修士論文

ソフトマクロＣＰＵ上での

ハード／ソフト協調設計と並列処理環境の実現

氏　　名	：	船附 誠弘
学 籍 番 号	：	6124050212-6
指 導 教 員	：	山崎 勝弘 教授
提 出 日	：	2007 年 2 月 16 日

立命館大学大学院　理工学研究科　情報システム学専攻

内容梗概

本論文では，年々市場規模が拡大している FPGA(Field Programable Gate Array)を用いたハードウェア/ソフトウェア協調設計を行い，アプリケーションの性能バランスを考慮したシステム構成を評価，実装した．また，協調設計のシステムを利用し，ソフトマクロ CPU のペリフェラルハードウェア IP として自作のプロセッサコアを接続することにより，ヘテロジーニアスなマルチコア環境の基礎を構築し，演算の並列分散処理を行った．

研究においては，開発ボードとして，Virtex-4 MB LX60 Development Board revision3 を用い，同じく Xilinx 社が自社デバイス用に提供するソフトマクロ CPU の MicroBlaze をメイン CPU として使用した．自作のハードウェアについては，Verilog-HDL で記述した設計データを合成することでメイン CPU と通信し，協調動作処理を行う．

セキュアハッシュアルゴリズム SHA-1 を対象に C 言語によるプログラミングでアプリケーションを記述し，MicroBlaze による処理負荷部の解析とハードウェアとソフトウェアの分割パターン探索を行った．パターン探索においては，回路規模，所要クロックサイクル数，使用メモリ量の 3 つを考慮し，実験の結果，使用メモリ量の変化が微小なことから，回路規模とクロックサイクル数の相関でコストパフォーマンスの高いパターンとハードウェア化の優先度を導出した．

また，上記実験をもとに FPGA での分割処理環境の構築と性能評価の手法を確立し，上記分割処理環境を基礎に自作プロセッサコアを MicroBlaze に接続することで，ヘテロジーニアスなマルチコアによる並列処理環境を実現した．スレーブは本研究室で以前設計した 3 種類のプロセッサ接続が確認でき，総和計算の並列化においてプロセッサ数に応じた性能向上が得られた．

この研究から，FPGA 上でのマルチコア構成での並列分散処理が可能であることを示した．また，汎用性を考慮したプロセッサインタフェースの策定や MicroBlaze によるデータ駆動制御などについて，性能向上を目的とした検討，考察を行った．

目次

図目次

表目次

はじめに

現在，半導体は日常に存在するあらゆる家電製品やパーソナルコンピュータ，自動車，通信機器にまで幅広く用いられており，世界における半導体市場規模が過去最高を更新し続けていることからも半導体が生活の中枢を担っていることは間違いない．このような半導体の生活への浸透に応じて，高機能化や高性能化，低消費電力やコストパフォーマンスなど，半導体製品に対する要求も多様化し，さらにはこれらの要求のバランスを取ることが望まれている．また，組み込み機器の開発サイクルにおいては，激しい競争の中で新製品の投入時期が早くなっており，製品開発の効率化が求められている．

しかし，膨らみ続けるシステム LSI の規模と開発サイクルの短期化，コスト削減による人的投資の困難といった背景の中で現状の設計効率では完全なカバーができず，開発現場は疲弊した状況であるといえる．システム LSI はハードウェアとソフトウェアの混在した製品であり，以前から本研究室ではハード/ソフト協調設計の開発効率の向上を目的に機能ブロック単位のハードとソフトの分割探索に着目し，コストや性能，消費電力などの様々な要求を満たす最適な構成をいかに早く，より正確に求めるかについて日々研究を行ってきた[1-5].

本論文ではソフトマクロ CPU である MicroBlaze を用いたアプリケーションの設計空間探索と並列処理環境の実現を目的とする．具体的には FPGA 上にハード/ソフト協調システムを実現し，機能ブロックのハードウェア化による様々なパターン毎の回路規模や所要クロックサイクル数，メモリ使用量の測定を行い，最適な構成の探索を行う．対象アプリケーションは米国標準技術研究所(NIST)によって 1995 年米国政府の標準ハッシュ関数として採用された SHA-1 を用いた. SHA-1 はハッシュ関数として世の中の様々なアプリケーションやプロトコルに用いられている．また，上記実験によって構成された FPGA システムを参考に専用ハードウェアの替わりにオリジナルのプロセッサを接続し，処理を分担して行う並列処理環境を提案し，実現，性能検証も行っている．

現在の高性能 FPGA ではプロセッサを複数台実装できる十分な回路規模があり，FPGA 最大の利点である回路の再構成が可能なことからデジタル家電製品においても ASIC で賄ってきた部分を，FPGA で実現されるケースが増えている．システム LSI におけるハードウェアアクセラレーションの恩恵は依然大きいが，アルゴリズムの進化などによる仕様変更には非常に脆く，専用ハードウェア設計に要するコストも大きくなることが予想できる．そこで FPGA でマルチコア環境を構築することで，処理のスループットを向上し，再利用性，柔軟性の高いソフトウェア資源を同じく再利用性，柔軟性の高いソフト・マクロ CPU で処理することでシステム設計における開発サイクルの短縮化に貢献することが目標である．

FPGA 上での並列処理環境の構築においては，MicroBlaze に複数のオリジナルプロセッサを直結する構造となっており，MicroBlaze を全体処理するマスタプロセッサとする．本

研究で接続するスレーブプロセッサには教育用マイクロプロセッサ KUE-CHIP2 [11-15]と RSIC アーキテクチャである SOAR，MON を用いた [16][17]．これらは高性能計算研究室の設計資産として HDL データで存在する．32 ビットに拡張した KUE-CHIP2 を複数台接続し，所要クロックサイクル数を計測することで並列処理環境の有効性の検証を行う．

また，並列処理環境の汎用性を追及するために異なるアーキテクチャのプロセッサコアを接続し，どのようなプロセッサも接続できるようなインタフェースの考察を行っている．さらに，FSL(Fast Simplex Link)による高並列構成の方法論についても言及を行い，並列処理環境の改善や有効性を高めるための検討を行う．

本論文の構成としては, 第 2 章では MicroBlaze を用いたハード/ソフト協調設計や FPGA システムの構成について述べ, 第 3 章で SHA-1 を対象アプリケーションとしたハード/ソフト協調設計手法の検証を，第 4 章では MicroBlaze と教育用プロセッサ KUE-CHIP2 による並列処理環境の構築と検証，そして第 5 章では FPGA マルチコア環境の高性能化や高並列構成への検討について述べる．

2. ソフトマクロCPU上でのハード/ソフト協調設計

2.1 ソフトマクロCPU上でのシステム構成

近年，FPGAの市場規模は年々上昇傾向にあり，それに伴って回路の大規模化や低消費電力化および低コスト化が進んでいる．今やシステムLSIの必要条件ともいえるプロセッサコアも１つのFPGA内に搭載が可能である．大手FPGAメーカー各社はデバイスの優位性を示す指標の１つとして自社デバイス専用のソフトマクロCPUをユーザに提供できることが当然となってきた．ソフトマクロCPUとは回路設計情報として提供され，FPGAに実装できるCPUであり，LUTやマクロブロックなどのFPGA内部の構成要素を活用してマイクロプロセッサの機能を実現することができる．本研究ではXilinx社のFPGAを用い，同社が提供するソフトマクロCPUのMicroblazeを用いてシステムを実現している． 図 1にMicroBlazeを用いたFPGAシステムの構成を示す．MicroBlazeはソフトウェアの実行とともにシステム全体の制御を行う．OPBにはホストPCとの通信を行うUARTやボードに付属している大規模SRAMとの通信を制御するDDR Controler，またGPIOと呼ばれるインタフェースを用いたハードウェアの接続が可能である．また，OPBを介さずにFSLというインタフェースを用いることでMicroBlazeとハードウェアを直結することが出来る．

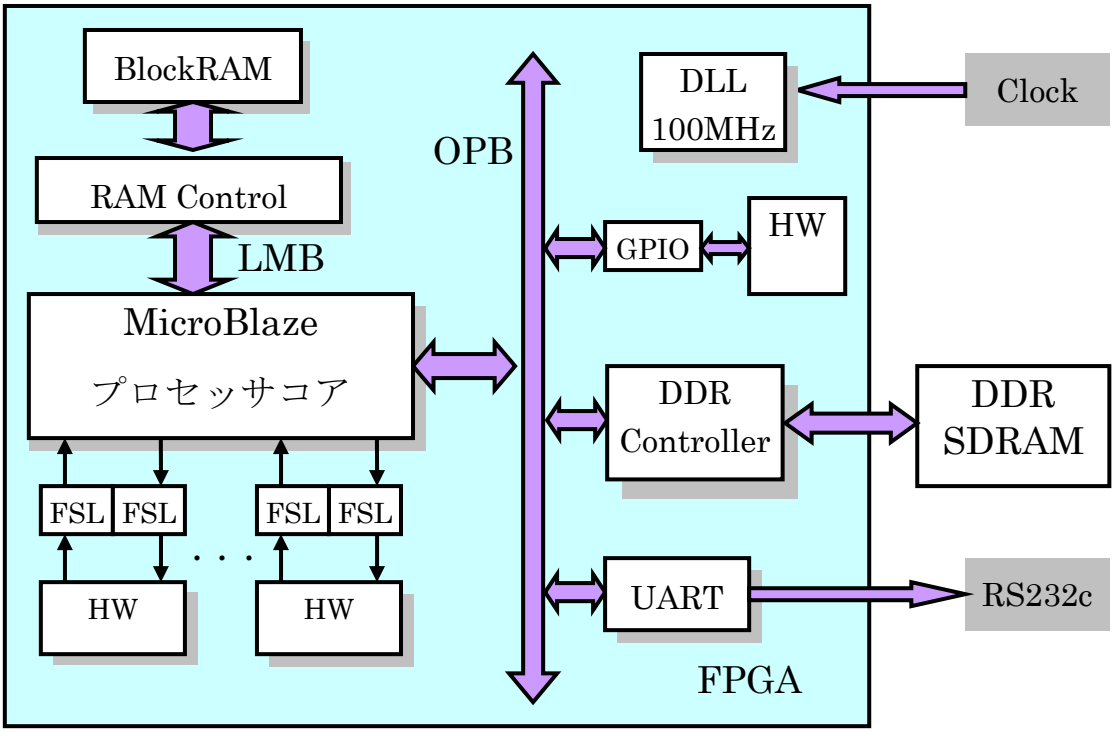

図 1 MicroBlaze を用いた FPGA システムの構成

MicroBlaze を用いたシステムの開発環境として XilinxPlatformStudio(XPS)が用意されており，実装する CPU コアの設定や周辺のモジュールとの接続設定を，XPS を用いてカスタマイズできる．ソフトマクロ CPU と ASIC で設計されたいわゆるハードマクロの汎用 CPU を比較した場合，ソフトマクロ CPU はソフトウェアメモリ使用量や浮動小数点ユニットなどの様々なオプションを自由に設定することができ，必要な機能だけを FPGA 上に実装するために非常にデバイスの使用効率が高い．またもう１つの利点として，CPU コアと外部メモリや I/O 機器などの接続を制御するバスの構成も FPGA で自由に定義することができるため，追加機能による仕様変更にも柔軟に対応することができ，自分で HDL や EDIF でハードウェアアクセラレーター設計し，ツールが提供するインタフェースを用いてシステムの高速化も容易に実現することができる．本研究では MicroBlaze にオリジナルの IP を接続することにより，FPGA を用いたシステムの有用性を追求していく．

2.2 ハード/ソフト協調設計フロー

図 2に本研究におけるアプリケーションのハード/ソフト協調設計のフローを示す．

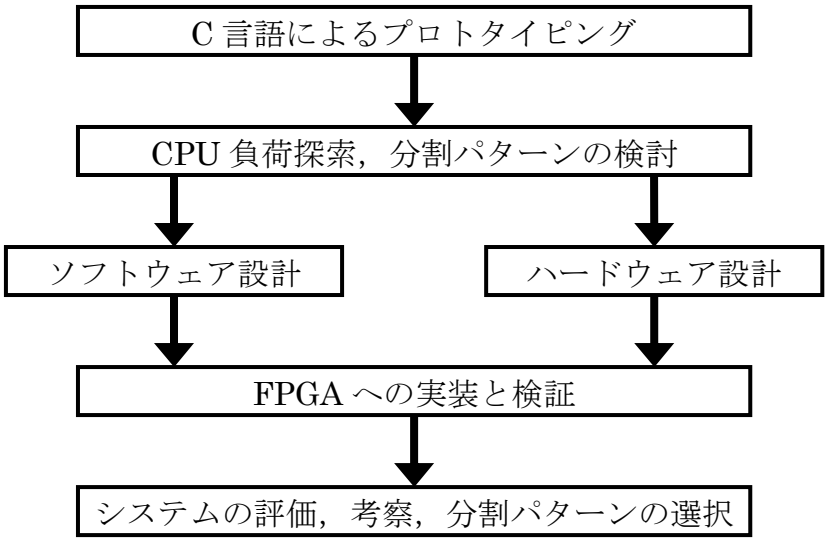

図 2 ハード/ソフト協調設計フロー

システム LSI にアプリケーションを実装するにはまず C 言語でアプリケーションのプロトタイピングが必要となる．このフェーズはシステム設計において最も重要であり，要求を正確に把握し仕様定義を行わなければならない．アプリケーションをどのようなモジュールに切り分けるのか，切り分けたモジュールの入出力仕様はどのようにするのかなど判断を誤るとウォータフォール設計である本フローでは後に大きな手戻りが生じてしまう．次に，プロトタイピングしたアプリケーションを MicroBlaze 上

で実行し，機能モジュール単位での処理負荷割合を測定する．測定にはハードウェアで実装したクロックカウンタを MicroBlaze に接続して確認することができ，所要クロックサイクル数からモジュールのハードウェア化に優先順位をつける．優先順位を元にハードウェアとソフトウェアの分割候補を決めた後，分割パターンに応じた設計を行う．ソフトウェアはプロトタイピングで設計したソースファイルを使用できるので実際には HDL によって専用ハードウェアを設計していく．設計が完了すれば分割パターン毎に FPGA 上で順次実装を行い，所要クロックサイクル数の改善や回路規模増加量, SW メモリ使用量などを分析して要求に最も適した分割パターンを決定し採用する．

2.3 XPSによるMicroBlazeシステムの開発

MicroBlaze を用いた FPGA システムの設計および実装は XPS で行うことができる．本研究では XPS8.1i を用いるが，論理合成，マッピング，配置配線などのツールは XPS と協調関係のある Xilinx 社の ISE8.1i から呼び出して使用している．個別に設計したハードウェア IP の検証は MentorGraphics 社の ModelSimXEⅢ6.0d を用いて行った．開発においてはまず，BSB(BaseSystemBuilder)ウィザードを用いてシステムの土台を設定する．BSB での設定は実装に使用する FPGA ボードの種類によってそれぞれ異なるので事前にボードの構成情報をベンダーから取得しておく必要がある．BSB により SW メモリ使用量やシステムクロック周波数，浮動小数点ユニットの有無，SRAM や rs232c などの外部周辺モジュールとの接続設定を対話形式で決定していく．次に MicroBlaze に接続する自作 IP の取り込みと結合を行う．事前に機能検証を行った HDL 記述に対して，MicroBlaze との通信インタフェースモジュールに組み込むことで追加することができる．統合されたハードウェア情報に論理合成，マッピング，配置配線を行い FPGA 書き込み用のビットストリームを生成する．この段階でハードウェアの実装が可能となる．次にソフトウェア側の設計では MicroBlaze 上で処理するための C プログラムを用意し，コンパイルを行う．自作 IP との通信を行うマクロは XPS のライブラリ内に保持されており，各ライブラリとのリンクを行うことによって実行制御が可能となる．コンパイルによって生成された実行形式ファイルが MicroBlaze の Block RAM に格納されるように FPGA ビットファイルの書き換えを行い，最終的な書き込み用データを生成する．コンフィギュレーションによって FPGA に書き込まれると同時に main 関数の処理が起動し，rs232c を通じて標準出力を確認することで，所要クロックサイクルの測定やプロトタイプとの一致検証を行うことができる．

3. セキュアハッシュSHA-1 のハード/ソフト協調設計手法の検証

3.1 SHA-1 のアルゴリズム

SHA-1(Secure Hash Algorithm 1)は，米国標準技術研究所(NIST)によって 1995 年米国政府の標準ハッシュ関数として採用されている．アルゴリズムは同じくハッシュアルゴリズムである MD4 から発展させたものであり，MD5 よりも攻撃に対する耐性が強いと考えられている．応用範囲としては TLS，SSL，PGP，SSH，S/MIME，IPSec などの様々なアプリケーションやプロトコルに採用されている．SHA-1 の仕様は FIPS180-2(Federal Information Processing Standards 180-2)に記載されている．

図 3にSHA-1 アルゴリズムのハッシュ生成フローを示す.

