情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 1　24–42 (Feb. 2011)

メニーコアアーキテクチャのHW評価環境 ScalableCoreシステム

高前田　伸也[†1]　佐藤　真平[†1]　藤枝　直輝[†1]
三好　健文[†2]　吉瀬　謙二[†1]

メニーコアプロセッサの動作を現実的な時間でシミュレートするため，我々はハードウェアによる高速プロトタイピングシステム構築手法の ScalableCore を提案している．ScalableCore は，シミュレーションノードである ScalableCore Unit とそれらの接続インタフェースである ScalableCore Board で構成される．ハードウェアによるシミュレータでは内在する並列性の活用によりソフトウェアによるシミュレータと比較してメニーコアアーキテクチャをより高速にシミュレーションすることができる．加えて，ScalableCore システムではシミュレーションノードの増減が容易であるため，シミュレーション対象であるアーキテクチャに対する高いスケーラビリティが得られる．本稿では，ScalableCore のコンセプトを議論し，その妥当性を述べる．また，提案コンセプトをもとに構築したメニーコアアーキテクチャM-Core のシミュレーション環境 ScalableCore システム Version 1.1 の実装を示す．本システムは多数の小容量 FPGA で構成されており，64 ノードのシミュレーションでは，M-Core のソフトウェアシミュレータ SimMc と比較して，14.2 倍の高速化を実現した．

ScalableCore System: Hardware Environment for Many-core Architectures Evaluation

Shinya Takamaeda,[†1] Shimpei Sato,[†1]
Naoki Fujieda,[†1] Takefumi Miyoshi[†2]
and Kenji Kise[†1]

In order to practically simulate many-core processor, the authors have proposed ScalableCore that is a hardware-based simulator. ScalableCore consists of both simulation nodes and connection interfaces of them named ScalableCore Unit and ScalableCore Board, respectively. Hardware-based simulator can simulate many-core architecture faster than software-simulator by employing the inherent fine-grain parallelism. Additionally, it is easy to increase/decrease the number of simulation nodes in ScalableCore system, so that it can achieve high scalability for the target many-core architecture. In this paper, the concept of ScalableCore is described and the adequacy is discussed. And on the concept, the implemented of ScalableCore system Version 1.1 for many-core architecture M-Core is shown. This system consists of many small FPGAs. The simulation speed for 64-nodes is 14.2 times faster than the corresponding software simulator; SimMc.

1. はじめに

メニーコアプロセッサやそれにともなうソフトウェアの研究・開発を効率的に行うためには，アイディアを検証する環境が必要となる．プロセッサアーキテクチャの場合，構成の変更が柔軟であること，特定のハードウェアが必要でないなどの理由からソフトウェアによるシミュレータを用いることが多い．しかし，プロセッサコア数や実行アプリケーションなどのシミュレーション規模が大きい場合，ソフトウェアシミュレータではシミュレーション速度の問題から現実的な時間でシミュレーションを行うことが困難である．我々は，ハードウェアによる高速プロトタイピングシステム構築手法の **ScalableCore** [1),2)] を提案している．ScalableCore は，タイル型アーキテクチャを主なターゲットとするメニーコアプロセッサ向けのシミュレーション環境構築手法である．本手法により，コア数に対するシミュレーション速度のスケーラビリティを満たしながら，構成の変更に対して柔軟かつ高速なシミュレーション環境の実現を目指す．

本稿では，まず，ScalableCore のコンセプトについて述べる．そしてそのコンセプトをもとに，多数の FPGA を用いて構築したメニーコアアーキテクチャM-Core [3)–5)] のシミュレーション環境 ScalableCore システム Version 1.1 の実装について述べる．本システムは ScalableCore システム開発のファーストステップであるため，高度な高速化手法は適用していない．しかし本システム上では実際に 64 ノードの M-Core プロセッサが動作しており，M-Core のソフトウェアシミュレータ SimMc と比べて 14.2 倍の高速化を達成している．

本稿の構成について述べる．まず，2 章では我々の提案する ScalableCore のコンセプトについて述べる．そしてコンセプトについて，スケーラビリティ，コスト，シミュレーショ

†1 東京工業大学大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†2 電気通信大学大学院情報システム学研究科
Graduate School of Information Systems, The University of Electro-Communications

ン速度などの点で議論する．3 章では M-Core のシミュレータとして実装した ScalableCore システム Version 1.1 の実装について述べる．特に，本システムの特徴である複数 FPGA の協調動作を実現する仕組みと，検証対象プロセッサの実装方法について解説する．4 章ではシミュレーション速度，消費電力などの点で本システムの評価を行う．5 章で関連研究について述べる．最後に，6 章で本稿をまとめる．

2. シミュレーション対象とコンセプト

本章では，主なシミュレーション対象であるメニーコアアーキテクチャM-Core の概要と，タイル型アーキテクチャ向けのシミュレーション環境構築手法の ScalableCore Version 1.1 のコンセプトについて述べる．そしてコンセプトに関する，M-Core アーキテクチャを実装対象としたときのメリット，デメリットを議論する．

2.1 メニーコアアーキテクチャM-Core

はじめに，M-Core アーキテクチャの概要について述べる．**図 1** に M-Core の構成を示す．M-Core はタイル状に配置されたノードをメッシュネットワークで接続する構成を採用しており，ノード数に対する高いスケーラビリティの実現を目指している．各ノードは DMA 転送を用いて明示的にデータのやりとりを行う．各ノードには，チップ上の X 座標と Y 座標の組で表現される固有の ID が割り当てられており，DMA 転送にはその ID を用いる．ノードには，計算ノード（図中 (a)），メモリノード（図中 (b)），パスノード（図中 (c)）の 3 種類がある．計算ノードはアプリケーションプログラムの実行およびパケットの転送を行う．メモリノードはオフチップのメインメモリに接続し，計算ノードとメインメモリのインタフェースの役割を果たす．パスノードはパケットの転送のみを行う．図 1 において，計算ノードは Comp. Node(1,1) から Comp. Node(M,N)，メモリノードは Memory Node(0,0)，パスノードは Path Node(1,0) から Path Node(M,0) および Path Node(0,1) から Path Node(0,N) のそれぞれが対応する．図 1 (a) に計算ノードのブロック図を示す．計算ノードは，コア（演算処理ユニット），ノードメモリ（各ノードが持つ小規模なメモリ），INCC（Inter Node Communication Controller，DMA 転送を管理する），ルータとで構成される．各ノードは独立したメモリアドレス空間を持ち，他のノードあるいはメインメモリへのアクセスは INCC を介した DMA 転送により行う．コアは，DMA 転送のステータス（転送先 ID，転送先アドレス，転送先ストライド，転送元アドレス，転送元ストライド，転送サイズ）を専用ライブラリ MClib を用いて INCC レジスタのメモリマップ領域に書き込むことで，DMA 転送を発行する．ルータは隣接する 4 つのノードのルータおよび自身のノードの INCC の 5 方向と通信をする．通信はパケット単位で行われ，制御は flit（flow control unit）単位で行う．M-Core には flit 幅が 38 ビットの版と 15 ビットの版が存在するが，今回の実装では flit 幅は 15 ビットとした．ノードメモリは小容量の高速な 32 ビット幅の 4 ポートの RAM である．4 ポートの内訳は，コアへの命令供給（Read ポート），コアによるデータアクセス（Read/Write ポート），INCC による読み出し（Read ポート）および INCC による書き込み（Write ポート）である．ノードメモリのアクセスレイテンシは 1 サイクルである．

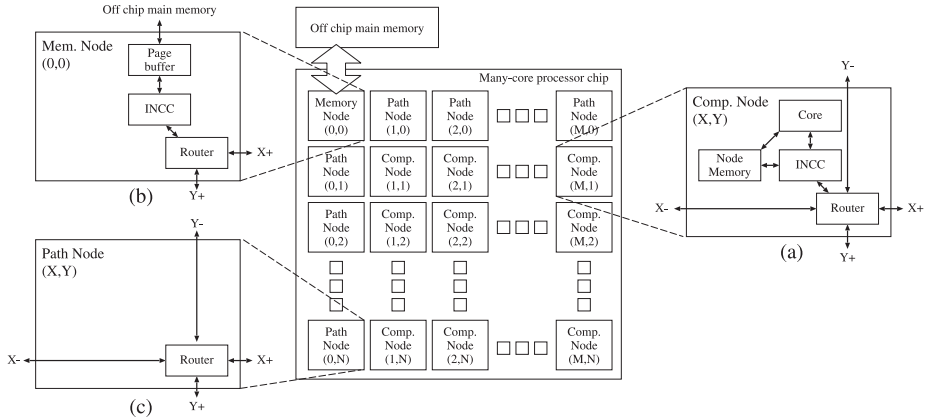

図 1　M-Core アーキテクチャの構成．計算ノード，パスノード，メモリノードの 3 つのノードで構成される
Fig. 1　Structure of M-Core architecture.

M-Core アーキテクチャのソフトウェアシミュレータとして SimMc[3),4)] がある．SimMc において，コアには MIPS32 ベースのシングルサイクルプロセッサが採用されており，コアに関しては命令レベルのシミュレーションが可能である．INCC とルータはサイクルレベルの実装がされているため，ネットワークに関してはサイクルレベルのシミュレーションが可能である．可読性を重視し，SimMc には処理の並列化などの高速化手法を適用していない．

2.2 コンセプトの提案

次に，我々が提案する ScalableCore のコンセプトについて述べる．

いくつかの異なる条件における評価を行うために，プロセッサコアやオンチップルータなどのモデルが変更できる，といった柔軟性がメニーコアアーキテクチャのシミュレーション環境には求められる．また，プロセッサアーキテクチャ研究においては，1 つの設定や条

件につき，複数のベンチマークで評価をとるのが一般的である．評価項目によりシミュレーションの規模は異なるが，1 つの評価をとるのに，合計おおよそ数十 G サイクル程度のシミュレーションを行うことが多く，そのシミュレーション時間は長い．特にメニーコアアーキテクチャのシミュレーションではシミュレーションに要する時間は長くなりがちである．たとえば，ソフトウェアシミュレータの SimMc を用いて 64 ノードの M-Core アーキテクチャの連続した 80G サイクルの挙動をシミュレーションする場合，そのシミュレーションにはおおよそ 2 週間もの期間を要する．そのため，メニーコアアーキテクチャのシミュレーション環境は高速にシミュレーションできることが好ましい．

ScalableCore Version 1.1（以下，単に ScalableCore と呼ぶ）は，**コア数に対するシミュレーション速度のスケーラビリティ**に主眼を置いた，シミュレーション環境構築手法である．ScalableCore は，タイル型アーキテクチャを主な対象とし，シミュレーション対象全体に対して小規模な FPGA を複数用いてシステムを構成する．各 FPGA 上にシミュレーション対象を部分的に実装し，それらを連携させることで対象全体のシミュレーションを行う．本稿ではこの手法に基づいたシステムを **ScalableCore システム**と呼ぶ．本コンセプトをもとに構築した ScalableCore システム Version 1.1 はタイル型アーキテクチャである M-Core のハードウェア評価環境として用いる．

