) = *(double*)(param[2])
                         + *(double*)(param[3]); /* sample of transaction */
}
```

kernel_top() が外側ループに対応しており，最内ループ kernel_cgra() を呼び出している．また，kernel_cgra() が，さらにトランザクション kernel_tcu1() を呼び出している．kernel_cgra() および kernel_tcu1() は，通常の C コンパイラによりコンパイルすることにより，ARM による実行が可能な記述となっている．一方，EMAX5/ZYNQ に対応したコンパイラは，ディレクティブ//EMAX5A begin から//EMAX5A end の間の C プログラムを EMAX5 の CGRA に写像可能な形に変換し，ARM からの起動手順を含むアセンブリ言語プログラムを埋め込む．

Table.3.1: ALU operations

unit	usage			mode	description
CEX	CEXE	((char*)ex, c3, c2, c1, c0, imm)		2bit 4in 16bit	word-wise(w1/w0) conditional execution
EX1,2,3	ex4.FMA	(Ull*)d|s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)		32bit*2*4 3in	floating-point s1+s2*s3
	ex4.FAD	(Ull*)d|s1, ((Ull*)s1, (Ull*)s2, -)		32bit*2*4 2in	floating-point s1+s2
	ex4.FML	(Ull*)d, ((Ull*)s1, (Ull*)s2, -)		32bit*2*4 2in	floating-point s1*s2
	FMA	(Ull*)d|&s1, (s1, s2, s3)		32bit*2 3in	floating-point s1+s2*s3
	FAD	(Ull*)d|&s1, (s1, s2, -)		32bit*2 2in	floating-point s1+s2
	FML	(Ull*)d, (s1, s2, -)		32bit*2 2in	floating-point s1*s2
EX1	ex4.ADD3	(Ull*)d|s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)		32bit*2*4 3in	integer add s1+s2+s3
	ex4.SUB3	(Ull*)d|s1, ((Ull*)s1, (Ull*)s2, (Ull*)s3)		32bit*2*4 3in	integer subtract s1-(s2+s3)
	ex4.ADD	(Ull*)d|s1, ((Ull*)s1, (Ull*)s2, -)		32bit*2*4 2in	integer add s1+s2
	ex4.SUB	(Ull*)d|s1, ((Ull*)s1, (Ull*)s2, -)		32bit*2*4 2in	integer subtract s1-s2
	ADD3	(Ull*)d|&s1, (s1, s2, s3)		32bit*2 3in	integer add s1+(s2+s3)
	SUB3	(Ull*)d|&s1, (s1, s2, s3)		32bit*2 3in	integer subtract s1-(s2+s3)
	ADD	(Ull*)d|&s1, (s1, s2, -)		32bit*2 2in	integer add s1+s2
	SUB	(Ull*)d|&s1, (s1, s2, -)		32bit*2 2in	integer subtract s1-s2
	CMP_{EQ|NE|LT|LE|GT|GE}	((Ull*)d, s1, s2)		32bit*2 2in	word-wise(w1/w0) compare and set 1*2bit-CC
	CMOV	(Ull*)d, (s1, s2, s3)		2,32bit*2 2in	word-wise(w1/w0) conditional move
	CCAT	(Ull*)d, (s1, s2, -)		32bit 2in	s1<<32|s2 concatenate
	MAUH3	(Ull*)d, (s1, exp, s2, exp, s3, exp)		16bit*4 3in	s1.exp+(s2.exp+s3.exp)
	MAUH	(Ull*)d, (s1, exp, s2, exp, -, -)		16bit*4 2in	s1.exp+s2.exp
	MSUH3	(Ull*)d, (s1, exp, s2, exp, s3, exp)		16bit*4 3in	s1.exp-(s2.exp+s3.exp)
	MSUH	(Ull*)d, (s1, exp, s2, exp, -, -)		16bit*4 2in	s1.exp-s2.exp
	MLUH	(Ull*)d, (s1, exp, s2, exp, -, -)		[11bit*4]*[9bit*2]	s1.exp*s2.exp
	MMRG	(Ull*)d, (s1, s2, s3)		8bit*2 3in	s1.b4|s2.b4|s3.b4|0→w1,s1.b0|s2.b0|s3.b0|0→w0
	MSSAD	(Ull*)d, (s1, s2, s3)		16bit*4	s1.h3+df(s2.b7,s3.b7)+df(s2.b6,s3.b6)→d.h3
				8bit*8 2in	s1.h2+df(s2.b5,s3.b5)+df(s2.b4,s3.b4)→d.h2
					s1.h1+df(s2.b3,s3.b3)+df(s2.b2,s3.b2)→d.h1
					s1.h0+df(s2.b1,s3.b1)+df(s2.b0,s3.b0)→d.h0
	MSAD	(Ull*)d, (s2, s3 -)		16bit*4	df(s1.b7,s2.b7)+df(s1.b6,s2.b6)→d.h3
				8bit*8 2in	df(s1.b5,s2.b5)+df(s1.b4,s2.b4)→d.h2
					df(s1.b3,s2.b3)+df(s1.b2,s2.b2)→d.h1
					df(s1.b1,s2.b1)+df(s1.b0,s2.b0)→d.h0
	MINL3	(Ull*)d, (s1, s2, s3)		16bit*4 3in	(s3.h3<s3.h2)?s1.h3|s3.h3:s2.h3|s3.h2→d.w1
					(s3.h1<s3.h0)?s1.h1|s3.h1:s2.h1|s3.h0→d.w0
	MINL	(Ull*)d, (s1, s2, -)		16bit*4 2in	(s1.h2<s2.h2)?s1.w1:s2.w1→d.w1
					(s1.h0<s2.h0)?s1.w0:s2.w0→d.w0
	MH2BW	(Ull*)d, (s1, s2, -)		16bit*4 2in	s1.b6|s1.b4|s2.b6|s2.b4|s1.b2|s1.b0|s2.b2|s2.b0
	MCAS	(Ull*)d, (s1, s2, -)		16bit*2 2in	(s1.h2<s2.h2)?0:0xff→d.b1
					(s1.h0<s2.h0)?0:0xff→d.b0
	MMID3	(Ull*)d, (s1, s2, s3)		8bit*8 3in	bytewise compare and output middle
	MMAX3	(Ull*)d, (s1, s2, s3)		8bit*8 3in	bytewise compare and output maximum
	MMIN3	(Ull*)d, (s1, s2, s3)		8bit*8 3in	bytewise compare and output minimum
	MMAX	(Ull*)d, (s1, s2, -)		8bit*8 2in	bytewise compare and output maximum
	MMIN	(Ull*)d, (s1, s2, -)		8bit*8 2in	bytewise compare and output minimum
EX2	AND	(Ull*)d, (s4)	cascadable	64bit 2in	logical and s1&s2
	OR	(Ull*)d, (s4)	cascadable	64bit 2in	logical or s1|s2
	XOR	(Ull*)d, (s4)	cascadable	64bit 2in	logical xor s1∧s2
	SUMHH	(Ull*)d, (-)	cascadable	16bit*4 1in	s1.h3+s1.h2→d.h3, s1.h1+s1.h0→d.h1
	SUMHL	(Ull*)d, (-)	cascadable	16bit*4 1in	s1.h3+s1.h2→d.h2, s1.h1+s1.h0→d.h0
	WSWAP	(Ull*)d, (-)	cascadable	32bit*2 1in	s1.w0|s1.w1
EX3	SLL	(Ull*)d, (s5)	cascadable	32bit*2 2in	32bit logical shift to left
	SRL	(Ull*)d, (s5)	cascadable	32bit*2 2in	32bit logical shift to right
	SRAA	(Ull*)d, (s5)	cascadable	32bit*2 2in	32bit arith shift to right (bit63,31 is ext.)
	SRAB	(Ull*)d, (s5)	cascadable	32bit*2 2in	32bit arith shift to right (bit55,23 is ext.)
	SRAC	(Ull*)d, (s5)	cascadable	32bit*2 2in	32bit arith shift to right (bit47,15 is ext.)
	SRAD	(Ull*)d, (s5)	cascadable	32bit*2 2in	32bit arith shift to right (bit39,7 is ext.)
	SRLM	(Ull*)d, (s5)	cascadable	16bit*4 2in	16bit logical shift to right

† exp is 3:64bit→16bit[4], 2:byte7,6,3,2→16bit[4], 1:byte5,4,1,0→16bit[4]
† (Ull*)s1,(Ull*)s2,(Ull*)s3,(Ull*)d are pointers and s1, s2, s3 are values
† c3, c2, c1, c0 are 1*2bit-values, and (char*)ex is a pointer
† .b[7:0] is 8bit portion, .h[3:0] is 16bit portion and .w1|w0 is 32bit portion in 64bit respectively
† d|s1 is d:normal operation, s1:accumulation. succeeding operations with s1 reffer the result.

3.1.2 Instruction format for CGRA

表 3.1 に，CGRA に写像可能な演算記述様式を示す．レジスタ幅は 64bit であり，基本データ型は，浮動小数点演算では単精度演算の 2 要素連結，整数演算では 32bit 幅の 2 要素連結，メディア演算では 16bit 幅の 4 要素連結または 8bit 幅の 8 要素連結である．AND，OR，XOR，SUMHH，SUMHL は，単独で前

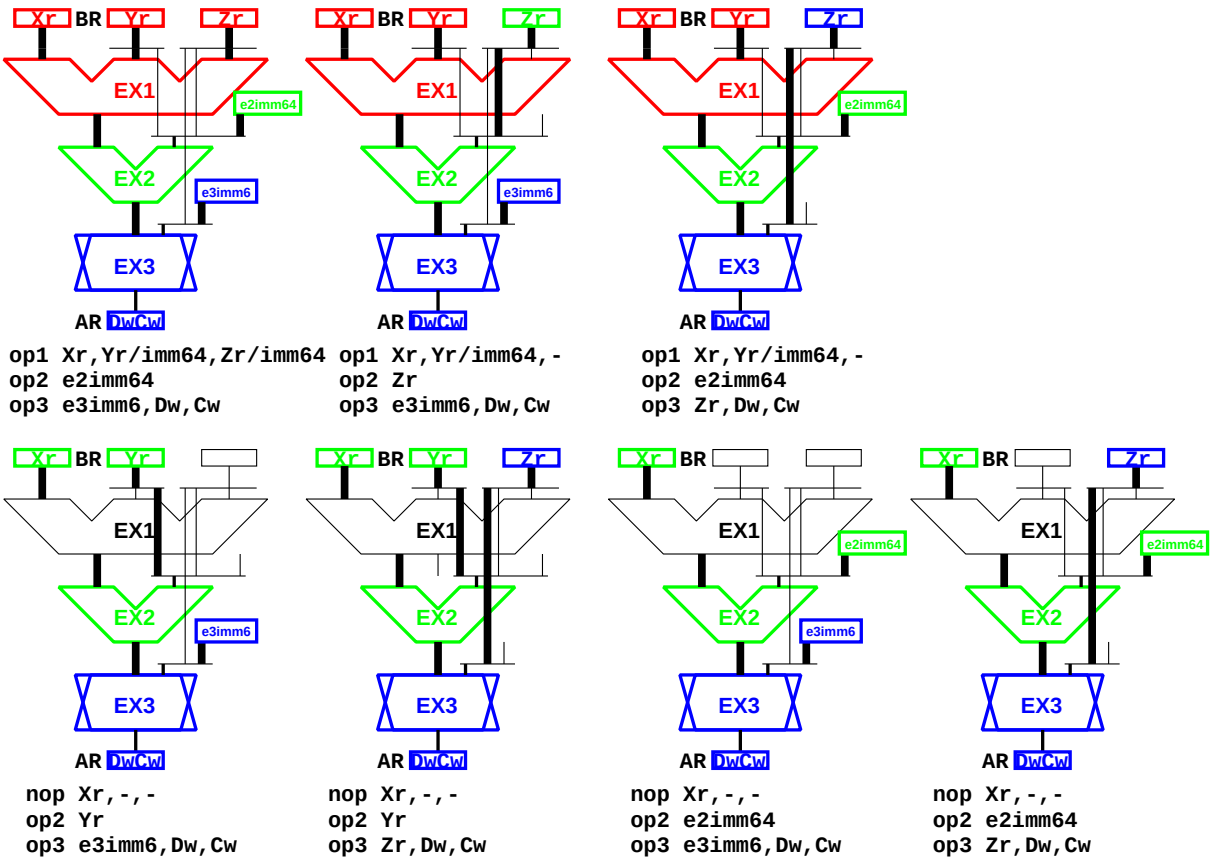


Figure.3.1: Combination of ALU operations and datapath from registers

Table.3.2: Memory operations

unit	usage		mode	description
LD	LDR	(-, (Ull*)d, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	64bit lmm	LMM rand-access
	LDWR	(-, (Ull*)d, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	s32bit lmm	LMM rand-access
	LDUWR	(-, (Ull*)d, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	u32bit lmm	LMM rand-access
	LDHR	(-, (Ull*)d, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	s16bit lmm	LMM rand-access
	LDUHR	(-, (Ull*)d, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	u16bit lmm	LMM rand-access
	LDBR	(-, (Ull*)d, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	s8bit lmm	LMM rand-access
	LDUBR	(-, (Ull*)d, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	u8bit lmm	LMM rand-access
	LDRQ	(-, (Ull*)d, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	64bit*4 lmm	LMM rand-access
	LDDMQ	(-, (Ull*)d, base(++), offs, msk, -, -, -, -, -, -)	64bit*4 mem	Direct access to MM
ST	STR	(ex, s, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	64bit lmm	LMM rand-access
	STWR	(ex(b0), s, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	32bit lmm	LMM rand-access
	STHR	(ex(b0), s, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	16bit lmm	LMM rand-access
	STBR	(ex(b0), s, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	8bit lmm	LMM rand-access
	STRQ	(-, (Ull*)s, base(++), offs, msk, top, len, blk, force-read, ptop, plen)	64bit*4 lmm	LMM rand-access
	TR	(ex(b0), (Ull*)s, -, -, -, func(), -, -, -, -, -)	64bit*4 exec	Send transaction

† ex is 2bit (b1 controls word1, b0 controls word0)
† msk is 15:64bit, 13:word1, 12:word0, 11-8:half3-0, 7-0:byte7-0 of offset
† base++ increments address by the size of element.
† LD with force-read=0 and ptop==NULL generates current(lmr) and reuse LMM. same as lmr in EMAX4
† LD with force-read=1 and ptop==NULL generates current(lmf) and !reuse LMM. same as lmf in EMAX4
† LD with force-read=0 and ptop!=NULL generates current(lmr) and next(lmp). mapdist=0 allowed
† LDDMQ set f=1 and p=1 in lmmc automatically
† ST with force-read=0 and ptop==NULL generates current(lmw) and reuse+wback LMM. same as lmw in EMAX4
† ST with force-read=1 and ptop==NULL generates current(lmx) and !reuse+wback LMM. same as lmx in EMAX4
† ST with force-read=0 and ptop!=NULL generates current(lmw) and prev(lmd). mapdist=0 allowed
† TR set f=1 and p=1 in lmmc automatically

段の出力レジスタを入力とできることに加え，integer-add/sub の出力 d を入力に指定することにより，ユニット内カスケード演算が可能である．さらに，SLL，SRL，SRLA，SRLB，SRLC，SRLD，SRLM は，単独で前段の出力レジスタを入力とできることに加え，integer-add/sub の出力 d や，AND，OR，XOR，

SUMHH，SUMHL の出力 D を入力に指定することにより，ユニット内カスケード演算が可能である．すなわち，ユニット内で最大，integer-add/sub，論理演算，シフト演算の順に 3 つの演算をカスケード実行できる．CMP，CEXE，CMOV は，条件付き実行のための記述である．CMP により条件コードを生成し，CEXE により複数の条件コードを組合せるロードストア用実行条件を生成でき，CMOV により単一の条件コードによる条件付き代入を行う．CEX の出力は前述のロード/ストア記述の実行条件 ex に使用できる．図 3.1 に，演算記述と演算器写像の関係を示す．

表 3.2 に，CGRA に写像可能なロード/ストア記述様式を示す．LDDMQ は外部メモリ直接参照，TR はトランザクション発行，残りは LMM のランダム参照用である．

プログラミングの際には，まず，ストアデータに着目し，LMM に対する単一ストアが，必ずストア幅にアラインされるよう留意する．例えば 32bit 整数データを 2 つ連結した 64bit データをストアする際には，64bit にアラインしなければならない．一方，ロードデータに関しては，1 つの LMM から 64bit データを 4 つ同時に読み出す際には 256bit 境界を意識しなければならないものの，2 ポートを利用して 2 箇所から 32bit データを読み出す際には 32bit 境界のみを意識すればよい．幅方向の 4 つの LMM に同じデータを格納することにより，一度に 8 箇所からのランダム読み出しが可能であり，ストアよりも境界に関する自由度が高い．この性質を利用すれば様々なステンシル計算を無理なく写像することができる．

3.1.3 Overlapping of prefetch and drain

ステンシル計算では，（1）直前の連続演算が LMM に格納した演算結果の主記憶への書き戻し，（2）LMM に揃っている入力データを利用した演算器の連続動作，および，（3）次の連続演算に必要なデータの主記憶から LMM へのプリフェッチの全てを同時動作させることにより，高効率を達成できる．本動作を利用する際の記述方法として，プログラマが（1）～（3）の全てを明示的に記述する方法と，（2）のみを記述し（1）（3）はコンパイラが自動生成する方法が考えられる．前者の場合，最初の連続動作では入力データが揃っていないため（3）のみを有効化し，同様に最終の連続動作では（1）のみを有効化する仕組みが必要となる．後者の場合，単純なステンシル計算でなければ（2）の情報から（3）を自動生成できず，離散ステンシルに対応できない．また，最終の（2）を検出して単独の（1）を自動的に追加することも困難である．以上のことから，EMAX5 では前者を採用している．以下は前述の kernel_cgra() を本動作に書き換えた例である．（1）に関しては，kernel_top() 内の post_processing 部分において，emax5_drain_dirty_lmm() を呼び出すことにより，STRQ の格納先 LMM に残っている演算結果を主記憶に書き出す動作を指示する．（3）に関しては，LDRQ の最後の 3 つの引数により，LMM からの読み出しと同時に，次段（段間の距離は，map_dist により指定）の LMM にプリフェッチするための先頭アドレス，距離，長さを指示する．

```
#define XSIZE 1024
#define YSIZE 1024
double A[YSIZE][XSIZE];
double B[YSIZE][XSIZE];
double C[YSIZE][XSIZE];
double D[YSIZE][XSIZE];

kernel_top(status) int *status; /* CPU always starts from kernel_top() */
{
  pre_processing;
  for (y=0; y<YSIZE; y++)
    kernel_cgra(y);
//EMAX5A drain_dirty_lmm ← (1) ★
  return (status);
}
kernel_cgra(y) int y; /* If CGRA is not available, CPU executes. */
{ /* D = A * B + C */
//EMAX5A begin map_dist=1
  while (loop--) { /* mapped to while() on BR[15][0][0] */
    mo4(LDRQ, 1, src1, a++, 0, msk, A[y], XSIZE, 0, 0, A[y+1]); /* prefetch   */← (3) ★
    mo4(LDRQ, 1, src2, b++, 0, msk, B[y], XSIZE, 0, 0, B[y+1]); /* prefetch   */← (3) ★
    mo4(LDRQ, 1, src3, c++, 0, msk, C[y], XSIZE, 0, 0, C[y+1]); /* prefetch   */← (3) ★
    ex4(FMA, dst1, src1, src2, src3);
    mo4(STRQ, 1, dst1, d++, 0, msk, NULL, XSIZE, 0, 0, D[y-1]); /* late drain */← (1) ★
  }
//EMAX5A end
}
```

3.2 Examples (2D-convolution)

3.2.1 16x16 畳み込み演算