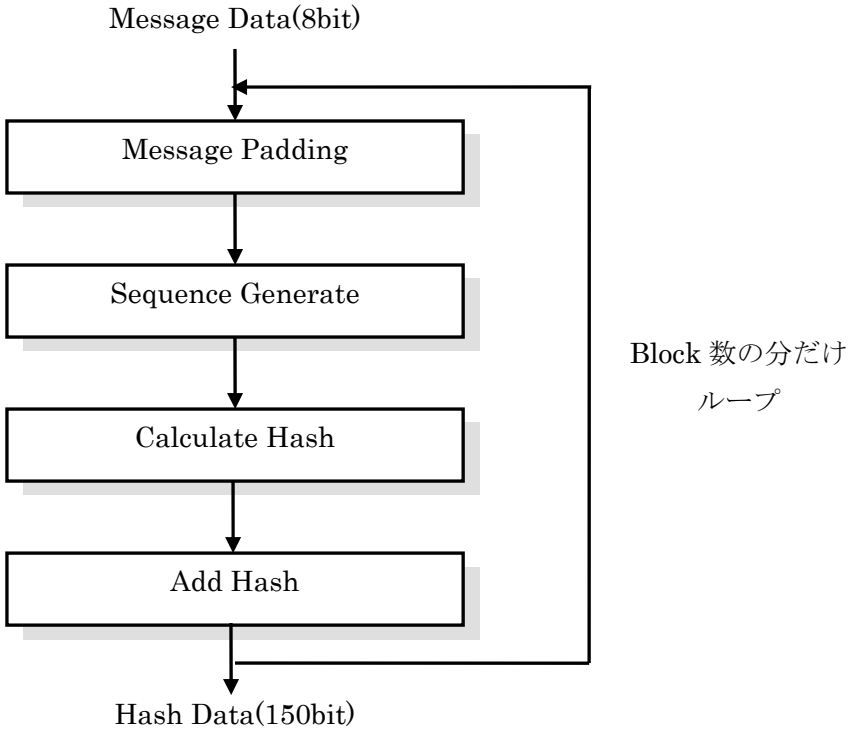

図 3 SHA-1 のハッシュ生成フロー

本研究においては入力のメッセージデータは 8bit 単位で受け取るものと想定している．SHA-1 アルゴリズムでは一般的にメッセージデータをパッキングし，512bit にまとめられたデータのことをブロックという．メッセージデータのパッキングを MessagePadding が担当する．一般的にメッセージデータ長は可変長であるので，生成されるブロックの個数はメッセージデータ長に依存関係がある. ブロック化されたデータから SequenceGenerate, CalculateHash を経てハッシュデータを生成し, ブロック個数の分だけ AddHash で更新することで最終的な 150bit のハッシュデータが生成される．ハッシュ生成における計算量はブロック個数分の処理で決まるので，ブロック個数に比例して大きくなる．以下，

MessagePadding，SquenceGeneate，CalcHash，AddHash の処理内容を説明していく．FIPS180-2 に記載されている SHA-1 の仕様には各機能ブロックの名称が明確に定義されていないため，本研究においては上記モジュール名を便宜的につけている．

3.1.1 Message Padding

Message Padding では入力されるメッセージデータを 512bit のブロックデータにパッケージングして出力する．通常は入力されてくる 8bit のメッセージを 64 個受け取って出力するだけの処理になるが，全メッセージ入力の受け取り後，以下のデータのパディングを行う．

- 最後のメッセージデータの直後に終端ビット'1'を付加
- ブロックの最後の 64bit にメッセージデータ長(bit)を付加

後者のメッセージデータ長の付加については最終ブロックにかかるメッセージデータ数によってはブロック長が変化するため，カウンティングによる制御が必要である．具体的な例を FIPS180-2 に記載されているメッセージデータを例として説明する．

(1) メッセージ"abc"(3 文字)の場合

メッセージデータ数が 3 文字のときのパディングの様子を図 4に示す．

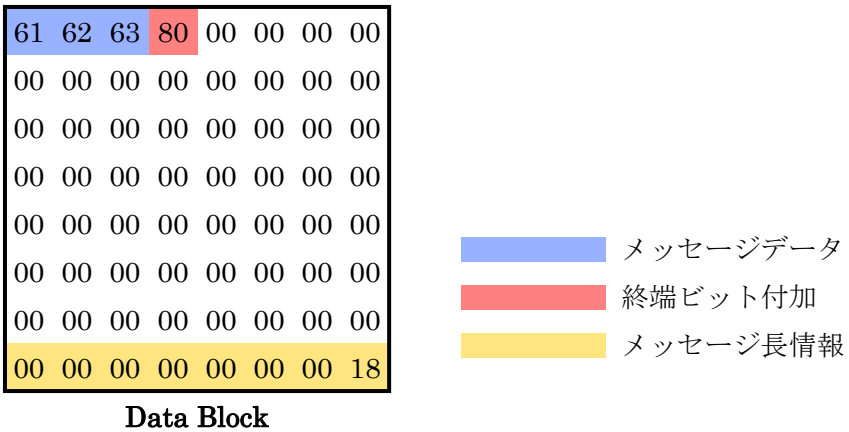

図 4 メッセージデータ数 3 のパディング

この場合ではメッセージデータ(16 進表示 61,62,63)が順次入力され，終端ビット'1'が挿入される．ただしメッセージデータは 8bit 単位なので実際は 16 進表示 80(2 進 10000000)が挿入されることになる．図における 00 は空白メッセージに相当し，8 文字以上の空白メッセージがあるためメッセージ長情報を同一ブロック内に挿入でき，パディングにおける制御は比較的容易である．

(2) メッセージ"abcdbcdecdefdefgefghfghighijhijkijkljklmklmnlmnomnopnopq"(56 文字)の場合

メッセージデータ数が 56 文字のときのパディングの様子を図 5に示す．この場合でもメッセージデータと終端ビットの挿入は 3 文字のときと同様に行われるが，この時点でメッ

セージ長情報を格納する 64bit分の空白が確保できていないことがわかる．このとき残りの空白にはメッセージ長情報は格納せずに新しいデータブロックを生成し，空白メッセージとメッセージ長を挿入して処理をしなければならない，この場合のパディングの制御は追加のブロック生成をしなければならないことや，新規ブロックに応じてプログラム全体の制御にもブロック数が増えることを把握させなければならず，煩雑になる．

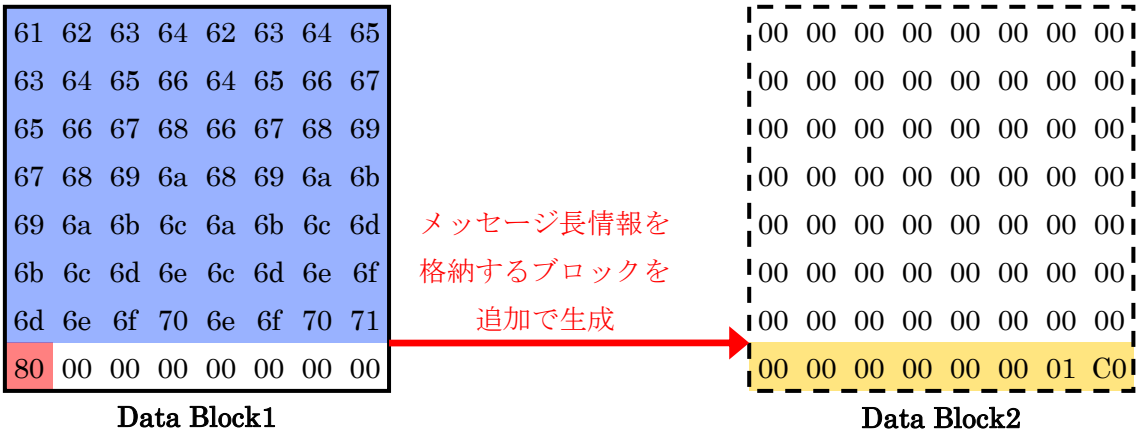

図 5　メッセージデータ数 56 のときのパディング

3.1.2　Sequence Generate

Message Paddingで生成されたブロックデータから 32bitを 1 ワードとする要素数 80 の配列状のワードシーケンスを生成する．その際ブロックデータは図 6のように 32bit単位の 16 個のワードデータに分割される．

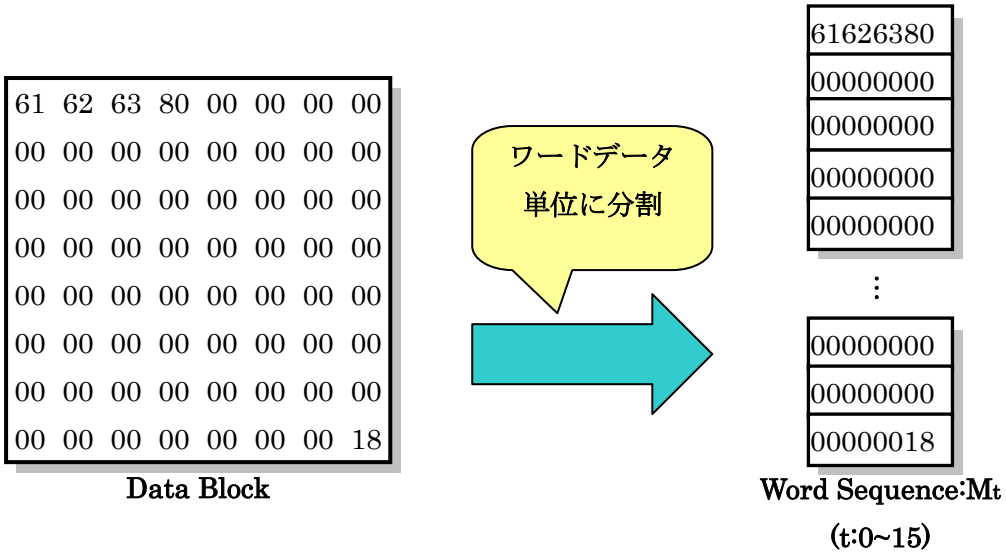

図 6　ブロックデータのワード分割

図 6で示す 16 個のワードデータを M_t としたとき，生成されるワードシーケンス W_t は次式のようになる．

$$W_t = \begin{cases} M_t & 0 \le t \le 15 \\ ROTL^1\left(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}\right) & 16 \le t \le 79 \end{cases} \quad (3.1)$$

式(3.1)における $ROTL^1$ は 1 ビットの右ローテート，⊕ は排他的論理和を意味する．ワードシーケンスの生成においてはメッセージデータそのものをワードとして出力した後，過去の出力ワードを抽出し，フィードバックを繰り返しながら新たなワードシーケンスを生成していき，要素数 80 になるまで処理を継続する。

3.1.3 Calculate Hash

Calculate Hash では Sequence Generate で生成したワードシーケンスを用いて一時的な 150bit のハッシュ値を求める．32bit の内部変数 a，b，c，d，e に対して演算を行うがまず初期化処理として次式のようにその時点で最新の中間ハッシュ値 $H_n^{(i-1)}$ を代入する．

$$\begin{aligned} a &= H_0^{(i-1)} & \left(H_0^{(0)} &= 67452301\right) \\ b &= H_1^{(i-1)} & \left(H_1^{(0)} &= efcdab89\right) \\ c &= H_2^{(i-1)} & \left(H_2^{(0)} &= 98badcfe\right) \\ d &= H_3^{(i-1)} & \left(H_3^{(0)} &= 10325476\right) \\ e &= H_4^{(i-1)} & \left(H_4^{(0)} &= c3d2e1f0\right) \end{aligned} \quad (3.2)$$

式中の i はブロックの番号であり，最初のブロック番号は 1 である．ブロック番号 1 に対応するハッシュ値は式中右側カッコ内の値であり．後述する Add Hash を経て更新される．初期化された中間変数とワードシーケンス配列を組み合わせた演算内容は以下の通りになる

$$\begin{aligned} &for\ \ t = 0\ \ to\ \ 79 \\ &\{ \\ &\quad T = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t \\ &\quad e = d \\ &\quad d = c \\ &\quad c = ROTL^{30}(b) \\ &\quad b = a \\ &\quad a = T \\ &\} \end{aligned} \quad (3.3)$$

処理内容としては各変数間のシフトを繰り返し，変数間に論理演算やテーブル参照などを行う．本研究ではこれらの処理の一部を Calculate Hash 内部でのサブモジュールとして配置しており，Calculate Hash 同様サブモジュール名も便宜的に命名している．モジュー

ル名と各処理の詳細を次に示す.

(1) SHA function

この処理は式(3.3)の $f_t()$ に相当する．処理内容はワードシーケンス配列の要素番号に応じて以下のように変化する.

$$f_t(x,y,z)=\begin{cases}(x\wedge y)\oplus(\neg x\wedge z) & 0\le t\le 19\\ x\oplus y\oplus z & 20\le t\le 39\\ (x\wedge y)\oplus(x\wedge z)\oplus(y\wedge z) & 40\le t\le 59\\ x\oplus y\oplus z & 60\le t\le 79\end{cases} \tag{3.4}$$

(2) Get Constants

この処理は式(3.3)の K_t に相当する．処理内容はワードシーケンス配列の要素番号に応じて以下のようなコンスタント値を返す.

$$K_t=\begin{cases}5a827999 & 0\le t\le 19\\ 6ed9eba1 & 20\le t\le 39\\ 8f1bbcdc & 40\le t\le 59\\ ca62c1d6 & 60\le t\le 79\end{cases} \tag{3.5}$$

(2) Rotate Left

この処理は式(3.3)の $ROTL^n$ に相当する．処理内容は n bit の右ローテートを行う.

$$ROTL^n(x)=(x<<n)\vee(x>>32-n) \tag{3.6}$$

3.1.4 Add Hash

Add Hash では Calculate Hash で生成した一時的な 150bit のハッシュ値を加算し, 中間ハッシュ値 $H_n^{(i-1)}$ を更新する.

$$\begin{aligned}H_0^{(i)}&=a+H_0^{(i-1)}\\ H_1^{(i)}&=b+H_1^{(i-1)}\\ H_2^{(i)}&=c+H_2^{(i-1)}\\ H_3^{(i)}&=d+H_3^{(i-1)}\\ H_4^{(i)}&=e+H_4^{(i-1)}\end{aligned} \tag{3.7}$$

3.2　CPU上での過負荷部解析と分割検討

ハードウェアとソフトウェアの分割パターンを検討する上で，処理時間は重要な要因のひとつとして挙げられる．そこでMicroBlazeを用いてSHA-1 における機能ブロック毎のCPU過負荷部解析を行った．具体的な解析方法として事前にVerilog-HDLでハードウェア設計したクロックカウンタをMicroBlazeに接続し，各機能ブロックが何サイクル処理に要したかを測定し，相対的な割合を求める．計測対象のメッセージデータはFIPS180-2 の仕様書にも記載されている”abc”の 3 文字分のデータを使用した．生成ブロック数が 1 つであるため，全体の所要クロックサイクル数を 100%として測定している．また，XPSによるC言語のコンパイルには最適化レベルが選択でき，今回はLevel：High(-O2)でコンパイルを行っている．FPGA実装後測定した各機能ブロックの所要クロックサイクル数を表 1に，CPU負荷割合をグラフ化したものを図 7に示す．

表 1　SHA-1 各機能ブロックの所要クロックサイクル数

module	Initialize	MessagePadding	SequenceGenerate	CalculateHash	AddHash	Total
Clock	43	1231	4279	7102	75	12724

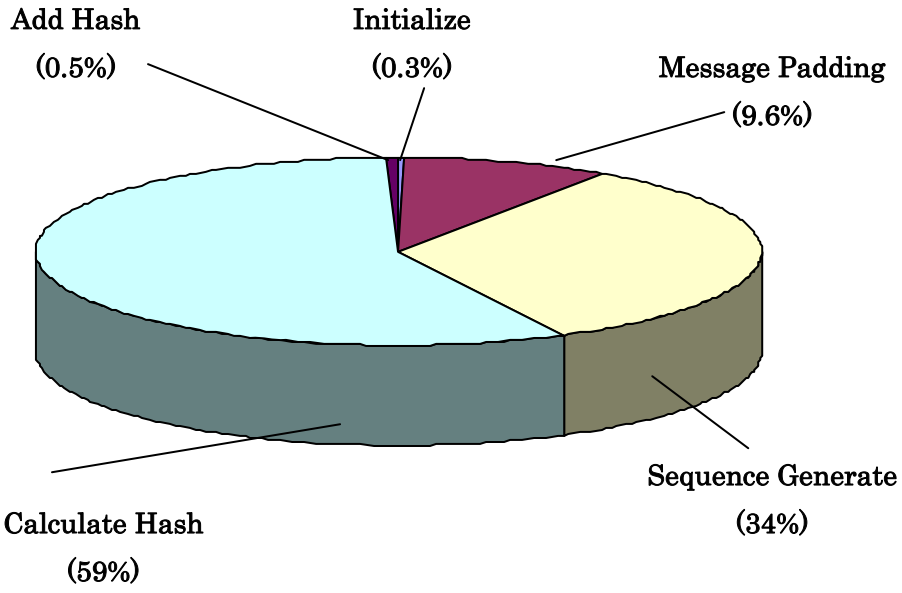

図 7　SHA-1 の負荷割合

図 7のグラフからCalculate HashとSequence Generateの負荷が大きいことがわかる．特にCalculate Hashが 59%と処理全体の半分以上の負荷を占めている．Calculate Hashは

3.1.3 で説明したように他のモジュールに比べて様々な演算要素が凝縮されており，さらに 3 つのサブモジュールが存在する．本研究ではMicroBlazeシステム上での処理内容と負荷の関連性を探索できることが望ましいと考え，Calculate Hash内部のサブモジュールをハードウェアとソフトウェアの分割対象に加えた．

Message Padding モジュールについては全体の処理負荷に対して約 1 割程度を占めているが，処理内容の性質上入力データそのものに加工を行うわけではなく，様々な制御情報からブロック情報の付加を行うため，ハードウェアでの実装は処理内容に対して複数の制御信号が必要となるのでソフトウェアで実装することにしている．

以上を考慮し，全体処理に対してハードウェア化する部分とソフトウェアで処理する部分の切り分けパターンを表 2のように選出した．

表 2　機能ブロック単位での切り分けパターン

	ハードウェア処理部	ソフトウェア処理部
S		制御 , MessagePadding SequenceGenerate , RotateLeft SHAfunction , GetConstant , AddHash
A	RotateLeft	制御 , MessagePadding , SequenceGenerate , SHAfunction , GetConstant , AddHash
B	RotateLeft , GetConstant	制御 , MessagePadding , AddHash , SequenceGenerate , SHAfunction
C	RotateLeft , GetConstant , SequenceGenerate	制御 , MessagePadding , SHAfunction , AddHash
D	RotateLeft , GetConstant , SequenceGenerate , SHAfunction	制御 , MessagePadding , AddHash
E	RotateLeft , GetConstant , SequenceGenerate , SHAfunction , AddHash	制御 , MessagePadding

表 2の切り分けパターンのうち，パターンSはすべてをソフトウェアで実行することを示し，検証における評価基準となる．Aから順に処理負荷割合の高いモジュールをハードウェア側に追加していき，パターンEがMicroBlaze上で最もハードウェア比率が高い構成となっている．これらのパターンに対して実行クロックサイクル数，実装回路規模，使用ソフトウェアメモリ量を解析し，最適なパターン探索を行う．

3.3　各モジュールのハードウェア設計

SHA-1 のハードウェア/ソフトウェア協調設計システムでは, MicroBlazeと各専用ハードウェアとの接続にFSL(Fast Simplex Link)を使用する．FSLを用いた専用ハードウェアの接続を図 8に示す.