ScalableCore システムの主な構成要素は，シミュレーション対象の一部分に対応するシミュレーションノード **ScalableCore Unit**（以下 Unit と略す）と，それらを接続するインタフェース **ScalableCore Board**（以下 Board と略す）である．ScalableCore の概要を**図 2** に示す．図は 4 × 4 の 16 ノードで構成される ScalableCore システムのモデルである．4 枚の Board を 1 つの Unit を取り囲むように配置し，各 Unit の I/O ポートと隣接する Unit の I/O ポートを Board 上の配線を介して接続する．隣接 Unit 間では適宜同期をとりながらシミュレーションを進める．各 Unit には FPGA および SRAM を搭載し，これらの上にプロセッサコアとシステム制御機構（ScalableCore Module）の実装を行う．各 Unit の電源は左から供給し，Unit 上に用意した電源用配線および Board を介して，右の Unit へと順に供給する．シミュレーション情報の出力には小型コマンドインタプリタ型ディスプレイを用いる．ディスプレイには Unit からシリアル通信で表示情報を転送する．

ScalableCore システム Version 1.1 では Node(1,1) から Node(M,N) で示される計算ノード群のシミュレーションを行う．本システムの各 Unit には計算ノード 1 つを実装し，シミュレーション対象のノード数に応じて接続する Unit 数を調整する．たとえば，図 2 のように 4 × 4 の 16 ノードの構成の場合，横方向に 4，縦方向に 4 の 16 個の Unit を並べ，Board

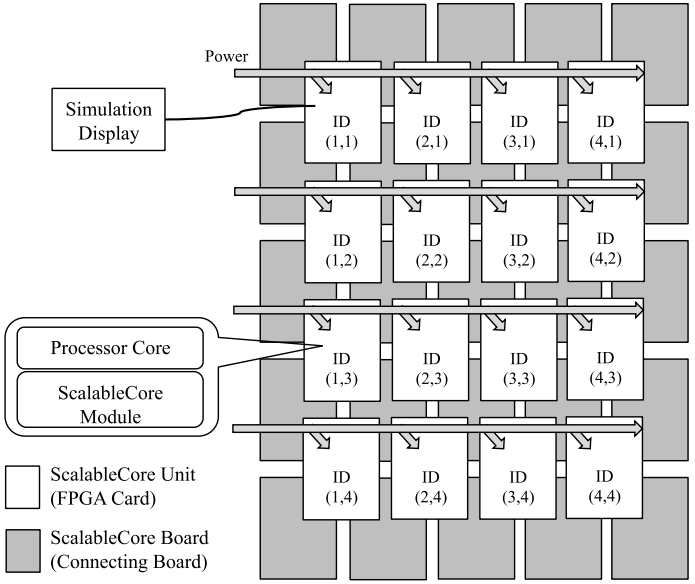

図 2　4 × 4 の ScalableCore システムの構成
Fig. 2　Formation of 4 × 4 ScalableCore system.

を用いて Unit 間を接続する．もし，4 × 5 の 20 ノード構成に変更する場合は，縦軸方向の Unit 数を 5 個にし，縦方向の接続に用いる Board 数は 6 個にする．このように M-Core アーキテクチャのようなタイルアーキテクチャであれば，シミュレーション対象のコア数に応じて，シミュレーションシステムの大きさを変更すればよい．

今回実装対象としなかったパスノードおよびメモリノードの実現方法について述べる．パスノードはルーティングのみを行うノードであるため，計算ノードからコアおよび INCC を取り除くことでパスノードを実現できる．メモリノードの実現は今後の課題である．

ScalableCore システムの利用目的とシナリオについて述べる．ScalableCore システムの典型的な利用目的はシミュレーションの高速化である．ScalableCore システムは高速にシミュレーションできるという利点を持つが，シミュレーション中の内部状態の読み出しが難しい．他方，ソフトウェアシミュレータはシミュレーション速度は低速であるが，シミュレーション中の内部状態の読み出しが容易という利点がある．これら 2 つの利点は開発を行ううえで有用であるため，ScalableCore システムはソフトウェアシミュレータを置き換

えるものではなく，ともに補完的な関係にある．想定される ScalableCore システムの利用シナリオとして以下が考えられる．ハードウェア動作のデバッグ時は，コンパイル時間が短く，内部状態を自由なタイミングで出力することが可能であるソフトウェアシミュレータを利用する．デバッグ完了後に，パラメータなどを変更しながらの大規模なシミュレーションには，高速にシミュレーションできる ScalableCore システムを利用する．

本システムを拡張することにより，M-Core アーキテクチャのプロセッサコアやオンチップルータのモデルを変更した場合の評価が可能である．具体的には，プロセッサコアのパイプライン化やスーパスカラ化した場合の評価や，オンチップルータの仮想チャネルの追加やパイプライン化した場合の評価などである．このように，モデルを変更する場合には，ScalableCore システムに含まれるハードウェア記述の変更および回路情報（bit ファイル）の再構成・再書き込みが必要になる．

2.3　コンセプトに関する議論

ハードウェアによる高速な評価環境はその特殊性から高価であることが多い．ScalableCore は速度とコストの両方で優れた評価環境の実現を目指している．本節では，M-Core アーキテクチャを実装対象としたときの，提案手法の妥当性について議論する．評価項目は，開発時間の長短を大きく左右する**回路合成時間**，システムを構成する際のコストの目安となる**コアあたりのコスト**，**システム全体の複雑性**，**シミュレーション速度**の 4 点である．この 4 点について，各コアがハードマクロ化されていないという条件で評価する．複数の FPGA を用いてプロセッサ全体を構成する提案手法と，大容量の FPGA 上にプロセッサ全体を実装する手法とを比較する．

2.3.1　回路合成時間

各コアがハードマクロ化されていないという条件の下，FPGA をターゲットとした回路合成時間について議論する．回路規模と回路情報の合成に要する時間の関係を知るために，Verilog HDL で記述されたマルチサイクルの MIPS プロセッサコアである MIPSCORE[6] を 1 単位として用いて，Spartan-3E 1600E を対象に，実装するコア数を変化させ回路合成を行い，要する時間を測定した．測定には，Intel®Core™2 Quad Q6700 2.66 GHz，メモリ 2 GB，Windows XP Professional SP3 という一般的な構成の計算機を用いた．合成ツールには Xilinx ISE 11.1 を用いた．

図 3 に合成対象の回路規模と回路合成時間の関係を示す．横軸は合成対象コア数，縦軸は回路合成時間である．ここでいう回路合成時間とは，1) HDL からネットリストを生成する “Synthesis”，2) ネットリストを対象デバイスにマップする “Map”，3) マップされたネットリストを実際に FPGA 内の要素に配置し配線を行う “Place and Route” の 3 つのフェーズに要する時間の合計のことを指す．グラフ上の数値はそれぞれの処理に要する時間を示している．回路合成時間は，タイルアーキテクチャの実装を想定し，実装するコアの個数のみを変更して計測した．そのため合成対象は構成の等しい要素を多く含む．しかしこの場合でも，全体規模が大きくなると，合成時間がほぼ線形に増加し，9 コアの場合には，1 コア時と比較して 8.15 倍の回路合成時間を要する．そのため大規模 FPGA にプロセッサ全体を実装する場合，対象アーキテクチャにかかわらず回路規模に応じた合成時間を要することが分かる．一方，提案手法のように対象アーキテクチャを部分的に実装し，それらを結合しプロセッサ全体を構築する場合には，同一構成要素の合成は 1 度で済み，同一構成要素の割合が高いほど，全体の合成時間が短縮できる．M-Core を実装する場合，プロセッサ内には構成が等しい要素を多く含むため，論理合成時間の短縮が期待できる．

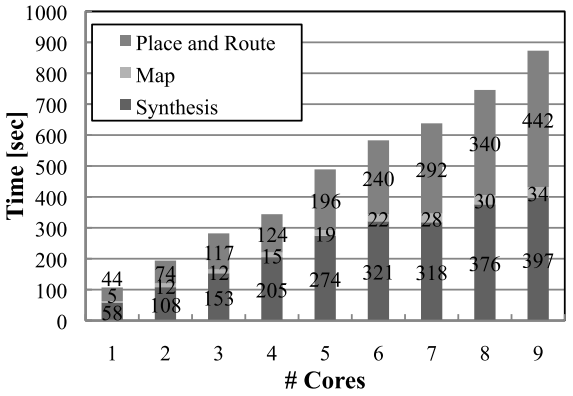

図 3　回路合成時間．Spartan-3E 1600E をターゲットにプロセッサコアの数を変化させて回路の合成を行った
Fig. 3　Compile time.

加えて，提案手法ではコア数の増減は利用する FPGA（=Unit）数を変更することで実現できるため，回路の再合成が必要ない．すなわちコア数に対するスケーラビリティの点で提案手法が優れているといえる．

2.3.2　コアあたりのコスト

1 コアあたりのコストについて議論する．1 コアあたりのコストとは，プロセッサコア 1 つあたりの金銭的コストのことを指す．FPGA チップはメモリなどのその他部品に比べ，一般に高価であるため，議論を簡単にするため，FPGA チップの金額をベースに議論を行う．

表 1　各 FPGA の 2010 年 1 月 18 日時点の Digi-Key Corp. での価格
Table 1　Price of each FPGA.

デバイス		価格［円］
Spartan-3E	XC3S500E	2,078
	XC3S1200E	3,808
	XC3S1600E	6,102
Virtex-5 LX	XC5VLX30	24,024
	XC5VLX50	37,838
	XC5VLX85	89,129
	XC5VLX110	132,132
	XC5VLX155	206,606
	XC5VLX220	348,348
	XC5VLX330	839,038

議論にあたり，エントリ向け小容量 FPGA の Xilinx Spartan-3E と高速動作可能な大容量 FPGA の Xilinx Virtex-5 のうち，いくつかの価格を調査した．価格を**表 1** に示す．価格は Digi-Key Corp.[7] のウェブサイトで調査した．1 つのデバイスで複数の選択肢（ピン数やパッケージ）がある場合には，その中で最も安価なものを選択した．その価格をもとに，1 FPGA 上に複数のプロセッサコアを実装する場合に，各 FPGA に実装可能なコア数と 1 コアあたりの価格を算出した．1 FPGA に集積できるコア数の算出には，M-Core アーキテクチャの計算ノードの合成結果のうち，使用率の最も大きかった LUT 使用量の見積り値を基準として用いた．Spartan-3E シリーズに関しては，XC3S500E をターゲットに論理合成した際の LUT 使用量見積りの 7,170 を，Virtex-5 LX シリーズに関しては，XC5VLX30 をターゲットに論理合成した際の LUT 使用量見積りの 5,533 を 1 ノードの大きさとして用い，1 つの FPGA チップに集積できる計算ノードの数を算出した．**図 4** に 1 FPGA に実装可能なノード数，**図 5** に 1 ノードあたりの価格を示す．横軸はデバイス，縦軸はそれぞれ，1 チップに実装可能なノード数と 1 ノードあたりのコストを示す．1 つの FPGA に実装可能なノード数はデバイスのグレードが上がるにつれて増加し，Virtex-5 XC5VLX330 では 37 ノードが 1 つの FPGA に実装可能である．1 ノードあたりの価格に着目する．エントリクラスの Spartan-3E では 1 ノードあたりの価格はほぼ一定であることが分かる．一方，高速動作が可能な Virtex-5 では，グレードが上がるにつれて 1 ノードあたりの価格が増加する．

単一の FPGA に多数のコアを実装するには，大容量の FPGA が必要となりコストが高くなる．一方，提案手法のように複数の FPGA を用いて実装する場合には，それぞれは比較的小容量の FPGA を用いることが可能であるため，全体のコストを抑えることが可能となる．

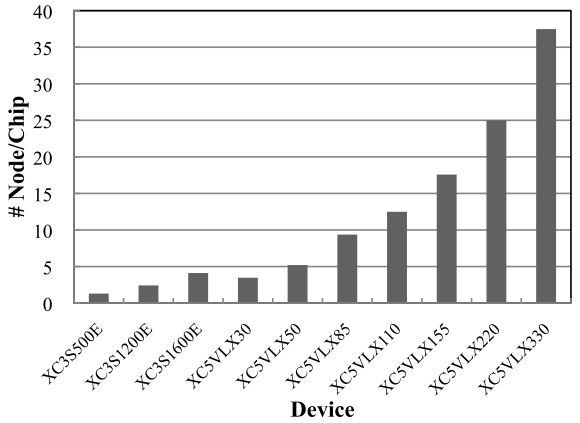

図 4　1 FPGA に実装可能なノード数
Fig. 4　The number of nodes per FPGA.