```
gather_x1(float *yi, float *yo)
{
  Ull  loop = image_WD/2-8;
  Ull  x = 8;
#ifndef EMAX
  /***********************************************/
  /* non EMAX5                                   */
  /***********************************************/
  while (loop--) {
    yo[x  ]
     = yim8[x-8]*SCON[  0]+yim8[x-7]*SCON[  1]+yim8[x-6]*SCON[  2]+yim8[x-5]*SCON[  3]+yim8[x-4]*SCON[  4]+yim8[x-3]*SCON[  5]+yim8[x-2]*SCON[  6]+yim8[x-1]*SCON[  7]
     + yim8[x+0]*SCON[  8]+yim8[x+1]*SCON[  9]+yim8[x+2]*SCON[ 10]+yim8[x+3]*SCON[ 11]+yim8[x+4]*SCON[ 12]+yim8[x+5]*SCON[ 13]+yim8[x+6]*SCON[ 14]+yim8[x+7]*SCON[ 15]
     + yim7[x-8]*SCON[ 16]+yim7[x-7]*SCON[ 17]+yim7[x-6]*SCON[ 18]+yim7[x-5]*SCON[ 19]+yim7[x-4]*SCON[ 20]+yim7[x-3]*SCON[ 21]+yim7[x-2]*SCON[ 22]+yim7[x-1]*SCON[ 23]
     + yim7[x+0]*SCON[ 24]+yim7[x+1]*SCON[ 25]+yim7[x+2]*SCON[ 26]+yim7[x+3]*SCON[ 27]+yim7[x+4]*SCON[ 28]+yim7[x+5]*SCON[ 29]+yim7[x+6]*SCON[ 30]+yim7[x+7]*SCON[ 31]
     + yim6[x-8]*SCON[ 32]+yim6[x-7]*SCON[ 33]+yim6[x-6]*SCON[ 34]+yim6[x-5]*SCON[ 35]+yim6[x-4]*SCON[ 36]+yim6[x-3]*SCON[ 37]+yim6[x-2]*SCON[ 38]+yim6[x-1]*SCON[ 39]
     + yim6[x+0]*SCON[ 40]+yim6[x+1]*SCON[ 41]+yim6[x+2]*SCON[ 42]+yim6[x+3]*SCON[ 43]+yim6[x+4]*SCON[ 44]+yim6[x+5]*SCON[ 45]+yim6[x+6]*SCON[ 46]+yim6[x+7]*SCON[ 47]
     + yim5[x-8]*SCON[ 48]+yim5[x-7]*SCON[ 49]+yim5[x-6]*SCON[ 50]+yim5[x-5]*SCON[ 51]+yim5[x-4]*SCON[ 52]+yim5[x-3]*SCON[ 53]+yim5[x-2]*SCON[ 54]+yim5[x-1]*SCON[ 55]
     + yim5[x+0]*SCON[ 56]+yim5[x+1]*SCON[ 57]+yim5[x+2]*SCON[ 58]+yim5[x+3]*SCON[ 59]+yim5[x+4]*SCON[ 60]+yim5[x+5]*SCON[ 61]+yim5[x+6]*SCON[ 62]+yim5[x+7]*SCON[ 63]
     + yim4[x-8]*SCON[ 64]+yim4[x-7]*SCON[ 65]+yim4[x-6]*SCON[ 66]+yim4[x-5]*SCON[ 67]+yim4[x-4]*SCON[ 68]+yim4[x-3]*SCON[ 69]+yim4[x-2]*SCON[ 70]+yim4[x-1]*SCON[ 71]
     + yim4[x+0]*SCON[ 72]+yim4[x+1]*SCON[ 73]+yim4[x+2]*SCON[ 74]+yim4[x+3]*SCON[ 75]+yim4[x+4]*SCON[ 76]+yim4[x+5]*SCON[ 77]+yim4[x+6]*SCON[ 78]+yim4[x+7]*SCON[ 79]
     + yim3[x-8]*SCON[ 80]+yim3[x-7]*SCON[ 81]+yim3[x-6]*SCON[ 82]+yim3[x-5]*SCON[ 83]+yim3[x-4]*SCON[ 84]+yim3[x-3]*SCON[ 85]+yim3[x-2]*SCON[ 86]+yim3[x-1]*SCON[ 87]
     + yim3[x+0]*SCON[ 88]+yim3[x+1]*SCON[ 89]+yim3[x+2]*SCON[ 90]+yim3[x+3]*SCON[ 91]+yim3[x+4]*SCON[ 92]+yim3[x+5]*SCON[ 93]+yim3[x+6]*SCON[ 94]+yim3[x+7]*SCON[ 95]
     + yim2[x-8]*SCON[ 96]+yim2[x-7]*SCON[ 97]+yim2[x-6]*SCON[ 98]+yim2[x-5]*SCON[ 99]+yim2[x-4]*SCON[100]+yim2[x-3]*SCON[101]+yim2[x-2]*SCON[102]+yim2[x-1]*SCON[103]
     + yim2[x+0]*SCON[104]+yim2[x+1]*SCON[105]+yim2[x+2]*SCON[106]+yim2[x+3]*SCON[107]+yim2[x+4]*SCON[108]+yim2[x+5]*SCON[109]+yim2[x+6]*SCON[110]+yim2[x+7]*SCON[111]
     + yim1[x-8]*SCON[112]+yim1[x-7]*SCON[113]+yim1[x-6]*SCON[114]+yim1[x-5]*SCON[115]+yim1[x-4]*SCON[116]+yim1[x-3]*SCON[117]+yim1[x-2]*SCON[118]+yim1[x-1]*SCON[119]
     + yim1[x+0]*SCON[120]+yim1[x+1]*SCON[121]+yim1[x+2]*SCON[122]+yim1[x+3]*SCON[123]+yim1[x+4]*SCON[124]+yim1[x+5]*SCON[125]+yim1[x+6]*SCON[126]+yim1[x+7]*SCON[127]
     + yizz[x-8]*SCON[128]+yizz[x-7]*SCON[129]+yizz[x-6]*SCON[130]+yizz[x-5]*SCON[131]+yizz[x-4]*SCON[132]+yizz[x-3]*SCON[133]+yizz[x-2]*SCON[134]+yizz[x-1]*SCON[135]
     + yizz[x+0]*SCON[136]+yizz[x+1]*SCON[137]+yizz[x+2]*SCON[138]+yizz[x+3]*SCON[139]+yizz[x+4]*SCON[140]+yizz[x+5]*SCON[141]+yizz[x+6]*SCON[142]+yizz[x+7]*SCON[143]
     + yip1[x-8]*SCON[144]+yip1[x-7]*SCON[145]+yip1[x-6]*SCON[146]+yip1[x-5]*SCON[147]+yip1[x-4]*SCON[148]+yip1[x-3]*SCON[149]+yip1[x-2]*SCON[150]+yip1[x-1]*SCON[151]
     + yip1[x+0]*SCON[152]+yip1[x+1]*SCON[153]+yip1[x+2]*SCON[154]+yip1[x+3]*SCON[155]+yip1[x+4]*SCON[156]+yip1[x+5]*SCON[157]+yip1[x+6]*SCON[158]+yip1[x+7]*SCON[159]
     + yip2[x-8]*SCON[160]+yip2[x-7]*SCON[161]+yip2[x-6]*SCON[162]+yip2[x-5]*SCON[163]+yip2[x-4]*SCON[164]+yip2[x-3]*SCON[165]+yip2[x-2]*SCON[166]+yip2[x-1]*SCON[167]
     + yip2[x+0]*SCON[168]+yip2[x+1]*SCON[169]+yip2[x+2]*SCON[170]+yip2[x+3]*SCON[171]+yip2[x+4]*SCON[172]+yip2[x+5]*SCON[173]+yip2[x+6]*SCON[174]+yip2[x+7]*SCON[175]
     + yip3[x-8]*SCON[176]+yip3[x-7]*SCON[177]+yip3[x-6]*SCON[178]+yip3[x-5]*SCON[179]+yip3[x-4]*SCON[180]+yip3[x-3]*SCON[181]+yip3[x-2]*SCON[182]+yip3[x-1]*SCON[183]
     + yip3[x+0]*SCON[184]+yip3[x+1]*SCON[185]+yip3[x+2]*SCON[186]+yip3[x+3]*SCON[187]+yip3[x+4]*SCON[188]+yip3[x+5]*SCON[189]+yip3[x+6]*SCON[190]+yip3[x+7]*SCON[191]
     + yip4[x-8]*SCON[192]+yip4[x-7]*SCON[193]+yip4[x-6]*SCON[194]+yip4[x-5]*SCON[195]+yip4[x-4]*SCON[196]+yip4[x-3]*SCON[197]+yip4[x-2]*SCON[198]+yip4[x-1]*SCON[199]
     + yip4[x+0]*SCON[200]+yip4[x+1]*SCON[201]+yip4[x+2]*SCON[202]+yip4[x+3]*SCON[203]+yip4[x+4]*SCON[204]+yip4[x+5]*SCON[205]+yip4[x+6]*SCON[206]+yip4[x+7]*SCON[207]
     + yip5[x-8]*SCON[208]+yip5[x-7]*SCON[209]+yip5[x-6]*SCON[210]+yip5[x-5]*SCON[211]+yip5[x-4]*SCON[212]+yip5[x-3]*SCON[213]+yip5[x-2]*SCON[214]+yip5[x-1]*SCON[215]
     + yip5[x+0]*SCON[216]+yip5[x+1]*SCON[217]+yip5[x+2]*SCON[218]+yip5[x+3]*SCON[219]+yip5[x+4]*SCON[220]+yip5[x+5]*SCON[221]+yip5[x+6]*SCON[222]+yip5[x+7]*SCON[223]
     + yip6[x-8]*SCON[224]+yip6[x-7]*SCON[225]+yip6[x-6]*SCON[226]+yip6[x-5]*SCON[227]+yip6[x-4]*SCON[228]+yip6[x-3]*SCON[229]+yip6[x-2]*SCON[230]+yip6[x-1]*SCON[231]
     + yip6[x+0]*SCON[232]+yip6[x+1]*SCON[233]+yip6[x+2]*SCON[234]+yip6[x+3]*SCON[235]+yip6[x+4]*SCON[236]+yip6[x+5]*SCON[237]+yip6[x+6]*SCON[238]+yip6[x+7]*SCON[239]
     + yip7[x-8]*SCON[240]+yip7[x-7]*SCON[241]+yip7[x-6]*SCON[242]+yip7[x-5]*SCON[243]+yip7[x-4]*SCON[244]+yip7[x-3]*SCON[245]+yip7[x-2]*SCON[246]+yip7[x-1]*SCON[247]
     + yip7[x+0]*SCON[248]+yip7[x+1]*SCON[249]+yip7[x+2]*SCON[250]+yip7[x+3]*SCON[251]+yip7[x+4]*SCON[252]+yip7[x+5]*SCON[253]+yip7[x+6]*SCON[254]+yip7[x+7]*SCON[255];
    x += 2;
  }
```

```
#else
  /***********************************************/
  /* EMAX5                                       */
  /***********************************************/
  Ull  AR[64][4];                   /* output of EX     in each unit */
  Ull  BR[64][4][4];                /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
  Ull  r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
  Ull c000=DCON[  0], c002=DCON[  1], c004=DCON[  2], c006=DCON[  3], c008=DCON[  4], c010=DCON[  5], c012=DCON[  6], c014=DCON[  7];
  Ull c016=DCON[  8], c018=DCON[  9], c020=DCON[ 10], c022=DCON[ 11], c024=DCON[ 12], c026=DCON[ 13], c028=DCON[ 14], c030=DCON[ 15];
  Ull c032=DCON[ 16], c034=DCON[ 17], c036=DCON[ 18], c038=DCON[ 19], c040=DCON[ 20], c042=DCON[ 21], c044=DCON[ 22], c046=DCON[ 23];
  Ull c048=DCON[ 24], c050=DCON[ 25], c052=DCON[ 26], c054=DCON[ 27], c056=DCON[ 28], c058=DCON[ 29], c060=DCON[ 30], c062=DCON[ 31];
  Ull c064=DCON[ 32], c066=DCON[ 33], c068=DCON[ 34], c070=DCON[ 35], c072=DCON[ 36], c074=DCON[ 37], c076=DCON[ 38], c078=DCON[ 39];
  Ull c080=DCON[ 40], c082=DCON[ 41], c084=DCON[ 42], c086=DCON[ 43], c088=DCON[ 44], c090=DCON[ 45], c092=DCON[ 46], c094=DCON[ 47];
  Ull c096=DCON[ 48], c098=DCON[ 49], c100=DCON[ 50], c102=DCON[ 51], c104=DCON[ 52], c106=DCON[ 53], c108=DCON[ 54], c110=DCON[ 55];
  Ull c112=DCON[ 56], c114=DCON[ 57], c116=DCON[ 58], c118=DCON[ 59], c120=DCON[ 60], c122=DCON[ 61], c124=DCON[ 62], c126=DCON[ 63];
//EMAX5A begin x1 mapdist=2
  while (loop--) {                              /* mapped to WHILE() on BR[15][0][0] stage#0 */
    mop(OP_LDR,  3, &BR[0][0][1], yim80++, 0, MSK_D0, yim80, 320/2, 0, 0, (Ull)NULL, 320/2); /* stage#0 */
    mop(OP_LDR,  3, &r1,          yim81++, 0, MSK_D0, yim80, 320/2, 0, 0, (Ull)NULL, 320/2); /* stage#0 */
    mop(OP_LDR,  3, &r2,          yim82++, 0, MSK_D0, yim80, 320/2, 0, 0, (Ull)NULL, 320/2); /* stage#0 */
    mop(OP_LDR,  3, &r3,          yim83++, 0, MSK_D0, yim80, 320/2, 0, 0, (Ull)NULL, 320/2); /* stage#0 */
    mop(OP_LDR,  3, &r4,          yim84++, 0, MSK_D0, yim80, 320/2, 0, 0, (Ull)NULL, 320/2); /* stage#0 */
    mop(OP_LDR,  3, &r5,          yim85++, 0, MSK_D0, yim80, 320/2, 0, 0, (Ull)NULL, 320/2); /* stage#0 */
    mop(OP_LDR,  3, &r6,          yim86++, 0, MSK_D0, yim80, 320/2, 0, 0, (Ull)NULL, 320/2); /* stage#0 */
    mop(OP_LDR,  3, &r7,          yim87++, 0, MSK_D0, yim80, 320/2, 0, 0, (Ull)NULL, 320/2); /* stage#0 */
    exe(OP_FMA, &r10, 0LL, EXP_H3210, BR[0][0][1], EXP_H3210, c000, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#1 */
    exe(OP_FMA, &r11, 0LL, EXP_H3210, r1,          EXP_H3210, c002, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#1 */
    exe(OP_FMA, &r12, 0LL, EXP_H3210, r2,          EXP_H3210, c004, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#1 */
    exe(OP_FMA, &r13, 0LL, EXP_H3210, r3,          EXP_H3210, c006, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#1 */
    exe(OP_FMA, &r20, r10, EXP_H3210, r4,          EXP_H3210, c008, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#2 */
    exe(OP_FMA, &r21, r11, EXP_H3210, r5,          EXP_H3210, c010, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#2 */
    exe(OP_FMA, &r22, r12, EXP_H3210, r6,          EXP_H3210, c012, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#2 */
    exe(OP_FMA, &r23, r13, EXP_H3210, r7,          EXP_H3210, c014, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#2 */
        :
    exe(OP_FMA, &r10, r20, EXP_H3210, BR[30][0][1],EXP_H3210, c240, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#31 */
    exe(OP_FMA, &r11, r21, EXP_H3210, r1,          EXP_H3210, c242, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#31 */
    exe(OP_FMA, &r12, r22, EXP_H3210, r2,          EXP_H3210, c244, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#31 */
    exe(OP_FMA, &r13, r23, EXP_H3210, r3,          EXP_H3210, c246, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#31 */
    exe(OP_FMA, &r20, r10, EXP_H3210, r4,          EXP_H3210, c248, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#32 */
    exe(OP_FMA, &r21, r11, EXP_H3210, r5,          EXP_H3210, c250, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#32 */
    exe(OP_FMA, &r22, r12, EXP_H3210, r6,          EXP_H3210, c252, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#32 */
    exe(OP_FMA, &r23, r13, EXP_H3210, r7,          EXP_H3210, c254, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#32 */

    exe(OP_FAD, &r10,  r20, EXP_H3210,  r21, EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#33 */
    exe(OP_FAD, &r12,  r22, EXP_H3210,  r23, EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#33 */
#if 1
    exe(OP_FAD, &AR[35][0],  r10, EXP_H3210,  r12, EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#35 */
    mop(OP_STR, 3, &AR[35][0], yoo++, 0LL, MSK_D0, (Ull)yoo, 152, 0, 0, yop, 152);                        /* stage#35 */
#else
    exe(OP_FAD, &r20,        r10, EXP_H3210,  r12, EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#34 */
    mop(OP_STR, 3, &r20,       yoo++, 0LL, MSK_D0, (Ull)yoo, 152, 0, 0, yop, 152);                        /* stage#34 */
#endif
  }
//EMAX5A end
//emax5_start((Ull*)emax5_conf_x1, (Ull*)emax5_lmmi_x1, (Ull*)emax5_regv_x1);
#endif
}
```

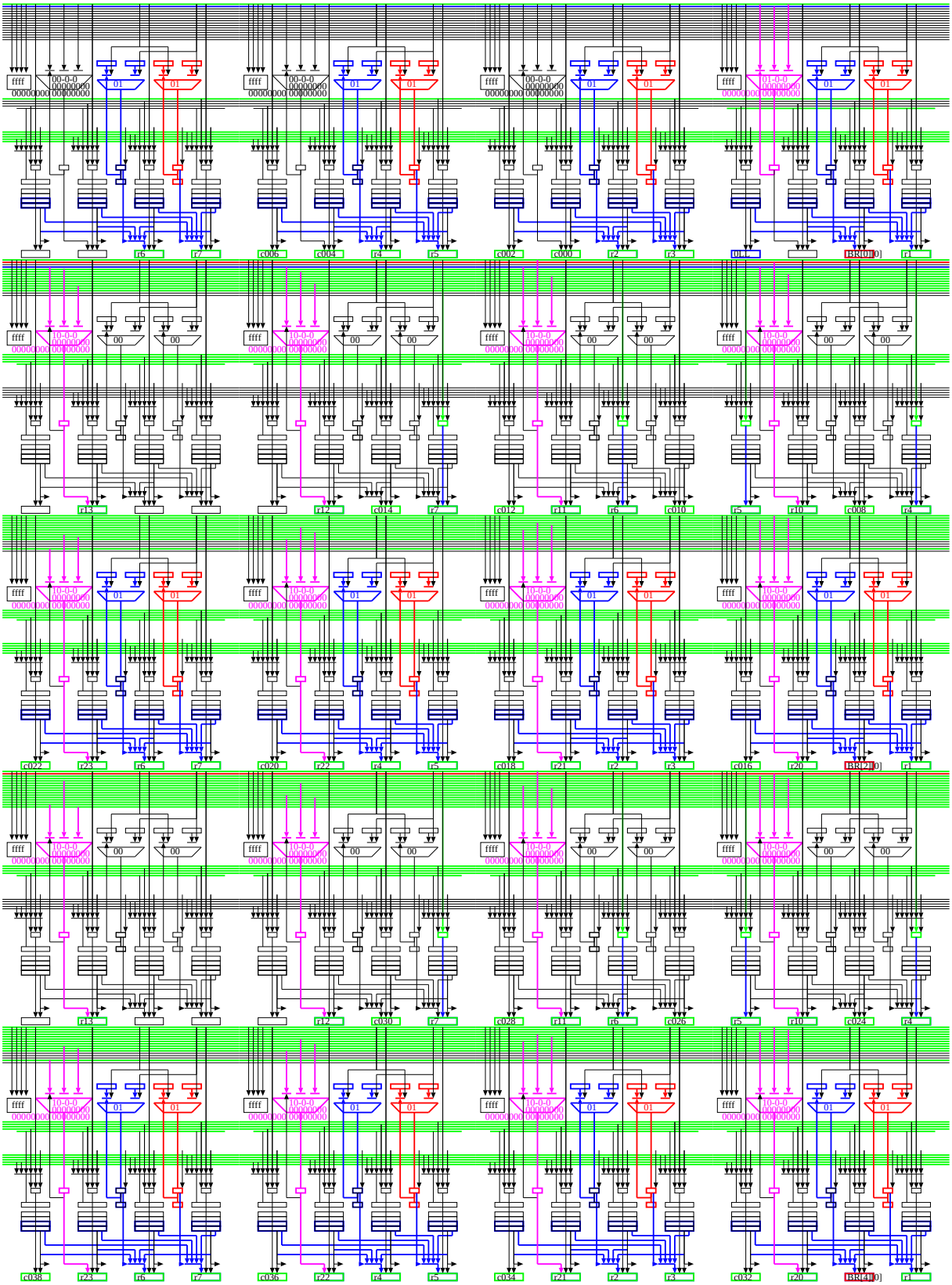


Figure.3.2: 16x16 畳み込み演算

3.3 Examples (2D-imaging)

3.3.1 Tone_curve

```
void tone_curve(r, d, t)
     unsigned int *r, *d;
     unsigned char *t;
{
#ifndef EMAX
  int j;
  for (j=0; j<WD; j++) {
    *d = ((t)[*r>>24])<<24 | (t[256+((*r>>16)&255)])<<16 | (t[512+((*r>>8)&255)])<<8;
    r++; d++;
  }
```

```
#else
  int  t1 = t;
  int  t2 = t+256;
  int  t3 = t+512;
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define TONE_CURVE_64BIT
#ifdef TONE_CURVE_32BIT
  int loop=WD;
//EMAX5A begin tone_curve mapdist=0
  while (loop--) {
    mop(OP_LDWR,  1, &BR[0][1][1], (Ull)(r++), 0LL,         MSK_D0, (Ull)r, 320/2,   0, 0, (Ull)NULL, 320/2);   /* stage#0 */
    mop(OP_LDUBR, 1, &BR[1][1][1], (Ull)t1,    BR[0][1][1], MSK_B3, (Ull)t1, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    mop(OP_LDUBR, 1, &BR[1][2][1], (Ull)t2,    BR[0][1][1], MSK_B2, (Ull)t2, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    mop(OP_LDUBR, 1, &BR[1][3][1], (Ull)t3,    BR[0][1][1], MSK_B1, (Ull)t3, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    exe(OP_MMRG, &r1, BR[1][1][1], EXP_H3210,  BR[1][2][1], EXP_H3210, BR[1][3][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
    mop(OP_STWR,  3, &r1,          (Ull)(d++), 0LL,         MSK_D0, (Ull)d, 320/2,   0, 0, (Ull)NULL, 320/2);   /* stage#2 */
  }
//EMAX5A end
#endif
#ifdef TONE_CURVE_64BIT
  Ull *rr = r;
  Ull *dd = d;
  int loop=WD/2;
//EMAX5A begin tone_curve mapdist=0
  while (loop--) {
    mop(OP_LDR,   1, &BR[0][1][1], (Ull)(rr++), 0LL,        MSK_D0, (Ull)r, 320/2,   0, 0, (Ull)NULL, 320/2);   /* stage#0 */
    mop(OP_LDUBR, 1, &BR[1][1][1], (Ull)t1,    BR[0][1][1], MSK_B3, (Ull)t1, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    mop(OP_LDUBR, 1, &BR[1][1][0], (Ull)t1,    BR[0][1][1], MSK_B7, (Ull)t1, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    mop(OP_LDUBR, 1, &BR[1][2][1], (Ull)t2,    BR[0][1][1], MSK_B2, (Ull)t2, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    mop(OP_LDUBR, 1, &BR[1][2][0], (Ull)t2,    BR[0][1][1], MSK_B6, (Ull)t2, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    mop(OP_LDUBR, 1, &BR[1][3][1], (Ull)t3,    BR[0][1][1], MSK_B1, (Ull)t3, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    mop(OP_LDUBR, 1, &BR[1][3][0], (Ull)t3,    BR[0][1][1], MSK_B5, (Ull)t3, 64/2,  0,  0, (Ull)NULL, 64/2);   /* stage#1 */
    exe(OP_CCAT,  &r1, BR[1][1][1], EXP_H3210, BR[1][1][1], EXP_H3210,        0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
    exe(OP_CCAT,  &r2, BR[1][2][1], EXP_H3210, BR[1][2][1], EXP_H3210,        0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
    exe(OP_CCAT,  &r3, BR[1][3][1], EXP_H3210, BR[1][3][1], EXP_H3210,        0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
    exe(OP_MMRG,  &r0,          r1, EXP_H3210, r2, EXP_H3210, r3, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
    mop(OP_STR,   3, &r0,          (Ull)(dd++), 0LL,        MSK_D0, (Ull)d, 320/2,   0, 0, (Ull)NULL, 320/2);   /* stage#2 */
  }
//EMAX5A end
#endif
#endif
}
```

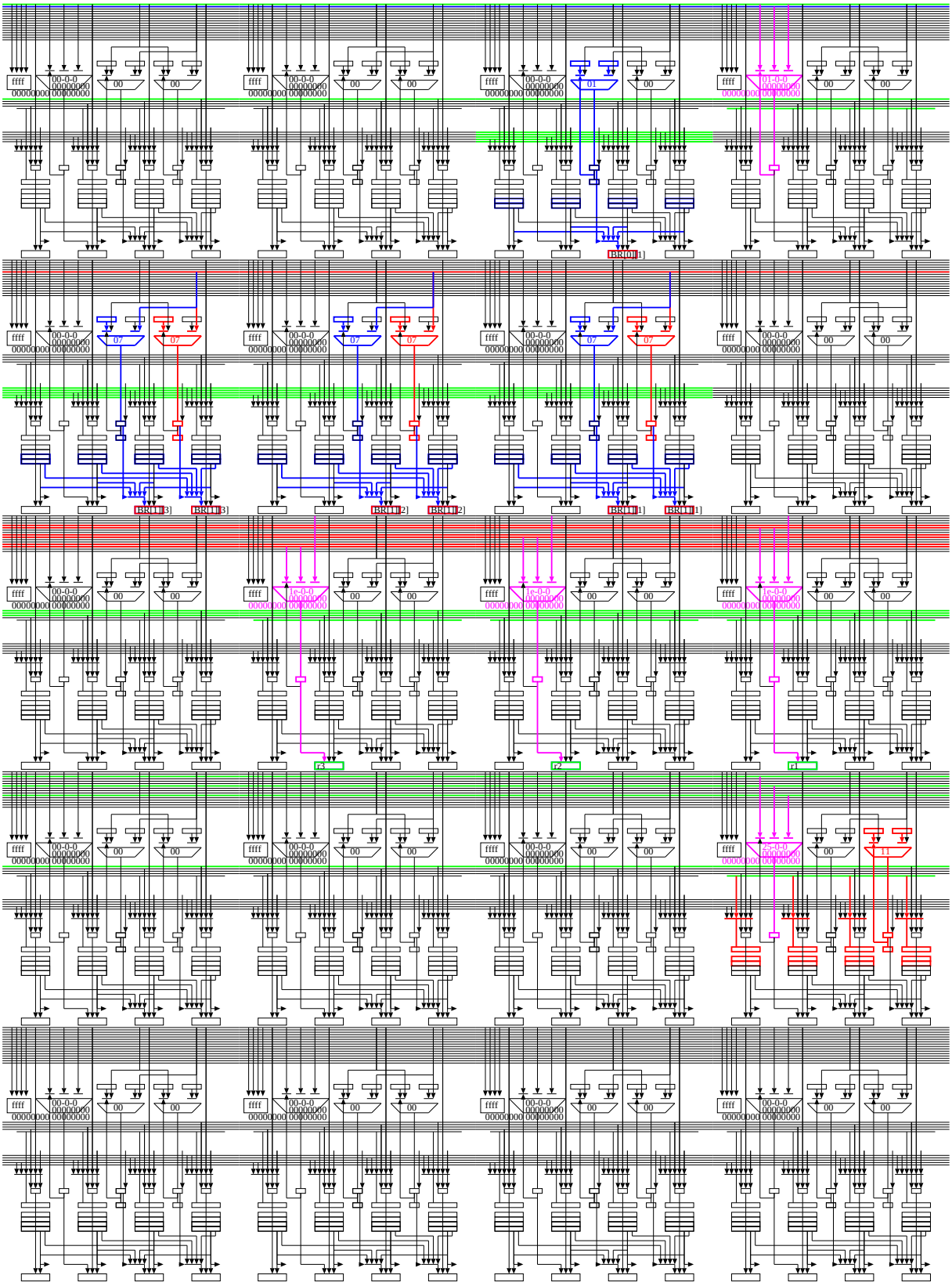


Figure.3.3: Tone_curve

3.3.2 Hokan1 with stencil