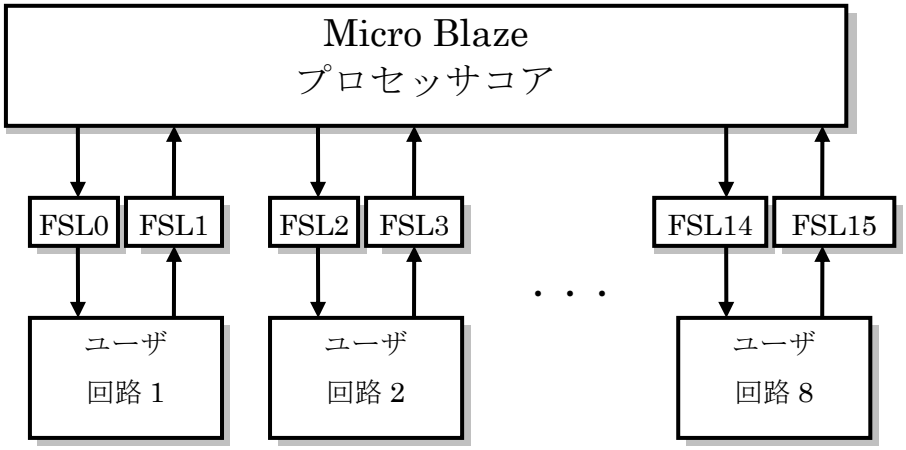

図 8　FSL によるハードウェアの接続

FSLはFIFOベースの通信インタフェースであり，一方向通信のため一つのユーザ回路の接続には基本的に 2 つのFSLを用いる．MicroBlazeでは最大 8 つまでユーザ回路をFSLで繋ぐことができ，各通信がポイント間で専用に用いられるため，アービトレーションの必要が無く高速な動作が可能である．図 9にFSLアーキテクチャのブロック図を，表 3に入出力リストをそれぞれ示す.

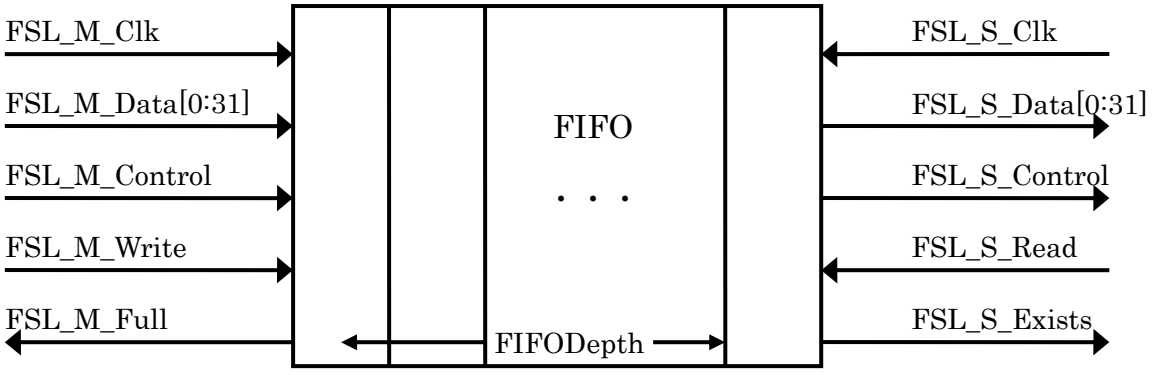

図 9　FSL のブロック図

FSL のデータバス幅は 8，16，32 ビット，FIFO の深さも共にコンフィギュレーションによって自由に定義することが可能である．MicroBlaze による FSL バスへの書き込みは put 命令によってレジスタの内容が FSL に転送される．この転送にかかる時間は通常 FSL の FIFO がフルでなければ 2 クロックサイクルで完了し, FIFO がフルの場合は書き込みが

行われず，FSL_M_Full が Low になるまでプロセッサが停止する．一方，MicroBlaze への FSL バスからの読み出しは get 命令によって FSL の内容が汎用レジスタに転送される．この転送にかかる時間は通常 FIFO にデータが存在すれば 2 クロックサイクルで完了し，FIFO にデータが無ければ FSL_S_Exists が High になるまでプロセッサが停止する．

表 3　FSL の入出力リスト

ポート名	I/O	ビット幅	内容
FSL_M_Clk	in	1	マスター側のクロック
FSL_M_Data	in	32	FIFO に書き込まれるデータ値
FSL_M_Control	in	1	FIFO に書き込まれる制御ビット値
FSL_M_Write	in	1	FIFO へのデータ書き込みを示す ACK 信号
FSL_M_Full	out	1	FIFO がフルであることを示す制御信号
FSL_S_Clk	in	1	スレーブ側のクロック
FSL_S_Data	out	32	FIFO から取り出されるデータ値
FSL_S_Control	out	1	FIFO から取り出される制御ビット値
FSL_S_Read	in	1	FIFO のデータ読み出しを示す ACK 信号
FSL_S_Exists	out	1	FIFO からデータを取り出せるかを示す信号

MicroBlazeにFSLによって自作した回路を接続する場合，設計する回路のインタフェースはFSLで通信できるように調整しなければならない. 本研究におけるFSLを用いたユーザ回路の構成を図 10に示す.

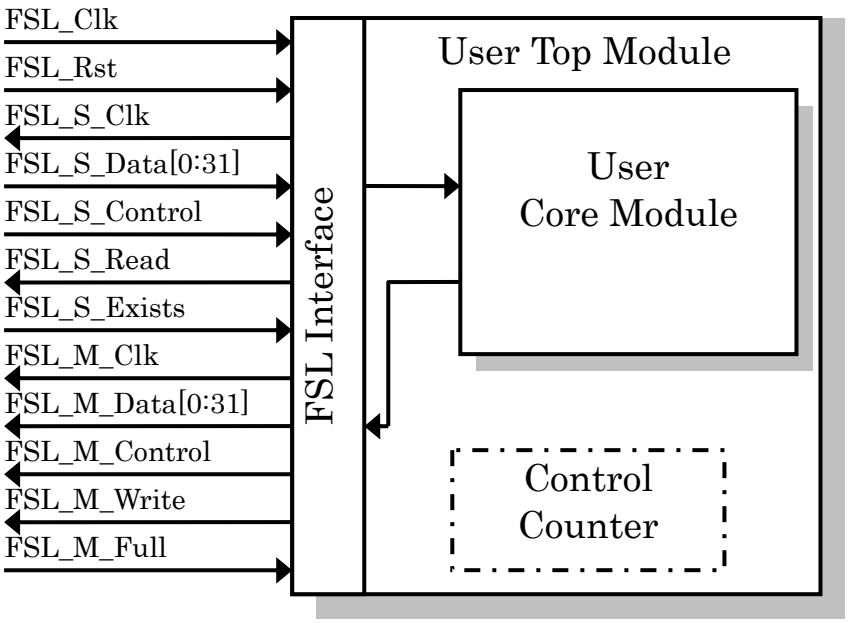

図 10　FSL を用いたユーザ回路の構成

ユーザ回路は FSL の通信を制御するトップモジュールとデータの演算処理を行う内部コアモジュールで構成される．SHA-1 のハードウェア化対象モジュールはすべて所要クロックサイクル数がデータに依存せず常に固定であるので，トップモジュールで入出力データ数をカウントしながら入出力や内部コア部分の制御を行っている．以下に SHA-1 のハードウェア化対象モジュールのコア部分の実現について述べる．

3.3.1 Sequence Generate

図 11にSequence Generateのブロック図を，表 4にその入出力を示す．Sequence Generateは 32 ビットの入力データを 16 サイクルかけてシフトレジスタに格納し, 80 サイクルかけて 32 ビットのワードシーケンスデータを出力する．シフトレジスタ中の 4 つの要素を抽出し，XORと 1 ビットの右ローテート処理を経てシフトレジスタの入力として挿入され続ける．

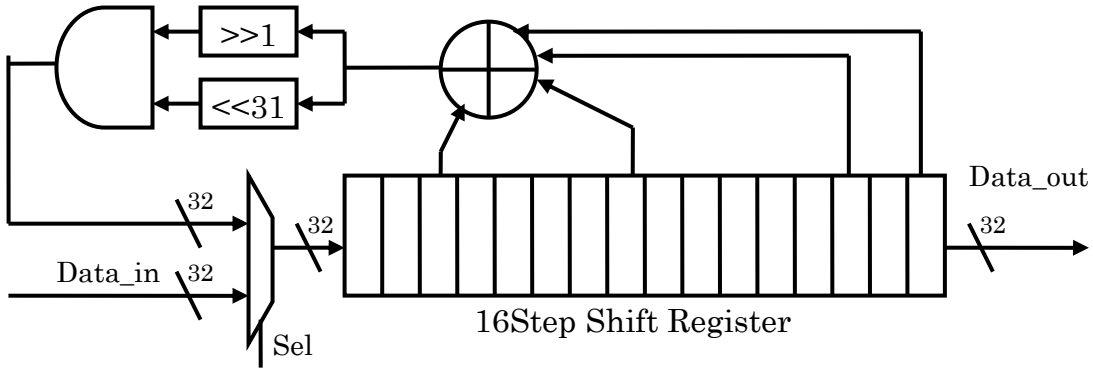

図 11　Sequence Generate のブロック図

表 4：Sequence Generate の入出力リスト

ポート名	I/O	ビット幅	内容
Data_in	in	32	メッセージワードデータ入力
Sel	in	1	シフトレジスタ入力の選択信号
Data_out	out	32	ワードシーケンス出力

3.3.2 SHA function

図 12にSHA functionのブロック図を，表 5に入出力リストを示す．SHA functionは 3 つのデータ入力に対する論理演算を行うモジュールで論理演算の内容は上位モジュールであるCalculate Hashのループが何回目であるかによって変化する．

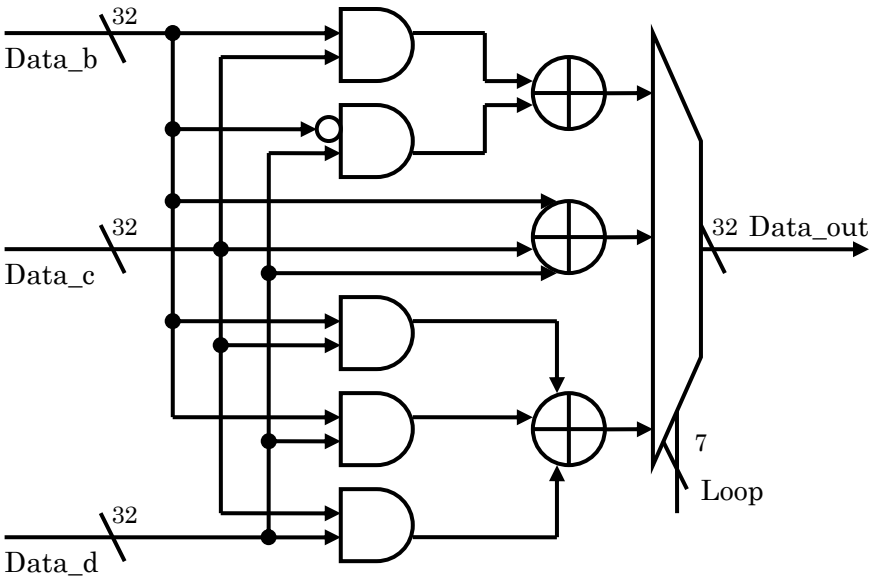

図 12　SHA function のブロック図

表 5　SHA function の入出力リスト

ポート名	I/O	ビット幅	内容
Data_b	in	32	レジスタ B からのデータ入力
Data_c	in	32	レジスタ C からのデータ入力
Data_d	in	32	レジスタ D からのデータ入力
Loop	in	7	ループの周回数(0~79)
Data_out	out	32	SHA function 演算出力

3.3.3　Get Constants

図 13にGet Constantsのブロック図を，表 6に入出力リストを示す．GetConstantsはCalculate Hashのループが何回目であるかという情報をもとに定数値を出力する．

表 6　Get Constants の入出力リスト

ポート名	I/O	ビット幅	内容
Loop	in	7	ループの周回数(0~79)
Data_out	out	32	SHA function 演算出力

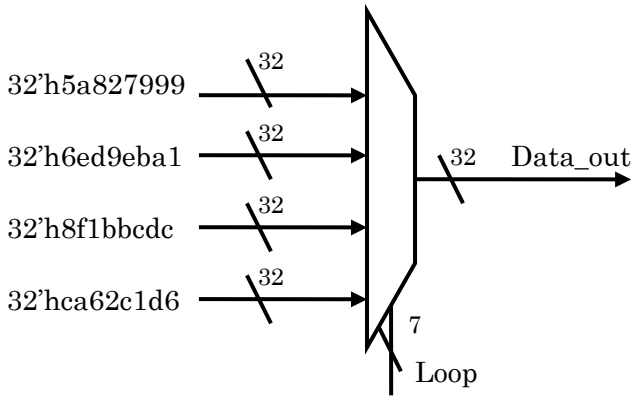

図 13　Get Constants のブロック図

3.3.4　Rotate Left

図 14にRotate Leftのブロック図を, 表 7に入出力リストを示す. Rotate LeftはCalculate Hash内部の右ローテート処理を行うモジュールでCalculate Hashでは 5 ビット，30 ビット二つのローテート処理がある.

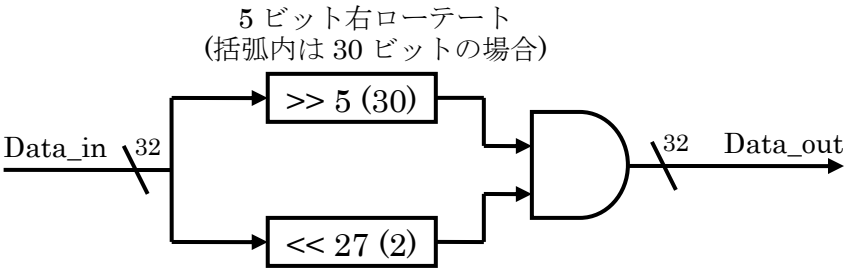

図 14　Rotate Left のブロック図

表 7　Rotate Left の入出力リスト

ポート名	I/O	ビット幅	内容
Data_in	in	32	Rotate Left モジュールデータ入力
Data_out	out	32	Rotate Left モジュールデータ出力

3.3.5　Add Hash

図 15にAdd Hashのブロック図を，表 8に入出力リストを示す．Add HashはCalculate Hashで生成された中間ハッシュデータを加算することによって更新するモジュールである．ハッシュデータは 150 ビットであるが，FSLのビット幅が 32 ビットであるため，設計したモジュールで 5 回処理を繰り返すことで更新を行う．

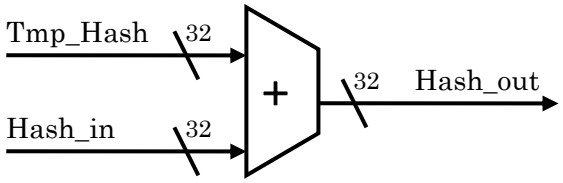

図 15　Add Hash のブロック図

表 8　Add Hash の入出力リスト

ポート名	I/O	ビット幅	内容
Tmp_Hash	in	32	Calculate Hash で生成された中間ハッシュ値
Hash_in	in	32	更新前のハッシュデータ
Hash_out	out	32	更新されたハッシュデータ出力

3.4　SHA-1 分割評価システムの構成

3.4.1　SHA-1 分割評価システム

今回設計したハードウェアとソフトウェアのパターン分割の検証は旧MEMEC社(現Avnet Japan)のFPGAボードに実装した. 図 16にSHA-1 分割評価システムの基本構成を示す.

