社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－ＳＬＤＭ－129／2007－ＥＭＢ－4（13）
2007／3／16

HW/SWコデザインにおける共有アドレス空間を介した統一的なモジュール間インタフェースの実現法

山脇 彰† 岩根 雅彦†

† 九州工業大学 工学部 〒804-8550 福岡県北九州市戸畑区仙水町 1-1
E-mail: †{yama,iwane}@ecs.kyutech.ac.jp

あらまし システムレベルデザインにおける抽象的なモジュール間インタフェースは，HW/SW コデザインシステムによって，HW では専用の入出力制御回路に変換される．ただし，メモリアクセスパタンや必要なデータ量が異なる処理ごとに最適な入出力制御回路を生成することは難しい．さらに，メモリレイテンシの隠蔽に対する取り組みはコデザインシステムではあまり考慮されていない．そこで、コデザインシステムのハードウェアモジュールとしてセミプログラマブル再構成可能プロセッサ (SPRCP) の導入を提案する．SPRCP はメモリアクセス用簡易プロセッサ (LSU) と，ハードウェア構成部 (EXU) がレジスタを挟んだ構成を持つ．EXU にとってデータ入出力は単なるレジスタ読み書きであり，柔軟なメモリアクセスは LSU がメモリアクセスプログラムを実行することによって提供される．LSU と EXU は並列に動作するため，処理とメモリアクセスをオーバラップできる．本稿を通して，メモリアクセスを隠蔽できる統一的な SW-HW 間，HW-HW 間インタフェースを，SPRCP が提供し得ることを示す．

キーワード コデザイン，インタフェース，再構成可能プロセッサ，メモリ，ハードウェア設計法

A Hardware Design Method Using Semi-Programmable Reconfigurable Processor

Akira YAMAWAKI† and Masahiko IWANE†

† Faculty of Engineering, Kyushu Institute of Technology
Sensui 1-1, Tobata-ku, Kitakyushu-shi, Fukuoka-ken, 805-8440 Japan
E-mail: †{yama,iwane}@ecs.kyutech.ac.jp

Abstract On a HW/SW co-design system, generating HW-HW and SW-SW interfaces is important to provide fair performance and hardware size. However, it is difficult to provide each process a specific interface circuit. Also, memory access latency becomes larger relative to the execution time reduced. This paper proposes introducing Semi-Programmable ReConfigurable Processor (SPRCP) to a HW/SW co-design system as a frame work of the hardware module. The SPRCP consists of a load/store unit and a hardware unit. The farmer is a tiny processor for memory access and the latter is a reconfigurable hardware unit. The register file is lies between them and provides the hardware unit an easy systematic interface. Instead of hardware unit, the load/store unit performs the flexible memory access and overlaps the memory accesses and hardware execution to hide memory access latency.

Key words codesign, interface, reconfigurable processor, memory, hardware design

1. まえがき

携帯電話，携帯情報端末 (PDA)，および，デジタル家電等の組込みシステムに対する高機能化，小型化，および，低消費電力化の要求は一層高まっている．それらの要求に対して，半導体集積度の向上を背景に，多くの機能を 1 チップ化したシステムオンチップ (SOC) が広く使用されるに至った．

大規模化する SOC に対して，製品のライフサイクルが短い組込みシステムでは，設計開発期間とコストの削減が大きな課題となっており，従来から，設計の抽象度を高める高位合成技術 [1]〜[3]，設計資産 (IP) の再利用 [4] や SoC のマルチプロセッサ化 [5] 等の取り組みが行われている．

その中で，ソフトウェア (SW) とハードウェア (HW) の設計を同時に進行する HW/SW コデザイン技術が注目されてい

る．概念自体は古くから存在するが[6]，近年，計算機性能の向上を背景に，比較的大規模なシステムを合成できるようになった[1]～[3]．高抽象度レベルにおいて，ソフトウェアとハードウェアは抽象化されたインタフェースを介して協調する．コデザインシステムの設計入力は多くがC言語であり，SW-HWとHW-HW間の通信はライブラリ関数によってモデル化される．通信インターフェースに関して，最終的に，ハードウェアには専用の入出力制御回路が，ソフトウェアにはその回路を駆動するためのデバイスドライバ(メモリマップドIOがベース)が挿入される[1],[3],[7]．

