解　説

4. 各種のハードウェアアルゴリズム

4.3 画像処理用 VLSI のハードウェアアルゴリズム†

松本恵治†† 抜山知二††

1. はじめに

LSI 製造技術の進展にともなって，LSI の集積度は飛躍的に増大しつつある．また LSI の応用分野の拡大により，ひとつの LSI に要求される機能も複雑かつ大規模になってきている．

しかしながら，製造技術が進歩し数多くのトランジスタをひとつの LSI に実現することが出来ても，LSI を実現するまでのさまざまな設計段階における機能検証方法や，開発期間や開発リソースの削減方法を十分に考慮しなければこれからの VLSI の実現は困難となりつつある．

本稿では，画像処理プロセッサのひとつである ImPP (Image Pipelined Processor) について，その機能概要とハードウェアアルゴリズムを紹介し，ImPP のハードウェアアルゴリズムが，VLSI の設計，検証において効率的であることを述べる．

2. ImPP の設計時の考慮事項

VLSI の機能や論理の設計段階では，以下に示す事項を考慮し行う必要がある．ImPP の開発に際してもこれらの事項を効果的に反映することを念頭におき開発を進めた．

（1） 単位面積当りの素子数改善

同一のチップサイズの LSI であっても，単位面積当りに集積する素子数が多ければ多くの機能を実現できることになる．したがって，単位面積当りに集積できる素子数を増やす努力は，VLSI の開発において考えなければならない事項のひとつである．

この集積密度の向上は，プロセス技術や，リゾグラフィ技術の進歩によるものが大きなウエイトを占めているのであるが，マスクの設計ルールが同じであれば，LSI を製造するマスク上に規則正しい配列の出来る機能とした LSI のほうが高い集積密度をもつ．ROM や RAM などのメモリは，ひとつのメモリセルを綿密にレイアウトし，それを規則正しく配列しているので集積密度は高い．マイクロプロセッサやその周辺機器制御用コントロール LSI は，メモリのみではその機能を実現できないが，表-1 に示すように，ROM や RAM を内蔵した LSI の集積密度は高い．

したがって，メモリに限らず，メモリ構成に近い構造，すなわちひとつの機能ブロックを設計し，これを規則正しく配列できる構成を採用できる機能仕様を考えることが VLSI の設計には大切である．

（2） 設計容易なアーキテクチャ

集積密度が向上し，ひとつの LSI に取り入れることのできる機能が大きくなると，回路設計や LSI を製造するためのマスク・レイアウト設計に要する時間の増大が問題となる．つまり，これらの設計に長時間を費やすと，LSI の完成時にすでに機能が陳腐化していることも生じよう．したがって回路設計が容易でレイアウト設計の期間短縮を図ることのできるような仕様の検討を十分に行う必要がある．このためには，

表-1 各種 LSI の集積密度の比較

LSI の種類	集積密度 (トランジスタ/mm²)
ROM/RAM 内蔵	
ROM 2KB, RAM 128 B	840
ROM 1KB, RAM 64 B	850
ROM 6KB, RAM 64 B	960
ROM/RAM なし	
割込コントローラ	290
グラフィックコントローラ	310
シリアルコントローラ	470

† Hardware Algorithm of An Image Processing VLSI by Keiji MATSUMOTO and Tomoji NUKIYAMA (Microcomputer Products Division, NEC Corporation).

†† 日本電気(株)マイクロコンピュータ技術本部

CAD の有効利用が図り易いことや，LSI を機能ブロックに分割して並行して設計ができること，さらに分割したブロック間の情報の受け渡しが単純であることなどが必要である．

（3） 機能検証が容易なアーキテクチャ

ひとつの LSI で実現できる機能が増大すると，その機能が所期に考えていた仕様通りに動作しているか否かの検証方法の容易さも，設計の容易さの問題と同様に事前に十分考慮すべき事項である．

LSI の検証には，LSI として製造される前の検証と製造後の検証とがある．特に，事前に行う検証が不十分であると，リワークのために多くの時間を費やさねばならない結果を招く．これらの検証を十分行うために，検証が容易である機能仕様の検討に加えて設計の階層化を図り，CAD ツールを有効に利用して LSI の製造前後の検証を迅速かつ容易にできるようにする必要がある．