MicroBlaze プロセッサコアには，メインメモリとして高速通信インタフェースであるLMB(Local Memory Bus)を介して BlockRAM を接続し，ここに命令とデータを格納している．SHA-1 の協調動作のうちソフトウェア部分の実行は MicroBlaze プロセッサコアで処理される．一方，ハードウェア処理の部分は，各モジュールを Verilog-HDL で設計し，プロセッサコアと直接接続するインタフェース FSL(Fast Simplex Link)を用いて実装した. この構成をベースとし，専用ハードウェアの着脱やソフトウェアプログラムの改変などで分割パターンの評価を行った.

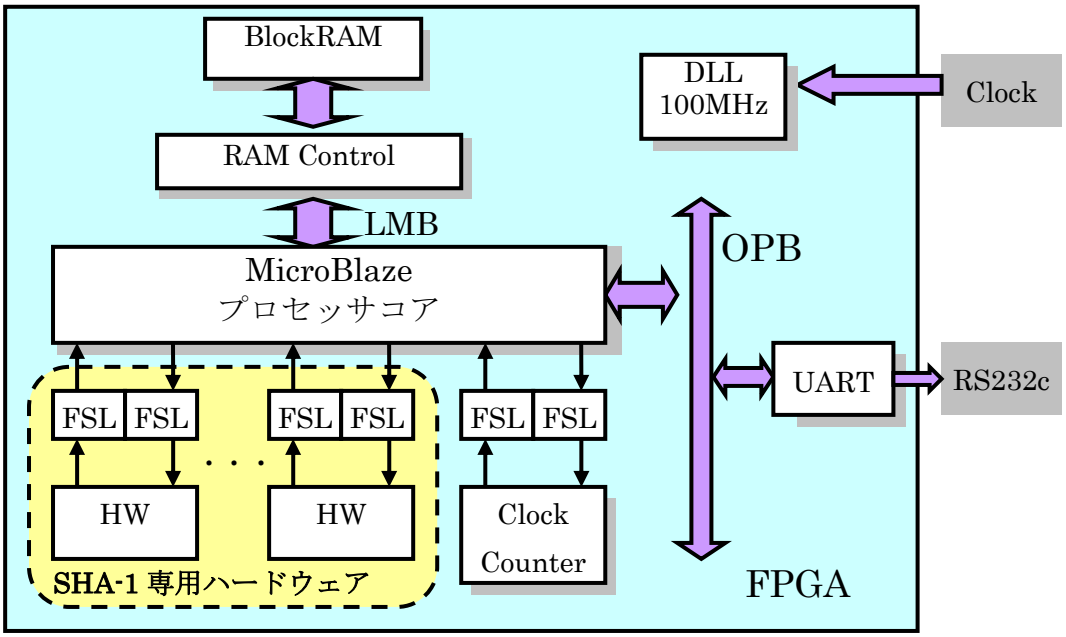

図 16　SHA-1 分割評価システム基本構成

3.4.2　SHA-1 分割評価システムの実行

SHA-1 分割評価システムではFPGAへのコンフィギュレーションの直後にソフトウェア記述のmain関数部の処理が始まる．そのときの実行命令とデータはBlockRAMに格納されているので，MicroBlazeはBrockRAMからロードしながらソフトウェア処理を行う．また，専用ハードウェアに処理をさせる場合は，対象となるデータをFSLに書き込むことでハードウェア処理部に送られる．MicroBlazeではFSL通信用ライブラリが用意されているため，送信回数分および受信回数分の送受信命令を繰り返すことで処理結果を得ることができる．図 17に具体的なハードウェアとの通信を行うためのプログラム記述を示す．

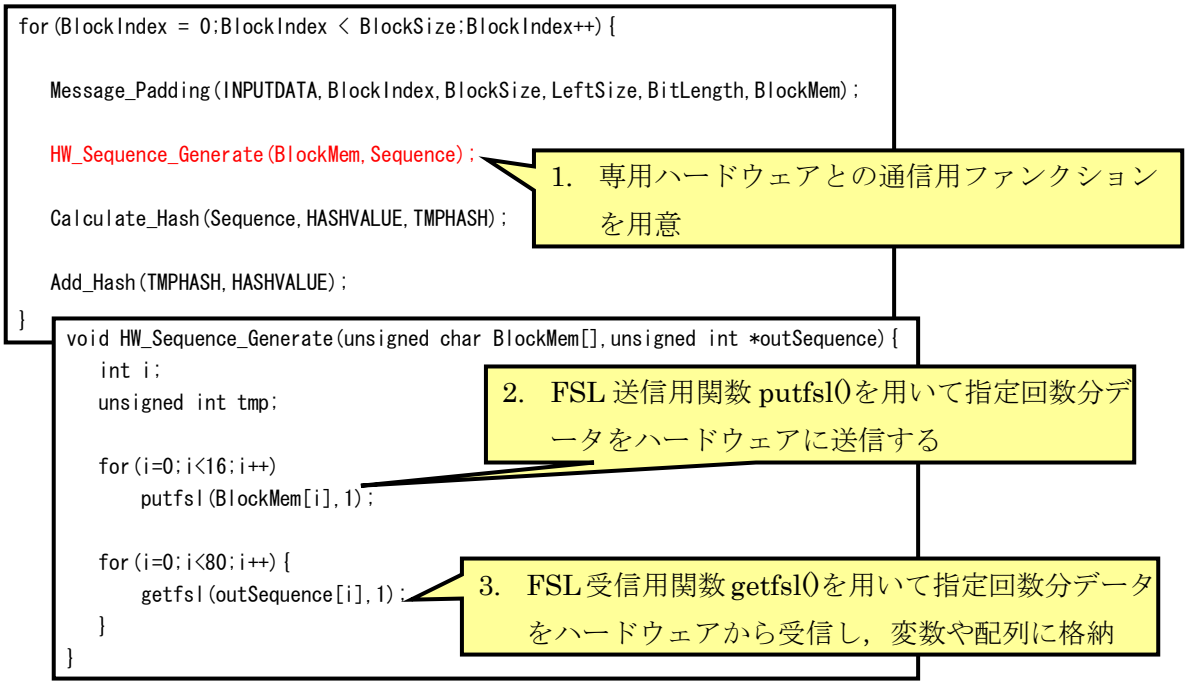

図 17　C プログラム上のハードウェアとの通信記述

FSLを用いたハードウェアとの通信は，MicroBlazeシステムの開発環境ツールであるEDKであらかじめ用意されている通信ライブラリを用いて，目的のハードウェアが接続されているFSLのIDとソフトウェア側の変数を指定することでデータのやり取りができる．今回はfsl.hで定義されているputfsl()とgetfel()を用いてプログラミングを行った．図 17ではSequence Generateをハードウェア化した場合のプログラムであり，通信用のファンクションを用意しておき，ファンクション内部ではputfsl()で 32 ビットブロックワードデータBlockMem[]を 16 回分連続転送し，次にgetfsl()で生成されたシーケンスをワード単位でoutSequence[]に 80 回読み出す．このときgetfsl()はブロッキングを行うため，ハードウェア側から出力データがまだ得られない場合，プロセッサの実行を停止してFSLからデータを取り出せるのを待つ．

評価における処理時間の計測にはハードウェアのクロックカウンタを用いている．クロックカウンタはFSLによって接続されており，ソフトウェア側で制御することで所要クロックサイクル数を取り出すことができる．図 18にCプログラム上でのクロックサイクル数計測の様子を示す．

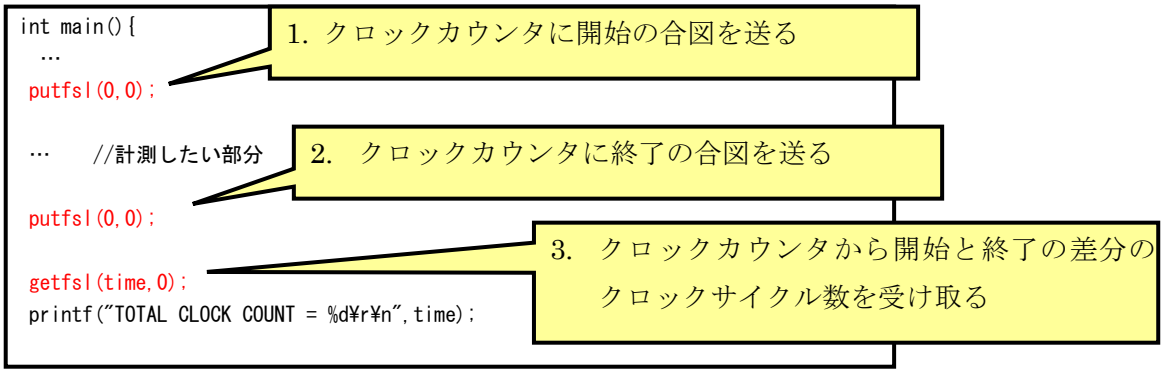

図 18　C プログラム上でのクロックサイクル数計測方法

実行結果やクロックサイクル数などの観測には FPGA 内部の UART(Universal Asynchronous Receiver Transmitter)を使用し，RS232c を介して PC 端末と接続している．ソフトウェアにて標準出力(printf 文)を記述した場合，シリアルケーブルを通じて，PC 画面上に出力される．

3.5　評価結果と考察

3.5.1　評価結果

表 2で選出した切り分けパターンに対する評価結果を表 9に示す．評価に用いたメッセージは”abc”3 文字で，1 ブロック分の処理に要するクロックサイクル数を計測している．実装規模は周辺回路を含めたMicroBlazeシステム全体の論理合成結果を示している．メモリ使用量はXPSでコンパイルしたときに表示される数値を用い，専用ハードウェアで生成されるメモリは含まない．　表 9のパターンHは制御を含めたすべてをハードウェア化した場合であり，MicroBlazeは使用せずISEで合成した結果を示している．

表 9　分割パターン毎の評価結果

	S	A	B	C	D	E	H
クロック サイクル数	12724	10962	10362	7640	7160	7202	144
使用メモリ量 (Byte)	7337	7301	7233	7069	6997	7005	0
回路規模 (Slices)	936	1070	1111	1648	1966	2051	894
速度向上率 (⊿clock / ⊿slices)	0	13.1	14.6	5.1	1.5	-3.5	-

3.5.2 考察

選出した切り分け結果についてまず使用メモリ量に着目すると S から E までの変化量が全体の 5%程度しか削減していないことがわかる．全体量が大きいのはあらかじめメッセージデータを格納していることやモジュール毎に入出力のデータ構造が異なるため，一時保存用のメモリ領域を用意しなければならないという SHA-1 の特徴を示していると考えられる．また，比較的単純な処理の繰り返しによってデータのランダマイズを行っているためメモリ使用量の変化が少ないと見られる．このことからメモリ使用量については判断基準からはずすことができる．

次に所要クロックサイクル数の推移に着目すると S→A と B→C でそれぞれ大きく削減されていることがわかる．前者は右ローテート処理であり，速度向上の要因としては MicroBlaze ではバレルシフタを内蔵していないことから複数ビットのローテート処理に時間がかかっていたことが考えられる．ハードウェアではローテート処理は配線を入れ替えるだけで実現できるため 1 サイクルで処理される(通信は含まない)．後者は Sequence Generate のハードウェア化で処理内容として 4 つのデータの XOR の後，1 ビット右ローテートした結果を出力する．ハードウェアではそれが 1 サイクルで 1 つの出力を得ることができるのに対して，ソフトウェア処理ではデータをメモリからロードし，XOR 演算は初回を除いて一度に 1 つずつしかできない．その後，右ローテート命令の実行後メモリへのストア命令をもってようやくひとつの出力生成が完了する．これらの過程のちがいから Sequence Generate のハードウェア化の速度向上が大きくなると考えられる．

一方で D→E では逆にクロックサイクルが増加してしまっている．これは ADD HASH モジュールのハードウェア化に相当し，FSL が一度に 32 ビットしか通信できないことから通信にかかるオーバヘッドが相対的に大きくなってしまい，ソフトウェア処理のほうが効率的に実行できていることがわかる．他のモジュールのハードウェア化については一定の速度向上が確認できたが，複数の論理演算を一括して行う C→D の SHA function の向上が予測よりも小さかった．これは 4 入力にかかる通信負荷の影響が考えられる．

回路規模の推移に着目すると B→C の増加量が大きいことがわかる．これはハードウェア内部に 16 ステップのシフトレジスタを実装していることが要因に挙げられる．その他のハードウェアについては論理演算器程度の実装のため増分は少ない結果となった．

これら 3 つの要素を総合的に見て，メモリ量の変化の少ない結果となったことを考慮すると，回路規模の増加に対する速度向上に着目した上で優先度を決めることが妥当と考えられる．表 9 のパターンでは，左から順に回路増加分に対する速度向上が高い．そのため実装するデバイスの容量に最も近いパターンを左から順に比較していくことで最もコストパフォーマンスの高いシステムとして最適分割パターンを検出することができると考えられる．

4. ソフトマクロCPU上での並列処理環境の実現

4.1 MicroBlazeとオリジナルプロセッサの接続

3 章ではSHA-1 を用いてアプリケーションのソフトウェア実行，ハードウェア実行のモジュール単位の切り分けについて述べた．プロトタイプの解析によって要求を満たす最適な構成を導出することで制約の厳しい組み込み機器の開発サイクルの短縮に一定の効果が期待できる．一方でFPGAのプロセス技術は現在も向上を続け，同サイズのCHIPにより多くの機能を搭載することが出来るようになった．また消費電力も新しいデバイスシリーズの登場とともに低く抑えられている．このような背景の中，複数のプロセッサコアをMicroBlazeに接続することでMicroBlazeがいわゆるマスタとなるマルチコア構造を構築し，ソフトウェア資源を利用し並列処理実行を行うことでシステムの高速化，開発負担の軽減に役立てることを目的に提案した．図 19に本研究におけるMicroBlazeを用いたFPGA上でのマルチコア構造のイメージを示す．

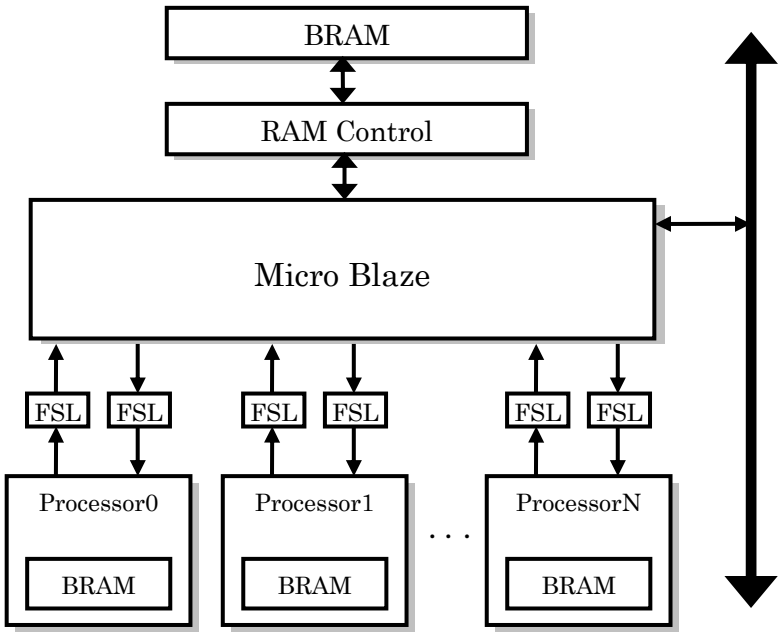

図 19　MicroBlaze を用いたマルチコア構造

各スレーブプロセッサには内部に専用のメモリが搭載されており，MicroBlaze 側のメインメモリに格納されている機械語命令やデータを FSL インタフェースで接続し，通信を行う．命令およびデータを受け取ったスレーブプロセッサは処理を行い，実行が終了した後 FSL を介して演算後の処理結果を返す．各結果は MicroBlaze が受け取り，スケジューリングなどによって最終的な処理結果として結合する．本研究ではこのような構造の MicroBlaze システムを構築しアプリケーションの処理実験などを参考に高効率で汎用性の高いシステム構造に関する考察を行う．

4.2 オリジナルプロセッサとのデータ通信と制御

図 20にFSLを用いたオリジナルプロセッサとのデータ通信構造を示す.