そのようなコデザインシステムにおいてハードウェアを生成する際に，以下のような課題が考えられる．通常，処理ごとにメモリのアクセスパターンは異なる．例えば，暗号処理や音声信号処理では1次元配列に対するストリームアクセスであるが，画像処理では2次元配列へのアクセス，ソート等はランダムアクセスになる．個々の場合に応じて最適な入出力制御回路を生成するのは難しいと考えられる．さらに，ハードウェアによる処理の高速化に対して，メモリアクセスレイテンシの隠蔽は重要であるが，従来のコデザインシステム[1]～[3]ではあまり考慮されていない．

そこで，HW/SWコデザインシステムにおけるハードウェアモジュールのフレームワークとしてセミプログラマブル再構成可能プロセッサ(SPRCP)[8],[9]の導入を提案する．SPRCPは，メモリアクセスを行う簡易プロセッサのロードストアユニット(LSU)と，ハードウェア構成部の実行ユニット(EXU)，および，LSUとEXU間のレジスタファイルからなる．EXUに実装するハードウェアはデータ入出力を単純なレジスタ読み書きのみで実現する．LSUは実行するプログラムに応じた様々なアクセスパターンで，メモリ上のデータをレジスタファイルに供給したり，レジスタファイルからメモリへ書き戻したりする．つまり，処理ごとに最適なメモリアクセスはLSUのプログラムをチューニングすることによって行い，EXUに実装されるハードウェアモジュールに対してはレジスタベースの統一的なインタフェースが提供される．さらに，LSUとEXUは並列に動作するので，メモリアクセスと演算処理のオーバーラップによってメモリアクセスレイテンシの隠蔽が可能となる．SPRCPはLSUによってDMAの機能を持つ．それゆえ，SPRCP群を共有アドレス空間にマッピングすることで，SW-HW，HW-HW間の通信は共有変数への読み書きをベースとしたプログラムによって統一的に扱われる．

以降，2章でSPRCPを導入したHW/SWコデザインの概要を示し，3章でSPRCPについて説明する．次に，4章では，簡単な例題を用いて実施したSPRCPへのマッピングに関するケーススタディについて述べる．最後に5章でむすぶ．

2. HW/SW協調設計

HW/SWコデザインでは，CやC++などの高級言語によるシステムレベルの設計データが，やがて，プロセッサが実行するマシン語と，ハードウェア化されるHDLプログラムに変換される．本節では，このフローにおいて，システムレベルデザインのハードウェア部がSPRCPにマッピングされるまでの過程を明確にする．

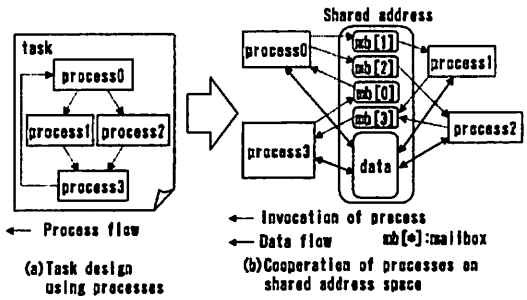

図1 HW/SWコデザインの概念
Fig.1 Concept of HW/SW design

システムレベル設計では，対象処理を実現するためにより小さな機能群を組み合わせてシステムを設計する．ここでは，図1(a)に示すように，対象処理をタスク，個々の機能をプロセスと呼ぶ．各プロセスは高級言語によって統一的に記述され，各プロセス間の通信はライブラリ等の抽象化されたインタフェースによって表現される．

プロセス間通信は，基本的に，共有アドレス上の共有変数を介して行われる．その概念を図1(b)に示す．ここでは，プロセスの起動や実行に必要なパラメータを受け渡すデータ構造をメールボックス(図中のmb)と呼ぶ．各プロセスが使用するデータは，メールボックスと同様に，共有アドレス上にマッピングされる．