```
void hokan1(c, p, s)
     unsigned int *c, *p;
     unsigned short *s; /*[WD/4][8];*/
     /*hokan1(&W[i*WD], &R[(i+j)*WD], &SAD1[i/4][j+4]);*/
{
#ifndef EMAX
  int j;
  for (j=0; j<WD; j++) {
    int j2 = j/4*4;
    int k = j%4*2;                                                                                                                                  /* j2+k:0,2,4,6; 4,6,8,10; 8,10,12,14; 12,14,16,18; */
    * s    += df(c[j2],p[j2+k-4]) + df(c[j2+1],p[j2+k-3]) + df(c[j2+2],p[j2+k-2]) + df(c[j2+3],p[j2+k-1]); /* p[-4],p[-3],p[-2],p[-1] -> p[-2],p[-1],p[0],p[1] */
    *(s+1) += df(c[j2],p[j2+k-3]) + df(c[j2+1],p[j2+k-2]) + df(c[j2+2],p[j2+k-1]) + df(c[j2+3],p[j2+k  ]); /* p[-3],p[-2],p[-1],p[ 0] -> p[-1],p[ 0],p[1],p[2] */
    s += 2;
  }
```

```
#else
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define HOKAN1_32BIT
#ifdef HOKAN1_32BIT
  Sll  j=-1;
  Uint *s0 = s;
  Uint *s1 = s;
  int  loop=WD;
//EMAX5A begin hokan1 mapdist=0
  while (loop--) {
    /*@0,1*/ exe(OP_ADD,     &j,           j,      EXP_H3210, 1LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@1,1*/ exe(OP_NOP,     &r10,         j,      EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND, ~3LL, OP_SLL, 2LL);
    /*@1,2*/ exe(OP_NOP,     &r11,         j,      EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND,  3LL, OP_SLL, 3LL);
    /*@2,0*/ exe(OP_ADD,     &r12,         (Ull)c, EXP_H3210, r10, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@2,1*/ exe(OP_ADD3,    &r13,         (Ull)p, EXP_H3210, r10, EXP_H3210, r11, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@3,0*/ mop(OP_LDWR, 1, &r0,          r12,         0LL,  MSK_D0,  (Ull)c, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@3,1*/ mop(OP_LDWR, 1, &r1,          r12,         4LL,  MSK_D0,  (Ull)c, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@3,2*/ mop(OP_LDWR, 1, &r2,          r12,         8LL,  MSK_D0,  (Ull)c, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@3,3*/ mop(OP_LDWR, 1, &r3,          r12,        12LL,  MSK_D0,  (Ull)c, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@4,0*/ mop(OP_LDWR, 1, &BR[4][0][1], r13,       -16LL,  MSK_D0,  (Ull)p, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@4,1*/ mop(OP_LDWR, 1, &r25,         r13,       -12LL,  MSK_D0,  (Ull)p, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@4,2*/ mop(OP_LDWR, 1, &r26,         r13,        -8LL,  MSK_D0,  (Ull)p, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@4,3*/ mop(OP_LDWR, 1, &r27,         r13,        -4LL,  MSK_D0,  (Ull)p, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@4,3*/ mop(OP_LDWR, 1, &r28,         r13,         0LL,  MSK_D0,  (Ull)p, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@5,0*/ exe(OP_MSSAD,   &r11,         0LL,    EXP_H3210, r0,  EXP_H3210, r25, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@5,1*/ exe(OP_MSSAD,   &r13,         0LL,    EXP_H3210, r1,  EXP_H3210, r26, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@5,2*/ exe(OP_MSSAD,   &r15,         0LL,    EXP_H3210, r2,  EXP_H3210, r27, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@5,3*/ exe(OP_MSSAD,   &r17,         0LL,    EXP_H3210, r3,  EXP_H3210, r28, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@6,0*/ exe(OP_MSSAD,   &r10,         0LL,    EXP_H3210, r0,  EXP_H3210, BR[4][0][1], EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@6,1*/ exe(OP_MSSAD,   &r12,         0LL,    EXP_H3210, r1,  EXP_H3210, r25, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@6,2*/ exe(OP_MSSAD,   &r14,         0LL,    EXP_H3210, r2,  EXP_H3210, r26, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@6,3*/ exe(OP_MSSAD,   &r16,         0LL,    EXP_H3210, r3,  EXP_H3210, r27, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@7,0*/ exe(OP_MAUH,    &r20,         r10,    EXP_H3210, r12, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@7,1*/ exe(OP_MAUH,    &r21,         r11,    EXP_H3210, r13, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@7,2*/ exe(OP_MAUH,    &r24,         r14,    EXP_H3210, r16, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@7,3*/ exe(OP_MAUH,    &r25,         r15,    EXP_H3210, r17, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@8,0*/ exe(OP_MAUH,    &r10,         r20,    EXP_H3210, r24, EXP_H3210, 0LL, EXP_H3210, OP_SUMHL,0LL, OP_NOP, 0LL);
    /*@8,1*/ exe(OP_MAUH,    &r11,         r21,    EXP_H3210, r25, EXP_H3210, 0LL, EXP_H3210, OP_SUMHH,0LL, OP_NOP, 0LL);
    /*@8,2*/ mop(OP_LDWR, 1, &r0,          (Ull)(s0++), 0LL,  MSK_D0, (Ull)s0, 320/2, 0, 1, (Ull)NULL, 320/2);
    /*@9,0*/ exe(OP_MAUH3,   &r1,          r0,     EXP_H3210, r10, EXP_H3210, r11, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
    /*@9,0*/ mop(OP_STWR, 3, &r1,          (Ull)(s1++), 0LL,  MSK_D0, (Ull)s1, 320/2, 0, 0, (Ull)NULL, 320/2);
  }
//EMAX5A end
#endif
#ifdef HOKAN1_64BIT
  int loop=WD/2;
//EMAX5A begin hokan1 mapdist=0
  while (loop--) {
  }
//EMAX5A end
#endif
#endif
}
```

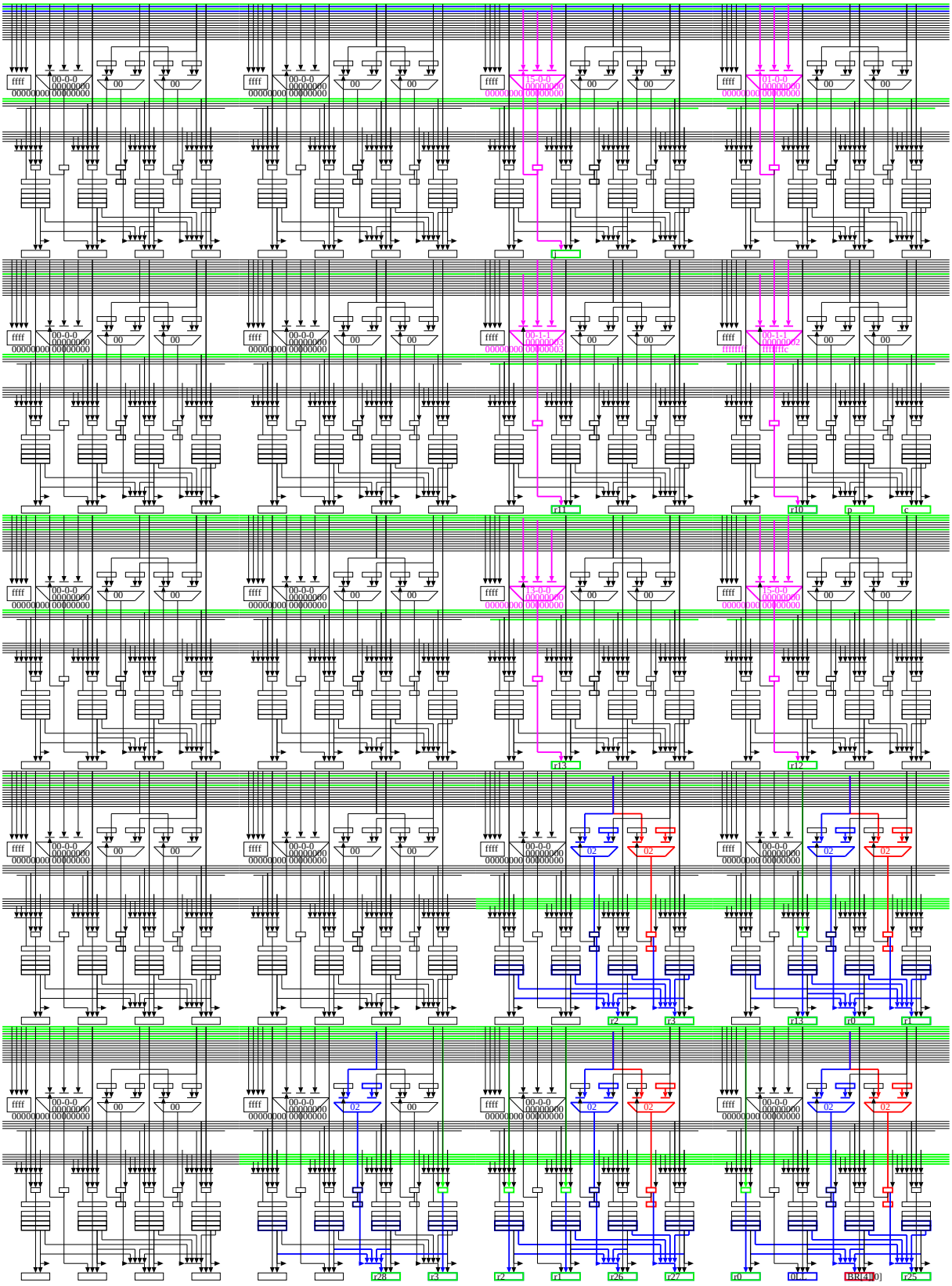


Figure.3.4: Hokan1

3.3.3 Hokan2 with stencil

```
void hokan2(s, sminxy, k)
     unsigned short *s; /*[WD/4][8];*/
     unsigned int *sminxy;
     int k;
{
#ifndef EMAX
  int j;
  for (j=0; j<WD; j++) { /* j%4==0 の時のみ sminxy[j] に有効値. 他はゴミ */
    int l1 = ((-2)<<24)|k|*(s  );
    int l2 = ((-1)<<24)|k|*(s+1);
    int l3 = ((-1)<<24)|k|*(s+2);
    int l4 = (( 0)<<24)|k|*(s+3);
    int l5 = (( 0)<<24)|k|*(s+4);
    int l6 = (( 0)<<24)|k|*(s+5);
    int l7 = (( 1)<<24)|k|*(s+6);
    int l8 = (( 1)<<24)|k|*(s+7);
    if ((sminxy[j]&0xffff) > *(s  )) sminxy[j] = l1;
    if ((sminxy[j]&0xffff) > *(s+1)) sminxy[j] = l2;
    if ((sminxy[j]&0xffff) > *(s+2)) sminxy[j] = l3;
    if ((sminxy[j]&0xffff) > *(s+3)) sminxy[j] = l4;
    if ((sminxy[j]&0xffff) > *(s+4)) sminxy[j] = l5;
    if ((sminxy[j]&0xffff) > *(s+5)) sminxy[j] = l6;
    if ((sminxy[j]&0xffff) > *(s+6)) sminxy[j] = l7;
    if ((sminxy[j]&0xffff) > *(s+7)) sminxy[j] = l8;
    s += 2;
  }
```

```
#else
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define HOKAN2_32BIT
#ifdef HOKAN2_32BIT
  Uint *s0 = s+0;
  Uint *s1 = s+2;
  Uint *s2 = s+4;
  Uint *s3 = s+6;
  Uint *smin0 = sminxy;
  Uint *smin1 = sminxy;
  int  loop=WD;
//EMAX5A begin hokan2 mapdist=0
  while (loop--) {
    /*@0,0*/ mop(OP_LDWR, 1, &r10,         (Ull)(s0++),             0LL, MSK_D0,  (Ull)s0, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@0,1*/ exe(OP_NOP,     &r28,  (-2LL<<24),    EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,   k, OP_NOP, 0LL);
    /*@0,1*/ mop(OP_LDWR, 1, &r12,         (Ull)(s1++),             0LL, MSK_D0,  (Ull)s0, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@0,2*/ exe(OP_NOP,     &r29,  (-1LL<<24),    EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,   k, OP_NOP, 0LL);
    /*@0,2*/ mop(OP_LDWR, 1, &r14,         (Ull)(s2++),             0LL, MSK_D0,  (Ull)s0, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@0,3*/ exe(OP_NOP,     &r31,  ( 1LL<<24),    EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,   k, OP_NOP, 0LL);
    /*@0,3*/ mop(OP_LDWR, 1, &r16,         (Ull)(s3++),             0LL, MSK_D0,  (Ull)s0, 320/2, 0, 0, (Ull)NULL, 320/2);
    /*@1,0*/ exe(OP_MINL3,   &r10,         r29,    EXP_H3210, r28, EXP_H3210, r10, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@1,1*/ exe(OP_MINL3,   &r12,         k,      EXP_H3210, r29, EXP_H3210, r12, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@1,2*/ exe(OP_MINL3,   &r14,         k,      EXP_H3210, k,   EXP_H3210, r14, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@1,3*/ exe(OP_MINL3,   &r16,         r31,    EXP_H3210, r31, EXP_H3210, r16, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@2,0*/ exe(OP_MINL,    &r20,         r10,    EXP_H3210, r12, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@2,2*/ exe(OP_MINL,    &r24,         r14,    EXP_H3210, r16, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@3,0*/ exe(OP_MINL,    &r10,         r20,    EXP_H3210, r24, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@3,0*/ mop(OP_LDWR, 1, &r11,         (Ull)(smin0++),    0LL, MSK_D0, (Ull)smin0, 320/2, 0, 1, (Ull)NULL, 320/2);
    /*@4,0*/ exe(OP_MINL,    &r31,         r10,    EXP_H3210, r11, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL);
    /*@4,0*/ mop(OP_STWR, 3, &r31,         (Ull)(smin1++),    0LL, MSK_D0, (Ull)smin1, 320/2, 0, 0, (Ull)NULL, 320/2);
  }
//EMAX5A end
#endif
#ifdef HOKAN2_64BIT
  int loop=WD/2;
//EMAX5A begin hokan2 mapdist=0
  while (loop--) {
  }
//EMAX5A end
#endif
#endif
}
```

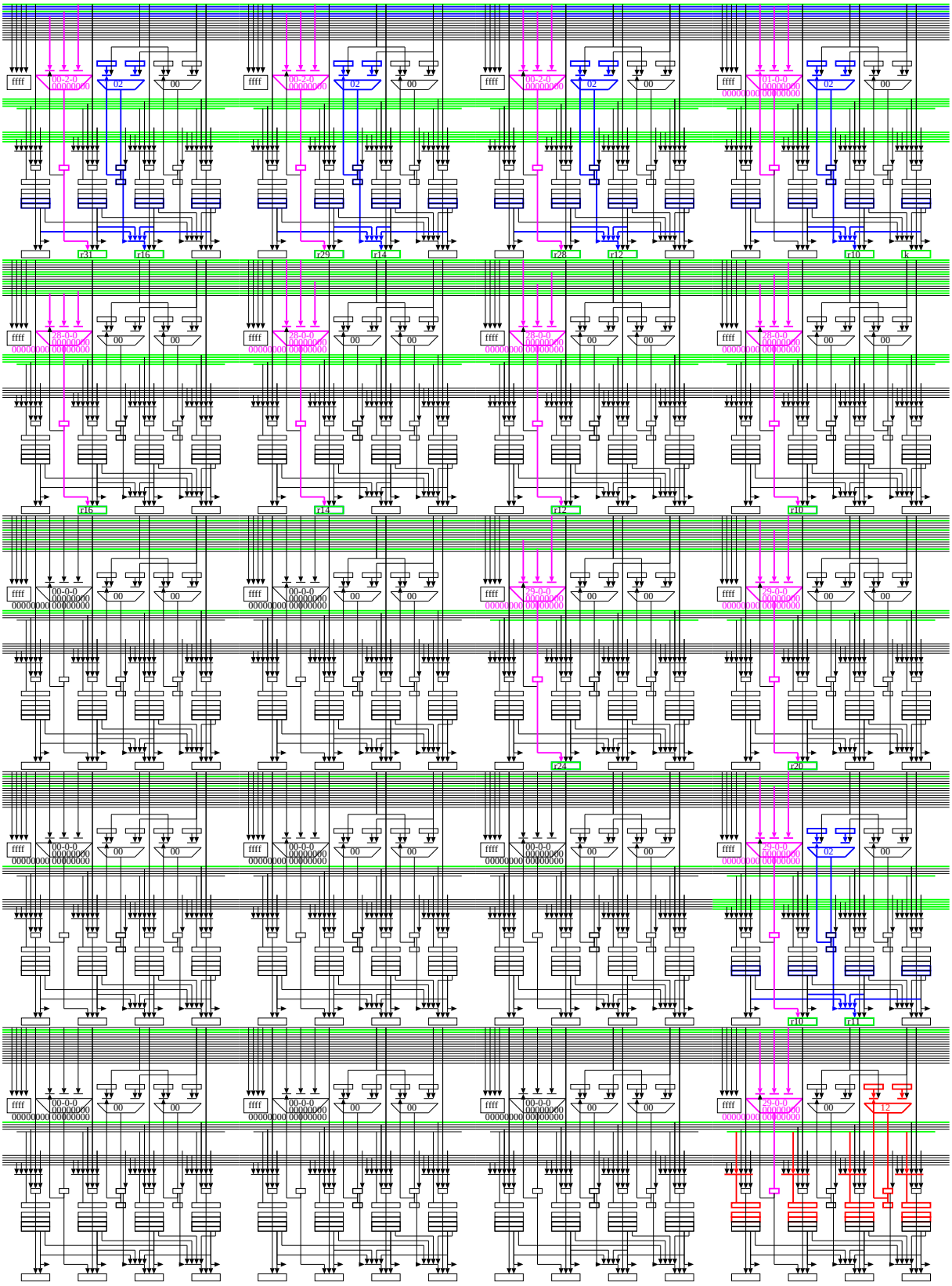


Figure.3.5: Hokan2

3.3.4 Hokan3 with stencil

```
void hokan3(sminxy, r, d, k)
     unsigned int *sminxy;
     unsigned int *r, *d;
     int k;
{
#ifndef EMAX
  int j;
  for (j=0; j<WD; j++) {
    int x = (int) sminxy[j/4*4]>>24;
    int y = (int)(sminxy[j/4*4]<<8)>>24;
    if (y == k) d[j] = r[j+x];
  }
```