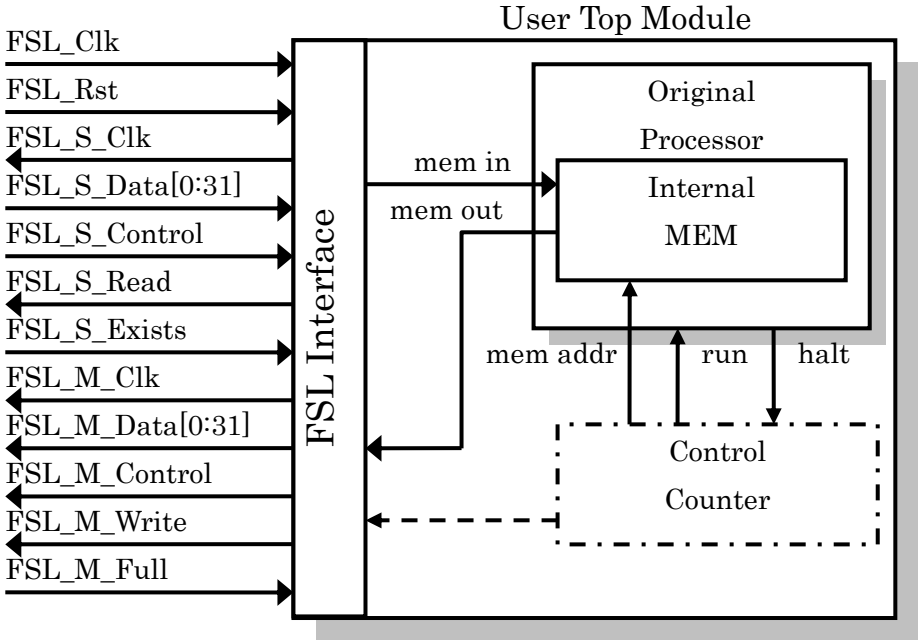

図 20　FSL を用いたオリジナルプロセッサのデータ通信構造

オリジナルプロセッサは FSL の通信を制御するトップモジュールと，内部にメモリを搭載したデータの演算処理を行うプロセッサモジュールで構成される．FSL を介して入力されたデータはオリジナルプロセッサ内部のメモリに書き込まれ，そのときのアドレスはトップモジュールの Control Counter で制御される．処理実行後に同じく Control Counter で指定したアドレスのデータが FSL を介してマスタである MicroBlaze に送信される．本研究で用いることが出来るプロセッサの必要要件として命令とデータが同一であること，内部メモリの場合は外部からのメモリアクセスが可能であること，またはプロセッサコアが外部(トップモジュール)のメモリを使用する場合に MicroBlaze と接続することが出来る．MicroBlaze メモリへの書き込みやメモリデータの MicroBlaze への送信のデータ量はトップモジュールの制御部の数値を変更することで変更することが出来るため，処理内容やプロセッサが使用するメモリ容量に応じて通信にかかるオーバーヘッドを軽減することが出来る．

図 21にFSLによるプロセッサ通信の状態遷移図を示す．FSLを用いた通信においてはシステムリセットによって待機状態であるIdleに遷移する．その後MicroBlazeからデータの送信要求を受けてデータが受け取り可能状態(FSL_S_Existsがアクティブ)のときにデータの読み出し状態であるReadに遷移する．この間受け取ったデータはプロセッサ処理用メモリのControlCounterで生成されたアドレスに書き込まれる．指定されたデータ数のメモリ書き込みが完了すると次はプロセッサの実行状態であるCalcに遷移する．Calcに遷移する

ときにプロセッサのrun信号をアクティブにすることで実行のトリガとする．そしてプロセッサでの実行が終了するときにアクティブになるHalt信号をトリガにMicroBlazeにメモリデータを送信するフェーズであるWriteに遷移する．ControlCounterで生成したアドレスのデータを読み出してFSLを介して送信し，指定送信回数に達すると再び待機状態であるIdleにもどる．このような状態の変化によってプロセッサの通信を行っている．

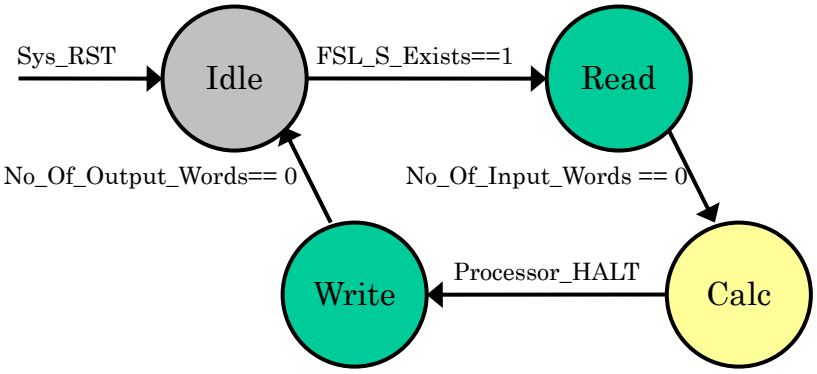

図 21　FSL によるプロセッサ通信の状態遷移図

4.3　教育用プロセッサKUE-CHIP2 とMicroBlazeの接続

4.3.1　KUE-CHIP2 のアーキテクチャと FSL への対応

MicroBlaze と接続するプロセッサに KUE-CHIP2(Kyoto University Education CHIP2)を採用し，VerilogHDL で記述されたソースコードの接続に対応するための修正を行った．KUE-CHIP2 はプロセッサアーキテクチャの教育を目的として開発された 8 ビットのマルチサイクルプロセッサであり，約 40 種のシンプルな命令セットと 2 つの汎用レジスタと非常にコンパクトな構成をしている．

KUE-CHIP2 をMicroBlazeと通信可能にするためにはKUE-CHIP2 の内部メモリに外部から書き込み，読み出しができるようにし，図 21の状態遷移図に対応してKUE-CHIP2 内部からのメモリアクセスと競合しないように配慮する必要がある．このような条件を満たすためにKUE-CHIP2 の外部とのインタフェース，内部構造の修正を行った．図 22，表 10 にFSL接続対応のKUE-CHIP2 の入出力を,図 23に内部ブロック構造をそれぞれ示す．KUE-CHIP2 の入出力はFSLでの制御に必要な最低限のポートに抑えている．外部からメモリへのアクセスを行うポートの他に図 21で示した状態遷移に応じて外部からのメモリアクセスの有効無効を制御するState信号や格納されたプログラムの全実行を指示するSS信号，実行終了を示すOP信号で構成される．全実行を行うSS信号に対する扱いとしてはKUE-CHIP2 内部に同期を行うモジュールによってSSのアクティブ状態がパルス状である場合やある程度連続している場合でも同様に扱い，プログラムの終了まで継続的に処理を続けるように設計を行っている．

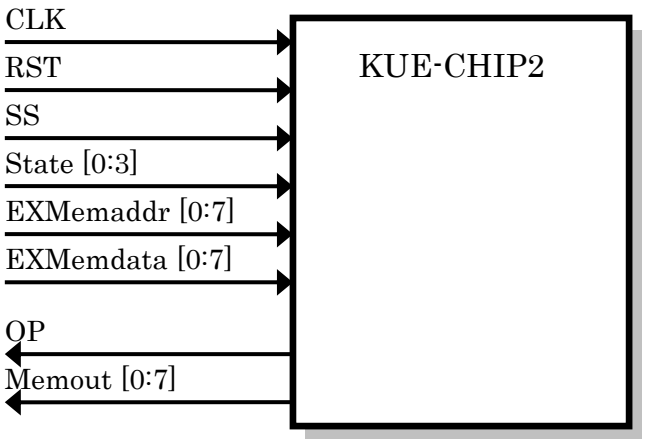

図 22　FSL 対応の KUE-CHIP2 の入出力

表 10　KUE-CHIP2 の入出力リスト

ポート名	I/O	ビット幅	内容
CLK	in	1	システムクロック
RST	in	1	負論理リセット信号
SS	in	1	プロセッサの処理スタート信号
State	in	4	待機，書き込み，実行，読み出し状態の通知
EXMemaddr	in	8	内部メモリ参照用アドレス
EXMemdata	in	8	内部メモリ書き込みデータ
OP	out	1	プロセッサ処理の終了通知
Memout	out	8	内部メモリデータ出力

図 23に示している内部構造の変更についてはBlockRAMを制御するモジュールであるIMEMのみの変更で対応可能である．KUE-CHIP外部から入力されるStateの状態でメモリが内部あるいは外部どちらからのアクセスに許可を与えるかを制御している．Stateが図21で示したRead，Writeの状態の時に外部からのメモリアドレスと書き込みデータを許可し，それ以外の状態で内部のメモリアドレスレジスタ(MAR)と内部バスのデータ入力をアクセスできるように制御することでMicroBlazeからFSLを介したメモリデータの通信とスレーブプロセッサのプログラム実行を一連の動作で実行が可能となった．

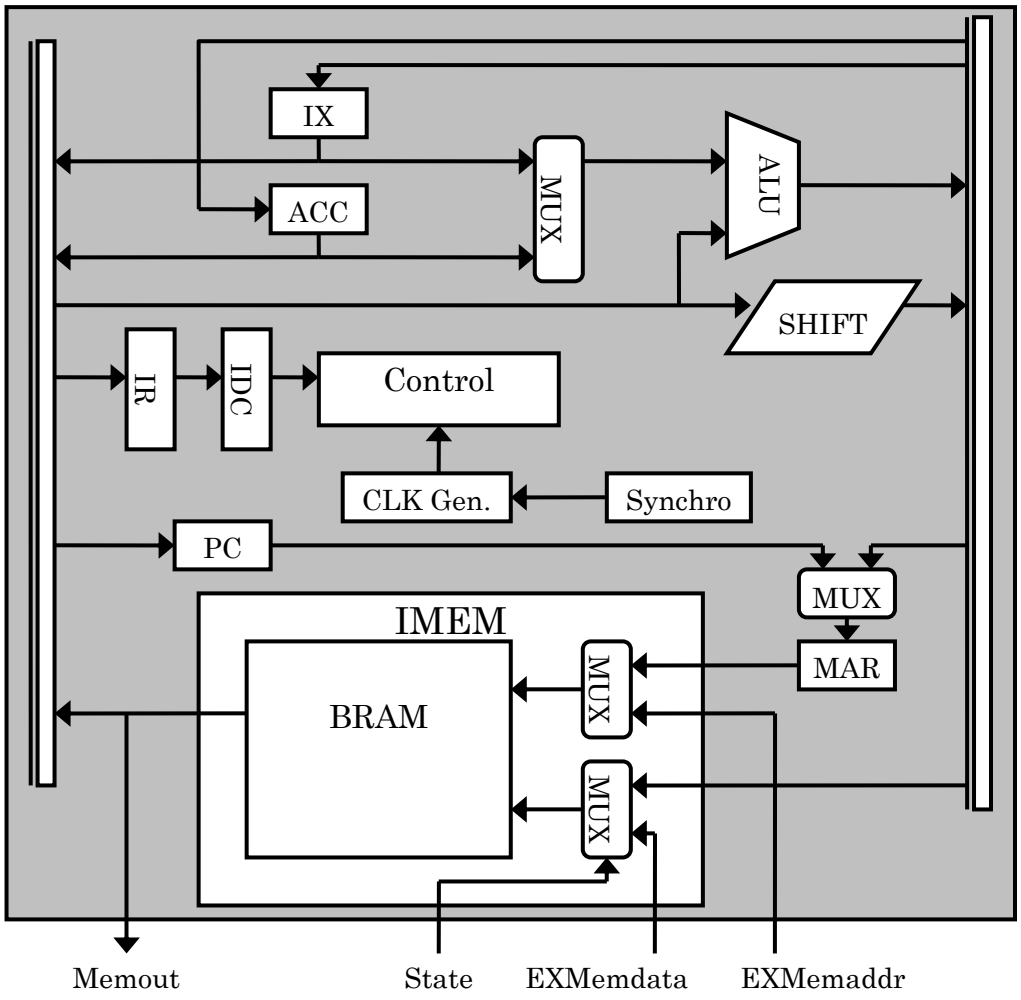

図 23　FSL 対応 KUE-CHIP2 の内部構造

4.3.2　実装結果と動作検証

4.3.1 で述べた KUE-CHIP2 の FPGA への実装を行った．対象デバイスは Virtex4 XC4VLX60 である．その結果 KUE-CHIP2 の使用スライス数は 429，最高動作周波数は 70.005MHz となった. 使用する FPGA ボードが供給するリファレンスシステムクロック周波数は 100MHz であり KUE-CHIP2 にそのままシステムクロックを供給しても動作が保証されない. そこでシステムクロック周波数を半分の 50MHz に下げることで安定したプロセッサの動作を確認することができた.

図 24に動作検証におけるマスタ(MicroBlaze)のプログラム例を示す．図はスレーブにバブルソートを処理させる例である．マスタではスレーブに送信するプログラムとデータを配列として格納しておき，FSL送信マクロであるputfsl()によってメモリ内容を順次送信する．あらかじめ決められた回数のデータ送信が終わると次はFSL受信マクロのgetfsl()でスレーブ処理後のメモリ内容を受け取る.

```
#include "stdio.h"
int main(){
          int i;
          unsigned int out;
//Bubule Sort
char INPUTDATA[64]={
  0x6c,0x2a,0xaa,0x1,0x7c,0x2b,0xc9,0x20,0x67,0x30,0xf7,0x31,0x3f,0x19,0x74,0x2c,
  0x67,0x31,0x77,0x30,0x64,0x2c,0x77,0x31,0x2f,0xba,0x1,0xfc,0x2b,0x31,0x8,0x35,
  0x29,0x6c,0x2b,0xaa,0x01,0x7c,0x2b,0x31,0x06,0x0f,0x08,0,0,0,0,0,
  0x10,0xff,0x40,0x80,0xc0,0xd8,0x7f,0xcd,0,0,0,0,0,0,0,0
};

  for(i=0;i<64;i++)      putfsl(INPUTDATA[i],0);
  for(i=0;i<64;i++)      getfsl(out,0);
  return(0);
}
```

図 24　マスタの C プログラム記述例

表 11に動作検証に用いたサンプルプログラムを示す．サンプルプログラムの動作検証についてはすべて 0 番地から 63 番地のKUE-CHIP2 内部メモリに書き込みを行い，プログラム実行終了後に同領域のメモリ内容をマスタであるMicroBlazeに送信する．以上の処理にかかるクロックサイクル数を 100MHzのシステムクロック周波数を基準に求め参考として示している．

表 11　プログラムの動作検証

プログラム名	サンプルデータの詳細	所要サイクル数
1 から N までの和	10 まで	1216
多倍長の加算	4 バイト	1126
ユークリッドの互除法	60 と 40(hex)	1108
バブルソート	データ数 8	2117

4.4　並列処理に向けたKUE-CHIP2 の 32 ビット拡張

4.4.1　KUE-CHIP2 の 32 ビット拡張

一般的に並列処理は通信時間を相対的に小さくできるような負荷の高い演算処理に対してその効果を発揮することができる．メモリ領域が小さく，なおかつデータのビット幅が 8 ビットである KUE-CHIP2 では高負荷の処理を行うのに適さない．そこで KUE-CHIP2 のデータ幅を 32 ビットに拡張を行うことにした．KUE-CHIP2 はメモリが小さいこともありメモリをほとんど使わない 1 から N までの和などが並列処理のベンチマークの一つにふさわしいがデータ幅が 8 ビットの場合加算できる値域はせいぜい数十程度あり，それを複数台で分割してもまったく効果が得られないことが自明である．32 ビットであれば数万まで

の和を求めることができ，尚且つ他の問題に適用可能と考えられる．

図 25，表 12に 32 ビット拡張したKUECHIP2 の入出力を示す．拡張したKUE-CHIP2 は命令セット自体まったく変更しておらず，データのみを 32 ビットで使えるように設計している．そのため，入出力インタフェースもメモリデータの入出力ポートのビット幅が 8 ビットから 32 ビットに変更しただけでその他のポートに関しては修正を行っていない．

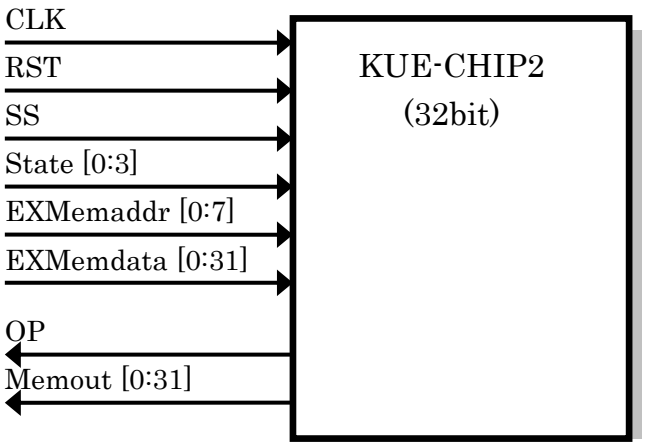

図 25　32 ビット拡張した KUE-CHIP2 の入出力

表 12　32 ビット拡張した KUE-CHIP2 の入出力リスト

ポート名	I/O	ビット幅	内容
CLK	in	1	システムクロック
RST	in	1	負論理リセット信号
SS	in	1	プロセッサの処理スタート信号
State	in	4	待機，書き込み，実行，読み出し状態の通知
EXMemaddr	in	8	内部メモリ参照用アドレス
EXMemdata	in	32	内部メモリ書き込みデータ
OP	out	1	プロセッサ処理の終了通知
Memout	out	32	内部メモリデータ出力

図 26に拡張KUE-CHIP2 の内部構造を示す．入出力インタフェースの変更が少ない半面で内部構造の変更はやや煩雑なものとなった．主な原因は命令，アドレス(256 ワード)が 8 ビット，データが 32 ビットと同一データパス上をビット幅の異なる信号が介在するためである．とくに入力バス，出力バスは通過する信号が非常に多いため，命令や実行フェーズに応じてデータの制御が難しく，設計に時間を要した．汎用レジスタやALU，SHIFTなどは単純なビット拡張を行うことで対応でき，IR，IDC，CONTROLのような制御系データパスやPC，MARなどのアドレスデータパスについては変更していない．