図1(b)の例では，以下のような流れでタスクが実現される．

(1) プロセス0がデータを加工し，プロセス1とプロセス2を，それぞれのメールボックス(mb[1], mb[2])を介して起動する．同時に，実行に必要なパラメタ(入出力データのアドレス，データサイズ，プロセス3のメールボックスアドレス等)もメールボックスに添付される．

(2) 起動されたプロセス1とプロセス2は並列に動作し，各々が機能に応じてデータを加工する．終了したプロセス1とプロセス2は，プロセス3のメールボックス(mb[3])にプロセス3の実行に必要なパラメタを書き込み起動する．

プロセスの起動はメールボックス上のフラグチェックによって行われる．起動条件が，図1(b)のプロセス1，2とプロセス3のように，複数プロセスの合流ならば，起動要求元に対応する全フラグが1になった場合に起動要求先が起動される．

その後，図2に示すように，ソフトウェアプロセスはCPUへ，ハードウェアプロセスはSPRCPへと割り当てられる．CPUのメールボックスは組込みメモリ等の実メモリに置かれる．ハードウェアプロセスのメールボックスはSPRCP内に置かれる．

次章では，ハードウェアプロセスとそのメールボックスが割りあてられたSPRCPについて説明する．

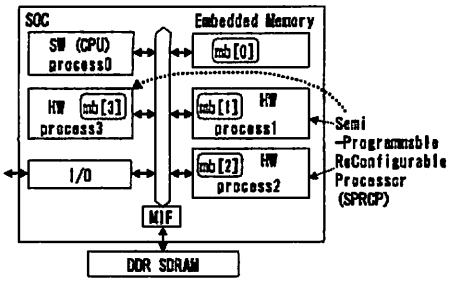

図 2 HW/SW の SOC への割り当て
Fig. 2 Assigning HW and SW to an SOC

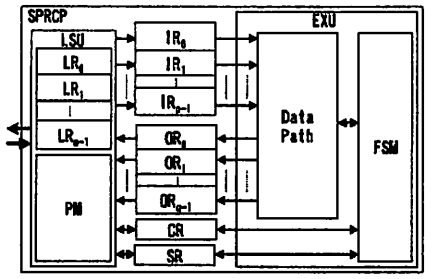

図 3 SPRCP の構成
Fig. 3 Organization of SPRCP

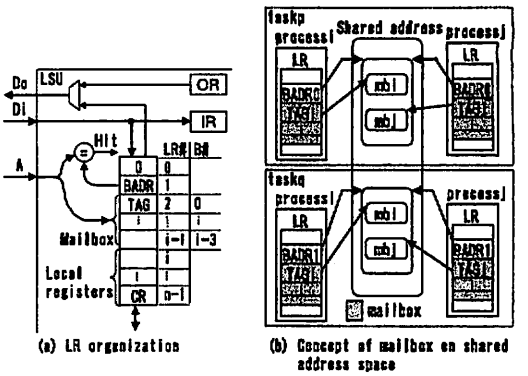

図 4 LR とメールボックスの概要
Fig. 4 Overview of LR and mailbox

3. セミプログラマブル再構成可能プロセッサ

3.1 概　　要

SPRCP の構成を図 3 に示す．SPRCP は，主に，メモリアクセスを実行する専用プロセッサ Load/Store Unit (LSU) と，ハードウェアが構成される演算処理部 EXecution Unit (EXU)，EXU に対してメモリへの透過的なインタフェースを提供する入出力レジスタ (IR, OR) からなる．また，LSU と EXU 間で動作を協調させるための同期レジスタ (SR) と制御レジスタ (CR) がある．SR はメモリ-レジスタ間と EXU-レジスタ間でのデータ転送の同期に使用される．CR は，LSU と EXU が相互で演算した結果を条件とした分岐に使用される．