3. ImPP の機能概略[1),2)]

ImPP のハードウェアアルゴリズムについて述べる前に，ImPP の機能について簡単に触れておきたい．ImPP は，多量のデータに対する繰り返し演算処理を高速に行わせることを目的として開発したプロセッサである．この目的達成のために，パイプライン方式と非ノイマン型アーキテクチャのひとつであるデータ駆動方式[3),4)] のアーキテクチャを採用している．従来のコンピュータのほぼすべてに用いられているノイマン型アーキテクチャのプロセッサでは，真に処理の対象となるデータ以外に命令コードやアドレス生成のためのデータもひとつのバスを介して転送する必要がある．各種の情報がひとつのバスに集中することにより，いわゆるノイマン・ボトル・ネックと呼ばれるバスの隘路が生じる．またプログラムの実行は逐次処理であり，プログラムの実行時に演算対象のデータが用意されていない場合は，データが用意されるまで他の処理を実行することはできない．一方データ駆動型のプロセッサでは，データ自身が処理内容を携えているのでノイマン型で問題となるバスの隘路は発生しない．またプログラムの実行は，データの流れにもとづいており演算の対象となるデータが揃った時点で処理を実行し，データが揃わない場合には，他の処理を実行するのでデータが揃うまで他の処理が待たされることはない．

ImPP は，図-1 に示すように，大きく 10 個のブロックで構成されている．

IC (Input Controller)：外部より ImPP へ入力するデータの制御を行う．

LT (Link Table)：ユーザにより書き換え可能な 128 語×16 ビットの DRAM (Dynamic RAM) で構成されるプログラム領域である．主としてデータの行き先情報を格納しておく．

FT (Function Table)：ユーザにより書き換え可能な 64 語×40 ビットの DRAM で構成されるプログラム領域である．主としてデータに対する機能的な処理

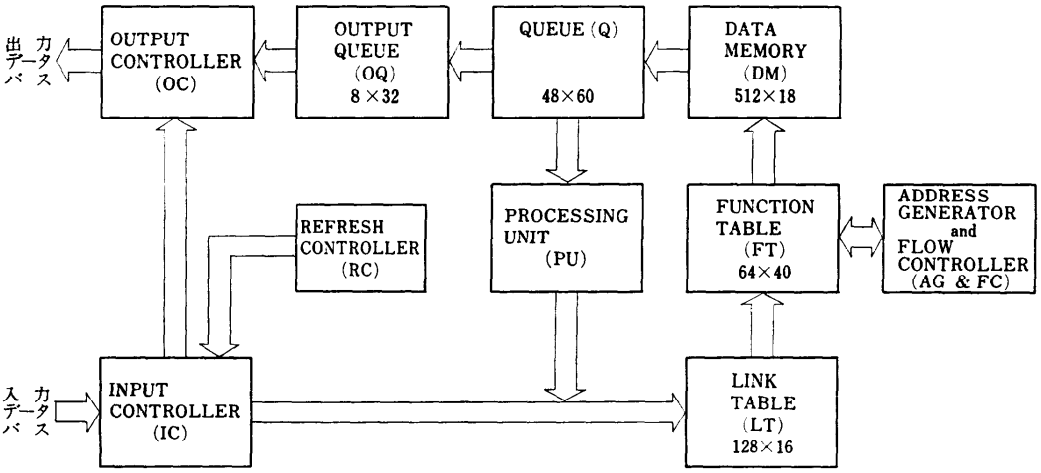

図-1 ImPP の内部ブロック図

情報を格納しておく.

AG & FC (Address Generator and Flow Controllor): DM アドレスの生成と，データの流れの制御を行うブロックである.

DM(Data Memory): 512語×18ビットの DRAM で構成され，演算対象となるデータの一時的格納や演算に必要な定数を格納する領域である.

Q (Queue): 48語×60ビットの DRAM で構成され，PU や OQ が稼動中のときに DM より送られてくるデータを一時的に格納する FIFO (First-In First Out) バッファである.

PU (Processing Unit): FT で読み出した処理内容に従って，単項または二項演算を実行するユニットである．論理演算，加減乗算，バレルシフト，大小比較，ビット操作など32種類の演算を実行する．また16ビット×16ビットのハードウェア乗算器を内蔵し，この乗算をひとつのパイプライン・ステージで実行する.

