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

2007－SLDM－130（4）
2007／5／10

SystemCを用いた動的再構成可能プロセッサのモデル開発

上田 浩司† 北道 淳司†† 黒田 研一†

† 会津大学大学院コンピュータ理工学研究科
†† 会津大学コンピュータ理工学部
〒965-8580 福島県会津若松市一箕町鶴賀
E-mail: ††{kitamiti,kuroken}@u-aizu.ac.jp

あらまし 近年, FPGA 技術を応用した動的再構成可能プロセッサ (DRP) が提案されている. DRP は独自の動的再構成可能アーキテクチャ (DRA) を持ち, 独自の設計自動化環境が用意されている. しかし, アプリケーションに特化した新規の DRA を含む動的再構成可能システム (DRS) のシステム設計において, 既存の DRP 用記述言語および CAD では対応できない可能性がある. 本稿では, システムレベルにおける汎用の DRS のモデリングのために開発している動的モジュールライブラリを利用した DRP のシステムレベルモデリングについて述べる. 動的モジュールライブラリは SystemC の拡張ライブラリであり, モジュールの動的な生成・削除およびポートの動的な接続・分断のモデリングが可能である. 提案プロセッサのアーキテクチャは, 基本となる MIPS 型アーキテクチャに対して, 動的に生成・削除される演算器のための命令およびそれらの生成・削除命令を追加したものである. 提案プロセッサのモデリングおよびそのシミュレーション結果について述べる.

キーワード 動的再構成可能プロセッサ, SystemC, システムレベル, モデリング

A Modeling for Dynamically Reconfigurable Processor using SystemC

Koji UEDA†, Junji KITAMICHI††, and Kenichi KURODA†

† Graduate School of Computer Science and Engineering, The University of Aizu
†† School of Computer Science and Engineering, The University of Aizu ,
Tsuruga, Ikki-machi ,Aizu-Wakamatsu City,Fukushima 965–8580 Japan
E-mail: ††{kitamiti,kuroken}@u-aizu.ac.jp

Abstract Recently, dynamically reconfigurable processors (DRPs) based on FPGA technology are proposed. DRPs are implemented on unique dynamically reconfigurable architecture, and a specialized design environment is provided for the DRP. In the case of the system design for new application specific dynamically reconfigurable system (DRS), existing description language and CAD system for existing DRA can not deal with this system design. In this paper, we describe the system level modeling of a DRP using a dynamic module library, which we have developed for the modeling of general purpose DRSs at the system level. The dynamic module library is an extended SystemC library, and enables the modeling of the dynamically generation and elimination of modules, ports and channels and the connection and dispatch between port and channel. The architecture of proposed processor is based on a MIPS type architecture and is appended the instructions, which are for the dynamically reconfigurable operational units and for the generation and elimination of them, and the hardware resources for the execution of appended instructions. We describe the proposed DRP model and its simulation results.

Key words Dynamically Reconfigurable Processor, SystemC, System Level, Modeling

1. はじめに

近年, FPGA 技術を利用した動的再構成可能プロセッサ (DRPs) が数多く提案されている [1] [2] [3] [4] [5] [6] . それぞれの DRP は独自の動的再構成可能アーキテクチャを持ち, 独自の設計自動化環境が用意され, それらに適したアプリケーションが実装され評価されている. しかし, アプリケーションおよびそれに特化した新規の DRP アーキテクチャの設計も含めた

設計においては, 既存 DRP のための記述言語および CAD では対応できない.

新規の DRP アーキテクチャの設計においては, システム開発初期の段階におけるシステムレベル設計 [7] [8] での特定の動的再構成可能アーキテクチャに依存しない設計環境の構築が必要となると考えられる. システムレベル設計は, 従来の RTL より抽象度の高いレベルでの設計であり, 記述量の削減, 既存の IP(Intellectual Property) の再利用, ハードウェア/ソフトウェアの協調設計が特徴として挙げられる. ソフトウェアとハードウェアの親和性の高い, C や C++ベースにして同時並列性や階層構造のようなハードウェア指向を持たせた SystemC [9] や SpecC [10], 従来のハードウェア記述言語である VerilogHDL を高い抽象レベルでシステムを記述するために必要な仕様を追加した SystemVerilog [11] などが提案されている.