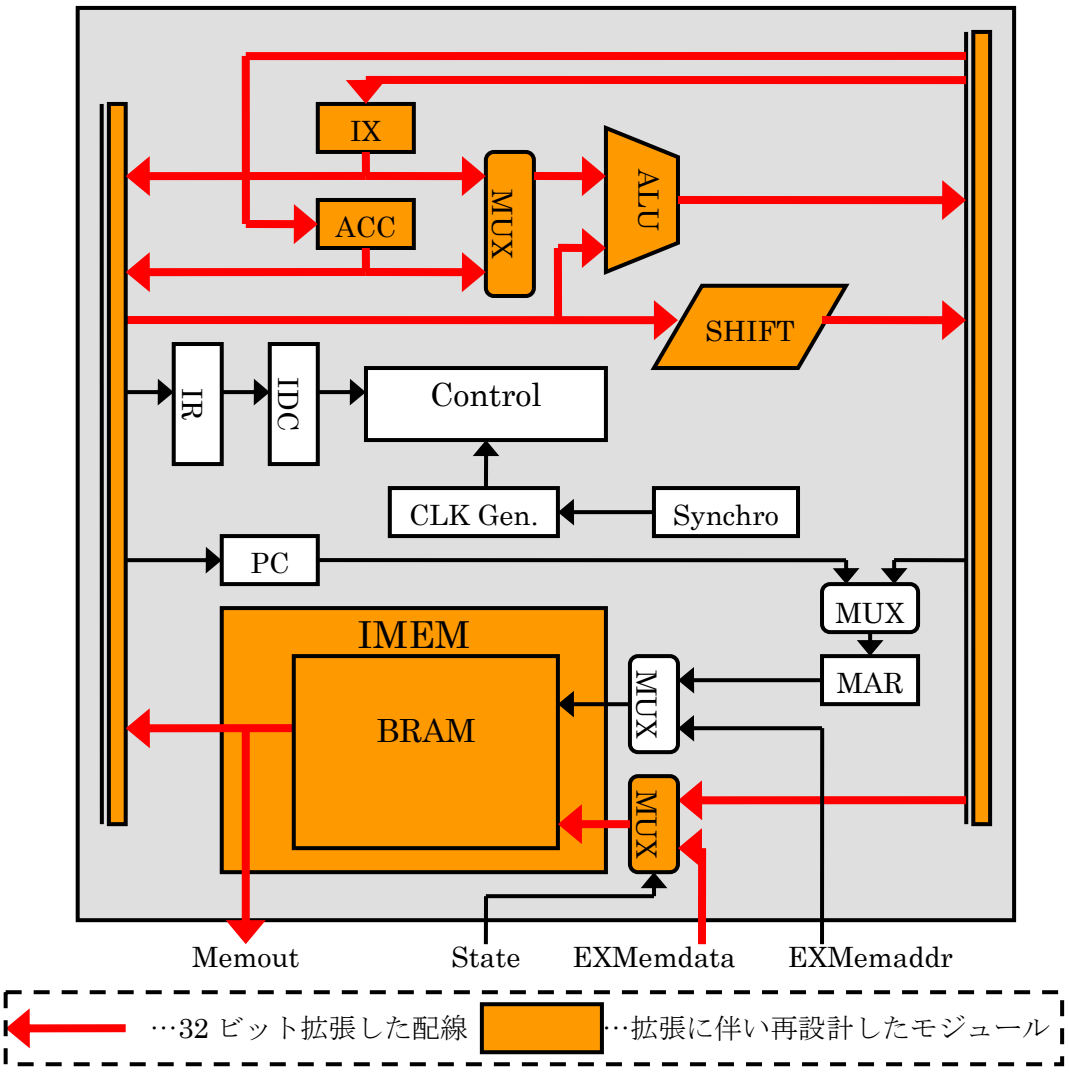

図 26　拡張 KUE-CHIP2 の内部構造

4.4.2　実装結果と動作検証

4.4.1 で述べた 32 ビットに拡張したKUE-CHIP2 をFPGAに実装した．対象デバイスは同じくVirtex4 XC4VLX60 でその結果拡張KUE-CHIP2 の使用スライス数は 741，最高動作周波数は 57.320MHzとなった．データのビット拡張による回路増加とともにクリティカルパスが増大していることがわかる．使用FPGAボードの供給リファレンスシステムクロック周波数は 100MHzであることから拡張前のKUE-CHIP2 と同様半分の 50MHzに下げることで安定したプロセッサの動作を確認することができた．表 13に動作検証に用いたサンプルプログラムを示す．

表 13　拡張版 KUE-CHIP2 のプログラム動作検証

プログラム名	サンプルデータの詳細	所要サイクル数
1 から N までの和	10 まで	1216
1 から N までの和	20000 まで	221106
ユークリッドの互除法	60 と 40(hex)	1108
バブルソート	データ数 8	2117

4.5　シングルサイクルプロセッサSOARとMicroBlazeの接続

KUE-CHIP2 に替わるスレーブプロセッサコアとしてSOARプロセッサの接続を試みた．SOARは本研究室で現在研究されているハード/ソフトコラーニングシステムの構築過程で設計されたシングルサイクルプロセッサであり，MIPSアーキテクチャをベースとした全部で 25 の命令を持つ．従来の命令長・データ長が共に 16 ビットであるが，本研究用にデータ長を 32 ビットに拡張した．図 27，表 14にSOARの入出力を示す．

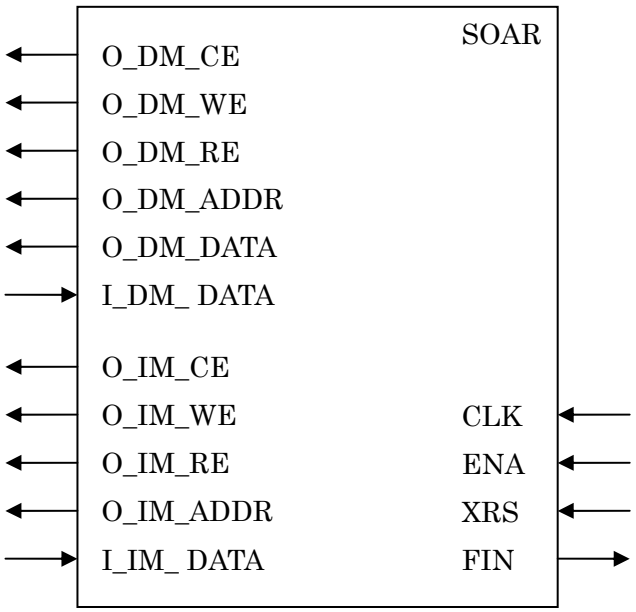

図 27　SOAR の入出力

表 14　SOAR の入出力リスト

入出力	ポート名	ビット幅	意味
output	O_DM_CE	1	データメモリのチップイネーブル
output	O_DM_WE	1	データメモリの書き込みイネーブル
output	O_DM_RE	1	データメモリの読み出しイネーブル
output	O_DM_ADDR	11	データメモリのアドレス
output	O_DM_DATA	32	データメモリへの書き込みデータ
input	I_DM_DATA	32	データメモリからの読み出しデータ
output	O_IM_CE	1	命令メモリのチップイネーブル
output	O_IM_WE	1	命令メモリの書き込みイネーブル
output	O_IM_RE	1	命令メモリの読み出しイネーブル
output	O_IM_ADDR	11	命令メモリのアドレス
input	I_IM_DATA	16	命令メモリへの読み込み命令
output	FIN	1	終了信号
input	CLK	1	クロック
input	XRS	1	リセット
input	ENA	1	イネーブル

SOARはKUE-CHIP2 と異なりデータメモリと命令メモリを別々に用意する必要がある．図 28にSOARにおけるプロセッサ間通信の状態遷移図を示す．メモリを物理的に 2 つに分けているため，それぞれのメモリへの書き込みフェーズが必要である．一方でMicroBlaze への送信フェーズではデータメモリからのみ実行結果を受け取ればよい．命令メモリの読み出しはデバッグ時のみ用いる．

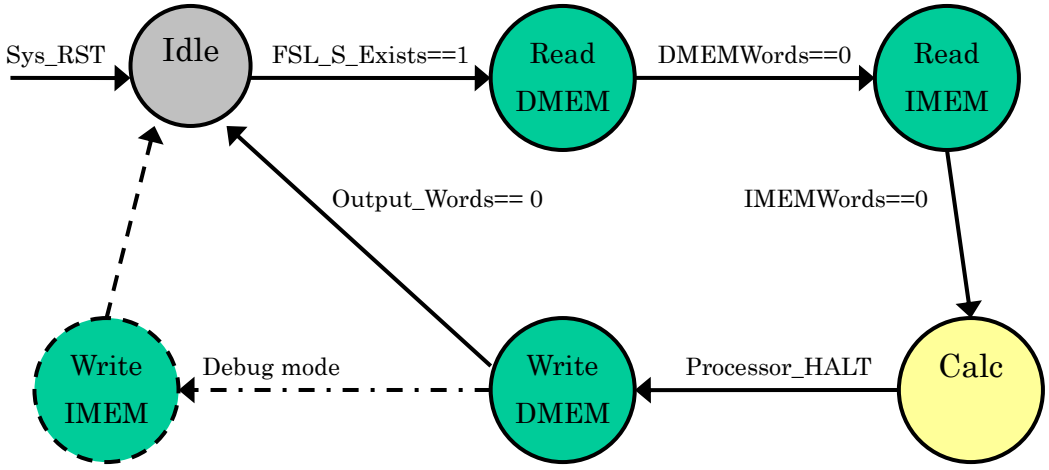

図 28　SOAR におけるプロセッサ通信の状態遷移

4.6　並列処理環境の実現と評価

4.6.1　拡張 KUE-CHIP2 による並列処理検証環境の実現

4.4 で述べた 32 ビット拡張したKUE-CHIP2 をスレーブプロセッサとしたFPGA上での並列処理環境を構築した．図 29にMicroBlazeを用いたKUE-CHIP2 並列処理環境を示す.

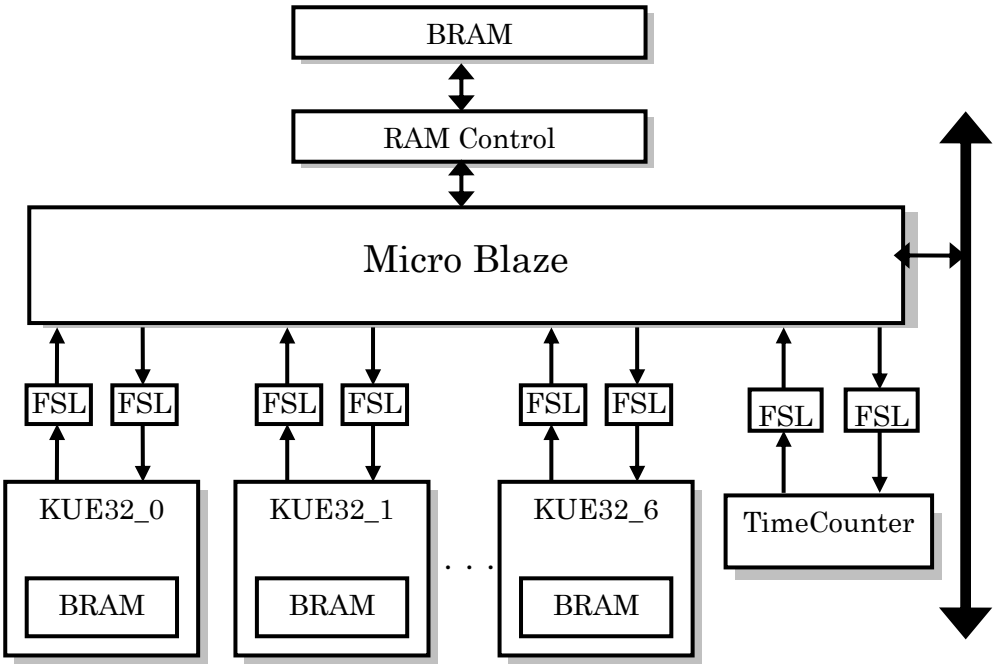

図 29　MicroBlaze を用いた KUE-CHIP2 並列処理環境

Microblaze は FSL インタフェースによるハードウェア接続が最大 8 つまで可能である．本研究においては所要クロックサイクル数の測定モジュールである TimeCounter をひとつ接続するため，各スレーブプロセッサに FSL を 1 組ずつ割り当てると最大 7 台の KUE-CHIP2 が接続できる．XPS による操作としては GUI 上でインスタンス化された KUE-CHIP2 をマウス操作のみで接続ができ,非常に容易である.マスタとなる MicroBlaze は各プロセッサに割り当てる命令とデータを保持しスケジューリングによって各プロセッサに動的に処理を割り当てることができる．なお、SOAR プロセッサの並列処理環境についても同様の構成で実現している.

4.6.2　N までの総和を用いた並列処理

図 29で示した並列処理環境の評価にKUE-CHIP2 教育用ボードリファレンスマニュアルに記載されているサンプルプログラムの 1 からNまでの総和をベンチマークに用いた.本研究で用いるKUE-CHIP2 はメモリ領域が小さく，上記プログラムは少ないメモリ領域で計算時間を稼ぐことができるため採用した．図 30に逐次処理によるNまでの総和のフローチャートを示す．処理の説明はKUE-CHIP2 のアーキテクチャを元にしている.

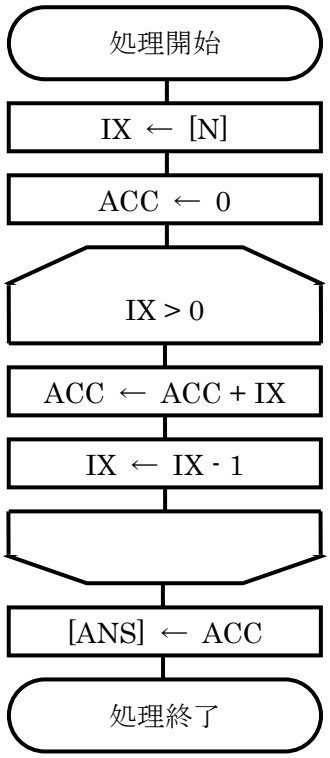

図 30　逐次処理による N までの総和のフローチャート

評価対象となる処理の流れとしてインデックスレジスタIXに総和の最大値Nを代入する．IXの値をもうひとつの汎用レジスタであるアキュムレータACCに加算し，IXの値をデクリメントしながらIXが 0 になるまで加算を繰り返す．最終的に総和がACCに代入されるので解を格納するメモリ領域[ANS]にデータをストアすることで処理は終了する．この処理をM個のKUE-CHIP2 で並列処理を行い，所要クロックサイクル数の測定を行う．図 31に並列化したNまでの総和のフローチャートを示す．並列処理の手法としては各プロセッサに与えるIXの初期値を変更し，ループ中で行う減算の値を従来の 1 から分割数であるMに変更することで各プロセッサが加算するデータを分担し，それぞれが部分和を得る．

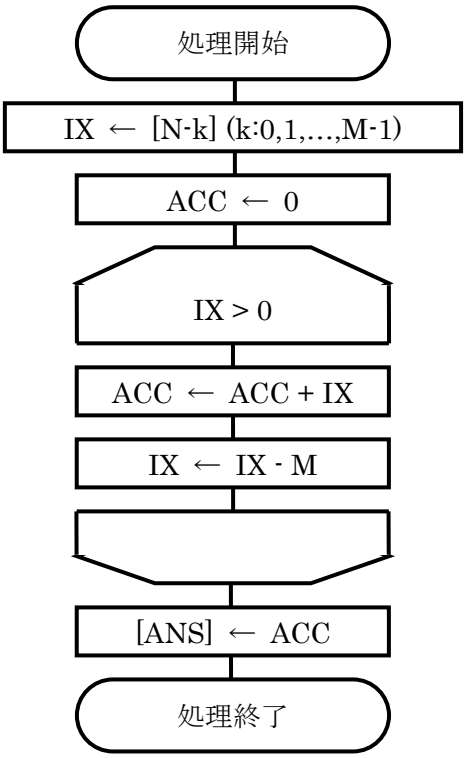

図 31　並列化によるスレーブプロセッサのフローチャート

図 32にマスタであるMicroBlazeによるスレーブへの処理割り当てのイメージを示す.