OQ (Output Queue): 8語×32ビットの SRAM (Static RAM) で構成され，ImPP より外部へデータを出力する際に外部バスがビジィのとき，一時的にデータを格納する FIFO バッファである.

OC (Output Controller): ImPP より外部へ出力するデータを制御する.

RC (Refresh Controller): 内部の DRAM をリフレッシュするためのコントロール部である.

これらのブロックの中で，LT, FT (AG & FC を含む)，DM は1パイプライン・ステージで，Q, PU は，2パイプライン・ステージでデータ処理を行う．LT, FT, DM, Q, PU で内部のパイプライン・リングを形成し，IC を介して ImPP に取り込まれたデータはこの内部の7段のステージによるパイプライン・リングを周ることにより，処理が施され，処理されたデータが OQ, OC を介して ImPP より出力される.

データ駆動の特徴であるデータの待ち合せ制御は，AG & FC と DM において処理される．AG & FC に到着したデータの演算相手がまだ DM に到着していない場合は，このデータは DM へ格納される．演算相手が，すでに到着している場合はこのデータを DM より読み出して，到着したデータとペアとなって PU へ送られ演算が施される．このように，データの流れにもとづく処理を可能にしている.

図-2 に，ImPP のチップ拡大写真を示す．ImPP は，1.75ミクロン・ルールの NMOS 技術を用い，

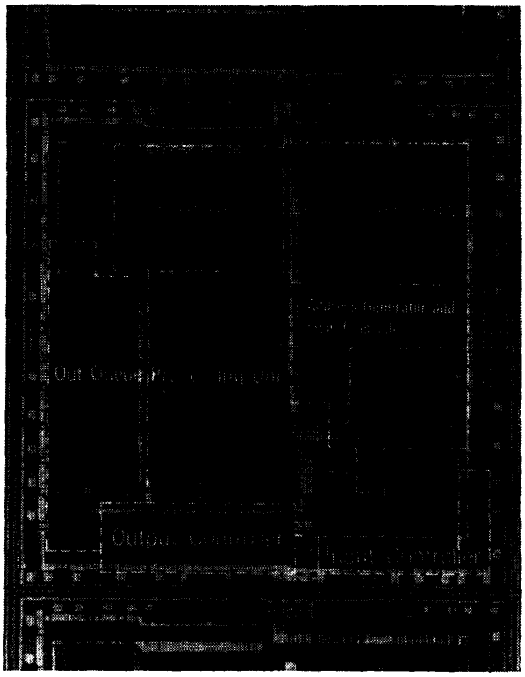

図-2 ImPP のチップ拡大写真

11万5千トランジスタを 6.93×6.99 mm^2 上に集積している.

4. ImPP のハードウェアアルゴリズム

ここでは，ImPP の具体的なハードウェアアルゴリズムの二，三の例について，先に述べた ImPP の設計時の考慮事項をもとに説明する.

4.1 規則性と集積密度の向上

ROM, RAM などのメモリは，規則性が良い．したがって，メモリを内蔵しているプロセッサは，集積密度が向上する．ImPP も，できる限り多くのメモリを利用することにより集積密度の向上を図っている.

LT, FT, DM および Q に合計 16.7K ビットの DRAM を，OQ に256ビットの SRAM を用いている．OQ に SRAM を用いたのは OQ のビット数が少ないことと，OQ 部は，後述する RAM のリフレッシュが行いづらいことにもとづいている．OQ 以外の RAM として DRAM を用いたのは，DRAM のビット・セル面積が，SRAM に比べて小さく集積密度が向上するからである．しかしながら DRAM の場合，メモリ・セルの内容を保持させるために定期的にセルをリフレッシュする必要があり，ImPP の各 DRAM について，リフレッシュ回路を設けるのは相

当の負担となる．ImPP は，パイプラインとデータ駆動方式により，この負担を軽減している．すなわち，ひとつのリフレッシュ用データが RC より定期的に出力され，このデータが順次パイプライン・リング上を移り行くことにより DRAM をリフレッシュできるのである．この場合，リフレッシュ用データ自身がリフレッシュすべき RAM のアドレスを携えている．