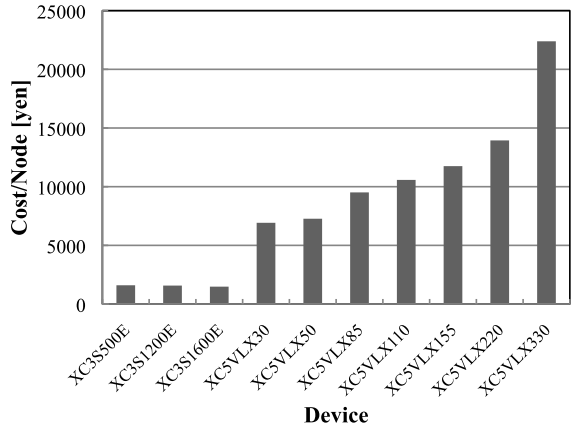

図 5　1 ノードあたりのコスト
Fig. 5　Cost per node.

2.3.3 複　雑　性

複雑性の議論を行う．複雑性とは，検証対象のアーキテクチャの実装のほかに，どのような実装オーバヘッドが生じるかを指す．

まず，ローカルメモリ（M-Core ではノードメモリ）の扱い方に着目する．大容量 FPGA にプロセッサ全体を実装する場合，ローカルメモリの総容量が大きいと FPGA 内部のブロック RAM のみでは不足する．その場合，FPGA 外部のメモリシステムを効率的に利用するために，仮想的なローカルメモリを構成するなどの工夫が必要となる．そのため，その制御の部分に複雑性が生じる．一方，複数の FPGA に分散してプロセッサを構成する場合，各 FPGA に 1 つの SRAM を接続すればよく，構成をシンプルにすることができる．

次にコア間通信に着目する．単一 FPGA にプロセッサ全体を実装する場合はコア間通信の実装が FPGA のチップ内で完結するためシンプルである．一方複数の FPGA に分散して実装する場合には，コア間通信を FPGA 間通信を含む形で再現する必要が生じる．また複数の FPGA を用いる場合の特有の問題として，FPGA 間の同期があげられる．サイクルレベルでシミュレートするには，システム全体で同期をとる必要がある．

したがって，複雑性の点ではトレードオフがある．

2.4　シミュレーション速度

シミュレーション速度の議論を行う．シミュレーション速度は実装依存であるため詳細に述べることは困難であるが，単一の大型 FPGA にシミュレーション対象全体を実装する方が，複数の FPGA に分散して実装するよりも高速であると考えられる．なぜならば，複数 FPGA にまたがって構成する場合には，FPGA 間通信がボトルネックになりうるからである．そのため，単一の FPGA に実装可能な大きさのアーキテクチャのシミュレーションに関しては提案手法が不利である．

以上 4 点の比較から，M-Core アーキテクチャを実装する場合には，回路合成時間，スケーラビリティおよびコストの点で提案手法が優れているといえる．しかし実装の複雑性とシミュレーション速度では，提案手法が優れているとはいいきれない．そのためバランスを十分に考慮し，構成を決定する必要がある．今回我々は，コア数に対するスケーラビリティを重視し，あらかじめ拡張可能な形でシミュレーションシステムを構築するために，ScalableCore を提案する．

3.　ScalableCore システムの実装

本章では，前章で述べたコンセプトに基づいて実装した ScalableCore システム Version 1.1 の説明を行う．前述のとおり，M-Core アーキテクチャの計算ノード群を実装対象としている．ここでは特に，拡張可能なシステムの構築方法と対象アーキテクチャの実装方法について詳しく述べる．

3.1　ハードウェアプラットフォーム

図 6 は，実際に制作した ScalableCore Board である．図 6 左は，ScalableCore システム中で左端に用いる Board である．この Board は，行方向で共有する電源（DC 5 V）の入力端子およびそのスイッチと，各ノードの初期メモリイメージを記録している MMC（Multi Media Card または SD カード）を挿入するスロットを搭載している．図 6 右は，左端以外で利用する Board である．図 6 中どちらの Board も隣接する Unit（FPGA カード）間を接続する役目がある．また，各 Unit に供給する電源（DC 3.3 V）を生成するために，3 端子レギュレータおよび周辺回路（コンデンサ）を搭載する．左端から入力された電源は Unit 上に用意された電源用配線を経由して，隣の Board へと供給される．システムの安定動作のために，Board 上の 3 端子レギュレータ用コンデンサは試行錯誤のうえ，選定した．GND は Unit を介してシステム全体で共有する．DC 5 V はシミュレーション情報表示用のコマンドインタプリタ型液晶[8] に供給する．MMC の各種信号線は Board に対して右下の Unit に接続される．**図 7** は，フリーソフトのプリント基板エディタ PCBE[9] を用いて作成した ScalableCore Board の設計図である．基板は 2 層基板であり，図はそのうちの片面のみを示している．M-Core アーキテクチャを実装する本システムでは，後述するローカルバリア同期を用いて，隣接 Unit とのみ同期をとるため，隣接 Unit 以外と直接通信を行う必要がない．そのため，Unit 間の信号は 1 つの Board しか経由しない．このように信号伝送路を短くすることで，ノイズの影響を受けにくくしている．設置面積を削減するため

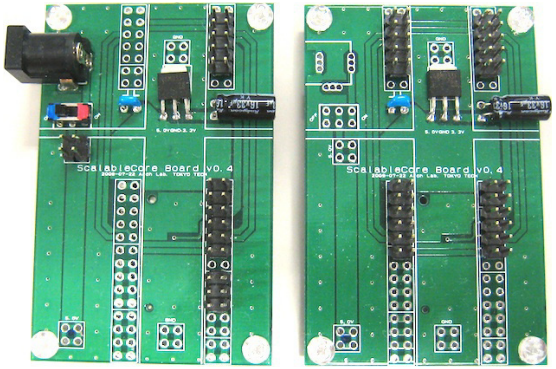

図 6　制作した ScalableCore Board．左は左端用，MMC（SD カード）が挿入可能．右は左端以外
Fig. 6　Our original ScalableCore Board.

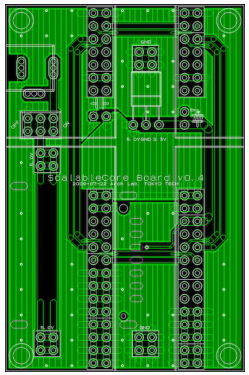

図 7　ScalableCore Board の設計図．設計にはプリント基板エディタ PCBE を用いた
Fig. 7　Design of ScalableCore Board.

図 8　制作した ScalableCore Unit（FPGA カード）．Xilinx Spartan-3E XC3S500E と 8 bit × 512 K の容量 512 KB の SRAM を搭載する
Fig. 8　Our original ScalableCore Unit (FPGA Card).

に，Board の裏面に小型の磁石を取り付け，ホワイトボードにシステム全体を貼り付け，システムを垂直に設置することを可能にしている．

Board 同様に ScalableCore Unit も設計・制作した．**図 8** は，実際に制作した ScalableCore Unit（FPGA Card）である．**図 9** は ScalableCore Unit の設計図である．Board 同様，PCBE を用いて設計した．基板は 2 層基板であり，図はそのうちの片面のみを示している．

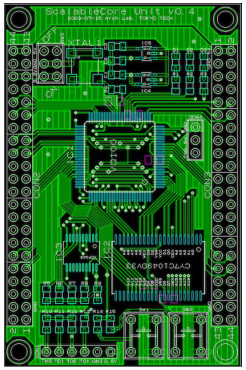

図 9　ScalableCore Unit の設計図．設計にはプリント基板エディタ PCBE を用いた
Fig. 9　Design of ScalableCore Unit.

Xilinx Spartan-3E の XC3S500E（50 万ゲート相当）と 8 bit × 512 K の容量 512 KB の SRAM を搭載する．加えて，電源スイッチ，LED，押しボタンスイッチ，27 MHz の水晶発振器と回路情報（bit ファイル）を記録する不揮発性の ROM である Xilinx XCF04S（コンフィギュレーション ROM）を搭載する．Unit は左端から供給された電源（DC 5 V）を横方向に供給する配線の役目も担っている．FPGA 間の通信は，使用するピン数を抑えるためにシリアル通信で行う．

アプリケーションの実行結果やシミュレーション状況の表示には，コマンドインタプリタ型の小型液晶ディスプレイを用いる．ディスプレイには Unit からシリアル通信で表示情報を転送する．

図 10 に 8 × 8 の 64 Unit のシステムの様子を示す．DC 5 V/3 A の安価な小型スイッチング電源[10] を 3 系統用いて 64 Unit を駆動している．図では 3 つの系統を Power Domain 1, Power Domain 2, Power Domain 3 で示している．図の左側に示した矢印は電源を分配していることを示している．今回用いたスイッチング電源では，1 系統で 30 Unit 程度まで安定駆動することを確認している．各 Unit 上の FPGA に M-Core の計算ノードを実装し，64 Unit を接続することで 64 ノードの M-Core 計算ノード群を実現している．各 Unit 上の計算ノードには前述のとおり，(X,Y) で表される ID が割り当てられる．ScalableCore システムでは，ID は初期化時に設定する．図 10 中の (1,1) の位置に相当する Board には SD カードが挿入されている．SD カードの制御線は (1,1) に相当する Unit に接続されており，

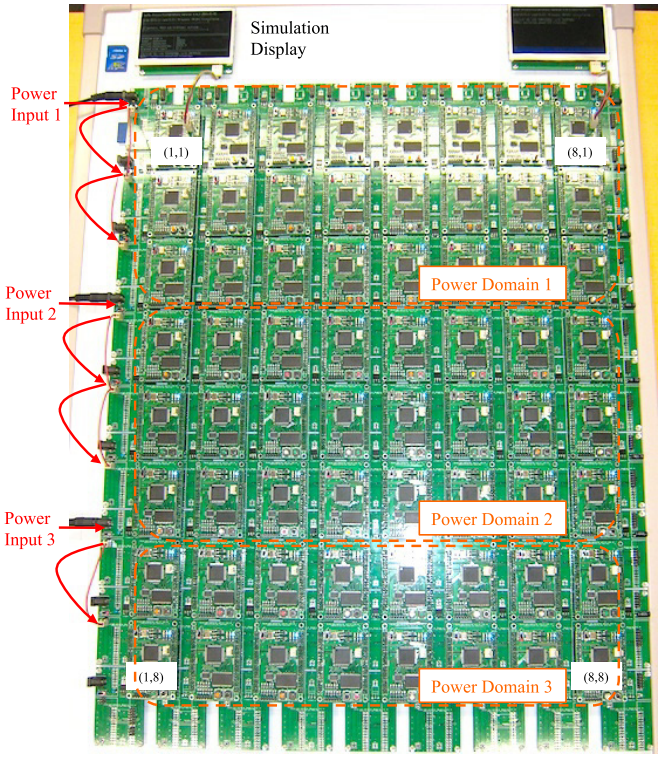

図 10　8 × 8 の 64 Unit の ScalableCore システム．DC 5 V/3 A の小型スイッチング電源を 3 系統用いて駆動している

Fig. 10　ScalableCore system of 8 × 8, 64 Units.