EXU には，データパスと，そのデータパス，および，LSU 間の協調動作を制御する FSM が実装される．LSU はプログラムメモリ (PM) に格納された命令を実行する．命令にはアドレス計算のための最低限の演算命令とメモリにアクセスするためのロード/ストア命令に加え，いくつかの SPRCP に特化した命令がある (3.2 で説明)．これにより，SPRCP はストリームアクセスだけではなく，任意のランダムアクセスにも柔軟に対応できる．また，LSU は，EXU とは独立して，メモリと IR, OR 間でデータを転送し，EXU は，LSU と独立に，IR, OR に対してデータを読み書きする．そのため，EXU の実行と LSU のメモリアクセスをオーバラップできる．

LSU は専用のレジスタファイル (図 3 の LR) を持ち，図 4 (a) に示すように，いくつかの部分に分けられる．0 番レジスタは MIPS アーキテクチャと同様に，常に 0 を供給するレジスタである．1 番レジスタは，アドレスデコードに使用され，SPRCP のメールボックスが存在するベースレジスタが格納される．LR1 の内容はハードウェア設計時に初期値としても設定でき，LSU のプログラムによっても動的に設定可能である．2 番レジスタから i-1 番レジスタまでがメールボックスとなり，残りは汎用のレジスタとして利用される．メールボックスの 0 番エントリ (LR2) は，SPRCP を起動するための情報を示すタグが格納される．タグの仕様はシステム設計者によって策定される．また，LR の最終レジスタには CR が割り当てられる．

図 4 (b) に LR を用いたメールボックスの概念を示す．あるタスクに属する各プロセスのメールボックスは，共有アドレス上で，各 SPRCP の LR1 が指す領域 (メールボックス領域) に存在する．メールボックス領域に存在する個々のメールボックスがタグによって指定される．タスクごとにベースアドレスを管理すれば，共有アドレス上で，メールボックスをタスクごとにグループ化できる．

3.2 SPRCP 命令

LSU はアドレス計算を実施する演算命令と，条件分岐命令，および，LR とメモリ間のロードストア命令を持つ．これらは一般的な RISC プロセッサと同等な命令であるため，ここでの説明は割愛する．

それらの命令に加えて，LSU と IR, OR 間のデータ転送を実施するための同期付きロード/ストア命令，EXU の演算結果を条件とした分岐命令を持つ (図 5)．各命令は，同期フィールド (sc) を持ち，これによって LSU と EXU が実施する同期の種類が指定される (同期は 3.3 で説明)．命令に同期を付加することによって，命令数と同期オーバヘッドの削減が図られる．

同期付き NOP 命令は，同期を取るためだけに使用される．同期付きロード命令 (LDWS) はメモリから IR に指定数分 (n) のワードを読み出し，同期付きストア命令 (STWS) は OR からメモリへ指定数分 (n) のワードを書き込む．同期付き条件分岐命令 (BCRWS) は，CR のビットパタンと命令の cc フィールドのビットパタンとを比較して，分岐を実施する．LDWS と STWS は，それぞれ，IR と OR を LR を介して間接的にアクセスする．これによって，IR と OR をリング状にアクセスで