ImPP は，RAM 以外にひとつの機能ブロックの設計により，そのブロックを繰り返し利用し，規則性と集積密度の向上も図っている，PU 部に必要な ALU (Arithmetic Logic Unit) や乗算器，AG & FC 部に必要な加減算器，各パイプライン・ステージにおけるデータ・ラッチなどにこの種の方法を利用している．ここでは，乗算器を取り上げてそのハードウェアアルゴリズムについて説明する．ImPP に用いている乗算器は，16 ビット×16 ビットの乗算をひとつのパイプライン・ステージで実行する．一般に乗算器に広く用いられているブース (Booth) のアルゴリズム[5),6)]を変形したものを ImPP に適用し，素子数の減量と高速化を図った．

一般に，N ビット×N ビットの並列乗算器をハードウェアで構成するには N^2 個の全加算器を必要とするが，乗数を M ビットごとにまとめて部分積を求めることにより全加算器の数を，おおよそ N^2/M 個に減らすことができる．ただし，部分積から発生した被乗数の生成のために別に全加算器が必要となるので，M の値を最適に選ぶ必要がある．さらに部分積に負数表現を許すことにより，被乗数の生成のための全加算器の数を減らすことができる．16 ビット×16 ビットの乗算で，M を 1～4 とし負数表現を許さないアレー方式の場合と，負数表現を許す場合とに必要な全加算器数をまとめたものが**表-2** である．表-2 より，$M=3$ とし，負数表現を許した場合に，ハードウェア量を最少にできる．ImPP には，このハードウェアアルゴリズムによる乗算器を採用している．

表-2　乗算器の全加算器数比較

方式 / M	アレー方式			負数表現を許す方式		
	部分積	被乗数生成	合計	部分積	被乗数生成	合計
1	240	0	240	—	—	—
2	126	16	142	168	0	168
3	95	50	145	122	16	138
4	60	96	156	108	50	158

このようにして実現した ImPP の規則化率，すなわち全体の素子数の中で人手によりレイアウトしないで済ますことのできる素子数の割合は，85%で有効領域の集積密度は約 3,000 トランジスタ/mm² である．この集積密度は，デザイン・ルールの改善による密度の向上が考えられるものの表-1 に示したものに較べて数倍向上していることがわかる．

4.2 設計容易な構造

ImPP は，パイプラインとデータ駆動方式を併用している．これらの方式を用いることにより，ImPP を設計容易な構造としている．すなわち各々のブロックに到着したデータには，処理の対象となるデータ以外にそのブロックでの処理内容をも携さえているからである．したがって，各々のブロックで独立して処理が可能で他のブロックの処理の実行に干渉することがない．各々のブロックで処理されたデータは，その形を変えながらパイプラインにより次のブロックへ送られる．この動作を繰り返しながら一連の処理が遂行される．

この様子を，LT 部を例にとって説明する．**図-3～図-5** に，LT 部のブロック図，入出力データ形式およびタイミング図をそれぞれ示す．LT 部は，図-3 に示すように 128 語×16 ビットの DRAM で構成され，LT 部に到着したデータにより DRAM の読み出し，書き込みおよびリフレッシュを行い，FT 部へ送り出される．LT 部のパイプライン・ステージは，二相クロック ϕ_1 と ϕ_2 で制御され，ϕ_1 で LT 部への入力データをラッチし，ϕ_2 でタイミングを調整し，