我々はシステムレベルにおける汎用の動的再構成可能システム (DRS) のモデリングのために, モジュールの動的な生成・削除およびポートおよびチャネルの動的な接続・分断のモデリングを可能にする動的モジュールライブラリを開発している. 最新の SystemC2.1.v1 では, 動的プロセスという機能が導入され, シミュレーション開始後に動的にプロセスを生成・削除することが可能である. しかし モジュール, ポートおよびチャネルに関してはシミュレーション開始後に生成・削除はできない. 提案ライブラリを用いることにより, シミュレーション開始後に, モジュール, ポートおよびチャネルの動的な生成・削除とポートとチャネルの動的な接続・分断が可能である. 提案ライブラリを用いることにより, 例えばマルチコンテキスト型 DRA の実装レベルでのデリング方法のひとつであるマルチプレクサ・デマルチプレクサを用いたコンテキスト切り替えによるモデリングと同等の記述量で, 同等のシミュレーション速度が可能である [12].

本論文では, 動的モジュールライブラリを用いた, 動的再構成可能プロセッサのシステムレベルモデリングについて述べる. 対象プロセッサは, MIPS 型アーキテクチャを基本とし, 動的に生成・削除される演算器のための命令およびそれらの生成・削除命令を追加している. 提案プロセッサのモデリングおよびそのシミュレーション結果について述べる.

以降, 2 章,3 章では, それぞれ提案するプロセッサの仕様とその実装について述べる. 4 章で評価, 5 章でまとめとする.

2. 提案プロセッサ仕様

提案する動的再構成可能プロセッサ (DRP) の仕様について述べる. 提案プロセッサはハーバードアーキテクチャを持つ. 提案プロセッサは, 動的に生成・削除可能な演算器を有し, 提案プロセッサの実装の一例として, 図 1 に示すようなプロセッサコアと複数の動的再構成可能な演算器を実装可能な動的再構成デバイスからなる構成があげられる. 本研究では, デバイスに依存しない, より抽象度の高いレベルでのモデリングを考える.

(a) プロセッサコア

本研究において提案する DRP は 32bit プロセッサである. 提案プロセッサは MIPS [14] のサブセット命令を実行することができ,5 段のパイプランを持つ. 整数演算命令,Load/Store, 分岐命令など約 50 命令をもち, 各命令は Inorder 実行される.

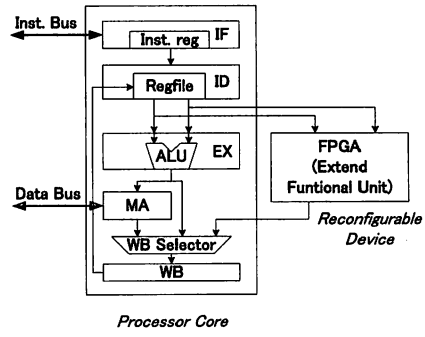

図 1 提案プロセッサの一構成

提案プロセッサにおいて,Load/Store 命令以外の命令は Memory Access(MA) ステージをスキップし, 演算器で求められたデータを Instruction Decode(ID) ステージや Execution(EX) ステージにフォワーディングする. 静的な内部演算器は整数 ALU のみとし, Load/Store, 乗除算以外の算術演算命令および乗除算命令は, それぞれ,5,4 および約 35 サイクルで実行する.

動的再構成に関係する命令は, 演算器を動的に生成する命令とそれらの演算器への命令がある. 動的に生成される演算器は動的再構成デバイス上に生成され,ID ステージから渡されるデータに対して演算を実行し WB ステージに結果を渡す. レジスタファイルと密結合な DRP としては PRISC [13] がある. PRISC における演算は 2 入力・1 出力のもののみであり, その再構成可能ユニット (PFU) において実行できる命令はシングルサイクルで終了する命令に限られている. 提案プロセッサでは, 演算は毎サイクルごとにレジスタファイルにアクセスすることができ, マルチサイクル実行が可能である.