システム初期化時にローカルメモリの初期イメージを読み込む．各 Unit にはコマンドインタプリタ型ディスプレイを接続することが可能であり，図では (1,1) と (8,1) の Unit に接続している．

各 Unit の FPGA の回路情報はシステムの電源投入時に各 Unit のコンフィギュレーション ROM から読み込まれる．そのためシミュレーションを行うには，あらかじめ書き込み端子（JTAG 端子）経由でコンフィギュレーション ROM に FPGA の回路情報を書き込む必要がある．各 Unit は独立した書き込み端子を持つため，書き込む際にはこの端子を介して回路情報を転送する．

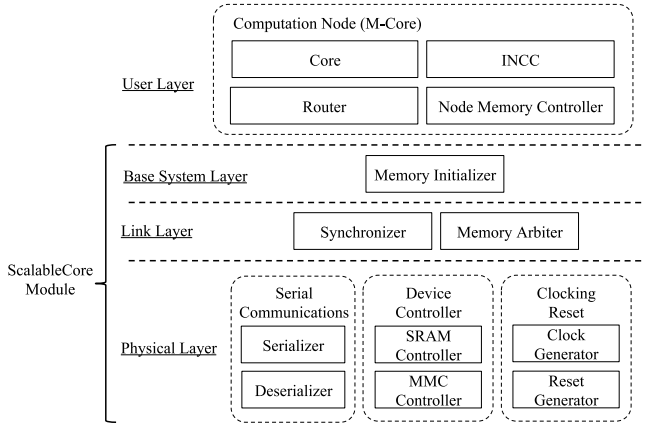

図 11　各 Unit に実装したモジュールの階層図

Fig. 11　Layer structure of modules on a unit.

3.2　ハードウェアアーキテクチャ

図 11 は，各 Unit に実装したモジュール構成の階層図である．すべてのモジュールを Verilog HDL で記述している．モジュールは**物理層**（Physical Layer），**リンク層**（Link Layer），**ベースシステム層**（Base System Layer），**ユーザ層**（User Layer）の 4 つの階層に分けられる．このうち，物理層，リンク層，ベースシステム層をまとめて ScalableCore Module と呼ぶ．ScalableCore Module はシステムの制御やシミュレーション対象アーキテクチャの実装をサポートする機能の提供を行う．一方，ユーザ層はシミュレーション対象のアーキテクチャに相当する階層であり，本システムでは，ユーザ層に M-Core の計算ノード（Core，INCC，Router，Node Memory Controller）を実装した．そして適切な数の ScalableCore Unit を ScalableCore Board を用いて接続し，M-Core 計算ノード群のプロトタイプを構成する．次節以降では，各階層の役割とその実装について述べる．

ここで，ScalableCore システムの実現に重要な**仮想サイクル**という概念を導入する．仮想サイクルとはシミュレーション対象における 1 サイクルに相当し，ScalableCore システムでは複数クロックサイクルをかけて 1 仮想サイクルの動作を進める．その様子を**図 12** に示す．1 仮想サイクル中では，対象アーキテクチャの動作はもちろん，それに付随する Unit 上の SRAM へのアクセスや Unit 間通信・同期などの処理が行われる．詳しくはユーザ層の項で述べるが，仮想サイクルの概念を用いることで，Unit 上の物理リソースによる制約

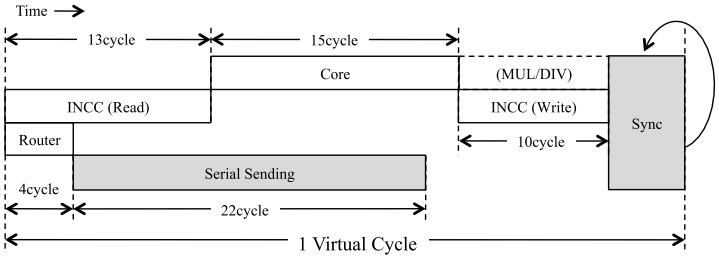

図 12　ユーザ層のモジュールの動作タイミング
Fig. 12　Timing chart of user layer modules.

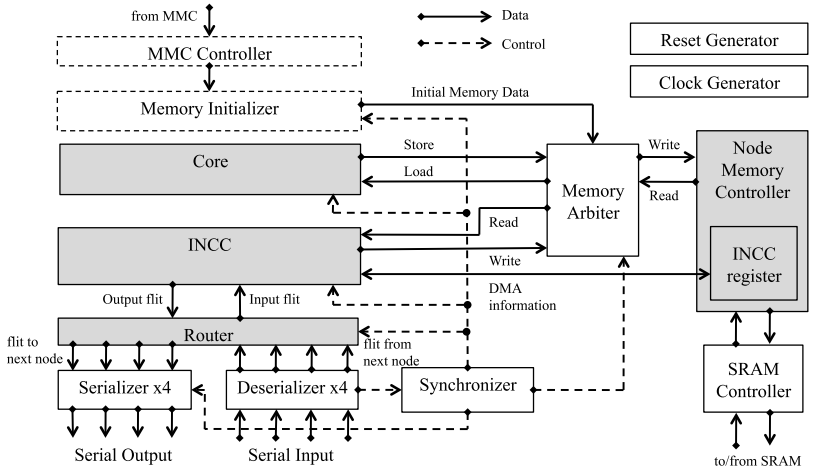

図 13　各 Unit に実装したハードウェアモジュールの構成図．灰色で示されるモジュールは M-Core アーキテクチャの計算ノードの要素である．点線で示されるモジュールは初期化時のみに動作する
Fig. 13　Structure of hardware modules on each unit.

を緩和し，ソフトウェアシミュレータに似た柔軟性を得ることが可能となる．加えて，仮想サイクル中の同期の段階で Unit 間の処理のずれを吸収することが可能となり，複数 FPGA にまたがったシステムが実現できる．

図 13 は各モジュールがどのようなデータのやりとりを行うかを示している．すべてのモジュールは基本的に，Clock Generator が生成する 45 MHz の単一クロックに同期して動作

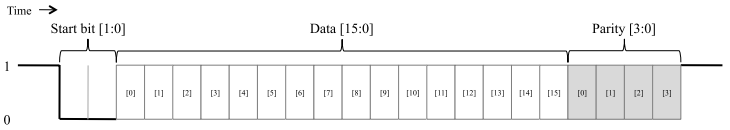

図 14　ScalableCore システム Version 1.1 に実装した調歩同期式シリアル通信機構のビット構成
Fig. 14　Data structure of asynchronous serial communications on ScalabeCore system Version 1.1.

する．図 13 において，灰色で示されるモジュールは M-Core の計算ノードの構成要素であり，ユーザ層に位置する．その他のモジュールは ScalableCore Module に相当する．ユーザ層に該当する計算ノードの各モジュールは，Synchronizer により 1 仮想サイクル中の動作タイミングが調整される．また，Memory Arbiter は SRAM Controller にアクセスする Core，INCC および Memory Initializer の調停を行う．計算ノードは 1 サイクルに最大 4 回（命令フェッチ，ロード/ストア，DMA リード，DMA ライト）の 32 ビット幅のメモリアクセスを行う仕様となっている．しかし Unit 上の SRAM は 8 ビット幅のメモリであるため，複数サイクルをかけてアクセスする必要がある．そのため Memory Arbiter がアクセスタイミングを調整し，仮想的にマルチポートの 32 ビット幅メモリを構成している．

以下に，図 13 の各モジュールの役割とその実装について，図 11 に示した階層別に述べる．

3.2.1　物　理　層

物理層は ScalableCore Module の中で最下層に位置し，デバイスの制御，通信機構の制御を行うモジュールが相当する．物理層にはシリアル通信機構の Serializer と Deserializer，SRAM Controller，MMC Controller，各 FPGA 内のクロックおよびリセット信号の管理を行う Clock Generator と Reset Generator が含まれる．

まず，シリアル通信機構の Serializer および Deserializer について説明する．隣接 4 方向の Unit との通信はシリアル通信により行われる．ScalableCore システム Version 1.1 では，実装を簡単にするために，調歩同期式のシリアル通信機構を，送信 4 系統，受信 4 系統の計 8 系統実装した．これらの動作周波数は 45 MHz である．

図 14 に示すように，スタートビット 2 ビット，データ部 16 ビット，パリティ部 4 ビットの計 22 ビット単位で通信が行われる．送信側はスタートビット，データ部，パリティ部の順にビット単位で隣接 Unit に送出する．受信側はデータリカバリ機構[11] を含む受信回路によりビット単位で受信する．

M-Core アーキテクチャでは，毎サイクル 15 ビット幅の flit および，ルータの入力バッファの状態を示す 1 ビットの Ready 信号を隣接ノード間で互いに通信する．本システムの

送信側は，上位のモジュールから入力される flit の 15 ビットと Ready 信号 1 ビットの計 16 ビットをシリアル通信のデータ部として，これらにパリティの 4 ビットを付加した計 20 ビットを隣接 Unit に送信する．受信側はパリティを用いて受信データの正誤の確認を行い，上位のモジュールへと flit データを渡す．もしノード間通信幅の異なるアーキテクチャを実装する場合には，このシリアル通信の転送サイズを変更する．

SRAM Controller は Unit に実装された非同期 SRAM の制御を行う．他のモジュールは本コントローラを介して同期式メモリとして SRAM を利用する．

MMC Controller は MMC の SPI モードをサポートし，クロック線，入力データ線，出力データ線，各 1 ビットの計 3 本で MMC を制御する．MMC にはあらかじめローカルメモリの初期イメージを記録しておき，初期化時に後述する Memory Initializer が本コントローラを介してメモリ初期イメージを読み出す．そのため，本コントローラは初期化が完了すると動作が停止する．

Clock Generator は，Unit 上に実装された 27 MHz の水晶発振器から DCM（Digital Clock Manager）[12] を介して 45 MHz のシステムクロック信号を生成するモジュールである．このクロック信号は Unit 内全体で利用される．Reset Generator はリセット信号の生成を行うモジュールである．