```
#else
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
  Ull  ex0;
#define HOKAN3_32BIT
#ifdef HOKAN3_32BIT
  Sll  j=-1;
  int  loop=WD;
//EMAX5A begin hokan3 mapdist=0
  while (loop--) {
    /*@0,1*/ exe(OP_ADD,       &j,    j,    EXP_H3210, 1LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@1,1*/ exe(OP_NOP,       &r12,  j,    EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND, ~3LL,          OP_SLL,  2LL);
    /*@1,2*/ exe(OP_NOP,       &r14,  j,    EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,   0LL,          OP_SLL,  2LL);
    /*@2,0*/ mop(OP_LDWR, 1,   &r16,  (Ull)sminxy,     r12, MSK_D0,    (Ull)sminxy,       320/2,     0, 0, (Ull)NULL,       320/2);
    /*@3,0*/ exe(OP_ADD,       &r13,  r,    EXP_H3210, r14, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,1*/ exe(OP_NOP,       &r17,  r16,  EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND,  0xff000000LL, OP_SRAA, 22LL);
    /*@3,2*/ exe(OP_NOP,       &r18,  r16,  EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_AND,  0x00ff0000LL, OP_SRAB, 16LL);
    /*@4,0*/ exe(OP_CMP_EQ,    &r10,  r18,  EXP_H3210, k,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,0*/ mop(OP_LDWR, 1,   &r16,  (Ull)r13,        r17, MSK_D0,    (Ull)r,        320/2,         0, 0, (Ull)NULL,       320/2);
    /*@5,0*/ cex(OP_CEXE,      &ex0,  0LL,  0LL, 0LL,  r10, 0x0002LL);
    /*@5,0*/ mop(OP_STWR, ex0, &r16,  (Ull)(d++),      0LL, MSK_D0,    (Ull)d,        320/2,         0, 0, (Ull)NULL,       320/2);
  }
//EMAX5A end
#endif
#ifdef HOKAN3_64BIT
  int loop=WD/2;
//EMAX5A begin hokan3 mapdist=0
  while (loop--) {
  }
//EMAX5A end
#endif
#endif
}
```

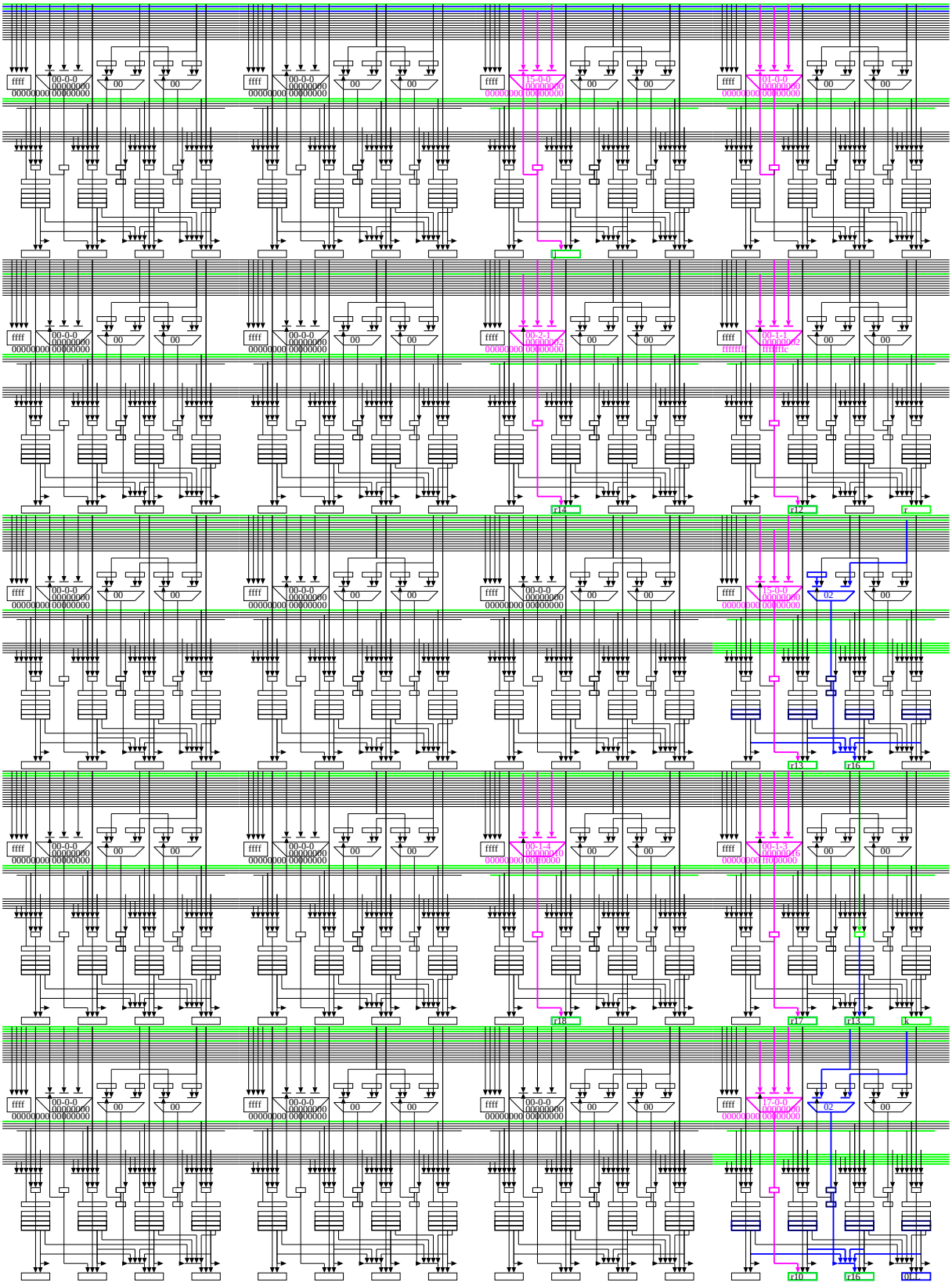


Figure.3.6: Hokan3

3.3.5 Expand4k with stencil

```
void expand4k(p, r, kad, sk1, sk2)
     unsigned int *p, *r;
     int kad, sk1, sk2;
{
#ifndef EMAX
  int j;
  unsigned int ph, pl, x;

  for (j=0; j<1024; j++) { /* 本当は 4095 まで */
    int p1 = j*WD/1024;
    int lfraq = (((j*WD)<<4)/1024)&15; /* 4bit */
    int lad = 16-ad(lfraq,8);
    int sl1 = ss(lfraq,8);
    int sl2 = ss(8,lfraq);
    int r1 = kad*lad; /* 4bit*4bit */
    int r3 = kad*sl1; /* 4bit*4bit */
    int r2 = kad*sl2; /* 4bit*4bit */
    int r5 = sk1*lad; /* 4bit*4bit */
    int r9 = sk1*sl1; /* 4bit*4bit */
    int r8 = sk1*sl2; /* 4bit*4bit */
    int r4 = sk2*lad; /* 4bit*4bit */
    int r7 = sk2*sl1; /* 4bit*4bit */
    int r6 = sk2*sl2; /* 4bit*4bit */
    ph = madd(mmul(b2h(p[p1     ], 1), r1), mmul(b2h(p[p1-1], 1), r2));
    ph = madd(mmul(b2h(p[p1   +1], 1), r3), ph);
    ph = madd(mmul(b2h(p[p1-WD  ], 1), r4), ph);
    ph = madd(mmul(b2h(p[p1+WD  ], 1), r5), ph);
    ph = madd(mmul(b2h(p[p1-WD-1], 1), r6), ph);
    ph = madd(mmul(b2h(p[p1-WD+1], 1), r7), ph);
    ph = madd(mmul(b2h(p[p1+WD-1], 1), r8), ph);
    ph = madd(mmul(b2h(p[p1+WD+1], 1), r9), ph);
    pl = madd(mmul(b2h(p[p1     ], 0), r1), mmul(b2h(p[p1-1], 0), r2));
    pl = madd(mmul(b2h(p[p1   +1], 0), r3), pl);
    pl = madd(mmul(b2h(p[p1-WD  ], 0), r4), pl);
    pl = madd(mmul(b2h(p[p1+WD  ], 0), r5), pl);
    pl = madd(mmul(b2h(p[p1-WD-1], 0), r6), pl);
    pl = madd(mmul(b2h(p[p1-WD+1], 0), r7), pl);
    pl = madd(mmul(b2h(p[p1+WD-1], 0), r8), pl);
    pl = madd(mmul(b2h(p[p1+WD+1], 0), r9), pl);
    *r = h2b(msrl(ph, 8), 1) | h2b(msrl(pl, 8), 0);
    r++;
  }
```

```
#else
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define EXPAND4K_32BIT
#ifdef EXPAND4K_32BIT
  Sll  j=-320;
  int  p0=p-320;
  int  p1=p;
  int  p2=p+320;
  int  loop=1024;
//EMAX5A begin expand4k mapdist=0
  while (loop--) {
    /*@0,1*/ exe(OP_ADD,       &j,    j,    EXP_H3210, 320LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
    /*@1,0*/ exe(OP_NOP,       &r0,   j,    EXP_H3210, 0LL,   EXP_H3210, 0LL, EXP_H3210, OP_AND, ~1023LL,  OP_SRL,  8LL);
    /*@1,1*/ exe(OP_NOP,       &r4,   j,    EXP_H3210, 0LL,   EXP_H3210, 0LL, EXP_H3210, OP_AND,  0x3c0LL, OP_SRL,  6LL);
    /*@2,0*/ exe(OP_ADD,       &r0,   p,    EXP_H3210, r0,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
    /*@2,1*/ exe(OP_MSUH,      &r1,   r4,   EXP_H3210, 8LL,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
    /*@2,2*/ exe(OP_MSUH,      &r2,   8LL,  EXP_H3210, r4,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
    /*@2,3*/ exe(OP_MSSAD,     &r3,   0LL,  EXP_H3210, r4,    EXP_H3210, 8LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
    /*@3,3*/ exe(OP_MSUH,      &r3,   16LL, EXP_H3210, r3,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);

    /*@4,1*/ exe(OP_MLUH,      &r21,  sk2,  EXP_H3210, r1,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
    /*@4,1*/ mop(OP_LDWR, 1,   &r10,  r0,   -1276, MSK_D0,   (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@4,2*/ exe(OP_MLUH,      &r22,  sk2,  EXP_H3210, r2,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
    /*@4,2*/ mop(OP_LDWR, 1,   &r11,  r0,   -1284, MSK_D0,   (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@4,3*/ exe(OP_MLUH,      &r23,  sk2,  EXP_H3210, r3,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
    /*@4,3*/ mop(OP_LDWR, 1,   &r12,  r0,   -1280, MSK_D0,   (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
         :
   /*@10,0*/ exe(OP_MAUH3,     &r19,  r13,  EXP_H3210, r14,   EXP_H3210, r15, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@10,1*/ exe(OP_MLUH,      &r21,  sk1,  EXP_H3210, r1,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@10,1*/ mop(OP_LDWR, 1,   &r10,  r0,    1284, MSK_D0,   (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
   /*@10,2*/ exe(OP_MLUH,      &r22,  sk1,  EXP_H3210, r2,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@10,2*/ mop(OP_LDWR, 1,   &r11,  r0,    1276, MSK_D0,   (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
   /*@10,3*/ exe(OP_MLUH,      &r23,  sk1,  EXP_H3210, r3,    EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@10,3*/ mop(OP_LDWR, 1,   &r12,  r0,    1280, MSK_D0,   (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);

   /*@11,1*/ exe(OP_MLUH,      &r13,  r10,  EXP_B5410, r21,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@11,2*/ exe(OP_MLUH,      &r14,  r11,  EXP_B5410, r22,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@11,3*/ exe(OP_MLUH,      &r15,  r12,  EXP_B5410, r23,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);

   /*@12,0*/ exe(OP_MAUH3,     &r20,  r13,  EXP_H3210, r14,   EXP_H3210, r15, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@12,1*/ exe(OP_MLUH,      &r13,  r10,  EXP_B7632, r21,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@12,2*/ exe(OP_MLUH,      &r14,  r11,  EXP_B7632, r22,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@12,3*/ exe(OP_MLUH,      &r15,  r12,  EXP_B7632, r23,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);

   /*@13,0*/ exe(OP_MAUH3,     &r21,  r13,  EXP_H3210, r14,   EXP_H3210, r15, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);

   /*@14,0*/ exe(OP_MAUH3,     &r21,  r17,  EXP_H3210, r19,   EXP_H3210, r21, EXP_H3210, OP_OR,   0LL,     OP_SRLM, 8LL);
   /*@14,1*/ exe(OP_MAUH3,     &r20,  r16,  EXP_H3210, r18,   EXP_H3210, r20, EXP_H3210, OP_OR,   0LL,     OP_SRLM, 8LL);

   /*@15,0*/ exe(OP_MH2BW,     &r31,  r21,  EXP_H3210, r20,   EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,     OP_NOP,  0LL);
   /*@15,0*/ mop(OP_STWR, 3,   &r31,  (Ull)(r++),     0LL,   MSK_D0,    (Ull)r,       1024/2,        0,  0, (Ull)NULL,       1024/2);
  }
//EMAX5A end
#endif
#ifdef EXPAND4K_64BIT
  int loop=WD/2;
//EMAX5A begin expand4k mapdist=0
  while (loop--) {
  }
//EMAX5A end
#endif
#endif
}
```

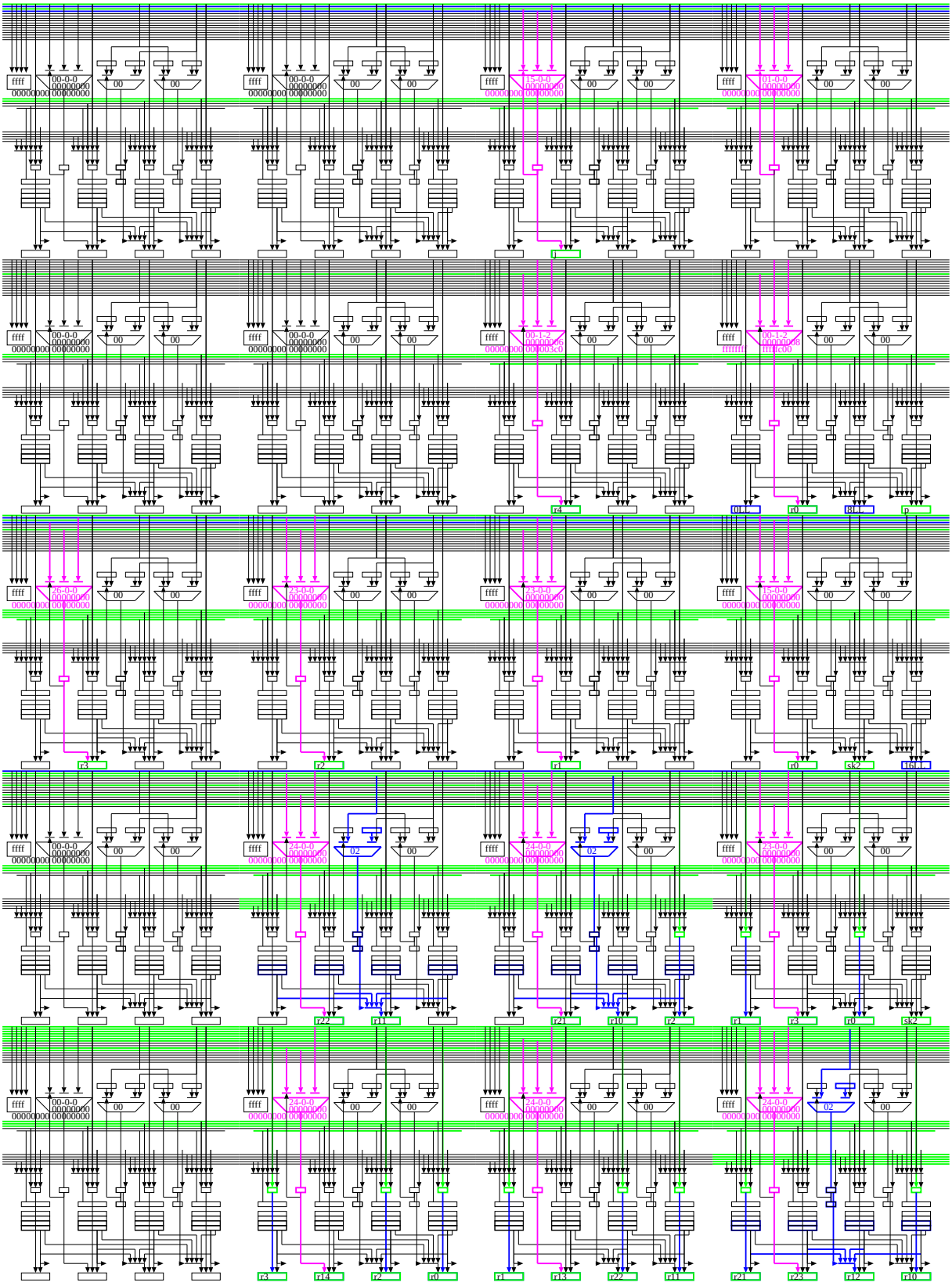


Figure.3.7: Expand4k

3.3.6 Unsharp with stencil

```
void unsharp(p, r)
     unsigned char *p;
     unsigned char *r;
{
#ifndef EMAX
  int t0,t1,t2;
  int j, k;
  int p0 = ((0  )*WD+(1  ))*4;  // p1 p5 p2
  int p1 = ((0-1)*WD+(1-1))*4;  // p6 p0 p7
  int p2 = ((0-1)*WD+(1+1))*4;  // p3 p8 p4
  int p3 = ((0+1)*WD+(1-1))*4;
  int p4 = ((0+1)*WD+(1+1))*4;
  int p5 = ((0-1)*WD+(1  ))*4;
  int p6 = ((0  )*WD+(1-1))*4;
  int p7 = ((0  )*WD+(1+1))*4;
  int p8 = ((0+1)*WD+(1  ))*4;
  for (j=0; j<WD; j++) {
    r[p0+0] = 0;

    t0 = p[p0+1];
    t1 = p[p1+1] + p[p2+1] + p[p3+1] + p[p4+1];
    t2 = p[p5+1] + p[p6+1] + p[p7+1] + p[p8+1];
    r[p0+1] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    t0 = p[p0+2];
    t1 = p[p1+2] + p[p2+2] + p[p3+2] + p[p4+2];
    t2 = p[p5+2] + p[p6+2] + p[p7+2] + p[p8+2];
    r[p0+2] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    t0 = p[p0+3];
    t1 = p[p1+3] + p[p2+3] + p[p3+3] + p[p4+3];
    t2 = p[p5+3] + p[p6+3] + p[p7+3] + p[p8+3];
    r[p0+3] = limitRGB(( t0 * 239 - t1 * 13 - t2 * 15 - t2/4) >> 7);

    p0+=4; p1+=4; p2+=4; p3+=4; p4+=4; p5+=4; p6+=4; p7+=4; p8+=4;
  }
```

```
#else
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define UNSHARP_32BIT
#ifdef UNSHARP_32BIT
  Sll  j=p-4;
  int  p0=p-1280;
  int  p1=p;
  int  p2=p+1280;
  int  *rr=r;
  int  loop=WD;
//EMAX5A begin unsharp mapdist=1
  while (loop--) {
    /*@0,1*/ exe(OP_ADD,       &j,    j,    EXP_H3210, 4LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@1,0*/ mop(OP_LDWR, 1,   &r1,   j,     -1276, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,1*/ mop(OP_LDWR, 1,   &r2,   j,     -1284, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,2*/ mop(OP_LDWR, 1,   &r5,   j,     -1280, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@2,0*/ exe(OP_MAUH,      &r11,  r1,   EXP_B5410, r2,  EXP_B5410, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@2,0*/ mop(OP_LDWR, 1,   &r6,   j,      4,    MSK_D0,    (Ull)p1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,1*/ exe(OP_MAUH,      &r12,  r1,   EXP_B7632, r2,  EXP_B7632, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@2,1*/ mop(OP_LDWR, 1,   &r7,   j,     -4,    MSK_D0,    (Ull)p1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,2*/ mop(OP_LDWR, 1,   &r0,   j,      0,    MSK_D0,    (Ull)p1,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@3,0*/ exe(OP_MLUH,      &r20,  r0,   EXP_B5410, 239, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,0*/ mop(OP_LDWR, 1,   &r3,   j,      1284, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@3,1*/ exe(OP_MLUH,      &r21,  r0,   EXP_B7632, 239, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,1*/ mop(OP_LDWR, 1,   &r4,   j,      1276, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@3,2*/ exe(OP_MAUH,      &r15,  r5,   EXP_B5410, r6,  EXP_B5410, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,2*/ mop(OP_LDWR, 1,   &r8,   j,      1280, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@3,3*/ exe(OP_MAUH,      &r16,  r5,   EXP_B7632, r6,  EXP_B7632, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@4,0*/ exe(OP_MAUH3,     &r11,  r3,   EXP_B5410, r4,  EXP_B5410, r11, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,1*/ exe(OP_MAUH3,     &r12,  r3,   EXP_B7632, r4,  EXP_B7632, r12, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@5,0*/ exe(OP_MLUH,      &r13,  r11,  EXP_H3210, 13,  EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@5,1*/ exe(OP_MLUH,      &r14,  r12,  EXP_H3210, 13,  EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@5,2*/ exe(OP_MAUH3,     &r15,  r7,   EXP_B5410, r8,  EXP_B5410, r15, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@5,3*/ exe(OP_MAUH3,     &r16,  r7,   EXP_B7632, r8,  EXP_B7632, r16, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@6,0*/ exe(OP_NOP,       &r7,   r15,  EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,   0LL,          OP_SRLM, 2LL);
    /*@6,1*/ exe(OP_MLUH,      &r17,  r15,  EXP_H3210, 15,  EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@6,2*/ exe(OP_NOP,       &r8,   r16,  EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,   0LL,          OP_SRLM, 2LL);
    /*@6,3*/ exe(OP_MLUH,      &r18,  r16,  EXP_H3210, 15,  EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@7,0*/ exe(OP_MSUH3,     &r10,  r20,  EXP_H3210, r7,  EXP_H3210, r17, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@7,2*/ exe(OP_MSUH3,     &r11,  r21,  EXP_H3210, r8,  EXP_H3210, r18, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@8,0*/ exe(OP_MSUH,      &r20,  r10,  EXP_H3210, r13, EXP_H3210, 0LL, EXP_H3210, OP_OR,   0LL,          OP_SRLM, 7LL);
    /*@8,2*/ exe(OP_MSUH,      &r21,  r11,  EXP_H3210, r14, EXP_H3210, 0LL, EXP_H3210, OP_OR,   0LL,          OP_SRLM, 7LL);

    /*@9,0*/ exe(OP_MH2BW,     &r31,  r21,  EXP_H3210, r20, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@9,0*/ mop(OP_STWR, 3,   &r31,  (Ull)(rr++),     0LL, MSK_D0,    (Ull)rr,       320/2,       0, 0, (Ull)NULL,       320/2);
  }
//EMAX5A end
#endif
#ifdef UNSHARP_64BIT
  int loop=WD/2;
//EMAX5A begin unsharp mapdist=0
  while (loop--) {
  }
//EMAX5A end
#endif
#endif
}
```

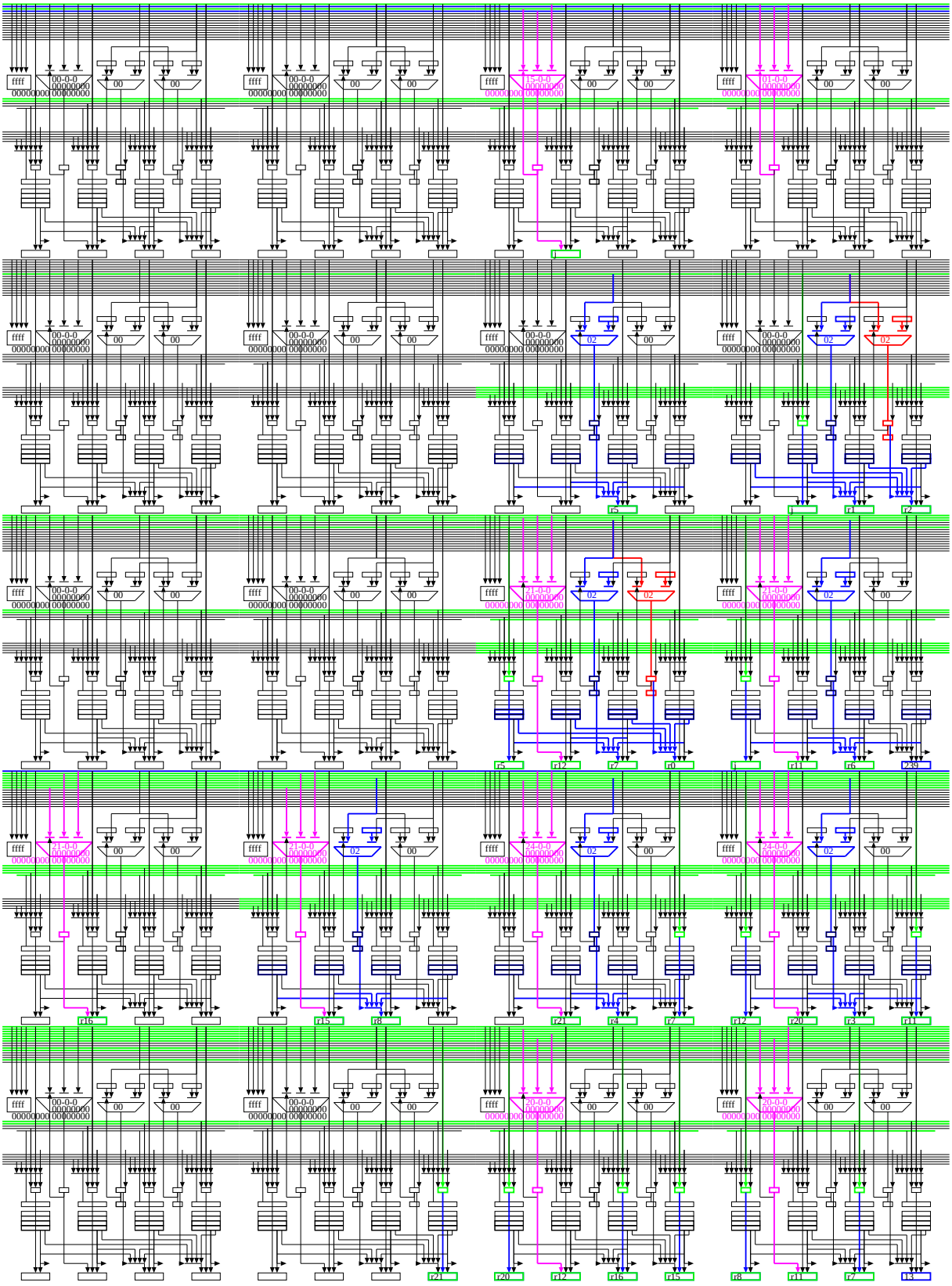


Figure.3.8: Unsharp

3.3.7 Blur with stencil