提案プロセッサでの処理は, 例外および割り込み処理を含む. 例外および割り込みは, オーバーフロー, マスク可能/不可能割り込み, システムコール, ブレイク, 予約命令である.

以下, 各ステージの処理の概要をまとめる.

Instruction Fetch(IF) ステージでは, 命令キャッシュへアクセスし命令をフェッチする. 同時に PC を更新する. また分岐または Jump 命令の実行時に, 分岐または Jump 先アドレスを算出する. 先行命令が ID ステージでストールするときは, フェッチした命令をプリフェッチバッファに入れる.

ID ステージでは, 命令をデコードする. EX,MA,WB ステージで用いる制御信号を生成し, データのハザードの検出, データフォワーディングの検出とその制御, 条件分岐命令用の分岐アドレスの算出, および例外の検出を行う. 割り込み処理は全てこのステージで行われる. マルチサイクル命令発行時には後続の命令の実行をストールさせる. Jump 命令はこのステージで Jump 先アドレスへ分岐し, 実行を完了する.

EX ステージでは, ALU,Shifter および動的に再構成される演算ユニットを用いて演算を行う. Load/Store 命令ではアドレスの計算およびストアデータの処理を行う. また, 条件分岐命

令の条件判定もこのステージで行う. フォワーディング時には, フォワーディングデータを受け取る. マルチサイクル命令実行完了時やメモリアクセス完了時は該当命令の後続の命令を再開させる. 分岐命令の場合はこのステージで分岐先アドレスへ分岐し, 実行を完了する.

MA ステージでは, データキャッシュへのアクセスを行う. アクセス中は後続の命令の実行をストールさせる. Load/Store 命令以外はこのステージは処理を行わない.

Write Back(WB) ステージでは, 一般の演算命令のときはレジスタファイルに実行結果を書き込む. 乗除算命令の時は HI レジスタ,LO レジスタに 1 クロック 32bit ずつ,2 クロックで書きこむ. また必要な時は,ID や EX ステージに実行結果をフォワーディングする.

浮動小数点用のレジスタファイル, 浮動小数点汎用レジスタ (Floating-point General Register:FGR) を持つ. 提案プロセッサは,32 ビット FGR を 32 個を持つ. また倍精度の浮動小数点を扱うために論理的なレジスタである浮動小数点レジスタ (Floating-Point Register:FPR) 16 個としても利用できる.FPR へのアクセスは, 隣接する 2 個の FGR に同時にアクセスすることで実現される. データキャッシュと FGR 間のデータや汎用レジスタと FGR 間を転送する命令が実装される.

例外や割り込み処理においては, 現在のプログラムカウンタをセーブし, その例外/割り込みの種類を原因レジスタに, 現在のステータスをステータスレジスタにそれぞれ保存する. 次のサイクルでその例外や割り込みに対応した例外分岐先アドレスを生成し, 分岐する.

(b) 動的再構成可能ユニットとプロセッサコアとのインターフェース

動的再構成可能ユニット (Dynamically Reconfigurable Execution Unit:DREU) は 2 入力, 1 出力のポートを持ち, 内部に複数の動的に生成される演算器を格納する.

動的再構成のための命令は, 演算器を動的に生成する命令とそれらの演算器に対する演算命令がある. 演算器を動的に生成する命令は 32 ビット命令であり, 6 ビットの Configuration Address フィールドと 3 ビットの Address フィールドをもつ. Configuration Address フィールドは, 生成される演算器の種類を表し,Address フィールドは, DREU 内の再構成ブロックを指定する. 演算器を生成する時間および演算に必要なクロック数は, 生成される演算器の種類ごとに定義されているものとし, パラメータとして与えることができる. プロセッサコアは, 指定されたサイクルの間,ID ステージをストールさせる.