3.2.2 リンク層

リンク層は物理層のモジュールを制御し，各モジュールのタイミング制御や Unit 間の同期を行うモジュールが相当し，ScalableCore システムの中枢を担う．図 11，図 13 中の Synchronizer は隣接 Unit 間の同期をとる．また，1 仮想サイクル中で各モジュールが動作するタイミングを調停する．Memory Arbiter はユーザ層のモジュール単位（Core，INCC など）で SRAM へのアクセスを制御する．各 ScalableCore Unit には 8 ビット幅・シングルポートの SRAM が搭載されている．しかし，実装対象の M-Core アーキテクチャが要求するメモリ幅およびポート数はこれよりも多い．そのため，1 仮想サイクル内で複数回の SRAM アクセスを行うことにより，ノードメモリの機能を実現する．M-Core における Core は，1 仮想サイクル中に命令フェッチおよびロード/ストアの最大 2 回の 32 ビットメモリアクセスを行う．そのため，1 仮想サイクル中で 8 サイクルが Core のメモリアクセス期間として割り当てられる．また，INCC は DMA リードおよび DMA ライトの最大 2 回の 32 ビットメモリアクセスを行う．そのため INCC も Core と同様に，1 仮想サイクル中で 8 サイクルが INCC のメモリアクセス期間として割り当てられる．Memory Arbiter は，Core と INCC のどちらのモジュールに SRAM へのアクセスを許可するかを切り替える．具体的には，アドレス信号および SRAM に対する書き込みデータ信号の選択，および SRAM からの読み込みデータ信号の伝達を行う．加えて，システムの初期化時は Memory Initializer が SRAM にメモリの初期イメージを書き込む．そのため，初期化時にのみ Memory Initializer のアクセスを許可する．

割り当てられた期間中に，どのタイミングでどのアドレスにアクセスするかは，ユーザ層の各モジュール記述で指定する．ScalableCore システム Version 1.1 の場合，Core は割り当てられたアクセス期間の 8 サイクルの間に，命令フェッチ（4 サイクル）およびロード/ストア命令によるデータアクセス（4 サイクル）を行う．INCC は割り当てられたアクセス期間の 8 サイクルの間に，DMA 転送による書き込み（4 サイクル）および読み出し（4 サイクル）を行う．これにより，8 ビット幅・シングルポートメモリから 32 ビット幅・4 ポートの仮想的なメモリを構成し，M-Core のノードメモリを実現する．

ある仮想サイクルのシミュレーションを正しく行うには，その前の仮想サイクルで他の Unit で生成されたデータがあらかじめ伝達されている必要がある．M-Core のように演算ユニットとオンチップネットワークのルータを持つノードがメッシュネットワークで接続されたアーキテクチャの場合，ノード間の通信はルータを介して行われる．そのため，1 つのルータを 1 つの Unit に実装する場合に，各 Unit は 1 仮想サイクルごとに同期をとる必要がある．ネットワークがメッシュ型であるため，N サイクル目に，あるノード A のルータに存在する情報が $N+1$ サイクル目までに伝達されうる範囲は，ノード A の内部またはノード A の隣接ノードのルータに限られる．ScalableCore システム Version 1.1 ではこの点に着目し，すべての Unit で同期をとるのではなく，隣接する Unit 間のみで同期をとる**ローカルバリア同期**を用いる．ローカルバリア同期により，各 Unit が行う処理で利用するデータの世代を揃え，正しくシミュレーションを進めることができる．

図 15 にローカルバリア同期の様子を示す．ユーザ層の各モジュールは Synchronizer の制御に従い，図 12 に示したタイミングで動作する．ユーザ層のモジュールが送信すべきデータの生成が完了すると，Serializer は隣接 Unit に対してデータ送信を開始する．そして Synchronizer は隣接 Unit からのデータ受信の完了，データ送信の完了，ユーザ層モジュールの動作完了を待機する（同期待ち）．これは図 12 中の “Sync” に対応する．同期が完了すると各モジュールの動作は次の仮想サイクルへと進む．この一連の処理により，隣接 Unit との正しいデータの授受が完了する．

ScalableCore システムでは，搭載する水晶発振器の差異や電源投入のタイミングなどの原因で，各 Unit が毎仮想サイクルに要する時間はまちまちである．そのため，ある Unit

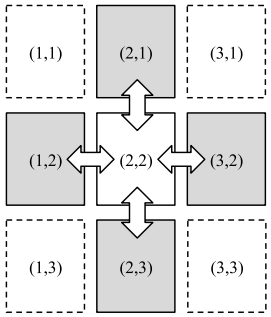

図 15　ローカルバリア同期．ノード (2,2) はノード (2,1)，(1,2)，(3,2)，(2,3) とのみ同期をとる
Fig. 15　Local barrier synchronization.

（Unit A とする）が仮想サイクル N における同期待ちを完了する前に，ある隣接 Unit（Unit B とする）は仮想サイクル N における同期を先に完了し，仮想サイクル $N+1$ のシミュレーションに進むことがある．そして，Unit B がさらに先行し，Unit B は仮想サイクル $N+1$ におけるデータを生成し，Unit A が仮想サイクル N における同期待ちを完了する前に，Unit A に仮想サイクル $N+1$ で生成されたデータを送信することがある．このような場合に，Unit B の仮想サイクル N のデータが消費される前に Unit B の仮想サイクル $N+1$ のデータによって上書きされるのを防ぐために，各 Unit の各受信ポートにはキューが実装されている．受信したデータは各受信ポートキューにエンキューされ，同期が完了した段階でデキューされる．これにより，データがとりこぼされることなく正しく同期が行われる．

3.2.3　ベースシステム層

ベースシステム層には物理層やリンク層が提供する機能を利用する ScalableCore Module が該当する．ローカルメモリの初期イメージを SRAM に書き込むために，Memory Initializer は，MMC Controller を制御する．メモリ初期イメージの転送完了後は動作を停止する．

3.2.4　ユーザ層

ユーザ層にはシミュレーション対象のモジュールが該当する．本システムでは M-Core アーキテクチャの計算ノード（Core，INCC，ルータ，ノードメモリ）を Verilog HDL で実装した．ただし，今回のバージョンでは Core の浮動小数点命令ユニット（FPU）は実装していない．ノードメモリは Unit 上の SRAM を利用して実現した．ユーザ層の Node Memory Controller は，メモリマップされた INCC レジスタへのアクセスとノードメモリへのアクセスの切替えを行う．

```
always @(posedge CLK or negedge RST_X) begin
    if(!RST_X) begin
        /* Reset statement */
        A <= 0;
    end else if(EN) begin
        /* assign statement */
        A <= IN_A + IN_B + IN_C;
    end
end
```

(a) Single-cycle model

```
always @(posedge CLK or negedge RST_X) begin
    if(!RST_X) begin
        /* Reset statement */
        A     <= 0;
    end else if(EN) begin
        /* assign statement */
        tmp0  <= IN_A + IN_B;
        state <= S1;
    end else if(state == S1) begin
        tmp1  <= tmp0 + IN_C;
        state <= S2;
    end else if(state == S2) begin
        A     <= tmp1;
        state <= S0;
    end
end
```

(b) Multi-cycle model

図 16　ユーザ層モジュールの記述方法
Fig. 16　Rule to describe a user layer module.

Core，INCC，ルータの各モジュールは，リンク層の項で述べたとおり，図 12 のように 1 仮想サイクル中であらかじめ決められた順番に動作する．メモリアクセスなどの 1 仮想サイクルで行わなければならない処理の競合の具合に応じて，動作する順番や同時に動作するモジュールを決定し，Synchronizer にその順序と重ね合わせを記述する必要がある．図 12 に示した各モジュールが動作するタイミングは，モジュール間で SRAM へのアクセス競合が発生しないように調整したものである．

ユーザ層のモジュールは Synchronizer の制御に従って動作する必要があるため，通常の HDL 設計とは異なった記述が必要となる．**図 16** に Verilog HDL によるサンプルコードを示す．図 16 (a) は対象アーキテクチャの 1 サイクルで行う処理を 1 仮想サイクル中の 1 サイクルで実現する場合（Single-cycle model）の記述方法，図 16 (b) は対象アーキテクチャの 1 サイクルで行う処理を 1 仮想サイクル中の複数サイクルで実現する場合（Multi-cycle model）の記述方法をそれぞれ示している．これらは IN_A，IN_B，IN_C の 3 つの入力の和をレジスタ A に代入する記述である．(a)，(b) 共通して，Synchronizer からの動作開始の合図である EN が 1 になると動作するように if 文を用いて記述している．(a) の Single-cycle model による実装では，前述の EN 信号により動作するかどうかを指定している点以外は，通常の HDL 記述と同じである．(b) の Multi-cycle model による実装では，モジュール内部でステートマシンを構成し，対象アーキテクチャでは 1 サイクルで完了する処理を複数サイクルに分割して処理している．ScalableCore システムでは，1 仮想サイクルが対象アーキテクチャの 1 サイクルに対応しているため，図 16 (b) の対象のアーキテク

チャでは 1 サイクルで完了する処理を複数サイクルの処理に分割して記述することが可能である．本実装では 14 サイクルのマルチサイクルプロセッサを用いて，M-Core のシングルサイクルプロセッサの機能を実現した．複数サイクルに分割して実装することで，前述のメモリポート問題など，Unit 上の物理リソースに起因する制約を緩和することが可能である．

もし，異なるアーキテクチャをシミュレーションする場合は，ユーザ層の構成を変更すればよい．その際も，図 16 に示したように，Synchronizer が指定するタイミングで動作するように記述すればよい．

3.3 動作フェーズ

ScalableCore システムは初期化フェーズとシミュレーションフェーズの 2 つの動作フェーズを持つ．初期化フェーズでは，ソフトウェアと連携し，隣接 Unit の確認と各 Unit の ID の設定，ローカルメモリの初期イメージの設定を行う．シミュレーションフェーズでは，各 Unit は前述のローカルバリア同期により，隣接する Unit と同期をとりながらシミュレーションを進める．以下に，それぞれのフェーズで，システムがどのような動作をするかを述べる．

3.3.1 初　期　化

初期化はユーザ層に実装した計算ノードモジュールを駆使して行われる．まず，各 Unit は電源投入完了後，隣接 4 方向に Unit が接続されているかどうかを確認する．各 Unit は電源が投入されると，隣接 4 方向に “Check-flit” と呼ばれる特殊なデータの送信を開始する．Check-flit は連続で 100 万回程度送信される．Check-flit の送信と同時に，それぞれの方向の Check-flit の受信回数を確認する．ある一定の回数（ここでは n とする）受信したとき，その方向の Unit の存在が確認できる．Check-flit の送信回数は，隣接 Unit が存在すると確証を得る受信回数 n に対して，十分大きくとることで電源投入時の遅延による隣接 Unit の検出ミスを防ぐ．隣接 Unit の検出が完了するとデータの送信を停止しスリープ状態に遷移する．

ScalableCore システム Version 1.1 では，図 2 において (1,1) で示される左端かつ上端に位置する Unit を Master Unit とする．また，それ以外の Unit は Worker Unit とする．

隣接 4 方向の Unit の検出が完了した後，すべての Unit は一定時間スリープする．この間に，隣接 4 方向の Unit 接続状況を用いて Master Unit の識別を行う．図 2 において (1,1) に位置する Master Unit のみが，隣接 4 方向のうち上方向と左方向に Unit が接続されていない．このことを利用し，隣接 4 方向のうち上方向と左方向に Unit が接続されていない Unit は，自身を Master Unit として識別する．上方向または左方向に Unit が接続されている Unit は，自身を Worker Unit として識別する．このように隣接 4 方向の Unit 接続状況のみを用いて（特別な通信を必要とせず）Master Unit と Worker Unit の識別を実現している．