```
void blur(p, r)
     unsigned int *p, *r;
{
#ifndef EMAX
  int j, k, l;

  int p0 = (0  )*WD  ;
  int p1 = (0  )*WD  ;
  int p2 = (0  )*WD-1;
  int p3 = (0  )*WD+1;
  int p4 = (0-1)*WD  ;
  int p5 = (0+1)*WD  ;
  int p6 = (0-1)*WD-1;
  int p7 = (0-1)*WD+1;
  int p8 = (0+1)*WD-1;
  int p9 = (0+1)*WD+1;
  for (j=0; j<WD; j++) {
    unsigned int s0,s1,s2,s3,s4,s5,s6,s7,s8;
    unsigned int t0,t1,t2;
    s0=p[p1];s1=p[p2];s2=p[p3];s3=p[p4];s4=p[p5];s5=p[p6];s6=p[p7];s7=p[p8];s8=p[p9];
    /*
       5 3 6      5<3<★      5 3 2      5<3<★      5    3      5<  <★      5
       1 0 2      1<0<2      - 0 -          0          0          0          0    0 中間値確定
       7 4 8      ★<4<8      1 4 8      ★<4<8      4    8      ★<  <8      8
                                                                          */
    t0 = pmax3(s5,s1,s7); t1 = pmid3(s5,s1,s7); t2 = pmin3(s5,s1,s7);      s5 = t0; s1 = t1; s7 = t2;
    t0 = pmax3(s3,s0,s4); t1 = pmid3(s3,s0,s4); t2 = pmin3(s3,s0,s4);      s3 = t0; s0 = t1; s4 = t2;
    t0 = pmax3(s6,s2,s8); t1 = pmid3(s6,s2,s8); t2 = pmin3(s6,s2,s8);      s6 = t0; s2 = t1; s8 = t2;

    t0 = pmin3(s5,s3,s6); t1 = pmid3(s5,s3,s6);                            s5 = t0; s3 = t1;
    t0 = pmin3(s1,s0,s2); t1 = pmid3(s1,s0,s2); t2 = pmax3(s1,s0,s2);      s1 = t0; s0 = t1; s2 = t2;
    t0 = pmid3(s7,s4,s8); t1 = pmax3(s7,s4,s8);                            s4 = t0; s8 = t1;

    t0 = pmax2(s5,s1);    t1 = pmin2(s5,s1);                               s5 = t0; s1 = t1;
    t0 = pmax3(s3,s0,s4); t1 = pmid3(s3,s0,s4); t2 = pmin3(s3,s0,s4);      s3 = t0; s0 = t1; s4 = t2;
    t0 = pmax2(s2,s8);    t1 = pmin2(s2,s8);                               s2 = t0; s8 = t1;

    t0 = pmin3(s5,s3,s2); t1 = pmid3(s5,s3,s2);                            s5 = t0; s3 = t1;
    t0 = pmid3(s1,s4,s8); t1 = pmax3(s1,s4,s8);                            s4 = t0; s8 = t1;

    t0 = pmax2(s5,s4);    t1 = pmin2(s5,s4);                               s5 = t0; s4 = t1;
    t0 = pmax3(s3,s0,s8); t1 = pmid3(s3,s0,s8); t2 = pmin3(s3,s0,s8);      s3 = t0; s0 = t1; s8 = t2;

    s5 = pmin2(s5,s3);    s8 = pmax2(s4,s8);

    r[p0] = pmid3(s5,s0,s8);
    p0++; p1++; p2++; p3++; p4++; p5++; p6++; p7++; p8++; p9++;
  }
```

```
#else
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define BLUR_32BIT
#ifdef BLUR_32BIT
  Sll  j=p-1;
  int  p0=p-320;
  int  p1=p;
  int  p2=p+320;
  int  loop=WD;
//EMAX5A begin blur mapdist=1
  while (loop--) {
    /*@0,1*/ exe(OP_ADD,       &j,    j,    EXP_H3210, 4LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@1,0*/ mop(OP_LDWR, 1,   &r7,   j,     -1276, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,0*/ mop(OP_LDWR, 1,   &r1,   j,     -1280, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,1*/ mop(OP_LDWR, 1,   &r5,   j,     -1284, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@2,0*/ exe(OP_MMIN3,     &r17,  r7,   EXP_H3210, r1,  EXP_H3210, r5,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@2,1*/ exe(OP_MMID3,     &r11,  r7,   EXP_H3210, r1,  EXP_H3210, r5,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@2,2*/ exe(OP_MMAX3,     &r15,  r7,   EXP_H3210, r1,  EXP_H3210, r5,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@2,2*/ mop(OP_LDWR, 1,   &r4,   j,      4,    MSK_D0,    (Ull)p1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,2*/ mop(OP_LDWR, 1,   &r0,   j,      0,    MSK_D0,    (Ull)p1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,3*/ mop(OP_LDWR, 1,   &r3,   j,     -4,    MSK_D0,    (Ull)p1,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@3,0*/ exe(OP_MMIN3,     &r14,  r4,   EXP_H3210, r0,  EXP_H3210, r3,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,1*/ exe(OP_MMID3,     &r10,  r4,   EXP_H3210, r0,  EXP_H3210, r3,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,2*/ exe(OP_MMAX3,     &r13,  r4,   EXP_H3210, r0,  EXP_H3210, r3,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@3,2*/ mop(OP_LDWR, 1,   &r8,   j,      1284, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@3,2*/ mop(OP_LDWR, 1,   &r2,   j,      1280, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@3,3*/ mop(OP_LDWR, 1,   &r6,   j,      1276, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@4,0*/ exe(OP_MMIN3,     &r18,  r8,   EXP_H3210, r2,  EXP_H3210, r6,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,1*/ exe(OP_MMID3,     &r12,  r8,   EXP_H3210, r2,  EXP_H3210, r6,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,2*/ exe(OP_MMAX3,     &r16,  r8,   EXP_H3210, r2,  EXP_H3210, r6,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
        :
    /*step-6*/
   /*@12,3*/ exe(OP_MMAX,      &r8,   r14,  EXP_H3210, r18, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
   /*@12,2*/ exe(OP_MMIN,      &r5,   r15,  EXP_H3210, r13, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

   /*@13,0*/ exe(OP_MMID3,     &r31,  r5,   EXP_H3210, r10, EXP_H3210, r8,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@13,0*/ mop(OP_STWR, 3,  &r31,  (Ull)(r++),      0LL, MSK_D0,    (Ull)r,       320/2,         0, 0, (Ull)NULL,       320/2);
  }
//EMAX5A end
#endif
#ifdef BLUR_64BIT
  int loop=WD/2;
//EMAX5A begin blur mapdist=0
  while (loop--) {
  }
//EMAX5A end
#endif
#endif
}
```

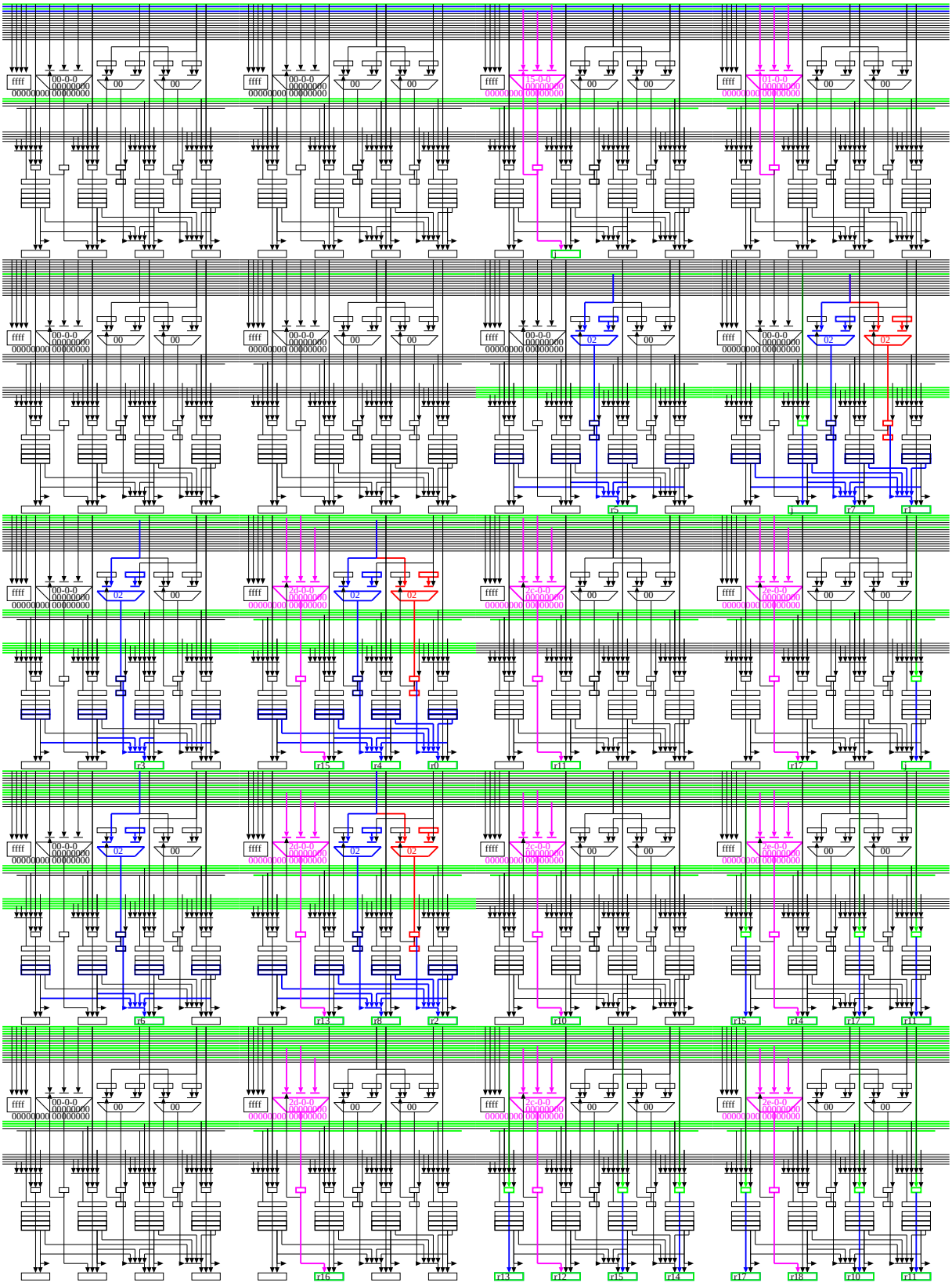


Figure.3.9: Blur

3.3.8 Edge with stencil

```
void edge(p, r)
     unsigned int *p;
     unsigned char *r;
{
#ifndef EMAX
  int j, k;

  int p0 = (0  )*WD  ;
  int p1 = (0-1)*WD-1;
  int p2 = (0+1)*WD+1;
  int p3 = (0-1)*WD  ;
  int p4 = (0+1)*WD  ;
  int p5 = (0-1)*WD+1;
  int p6 = (0+1)*WD-1;
  int p7 = (0  )*WD-1;
  int p8 = (0  )*WD+1;
  for (j=0; j<WD; j++) {
    int d1 = df(p[p1]&MASK,p[p2]&MASK)
           + df(p[p3]&MASK,p[p4]&MASK)
           + df(p[p5]&MASK,p[p6]&MASK)
           + df(p[p7]&MASK,p[p8]&MASK);
    /* 0 < d1(42) < 256*2*4 */
    r[p0] = d1 < EDGEDET ? 0 : PIXMAX;
    p0++; p1++; p2++; p3++; p4++; p5++; p6++; p7++; p8++;
  }
```

```
#else
  Ull  AR[16][4];    /* output registers in each unit */
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define EDGE_32BIT
#ifdef EDGE_32BIT
  Sll  j=p-1;
  int  p0=p-320;
  int  p1=p;
  int  p2=p+320;
  int  loop=WD;
//EMAX5A begin edge mapdist=1
  while (loop--) {
    /*@0,1*/ exe(OP_ADD,       &j,    j,    EXP_H3210, 4LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@1,0*/ mop(OP_LDWR, 1,   &r5,   j,     -1276, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,0*/ mop(OP_LDWR, 1,   &r3,   j,     -1280, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,1*/ mop(OP_LDWR, 1,   &r1,   j,     -1284, MSK_D0,    (Ull)p0,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@2,0*/ exe(OP_NOP,    &AR[2][0],0LL,  EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,  0LL,          OP_NOP,  0LL);
    /*@2,0*/ mop(OP_LDWR, 1,   &r8,   j,      4,    MSK_D0,    (Ull)p1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,0*/ mop(OP_LDWR, 1,   &r7,   j,     -4,    MSK_D0,    (Ull)p1,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@3,0*/ exe(OP_NOP,    &AR[3][0],0LL,  EXP_H3210, 0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,  0LL,          OP_NOP,  0LL);
    /*@3,0*/ mop(OP_LDWR, 1,   &r2,   j,      1284, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@3,0*/ mop(OP_LDWR, 1,   &r4,   j,      1280, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@3,1*/ mop(OP_LDWR, 1,   &r6,   j,      1276, MSK_D0,    (Ull)p2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@3,2*/ exe(OP_MSSAD,     &r7,   0LL,  EXP_H3210, r7,  EXP_H3210, r8,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@4,0*/ exe(OP_MSSAD,     &r1,   0LL,  EXP_H3210, r1,  EXP_H3210, r2,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,1*/ exe(OP_MSSAD,     &r3,   0LL,  EXP_H3210, r3,  EXP_H3210, r4,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,2*/ exe(OP_MSSAD,     &r5,   0LL,  EXP_H3210, r5,  EXP_H3210, r6,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@5,0*/ exe(OP_MAUH,      &r1,   r3,   EXP_H3210, r1,  EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@5,1*/ exe(OP_MAUH,      &r5,   r7,   EXP_H3210, r5,  EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@6,0*/ exe(OP_MAUH,      &r1,   r5,   EXP_H3210, r1,  EXP_H3210, 0LL, EXP_H3210, OP_SUMHL,0LL,         OP_NOP,  0LL);

    /*@7,0*/ exe(OP_MCAS,      &r31,  r1,   EXP_H3210, 64,  EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@7,0*/ mop(OP_STBR, 3,   &r31,  r++,    0LL,  MSK_D0,    (Ull)r,       80/2,     0,  0, (Ull)NULL,      80/2);
  }
//EMAX5A end
#endif
#ifdef EDGE_64BIT
  int loop=WD/2;
//EMAX5A begin edge mapdist=0
  while (loop--) {
  }
//EMAX5A end
#endif
#endif
}
```

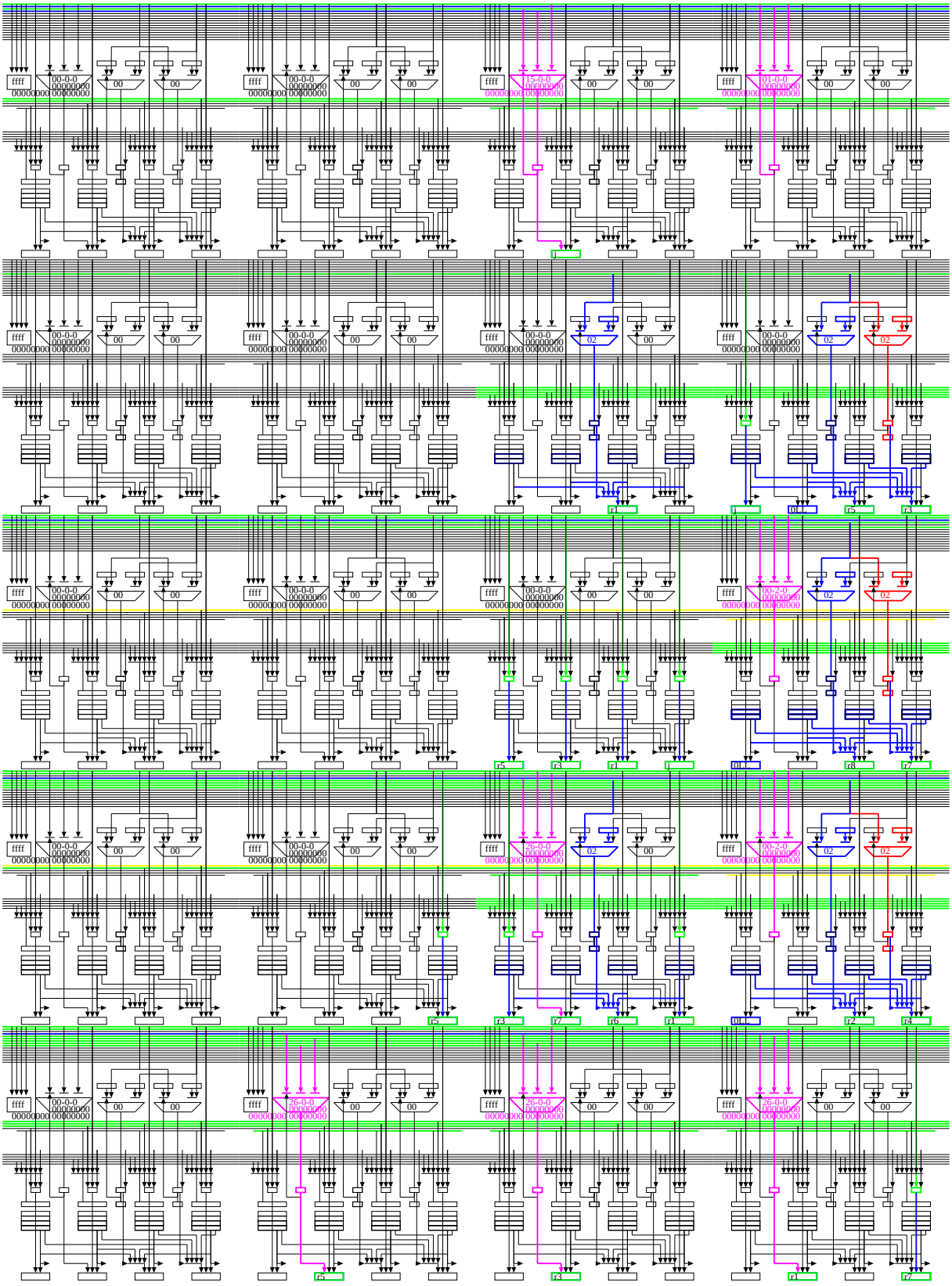


Figure.3.10: Edge

3.3.9 Stereo with stencil

```
void wdifline(u1, u2, d, w)
     unsigned int *u1, *u2, *d;
     int w;
{
#ifndef EMAX
  int j;

  for (j=0; j<w; j++) { /* one scan-line */
    *d += wdif(WIN*2,u1,u2);
    u1++;
    u2++;
    d++;
  }
```

```
#else
  Ull  AR[16][4];    /* output registers in each unit */
  Ull  BR[16][4][4]; /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15, r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
#define WDIFLINE_32BIT
#ifdef WDIFLINE_32BIT
  Ull u1m1 = u1-1;
  Ull u2m1 = u2-1;
  int *d0 = d;
  int *d1 = d;
  int  loop=WD;
//EMAX5A begin wdifline mapdist=0
  while (loop--) {
    /*@0,1*/ exe(OP_ADD,       &u1m1, u1m1, EXP_H3210, 4LL, EXP_H3210,   0,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@0,2*/ exe(OP_ADD,       &u2m1, u2m1, EXP_H3210, 4LL, EXP_H3210,   0,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@1,0*/ mop(OP_LDWR, 1,   &r2,   u1m1,   0,  MSK_D0,    (Ull)u1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,0*/ mop(OP_LDWR, 1,   &r3,   u1m1,   4,  MSK_D0,    (Ull)u1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,1*/ mop(OP_LDWR, 1,   &r4,   u1m1,   8,  MSK_D0,    (Ull)u1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,1*/ mop(OP_LDWR, 1,   &r5,   u1m1,   12, MSK_D0,    (Ull)u1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,2*/ mop(OP_LDWR, 1,   &r6,   u1m1,   16, MSK_D0,    (Ull)u1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,2*/ mop(OP_LDWR, 1,   &r7,   u1m1,   20, MSK_D0,    (Ull)u1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,3*/ mop(OP_LDWR, 1,   &r8,   u1m1,   24, MSK_D0,    (Ull)u1,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@1,3*/ mop(OP_LDWR, 1,   &r9,   u1m1,   28, MSK_D0,    (Ull)u1,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@2,0*/ mop(OP_LDWR, 1,   &r12,  u2m1,   0,  MSK_D0,    (Ull)u2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,0*/ mop(OP_LDWR, 1,   &r13,  u2m1,   4,  MSK_D0,    (Ull)u2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,1*/ mop(OP_LDWR, 1,   &r14,  u2m1,   8,  MSK_D0,    (Ull)u2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,1*/ mop(OP_LDWR, 1,   &r15,  u2m1,   12, MSK_D0,    (Ull)u2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,2*/ mop(OP_LDWR, 1,   &r16,  u2m1,   16, MSK_D0,    (Ull)u2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,2*/ mop(OP_LDWR, 1,   &r17,  u2m1,   20, MSK_D0,    (Ull)u2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,3*/ mop(OP_LDWR, 1,   &r18,  u2m1,   24, MSK_D0,    (Ull)u2,       320/2,    0, 0, (Ull)NULL,       320/2);
    /*@2,3*/ mop(OP_LDWR, 1,   &r19,  u2m1,   28, MSK_D0,    (Ull)u2,       320/2,    0, 0, (Ull)NULL,       320/2);