DREU 上に実装される演算器を利用する演算命令は 32 ビット長の命令であり, 3 ビットの Address フィールドによって, どの再構成ブロックにおいて実行するかを指定する. 命令は,2 もしくは 3 オペランド命令をもち, ID ステージで解釈され, レジスタファイル内のデータを元に指定された演算器が演算を実行し, 結果を 32bit づつ WB ステージに転送する.

これらの命令の実行は図 2 に示すように行われる. 図 2 には,ID ステージと DREU がモジュールとして表されている. 本研究におけるモデリングでは, 動的再構成に関する動作は抽象化され, 以下のようにシステムが構成される.

演算器を動的に生成する命令の場合,ID ステージは動的再構成開始イベント Config_en と, 生成すべき演算器の種類を表す Config_adr を DREU に転送する.DREU は演算器の生成に必要な時間 Creating_time を経過させる. 既に構成されている演算器を削除する必要がある場合は, 時間 Deleting_time も経過させる. これらの時間が経過した後,DRUE は ID ステージに構成終了イベント Genend を発行する.

特定の動的再構成デバイスが確定されたレベルでは, 以下のようにシステムが構成される. 演算器を動的に生成する命令の場合,ID ステージは DREU の構成情報が格納されたアドレス Config_adr を構成情報が格納されているメモリ Config_Mem に発行する. Config_Mem は, アドレス Config_adr に納められている構成情報 Config_data を DREU に転送する. 以下 DREU 内で上記と同じ手順が実行される.

全ての DREU がコンテキスト内に設定されており, 動的再構成はコンテキストの切替のみ (外部からの構成情報のロードがない) であるようなシステムのモデリングにおいては,Creating_time および Deleting_time は 0 に指定される.

演算命令の場合,ID ステージは処理を行うソースオペランド (浮動小数点もしくは整数)FP_data あるいは INT_data および DREU を制御する制御信号 Dreu_op(DREU で実行する命令), Dreu_adr(どの動的再構成ブロックで実行するかのアドレス) が発行される. 演算器での演算に必要な時間は, パラメータ Running_time で表される. 演算終了後, 演算結果は WB ステージに転送されると同時に,ID ステージに動作完了信号 Exend が発行される.

再構成の状況は DREU 内にあるテーブルにより管理する. テーブルは Configuration Address と DREU 内の Address を保持しており, どの演算器がどの Address に生成されているかを保持する.

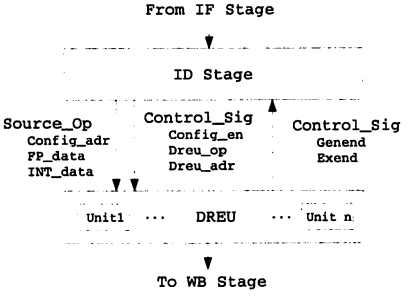

図 2 プロセッサコアと DREU とのインターフェースの概要

3. 提案プロセッサのモデリング

提案プロセッサのモデリングについて述べる. モデリングには SystemC およびモジュールおよび動的モジュールライブラリ [12] を用いた.

3.1 動的モジュールライブラリ

システムレベルにおいて, モジュールおよびインターフェースの動的な生成・削除を含むシステムのモデリングを可能にする動的モジュールライブラリについて述べる. 動的再構成可能システム (DRS) において, モジュールおよびインターフェースは, システムの動作中に生成・削除される. システムレベルにおいてこれらの振舞を自然に表現するためには, モジュール, ポートおよびチャネルが動的に生成・削除されなければならない. しかし, 最新の SystemC 2.1.v1 を用いたとしても, シミュレーションが開始された後, モジュール, ポートおよびチャネルを生成したり削除することはできない.

我々は, 動的に生成・削除可能なモジュール, ポートおよびチャネルとそれらの対する処理を含む動的モジュールライブラリを開発した. 提案ライブラリは, モジュールの活性時間の記録などのプロファイリングの機能を実現している. 提案ライブラリの詳細は論文 [12] に述べられている.