識別後，Master Unit は MMC Controller，Memory Initializer を介して，MMC からメモリの初期イメージを読み出す．

各 Unit には ID とシステムの大きさ（Rank:(X,Y) で示される）が設定される．M-Core で動作するソフトウェアの記述では，ID，Rank をともに用いるため，これらの情報の設定は必須である．これらの情報の設定は起動時に動的に行われる．動的に行うことで，シミュレーションシステムのサイズを任意に変更することが可能となる．Master Unit が自ノードのメモリイメージの設定を終えると，ID と Rank の設定を開始する．以下に Rank 取得から ID 設定の間の動作について述べる．**図 17** に Rank 取得・ID 設定時の様子を示す．

各 Unit のルータは起動時から動作しているが，ID は未設定である．各ルータには ID が未設定の状態でパケットを受信すると，そのパケットの宛先を自分の ID として取り込むような機構を実装し．それを利用して Master Unit は，左上の Unit を (1,1) としたときに X 座標が 1 である Unit と Y 座標が 1 である Unit の ID を Master Uni に近い順に設定する（図 17 (a)）．その後，ID を設定した Unit のノードメモリに対して DMA 転送を用いて，メモリイメージを転送する．Master Unit 以外の Core は起動時は停止しておき，各 INCC には DMA 転送サイズが 0 である DMA 転送を受信した場合に Core を起動するような機構を実装しておく．そして Master Unit は，メモリイメージの設定が完了した Unit の Core を転送サイズ 0 の DMA 転送を用いて起動する．起動した Core はソフトウェアに従って動作し，Master Unit に対して，自らの ID が設定されたことに対するレスポンスを DMA 転送を用いて送信する．Master Unit はレスポンスのうち，X 座標，Y 座標それぞれの値が最も大きいものをシステムの Rank として記録する（図 17 (b)）．以上の流れにより，Master Unit はシステムの Rank を知ることが可能である．その後，Master Unit は取得した Rank をもとに全体の ID の設定を行う（図 17 (c)）．次に Master Unit は全 Unit にメモリイメージを DMA 転送を用いて送信した後，サイズ 0 の DMA 転送を用いて各 Unit を起動する．各 Unit が起動時に実行する初期化関数の内部で，Rank が設定されるまでシミュレーションを開始しないように記述しておく．そして最後に Master Unit は全 Unit に Rank を書き込む（図 17 (d)）ことでシミュレーションを開始する．

3.3.2 シミュレーション

シミュレーションフェーズでは実際に各 Unit はプログラムを実行してシミュレーションを進める．計算ノードの各モジュール（Core，INCC，Router）は，1 仮想サイクルの中で

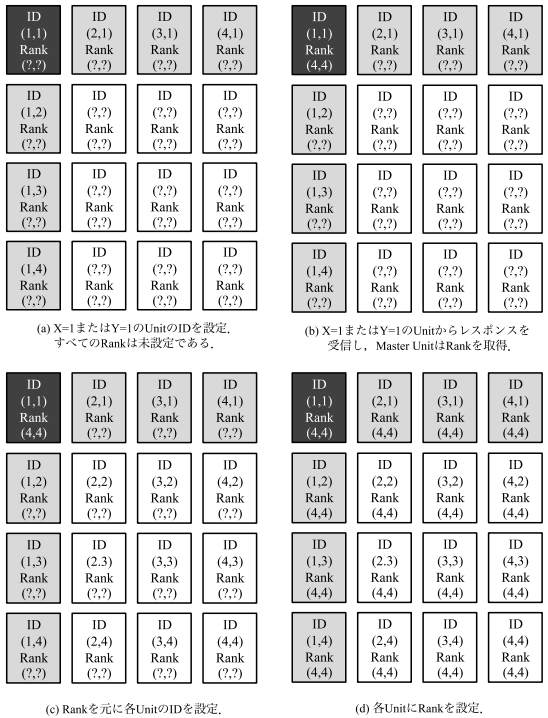

図 17　Rank 取得時および ID の設定時の様子．黒い Unit は Master Unit，灰色の Unit は Rank 取得時に ID が設定される Worker Unit，白い Unit は Master Unit が Rank を取得した後で ID が設定される Worker Unit である

Fig. 17　Behavior on rank acquirement and ID configuration.

図 12 に示すようなタイミングで動作する．仮想サイクル中で各モジュールが動作するタイミングは前述のとおり Synchronizer によって制御される．図中の “Serial Sending” のタイミングで，Serializer が動作し隣接 Unit に対してデータを送信する．送信が完了すると，前述のとおり Synchronizer は隣接 Unit からのデータ受信と内部処理の完了を待機し，隣接 Unit 間で同期をとる．同期が完了すると次の仮想サイクルの処理へと進む．

ローカルバリア同期により，仮想サイクルレベルで動作の一貫性を保証しているため，同じアプリケーションを用いるシミュレーションを行う限りはシミュレーション結果は同じとなる．

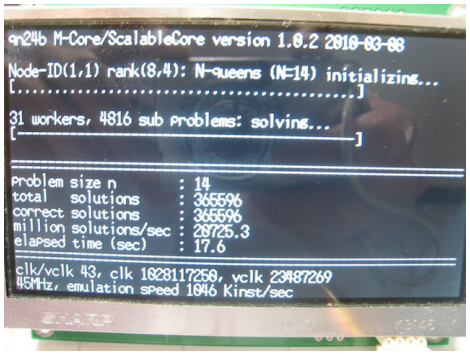

図 18　8 × 4 構成，32 ノードの ScalableCore システム上で N-Queen（N = 14）が動作している様子．ノード (1,1) の出力画面である．画像下の vclk の値がシミュレーションに要した仮想サイクル数である

Fig. 18　N-Queen (N = 14) works on 8 × 4, 32-node ScalableCore system.

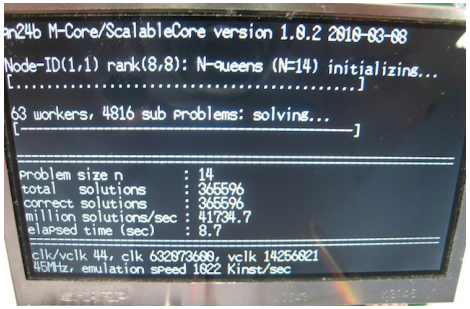

図 19　8 × 8 構成，64 ノードの ScalableCore システム上で N-Queen（N = 14）が動作している様子．ノード (1,1) の出力画面である．画像下の vclk の値がシミュレーションに要した仮想サイクル数である

Fig. 19　N-Queen (N = 14) works on 8 × 8, 64-node ScalableCore system.

図 18 に N-Queen（N = 14）が 8 × 4 構成の 32 ノードの ScalableCore システムで動作している様子を示す．また，**図 19** に，同様のアプリケーションが 8 × 8 構成の 64 ノードの ScalableCore システムで動作している様子を示す．32 ノードの場合に実行サイクル数（図中の vclk：仮想サイクル数）は 23,487,269 サイクルを要していたが，64 ノードの場合は 14,256,021 サイクルに減少している．このようにノード数を変えると，うまく並列化されたアプリケーションであれば実行サイクル数が大きく変化する．

次に，ScalableCore システムにおいて，内部状態を観測する方法について述べる．ScalableCore システム Version 1.1 では，内部状態を観測する 1 つの手段を提供している．具体的には，プログラムの実行サイクル数がメモリマップされており，ロード命令を介してその値を知ることができる．その他の内部状態を観測する場合には，カウンタ回路などを追加したうえで，システムに搭載されているシリアル出力端子（ディスプレイ出力と兼用）を介してそのカウンタの値を出力することで，ユーザは内部状態を知ることができる．また，実行サイクル数と同様に必要な情報をメモリマップしたり，システムに搭載されている LED に出力したりすることで内部状態を観測することができる．

4. 評　　価

本章では前章で実装した ScalableCore システムの評価を行う．評価項目は，シミュレーション速度，消費電力量，Energy-Delay Product（電力量と時間の積），コスト性能比，占有面積，回路合成時間，複雑性，回路構成の変更に要する作業量の 8 つである．

シミュレーション速度，電力量，Energy-Delay Product の評価においては，2009 年 12 月に購入した HP Compaq Business Desktop dc5800[13] において M-Core のシミュレータ SimMc Version 1.0 を動作させた場合との比較を行う．実験環境を以下に示す．OS は Debian GNU/Linux 5.0.3（Linux kernel 2.6.26），SimMc をコンパイルするコンパイラは gcc 4.3.2（最適化オプション：O3），アプリケーションプログラムをコンパイルする MIPS クロスコンパイラは gcc 4.2.4（最適化オプション：O3），プロセッサは Intel®Core™2 Duo E8400 3.00 GHz，メモリは 4 GB 搭載する．価格は 69,800 円であった．

ScalableCore システムでは 4 ノードから 64 ノードの間でノード数を変更して，シミュレーション速度を測定した．なお，実装したシステムは非常に安定して動作しており，数日間の連続動作においてもエラーなく動作している．

4.1 シミュレーション速度

図 20 に，SimMc Version 1.0 および ScalableCore システムそれぞれで Equation Solver Kernel[14] を実行した場合のシミュレーション速度と SimMc に対する ScalableCore システムの速度向上率を示す．グラフの横軸はシミュレーション対象のノード数，縦軸はシミュレーション速度および速度向上率を示す．グラフ中では実測値（4 ノードから 64 ノードの間）を実線・色の濃い棒グラフで，見込み値（64 ノードから 100 ノードの間）を破線・色の薄い棒グラフで示している．ScalableCore システムのシミュレーション速度は動作周波数および 1 仮想サイクルに要する実クロック数に依存する．ScalableCore システム Version 1.1

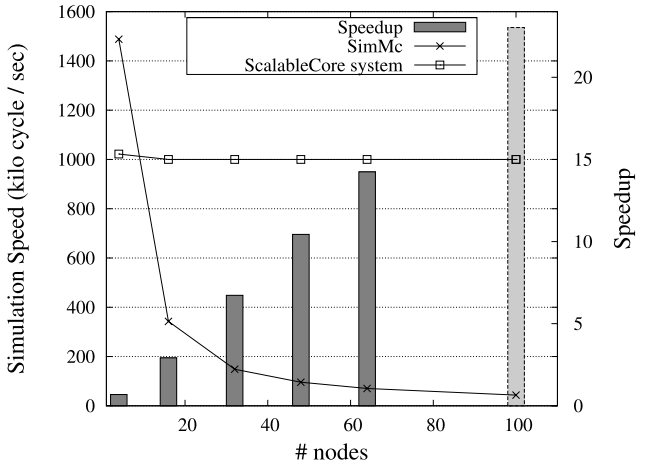

図 20　SimMc Version 1.0 および ScalableCore システムのシミュレーション速度および速度向上率．グラフ中の実線・色の濃い棒グラフは実測値，破線・色の薄い棒グラフは見込み値である
Fig. 20　Simulation speed of SimMc and ScalableCore system. And speedup rate.