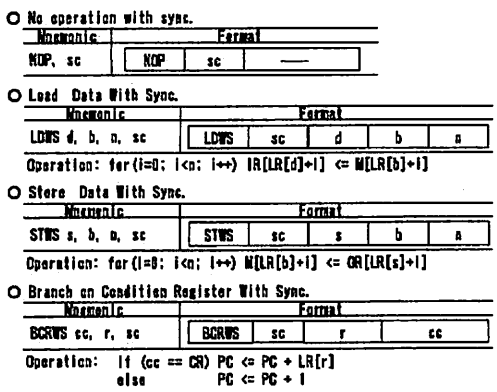

図 5 LSU 命令

Fig. 5 LSU instructions

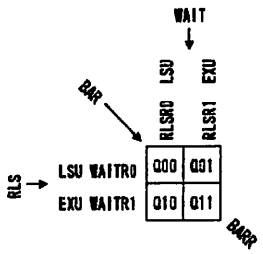

図 6 軽量同期機構

Fig. 6 Light-weight Synchronization Mechanism

きるため，IR と OR 上でダブルバッファを容易に構成できる．

3.3 同期機構

LSU と EXU 間の同期は図 3 中の SR (同期レジスタ) を介して実施される．同期には，リリース同期 (RLS)，ウェイト同期 (WAIT)，及び，バリア同期 (BAR) があり，図 6 に示す軽量同期機構によってそれらの機能が提供される．SR は 3 つの同期ごとに RLSR，WAITR，BARR に分かれる．軽量同期機構は 3 つの同期レジスタを構成する 2 × 2 のフリップフロップ (RLSR は列方向，WAITR は行方向，および，BARR は対角線方向) と制御回路からなる．さらに，RLSR と WAITR は LSU 用 (RLSR0，WAITR0) と EXU 用 (RLSR1，WAITR1) に分かれる．各レジスタともに 0 番ビットが LSU，1 番ビットが EXU に対応する．軽量同期機構に対して発行される同期の種類は，LSU では図 5 の sc，EXU ではアクセスされたレジスタによって識別される．同期が発行されると，同期の種類と各 SR のビットパタンに応じて発行元に同期待ちかどうかが出力される (同期待ちでは LSU はストールされ，EXU は待ち状態となる)．

RLS では，RLSR の起動先ビットが 1 ならば，その発行元は待たされる (これは，データがまだ消費されていないことを示し，データの逆依存，出力依存を満たすためである)．起動先のビットが 0 だった，もしくは，0 になった場合，LSU の命令または EXU の演算が完了した後に RLSR の当該ビットが 1 にセットされる (IR，または，OR 上に有効なデータが存在することを相手に通知する)．

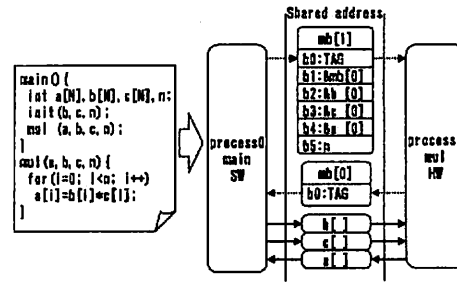

図 7 マッピング例：HW/SW 分割とメールボックスの設定

Fig. 7 Example of Mapping to the SPRCP; Dividing HW and SW

WAIT では，WAITR の自身に対応するビットが 0 ならば，その発行元は待たされる (まだ，データが生産されておらず，データの真依存を満たすためである)．WAITR 上の自身のビットが 1 だった，もしくは，1 になった場合 (有効なデータが IR，OR 上に存在する場合)，WAITR の当該ビットを 0 にリセットして，LSU の命令，または，EXU の演算が実行される．

BAR では，BARR の全ビットが 1 でなければ，発行元に対応するビットが 1 になり，発行元は待たされる．全ビットが 1 になったら，全ビットが 0 にリセットされ，待たされていた発行元はバリア同期を抜ける．

4. SPRCP へのマッピング

4.1 基本手法

本章では，簡単な例題を通して，如何に LSU のプログラムと EXU 上のハードウェアが記述し得るかを示す．以降では，擬似コードを用いて LSU のプログラムと EXU の振る舞いを説明する．図 7 にその例題を示す．対象となるタスクは配列 b，c を初期化し，両者を掛け合わせて配列 a に格納する．そして，main 関数がソフトウェアプロセス (プロセス 0)，掛け算の関数 mul がハードウェアプロセス (プロセス 1) として分割される．プロセス 1 用のメールボックス (mb[1]) には，起動用タグ，プロセス 0 のメールボックス (mb[0]) のアドレス，続いて，配列 b，c，a のアドレスと，最後に，配列のサイズ (n) が割り当てられる．プロセス 0 のメールボックスには mul の完了を知らせるタグのみが割り当てられる．

ソフトウェアプロセスとハードウェアプロセスの分割，および，メールボックスの割り当てが完了したら，各プロセスの内容をより具体的に記述していく．その結果を図 8 に示す．ソフトウェアにおける mul は mb[1] の設定と (4〜9 行目)，mb[0] を用いた mul の完了検出コードに置き換わる (10〜11 行目) (注1)．