システムレベルでのモデリングにおいて提案ライブラリを用いると, 通常の *sc_module* クラスに加えて,dynamic module クラスなどを用いることができる. これのクラスは, 動的モジュール, ポートおよびチャネルに対して, 生成・削除のための,*newmod()* あるいは *delmod()* のメソッドなどを提供する.

動的モジュールでは, 入出力ポートの宣言として用いられる *sc_in* の代わりに *dc_in* が, チャネルの宣言として用いられる *sc_fifo* の代わりに *dc_fifo* を用いて宣言する. これらのポートやチャネルは, シミュレーション後に, 動的に生成・削除が可能であり, 動的な接続・分断が可能である.

例えばマルチコンテキスト型 DRA を用いたモデリングでは, 動的モジュールにおける生成あるいは削除期間の動作は, 外部システムからのコンテキストのローディング, コンテキストやレジスタの初期化, 再構成を行うレジスタデータの退避に対応する. それらに加えて, 動的再構成時のエラー処理についても表現する必要がある場合がある. これらの処理のモデリングのために, 提案ライブラリにおける動的モジュールは, あらかじめ用意された 3 つの状態 (*creating*, *running* および *deleting*) を使用できる. 各状態における処理は, 動的モジュールの生成時, 通常動作時, 削除時の処理を定義することができ, 設計者は各状態での処理をユーザプロセスとして宣言できる. 生成および削除に必要な時間を, それらのモジュールの大きさやアーキテクチャの特徴に応じてそれぞれ *creating_time* および *deleting_time* というパラメータとして記述できる.

動的モジュールの処理のオーバヘッドによってシミュレーション時間が増大する可能性もあるが, 動的モジュールにはいくつかの高速化手法が組み込まれており, 必要に応じて必要なものを適用することができる.

3.2 動的再構成プロセッサのモデリング

SystemC および動的モジュールライブラリを用いた提案プロセッサのモデリングについて述べる. プロセッサコアの 5 段のパイプラインはそれぞれのステージごとにデータパス部と制御部のモジュールからなる. 動的に再構成される実行ユニットとそれに関係するプロセッサコアのデータパス部と制御部について述べ, 動的再構成に関係のない IF,MEM,WB ステージ,EX ステージにおける通常の命令を実行する部分に関する内容は特に言及しないかぎり省略する.

(a) 実行ユニット

動的に生成される演算器として, 浮動小数点加算・減算・除算・乗算の 4 種類を用いる. 浮動小数点フォーマットとして,IEEE754 を採用する. 入力データは単精度もしくは倍精度の浮動小数点とし, 不正規化数は扱わない. 丸め方式は最近点への丸めを用いる. 各演算の実行サイクル数はパラメータとして与えられる. DREU 内の動的再構成ブロック数は 2 とする.

浮動小数点の四則演算を行う実行ユニット 4 種のモデルについて述べる. 図 3 に浮動小数点乗算ユニットのモデルの一部を示す. Dynamic_Mulf クラスは,Dynamic Module クラスの派生クラスとして宣言される. このクラスは,dc_in あるいは dc_out の動的に再構成可能な入出力ポートをもち,running などのプロセスをもつ.

running プロセスは, 浮動小数点の乗算を実行する. 入力ポート *fsin* および *ftim* の値から演算結果をもとめ出力ポート *fout* に結果を出力する.

```
#include "DynamicModule.h"

class Dynamic_Mulf: public DynamicModule
{
public:
  dc_in<sc_uint<DWORD> > fsin,ftin;
  dc_out<sc_uint<DWORD> > fout;
  dc_in<bool> dc_en;

  SC_HAS_PROCESS(Dynamic_Mulf);
  Dynamic_Mulf(const char *name, int ctime,
               int dtime, sc_time_unit tunit):
    DynamicModule(name, ctime, dtime, tunit)
  {

  }
  ~Dynamic_Mulf(){};

  void running();
  void creating();
  void deleting();
};

void Dynamic_Mulf::running(){
      :
  while(true){
    while(dc_en.read()==false){
      if( dwait(dc_en.value_changed_event()) )return;
    }

    fs_signTmp = fsin.read()[31];
    ft_signTmp = ftin.read()[31];
    fs_expTmp =  fsin.read().range(30,23);
    ft_expTmp =  ftin.read().range(30,23);
    fs_manTmp =  fsin.read().range(22,0);
    ft_manTmp =  ftin.read().range(22,0);

       :

    dwait(mulf_etime, SC_NS);
    fout.write( (sign1_tmp, exp5_tmp, man5_tmp) );
    dwait(mulf_waits, SC_NS);
  }
}
```