はシステム全体が 45 MHz の単一のクロックに同期して動作している．1 仮想サイクルに要するクロック数は 16 ノードから 64 ノードまでの構成で最大 45 サイクルであった．ノード数を増加させた場合，1 仮想サイクルに要するクロック数は増加する可能性がある．しかしながら，16 ノードから 64 ノードの構成において 1 仮想サイクルに要するクロック数は変化しなかったため，100 ノードの場合においても 45 サイクルであるとして評価を行った．

SimMc のシミュレーション速度は 4 ノードのシミュレーション時で 1,487 Kcycle/sec である．そしてシミュレーション対象のノード数が増加するに従い速度は低下し，64 ノードでは 70 Kcycle/sec，100 ノードでは 43 Kcycle/sec で動作する．これらは 4 ノードのシミュレーション時と比較してそれぞれ 1/21 倍，1/34 倍の速度である．一方，ScalableCore システムのシミュレーション速度はほぼ一定で，4 ノードのシミュレーションでは 1,022 Kcycle/sec，それ以上では 1,000 Kcycle/sec で動作する．

64 ノードでのシミュレーションにおいて，ScalableCore システムは SimMc に対して 14.2 倍の高速化を達成した．また，100 ノードでのシミュレーションにおいては約 23 倍の高速化が見込まれる．これにより，ソフトウェアシミュレータの SimMc では約 2 週間を要する 64 ノードの M-Core アーキテクチャの 80G サイクルのシミュレーションを約 1 日で行うこ

とが可能になった．

本実装および比較対象である SimMc における Core は，どちらも 1 サイクルに 1 命令を実行するモデルとなっている．もし，Core のパイプライン化やスーパスカラ化などによりマイクロアーキテクチャを評価する詳細なシミュレーションを行うとすると，ソフトウェアシミュレータの SimMc は大幅にシミュレーション速度が低下すると考えられる．一方ハードウェア実装である ScalableCore システムは，1 仮想サイクル中で Core の動作が占める実サイクル数の割合を変更しなければ，シミュレーション速度は変化しない．そのため，詳細なシミュレーションになるほど，ScalableCore システムが速度面で有利である．

4.2　消費電力量・Energy-Delay Product

図 21 に，各ノード数における，1 キロサイクルのシミュレーションを行った場合に要する平均的なエネルギー（消費電力量）とその相対効率のグラフを示す．横軸はシミュレーション対象のノード数，縦軸はエネルギーと相対効率を示す．**図 22** に，各ノード数における，1 キロサイクルのシミュレーションを行った場合の Energy-Delay Product とその相対性能のグラフを示す．グラフの横軸はシミュレーション対象のノード数，縦軸は Energy-Delay Product と相対効率を示す．評価条件を以下に示す．ベンチマークとしてはシミュレーション速度の評価と同様に Equation Solver Kernel を用いた．消費電力量および Energy-Delay Product を算出するために，ScalableCore システムおよびソフトウェアシミュレータを実行する比較対象の計算機の消費電力を測定した．ScalableCore システムの消費電力はスイッチング電源アダプタと交流 100 V の電源端子の間に電力計を挿入し測定した．比較対象の計算機の電力も同様に，計算機の電源入力と交流 100 V の電源端子の間に電力計を挿入して測定した．比較対象の計算機の消費電力はシミュレーションノード数にかかわらず 84 W であった．ScalableCore システムの消費電力は 64 ノードシミュレーション時で 51 W であった．この値をもとに 1 Unit あたりの消費電力を算出した．そして，これらの値と前節において測定したシミュレーション速度をもとに，1 キロサイクルのシミュレーションを行う場合に各ノード数における ScalableCore システムおよびソフトウェアシミュレータを実行する計算機が要する平均的な消費電力量を算出した．具体的には，次の式より算出した．

(1 キロサイクルのシミュレーションに要するエネルギー) [J]

= ((消費電力) [W]/(シミュレーション速度) [Kcycle/sec]) × 1 [Kcycle]

また，算出したエネルギーおよび測定したシミュレーション速度をもとに，1 キロサイクルのシミュレーションを行う場合に各ノードの ScalableCore システムおよびソフトウェアシミュレータを実行する計算機における平均的な Energy-Delay Product を算出した．具体的

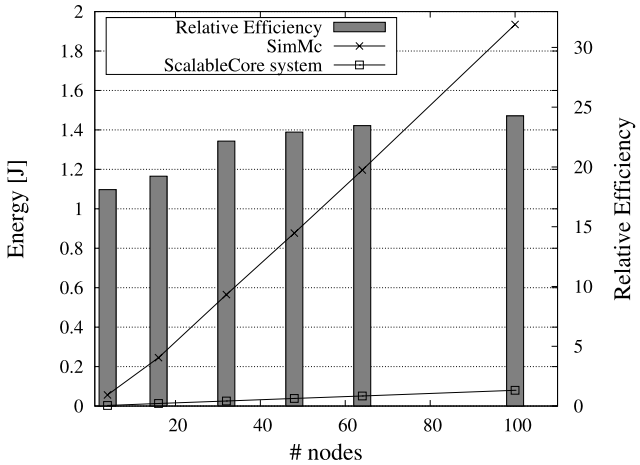

図 21　1 キロサイクルのシミュレーションに要する各ノード数における消費電力量とその相対効率
Fig. 21　Power consumption and its relative efficiency in 1 kilo cycles simulation.

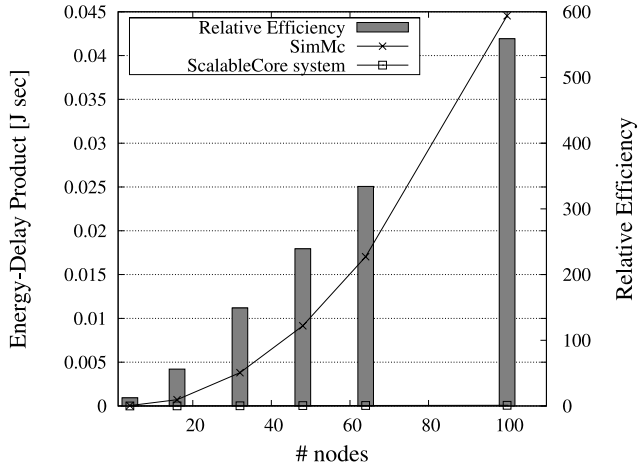

図 22　1 キロサイクルのシミュレーションを行った場合の各ノード数における Energy-Delay Product とその相対効率
Fig. 22　Energy-Delay Product and its relative efficiency in 1 kilo cycles simulation.

には，次の式より算出した．

(1 キロサイクルのシミュレーション時の Energy-Delay Product) [Jsec]
＝(1 キロサイクルのシミュレーションに要するエネルギー) [J]
　× (1 [Kcycle]/(シミュレーション速度) [Kcycle/sec])

まずエネルギーについて述べる．SimMc ではシミュレーションノード数が増加するとシミュレーション速度が低下するため，それに応じて 1 キロサイクルのシミュレーションに要するエネルギーは増加する．64 ノードのシミュレーション時には 4 ノードシミュレーション時と比較して 21 倍のエネルギーを要する．一方，ScalableCore システムにおいては，シミュレーション速度はほぼ一定であるがノード数に応じて消費電力が増加するため，やはりノード数の増加に応じて 1 キロサイクルのシミュレーションに要する消費エネルギーは増加する．64 ノードのシミュレーション時には 4 ノードシミュレーション時と比較して 16 倍のエネルギーを要する．これらの結果から，SimMc に比べ ScalableCore システムの電力効率は，64 ノードのシミュレーション時で約 23 倍優れていることが確認できた．

次に Energy-Delay Product について述べる．SimMc のシミュレーション速度がノード数にほぼ反比例して低下する一方，ScalableCore システムのシミュレーション速度がほぼ一定であることから，ノード数の増加に応じて Energy-Delay Product の相対性能は向上し，64 ノードのシミュレーション時に SimMc 比の約 334 倍の効率を達成することができた．

このように ScalableCore システムは電力面で優れているといえる．

4.3　コスト性能比 · 占有面積

今回の実装の場合，基板代・部品代・業者による実装代を含めた価格は，ScalableCore Unit が約 4,000 円，ScalableCore Board が約 500 円であった．$8 \times 8 = 64$ ノードのシステムのコストは約 301,500 円となる．一方，比較対象の計算機の価格は 69,800 円であるので，64 ノードをシミュレーションする ScalableCore システムは 4.3 倍高価である．しかし性能は 64 ノードの ScalableCore システムは 14.2 倍優れているため，コスト性能比で考えると $14.2/4.3 = 3.3$ 倍，ScalableCore システムは優れているといえる．このように我々が構築した ScalableCore システムは，コスト性能比において優れている．

今回の実装の場合，基板のサイズが 71 mm × 47 mm である．したがって，$m \times n$ ノードのシステムは $(71 + 73 \times n) \times (47 + 59 \times m)\,\mathrm{mm}^2$ を占有する．たとえば 14×14 ノードのシステムと A0 用紙がほぼ同等の面積となる．今回実装した ScalableCore システムは，ScalableCore Board 裏面に実装した磁石により，ホワイトボードなどに貼り付けることができる．幅 900 mm のホワイトボードがあれば，$8 \times 8 = 64$ ノードを貼り付けることが可能である．このように ScalableCore は専有面積はそれほど大きくない．

4.4　回路合成時間

ScalableCore システム Version 1.1 の回路合成について述べる．ScalableCore システム Version 1.1 の各 FPGA に書き込む回路情報（bit ファイル）の合成に要する時間を測定した．測定には，Intel Core2 Quad Q6700 2.66 GHz，メモリ 2 GB，Windows 7 Professional という一般的な構成の計算機を用いた．合成ツールには Xilinx ISE 11.5 を用いた．測定の結果，合成時間はおよそ 10 分であった．ScalableCore システムでは，複数 Unit の回路構成が同じ場合，同一の bit ファイルがそのまま利用できる．そのため，M-Core アーキテクチャをシミュレーションする場合，シミュレーション対象のコア数が増加しても回路合成時間が増加しないというメリットがある．このことから，ScalableCore システムの回路合成時間は大きな問題とならない．

4.5　複　雑　性

ScalableCore システム Version 1.1 の複雑性について，まずローカルメモリ周りの再現と同期処理の 2 点について述べる．ScalableCore システム Version 1.1 では，各ノードが持つローカルメモリを各 Unit が搭載する SRAM を利用してシミュレートする．これにより，各ノードのローカルメモリに関する処理を Unit ごとに独立にシミュレーションすることができ，ローカルメモリに関わる複雑性を軽減する．また ScalableCore システム Version 1.1 は同期処理の実装はシンプルであり，その複雑性は低い．各 Unit 間はサイクルレベルで正確なシミュレーションを行うための同期機構を搭載している．ScalableCore システム Version 1.1 では，各ノードがメッシュネットワークで接続されている M-Core アーキテクチャをシミュレートするため，実際に各 Unit が処理やデータ転送の完了を待つ必要があるのは，隣接する 4 ノードのみである．

次に，ScalableCore システムのシミュレーションターゲットの回路記述に関する複雑性について述べる．ScalableCore システム Version 1.1 では，多ポートメモリなど，本質的に FPGA に実装が難しいものを仮想サイクルという仕組みを用いて実現している．ターゲットシステムにおいて複数のユニットから同一のユニットに対して，1 サイクルの間に同時に行われるアクセスを，ScalableCore システムでは 1 仮想サイクル内の複数のクロックサイクルに分割して取り扱う．各デバイスからのアクセスを一貫性が維持できるように分割し，仮想サイクル内の各クロックサイクルにスケジューリングする．これにより，ソフトウェアシミュレータのように機能ユニットを記述することができる．このように，ScalableCore システムは，複数 SRAM を用いたメモリユニットの独立性，シンプルな同期機構，仮想サイ