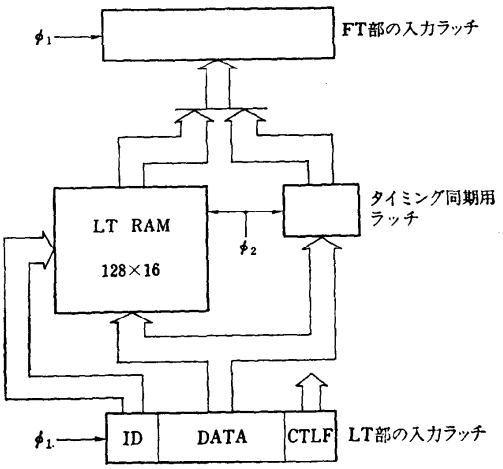

図-3　LT 部のブロック図

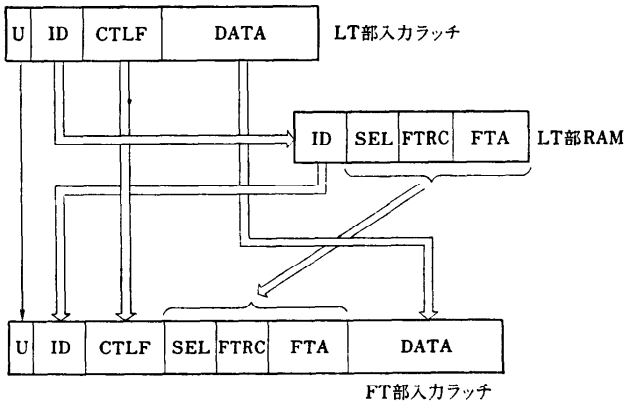

図-4 LT 部の入出力データ形式

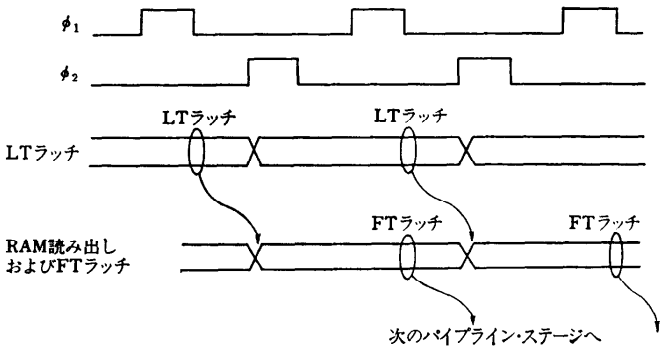

図-5 LT 部のタイミング図

LT 部からの出力データを ϕ_1 で FT 部への入力データとしてラッチする．図-4 に示すように，LT 部の入力ラッチにラッチされたデータには，'DATA' 以外に 'ID', 'CTLF' と 'U' 情報を含んでいる．'ID' 情報により DRAM をアドレスし，'CTLF' と 'U' 情報に従って書き込み，読み出しやリフレッシュの処理内容を決める．処理内容が読み出しの場合は，DRAM の内容を読み出し FT 部以降の処理に必要な情報を付加し FT 部へ送る．この DRAM の内容には FT 部の DRAM のアドレス情報 (FTA)，パイプライン・リングを一周し再度 LT 部の DRAM をアドレスするためのアドレス情報 (ID) などがある．このように，LT 部での処理は，前段のブロックより送られてきたデータと，パイプライン・クロック ϕ_1 と ϕ_2 とによってのみ制御できるので他のブロックからの制御信号にもとづく処理や，他のブロックへ影響を与える制御信号の発生はない．LT 部以外に，FT，DM，Q の各ブロックは，基本的にこの構成を採っている．

AG & FC と PU 部は，ALU や乗算器を含んでいるがそのブロックに到着したデータによって処理を遂行できる構造にかわりはない．

各ブロック間に複雑な制御情報の引き渡しが不要のために，回路設計やレイアウト設計は容易で，かつ各々のブロックに分割して設計を進めることができる．さらに素子ディレイなどを考えた処理スピードに対する考察もパイプライン・クロック ϕ_1 と ϕ_2 以外のタイミングは不要なので，容易に行うことができる．

4.3 検証容易な構造

パイプラインとデータ駆動方式を採用している ImPP は，設計が容易な構造となっていると同時に検証も容易な構造となっている．すなわち ImPP を各々のブロックに分割して検証を行うことができ，検証作業の並列化を図ることができる．また，分割して検証を行ってもブロック間に複雑なタイミングによる制御情報の受け渡しがないので検証もれが少ない．しかしながら，パイプライン方式を採用しているためにパイプライン・ステージの途中のブロックに不具合が生じた場合，その後続のブロックの機能検証が非常に困難となる．このことは試作後の LSI を検証する場合に顕著である．つまり，試作後の LSI の機能検証は，LSI への入力データに対して生じる結果の出力データのみで判定しなければならない．特にパイプライン方式の LSI の場合，パイプライン・ステージの途中のブロックで不具合が生じ出力データが期待値と合致しない時，その出力データより不具合個所を見い出すことが困難である．