LSU プログラムでは，メールボックスを構造体 mb と表し，その各フィールドを mb.b* として記述する．また同期付きロード命令と同期付きストア命令をマクロとして関数の形で表現す

(注1)：例題には無いが，9 行目と 10 行目の間にその他の有益な処理を挿入することができる．

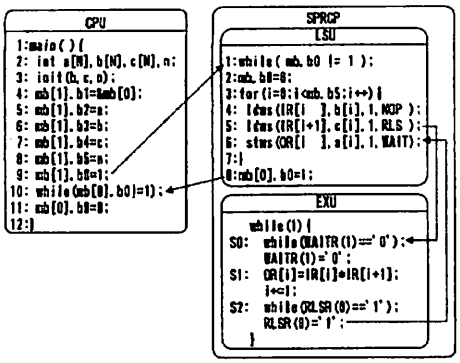

図 8 マッピング例：CPU，LSU，および，EXU のプログラミング

Fig. 8 Example of Mapping to the SPRCP; CPU program, LSU program and EXU behavior

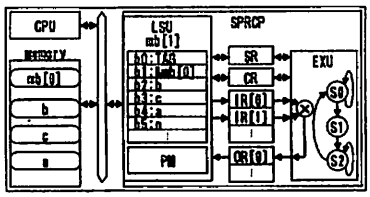

図 9 マッピング例：SOC へのマッピング

Fig. 9 Example of Mapping to the SPRCP; SOC assignment

る．まず，LSU は mb.b0 (タグ) 上で条件が満たされるまでスピンする (1 行目)．その後，配列 b を IR へ読み出し (4 行目)，配列 c を IR へ読み出すと同時に RLS によって EXU を起動する (5 行目)．その後，EXU の乗算の完了を WAIT で待ち，その完了後に配列 a へ OR の内容を書き出す (6 行目)．これを配列サイズ分繰り返し (3〜7 行目)，その後，mb[0].b0 (タグ) に完了の通知を書き込む (8 行目)．

EXU では，振る舞いを表す擬似コードに対応する状態を S* として表す．EXU は S0 において，WAITR の自身のビット (1 番ビット) が 1 になるまで待ち，1 になったら (つまり，LSU によって起動されたら)，WAITR の自身のビットをリセットして次状態に遷移する．そして，S1 で，配列 b と c の要素を IR から読み出し，掛け合わせてから OR に格納する．S2 では，RLSR の LSU 側のビット (0 番ビット) を調べ，1 であった (つまり，以前のデータがまだメモリに書き戻されていない) ならば，0 になるまで待つ．そうでなければ，RLSR の 0 番ビットを 1 にセットし，OR の書き戻しを LSU に通知する．

以上から得られた SOC の構成例を図 9 に示す．mb[1] は，LSU 内の LR に割り当てられるため，タグ上でのスピンはチップ内結合網に影響を与えない．また，CPU はメールボックス (mb[0]) をキャッシュでき，その場合も，CPU によるスピンはチップ内結合網に影響を与えない．ただし，キャッシュにメモリの一貫性を保つ機構を要する．

4.2 並列化の適用

ソフトウェアをハードウェア化する主な利点は，処理に内在

```
for (i=0; i<n; i+=2){
 a[i  ]=b[i  ]*c[i  ];
 a[i+1]=b[i+1]*c[i+1];
}
```

(a) Loop Unrolling

```
1: while(mb.b0==' 0');
2: mb.b0=0;
3: for(i=0; i<n; i+=2){
4:   ldws(IR[j  ],b[i],2,NOP );
5:   ldws(IR[j+2],c[i],2,RLS );
6:   stws(OR[i  ],a[i],2,WAIT);
7:   j+=4;
8: }
9: mb[0].b0=1;
```

(b) LSU program

```
   while(1){
S0:  while(WAITR(1)==' 0');
     WAITR(1) = ' 0';
S1:  OR[i  ]=IR[j  ]*IR[j+2];
     OR[i+1]=IR[j+1]*IR[j+3];
     i+=2; j+=4;
S2:  while(RLSR(0)==' 1');
     RLSR(0)=' 1';
   }
```