クルの導入によるソフトウェアシミュレータのような機能ユニット記述など，その複雑性は高くないといえる．

4.6 回路構成の変更に要する作業量

ScalableCore システム Version 1.1 では，各 Unit が搭載するコンフィギュレーション ROM に各 FPGA の回路情報（bit ファイル）を書き込む必要がある．この書き込みは各 Unit が持つ書き込み端子（JTAG 端子）経由で行う．

64 個の Unit のコンフィギュレーション ROM に 1 つ 1 つ順番に書き込む（1 つのコンフィギュレーションが完了したら，ケーブルを差し替えて次のコンフィギュレーションを行う）ような場合に要する時間は，書き込み用ケーブルの差し替えの時間を含めて，おおよそ 15 分程度である．一方で，簡単なデバッグでは 4 個程度の Unit のコンフィギュレーション ROM に書き込んで確認することが多い．この場合に要する時間はおおよそ 1 分程度であるため，多くの場合で書き込み時間は問題にはならない．

しかし，数百個の Unit のコンフィギュレーション ROM への書き込みにはより長い時間を要する．このため，現在の方法は効率的ではない．書き込み方式の効率化は今後の課題である．

5. 関 連 研 究

コンピュータアーキテクチャの研究やソフトウェアの性能向上のための解析の道具としてのプロセッサのシミュレーション環境を構築する研究は数多くある．近年では，コア数増加の流れにともない，マルチコア/メニーコアを対象としたシミュレーション環境の研究が多く行われている．マルチコア/メニーコア上の複数のコアを用いて効率良くプログラムを動作させるためには，コンピュータアーキテクチャだけではなく，その上で動作するミドルウェアやアプリケーションを含めた研究開発が必要である．そのため，特にマルチコア/メニーコア向けのフルシステムシミュレーション環境の構築が今後ますます重要になってくる．

ソフトウェアシミュレーションのかわりに，柔軟に構成を変更可能な FPGA を用いることで高速なシミュレーションの実現を目標とする研究には，RAMP [15] をはじめとして，bluespec [16]，FAST [17] や，ATLAS [18]，ProtoFlex [19] などがある．bluespec [16] では，System Verilog をベースとした高位合成により，柔軟なハードウェア検証を可能としている．FAST は，4 個の Processor Tile を持つシミュレーションプラットフォームであり，チップマルチプロセッサにおけるメモリシステムの研究に適している．RAMP は，ソフトウェアとハードウェアの両面からマルチコア/メニーコアの研究を加速するためのプラットフォームの研究を推進している．ATLAS では，トランザクショナルメモリをサポートした CMP についての実用的なシミュレーション環境を提案している．これは，ソフトウェアシミュレーションである TASSEL と比較して 100 倍の高速化を実現している．また，ProtoFlex では，OS の動作をも模倣するフルシステムシミュレーションの 38 倍の高速化を実現している．ATLAS および ProtoFlex では，機能レベルのシミュレーションをソフトウェアに任せることで，複雑な実装を回避している．しかし一方で，ソフトウェアとのハイブリッドシミュレーション環境では，現実的なハードウェアリソース量では実現しえないような不当な仮定をおいてしまう可能性がある．そのため，ハードウェアリソース量を評価するために，すべてをハードウェアとして実現可能な RTL ベースのシミュレーション環境と，それを現実的な時間で運用するための FPGA によるプラットフォームがより好ましい．

これらシステムの多くは大規模 FPGA を用いており，小規模な FPGA を用いてあらかじめ拡張可能な形でシステムを構成する点で ScalableCore は異なる．我々が実装した ScalableCore システム Version 1.1 は柔軟性を得るために仮想サイクルの概念を導入した．1 仮想サイクル中で複数の実サイクルで 1 機能を再現することが可能であるため，柔軟性は高くなる反面，現実的なハードウェアとはいい難い実装となる場合がある．ハードウェア量の評価や，プロセッサのパイプライン化，スーパスカラ化などの，より厳密なモデルによる動作検証を行う場合には，慎重に記述する必要がある．

6. ま と め

我々は，ハードウェアによる高速プロトタイピングシステム構築手法である ScalableCore を提案している．小容量の FPGA を搭載したシミュレーションノード “ScalableCore Unit” を，4 方向に拡張可能な接続インタフェース “ScalableCore Board” で接続することで，スケーラビリティを満たしながら，柔軟かつ高速なシミュレーション環境の実現を目指す．本稿では，メニーコアアーキテクチャM-Core をシミュレーション対象アーキテクチャとしたときの，小容量の FPGA を多数接続する提案手法の得失について議論し，回路合成時間，1 シミュレーションユニットあたりのコストにおいて提案手法が有利であることを明らかにした．そして実際に構築した M-Core アーキテクチャの計算ノード群のシミュレーション環境である ScalableCore システム Version 1.1 の実装について述べた．ScalableCore システムの動作を目的としたファーストステップの実装であるため，高度な高速化手法は適用していない．しかしながら，実装したシステムは M-Core アーキテクチャのソフトウェアシミュレータ SimMc に比べて，64 ノードのシミュレーションでは 14.2 倍の高速化を達成し

た．実装したシステムは非常に安定して動作しており，数日間の連続動作においてもエラーなく動作している．対象アーキテクチャにおける 1 サイクルに相当する “仮想サイクル” の概念を導入することで，ある程度の柔軟性を得ることが可能となった．また，仮想サイクルごとに同期を行うことにより，複数の FPGA に分散したシステム構成が可能となった．

謝辞　本研究の一部は，科学技術振興機構・戦略的創造研究推進事業（CREST）の「アーキテクチャと形式的検証の協調による超ディペンダブル VLSI」の支援による．またシステムの開発にご協力いただいた東京工業大学の姜軒さんに深く感謝いたします．

参 考 文 献

1) 高前田伸也，渡邉伸平，吉瀬謙二：メニーコアプロセッサの高速プロトタイピングシステム ScalableCore の提案，情報処理学会第 71 回全国大会 (2009).
2) 高前田伸也，渡邉伸平，姜　軒，藤枝直輝，植原　昂，三好健文，吉瀬謙二：メニーコアアーキテクチャ研究のためのスケーラブルな HW 評価環境 ScalableCore システム，情報処理学会研究報告 2009-ARC-185 (2009).
3) 植原　昂，佐藤真平，高前田伸也，渡邉伸平，吉瀬謙二：メニーコアプロセッサの HW/SW 研究開発を加速する実用的な基盤環境，先進的計算基盤システムシンポジウム SACSIS2009 論文集 (2009).
4) Uehara, K., Sato, S., Miyoshi, T. and Kise, K.: A Study of an Infrastructure for Research and Development of Many-Core Processors, *Workshop on Ultra Performance and Dependable Acceleration Systems Held in Conjunction with PDCAT'09*, pp.414–419 (2009).
5) 植原　昂，佐藤真平，佐野伸太郎，吉瀬謙二：メニーコアプロセッサの研究・教育を支援する実用的な基盤環境 M-Core，情報処理学会研究報告 2009-ARC-188 (2010).
6) 渡邉伸平，藤枝直輝，若杉祐太，高前田伸也，森　洋介，吉瀬謙二：MIPS システムシミュレータ SimMips を活用した組込みシステム開発の検討，情報処理学会研究報告，Vol.2008, No.116, pp.23–28 (2008).
7) Digi-Key Corp. http://jp.digikey.com/
8) インテグラル電子カラー液晶ユニット ITC-2432-035F. http://www.intgrl.co.jp/product/lcd/color/command/ITC-2432-035F/index.html
9) プリント基板パターンエディタ PCBE. http://www.riric.jp/electronics/design/editor/pcbe.html/
10) 秋月電子通商スイッチングアダプター 5V/3A. http://akizukidenshi.com/catalog/g/gM-02191/
11) Data recovery, *Xilinx Application Note*, Vol.224 (2005).
12) Using Digital Clock Managers (DCMs) in Spartan-3 FPGAs, *Xilinx Application Note*, Vol.462 (2006).
13) HP Compaq Business Desktop dc5800. http://h50146.www5.hp.com/products/desktops/old/dc5800sf_ct/core_duo_model.html
14) Sing, J.P., Culler, D.E. and Gupta, A.: *Parallel Computer Architecture: A Hardware/Software Approach*, Morgan Kaufmann (1998).
15) RAMP: Research Accelerator for Multiple Processors. http://ramp.eecs.berkeley.edu/
16) Nikhil, R.: Bluespec System Verilog: Efficient, correct RTL from high level specifications, *Formal Methods and Models for Co-Design, 2004. MEMOCODE '04. Proc. 2nd ACM and IEEE International Conference on*, pp.69–70 (2004).
17) Davis, J.D., Richardson, S.E., Charitsis, C. and Olukotun, K.: A chip prototyping substrate: The flexible architecture for simulation and testing (FAST), *SIGARCH Comput. Archit. News*, Vol.33, No.4, pp.34–43 (2005).
18) Wee, S., Casper, J., Njoroge, N., Tesylar, Y., Ge, D., Kozyrakis, C. and Olukotun, K.: A practical FPGA-based framework for novel CMP research, *FPGA '07: Proc. 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, New York, NY, USA, ACM, pp.116–125 (2007).
19) Chung, E.S., Papamichael, M.K., Nurvitadhi, E., Hoe, J.C., Mai, K. and Falsafi, B.: ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs, *ACM Trans. Reconfigurable Technol. Syst.*, Vol.2, No.2, pp.1–32 (2009).

(平成 22 年 5 月 7 日受付)
(平成 22 年 11 月 30 日採録)

高前田伸也（学生会員）
2009 年東京工業大学工学部情報工学科卒業．現在，同大学大学院情報理工学研究科修士課程在学中．メニーコアプロセッサおよび FPGA システムに関する研究に従事．

佐藤　真平（学生会員）

2007 年東京工業大学工学部情報工学科卒業．2009 年同大学大学院情報理工学研究科修士課程修了．現在，同大学院情報理工学研究科博士後期課程在学中．2010 年より日本学術振興会特別研究員 DC2．メニーコアプロセッサにおけるネットワークオンチップとディペンダビリティに関する研究に従事．

藤枝　直輝（学生会員）

2008 年東京工業大学工学部情報工学科卒業．2010 年同大学大学院情報理工学研究科修士課程修了．現在，同大学院博士後期課程在学中．MieruPC 株式会社代表取締役．プロセッサアーキテクチャ，計算機システムに関する研究に従事．

三好　健文（正会員）

2003 年東京工業大学工学部電気電子工学科卒業，2005 年同大学大学院電子機能システム専攻修士課程修了，2007 年同大学院物理情報システム専攻博士課程修了．博士（工学）．2010 年より電気通信大学大学院情報システム学研究科助教．現在に至る．メニーコアプロセッサ，HW/SW 協調設計，リコンフィギュラブルシステムに関する研究に従事．IEEE，ACM，電子情報通信学会各会員．

吉瀬　謙二（正会員）

1995 年名古屋大学工学部電子工学科卒業．2000 年東京大学大学院情報工学専攻博士課程修了．博士（工学）．同年電気通信大学大学院情報システム学研究科助手．2006 年東京工業大学大学院情報理工学研究科講師．計算機アーキテクチャ，メニーコアプロセッサアーキテクチャ，並列処理に関する研究に従事．電子情報通信学会，IEEE-CS，ACM 各会員．