    /*@3,0*/ exe(OP_MSAD,      &r2,   r12,  EXP_H3210, r2,   EXP_H3210,  0,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,1*/ exe(OP_MSAD,      &r3,   r13,  EXP_H3210, r3,   EXP_H3210,  0,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,2*/ exe(OP_MSAD,      &r4,   r14,  EXP_H3210, r4,   EXP_H3210,  0,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@3,3*/ exe(OP_MSAD,      &r5,   r15,  EXP_H3210, r5,   EXP_H3210,  0,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@4,0*/ exe(OP_MSSAD,     &r6,   r2,   EXP_H3210, r16,  EXP_H3210, r6,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,1*/ exe(OP_MSSAD,     &r7,   r3,   EXP_H3210, r17,  EXP_H3210, r7,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,2*/ exe(OP_MSSAD,     &r8,   r4,   EXP_H3210, r18,  EXP_H3210, r8,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@4,3*/ exe(OP_MSSAD,     &r9,   r5,   EXP_H3210, r19,  EXP_H3210, r9,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@5,0*/ exe(OP_MAUH3,     &r31,  r6,   EXP_H3210, r7,   EXP_H3210,  r8,  EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);

    /*@6,0*/ exe(OP_MAUH3,     &r1,   r31,  EXP_H3210, r9,   EXP_H3210,  0,   EXP_H3210, OP_SUMHL,0LL,         OP_NOP,  0LL);
    /*@6,0*/ mop(OP_LDWR, 1,   &r0,   d0++,   0,  MSK_D0,    (Ull)d0,       320/2,    0, 1, (Ull)NULL,       320/2);

    /*@7,0*/ exe(OP_ADD,       &r31,  r0,   EXP_H3210, r1,   EXP_H3210,  0,   EXP_H3210, OP_NOP,  0LL,          OP_NOP,  0LL);
    /*@7,0*/ mop(OP_STWR, 3,   &r31,  d1++,   0,  MSK_D0,    (Ull)d1,       320/2,    0, 0, (Ull)NULL,       320/2);
  }
//EMAX5A end
#endif
#ifdef WDIFLINE_64BIT
  int loop=WD/2;
//EMAX5A begin wdifline mapdist=0
  while (loop--) {
  }
//EMAX5A end
#endif
#endif
}
```

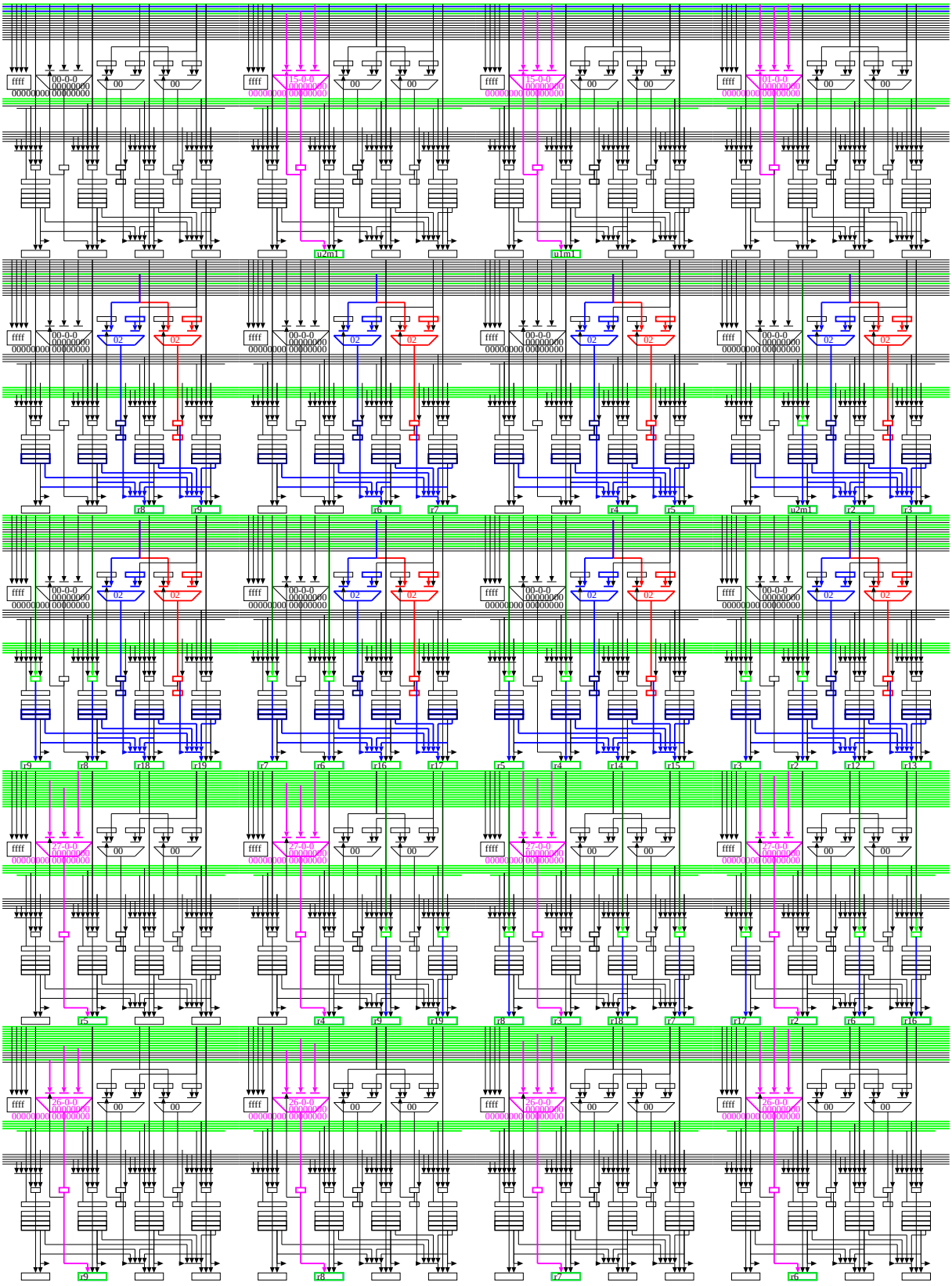


Figure.3.11: Stereo with stencil

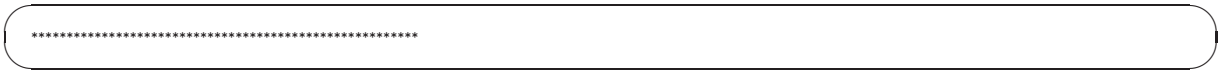
3.4 Examples (3D-floating-point)

3.4.1 Grapes with stencil

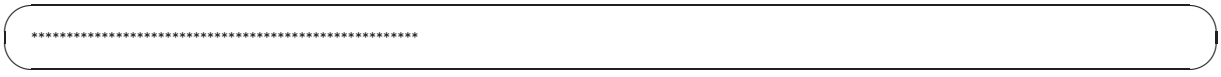
3.4.2 Jacobi with stencil

3.4.3 Fd6 with stencil

3.4.4 Resid with stencil



3.4.5 Wave2d with stencil



3.5 Examples (4D-imaging)

3.5.1 Gather with stencil

```
gather_x1(int yin, int yout)
{
#ifndef EMAX
  /*********************************************/
  /* non EMAX5                                 */
  /*********************************************/
  int x, dx, dy, w, pix;
  int cvalR, cvalG, cvalB;

  for (x=36; x<FWD-36; x++) {
#ifdef PRECISE_SCALE
    int image_center = yin+((x/coresize*WINSIZE+(offset*(coresize-(x%coresize)))/coresize+shift_x)*1021/1024);
#else
    int image_center = (x>>4)*WINSIZE + ((((~x&15)*offset)>>4) + shift_x + yin;
#endif
    /* 256  512 256 */
    pix = ACCI[image_center+smallwin_offset_y*(-1)+smallwin_offset_x*(-1)]; w = 16; cvalR =((pix>>24)&255)*w; cvalG =((pix>>16)&255)*w; cvalB =((pix>> 8)&255)*w;
    pix = ACCI[image_center+smallwin_offset_y*(-1)+smallwin_offset_x*( 0)]; w = 32; cvalR+=((pix>>24)&255)*w; cvalG+=((pix>>16)&255)*w; cvalB+=((pix>> 8)&255)*w;
    pix = ACCI[image_center+smallwin_offset_y*(-1)+smallwin_offset_x*( 1)]; w = 16; cvalR+=((pix>>24)&255)*w; cvalG+=((pix>>16)&255)*w; cvalB+=((pix>> 8)&255)*w;
    /* 512 1024 512 */
    pix = ACCI[image_center+smallwin_offset_y*( 0)+smallwin_offset_x*(-1)]; w = 32; cvalR+=((pix>>24)&255)*w; cvalG+=((pix>>16)&255)*w; cvalB+=((pix>> 8)&255)*w;
    pix = ACCI[image_center+smallwin_offset_y*( 0)+smallwin_offset_x*( 0)]; w = 64; cvalR+=((pix>>24)&255)*w; cvalG+=((pix>>16)&255)*w; cvalB+=((pix>> 8)&255)*w;
    pix = ACCI[image_center+smallwin_offset_y*( 0)+smallwin_offset_x*( 1)]; w = 32; cvalR+=((pix>>24)&255)*w; cvalG+=((pix>>16)&255)*w; cvalB+=((pix>> 8)&255)*w;
    /* 256  512 256 */
    pix = ACCI[image_center+smallwin_offset_y*( 1)+smallwin_offset_x*(-1)]; w = 16; cvalR+=((pix>>24)&255)*w; cvalG+=((pix>>16)&255)*w; cvalB+=((pix>> 8)&255)*w;
    pix = ACCI[image_center+smallwin_offset_y*( 1)+smallwin_offset_x*( 0)]; w = 32; cvalR+=((pix>>24)&255)*w; cvalG+=((pix>>16)&255)*w; cvalB+=((pix>> 8)&255)*w;
    pix = ACCI[image_center+smallwin_offset_y*( 1)+smallwin_offset_x*( 1)]; w = 16; cvalR+=((pix>>24)&255)*w; cvalG+=((pix>>16)&255)*w; cvalB+=((pix>> 8)&255)*w;
    ACCO[(yout+x)] = ((cvalR>>8)<<24) | ((cvalG>>8)<<16) | ((cvalB>>8)<<8);
  }
```

```
#else
  /*********************************************/
  /* EMAX5                                     */
  /*********************************************/
#undef EMAX_BASE32
#define EMAX_BASE64
#ifdef EMAX_BASE32
#endif
#ifdef EMAX_BASE64
  Ull  loop = 1528/2;
  Ull  x = 34;

  Uint *ym_xm   = ACCI         -smallwin_offset_y-smallwin_offset_x;
  Uint *ym_xz   = ACCI         -smallwin_offset_y                  ;
  Uint *ym_xp   = ACCI         -smallwin_offset_y+smallwin_offset_x;
  Uint *yz_xm   = ACCI                           -smallwin_offset_x;
  Uint *yz_xz   = ACCI                                             ;
  Uint *yz_xp   = ACCI                           +smallwin_offset_x;
  Uint *yp_xm   = ACCI         +smallwin_offset_y-smallwin_offset_x;
  Uint *yp_xz   = ACCI         +smallwin_offset_y                  ;
  Uint *yp_xp   = ACCI         +smallwin_offset_y+smallwin_offset_x;
  Uint *acci_ym = ACCI+yin     -smallwin_offset_y;
  Uint *acci_yz = ACCI+yin;
  Uint *acci_yp = ACCI+yin     +smallwin_offset_y;
  Ull  *acco_base = (Ull*)(ACCO+yout);
  Ull  *acco      = (Ull*)(ACCO+yout+x);
  Ull  AR[16][4];                     /* output of EX     in each unit */
  Ull  BR[16][4][4];                  /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
  Ull  r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
  Ull  c0, c1, c2, c3, ex0, ex1;
```

```
//EMAX5A begin x1 mapdist=0
  while (loop--) { /* mapped to WHILE() on BR[15][0][0] stage#0 */
    exe(OP_ADD,   &x,             x,  EXP_H3210,         2LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,  OP_NOP, 0LL); /* stage#0 */
    exe(OP_SUB,   &r1,         -1LL,  EXP_H3210,           x, EXP_H3210, 0LL, EXP_H3210, OP_AND,  15LL, OP_NOP, 0LL); /* stage#1 */
    exe(OP_NOP,   &r2,            x,  EXP_H3210,         0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,   0LL,  OP_SRL, 4LL); /* stage#1 */
    exe(OP_MLUH,  &r3,           r1,  EXP_H3210, (Ull)offset, EXP_H3210, 0LL, EXP_H3210, OP_OR,   0LL,  OP_SRL, 4LL); /* stage#2 */
    exe(OP_MLUH,  &r4,           r2,  EXP_H3210,        75LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,  OP_NOP, 0LL); /* stage#2 */
    exe(OP_ADD,   &r5, (Ull)shift_x,  EXP_H3210,    (Ull)yin, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL,  OP_NOP, 0LL); /* stage#2 */
    exe(OP_ADD3,  &r0,           r3,  EXP_H3210,          r4, EXP_H3210,  r5, EXP_H3210, OP_OR,   0LL,  OP_SLL, 2LL); /* stage#3 */
    mop(OP_LDR,   1, &BR[4][0][1],  r0, (Ull)ym_xm, MSK_D0, (Ull)acci_ym, 7240/2, 0, 0, (Ull)NULL, 7240/2);        /* stage#4 */
    mop(OP_LDR,   1, &BR[4][1][1],  r0, (Ull)ym_xz, MSK_D0, (Ull)acci_ym, 7240/2, 0, 0, (Ull)NULL, 7240/2);        /* stage#4 */
    mop(OP_LDR,   1, &BR[4][2][1],  r0, (Ull)ym_xp, MSK_D0, (Ull)acci_ym, 7240/2, 0, 0, (Ull)NULL, 7240/2);        /* stage#4 */
    exe(OP_MLUH,  &r10,     BR[4][0][1],  EXP_B5410,        16LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#5 */
    exe(OP_MLUH,  &r11,     BR[4][1][1],  EXP_B5410,        32LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#5 */
    exe(OP_MLUH,  &r12,     BR[4][2][1],  EXP_B5410,        16LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#5 */
    exe(OP_MLUH,  &r13,     BR[4][0][1],  EXP_B7632,        16LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#6 */
    exe(OP_MLUH,  &r14,     BR[4][1][1],  EXP_B7632,        32LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#6 */
    exe(OP_MLUH,  &r15,     BR[4][2][1],  EXP_B7632,        16LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#6 */
    exe(OP_MAUH3, &r20,  r10, EXP_H3210,  r11, EXP_H3210,  r12, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);                   /* stage#6 */
        :
    exe(OP_MAUH3, &r23,  r13, EXP_H3210,  r14, EXP_H3210,  r15, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);                   /* stage#9 */
    exe(OP_MLUH,  &r10,     BR[8][0][1],  EXP_B5410,        16LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#9 */
    exe(OP_MLUH,  &r11,     BR[8][1][1],  EXP_B5410,        32LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#9 */
    exe(OP_MLUH,  &r12,     BR[8][2][1],  EXP_B5410,        16LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#9 */
    exe(OP_MLUH,  &r13,     BR[8][0][1],  EXP_B7632,        16LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#10 */
    exe(OP_MLUH,  &r14,     BR[8][1][1],  EXP_B7632,        32LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#10 */
    exe(OP_MLUH,  &r15,     BR[8][2][1],  EXP_B7632,        16LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,  0LL, OP_NOP, 0LL); /* stage#10 */
    exe(OP_MAUH3, &r24,  r10, EXP_H3210,  r11, EXP_H3210,  r12, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);                   /* stage#10 */
    exe(OP_MAUH3, &r25,  r13, EXP_H3210,  r14, EXP_H3210,  r15, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);                   /* stage#11 */
    exe(OP_MAUH3, &r30,  r20, EXP_H3210,  r22, EXP_H3210,  r24, EXP_H3210, OP_AND, -1LL, OP_SRLM, 8LL); /* stage#12 */
    exe(OP_MAUH3, &r31,  r21, EXP_H3210,  r23, EXP_H3210,  r25, EXP_H3210, OP_AND, -1LL, OP_SRLM, 8LL); /* stage#12 */
    exe(OP_MH2BW, &r1,  r31, EXP_H3210,  r30, EXP_H3210,  0LL, EXP_H3210, OP_WSWAP, 0LL, OP_NOP, 0LL);  /* stage#13 */
    mop(OP_STR,     3,  &r1, (Ull)(acco++), 0LL, MSK_D0, (Ull)acco_base, 1528, 0, 0, (Ull)NULL, 1528);       /* stage#13 */
  }
//EMAX5A end
//emax5_start((Ull*)emax5_conf_x1, (Ull*)emax5_lmmi_x1, (Ull*)emax5_regv_x1);
#endif
#endif
}
```

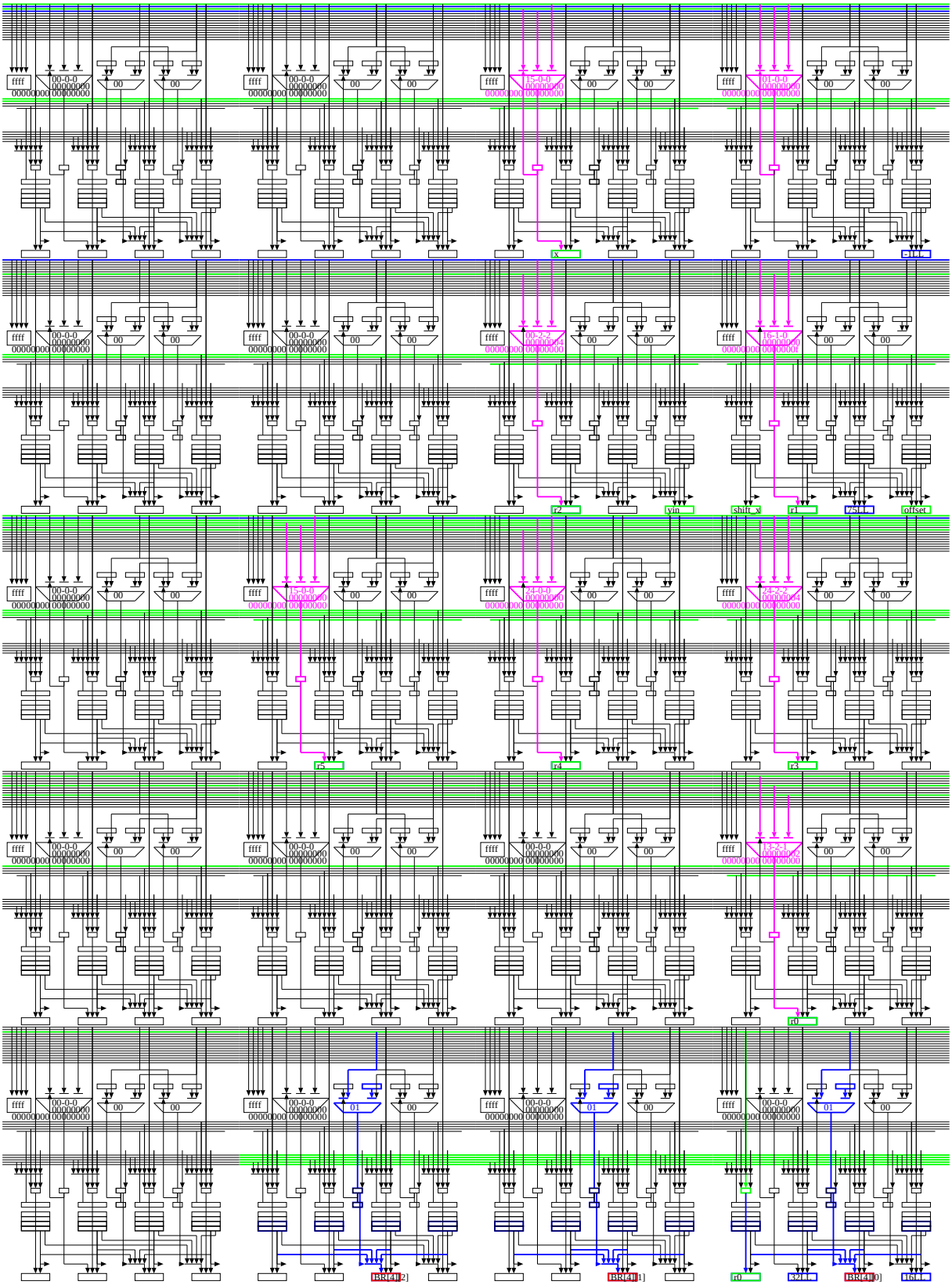


Figure.3.12: Gather

3.5.2 Gdepth with stencil