図 3 動的モジュールのモデル

(b)DREU における動的再構成の制御

DREU における, モジュールの生成・削除の制御のモデリン

グについて述べる. DREU の制御を図 4 に示す.

DREU における制御部は,Idle, Creating, Running, Deleting の 4 状態をもつ. (1)Reset 信号によって, 状態 Idle に遷移する. (2)ID ステージからの再構成要求信号 Config_en により, 状態 Creating に遷移する. この状態で, 再構成に必要な Creating_time 時間停止する. (3) その後, 再構成完了信号 (Genend) を ID ステージに送信し, 状態 Running に遷移する. (4,5) 状態 Running は, 通常の演算を行う状態である. 演算を実行し,Running_time で指定したサイクル数を経過させる. その後, WB ステージへ演算結果を出力するとともに,ID ステージへ演算完了信号 (Exend) を送信し, 次の命令とデータを待つ. 演算は上記 (a) で述べた演算ユニットで行われる. (6) 再構成要求信号の到着により, 状態 Deleting へ遷移し,Deleting_time サイクル経過後, 演算器を削除する. (7) 要求に応じ演算器を新たに再構成するために状態 Creating に遷移する. (6) や (7) の遷移は, 既に演算器が存在している状況において, 新しい再構成要求信号を受け取った際の動作を示している.

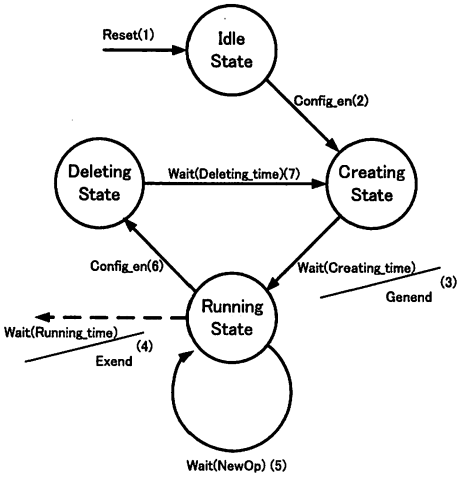

図 4 DREU における制御部

DREU における動的再構成に関するモデルについて述べる. 上記の手順 (6) の一部を 5 に示す. 使用中の DREU のブロック番号 i の演算器を表す unit[i] に対して, 新たな Dynamic_Addf モジュールの書き込み要求の処理が記述されている. もともと存在する動的モジュールに対して,(1) 分断を行うメソッド detach により, チャネル dina,dinb,dout などと, 動的ポート fsin,stin,fout などを切り離す. (2)delmod メソッドによりモジュールを消滅させ, 消滅に必要な時間を経過させる. (3)new により新たにモジュールを生成させる. (4)bind メソッドによりチャネル dina,dinb,dout などと, 新たに生成された動的ポートを接続する. (5) 生成に必要な addf_ctime 時間を経過させる.

チャネル dina,dinb,dout などは,DRUE の外部ポートに静的に接続される.