このような状況を避けるために，ImPP はテスト用バスをパイプライン処理のバス以外に備えている．この様子を LT 部を例にとって図-6 に示す．テスト用バスは，テストモード時 ImPP の入力データ端子と出力データ端子を結び，各々のブロックに共通に配線されている．また，テスト用バスは各々のブロックのパイプライン入力ラッチへのデータの設定や，各々のブロックからのデータの読み出しをテスト用の制御信号により時分割に行う．したがってテストモード時の LT 部のテストには，LT 部の入力ラッチにデー

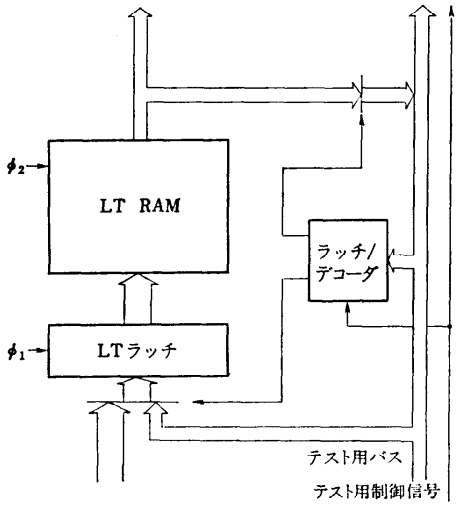

図-6 LT 部のテスト回路

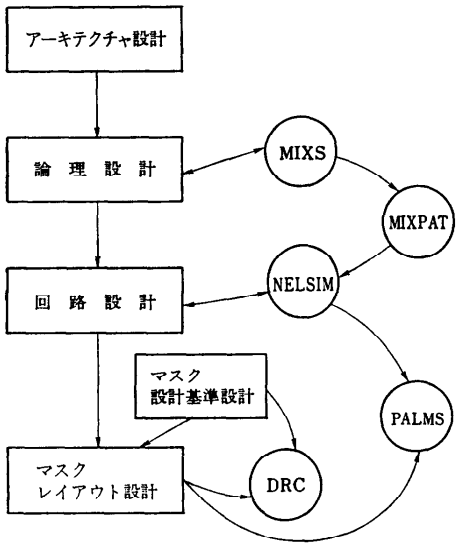

図-7 LSI の設計階層構造と ImPP で用いた CAD ツール

タを設定し，LT 部の DRAM に対するデータの書き込みや読み出し処理をさせ，その後 LT 部からの出力情報をテスト用バスを介して読み出すことにより LT 部の機能を検証する．こうすることにより，LT 部を他のブロックとは独立して検証できるのである．どのブロックも他のブロックとは独立した形で検証可能で試作後の LSI の検証を容易にし，検証もれによるリワークのためのリソースを削減している．

5. ImPP の階層設計

LSI の設計には，パイプラインやデータ駆動方式などの基本的な動作を規定するアーキテクチャ設計，各ブロックの動作タイミングや動作論理を設計する論理設計，論理設計をもとに LSI を実現するために，具体的回路を作成する回路設計，回路図をもとに LSI を製造するために必要なマスクを作るためのマスク・レイアウト設計などがある．この他に，レイアウト設計の基準を決めるための設計基準設計も，LSI の設計には大切な設計のひとつである．

LSI の誕生の頃は，LSI の規模が小さくひとりの設計者がこれらすべての段階の設計を行うことが可能であった．しかしながら ImPP のように規模が大きくなると，ひとつの段階の設計であってもひとりの設計者では，設計が不可能である．したがって，パイプラインとデータ駆動方式を採用し，各段階での設計において作業分割を行ってもスムーズに作業を進めることができるよう考慮している．さらに，ImPP は，これらの各段階の設計の階層化を図っている．設計の階層化を行う場合，各階層間の情報の受け渡しが不十分であると各階層での検証が十分に行われたとしても，試作後の LSI が所期の仕様通りに動作しない結果となる．

ImPP の設計においては，各階層間の情報が確実に受け渡されていることを検証するために CAD ツールを駆使している．図-7 に，LSI 設計の階層構造と検証に用いた CAD ツールの関係を示した．アーキテクチャ設計と論理設計階層間の情報伝達や検証のための CAD ツールは実用レベルのものはない．論理設計では，設計したものを MIXS (Mixed Level Simulation)[7] の FDL (Functional Description Language) を用いて記述し，シミュレーションを行い論理の正しさを検証する．回路設計においては，NELSIM (Network Logic Simulator)[8] を用いゲートレベルでのシミュレーションを行い，回路の正しさを検証する．論理と回路の設計階層間の検証には，論理設計において MIXS で論理検証に用いた ImPP に対する入出力情報を NELSIM 用に変換するツール MIXPAT を用い，NELSIM でシミュレーションし，ふたつのシミュレーションによる出力結果を比較照合することにより行っている．さらに，回路とレイアウトの設計階層間の検証は，PALMS (Pattern-Logic Matching Test System)[9] を用いてレイアウト設計