(c) EXU behavior

図 10 ループ展開による並列化

Fig. 10 Parallelization using Loop Unrolling

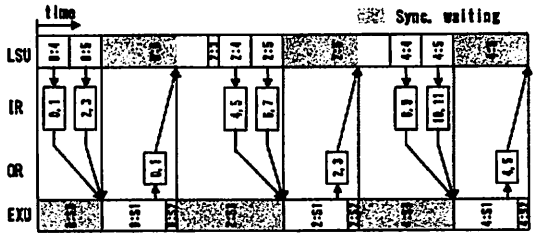

図 11 ループ展開を適用した SPRCP の動作

Fig. 11 Execution Snapshot of SPRCP with loop-unrolling

する多重の並列性を引き出すことによって高性能化が図れる点である．特にループは並列化によって大きな性能向上が望める部分である．前節では，基本的な SPRCP へのマッピング方法を示すことが目的であり，ループに関して並列化を行わなかった．本節では，基本的手法に対して並列化を導入した SPRCP へのマッピング方法について示す．

ここではループを並列化するにあたって高位合成において一般的に用いられているループ展開を利用する [1], [2]．その概要を図 10 に，SPRCP の動作の様子を図 11 に示す．これは展開数が 2 の例であり，内部の 2 つの式を並列実行することによって理想的には 2 倍の性能向上が期待される(注2)．図 11 の実線はデータの流れを，点線は RLS，および，WAIT の様子 (矢印の元がリリース側，先がウェイト側) を表す．LSU におけるコロンの左側はループの添え字，右側は図 10 (b) の行番号であり，EXU におけるコロンの左側はループの添え字，右側は図 10 (c) 状態である．IR と OR の数値はレジスタ番号を表す．

LSU プログラムに関しては，図 10 (b) に示すとおり，同期付きロード/ストア命令で転送ワード数を指定できる利点が活用されている．ループ展開によって配列 b と c の要素を 2 個ずつ読み出す必要があり，転送回数を 2 にするだけで対応できている．それに伴い，一度の繰り返しで IR が 4 つ分占有されるが，添え字の更新を +4 に変更することで容易に対応可能である．EXU でも，LSU プログラムと同様に，添え字の更新を変

(注2)：一般的には展開数倍の性能向上が期待される．ただし，展開数の増加に伴ってハードウェア量も増加するため，性能とハードウェア規模とのトレードオフが必要となる

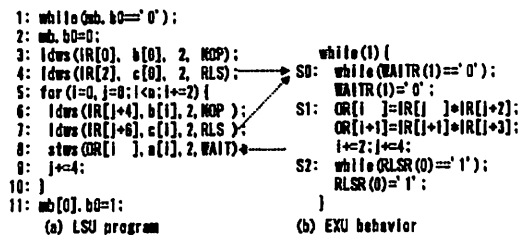

図 12 ダブルバッファリングの適用

Fig. 12 Applying Double Buffering On SPRCP

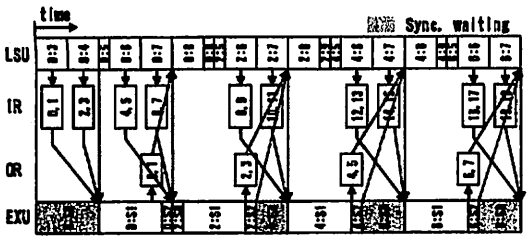

図 13 ダブルバッファリングを適用した SPRCP の動作

Fig. 13 Execution Snapshot of SPRCP with double-buffering

更するだけである．任意の展開数でも同様に SPRCP へマッピング可能と考えられる．

4.3 ダブルバッファリングによるレイテンシ隠蔽

ループ展開を適用した場合，理想的には，並列化前に対して性能を展開数倍向上できる．ただし，図 11 に示すとおり，実際は，メモリアクセスレイテンシによって，理想的な性能向上は達成できない．特に，図 11 のような場合，LSU がデータを読み出した後でしか EXU は演算を開始できないため，メモリアクセスレイテンシの悪影響を完全に被ることになる．