(c)ID ステージにおける制御

プロセッサコアの ID ステージにおける制御について述べる.

```
// If unit[i] is used and reconfigured to Dynamic_Addf
          unit[i]->fsin.detach(dina);
          unit[i]->ftin.detach(dinb);
          unit[i]->fout.detach(dout);
          unit[i]->dc_en.detach(ex_en);
          unit[i]->delmod();
          wait(unit[i] -> dtime, SC_NS);
          unit[i] = new Dynamic_Addf("addf",
                    addf_ctime, addf_dtime, SC_NS);
          unit[i]->fsin.bind(dina);
          unit[i]->ftin.bind(dinb);
          unit[i]->fout.bind(dout);
          unit[i]->dc_en.bind(ex_en);
          unit[i]-> dtime = addf_dtime;
          wait(addf_ctime, SC_NS);
        }
                              :

//Connection Input Port and Channel
void Dynamic_Control::data_in() {
  while(true) {
    wait();
    dina = in0.read();
    dinb = in1.read();
  }
}

//Connection Channel and Output Port
void Dynamic_Control::data_out(){
  while(true) {
    wait();
    if(exend_tmp==true){
      out.write(dout.read());
    }
  }
}
```

図 5 動的モジュールの制御のモデル

動的再構成に関する命令は, もともとの MIPS において使用されていない命令コードを用いて表される. ID ステージは,IF ステージから転送された命令に対して, 命令コードにしたがって各命令をインオーダに発行する.

演算器を動的に生成する命令が発行された場合, 後続の命令をストールさせ,DREU からの再構成完了イベント Genend により命令の発行を再開する. DREU において演算を実行する際も同様に後続の命令をストールさせ, 実行完了イベント Exend により命令の発行を再開する. これらの命令の実行中に割り込みが起きた場合は, 実行中の命令全ての実行を完了した後, 割り込みに対応した処理を行う.

DREU 内部の演算器への演算命令の場合, ソースオペランドを DREU に転送し, DREU での実行完了イベント Exend を待つ. イベント Exend の到着を受けて, ID ステージは,EX ステージへ DREU での演算結果を書き込むための書き込み許可信号および結果を格納するレジスタファイルのアドレスを発行する. なお, イベント Exend は,DREU での実行が完了する 1 サイクル前に送信されるため, DREU からの結果の出力に合わせて実行結果のレジスタファイルへの書き込みが完了する.

4. 評　　価

提案した動的再構成可能プロセッサに対して, SystemC 2.1.v1 と動的モジュールライブラリを用いてモデリングを行い, シミュレーションによる評価を行った. シミュレーション環境は, Xeon 3.2GHz, Mem 4GB, Linux 2.6.9, GCC 3.4.6 である. 評価は, 標準の SystemC を用いて記述した Mux モデルとの比較であ

る. Mux モデルは, 複数の静的モジュールをマルチプレクサ/デマルチプレクサで切り替え, 動的再構成を仮想的に実現する方法である. 動的モジュールライブラリによるモデルと,Mux モデルのコード量を表 1 に示す. コード量は, コメント, 空行を取り除いた行数である.

	S Modules	D Modules	Control
Mux model	8248	787	507
Proposed method	8248	751	588

表 1 モデルのコード量

表 1 において,"S Modules"は, 静的モジュールで構成されるデータパスおよび制御部であり, 提案モデルと Mux モデルにおける共通部分である. "D Modules"は動的再構成に関する部分である. 提案モデルでは動的モジュールの記述であり,Mux モデルでは静的な演算モジュールとマルチプレクサ/デマルチプレクサの記述を含む."Control"は動的再構成の制御に関する部分であり, 提案モデルではモジュール, ポートおよびチャネルの生成・削除, 接続・分断の記述を含む.

"D Modules"ではマルチプレクサ/デマルチプレクサなどの記述を含む Mux モデルが若干多くなるが,"Control"では, モジュールなどの生成/削除/接続/分断の記述を含む提案モデルが多くなる. 全体においては, 双方の記述量の違いはわずかである. しかし, 提案モデルは, 特定のデバイスに依存せず, モジュールの生成時における初期化の動作や生成中や削除中の外部からのエラーに対する動作などを自然に記述できる.

次に, 提案モデルと Mux モデルのシミュレーション時間 (秒) を計測した. 計測において, 演算器を動的に生成する命令と生成した演算器における演算命令の実行を 10000 回行った. その結果を表 2 に示す.