```
gdepth_x1(int yin, int yout)
{
#ifndef EMAX
  /*********************************************/
  /* non EMAX5                                 */
  /*********************************************/
  int i, j;
  int x, dx, dy;
  int cvalR, cvalG, cvalB;
  Uint pix00, pix01, pix02, pix03, pix04, pix05, pix10, pix11, pix20, pix21;

  for (x=36; x<FWD-36; x++) {
#ifdef PRECISE_SCALE
    int image_center = yin+((x/coresize*WINSIZE+(offset*(coresize-(x%coresize)))/coresize+shift_x)*1021/1024);
#else
    int image_center = (x>>4)*WINSIZE + (((~x&15)*offset)>>4) + shift_x + yin;
#endif
    Uint sad = 0;
    cvalR=0;
    cvalG=0;
    cvalB=0;
    pix10 = ACCI[image_center                 -smallwin_offset_x-image_WD-1];                      /*  |             |  */
    pix11 = ACCI[image_center                 -smallwin_offset_x-image_WD+1];                      /*  ├             ┤  */
    pix00 = ACCI[image_center                                   -image_WD-1];sad  = sad(pix00, pix10); /*  |++    **    ++|  */
    pix01 = ACCI[image_center                                   -image_WD+1];sad += sad(pix01, pix11); /*  |    center     |  */
    pix20 = ACCI[image_center                 +smallwin_offset_x-image_WD-1];sad += sad(pix00, pix20); /*  |    r2  r1     |  */
    pix21 = ACCI[image_center                 +smallwin_offset_x-image_WD+1];sad += sad(pix01, pix21); /*  ├             ┤  */
                                                                                                   /*  ├─────────────┤  */
    pix10 = ACCI[image_center-smallwin_offset_y-smallwin_offset_x-image_WD-1];sad += sad(pix00, pix10); /*  |++         ++|  */
    pix11 = ACCI[image_center-smallwin_offset_y-smallwin_offset_x-image_WD+1];sad += sad(pix01, pix11); /*  |             |  */
    pix20 = ACCI[image_center-smallwin_offset_y+smallwin_offset_x-image_WD-1];sad += sad(pix00, pix20); /*  |             |  */
    pix21 = ACCI[image_center-smallwin_offset_y+smallwin_offset_x-image_WD+1];sad += sad(pix01, pix21); /*  ├             ┤  */
                                                                                                   /*  ├─────────────┤  */
    pix10 = ACCI[image_center+smallwin_offset_y-smallwin_offset_x-image_WD-1];sad += sad(pix00, pix10); /*  ├             ┤  */
    pix11 = ACCI[image_center+smallwin_offset_y-smallwin_offset_x-image_WD+1];sad += sad(pix01, pix11); /*  |++         ++|  */
    pix20 = ACCI[image_center+smallwin_offset_y+smallwin_offset_x-image_WD-1];sad += sad(pix00, pix20); /*  |             |  */
    pix21 = ACCI[image_center+smallwin_offset_y+smallwin_offset_x-image_WD+1];sad += sad(pix01, pix21); /*  |             |  */
                                                                                                   /*  ├─────────────┤  */
    pix10 = ACCI[image_center                 -smallwin_offset_x        -1];                       /*  |             |  */
    pix11 = ACCI[image_center                 -smallwin_offset_x        +1];                       /*  ├             ┤  */
    pix02 = ACCI[image_center                                           -1];sad += sad(pix02, pix10); /*  |             |  */
    pix03 = ACCI[image_center                                           +1];sad += sad(pix03, pix11); /*  |++    **    ++|  */
    pix20 = ACCI[image_center                 +smallwin_offset_x        -1];sad += sad(pix02, pix20); /*  |    center     |  */
    pix21 = ACCI[image_center                 +smallwin_offset_x        +1];sad += sad(pix03, pix21); /*  ├    r4  r3   ┤  */
                                                                                                   /*  ├─────────────┤  */
    pix10 = ACCI[image_center-smallwin_offset_y-smallwin_offset_x        -1];sad += sad(pix02, pix10); /*  |             |  */
    pix11 = ACCI[image_center-smallwin_offset_y-smallwin_offset_x        +1];sad += sad(pix03, pix11); /*  |++         ++|  */
    pix20 = ACCI[image_center-smallwin_offset_y+smallwin_offset_x        -1];sad += sad(pix02, pix20); /*  |             |  */
    pix21 = ACCI[image_center-smallwin_offset_y+smallwin_offset_x        +1];sad += sad(pix03, pix21); /*  ├             ┤  */
                                                                                                   /*  ├─────────────┤  */
    pix10 = ACCI[image_center+smallwin_offset_y-smallwin_offset_x        -1];sad += sad(pix02, pix10); /*  ├             ┤  */
    pix11 = ACCI[image_center+smallwin_offset_y-smallwin_offset_x        +1];sad += sad(pix03, pix11); /*  |             |  */
    pix20 = ACCI[image_center+smallwin_offset_y+smallwin_offset_x        -1];sad += sad(pix02, pix20); /*  |++         ++|  */
    pix21 = ACCI[image_center+smallwin_offset_y+smallwin_offset_x        +1];sad += sad(pix03, pix21); /*  |             |  */
                                                                                                   /*  ├─────────────┤  */
    pix10 = ACCI[image_center                 -smallwin_offset_x+image_WD-1];                      /*  |             |  */
    pix11 = ACCI[image_center                 -smallwin_offset_x+image_WD+1];                      /*  ├             ┤  */
    pix04 = ACCI[image_center                                   +image_WD-1];sad += sad(pix04, pix10); /*  |             |  */
    pix05 = ACCI[image_center                                   +image_WD+1];sad += sad(pix05, pix11); /*  |    center     |  */
    pix20 = ACCI[image_center                 +smallwin_offset_x+image_WD-1];sad += sad(pix04, pix20); /*  |++    **   ++|  */
    pix21 = ACCI[image_center                 +smallwin_offset_x+image_WD+1];sad += sad(pix05, pix21); /*  ├    r6  r5   ┤  */
                                                                                                   /*  ├─────────────┤  */
    pix10 = ACCI[image_center-smallwin_offset_y-smallwin_offset_x+image_WD-1];sad += sad(pix04, pix10); /*  |             |  */
    pix11 = ACCI[image_center-smallwin_offset_y-smallwin_offset_x+image_WD+1];sad += sad(pix05, pix11); /*  |             |  */
    pix20 = ACCI[image_center-smallwin_offset_y+smallwin_offset_x+image_WD-1];sad += sad(pix04, pix20); /*  |++         ++|  */
    pix21 = ACCI[image_center-smallwin_offset_y+smallwin_offset_x+image_WD+1];sad += sad(pix05, pix21); /*  ├             ┤  */
                                                                                                   /*  ├─────────────┤  */
    pix10 = ACCI[image_center+smallwin_offset_y-smallwin_offset_x+image_WD-1];sad += sad(pix04, pix10); /*  ├             ┤  */
    pix11 = ACCI[image_center+smallwin_offset_y-smallwin_offset_x+image_WD+1];sad += sad(pix05, pix11); /*  |             |  */
    pix20 = ACCI[image_center+smallwin_offset_y+smallwin_offset_x+image_WD-1];sad += sad(pix04, pix20); /*  |             |  */
    pix21 = ACCI[image_center+smallwin_offset_y+smallwin_offset_x+image_WD+1];sad += sad(pix05, pix21); /*  |++         ++|  */
    if (SAD[yout+x] > (255*6*3*6)/200 && sad < SAD[yout+x]) {
      SAD[yout+x]  = sad;
      ACCO[yout+x] = offset;
    }
  }
```

```
#else
  /***********************************************/
  /* EMAX5                                       */
  /***********************************************/
#undef EMAX_BASE32
#define EMAX_BASE64
#ifdef EMAX_BASE64
  Ull  loop = 1528/2;
  Ull  x = 34;
  Uint *yzm_xm_m4 = ACCI-image_WD                -smallwin_offset_x-1;  Uint *yzm_xm_p4 = ACCI-image_WD                -smallwin_offset_x+1;
  Uint *yzm_xz_m4 = ACCI-image_WD                                  -1;  Uint *yzm_xz_p4 = ACCI-image_WD                                  +1;
  Uint *yzm_xp_m4 = ACCI-image_WD                +smallwin_offset_x-1;  Uint *yzm_xp_p4 = ACCI-image_WD                +smallwin_offset_x+1;
  Uint *ymm_xm_m4 = ACCI-image_WD-smallwin_offset_y-smallwin_offset_x-1;  Uint *ymm_xm_p4 = ACCI-image_WD-smallwin_offset_y-smallwin_offset_x+1;
  Uint *ymm_xp_m4 = ACCI-image_WD-smallwin_offset_y+smallwin_offset_x-1;  Uint *ymm_xp_p4 = ACCI-image_WD-smallwin_offset_y+smallwin_offset_x+1;
  Uint *ypm_xm_m4 = ACCI-image_WD+smallwin_offset_y-smallwin_offset_x-1;  Uint *ypm_xm_p4 = ACCI-image_WD+smallwin_offset_y-smallwin_offset_x+1;
  Uint *ypm_xp_m4 = ACCI-image_WD+smallwin_offset_y+smallwin_offset_x-1;  Uint *ypm_xp_p4 = ACCI-image_WD+smallwin_offset_y+smallwin_offset_x+1;
  Uint *yzz_xm_m4 = ACCI                         -smallwin_offset_x-1;  Uint *yzz_xm_p4 = ACCI                         -smallwin_offset_x+1;
  Uint *yzz_xz_m4 = ACCI                                           -1;  Uint *yzz_xz_p4 = ACCI                                           +1;
  Uint *yzz_xp_m4 = ACCI                         +smallwin_offset_x-1;  Uint *yzz_xp_p4 = ACCI                         +smallwin_offset_x+1;
  Uint *ymz_xm_m4 = ACCI         -smallwin_offset_y-smallwin_offset_x-1;  Uint *ymz_xm_p4 = ACCI         -smallwin_offset_y-smallwin_offset_x+1;
  Uint *ymz_xp_m4 = ACCI         -smallwin_offset_y+smallwin_offset_x-1;  Uint *ymz_xp_p4 = ACCI         -smallwin_offset_y+smallwin_offset_x+1;
  Uint *ypz_xm_m4 = ACCI         +smallwin_offset_y-smallwin_offset_x-1;  Uint *ypz_xm_p4 = ACCI         +smallwin_offset_y-smallwin_offset_x+1;
  Uint *ypz_xp_m4 = ACCI         +smallwin_offset_y+smallwin_offset_x-1;  Uint *ypz_xp_p4 = ACCI         +smallwin_offset_y+smallwin_offset_x+1;
  Uint *yzp_xm_m4 = ACCI+image_WD                -smallwin_offset_x-1;  Uint *yzp_xm_p4 = ACCI+image_WD                -smallwin_offset_x+1;
  Uint *yzp_xz_m4 = ACCI+image_WD                                  -1;  Uint *yzp_xz_p4 = ACCI+image_WD                                  +1;
  Uint *yzp_xp_m4 = ACCI+image_WD                +smallwin_offset_x-1;  Uint *yzp_xp_p4 = ACCI+image_WD                +smallwin_offset_x+1;
  Uint *ymp_xm_m4 = ACCI+image_WD-smallwin_offset_y-smallwin_offset_x-1;  Uint *ymp_xm_p4 = ACCI+image_WD-smallwin_offset_y-smallwin_offset_x+1;
  Uint *ymp_xp_m4 = ACCI+image_WD-smallwin_offset_y+smallwin_offset_x-1;  Uint *ymp_xp_p4 = ACCI+image_WD-smallwin_offset_y+smallwin_offset_x+1;
  Uint *ypp_xm_m4 = ACCI+image_WD+smallwin_offset_y-smallwin_offset_x-1;  Uint *ypp_xm_p4 = ACCI+image_WD+smallwin_offset_y-smallwin_offset_x+1;
  Uint *ypp_xp_m4 = ACCI+image_WD+smallwin_offset_y+smallwin_offset_x-1;  Uint *ypp_xp_p4 = ACCI+image_WD+smallwin_offset_y+smallwin_offset_x+1;
  Uint *acci_yzm  = ACCI+yin-image_WD;        Uint *acci_ymm  = ACCI+yin-image_WD-smallwin_offset_y;  Uint *acci_ypm  = ACCI+yin-image_WD+smallwin_offset_y;
  Uint *acci_yzz  = ACCI+yin;                 Uint *acci_ymz  = ACCI+yin         -smallwin_offset_y;  Uint *acci_ypz  = ACCI+yin         +smallwin_offset_y;
  Uint *acci_yzp  = ACCI+yin+image_WD;        Uint *acci_ymp  = ACCI+yin+image_WD-smallwin_offset_y;  Uint *acci_ypp  = ACCI+yin+image_WD+smallwin_offset_y;
  Ull  *sadi_base = (Ull*)(SAD+yout+x);       Ull  *sadi      = (Ull*)(SAD+yout+x);                  Ull  *sado_base = (Ull*)(SAD+yout+x);
  Ull  *sado      = (Ull*)(SAD+yout+x);       Ull  *acco_base = (Ull*)(ACCO+yout+x);                 Ull  *acco      = (Ull*)(ACCO+yout+x);
  Ull  AR[16][4];                    /* output of EX     in each unit */
  Ull  BR[16][4][4];                 /* output registers in each unit */
  Ull  r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, r10, r11, r12, r13, r14, r15;
  Ull  r16, r17, r18, r19, r20, r21, r22, r23, r24, r25, r26, r27, r28, r29, r30, r31;
  Ull  c0, c1, c2, c3, ex0, ex1;
```

```
//EMAX5A begin x1 mapdist=0
  while (loop--) {                                          /* mapped to WHILE() on BR[15][0][0] stage#0 */
    exe(OP_ADD,  &x,             x,  EXP_H3210,         2LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,   0LL, OP_NOP, 0LL); /* stage#0 */
    exe(OP_SUB,  &r1,         -1LL,  EXP_H3210,           x, EXP_H3210, 0LL, EXP_H3210, OP_AND,  15LL, OP_NOP, 0LL); /* stage#1 */
    exe(OP_NOP,  &r2,            x,  EXP_H3210,         0LL, EXP_H3210, 0LL, EXP_H3210, OP_OR,    0LL, OP_SRL, 4LL); /* stage#1 */
    exe(OP_MLUH, &r3,           r1,  EXP_H3210, (Ull)offset, EXP_H3210, 0LL, EXP_H3210, OP_OR,    0LL, OP_SRL, 4LL); /* stage#2 */
    exe(OP_MLUH, &r4,           r2,  EXP_H3210,        75LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP,   0LL, OP_NOP, 0LL); /* stage#2 */
    exe(OP_ADD,  &r5, (Ull)shift_x,  EXP_H3210,    (Ull)yin, EXP_H3210, 0LL, EXP_H3210, OP_NOP,   0LL, OP_NOP, 0LL); /* stage#2 */
    exe(OP_ADD3, &r0,           r3,  EXP_H3210,          r4, EXP_H3210,  r5, EXP_H3210, OP_OR,    0LL, OP_SLL, 2LL); /* stage#3 */
    mop(OP_LDR,  1, &BR[4][0][1], r0, (Ull)yzm_xm_m4, MSK_D0, (Ull)acci_yzm, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#4 */
    mop(OP_LDR,  1, &BR[4][0][0], r0, (Ull)yzm_xp_m4, MSK_D0, (Ull)acci_yzm, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#4 */
    mop(OP_LDR,  1, &BR[4][1][1], r0, (Ull)yzm_xz_m4, MSK_D0, (Ull)acci_yzm, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#4 */
    mop(OP_LDR,  1, &BR[4][1][0], r0, (Ull)yzm_xz_p4, MSK_D0, (Ull)acci_yzm, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#4 */
    mop(OP_LDR,  1, &BR[4][2][1], r0, (Ull)yzm_xp_m4, MSK_D0, (Ull)acci_yzm, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#4 */
    mop(OP_LDR,  1, &BR[4][2][0], r0, (Ull)yzm_xp_p4, MSK_D0, (Ull)acci_yzm, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#4 */
    exe(OP_MSSAD,&r14,   0LL, EXP_H3210, BR[4][0][0], EXP_H3210, BR[4][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);   /* stage#5 */
    exe(OP_MSSAD,&r15,   0LL, EXP_H3210, BR[4][0][1], EXP_H3210, BR[4][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);   /* stage#5 */
    exe(OP_MSSAD,&r16,   0LL, EXP_H3210, BR[4][2][0], EXP_H3210, BR[4][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);   /* stage#5 */
    exe(OP_MSSAD,&r17,   0LL, EXP_H3210, BR[4][2][1], EXP_H3210, BR[4][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);   /* stage#5 */
        :
    mop(OP_LDR,  1, &BR[9][0][1], r0, (Ull)ypz_xm_m4, MSK_D0, (Ull)acci_ypz, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#9 */
    mop(OP_LDR,  1, &BR[9][0][0], r0, (Ull)ypz_xm_p4, MSK_D0, (Ull)acci_ypz, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#9 */
    mop(OP_LDR,  1, &BR[9][2][1], r0, (Ull)ypz_xp_m4, MSK_D0, (Ull)acci_ypz, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#9 */
    mop(OP_LDR,  1, &BR[9][2][0], r0, (Ull)ypz_xp_p4, MSK_D0, (Ull)acci_ypz, 7240/2, 0, 0, (Ull)NULL, 7240/2);          /* stage#9 */
    exe(OP_MSSAD,&r24,   r14, EXP_H3210, BR[9][0][0], EXP_H3210, BR[7][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);   /* stage#10 */
    exe(OP_MSSAD,&r25,   r15, EXP_H3210, BR[9][0][1], EXP_H3210, BR[7][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);   /* stage#10 */
    exe(OP_MSSAD,&r26,   r16, EXP_H3210, BR[9][2][0], EXP_H3210, BR[7][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);   /* stage#10 */
    exe(OP_MSSAD,&r27,   r17, EXP_H3210, BR[9][2][1], EXP_H3210, BR[7][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);   /* stage#10 */
    mop(OP_LDR,  1, &BR[10][0][1], r0, (Ull)yzp_xm_m4, MSK_D0, (Ull)acci_yzp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#10 */
    mop(OP_LDR,  1, &BR[10][0][0], r0, (Ull)yzp_xm_p4, MSK_D0, (Ull)acci_yzp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#10 */
    mop(OP_LDR,  1, &BR[10][1][1], r0, (Ull)yzp_xz_m4, MSK_D0, (Ull)acci_yzp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#10 */
    mop(OP_LDR,  1, &BR[10][1][0], r0, (Ull)yzp_xz_p4, MSK_D0, (Ull)acci_yzp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#10 */
    mop(OP_LDR,  1, &BR[10][2][1], r0, (Ull)yzp_xp_m4, MSK_D0, (Ull)acci_yzp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#10 */
    mop(OP_LDR,  1, &BR[10][2][0], r0, (Ull)yzp_xp_p4, MSK_D0, (Ull)acci_yzp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#10 */
    exe(OP_MSSAD,&r14,   r24, EXP_H3210, BR[10][0][0], EXP_H3210, BR[10][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#11 */
    exe(OP_MSSAD,&r15,   r25, EXP_H3210, BR[10][0][1], EXP_H3210, BR[10][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#11 */
    exe(OP_MSSAD,&r16,   r26, EXP_H3210, BR[10][2][0], EXP_H3210, BR[10][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#11 */
    exe(OP_MSSAD,&r17,   r27, EXP_H3210, BR[10][2][1], EXP_H3210, BR[10][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#11 */
    mop(OP_LDR,  1, &BR[11][0][1], r0, (Ull)ymp_xm_m4, MSK_D0, (Ull)acci_ymp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#11 */
    mop(OP_LDR,  1, &BR[11][0][0], r0, (Ull)ymp_xm_p4, MSK_D0, (Ull)acci_ymp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#11 */
    mop(OP_LDR,  1, &BR[11][2][1], r0, (Ull)ymp_xp_m4, MSK_D0, (Ull)acci_ymp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#11 */
    mop(OP_LDR,  1, &BR[11][2][0], r0, (Ull)ymp_xp_p4, MSK_D0, (Ull)acci_ymp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#11 */
    exe(OP_MSSAD,&r24,   r14, EXP_H3210, BR[11][0][0], EXP_H3210, BR[10][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#12 */
    exe(OP_MSSAD,&r25,   r15, EXP_H3210, BR[11][0][1], EXP_H3210, BR[10][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#12 */
    exe(OP_MSSAD,&r26,   r16, EXP_H3210, BR[11][2][0], EXP_H3210, BR[10][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#12 */
    exe(OP_MSSAD,&r27,   r17, EXP_H3210, BR[11][2][1], EXP_H3210, BR[10][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#12 */
    mop(OP_LDR,  1, &BR[12][0][1], r0, (Ull)ypp_xm_m4, MSK_D0, (Ull)acci_ypp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#12 */
    mop(OP_LDR,  1, &BR[12][0][0], r0, (Ull)ypp_xm_p4, MSK_D0, (Ull)acci_ypp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#12 */
    mop(OP_LDR,  1, &BR[12][2][1], r0, (Ull)ypp_xp_m4, MSK_D0, (Ull)acci_ypp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#12 */
    mop(OP_LDR,  1, &BR[12][2][0], r0, (Ull)ypp_xp_p4, MSK_D0, (Ull)acci_ypp, 7240/2, 0, 0, (Ull)NULL, 7240/2);         /* stage#12 */
    exe(OP_MSSAD,&r14,   r24, EXP_H3210, BR[12][0][0], EXP_H3210, BR[10][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#13 */
    exe(OP_MSSAD,&r15,   r25, EXP_H3210, BR[12][0][1], EXP_H3210, BR[10][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#13 */
    exe(OP_MSSAD,&r16,   r26, EXP_H3210, BR[12][2][0], EXP_H3210, BR[10][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#13 */
    exe(OP_MSSAD,&r17,   r27, EXP_H3210, BR[12][2][1], EXP_H3210, BR[10][1][1], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#13 */
    exe(OP_MAUH, &r24,   r14, EXP_H3210,  r15, EXP_H3210, 0LL, EXP_H3210, OP_NOP,   0LL, OP_NOP, 0LL);               /* stage#14 */
    exe(OP_MAUH, &r26,   r16, EXP_H3210,  r17, EXP_H3210, 0LL, EXP_H3210, OP_NOP,   0LL, OP_NOP, 0LL);               /* stage#14 */
    exe(OP_MAUH, &r30,   r24, EXP_H3210,  r26, EXP_H3210, 0LL, EXP_H3210, OP_SUMHL, 0LL, OP_NOP, 0LL);               /* stage#15 */
    mop(OP_LDR,  1, &BR[15][1][1], (Ull)(sadi++), 0LL, MSK_D0, (Ull)sadi_base, 1528/2, 0, 1, (Ull)NULL, 1528/2);            /* stage#15 */
    exe(OP_CMP_LT, &c0, r30,          EXP_H3210,     BR[15][1][1], EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#16 */
    exe(OP_CMP_GT, &c1, BR[15][1][1], EXP_H3210, (137LL<<32)|137LL, EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#16 */
    exe(OP_CCAT, &r31,  (Ull)offset,  EXP_H3210, (Ull)offset,       EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* stage#16 */
    cex(OP_CEXE, &ex0,   0, 0, c1, c0, 0x8888);                                                                         /* stage#17 */
    mop(OP_STR,  ex0, &r30, (Ull)(sado++), 0LL, MSK_D0, (Ull)sado_base, 1528/2, 0, 1, (Ull)NULL, 1528/2); /* stage#17 */
    cex(OP_CEXE, &ex1,   0, 0, c1, c0, 0x8888);                                                                         /* stage#17 */
    mop(OP_STR,  ex1, &r31, (Ull)(acco++), 0LL, MSK_D0, (Ull)acco_base, 1528/2, 0, 1, (Ull)NULL, 1528/2); /* stage#17 */
  }
//EMAX5A end
//emax5_start((Ull*)emax5_conf_x1, (Ull*)emax5_lmmi_x1, (Ull*)emax5_regv_x1);
#endif
#endif
}
```

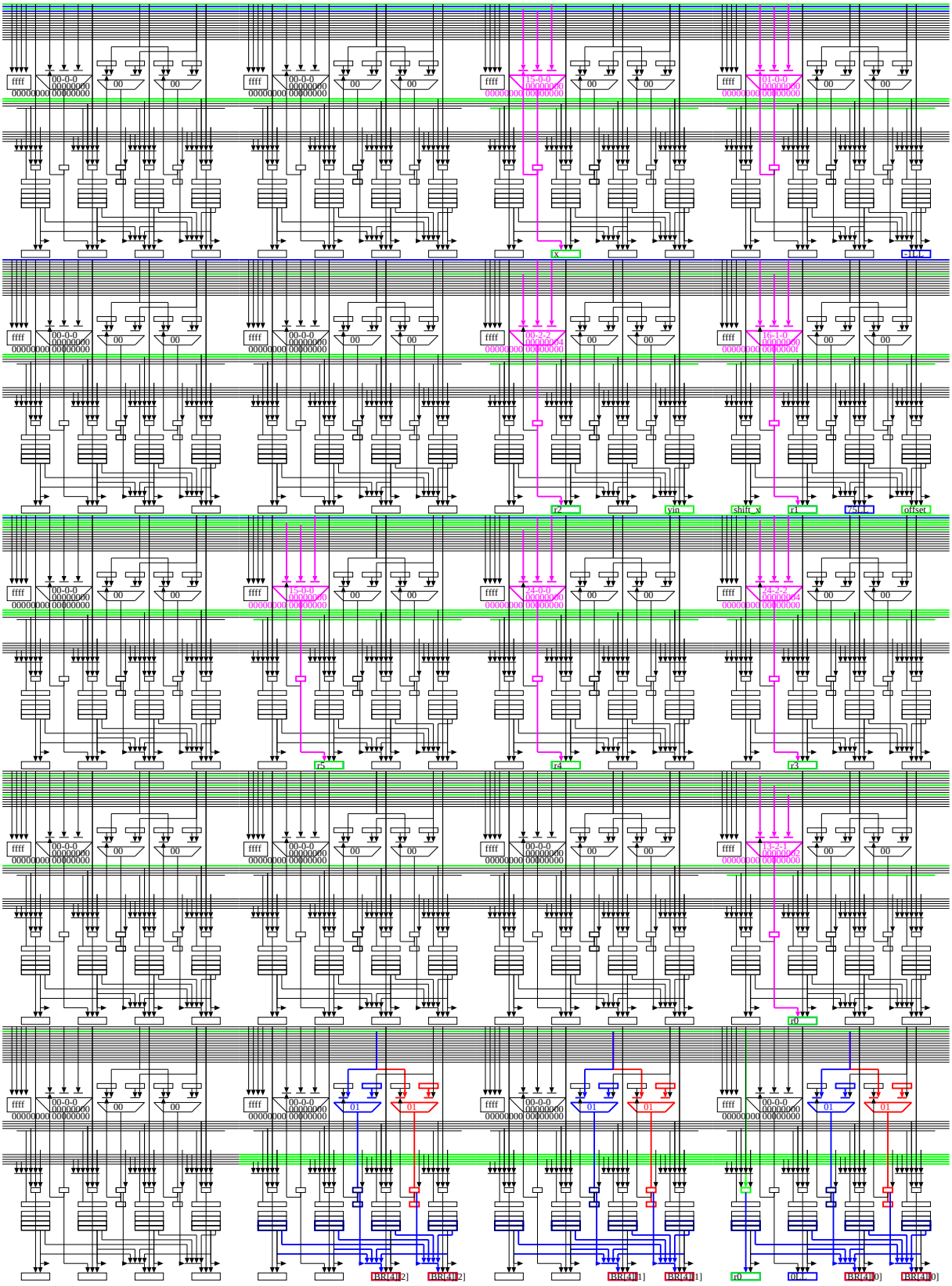


Figure.3.13: Gdepth

3.6 Examples (graph processing)

3.6.1 Triangle counting kernel0 with TCU