情報より回路レベルの接続情報を生成し，比較照合することにより行っている．また，レイアウト設計が設計基準を満足しているか否かの検証には，DRC (Design Rule Check) を用いて行っている．

以上のように，ImPP は，その規模が大きいために設計の階層化を図り，かつ各設計フェーズにおいては，機能分割し設計作業の並列化を図っている．検証面では，各設計フェーズでの検証はもとより，各設計階層間の検証にも，CAD ツールを活用することにより，機能検証の確度を向上させている．

6. おわりに

以上 ImPP のアーキテクチャを示し，ImPP のパイプラインとデータ駆動方式によるハードウェアアルゴリズムが，VLSI の設計に向くことを示した．

今後は，ひとつの LSI チップに要求される機能が，さらに増大し，チップに盛り込むトランジスタ数は，増々多くなろう．このような状況ではチップ上に実現された機能が設計者の意図に沿ったものかの検証方法や，設計開発期間の短縮方法の充実が望まれる．ひとつには，本稿で述べたように，アーキテクチャ面での工夫と，設計の階層化と CAD ツールの活用により，設計や検証を容易にし，設計開発期間の短縮を図ることが掲げられる．しかしながら，現状では人手に頼らざるを得ない設計が多く，人的ミスが避けられないことや，検証面での検証するテストパターンは人手によって作成しなければならず，十分とは言えない．したがって，これらの人手に頼る部分を排除した自動化された設計や検証，すなわちシリコン・コンパイラや自動検証システムなどの実用化が待たれるところである．これからは，これらの自動化に適した VLSI アーキテクチャの検討も進展するであろう．

参 考 文 献

1) Nukiyama, T., Kusano, T., Matsumoto, K., Kurokawa, H., Hoshi, T., Goto, H. and Temma, T.: A VLSI Image Pipeline Processor, Digest of Technical Papers, IEEE Solid-State Circuits Conference, pp. 208-209 (Feb. 1984).
2) 松本，黒川，大内，拔山，星，天満，岩下：画像処理分野をねらったデータフロー型プロセッサ LSI，日経エレクトロニクス，No. 340, pp. 181-218 (1984).
3) Dennis, J. B. and Misunas, D. P.: A Preliminary Architecture for A Basic Data Flow Processor, Proc. of the 2nd Annual Symposium on Computer Architecture. pp. 126-132 (Jan. 1975).
4) Temma, T., Hasegawa, S. and Hanaki, S.: Dataflow Processor for Image Processing, Proc. of Mini and Microcomputers, Vol. 5, No. 3, pp. 52-56 (Jan. 1980).
5) Macsorley, O. L.: High-Speed Arithmetic in Binary Computers, Proc. of the IRE, Vol. 49, pp. 67-90 (Jan. 1961).
6) Rubinfield, L. P.: A Proof of the Modified Booth's Algorism for Multiplication, IEEE Trans. Comput., Vol. C-24, No. 10, pp. 1014-1015 (Oct. 1975).
7) Sasaki, T., Yamada, A., Kato, S., Nakazawa, T., Tomita, K. and Nomizu, N.: MIXS: Mixed Level Simulator for Large Digital System Verification, Proc. of 17th Design Automation Conference, pp. 626-633 (June 1980).
8) Sakuma, H., Fujinami, Y. and Kurobe, T.: NELSIM: A Hierachical VLSI Design Verification System, Proc. of VLSI '83 International Conference, pp. 109-118 (Feb. 1983).
9) Kishimoto, A., Kawanishi, H., Yoshizawa, H., Ohno, H., Fujinami, Y. and Kani, K.: An Interconnection Check Algorism for Mask Pattern, Proc. ISCAS, pp. 669-672 (1979).

(昭和 60 年 4 月 24 日受付)