Mux model	6.13
(1)	6.90
(2)	6.72

表 2 シミュレーション実行時間

表 2 において,(1) は動的モジュールライブラリの高速化手法を用いない実行である. (2) は動的モジュールを再利用するという高速化手法を採り入れた. シミュレーション時間は,Mux モデルが一番高速であるが, 高速化手法を用いた (2) との差はわずかである.

5. ま と め

動的モジュールライブラリを用いた動的再構成可能プロセッサのモデリングについて述べた. 開発したモデルは, モジュール, ポートおよびチャネルの動的な生成・削除を自然にモデリングすることができる. またマルチプレクサ・デマルチプレクサを用いたモデルと同等の記述量およびシミュレーション時間を達成可能であることを示した.

提案モデルは, プロセッサコアが RTL に近く, 動的再構成に関する部分が抽象化されたものであり, システム全体のモデリングレベルが統一されていない. 今後, システム全体について記述レベルを統一し, より洗練されたものにしたい. プロセッサおよび周辺モジュールをふくめた規模のモデリングへの適用を行い, 提案手法を評価したい.

また, 動的モジュールを用いたモデルから静的なデバイスと動的なデバイスへ実装するためのシステム分割の手法および実際の動的再構成可能デバイスへの動作合成についても検討したい. 当初は特定の動的再構成可能デバイスに限定し, これらの問題に取り組んでいきたいと考えている.

謝辞 この研究は栢森情報科学振興財団の助成を受けて遂行された.

文 献

[1] M.B.Gokhale and P.S.Graham, "Reconfigurable Computing : Accelerating comutation with field-programmable gate arrays," SPRINGER, 2005.

[2] H. Nakano, T. Shindo, T. Kazami, and M. Motomura, "Development of dynamically reconfigurable processor lsi," NEC Technical Journal,Vol.56,No.4,pp.99-102, 2003.

[3] T.Toyo, H.Watanabe, and K.Shiba. "Implememtation of Dynamically Reconfigurable Processor DAP/DNA-2," Proc, of IEEE VLSI-TSA International Symposium on VLSI Design, Automation & TEST, pp.321-322, Apr.2005.

[4] J.E.Carrillo, P.Chow, "The Effect of Reconfigurable Units in Superscalar Processors," Proc. of the 2001 ACM/SIGDA 9th International Symposium on Field Programmable Gate Arrays, pp.141-150, Feb.2001.

[5] T.Ito, K.Ono, M.Ichikawa, Y.Okuyama, and K.Kuroda, "Reconfigurable Instruction-Level Parallel Processor Architecture," ACSAC2003(LNCS 2823), pp.208-220, Sep.2003.

[6] PACT XPP Technologies,"Reconfiguration on XPP-III Processors White Paper," http://www.pactxpp.com/main/download/XPP-III_reconfiguration_WP.pdf.

[7] H.Kurosaka, K. Takemura, and M.Tachibana, "System Level Design Flow and Design Language," IPSJ Magazine, Vol.45,No.5,pp.456-463, May.2004

[8] R. Bergamaschi and W. Lee, "Designing Systems-on-Chip Using Cores," Proc. of the 37th conference on Design automation, pp.430-425, Jun.2000.

[9] Open SystemC Initiative, "SystemC 2.1 Language Reference Manual," http://www.systemc.org.

[10] SpecC Technology Open Consortium, http://www.specc.org.

[11] Accellera Organization, Inc., http://www.systemverilog.org.

[12] K. Asano, J. Kitamichi, and K. Kuroda, "Proposal of Dynamic Module Library for System Level Modeling and Simulation of Dynamically Reconfigurable Systems,"20th Int. Conf. on VLSI DESIGN(VLSI DESIGN 2007),B2,5019, 2007.

[13] R.Razdan and M.Smith., "A high-performance microarchitecture with hardware-programmable functional units," Proc. of 27th the Annual International Symposium on Microarchitecture,pp.172-180, Nov.1994.

[14] G. Kane and M. Maekawa, "mips RISC Architecture-R2000/R3000-," Kyouritsu Syuppan, 1992.