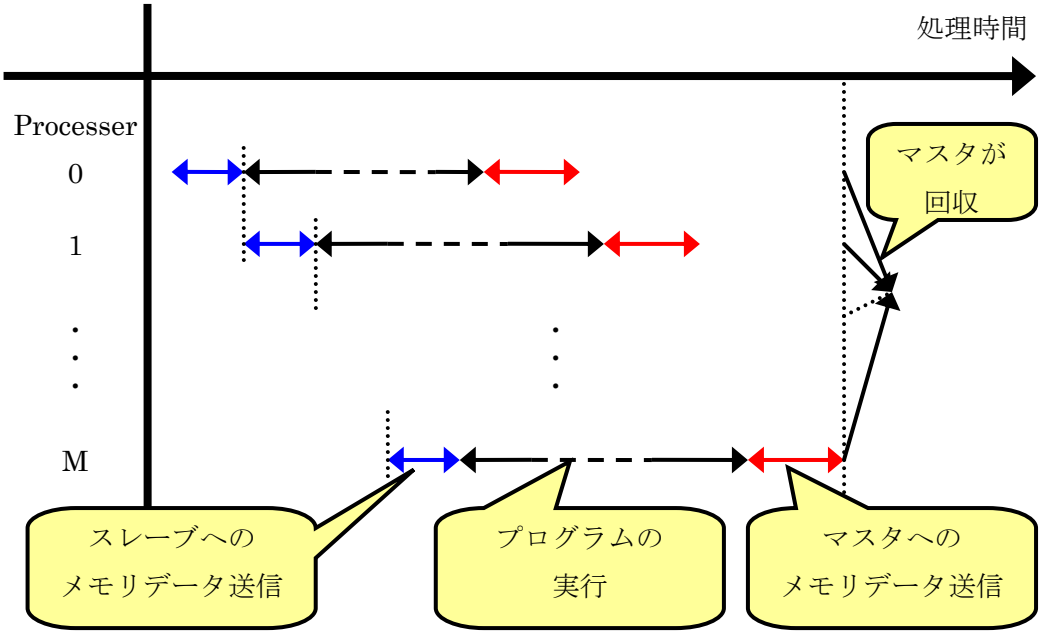

図 32　MicroBlaze によるスレーブへの処理割り当て

MicroBlaze のプログラムではまずプロセッサ 0 に処理内容を示す命令コードと処理をする対象のデータを FSL の通信マクロである putfsl()で送信する．規定数の送信が終わると次はプロセッサ 1 に命令とデータを送り始める．このときプロセッサ 0 は命令とデータを受け取って数サイクル以内に自立的に処理を開始する．すべてのスレーブプロセッサに処理を割り当てた後に今度は getfsl()を用いてプロセッサ 0 から順に結果を受け取る．getfsl()はスレーブプロセッサが処理を完了するまで受け取りを待つブロッキング方式なので同期をとって受信することが容易である．すべてのスレーブプロセッサの結果を回収後に MicroBlaze で各結果の結合を行い最終的な解を得る．

4.5.3 並列処理環境の評価結果

(1) KUE-CHIP2 の結果

表 15にNまでの総和のスレーブ数と処理時間，図 33に並列処理による速度向上のグラフをそれぞれ示す．並列化における命令とデータはあらかじめマスタプログラム内に配列として格納されているものとし，スレーブへのプログラムの送信開始からクロックカウントを開始し，マスタがすべてのスレーブから部分和を回収し，部分和を合計した時点までをクロックカウントの計測時間とする．表 15はスレーブのみが処理を行った場合を指し，計算量の指標になるNの値と実行プロセッサ数毎の実行結果を示した． 図 33は表 15内の速度向上比をNの値ごとにグラフ化した．なお，通信は必要最低限のワード数(送信 13 ワード，受信 1 ワード)に最適化した．

表 15 KUE-CHIP2 によるNまでの総和のクロックサイクル数

スレーブ数 N の値	1	2	3	4	5	6	7
5000	55167 (1)	27815 (1.98)	18789 (2.93)	14361 (3.84)	11759 (4.69)	10070 (5.47)	8909 (6.19)
20000	220167 (1)	110315 (1.99)	73789 (2.98)	55611 (3.95)	44759 (4.91)	37570 (5.86)	32482 (6.77)
40000	440167 (1)	220315 (1.99)	147126 (2.99)	110611 (3.97)	88759 (4.95)	74233 (5.92)	63909 (6.88)

括弧内は速度向上比

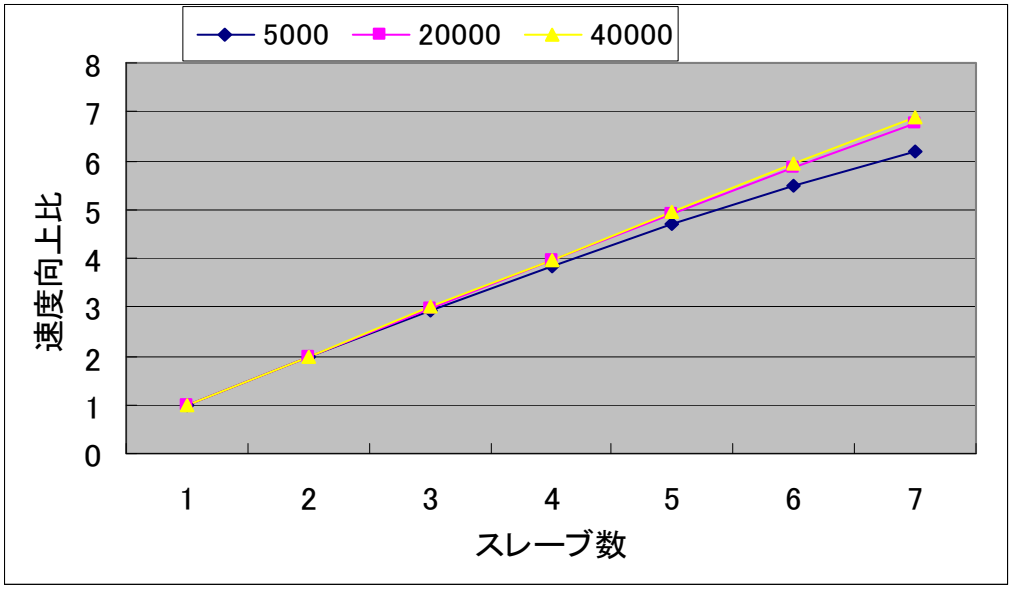

図 33　KUE-CHIP2 による並列処理の速度向上

(2)　SOAR の結果

表 16にNまでの総和のスレーブ数と処理時間，図 34に並列処理による速度向上のグラフをそれぞれ示す．アルゴリズムはKUE-CHIP2 と同様にし，SOAR命令セットでアセンブリコードを作成してMicroBlaze上のCプログラムに配列として格納した．SOARの通信ワード数は(送信 7 ワード，受信 1 ワード)に最適化した．

表 16　SOAR によるNまでの総和のクロックサイクル数

スレーブ数 N の値	1	2	3	4	5	6	7
5000	15091 (1)	7680 (1.37)	5297 (1.99)	4146 (2.55)	3493 (3.03)	3089 (3.42)	2829 (3.74)
20000	60091 (1)	30180 (1.99)	20297 (2.96)	15396 (3.90)	12493 (4.80)	10589 (5.67)	9258 (6.49)
40000	120091 (1)	60180 (1.99)	40298 (2.98)	30396 (3.95)	24493 (4.90)	20588 (5.83)	17829 (6.73)

括弧内は速度向上比

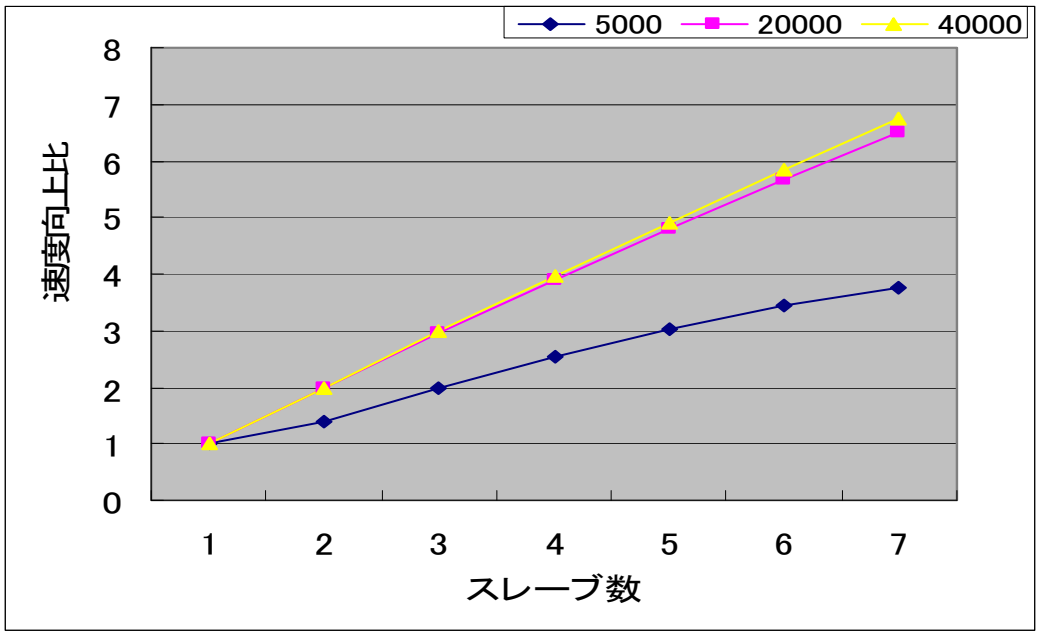

図 34　SOAR による速度向上

(3)　MicroBlaze との比較

表 17にMicroBlazeと組み合わせた処理時間を示す．表ではMicroBlazeのみで実行した場合とKUE-CHIP2 およびSOAR7 台とMicroBlazeの計 8 台で均等分割して実行した場合の結果を示している．

表 17　マスタと組み合わせた場合のクロックサイクル数

N の値	MicroBlaze のみ	KUE-CHIP2×7 + MicroBlaze	SOAR×7 + MicroBlaze
5000	30003	7931 (3.78)	3853 (8.62)
20000	120003	28556 (4.20)	13228 (8.78)
40000	280003	56056 (4.99)	25728 (11.5)

括弧内は MicroBlaze のみとの速度向上比

(4) 考察

スレーブ数 1 のときの結果を見ると N の値が 5000，20000，40000 と増えるのに比例した所要クロックサイクル数が観測される．各々の FSL を用いた通信回数が同じことと計測結果の下 3 桁の値がいずれも同じことから 1 プロセッサへの FSL での送受信の所要サイクル数がわかる．このことに着目すると，N がいずれの値の場合でもスレーブが増えるに連れて所要クロックサイクル数が減少している．しかし，計算量が相対的に少ない 5000 までの和の場合はプロセッサ数の増加とともに FSL での通信時間が多くなり，速度向上比が他の場合と比べて若干低くなっていることが観測される．一方，最も計算量の多い 40000 までの和では 7 個のプロセッサ実行時でもほぼプロセッサ数分の速度向上が得られた．これ

らのことから，本研究で実装した並列処理環境は，ひとつのスレーブプロセッサ処理量を基準としたとき，FSL を用いた通信データ量の効率化や並列処理対象の計算量しだいで十分な並列化効果を得ることが可能であることが示された．

また，MicroBlazeのみの処理時間とスレーブプロセッサ 1 つの処理時間を比較するとKUE-CHIP2 では約 1.8 倍，SOARでは約 0.5 倍のスループットの差があることがわかる．これはKUE-CHIP2 がマルチサイクルアーキテクチャであるのに対しMicroBlazeが 3 段パイプラインアーキテクチャと処理効率に差があることを示している．大きな計算量においては並列処理環境が十分に作用しており，KUE-CHIP2 の場合，スレーブ 2 つによってFSLの通信を含めても，MicroBlaze以上の性能を出すことができた．一方，SOARがシングルサイクルアーキテクチャのため，所要クロックサイクル数が少ない．40000 までの和についてはスレーブ 7 つとMicroBlazeでは最大 2.3 倍の速度向上を得ている．KUE-CHIP2 は本来教育目的に開発されたプロセッサアーキテクチャであり，スループットを追及したものではない．しかしながら本研究で構築した並列処理環境を用いることで，MicroBlazeを超えるスループットが出せることを示した．これは今後スレーブプロセッサのアーキテクチャを改良することで，さらに並列処理の恩恵を得ることが十分に考えられる．表 17ではスレーブへの処理割り当ての直後にMicroBlaze本体も処理の一部を担当することで 8 個のプロセッサで処理ができる．KUE-CHIP2 ではマスタとスレーブ両方組み合わせて処理を行うことで，よりいっそうの速度向上を得ることができる．一方，SOARではMicroBlazeとの均等処理割り当てで性能がでなかった．この実験ではマスタとスレーブは同じ動作周波数で動作させているため，MicroBlazeのスループットが低下していることがわかる．MicroBlazeを含めて性能を向上させたい場合は，マスタとスレーブでそれぞれ実現できる動作周波数に分周することで可能と推測できる．

5. FPGAマルチコア環境の機能拡張と検討

5.1 スレーブプロセッサ標準接続インタフェース

4 章の並列処理環境の構築においてはKUE-CHIP2，SOARを個別に接続インタフェースを設計し，実装と検証を行った．KUE-CHIP2 とSOARではメモリアクセスの方法が異なっているため，実質は設計をやり直していた．そこで，本研究で構築した並列処理環境に標準となるインタフェースを策定することにし，ハード・ソフトコラーニングシステムで用いられているプロセッサデバッガ[18]に着目した．プロセッサデバッガはFPGA実装時に外部からの操作でプロセッサの動的検証ができるツールで，外部から命令，データメモリの読み書きが出来る点で本研究のスレーブプロセッサの接続インタフェースへの組み込みに適している．図 35表 18にプロセッサデバッガに基づいた標準のプロセッサインタフェースを示す．入出力の構成はハーバードアーキテクチャを想定しており，データメモリ，命令メモリにアクセスするポートとプロセッサ実行を制御するENA，プロセッサ実行終了を示すFINと非常にシンプルな構成となっている．

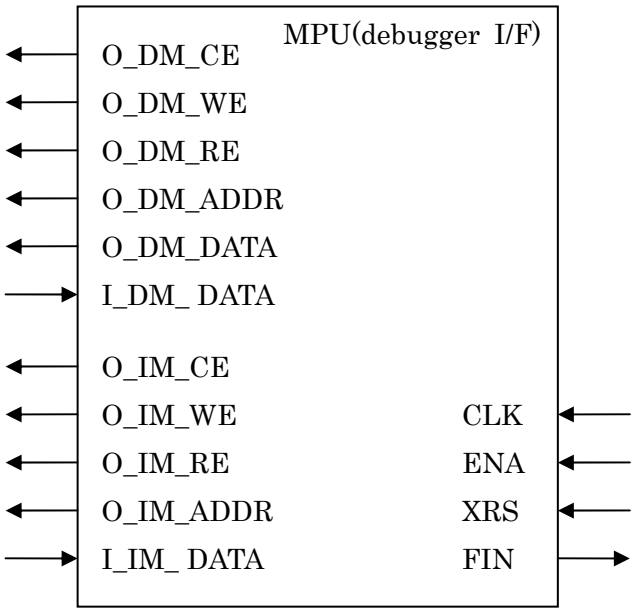

図 35　プロセッサデバッガに基づくプロセッサインタフェース

表 18　プロセッサデバッガに基づくプロセッサ入出力

入出力	ポート名	ビット幅	意味
output	O_DM_CE	1	データメモリのチップイネーブル
output	O_DM_WE	1	データメモリの書き込みイネーブル
output	O_DM_RE	1	データメモリの読み出しイネーブル
output	O_DM_ADDR	11	データメモリのアドレス
output	O_DM_DATA	32(16)	データメモリへの書き込みデータ
input	I_DM_DATA	32(16)	データメモリからの読み出しデータ
output	O_IM_CE	1	命令メモリのチップイネーブル
output	O_IM_WE	1	命令メモリの書き込みイネーブル
output	O_IM_RE	1	命令メモリの読み出しイネーブル
output	O_IM_ADDR	任意	命令メモリのアドレス
input	I_IM_DATA	16(32)	命令メモリへの読み込み命令
output	FIN	1	終了信号
input	CLK	1	クロック
input	XRS	1	リセット
input	ENA	1	イネーブル

表 19に策定した標準インタフェースに準拠したプロセッサの実装結果を示す．本研究で用いたSOARはプロセッサデバッガに対応しているため，16 ビット版，32 ビット版それぞれ図 35の入出力で並列処理環境と接続ができた．また，プロセッサデバッガはシングルサイクル以外のアーキテクチャにも対応しており，マルチサイクルアーキテクチャで設計されたMONIプロセッサの動作確認ができた．プロセッサデバッガに準拠しているため，プロセッサデバッガを用いて設計したプロセッサであれば基本的に接続が可能であり，パイプラインやスーパスカラプロセッサの接続が期待できる．

表 19　標準インタフェースによるプロセッサの実装結果

プロセッサ名	bit 幅	アーキテクチャ	回路規模(slice)	最高動作周波数(MHz)
SOAR	16	シングルサイクル	775	34.841
SOAR	32	シングルサイクル	1000	33.325
MONI	16	マルチサイクル	1032	76.974

5.2　プロセッサ間データ通信の動的可変量化

4 章で述べた並列処理環境は元々3 章で述べたハード・ソフト協調設計システムをベースにしているため，マスタ・スレーブ間のデータ転送量はスレーブのHDL記述にパラメータとして指定する必要があり，コンフィギュレーション時に設定するなど，静的なデータ転送量の指定しか出来なかった．並列処理環境の構築目的はSW資源の有効活用とアクセラレーションのため，プロセッサ資源を再利用する上で動的に命令やデータをやり取りするが，転送量が固定されるとシステムの効率が非常に悪くなる．これは 1 つのプロセッサに大規模なプログラムと小規模のプログラムをそれぞれ実行する必要があるとき，静的指定の場合，転送量を大きなプログラムに合わせる必要があり，その結果小さなプログラムの転送に関して余分クロックサイクル数を消費することになり，スループットが悪くなる．そこでMicroBlaze側のソフトウェア側から自由にデータ転送量を指定できるようにすることを考え，動作確認を行った．図 36に可変量データ転送を用いた通信状態遷移を示す．