```
void tri_kernel0(struct param_bfs *param)
{
  volatile int i, j, pid, qid, MVL, MEL;
  volatile struct vertex *p, *np, *q;
  volatile struct neighborvertex *n;

  i = param->i;
  p = param->p;
  np = param->nextp;
  MVL = param->maxvlist;
  MEL = param->maxelist;
  pid = p->id;

#ifndef EMAX
    for (j=0; j<p->nedges; j++) {                    /* R 0段:最内ループ 256 回転程度 */
      n = p->npage[j/MAXNV_PPAGE]+(j%MAXNV_PPAGE);
      q = n->vp;                                     /* R 0段:neighborvertex 全体を配置 pointer を使い参照 */
      qid = n->id;                                   /* R 0段:同上 */
      if (!q->parent) {                              /* R 1段:vertex 全体を配置 pointer->pointer を使い参照 */
        /************************/
        while (cmpxchg(&Sem0, -1, param->th) != -1);
        /************************/
        if (!q->parent) {                            /* R 1段:同上 */
          if (nnextfrontiers >= MVL) {
            printf("vlist[%d] exhausted\n", MVL);
            exit(1);
          }
          q->parent = pid;                           /* W 2段:verex 更新 */
          q->depth  = depth;                         /* W 2段:同上 */
          q->findex = nnextfrontiers;                /* W 2段:同上 */
          nextfrontier[nnextfrontiers] = q;          /* W 2段:next_frontier[] 更新 */
          nnextfrontiers++;                          /* W 2段:同上 */
          nnextfrontiers__neighbors+=q->nedges;
        }
        /************************/
        /*cmpxchg(&Sem0, param->th, -1);*/
        release(&Sem0, -1);
        /************************/
      }
      else if (q->depth==depth-1 && q->findex<i) {   /* R 1段:vertex 全体を配置 pointer->pointer を使い参照 */
        /************************/
        while (cmpxchg(&Sem1, -1, param->th) != -1);
        /************************/
        if (nfrontier_edges >= MEL) {
          printf("elist[%d] exhausted\n", MEL);
          exit(1);
        }
        frontier_edge[nfrontier_edges].src = (pid<qid)?p:q; /* W 2段:frontier_edge[] 更新 */
        frontier_edge[nfrontier_edges].dst = (pid<qid)?q:p; /* W 2段:同上 */
        nfrontier_edges++;                                  /* W 2段:同上 */
        nfrontier_edges__neighbors+=((pid<qid)?p:q)->nedges;
        /************************/
        /*cmpxchg(&Sem1, param->th, -1);*/
        release(&Sem1, -1);
        /************************/
      }
    }
```

```
#else
  Ull  AR[64][4];    /* output registers in each unit */
  Ull  BR[64][4][4]; /* output registers in each unit */
  struct neighborvertex  *r0     =NULL; /* n */
  struct neighborvertex **r0_top =p->npage;
  Uint                   r0_len =p->nedges*4;
  Uint                   pnedges=p->nedges;
  struct neighborvertex **r0_ntop=np?np->npage:NULL;
  Uint                   r0_nlen=np?np->nedges*4:NULL;
  Ull                    r2[4], r3[4], r4[4], r6, r7, r8;
  Ull                    depth_1=depth-1;
  Ull                    c0, c1, c2, c3, ex0, ex1;
  int loop=p->nedges;
  void tri_kernel0_trans0();
  void tri_kernel0_trans1();
//EMAX5A begin tri_kernel0 mapdist=1
  while (loop--) {
/*0,0*/ mo4(OP_LDRQ,    1,      BR[0][0],    (Ull)(r0++), 0LL,         MSK_D0,    (Ull)r0_top, r0_len, 2, 0, (Ull)r0_ntop, r0_nlen);
/*1,1*/ mo4(OP_LDDMQ,   1,      BR[1][1],    BR[0][0][3], 32LL,        MSK_D0,    (Ull)NULL, 0, 0, 0, (Ull)NULL, 0);
/*1,2*/ exe(OP_CMP_LT,  &c0,    pid,         EXP_H3210,   BR[0][0][2], EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
/*2,0*/ exe(OP_CMOV,    &r6,    c0,          EXP_H3210,   p,           EXP_H3210, BR[0][0][3], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
/*2,1*/ exe(OP_CMOV,    &r7,    c0,          EXP_H3210,   BR[0][0][3], EXP_H3210, p,           EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
/*2,2*/ exe(OP_CMOV,    &r8,    c0,          EXP_H3210,   pnedges,     EXP_H3210, BR[1][1][0], EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);
/*2,3*/ exe(OP_CMP_EQ,  &c0,    BR[1][1][1], EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* q->parent==0? */
/*3,0*/ exe(OP_ADD,     &r3[0], BR[0][0][3], EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* address(q) */
/*3,1*/ exe(OP_ADD,     &r3[1], BR[1][1][0], EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* q->nedges */
/*3,2*/ exe(OP_ADD,     &r3[2], pid,         EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* pid */
/*3,2*/ cex(OP_CEXE,    &ex0,   0,  0,  0, c0, 0xaaaa);
/*3,2*/ mo4(OP_TR,      ex0,    r3,          (Ull)NULL,   0LL,         0LL,       (Ull)tri_kernel0_trans0, 0, 0, 0, (Ull)NULL, 0);
/*4,0*/ exe(OP_CMP_NE,  &c0,    BR[1][1][1], EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* q->parent!=0 */
/*4,1*/ exe(OP_CMP_EQ,  &c1,    BR[1][1][2], EXP_H3210,   depth_1,     EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* q->depth==depth-1 */
/*4,2*/ exe(OP_CMP_LT,  &c2,    BR[1][1][3], EXP_H3210,   i,           EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* q->findex<i */
/*5,0*/ exe(OP_ADD,     &AR[5][0], r6,       EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* src */
/*5,1*/ exe(OP_ADD,     &AR[5][1], r7,       EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* dst */
/*5,2*/ exe(OP_ADD,     &AR[5][2], r8,       EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* nen */
/*5,3*/ exe(OP_ADD,     &AR[5][3], 0LL,      EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL); /* dummy */
/*5,3*/ cex(OP_CEXE,    &ex1,   0, c2, c1, c0, 0x8080);
/*5,3*/ mo4(OP_TR,      ex1,    AR[5],       (Ull)NULL,   0LL,         0LL,       (Ull)tri_kernel0_trans1, 0, 0, 0, (Ull)NULL, 0);
  }
//EMAX5A end
#endif
}
```

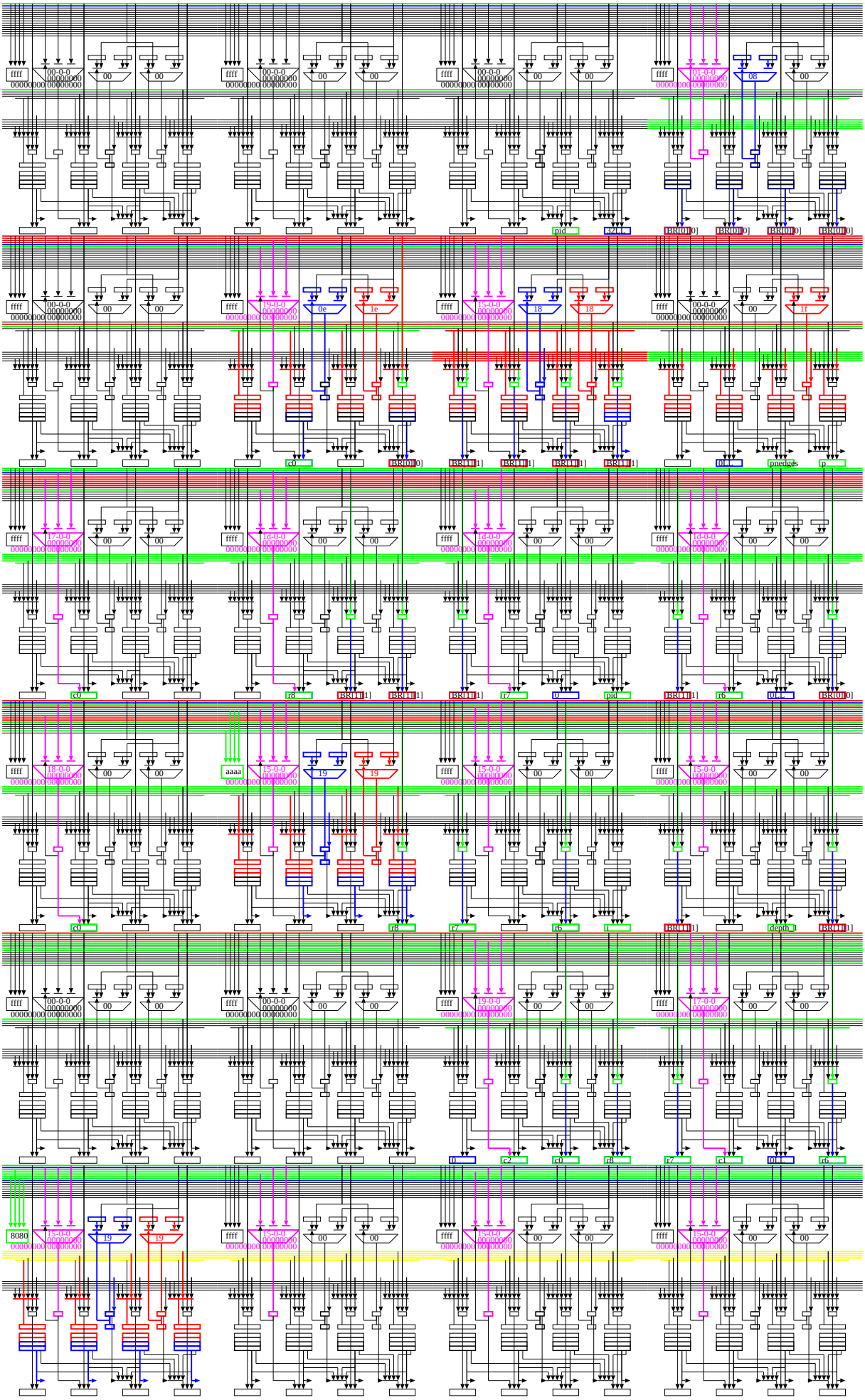


Figure.3.14: Triangle counting kernel0 with TCU

3.6.2 Triangle counting kernel1 with TCU

```
void tri_kernel1(struct param_tricount *param)
{
  /* search triangle in {frontier,next} */
  /* case 1: e ∈ frontier, v ∈ prev     */
  /* case 2: e ∈ frontier, v ∈ frontier */
  /* case 3: e ∈ frontier, v ∈ next     */
  int i, j, pid, qid, sdepth, tdepth;
  struct vertex *p, *np, *q, *t;
  struct neighborvertex *n;

  p = param->p;
  np = param->nextp;
  t = param->t;
  pid = p->id;
  sdepth = p->depth;

#ifndef EMAX
    int tricount = 0;
    for (j=0; j<p->nedges; j++) {                   /* R ０段:最内ループ 256 回転程度 */
      n = p->npage[j/MAXNV_PPAGE]+(j%MAXNV_PPAGE);
      q = n->vp;                                    /* R ０段:neighborvertex 全体を配置 pointer を使い参照 */
      qid = n->id;                                  /* R ０段:同上 */
      tdepth = q->depth;                            /* R １段:vertex 全体を配置 pointer->pointer を使い参照 */
      if ((tdepth==sdepth-1)||(tdepth==sdepth+1)||(tdepth==sdepth && qid<pid)) { /* R ２段:比較 */
        if (search_nvertex(t->nhashtbl, qid))       /* R ３段:HASH-SEARCH/CAM-SEARCH */
          tricount++;                               /* W ４段:カウンタ更新 */
      }
    }
    param->tricount += tricount;
```

```
#else
  Ull  BR[16][4][4]; /* output registers in each unit */
  struct neighborvertex  *r0     =NULL; /* n */
  struct neighborvertex **r0_top =p->npage;
  Uint                   r0_len =p->nedges*4;
  Uint                   pnedges=p->nedges;
  struct neighborvertex **r0_ntop=np?np->npage:NULL;
  Uint                   r0_nlen=np?np->nedges*4:NULL;
  Ull                    r2[4], r3[4], r6, r7, r8;
  Ull                    sdepth_m1=sdepth-1;
  Ull                    tnhashtbl=t->nhashtbl;
  Ull                    sdepth_p1=sdepth+1;
  Ull                    c0, c1, c2, c3, ex0, ex1;
  int loop=p->nedges;
  void tri_kernel1_trans0();
//EMAX5A begin tri_kernel1 mapdist=1
  while (loop--) {
/*0,0*/ mo4(OP_LDRQ,    1,      BR[0][0],    (Ull)(r0++), 0LL,         MSK_D0,    (Ull)r0_top, r0_len, 2, 0, (Ull)r0_ntop, r0_nlen);
/*1,1*/ mo4(OP_LDDMQ,   1,      BR[1][1],    BR[0][0][3], 32LL,        MSK_D0,    (Ull)NULL, 0, 0, 0, (Ull)NULL, 0);
/*2,0*/ exe(OP_CMP_EQ,  &c0,    BR[1][1][2], EXP_H3210,   sdepth_m1,   EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);       /* q->depth==p->depth-1 */
/*2,1*/ exe(OP_CMP_EQ,  &c1,    BR[1][1][2], EXP_H3210,   sdepth_p1,   EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);       /* q->depth==p->depth+1 */
/*2,2*/ exe(OP_CMP_EQ,  &c2,    BR[1][1][2], EXP_H3210,   sdepth,      EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);       /* q->depth==p->depth   */
/*2,3*/ exe(OP_CMP_LT,  &c3,    BR[0][0][2], EXP_H3210,   pid,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);       /* qid<pid              */
/*3,0*/ exe(OP_ADD,     &r3[0], tnhashtbl,   EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);       /* t->nhashtbl */
/*3,1*/ exe(OP_ADD,     &r3[1], BR[0][0][2], EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);       /* qid */
/*3,2*/ exe(OP_ADD,     &r3[2], 0LL,         EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);       /* dummy */
/*3,3*/ exe(OP_ADD,     &r3[3], 0LL,         EXP_H3210,   0LL,         EXP_H3210, 0LL, EXP_H3210, OP_NOP, 0LL, OP_NOP, 0LL);       /* dummy */
/*3,3*/ cex(OP_CEXE,    &ex0,   c3, c2, c1, c0, 0xfeee);
/*3,3*/ mo4(OP_TR,      ex0,    r3,          (Ull)NULL,   0LL,         0LL,       (Ull)tri_kernel1_trans0, 0, 0, 0, (Ull)NULL, 0);  /* r3[1]<-(nhashtbl) r3[0]<-(qid) */
  }
//EMAX5A end
#endif
}
```

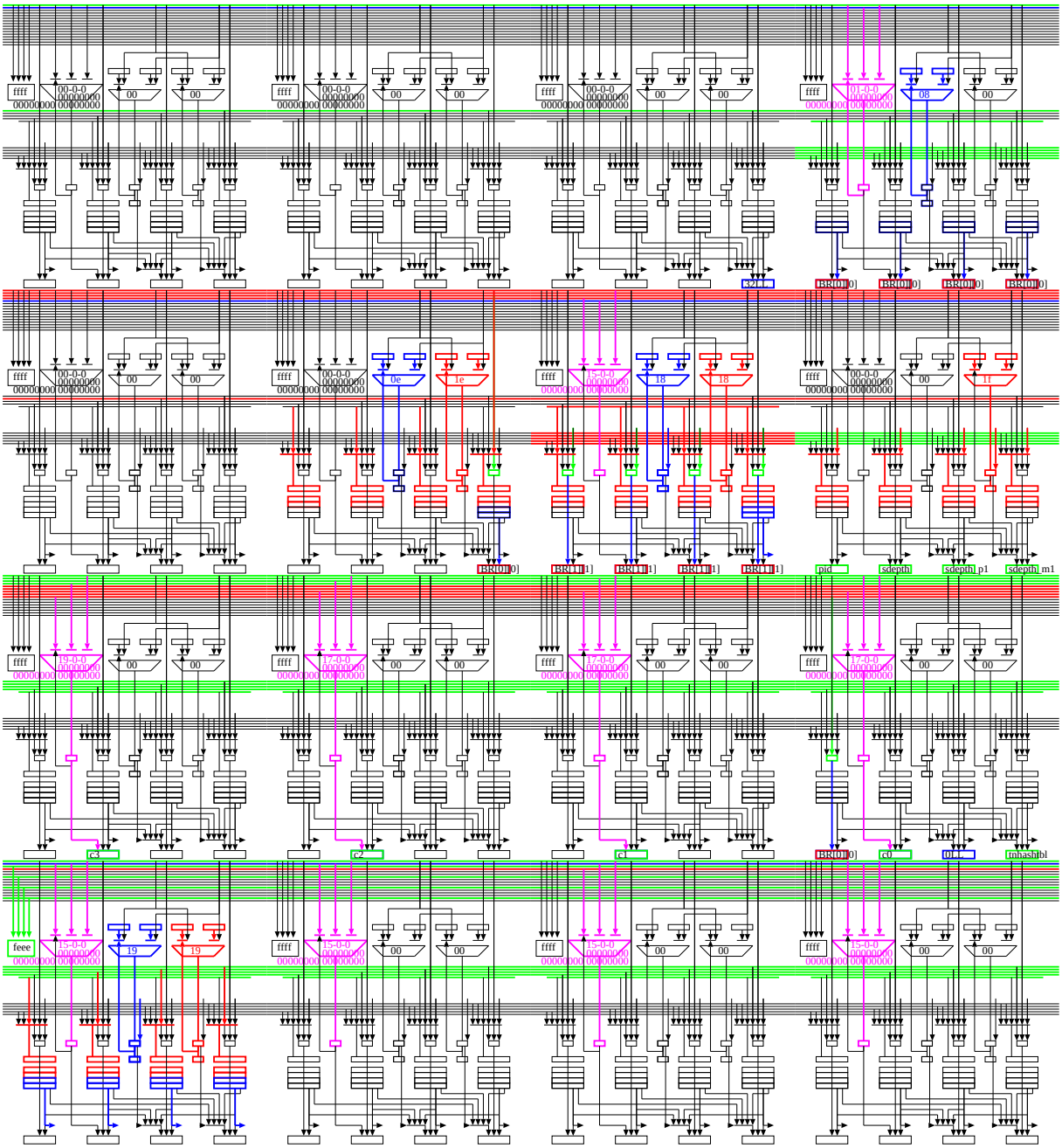


Figure.3.15: Triangle counting kernel1 with TCU

3.6.3 Dijkstra kernel with TCU

****

3.7 Compiling application programs

See “all:” tag in each Makefile-zynq.emax5.

3.8 Executing application programs on simulator

See “run:” tag in each Makefile-zynq.emax5.

Appendix A

References

本章では，関連仕様書・規格，参考文献，関連ソースプログラム，および，ツールチェインを列挙する．

A.1 EMAX5/L2

- EMAX5/L2 基本特許 proj-arm64/doc/pat35.tgz

A.2 EMAX5/ZYNQ

- ARMv8 アーキテクチャ仕様書 ..... proj-arm64/doc/arm/DDI0487A_f_armv8_arm.pdf
- ARM Cortex-A53 MPCore Processor Technical Reference Manual
  ..... proj-arm64/doc/arm/ARM-CORTEX-A53_R0P4.pdf
- ZYNQ Ultrascale+ SoC Technical Reference Manual
  ..... proj-zcu102/doc/ug1085-zynq-ultrascale-trm.pdf
- ZYNQ-7000 Technical Reference Manual proj-zc706/doc/xilinx/ug585-Zynq-7000-TRM.pdf
- AMBA AXI4 and ACE Protocol Specification ..... proj-arm64/doc/arm/AXI4_specification.pdf
- EMAX5/ZYNQ 仕様書 proj-arm64/doc/emax5/emax5.pdf
- EMAX5 C 言語ディレクティブ変換 proj-arm64/src/conv-c2b/conv-c2b
- EMAX5 シミュレータ proj-arm64/src/bsim/bsim
- プログラム例（畳み込み演算） proj-arm64/sample/conv16/conv16.c
- プログラム例（画像フィルタ） proj-arm64/sample/filter/filter.c
- プログラム例（浮動小数点ステンシル） proj-arm64/sample/stencil-pipe/stencil.c
- プログラム例（4D 画像処理. 再生） proj-arm64/sample/4dimage/gather.c
- プログラム例（4D 画像処理. 距離画像） proj-arm64/sample/4dimage/gdepth.c
- プログラム例（グラフ処理） proj-arm64/sample/tricount8/tricount.c