そこで，IR をダブルバッファとして利用し，メモリアクセスの隠蔽を図る．その概要を図 12 に示す．図 12 (a) に示すとおり，LSU プログラムでは，メインループの開始前に第 1 イタレーションで使用する配列 b と c の要素を IR にローディングする (3〜4 行目)．その後は，メインループ内で，次イタレーションで使用される配列 b と c の要素を次イタレーションで使用される IR にローディングする (6〜7 行目)．IR はリング構造なので，ダブルバッファを形成するために IR をバルク単位で切り替える手間が必要なく，次々と IR へデータを供給するだけでよい．また，図 10 (b) と図 12 (b) から，EXU の振る舞いを変更せずに LSU プログラムのみをチューニングするだけで性能の最適化が行え得ることも確認できる．結果として，図 13 に示すとおり，図 11 と比較して，大幅な性能向上が期待される．

5. むすび

HW/SW コデザインシステムにおけるハードウェアモジュールのフレームワークとして SPRCP の導入を提案した．SPRCP の導入は以下のような利点をもたらす．

・LSU によってストリームからランダムにわたった様々なメモリアクセスパタンに柔軟に対応できる．

・EXU 上に構成されるハードウェアはレジスタ読み書きとフラグチェックの単純なインタフェースを持てば良い．

・CPU(SW)–SPRCP(HW), SPRCP(HW)–SPRCP(HW) 間の通信は，共有アドレス空間上のメールボックスとデータに対する読み書きをベースとしたプログラムによって統一的に扱える．

・LSU と EXU を適切に協調させることによってメモリアクセスレイテンシを隠蔽できる．

SPRCP へのマッピングに関するケーススタディを通して，ハードウェアの生成に並列化を施した場合にもマッピング可能であること，ダブルバッファリングによりメモリアクセスレイテンシの隠蔽が可能なことを確認した．さらに，EXU 上のハードウェアを変更せずに，LSU のプログラムをチューニングすることで，性能の最適化が行えることも確認した．なお，先行研究において，LSU とレジスタ (すなわち EXU 以外) を FPGA(Virtex4 XC4VLX25) に実装したところ，5%程度のリソース使用量であることが確認されている [8], [9]．今後の課題として，より多くのアプリケーションを用いた記述実験と実機での評価があげられる．また，高位合成ツール (ImpulseC [1]) と LSU を結合するための環境を現在開発中である．

文 献

[1] D. Pellerin and S. Thibault: "Practical FPGA Programming in C", Prentice Hall (2005).

[2] S. Gupta, N. Savoiu, N. Dutt, R. Gupta and A. Nicolau: "Using global code motions to improve the quality of results for high-level synthesis", IEEE Trans.on Computer-Aided Design of Integrated Circuits and Systems, **23**, 2, pp. 302–312 (2004).

[3] 本田, 富山, 高田："システムレベル設計環境:SystemBuilder", 信学論, **J88–D–I**, 2, pp. 163–174 (2005).

[4] 遠藤, 小泉："再利用モジュールのオンライン評価を取り入れたハードウェア・ソフトウェア協調設計方式とその検証", 電学論 C, **124**, 11, pp. 2249–2259 (2004).

[5] A. A. Jerraya and W. Wolf: "Multiprocessor Systems-on-Chips", Morgan Kaufmann Publishers (2005).

[6] R. K. Gupta and G. D. Micheli: "Hardware-Software CO-synthesis for Digital Syste", IEEE Design and Test, **10**, 3, pp. 29–41 (1993).

[7] M. Luthra, S. Gupta, N. Dutt, R. Gupta and A. Nicolau: "Interface Synthesis using Memory Mapping for an FPGA Platform", Proc.of the 21st International Conference on Computer Design, pp. 140–145 (2003).

[8] 下尾, 山脇, 岩根："プログラマブルなロードストアユニットと演算部が協調する再構成プロセッサアーキテクチャ", FIT2006 第 5 回情報科学技術フォーラム, pp. 153–156 (2006).

[9] 菱川, 山脇, 岩根："再構成プロセッサに適したロードストアユニットの設計", 第 59 回九州連大, p. 158 (2006).