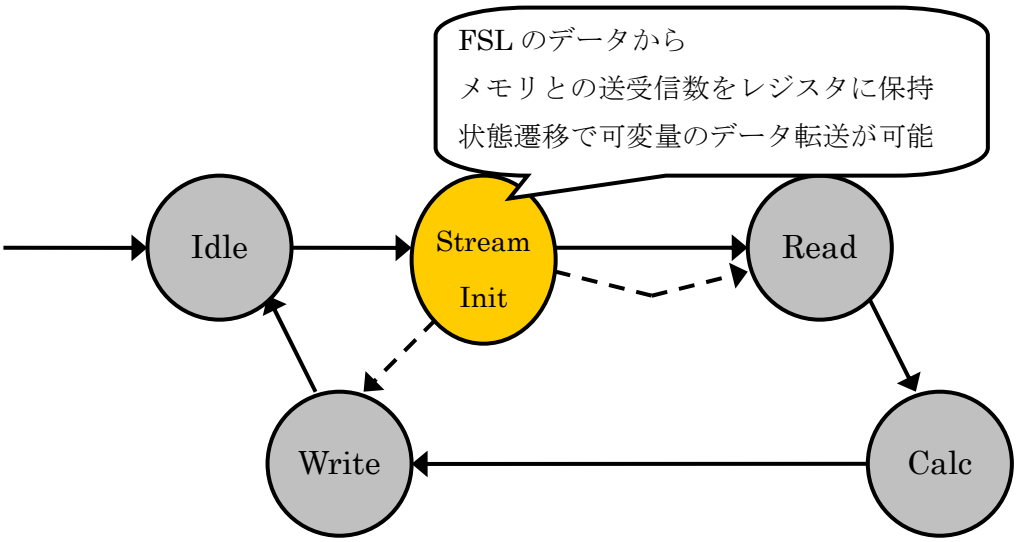

図 36　可変量データ転送を用いた通信状態遷移

図 36は図 20のコントロール部の状態遷移を示しており，図では簡略化のためデータメモリ，命令メモリの区別はしていない．従来の状態遷移に対してデータ量情報を受け取るフェーズ(StreamInit)を追加した構造になっている．このフェーズで受信したデータは状態遷移を管理するモジュールにあるレジスタに保持される．メモリの書き込み，読み出しフェーズはそれぞれ保持されたレジスタを参照し，デクリメントを繰り返しその値が 0 になれば次のフェーズに遷移するように設計することでMicroBlazeとの可変量データ通信を実現することができた．

プロセッサ間の可変量データ通信インタフェースを実現することにより，MicroBlaze上で動作するソフトウェアプログラムは図 37のようになる．処理内容は図 24と同様である

が，大きな違いはputfslとgetfslの回数をMicroBlaze上のプログラム側でそれぞれ指定している点が挙げられる．従来の方法ではHDLによって静的に指定していたため，Cプログラムからでは指定ができず，動的なプログラムの再割り当てに適さない．可変通信量設定を可能にすることで並列処理環境におけるスレーブプロセッサの再利用性の向上が得られる．

```
//Bubule Sort
char INPUTDATA[64]={
  0x6c,0x2a,0xaa,0x1,0x7c,0x2b,0xc9,0x20,0x67,0x30,0xf7,0x31,0x3f,0x19,0x74,0x2c,
  0x67,0x31,0x77,0x30,0x64,0x2c,0x77,0x31,0x2f,0xba,0x1,0xfc,0x2b,0x31,0x8,0x35,
  0x29,0x6c,0x2b,0xaa,0x01,0x7c,0x2b,0x31,0x06,0x0f,0x08,0,0,0,0,0,
  0x10,0xff,0x40,0x80,0xc0,0xd8,0x7f,0xcd,0,0,0,0,0,0,0,0
};
int READ=56;
int WRITE=8;

  putfsl(READ,0);          //READ でのデータ送信量の指定
  putfsl(WRITE,0);         //WRITE でのデータ受信量の指定
  for(i=0;i<READ;i++)      putfsl(INPUTDATA[i],0);
  for(i=0;i<WRITE;i++)      getfsl(out,0);
  return(0);
}
```

図 37　MicroBlaze 上での可変量通信

本章では，32 ビット拡張版KUE-CHIP2 のインタフェースに対して可変量通信のカスタマイズを行い，従来の固定通信量の場合と比較を行った．実験の内容は表 13で示した 1 からNまでの和(20000 までの和は除く)，ユークリッド互除法，バブルソートを 1 つのスレーブプロセッサで順次行いすべて終了するまでの所要クロックサイクル数を測定して比較した．固定量通信と可変量通信の比較結果を表 20に示す．各プログラムの実行結果は 16 進数表示 30 番地を先頭として格納されるようにコーディングしており，Writeフェーズにおけるメモリのアドレスは 30 番地から開始するようにしている．

表 20　固定量通信と可変量通信の比較

データ転送量(ワード)	固定量通信(従来方法)	可変量通信
1 から N までの和	64 (R:56 , W:8)	17 (SI:2 , R:14 , W:1)
ユークリッド互除法	64 (R:56 , W:8)	21 (SI:2 , R:18 , W:1)
バブルソート	64 (R:56 , W:8)	66 (SI:2 , R:56 , W:8)
データ転送量の合計	192	104
所要クロックサイクル数	2452	2256

(R：READ　W：WRITE　SI：StreamInit の転送量)

従来の設計を基にした固定量の通信方法では，1 つのプロセッサに複数種のプログラムを

動的に割り当てて実行する場合に，最も転送量の多いプログラムに基準を合わせなければ正常な動作を得られない．そのため表ではバブルソートのプログラムとデータの送信，及び実行後のデータ回収量が最も多く，データ転送量が相対的に少ない 1 から N までの和，ユークリッド互除法もバブルソートの転送量に合わせるしかなく，必要なデータ送信の後に無効データのパディングをして対応するなど，合計の所要クロックサイクル数が 2452 に至った．

一方で今回提案した可変量通信インタフェースによるデータ転送量は，各プログラムの実行に対して図 36のStreamInitに対応する 2 ワード分の転送量情報を必要とするが，転送量が各プログラムの必要数に最適化されるため，固定量通信と比べても半分程度に削減され，所要クロックサイクル数も 2256 まで削減できていることがわかる．実験ではスループットの改善が 10 パーセント弱であったが，マルチスレッド単位の分割処理を想定した場合，プログラム量や処理負荷によってはさらにスループットに差が現れる可能性が高い．このことからプログラムの動的割り当てにおける可変量通信方式が本研究の並列処理環境において一定の有効性が示された．

表 20の実験の意義は可変量通信の有効性以上に，MicroBlazeからスレーブプロセッサに対して制御情報を送信して管理ができる事の方が大きい．今後はアドレス情報を用いたランダムアクセスやスレーブプロセッサに様々なモードを用意した上でMicroBlaze側から自由に制御するなど，MicroBlazeからより細かなシステム制御ができるようになることが期待できる．

5.3　FSLを用いた高並列度の処理環境の検討

本研究ではXilinx社のソフトマクロCPUのMicroBlazeを用いたシステムを構築し，マスタ・スレーブ間のプロセッサ接続にはFSLを用いた．MicroBlazeに直接接続できるFSLは 8 組であり，1 スレーブにつき 1 組のFSLを割り当てると最大 8 つまでしか接続できないことになる．そこで，より多くのスレーブプロセッサを用いた並列処理を実現するために 1 つのFSLに複数のスレーブプロセッサを接続することを提案し，検討を行った．図 38に 1 組のFSLに 4 つのプロセッサコアを接続する場合のブロック図を示す．

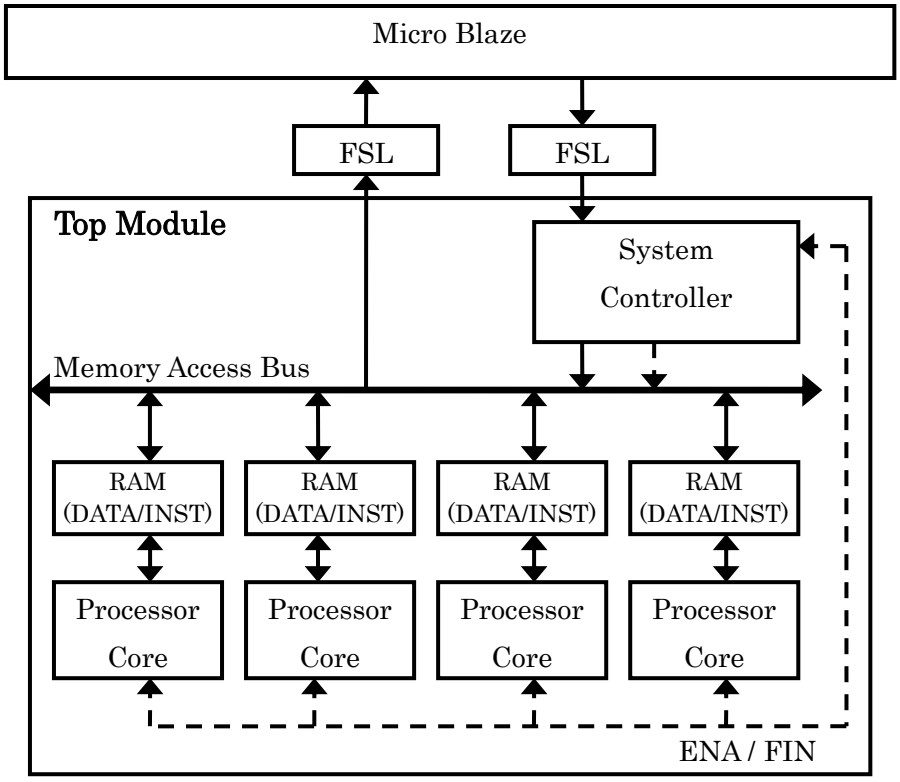

図 38　FSL を用いた高並列処理環境の構造

全体の制御はSystem Controllerモジュールが担当し，メモリアクセスバスの制御や各スレーブプロセッサコアに対するENAやFIN信号の制御を行う．全体制御の方法としては，FSLからのデータ取得回数のカウンティングや状態遷移で行うとするが，5.2 で述べたデータの可変量通信やその応用として，データ駆動方式でMicroBlazeから様々な制御情報を受け取って自由且つ動的に並列度を指定したり，あるいはSIMDのように命令メモリを共有するなど，様々なシステムの発展の可能性が考えられる．図 38の場合では同様の構造をFSLで 8 組用いた場合に理論上は 32 個のスレーブプロセッサで実行できることになり，高並列の処理環境の実現が考えられる．もちろん上記構造は接続できるプロセッサ数の際限はなく，実装するFPGAデバイスの容量に応じて増やすことが出来る．上記提案はまだ検討段階

であるが，理論はこれまで研究で実現した並列処理環境の応用に基づいて考案しているため，提案する高並列度の処理環境は十分に実現が可能であると考えられる．

6. おわりに

本研究では SHA-1 をターゲットアプリケーションとしたハード/ソフト最適分割を探索し，様々な分割パターンの切り分けや FPGA 上での実装による評価を行い，最適なハードウェアとソフトウェアの構成について考察を行った．

また，この過程で用いられたシステム構成からソフトマクロ CPU にオリジナルプロセッサを複数個接続し，ヘテロジーニアスなマルチコア並列処理環境を実現して，スレーブプロセッサ数に応じた速度向上を確認した．このことから本研究で構築した並列処理環境において，FPGA 上での再利用性の高いスレーブプロセッサによるソフトウェア資源の並列実行や動的なタスク割り当てが可能であることを示した．接続したスレーブプロセッサとして，KUE-CHIP2 以外に本研究室の設計資産である RISC アーキテクチャ SOAR, MONI の動作確認など，本研究で構成した並列処理環境に様々なアーキテクチャのプロセッサが接続可能であることも示すことが出来た．

本研究は FPGA による変更容易な超並列構成の実現を目標にしている．今後の研究課題としては，スレーブプロセッサとの接続インタフェースの標準化をこれからも進め，パラメータ調整のみでより多くのプロセッサ接続を可能にすることが挙げられる．また，MicroBlaze の FSL インタフェースは最大 8 つまで使用できるが，一つの FSL に対して複数のプロセッサを接続することが原理的に可能であり，今回使用した FPGA デバイスには，32 ビット拡張 KUE-CHIP2 サイズであれば 64 個は実装が可能である．このようなより多くのプロセッサ接続を実現し，並列度を自由に設定できるようにすることも課題の一つに挙げられる．そしてこのような並列実行環境を構築していく上で並列効果を得ることが出来るアプリケーションプログラムの探索も本研究の大きな目標の一つである．

高性能計算研究室ではハード/ソフトコ・デザインと並列処理を大きな柱とした研究をしており，本研究では両方の融合と高性能，汎用化を追及することを大きな目標として掲げた．研究はまだ発展途上の段階であり，今後より一層発展することを期待したい．

謝辞

本研究の機会を与えてくださり，貴重な助言，ご指導をいただきました山崎勝弘教授，小柳滋教授に深く感謝いたします.

また同じ Micro Blaze を用いた研究に従事している Hoang Anh Tuan 氏，梅原直人氏，並列処理環境の構築において SOAR プロセッサの提供と接続に対する助言をいただいた難波翔一朗氏，数多くの技術的な助言を頂いた中谷嵩之氏，ならびに本研究に関して貴重なご意見をいただきました高性能計算研究室の皆様に深く感謝致します.

参考文献

[1] 古川達久 : FPGA 上でのソフト・マクロ CPU によるハードウェア/ソフトウェア分割手法の研究,立命館大学理工学研究科修士論文,2005.

[2] 梅原直人 : ハード/ソフト最適分割を考慮した AES 暗号システムと JPEG エンコーダの設計と検証,立命館大学理工学部卒業論文,2005.

[3] 的場督永 : ハード/ソフト最適分割を考慮した JPEG エンコーダの協調設計,立命館大学理工学部卒業論文,2005.

[4] 梅原直人,古川達久,的場督永,山崎勝弘,小柳滋 : ソフト・マクロ CPU を用いた回路設計とハードウェア/ソフトウェア最適分割法の検討,情報処理学会関西支部大会,VLIS システム研究会,C-06,2005.

[5] 梅原直人,山崎勝弘 : 設計仕様の解析によるハードウェア/ソフトウェア最適分割手法の検討,情報処理学会関西支部大会,VLSI システム研究会,C-08,2006

[6] 船附誠弘,山崎勝弘,小柳滋 : Handel-C による SHA-1 の設計とハードウェア/ソフトウェア最適分割の検討,FIT2006,C-001,2006.

[7] Xilinx : http://www.xilinx.com

[8] Avnet Japan : http://www.jp.avnet.com

[9] Avnet Japan : Xilinx Micro Blaze Development Workshop テキスト,2006.

[10]FIPS180-2 SECURE HASH STANDARD : http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf.

[11]船附誠弘 : 教育用マイクロプロセッサの設計と FPGA 上での検証,立命館大学理工学部卒業論文,2005.

[12]船附誠弘,中谷嵩之,山崎勝弘,小柳滋 : 教育用マイクロプロセッサの設計と FPGA ボード上での検証,FIT2005,C-001,2005.

[13]神原弘之,越智弘之,澤田宏,浜口清治,岡田和久,上嶋明,安浦寛人 : KUE-CHIP2 設計ドキュメント version1.10,京都高度技術研究所,1993.

[14]神原弘之,越智弘之,澤田宏,浜口清治,岡田和久,上嶋明,安浦寛人 : KUE-CHIP2 教育用ボードリファレンスマニュアル version1.11,京都高度技術研究所,1993.

[15]山崎勝弘 : 電子情報デザイン実験Ⅲ「教育用ボードコンピュータ」テキスト,立命館大学理工学部電子情報デザイン学科,2006.

[16]難波翔一朗 : FPGA ボード上での単一サイクルマイクロプロセッサの設計と検証,立命館大学理工学部卒業論文,2005.

[17]志水健太 : ハード/ソフト協調学習システム上でプロセッサの設計とプロセッサデバッガによる検証,立命館大学理工学部卒業論文,2007.

[18]難波翔一朗：プロセッサ設計支援ツールの実装とハード/ソフト協調学習システムの評価,立命館大学理工学研究科修士論文,2007.
[19] これからが面白いプロセッサアーキテクチャ：
http://www.arch.cs.titech.ac.jp/event/fit2006.html
[20]Cell Broadband Engine 公開情報：http://cell.scei.co.jp
[21]宮岡祐一郎,戸川望,柳澤政生,大附辰夫:不規則なデータパスを持つプロセッサのハードウェア/ソフトウェア協調合成手法,回路とシステム(軽井沢)ワークショップ,2003
[22]小田雄一,宮岡祐一郎,戸川望,橘昌良,柳澤政生,大附辰夫:システムLSI設計における定性的側面を考慮したハードウェア/ソフトウェア分割システム,情報処理学会 DA シンポジウム 2003,pp.169-174,2003.